Chapter 5

Path Planning: Roadmap Methods

5.1 General Concepts

5.2 Voronoi Roadmaps

A *Voronoi roadmap* is based on the topological notion of retraction. It corresponds to a skeleton-like structure that is obtained by incrementally shrinking the free space. The shrinking occurs by repeatedly removing a thin strip from the boundary of the free space. Strips are removed until a one-dimensional structure remains.

The resulting roadmap has the following property (see Figure 5.1. For a given point in the free space, consider the closest point or points in the obstacle region. For any point in the Voronoi roadmap there are two or more closest points to the obstacle region. In other words, the Voronoi roadmap yields solutions for which a robot has maximum clearance.

The obstacle region is specified as a set of polygons. Each polygon is specified by an list of vertices. Each edge of the polygon connects two vertices in the list. The terms *edge* and *vertex* will be used to refer to the edges and corners of the boundary of the obstacle region. There will be three possible cases when there are two closest points: 1) both points are vertices; 2) one point is a vertex and the other lies in the interior of an edge; and 3) each point lies in the interior of an edge.

One naive way to construct the Voronoi roadmap is as follows. Note that the obstacle region is specified by a set of edges. For every possible pair of edges, generate a line as shown in Figure 5.2.a. The edges that specify the obstacle region specified by endpoints. For every possible pair of endpoints, generate a line as shown in Figure 5.2.b. Also, for every possible combination of endpoint and edge, generate a quadratic curve as shown in Figure 5.2.c. The Voronoi roadmap will be included amongst the curves that were drawn in each of the cases of (edge,edge), (point, point), and (point,edge) pairings. The pieces of these curves that actually belong to the Voronoi roadmap can be determined by computing pairwise intersections of the curves. This naive method takes $O(n^4)$ time. By realizing that there are only $O(n)$ curves in the Voronoi roadmap, it is possible to compute the Voronoi roadmap in $O(n^2)$ time. The best complexity possible has been achieved in several, more involved, methods. The resulting computation time is $O(n \lg n)$.

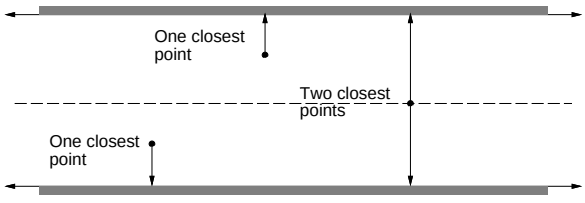


Figure 5.1: example.

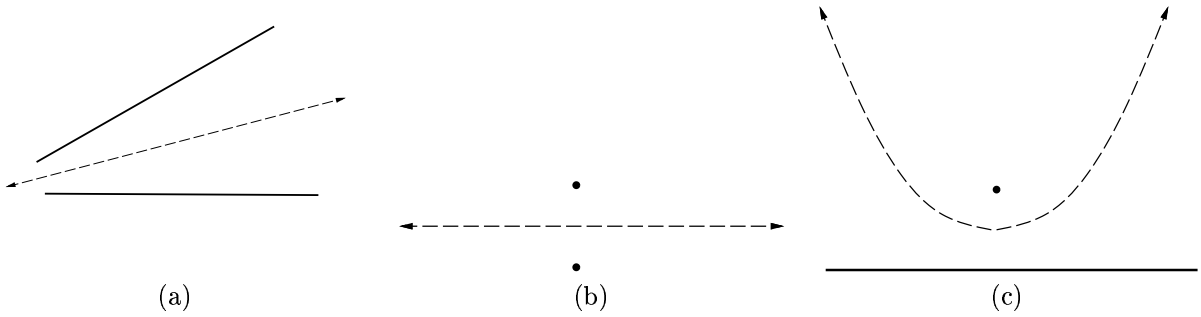


Figure 5.2: Voronoi roadmap pieces are generated in one of three possible cases: a) between two edges, b) between two points, and c) between a point and an edge. The third case leads to a quadratic curve.

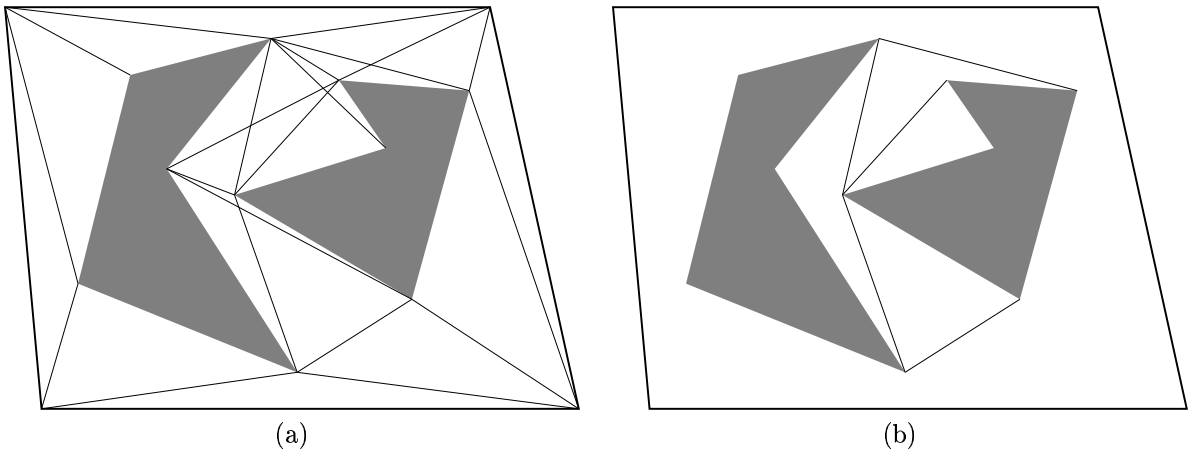


Figure 5.3: a) A visibility graph is constructed by joining obstacle region vertices; b) It is generally possible to reduce the number of edges in the visibility roadmap.

5.3 Visibility Roadmaps

A visibility roadmap is obtained by generating all line segments between pairs of obstacle region vertices. Any line segment that lies entirely in the free space is added to the roadmap. Figure 5.3 shows an example. When a path planning is given, the initial position and goal position are also treated as vertices (in other words, they are connected to other vertices, if possible). This generates a connectivity graph that can be searched for a solution. The naive approach to constructing this graph takes time $O(n^3)$. The sweep-line principle can be applied to yield a more efficient algorithm.

There are two important notes about visibility roadmaps:

- The shortest-path solutions found in the roadmap will actually be the shortest-path solutions for the original problem. In addition, the paths "touch" the obstacle region. This is not acceptable in terms of the original problem, and the resulting paths should be modified.

- Many edges can be removed from the visibility roadmap, and optimal solutions will still be obtained. Any edge can be removed if the following property holds: extend the edge in both direction by a small amount. If either end "pokes" into the obstacle region, then the edge can be removed. Figure **??**.b shows the edges that remain after this removal of performed.

5.4 Cell Decomposition

5.5 Canny's Algorithm

This method was not covered, but it is the most-efficient, complete algorithm known [**?**]. Although it is the most impressive algorithm theoretically, it is not been used in practice because of performance limitations and difficulty of implementation.

5.6 Probabilistic Roadmaps (PRMs)

Most path planning algorithms construct a representation of $\mathcal{C}_{free}$ in the form of a graph. Let $G(V, E)$ represent an undirected graph in which V is the set of vertices and E is the set of edges. Each vertex of V corresponds to a configuration in $\mathcal{C}_{free}$, and each edge in E corresponds to a collision-free path between a pair of configurations in $\mathcal{C}_{free}$.

There are typically two phases to path planning. The graph G is constructed in a *preprocessing phase*. In a *query phase*, G is used to solve a particular path planning question for a given q_{init} and q_{goal}.

The following algorithm builds a graph, G, in $\mathcal{C}_{free}$:

```
BUILD_ROADMAP(A, O)
 1  G.init();
 2  for i = 1 to M
 3      q ←RAND_FREE_CONF(q);
 4      G.add_vertex(q);
 5      for each v ∈ NBHD(q,G)
 6          if ((not G.same_component(q, v)) and CONNECT(q, v)) then
 7              G.add_edge(q, v);
 8  Return G;
```

The graph resulting graph will have M vertices. Each of the major components in described in more detail below. Sometimes it is preferable to replace the condition (**not** G.same_component(q, v)) with G.vertex_degree(q) $< K$, for some fixed K (e.g., K = 15). The same_component condition checks whether q and v are in the same connected component of G. If they are, then it is generally not necessary to attempt to connect q to v. However, it might be too costly to evaluate this condition each time, and one might prefer shorter paths. In this case, it would be appropriate to increase the density of edges in G. A limit K can be set on the maximum allowable degree for any vertex in G.

A random configuration in $\mathcal{C}_{free}$ is simply found by picking a configuration at random, and using a collision detection algorithm to test whether the configuration lies in $\mathcal{C}_{obs}$ or $\mathcal{C}_{free}$. If it lies in $\mathcal{C}_{obs}$, then a new random configurations are chosen, until one lies in $\mathcal{C}_{free}$. The running time of this algorithm will obviously be affected by the amount of freedom the robot has to move in the world. If most configurations are in collision, this algorithm will spend more time trying to find a free configuration. The algorithm is outlined below.

```
RAND_FREE_CONF()
 1  repeat
 2      q ←RAND_CONF();
 3  until q ∈ C_free;
 4  Return q;
```

The following algorithm returns an ordered list of vertices in $G(V, E)$, whose corresponding configurations are within a distance ϵ of q. The distance threshold, ϵ, and the metric $\rho(q, v)$ are selected in advance.

```
NBHD(q,G)
```

```
1  Q.init();
2  for each v ∈ V
3      if ρ(q, v) < ε then
4          Q.insert();
5  Return q;
```

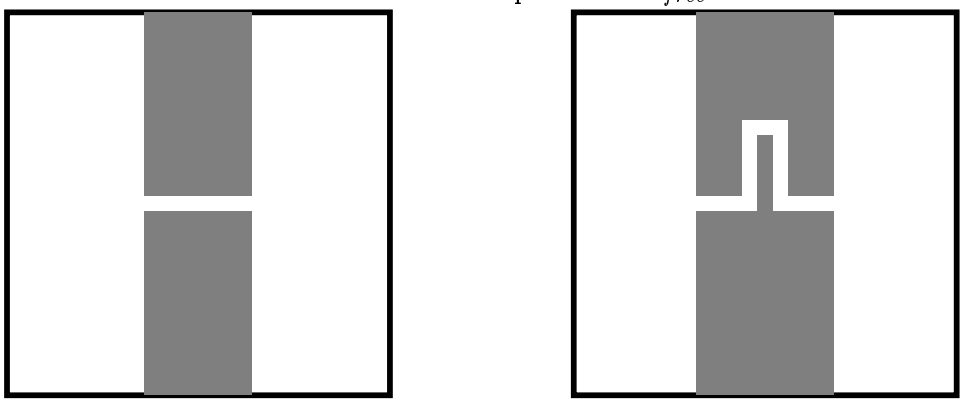
Is it assumed that Q is a priority queue, with elements sorted in increasing order according to $\rho(q, v)$. This ordering will cause the roadmap building algorithm to attempt connections to closer vertices first, before proceeding to more difficult connections.

The CONNECT(q, v) algorithm is considered conceptually as a *local* planner. The simplest form of a local planner is one that simply connects q to v by a "straight line" in $\mathcal{C}_{free}$. This line is defined by performing interpolation between q and v. If this line intersects $\mathcal{C}_{obs}$, then an edge cannot be made in G to connect q and v. One way to ensure that the entire line lies in $\mathcal{C}_{free}$ is to perform collision detection incrementally along the line from q to v. In each step, the robot is moved a small amount, and the configuration is checked for collision. An incremental collision detection algorithm would be suitable for this phase of the algorithm. Alternatively, it may be preferable to precompute a bitmap for fast collision detection. If the steps are made small enough so that no point on $\mathcal{A}$ is displaced by more than some $\delta_1 > 0$, then one can guarantee that a collision not occur in any intermediate configurations if the robot is at least δ_2 away from the obstacles in the world. The robot displacement metric can be used to determine δ_1, and the incremental collision detection algorithm can be used to determine δ_2.

In the query phase, an initial configuration, q_{init}, and a goal configuration, q_{goal}, are given. To perform path planning, the first step is to pretend as if q_{init} and q_{goal} were chosen randomly for connection to G. The NBHD and CONNECT algorithms can be used to attempt connection. If these succeed in connecting q_{init} and q_{goal} to other vertices in G, a graph search is performed for a path that connects the vertex q_{init} to the vertex q_{goal}. The path in the graph corresponds directly to a path in $\mathcal{C}_{free}$, which is a solution to the query. Unfortunately, if this method fails, it cannot be determined conclusively whether a solution exists. One approach could be to add more random vertices to G, and then try again to solve the path planning problem.

In general, the best one can assure is that the probabilistic roadmap planning algorithm in *probabilistically complete*. This is a fairly weak statement, which simply implies that the probability of finding a solution (if one exists) converges to one, while the running time approaches infinity. This seems to guarantee that a solution will be found, but this guarantee could only be made "after" running the algorithm forever. Thus, the algorithm cannot conclude that a solution does not exist after a finite period of time.

It is very challenging to analyze the performance of randomized path planning algorithms. Among these algorithms, the strongest analysis thus far exists for the randomized roadmap approach. The concept is based on the notion of ϵ-goodness. Consider cases in which the CONNECT algorithm will be unlikely to make a connection, even though a connection exists. The following examples are extremely difficult (especially the right one) because of the narrow corridor that links two portions of $\mathcal{C}_{free}$.



These are examples in a 2D configurations, and in higher dimensions, the problem can become even more difficult. Many planning problems involve moving a robot through an area with tight clearance. This will generally cause narrow channels to form in $\mathcal{C}_{free}$, which leads to a challenging planning problem for the randomized roadmap algorithm.

Let $S(q)$ denote the set of all configurations that can be connected to q using the CONNECT algorithm. Intuitively, this can be considered as the set of all configurations that can be "seen" using line-of-sight visibility. The ϵ-goodness is defined in terms of the following parameter,

$$\epsilon = \min_{q \in \mathcal{C}_{free}} \frac{\mu(S(q))}{\mu(\mathcal{C}_{free})},$$

in which μ represents the volume (or more technically, the measure). Intuitively, ϵ represents the small fraction of $\mathcal{C}_{free}$ that is visible from any point. In terms of ϵ and M (the number of vertices in G), a theorem has been established recently that yields the probability that a solution will be found. The main difficulties are that the ϵ-goodness concept is very conservative (it uses worst-case analysis over all configurations), and ϵ-goodness is defined in terms of the structure of $\mathcal{C}_{free}$, which cannot be computed efficiently. The result is interesting for gaining a better understanding of the algorithm, but the theorem is very difficult to use in a particular application to determine whether the algorithm will perform well.

5.7 Probabilistic Roadmap Variants

The following variants were covered:

- Node enhancement
- Visibility PRM
- Obstacle-based PRM
- Lazy PRM

These are covered in detail in papers that can be downloaded from the course web page.

Bibliography