Voronoi-Based Motion Planning Analysis

Bruno Carrillo

May 2025

ABSTRACT. This project investigates the application of Voronoi diagrams in robotic motion planning, focusing on the computational implications of environment complexity. Using a Python-based simulation, we explore how obstacle count, shape complexity, and geometric perturbations affect Voronoi diagram generation and path-finding performance using Dijkstra's algorithm. We compare the impact of two obstacle representations—vertices and sampled edge points—on the resulting road map structure and runtime. Our results show that increasing obstacle complexity and edge sampling significantly increase computation time while improving path quality. We also find that Voronoi-based paths are sensitive to small perturbations in obstacle geometry. These findings highlight key trade-offs in geometric planning and underscore the importance of careful site selection in Voronoi-based navigation systems.

Contents

1 Introduction

Motion planning is a core challenge in robotics, involving the computation of collision-free paths from a starting location to a goal location within an environment filled with obstacles. Efficient and reliable motion planning is critical in applications such as autonomous vehicles, robotic arms, drones, and warehouse automation. The primary goal is to generate paths that are both collision-free and optimal (shortest, safest, or fastest) while accounting for the environment's geometry and constraints.

This project is meant to explore the application of Voronoi diagrams as a computational geometry tool for motion planning. The focus is on analyzing how the structure of the environment, specifically the number and complexity of obstacles, affects the computation time and behavior of path-finding algorithms. We also examine how changes in obstacle representation influence the outcome, by slightly shifting the obstacles and comparing the use of obstacle vertices and sampled edge points.

2 Background

2.1 Robotic Motion Planning

Robotic motion planning refers to the process of determining a path or trajectory that a robot should follow to move from a starting location to a goal location without colliding with any obstacles. It involves both global planning (finding a path throughout the entire environment) and local planning (reacting to nearby changes). Motion planning algorithms often rely on geometrical and topological properties of the environment.

There are several kinds of motion planning algorithms, including:

- Grid-based methods (e.g. A*)
- Sampling-based methods (e.g. PRM, RRT)
- Road map-based methods (e.g. Visibility graphs, Voronoi diagrams)

Voronoi-based planning provides a structured road map that maximizes clearance from obstacles, making it especially useful in environments where safety margins are important.

2.2 Voronoi Diagrams

A Voronoi diagram is a partitioning of a plane into regions based on the distance to a specified set of points called sites. Each region corresponds to one site and consists of all points closer to the site than to any other. Formally, given a set of points $S = \{s_1, s_2, \ldots, s_n\}$, the Voronoi region $\text{Vor}(s_i)$ is defined as:

$$\text{Vor}(s_i) = \{p \in \mathbb{R}^2 | \text{dist}(p, s_j), \forall j \neq i\}$$

Voronoi diagrams are constructed by computing the perpendicular bisectors between each set of pair sites and using them to define region boundaries. The result is a network of edges that form a graph with useful properties:

- The edges lie equidistant from two or more sites.
- The paths formed by the Voronoi diagrams ensure that the distance between obstacles is maximized.

In this project, we use the Voronoi diagram to build a road map for motion planning. By filtering out edges that intersect obstacles, we construct a graph that guides the robot along collision-free and high-clearance paths.

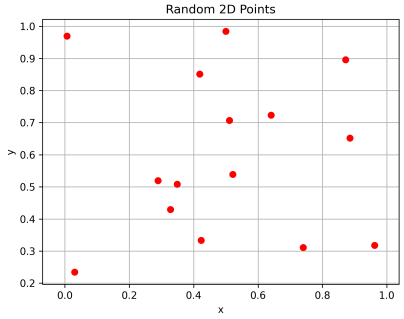

Figure 1: An example of randomly generated points

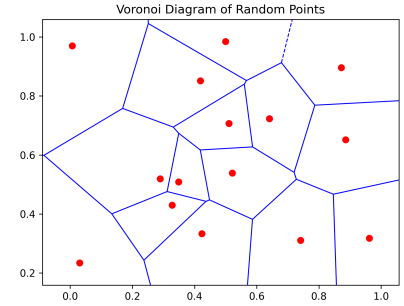

Figure 2: An example of a Voronoi diagram generated from random points

3 Methodology

We constructed a Python-based simulation using the following libraries:

- **Shapely** for geometric modeling of polygonal obstacles
- **Scipy.spatial** for generating Voronoi diagrams from point sites
- **NetworkX** for representing the Voronoi graph and running Dijkstra's algorithm
- **Matplotlib** for visualizations

Three key experimental questions guided the design:

1. How does computation time scale with the number of obstacles and their complexity?
2. How sensitive is the Voronoi diagram to small shifts in obstacle shape and position?
3. How does using obstacle vertices versus edge sample points affect computation time and path-finding behavior?

To answer these, we created randomized sets of non-overlapping polygonal obstacles with controlled complexity and count. Two sets of points were extracted from each: one using only vertices and one using densely sampled edge points. Voronoi diagrams were computed and filtered (to ensure no edges went through the interior of an obstacle), and Dijkstra's algorithm was run between a fixed start and a goal. This process was repeated 5 times per experiment to find the average runtime.

4 Assumptions

To simplify the scope and focus of this project, we make several assumptions about both the obstacles in the environment and the robot's capabilities. These assumptions are typical in foundational computational geometry applications and allow us to analyze core behaviors without involving complex real-world dynamics.

The obstacle assumptions are:

- All obstacles are assumed to be static and fully known in advance.
- The workspace is confined to a 2D plane.
- Obstacles are represented as simple, non-overlapping polygons.
- Obstacles are generated within a predefined bounding box, which serves as the robot's environment.

- The robot is treated as a point, so there is no explicit safety margin added to the obstacles for clearance.

The robot assumptions are:

- The robot is modeled as a single point, which greatly simplifies the computation and means collision checking only needs to ensure the path does not intersect obstacle interiors.
- It is assumed the robot can follow any planned path exactly, without error due to drift, sensor noise, or dynamic constraints.
- The start and goal positions are guaranteed to be placed in obstacle-free regions to avoid cases with ambiguous placements.
- The robot is not subject to turning radius, velocity, or acceleration constraints. This allows us to use Dijkstra's algorithm without making modifications.

5 Limitations

While the approach and implementation of this project offer valuable insight into Voronoi-based motion planning, several limitations should be noted. These reflect simplifications made to focus on core computational geometry aspects, but they also indicate opportunities for future improvement and extension.

One limitation is the simulated environment being static. All obstacles are assumed to be known and unmoving which excludes real-world scenarios where new obstacles can appear, move, or be partially unknown due to sensor limitations. The environment is also limited to 2D space. In real applications, 3D environments would introduce further complexity.

Another limitation is the simplified robot model. Since the robot is modeled as a point, this neglects size, shape, and motion constraints. In practice, most robots require a safety margin around obstacles and may not be able to make sharp turns or follow arbitrary paths.

6 Results and Analysis

6.1 Increasing Obstacle Count

As the number of obstacles increased (from 10 to 42), the computation time for Dijkstra's algorithm increased approximately linearly when using vertices. More obstacles lead to more sites, producing a denser Voronoi graph and increasing the number of nodes explored during path finding.

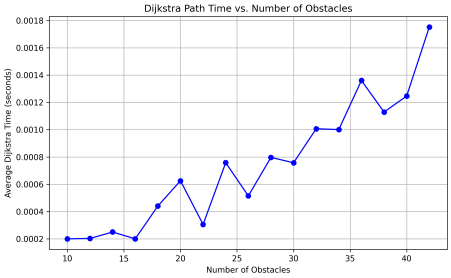

Figure 3: Graph comparing runtime to number of obstacles

One issue that arose when this experiment was running was the possibility that not every obstacle ended up being placed due to the potential of overlapping when more and more obstacles were generated. This becomes more apparent when reaching obstacle counts of around 40. This leads to the possibility of placing fewer obstacles than required being placed, which could lead to a lower runtime. Taking this into account, an overall increase in runtime can be observed as the obstacle count increases.

6.2 Increasing Obstacle Complexity

When increasing the polygonal complexity (3 to 24 sides), we observed that the number of Voronoi sites increased, and filtering edges became more computationally expensive. Dijkstra's runtime also increased due to a more irregular graph structure.

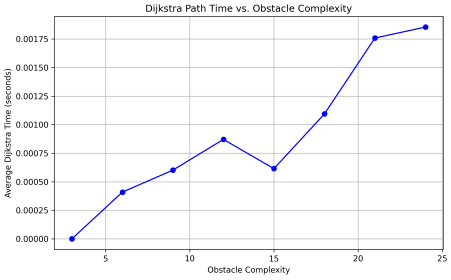

Figure 4: Graph comparing runtime to complexity of obstacles

To limit the probability that there was no path, the number of obstacles was limited to 20 obstacles. This was to try and get a more accurate average runtime for each level of complexity.

6.3 Slightly Shifting Obstacle Points

Small perturbations to obstacle vertices (random shifts of 0.1 units) produced noticeably different Voronoi diagrams. In many cases, paths changed shape or length, and some test cases failed to find valid paths due to topological changes in the graph. This highlights the sensitivity of Voronoi-based planning to minor geometric changes.

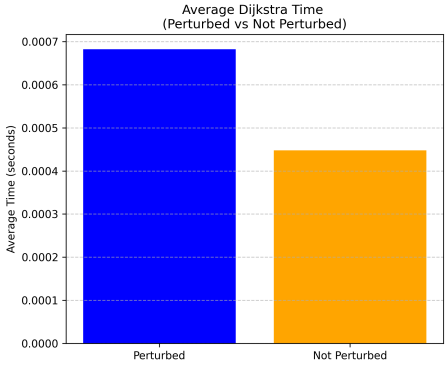

Figure 5: Graph comparing runtime of perturbed vertices to non-perturbed vertices

Runtimes between perturbed and non-perturbed environments were similar, but after running the same test a handful of times, we can see that the runtime of perturbed environments is, on average, higher than the runtime of non-perturbed environments.

6.4 Obstacle Vertices vs Edges

Using only vertices produced faster runtime but less complete Voronoi coverage, sometimes missing narrow paths. Sampling edges (e.g. 10 samples per unit length) improved path quality and robustness at the cost of higher computation time. Graph size and Dijkstra traversal time grew significantly with denser sampling.

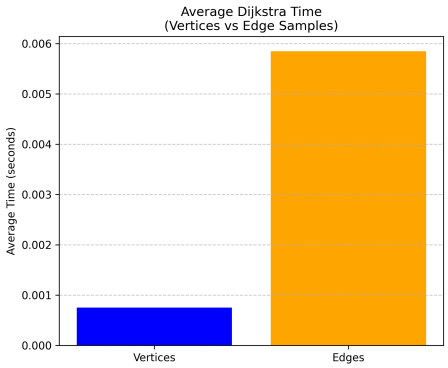

Figure 6: Graph comparing runtime of Voronoi diagrams built using vertices vs sampled edges

This larger runtime can most likely be attributed to an increase in the number of sites due to using 10 samples per edge instead of using the entire edge itself. This leads to more complex Voronoi diagrams which take longer to compute.

7 Conclusion and Future Work

This project demonstrates the effectiveness of Voronoi diagrams for motion planning in 2D environments and highlights the computational trade-offs involved. Denser representations provide more robust path-finding at the cost of longer runtime. Perturbation sensitivity suggests there are limitations in precision-dependent environments.

Future work could include modeling the robot as a polygon and using buffered obstacles, using true segment-based Voronoi diagrams through CGAL, or exploring dynamic environments. Additionally, these tests can be run using a different path-finding algorithm such as A* to compare the results to those using Dijkstra's.

8 Appendix

Tools and Parameters:

- Environment: Python 3.12.10, Jupyter Notebook
- Libraries: shapely, scipy, matplotlib, networkx
- Sampling density: 10 samples per unit edge length
- Perturbation magnitude: uniform random shifts in $[-0.1, 0.1]$
- Obstacles: Random non-overlapping polygons with 3-24 vertices
- Metrics: Runtime (in seconds), success rate, and path length

[Car25]

References

[Car25] Bruno Carrillo. *Voronoi-Based-Planned-Robotic-Motion*. Version 1.0.0. May 2025. URL: `https://github.com/brunocar004/Voronoi-Based-Planned-Robotic-Motion`.