On Path Planning using Point-based Minkowski Sum for Rigid Robots

Abhishek Ninad Kulkarni
Worcester Polytechnic Institute
Email: ankulkarni@wpi.edu

***Abstract*—In this paper, we address the problem of efficient computation of complete Configuration Space (C-space), for a rigid robot in an environment with polygonal (or polyhedral) obstacles. We introduce a new distributed $O(n)$ algorithm for the computation of Minkowski Sum (M-Sum) that only depends on the geometry of obstacle and not its dimensions. We achieve this by using the sampling theory to reduce the geometric and computational complexity of the problem and then proving that the output of the proposed algorithm approximates the true M-Sum within a finite error, which depends on the initial sampling density. This makes the use of *complete*, deterministic roadmap algorithms for solving the problem of path planning feasible in real-time. We prove the correctness of the proposed algorithm in this paper and provide two examples to illustrate how it works and to demonstrate how to solve the path planning problem for a mobile robot using this algorithm. An implementation of this algorithm can be found at https://github.com/abhibp1993/point-based-msum.git**

I. Introduction

Path-planning is a fundamental and well-studied problem in robotics. Several approaches have been proposed so far, ranging from purely geometric solutions to sampling based solutions. However, finding a *complete* algorithm suitable for solving the problem in real-time is still an open problem. We attempt to address this problem in the following paper.

The *complete* algorithms have a desirable property of finding a solution or reporting a failure within a finite amount of time. The deterministic roadmap algorithms like Visibility Graph, Generalized Voronoi Graph (GVG) are the examples of *complete* algorithms. However, it is noted that while these algorithms are computationally efficient, they require a complete representation of C-space, which is a computationally expensive operation. It is known that M-Sum computation grows exponentially as the dimensionality increases [] and so does the dependent *complete* path-planning algorithms[1].

On the contrary, the *probabilistically complete* algorithms like RRT* sample randomly in the C-space and perform collision checking in task-space. We note that the use of sampling theory makes these algorithms more computation friendly. This, however, results in loss of *completeness*, i.e. a *probabilistically complete* algorithm may take infinite time to converge to a solution, and may never terminate if it does not exist.

In this work, we integrate the aforementioned two observations about the *complete* algorithms and use of sampling theory in *probabilistically complete* algorithms into an algorithm that is *complete* as well as computationally efficient. We achieve this by sampling over the robot and obstacle boundaries in task-space to directly capture the collision information instead of sampling in C-space.

A. Literature Review

The configuration space (C-space) is a fundamental concept in motion planning. It represents the set of transformations that can be applied to robot. Therefore, by finding a continuous path from a start to goal in configuration space, we get the sequence of transformations that must be applied to move the robot to a desired configuration. The configurations of the robot that result in collision define the obstacle-space (C-obs). Note that C-obs is always contained in C-space.

a) Roadmaps: Given the complete C-space with explicit representation of C-obs, the roadmap algorithms like Visibility Map, Generalized Voronoi Diagram provide a simple and efficient way to find a path from start to goal configuration. It is known that these algorithms have polynomial runtime; for example, the visibility map algorithm can be implemented in $O(n^2 log(n))$ [2], and are known to be complete [3].

b) Sampling-based Motion Planning: Sampling based approaches avoid explicit construction of C-space and have been proved to be useful in many cases. The algorithms exploit the collision detection algorithms, which detect whether a given configuration is in collision or not by randomly sampling the C-space. If the sampled configuration results in collision, then it is ignored else it is used to construct a path to the goal.

Algorithms like RRT, Bi-RRT, RRT* are based on this approach. Most of these algorithms suffer from a major disadvantage of being probabilistically complete, i.e. they will eventually find a path, if it exists. However, they may run forever, if the path does not exist.

c) Minkowski Sum: Minkowski sum defines the transformation from the workspace to the configuration space. Several algorithms have been proposed over the past years for efficient computation of M-sum. If $\mathscr{P}$ and $\mathscr{A}$ are two N-dimensional polytopes with m and n edges respectively, then it is proved that the Minkowski sum can have a $O(m+n)$, $O(mn)$ or $O(m^2n^2)$ edges based on whether $\mathscr{P}$, $\mathscr{A}$ are convex or concave polyhedra, and in worst case can consume $O(m^3n^3)$-time. [2], [4], [5]. Furthermore, we note that these algorithms are not parallelization-friendly.

d) Point-based Minkowski Sum: To achieve faster computation, Lien [6] introduces the concept of sampling the boundaries of polyhedra, computing the M-sum and then

reconstructing the geometric boundary of the sum from point set. This results in a $O(mnT_{filter})$ algorithm, where the T_{filter} is the time complexity of processing a single sample point. This also exposes a parallelization-friendly structure, because each sample point can be processed independently. The work in this project is largely inspired by Lien's work.

The paper is further organized as follows. The next section II introduces the basic notation. The following section III formalizes the problem statement. The section IV explains the proposed algorithm and proves its correctness. Finally the section V presents two experiments to demonstrate the algorithm, and we conclude with some discussion about future directions in the section VI.

II. Preliminaries

As mentioned in the introduction, we sample the robot and obstacle in task-space. Therefore, in this paper, we consider the robot and obstacles that are either polygons or polyhedra. We note that the proposed algorithm and related theorems apply to any closed geometric shapes in 2D or 3D. Throughout the paper, we shall denote the polygonal (or polyhedral) robot by $\mathscr{A}$ and obstacle by $\mathscr{P}$. Further, we shall denote the geometric boundaries of $\mathscr{A}$ and $\mathscr{P}$ by $\partial\mathscr{A}$ and $\partial\mathscr{P}$ respectively.

We use equispaced sampling for generating the point-samples of robot and obstacle. This choice is motivated by the resultant simplifications in the following analysis.

Definition 1 (Equispaced Sampling in 2D)**.** *Let $f(x),\ x\in[0,1]$ represent a finite-length curve in 2D Cartesian space. Let $x_1,x_2,...,x_n$ be points satisfying $f(x_i)=0,\ i=1,2,...,n$. We say x_i to be equispaced samples of f, if and only if $\exists h>0$ such that $x_1=f(0)$ and $x_n=f(1)$ and $\forall i,\ x_{i+1}=x_i+h$. We denote the set of $x_1,x_2,...,x_n$ by $\mathfrak{S}(f)$.*

For a straight line in 2D, the set of n equispaced samples can be trivially by represented as $x_{i+1}=x_i+\frac{1}{n}$. For non-trivial polynomial curves, such samples may be generated using Chebyshev polynomial interpolation [7].

The boundaries of surfaces in 3D are 2D surfaces. Therefore, we slightly abuse the notation and define equispaced points on a surface in 3D Cartesian space over a triangulation of the surface. A triangulation is defined as the division of surface into a set of triangles (in 3D) such that each triangle side is entirely shared by two adjacent triangles [8]. It must be noted that only *compact* surfaces are triangulation-friendly, therefore, we will make this assumption in the following paper.

Definition 2 (Equispaced Sampling in 3D)**.** *Let $\triangle ABC$ be a triangle. Let the sides of $\triangle ABC$ satisfy the relation $l(\overline{AB})<l(\overline{BC})<l(\overline{CA})$. Let $\mathfrak{S}(\overline{AB})=x_i^{\overline{AB}}$ and $\mathfrak{S}(\overline{BC})=x_i^{\overline{BC}}$ for $i=1,2,...,n$ denote the equispaced samples of $\overline{AB}$ and $\overline{BC}$ respectively. Then the equispaced sample $\triangle ABC$ is defined as* $\mathfrak{S}(ABC)=\mathfrak{S}(\overline{AB})\cup\mathfrak{S}(\overline{AB})\cup\mathfrak{S}(\overline{AB})\cup(\bigcup_{i=1}^{n}\mathfrak{S}(\overline{x_i^{AB}x_i^{BC}}))$.

The cumulative equispaced sample of the surface is defined to be the union of equispaced samples of each triangle in its triangulation. It may be noted that above definition uses Basic Proportionality Theorem (BPT) to recursively generate the point-samples inside the triangle. Also, if the given surface to be sampled is a polygon in 3D, then Delaunay Triangulation provides an efficient algorithm for triangulation of surface [2].

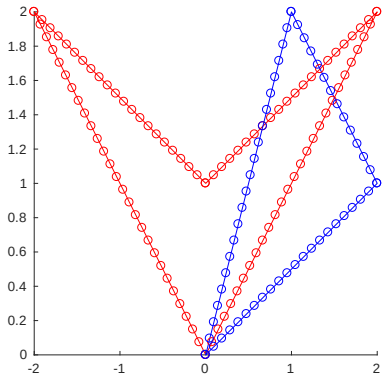

Fig. 1: δ-Sample of Triangle and Concave Polygon

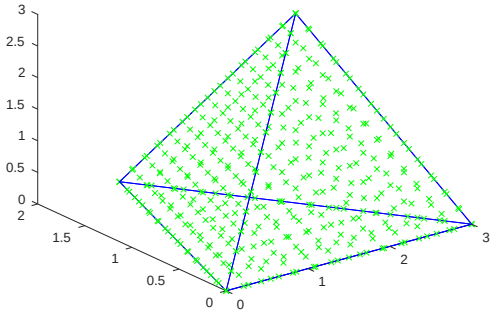

Fig. 2: δ-Sample of Tetrahedron

Finally, we define a δ-sample of a curve in 2D as its equispaced sample such that $h<2\delta$. Similarly, for a 3D surface, we call a equispaced sample as δ-sample if smallest side of each triangle in its triangulation is a δ-sample. See Fig. 1, 2.

Next, we define the notions of convex and concave sets in 2D, 3D. A closed set S in 2D or 3D Cartesian space is called convex if and only if for any arbitrary two points in S, the line joining the points lies entirely in the interior of S. A non-convex set with no self-intersecting edges is called concave set. Convex and concave polygons are convex sets with straight edges.

Given a set S of arbitrary n-points in a plane, the convex hull of S is defined as intersection of all closed half-planes that contain all points in S [9]. The concave-hull or α-hull is the generalization of convex hull, by introducing a parameter α. It is defined as follows.

Definition 3 (Equispaced Sampling in 3D)**.** *(Adopted from [9]) Let S be a set of n-points in a plane. Let $\alpha<0$ be a real number. The α-hull of S is defined as intersection of all closed*

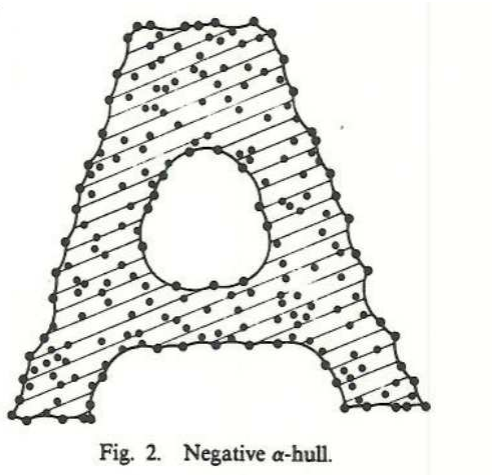

Fig. 3: Concave Hull of Set of Points (Adopted from [9])

complements of discs of radius $1/\alpha$ *that contain all points in S.*

Figure 3 shows a typical α-hull or concave-hull of a set of points. We also note that $\alpha = 0$ results in convex-hull of S. The idea of concave-hull will be used for reconstructing the geometric shape of M-Sum from the point sample.

We finally define the M-Sum operation over two sets $\mathscr{A}$ and $\mathscr{P}$ as

$$M = \mathscr{A} \oplus \mathscr{P} = \{\dashv + \sqrt{} \mid \dashv \in \mathscr{A} \ \dashv \backslash \lceil \sqrt{} \in \mathscr{P}\} \qquad (1)$$

where a and p are points contained in the sets $\mathscr{A}$ and $\mathscr{P}$ respectively.

We note that, for robot motion planning, the obstacle in C-space is represented by $-\mathscr{A} \oplus \mathscr{P}$ where $-\mathscr{A}$ is inversion operation of $\mathscr{A}$ about its origin. Based on whether $\mathscr{A}$ and $\mathscr{P}$ are convex or concave sets, we have the following results.

Theorem 1 (M-Sum of Convex Sets)**.** *Let $\mathscr{A}$ and $\mathscr{P}$ be two convex sets. Then, the M-Sum $\mathscr{M} = -\mathscr{A} \oplus \mathscr{P}$ is convex.*

Proof. The proof follows naturally by observing that an arbitrary set S is convex if it satisfies $S = tS + (1-t)S$, $for\, t \in [0,1]$. Therefore, we can write the M-Sum as $\mathscr{A} \oplus \mathscr{P} = \sqcup(\mathscr{A} \oplus \mathscr{P}) + (\infty - \sqcup)(\mathscr{A} \oplus \mathscr{P}) = (\sqcup\mathscr{A} + (\infty - \sqcup)\mathscr{A}) \oplus \sqcup\mathscr{P} + (\infty - \sqcup)\mathscr{P}$. □

For polygons or polyhedra, using the above theorem the M-Sum operation reduces to computing the vector sum of vertices.

With these definition, we next formulate the problem statement.

III. Problem Formulation

In this paper, we restrict ourselves to polygonal or polyhedral robots and obstacles for simplicity in analysis. However, it can be noted that the presented results naturally generalize to any *compact* surfaces in 2D or 3D.

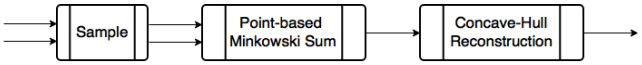

Fig. 4: Proposed Methodology for M-Sum

Let $\mathscr{A}$ and $\mathscr{P}$ represent polygonal (or polyhedral) robot and obstacle respectively. The sets $\mathscr{A}$ and $\mathscr{P}$ can be convex or concave. Then our objectives can be stated as follows:

- Use sampling theory effectively to introduce parallelism into the M-Sum computation.
- Propose a framework for efficiently computing M-Sum $\mathscr{M}$, given $\mathscr{A}$ and $\mathscr{P}$.
- Prove the equivalence of the proposed method with true M-Sum.

IV. Solution Approach

We propose a three step process as shown in 4. The sampling block inputs the robot and obstacle polygons (or polyhedra), $\mathscr{A}$ and $\mathscr{P}$, and outputs the boundary point-samples $\mathfrak{S}(\mathscr{A})$ and $\mathfrak{S}(\mathscr{P})$ respectively. The Distributed Minkowski Sum block computes the point-based M-Sum and returns a point-sample, $\mathfrak{S}(\mathscr{M})$. The final Reconstruction block reconstructs the geometric boundary represented by point-sample, $\tilde{\mathscr{M}}$. Note that, to prove the equivalence, we need to show that $\tilde{\mathscr{M}} \approx \mathscr{M}$.

We start by demonstrating how to reduce the computations required for M-Sum algorithm.

Lemma 1 (Sufficiency of Boundary)**.** *The boundary of the resultant M-Sum, $\mathfrak{S}(\mathscr{M})$, of $\mathscr{A}$ and $\mathscr{P}$ is defined only by the points in $\mathfrak{S}(\mathscr{A})$ and $\mathfrak{S}(\mathscr{P})$.*

Proof. The proof uses the concept of winding number. Using the argument as presented in [10], it can be shown that the winding number of point p with respect to the M-Sum, $\mathscr{M}$, is equal to product of individual polygon tracings, i.e. closed polygonal loops around the point.

Therefore, for every point in the interior of $\mathscr{A}$ or $\mathscr{P}$, the corresponding point in $\mathscr{M}$ will lie in the interior of the $\mathscr{M}$, and similar argument applies to every exterior point of $\mathscr{A}$ and $\mathscr{P}$. Therefore, every boundary point on $\mathscr{M}$ must be vector sum of a one point each from $\mathscr{A}$ and $\mathscr{P}$.

□

The Lemma 1 essentially reduces the problem of computing M-Sum for each interior point in $\mathscr{A}$ and $\mathscr{P}$ to computing the M-Sum for each point on the respective boundary sets, $\mathfrak{S}(\mathscr{A})$ and $\mathfrak{S}(\mathscr{P})$. Next, we show a relation between sampling density of $\mathfrak{S}(\mathscr{A})$ and $\mathfrak{S}(\mathscr{P})$ and the resultant density in $\mathfrak{S}(\mathscr{M})$.

Lemma 2 (M-Sum is δ-sample)**.** *Let $\mathfrak{S}(\mathscr{A})$ and $\mathfrak{S}(\mathscr{P})$ be two δ-samples. Then the M-Sum, $\mathfrak{S}(\mathscr{M})$, is also a δ-sample.*

Proof. We note from 1 that in the M-Sum of $\mathscr{A}$ and $\mathscr{P}$, every point on $\mathscr{A}$ boundary is summed with every point on boundary of $\mathscr{P}$. Consider the M-Sum of the edge-sample of $\mathscr{A}$ with a point from $\mathscr{P}$. It can be clearly seen that, for every point in

$\mathscr{M}$, which is sum of point from selected edge-sample and the point, there exists a point closer than 2δ distance from it. This above argument follows directly from definition of δ-sample and M-Sum. Hence the lemma holds true. □

Next we show that, it is possible to accurately reconstruct the boundary of a delta sample by choosing appropriate parameter, α, for concave hull construction.

Theorem 2 ($\alpha = \delta$)**.** *The boundary of a δ-sample can be reconstructed faithfully by using $\alpha = \delta$.*

Proof. The proof follows naturally by the definition of α-hull. □

Corollary 1 (Reconstruction Error is Bounded by δ)**.** *For any point p on δ-hull $\tilde{\mathscr{M}}$, the error from true M-Sum $\mathscr{M}$ is bounded by δ.*

Proof. By definition of α-hull, all points that lie on a disc of radius α or closer are included in reconstructed boundary. By Theorem 2, we have all points on δ-sample will lie on the reconstructed boundary of M-Sum. Therefore, the sampling error in the original polygons (or polyhedra) is unaffected during the M-Sum computation as per proposed method, and therefore is equal to δ. □

With Theorem 2, Corollary 1, we can now guarantee that the proposed method will compute the M-Sum boundary of given two polygons (or polyhedra), within a bounded error.

A. Complexity Analysis

We observe that use of sampling introduces parallelism into the solution and permits use of distributed computing. Given two point-samples of robot and obstacle $\mathfrak{S}(\mathscr{A})$ and $\mathfrak{S}(\mathscr{P})$, we observe that the M-Sum computation requires to compute vector sum of each point a in robot point-sample $\mathfrak{S}(\mathscr{A})$ with every point p in obstacle point sample $\mathfrak{S}(\mathscr{P})$. This appears as a nested loop structure. However, we observe for two points a_1 and a_2 in $\mathfrak{S}(\mathscr{A})$, $a_1 \oplus \mathfrak{S}(\mathscr{P})$ and $a_2 \oplus \mathfrak{S}(\mathscr{P})$ are independent of each other. Therefore, it possible to distribute this computation over independent threads. Therefore, the time-complexity of M-Sum is $O(N)$ where N is the number of points in $\mathfrak{S}(\mathscr{P})$.

The complexity of α-hull is known to be $O(N \log N)$ for 2D and $O(N^2)$ in higher dimensions [9].

V. Experiments

We illustrate the method with two examples. The first experiment demonstrates each step in proposed method, namely sampling, point-based M-Sum and reconstruction. The second experiment demonstrates a complete motion planning solution using the proposed method for a ground based mobile robot.

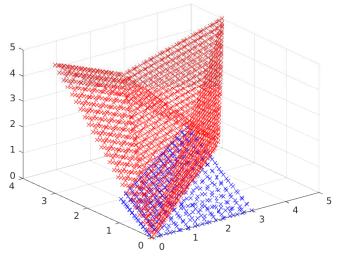

Fig. 5: Sample of Tetrahedral Robot and Concave 3D Obstacle

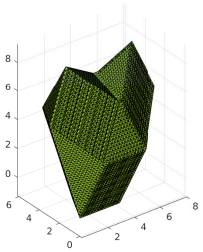

Fig. 6: Concave Hull of Generated M-Sum Point-Sample

A. Experiment 1: 3D Robot without Rotation

Consider a tetrahedral robot and obstacle given by

$$\mathscr{A} = \{(\prime,\prime,\prime),(\ni,\prime,\prime),(\infty,\in,\prime),(\in,\infty,\ni)\} \quad (2)$$

$$\mathscr{P} = \{\} \quad (3)$$

We choose tetrahedron for this illustration, as the facets are inherently triangles, thus, eliminating need for explicit triangulation of surfaces. The robot and its δ-sample is shown in Fig. 5.

Next, we compute the point-based M-Sum using MATLAB's Parallel Computing Toolbox with 4 threads. The output after computing α-hull with $\alpha = \delta$ on computed M-Sum point-sample is shown in the Fig 6.

B. Experiment 2: Mobile Robot Path Planning

In this section, we shall illustrate how the path planning problem can be solved in a *complete* manner using the proposed algorithm for point-based M-Sum computation.

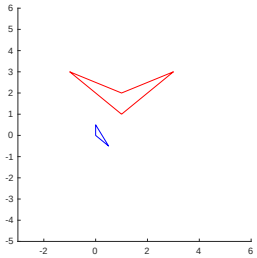

Fig. 7: Position of Mobile Robot and Obstacle

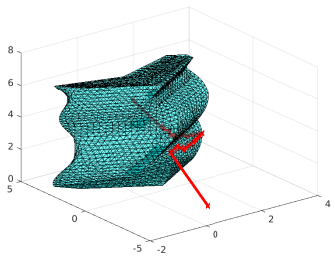

Fig. 8: Twisted Tower Representation of Configuration Space with Path shown in Red.

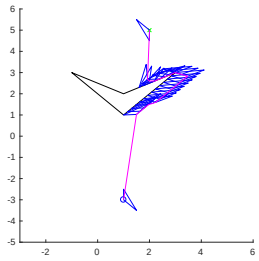

Fig. 9: Path in Task-Space

Consider a triangular robot and concave obstacle as shown in Fig. 7. The coordinates of robot are given by $\{(0,0),(0.5,-0.5),(0,0.5)\}$, while the obstacle coordinates are $\{(1,1),(3,3),(1,2),(-1,3)\}$.

To compute a path plan for the robot to go from configuration $(x,y,\theta) = (1,-3,0)$ to goal configuration $(x,y,\theta) = (2,5,\pi)$, we compute the M-Sum using the proposed algorithm.

In this problem, we use a bug-like algorithm where, in C-space, we try to take steps of predefined size towards the goal from given start configuration by moving along the line directly joining them. If we collide, i.e. the step leads to a point inside the concave hull of the computed M-Sum point-sample then we generate a graph using the points lying on boundary of concave hull of the M-Sum point-sample. The edges of this graph are defined by the edges of triangulation of the boundary points. We then find the path as shown in Fig. 9.

We note that it is possible to use deterministic and *complete* algorithms like cell decomposition or visibility graphs to find collision-free path in the given scenario.

VI. Discussion and Conclusion

We have proposed an algorithm for computation of M-Sum that supports parallelism and runs in $O(N)$-time. It may be noted that N grows with the increase in perimeter or surface area of obstacle, based on whether we are dealing with 2D or 3D space. This means, that the improvement in the time-complexity is achieved at the expense of memory resources. However, by use of shared memory in distributed computing this computation takes relatively less time and thus, may be possible to be used in a real-time application as demonstrated in Experiment V-B.

We have presented the proof-of-concept for this method in this paper. The Theorem 2 and Corollary 1 prove that the proposed algorithm can generate sufficiently close approximation of the M-Sum by choosing appropriate value of δ.

We further note that, in practical robotics application, we may not need to generate the point-sample for the obstacle. This sample is available from the point-cloud generated by proximity sensors like depth-cameras or LIDARs. Thus, it is possible to update the C-space directly as these points become available in real-time. Furthermore, for each new observed point, it takes $O(1)$-time to project it into the C-space, which make the proposed algorithm ideal for active sensing based planning approaches.

The future work for this project involves to design a complete motion planning framework, which uses the raw point-cloud data and returns plans the motion for robot in real-time and implement it on a real robot.

References

[1] G. Varadhan, S. Krishnan, T. Sriram, and D. Manocha, "A simple algorithm for complete motion planning of translating polyhedral robots," *The International Journal of Robotics Research*, vol. 25, no. 11, pp. 1049–1070, 2006.

[2] J. o'Rourke, *Computational geometry in C*. Cambridge university press, 1998.

[3] H. M. Choset, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[4] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM*, vol. 22, no. 10, pp. 560–570, Oct. 1979. [Online]. Available: http://doi.acm.org/10.1145/359156.359164

[5] D. Halperin, "Robust geometric computing in motion," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 219–232, 2002.

[6] J.-M. Lien, "Point-based minkowski sum boundary," in *Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference on*. IEEE, 2007, pp. 261–270.

[7] G. W. Stewart, *Afternotes on numerical analysis*. SIAM, 1996.

[8] "Wolfram math: Triangulation." [Online]. Available: http://mathworld.wolfram.com/Triangulation.html

[9] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel, "On the shape of a set of points in the plane," *IEEE Transactions on information theory*, vol. 29, no. 4, pp. 551–559, 1983.

[10] G. Ramkumar, "An algorithm to compute the minkowski sum outerface of two simple polygons," in *Proceedings of the twelfth annual symposium on Computational geometry*. ACM, 1996, pp. 234–241.

[11] F. Aurenhammer, "Voronoi diagramsa survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.

[12] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[13] Matlab - alphashapes. [Online]. Available: https://www.mathworks.com/help/matlab/ref/alphashape.html