Path Planning in Complex Environments with Superquadrics and Voronoi-Based Orientation

Lin Yang[1†], Ganesh Iyer[2†], Baichuan Lou[1], Sri Harsha Turlapati[1], Chen Lv[1], Domenico Campolo[1*]

Abstract—Path planning in narrow passages is a challenging problem in various applications. Traditional planning algorithms often struggle in complex environments such as mazes and traps, where navigating through narrow entrances requires precise orientation control. Additionally, perception errors can lead to collisions if the planned path is too close to obstacles. Maintaining a safe distance from obstacles can enhance navigation safety. In this work, we present a novel approach that combines superquadrics (SQ) representation and Voronoi diagrams to solve the narrow passage problem. The key contributions are: i) an efficient and generalizable path planning framework, integrating SQ with Voronoi diagrams to provide a computationally efficient and differentiable mathematical formulation, enabling flexible shape fitting for various robots and obstacles. ii) improved passage feasibility, achieved by expanding SQ's minor axis to eliminate impassable narrow gaps while maintaining feasible paths along the major axis, which aligns with the Voronoi hyperplane and follows maximum clearance paths. iii) enhanced robustness in complex environments, by integrating multiple connected obstacles into the same Voronoi region, handling trap scenarios. We validate our framework through a 2D object retrieval task and 3D drone simulation, demonstrating that our approach outperforms classical planners and a cutting-edge drone planner in various key aspects. The video can be found at `https://youtu.be/kbvQRxRACvQ`

Keywords— Voronoi diagrams, Superquadrics, Motion planning, Narrow passage.

I. INTRODUCTION

Path planning in complex environment such as narrow passages and traps poses challenges across various applications, including robotic manipulation [1], [2], mobile robotics [3], and drone navigation [4]. The task involves guiding a robot to navigate through a narrow passage without colliding with obstacles. The challenge arises from that not all robots or obstacles can be approximated as spheres or grid, though this is a common simplification in mobile robots. This approximation allows for easier handling of collision detection and path planning algorithms, as it reduces the complexity of the vehicle's geometry. However, in many cases, robots must find a path through the passage while maintaining a correct orientation. For example, as depicted in Fig. 1a, a manipulator attempts to retrieve an object from a crowded shelf. Some passages may be too narrow for the robot to traverse, while others are wide enough to accommodate the object's shorter axis length. However, due

to perception uncertainties or control inaccuracies, some of these trajectories may result in collisions with obstacles if they are too close to them. Similarly, in drone navigation, certain passages may require the drone to tilt at specific angles to pass through, while in other cases, the robot must recognize that certain passages are impassable, as illustrated in Fig.1b.
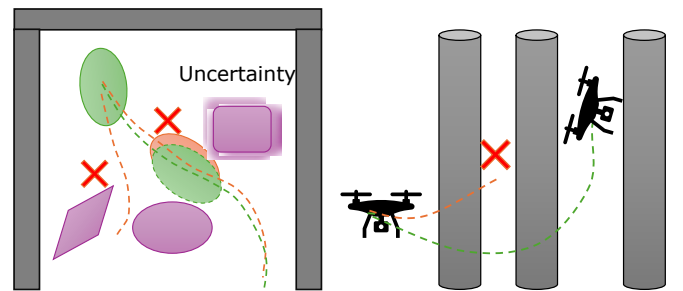
(a) Uncertainty in real-world with narrow passage: potential failure due to perception errors.

(b) 3D Navigation through narrow passages: a drone pass through narrow pillars.

Fig. 1: Navigating narrow passages in 2D and 3D: path feasibility, orientation decision, and safety problem due to uncertainty.

Many traditional path planning algorithms are popular and widely applied due to their favorable performance. However, some fall short when dealing with complex environments like mazes, traps, and narrow passages. For example, while sampling-based planning methods perform well in high-dimensional tasks, they struggle with efficiently sampling and growing a path through narrow passages [5]. Similarly, artificial potential field methods are widely used for obstacle avoidance [6], but they are prone to getting stuck in local optima. One effective approach for navigating narrow passages and mazes is the Voronoi diagram [7]. This method provides the safest solution for the planner by ensuring maximum clearance, which means the robot remains as far as possible from all obstacles. Additionally, some algorithms have already achieved near real-time efficiency in mobile robot path planning [8], [9] with the help of Voronoi diagram. However, these approaches lack pose control because circular mobile robots typically do not need orientation control in such environments.

The choice to approximate objects is crucial. A model that is too simplified (e.g., a sphere) may prevent the robot from passing through certain narrow passages. For instance, if the green ellipse in Fig. 1a is approximated as a sphere, the robot would lose the opportunity to retrieve the object.

[1] Authors are with the School of Mechanical and Aerospace Engineering, Nanyang Technological University (NTU), Singapore.

[2] Authors are with the School of Mechanical Engineering, Indian Institute of Technology, Bombay, India.

[†] These authors contributed equally to this work

[*] Corresponding author: `d.campolo@ntu.edu.sg`

On the other hand, using an accurate model can result in computational overhead during collision detection. Orthey et al. [10] successfully tackled narrow passage problems by employing an efficient yet complex approach. Their method, while powerful, requires numerous patterns and a detailed CAD model of the object for collision detection.

In contrast, we prioritize a balance between simplification and efficiency by representing objects using superquadrics (SQ) [11], which provide both differentiability and a convenient parametrization. SQ can describe a wide range of geometric shapes. For example, Ruan et al. [12] address narrow passage problems using SQ to represent both robots and obstacles, employing Minkowski sums to avoid collisions across various robot orientations. Their approach achieved effective planning for self-collision avoidance. Zhao et al. [13] simplify the Minkowski sum via utilizing the formulation of SQ. For now, our task is simpler than [12] and thus, we do not require extensive computation with Minkowski sums. Additionally, we avoid over-simplification, as many researchers use Minkowski sums to grow obstacle sizes by sweeping the maximum size of the robot, ensuring the remaining map is navigable with any robot pose.

In this work, we propose an efficient approach for addressing the narrow passage problem in both 2D and 3D scenarios by integrating SQ representation with Voronoi diagrams. Our method offers three key advantages:

i) **An efficient path planning framework:** SQs can model various geometries, ensuring adaptability to various shapes while maintaining differentiability. Parameterizing SQs in polar coordinates enables efficient hyperplane computation and Voronoi diagram generation.

ii) **Improved passage feasibility and orientation guidance:** Expanding the SQ's minor axis eliminates impassable gaps while preserving feasible paths. Aligning with Voronoi hyperplanes ensures maximum clearance paths and naturally guides the robot's orientation.

iii) **Enhanced robustness in complex environments:** Segmenting connected obstacles into the same Voronoi region prevents the robot from getting trapped. Validated in 2D and 3D scenarios, our method successfully identifies the correct passage and orientation while avoiding traps. In 3D simulations, it outperforms a cutting-edge drone planner, finding passable paths in a narrow-passage world.

II. SUPERQUADRICS AND VORONOI DIAGRAM

In this section, we present the mathematical definitions of Superquadrics (SQ) (known as Superellipses in 2D) and the Voronoi diagram. Superquadrics form a family of three-dimensional shapes that generalize basic quadric surfaces, such as ellipsoids, elliptic paraboloids, and hyperbolic paraboloids. These surfaces are generated by the spherical product of two parametric two-dimensional curves [11].

A. Superellipses and superquadrics

Superquadrics are widely utilized to represent geometry of objects in real world [14]–[16]. The 2D analogue of

superquadrics are superellipses,

1) Superellipses: Superellipses is implicitly defined via equation:

$$\left(\frac{x}{a_1}\right)^{\frac{2}{\epsilon}} + \left(\frac{y}{a_2}\right)^{\frac{2}{\epsilon}} = 1 \tag{1}$$

Here, ϵ represents the exponent of the superellipse, which controls the shape of the superellipse. a_1, a_2 control the size. Meanwhile, define a inside-outside function $F(x, y)$ as

$$F(x, y) = \left(\frac{x}{a_1}\right)^{\frac{2}{\epsilon}} + \left(\frac{y}{a_2}\right)^{\frac{2}{\epsilon}} - 1 \tag{2}$$

The function $F(x, y)$ possesses a very useful property. For a point (x_0, y_0), if $F(x_0, y_0) > 0$, then it is outside the SQ, if $F(x_0, y_0) = 0$ then it is on the surface, and if $F(x_0, y_0) < 0$, then it is inside. For this reason, the function F is also referred to as the **inside-outside** function.

Meanwhile, analogous to a circle, superellipse can be parameterized as [11],

$$\mathbf{r}(\omega) = \begin{bmatrix} a_1 \cos^\epsilon \omega \\ a_2 \sin^\epsilon \omega \end{bmatrix}, \ \omega \in [-\pi, \pi] \tag{3a}$$

The exponentiation with power ϵ here is the signed power function, i.e

$$\cos^\epsilon \omega = \text{sign}(\cos \omega) \cdot |\cos \omega|^\epsilon \tag{4}$$

2) Superquadrics: In 3D case, the implicit function of superquadrics is defined as,

$$\left(\left(\frac{x}{a_1}\right)^{2/\epsilon_2} + \left(\frac{y}{a_2}\right)^{2/\epsilon_2}\right)^{\epsilon_2/\epsilon_1} + \left(\frac{z}{a_3}\right)^{2/\epsilon_1} = 1 \tag{5}$$

So we have a similar inside-outside function $F(x, y, z)$ (analogy to Eq. 2 for 3D case)

$$F = \left(\left(\frac{x}{a_1}\right)^{2/\epsilon_2} + \left(\frac{y}{a_2}\right)^{2/\epsilon_2}\right)^{\epsilon_2/\epsilon_1} + \left(\frac{z}{a_3}\right)^{2/\epsilon_1} - 1 \tag{6}$$

$F(x, y, z)$ is still **inside-outside** function in 3D case. Meanwhile, Eq. 5 can be parameterized by

$$\mathbf{r}(\eta, \omega) = \begin{bmatrix} a_1 cos^{\epsilon_1}\eta \ cos^{\epsilon_2}\omega \\ a_2 cos^{\epsilon_1}\eta \ sin^{\epsilon_2}\omega \\ a_3 sin^{\epsilon_1}\eta \end{bmatrix},$$
$$\eta \in [-\pi/2, \pi/2], \ \omega \in [-\pi, \pi] \tag{7a}$$

Here, (a_1, a_2, a_3) are the size parameters, while (ϵ_1, ϵ_2) are the shape parameters. Thus, in local coordinates of the superquadric, 5 parameters are needed to define it completely.

3) Superellipses and Superquadrics in general position: To accommodate any orientation and position, translation and rotation operations must be applied to the superellipses and superquadrics. Transformation matrices are required to represent superellipses and superquadrics in the world frame. Therefore, to fully describe a SQ, the parameter set is defined as $SQ := [\epsilon, \mathbf{a}, \mathbf{p}, \boldsymbol{\theta}]^T$, where $\mathbf{T}(\mathbf{p}, \boldsymbol{\theta}) = [\mathbf{R}(\boldsymbol{\theta}) \in SO(3), \mathbf{p} \in \mathbb{R}^3] \in SE(3)$ is the transformation matrix.

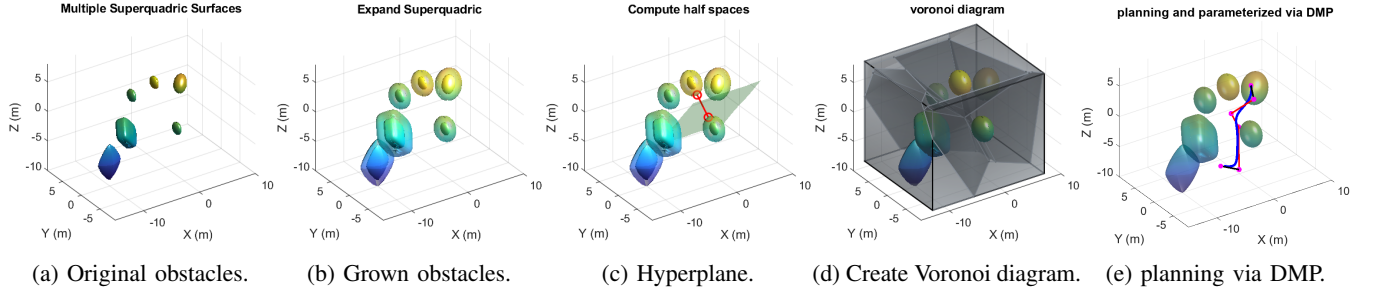| (a) Original obstacles. | (b) Grown obstacles. | (c) Hyperplane. | (d) Create Voronoi diagram. | (e) planning via DMP. |

Fig. 2: An entire process of our algorithm: a): approximate the environment by SQs. b): Grown obstacles with shortest axis length of robot. c): Hyperplane between two SQs and closest point. d): Create Voronoi diagram with hyperplane. e): Utilize graph planning to find edges, then parameterized by DMP.

B. Voronoi Diagrams

Fundamental data structures play a pivotal role in computational geometry, enabling efficient algorithms for solving various geometric problems. One such essential structure is the Voronoi diagram, which partitions the Euclidean space $W \in \mathbb{R}^n$ into m half spaces $\nu = \{V_1, V_2, \ldots, V_m\}$, often referred to as cells or regions,

Each cell is defined such that every point within it is closest to the corresponding disjoint convex obstacles $\mathcal{O} = \{O_1, O_2, \ldots, O_m\}$ (i.e., $d(O_i, O_j) > 0 \; \forall i \neq j$), based on maximum margin separating hyperplane, to be

$$HP_{i,j} := \left\{ q \in W \mid \|q - p_i^*\| = \|q - p_j^*\| \right\}, \quad (8a)$$

$$V_i := \left\{ q \in W \mid \|q - p_i^*\| \leq \|q - p_j^*\| \right\}, \quad (8b)$$

$$(p_i^*, p_j^*) = \underset{p_i \in \mathcal{O}_i, p_j \in \mathcal{O}_j}{\arg\min} \; d(p_i, p_j) \quad \forall j \neq i \quad (8c)$$

where $HP_{i,j}$ denotes the hyperplane (or boundary) shared by two Voronoi regions V_i and V_j. Traveling along this hyperplane ensures that the path remains equidistant from the obstacles associated with V_i and V_j, thus providing a locally maximal clearance from these obstacles [8].

III. ROBOT NAVIGATION IN GENERALIZED WORLD

Many objects can be approximated by one or several SQs [14]. Unlike many approaches that assume all obstacles are disjoint and that the distance between obstacles is greater than the robot's maximum diameter [8], our SQ-based representation allows for a more generalized and realistic world.

A. Eliminate impossible passage by expanding

In robot planning, mobile robots [8] and drones [17] are commonly approximated by a sphere, after which collision checks or Minkowski sums are applied to all obstacles.

However, this method restricts the ability of non-spherical robots, such as a truck or a peg-like object, from passing through narrow passages. Therefore, in our approach, the first step is to approximate the robot, object, or drone using a Superquadric, represented as $SQ_r = [\boldsymbol{\epsilon}_r, \mathbf{a}_r, \mathbf{p}_r, \boldsymbol{\theta}_r]^T$, while the other obstacles are represented as $SQ_o = [\boldsymbol{\epsilon}_o, \mathbf{a}_o, \mathbf{p}_o, \boldsymbol{\theta}_o]^T$. For example, approximate some obstacles like Fig. 2a.

We arrange the components of a_i in ascending order, denoted as $\mathbf{a} = [a_1, a_2, a_3], a_i < a_{i+1}$ so $a_{r,1}$ is the shortest

axis of SQ_r. We expand this shortest axis to evaluate whether the robot can navigate through a narrow passage.

While the original concept of Voronoi-based planning is effective for point robots, it cannot be directly applied to robots with extended bodies. The Minkowski sum, a fundamental operation in computational geometry and robotics, is commonly used to handle this situation. Given two sets A and B in a vector space, the Minkowski sum $A+B$ is defined as the set of all points that can be expressed as the sum of a point from A and a point from B: $A+B = \{a+b \mid a \in A, b \in B\}$. Instead of applying the Minkowski sum directly, we leverage the properties of Superquadrics to achieve similar results by expanding $\mathbf{a}_o$ to achieve similar performance of Minkowski sum. This expansion yields an adjusted obstacle, represented SQ_o^M, where the size parameters are modified accordingly to reflect the expanded configuration.

$$\mathbf{a}^M = \mathbf{a} + a_{r,1}\mathbf{I} \quad (9)$$

where $\mathbf{I}$ denotes the identity matrix. Now the robot is reduced to a point, as shown in Fig. 2b, on which we can apply standard voronoi diagram.

B. Generate Voronoi diagram

One advantage of using SQ is that they can be parameterized by angular coordinates, making them continuous and differentiable functions. This property allows us to efficiently compute the minimum distance between SQ pairs using classical optimization algorithms (e.g., Newton's method) to identify the closest points between each SQ pair. As illustrated in Fig. 2c, two obstacles are represented by two SQs. The red circles indicate the points (p_i^*, p_j^*), where the distance $d(p_i, p_j)$ is minimized (Eq. 8), and the green plane represents the hyperplane that separates the two regions.

Meanwhile, we evaluate whether $F_i(p_j^*) \leq 0$. If this condition holds, the two SQs overlap, and we assign them to the same cluster C_i. Otherwise, we separate two SQ as two clusters C_i and C_j. The logic is implemented in Alg. 1.

After separating the clusters based on the overlapping of SQs, we proceed to compute the elements in Eq. 8 to create the diagram. For any the hyperplane $HP_{i,j}$ between SQ i and SQ j can be expressed at the center of the $\frac{p_i^* + p_j^*}{2}$. Meanwhile,

Algorithm 1 Generate cluster

Input: Obstacles after mink sum $\mathbf{S}Q_o^M$, the world space W.
1: **procedure** CREATE CLUSTER($\mathbf{S}Q_o^M, W$)
2:     Cluster $C \leftarrow \emptyset$, cluster index $id = 1$.
3:     **for** $i = 1, \ldots, N$ **do**
4:        **if** $SQ_{o,i}^M \notin C$ **then**
5:           $id \leftarrow id + 1$
6:           $C_{id} \leftarrow \emptyset$
7:        **for** $j = i, \ldots, N$ **do**
8:           $(p_i^*, p_j^*) \leftarrow \underset{p_i \in SQ_{o,i}^M, p_j \in SQ_{o,j}^M}{\arg\min} d(p_i, p_j),$
9:           **if** $F_i(p_j^*) \leq 0$ **then**
10:              $C_i \leftarrow C_i \cup SQ_{o,i}^M$
11:        $C \leftarrow C \cup C_i$
12:     **return** C

Algorithm 2 Generate Voronoi diagram

Input: Cluster $C = [C_1, \ldots, C_K]$, the world space W.
1: **procedure** CREATE VORONOI(C, W)
2:     **for** $i = 1, \ldots, K$ **do**
3:        $C_i = C(i)$
4:        **for** $j = i, \ldots, K$ **do**
5:           $C_j = C(j)$
6:           **if** $i \neq j$ **then**
7:              $(p_i^*, p_j^*) \leftarrow \underset{\mathbf{p}_i, \mathbf{p}_j}{\arg\min} d(\mathbf{p}_i, \mathbf{p}_j),$ where
                 $\mathbf{p}_i \in SQ_{o,i}^M \in C_i, \ \mathbf{p}_j \in SQ_{o,j}^M \in C_j$
8:        $HP_{i,j} \leftarrow$ Eq. 12.
9:        $V_i = \bigcup_{j=0}^m W \cap HP_{i,j}$ Eq. 13.
10:     **return** V
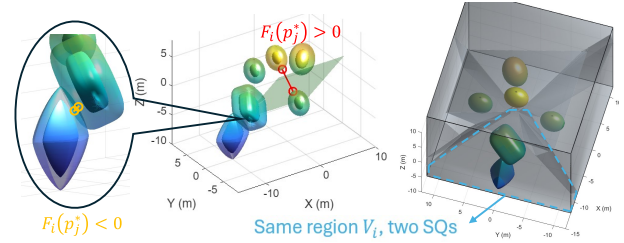

Fig. 3: Clusters are partitioned using Alg. 1, and adjacent SQs are grouped into the same region as Alg. 2.

define the normalize function $\hat{n}(x)$ as

$$\hat{n}(\mathbf{x}) = \frac{\mathbf{x}}{\|\mathbf{x}\|} \tag{10}$$

Thus, the normal vector of $HP_{i,j}$ equals,

$$\mathbf{n}_{i,j} = \hat{n}(\mathbf{p}_i^* - \mathbf{p}_j^*) \tag{11}$$

Following the definition of hyperplane, we get

$$HP_{i,j} = \{x | \mathbf{n}_{i,j}^T x \leq \mathbf{n}_{i,j}^T (\frac{\mathbf{p}_i^* + \mathbf{p}_j^*}{2})\} \tag{12}$$

A property of voronoi diagrams is that, for each object SQ_o, the intersection of all the spaces created by the bisector hyperplanes that contain the object forms the polyhedral cell corresponding to that object. The collection of all such polyhedral cells constitutes the complete Voronoi diagram. Initially, We define a bounding box (world space W) to appropriately represent the workspace. Consequently, the region V_i corresponding to each cluster is represented as the intersection of all the hyperplanes that define its polyhedral cell within this bounding box.

$$V_i = \bigcup_{j=0}^m W \cap HP_{i,j}. \tag{13}$$

The result of Eq. 13 is shown as Fig.2d. We observe most obstacles belong to one region, while there are two overlap obstacles belongs to same region. The steps to generate a Voronoi diagram with cluster is shown in Alg. 2.

The diagrams illustrating the performance of Alg. 1 and Alg. 2 are shown as Fig. 3.

C. Generate graph via Voronoi diagram

We create a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each $\mathcal{V}_i$ corresponds to a polygon with multiple vertices and edges. Each vertex of the polygon is represented as a node v_i in a graph $\mathcal{V}$, where nodes are connected by edge on polygon $\mathcal{E}$. The weight assigned to each edge corresponds to the Euclidean distance between the connected nodes.

We notice that there are certain regions (holes) in the diagram that do not belong to any cell V_i. This occurs due to

the linear approximation used in constructing the diagram, but it has few impact since our objective is to identify safe paths along the roadmap. Thus, we connect nodes where the distance between them is smaller than a predefined threshold h, which is determined based on the overall size of the workspace. In other words, this is used to connect the nodes near the hole to maintain a practical and navigable graph structure.

$$\mathcal{E} = \mathcal{E} \cup \{v_i, v_j\}, \ \forall v_i, v_j \in V, \|v_i - v_j\| \leq h \tag{14}$$

Given any initial pose and target pose, we first project these poses onto their respective closest edges $\mathcal{E}$. Once projected, we determine the shortest path through the graph by classical graph planning algorithm (e.g., Dijkstra) to find solution path via nodes $\mathcal{V}_{sol} = [v_1, v_2, \ldots, v_i]^T$.

D. Planning with orientation

In subsection A, we determined the longest axis of the robot SQ_r. To control the robot's orientation while navigating through narrow passages, the robot should move along the direction of its longest axis. This direction is computed based on the nodes in the path solution, $\mathcal{V}_{sol}$, which provides the optimal orientation for each step of the path.

1) 2D: In the 2D case, we plan the robot's motion using $\mathbf{T}(\mathbf{p}, \boldsymbol{\theta}) \in SE(2)$, where $\boldsymbol{\theta} \in \mathfrak{so}(2)$. For each edge in the path, the direction is determined by the orientation of the robot's longest axis relative to the edge,

$$\Delta v := [\Delta v_x, \Delta v_y]^T = v_{i+1} - v_i \in \mathbb{R}^2 \tag{15a}$$

$$\boldsymbol{\theta} = \arctan(\Delta v_y, \Delta v_x) \tag{15b}$$

2) 3D: In 3D case, we are planning for $\mathbf{T}(\mathbf{p}, \boldsymbol{\theta}) \in SE(3)$, where $\boldsymbol{\theta} \in \mathfrak{so}(3)$. Since we are navigating through nodes located on the surfaces of polyhedra, these surfaces provide the maximum clearance orientation for the robot. To ensure efficient navigation, the robot's (SQ_r) longest axis is aligned forward along the direction of travel, while the shortest axis is oriented perpendicular (normal) to the halfplane $HP_{i,j}$. This configuration maximizes the robot's ability to fit through narrow passages and maintain optimal clearance. The rotation vector required to adjust the robot's orientation is computed based on the vertices of the surfaces, ensuring that the robot aligns itself as it moves from node to node on the surface.

$$
\begin{aligned}
\mathbf{r}_1 &= \hat{n}(v_{i+1} - v_i) \\
\mathbf{r}_3 &= \mathbf{n}_{i,j}, \; v_{i+1}, v_i \in HP_{i,j} \\
\mathbf{r}_2 &= \mathbf{r}_3 \times \mathbf{r}_1 \\
\mathbf{R} &= [\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3]
\end{aligned}
\tag{16}
$$

After determining the rotation matrix at each edge, we compute the corresponding Lie algebra $\boldsymbol{\theta} \in \mathfrak{so}(3)$. This is done by applying the matrix logarithm $\log$ to the rotation matrix, which gives a skew-symmetric matrix, then $\vee$ represents extract skew-symmetric matrix to vector.

$$
\boldsymbol{\theta} = \log_{\vee} \mathbf{R} \tag{17}
$$

E. Filtering unsafe turning points

When the robot navigates through narrow passages, sudden orientation changes at turning points may lead to unintended collisions due to the extension of its major axis $a_{r,3}$. To address this, we apply a conservative check to filter out unsafe turning nodes in the path planning graph.

For each candidate turning point v_i, we define a circular safety region whose diameter equals the length of the robot's major axis $a_{r,3}$. If this circle overlaps any obstacle SQ_o, the point is removed from the planning graph. This simple yet effective improves the robustness of the planner by accounting for possible collisions during orientation adjustments, which are not fully captured by purely axis-aligned Voronoi alignment.

F. Smooth Interpolation Using DMP

Now that we have the orientation at each edge and the nodes as via points, we need a smooth trajectory for the robot to follow. First, we interpolate all the nodes using polynomials to obtain the trajectory $\mathbf{p}(t)$. However, this trajectory may still appear zigzagged, and it is not necessary to pass through every node, especially in areas with many redundant nodes (e.g., holes in the diagram). To address this, we employ DMP to learn a smooth trajectory. DMP is a well-established technique for parameterizing and smoothing point trajectories [18], and it has also been extended to manipulation planning scenarios where object geometry are considered [2], [15], [19]. Classical DMP can be used to map a finite dimensional set of parameters $\mathbf{W} \in \mathbb{R}^{K \times P}$ where P

is the number of Radial Basis Functions (RBFs) per degree-of-freedom (DOF) into smooth and differentiable functions.

$$
\text{DMP} : (\mathbf{W}, \mathbf{u}_{start}, \mathbf{u}_{goal}, T) \mapsto \mathbf{u}_{\mathbf{W}}(t), \quad t \in [0, T]. \tag{18}
$$

where $\mathbf{u}_{\mathbf{W}}(t)$ represents the desired pose. We first augment position and lie algebra, so we have dataset $\mathbf{D} = [\mathbf{p}(t), \boldsymbol{\theta}(t)], t \in [0, T]$, then we apply locally weighted regression (LWR) [18] to learn parameter $\mathbf{W}$ from dataset $\mathbf{D}$, which is used to generate the new trajectory.

$$
\mathbf{W} = \text{LWR}(\mathbf{D}) \tag{19}
$$

The resulting trajectory is shown in Fig. 2e, where the blue line represents the original edges of the solution path, and the pink trajectory generated by DMP shows the smoothed path. In practice, we choose the number P of RBFs sufficiently large to ensure that the generated continuous trajectory closely follows the original discrete Voronoi path, while maintaining smoothness for safe execution.

IV. EXPERIMENT: MOTION PLANNING IN CLUTTERED 2D ENVIRONMENTS

Retrieving a target object from clutter is a common task in daily life (e.g., from shelves). However, due to height restrictions, retrieving objects from the top is not feasible, making it challenging to find a collision-free path for retrieval [20]. We designed a toy experiment to simulate a similar scenario and test the performance of our algorithm in a 2D environment.

A. Experiment design and results

The process of this experiment is illustrated in Fig. 4. We set up a table with several objects, some of which are placed to create narrow passages that too difficult to pass through. A Kinova robot equipped with a depth camera is positioned on the table, and the camera is calibrated. The robot raises to its highest position while keeping its gaze downwards, and images are captured.

We segment the objects using the SAM toolbox in MATLAB package [21], from which we obtain point clouds representing the boundaries of all obstacles. Superellipse fitting is then applied to these point clouds to obtain the obstacle representation [14]. Next, we applied our framework with the help of bensolve toolbox [22], and one result is shown at Fig. 4. It is called narrow passage scenario since some passages between obstacles are impossible to pass through. We manually select an object SQ_r for retrieval. The robot moves to $\mathbf{p}_r, \boldsymbol{\theta}_r$ to grasp the object. Subsequently in the right bottom figure in Fig. 4, the size of other obstacles are expanded by $a_{r,1}$, which eliminates impossible passages. After expansion, objects close to each other overlap, indicating that they belong to the same Voronoi region V_i (Eq. 13). Planning with orientation ensures that the long axis of the object aligns with the edge in Voronoi diagram (Eq. 15). The final trajectory is sent to the robot, which executes the trajectory, allowing the object to safely pass through the cluttered environment.
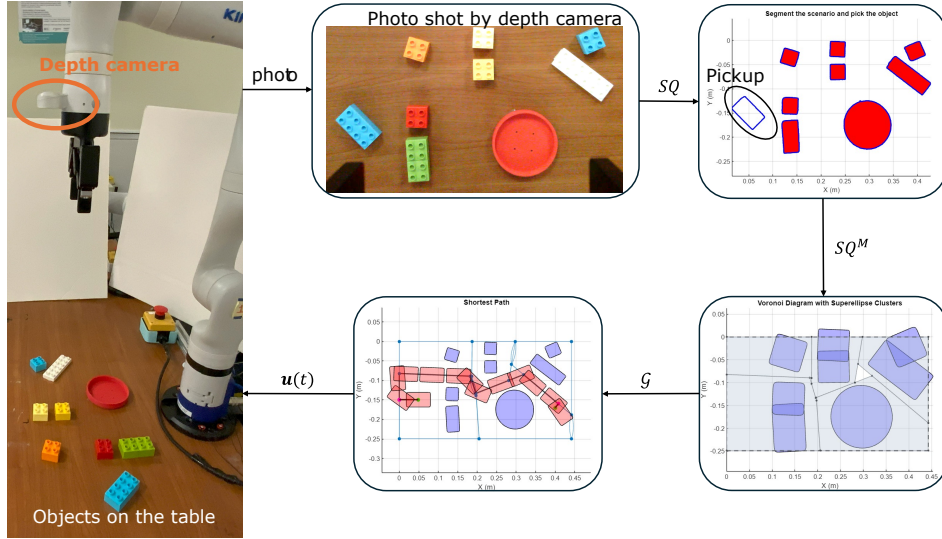
Fig. 4: Process for executing the narrow-passage experiment. Step 1: The Kinova robot scans a table crowded with objects. Step 2: Objects are segmented, and the target object is selected as SQ_r. Step 3: The remaining obstacles SQ_o are expanded, where the impossible passage are eliminated. Step 4: Trajectory planning is performed with the desired orientation.

B. Benchmark for 2D scenario

To further validate our method, we compare it against three classical motion planners: A*, RRT* and Genetic Algorithm (GA). We design three 2D benchmark scenarios: narrow passage (Fig. 4), T-block, and U-block (Fig. 5), which are well-known challenging cases for motion planning. The results of our experiments are summarized in Table I, where [−] represents failure in the given time. All computations are performed in MATLAB. We randomly selected start and target points five times in each map to make comparisons.

Method	Metric	Narrow	T-block	U-block
A*	planning time (s)	0.42	0.16	0.21
	arc length (m)	0.41	0.42	0.38
	min-distance (mm)	0.7	1.5	1.0
RRT*	planning time (s)	8.16	2.71	5.26
	arc length (m)	0.76	0.92	0.88
	min-distance (mm)	4.1	2.1	1.18
GA	planning time (s)	5.78	2.94	-
	arc length (m)	0.45	0.57	-
	min-distance (mm)	1.1	2.3	-
Proposed	planning time (s)	0.065	0.095	0.083
	arc length (m)	0.61	0.52	0.51
	min-distance (mm)	14.2	23.5	26.3

TABLE I: Benchmark for 2D scenario.

In this comparison, we evaluate three key metrics: planning time, arc length, and minimum distance. Planning time refers to the duration of the algorithm's execution, arc length represents the total trajectory length, and min-distance refers to the closest distance between the robot SQ_r to any obstacle SQ_o.

Our results show that our algorithm consistently completes planning within $0.1s$ for all 2D scenarios, whereas the classical planners require significantly more time. Additionally, our method achieves the largest min-distance from obstacles

among all tested planners. This advantage stems from the Voronoi diagram's inherent property of maximum clearance path (Eq. 8), which is particularly crucial in the tasks involving small objects. Given that perception errors are typically in the millimeter range [23], maintaining a safe clearance is essential. In contrast, the other three classical planners frequently generate trajectories very close to obstacles (less than 10 mm), increasing the risk of collisions. Our method, however, consistently maintains a clearance of over 10 mm, ensuring safer navigation.
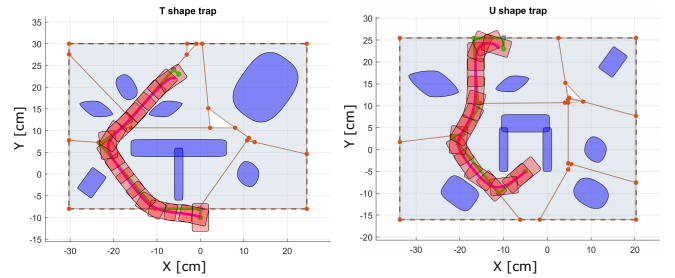
(a) T block scenario.

(b) U block scenario.

Fig. 5: Our method in U-block and T-block scenarios, where closed obstacles are segmented into the same Voronoi region.

V. SIMULATION: DRONE NAVIGATION IN 3D ENVIRONMENTS

In this section, we apply our algorithm into 3D world. A relevant example is drone navigation in a crowded scenario, where all obstacles, including the drone itself (modeled as an ellipsoid), can be approximated by SQ.

A. Experiment setup

We designed four challenging scenarios: narrow-passage, trap, complex, and common: a) Narrow-passage scenario

(Fig. 6a): This setup consists of several pillar-like obstacles, a common real-world configuration [4]. We manually position four SQs as pillars, creating a passage just wide enough for the drone to pass through. Specifically, the gap between obstacles is smaller than the drone's long axis but larger than its short axis, requiring the drone to adjust its orientation to navigate successfully. The other two passages are too narrow for passage. *b*) Trap scenario (Fig. 6b right): The drone starts inside a confined space, or "trap," with only one exit. The opening is positioned on the side opposite to the target location, making path planning more challenging. *c*) Complex scenario (Fig. 6b left): This environment contains a large number of obstacles, significantly increasing the difficulty of finding a feasible trajectory. *d*) Common scenario: A simpler environment with fewer obstacles, representing a common case for comparison.

We compare our method against three classical motion planners: A*, RRT*, and GA as in Section IV-B. Additionally, we evaluate our approach against GCOPTER, a cutting-edge algorithm, using its open-source implementation [17]. To integrate GCOPTER, we first generate the map in MATLAB and convert it into a point cloud, which serves as input for GCOPTER's environment in Ubuntu 20.04. The results are then collected and analyzed in MATLAB. For each scenario, we randomly select start and target positions five times to ensure a robust comparison.

B. Result analysis

The results are summarized in Table II, where $[-]$ represents failure in the given time. and some representative results are visualized in Fig.6. we get the following conclusions:

Method	Metric	Narrow	Trap	Complex	Common
A*	planning time (s)	3.58	3.77	4.25	8.06
	arc length (m)	22.4	28.8	29.0	42.8
	min-distance (m)	0.12	0.25	0.65	0.49
RRT*	planning time (s)	8.92	-	1.39	1.44
	arc length (m)	21.4	-	28.8	43.5
	min-distance (m)	0.22	-	0.62	0.51
GA	planning time (s)	138.0	-	24.89	27.5
	arc length (m)	24.3	-	30.1	44.9
	min-distance (m)	0.23	-	0.71	0.55
GCOPTER*	planning time (s)	-	0.032	0.034	0.028
	arc length (m)	-	30.2	27.4	41.3
	min-distance (m)	-	0.62	0.87	0.53
Proposed	planning time (s)	0.26	0.28	0.42	0.16
	arc length (m)	33.47	39.3	32.0	51.2
	min-distance (m)	0.28	0.95	1.46	1.71

TABLE II: Benchmark for 3D scenario.

- **Safer path selection:** Our method consistently selects paths that keep away from obstacles due to the maximum clearance property (Eq. 8). As shown in Table II, our method achieves the largest minimum distance from obstacles compared to other planners. This behavior is particularly evident in the Complex scenario (Fig. 6b left), where our algorithm prioritizes a longer but safer route, avoiding the densely packed obstacles in the center. A similar trend is observed in the Trap scenario
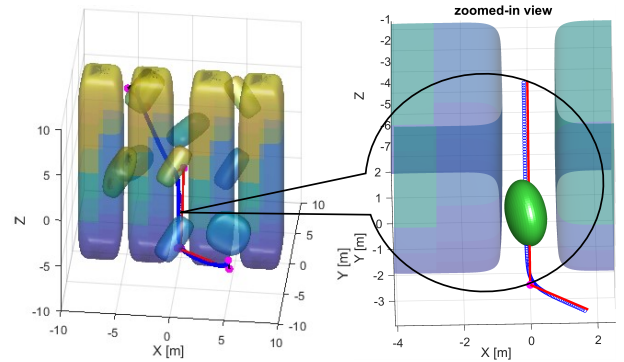
(a) Our method in the narrow-passage scenario, where the drone tilts to navigate through a tight gap between obstacles.
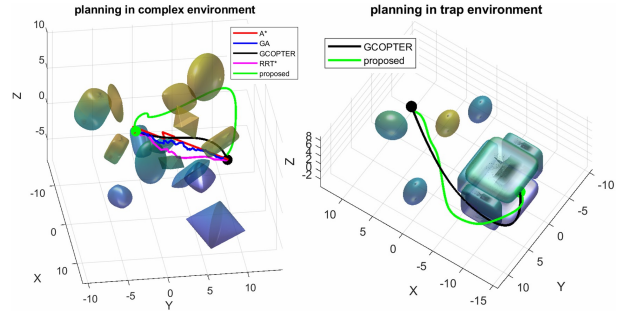
(b) Comparison of our method vs. GCOPTER and classical planners in complex and trap scenario.

Fig. 6: Comparison of our method with GCOPTER and classical planners in various 3D scenarios.

(Fig. 6b right), where our method takes a slight detour to safely exit the trap, in contrast to GCOPTER, which follows a more direct yet riskier path.

- **Comparison with GCOPTER:** Our method successfully identifies and eliminates impassable gaps while correctly selecting the only viable narrow passage, as shown in Fig. 6a. The drone also tilts at an angle to navigate through the passage, as seen in the zoomed-in view of Fig. 6a. In this figure, the green ellipsoid represents the drone, while the transparent shapes depict surrounding obstacles. Our planner ensures that the drone's short axis aligns with the Voronoi hyperplane normal (Eq. 16), enabling it to pass through tight spaces efficiently. In contrast, the open-source GCOPTER algorithm does not achieve similar results in this environment due to its spherical modeling of the drone. One limitation of our method is that it is slightly slower than GCOPTER, though both complete execution within less than 0.5 second.

- **Comparison with classical planners:** Our method successfully navigates the Trap scenario by treating trap-like structures as Voronoi regions V_i (Eq.13), allowing the planner to escape efficiently. In contrast, classical methods such as RRT* and GA fail to exit the trap within the given time limit, as shown in Fig. 6b. While A* eventually finds an escape route, it requires

significantly more time. In general, our method proves much more efficient in 3D environments than classical planners. While RRT*, GA, and A* require more than 1 second, our algorithm completes planning in under 0.5 second, demonstrating its computational efficiency.

Limitation: Although our approach can handle moderate non-convexity by clustering adjacent obstacles, it faces challenges in highly non-convex environments. Specifically, when the obstacle configurations become too intricate or require merging regions, the current formulation is constrained by the convex polygonal representation in Bensolve. This limitation prevents merging Voronoi regions to form a non-convex space.

VI. Conclusion

This study introduces an effective path planning approach that integrates SQ modeling with Voronoi diagrams to achieve maximum clearance navigation. By leveraging the differentiable SQ formulation, the computation of Voronoi hyperplanes becomes efficient, thereby enhancing both safety and orientation guidance. Moreover, the flexibility of SQ representations allows the method to approximate a broad range of robot and object geometries (e.g., trap) within a single Voronoi region. Furthermore, by incorporating Minkowski sums, our approach eliminates impractical narrow gaps by merging close obstacles into a same Voronoi region. Due to its mathematical simplicity and generalizability, the proposed method is applicable to both 2D and 3D environments, making it a versatile solution for complex path planning tasks.

We validated our algorithm in 2D and 3D scenarios, including narrow passages, traps, and complex obstacle configurations. Our approach was compared with classical planners (A*, RRT*, GA) and a cutting-edge drone planner (GCOPTER). The results demonstrate that our method is faster than classical methods and effectively navigates through traps and narrow passages, whereas other planners exhibit various limitations.

Future work will focus on addressing our limitation by exploring ways to handle non-convex Voronoi regions in highly intricate environments. Additionally, we will investigate how to reduce unnecessary turns while keep safety. Meanwhile, we aim to extend the planner to include robotic manipulators, allowing for both end-effector and manipulator path planning to achieve self-collision avoidance.

References

[1] L. Yang, M. Z. Ariffin, B. Lou, C. Lv, and D. Campolo, "A planning framework for robotic insertion tasks via hydroelastic contact model," *Machines*, vol. 11, no. 7, p. 741, 2023.

[2] L. Yang, S. H. Turlapati, Z. Lu, C. Lv, and D. Campolo, "A planning framework for stable robust multi-contact manipulation," *arXiv preprint arXiv:2504.02516*, 2025.

[3] Y. Chen, Z. Zhang, Z. Wu, Z. Miao, H. Zhang, and Y. Wang, "Set: Sampling-enhanced exploration tree for mobile robot in restricted environments," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 10, pp. 10467–10477, 2023.

[4] F. Gao, L. Wang, B. Zhou, X. Zhou, J. Pan, and S. Shen, "Teach-repeat-replan: A complete and robust system for aggressive flight in complex environments," *IEEE Transactions on Robotics*, vol. 36, no. 5, pp. 1526–1545, 2020.

[5] J. O. Jimenez and W. Suleiman, "Visualizing high-dimensional configuration spaces: A comprehensive analytical approach," *IEEE Robotics and Automation Letters*, 2024.

[6] T. T. Mac, C. Copot, D. T. Tran, and R. De Keyser, "Heuristic approaches in robot path planning: A survey," *Robotics and Autonomous Systems*, vol. 86, pp. 13–28, 2016.

[7] Z. Lei, B. Y. Tan, N. P. Garg, L. Li, A. Sidarta, and W. T. Ang, "An intention prediction based shared control system for point-to-point navigation of a robotic wheelchair," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 8893–8900, 2022.

[8] O. Arslan and D. E. Koditschek, "Sensor-based reactive navigation in unknown convex sphere worlds," *The International Journal of Robotics Research*, vol. 38, no. 2-3, pp. 196–223, 2019.

[9] X. Chen, B. Zhou, J. Lin, Y. Zhang, F. Zhang, and S. Shen, "Fast 3d sparse topological skeleton graph generation for mobile robot global planning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10283–10289, IEEE, 2022.

[10] A. Orthey and M. Toussaint, "Section patterns: Efficiently solving narrow passage problems in multilevel motion planning," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1891–1905, 2021.

[11] A. Jaklic, A. Leonardis, and F. Solina, *Segmentation and recovery of superquadrics*, vol. 20. Springer Science & Business Media, 2000.

[12] S. Ruan, K. L. Poblete, H. Wu, Q. Ma, and G. S. Chirikjian, "Efficient path planning in narrow passages for robots with ellipsoidal components," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 110–127, 2022.

[13] L. Zhao, J. Zhao, Z. Liu, D. Yang, and H. Liu, "Solving the real-time motion planning problem for non-holonomic robots with collision avoidance in dynamic scenes," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10510–10517, 2022.

[14] W. Liu, Y. Wu, S. Ruan, and G. S. Chirikjian, "Robust and accurate superquadric recovery: A probabilistic approach," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2676–2685, 2022.

[15] L. Yang, S. H. Turlapati, C. Lv, and D. Campolo, "Planning for quasi-static manipulation tasks via an intrinsic haptic metric: A book insertion case study," *IEEE Robotics and Automation Letters*, 2025.

[16] L. Yang, H.-T. Nguyen, C. Lv, D. Campolo, and F. Cardin, "An energy-based numerical continuation approach for quasi-static mechanical manipulation," *Data-Centric Engineering*, vol. 6, p. e18, 2025.

[17] Z. Wang, X. Zhou, C. Xu, and F. Gao, "Geometrically constrained trajectory optimization for multicopters," *IEEE Transactions on Robotics*, vol. 38, no. 5, pp. 3259–3278, 2022.

[18] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[19] H.-T. Nguyen, L. Yang, C. Lv, and D. Campolo, "Quasi-static contact-rich manipulation using dynamic movement primitives and haptic potential map," *Data-Centric Engineering*, vol. 6, p. e41, 2025.

[20] C. Nam, J. Lee, S. H. Cheong, B. Y. Cho, and C. Kim, "Fast and resilient manipulation planning for target retrieval in clutter," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3777–3783, IEEE, 2020.

[21] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, *et al.*, "Segment anything," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4015–4026, 2023.

[22] D. Ciripoi, A. Löhne, and B. Weißing, "Calculus of convex polyhedra and polyhedral convex functions by utilizing a multiple objective linear programming solver," *Optimization*, 2019.

[23] A. Morar, A. Moldoveanu, I. Mocanu, F. Moldoveanu, I. E. Radoi, V. Asavei, A. Gradinaru, and A. Butean, "A comprehensive survey of indoor localization methods based on computer vision," *Sensors*, vol. 20, no. 9, p. 2641, 2020.