A Voronoi Diagram–Visibility Graph–Potential Field Compound Algorithm for Robot Path Planning

Ellips Masehian* and M. R. Amin-Naseri
Faculty of Engineering
Tarbiat Modarres University
Tehran, 14115-143, Iran
e-mail: masehian@modares.ac.ir

Received 14 April 2003; accepted 15 January 2004

Numerous methods have been developed to solve the motion planning problem, among which the Voronoi diagram, visibility graph, and potential fields are well-known techniques. In this paper, a new path planning algorithm is presented where these three methods are integrated for the first time in a single architecture. After constructing the generalized Voronoi diagram of \( C \)-space, we introduce a novel procedure for its abstraction, producing a \( \text{pruned generalized Voronoi diagram} \). A broad freeway net is then developed through a new \( \alpha\text{-MID} \) (maximal inscribed discs) concept. A potential function is assigned to the net to form an obstacle-free network of valleys. Afterwards we take advantage of a bidirectional search, where the visibility graph and potential field modules execute alternately from both start and goal configurations. A \( \text{steepest descent mildest ascent} \) search technique is used for local planning and avoiding local minima. The algorithm provides a parametric tradeoff between safest and shortest paths and generally yields shorter paths than the Voronoi and potential field methods, and faster than the visibility graph. It also performs well in complicated environments.

1. INTRODUCTION

The general motion planning problem is shown to be PSPACE-hard and PSPACE-complete with an example of a many-jointed object constrained to move within a complex system of narrow channels.\(^1\) This exponential space requirement stems from the vast number of different configurations these planners have to cope with. In addition to the storage problem, it is also proved that the path planning problem is NP-complete.\(^2\) This means that a path planner’s running time is exponential in the degrees of freedom. Therefore the exponential time and storage requirements together create a challenging research problem.

Collision avoidance is the most important factor of motion planning. A planner should guarantee a collision-free movement of the robot; otherwise the system will fail to function properly due to a break-
down of the hardware. This must be done automatically by the planner or by a human programmer.

The major solution methods are variations of a few general approaches: skeleton (e.g., visibility graph and Voronoi diagram), cell decomposition (object-dependent and object-independent), potential fields, and mathematical programming. Most classes of motion planning problems can be solved using these approaches. Other approaches are randomized kinodynamic planning,\(^3\) randomized planning, probabilistic roadmap planners,\(^4\) sensor based motion planning,\(^5\) and methods dealing with different constraints such as nonholonomic, speed, inertia, etc. A good survey on some methods can be found in refs. 1 and 6.

The first section of this paper deals with a brief description of Voronoi diagrams, visibility graphs, and potential fields. The outline of our algorithm (which we call the VVP algorithm for simplicity) is described in Section 2. In Section 3 we go over the details of the preprocessing phase of the algorithm, where some new concepts such as the pruned generalized Voronoi diagram (PGVD) and \(\alpha\)-MID (\(\alpha\) is a scalar) are developed. Section 4 presents a bidirectional search method based on visibility rays and steepest descent mildest ascent search. Also a new version of the potential field is proposed, where no repulsive potential is associated with \(C\)-obstacles. The experimental results and comparisons are given in Sections 5 and 6, respectively. More issues are discussed in Section 7. The conclusion comes in Section 8.

1.1. Voronoi Diagram

The Voronoi diagram is defined as the set of points that are equidistant from two or more object features. The Voronoi diagram partitions the space into regions, where each region contains one feature. For each point in a region, this feature is the closest feature to the point than any other feature.

The Voronoi diagram is attractive in two respects: there are only \(O(n)\) edges in the Voronoi diagram, and it can be efficiently constructed in \(\Omega(n \log n)\) time, where \(n\) is the number of features.\(^7\) By using Dijkstra’s method the Voronoi diagram can be searched for the shortest path in \(O(n^2)\) time. Another advantage of Voronoi methods is the fact that the object’s initial connectedness is directly transferred to the diagram,\(^5\) whereas other path planning methods have to restore connectedness artificially in a postprocessing step.

From a practical point of view, Voronoi diagrams seem to be inefficient in high (especially more than three) dimensions and they require very complex data structures. Moreover, the form of the diagram differs when different features of the objects are presumed as sites, e.g., vertices, edges or the whole object. In spite of these facts, Voronoi diagrams have found a wide application in motion planning problems. Especially when the edges of the convex obstacles are taken as sites, and the configuration space \(C=\mathbb{R}^2\), the Voronoi diagram of free space \(V(C_{free})\) consists of a finite collection of straight line segments and parabolic curve segments (arcs), referred to as medial axis transform, or, more often, generalized Voronoi diagram (GVD).

1.2. Visibility Graph

The visibility graph is the collection of lines in the free space that connect a feature of an object to that of another. In its principal form, these features are vertices of polygonal obstacles, and there are \(O(n^2)\) edges in the visibility graph, which can be constructed in \(O(n^2)\) time and space in 2D, where \(n\) is the number of features.

An extension to the simple visibility graph is the generalized visibility graph, where the obstacles are generalized polygons, i.e., regions bounded by straight segments and/or circular arcs. So the features can be both vertices on sharp corners and points located on circular edges of obstacle boundary. Let \(n\) be the total number of vertices of \(CB\), the union of all \(C\)-obstacles. Then the number of nodes in the graph is \(O(n^2)\) and the number of links is also \(O(n^2)\). The generalized visibility graph can be constructed in \(O(n^3)\) time and its search performed in \(O(n^2)\) time. The shortest path can be found in \(O(n^2 \log n)\) time using the \(A^*\) algorithm with the Euclidean distance to the goal as the heuristic function.\(^8\)

1.3. Potential Fields

A robot in the potential fields method is treated as a point represented in configuration space, and as a particle under the influence of an artificial potential field \(U\) whose local variations reflect the “structure” of the free space.\(^9\) In order to make the robot attracted toward its goal configuration, while being repulsed from the obstacles, \(U\) is constructed as the sum of two elementary potential functions: attractive potential associated with the goal configuration \(q_{goal}\) and repulsive potential associated with the \(C\)-obstacle region. Motion planning is performed in an iterative fashion. At each iteration, the artificial force induced
by the potential function at the current configuration is regarded as the most appropriate direction of motion, and path planning proceeds along this direction by some increment.

The most serious problem with the potential fields method is the presence of local minima caused by the interaction of attractive and repulsive potentials, which results in a cyclic motion. The routine method for getting free is to take a random step outwards of the minimum well. Other drawbacks are:

(i) no passage between closely spaced obstacles,
(ii) oscillations in the presence of obstacles,
(iii) oscillations in narrow passages,
(iv) nonsmooth movements of the robot when trying to extricate from a local minimum,
(v) overlapping of different obstacles repulsive potential when they are adjacent to each other, and
(vi) difficulty in defining potential parameters properly.

Nevertheless, the potential fields method remains as a major path-planning approach, especially when high degrees of freedoms are involved.

2. THE VVP ALGORITHM

Considering the attributes of the three aforementioned path planning techniques, they are incorporated in a single model to benefit from advantages of each. To accomplish this, we tailored them and added a number of complementary new procedures to generate a valid and high quality path.

The algorithm is principally developed based on the following assumptions:

(i) The workspace is two-dimensional, where the robot is considered as a point. For real applications, this supposition can be attained by expanding the obstacles using Minkowski set difference.
(ii) The map of workspace is known a priori.
(iii) The obstacles are static.

The algorithm has two major stages: preprocessing phase and search phase. The latter contains two modules: visibility and potential field. A description of the algorithm steps will be presented below. Subsection numbers point to further details discussed in Sections 3 and 4.

2.1. Preprocessing Phase

In this phase we establish a guaranteed no-obstacle area for robot navigation. The main steps are the following.

P1) Define the obstacles in a distributed (grid-based) form (Section 3.1), and specify start and goal points.

P2) Construct a simple Voronoi diagram by taking the obstacle pixels as features, and convert it to a generalized Voronoi diagram (GVD) (Section 3.2).

P3) Connect the start and goal points to the GVD by edges totally laid in \(C_{\text{free}}\). If this operation fails, report that no valid path exists. This step implicitly preserves the exactness of the algorithm.

P4) Prune (trim) the GVD edges stretched toward dead-end regions, forming a PGVD (Section 3.3). This is a very important step in our algorithm, since a good abstraction of the environment results and most relatively “deep” local minimum wells are excluded from the search space.

P5) Form an obstacle-free C-space region by developing \(\alpha\)-scaled \((0<\alpha<1)\) maximal inscribed discs (\(\alpha\)-MIDs) centered on PGVD points (Section 3.4). This region is nonconvex and reflects the topology of the \(C_{\text{free}}\).

P6) Associate an attractive (negative) potential to \(\alpha\)-MID region (Section 3.5). The result is an obstacle-free network of freeway valleys which serves as the navigation area of the robot.

P7) The global search process is performed bidirectionally, i.e., from both start \((S)\) and goal \((G)\) points. Thus we initialize two trajectory sets related to those points: \(\text{Traj}(S)\) and \(\text{Traj}(G)\). These sets keep the track of paths spanned from \(S\) and \(G\), respectively.

For the very beginning, set \(\text{Traj}(S) = \{\text{Start point}\}\) and \(\text{Traj}(G) = \{\text{Goal point}\}\).

2.2. Search Phase

This phase is designed to progressively build up a start-to-goal path. In fact, the global problem is decomposed to a number of smaller path planning tasks, having intermediate milestones as temporary start and goal points. Through this iterative process the global path is incrementally constructed. Figure 1 demonstrates the overall sequence of operations.

The main modules included in this phase are visibility and potential field, which are executed...
iteratively toward the construction of the final path. The termination condition is satisfied when $\text{Traj}(S)$ and $\text{Traj}(G)$ get connected. Steps S1, S2 and S3 constitute the visibility module, and the remaining steps are put into action for the potential field part, as described below.

S1) Perform a visibility scan. Each iteration starts with this procedure. The scan is concurrently implemented for the endpoints (i.e., last elements) of both $\text{Traj}(S)$ and $\text{Traj}(G)$. A radial sweeping ray technique is applied to collect information about the surrounding obstacle boundaries and the opposite trajectory (Section 4.1).

Suppose that the ray sweeping operation is performed from $p$ and $q$, the endpoints of $\text{Traj}(S)$ and $\text{Traj}(G)$, respectively. Consequently, four incidences may occur (Figure 2):

S1-a) A subset of points in $\text{Traj}(G)$ is visible from $p$, but no point from $\text{Traj}(S)$ is visible from $q$ [Figure 2(a)]. In this case, by a straight line, connect $p$ to a visible point in $\text{Traj}(G)$, say $q'$, which is nearest to the goal point (i.e., has the smallest ordinal rank in $\text{Traj}(G)$ among the visible points], and truncate all elements in $\text{Traj}(G)$ located after $q'$. Note that the goal point might be visible itself, which in that instance, point $p$ is directly connected to the $G$ [as in Figure 2(c)].

S1-b) A subset of points in $\text{Traj}(S)$ is visible from $q$, but no point from $\text{Traj}(G)$ is visible from $p$ [Figure 2(b)]. Since this is similar to the previous case, act in the same way, but swap the $p$ and $q$, also $\text{Traj}(S)$ and $\text{Traj}(G)$.

S1-c) Subsets of points in both $\text{Traj}(G)$ and $\text{Traj}(S)$ are visible from $p$ and $q$, respectively [Figure 2(c)]. In this case, define the following criterion to determine the endpoints of connecting line:

\[
\min\{|\text{Traj}(S)| + \|p - q'\| + |q' \in \text{Traj}(G)|, \\
|\text{Traj}(G)| + \|q - p'\| + |p' \in \text{Traj}(S)|\},
\]

(1)

where $|\text{Traj}(S)|$ means the cardinality (or length) of $\text{Traj}(S)$, $\|p - q'\|$ is the Euclidean distance of $p$ and $q'$, and $|q' \in \text{Traj}(G)|$ indicates the ordinal position of $q'$ in $\text{Traj}(G)$ (i.e., the distance of $q'$ to $G$ via trajectory). Among $pq'$ and $qp'$, the line providing the minimum value for the above criterion will be selected to

Figure 2. Four different combinations of $\text{Traj}(S)$ and $\text{Traj}(G)$ in the visibility scan.
connect Traj(S) and Traj(G). Again truncate the elements of the trajectory located after the connection point \( p' \) or \( q' \), according to drawn line.

S1-d) If none of the Traj(S) and Traj(G) are visible to each other’s endpoints, then for both \( p \) and \( q \), determine those rays that are tangent to visible \( C_{\text{obs}} \) boundary (calculated in step P5). Note that this boundary is at a safe distance from actual obstacles’ edges. The intersection of these rays and the free space’s boundary produces two sets of critical points, \( R(p) \) and \( R(q) \). Figure 2(d) shows the result of a visibility scan from \( q \), which consequently renders four visible obstacle vertices in \( R(q) = \{1,2,3,4\} \).

Now among all combinations of the elements of \( R(p) \) and \( R(q) \), select the closest (i.e., nearest) pair. Mathematically speaking, select \( x \) and \( y \) such that \( \{(x,y)|\forall x,u \in R(p);y,v \in R(q);\|x-y\|\leq\|u-v\|\} \), where \( \|\| \) shows Euclidean distance. The total number of combinations to be evaluated is \( |R(p)| \times |R(q)| \), where \( |\| \) is the cardinality of sets. This operation determines the most mutual points that Traj(S) and Traj(G) must extend toward via two straight lines.

S2) Map the line segment(s) found in step S1 to the configuration space grid. Through a fine-enough discretizing operation, new points are added to Traj(S) and or Traj(G).

If any of the cases \( a \), \( b \), or \( c \) in step S1 hold, then terminate the search phase and go to step S10. For case \( d \) continue with next step.

S3) Since all the points in Traj(S) and Traj(G) lie on the bottom of roadmap valleys (calculated in step P6), in order to mark the valleys as traversed, increase the potentials of trajectory points and their surroundings to “fill” the width of valleys (Section 4.2). This is an effective operation for preventing the planner from searching every possible configuration in \( C_{\text{free}} \).

The following steps represent the potential field module. It is applied in two directions: first the trajectory stemmed from the start point [i.e., Traj(S)] is extended (steps S4 to S6), then Traj(G) is stretched out (step S7 to S9).

S4) This step constructs a potential field for performing steepest descent mildest ascent search from the endpoint of Traj(S). For this purpose, we set the endpoint of Traj(G) as a temporary goal. Then we add a paraboloid function with a minimum apex on the temporary goal to the potential manifold gained in step P6. This paraboloid is expressed as

\[
U_{\text{goal}}(x,y) = \xi [(x-x_{\text{goal}})^2 + (y-y_{\text{goal}})^2],
\]

where \( \xi \) is a constant. The global minimum of the resulting potential is located on the temporary goal point. Note that unlike the goal potential for classical potential fields method which consists of conic and parabolic wells, this goal potential is simply a paraboloid. We have also successfully implemented a Gaussian bell-shaped potential described by \( U_{\text{goal}}(x,y) = \xi e^{\delta (x-x_{\text{goal}})^2 + (y-y_{\text{goal}})^2} \), in which \( \xi \) and \( \delta \) are constants.

To apply the paraboloid potential, we graduate the configuration space in a fine-enough resolution; then, assigning every grid cell as \((x_i, y_i)\), the potential is calculated numerically (Section 4.3). Due to the numerical nature of the model, working with these complex functions is extremely easy (only a simple addition of corresponding grid values is sufficient).

S5) Now the steepest descent mildest ascent search is performed with setting the endpoint of Traj(S) as temporary start, and the endpoint of Traj(G) as temporary goal points. This step contains a gradient search for selecting the next grid-cell to proceed. New points are appended to Traj(S). Also, in order to provide a mechanism for escaping from local minima, the potentials of visited points and their adjacent cells across the roadmap valleys are elevated (Section 4.2). This operation hinders the robot to be attracted to a local minimum for many times.

S6) Repeat the step S5 until one of the following situations take place:

(a) If before the occurrence of case \( b \) below, the last point of Traj(S) comes close to any point in opposite trajectory, Traj(G), the search phase is completed. First truncate the elements of Traj(G) located after the connection point, then go to step S10.

(b) The grid-cell wavefront distance between the last point of Traj(S) and the free space boundary, \( \partial C_{\text{free}} \), exceeds a certain limit. That is, \( |\text{LAST}(\text{Traj(S)}) - \partial C_{\text{free}}| > d \). Through experimentation \( d = 3 \) was found appropriate.

The steepest descent search for Traj(S) is now terminated and therefore the searching process is shifted to
Before we develop the model, it is necessary to mathematically represent the workspace (or configuration space): centralized and distributed. In centralized representation, the geometric primitives are modeled as sets of algebraic equations, each for an edge. Often, these equations are linear, approximating objects to polygons. The great advantage of this type of representation is that the geometric primitives can be defined precisely at any scale. On the other hand, due to the unstructured nature of centralized representation, assessing the occupancy of a given location in space requires scanning the list of all objects present in the scene, which requires a time linear in the number of geometric primitives. For path planning, a number of collision detections are to be made, ranging from a few hundred for simple cases to a few hundreds of thousands for complex environments. Therefore a considerable amount of time is spent on this task.

In distributed representation, the workspace \( \mathcal{W} \) is modeled as a bitmap array, by the following function: \( \mathcal{Z}: \mathcal{W} \rightarrow \{0, 1\} \), and \( x \rightarrow \mathcal{Z}(x) \).

The subset of points \( x \) for which \( \mathcal{Z}(x) = 1 \) represents the workspace obstacles and the subset of points \( x \) such that \( \mathcal{Z}(x) = 0 \) represents the empty workspace. In other words, \( \mathcal{W}_{\text{empty}} = \{x | \mathcal{Z}(x) = 0\} \).

Due to this simple representation, any location can be determined rapidly to be empty or occupied with obstacles, in a time constant in the number and shape of obstacles. The major drawback of distributed representation is the high memory requirement for the bitmap array, particularly when the resolution is high.

To take advantage of the low-time collision detection, and to facilitate the calculation of the Voronoi diagram which plays a central role in our algorithm, we adopt the distributed representation for our algorithm, shown for a sample workspace in Figure 4(a).

### 3.2. Generalized Voronoi Diagram

The main reason for incorporating the Voronoi concept in our algorithm is its property of lying on the maximum clearance from the obstacles. This property helps the robot to navigate at a safe distance from obstacles, making it less prone to be trapped in local minimum wells.

To start with, we construct a simple Voronoi diagram based on the input workspace using a common algorithm, say Fortune’s, in a time order of \( \Omega(n \log n) \). Note that the distributed nature of the workspace representation permits such an operation, where every single point lying in obstacle space is assumed a feature. To avoid the formation of unbounded Voronoi cells, the environment is
considered to be confined within a rectangular border, being itself an obstacle. However, the bisector lines of border points are deleted due to their extension to infinity. The simple Voronoi diagram of the workspace in Figure 4(a) is depicted in Figure 4(b).

Because we are interested in calculating the generalized Voronoi diagram, we have devised a novel procedure that converts a simple Voronoi diagram to generalized form for distributed workspace representation. For this purpose, we perform a neighborhood
check for each Voronoi vertex, i.e., points having more than two equidistant sites. Regarding the resolution of the uniform grid representing the workspace, we mark those Voronoi vertices \( V(v_x, v_y) \) which have exactly four neighbors located on the four corners of a \([v_x \pm \text{res}/2, v_y \pm \text{res}/2]\) frame [Figure 5(a)].

The marked vertices and the Voronoi edges stemmed from them are then deleted. Figure 5(b) demonstrates the result of this operation applied to the simple Voronoi diagram in Figure 4(b).

As can be seen in Figure 5(b), the stepwise approximation of the sloped edge in the lower-right obstacle causes the formation of some redundant branches connecting the obstacle to the main GV diagram. The same problem can arise when there is a degree of uncertainty in defining workspace objects exactly. This is originated from the nature of Voronoi diagram, which is noise-sensitive. However, such inaccuracies do not affect our algorithm due to a pruning operation performed after constructing the GV.

To this end we did not need the information about the start and goal points. Now this knowledge is incorporated into the model through a simple connecting algorithm, where the start and goal positions are connected to the generalized Voronoi diagram via the closest point. The connecting edges are also included in the diagram.

---

**Figure 4.** (a) Distributed representation of a sample workspace. The resolution is 0.5 unit × 0.5 unit. (b) Simple Voronoi diagram with obstacle points taken as features. The points are aggrandized in size and drawn as squares.

**Figure 5.** (a) Voronoi vertices located at the center of obstacle point frames. The vertices having four obstacle neighbors (shown as squares) are to be deleted. (b) Generalized Voronoi diagram (GVD).
3.3. Pruning Procedure

Before we continue, let us introduce some notation. Suppose \( W \) is the workspace, and \( B \) the set of obstacles. Then \( V(B) \) denotes the (generalized) Voronoi diagram of \( B \), \( VV(c_i, c_j) \) is the Voronoi edge with \( c_i \) and \( c_j \) being its two closest obstacles, and \( VV(ve_i, ve_j) \) stands for the Voronoi vertex connecting the two Voronoi edges, \( ve_i \) and \( ve_j \).

Definition 1: We consider two Voronoi vertices \( VV_i(ve_k, ve_l) \) and \( VV_j(ve_m, ve_n) \) to be neighbors, iff \( \{ve_k, ve_l\} \cap \{ve_m, ve_n\} \neq \emptyset \), and \( VV_j = N(VV_i) \) shows this neighborhood.

Definition 2: The neighborhood degree of a vertex \( ND(VV_i) \) is defined as the number of vertices sharing a Voronoi edge with vertex \( VV_i \).

In this way, the points \( x \in V(B) \) are classified in three groups:

1. Points having \( ND(x) = 1 \) are end-vertices.
2. Points having \( ND(x) = 2 \) are located on a Voronoi edge.
3. Points having \( ND(x) = k (k \geq 3) \) are meet points, where \( k \) or more edges are conjoined.

The pruning operation is now proposed according to the algorithm in Figure 6.

Taking advantage of the distributed representation of the workspace, the scanning operation in step 1 of Figure 6 can be performed in a faster way, simultaneous with the process of converting the simple Voronoi diagram to the generalized form. There we performed a neighborhood checking operation which culminated in removal of Voronoi vertices having exactly four obstacle point neighbors \( [V(ve_x, ve_y) \) in Figure 5(a)]. Now the vertices having three obstacle point neighbors \( [e.g., V'(ve_x', ve_y')] \) are the very end-vertices of \( V(B) \) and hence must be included in \( P \).

Note that the previously mentioned problems of redundant Voronoi edge formations in sloped obstacles and the hypersensitivity of the Voronoi diagram to minor uncertainties in workspace definition are resolved after running the pruning procedure.

The pruning operation is an important stage in our algorithm since it truncates all paths toward collision with obstacles and dead-end traps and therefore reduces the search space drastically. After assigning two arbitrary locations as start and goal points, the result of the pruning operation performed on our example is shown in Figure 7.

3.4. α-MID Region Construction

As noted in Section 1.1, the generalized Voronoi diagram is also known as the medial axis transform (MAT). This concept first appeared in the literature in 1967, when H. Blum introduced the notion of a skeleton in his paper about MAT.12 There he compared the symmetric or medial axis transform with a grass fire phenomenon, where the fire on the borders of a grass field advances toward the center. The fire fronts will meet and quench in some points which form the medial axis. Blum showed that these points are the centers of maximal inscribed discs (MIDs).

Figure 8 shows the maximal inscribed discs and medial axis of an L-shaped environment.

In order to express the MID mathematically, we need to define some terms:

Let \( \mathcal{W} \) stand for the workspace and \( \mathcal{C} \) its configuration space. Then \( \mathcal{C}_{\text{free}} \) represents the free configura-
The maximal inscribed disc (MID) is defined as the locus of points in $C_{obs}$ such that $r_{LMD}(x)$ for every point in $C_{free}$. Let

$$\text{clearance}(q) = \min_{p \in \beta} \|q - p\|, \forall q \in C_{free}. \quad (3)$$

Then the nearest point(s) on obstacle boundary is defined as

$$\text{Near}(q) = \{p \in \beta : \|q - p\| = \text{clearance}(q)\}. \quad (4)$$

The medial axis is the set of points such that

$$MA = \{q \in C_{free} : \|\text{Near}(q)\| > 1\}. \quad (5)$$

Equation (5) implies that the medial axis is the locus of points in $C_{free}$ having the maximum clearance from obstacles boundary. Here $\|\|$ shows the cardinality of set.

Now we introduce the locally maximal disc (LMD) for every point in $C_{free}$ as

$$\text{LMD}(x) = \{q : \|x - q\| < \text{clearance}(x) \land x, q \in C_{free}\}. \quad (6)$$

The maximal inscribed disc (MID) is defined as

$$\text{MID}(x) = \{\text{LMD}(x) : r_{LMD(x)} > r_{LMD(y)}\} \quad x \in MA \land y \in N(x) \quad (7)$$

in which the $r_{LMD(x)}$ is the radius of the locally maximal disc centered at $x$, and $N(x)$ denotes the neighborhood of $x$.

Now the $\alpha$-maximal inscribed disc is defined according to

$$\alpha\text{MID}(x) = \{\text{LMD}(x) : r_{LMD(x)} = \alpha \times r_{\text{MID}(x)}\} \quad x \in MA, 0 < \alpha < 1 \quad (8)$$

The loci of centers of maximal inscribed discs form the medial axis, and the transformation of an object to its medial axis is called the medial axis transform (MAT).

The medial axis for the case of a simple polygon has been calculated in linear time complexity, which is better than the generally known $O(n \log n)$. A number of works have utilized the medial axis concept for path planning. The mathematical properties of medial axis are well studied in ref. 17. It is important to note that this concept can be extended to three-dimensional space in a straightforward way (Section 7.5). The centers of maximal inscribed balls (MIBs) in a 3D object form a 2D medial axis, also called the medial surface.

Now we deal with the algorithm and explain why and how the MID was implemented in our work. As noted in Section 1.1, Voronoi diagrams lie on the maximum clearance of objects. Although this property offers some advantages in the sense of path safety, it elongates the path, especially in workspaces where the obstacles are located quite far from each other. Besides, the path generated thus usually has acute angles around Voronoi vertices, making it useless for robots with nonholonomic or rotational constraints.

In order to compensate these shortcomings and improve our previous version of the algorithm, we tried to build a network of wide channels based on the GVD. These channels have the Voronoi edges as their skeleton and provide sufficient space for the robot to navigate along shorter paths and maneuver freely. However, due to the different sizes of free spaces between various obstacles or passages, the widths of the channels must vary accordingly.

Considering this matter, we utilized a “reverse” MAT approach. That is, instead of retracting the workspace to its medial axis, we dilated and expanded the medial axis (i.e., GVD) to build a region reflecting the topology of actual workspace (e.g., connectedness, convexity, etc.). Hence the MID concept was incorporated to construct an obstacle-free network of channels.

For this purpose, we calculated the radii of the maximal inscribed discs centered on each point of generalized Voronoi diagram. Meanwhile, in order to maintain a safe distance to obstacle borders, we
multiplied each MID radius by a scalar factor, named $\alpha(0 < \alpha < 1)$, and integrated all these discs in a single region, named the $\alpha$-MID region, as defined below:

**Definition 3:** The $\alpha$-MID region of a configuration space is the union of all $\alpha$-maximal inscribed discs centered on medial axis (MA) points expressed as

$$
\text{Region}(\alpha\text{MID}) = \{ \bigcup_{x \in \text{MA}} \alpha\text{MID}(x) \}. \quad (9)
$$

Since the robot is supposed to navigate within the $\alpha$-MID region’s borders, it is important to make sure that the $\alpha$-MID region lies totally in free $C$-space. Theorem 1 investigates this property.

**Theorem 1:** The $\alpha$-MID region is obstacle-free.

**Proof.** Suppose that the $\text{Region}(\alpha\text{MID})$ contains obstacle points. Then there should be some $x$ for which $\alpha\text{MID}(x) \cap C_{\text{obs}} \neq \emptyset$.

If $x \in C_{\text{free}}$ then according to (7), no locally maximal disc can be defined. So this supposition is not true.

If $x \in \text{MA}$, then the radius of $\alpha\text{MID}(x)$ must be greater than that of $\text{MID}(x)$. This is not possible because $0 < \alpha < 1$ according to (8).

If $x \not\in \text{MA}$ but $x \in C_{\text{free}}$ then the locally maximal disc centered on $x$ cannot have a radius greater than $\text{clearance}(x)$ as in (6).

Therefore there is no $x$ in $C$-space for which the $\alpha\text{MID}(x)$ overlaps an obstacle point. This figures out that the $\alpha$-MID region is obstacle-free.

An interesting property of the $\alpha$ is that it offers a balance between the roadmap and full $C_{\text{free}}$ concepts. If we set $\alpha = 0$, the resulting region will turn to the medial axis roadmap. For $\alpha = 1$, the region’s borders will be tangent to obstacles. Based on experiments, we recommend $\alpha \in [0.5, 0.8]$.

The $\alpha$-MID region for Figure 7 is calculated and depicted in Figure 9(a). Note that for our algorithm, the $\alpha$-MID discs are centered on points of the pruned generalized Voronoi diagram (PGVD). Another example is given in Figure 9(b).

Figures 9 also indicate the $\alpha$-MID region’s attribute of smoothening the Voronoi roadmap’s sharp corners and local irregularities. As a result, even the narrowest passages are widened and become more navigable.

### 3.5. Valley Potentials

Unlike the conventional potential fields concept where there are two kinds of attractive and repulsive potentials associated with goal and obstacles, respectively (Section 1.3), our algorithm makes use of two attractive potentials, related to the $\alpha$-MID region and temporary goals (steps P6 and S4 in Section 2). By this, we avoid some known problems of the standard potential fields related to the calculation of repulsive forces for each obstacle, and also their integration into a single function which usually gives rise to complexities due to their overlapping and parameter setting. Moreover, the problem of narrow corridors, where most potential field algorithms cause the robot to oscillate, is fixed in our method.

The first calculated attractive potential pertains to the $\alpha$-MID region. Since the conventional potential fields method suffers from local minima, we tried to hamper the robot to be absorbed to local traps and wells in the VVP algorithm. Since the $\alpha$-MID region keeps a safe $(1-\alpha)\%$ distance from obstacles, colliding
with obstacles will never arise in our context. Moreover, because we prune the GVD such that all Voronoi edges toward obstacles (mainly leading to concave corners or dead-ends) are eliminated from the graph, the possibility of the robot to get trapped in local minima reduces drastically. So, we try to “encourage” the robot to move along the α-MID valleys. This is done by associating an attractive potential with the points of valleys. The following function gives the desired result (s is the depth of the valley):

\[
U(x_i, y_i) = \begin{cases} 
-s & \text{if } (x_i, y_i) \in \{\alpha\text{-MID}\}, \\
0 & \text{otherwise}. 
\end{cases}
\] (10)

By applying a fine-enough grid resolution and using the MATLAB™ software, we performed this operation for α-MID regions of Figure 9 in a fraction of a second and gained the manifolds shown in Figure 10.

The preprocessing phase terminates with the construction of roadmap valleys’ potential field. Since the subsequent search’s strategy is bidirectional, it remains to initialize two sets of Traj(S) and Traj(G) as trajectories keeping the track of paths spanned from the start and goal points, respectively. At the outset, set Traj(S) = {Start point} and Traj(G) = {Goal point}.

4. SEARCHING PHASE

This section deals with the details of search phase steps. The search phase integrates two approaches of the visibility graph and potential fields in an iterative process to incrementally build up two trajectories from start and goal points. This process ends whenever those two trajectories are seen, or simply get in touch with each other. We characterize being seen as being able to draw a straight line in free space between two trajectories. The following subsections describe some details for visibility and potential fields modules.

4.1. Visibility Scan

After setting up the potential manifold of valleys, the search phase begins in an iterative manner (see Figure 1). Each iteration starts with a visibility scan performed concurrently for both endpoints of Traj(S) and Traj(G). For this purpose, a ray sweeping technique is used to collect information about the surrounding valley borders and probably the opposite trajectory.

The aim of this procedure is to determine whether the opposite trajectory is visible from the current point or not. If it is visible, then the search phase is over, and we treat according to step S1 in Section 2. If not, we have to find the boundary vertices as seen from the current point, as described below.

By applying a polar coordinate system with the origin defined on the vantage point [e.g., endpoint of Traj(S)], the radial distances to valley borders (\(\partial C_{\text{free}}\)) are calculated for \([0, 2\pi]\) and integrated in an array. Figure 11(a) shows the \(C_{\text{free}}\) valleys and the position of the point considered for a visibility scan in a sample problem. Figure 11(b) plots the radial Euclidean distances from that point to its surroundings.

Subsequent to the calculation of distances (ρ)
between the vantage point and \( \partial C_{\text{free}} \) for any angle \( \theta \in [0,2\pi) \), we convert and map these data to Cartesian coordinates [Figure 12(a)].

Since the \( \partial C_{\text{free}} \) boundary generally has a complex geometrical shape and lacks definite vertices as in polygonal objects, we take advantage of the ray sweeping data to determine the boundary points being tangent to any ray emanated from the vision source point. The tangent ray is defined as follows:

**Definition 4:** Let \( q \) be the ray sweeping source point, \( X \) a point on the \( \partial C_{\text{free}} \) boundary, and \( p_X \) a ray passing through \( q \) and \( X \). Then \( p_X \) is tangent to \( \partial C_{\text{free}} \) at \( X \) if and only if in a neighborhood \( U \) of \( X \) the interior of \( C_{\text{free}} \) lies entirely on a single side of \( p_X \).

In order to find the tangent rays and their touching boundary points, we apply a difference function for successive adjacent rays. We define the ray difference variables as \( \Delta p_\theta = p_{\theta+1} - p_\theta \) for \( \theta \in [0,2\pi] \) and integrate them in an array plotted in Figure 12(b). By applying a high-pass band filter, the peaks of the ray difference array are determined. These peaks imply abrupt and large differences in successive ray magnitudes and therefore indicate the points where sweeping rays leave (positive peaks) or meet (negative peaks) a convex contour on \( \partial C_{\text{free}} \) (based on anticlockwise rotation of rays) [Figure 13(a)]. The boundary points corresponding to the tangent rays are treated as boundary vertices visible from the vantage point, \( q \). These points are called critical points and form the set \( R(q) \) (see step S1-d in Section 2). The tangent rays and critical points are shown in Figure 13(b).

**Figure 11.** (a) An obstacle-free network of valleys. The point for which the visibility scan must be executed is located in the lower-right part of the free space. (b) Polar representation of radial distance (i.e., ray magnitudes) of the point from \( C_{\text{free}} \) boundary.

**Figure 12.** (a) The Cartesian representation of the magnitudes of rays in Figure 11(b). (b) Magnitude difference of sweeping rays for successive angles. The three peaks show tangent rays.
By concurrently implementing the visibility scan for both ends of Traj(S) and Traj(G), we discover that either there exists a line which connects the two trajectories (and lies entirely in \( C_{\text{free}} \)), or none of them is within the scope of the other’s endpoint. If the first case holds, then the search phase terminates. For the latter case, critical points of the two sets \( R(p) \) and \( R(q) \) are calculated and matched to find the closest pair, one point from each. These points determine the two positions which the two trajectories must extend toward.

4.2. Valley Filling

As soon as new points are appended to the trajectories, the navigated valleys must be distinguished by “elevating” their depths in order to prevent the robot to retraverse them later.

The valley filling technique is somehow a micro-visibility process; it flags the neighboring configurations as “seen,” and excludes them from the search space. This process is comparable to walking in a long corridor while trying to get out by reaching an open door or a junction. Naturally one does not consider the tiles across the corridor and near his feet as promising cells leading to a desired destination. Rather, he deems those points as traversed (though physically not indeed), and continues his wall-following motion. This is done in filling technique by “elevating” the potentials of those cells, making them less attractive. Since in a steepest descent context the robot occupies the cell with the least potential value across the valley, the filling procedure does not affect the path length adversely.

The filling procedure is applied immediately after the visibility scan.

**Figure 13.** (a) The total visible configurations from the vantage point. (b) The tangent rays and their corresponding boundary vertices (critical points).

Figure 14. Three iterations from the valley-filling process. As new points (black dots) are appended to the trajectory, the cells across the channel are elevated in potential, so that the planner is encouraged to move along the valley’s main direction. Points on the medial axis are shown in white, except for the point \( q \) which is nearest to trajectory’s endpoint \( p \) (shown in black). The elevated rack is highlighted in each iteration.
after a new point is appended to a trajectory. So it is performed in a layer-by-layer manner. Suppose that a point \( p \) is just added to an existing trajectory array [Figure 14(a)]. In order to mark and elevate the potentials of grid-cells across the \( C_{\text{free}} \) valley, we must find a line passing from \( p \) and perpendicular to the local direction of the channel. To do this, the point \( p \) must be connected to its nearest point \( q \) on the medial axis (skeleton) of the valley. Through an interpolation and extrapolation, the cells along this line are found and increased in potential. The amount of this increase is set to about \( \frac{1}{3} \) of the valley depth [i.e., \( s \) in (10)]. Figure 14 shows three consecutive iterations of filling operation.

For a better understanding of the function of this process, imagine that an attractive potential (i.e., a local minimum) is located in the upper-end of the narrow channel in Figure 14(a). Then according to the steepest descent search, the trajectory points should move towards it, which is of course hopeless. However, the elevated barrier created in each iteration blocks this motion, and forces the planner to take a mildest ascent step and run off the fatal situation.

For channels of uniform width, this method fills the cells thoroughly and compactly, but it may cause porosities in curved and bent valleys, or leave unfilled areas behind, as in Figure 14 or 15(b). The case in Figure 15(b) stems from the fact that for two successive trajectory points, their respective nearest medial axis points are not adjacent.

Although this does not cause a serious problem most of the time, we will present a variation to this procedure to overcome such conditions: First a square (or rectangular) frame with a symmetrical center on the medial point \( q \) is defined [dashed line in Figure 15(c)]. This frame is partitioned into two hyperplanes by the connecting line \( pq \). The hyperplane that contains the penultimate trajectory point is therefore the “backward” region which may contain some unfilled cells. Next, elevate the potentials of those cells confined within the frame and valley border. The magnitude of this frame can be set such that all the unfilled cells can be covered. However, a size equal to the valley width in that point suffices. The yet unfilled area in Figure 15(c) will not cause any problem since it is far from trajectory points.

The implemented valley filling routine provides some advantages for our algorithm. First, it reduces the potential searching time significantly by discarding the configurations in \( C_{\text{free}} \) which have normal vectors pointing toward a local minimum, and so obviates the random or “Brownian” steps. Second, this technique enables the planner to perform a “hill climbing” operation for coping with the attraction of a nearby local minimum and, as such, is a subtle way to avoid exhaustively filling up dead-ends or saddle point regions and the consequent path smoothing operations.\(^{19} \)

Mathematically speaking, suppose that the planner incrementally builds up a search tree and adopts a best first strategy to find the goal point. This task becomes time-consuming when the tree has many branches. Now the valley filling process curtails most of the nonpromising branches and directs the planner along an effective branch leading to another valley. In other words, this technique converts a breadth-first or best-first search into a depth-first search.

Our experiments showed that the filling process aids the robot considerably, especially in departing from deep local minimum wells.

\( \text{Figure 15.} \) An unfilled area is originated from the fact that for two successive trajectory points, their respective nearest medial axis points are not adjacent. To resolve this problem, a frame is defined around the medial point \( q \) (drawn by dashed line), and the unfilled area confined within this frame is elevated in potential.
4.3. Potential Field Module

The bidirectional nature of the VVP algorithm requires that for each iteration, the valley potentials manifold be numerically added to a paraboloid with a minimum apex on a temporary goal point (see Section 2, step S4). For instance, when extending Traj(S), the temporary goal is to reach the endpoint of Traj(G), and vice versa. Figure 16 shows the summation of a paraboloid function to the valley potentials.

To maintain the movement of the robot, we take advantage of the descent search method, which is the simplest and fastest searching strategy in numerical contexts. We define the neighborhood of each cell to be 2-neighbors. Points defined by \((x \pm 1, y \pm 1)\) are 2-neighbors of \((x, y)\). The number of neighbors of a cell is thus \(3^2 - 1 = 8\). For a \(k\)-dimensional space, it would be \(3^k - 1\).

The search begins at the start point, with examining all the neighbors of current cell. The algorithm chooses the neighboring cell with lowest potential among all neighbors as the next location of robot. This is the simple steepest descent method, which is often prone to stopping at a local minimum. To cope with this problem, we permitted the algorithm to take ascending steps, in order to be able to exit from local minima. The amount of ascension is kept minimal. So the concept used here is “steepest descent, mildest ascent” motion. This step is comparable to the random walks made to extricate the robot from a trap or relocate it in another region of the potential function, but is more directed than the random walk.

However, in order to prevent the robot from looping (i.e., the perpetual fluctuation of robot between two neighboring cells), we assign a relatively high potential to all visited cells, but still lower than the potentials of points in \(C_{obs}\) (see valley filling in Section 4.2). Therefore the robot may not return immediately to a local minimum after it has been once there, simply because it is not a local minimum anymore! The height to which a visited point is elevated has been experimentally set to around \(\frac{1}{2}\) of the valley depth. Figure 17 illustrates a scene from potential search. The trajectory points are chosen on the basis of the least potential among neighboring cells.

5. EXPERIMENTATION

In this section we intend to demonstrate the algorithm’s function through examples. Figure 18 illustrates the path planning process for a sample problem.

After preparing the valley potentials [Figures 18(a)–18(c)], the search phase is accomplished in three iterations. The bidirectional progression of trajectories is plainly shown in Figures 18(d)–18(f). The tangent rays in the visibility scan are drawn in black, and the points created by potential field module are shown white. The \(C_{free}\) region is light-colored, and the “filled” area has a darker shade. Figure 18(d) indicates the development of Traj(S) (upper-right), and Traj(G) (lower-left) trajectories in iteration 1, by first performing a visibility scan, then a potential field search. The visibility scan matches with case S1-d, where none of the two trajectories is in the scope of another. Hence, six possible pairs of critical points (two for G, and three for S) are evaluated and the closest pair is selected as the destination of trajectories. The filling procedure is then implemented for the drawn lines (darker area in \(C_{free}\) according to step S3).

The potential field module now starts with performing a steepest descent mildest ascent search from
the endpoint of \( \text{Traj}(S) \) toward the endpoint of \( \text{Traj}(G) \), the temporary goal. This requires a superimposition of a paraboloid function with a minimum apex on \( \text{END}(\text{Traj}(G)) \) on the “flat” potential manifold in Figure 18(c) (step S4). This search generates points directed to the temporary goal, elevates the potentials across the current valley, and stops after a few repetitions upon detaching enough from the \( \partial \mathcal{C}_{\text{free}} \) (case S6-b). These points are appended to \( \text{Traj}(S) \).

The same operation is carried on from \( \text{END}(\text{Traj}(G)) \) to the new endpoint of \( \text{Traj}(S) \), which now includes recently added potential search points. Note that in Figure 18(d), due to the filling operation executed before and during the potential field module, the steepest descent search does not “fill up” the nearby minimum well and thus avoids an entrapment in local minimum around point \( G \). Rather, it utilizes the mildest around point ascent concept, and exhibits a hill climbing behavior. This case shows the importance and effectiveness of the filling procedure, which helps the planner substantially through the whole process.

Figure 18(e) illustrates the second iteration. It is performed in the same fashion as the first iteration. Note the wall-following function of the potential module before detachment from \( \mathcal{C}_{\text{free}} \) border. Figure 18(f) displays the case S1-c occurred in third iteration, where both trajectories are being seen by each other’s endpoints. By applying the criterion (1) it becomes evident that the endpoint of \( \text{Traj}(S) \) must be connected to a visible point in \( \text{Traj}(G) \) with least distance to \( G \) [i.e., having the least ordinal rank in a visible subset of \( \text{Traj}(G) \)]. The remaining points to the end of \( \text{Traj}(G) \) are truncated afterwards.

Eventually the reversely ordered \( \text{Traj}(G) \) is concatenated to the \( \text{Traj}(S) \) and yields the final path from start to goal [Figure 19(a)].

Now we are going to present another example aiming to highlight the effects of the pruning procedure. Figure 19(b) introduces the workspace and start and goal points for this problem. This variant differs slightly from the previous example. The pruned generalized Voronoi diagram and \( \alpha \)-MID region are depicted in Figure 20(a). Despite the minor modification of the workspace, the region varies obviously due to the pruning and exclusion of dead-end-leading branches of the GVD. The remaining steps are performed based on this abridged region, and we find the final path in three iterations [Figures 20(c)–20(e)].

Through the above two examples we tried to clarify the VVP algorithm’s function, as well as
emphasize the pivotal role of the pruning procedure in reducing the search space. Other examples are presented in Figure 21 to demonstrate the quality of the generated paths for maze-like problems. The meeting point of two trajectories is shown by a color contrast. The search took 7 s and five iterations for Figure 21(a), and 5 s and three iterations for Figure 21(b). It should be noted that we gain the same final path when the start and goal points are swapped.

6. COMPARISON

In order to evaluate our algorithm and compare it with existing methods, we categorized path planning problems of different sizes and types in four groups, 1 to 5 obstacles, 6 to 10 obstacles, 11 to 20 obstacles, and maze-like problems, and designed five problems for each category and solved by different methods. The obstacles are convex and are either disjoint or combined with each other to form complex concave shapes [e.g., the problem discussed in Figure 18(a) lies in the second group, with 7 convex obstacles].

Table I shows the result of our experimentation for VVP algorithm, where 20 different problems are solved and the mean CPU time of each category is calculated for every single component.

The experiments were run in MATLAB™ context.
with a 1400 MHz AMD Athlon™ processor. Since the program was not compiled into machine language and each component was run as a script file, the computation time could still be improved.

As it is observed, the preprocessing times increase with the complication of workspace. Since for more complex problems we expanded the workspace dimension and obstacles’ number and intricacy, the grid-size and GVD length also grew consequently. Therefore, the mean CPU times for pruning and constructing α-MID region and valley potentials increased as well (almost linearly). In other words, it is mainly the resolution of the workspace representation grid that determines the preprocessing time. However, the searching phase did not seem to be affected greatly by the complexity. The number of iterations also remained within normal limits.

In order to compare the VVP algorithm with its parent methods, i.e., visibility graph, Voronoi diagram, and potential fields approaches, we solved the 20 problems in different categories mentioned earlier by these four techniques and calculated the lengths of produced paths. Then for each problem, the lengths of paths were normalized through a uniform scale to set up a proper benchmark for comparison. Because the paths generated by the visibility graph technique are optimal, we assumed their length as base index (=100) and calculated the proportion of other paths to the base indices. Although the visibility graph method yields the shortest path, its computational time is high (see Section 1.2 for time complexity). Figure 22 sketches our results. The value for α in the VVP algorithm was set to 0.7 and the distance of influence for obstacles in the potential field method was a 5% offset from obstacle borders.

Table I. Mean processing times (in seconds) for different problem categories computed for each component of the VVP algorithm.

<table>
<thead>
<tr>
<th>No. of convex obstacles</th>
<th>Preprocessing phase</th>
<th>Search phase</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Generalized Voronoi diagram</td>
<td>Pruning generalized Voronoi diagram</td>
</tr>
<tr>
<td>1–5</td>
<td>0.32</td>
<td>0.60</td>
</tr>
<tr>
<td>6–10</td>
<td>0.51</td>
<td>0.60</td>
</tr>
<tr>
<td>11–20</td>
<td>1.02</td>
<td>1.88</td>
</tr>
<tr>
<td>Mazes</td>
<td>2.04</td>
<td>5.68</td>
</tr>
<tr>
<td>Mean</td>
<td>0.97</td>
<td>2.19</td>
</tr>
</tbody>
</table>

Figure 21. Maze-like problems solved by the VVP algorithm. Two trajectories (with different colors) are extended toward each other.
found shorter paths than the Voronoi and potential fields methods most of the time. The path length in the VVP could be made even shorter by increasing the value of \( \alpha \), of course with the risk of safety loss.

As can be seen, the Voronoi diagram yields considerably longer paths for sparse environments, but is improved for mazes. Nevertheless, our algorithm performs steadily and produces shorter paths than the other two methods.

The potential field approach usually evaluated a great number of configurations and was directed by best-first search principle. The results presented here have undergone a postprocessing step. We also implemented random, as well as “Brownian,” motions techniques, for escaping from local minima, but since these methods are not complete, they did not produce a valid path within a reasonable time limit for a number of problems, especially for mazes, and therefore were not selected for comparison. Unlike most numerical potential field techniques where the generated path needs to be smoothed using classical variational calculus methods, our algorithm barely requires such postprocessing steps, which in turn improves the path planning time.

We also summarized the above results in Table II, where the mean values of path lengths are calculated for each category and compared. To evaluate the computational times for these algorithms refer to discussions in Table I and Sections 1.1, 1.2, 1.3, and 7.4.

To conclude our observations, we can state that our proposed algorithm offers an effective tradeoff between the shortest and safest paths in an order of subquadratic time complexity. The extent of this tradeoff is determined by manipulating the parameter \( \alpha \) when constructing the navigational area.

**Figure 22.** Relative path lengths generated by the visibility graph, Voronoi diagram, potential fields, and VVP algorithm methods for 20 problems in four categories. The lengths of the paths created by the visibility graph are taken as base indices and set to 100. The diagram shows the proportion of path lengths.

**Table II.** Path lengths (in equal units) generated by different approaches and their relative proportion.

<table>
<thead>
<tr>
<th>No. of convex obstacles</th>
<th>Visibility graph</th>
<th>Voronoi diagram</th>
<th>Potential fields</th>
<th>VVP algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Length</td>
<td>%</td>
<td>Length</td>
<td>%</td>
</tr>
<tr>
<td>1–5</td>
<td>16.68</td>
<td>100</td>
<td>23.32</td>
<td>139.8</td>
</tr>
<tr>
<td>6–10</td>
<td>20.25</td>
<td>100</td>
<td>25.23</td>
<td>124.6</td>
</tr>
<tr>
<td>11–20</td>
<td>25.96</td>
<td>100</td>
<td>34.42</td>
<td>132.6</td>
</tr>
<tr>
<td>Mazes</td>
<td>46.98</td>
<td>100</td>
<td>55.09</td>
<td>117.2</td>
</tr>
<tr>
<td>Mean</td>
<td>27.46</td>
<td>100</td>
<td>34.51</td>
<td>125.7</td>
</tr>
</tbody>
</table>
7. DISCUSSION

In this section we point out some issues related to the algorithm, such as its strategy to overcome local minima, the effectiveness of the applied bidirectional search, robustness, time complexity, and the possibility of generalizing to high-order spaces.

7.1. Dealing with Local Minima

The VVP algorithm’s strategy to bypass local minima is accomplished in two levels: macro and micro. In the macro level, realized by the procedure of GVD pruning, all Voronoi edges leading to obstacle boundaries are truncated. Since the $\alpha$-MIDs and subsequently all navigation valleys are constructed based on PGVD points, this operation has a substantial influence on the whole search phase. It significantly reduces the search space and hinders the robot’s future entrapments in local minima.

The micro-level local minimum resolving occurs during the steepest descent mildest ascent search. During the process of potential elevation of traversed configuration points, the local minimum point increases in potential value and therefore stops to act as a trapping well.

7.2. Global Search Approach

In primary versions of our algorithm we implemented a unidirectional search from start to goal. However, through experiments we noticed that this approach proved to be inefficient for a class of complex problems. The complication originated from the fact that the robot did not have a sufficient global vision of the C-space, and therefore was susceptible to deviate from promising routes and wander in other areas of search space.

This circumstance drove us to apply a bidirectional search method. The almost simultaneous movements of the start and goal configurations provide a rapid convergence for the algorithm, as well as a better quality for the path. The iterative nature of VVP algorithm generates intermediate milestones acting as temporary goal configurations. By decomposing the overall problem to a number of low level temporary-goal-reaching tasks, the algorithm is capable of resolving more complex problems. We observed that the bidirectional method exhibits a significantly higher performance than the unidirectional approach.

7.3. Robustness

The robustness and fault tolerance of the algorithm are reflected in its ability to overcome special situations which hamper the fluent progression of the algorithm. In the preprocessing phase, the erroneous and inexact development of navigation area may cause dysfunctions in search phase. Nevertheless, the pruning operation discards many redundant GVD branches, which may be connected to concave corners of obstacles, stretched toward dead-end passages, or generated due to noisy and inaccurate representation of the workspace (Section 3.3). The non-smoothness of the PGVD itself is corrected further by constructing the $\alpha$-MID region.

In the visibility module, when no convex contour of the $\partial_{\text{free}}$ is sensed, or the location of the vantage point is unsuitable or contiguous to the border, the visibility scan may return no critical point. However, the subsequent potential search settles this problem by either wall-following motion or localizing the next trajectory points strategically far enough from border.

The filling procedure in the potential field module also may bring about circumstances where all the eight neighboring cells (2-neighbors) of the current point are filled (i.e., elevated in potential), and therefore a reliable step cannot be taken. In this case the algorithm expands the neighborhood frame to 24 cells ($5 \times 5 - 1$) and searches it for the least potential. If it fails again, the potential search stops and the algorithm continues with the next iteration.

If for any reason one of the trajectories stops expanding further (this case hardly occurs), the algorithm will unidirectionally continue building the path till two trajectories meet.

7.4. Time Complexity

Due to the compound nature the VVP algorithm its time-complexity order cannot be calculated straightforwardly. Hence we have to decompose it to its modules and discuss each one singly.

In order to analyze the complexity of the preprocessing phase, it is essential to determine the time complexity of constructing and pruning the Voronoi diagram. For this purpose, we need first to study the complexity of the problem’s size. The following lemma deals with this issue:

**Lemma 1:** The Voronoi diagram has $O(n)$ many edges and vertices, in which $n$ is the number of Voronoi sites.

**Proof.** By the Euler formula for planar graphs,
the following relation holds for the numbers \( v, e, f \), and \( n \) of vertices, edges, faces, and connected components:

\[
v - e + f = 1 + c.
\]  

(11)

Since the Voronoi diagram is a connected graph, \( c = 1 \). The Voronoi graph has \( n + 1 \) faces; \( n \) for the number of Voronoi sites, and 1 for the exterior of the graph. So \( f = n + 1 \). From (11) we will have \( v - e = 2 - (n + 1) = 1 - n \). By noting that each face has at least 3 incident edges, and each edge is counted no more than twice in the counting process, we obtain

\[
e \geq 3(n + 1)/2,
\]  

(12)

\[
e \geq 3v/2,
\]  

(13)

\[
v \leq 2n - 2,
\]  

(14)

\[
e \leq 3n - 3.
\]  

(15)

Adding up the numbers of edges contained in the boundaries of all \( n + 1 \) faces results in \( 2e \leq 6n - 6 \), because each edge is again counted twice. Thus, the average number of edges in the boundary of a Voronoi region is bounded by \((6n - 6)/(n + 1)<6\). The same bound applies to Voronoi graph.

The following lemmas and the subsequent theorem deal with the time complexity analysis of the compound algorithm.

**Lemma 2**: The time complexity of the preprocessing phase of the VVP compound algorithm is \( O(n \log n) \).

**Proof**: The preprocessing phase can be broken down into four basic operations: (i) GVD construction, (ii) Pruning operation, (iii) \( \alpha \)-MID region computation, and (iv) Valley potential values calculation.

The generalized Voronoi diagram can be constructed in \( O(n \log n) \) time by a retraction approach. Since the number of Voronoi vertices is in \( O(n) \), and the calculation of neighborhood degrees (see definition 2) of all vertices takes \( O(n) \) time. This gives the time order for pruning procedure.

The time required for establishing the \( \alpha \)-MID region is \((L/\lambda) \times O(1)\), where \( L \) is the total length of PGVD edges, and \( \lambda \) is the discretization resolution of Voronoi edges (note that the environment has distributed representation). The time required to calculate the valley potentials is constant for each workspace grid point. Therefore the total time complexity for the preprocessing phase is in the order of \( O(n \log n) \).

**Lemma 3**: The time complexity of the searching phase of the VVP compound algorithm is \( O(n) \).

**Proof**: The searching phase has two main modules executed \( k \) times (\( k \) is the number of iteration): (1) visibility module, and (2) potential search module.

During the search phase, the visibility radial sweep operation is performed 2 times for each iteration (for Traj(S) and Traj(G) separately). Because the radial scan has constant time complexity (depending on the number of sensors, i.e., radial rays), the time required for the visibility scan is \( 2k \times O(1) \).

In order to calculate the time required for pair checking in the visibility module, suppose that \( P(x) = \{ \rho_1, \rho_2, \ldots, \rho_{\ell} \} \) is the set of visibility rays emanated from the robot at location \( x \). Let \( H(B_i) = \text{closure}(B_i[i \in m]) \), where \( B_i \) is the \( i \)-th obstacle, \( m \) is the number of convex objects in \( C_{\text{obs}} \), and \( H(B) \) is the convex hull of the set. The set of rays extended from the robot and tangent to each convex obstacle is

\[
\text{Tr}(x) = \{ \rho_j \parallel \rho_j \cap H(B_j) \} = 1, \rho_j \in P(x), i \in m
\]  

(16)
time complexity is linear \((O(n))\) in the number of configurations in the grid and is independent of the number and shape of the obstacles. Therefore, it is concluded that the time complexity of the search phase is in the order of \(O(n)\).

The total time complexity of the VVP compound algorithm is now determined by theorem 2.

**Theorem 2:** The time complexity of the VVP compound algorithm is \(O(n \log n)\).

**Proof.** Lemmas 2 and 3 indicate that the time complexity of the VVP algorithm is \(O(n \log n)\) in the preprocessing phase, and \(O(n)\) in the search phase; therefore, the overall time complexity of the VVP algorithm is \(O(n \log n)\).

We are now able to compare the time complexities of different methods.

Table III provides the time complexity of construction and search procedures for four path planning methods.

The path length information is taken from Table II. We ranked each method according to its constructing and searching times, as well as its generated path’s length. Although the potential fields method solves the problems in a linear time order, its generated path is usually longer, and needs a post-processing step. On the other hand, although the visibility graph method finds the optimum path, the time for constructing and searching the graph is considerably high. This may not cause serious problems for simple polygons, but for real-world curved obstacles, their borders have to be approximated to numerous facets, which in turn increases the construction and search times.

The VVP compound method ranks high thanks to its low computational time and generated short paths. It truly takes advantage of the superiorities of its parent methods; that is, low construction time from the GVD, low search time from the PF, and short paths from the VG. It provides an effective balance between computational speed and path quality. By selecting different values for \(a\in(0,1)\), the VVP method acts similar to one of the above methods.

### 7.5. Extension to Higher Spaces

The VVP algorithm has the potential to be extended to three and higher dimensional spaces. Though the full \(n\)-dimensional implementation of the algorithm is among our future research, we will briefly discuss here the possibility of its extension to 3D.

Let us decompose the algorithm to its main building blocks; i.e., Voronoi diagram, maximal

<table>
<thead>
<tr>
<th>Path planning method</th>
<th>Constructing</th>
<th>Searching</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time complexity</td>
<td>Rank</td>
</tr>
<tr>
<td>GVD</td>
<td>(O(n \log n))</td>
<td>2</td>
</tr>
<tr>
<td>PF</td>
<td>(O(n))</td>
<td>1</td>
</tr>
<tr>
<td>VG</td>
<td>(O(n^2))</td>
<td>4</td>
</tr>
<tr>
<td>VVP</td>
<td>(O(n \log n))</td>
<td>2</td>
</tr>
</tbody>
</table>

*After post-processing and path smoothing.
inscribed discs, visibility graph, and potential fields. Each of these elements can be generalized to higher spaces:

(i) The Voronoi diagram in 3D space is the set of polygons (facets) being the loci of points equidistant from obstacle surfaces. A number of algorithms have been developed for 3D Voronoi computation with $O(n^2)$ time complexity. This type of Voronoi generalization offers an infinite number of possible navigable paths through Voronoi surfaces, which increases the search space tremendously. To avoid this, an alternative method for generalizing the Voronoi diagram to higher spaces is adopted as follows:

**Definition 5**: A generalized Voronoi graph (GVG) in $n$-D space is the locus of points being equidistant from $n$ or more obstacle features.

This extension maintains the one-dimensional nature of the Voronoi roadmap, which consists of linear and curved segments in the space. Since the resulting graph connects only the vertices, and not the edges of the obstacles, it may have disjoint subgraphs. However, in order to retain its connectivity, the hierarchical generalized Voronoi graph (HGVG) is developed in Ref. 20, which takes advantage of some “bridge” edges (called GVG$^3$).

Figure 24 demonstrates a 3-dimensional environment and its medial axis. The MA (or GVG) is constructed incrementally using a new algorithm which is the 3D version of the work presented in ref. 15. In these algorithms, the obstacles are represented in centralized (hence, precise) form (see Section 3.1), but the MA is built on grid-points, which makes it compatible with the VVP algorithm. Note that the spatial lines are the edges of Voronoi facets.

(ii) Due to the one-dimensional nature of the GVG roadmap, the pruning procedure is still applicable to 3D context. Figure 25 depicts the result of pruning the GVG in Figure 24, after selecting arbitrary positions for start and goal points. Similar to the 2D case, the pruning procedure reduces the search space considerably in 3D too.

(iii) As noted in Section 3.4, maximal inscribed discs can easily be generalized to 3D space, resulting in maximal inscribed balls (MIBs). In the same manner, we can extend the $\alpha$-MID concept to $\alpha$-MIB concept by constructing spheres centered on Voronoi points. Figure 26 illustrates the union of all $\alpha$-maximal inscribed balls centered on the points of the pruned GVG and with $\alpha=0.5$. The produced region is a network of “hose-like” obstacle-free navigable channels. Greater values for $\alpha$ cause “fatter” tubes, and freer space for robot’s maneuvering.

Due to its one-dimensional nature, the generalized Voronoi graph usually “under-represents” the topology of the workspace, thus making a more limited subset of $C_{\text{free}}$ accessible for path planning. In some cases, it is an $n$-partite graph that does not reflect certain areas of the $C$-space at all. In other instances, to keep the maximum clearance, it generates lengthy and unnecessary detours, especially in sparse environments. Moreover, the sharp angles in intersection (or meet) points are permanent impediments to smooth navigation.

Now it is worth noting that the $\alpha$-MIB structure compensates most of these drawbacks. Since the $\alpha$-MIB system dilates each GVG branch according to its clearance from the obstacles and with a scale of $\alpha$, 
many channels will intersect and overlap, and thus occupy the emptiness between GVG branches (compare Figures 25 and 26). This operation also enables the planner to select short-cut and smooth paths for the robot.

(iv) The visibility scan in 3D can be applied via “sweep surfaces” instead of sweep rays in the 2D method. The robot should scan the space inside the $\alpha$-MID region to find tangent surfaces (compare with Definition 4). This will create visibility planes as supports for robot trajectory.

(v) The capability of the potential field approach for being generalized to higher spaces is indeed its prominent advantage. However, the steepest descent mildest ascent search will take more calculation time compared to the 2D case, as it must evaluate 26 neighboring points instead of 8 points.

The search phase can be performed similar to the 2-dimensional VVP method; i.e., the visibility and potential field modules will execute alternately, and the valley filling procedure will convert to “tube filling.”

We conclude from the above discussions that the extension of the VVP algorithm to at least 3D workspace is possible.

8. CONCLUSION

In this paper the authors have presented a new algorithm (VVP) for the path-planning problem. Three conventional and well-known techniques, i.e., visibility graph, Voronoi diagram, and potential fields, are integrated in a single architecture to provide a high quality path in a short time. The new ideas developed in this paper can be briefed as follows:

(i) The integration of the Voronoi diagram, maximal inscribed discs, visibility graph, and potential field concepts in a single path planning algorithm, for the first time.

(ii) Proposing a procedure to convert simple Voronoi diagrams to generalized Voronoi diagrams for distributed workspace representations.

(iii) Introducing the pruned generalized Voronoi diagram (PGVD) which is computed by truncating the edges of generalized Voronoi diagram that stretch toward cul-de-sacs.

(iv) Developing a robust $C_{free}$ region based on $\alpha$-maximal inscribed balls and constructing potential valleys where no repulsive forces are associated with obstacles.

(v) Implementing a steepest descent, mildest ascent search and potential elevation technique within the framework of the potential field approach to build a collision-free goal-directed path.

(vi) Implementing visibility scan–potential field search methods alternately and bidirectionally through constructive iterations.

(vii) Introducing the concept of $\alpha$-MIB region, and investigating the extension of the VVP algorithm to 3D workspace.
We have successfully implemented our proposed algorithm in assorted types of problems and have compared the results with the visibility graph, Voronoi diagram, and potential field approaches. The results indicate that the VVP algorithm offers an effective tradeoff between the shortest and safest paths in order of subquadratic time complexity.

REFERENCES