MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A. I. Memo No. 760 February, 1984

The Find-Path Problem in the Plane

Van-Duc Nguyen

Abstract. This paper presents a fast heuristic algorithm for planning collision-free paths of a moving robot in a cluttered planar workspace. The algorithm is based on describing the free space between the obstacles as a *network of linked cones*. Cones capture the *freeways* and the *bottle-necks* between the obstacles. Links capture the *connectivity* of the free space. Paths are computed by intersecting the *valid configuration volumes* of the moving robot inside these cones and inside the regions described by the links.

Acknowledgements. This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's Artificial Intelligence research is provided in part by the System Development Foundation, in part by the Office of Naval Research under Office of Naval Research contract N00014-81-K-0494, and in part by the Advanced Research Projects Agency under Office of Naval Research contracts N00014-80-C-0505 and N00014-82-K-0334.

Table of Contents

1. INTRODUCTION

1.1. The Find-Path Problem in the Plane

This paper addresses the problem of planning collision-free paths for a moving robot in a planar workspace cluttered by known obstacles. The goal is to build a Path Planner which:

* takes as inputs known positions of the obstacles in the workspace, and the start and goal configurations of the moving robot,

* then outputs a collision-free path of the robot between the start and goal configurations.

Figures 1.1 illustrate paths computed by the Path Planner. The planar workspace is bounded by four walls, and has five obstacles inside it. The paths are shown by displaying consecutive configurations of the moving robot. The starting and ending configurations of the robot are shown bold. With each path, we also display the trace of the center of the robot. You are invited to follow this trace to find out which translations and rotations the robot has made.

We assume that the moving robot is convex. A non-convex robot can be approximated by its convex hull [7]. All the obstacles are also assumed to be convex and non-overlapping. This is not a severe restriction since we can eliminate concavities in an obstacle by splitting it into smaller convex obstacles, or by taking its convex hull. We consider the workspace boundary as the juxtaposition of four separate walls, otherwise the square boundary would be a concave obstacle.

1.2. How the Path Planner Works

The Path Planner proceeds in two main steps:

1. Model the moving robot and describe the free space between the obstacles.

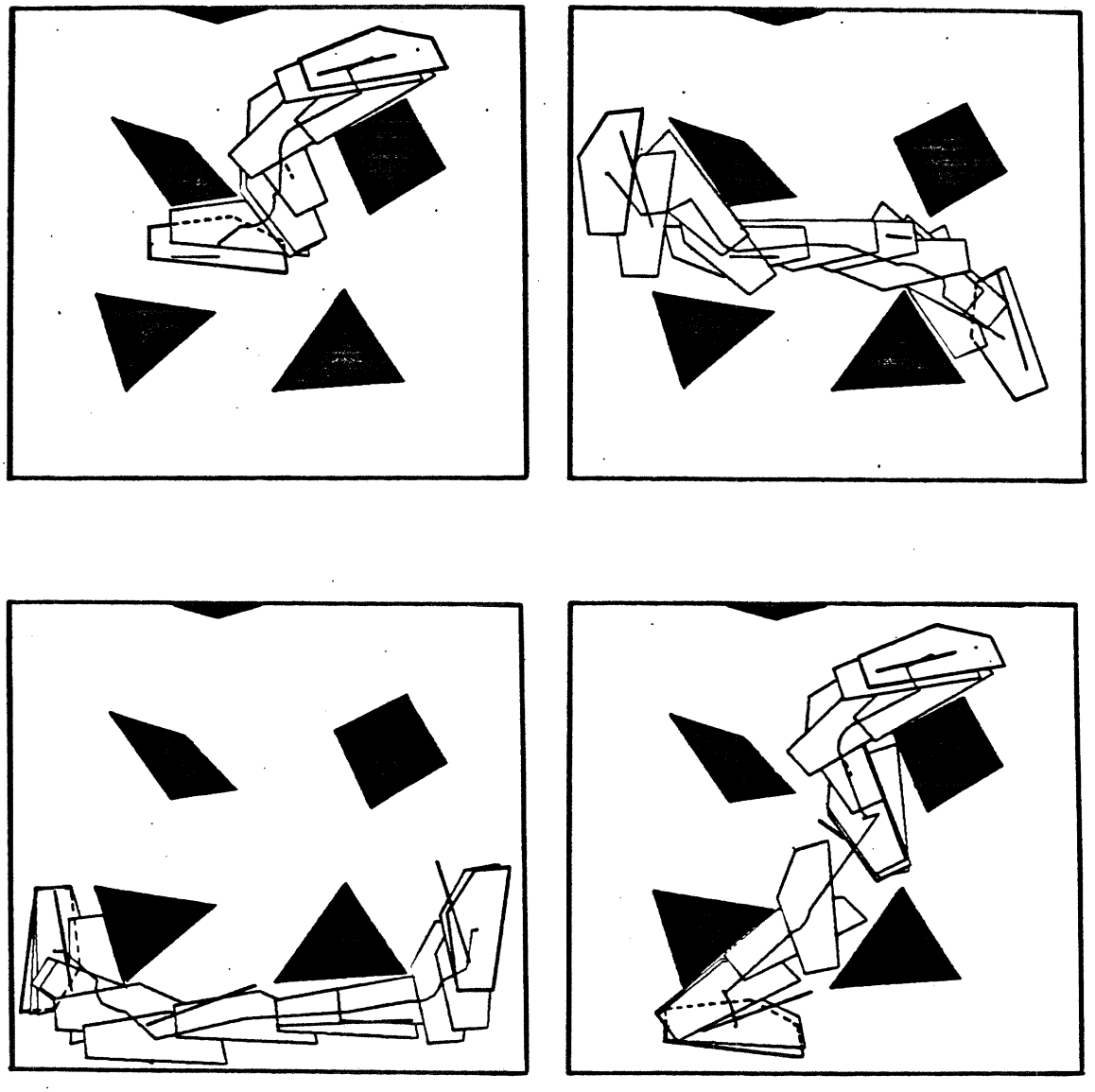
2. Compute a collision-free path.

The description of the robot and the free space can be seen as a preprocessing step for the second step.

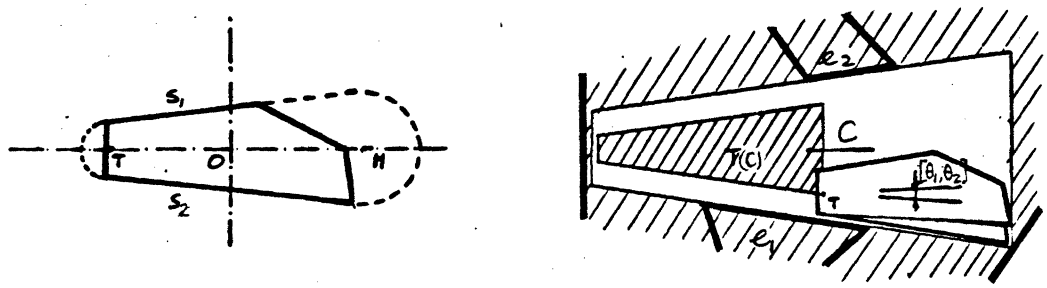
Model the Robot

We are motivated by the belief that the shape of the moving robot gives useful hints to the computation of paths. By modeling the moving robot, we find salient features such as the sliding edges, and the head and tail points of rotation. In figure 1.2.a, the sliding edges s_1 and s_2 in some way capture the long-versus-narrow shape of the moving robot M. The two sliding edges s_1 and s_2 suggest a useful range of orientation $[\theta_1, \theta_2]$ for which the robot M aligns itself with the length of the cone C. See figure 1.2.b. The range $[\theta_1, \theta_2]$ is called the alignment range of the robot inside the cone C. The robot will translate inside the cone at any orientation inside the alignment range.

Next, the rotation of the robot inside the cone is done about two points H and T called the head and tail of the robot. The points H and T are centers of the two



Figures 1.1 Some example paths between different start and final configurations of the robot.

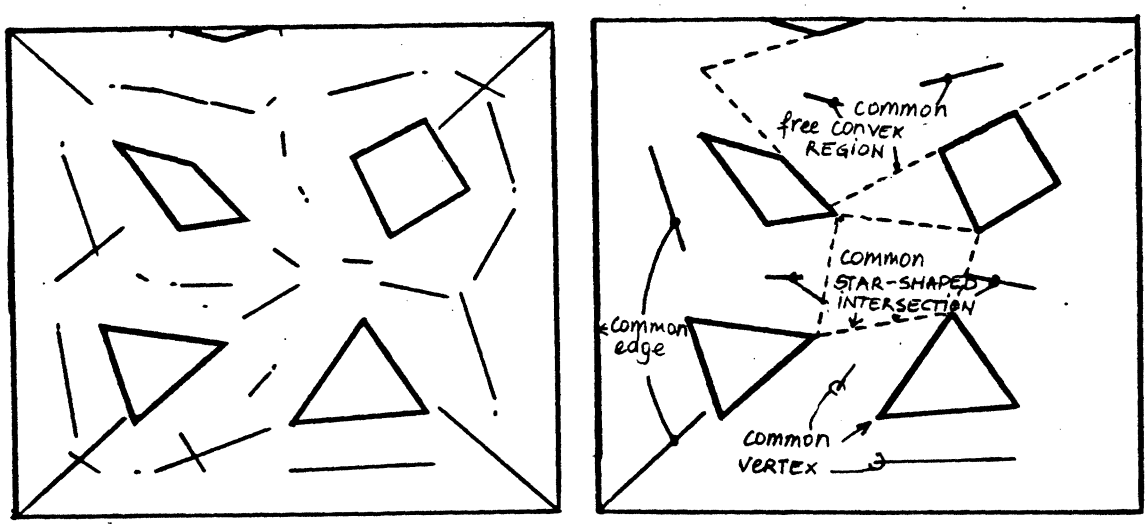


a. b.

Figures 1.2 The shape of the moving robot hints the set of valid motions inside a cone.

a. The sliding edges of the moving robot pinpoint the alignment range inside a cone. The robot will rotate about point H/T when it is inside a cone.

b. The set of valid motions of the robot inside a cone is composed of all rotations inside the alignment range, and all translation inside the translation polygon.



a. b.

Figures 1.3 Describing the free space as a network of linked cones.

a. The set of overlapping cones captures the free space between the obstacles.

b. Links capture the connectivity between the cones. Two cones are linked if they share a common vertex, or a common edge, or if they are part of a same free convex region, or a same star-shaped intersection.

the *alignment range* of the robot inside the cone C. The robot will translate inside the cone at any orientation inside the alignment range.

Next, the rotation of the robot inside the cone is done about either of the two points H and T, called the head and tail of the robot. The points H and T are centers of the two circles tangent to the sliding edges. See figure 1.2.a. They are also points for which the annulus swept by the robot (from θ_1 to θ_2) is small. In figure 1.2.b, the robot is shown rotating and translating inside a cone C. Rotation is done about the tail T and within the alignment range $[\theta_1, \theta_2]$. Translation is such that the tail T is within the shaded region, called the *translation polygon* $T(C)$. The sliding edges and the head/tail point of rotation capture a good range of valid configurations of the robot inside the cone C.

Describe the Free Space between the Obstacles

We believe that the free space between the obstacles gives useful hints to the computation of paths. The free space between the obstacles is described by a set of *overlapping cones*. A cone captures a *freeway* between two facing edges [1], or a *bottle-neck* between an edge and a facing corner. In figure 1.2.b, the cone C describes the freeway between the facing edges e_1 and e_2. A cone is bounded on the two sides by the two obstacle edges, and at the two ends either by two arcs of circle, or by two approximating straight lines. Figure 1.3.a shows all the overlapping cones which capture freeways and bottle-necks in the example free space. A freeway is pictured by a bisecting segment between the facing edges. Similarly, a very short segment pictures a bottle-neck between an edge and a facing corner.

These overlapping cones are linked into a graph. While a cone captures a convex local region, a *link* captures the *connectivity* between two nearby cones. Two cones are linked if they share a common edge, or a common corner. See figure 1.3.b. A common-edge link suggests a sliding motion of the robot inside some freeway along the common edge. This freeway is deducible from the free space description. A common-corner link suggests a rotation about the common corner, and inside some V-shaped region around the corner. The V-shaped region is computable from the free space description.

Two cones can also be linked if they are part of the same *free convex region* or the same *star-shaped intersection*.[3] A free convex region describes a local space in between many edges at a time. Figure 1.3.b illustrates a free convex region bounded by five facing edges. A path of the robot through a common free convex region consists generally of a realignment, followed by a translation inside the common convex region. A star-shaped intersection captures a region in between many vertices. It can be considered as the dual of a free convex region. In figure 1.3.b, we have an intersection bounded by four facing corners. The path of the robot through a star-shaped intersection is broken into paths about the two nearest vertices, and a path inside the intersection. We'll see in chapter 3 that the above two links are respectively the extension of common-edge and common-vertex links.

[3] See section 3.3, 3.4 for formal definitions of free convex regions, and star-shaped intersections.

The free space between the obstacles is described by a *network of linked cones*. The network of linked cones corresponds closely to the Voronoi diagram [5] for the same free space. We'll search this network of linked cones for a short path of the moving robot.

Compute a Path Using Four Experts and an A*-Search

The path of the moving robot is composed of paths inside the cones and paths along the links which connect nearby cones. Paths inside a cone are composed of all rotations 'inside' the alignment range, and all translations 'inside' the translation polygon. Paths along the links are computed by four experts corresponding to the four types of links. To move along a link between two cones, the robot typically:

* from the current cone, enters the transition region described by the link,

* translates and rotates inside the transition region aiming towards the next cone,

* from the transition region, enters the next cone.

The computation of a path along a link between two cones proceeds in three steps:

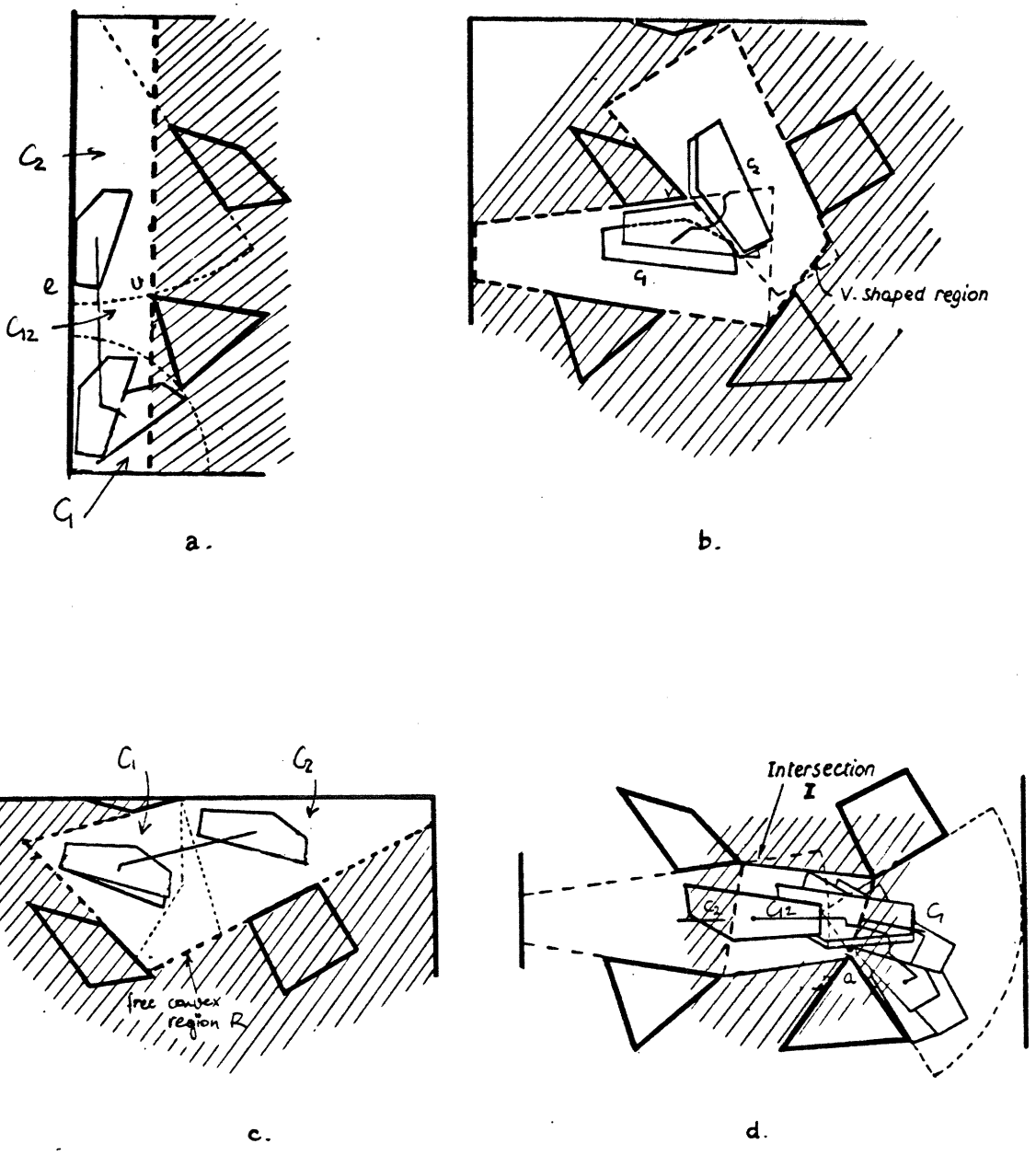
1. Find the transition region from the free space description.

2. Compute the set of valid configurations of the robot inside the transition region, and inside the next cone. We'll call CC_1, CC_2, and CR_{12}, the set of valid configurations, or valid *configuration volumes* of the robot in respectively the current cone, the next cone, and the transition region.[4]

3. Find a path of the robot by doing a search for the nearest intermediate configurations. The intermediate configurations are found by intersecting CC_1 with CR_{12}, and CR_{12} with CC_2. The path of the robot between two intermediate configurations is a straight-forward translation and rotation.

Figure 1.4.a illustrates a path along edge e from cone C_1 to cone C_2. Inside the cone C_1, the moving robot rotates to the nearest orientation θ_{12}, and translates until it is inside the transition cone C_{12}. Then, the robot translates inside the cone C_{12} until it is completely inside the next cone C_2. In our example, the free space for transition is the bottle-neck between edge e and corner v. This bottle-neck is described by the cone C_{12} which is found by finding the most constraining cone between the two cones C_1 and C_2. Since the two cones C_1 and C_2 share a common edge, there exists always an orientation θ_{12} common to the two alignment ranges of CC_1 and CC_2. CR_{12} is in this case the set of valid positions of the robot at that common orientation θ_{12}, and inside the transition cone C_{12}. We are thus reduced to intersecting the three translation polygons of CC_1, CR_{12}, and CC_2. This intersection gives us the intermediate configurations of the robot, and specifies a short path from C_1 to C_2.

[4] CC is the short for configuration cone, which is the valid configuration volume of the robot inside some cone. Similarly, CR is the short for the valid configuration volume of the robot inside a transition region.



Figures 1.4 Experts compute local paths from one cone to the next.

a. A path along an edge and through a bottle-neck.

b. A path around a corner.

c. A path through a free convex region.

d. A path through a star-shaped intersection.

Figure 1.4.b pictures a typical path around a corner V. First the robot approaches the corner V. Next, it rotates about V and some point on its sliding edge until it is in the nearest orientation valid in the cone C_2. Finally, it departs from the corner, and translates until it is completely inside the cone C_2. The free space of transition is the V-shaped region. This V-shaped region can be computed from the vertices and edges facing to corner V, and 'between' the cones C_1 and C_2. The V-shaped region serves as transition medium connecting C_1 to C_2.

Figure 1.4.c shows a path through a free convex region R. The robot rotates to the nearest orientation θ_{12} common to both alignment ranges of CC_1 and CC_2. Then it translates inside the convex region R until it is completely inside the next cone C_2. The convex region R is previously computed by the free space description, and so can be retrieved from the link. We demonstrate in chapter 4 that the common orientation θ_{12} always exists unless one or both the two configuration volumes CC_1 and CC_2 are empty. The computation of a path through a free convex region proceeds exactly as the computation of a path along a common edge and through some transition cone.

Figure 1.4.d shows a path through a star-shaped intersection I. The robot goes around vertex a and partially enters the intersection I. It then translates within the convex region formed by the cone C_2 and the intersection I until it reaches the next cone C_2. The free space of transition is the fake cone C_{12} which describes the freeway linking the facing ends of the two cones C_1 and C_2. The path is composed of:

* a subpath from cone C_1 to the fake cone C_{12} going around the fake corner a

* a subpath from the fake cone C_{12} to the cone C_2 going through the fake convex region R formed by the union of C_2 and I.

Paths between linked cones are computed by the four experts. A path between two arbitrary configurations is found by an A^*-search [13] which probes the network of cones, and computes paths between successive linked cones. In figures 1.1, we have displayed the intermediate cones and the start and final cones for each path. You are invited to decipher from these cones the type of link, and the identity of the expert used.

1.3. Main Features of the Path Planner

1. A good description of the free space between the obstacles. Cones, free convex regions, star-shaped intersections, and V-shaped regions aim at describing the local constraints of the free space. They capture large regions in which the robot can move. We have here a 2D description of the free space instead of the complete 3D description in the configuration space.

2. The matching of the features of the robot and of the free space results in good configuration volumes. Paths are found relatively fast by intersecting the

valid configuration volumes of the robot inside convex regions. Dealing with non convex regions such as V-shaped regions, is however difficult.

3. The Path Planner can find paths in very cluttered environments. See figure 5.7. In cluttered workspaces, cones are more useful than free convex regions, or star-shaped intersections. Paths along edges, and around corners are more crucial than paths through a convex region or star-shaped intersection. The reverse holds for sparse workspaces.

1.4. Relation to Other Work

We can distinguish the different algorithms[5] for solving the Find-Path problem based on:

1. The range of application. — The obstacles and the robot can be planar or spatial. The moving object can be an articulated arm or a moving robot.

2. The nature of the 'free space'. — In the literature, 'free space' can be the free multi-dimensional volumes of the configuration space, or the free planar/spatial regions between the obstacles.

3. How the constraints on motions are expressed. — Constraints can be expressed by capturing the CO or CI volumes in the configuration space. The CO volumes are also called grown obstacles. They describe the forbidden volumes for which the robot collides with the obstacles. CI volumes represent the configuration volumes for which the robot is collision-free.

Udupa's algorithm [20] is designed for the Stanford arm. To compute the CO volumes, Udupa approximates the boom and the forearm by cylinders. He then tesselates the joint space of the arm into collision-free regions. A path is found by recursively modifying the straight line path until it totally lies in the free joint space.

Moravec [12] bounded all the obstacles and the moving robot by circles. The moving circle is shrunk to its center and the obstacle circles are inversely expanded. The problem is reduced to finding the path for the center of the moving circle staying outside of the grown circles. Rotations are in this case ignored.

Lozano-Pérez and Wesley [11] find paths for cartesian manipulators and moving robots. The CO-obstacles represent the exact constraints on the position of the robot [9, 10]. In three dimensions, paths with translation only are found by searching through the free polyhedral cells which tesselate the xyz-space outside of the CO-obstacles. For the planar case, paths are searched through a Visibility graph connecting all the vertices of the CO-obstacles which 'see' each other. For paths with rotation, Lozano-Pérez splits the rotation range into a fixed number of slices, and within each slice compute the grown obstacles. These CO-obstacles are used to define several free space descriptions.

[5] A more detail overview of the mentioned algorithms can be found in [8]

Recently, Brooks and Lozano-Pérez [3] developed a subdivision algorithm for computing with the curved surfaces of the grown obstacles. The algorithm is general, and can find hard paths for 2D moving robots.

All the above algorithms partition the configuration space at different degree of complexity. Instead of tesselating the configuration space, Donald [4] divides the free space between the obstacles into non-overlapping regions. From these regions, a channel is suggested and a path within this channel is rectified and interpolated until it lies outside all the CO-obstacles.

Brooks [1] captures good freeways between the obstacles by overlapping generalized cones. Along the spine of a cone, a restricted range of orientation and translation of the moving robot is computed. This range of valid configurations of the moving robot is a restricted subset of the CI-volume of the robot moving inside the cone. The planar path is composed of translation along the spines of the cones, and rotation at the intersections of these spines.

Brooks [2] uses cones to find quick paths for the Puma arm. Cones describe freeways for the hand and payload ensemble, and the freeways for the upperarm in the configuration space. A path is found by concurrently searching the two freeway-spaces when the forearm is moved.

Schwartz and Sharir demonstrate the existence of a polynomial time algorithm for planning the path of a planar moving robot [15], and of a general hinged device [16].

Our work is the extension of Brooks' find-path algorithm using generalized cones. We aim at:

1. Capturing more and larger CI-volumes of the robot. To this end, we devise an elaborate description of the free space between the obstacles. The network of linked cones has many features of the Voronoi diagram of the same free space. There are many more CI-volumes because the local regions are multiply described by overlapping cones, free convex regions, V-shaped and star-shaped regions. The CI-volumes are larger because the features of the moving robot 'match' the features of the free space. An example of such match is the alignment range of the robot inside a cone.

2. Capturing a larger set of motions by using four different path experts. The experts 'match' the path computation and the free space of transition along a link.

2. MODELING THE ROBOT

2.1. The two Sliding Edges and the Main Axis

Definitions

The two *sliding edges* are the two edges perpendicular to which the robot M has the two smallest cross sections. The main axis is chosen to be the bisector of the small angle between the two sliding edges. This axis will be the x-axis of a frame F fixed to the moving robot. The x-axis is oriented from the small end to the big end of M. The y-axis will be the other perpendicular axis which goes through the center of the smallest enclosing circle. Figure 2.1.a illustrates.

The Sliding Edges Pin-point the Alignment Range Inside a Cone

Since the cross-section of M is one of the two smallest cross-sections along the sliding edge, we have the following corollary:

Corollary: For fixed translation, the robot sweeps one of the two smallest spaces if and only if one of its sliding edges is parallel to the direction of translation.

A cone C is assumed to be narrower between its two facing edges e_1 e_2 than between its two ends. We deduce that when M is inside a cone, M is constrained more on the two sides by the two edges of cone C than at the two ends, by the length of the cone, so M 'should align itself' with the spine of the cone. Moreover, it is desirable that robot M traverse the cone C from one end to the other. So from the above corollary, M should translate with one of its sliding edges parallel to the facing edge e_1 or e_2 of C, or parallel to any direction in between edges e_1 and e_2. This gives a heuristic range of orientation $[\theta_1, \theta_2]$ called the *alignment range*, and has the effect of aligning the long robot M with the length of the cone C.

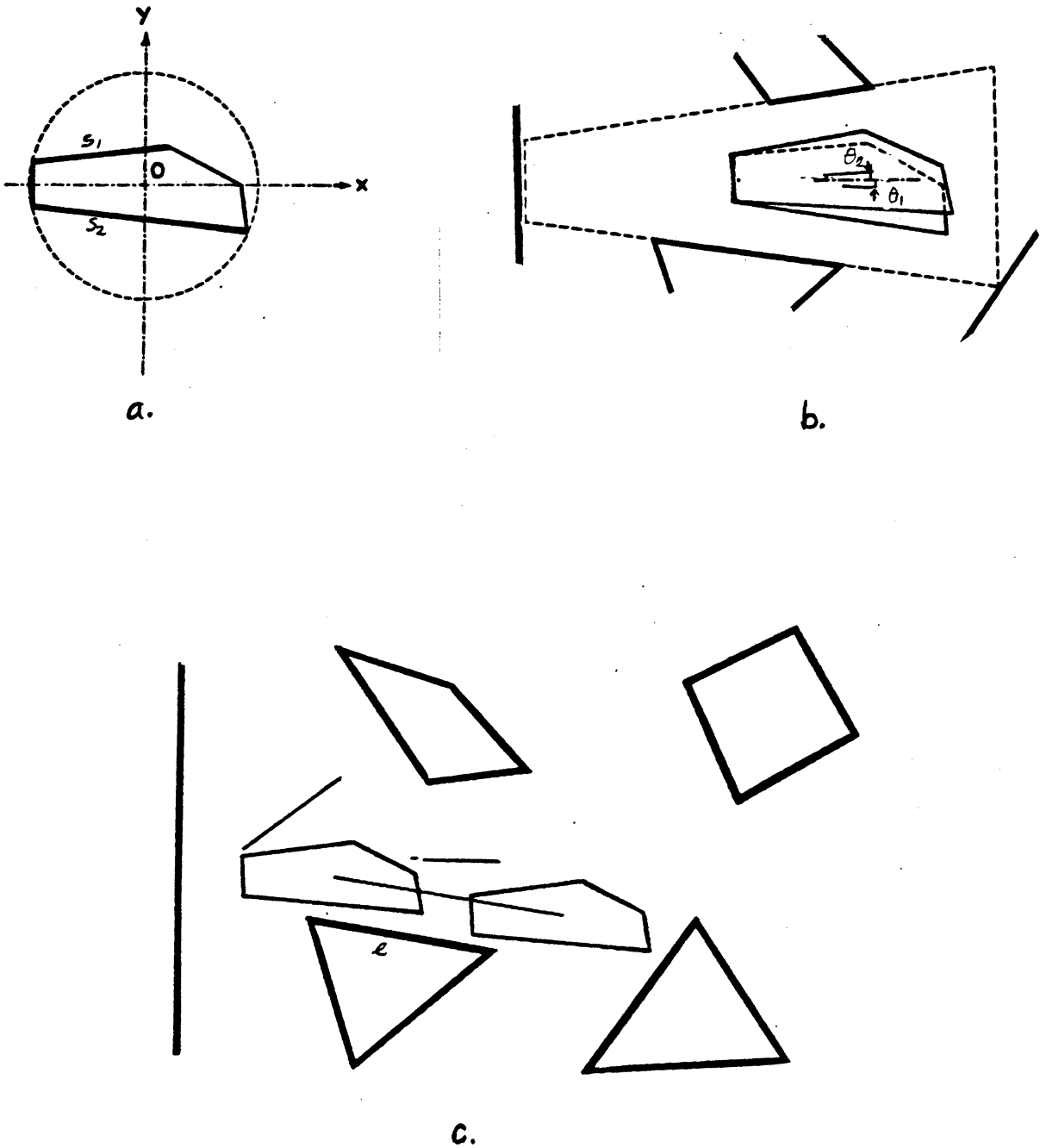
Sliding Heuristic: Inside a cone, the robot should align itself with the length of the cone. The alignment range $[\theta_1, \theta_2]$ is such that for orientation θ_1 or θ_2, the moving robot has one of its sliding edges parallel to one of the two facing edges of the cone.

Figure 2.1.b illustrates the alignment range of M inside a cone C. Figure 2.1.c shows an example where the Sliding heuristic helps find good paths. The two sliding edges can be computed in $O(|vertices(M)|)$ time using Shamos' diameter algorithm [17].

2.2. The Head and Tail Points of Rotation

Definitions

We designate by *head* (resp. *tail*) the big (resp. small) end of the moving robot. The head (resp. tail) also designates the point H (resp. T) about which the robot is rotating inside a cone. The points H and T are centers of two small arcs of circle which are tangent to the sliding edges and enclose the robot. We call these arcs the *head* and *tail bounding arcs*. See figure 2.2.a. H and T are also points for which the



Figures 2.1

a. The two sliding edges along which the robot has the two smallest cross sections.

b. The sliding edges and the alignment range of the robot inside a cone.

c. An example path of the robot sliding along an edge of an obstacle.

annulus swept by the robot (from θ_1 to θ_2) is not only easy to compute but also has a small convex hull.

Head-Tail Heuristic: Cones usually have a small end. To maximize the ability of the robot M to 'sneak through' this small end, the robot M should rotate about a point at one of its two ends. We choose these points of rotation to be the centers of the head and tail bounding arcs.

The sliding heuristic and the head-tail heuristic result in a combination of an alignment range $[\theta_1, \theta_2]$ and a *translation polygon* $T(C)$. This combination turns out to be a simple and powerful expression of the range of configurations (x, y, θ) of robot M in a cone C. See figure 2.2.b. The head and tail points of rotation can be computed in $O(|vertices(M)|)$ by finding the circles which are tangent to the sliding edges, and which go through the vertices of the robot. The tail/head point of rotation corresponds to the circular arc which includes the tail/head end of the robot.

2.3. Points of Rotation on the Sliding Edges

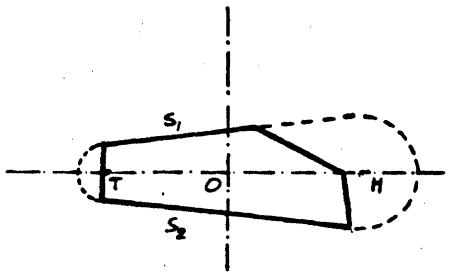
Navigation Around a Corner

A convex corner can be seen as a break[6] between two convex regions in which the planning of translational paths is relatively easy. The break becomes obvious from the fact that the two alignment ranges $[\theta_1, \theta_2]$ in the two nearest cones linked by the corner typically do not overlap. So circumventing a convex corner generally requires rotation of the moving robot from one range to the other. A simple approximation to the free space around the corner is a *disc segment* centered on that corner. Figure 2.3.a illustrate the approximation of the free space around a corner by a disc segment. For the same amount of rotation, we intuitively favor a rotation about the corner and some point on the sliding edge of M over a rotation about the center of the smallest enclosing circle. We conclude that rotation of M should be allowed about the corner and some point on the sliding edge of M. We'll accept this as the Corner heuristic.

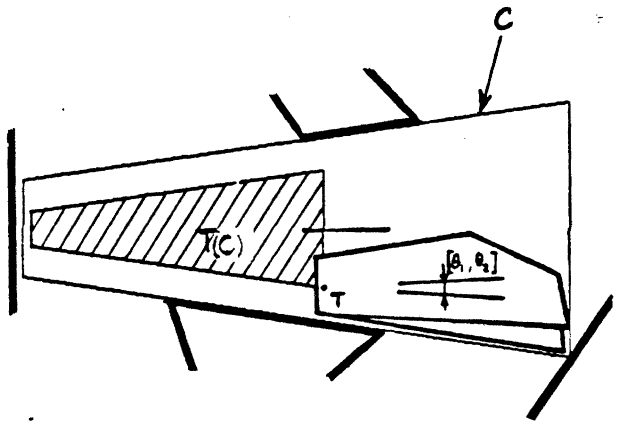
Corner Heuristic: Navigation around a convex corner should be achieved with a rotation about the corner and some point on the nearest sliding edge of the robot.

Figure 2.3.a illustrates an example of cornering about a vertex V. The transition path has a rotate-slide-rotate flavor. The robot corners the vertex V by first rotating about V and a point on the sliding edge, then it slides with the corner V in contact with the sliding edge, and finally rotates once more about V and a second point on the sliding edge.

[6] From another point of view, Lozano-Pérez [3, 9] differentiates two types of constraints. Type A constraint is the interaction of an edge of the moving robot with a corner of the obstacle. Type B constraint is the interaction of a vertex of the robot with an edge of the obstacle. So, a convex corner is a type A constraint 'separating' two type B constraints.



a.

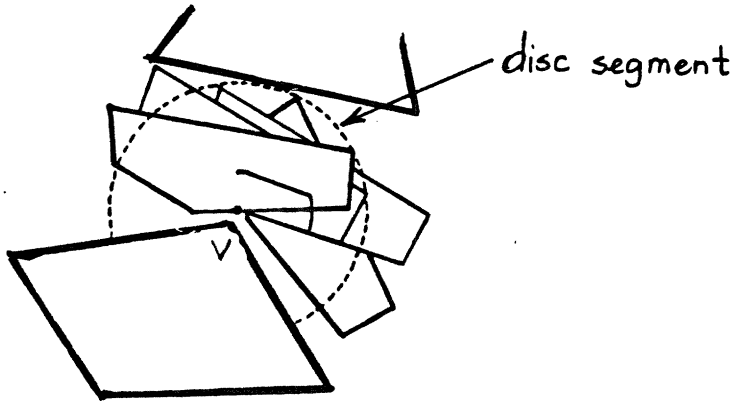


b.

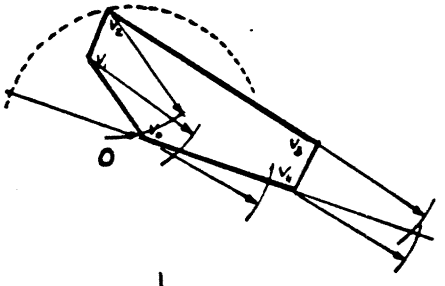
Figures 2.2

a. The head and tail bounding arcs.

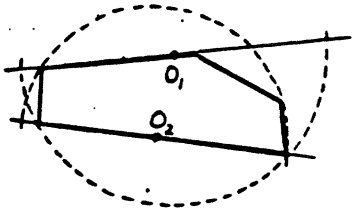
b. The configuration volume of the robot inside a cone is a combination of an alignment range and a translation polygon.



a.



b.



c.

Figures 2.3

a. The robot rotates, slides, and then rotates about a convex corner.

b. The Free Space determines the points of rotation on the two sliding edges.

c. The centers of the two smallest bounding half circles.

The Disc Segment About the Corner Pinpoints the Point of Rotation

The points of rotation on the sliding edge can be computed from the disc segment around the vertex in $O(|vertices(M)|)$ time. Let r be the radius of the disc segment which captures the free space around the vertex V. For each vertex v_i of M, we find on the sliding edge the center O_i of a circle having radius r, and going through v_i. Then we choose as the point of rotation the center of the half-circle which totally encloses the head (or tail) of M. See figure 2.3.b.

We can also enclose the robot by two smallest bounding half circles. The centers O_1 and O_2 of these half circles will be points of rotation for easy cornerings. Figure 2.3.c illustrates.

3. DESCRIPTION OF THE FREE SPACE

3.1. The Voronoi Diagram Describes the Free Space

This section presents the Voronoi diagram which is well known in Computational Geometry as the mathematical description of the free space between points and obstacles. Since we deal with 2D-obstacles, we'll consider the generalized Voronoi diagram [5] instead of the regular Voronoi diagram of a set of points.

Definition of the Voronoi Diagram

The *Voronoi diagram* of a set of convex obstacles is a decomposition of the free space between these obstacles into Voronoi regions R_i, such that each point in the interior of R_i is nearest to exactly one obstacle. Each Voronoi region is bounded on the outside by Voronoi edges, and on the inside by edges of the enclosed obstacle. All Voronoi regions are bounded because our workspace is bounded.

Figure 3.1.a illustrates the Voronoi diagram for our workspace example. The Voronoi edges are shown with curved lines, and can be seen as spines of the free space between the obstacles. The Voronoi diagram is sometimes viewed as describing the paths of maximal clearance for a point between the obstacles.

Characteristics

1. Each Voronoi region is closed and the boundaries of the Voronoi regions are connected. This results from the fact that we have a partition of a bounded workspace.

2. The interactions between edges and vertices produce three types of Voronoi edges. See figure 3.1.a.

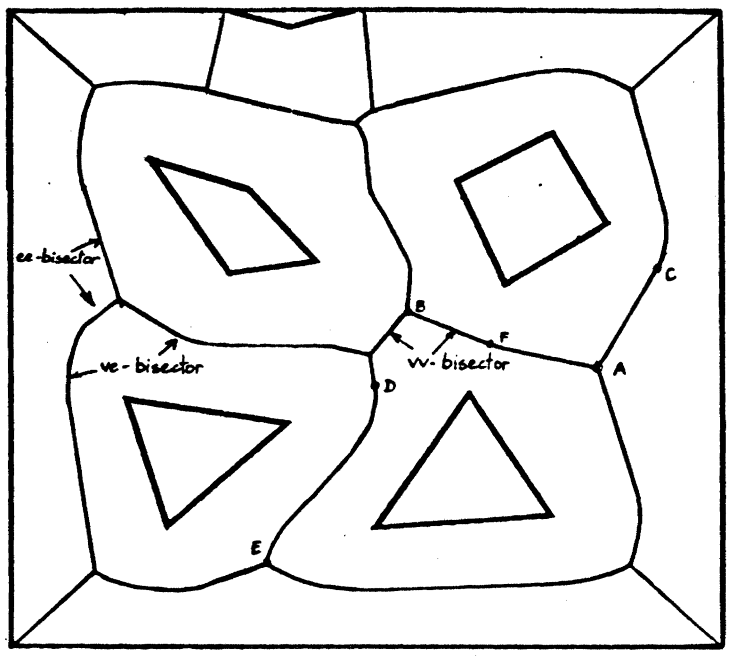
a) *ee-bisector*, or bisecting segment between two facing edges. — Each point on the bisecting segment is equidistant and nearest to two points respectively on the two facing edges. The ee-bisector splits the freeway between the two edges into two regions, each closest to one of the facing edges.

b) *vv-bisector*, or perpendicular bisector between two vertices. — The vv-bisector splits the free space around the two vertices into two regions closest to one vertex at a time.

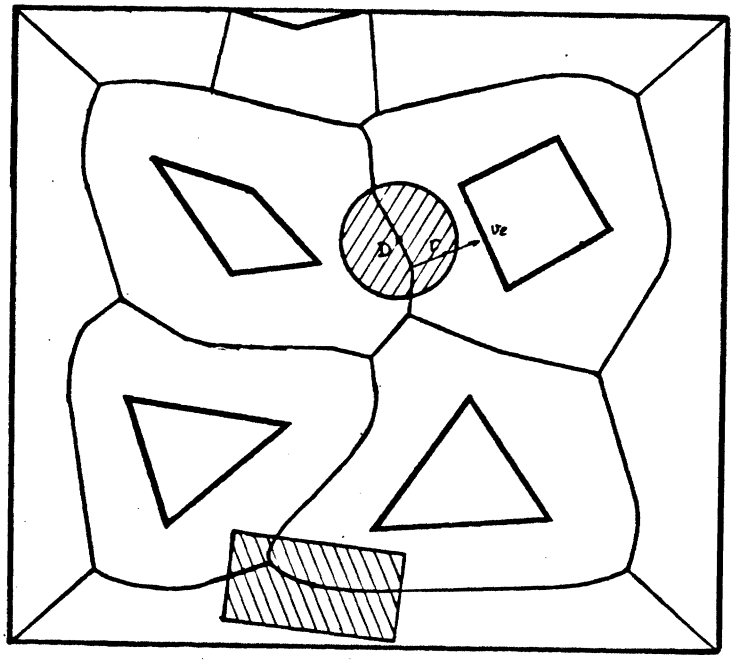
c) *ve-bisector*, or parabolic arc between an edge and a facing corner. — The arc of the parabola has an axis of symmetry going through the vertex, and perpendicular to the facing edge. We'll call an arc of a parabola which cuts its symmetry axis, that is, has arcs on both sides of its symmetry axis, a *2-sided* arc of parabola. A parabolic arc which does not cut its symmetry axis is called *1-sided* arc of parabola.

3. We have six types of intersections from the pairing of the Voronoi edges. See figure 3.1.a. for illustrations of the different types of intersection points: $a, ..., f$.

a) Intersection between two ee-bisectors. — This type of point tells us the presence of a convex region partially constrained by the corresponding facing edges.



a.



b.

Figures 3.1

a. The Voronoi diagram decomposes the free space into regions closest to various obstacles.

b. The Voronoi diagram hints at a path for a disc.

The most characteristic example is the intersection of the ee-bisectors of three facing edges. The point of intersection is the center of gravity of the triangular region enclosed by the three facing edges.

b) Intersection between two vv-bisectors. -- This type point indicates the presence of a star-shaped intersection, very much like a street intersection. The simplest example is the intersection of vv-bisectors from three facing vertices.

c) Intersection between an ee-bisector and a ve-bisector. -- Since the Voronoi boundaries and the obstacles are connected, we deduce that there must be a common edge. As we go through the intersection point, from the ee-bisector to the ve-bisector, we are constrained on one side by the same edge.

d) Intersection between a ve-bisector and a vv-bisector. — There must be a common vertex for the same reason as above. The picture is a transition from the vv-bisector to the ve-bisector via a common vertex.

e) Intersection between two ve-bisectors. — We can similarly prove that there must be either a common vertex or a common edge.

f) Intersection between an ee-bisector and a vv-bisector. — We can prove that the vertices must be on the corresponding edges. This intersection point can be seen as a superposition of a point of type c) and a point of type d).

The Voronoi Diagram as a Criterion for a Free Space Description

The set of Voronoi edges forms a network of legal paths for a disc staying as far away from the obstacles as possible [14]. We can plan path for a disc by sliding the center D of the disc on the Voronoi edges. See figure 3.1.b. The Voronoi diagram reduces the representation of the $2D$-space between the obstacles to the representation of the $1D$-space along the Voronoi edges. Associated with each Voronoi edge is a minimum free radius. This reduction is suitable only for discs because they are rotationally symmetric.

We can approximate a polygonal robot by a disc, and so plan the path of that disc along the edges of the Voronoi diagram. But still, the Voronoi diagram is of limited utility not only because of its severe limitations when the moving robot is long, but also because of its expensive computation,[7] and its high sensitivity to small local variations.[8] So, the Voronoi diagram will serve not as a direct description of the free space, but as a goal for any high level description of the free space. The following sections develop a description of the free space as a network of linked cones which functionally captures local regions of the free space, and most of the features of the Voronoi diagram.

[7] The Voronoi Diagram costs $O(n \log n)$, but the constant terms are high.[5]

[8] Cones are also sensitive to small variations although to a lesser degree. In these cases, cones are less useful than free convex regions and star-shaped intersections which describe clusters and branchings of small cones.

3.2. Cones Capture Freeways or Bottle-Necks in the Free Space

The cone is the most basic unit in our representation. It represents a convex local region of free space known as a *freeway* [1] or *bottle-neck*.

Definitions

An edge e divides the plane into two half-planes. One of the half-planes is outside of the obstacle which has edge e. We call this half-plane the *free half-plane* of edge e. The normal of edge e points outward to the free half-plane of e. Two edges e_1 and e_2 *face* one another if and only if part of e_1 is in the free half-plane of e_2 and vice versa.

A *cone* C is a region of free space constrained by two facing edges e_1 and e_2. These edges are called *generating edges* of the cone. If the two generating edges are not parallel, they will intersect at a point called the *peak* of the cone. The peak is chosen arbitrarily far on one side if the two generating edges are parallel. Each cone has an axis or *spine* δ which bisects the angle between the two generating edges e_1, e_2, and is inside cone C. We choose to orient the spine δ from the small end, or peak, to the big end of the cone. A cone is bounded on the two sides by the lines which extend the two generating edges, and at the two ends by two *bounding arcs* of circle centered on the peak. These two arcs are computed by pruning (see below) the cone with all nearby obstacles. See figures 3.2.

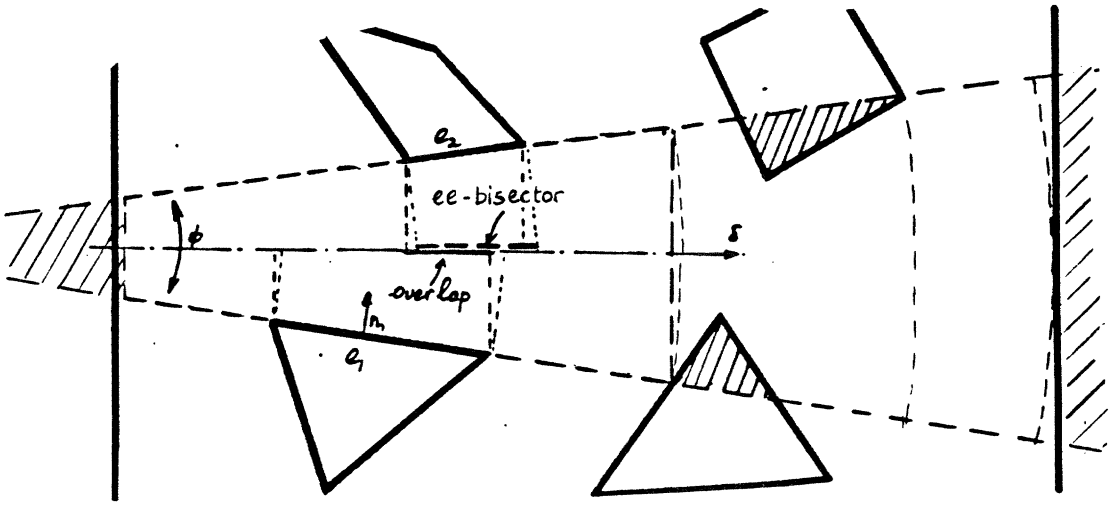
With the generating edges and the peak known, the cone region is completely specified by $[r_1, r_2]$, where r_1 and r_2 are the radii of the two arcs at the two ends of the cone. When the disk segment in between the two generating edges intersects with an obstacle, we compute the segment $[\rho_1, \rho_2]$ characterizing the annulus which overlaps with the obstacle. Then, we delete the segment $[\rho_1, \rho_2]$ from the cone region $[r_1, r_2]$. This is called the *pruning* of the cone with the obstacles.

We make the annular cone region convex by replacing the small arc of radius r_1 by a segment perpendicular to the spine δ and going through the point where the small arc cuts the spine. We can further make this convex boundary polygonal by replacing the big arc of radius r_2 by one or two segments.

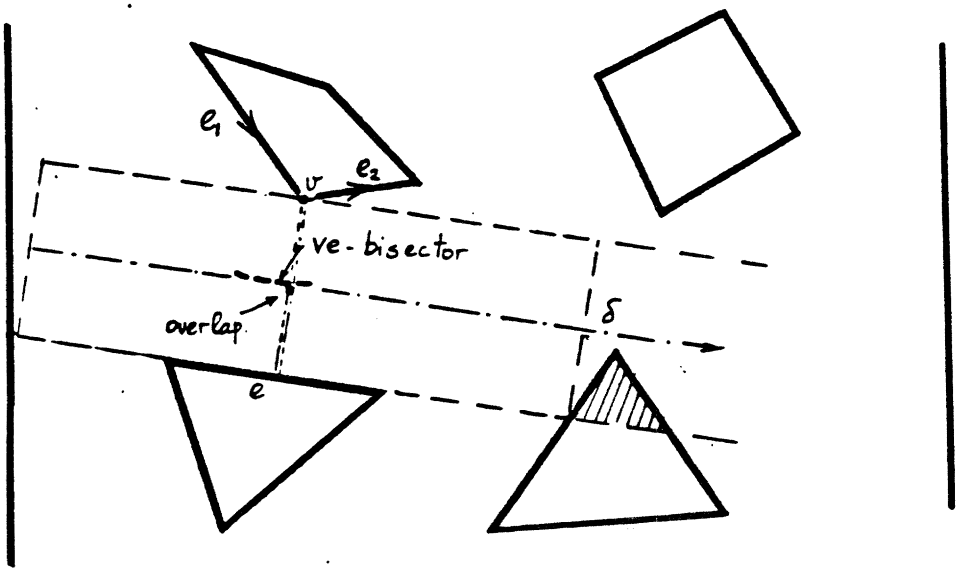
Properties

1. The two generating edges, e_1 and e_2, enclose a free region in between them. — The region has the shape of an annular segment centered at the peak. The annulus is bounded by e_1 and e_2, since there are portions of e_1 and e_2 on its boundary. After the pruning operation, this annulus is guaranteed to be obstacle-free.

2. The projections of the two generating edges on the spine of the cone overlap. — We project the edges e_1 and e_2 along a direction perpendicular to the spine δ. The overlap of the projections guarantees the existence of points on spine δ which are equidistant to points on the edges e_1 and e_2. The existence of the overlap is a criteria for discarding irrevelant cone regions. In figure 3.2.a, the annular segment on the far right is completely irrelevant, although obstacle-free. It can be shown



a.



b.

Figures 3.2

a. A cone describes a freeway between two facing edges.

b. A cone describes a bottle-neck between an edge and a facing corner.

that the set of 'freeway cones' which satisfy this overlap property describes all the ee-bisectors in the Voronoi diagram.

3. There is at least one point P on the spine of the cone C and inside C for which the distance from point P to one of the generating edges is less than the distance from P to the nearest bounding arc. In common terms, we can say that it is tighter between the generating edges than between the two ends of the cone. This will be known as the '*Narrowness Property*'.

4. There is a special case of a cone between an edge and a facing vertex. — This is a bottle-neck between the edge e and the vertex v. We see the vertex v as the limiting case of an edge e_v parallel to the edge e, with length shrinking to zero. From that observation, we create a fake generating edge e_v with almost zero length and direction parallel to the facing edge e. A cone is then built between the two generating edges e_v and e. It can be proved that the 'bottle-neck cones' capture all the 2-sided arcs of parabola in the Voronoi diagram.

Finding All Valid Cones in the Free Space

Algorithm:

• Cones between pairs of facing edges. — For each pair of obstacle edges e_1, e_2 do:

1. Check that edges e_1, e_2 face one another.

2. Compute the axis δ bisecting the angle between e_1 and e_2, and check that the projections of e_1 and e_2 on the spine δ overlap. We then build a cone C with the two generating edges e_1, e_2, and axis δ.

3. Prune cone C with all other obstacles. Each time, calculate the new bounding arcs centered at the peak of C.

4. If the region bounded between the small and big arcs is non null, output C.

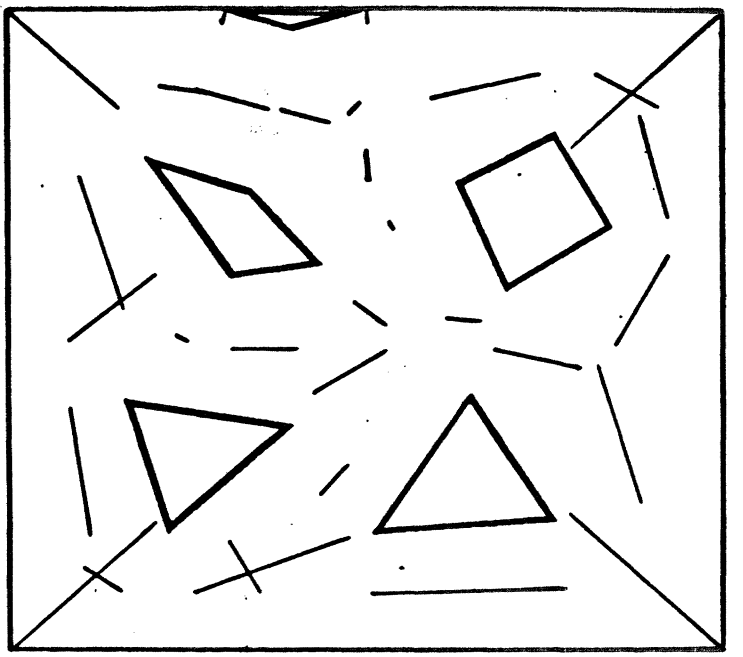
• Cones between an edge and a facing vertex. — For each pair of vertex v, edge e do:

1. Check that the vertex v is in the free half-plane of edge e. Then, look at the edges linked by vertex v, and check that we have a bottle-neck between edge e and vertex v. A simple check ensures that the dot product of the normal of edge e with the left directed edge e_1 be negative. Similarly, the dot product of the normal of e with the right directed edge e_2 must be positive.

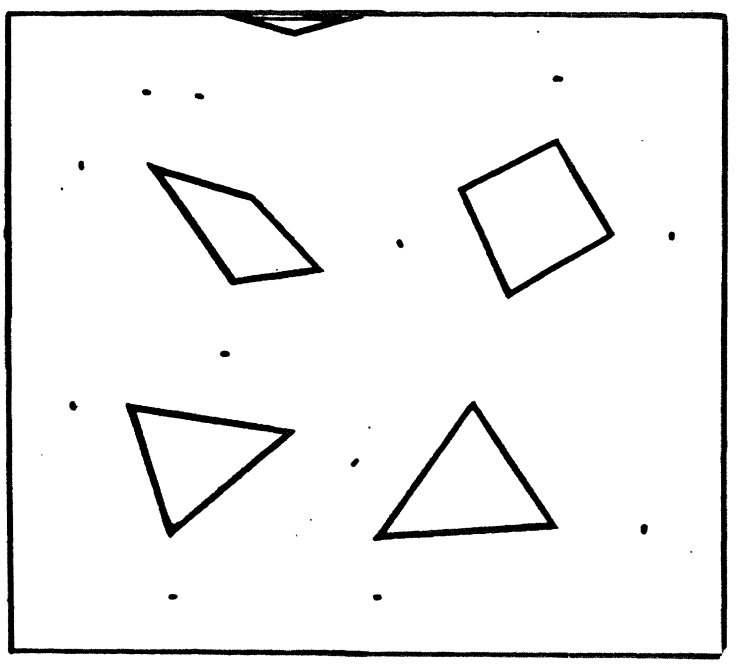
2. Create a very short fake edge e_v going through vertex v and parallel to edge e. Build a cone C with generating edges e, e_v, and axis δ parallel to e and in between e and e_v.

3. Prune cone C and output it as in the two edge cone case. ∎

Complexity: Let n be the number of edges. There are also n vertices. There will be $\binom{n}{2}$ edge-edge or edge-vertex pairs. For each pair, the pruning of the cone



a.



b.

Figures 3.3

a. Finding all cones from pairs of facing edges.

b. Finding all cones from pairs of edge-corner.

with all other edges costs $O(n)$ time. So the above two algorithms have $O(n^3)$ time complexity.

In addition to finding cones as in the above algorithm, we use a heuristic implementation of the Narrowness property to reduce the number of cones, while keeping most of the cones corresponding to Voronoi edges. Our implemented version compares the small end of the cone with the length of the cone. If the length of the cone is less than half the smallest distance between the overlapping segments of the generating edges, then the cone is discarded. This loose heuristic reduces the number of cones by a third in this example.

Figures 3.3.a shows all the regular cones between facing edges after the application of the above heuristic implementation of the Narrowness property. Figures 3.3.b shows all the cones between edges and facing vertices. Each cone is shown by displaying the overlap of the generating edges when projected on the cone axis. The top of figures 3.6 shows all the valid cones. This set of cones coincides closely to the exact Voronoi diagram in figure 3.1.a.

3.3. Convex Regions Capture Clusters of Overlapping Cones

Definitions

1. A *free convex region* is a region R of the free space, limited by a closed convex polygonal boundary and having no obstacles in it. Each edge of the convex boundary must overlap with some obstacle edge. We call each such obstacle edge, a *constraining edge* of the free convex region R.

2. We can alternatively define a free convex region R as the intersection of the free half-planes of facing edges. This intersection of the free half-planes must be bounded, and must contain no obstacles.

Characteristics

1. A free convex region is a region of the free space bounded by a *maximal set* of *facing edges*. Any two edges in this set must face one another. The set S of facing edges delimits a convex boundary B which overlaps with all the facing edges in S. This set S is maximal in the sense that any other obstacle edge not in S will not intersect with the convex boundary B.

Combining this characteristic and definition 2, we observe that a maximal set of facing edges defines a free convex boundary if it includes no obstacles.

2. A free convex region includes multiply overlapping cones. — A cone C is a *member cone* of a convex region R, if 1) the cone region of C overlaps with R, and 2) both of the generating edges of C are also constraining edges of R. In figure 3.4, the convex region R' captures a cluster of three highly overlapping cones.

3. A free convex region provides an *extension* for its member cones. — The cone is good for capturing the tightest local constraints on the moving object by its two generating edges. To ensure that we have a closed and free region, we have arbitrarily bounded the cone with two arcs. This bounding operation at times drastically reduces the cone region. The cone region may be too small to enclose

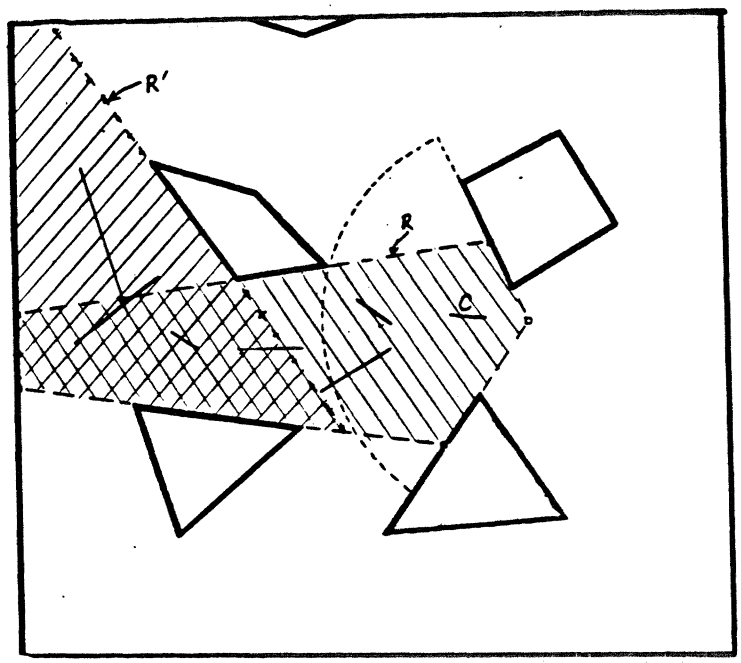


Figure 3.4 A convex region encloses multiply overlapping cones.

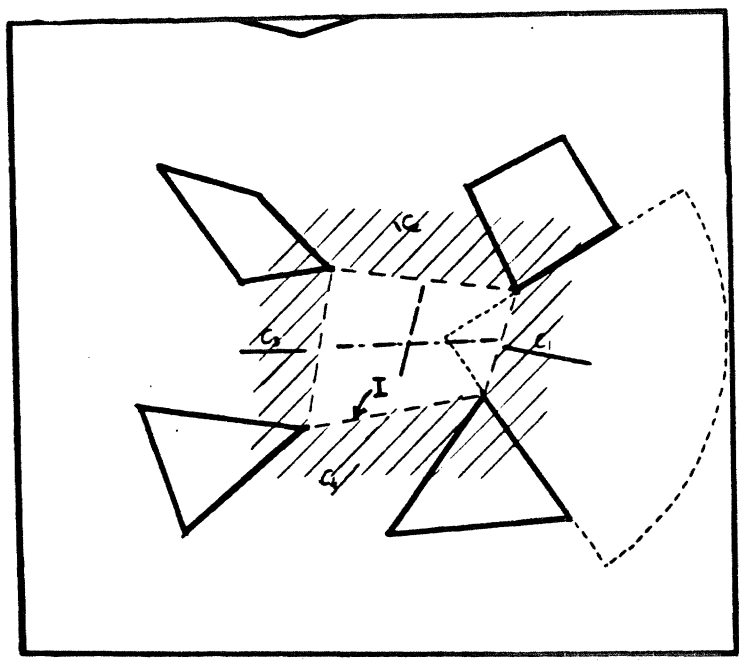


Figure 3.5 A star-shaped intersection captures the branching of cones.

the moving robot M, and is therefore useless. In this case, we can extend the cone region beyond its larger arc by the overlapping free convex region.

4. A free convex region can correspond, on the Voronoi diagram, to an intersection of two or more ee-bisectors. The convex region R' in figure 3.4 corresponds to an intersection of three ee-bisectors.

Convex Regions Localize the Search for Cones

From the second definition of a convex region, we note that in some way, a convex region is a 'maximal' set of facing edges. A cone has only two facing obstacle edges, and can be considered as a 'minimal' set of facing edges. This strong relationship between cone and convex region motivates a search for cones which are local to some convex region only.

In our implementation, all the convex regions are found first. Then, we find a cone from each pair of facing edges belonging to some convex region. This locality property has proved very useful. As example, in figure 3.3.a, we nicely eliminate the possibility of a cone between the middles of the two vertical boundaries.

Finding All Free Convex Regions

For each obstacle edge e, the idea is to do a depth-first search for all convex boundaries starting from edge e. If the convex boundary includes an obstacle-free region, then it defines a free convex region. The depth-first search has exponential growth in the number of edges of the workspace W, because the search tree has order $O(|edges(W)|!)$ leaves.

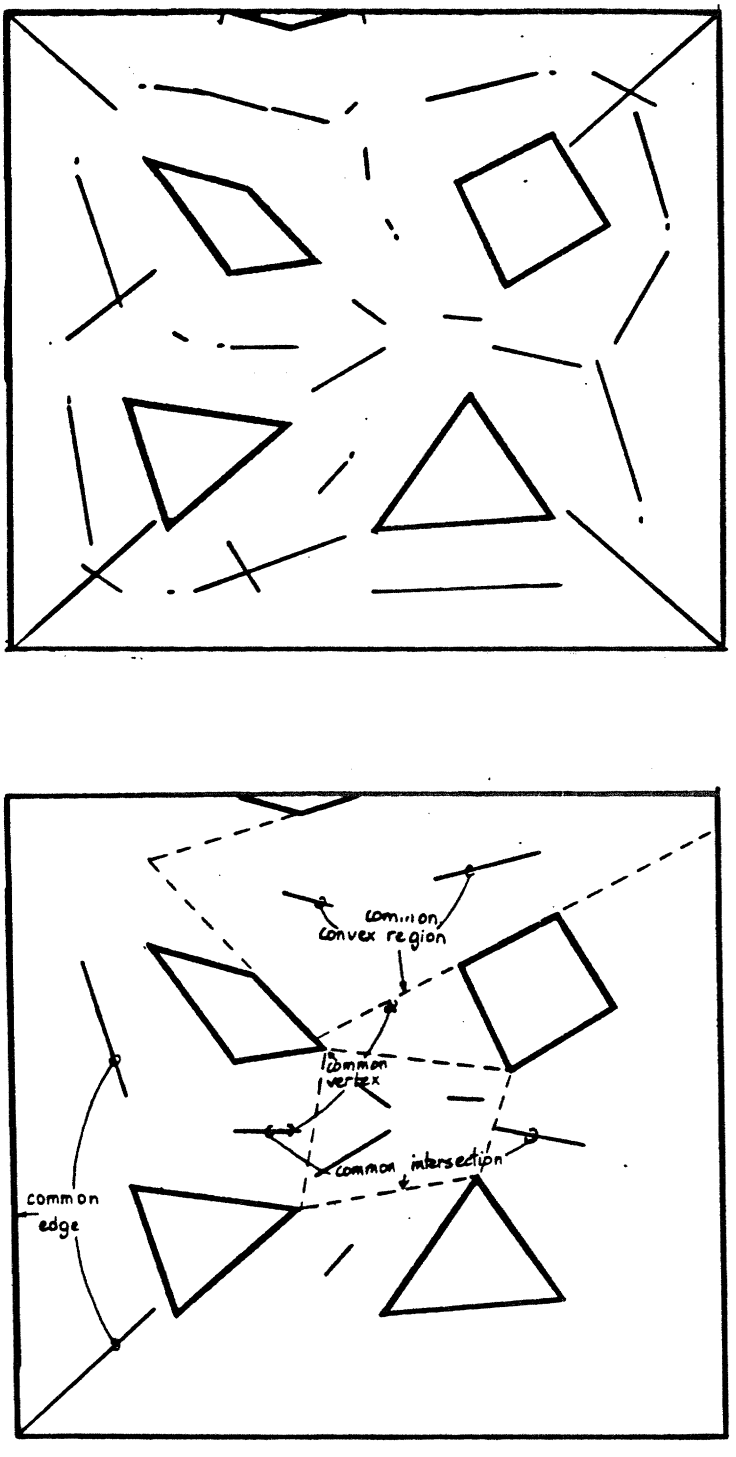
Finding convex boundaries or maximal sets of facing edges is an instance of the Clique problem. So, the finding of convex boundaries is NP-complete. The set of free convex regions is included in the set of legal convex boundaries. Whether the finding of free convex regions is also NP-complete remains an open question.

3.4. A Star-Shaped Intersection Captures Branching of Cones

Definitions

A vertex *faces* an edge if and only if the vertex is in the free half-plane of the edge. Two vertices v_1 and v_2 face one the other, if and only if vertex v_1 faces either of the two obstacle edges connected by v_2, and vice versa.

A *star-shaped intersection* I is an obstacle-free convex region whose vertices all face each other. This convex region I is bounded by artificial edges joining consecutive corners. These corners are called *constraining corners* of the star-shaped intersection I. Since we build our free space description based on cones, we restrict ourselves to finding star-shaped intersections that have a cone branching out of each of its artificial edges. These cones are *member cones* of the star-shaped intersection. The member cones branch-out from the intersection in a star-like fashion, and this motivates the name star-shaped intersection. In figure 3.5, the member cones are C_1, C_2, C_3 and C_4; these cones branch out of a four-vertex star-shaped intersection.



Figures 3.6 The network of linked cones describes the free space between the obstacles.

Characteristics

1. A star-shaped intersection is a convex region of the free space bounded by a *maximal set* of *facing vertices*. Conversely, a maximal set of facing vertices delimits a star-shaped intersection only if the vertices enclose an obstacle-free convex region.

2. A star-shaped intersection includes the ends of each of its member cones. We need star-shaped intersections, because a cone can be very poor at capturing the free space at its two ends. As an example, the small end of cone C_1 is determined not by the obstacles at that end, but by the imaginary extensions of the generating edges. We note also that only the concept of star-shaped intersection captures possible transitions between the cones C_1, and C_3 and between the cones C_2, and C_4.

3. A star-shaped intersection can correspond to a point on the Voronoi diagram, where at least two vv-bisectors intersect. The star-shaped intersection I in figure 3.5 corresponds to two intersection points, each point is the intersection of three vv-bisectors.

The finding of all star-shaped intersections is similar to the search for all free convex regions.

3.5. Capturing the Connectivity of Free Space by Links

A cone captures local constraints of a region by means of its two generating edges and two bounding arcs of circle. In the two previous sections, we saw that a convex region captures a cluster of overlapping cones, and its dual, a star-shaped intersection captures a branching of cones. This section shows how global features are captured by building a network of linked cones. Two cones are linked if they share a common edge or corner, or if they are part of a same convex region, or if they branch out from a same star-shaped intersection. The top figure 3.6 illustrates the complete set of overlapping cones. Note how well cones capture the freeways and bottle-necks of the free space between the obstacles. The bottom figure shows one of each of the four types of links.

3.5.1. Two Cones Sharing a Common Edge

Alignment of the Robot with the Common Edge.

Theorem: Let C_1 and C_2 be two cones which share a common generating edge. The two alignment ranges of robot M in the cones C_1 and C_2 always overlap, unless one of the ranges is null.

Proof: The common orientation is such that the common edge e is parallel to the nearest facing sliding edge of M. See figure 3.7.∎

Details of how M moves from C_1 to C_2 are delayed until section 5.2. For the moment, we say that a common-edge link between two cones suggests that M be 'aligned' with the common edge, and slid along that edge. The common-edge link is very useful when the workspace is cluttered with long obstacles. In our example, it is most useful for sliding along the boundaries of the workspace.

The Freeway Along the Common Edge

Theorem: Let C_1 and C_2 be two cones which share a common generating edge e. Assuming that obstacles do not overlap, one of the followings is always true:

— The two cones C_1 and C_2 share a common convex region.

— There is a tight freeway 'between' the cones C_1 and C_2. This tight freeway is described by a cone.

— There is an obstacle 'between' the cones C_1 and C_2. This obstacle touches the common edge e.

Proof: Remember from the end of section 3.3, that the search for cones is done local to some convex region. So, cone C_1 (resp. C_2) must belong to a convex region R_1 (resp. R_2). Two cases can arise:

1. The two convex regions R_1 and R_2 are the same. In other words, the two cones share a same convex region.

2. The two convex regions are different. These two regions can either overlap or not.

Consider the case when the two regions R_1 and R_2 overlap. Since both R_1 and R_2 are maximal and distinct convex regions, the overlap is not equal to either the region R_1 or region R_2. The union of the two regions R_1 and R_2 is connected and non-convex. The union has at least one concavity at the place where the two convex regions meet. Either a corner v' or an edge e', facing the common edge e, causes the concavity at the overlap. Between this facing corner v' or edge e' and the common edge e, there must be a cone C, since we have here respectively a bottle-neck or a narrow freeway connecting C_1 and C_2. Figure 3.7.a illustrates the case of a bottle-neck cone C describing the concavity between the two convex regions R_1 and R_2.

Next, consider the case when the two regions R_1 and R_2 do not overlap. Since the search for convex regions which have edge e as constraining edge is complete, one of the two following subcases must be true:

a. There is an obstacle in between the two convex regions R_1 and R_2, and this obstacle touches the edge e. See figure 3.7.b

b. There is at least one convex region R' between R_1 and R_2. These convex regions R', R_1 and R_2 may or may not overlap. If they overlap, there will be more than one cone. From these cones, we choose the one which has the smallest cross-section perpendicular to edge e. If the union of the regions R's with R_1 and R_2, is not connected, then there must be at least one obstacle which touches edge e.∎

For later path computations, we order the cones C_e along edge e. A cone C_1 is ordered before cone C_2, if as we walk on edge e, keeping the free half-plane of e on our right, the motions of robot M will be constrained by the generating edges of C_1 before those of C_2.

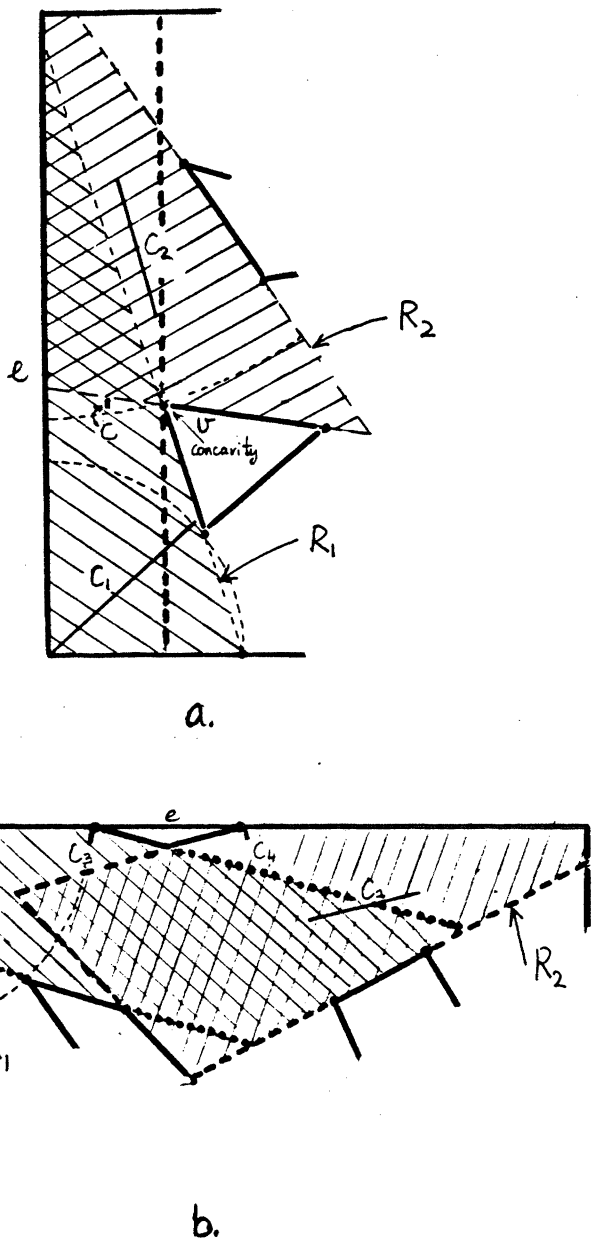


Figure 3.7 A cone describes the free space along a common edge.

3.5.2. Two Cones Sharing a Common Corner

Take as an example figure 3.8. The two cones C_1, C_2 are connected through a common corner V. Because the corner V is convex, the two alignment ranges of M in C_1 and C_2 often do not overlap. Furthermore, the two regions of C_1, C_2 often do not intersect. Even if the two cone regions do intersect, the intersection is generally not big enough to contain the robot. So, not only is there no common orientation of M, but also the free space around a corner is poorly described by the intersection of the cone regions. This is a limitation of using cones to find paths around a corner. The convexity of the corner is responsible for that limitation, and in some sense is a break between the cones.

V-shaped Region Around a Corner

We capture the free space around the corner by a V-shaped region. The V-shaped region connects the two cones, and is bounded on the outside by a convex boundary. From the edges which face the convex corner V, we use a depth-first search to find a convex boundary which connects the outside edges e_1 and e_2 of the two cones. The depth-first search is similar to the one which finds all the convex regions by looking at the facing edges.

3.5.3. Two Cones Included in the Same Convex Region

Common Orientation in a Convex Region.

Theorem: Let C_1 and C_2 be two cones which share a common convex region R. The two alignment ranges of robot M inside the cones C_1 and C_2 always overlap, unless at least one of the two ranges is null.

Proof: For illustrations, please refer to figures 3.9.

1. Since the region inside the boundary of R is convex, the segment joining any two points which are inside the convex region R, is totally inside R. Let TH be a moving segment. We rotate TH around its two ends T and H. Choose any two points P Q inside R. If the ranges of rotation of TH around T-P, and H-Q are non null, they always overlap since the longer of the segments PQ and TH is inside the convex region. See figure a. There is no common orientation when the matching of the points of rotation T and H on the robot and P and Q reverses, or when M does not fit inside R.

2. Now, consider a rectangular robot M. We can shrink the rectangular robot to the segment TH between the tail T and head H of M. We also shrink the convex region R by half the width of M. This case of a rectangular robot thus reduces to the previous case of a segment TH. See figure b.

3. Finally, consider the case of an arbitrary convex robot with its head H and tail T. Let's call θ, the angle between the two sliding edges of the robot M. The robot M can be bounded by the two sliding edges and the two bounding arcs centered at T and H, of radius r_T and r_H, and tangent to the sliding edges. From this bounded shape, we have two rectangular robots M_T, M_H with same tail T and head H, and with respective width r_T and r_H. See figure 3.9.c.

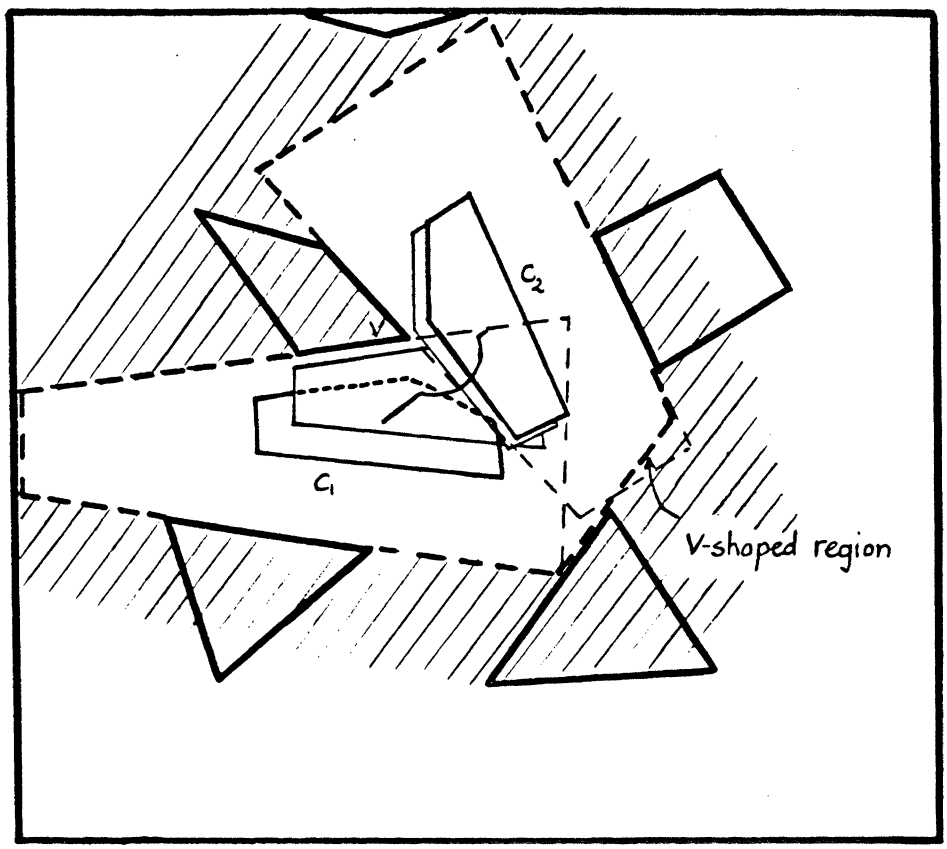
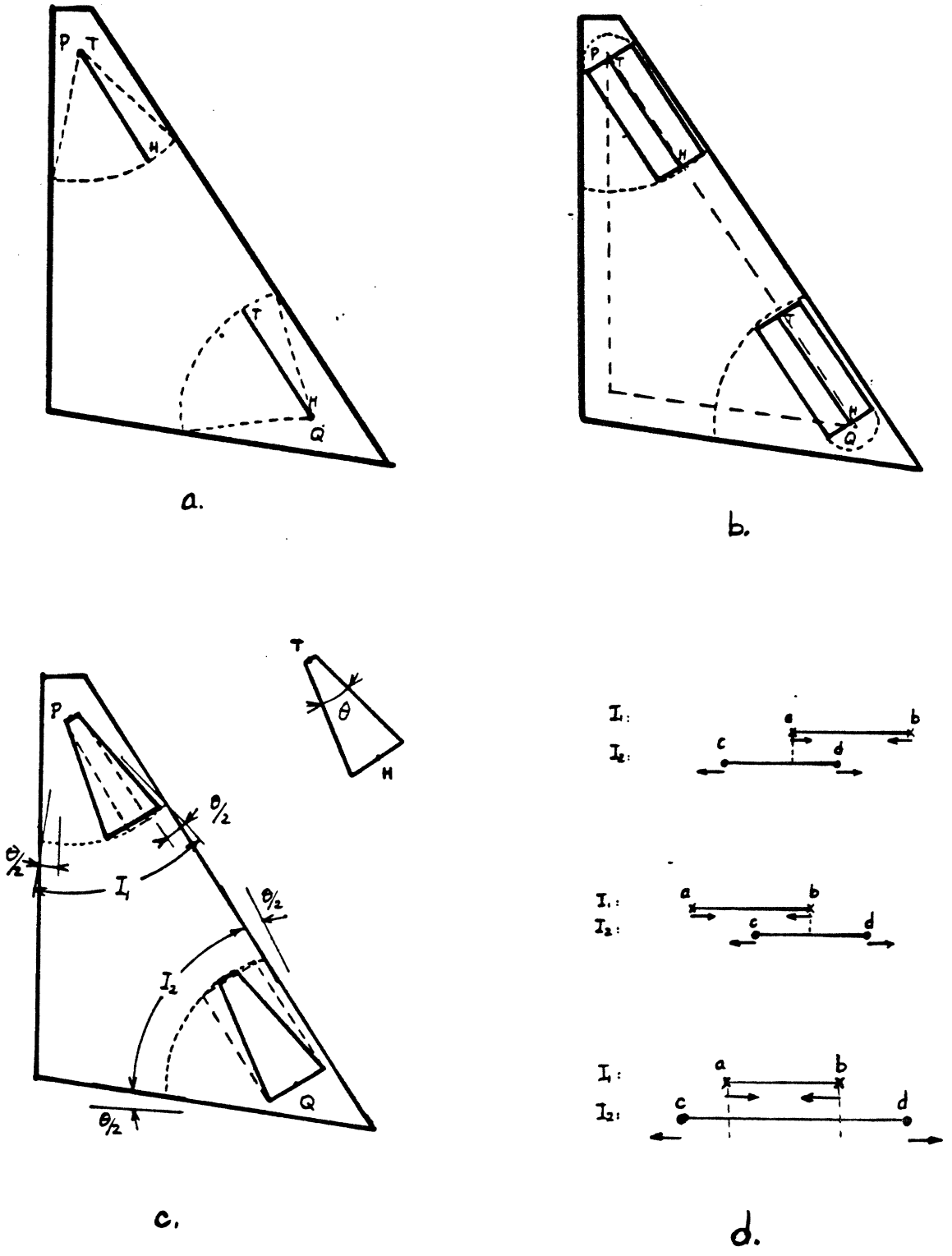


Figure 3.8 A V-shaped intersection describes the free space around a convex corner.



Figures 3.9 The alignment ranges of the robot inside two cones which share a same free convex region always overlap.

– Let I_1 (resp. I_2) be the orientation range of the rectangular robot M_T (resp. M_{II}) about P-T (resp. Q-II). Similar to the above, we can shrink the two rectangular robots to the segment TII, and correspondingly shrink the convex region R. The overlap between the ranges I_1 and I_2 thus becomes obvious. Each of these ranges is a connected interval, and so can be represented by a segment. See figure d.

– We remark that with the real robot M, the range I_2 is larger. More concretely, the two leftmost and rightmost ends of the interval I_2 move 'outward' respectively by $-\theta/2$ and $\theta/2$. This new alignment interval I'_2 includes the old interval I_2, and remains connected.

– For the alignment range I_1, its two leftmost and rightmost ends move 'inward' respectively by $\theta/2$ and $-\theta/2$. This new interval I'_1 remains non null and connected.

– Figures d show the three cases based on the assumption that the ranges I_1 and I_2 originally overlap. In all three cases, we can always find an orientation common to the shifted ranges I'_1 and I'_2. So, the two alignment ranges I'_1 and I'_2 overlap. ∎

3.5.4. Two Cones Branching from the Same Intersection

The link between two adjacent cones branching from a same star-shaped intersection is already captured by the common-vertex link. We need only link two cones that share with the star-shaped intersection four different constraining corners. The free space in the star-shaped intersection which is constrained by these four facing corner can be described by a cone. Cone C_{13} in figure 3.10 is an example of such cone linking the two member cones C_1 and C_3. We'll use cone C_{13} as an intermediary cone to find path from cone C_1 to cone C_3 through the intersection.

3.6. Is the Network of Linked Cones Voronoi-Complete?

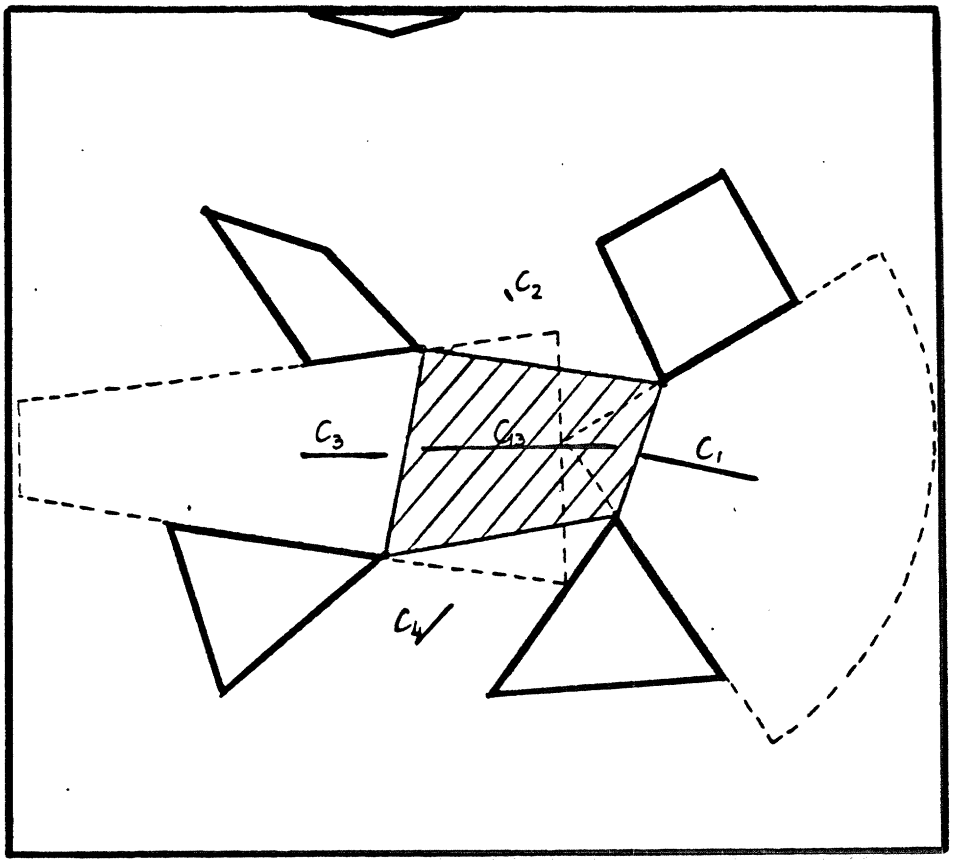
Voronoi-Completeness

The description of a region is *Voronoi-complete* if the description captures all the features of the Voronoi diagram of that same region. By features of the Voronoi diagram, we mean the Voronoi edges and the Voronoi points of intersection. From a point P on a Voronoi edge, we can find all the points on the obstacles which are closest to P. Loosely, we say that a Voronoi-complete description captures all the closest obstacle feature for a point.

Cones Capture some of the Voronoi Edges

The algorithm presented in section 3.2 can find all the ee-bisectors between facing edges. The algorithm in section 3.2 can also find all the 2-sided parabolic arcs between an edge and a facing corner.[9] The requirement that the corner and the edge form a bottle-neck is the necessary condition for the existence of a 2-sided parabolic arc. The overlap on the spine of the cone is another necessary condition

[9]The algorithm can fail to capture an ee-bisector, or a 2-sided parabolic arc only because of the heuristic implementation of the Narrowness property.



Figures 3.10 The free space in a star-shaped intersection is captured by a fake cone.

for the parabolic arc to cut its symmetry axis. Cones also capture a few 1-sided parabolic arcs when these arcs are continued by an ee-bisector.

Only vv-bisectors do not always have corresponding cones. In our opinion, this makes perfect sense since the direction of translation is not constrained by the two facing vertices. The region associated with a vv-bisector is captured partly either by the extension of a neighboring cone or by a star-shaped intersection.

Links Describe some of the Voronoi Intersections

The correspondence between links and Voronoi intersection points is weaker than the correspondence between cones and Voronoi edges. There are two reasons: the incomplete description of Voronoi edges by cones, and the approximation of Voronoi edges by spines of cones. However, there exists a close correspondence between free convex region and intersection of at least two ee-bisectors. Similarly, star-shaped intersection corresponds closely to intersection of at least two vv-bisectors. Common-corner and common-edge links describe some of the other types of intersection points.

The free space between five obstacles is transformed into a network of linked cones in about 40 seconds, on a single user Lisp machine. As the number of obstacles gets larger, the time spent grows rapidly because of the depth-first search which finds all the free convex regions. A better algorithm is needed. This depth-first search takes 15 seconds for our workspace example. But after we have found all the convex regions, the search of cones and star-shaped intersections, and the linking between pairs of cones are done locally. Cones are found only among edges that form some convex region. Star-shaped intersections are found only by searching among cones that are already linked by their vertices.

4. CONFIGURATIONS OF THE ROBOT

4.1. Configuration Space and Volume

We call the triple (x, y, θ) the *configuration* z of the robot . Since the robot M will be rotating about different points, the triple (x, y, θ) is associated with some point of rotation O fixed to M. $(M)_z$ denotes space occupied by the robot M in configuration z. A configuration z is said to be *valid* (resp. *forbidden*) if the robot M in configuration z, $(M)_z$, does not (resp. does) intersect with the obstacles.

The space of configurations of M is called the *configuration space*, or *CSpace*. This space has three dimensions x, y, and θ corresponding respectively to two degrees of freedom for translation, and one degree of freedom for rotation. A *θ-plane* is the plane which is parallel to the xy-plane and cuts the θ-axis at θ. If the workspace of the moving robot M is bounded, the *CSpace* is also bounded. If we cut this bounded *CSpace* by a θ-plane, we will get a bounded planar *θ-cross-section.*

We call *configuration volume* a bounded volume of configurations of M in *CSpace*. A configuration volume is said to be valid if and only if all configurations in the volume are valid. Examples of configuration volumes are:

— a segment $I(\theta_1, \theta_2)$ representing the range of orientation of M about a fixed point O coinciding with some point P in the workspace. See figure 4.1.b.

— a planar convex polygon T representing the translation polygon of M at some fixed orientation θ in a cone C. Figure 4.1.b illustrates.

The range of orientation is denoted by I. We need to specify the point P in the workspace which coincides with reference point O when M rotates. We sometimes write O-P, the point about which M rotates. A range of orientation I is represented by as a set of non-overlapping intervals between 0 and 2π.

The translation polygon is denoted by T. Since a translation polygon limits a convex region, it is represented as a circular list of vertices $\{P_1, \ldots, P_k\}$.

4.2. The Slice Projection Method Favors Translation

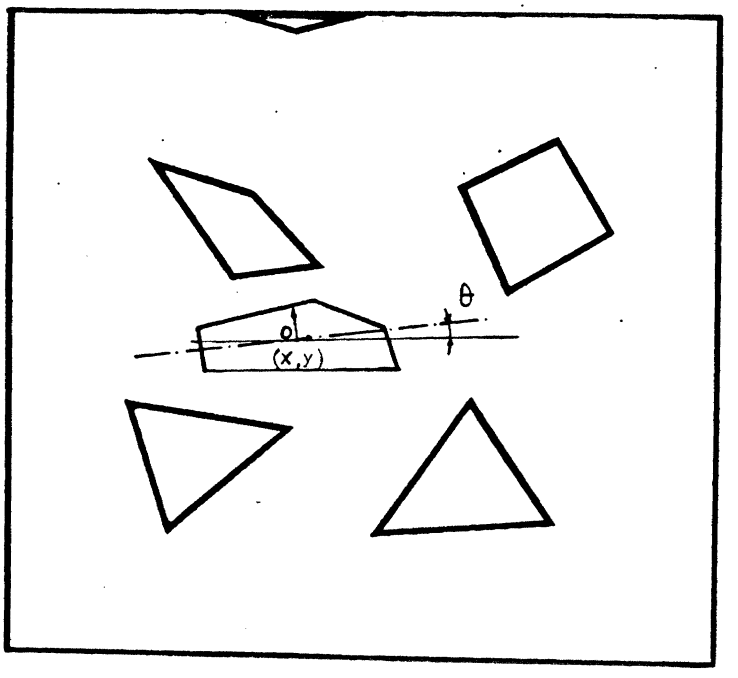
Capturing the Constraints with CI and CO

We recall two constructs from Lozano-Peréz's Slice Projection method [10, 11], and two important theorems:

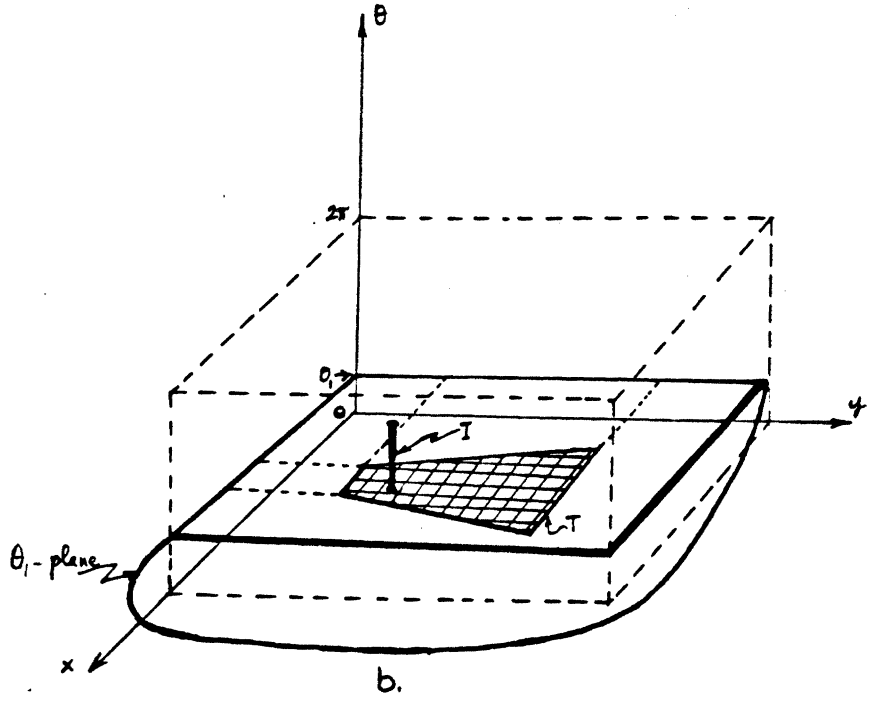
- The *CSpace interior* to B, denoted $CI(B)$, is the set of configurations z of M for which $(M)_z$ is completely inside the convex region B.

$$CI(B) \equiv \{ z \in CSpace \mid (M)_z \subseteq B \}. \quad (1)$$

- The *CSpace obstacle* due to B, denoted $CO(B)$, is the set of configurations z of M for which $(M)_z$ either touches or overlaps with B.



a.



b.

Figures 4.1

a. The point of rotation O on the robot is used as reference point for computing configurations.

b. Configuration volumes capture sets of valid motions. Alignment interval and translation polygon are examples of valid configuration volumes.

$$CO(B) \equiv \{ z \in CSpace \mid (M)_z \bigcap B \neq \emptyset \}. \quad (2)$$

In general, the CI and CO are some three dimensional non convex volumes with curved surfaces. However, if the obstacles and the moving robot are convex, it has been demonstrated that the CI and CO always have convex polygonal θ-cross-sections. In other words, for fixed orientation θ of the robot M, the valid/forbidden ranges of translation of M can be expressed exactly by a set of convex polygons. See figure 4.2.a.

Theorems:

If the boundary of region B and the moving robot M are convex polygons, the $CSpace$ of the robot M interior to B, denoted $CI(B)$, always has bounded convex polygonal θ-cross-sections.

If the obstacle B and the moving robot M are convex polygons, the $CSpace$ obstacle of the robot M due to B, denoted $CO(B)$, always has bounded convex polygonal θ-cross-sections.

Corollary: Let z_1 and z_2 be two configurations of robot M for which $(M)_{z_1}$ and $(M)_{z_2}$ are completely inside the convex region R. If the two configurations z_1 and z_2 have the same orientation θ, then the translation of M between the configurations z_1 and z_2 is collision-free.

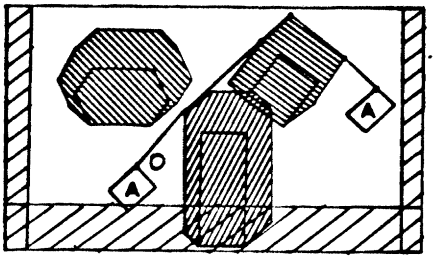
There are efficient algorithms [9] which compute the $CO_\theta(B)$ in $O(n)$ time. $CI_\theta(B)$ can be computed in $O(n \log n)$ time, where n is the number of edges of M and B.

Paths with Translation Only are Easy

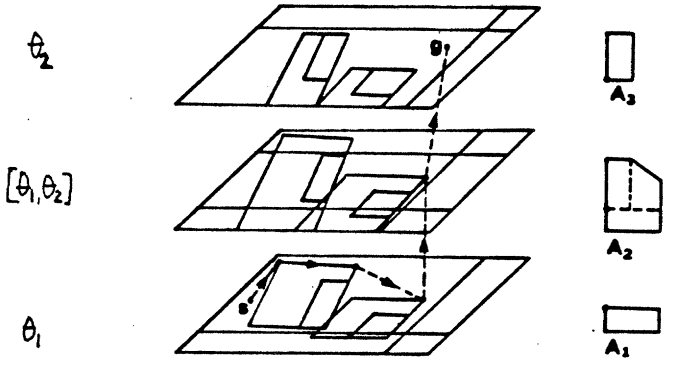
At fixed orientation θ of robot M, the CI of M inside the workspace boundary and the CO of M due to the convex obstacles result respectively in valid and forbidden convex polygons. For the path of the robot M to be collision-free, the point O of M has to be at all times inside the CI-polygon, and outside of all the CO-polygons. In the literature, the CO-polygons are called *grown obstacles*, because we shrink the moving object M to its reference point O and inversely grow the obstacles. This is the exact description of the constraints on the translation of M at fixed orientation θ [9].

We can connect all pairs of vertices of the CO-polygons which 'see' each other. By 'see', we mean that the line segment which joins the two vertices does not cut any CO-polygon, and the two vertices are inside the CI-polygon. We now have a graph connecting all visible vertices of the grown obstacles. Lozano-Peréz calls such graph a *Visibility graph.*

Finding a shortest safe translational path of M is equivalent to searching a path of point O through the above Visibility graph. We have thus reduced the



a.



b.

Figures 4.2

a. Finding translational path for a robot is reduced to finding path for a point outside of the grown obstacles.

b. An example path found by the Slice projection method in C-Space. (reprinted from [11]).

problem of planning collision-free translational paths for M into the equivalent problem of searching for a path of O through a graph [9, 10, 11].

Dealing with rotation turns out to be difficult because of the shape of the $3D$ CI and CO volumes. — A previous approach was to split the range of rotation $[0, 2\pi]$ into a fixed number of slices [10, 11]. Within each slice, we find the bounded robot, and use it to grow the obstacles. In the $CSpace$, a slice $[\theta_1, \theta_2]$ is bounded by two parallel θ_1- and θ_2-planes. There will be paths of O inside each slice, and paths between two consecutive slices.

To compute the CI and CO for a small orientation range, we can equivalently cut a slice from the CI and CO volumes. Then, we project everything in the slice onto a common θ-plane, and take the union of all projections. This process motivates the name of Slice Projection method in $CSpace$. The slicing of the orientation range looks more like a fix-up than an effective method for planning paths with rotation. The Slice Projection method still has a 'translation-flavor'. In figure 4.2.b, the slice is the range between the two orientations θ_1, and θ_2.

A recent method finds paths with rotation by computing exactly the CI and CO volumes for a given resolution. Lozano-Peréz and Brooks avoids the difficulty of representing curved surfaces by subdividing the CI and CO volumes into rectangloid cells [3]. This method is general but slow.

Both methods can be slow, although given enough time and space, they can refine the slicing and the subdivision of the cells to find a path if one exists. More importantly, both methods approach the Find Path problem by subdividing the search space, rather than trying to exploit the features of the free space and of the moving robot to find paths. We aim neither for generality, nor for completeness of paths. We aim to use the description of the free space and of the moving robot to find paths quickly.

Capturing Good Configuration Volumes

With our aim set at using the features from the free space and from the moving robot to find paths, we remark that:

1. We can approximate the CI and CO volumes by straight parallelepipeds which have constant θ-cross-section over a certain range of orientation of M. With such straight parallelepipeds, the range of orientation is rendered independent from the range of translation. This is exactly the slice projection, except there is only one slice, and the slicing interval is determined by the free space and the robot.

2. We can capture a good configuration volume in one slice because the compactness of the exact CI and CO volumes depends heavily on the choice of the reference point.

The above two observations confirm that we need to 'somehow' find some appropriate reference point O on M, such that the CI and CO are well approximated by nice straight parallelepipeds. Looking ahead, figure 4.4.b illustrates a restricted configuration volume of the robot inside a cone. This volume represents the valid

configurations for which M does not collide with the boundaries of the cone C. This restricted volume is a good approximation to the real $3D$ CI-volume. From this point on, we always denote by configuration volume some restricted straight parallelepiped approximating the exact $3D$ volume. We also deal exclusively with valid configuration volumes.

4.3. The Radius Function Method Favors Rotation About a Point

The Radius Function Finds the Shift Distance

The *radius function* $R(\xi)$ [1] is the minimum distance from center of rotation O-P to some wall w with inward normal orientation ξ, such that M rotates about O-P without bumping into w. The radius function can also be defined as:

$$R(\theta, \xi) = \max_{1 \leq j \leq m} d_j \cos[(\theta + \eta_j) - \xi]. \tag{3}$$

where θ is the orientation of M relative to the world frame W, ξ is the angle of the inward normal of wall w relative to W, d_j is the length of the ray from O on M to a vertex v_j of M, and η_j is the angle of that ray with the x-axis of M. Figure 4.3 illustrates.

An interesting point is that the angles ξ and θ just offset the angles η_j of the rays of M. Such offset operation does not depend on the number of vertices of M, and so has constant time complexity. Computing the radius function has $O(|vertices(M)|)$ time complexity.

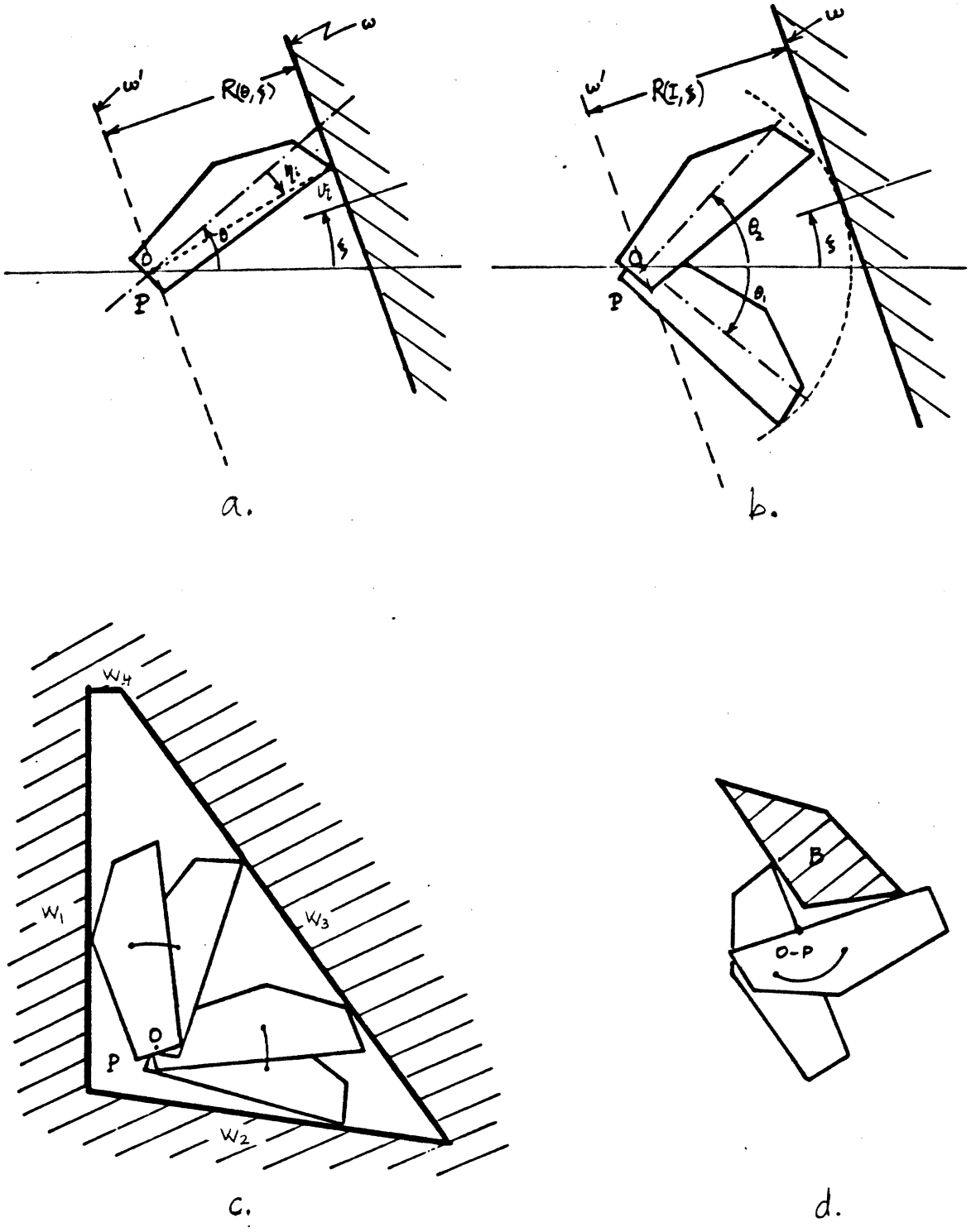
The radius function is the solution to the problem of determining the minimum free half plane which the robot M can occupy. The radius function is related to the support function [17] of a convex polygon, and to the shift distance of edges in the computation of the grown obstacles.

To see the concept of shift distance, let's slide robot M along the wall w, keeping M at fixed orientation θ. The reference point O draw a line parallel to the wall, distant from the wall by $R(\theta, \xi)$. We call the distance $R(\theta, \xi)$, the *shift distance* due to wall w. In some way, the reference point O feels an imaginary wall w' translated by the shift distance $R(\theta, \xi)$.

We now slide M along the wall w, and allow M to rotate within some constant interval I equal to $[\theta_1, \theta_2]$. See figure 4.3.b. We get the shift distance for the bounded moving object as:

$$R(I, \xi) = \max_{\substack{1 \leq j \leq m \\ \theta_1 \leq \theta \leq \theta_2}} d_j \cos[(\theta + \eta_j) - \xi]. \tag{4}$$

The *cosine* function is monotonically decreasing over $[0, \pi]$, and monotonically increasing over $[\pi, 2\pi]$. So the maximum of the *cosine* function in an interval can



Figures 4.3

a, b. The Radius function finds the shift distance. The inverse radius function finds the range of rotation about a point.

c. The range of rotation of the robot inside a convex region.

d. The range of rotation of the robot outside a convex obstacle.

be computed in constant time. The shift distance of the bounded moving robot $R(I, \xi)$ is therefore computable in $O(|vertices(M)|)$ time.

The Inverse Radius Function Finds the Orientation Range

In figure 4.3.a, the robot M rotates about point O-P. The robot is constrained on one side by an infinite wall w with inward normal orientation ξ. The reverse problem of finding the valid range of orientation θ of M can be formulated with the inverse radius function [1] as follows:

$$R^{-1}(r) \equiv \{\theta \mid R(\theta, \xi) \leq r\}. \tag{5}$$

where r is the distance from point O-P to wall w.

Brooks [1] noted that $R^{-1}(r)$ can be easily found by computing for each vertex v_j of M, the interval $[-\eta_j - |cos^{-1}(r/d_j)|, -\eta_j + |cos^{-1}(r/d_j)|]$, and removing it from the interval $[0, 2\pi]$. Then we offset the resulting interval by ξ. The resulting interval of orientation is denoted $I(w)$.

$$I(w) = R^{-1}(distance(P, w)). \tag{6}$$

There are m vertices for M and in the worst case, we may have to delete the newly found interval with $(m-1)$ other disjoint intervals. So the inverse radius function $R^{-1}(r)$ has complexity $O(m^2)$. $I(w)$ has at most m disjoint intervals.

This algorithm only holds for a convex robot and infinite wall, in which case the robot M touches the wall w if and only if at least one of its vertices touches the wall w. If the wall w is finite, or if the robot M has a concavity, M can collide the wall w with one of its edges, and the range $I(w)$ might be larger.

Robot Constrained by an Enclosing Convex Region

Theorem: The range of orientation of robot M rotating about a fixed point O-P and remaining inside a convex region B bounded by n walls $w_1, \ldots, w_n$ is the intersection of n intervals $I(w_1), \ldots, I(w_n)$, respectively being the range of orientation of M about O-P not bumping into walls $w_1, \ldots, w_n$.

$$I_{in}(B) = \bigcap_{i=1}^{n} I(w_i). \tag{7}$$

Proof: The space inside a convex region B bounded by m walls $w_1, \ldots, w_n$ is the intersection of their respective n free half-planes. So, M can rotate without colliding inside B if and only if M can rotate in all the n free half-planes.

The robot M touches the boundary of the convex region B if and only if at least a vertex of M touches some wall w_i of B, because both the robot M and the region B are convex. In other words, the collision of the robot M with the

boundary of B is equivalent to the collision of some vertex of M with some infinite wall of B. So, the range of rotation of the robot can be calculated by looking at the vertices of the robot, and by using the inverse radius function R^{-1}. The ranges of orientation are $I(w_i)$, where w_i is an infinite wall of the convex region B. See figure 4.3.c.∎

The computation of the n intervals $I(w_i)$ costs $O(n.m^2)$ time. The intersection of the n intervals, each having at most m disjoint subintervals, costs $O(n.m)$ time. So the range $I_{in}(B)$ costs $O(n.m^2)$ time.

Robot Constrained by a Nearby Convex Obstacle

Theorem: The range of orientation of robot M rotating about a fixed point O-P and being outside a convex obstacle B formed by n edges $b_1, \ldots, b_n$ is the union of:

1. n intervals $I(b_1), \ldots, I(b_n)$, respectively being the range of orientation of M, rotating about O, whose vertices do not bump into the infinite lines containing the edges of B, $b_1, \ldots, b_n$.

2. m negated intervals $I(e_1), \ldots, I(e_m)$, respectively being the negated range of orientation of B, rotating about P whose vertices do not bump into the infinite lines containing the edges of M, $e_1, \ldots, e_m$.

$$I_{out}(B) = [\bigcup_{i=1}^{n} I(b_i)] \quad \bigcup \quad [\bigcup_{j=1}^{m} -(I(e_j))]. \tag{8}$$

Proof: We view the space outside a convex obstacle B bounded by n edges $b_1, \ldots, b_n$ as the union of n corresponding free half planes. So, M can rotate outside of B if and only if M can rotate in one of the n free half-planes. This proves the three union signs.

However, because the edges b_i are finite, M can have one of its vertices touching an edge of B, as well as having one of its edges touching a vertex of B. See figure 4.3.d. The first type of collision corresponds to vertices of M touching infinite edges of B. This type of collision is responsible for the first union in equation (8). To see the meaning of the second union, we fix the robot M and rotate the obstacle B about P. The rotation of B stops whenever a vertex of B touches an infinite edge of M. We also have to negate the second set of intervals because the relative rotation between M and B is reversed. ∎

The computation of the n intervals $I(w_i)$ costs $O(n.m^2)$ time. The union of the n intervals, each having at most nm disjoint subintervals, costs $O(n^2.m)$ time. So the range $I_{out}(B)$ costs $O(n^2.m^2)$ time.

Configuration Range of the Robot Rotating About a Point

The *Configuration Range* of the robot M rotating about a point P, and constrained by B is denoted by $CP(B)$. $CP(B)$ is the set of valid configurations

of robot M, for which M does not collide with B when it rotates about O-P. B can be an infinite wall, a convex boundary, or a convex obstacle as above. $CP(B)$ is specified by a range of orientation $I(B)$, and a single vertex polygon $\{P\}$. $I(B)$ is computed as in equations (6), (7), (8).

In the *CSpace*, the configuration range $CP(B)$ is pictured by a segment parallel to the θ-axis going through point P.

4.4. Configurations of the Robot Inside a Cone

This section combines the results from the previous sections, with the heuristics from chapter 2 and 3, to build a new type of configuration volume called a *Configuration Cone*. A Configuration Cone is a restricted configuration volume which captures a large range of translation and rotation of the robot M inside a cone. Refering to figure 4.4.a, the valid motions of M inside cone C are characterized by the alignment range $[\theta_1, \theta_2]$, and the translation polygon $\{P_1, P_2, P_3, P_4, \}$. The point of rotation on the robot is T, the center of the tail bounding arcs. Inside cone C, the robot can rotate between θ_1 and θ_2, while its point T can translate to any point inside the polygon $\{P_1, P_2, P_3, P_4, \}$.

Alignment and Rotation Point Inside a Cone

The Sliding heuristic of chapter 2 pinpoints the two orientations θ_1, θ_2 of robot M, for which a sliding edge of M is parallel to the nearest corresponding generating edge e_1 or e_2 of cone C. The Narrowness property of chapter 3 observes that a semi-Voronoi cone C is narrower between the two generating edges e_1, e_2, than between the small and large bounding arcs. So, for robot M to be inside and translate through the narrow cone region, M should align itself with the spine of C. The Sliding heuristic and Narrow property support the need for the alignment range $[\theta_1, \theta_2]$ as the range of orientation of M inside cone C.

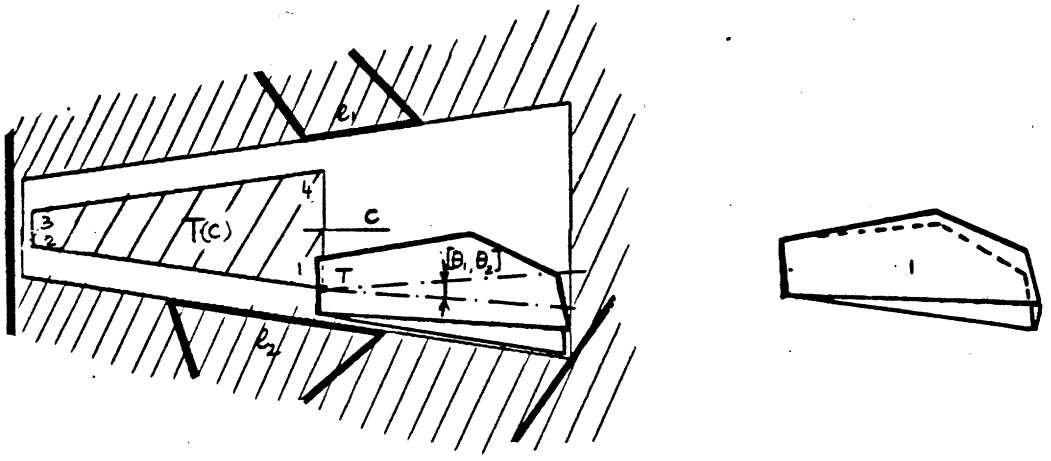
$$I(C) = [\theta_1, \theta_2]. \tag{9}$$

If the two sliding edges of M are already known, the computation of the alignment range $I(C)$ does not depend on the number of vertices of M. So, the computation of $I(C)$ is done in constant time.

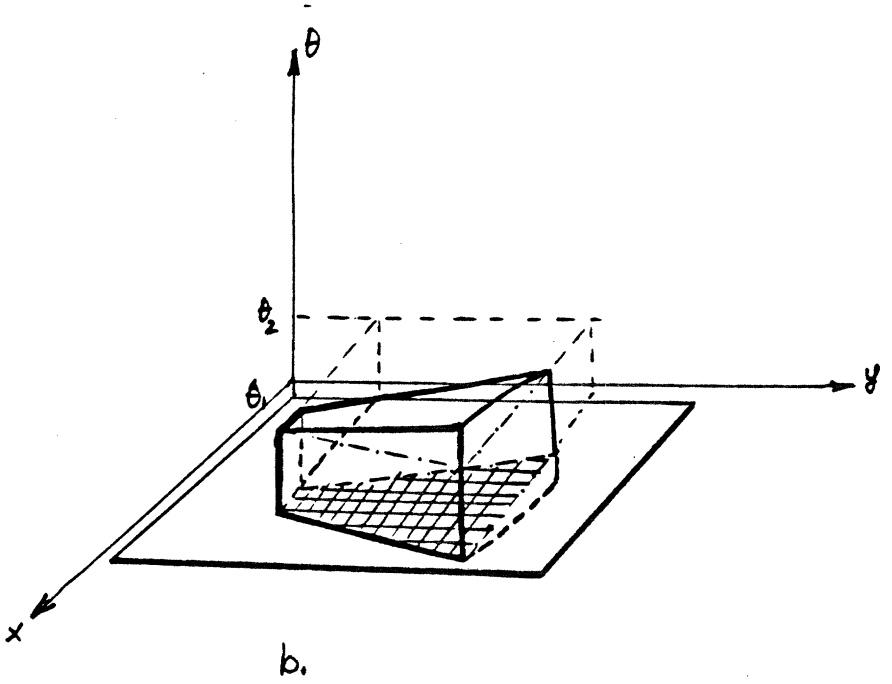
Translation Polygon Inside a Cone

The Head-Tail heuristic of chapter 2 observes that the small end of the cone can be easily sneaked through if robot M rotates only about its head or tail ends. We choose either H or T depending on which is nearer to the small end of the cone.

Figure 4.4.a shows a cone C bounded by four segments. The bounding arc at the large end of the cone is approximated by one segment. As the moving robot M rotates from θ_1 to θ_2, it sweeps some volume which can be bounded by taking the convex hull. See figure 4.4.b. We use this convex bounded robot to compute the



a.



b.

Figures 4.4

a. The alignment range and the translation polygon of the robot inside a cone.

b. The configuration cone of the robot is a good approximation to the real CI-volume.

CI inside the restricted cone region R. Such CI operation is equivalent to shifting the edges of R by the shift distance calculated over the alignment range $I(C)$. The result is a convex polygon of translation constraining where the reference point T of M can be.

$$T(C) \equiv CI(C). \tag{10}$$

where the moving robot is the bounded sweep of M in the alignment range $I(C)$.

Since the shift distance can be computed in $O(|vertices(M)|)$, and the cone region R is chosen to have from three to five vertices, the translation polygon $T(C)$ is computable in $O(|vertices(M)|)$ time.

Theorem: The robot M is collision-free inside cone C, if its orientation θ is inside the alignment range $I(C)$, and if its reference point is inside the translation polygon $T(C)$. Both $I(C)$ and $T(C)$ are constructed as above.

Configuration Cone

Definition: The *Configuration Cone* of robot M, rotating and translating inside a cone C, denoted $CC(C)$, is specified by an alignment range $I(C)$, and a convex translation polygon $T(C)$. The reference point on the robot M is either H, or T, the center of respectively the head or tail bounding arcs.

$$CC(C) \equiv \{I(C), \quad T(C)\}. \tag{11}$$

In the *CSpace*, $CC(C)$ is the sweep of the translation polygon $T(C)$ along $I(C)$. Configuration cone captures a good slice of the CI-volume $CI(C)$.

The alignment range $I(C)$ is computable in constant time. The translation polygon is computable in $O(|vertices(M)|)$ time. So, the Configuration Cone $CC(C)$ is computable in $O(|vertices(M)|)$ time.

5. EXPERTS COMPUTE LOCAL PATHS

For each type of link, there is an expert which plans paths between linked cones. Roughly, each path expert works as follows:

1. From the starting and ending cones C_1 and C_2, the expert computes the two configuration cones CC_1 and CC_2. From the free space description, we retrieve or compute the transition region describing the link between C_1 and C_2. From this transition free space, the expert computes some valid configuration volume CC_{12}.

2. The path expert finds local paths of robot M along the link between C_1 and C_2 by intersecting successive pairs of configuration volumes: CC_1 with CC_{12}, and CC_{12} with CC_2.

3. The intersection of two valid configuration volumes gives an infinite number of valid paths. The path expert chooses one path by using a local search which looks first for the smallest rotation, then for the smallest translation.

As an example, let's consider the path of the robot M through a convex region R, illustrated in figure 5.1.a. The configuration cones CC_1 and CC_2 describe all the rotations and translations of M inside the cones C_1 and C_2. The transition region here is the convex region R. From the convex region R, we compute the CI of M inside R, and at some fixed orientation common to both the alignment ranges of CC_1 and CC_2. The intersection of CC_1 and CI gives the intermediate configuration of M, and specifies the rotation about O_1-P_1 in cone C_1. The intersection of CI with CC_2 gives the final configuration of M in cone C_2, and specifies the translation from C_1 to C_2. Note that we have opted for the smallest rotation in C_1, then the shortest translation from C_1 to C_2.

Local paths of the robot are computed by the four experts. The whole path between two arbitrary configurations of the robot is found by an A^*-search [13]. The A^*-search probes the network of linked cones, and uses the four experts to plan paths along the links between the cones. From the search point of view, cones are 'nodes', and links are 'edges' between the nodes.

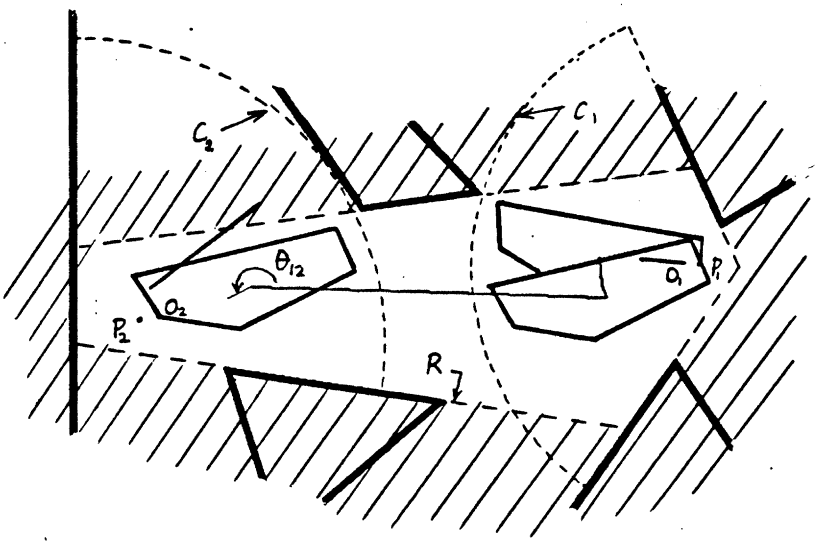
5.1. Traversing a Free Convex Region

Using the Free Convex Region as a Transition Medium

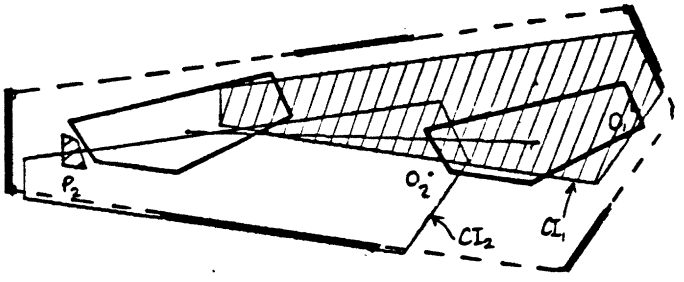
Figure 5.1.a illustrates a path through a free convex region which does not totally contain the two linked cones. From cone C_1, the robot gets inside of the free convex region R, then translates inside R to reach cone C_2.

The path between the cones C_1 and C_2 through the convex region R is composed of:

1. A rotation about the current point O_1-P_1 to the nearest orientation θ_{12} common to the two alignment ranges $I(C_1)$, $I(C_2)$.



a.



b.

Figures 5.1 Traversing a convex region.

a. A path between two cones partially wrapped by a convex region.

b. Find the translation by computing the CI of the robot inside the convex region.

2. A translation inside cone C_1 to some nearest point P_{12}, such that for configuration z_{12}, $(M)_{z_{12}}$ is not only inside the cone C_1, but also inside the convex region R. In our example, P_{12} coincides with P_1.

3. A translation inside the convex region R to some nearest point P_2, such that for configuration z_2, $(M)_{z_2}$ is not only inside the convex region R, but also inside the cone C_2.

Proof: We prove that the above path is collision-free.

1. Since the alignment range $I(C_1)$ is a connected interval, and since before and after the rotation of M, the orientation of M is both inside $I(C_1)$, the rotation is collision-free.

2. The two points P_1 and P_{12} are both inside the translation polygon $T(C_1)$. So the translation inside cone C_1 is collision-free.

3. The robot M is completely inside the convex region R, and has the same orientation in both configurations z_{12} and z_2. From the corollary in section 4.2, the translation inside the convex region R is collision-free. ∎

Computation: We show how the points P_{12} and P_2 are determined.

Call CI_1, the shrunk convex region obtained by taking the CI with M at fixed orientation θ and inside convex region R. The overlap of the translation polygon $T(C_1)$ and CI_1 is a convex polygon, representing the set of points P_{12}s for which $(M)_{z_{12}}$ is both inside the cone C_1 and inside the convex region R. From this overlap, we can choose the point P_{12} which is nearest to point P_1.

We find point P_2 similarly. Also, we don't need to shrink the convex region again, because CI_2 is the region CI_1 translated by the vector connecting the points of rotation O_1, O_2 when M is at orientation θ_{12}.

Complexity of Finding a Path Through a Convex Region

Assume that the robot and the workspace has $O(n)$ edges. — The CI operation can be done in $O(n \log n)$ time. The shrunk convex region CI_θ has $O(n)$ edges.

— The intersection of two convex n-gons can be computed in $O(n)$ time. Since the translation polygon $T(C)$ has at most five edges, the overlap between $T(C)$ and the CI_θ is computable in $O(n)$ time.

So the path through a convex region is computable in $O(n \log n)$ time.

5.2. Sliding Along an Edge

Sliding Along an Edge and Going Through a Bottle-Neck

Figure 5.2.a shows a path along an edge and through a bottle-neck between two cones. From section 3.6, we know that with proper matching of the points of rotation on M, the two alignment ranges in the two cones always overlap. The

common orientation is such that the common edge is parallel to the facing sliding edge of the robot. The rotation inside the alignment range is collision-free.

Translation is harder because the cone regions often do not overlap, Even if the cone regions do intersect, their union is generally non convex. The planning of the path along an edge proceeds in two steps:

1. Find the description of the free space along the edge, and between the two cones. — We look for the tightest freeway along the edge, or an obstacle which touches the edge. In the figure, the tightest freeway is the bottle-neck B, between the edge e and vertex v. The bottle-neck B is used as transition medium between the cones. If there is neither a tightest freeway nor a vertex touching the common edge, then from the theorem in 3.5.1, the two cones must share a convex region. This convex region is used as the transition medium.

2. Compute a path for the robot. — As in finding a path through a convex region, from cone C_1 the robot first gets inside of the bottle-neck B, then it translates inside the bottle-neck until it reaches the next cone C_2. The path is composed of the familiar one of a rotation, and two translations. In case the two cones share a convex region, the path through that convex region is computed as in the previous section.

We'll concentrate our discussion to the first step. The second step is similar to the planning of a path through a convex region. See section 5.1.

The Free Space of Transition Along the Edge

The theorem in section 3.5.1 tells us that the free space along an edge e is completely described by the convex regions and the cones having e as generating edge. In between the two cones C_1 and C_2, there is either a freeway, or an obstacle, or else the two cones must share a convex region.

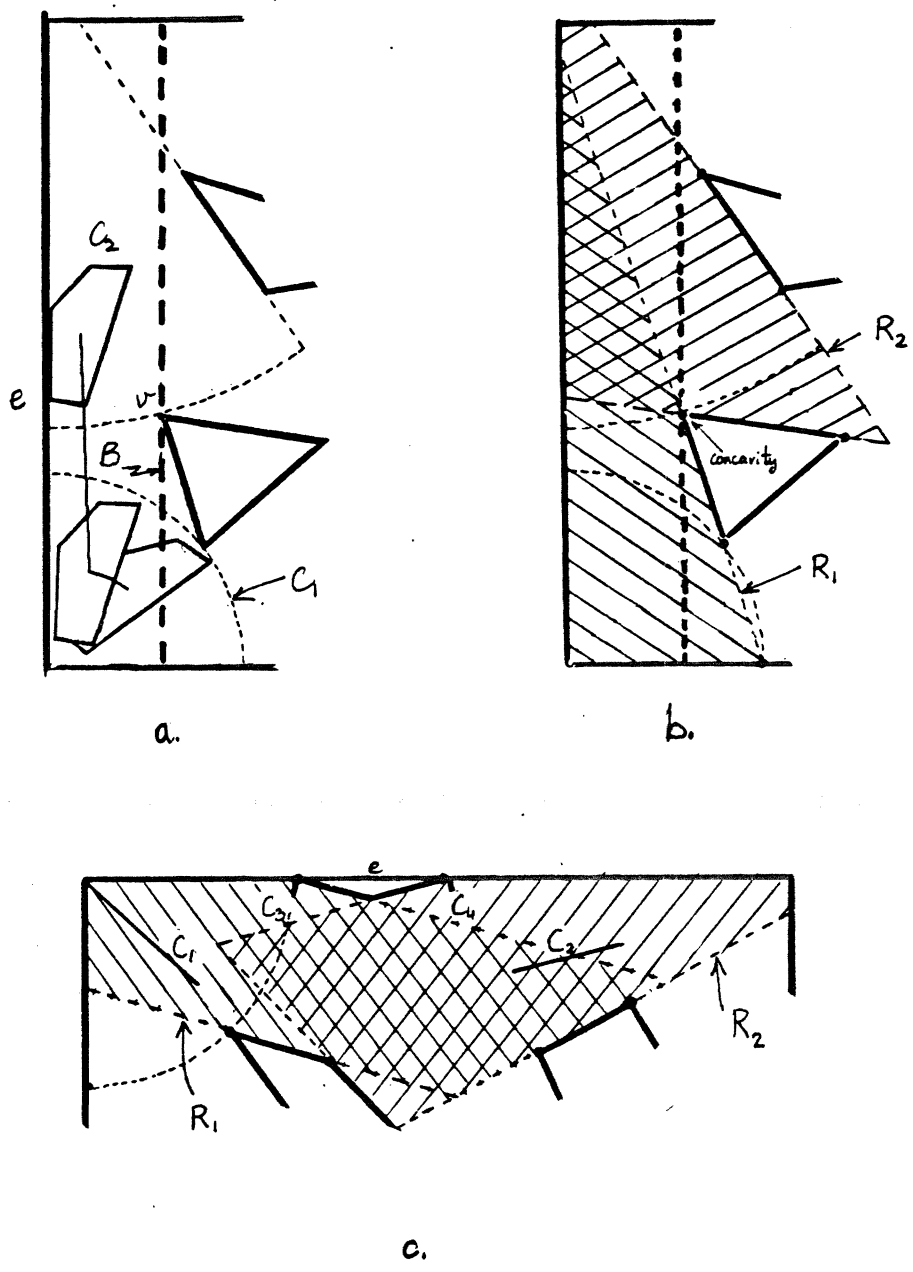
Computation: We recover the description of the free space between the two cones C_1 and C_2 as follows.

1. If the two cones C_1 and C_2 share a convex region, then return the convex region; else, continue with steps 1. and 2.

2. We find all cones C between the two cones C_1 and C_2. From these cones, we find all the generating edges f which face the common edge e. Next, from these edges, we find all of their vertices v; then for each vertex, we compute its distance to the common edge e.

From the two vertices of the edge f_1 of C_1, and their corresponding distance, we choose an end vertex v_1 which is nearer to e. Similarly, we find the end vertex v_2, which is the nearer vertex from the edge f_2 of cone C_2.

3. From the vertices v in between and including v_1 and v_2, a case analysis will dispatch to either:



Figures 5.2 Sliding along an edge.

a. The robot slides along an edge and passes through a bottle-neck.

b. A freeway along the common edge and describing a bottle-neck in between the two cones.

c. An obstacle touching the common edge of the two cones.

a. there is a corner v which is nearest to the common edge e, and is between[10] v_1 and v_2. The free space of transition is the cone which describes the bottle-neck between vertex v and edge e. See figure 5.2.b.

b. there is an edge f which is parallel to the common edge e, and nearest to e. Both the vertices of this edge must be between the two end-vertices v_1 and v_2. The free space between C_1 and C_2 is the cone describing the freeway between f and e.

c. there is an obstacle which touches the common edge if there is a vertex in between v_1 and v_2, whose distance to the common edge is zero. There is no transition region, and no path. See figure 5.2.c.

Proof: We prove that the steps 2 and 3 effectively find a cone or an obstacle in between two cones which share a common edge and have no common convex region.

1. Remember that the cones which share a common edge e are ordered along the edge e. From this ordered list, we can find all the cones C which are between the two cones C_1 and C_2. Among these cones Cs, we'll prove that there must be at least one cone which constraints most the translation along the common edge.

The theorem in section 3.5.1 says that there must be either a freeway or an obstacle which touches the common edge.

a. If such a freeway exists, it must be described by some cone B since cones capture all the freeways and bottle-necks in the free space. The cone B results from the interaction of the common edge with either a facing corner or a facing edge. The corner or the facing edge creates a concavity in between the two cones. So, assuming that the ordering of the cones reflects the ordering of the constraints along the common edge, the cone B which describes the break between the two convex regions R_1 and R_2, must necessarily be in between C_1 and C_2.

b. If such an obstacle exists, then there must be at least two cones between the two edges of the obstacle and the common edge e. We have two convex regions, which overlap the common edge on the two sides of the obstacle. Note that there is always a cone between two consecutive constraining edges of a convex regions if these two edges intersect. In figure 5.2.c, the two cones between the touching obstacle and the common edge are denoted by C_3 and C_4. Once more, assuming that the ordering of the cones along the common edge e reflects the ordering of the constraints along this edge, the two cones between the obstacle and the common edge must be in between the cones C_1 and C_2.

2. The cone which captures the tightest constraint on the translation of M between C_1 and C_2 can be singled out by finding the edge or vertex nearest to the common edge e. Since we assume the obstacles do not overlap, the detection of an edge or a vertex nearest to a common edge can be done by looking at the vertices between v_1 and v_2 only. ∎

[10] 'Between' means the projection of corner v on edge e is in the segment formed by the two projections of v_1 and v_2 on e.

Complexity of Finding a Path Along an Edge

— The above scheme finds the free space along the edge e in time linear to the number of cones Cs which share e as common edge. There are at most as many cones which share a same edge as there are edges in the workspace. So, the above scheme is computable in $O(n)$ time, n is the number of edges in the workspace.

— Path computation through a common free convex region, or a transition cone costs $O(n \log n)$ time.

So, the path along an edge is computable in $O(n \log n)$ time.

5.3. Circumventing A Corner

Approach, Rotate and Slide, then Depart

The planning of a path around a convex corner proceeds in two steps:

1. Find the description of the free space around the corner, and between the two cones. From section 3.5.2, the free space around a convex corner is described by a V-shaped region. This region connects the two cones, and has a concavity only at the corner.

2. Compute a path for the robot. Because of the concavity of the V-shaped region at the corner, the path around a convex corner is broken into three sub-paths:

a) Approach the corner. — Like previous paths, from the current cone, the robot gets inside of the V-shaped region. Then, the robot translates inside the V-shaped region, and approaches the corner.

b) Rotate and slide about the corner. — The robot rotates about the corner and possibly slides on this corner. This rotation bridges the gap between the two alignment ranges of the robot inside the two cones. After this step, the robot is in some orientation which is inside the alignment range of the next cone.

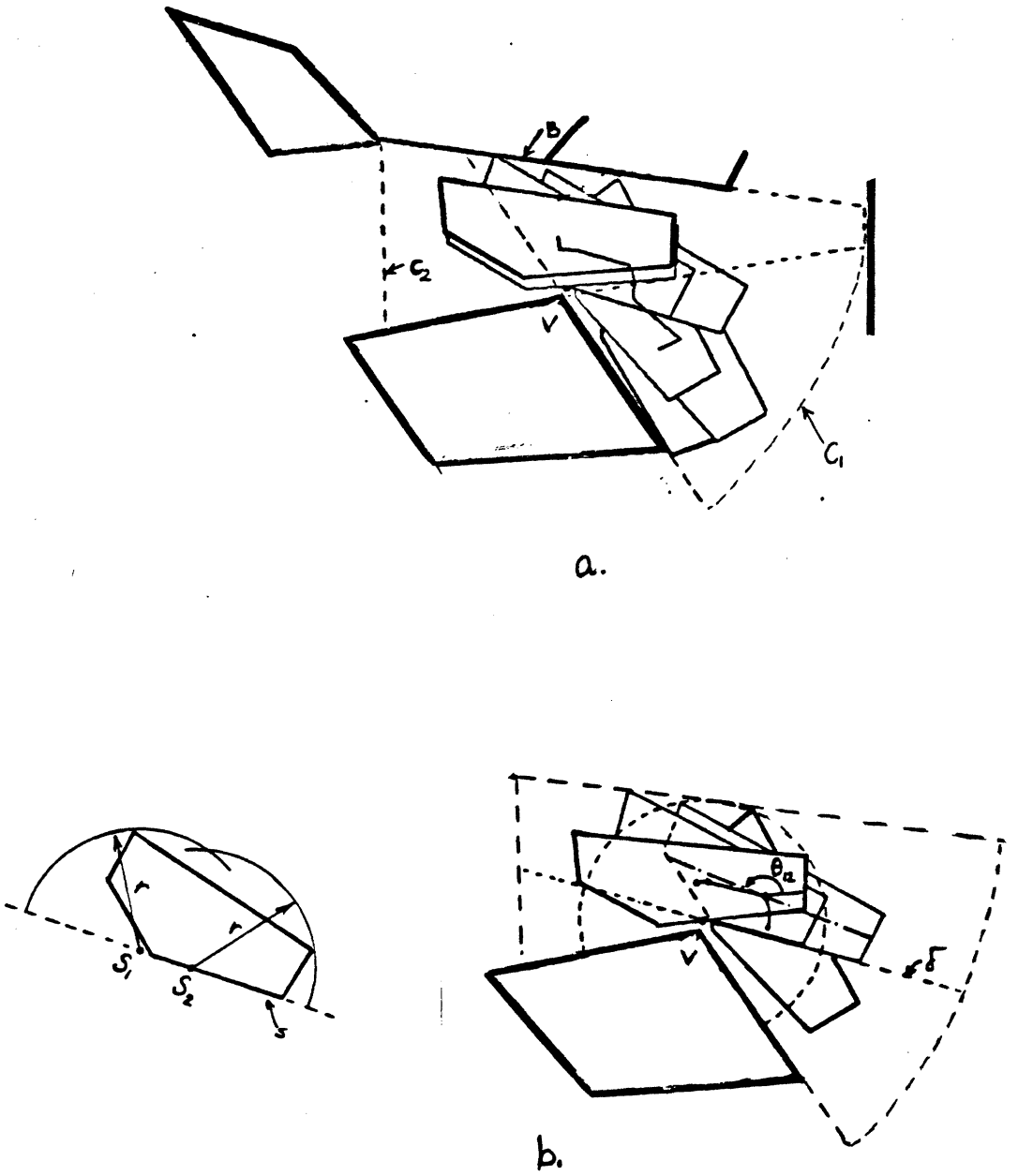
c) Depart from the corner. — Like the reverse of the approach path, the robot departs from the corner, then moves inside the V-shaped region until it is inside of the next cone.

The complete path of the robot around the corner V is shown in figure 5.3.a. The subpaths are shown in figures 5.3.b, and 5.4

Because cone poorly describes the free space around a corner, finding the V-shaped region is not supported by the network of linked cones. We have used a depth-first search to implement the search for a V-shaped region around a corner v. From the edges and vertices which face corner v, the depth-first search finds a convex boundary which connects the ends of the two cones.

Rotate and Slide About the Corner

Let s be the sliding edge that faces the convex corner. From the radius r of the disc segment D, we can find the two points of rotation S_1 and S_2 on the sliding edge s such that:



Figures 5.3 Navigating around a convex corner.

a. A path around a convex corner. From the current cone, the robot approaches the corner, rotates and slides about the corner, then it departs from the corner and enters the next cone.

b. The robot rotates and slides about the corner. The points of rotation are specified by the free space around the vertex.

— The head (or tail) of the robot is collision-free inside the disc segment D, when the robot is rotating about point S_1-V.

— The tail (or head) of the robot is collision-free inside the disc segment D, when the robot is rotating about point S_2-V.

In figure 5.3.b, the point S_1 (resp. S_2) corresponds to the head (resp. tail) of the robot rotating freely in the disc segment D. The two points of rotation correspond successively to one of the ends of the robot moving inside the disc segment. In the computation of the points of rotation S_1 and S_2, there are three simple cases:

a. The radius r of the disc segment is smaller than the cross-section associated with s. In this case, the disc segment is too small for the robot to sneak through it. There is no point S_1 and S_2, and no path around the corner.

b. The radius r of the disc segment is greater than the radius of the half bounding circle corresponding to the sliding edge s. In this case, the whole robot can rotate freely inside the disc segment. The two rotation points S_1 and S_2 coincide with the center of the bounding half circle.

c. The radius r of the disc segment is greater than the cross-section, but smaller than the radius of the bounding half circle. In this case, the disc segment can contain only one end of the robot at a time. There are two different points of rotation S_1 and S_2 on the sliding edge s. See figure 5.3.a.

Computation: Recall that the alignment ranges of the robot inside the two cones often do not overlap. The gap between the two alignment ranges is bridged by a simple rotation about S-V in case b, and by a rotate-slide-rotate motion in case c. The rotate-slide-rotate motion is computed as follows:

1. Find the configuration range CP_1 of the robot, rotating about S_1-V, being outside of the corner's obstacle C, and inside of the convex boundary B of the V-shaped region. Similarly, we find the configuration point CP_2 for the robot M rotating about S_2-V. The range of orientation in CP_1 always overlaps with the alignment range in CC_1. There is always the obvious common orientation which makes the sliding edge s parallel to the generating edge e_1 of the cone C_1. The same observation holds for the configurations CP_2 and CC_2.

2. If the ranges of orientation in the two configuration ranges CP_1 and CP_2 overlap, then the path of the robot is composed of:

— a rotation about S_1-V from the current orientation of robot M to the nearest orientation θ_{12} common to the two ranges orientation in CP_1 and CP_2.

— a translation at orientation θ_{12} until the point S_2 on M coincides with the corner V. The vector of translation is the vector from point S_1 to point S_2 in the current frame of the robot. In this translation, the robot 'slides' on the corner V.

— a rotation about S_2-V from orientation θ_{12} to the nearest orientation in the alignment range $I(C_2)$.

Proof: The above path is collision-free.

— The two rotations about S_1-V, and S_2-V is collision-free, because they are valid motions respectively in the configuration range CP_1 and CP_2.

— The sliding of the robot on the corner V is also collision-free. To see this, we artificially create a region R bounded by the convex boundary B of the V-shaped region and a line λ. The line λ extends the sliding edge s of the robot at current orientation θ_{12}. See figure 5.3.b. Since the obstacle C is outside of the free half-plane of λ, and since the boundary B of the V-shaped region is convex, the region R as constructed is convex. With such convex region R, the robot is completely inside R before and after the translation. So, from the corollary in section 5.1, the translation is collision-free. ∎

Complexity: The case c has the longest time.

– The convex boundary has $O(n)$ facing edges and vertices.

– From section 4.3, the configuration range of the robot rotating outside of an obstacle C is computable in $O(n^3)$ time. The configuration range of the robot rotating inside a convex boundary B is computable in $O(n^3)$ time. So, CP_1 and CP_2 cost $O(n^3)$ time.

We deduce that, the 'rotate-slide-rotate' path about the corner is computable in $O(n^3)$ time.

Approach and Depart from a Corner

We have seen how the robot can turn itself around the corner. From the current cone C_1, we have some configuration cone CC_1. From the rotation of the robot about S_1-V, we have a configuration point CP_1. We now plan the approach path which links the two configuration volumes CC_1 and CP_1. We remember that the alignment range in CC_1 always overlap the range of orientation in CP_1. However, the two translation polygons generally do not intersect. So, we need to translate the robot through some region of transition. The transition region here is the V-shaped region.

The path of the robot approaching a convex corner is broken into two subpaths:

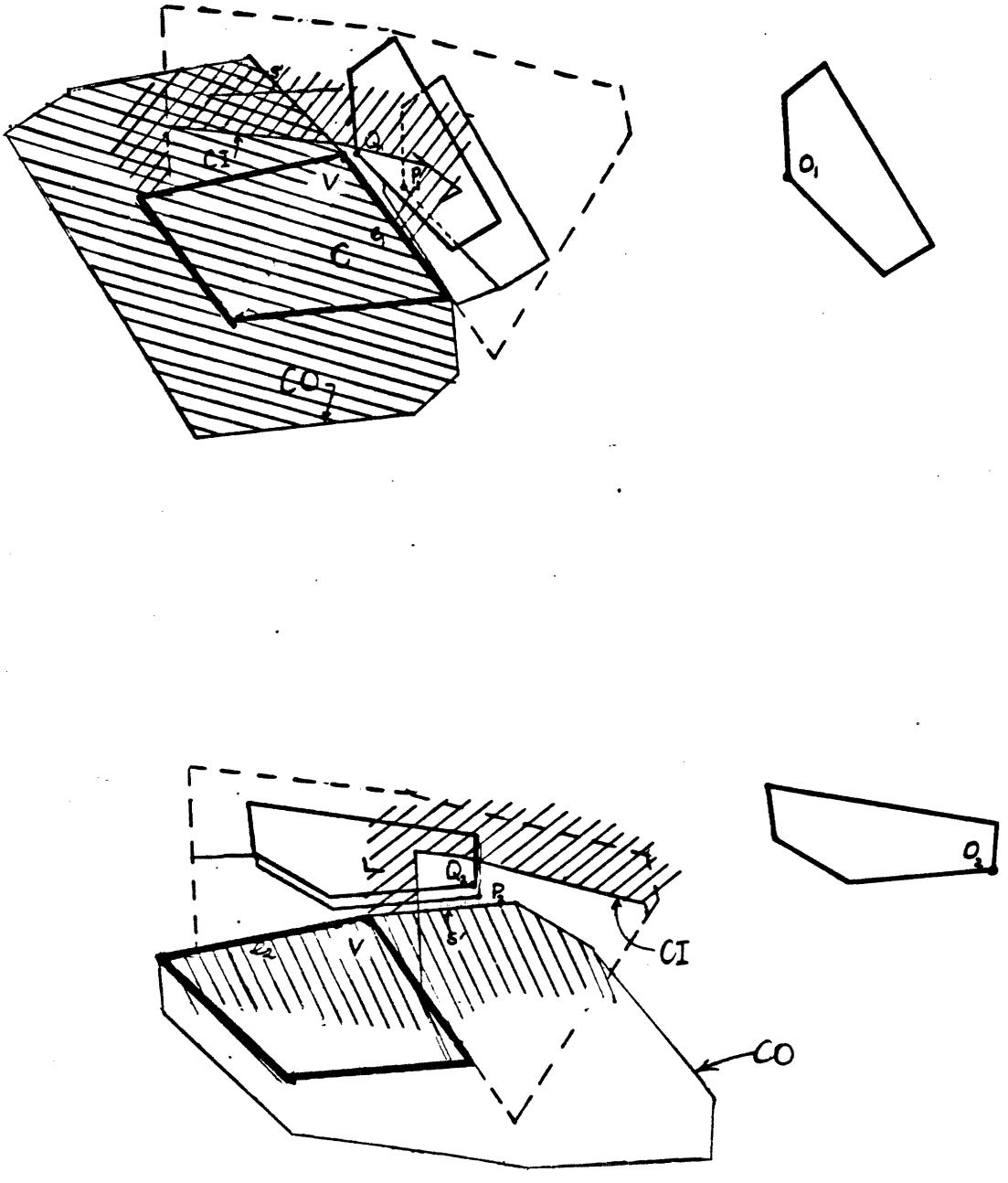
1. From the cone C_1, the robot gets inside of the V-shaped region. — First, the robot rotates inside the cone C_1 until it is in some orientation inside CP_1. Then, the robot translates inside the cone C_1 until it is completely inside the V-shaped region.

2. The robot translates inside the V-shaped region until the point of rotation S_1 on its sliding edge coincides with the corner V.

The top of figures 5.4 shows an example of an approach path. The bottom figure shows an example of a depart path. Depart paths are computed similarly to approach paths

Computation:

1. The first subpath is very similar to the subpath the robot uses to get inside a convex region from some current cone. To make the reduction, we construct a



Figures 5.4 The robot approaches and departs from the corner.

convex region R_1 from the non convex V-shaped region. The convex region R_1 is called the *approach region.*

The approach region R_1 is the intersection of the V-shaped region and the free half-plane of edge e_1. Edge e_1 is the constraining edge of the cone C_1 connected by the vertex V. Because the V-shaped region is concave only at V, the approach region R_1 as constructed is always convex. This subpath through the approach region R_1 is computed as in section 5.1.

2. Restricting the non convex V-shaped region to some convex region does not work for the second subpath. However, a path with translation only can be easily found by computing a three-point Visibility graph, (see below) and checking that the robot does not collide with the obstacles.

Proof: We'll show how the second subpath is contructed, and prove that it is collision-free. Throughout the following discussion, the orientation of the robot remains constant. Let's O_1 be the reference point on the robot. We shrink the moving robot to the reference point O_1, and inversely grow the boundary B of the V-shaped region, and the obstacle C. The grown obstacles are computed by taking the CI of the robot inside the convex boundary B, and the CO of the robot due to the obstacle C. In figure 5.4, the reference point O_1 on the robot is chosen such that the CO-obstacle has edge e_1 unshifted. The reference point O_1 describes a valid configuration of the robot if and only if point O_1 is outside of the CO-obstacle and inside of the CI-boundary.

The Visibility graph is fairly easy to compute. We want to link the current position P_1 of point O_1 with its final position Q_1, possibly through intermediary via-points.

— The straight translation is represented by the segment P_1Q_1. Since the robot is collision-free when its reference point O_1 coincides with the points P_1 and Q_1, the points P_1 and Q_1 must be outside of the CO-obstacle and inside of the CI-boundary. Because the CI-boundary is convex, the segment joining P_1 and Q_1 is totally inside the CI-boundary. We just need to check whether this segment cuts the CO-obstacle. If P_1Q_1 does not cut the CO-obstacle, or equivalently does not cut the edge e_1, then the straight translation represented by the segment P_1Q_1 is collision-free.

— The interesting via-points are restricted to the shifted point V' of the corner V due to the reference point O_1. Due to our choice of O_1, the edge e_1 and the vertex V are left unshifted. So, V' coincides with the corner V. Since the point V is already on the boundary of the CO-obstacle, it is a valid via-point if it is inside the CI-boundary.

If V is inside the CI-boundary, then the segments P_1V and VQ_1 represent collision-free translations inside the V-shaped region. To see this, we note that the segment P_1V is completely outside of the CO-obstacle because the segment is inside the free half-plane of edge e_1, and this free half-plane does not intersect with the convex CO-obstacle. The segment VQ_1 is also outside of the CO-obstacle

because the segment VQ_1 is in the free half-plane of edge s' on the CO-obstacle. Edge s' is the segment drawn by the reference point O_1 when the robot slides its sliding edge s on the corner V. Next, the two segments P_1V and VQ_1 are totally inside of the CI-boundary, because 1) the CI-boundary is convex and 2) all of the end points of the segments are inside of the CI-boundary.

From the three points P_1, V, and Q_1, we find the shortest translational path for the robot. The above discussion suggests that we can avoid the computation of the CI and CO, and just check that the robot is collision-free when its reference point O_1 coincides with the corner V. Such check can be done in $O(n \log n)$ time [19]. ∎

Complexity: Like a path through a convex region, the first subpath costs $O(n \log n)$ time. For the second subpath, there is a collision check of $O(n \log n)$ time. So, the path of the robot approaching/departing from a convex corner is computable in $O(n \log n)$ time.

5.4. Going Through a Star-Shaped Intersection

An Example

Figure 5.5.a shows a path through the star-shaped intersection I. The planning of a path through an intersection proceeds in two steps:

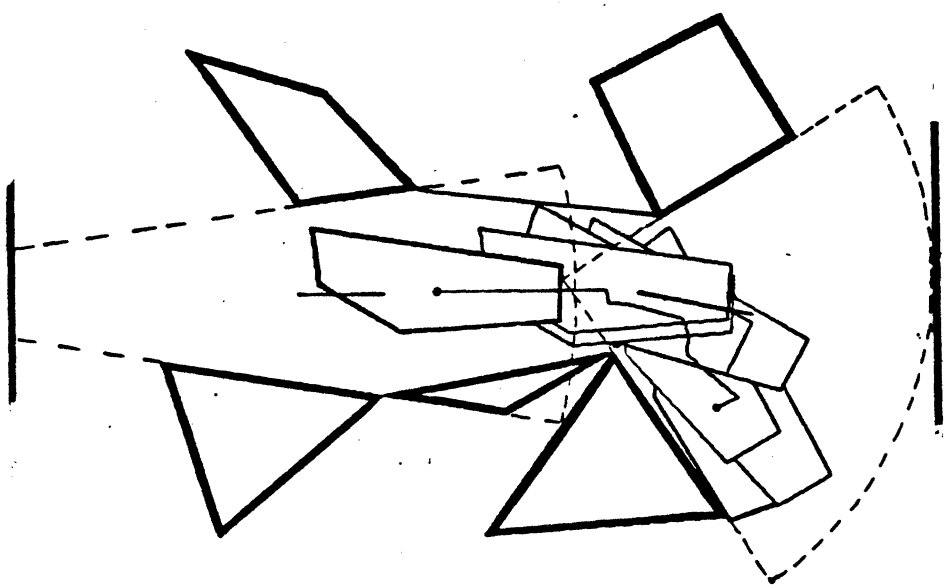
1. Retrieve from the description of the free space the intersection between the two cones. In section 3.5.4, we have discussed that the free space between the two branching cones C_1 and C_2 is captured by a fake cone C'. In figure 5.5.b, the fake cone C' is shown having generating edges ad and bc. Then, a case analysis classifies the relationship between the cones C_1-C', and C'-C_2 as:

— two cones sharing a fake corner. In the figure, the two cones C_1 and C' share a fake corner a. This fake corner has on one side a fake edge ad, and on the other side a real edge e_A. The V-shaped region around the corner a is used as transition medium for computing of a path from the fake cone C_1 to the next cone C'.

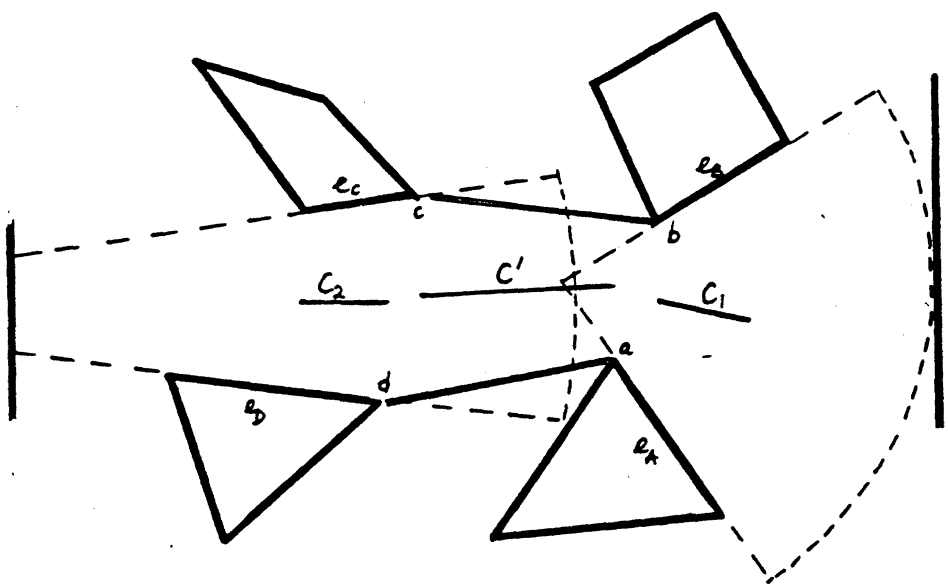
— two cones sharing a fake common convex region. In the figure, the two cones C' and C_2 share a fake convex region locally bounded by ad, bc, e_C, and e_D. This convex region is used as transition medium for computing a path from the fake cone C' to the next cone C_2.

2. Call the appropriate path expert to plan two subpaths: from the current cone C_1 to the fake cone C' of the intersection, then from the fake cone C' to the next cone C_2. Next, link the two subpaths to obtain the whole path going through the star-shaped intersection. See figure 5.5.a.

We discuss only the case analysis in step 1, This case analysis aims at capturing the local connectivity between the cones C_1-C', and between the cones C'-C_2. For the path computation in step 2, all the details can be found from the previous sections.



a.



b.

Figures 5.5 Going through a star-shaped intersection.

a. A path going through an intersection.

b. The free space in the intersection is described by a fake cone.

Local Links between the Cones in an Intersection

Since the three cones C_1, C', and C_2 do not have any common edge, the kinds of link which may be relevant are: common convex region and common corner. Either of the two links is decided from looking at the pair of constraining vertices shared by the two consecutive cones. The case analysis is as follows:

1. If the two vertices form two concave corners, then the two cones share a fake convex region. The fake convex region is formed from the four generating edges of the cones. In figure 5.5.b, the vertex c shared by the cones C_2 and C' forms a concave corner between the edges e_C and bc, bc is a fake edge. Similarly, the vertex d forms a concave corner between the edges e_D and ad. The fake convex region R is constrained by the two edges e_C, e_D from cone C_2, and the two edges ad and bc from the fake cone C'. The fake convex region R can have obstacles at the two far ends of the two cones, but never in between the two cones.

2. if one of the two vertices form a convex corner, then the two cones share a fake convex corner. In the figure, the two vertices a and b form two convex corners. The vertex a is chosen as vertex of transition because a path around a is more likely to be shorter. The V-shaped region describing the free space around the vertex A can be found from the edges e_B and bc, and the two cones C' and C_2 in constant time.

Complexity of a Path Through an Intersection

Let n be the number of edges in the workspace. A subpath through a fake convex region R can be computed in $O(n \log n)$ time. The V-shaped region describing the free space about a fake convex vertex can be computed in constant time. A subpath around a convex corner can be computed in $O(n^3)$ time. So, the path of the robot through a star-shaped intersection is computable in $O(n^3)$ time.

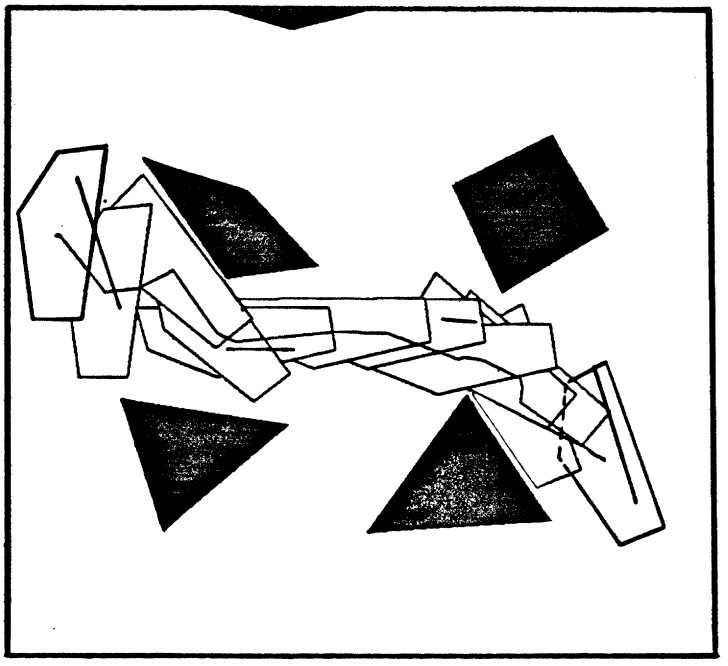
5.5. The A*-Search Finds Global Paths

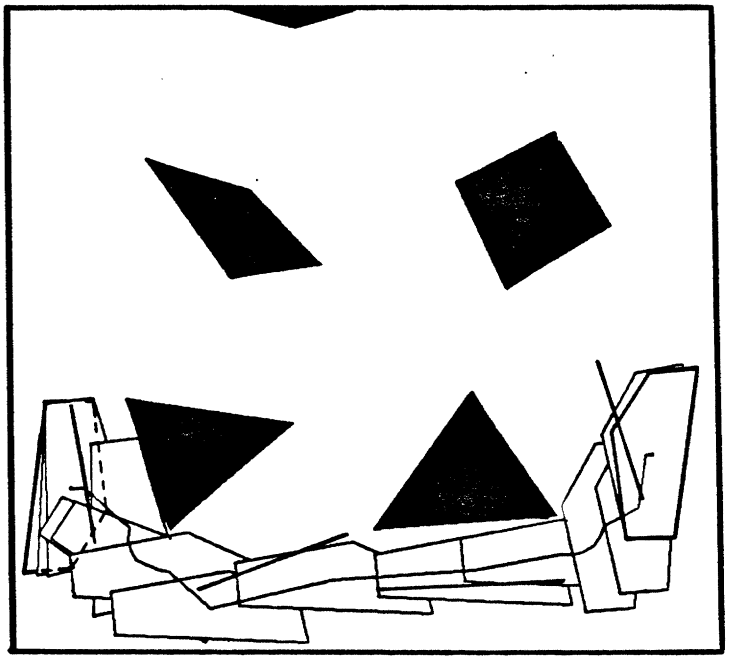
We have seen how the four experts compute local paths along the links between the cones. These short pieces of path are linked from one to the other, as an A^*-search [13] probes the network of linked cones. In our search network, cones are considered as nodes, and links as paths. Figures 5.6. shows two paths found by the A^*-search. The search for a collision-free path of the moving robot proceeds in two steps:

1. Identify the start and final cones. The start (resp. final) cone is a cone which totally encloses the moving robot when this later is in its start (resp. final) configuration.

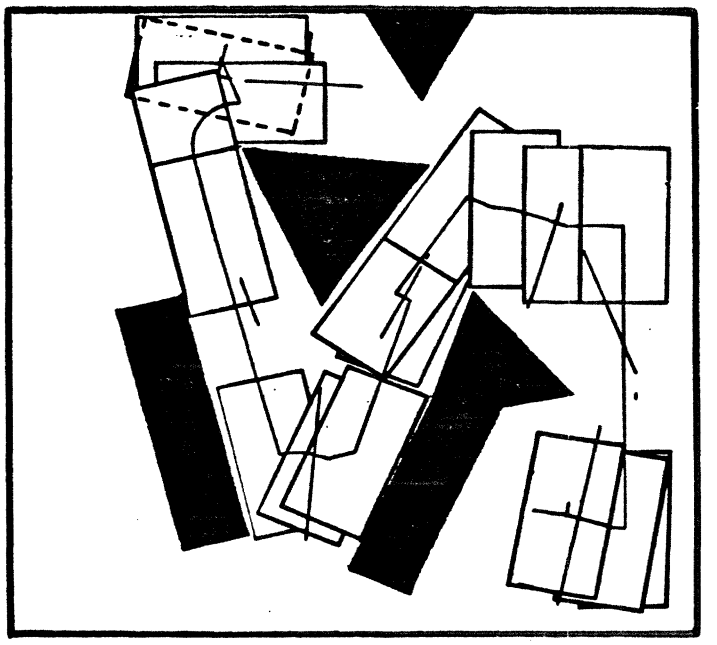
2. Search the network of linked cones, from the starting cone to the final cone. This step is done by an A^*-search, with a cost function depending both on translation and rotation.

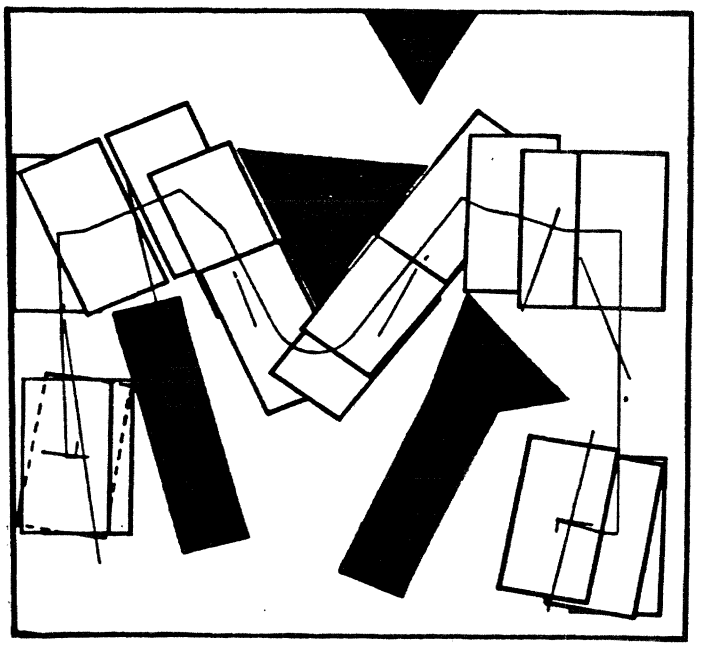
In the current Path Planner, the search for a path can fail when:





Figures 5.6 Some examples of paths found by the A^*-Search.





Figures 5.7 Some paths in cluttered environments.

— There is no identifiable starting (or ending) cone. In this case, the moving robot M either overlaps with some obstacle, or overlaps with many cones without being inside any of them. The validity of the starting (or ending) configuration can be checked by intersecting the robot with all the obstacles in the workspace. If the robot overlaps with many cones without being inside any of them, a local expert can find a translational path, after which the robot is inside one of these cones. This expert is not implemented in the current version.

— Some intermediary cones are too small, and the search gets lost. This is a serious limitation of our method. Our method depends on capturing convex local regions which are big enough so that we can find good sets of valid motions for the robot inside these regions. A small cone which cannot contain the robot has an empty configuration cone. In the search network, this corresponds to a node through which there is no path. Fortunately, many of these cones can be extended by the neighboring free convex regions. This kind of extension is provided directly by the free space description. Free convex regions have proved to be a good back up for small cones. But the free convex regions can also be many, and too small to enclose the robot. We see this case when the obstacles have many small edges like the teeth of a saw blade. It seems that a preprocessing step which smoothes out small local variations is necessary.

Both the description of the free space and the search for path can be badly fooled if the obstacles have many small edges and corners. In finding the convex regions and cones, in computing the alignment ranges of the robot inside the cones, we have relied on the edges of the obstacles. We have implicitly made the assumption that the edges of the obstacles are good constraints for translation. With the edges as good constraints for translation, a convex corner connecting two edges is seen as a 'break' or 'discontinuity' in translational constraints. Many small edges and corners means many cones, and many of them are too small to contain the moving robot. The Voronoi diagram of the free space between such obstacles is also fuzzy. The Voronoi diagram has a lot of short edges, and points of intersection. However, we can always smooth out small edges and approximate the obstacles with some fixed number of edges.

The transformation of the free space between the obstacles into a network of linked cones takes about 40 seconds. The search for a path takes typically from one to three minutes in compiled LISP on an MIT Lisp Machine.

6. CONCLUSION

Problems for Future Research

1. Find efficient algorithms for finding the free convex regions, and the star-shaped intersections, or prove that the problem is NP-complete.

2. Investigate the possibility of having a *CO*-type expert to compute the path around a convex vertex. It seems that we have pushed too far the idea of capturing simple convex regions and computing the *CI* of the robot inside these regions to find paths.

3. Extend the obtained results to planning paths for moving robots, and articulated arms in $3D$.

Extensions to a 3D Path Planner

A generalized cone in three dimensions describes a freeway between two 'opposite' faces, or a bottle-neck between a face and an edge. A free convex region, (resp. a star-shaped intersection) describes the convex polyhedral region between many faces, (reps. edges). The four types of links can be generalized similarly.

In two dimensions, we observe that the sparser is the workspace, the more the cones overlap one another, and the less useful are the paths along the edges and around the vertices. From this observation, we predict that in three dimensions, the cones will highly overlap one another, and that the common edge and common face links will not be as useful as the other two links. In other words, the free space between the obstacles should be described as the union of non overlapping cones between faces. A cone can be extended by its adjacent neighbors, forming a channel. The robot should translate in the middle of these channels rather than follow the faces and go around the edges of the obstacles. Constraints in the *CSpace* should be expressed by the *CO*-volumes, because the channels, or unions of the adjacent cones, are generally non convex. This direction of research was investigated by Donald [4].

We are confident that the description of the $3D$ space between the obstacles as a graph of non overlapping cones will yield a faster Path Planner. Such graph localizes the computation for paths.

References

[1] **Brooks, R A.** "Solving the find-path problem by good representation of free space," *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-13** (1983).

[2] ___ "Planning collision free motions for pick and place operations," *IEEE Transactions on Systems, Man, and Cybernetics* (May, 1983).

[3] **Brooks, R A and Lozano-Pérez, T.** "A subdivision algorithm in configuration space for findpath with rotation," *International Joint Conference on Artificial Intelligence* , Karlsruhe, Germany, 1983.

[4] **Donald, B R.** "Hypothesizing channels through free-space in solving the find-path problem," *IEEE Transactions on Systems, Man, and Cybernetics* (June, 1983).

[5] **Drysdale, R L.** Generalized Voronoi digrams and geometric searching, Department of Computer Science, Stanford University, 1979.

[6] **Forbus, Kenneth D.** "A study of qualitative and geometric knowledge in reasoning about motion," Massachusetts Institute of Technology Artificial Intelligence Laboratory, AI-TR-615, 1981.

[7] **Graham R L.** "An efficient algorithm for determining the convex hull of a finite planar set," *Information Processing Letters* **1** (1972).

[8] **Lozano-Pérez, T.** *Robot motion: planning and control* , Brady, M. et al. eds., MIT Press, Cambridge, MA, 1983, Chapter 6: Task planning.

[9] ___ "Spatial planning: a configuration space approach," *IEEE Transactions on Computers* **C-32** (February, 1983).

[10] ___ "Automatic planning of manipulator transfer movements," *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-11, No. 10** (1981).

[11] **Lozano-Pérez, T and Wesley, M A.** "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM* **22**, 10 (1979).

[12] **Moravec, H P.** "Obstacle avoidance and navigation in the real world by a seeing robot rover," *Communications of the ACM* (Sept. 1980).

[13] **Nilsson, N.** *Principles of artificial intelligence* , Tioga Publishing Co., Palo-Alto, 1980.

[14] **Ó'Dúnlaing, C. and Yap, C.** "The Voronoi method for motion planning: I. The case of a disc," Courant Institute of Mathematical Sciences, New York University, Report No. 53, 1983.

[15] **Schwartz, J T and Sharir, M.** "On the piano movers' problem, I: The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers,"

Courant Institute of Mathematical Sciences, New York University, Report No. 39, 1981.

[16] **Schwartz, J T and Sharir, M.** "On the piano movers' problem, II: general techniques for computing topological properties of real algebraic manifolds," Courant Institute of Mathematical Sciences, New York University, Report No. 41, 1982.

[17] **Shamos, M I.** "Geometric complexity," *7th Annual ACM Symposium on Theory of Computing* (1975).

[18] **Shamos, M I and Hoey, D.** "Closest-point problems," *16th Annual IEEE Symposium on Foundation of Computer Science* (1975).

[19] **Shamos, M I and Hoey, D.** "Geometric intersection problems," *17th Annual IEEE Symposium on Foundation of Computer Science* (1976).

[20] **Udupa, S.** Collision Detection and Avoidance in Computer-Controlled Manipulators, Ph.D Thesis, Department of Department of Electrical Enginnering, California Institute of Technology, 1977.

[21] **Winston, P H.** *Artificial Intelligence* , Addison Wesley, 1979.

Acknowledgements

I would like to thank Professor Tomás Lozano-Pérez for his generous guidance and help. My debt to Lozano-Pérez can never be repaid. A lot of the ideas presented here came from discussion with Lozano-Pérez, and so should be credited to him.

Rodney Brooks introduced me to cones, and much of the free space description should be credited to him as well.

Tomás Lozano-Pérez, Bruce Donald, Michael Brady read drafts and provided valuable comments.

I would like also to express my deepest thanks to my family and my fiancée who have been encouraging me all along.