# Algorithms for Optimal Design of Robots in Complex Environments

**Krasimir Kolarov**

**Interval Research Corp.**
**1801-C Page Mill Road**
**Palo Alto, CA 94304**

## Abstract

*The goal of our work is to find the optimal design of a robot that can reach everywhere in an environment with obstacles without collisions. The main questions we are concerned with are: what is the most appropriate type for the links of the robot? what is the minimum number of links that are needed to cover every point in the environment? and what is the best placement for the robot?*

*We describe some algorithms for finding the set of points in the environment that can reach all other points with a minimum number of links. Initially the obstacles are modeled as convex polygons and subsequently we discuss extensively the modifications that those algorithms require to cover curvilinear, non-convex and three-dimensional obstacles.*

*We derive several theorems that establish upper and lower bounds on the number of links for both planar and spatial cases. We describe some algorithms for minimizing the upper bounds to the optimal number of links for the environment.*

*We generalize the basic problem for the cases when both the robot and the environment are designed simultaneously, when we deal with moving robots and obstacles, and when we have multiple robots or robots with variable structure.*

## 1. Introduction

Robots are used in industrial automation to perform repetitive tasks or to lift, move and manipulate objects that are heavy and dangerous to handle. They also work underwater, in space, and other environments hostile to human life. To achieve their purpose well, robots should have a high degree of autonomity, good reasoning and learning capabilities and appropriate construction for their tasks. However the vast majority of the existing industrial robots perform simple tasks and have similar structures. Accordingly the majority of the research in the area of robotics has been concentrated on incorporating a high degree of intelligence, learning capabilities, control and mobility in the existing industrial robots. This is understandable, considering that the manufacturers and the users of robots systems would like their robots to have better capabilities.

Another way to improve performance is to design a robot for its specific workspace. That is why the main problem we are going to discuss in this paper, is how to design a robot that is most appropriate for a given workspace. In doing so we will introduce techniques similar to the ones used in "robot motion planning" and "computational geometry".

Researchers in the area of robot motion planning are aiming at providing robots with good reasoning capabilities (see [12]). Given a robot working in some given environment, they are trying to come up with some good algorithms and techniques that will allow the robot to grasp an object from one place and transfer it to another without colliding with the other moving or stationery objects in the environment. This operation is to be done in an optimal fashion (see [4]). Often this proves to be a difficult task, especially when the structure and the kinematics of the robot are not appropriate for the layout of the environment and the activities performed in it.

One can try to use a design optimization approach for the planning of a robot path which avoids the obstacles (see [21]), or to identify the kinematics of the robots with respect to their tasks [1]. However these approaches still do not guarantee the best fit between the environment and the robot operating in it, because they are both given appriori. In order to have an opportunity for optimization, we need to fix at most one of the components, and find the other one in such way that it best fits. We believe that the fixed component (if any) should be the environment, and the robot should be the one that varies in structure and placement to best fit the environment. Furthermore, we will also consider the problem when both the environment and the robot are allowed to vary in order to achieve more efficient operation of the robot.

The different aspects of optimal design of robots have been considered in the literature (see [3], [16] and [17]). We will concentrate in this paper on finding the best structure for a robot working in an environment with obstacles, and the best place for this robot in the environment. What we want to achieve is full autonomy, that is we want the robot to be able to reach every point in the environment, except of course the points interior to the obstacles. Thus the basic problem we will consider is:

Given an environment containing obstacles, what are the minimum number of links and the best placement of a manipulator operating in this environment that will allow it to reach any point in the environment that does not belong to the obstacles.

The set of points in the environment that do not belong to the obstacles is usually called "free space". We will use this term here and we will denote this space with "*E*". There are different types of joints that could be used between the links -- revolute, prismatic, spherical, ball-and-socket, etc. The two most widely used joints are revolute and prismatic. We will combine those two joints in our robot in order to achieve a larger "reachable space" (the points that can be reached by the end-effector of the robot) and greater flexibility to reach around obstacles.

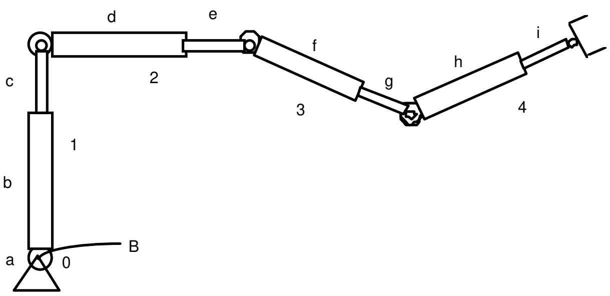


Fig.1 A robot with four telescoping links.

The primary joint type we will employ is a one degree of freedom (dof) prismatic joint which folds into itself like a telescope to some minimal length. Our robot is formed by connecting each telescopically jointed link to a revolute joint at its other end. The revolute axes are perpendicular to the prismatic axes, as in Figure 1. We will call the links thus formed "telescoping links".

Each telescopic joint connects two links. So in Figure 1 we have 9 links (*a, b, c, d, e, f, g, h, i*) including the ground link but not the end-effector. However in this paper it will be more convenient to think of a link in a different way. Since the only function of the prismatic joint is to change the link's length, we consider the telescoping joint to be part of a link's internal structure. Hence we will say we have in Figure 1 only 4 moving links (*1, 2, 3, 4*). In the plane, these types of links give more flexibility than a traditional fixed link-length, revolute-jointed manipulator. Such designs also make it easier to extend our analysis to other types of environments, such as ones where the obstacles can change their place and shape and the robot can move around rather than being on a fixed base. These structures can also be easily generalized to three dimensional spaces. In the limit, when the number of telescoping links goes to infinity, we can use these structures to consider problems similar to the ones discussed in [5].

Links with adjustable length are used in the "Adjustable Robotic Mechanisms" for low-cost automation, considered in [11] as well as in most of the walking robotic devices. Telescoping structures are especially popular in construction and building automation.

Once we have decided on the construction for the robot, the problem we want to solve becomes the one of finding the minimum number of telescoping links (we refer to them from here on simply as "links") that are needed to cover the whole free space.

In Section 2 of this paper we discuss this problem for a planar environment and obstacles represented as convex polygons. Using notions from computational geometry, we describe, in Section 2.1, an algorithm that finds the minimum number of links that cover *E* for any point of the free space. The complexity of this algorithm is in the worst case $O(m.n.\log n)$, where $m$ is the number of obstacles and $n$ is the number of vertices of the environment and the obstacles. Section 2.2 describes algorithms for finding the set of points that are most appropriate for best placement of our robot in the environment.

Section 3 generalizes the discussions in Section 2 for more complex shapes of the obstacles and for higher dimensions of the environment. In this, an algorithm for general convex obstacles is described in Section 3.1, which is primarily based on a discussion of generalized convex obstacles. The discussion of general non-convex obstacles in Section 3.2 shows that in our problem sometimes the presence of reflex vertices simplifies the work of the algorithm. Finally a spatial (3-Dimensional) environment is considered in Section 3.3, in which it is shown that the algorithms developed in Section 2 can be easily extended to $2^{1/2}$D environments using horizontal slices of the environment.

Sections 2 and 3 give a complete algorithmic treatment of our problem, i.e. if we are given an environment with obstacles, applying those algorithms we can choose the best place for the robot operating in this environment and determine the minimum number of links that the robot needs.

Section 4 derives several estimates for this minimum number of links in terms of the number of obstacles and vertices in the environment, using a more theoretical approach. Several theorems estimating this number for the planar (Section 4.1) and the spatial (Section 4.2) case are derived. Some algorithms for optimizing these estimates are discussed in Section 4.3.

We also consider the case when we design both the robot and the environment simultaneously. In Section 5 it is shown that if certain conditions are satisfied, we can always place the obstacles in the environment in such a way that the best designs of the robots contain only three prismatic links (in the planar cases) or three telescoping links (in the spatial cases). A brief discussion of moving robots and obstacles, multiple robots and reconfigurable structures can be found in [7].

Finally Section 6 summarizes our results and makes some suggestions for future research in this area.

## 2. Convex Environment and Obstacles

In this section all obstacles will be represented by convex polygons in a two-dimensional environment. For simplicity of the figures illustrating the material, we will consider a rectangular outside boundary of the environment, although all algorithms are implemented for any convex outside boundary.

Our optimization problem is to find a set of points in the free space such that, if we place the base of the robot in any point of this set , we will need a minimum number of telescoping links to reach all the other points in the free space. In fact we have to solve two distinct problems:

I. How to find this set of points?

II. If we place the base of the robot at a point in the set above, how to find the minimum number of links needed to reach all points in the free space?

We will start with problem II which we will show to be equivalent to finding the visibility polygons for a given point in the environment.

### 2.1 Visibility From a Point

Let us denote with point $B$ the base point of the robot and by $P_{ij}$ the vertices of the polygons, where the first index $i = 0,1,2,...,m$ denotes the obstacle ($0$ for the environment $EN$ ) and $j = 1,2,...,m_i$ denotes the vertex of an obstacle. We will increase $j$ clockwise for the obstacles and counterclockwise for $EN$. Let $n$ be the total number of vertices in the environment.

We want to determine the minimum number of links that are needed to reach from point $B$ any other point in the free space $E$. All points in $E$ that can be seen from point $B$ with one link (as mentioned before by 'link' we understand 'telescoping link') are depicted in Figure 2.1. Assuming no lower and upper bounds on the length of a link, those points constitute the shaded region in Figure 2.1 , which is sometimes referred to in the literature as the visibility polygon $V(B)$ for point $B$.

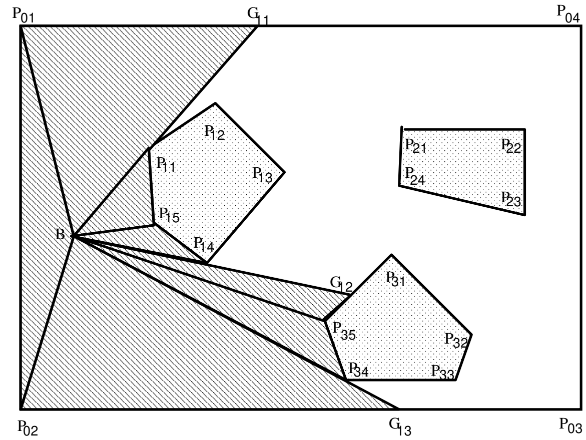


Fig.2.1 Region with Visibility 1 from point *B*.

If we consider the obstacles in our environment as holes we can formulate our problem as a visibility problem for polygons with holes. Suri and O'Rourke [22] consider visibility polygons with holes, and show (theoretically) an algorithm that runs in $O(n.logn)$ time which they prove optimal by reduction from the problem of sorting $n$ positive integers. Lee and Chen [14] present an algorithm for computing the visible edges of a set of $m$ non-overlapping convex polygons that computes only the first visibility region from a given point. We want an algorithm that builds all the visibility polygons from a given point.

In our case all points that can be reached from $B$ with two links are those that can be seen from the points in $V(B)$ with one link - we denote this set of points with $V_2(B)$. This set is depicted as the shaded area on Figure 2.2.

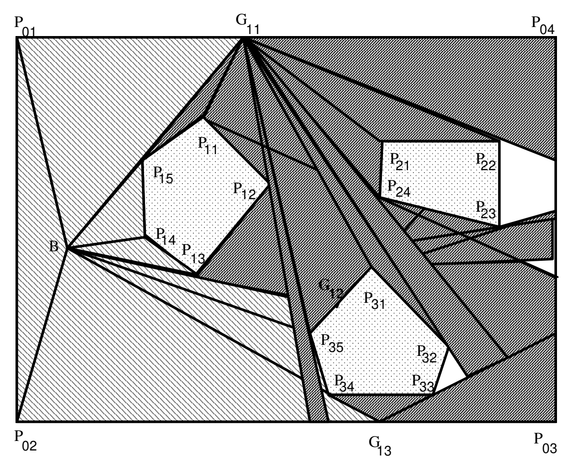


Fig.2.2 The visibility from point B is three.

In the same manner we can build the set $V_3(B)$ and so on. We continue this procedure until all the points in the free space $E$ are included in one of the visibility polygons $V_i(B)$. We will denote the first number $i$ for which all the points in $E$ are covered as $VMIN$, which stands for "minimal visibility".

In the computational geometry literature this is also known as the "link radius" of the environment, while the link distance between two points is the minimum number of straight segments (links) that connect the points without collisions with the obstacles.

The link center of a simple polygon P is the set of all points in P whose maximal link distance to any other point of P is the smallest possible. Our problem (#I) can be formulated to be the same as finding the link center for a polygon with holes. We do not know of any published research that deals with this problem.

In addition to the above notations, we can define the term "link diameter". This will be the maximal link distance between any two points in the polygon. In our notations we will also use *VMAX* (from maximal visibility) as a term for the link diameter in an environment with obstacles.

In Chapter 2 of [7] and in [8] we have described in detail an algorithm that builds all visibility polygons from a given fixed base point $B$ in an environment with obstacles. The basic approach for building the visibility regions at a certain level is to draw tangents from certain points, called "generating points", towards the vertices of the environment and look for intersections of these tangents with the edges of the environment. In our example point B is the generating point for the visibility polygon on level 1, while points $G_{11}$, $G_{12}$ and $G_{13}$ are the generating points for the second level. The visibility polygons are sets of triangles that are formed by the tangents from the generating points and the edges of the obstacles and the environment. In addition, for completeness of the algorithm, we include generating points and triangles created by the mutual tangents between the obstacles.

The algorithm terminates when the union of the visibility regions $V_i(B)$ $(i=1,2,...,k)$, covers the whole free space with a minimal link distance $k$. Theorem 2.1 in [7] shows that this is equivalent to have the entire length of all the edges of the environment be reached from point $B$ with $k$ links.

The worst case complexity of our algorithm is $O(m.n.\log n)$ where $m$ is the number of obstacles. In addition to the minimal number of links $k$ for point $B$, we can automatically extract for <u>every</u> point in $E$: the visibility of this point from the base point; the visibility triangle it corresponds to and a possible link path from the base point to this point. We also find as a by-product of our program the minimum and the maximum lengths of the links for each level. This is useful information for planning the actual sizes of the links.

To be able to model the links as straight line segments we initially grow the obstacles by the width of the links. In fact we can also grow the obstacles with an additional safety margin if we do not want a hard contact between the links and the objects in the environment.

In the following section we consider the question of choosing a base point.

### 2.2 Link Center of the Environment

Once we know how to find the minimum number of links for a given point, we need to decide where to place the base point so that we have the smallest link radius overall (question I). As we can see from Figure 2.3 different points have different visibility of the environment. If we place the base of the robot at point $A$ than we can see (reach) all points in the free space with two telescoping links. Point $B$ however requires at least 3 links to reach the inside points in the shaded triangle.

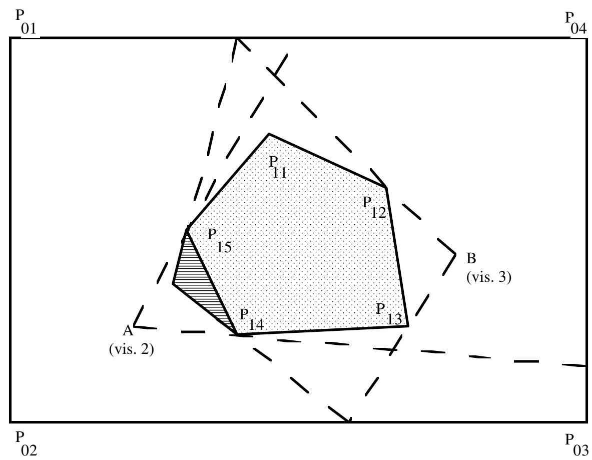


Fig.2.3 Points with different visibilities.

The algorithm for building the link center of the environment (the set of points that can reach all other points with minimal number of links) is based again on Theorem 2.1 from [7]

Theorem 2.1. If the entire length of all the edges of the environment $E$ can be reached from some point $B$ with $k$ revolute-jointed telescopic links, then all points in $E$ can be reached from $B$ with $k$ revolute-jointed telescopic links.

We can derive that he link center of $E$ is the intersection, which is not empty, of the same lowest level $k$ visibility regions for all points on the edges of $E$. This is true because if a point $A$ belongs to the link center with a link radius $k$, that means that this point can reach all the other point in $E$ with maximum $k$ links. Thus it can reach all points on the edges of $E$ with $k$ links, i.e. it can be reached from all those points with $k$ links as well. Then this point $A$ belongs to the intersection of the $k$-th visibility regions of all those points of the edges. As we show in Theorem 2.3 in [7] this $k$ is necessarily the minimal one for which this event occurs. Thus the general algorithm for building the link center is as follows:

```
begin
  k:= 0;
```

```
  repeat
      k:= k+1;
      S_k:=E;    {E is the whole free space}
      for  i=1,...,m do
         for j=1,...,m[i] do
             begin
             Build    visibility    region
V_k(e_ij);
      {e_ij  denotes the edge P_ij P_ij+1 }
             S_k = S_k ∩ V_k(e_ij);
             end;
  until  S_k  is non-empty ;
end;  {S_k is the exact link center}
```

As we can see from this algorithm we actually build the visibility polygons from all edges in the environment rather than using the infinite number of points along those edges. Such visibility polygons consist of the set of points in the environment that can see the entire edge, not just a point on the edge. As we can see in Figure 2.4 this set is the simply-connected component of the intersection of the visibility polygons of the two end-points of the edge.

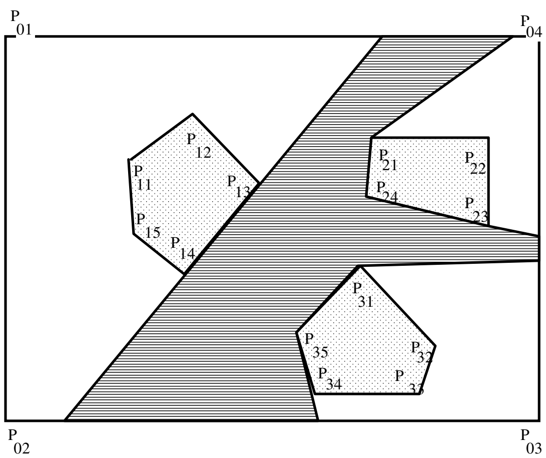


Fig.2.4 Visibility polygon of edge $P_{13}P_{14}$ .

This property is formally proved in [7] in Theorem 2.4. There, a detailed algorithm is explained of how to actually build such visibility region of an edge by going around clockwise along the visibility triangles defined by the two end points and merging them in a correct order. This algorithm builds the first visibility polygon $V_1(e)$ and has the same complexity as the one for visibility from a point.

One might think that we could find the next level visibility polygon by taking the generated points from the first level and use them to form the next level. However we can show that there exist points that cannot see any of the vertices of the visibility polygon $V_1(e)$ of an edge $e$ with $k$ links, but can see the edge $e$ with $k+1$ links.

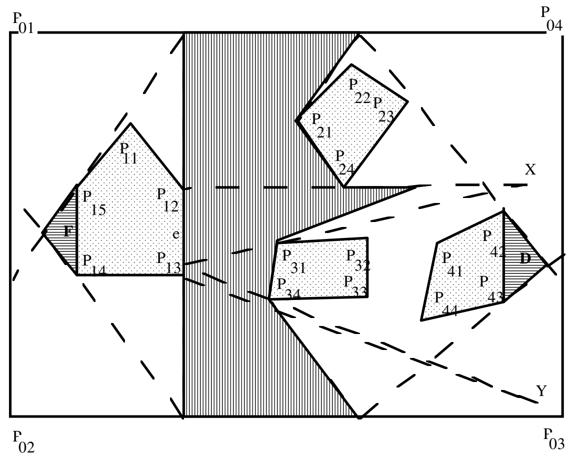


Fig.2.5 Partial visibility of higher order.

Consider the example in Figure 2.5. The visibility polygon $V(P_{12}P_{13})$ of the edge $P_{12}P_{13}$ is the vertically shaded area on the figure. We can draw the tangents from the vertices of this polygon, and determine the set of points that can see at least one point from $V(P_{12}P_{13})$ with one link. Those points can see the entire edge $P_{12}P_{13}$ with two links. There are two horizontally shaded triangles $D$ and $F$ on the figure that do not belong to any of the areas above, i.e. we might think that we need at least three links for the points inside those triangles to reach the entire edge $P_{12}P_{13}$ . However this is not the case for the points in the right triangle $D$. They can see both points $X$ and $Y$ with only one link, and every point on the edge $P_{12}P_{13}$ can be seen from either $X$ or $Y$ with one link. Consequently the points in $D$ can see the whole edge $P_{12}P_{13}$ with two links. In other words when we go to higher levels of visibility the problem becomes much more complicated, because one point can see part of the edge from one side of the some obstacle, while the other part (or parts) of the edge are seen from another side of the obstacles. To deal with this theoretically we introduce a construction called "X-form" (see [7], [10]).

The terminology for the X-form is illustrated in Figure 2.6. The part of the edge that we want to see will be called the ''base'' of the X-form and the opposite side of the form will be called the ''top''. The tangents that outline the X-form are called ''generating lines'' of the form, and their intersection point $V$ is the ''vertex'' of the X-form. The part inside the X-form between the vertex, the generating lines and the top is denoted as the ''upper part'', and the corresponding part for the base is called the ''lower part'' of the form. Any line $l$ that intersects the lower part, passes through the vertex, or intersects the upper part but not the top of the X-form, is

**said to ''cross'' the form. Let *U* be a point in the free space *E* for which there exists a line *l* that crosses the X-form. We will denote the intersection points of *l* with the X-form with *A* and *B* (if *l* crosses through the vertex *V* only, then *A* = *B* = *V* of the X-form).**

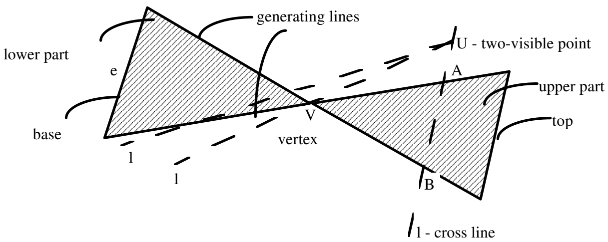


**Fig.2.6 X-form of an edge.**

**We will say that *U* is a ''two-visible'' point for the X-form if:**

**– when *l* crosses the lower part, the segment *UA* is entirely in free space, i.e. does not intersect any of the obstacles and is inside the outside boundary;**

**– when *l* goes through the vertex *V* of the form, the segment *UV* is entirely in free space;**

**– when *l* crosses the upper part the segment *UB* is in free space.**

**From this definition follows that if *U* is two-visible for an X-form, then *U* can reach all points in the X-form with two telescoping links. Consequently a point *U* will be able to see the entire edge *e* of some obstacle with two links if there exist one or more X-forms that are two-visible from *U* and whose bases cover *e* completely.**

**If we want to check whether a point *U* can see an edge *e* with two links, we can apply the following hierarchical algorithm:**

**– Form all X-forms that are based on the whole edge *e* .**

**– Check if any of those forms is two-visible from the point *U* . If yes we are done and two links are enough for *U* to see *e*. Exit.**

**– If the above condition is not satisfied then draw all common tangents between the obstacles (both inside and outside), that intersect the edge *e* .**

**– Those tangents will form X-forms with bases on parts of the edge *e*. Find all forms that are crossed by the lines from *U* to the vertices of *E*.**

**– Form the union of the bases of all X-forms above that are two-visible from *U* .**

**– If this union covers the whole edge *e* then we are done, and point *U* can see the whole edge *e* with two links. Exit.**

**– If the union is part of *e* then *e* is not completely visible from *U* with two links. If the union is empty we cannot say anything. Exit.**

**– If however this union covers at least one point *W* of *e* then we can conclude that *e* can be reached completely from *U* with three links, because on the third level all points of *e* can be reached from *W* with one link. Exit.**

**The work of this algorithm is illustrated on Figure 2.7. Although there is no line through point *U* that crosses either of the X-forms *X1*, *X2* or *X3* , using common tangents we can see that the two X-forms *X11* and *X12* are two-visible from *U* , and the union of their bases is the edge *e* . We conclude that *U* can see the entire edge *e* with two links.**

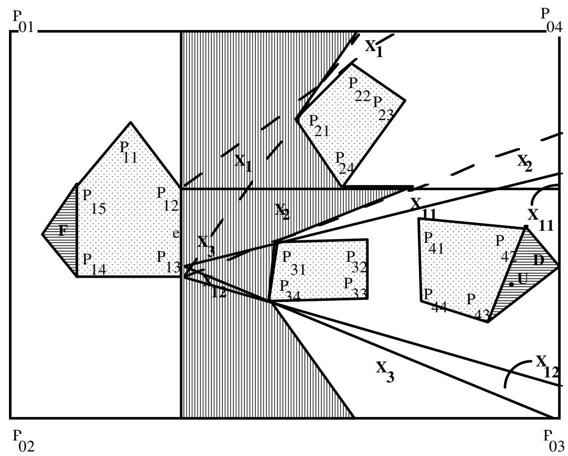


**Fig.2.7 Three-visible polygons of an edge.**

**More details on the actual implementation of X-forms can be found in [7]. We can use the procedure above for finding third, fourth, etc. level visibility polygons of an edge (if necessary) and the criterion for termination is the same as for discrete points. In our example the maximum number of links necessary is 3.**

**The result of the application of the algorithm for finding the link center for the environment in Figure 2.1 is depicted in Figure 2.8. The link center is the set of points that belong to the unshaded area in the figure. Thus if we place the base of our robot in any point in this area, we are guaranteed to reach all other points in the environment with minimum 2 telescopic links. The choice of a particular point can be made using heuristics, knowledge of the environment, or by request.**

**As we can see on Figure 2.8, small part of the shaded area in the free space is darker than the rest of it. This is the difference between the exact link center in *E* and an approximate link center (ALC) built using the vertices and the midpoints of the edges in *E*.**

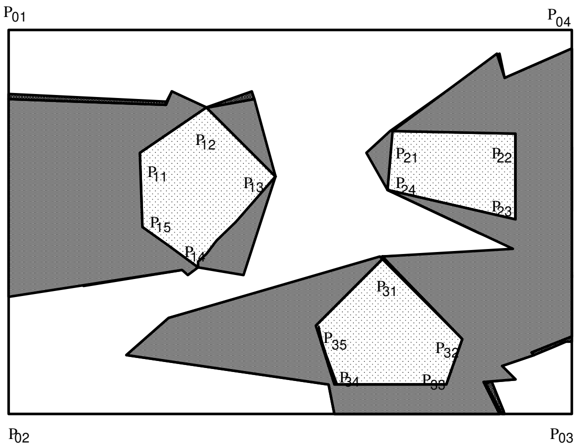

Fig.2.8 Link center for convex obstacles.

As we mentioned before, the exact link center requires the use of all (infinite) number of points on the edges of the obstacles. However for some cases we can use approximations to this link center by using finite number of points along each edge. For the figure above we have used for each edge its end-points and its mid-point. The advantage of this approximation is that we don't have to go through the complicated process of building and intersecting X-forms. The complexity of this ALC algorithm is the same as the one for the exact link center, but ALC is faster and straightforward extention of the point visibility and the results are very close as illustrated in Figure 2.8 (the exact link center is a subset of any of the ALC's).

The algorithms for visibility from a point and building of an ALC are implemented in Think Pascal and running in real time on a Macintosh computer. Very detailed description of the procedures for building of the exact link center can be found in [7]. In the next section we will extend our analysis to general types of obstacles and environments.

## 3. General Shape and Dimension of the Obstacles

In this section we will use the algorithms for visibility and link properties above to generalize our discussion going from convex polygons to generalized polygons, to general convex obstacles, to general planar obstacles and finally to a 3D environment and obstacles.

### 3.1 Algorithms for Convex Obstacles

As a first step we will assume that the shape of the obstacles can include both straight edges and arcs of circles. A convex two-dimensional object, whose edges are either straight segments or arcs of circles is called a "generalized polygon" and is extensively discussed in [13].The significance of generalized polygons comes from the notion of "Configuration space" (or C-space) in motion planning (see [12]).

We can approximate the shape of a moving robot in a 2D environment with its minimal bounding circle. Now instead of planning the motion of a circle around polygonal obstacles, we can grow the obstacles with the radius of the circle and plan the motion of a point around these grown obstacles (known as C-obstacles). In this case these C-obstacles are actually generalized polygons, i.e. there edges are either straight segments or arcs of circles.

We will also use generalized polygons as a stepping stone to more general obstacles.

The theory for generalized polygons is analogous to the treatment of convex polygons. To start with, the visibility region, from a point in an environment with obstacles represented as generalized polygons, is formed by drawing the tangents from the point to the obstacles. For example, the visibility polygon V(P) is the shaded area in Figure 3.1. The only difference between this area and the one in Figure 2.1 is that now we also have tangents from the point to an arc of circle. While the equation of the tangent in the case of convex polygons follows simply from the fact that the line passes through two known points, for generalized polygons this equation is a little more difficult to obtain. The corresponding equations, for this case, are derived in [7].

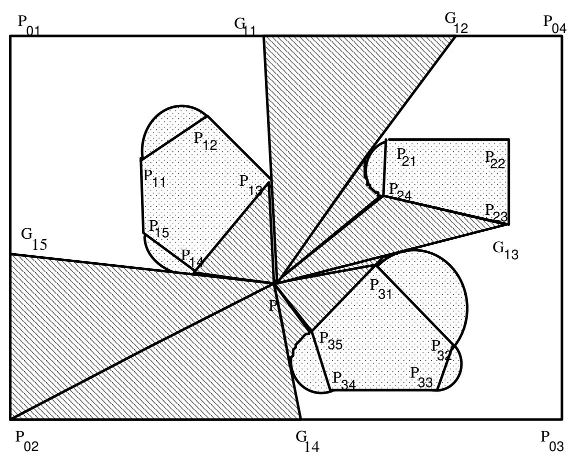

Fig.3.1 Visibility for generalized polygons.

The second visibility polygon $V_2(P)$ will be formed by the union of the visibility polygons $V(G_{11})$ , $V(G_{12})$ , $V(G_{13})$ , $V(G_{14})$ and $V(G_{15})$ for all generating points. As in the case of convex polygons, we need to consider the mutual tangents between the obstacles for the generation of the higher level visibility polygons. We continue this way until we cover the entire environment. In our example, the maximum number of links needed for $P$ is 3.

Theorem 2.1 from Section 2 is valid for generalized polygons provided that now an edge of the free space $E$ is either a straight segment or an arc of circle. Thus to complete the algorithm for visibility from a point, in this case we need to add an additional criteria for ending the algorithm. While the edges represented by straight segments can be treated analogously to the algorithm in Section 2, we need different considerations for the circular edges.

The complexity of the algorithm for visibility from a point is the same as the one for convex polygons because the only differences are in calculating the tangents, which takes constant time.

In building an ALC for convex polygons we used a property (Theorem 2.2 in [7]) that if we can reach all the vertices of the polygons with $k$ links than we need maximum $k+1$ links to reach all points in the free space. However this is not true in this case because hat the points on an arc of circle cannot be seen with a straight line from the arc's vertices. In [7] we have outlined a procedure of dealing with this difficulty where using the tangents to the arc at its vertices, we search for an area in free space that contains a point or a set of points that can see the whole arc with one link only. In the worst case we might need to subdivide the arc into sub-arcs for analysis, but overall because of the connectedness of the environment, at the end we will get $l$ points $D_i$, $(i=1,...,l)$ forming a set, from which all points of the arc $c$ are visible. Every point of the arc can be seen with one link from at least one of the points $D_i$. This procedure does not affect the overall complexity because it is done once in the beginning of the main algorithm.

As a criteria for "end", in the algorithm for the visibility from a point, for the edges that are arcs of circles we use Lemma 3.1 from [7]

Lemma 3.1 If at level $k$ all points $D_{ij}$ for every arc have been reached, than the next level is necessarily the last, i.e. the visibility is maximum $k+1$.

We can also use this criteria for building the visibility polygons for all vertices and all points $D_i$ in the environment and then intersecting the corresponding levels bottom-up until the first non-empty intersection. This will be an approximate link center for this environment. If we apply this procedure to our example, for all points $P_{ij}$ and $D_{ij}$ $(i=1,...,m;\ j=1,...,m_i)$ and intersect the corresponding triangles, the result is shown in Figure 3.2. The approximate link center on the figure is the unshaded portion of the free space. As we can see it is considerably smaller than the one in Figure 2.8.

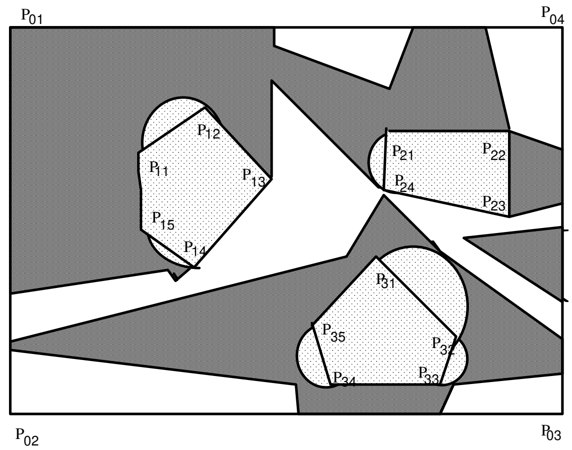


Fig.3.2 An ALC for generalized polygons.

To build the exact link center we need to build the visibility polygons from all edges in the free space. The visibility polygon of a circular arc is quite different however then the one for a line segment. In Figure 3.3 all points that can see the whole arc $Arc(P_{21}\ P_{24})$ with one link are shaded vertically.

The first characteristic of the shaded points is that they belong to the top part $B$ in Figure 3.3, which we will call "hat" $H_{ij}$ for every circular edge $e_{ij}$. In fact this hat is formed by intersection of the tangent lines $t_1$ and $t_2$ with the outside boundary and the vertices of the outside boundary between those intersections. In Figure 3.3 the hat is the polygon $D_{24}\ P'_{21}\ P_{12}\ P''_{24}\ P'_{24}$. The hat is non-empty if the vertex $D_{ij}$ is in free space and the circular triangle $P_{21}\ D_{24}\ P_{24}$ is also in free space. However if this hat is not completely in free space, we need to take just the part that is in free space and can see the whole edge with one link. Thus we can prove the following:

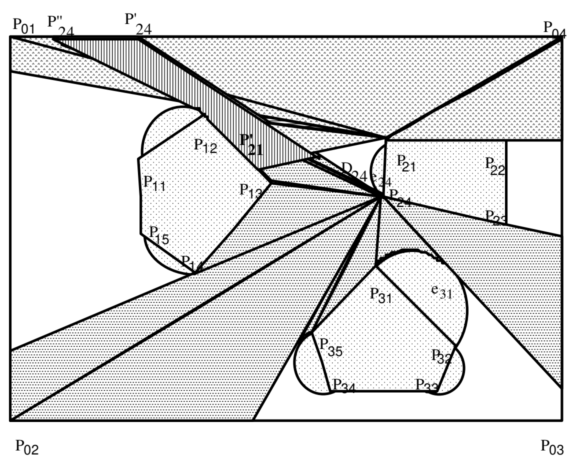


Fig.3.3 First Visibility region for an arc.

**Theorem 3.1** For every circular edge $c_i = Arc(P_iP_j)$ , $V(c_i)$ is the simply-connected component of the intersection $V(P_i) \cap V(P_j)$ that is contained in the non-empty hat $H(c_i)$.

We can build the shaded region on Figure 3.3 using the same algorithm as the one described in Section 2.2. The only difference is that now some of the edges of the resulting region are arcs of circles and instead of starting with the vertices $P_i$ and $P_j$ , we start and end with the point $D_i$ . If an arc has an empty hat (see arc $e_{31}$ in Figure 3.3), then there are no points that can see the whole arc with one link. In that case we can find the points that see the whole arc with two links by subdividing the arc into two (four, etc.) sub-arcs considering the intersection of the visibility polygons of the hats of the sub-arcs. This procedure is finite because there is no intersection between the obstacles and the outside environment, and every circular arc can be infinitely closely approximated with a finite polygonal chain.

For every circular edge after we find the first visibility region (in the case of non-empty hat) or the second visibility region when the first one is empty, we can build the rest of the visibility regions by using the X-forms introduced in Section 2.2. We have to add some conditions that reflect the specifics of the circular edges, explained in detail in [7].

Figure 3.4 illustrates the different visibility regions for edge $e_{24}$. In this the unshaded area represents $V_2(e_{24})$ .

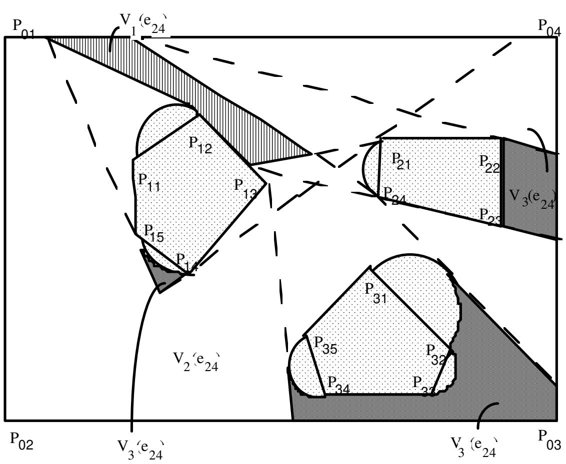


Fig.3.4 Visibility regions for circular edge.

After we build the visibility polygons for all edges (straight and circular), we start intersecting them bottom-up from level two, until we obtain a non-empty intersection. For our example the link radius is again two, but this time the link center, shown as the unshaded portion of the free space in Figure 3.5, is significantly smaller than both the one for straight edges in Figure 2.8 and the ALC for generalized polygons in Figure 3.3.

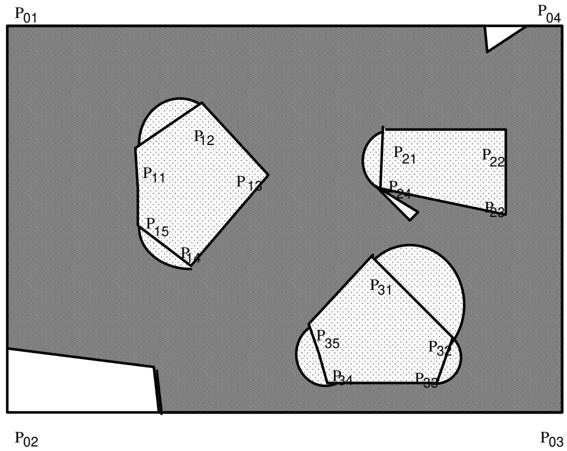


Fig.3.5 Exact link center for circular edges.

So far, in this Section, we have discussed visibility and link properties for generalized polygons. We emphasize that the procedure, described above, is valid for <u>any</u> set of convex obstacles. We never really actively used the fact that the edges are arcs of circles, as opposed to arcs of ellipses for example. All considerations about X-forms and visibility can be applied for any convex curved edges, because we can always draw and reason about the tangents. In addition for every convex obstacle we can use some reasonable heuristics to subdivide it in parts that are very closely approximated with straight lines and curved segments. Of course we need to trade-off between the closeness of the approximation and the number of edges that are introduced, because we want the program to run in reasonable time.

Thus using the considerations above and the detailed formulae in [7] we can conclude the description of our algorithm for <u>general convex</u> obstacles and environment. In the next section we will outline a method for dealing with non-convex two-dimensional objects.

### 3.2 Some Results for Non-Convex Obstacles

We start with non-convex polygons and we will point out the necessary modifications to the previous properties to accommodate this more general case. After that, using the results from Section 3.1, we will extend our treatment to general non-convex obstacles.

We will consider three general ways to deal with non-convex polygons:

- We can decompose them into convex components and use the properties of convex polygons to make conclusions for the non-convex ones.
- We can take the convex hulls of the polygons, solve the problem for them and then consider separately the concavities of the obstacles.
- We can try to deal directly with them, by modifying the actions of the algorithms

**for convex polygons, depending on whether we deal with convex or reflex vertices or edges.**

**The first method of decomposing the non-convex objects is widely used in the literature. We are looking for a subdivision where the polygon is partitioned into the minimum possible number of convex components.**

**There are two basic methods of minimum decomposition in convex polygons: 1) If we do not want to add new vertices to the partitioning, we can use the algorithm in [6] for decomposing a simple polygon with n sides into minimum number of convex pieces in time $O(n^3 . \log\ n)$. 2) If we do add additional vertices in the decomposition, we can sometimes achieve even a smaller number of convex components. Those additional vertices are called "Steiner" points, and if we allow them, we can use the algorithm in [18] for partitioning into the minimum number of convex pieces, which requires $O(n^3)$ time. After the partition we can use all the algorithms described so far to build visibility polygons and link center for this convex pieces.**

**In the second method we deal with non-convex polygons by first building the convex hulls of the obstacles and applying our method for them directly. Then we apply any of the algorithms mentioned in Section 2.1 for visibility and link properties for the concavities in the original obstacles, i.e. the differences between the convex hulls and the obstacles.**

**Both methods above cause more computations than if we deal with the non-convex obstacles directly - the third method described in [7].**

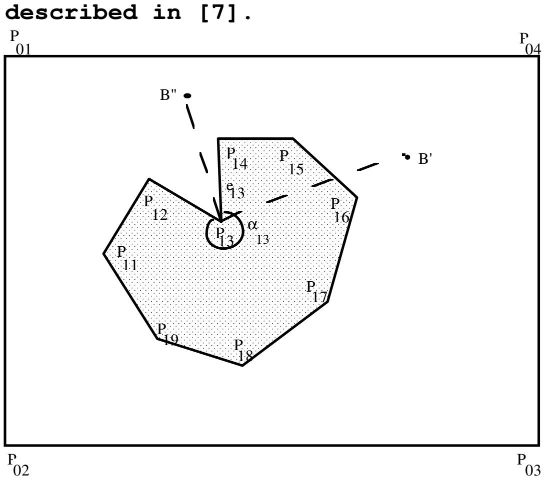


**Fig.3.6 'Inside visible' or 'not visible' reflex vertices.**

**In terms of the algorithm in Section 2.1 for visibility polygons from a point, the presence of reflex vertices does not introduce new generating points. If we look in Figure 3.6 we will see that for every point $B$ in the free space, point $P_{13}$ is either "not visible" (e.g. point $B''$) or "inside visible" (e.g. point $B''$ ) from that point. Thus the reflex vertices just form visibility triangles on the current level and do not introduce new generating points on the next level.**

**Another important characteristic for non-convex polygons is that it is sufficient to be able to see the convex edges of the environment in order to see all points in $E$. The reasoning behind this is similar to the one in [15], where it is proven that the link center of a simple polygon $P$ is built by considering the visibility regions only for the convex vertices of $P$. In fact for our case Theorem 2.1 can be reformulated by substituting "<u>all convex</u>" edges in $E$ instead of "<u>all</u>" edges in $E$. In that way the criteria for the end at level $k$ is to be able to reach the lines of the convex edges in free space at level $k - 1$ . The validity of these statements comes from the fact that a reflex edge lies in the halfplane of the neighboring edge that points to the outside of the polygon. That is why a reflex edge can be seen almost always from the first preceding convex vertex in the chain of vertices. If another obstacle is obstructing this view, the convex vertices of the obstructing obstacle will be able to see those reflex edges of the original obstacle.**

**These situations are illustrated and analyzed in detail in [7]. As it is shown there and from the discussion so far it is clear that the results in Section 2 for visibility polygons from a point are also valid for non-convex polygonal obstacles. The complexity of the algorithm here is the same as in Section 2.1, but in this case we can spend even less time because of the cutting-off of some of the checks for reflex vertices.**

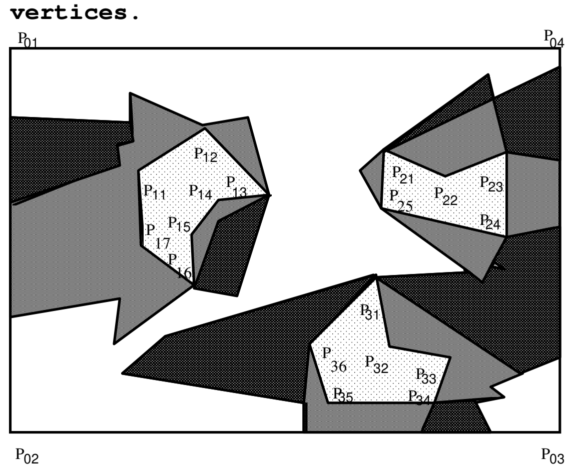


**Fig.3.7 Link center for non-convex obstacles.**

**The approximate link center can be built analogously to the one in Section 2.2, but now again we need to consider only <u>convex</u> vertices, rather then all vertices in the environment. The same is true for the exact link center, where we will build only the visibility polygons of all convex edges in $E$. The X-forms and the higher level visibility polygons of an edge are treated the same way.**

**In Figure 3.7 the unshaded area is the exact link center for this environment,**

while the unshaded area and the darkly shaded one constitute the ALC for the vertices and midpoints of the edges. If we compare the two we can see that additional reflex edges do not influence the exact link center. That is, if we have found the exact link center for the convex hulls of the obstacles and an ALC for the vertices of the non-convex obstacles, the exact link center for the non-convex obstacles is simply the intersection of the afore-mentioned link centers.

If we are dealing with general non-convex objects, not necessarily polygons, we can use the method, developed in Section 3.1. We approximate as close as possible the obstacles with "generalized non-convex polygons". (In fact the term "generalized polygons" refers to general simple polygons, not necessarily convex ones.) Now that we have generalized polygons, we can process each of the curved edges in the same manner as in Section 3.1. We point out that in this case we need to consider both the convex and the reflex curved edges. We illustrate the general algorithm for an approximate link center and the exact link center in an environment with non-convex obstacles in Figure 3.8. These obstacles are actually a combination of the obstacles in Figure 3.5 and Figure 3.7. Ss we can see the exact link center and the ALC here are very close to the link centers from Figure 3.5 and Figure 3.2.

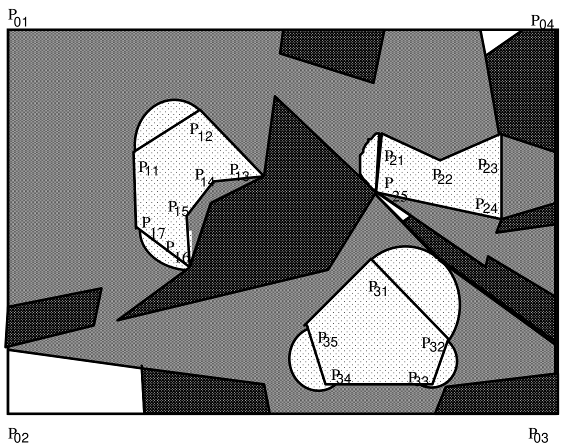

Fig.3.8 Link centers for general obstacles.

This completes our discussion of the two-dimensional case of an environment with obstacles. In the next section we will analyze the three-dimensional case.

### 3.3 3D Obstacles and Environment

As we have shown in this section so far, the transition from convex polygonal objects to general convex ones and then to general non-convex objects, requires only some additions and modifications to our program in Section 2. However the step from a two-dimensional to three-dimensional environment is not that obvious and easy to realize. In this Section we will discuss first an approximation to the real 3D case, which can be discussed conveniently using the 2D model we have developed. Next we will point out an approach for dealing with the real 3D case.

First of all , we need to determine the structure of our robot in 3D environments. It is clear that we cannot use only telescoping links because they are two-dimensional. One easy generalization of the existing structure is to add a prismatic joint between the manipulator's base and the ground in such a way that the translation action raises and lowers the entire planar manipulator structure normal to its original plane of action. In this way we will be able to reach any point in the free 3D environment by reaching first the horizontal plane it lies in and then reach the point itself with the telescoping links. This, of course, is not a general 3D solution, because we might be able to reach a point faster moving diagonally between horizontal planes. However this $2^{1/2}$D model of the environment is very convenient for direct implementation of our algorithm from Section 2, and we can use it to find an appropriate ALC for the 3D environment. That is why, we will concentrate first on this model.

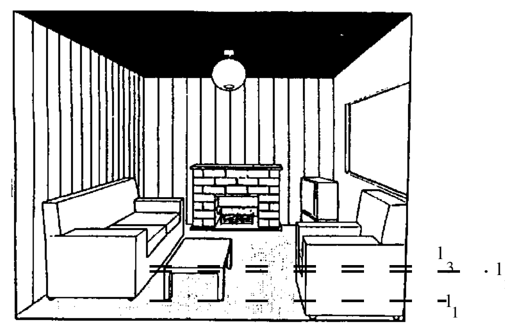

Fig.3.9 Typical 3D environment.

We will call each horizontal plane, on which the robot is working, a "horizontal layer" or "horizontal slice" of the environment. In a typical office or plant environment, the 3D objects usually have considerably uniform vertical structure. For example in Figure 3.9 we have a sofa, a coffee table, an easy chair , a TV and a fireplace. We can describe the coffee table in $2^{1/2}$ D with three horizontal slices: one at the bottom of the legs, one at the transition from legs to surface, and one at the top of the surface of the table. Those slices are represented by the lines $l_1$ , $l_2$ and $l_3$ respectively. Similarly we can find few horizontal slices that will define the other objects in the environment. If we take all slices for all the objects we will have a number of horizontal slices, say $k$ , that will model the whole environment and its obstacles. Every slice at a height $h$ is a 2D environment with obstacles, representing the horizontal section of the original obstacles at height $h$. For every slice we can build the link center for that environment. For example if we take the layer, corresponding to the height $h_t$ of the top of the coffee table, the link center is depicted in Figure 3.10.

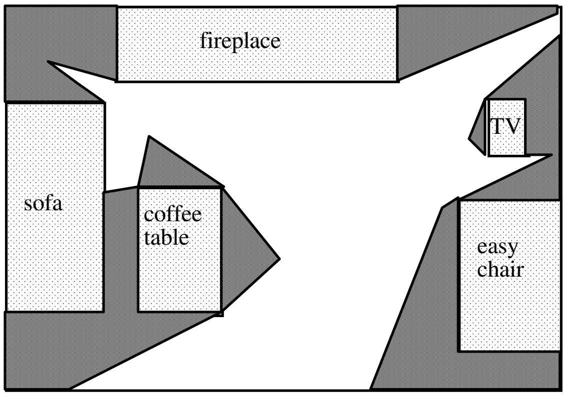

Fig.3.10 Link center for a horizontal slice.

We can apply the same procedure for every layer and find its corresponding link center. At the end we project all these link centers, that correspond to the same link radius, on one horizontal plane, and the intersection of the link centers for all layers will define a link center for the entire environment. The $2^{1/2}$ D link center will be the cylinder (or cylinders) with axial cross section the link center formed by the above intersection. The cylinder height is the same as the maximum height of the environment. This means that, if we put the fixed base of the first vertical prismatic link of the robot in any point of the base-plane link center, any point in the free space of the environment can be reached with our robot with a "sub-optimal" number of links. We say "sub-optimal", because we approximate the objects as being quite uniform vertically.

In [7] we discuss in detail this algorithm as well as the questions of: how do we choose the heights of the slices for each object, how many slices are going to be used for the overall environment with obstacles, how do we find the intersection of the link centers.

Sometimes, instead of using a large number of horizontal slices to closely model the shape of the obstacles, we can use a combination of horizontal and vertical slices. For every object we can take a few characteristic top-to-bottom, left-to-right and front-to-back slices. Of course the top-to-bottom slices are most important, because they reflect the structure of our robot. However, the vertical slices allow us to more easily discover holes or concavities in the objects, i.e. we can use them to determine the shape of the obstacles.

If we want to use the vertical slices for link center construction, we need to be able to have telescoping links in the vertical direction. This brings us to the problem of determining the best structure for our robot for real 3D obstacles. We need to generalize the telescoping structure in three dimensions. The reachable workspace for a telescoping link is a disk with center at the base of the link and radius equal to the maximum length of the link. In 3D we want the reachable space for the link to be a sphere. To achieve this in practice, we can define a RRP (revolute-revolute-prismatic) link, and call it a telescoping link in 3D. The joint's two revolute degrees of freedom are at the base joint, and the one prismatic stretches in and out like a telescope. This structure with the outbound of its reachable workspace is depicted in Figure 3.11, where point $O$ is the base of the link and point $P$ is its end-point. The telescoping link is stretched at distance $r$. The horizontal plane is the $O_{yz}$ plane and the projection of the link $OP$ on the horizontal plane is $OP_1$. The spherical coordinates $\alpha$, $\beta$ and $r$ uniquely determine the position of point $P$ with respect to point $O$.

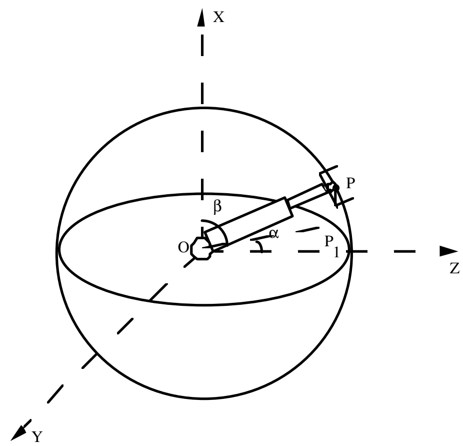

Fig.3.11 A 3D telescoping link.

Using those links we can generalize the procedure for building the visibility polygons described so far. We can draw tangents from a point to a 3D figure and they will correspond to reaching from the point with a 3D telescoping link. Of course now there can be an infinite number of tangents from a point to an obstacle. these are the generating lines of the cone with vertex the given point and base the element of the 3D obstacle. In fact the mutual tangents between two polyhedral obstacles can also define a whole plane if the corresponding edges are intersecting lines (see [7]).

The actual implementation of the visibility algorithm in 3D requires a lot of work and depends strongly on the method of representation of the 3D objects in the algorithm. CAD software packages have been built for ray tracing of 3D objects. In [7] we have outlined few ideas for extending of the algorithms, described so far in this paper, to 3D environments. Those include determining visibility from a point using the normal vectors to the faces of the obstacles (with silhouette curves and sweeping lines), using breaks of continuity in the ray tracing for higher level visibility regions, and using visibility from a face for the link center. Some consideration is also given to non-convex and non-polyhedral bodies.

So far we have presented algorithms that build geometrically different sets of points related to the visibility and reachability of points in environments cluttered with obstacles. In the next Section we will introduce a complementary problem - the one of fast mathematical determination or estimation of the minimum necessary number of links to cover the whole free space of a given environment.

## 4. Best Estimates of the Number of Links

In this Section we want to derive formulas that estimate the minimum number of links for $E$ in terms of the number of obstacles and the number of vertices for each obstacle. To do this we need to find characteristic points in the environment that can see large parts of it and then estimate the number of links needed to move between those points. This approach is closely related to the problem of guard placement in an art gallery.

### 4.1 The Planar Case

As described in [19], the art gallery is a simple polygon, with or without holes, and we want to find how many guards are needed and where they should be placed so that they can see the entire inside of the gallery. In Figure 4.1 the vertices of the gallery are $P_1, P_2, ..., P_{10}$.

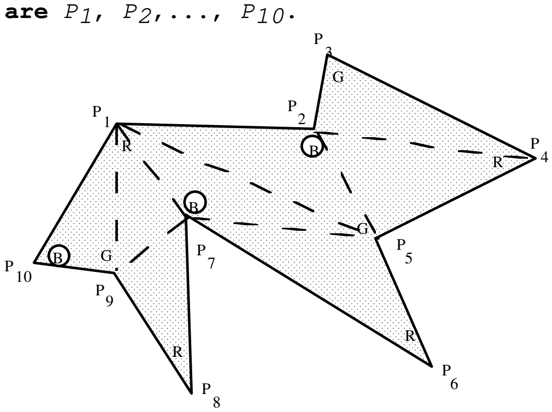


**Fig.4.1 Triangulation and three-coloring of an object.**

The basic approach taken in [19] is first to triangulate the free space $E$, i.e. to connect all vertices of $E$ with straight edges, so that they form triangles that cover $E$. Then the vertices are three-colored in such a way that in every triangle the three different vertices have different colors. (It can be shown that there always exist triangulations for which three-coloring is possible.) Finally the guards are placed at those vertices, having the color that appears the least number of times.

If we apply this procedure to the example in Figure 4.1, where $R$, $B$ and $G$ denote red, blue and green respectively, we obtain three green, three blue and four red vertices. Consequently we can place the guards at either the blue or the green vertices. In this case we choose the blue vertices. If we place three guards at vertices $P_2$, $P_7$ and $P_{10}$, they can see the entire inside of the polygon. The guard at $P_2$ sees triangles $P_2P_3P_4$, $P_2P_4P_5$ and $P_2P_5P_1$. The guard at $P_7$ sees triangles $P_7P_1P_5$, $P_7P_5P_6$, $P_7P_8P_9$ and $P_7P_9P_1$. Finally the guard at $P_{10}$ sees triangle $P_{10}P_1P_3$. The union of all these triangles covers the polygon completely.

Using this procedure we can estimate the number of guards needed to cover the whole polygon. This follows from the basic theorem in [19], the Chvatal's Art Gallery Theorem, which states that $[n/3]$ guards are occasionally necessary and always sufficient to see the entire interior of a polygon of $n$ edges. Here the italic square brackets, $[?]$ denote the largest integer less than or equal to $x$.

The term "polygon with holes" in [19] corresponds to our term "environment with obstacles". as we have shown in [9] and [7], a polygon with holes can be triangulated and the triangulation will have $n+2m-2$ triangles.

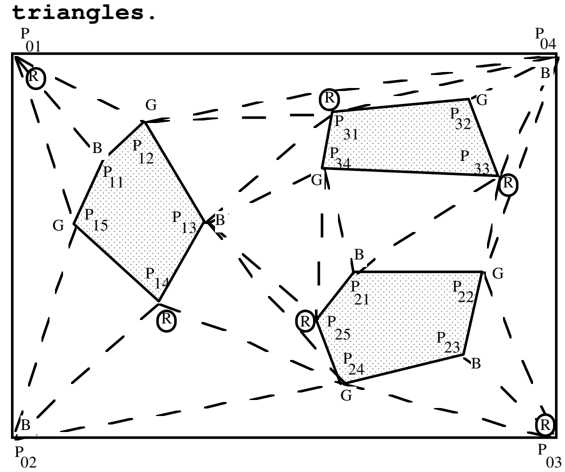


**Fig.4.2 Guards placement in the environment.**

In Figure 4.2 we illustrate an environment with 3 obstacles and 18 vertices overall. It is triangulated in 22 triangles. After three-coloring we have chosen the 6 red vertices and placed guards there.

The main result for polygons with holes, proved in [19], is Theorem 5.1 that states that $[(n+2h)/3]$ vertex guards are sufficient to cover the whole free space $E$. However it is clear that this is not a necessary condition because for example everything that the guard at $P_{25}$ can see, can also be seen by the guard at $P_{14}$, thus the guard at $P_{25}$ is not needed. Conjecture 5.1 in [19] states that $[(n+h)/3]$ vertex guards will be actually sufficient, however this has not been proven in general.

In what follows, we will connect our visibility of the environment and the minimum number of links covering an environment with obstacles to the guard placement theory. We will recall from Section 1 that the minimal visibility, VMIN, denotes the minimum number of links needed to cover the whole free space E. *VMAX* (maximal visibility) is the minimum number of links that are needed so that every point

**in $E$ can see every other point in $E$. We remember that all points in the link center have visibility of the environment equal to $VMIN$ and naturally always $VMIN \leq VMAX$.**

**In [9] and [7] we have proven a theorem that establish the connection we are looking for:**

**Theorem 4.1 In the worst case, $VMIN$ of the free space $E$ is equal to the number of guards covering $E$.**

**The proof of this theorem is based on a graph in which the vertices represent the guards in the gallery and an edge between two vertices indicates a relation of neighborhood. A guard is a neighbor with another one if it is a vertex in a triangle that has a side common to a triangle with a vertex at the other guard. The graph for Figure 4.2 is shown in Figure 4.3.**

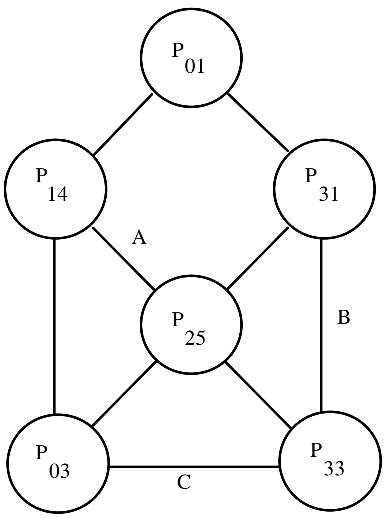


**Fig.4.3 The graph of neighboring guards.**

**The longest possible chain of edges is one that connects all the guards. We can think of it as one big obstacle. For example if the edges $A$, $B$ and $C$ in Figure 4.3 did not exist, we would have had one long chain visiting all the guards. In [7] we actually show an example, where the distance between two guards is $s-1$, where $s$ is the number of guards. Thus we can show that there is always a guard $Q$ in this conectivity graph that can reach all the rest of the guards with maximum $[(s-1)/2]$ edges. Finally every edge in the graph actually represents maximum two necessary links and thus all points in $E$ can be seen from guard $Q$ with a maximum of $2*[(s-1)/2] + 1 = s$ links.**

**We can also use similar arguments as above to prove that $VMAX = 2s$.**

**From Theorem 4.1 it follows that all results that are proven in [19] for polygons with holes can be directly transformed to results for $VMIN$. In particular we can prove:**

**Theorem 4.2 For planar environments $VMAX \leq 2*[(n+2m)/3]$ and $VMIN \leq [(n+2m)/3]$.**

**It is clear that, for a given number of obstacles $m$, if we have smaller $n$ we will need less links to cover $E$. Consequently if we can enclose the obstacles in polygons with less edges that can be placed in $E$ without overlapping, we will get better results. This is the basis for yet another approach extensively discussed in [9] and [7]. In that we first enclose all obstacles in minimal bounding triangles. We use the fact that if we can reach all vertices of the triangles with $k$ links, then we can reach all edges of the original convex obstacles with at most $k + 1$ links (Lemma 4.1 in [7]). Then by using graphical and algorithmic arguments, we show that:**

**– for a triangle in a convex environment we have $VMAX = 2$ and $VMIN = 1$;**

**– for two triangles $VMAX \leq 3$, $VMIN = 2$;**

**– for three triangles $VMAX \leq 4$, $VMIN = 2$;**

**By induction we can prove:**

**Theorem 4.3 In an environment with $m$ triangular obstacles we can reach all the vertices of the obstacles with $VMAX \leq m+1$ and $VMIN \leq [(m + 1)/2]+1$ links.**

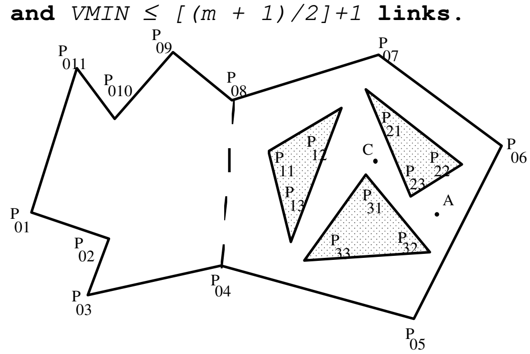


**Fig.4.4 A non-convex outside boundary.**

**If we want to include the case when the outside boundary is general convex or non-convex curve, we can formulate:**

**Theorem 4.4 For a general outside boundary shape we can reach all the vertices of the $m$ triangular obstacles with:**

$VMAX \leq m+1+2*[(n-3m-1)/3]$ **and**

$VMIN \leq min(max(m+1,2[(n-3m-1)/3]),[(4n-9m+5)/6])$

**The detailed proof of this theorem can be found in [7]. It is based on analysis of the worst case situation where the environment can be subdivided into a big non-convex part $S$ with almost no obstacles and another part with obstacles (see Figure 4.4). The part without obstacles can be treated as a general polygon with $p$ vertices and no holes. Such polygon can be covered by $[p/3]$ guards (see [19]). For the part with holes we can use Theorem 4.3.**

**Sometimes triangular enclosures in 2D take a lot of space and it might happen that the resulting triangles intersect each other or the outside boundary. In that case we can**

**try to use tighter enclosures with rectangles. Following exactly the same kind of reasoning as above we can show that Theorem 4.3 and Theorem 4.4 are also valid for rectangular obstacles.**

**If we cannot enclose the obstacles in rectangles, we can still apply this approach to derive upper bounds for** $VMIN$ **and** $VMAX$ **. If** $m_i = 2*k_i + 1$ **or** $m_i = 2*k_i + 2$ **for** $i = 1,2,...,m$ **(here** $m_i$ **is the number of vertices of obstacle** $i$**) then we can use the following property: To go around a polygon with** $n$ **vertices we need at most** $[n/2]$ **links. If we start with any vertex of the polygon, it can see two other vertices on the first level (that is with one link). These vertices can see two other vertices on the next level, and so on until we finish all the vertices. Thus we can show that for a convex outside boundary:**

**Theorem 4.5** $VMAX \leq k_1 + k_2 + ... + k_m + 1$ ,
$VMIN \leq [( k_1 + k_2 + ... + k_m + 1)/2] + 1$ **to see all the vertices.**

**In the general case we can always decompose the environment into sub convex parts with obstacles and non-convex parts with no obstacles by using the vertices that are the closest to the non-convex parts. Let us say there are** $s$ **non-convex parts with no obstacles, and** $l$ **separate sub convex parts with obstacles. In Figure 4** $s=1$ **and** $l=1$ **. Then for each sub convex part with obstacles we need** $[n_j/3]$ **links for** $j = 1,2,...,l$ $(l \leq h)$ **and for each non-convex part** $i$ **we need a maximum of** $[n_i/2]$ $(i = 1,2,...,s,\ s$ ³ $0)$ **links to cover it. Using Theorems 4.4 and 4.5 we can prove:**

**Theorem 4.6 For a general shape of the outside boundary and the obstacles:**

$$VMAX \leq \sum_{j=1}^{l}[n_j/3] + \sum_{i=1}^{s}[n_i/2] + 1 \quad \textbf{and}$$

$$VMIN \leq [( \sum_{j=1}^{l}[n_j/3] + \sum_{i=1}^{s}[n_i/2] + 1)/2] + 1$$

.

**We add one link at the end to cover the unreached edges between the last covered vertices. If we approximate above the expression for** $VMIN$ **we get an estimate** $[(n + 6)/4]$ **which is better than** $[(n + m)/3]$ **for every** $m$ ³ $2$ **.**

**In the next section we will discuss the extension of this analysis to 3D environments.**

### 4.2 The Spatial Case

**The 3D equivalent to triangulation is tetrahedralization. However as opposed to the 2D case, [19] shows that for the same polyhedron we can have tetrahedralizations with different numbers of tetrahedra. Moreover there exist non-tetrahedralizable polyhedra. In fact deciding whether or not a polyhedron can be tetrahedralized is NP-complete. Consequently we cannot readily use tetrahedralization to estimate the number of links that we need in three-dimensional environments.**

**However to obtain some estimates on the number of links needed to cover a 3D free space, we can use Theorem 10.1 in [19] which states that** $[(2*F-4)/3]$ **vertex guards are sometimes necessary and always sufficient to see the exterior of a convex polyhedron of** $F$ **faces for** $F$ ³ $10$ **. The necessity holds for** $F$ ³ $5$ **, and it is conjectured in [19] that the sufficiency also holds for that range.**

**Using the approach from Section 4.1 we get:**

**Theorem 4.7 In three dimensional space**

$$VMAX \leq 2*[(2* \sum_{i=1}^{m}F_i - 4m /3] \quad \textbf{and}$$

$$VMIN \leq [(2* \sum_{i=1}^{m}F_i - 4m)/3]$$

**, where** $F_i$ **is the number of faces on the** $i$ **-th obstacle.**

**To derive another estimate of the number of links necessary to cover the free space in 3D, we can use the approach in [2]. There it is proven that using Steiner points as well as the vertices of the obstacles, we can construct an algorithm that decomposes a polytope (in general in 3D a polytope is a bounded polyhedra) with** $n$ **vertices and** $r$ **reflex edges into** $O(n + r^2)$ **tetrahedra in time** $O(n*r + r^2*log(r))$.

**Let us denote with** $v_i$ **and** $r_i$ **,** $(i=0,1,...,m)$ **the number of vertices and reflex edges of obstacle** $i$ **(**$0$ **stands for the outside boundary). Let us connect the outside boundary along a convex edge with the first obstacle (randomly chosen) along one of its convex edges. Then we connect in the same way the first obstacle to a neighboring one, etc. until we connect the one before the last obstacle to the last one. Then the connected figure will have** $N = \sum_{i=0}^{m}v_i + 4m$ **vertices and maximum** $R = \sum_{i=0}^{m}r_i + 2m-1$ **reflex edges (if all the connections between the obstacles, except the one to the outside boundary, create reflex edges).**

**For example consider two cubes** $CB_1$ **and** $CB_2$ **inside a parallelepiped** $P$**, Figure 4.5. We connect the cubes with a plane through the edges** $P_{13}P_{17}$ **and** $P_{24}\ P_{28}$ **, and cut them along these edges. In the same way we connect the cube** $CB_1$ **to the parallelepiped** $P$ **along the edges** $P_{02}\ P_{06}$ **and** $P_{11}\ P_{15}$ **. The resulting figure is a polytope. Each of the cubes and the parallelepiped had eight vertices, twelve edges and six faces. The polytope has** $8+8+8+4*2=32$ **vertices, because at every cut we introduce** $4$ **new vertices**

and we have $m=2$ cuts. Out of them there are $0 + 0 + 0 + 3 = 3$ reflex edges.

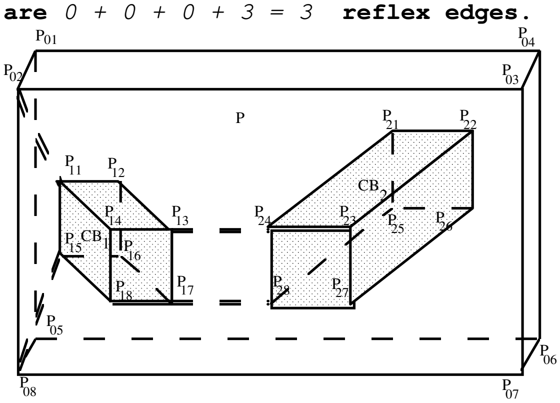


Fig.4.5 Decomposition of a 3D environment.

Using the consideration above and four-coloring of the vertices we can prove:

**Theorem 4.8** For a polytope with polygonal obstacles:

$$VMAX \leq 2[(O((\sum_{i=0}^{m} v_i +4m)+(\sum_{i=0}^{m} r_i +2m-1)^2)+3)/4],$$

$$VMIN \leq [(O((\sum_{i=0}^{m} v_i +4m)+(\sum_{i=0}^{m} r_i +2m-1)^2)+3)/4],$$

where $v_i$ and $r_i$ are the number of vertices and reflex edges respectively of the obstacle $i$.

In analogy with our discussion of the 2D case, we can envision future analysis of general 3D obstacles with straight edges and faces by approximating them with triangular wedges and parallelepipeds. In the next section we propose some heuristic procedures to lower, even more, the estimates we derived in section 4.1 to the optimal values for $VMAX$ and $VMIN$.

### 4.3 Optimizing the Number of Links

The estimates that we derived in section 4.1 are only upper bounds and in some cases they can be much higher than the actual maximums. For example for the environment in Figure 4.2, if we apply our results we obtain $VMAX \leq 7$, $VMIN \leq 4$. However using the algorithm for the link center from Section 2 we find that the actual values are twice smaller: $VMAX = 3$ and $VMIN = 2$. We can not always find the optimal numbers exactly, because that would be equivalent to finding the optimal number of guards in an art gallery, which was proven to be NP-hard in [20]. However we describe in [7] and [8] an algorithm that can find a sub optimal solution which in the majority of the cases is equal to the optimal one.

This algorithm uses triangulation of the environment to find the minimal visibility $k$ for all edges. Every edge is a root of a tree with the vertices of depth 1 being the ones that are nodes of the triangles including this edge in all possible triangulations. For the example in Figure 4.2, edge $P_{25}P_{21}$ has in its tree vertices $P_{21}$, $P_{25}$, $P_{02}$, $P_{14}$, $P_{13}$, $P_{12}$ and $P_{34}$. The next level are the vertices that can see the vertices of the previous level with one link, and so on until we include all the vertices in the environment. In our example the only vertex on level 3 is $P_{23}$. However if we denote with $A$ the intersection point of the lines containing edge $P_{21}P_{25}$ and edge $P_{23}P_{24}$, then $A$ can see the root of our tree with one link, and it can see point $P_{23}$ with one link. Thus the actual depth of point $P_{23}$ is two, and the optimal number of links, $p_1$, that are needed to see the whole free space from edge $P_{21}P_{25}$ is two. After applying this relatively fast procedure for the other edges, we can find $p_2$, $p_3$, ..., $p_h$. Then it is clear that $VMIN = min(p_1, p_2, ..., p_h)$ and $VMAX = max(p_1, p_2, ..., p_h)$. In our example as we mentioned before $VMIN = 2$ and $VMAX = 3$.

Algorithmically this procedure can be realized using a module for determining whether an edge is visible from a given point with one link. We can optimize the running time by arranging all vertices and edges in $E$ in a "visibility table" that can be processed in parallel to find the overall number of links. The method for building and analyzing this table is explained in detail in [7]. In addition the table allows for look-up estimates for the values of $VMIN$ and $VMAX$. Although this procedure is faster than the method in Section 2, it is not a constructive one. We don't know exactly where the link center is, we can just say which vertices lie in it. For an arbitrary point (different than the vertices) we cannot say what is its visibility and how many links are needed to reach some particular other point. Thus each of the methods has its advantages and serves its purpose.

The reason the approach in Section 4 does not give us optimal values is that we have restricted ourselves to vertex guards that are stationary points. With the telescoping links we are moving from region to region. Our visibility reflects the optimal solution for this environment, while the number of guards is just a solution for the art gallery problem. In the following algorithm we will remove the restriction on the guards to stay on the vertices of the environment, which will hopefully allow us to find the optimal number of guards for most cases. As before we first triangulate the environment, tricolor the vertices, choose the color with minimal number of occurrences and place the guards there.

In several instances the triangles that are completely visible by one vertex guard belong to another guard's patrol area. For example in Figure 4.2 triangles $P_{13}P_{14}P_{24}$ and $P_{13}P_{24}P_{25}$ are both visible to guards

$P_{14}$ and $P_{25}$. It is clear that if all the triangles visible to one guard are also visible to another guard, then this second guard is unnecessary. In our example everything visible to guard $P_{25}$ is visible to $P_{14}$. Similarly, everything visible to guard $P_{31}$ is visible to $P_{01}$ or $P_{33}$. Consequently one of the guards $P_{31}$ and $P_{25}$ (say $P_{25}$ ) can be eliminated, and we still have complete cover of $E$, now with only five vertex guards.

Now the next step is to introduce non-vertex guards (Steiner points). Consider the guard at $P_{33}$. In the area covered by it there are two patches that cannot be seen by the rest of the guards. One of these patches is based on the edge $P_{21}P_{22}$ , and is depicted on Figure 4.6 as the shaded area *1*.

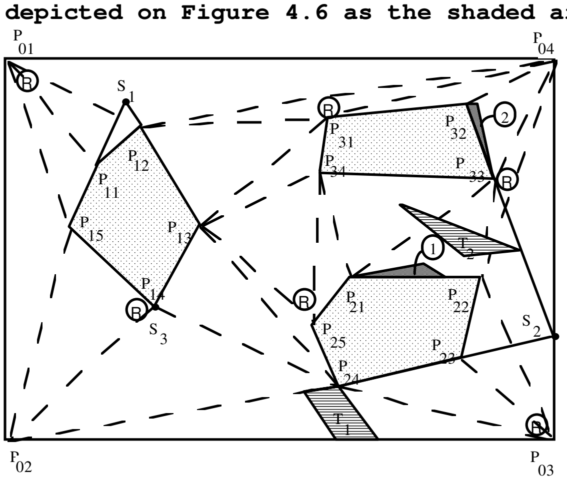


Fig.4.6 Optimal number of guards.

If we move the guard at $P_{01}$ to a position from which it can see $P_{12}P_{13}$, the guards at $P_{33}$ and $P_{31}$ will be redundant from this side. The new guard should also be able to see $P_{11}P_{15}$, consequently we can place it at the intersection of the lines $P_{15}P_{11}$ and $P_{13}P_{12}$ . If this point is in $E$ , which in our case is conveniently true, this gives point $S_1$. Similarly, the other area that is seen by $P_{33}$, but not by the other guards, is the shaded area *2* based on the edge $P_{32}$ $P_{33}$ . Therefore we can move the guard in $P_{03}$ to point $S_2$, the intersection of the lines $P_{24}P_{23}$ and $P_{32}P_{33}$, which is also inside $E$ . Finally with guards at $S_1$, $S_2$ and $S_3=P_{14}$ we can cover the whole environment $E$ . This is the optimal number of guards because, by using arguments similar to those above we can show that it is not possible to reduce the number of guards by moving them around. Detailed description of this algorithm is given in [7].

We would like to point out that although we found the minimum possible number of guards for this environment, the visibility here is actually even smaller - it is two. In fact every point that can see all the guards with only one link will be able to see the entire free space with a maximum of two links. In Figure 4.6 those points form the horizontally shaded areas $T_1$ and $T_2$ . These areas are actually part of the link center for this environment, as we can see in Figure 2.8 from Section 2.2. Of course they do not define the entire link center, because there might be other possible triples of guards covering the entire free space. This is still a fast way to get parts of the link center, provided we have managed to optimize the number of guards.

So far we have dealt with the situation when the environment is fixed and we are designing a stationary robot to operate in it. In the next chapter we present some extensions that look at our basic problem from a different point of view.

## 5. Extensions of the Basic Design Problem

There are several directions in which we can extend our analysis. Those including allowing the robot and/or obstacles to change their shape and/or position in the environment. The case of moving robots and obstacles as well as reconfigurable robots is discussed in [7] We will briefly mention here that all estimates in Section 4 are valid for both moving and stationary obstacles. We can look at the problem of moving obstacles with a fixed robot, as dual to the problem of fixed obstacles and a moving robot, because all that matters is the relative position of the obstacles and the robot. For a complicated environment with a large number of obstacles we may have a large link radius. That means that if we place the robot in a point of the link center, we will need a large number of telescoping links to reach all the points in the environment. From a practical point of view such a robot may be difficult to build because of the masses of the links, the actuator mechanisms, controls and so on. In that case we can introduce limited mobility for more manageable structure of the robot. Alternatively we can use multiple robots operating in sub-spaces of the environment. Those robots can have reconfigurable structure (which is easy to achieve with our telescoping links) and can cooperate at their common boundaries.

In this paper we will concentrate on the situation that arises when we try to design both the robot and the environment simultaneously. Let us say that we have a set of objects (e.g. machines) that we want to place in an environment (e.g. a factory floor) that is going to be completely automated with robots. In this case we might want to take advantage of the fact that we know the basic structural elements of the robots (e.g. telescoping links) and place the objects accordingly.

To start with, the objects in the environment can have different shapes and dimensions. It is difficult to reason directly about the general case of arbitrary obstacles and in our formulation the exact initial position of the objects is not that important. That is why we will outline here an approach that uses approximations of the obstacles with rectangles, in 2D (or parallelepipeds in 3D), to find simultaneously the best design of the environment and the robot. The reason we are using rectangles is because they are easy to build and rectangular shapes are easier to place and analyze, especially if we think of the outside boundary of the environment as a rectangle. In fact we can use rotating calipers (see [23]) to find in linear time the smallest area rectangle enclosing given polygon.

If we assume that we have enclosed all the obstacles in rectangles and we have a rectangular bounding environment, the problem now is to place those rectangles in the environment in such a way that the robot design problem gives the best result overall. We will describe a method to place these rectangles, inside the boundary rectangle with sides parallel to each other.

We introduce the following notations: $R_i$ denotes the enclosing rectangle for the obstacle $OB_i$ , with $i = 1, 2, ..., m$. For every rectangle $R_i$ , $a_i$ and $b_i$ are the lengths of the sides, where $a_i \geq b_i$ . We renumber the obstacles so that they are in decreasing order with respect to $b_i$. $R_0$ is the rectangular approximation of the outside boundary, $a_0$ and $b_0$ are the lengths of its sides, and $a_0 \geq b_0$ . A simple condition that can be checked to verify that all enclosing rectangles can fit in the boundary rectangle, is ***A*** : $\sum_{i=1}^{m} a_i * b_i < a_0 * b_0$, i.e., the sum of the areas of the rectangular obstacles must be less than the area of the rectangular boundary. For the rest of this discussion we will refer to the enclosing rectangular obstacles as "obstacles" and to the rectangular boundary as "boundary". Even though condition ***A*** might be satisfied, we might not be able to place the obstacles inside the boundary without overlapping. We will assume that there is enough room for the obstacles. This assumption is a reasonable one, because in a realistic environment there is usually ample space for transportation and manipulation.

Let us first sort the sides $b_i$ of the rectangles in decreasing order. We try to place the rectangles in the environment in horizontal layers where each layer has obstacles with similar height (see Figure 5.1). In other words we take the obstacle with the largest height and place it in the lower left corner so that there is distance $\varepsilon$ from the bottom and the left side of the boundary. This distance is such that the robot can fit and reach around the obstacle. We will always leave this distance between the obstacles and between the layers also.

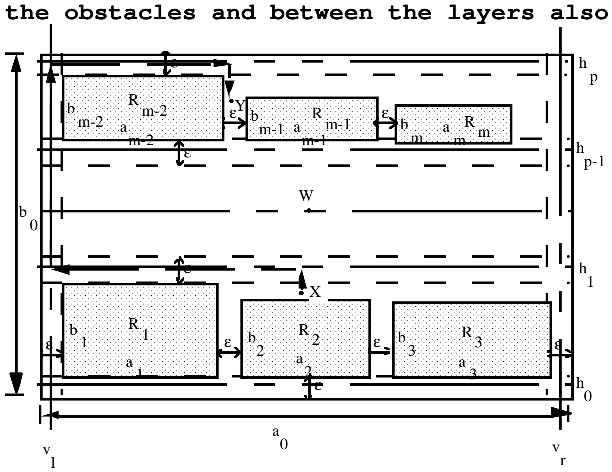


Fig.5.1 Placement of rectangular obstacles.

We place the next obstacle to the right of the first one on the same horizontal base at distance $\varepsilon$ from the first one. Similarly we place the next obstacles until we get to the right edge of the boundary. We make sure that there is at least $\varepsilon$ distance from the last obstacle in the layer to the right edge of the boundary. Then we continue with the next layer and so on, until we place all the obstacles. At the end if there is empty space above the last layer we can translate half of the layers up in order to free the center or we can space the layers more generously.

For each layer we will have a condition:

$$\textbf{H} : \quad \sum_{s=l_{i-1}+1}^{l_i} a_s \; + \; (l_i - l_{i-1} + 1) * \varepsilon \leq a_0$$

where $l_i$ is the last index of the rectangles in layer $i$ . The height of each layer is $b_i$ and the vertical condition for existence of such placement is:

$$\textbf{U} : \quad \sum_{i=1}^{p} b^i \; + \; (p + 1) * \varepsilon \leq b$$

where $p$ is the number of layers. If condition **U** is satisfied we can fit the obstacles using this algorithm.

We will label the lines along the middle of the horizontal passages between the layers "horizontal freeways" $h_i$ , $(i=0,1,...,p)$ . We define the lines $v_l$ and $v_r$. as "vertical freeways". Here $v_l$ is the mid-line between the left edge of the boundary and the line defined by the left edges of the first obstacles for all layers. In Figure 5.1 $v_r$ is the mid-line between the right edge of the outside boundary and the line defined by the furthest to the right , right edge of the last obstacles for all layers. We will use the horizontal and the vertical freeways to reach from point to point in the free space.

The link distance between any two points in the free space is maximum five. If we take points $X$ and $Y$ in Figure 5.1 for example, we need one link to reach from $X$ to the horizontal freeway above it, a second link to go to the extreme left (or right) on the freeway, a third link to go along the vertical freeway to the horizontal freeway above point $Y$, a fourth link to move along the freeway until we get directly above $Y$ and a fifth link to actually reach $Y$.

Alternatively, we can always choose a point that can reach all other points in the free space with three links only. For example we can take any point $Z$ along $v_1$. Then to reach point $X$ for example we can move with one link along the $v_1$ until we reach the horizontal freeway corresponding to $X$. Then we use a second link to move along this horizontal freeway until we get above $X$ and the third link reaches $X$. Thus we have proven:

Theorem 5.1. For a planar environment, if we can place the obstacles in such a way that condition U is satisfied then: $VMIN \leq 3$ and $VMAX \leq 5$.

If we want to put the base of the robot in a more central location we can take a point at the middle of the horizontal freeway that is closest to the middle of the vertical sides of the outside boundary, for example point $W$ in Figure 5.1. In that case we need a maximum of four links to reach all the points in the free space--one to reach the point of intersection of the horizontal freeway with $v_1$ and then three more to reach all the other points. However, if this is an important condition for us, we can slightly modify our algorithm by introducing a vertical freeway $v_C$ through the geometric center of the bounding rectangle, thus achieving maximum 3 links for the central point.

We note that once we have distributed the obstacles in the horizontal layers, we can reshuffle the obstacles in each layer so that they have more convenient placing within the corresponding layer. We can also incorporate some practical considerations in the initial numeration of the obstacles.

Sometimes it is also convenient to place the obstacles along the whole boundary of the environment. In that case we can use rectangular layers instead of horizontal ones. If we can place the obstacles in $p$ rectangular layers, Theorem 5.2 in [7] shows that $VMIN \leq [(p+1)/2]+3$ and $VMAX \leq p+3$ .

There is a very important difference between the estimates in Chapter 4 and Theorem 5.1. In our discussion we have used the term "links" to denote "telescoping links", i.e. links with two degrees of freedom (one revolute and one prismatic). However the algorithms we discussed in this chapter only need links with one degree of freedom (prismatic links). We are moving along freeways that are perpendicular to each other and we do not really need the rotation of the links. So if we use a central highway and if condition U is satisfied, we can design the environment in such a way that no matter how many obstacles we have, a manipulator with three prismatic links is sufficient to cover the whole environment. This is a very strong result, even more because prismatic structures are easier to analyze and control than revolute.

This result can also be directly transferred to 3D environments. In 3D we will approximate the obstacles with parallelepipeds (or if possible with cubes). This approximation is also an open problem, and we do not know of substantial results and algorithms that achieve it. However, if we can approximate the original obstacles and the outside boundary with parallelepipeds, we can subdivide the environment into layers parallel to the $X$-$Y$ plane, and arrange the obstacles according to their height (which can be the smallest dimension of the parallelepipeds) and place them in order in layers. We can prove Theorems 5.1 and 5.2 for this case, the same way as in 2D.

## 6. Future Directions

The problem we discussed in this paper was: given an environment with obstacles, what is the optimal design of a robot that can reach everywhere in this environment without collision with the obstacles. There are several open problems which can be a topic for future research:

- the problem of 3D modeling of the obstacles for convenient application of the link center algorithm to obstacles with general shape;
- the problem of deriving estimates for the number of links in 3D using enclosures in simpler shaped obstacles;
- the problem of deriving a general condition, through optimization, for best fitting of rectangular obstacles in the environment;

In addition to those problems, it would be interesting to consider further generalizations of the problems discussed in this paper. Those include:

investigate the affect of the kinematic and dynamic properties of the telescoping links. One could also consider the affect of joint limits and singularities on the algorithms in Section 2.

incorporate uncertainty and imprecise knowledge of the shape and the position of the obstacles.

generalize to any $n$-dimensional space by using either topology of semi-algebraic sets.

The ideas for robot placement and design that we discussed can be used in space robotics, factory automation , construction industry and transportation.

## References

[1] David Bennett and John Hollerbach, "Identifying the Kinematics of Robots and their Tasks," *Proceedings of the IEEE International Conference on Robotics and Automation*, 1989, pp.580-586.

[2] Bernard Chazelle and Leonidas Palios, "Triangulating a Nonconvex Polytope,"

*Proceedings 5th ACM Symposium on Computational Geometry*, **1989, pp.393-400.**

**[3] P.Chedmail and Ph.Wenger, "Design and Positioning of a Robot in an Environment with Obstacles using Optimal Research,"** *Proceedings of the IEEE International Conference on Robotics and Automation*, **1989, pp.1069-1074.**

**[4] Yao-Chon Chen and Mathukumalli Vidyasagar, "Optimal Trajectory Planning for Planar n-Link Revolute Manipulators in the Presence of Obstacles,"** *Proceedings of the IEEE International Conference on Robotics and Automation*, **1988, pp.202-208.**

**[5] Gregory Chirikjian and Joel Burdick, "Parallel Formulation of the Inverse Kinematics of Modular Hyper-Redundant Manipulators,"** *Proceedings of the IEEE International Conference on Robotics and Automation*, **1991, pp.708-713.**

**[6] Marc Keil, "Decomposing a Polygon into Simpler Components,"** *SIAM Journal on Computing*, **Vol.14, No.4, 1985, pp.789-817.**

**[7] Kolarov, K.** *Optimal geometric Design of Robots for Environments with Obstacles*, **PhD Thesis, Stanford University, 1992.**

**[8] Kolarov, K. and Roth, B. "On the Number of Links and Placement of Telescoping Manipulators in an Environment with Obstacles",** *Proc. 5th International Conference on Advanced Robotics*, **Pisa, Italy, 1991, pp.988-993.**

**[9] Kolarov, K. and Roth, B. "Best Estimates for the Construction of Robots in Environments with Obstacles",** *IEEE International Conference on Robotics and Automation*, **Nice, France, 1992.**

**[10] Kolarov, K. and Roth, B. "Best Placement of Telescoping Robots in Environments with Obstacles", Bangalore, India, 1993.**

**[11] Sridhar Kota and Tushchai Chuenchom, "Adjustable Robotic Mechanisms for Low-Cost Automation,"** *Cams, Gears, Robot and Mechanism Design*, **ASME 1990, pp.297-306.**

**[12] Jean-Claude Latombe,** *Robot Motion Planning*, **Kluwer Academic Publishers, 1991.**

**[13] Jean-Paul Laumond, "Obstacle Growing in a Non-Polygonal World,"** *Information Processing Letters* ***25***, **April 20, 1987, pp.41-50.**

**[14] D.Lee, I.Chen, "Display of Visible Edges of a Set of Convex Polygons,"** *Computational Geometry*, **G.Toussaint (Ed.),1985, pp.249-265.**

**[15] W.Lenhart, R.Pollack, J.Sack, R.Seidel, M.Sharir, S.Suri, G.Toussaint, S.Whitesides and C.Yap, "Computing the Link Center of a Simple Polygon,"** *Proceedings of the 63rd ACM Symposium on Computational Geometry*, **1987, pp.1-10.**

**[16] O.Ma and J.Angeles, "Optimum Architecture Design of Platform Manipulators,"** *Proceedings of the 4th International Conference on Advanced Robotics*, **1991, pp.1130-1135.**

**[17] J.A. Pamanes and Said Zeghloul, "Optimal Placement of Robotic Manipulators Using Multiple Kinematic Criteria,"** *Proceedings of the IEEE International Conference on Robotics and Automation*, **1991, pp.933-938.**

**[18] John Reif and James Storer, "Shortest Paths in Euclidean Space with Polyhedral Obstacles,"** *Technical Report CS-85-12*, **Computer Science Department, Brandeis University, 1985.**

**[19] Joseph O'Rourke,** *Art Gallery Theorems and Algorithms*, **Oxford University Press, 1987.**

**[20] Joseph O'Rourke and Kenneth Supowit, "Some NP-Hard Polygon Decomposition Problems,"** *IEEE Transactions on Information Theory*, **Vol. IT-29, No.2, March 1983, pp.181-190.**

**[21] E.Sandgren and S.Venkataraman, "Robot Path Planning and Obstacle Avoidance: A Design Optimization Approach,"** *15th Design Automation Conference* **1989, pp.169-175.**

**[22] Subhash Suri and Joseph O'Rourke, "Worst-Case Optimal Algorithms For Constructing Visibility Polygons with Holes,"** *Proceedings of the 2nd ACM Symposium on Computational Geometry*, **1986, pp.14-23.**

**[23] T.Toussaint, "Solving Geometric Problems with the 'Rotating Calipers',"** *Proceedings IEEE MELECOM '83*, **Greece.**