CHAPTER IV

REGRASP PLANNING FOR A POLYGON WITH A LARGE NUMBER OF EDGES

4.1 Introduction

This chapter addresses the problem of regrasp planning for a polygon with a large number of edges. We consider the case that uses minimum number of fingers to grasp and regrasp, i.e., 2-finger grasps is taken into account and one additional finger is used for a finger switching. Therefore, a hand applied in this work is assumed to equip with three fingers. To obtain the complete structure for regrasp planning of an object, all force closure grasps are computed in grasp space. Instead of naively constructing a switching graph from all uncombined sets of grasps computed from all combination of polygonal edges, we propose to merge sets of grasps that connect to one another into a connected set. In grasp space, a connected set allows continuous changing among any grasping configurations in the set, i.e., it allows continuous movements of fingers across polygonal edges. The obtained connected sets are then used to construct nodes of a switching graph. We propose an output sensitive algorithm which efficiently computes all edges of a switching graph. Regrasp planning then can be formulated as a graph search problem where nodes of the graph represent connected sets of force closure grasps while an edge connects two sets that can be changed between each other by finger switching. A method for finding the optimal solution of a finger switching is also presented.

4.2 Representing force closure grasps

In this section, we describe how to represent and construct the configuration space that characterizes all force closure grasps. The polygonal boundary of the rigid object to be grasped must not be self-intersecting but could be broken into many simple polygons. It is sufficient to consider only the case that the boundary is composed of at most two simple polygons: if there are more than two simple polygons that define the boundary, we can pick two at a time and run the same algorithm over all possible pairs.

The configuration space of the problem is clearly the contact space. A configuration

consists of two parameters, each of which defines where each finger is placed along the boundary of the grasped object. This section describes how to represent and construct the configuration space that characterizes all force closure grasps.

The entities of a polygon needed in our discussion are defined as follows. A simple polygon P is described by n distinct vertices $\boldsymbol{v}_i \in \mathbb{R}^2$ where $i \in \mathbf{Z}_n$ [1]. It is assumed that $\boldsymbol{v}_i$ are arranged counterclockwise if they represent the outer boundary of the object, or arranged clockwise if they represents the hole inside the object. Edges E_i are line segments with endpoints $\boldsymbol{v}_i$ and $\boldsymbol{v}_{i+1}$. Every point $\boldsymbol{p}$ on the boundary of P can be mapped to the length of curve measured counterclockwise from $\boldsymbol{v}_0$ to $\boldsymbol{p}$ along the boundary. We will write $length(\boldsymbol{p})$ to represent such length. Lengths of E_i can be computed by the equation $l_i = \|\boldsymbol{v}_{i+1} - \boldsymbol{v}_i\|$. It is obvious that $L_i = length(\boldsymbol{v}_i) = \sum_{j \in \mathbf{Z}_i} l_j$. We denote by L the total length of the boundary of P, which can be computed by $L = \sum_{i \in \mathbf{Z}_n} l_i$.

Next, let us define the tangents of E_i as $\boldsymbol{t}_i = (\boldsymbol{v}_{i+1} - \boldsymbol{v}_i)/l_i$. The normal vectors $\boldsymbol{n}_i$ of E_i are unit vectors that are perpendicular to $\boldsymbol{t}_i$ and point inward. Note that $\boldsymbol{n}_i$ can be obtained by rotating $\boldsymbol{t}_i$ counterclockwise by $\pi/2$ radian. The cone of forces C_i that can be exerted on the edge E_i is defined by two vectors $\boldsymbol{n}_i + (\tan\alpha)\boldsymbol{t}_i$ and $\boldsymbol{n}_i - (\tan\alpha)\boldsymbol{t}_i$ where $\alpha \in [0, \pi/2)$ is the half-angle of the friction cone.

Since we deal with two fingers that might not reside on the same polygon, two sets of entities for different polygons are needed. Let all entities defined above correspond to the polygon P which is in contact with the first finger, and let n', $\boldsymbol{v}'_i$, E'_i, $length'$, l'_i, L'_i, L', $\boldsymbol{t}'_i$, $\boldsymbol{n}'_i$ and C'_i be defined similarly for the polygon P' in contact with the second finger. If the two fingers are on the same polygon, then $n = n'$, $length = length'$, $L = L'$ and $X_i = X'_i$ where $X = \boldsymbol{v}, E, l, L, \boldsymbol{t}, \boldsymbol{n}$ or C.

The configuration space $\mathbf{C}$ of the two fingers is $[0, L) \times [0, L')$. Given a configuration $(u, u') \in \mathbf{C}$, we say that (u, u') composes a 2-finger grasp if and only if the two contact points $length^{-1}(u)$ and $(length')^{-1}(u')$ achieve force closure. (Recall that $length$ is a function that maps a vertex into a number, so $length^{-1}$ gives a vertex.) Let the *graspable set* $G \subseteq \mathbf{C}$ be the set of all configurations that compose 2-finger grasps (Obviously, G is the set of all configurations contained in all force closure regions mentioned in Section ??). Also let the *graspable subsets* $G_{i,j}$ be graspable regions on edges E_i and E'_j defined by

$$G_{i,j} = G \cap ([L_i, L_{i+1}] \times [L'_j, L'_{j+1}]).$$

[1] $\mathbf{Z}_n$ is a group of non-negative integers less than n. Addition and subtraction are computed modulo n.

4.2.1 Computing $G_{i,j}$

Clearly each $G_{i,j}$ corresponds to configurations whose one finger is on E_i and the other is on E'_j. This problem of finding all force closure grasps on a pair of edges has been well studied. Using Proposition 3.1, it has been shown in (Faverjon and Ponce, 1991) that $G_{i,j}$ can be defined by eight linear inequalities in the parameters u and u'. Here, let us present an easier method to define $G_{i,j}$. We define the inverted force cone $-C'_j$ as $\{-\boldsymbol{x} \mid \boldsymbol{x} \in C'_j\}$. It was shown in (Nguyen, 1988b) that emptiness of $C_{i,j} = C_i \cap (-C'_j)$ implies emptiness of $G_{i,j}$. If $C_{i,j}$ is not empty, we claim that $G_{i,j}$ can be defined by no more than six points on the bounding rectangle. The claim is justified as follows.

Since a 2-finger grasp can be either *compressive*(squeezing grasp) or *expansive*(stretching grasp), we define for simplicity $DC_{i,j} = C_{i,j} \cup (-C_{i,j})$ as the double-sided cone of $C_{i,j}$ so that both the stretching and squeezing cases can be dealt with together. Now we prove the above claim by examining $DC_{i,j}$ centered on E_i. Let us first extend both sides of the edges E_i and E'_j to infinity, choose an arbitrary real number u, find $\boldsymbol{p}(u) = length^{-1}(u)$ on E_i, then let $DC_{i,j}(u)$ be the cone $DC_{i,j}$ centered at $\boldsymbol{p}(u)$. The intersection $I(u)$ of E'_j and the cone $DC_{i,j}(u)$ is a line segment on E'_j which represents the region that the second finger can be placed to achieve force closure with the first finger at $\boldsymbol{p}(u)$. This means for a given position of the first finger u, $length'(I(u))$ is the corresponding graspable interval in the second finger's configuration space (Fig. 4.1(a, b)).

It is easy to see that if u moves by Δu, $p(u)$ will move in the direction of $\boldsymbol{t}_i$ by the same distance Δu, the endpoints of $I(u)$ will move in the direction of $-\boldsymbol{t}'_j$ by the distance proportional to Δu, and the endpoints of $length'(I(u))$ will move in the $-\Delta u$ direction by the distance proportional to Δu (Fig. 4.1(c, d)). These linear relationships imply that the graspable region is bounded by two straight lines. It is now obvious that cutting E_i and E'_j to their original lengths is equivalent to imposing four rectangular constraints $u \geq L_i$, $u \leq L_{i+1}$, $u' \geq L'_j$ and $u' \leq L'_{j+1}$ in the (u, u')-space (Fig. 4.1(e)). Therefore, $G_{i,j}$ can be defined with no more than six points on the bounding rectangle. In the real implementation, all defining points of $G_{i,j}$ can be found by computing endpoints of four intersections: $DC_{i,j}(\boldsymbol{v}_i) \cap E'_j$, $DC_{i,j}(\boldsymbol{v}_{i+1}) \cap E'_j$, $DC_{i,j}(\boldsymbol{v}'_j) \cap E_i$, and $DC_{i,j}(\boldsymbol{v}'_{j+1}) \cap E_i$ (Fig. 4.2).

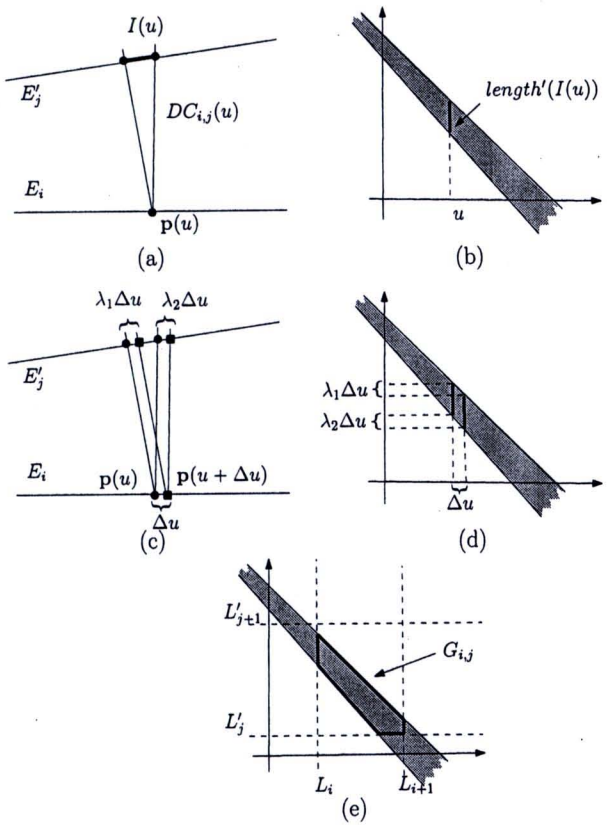

Figure 4.1: Computing $G_{i,j}$: (a) $I(u)$ is the graspable region of the second finger when the first finger is at $\boldsymbol{p}$. (b) $length'(I(u))$ is an interval in u' configuration space. (c) Endpoints of $I(u)$ move by the distances proportional to Δu. (d) Endpoints of $length'(I(u))$ move by the same distances as endpoints of $I(u)$, giving two straight lines bounding the graspable region. (e) $G_{i,j}$ is the result of cutting the infinite area by the rectangle.

4.2.2 Extending Configuration Space

Since each finger can be positioned anywhere in its ICR in order to form a force closure grasp, and its position can be represented by one parameter, it is clear that the ICRs can be depicted in the configuration space as rectangles whose sides are parallel to u and u' axes. The mapping results in one rectangle if the ICRs do not contain v_0 or v_0', two rectangles if the ICRs contain v_0 or v_0' but not both, or four rectangles if the ICRs contain both v_0 and v_0' (Fig. 4.3). To eliminate the need to find ICRs with multiple rectangles, we extend the domain of $length^{-1}$ and $(length')^{-1}$ to the whole real line so they both become functions with periods L and L' respectively. ($length$ and $length'$ are no longer one-to-one.) The new G in the expanded configuration space $\mathbb{R}^2$ can be defined from the old G with these periodic relations:

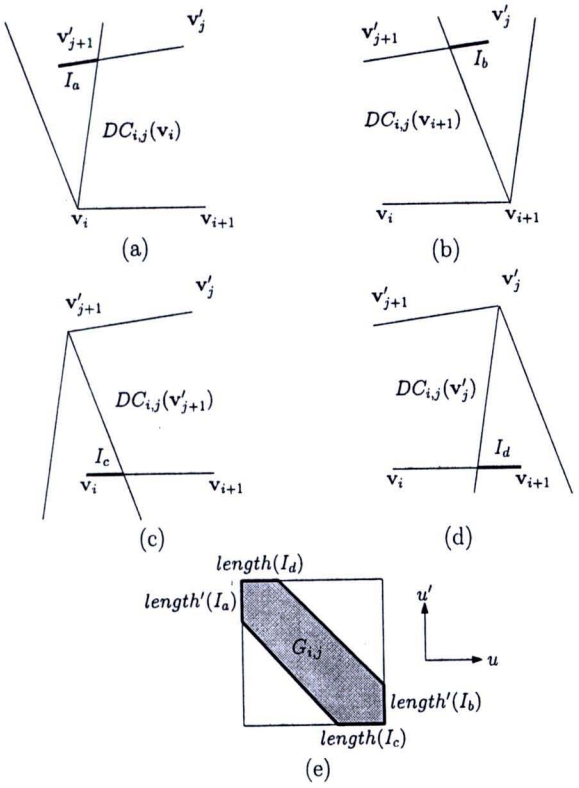

Figure 4.2: Extreme points of $G_{i,j}$: Four intersections I_a, I_b, I_c and I_d are shown in (a), (b), (c) and (d). $G_{i,j}$ can be immediately defined by these intersections as shown in (e).

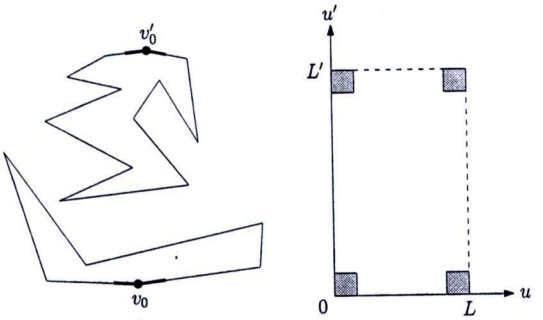

Figure 4.3: The independent contact region shown in thick lines (left) maps to four rectangles in the configuration space (right).

- $(u, u') \in G \Leftrightarrow (u + L, u') \in G$.
- $(u, u') \in G \Leftrightarrow (u, u' + L') \in G$.

We claim that despite infiniteness of G, the ICRs have one corresponding rectangle within $G \cap [0, 2L] \times [0, 2L']$. This is easily proved by the following argument.

- Suppose a position of one finger is given, there must be some positions of the second finger that do not form force closure with the first finger.
- It follows that all u-constant line segments in G are shorter than L' and all u'-constant line segments in G are shorter than L.
- Since the ICRs can be mapped into some rectangles in G whose sides are axis-parallel segments in G, one of these rectangles must lie inside $[0, 2L] \times [0, 2L']$.

The special case where the two fingers touch the same polygon can be handled with a smaller configuration space. G will be symmetric about the axis $u = u'$, which means we can cut out one half of G that lies above (or below) the line $u = u'$ (Fig. 4.4(a)). The remaining part of G above $u' > L$ (or to the right of $u > L$) can also be eliminated because to every rectangle crossing the line $u' = L$ (or the line $u = L$), there corresponds a rectangle in $[0, 2L] \times [0, L]$ (or $[0, L] \times [0, 2L]$) that represents the same configurations (Fig. 4.4(b)). Finally, the region to the right of the line $u = u' - L$ (or above the line $u = u' + L$) is redundant because no point on this line is in G (Fig. 4.4(c)). Therefore, the region of consideration is the shaded portion as shown in Fig. 4.4(d).

4.2.3 Constructing G

Now we know that each $G_{i,j}$ contains at most six defining vertices, so all $G_{i,j}$ can be constructed within $O(nn')$ time. In the final algorithm, we will need the polygonal representation of G, so adjacent $G_{i,j}$ must be merged together into big pieces. Many simple polygons may be needed to define G because G does not have to be simple nor connected.

A vertex of some $G_{i,j}$ is a *defining vertex of* G, or *defines* G, if it is a vertex on a boundary of G. It can be observed that a vertex v of some $G_{i,j}$ defines G if and only if one of the following is true:

- v is not at a corner of the bounding rectangle.
- v is a corner of four bounding rectangles (one contains $G_{i,j}$ and the other three are adjacent), but is not contained in some $G_{k,l}$ bounded by these rectangles.

Note that if $G_{i,j}$ is neither empty nor full ("full" means $G_{i,j} = [L_i, L_{i+1}] \times [L'_j, L'_{j+1}]$), at least one vertex of $G_{i,j}$ must be a defining vertex of G.

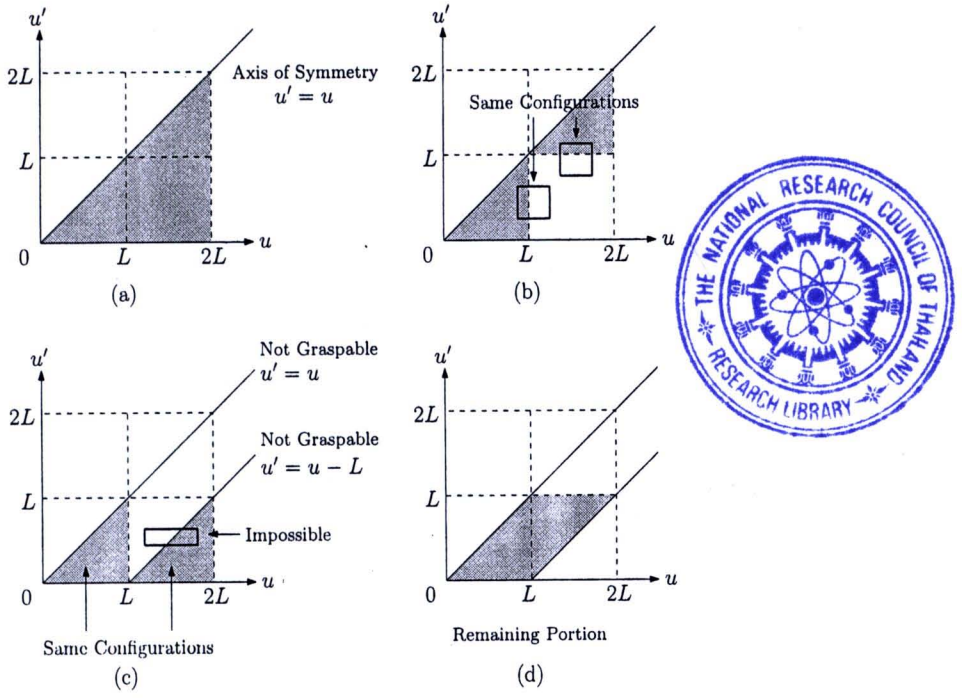

Figure 4.4: (a) The axis of symmetry is $u' = u$. (b) For each rectangle that crosses the line $u' = L$, there corresponds another rectangle in $[0, 2L] \times [0, L]$ that crosses the line $u = L$. (c) The line $u' = u$ and $u' = u - L$ never intersect G, and the part of G to the right of $u' = u - L$ represents the same configurations as the remaining portion in $[0, L] \times [0, L]$. (d) The remaining portion to consider is shown in the shaded area.

The algorithm to find all simple polygons that define boundaries of G is described as follows. Let us first attach a state "used/unused" to all vertices of all $G_{i,j}$. All vertices are initialized as "unused". We scan through all values of i and j, and do the following:

- While $G_{i,j}$ has an "unused" vertex v that defines G,
  - It is clear that v is on a boundary of G, so we can *trace* the boundary of G from v until we get back at v.
  - All vertices traced along the way defines a simple polygon which is a boundary of G. Mark these vertices as "used".

Note that *tracing* the boundary of G from $G_{i,j}$ may involve many $G_{k,l}$.

The tracing process can be simplified by first defining adjacencies of vertices that define G. Situations when two vertices v_1 and v_2 *that define* G are adjacent in G are listed below:

- If v_1 and v_2 are adjacent in the polygonal representation of $G_{i,j}$ and they lie on different sides of the bounding rectangle of $G_{i,j}$, they are adjacent in G (Fig. 4.5(a)).

- If v_1 and v_2 are adjacent in the polygonal representation of $G_{i,j}$ and they lie on the same side of the bounding rectangle of $G_{i,j}$, we assume without loss of generality that $v_1, v_2 \in \{L_i\} \times [L'_j, L'_{j+1}]$. v_1 and v_2 will be adjacent in G if $G_{i-1,j} \cap \overline{v_1 v_2} = \varnothing$ (Fig. 4.5(b)).

- If v_1 and v_2 belong to different pieces, i.e. $G_{i,j}$ and $G_{k,l}$, we assume without loss of generality that $v_1 \in G_{i,j}, v_2 \in G_{i-1,j}$. v_1 and v_2 can be adjacent in G if and only if they lie on the same segment $\{L_i\} \times [L'_j, L'_{j+1}]$ and

  - $\overline{v_1 v_2} \subseteq G_{i-1,j}$ and $\overline{v_1 v_2} \cap G_{i,j} = \{v_1\}$, or
  - $\overline{v_1 v_2} \subseteq G_{i,j}$ and $\overline{v_1 v_2} \cap G_{i-1,j} = \{v_2\}$ (Fig. 4.5(c)).

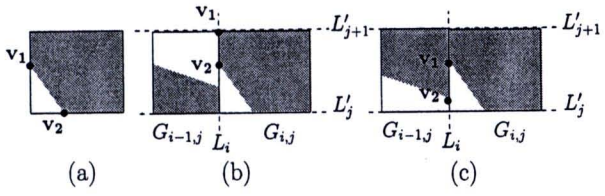

Figure 4.5: Adjacencies of vertices

A vertex on a boundary of G defines either an outer boundary or a hole of G. It is necessary to distinguish that the vertex defines an outer boundary or a hole. A sequence of vertices representing a polygon starting at v is obtained when a tracing gets back at v but we do not exactly know the orientation of the sequence. We firstly assume that it represents a simple closed polygon and then rearrange it into a counterclockwise order. To rearrange it, we start with the vertex that is extreme in u' direction denoted by v_i. Let its adjacent vertices be v_{i-1} and v_{i+1}. The cross product of $\overline{v_{i-1}v_i}$ and $\overline{v_i v_{i+1}}$ is calculated. Since the internal angle at the extreme vertex is convex and strictly less than π, they orient counterclockwise when the cross product is positive. If the cross product is negative, the sequence of the vertices is reversed.

The rearranged sequence defines an outer boundary when two adjacent vertices v_i and v_{i+1} of the sequence form the segment $\overline{v_i v_{i+1}}$ such that the graspable region is on the left side of $\overline{v_i v_{i+1}}$, whereas the sequence represents a hole when the graspable region is on the right side of $\overline{v_i v_{i+1}}$. All vertices in the sequence are identified that they define an outer boundary or a hole for further use.

The number of vertices defining G can be decreased by eliminating collinear vertices. Let a connected boundary of G consist of n vertices $v_i \in \mathbb{R}^2$ where $i \in \mathbf{Z}_n$. A vertex v_i can be eliminated when the slope of $\overline{v_{i-1}v_i}$ is equal to the slope of $\overline{v_iv_{i+1}}$.

4.3 Finger Switching

Regrasp process which changes grasping configuration by placing an additional finger on desired contact point and then releasing one finger of the initial grasp is called finger switching. Intuitively, considering grasps on two different grasp sets, a finger switching can be performed when there exists a common contact point on the grasped object. In grasp space, the common contact points are computed in subspaces of one parameter. It requires projections of two grasp sets onto the subspaces. The projections are then checked for the intersection which indicates a set of common contact points. This operation involves with an edge in the switching graph. Considering two grasp sets associated with two nodes, existence of finger switching between these sets indicates an edge linking the two nodes.

Finger switching requires that one non-switching contact points must remain the same during the process. Formally, there will be an edge connecting a node v_a and a node v_b when there exists a couple of points $(length^{-1}(u_a), length^{-1}(u'_a))$ where $(u_a, u'_a) \in P_a$ and a couple $(length^{-1}(u_b), length^{-1}(u'_b))$ where $(u_b, u'_b) \in P_b$ such that $u_a = u_b$ or $u'_a = u'_b$.

To check whether there exist grasps from two grasp sets that can switch to each other, we consider two polygons representing these grasp sets. The projection $\pi_u(P_a)$ of P_a on the axis of parameter u_a (Fig. 4.6(a)) represents the set of points on the object that are possible to form 2-finger grasps with some points corresponding to the projection $\pi_{u'}(P_a)$ of P_a on the axis of parameter u'_a. Similarly, the projections of P_b (Fig. 4.6(b)) represents the subspaces of 2-finger grasps . Note that $\pi_u(P_a)$ and $\pi_u(P_b)$ are in the same subspace, if the intersection between these two projections is not empty (Fig. 4.6(c)), then there exists points $length^{-1}(u_a)$ on the object that form 2-finger grasps with $length^{-1}(u'_a)$ and $length^{-1}(u'_b)$ concurrently when $u_a = u_b$ is satisfied. We apply the same process to check the existence of finger switching on the space of parameters u'_a and u'_b by considering $\pi_{u'}(P_a)$ and $\pi_{u'}(P_b)$.

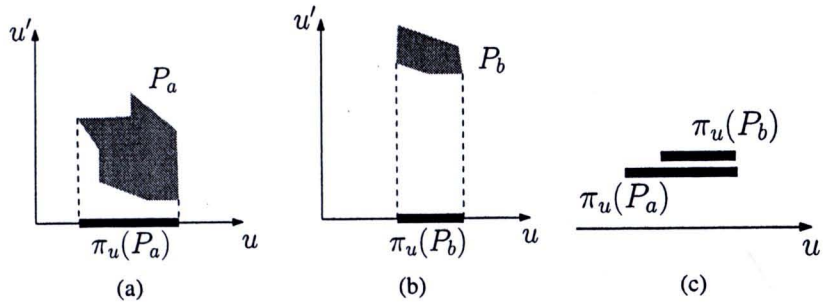

Figure 4.6: Finger switching: The projection of (a) P_a, (b) P_b on u parameter space. (c) Overlapping projections

4.4 Finger Aligning

Finger aligning is a process for repositioning fingers by rolling or sliding them along edges of a polygon while maintaining a force closure grasp during the repositioning process. By applying this operation, we can change grasping configuration with in the same connected set of grasps. This expresses the direct relation between finger aligning and a node of the switching graph.

Finger aligning is necessary as exemplified in the following instance. Let us consider Fig. 4.7(a). Obviously, because the current position of finger 1 cannot form a 2-finger grasp with the upper part, it is not possible to switch directly from the current grasp to a grasp that one finger is placed on the upper part using finger switching. However, somehow if the hand can continuously adjust finger 1 and 2 to change from the current grasp to a new grasp in Fig. 4.7(b), a finger switching can be performed to switch to another grasp by placing finger 3 on the upper part to form a 2-finger grasp with finger 1 (Fig. 4.7(c)) and then releasing the finger 2 (Fig. 4.7(d)).

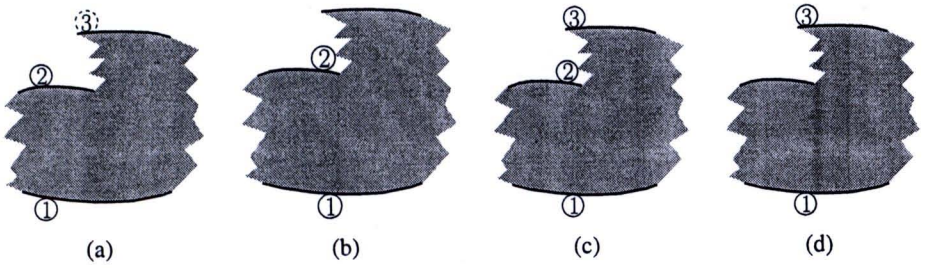

Figure 4.7: Finger aligning

Finger aligning is trivial from the construction of connected grasp sets $P_1, \ldots, P_m$. Each node in the switching graph corresponds to exactly one grasp set. Every grasp in each node can be repositioned to another grasp of the same node by finger aligning

because of continuity in a connected set of 2-finger grasps.

4.5 Constructing Switching Graph

To construct a switching graph, all of its vertices and edges have to be found. We compute all $G_{i,j}$ and construct G to identify all connected polygons $P_1, \ldots, P_m$. Each connected polygon is associated with a node of the switching graph.

To construct all edges, all polygons have to be checked for finger switching among them. Instead of exhaustively testing all pair of polygons, we apply a sweep algorithm to find overlaps among the projections of the polygons in a parameter subspace. Let the projection $\pi_u(P_a)$ of a polygon P_a on the axis of parameter u_a be represented by an interval $(l_a : h_a)$ where l_a is the lower endpoint and h_a is the higher endpoint. The lower endpoint and the higher endpoint are obtained from the leftmost vertex and the rightmost vertex of P_a respectively. The intervals of all polygons are used in our algorithm. We firstly sort all endpoints in increasing order and store the sorted endpoints into an event queue E. A priority queue Q is used to store intervals and identify overlaps among intervals. The priority of a interval is based on the value of its higher endpoint, less value has more priority. We are now ready to start the sweeping process from the first element in E. An endpoint is dequeue from E. If it is a lower endpoint, its associated interval is added into Q. Otherwise, if it is a higher endpoint, its associated interval is dequeued from Q. Clearly, the interval has the highest priority, we can use *ExtractMin* operation of the priority queue to remove it from Q. Let the dequeued interval be $(l : h)$. All remaining intervals in Q have the lower endpoints that less than h and the higher endpoints that higher than h therefore they overlap the interval $(l : h)$. As a consequence, the associated node of the interval $(l : h)$ has edges linking nodes associated with the remaining intervals in Q. The process is repeated from dequeuing E and so on until E is empty. We also have to check overlaps in the subspace u' using the same algorithm. The pseudocode of edge construction is as follows.

1: $E = Sort(l_1, h_1, l_2, h_2, \ldots, l_m, h_m)$
2: **while** E is not empty **do**
3: $\quad e = E.Dequeue()$
4: $\quad$ **if** e is lower endpoint **then**
5: $\quad\quad Q.Insert$(interval of e)
6: $\quad$ **else**
7: $\quad\quad Q.ExtractMin()$
8: $\quad\quad L =$ all remaining elements of Q

```
9:        v = the associated node of e
10:       for all i ∈ L do
11:         n = the associated node of i
12:         link(v, n)
13:       end for
14:     end if
15: end while
```

The construct of the event queue E takes $O(m \log m)$ running time. A priority queue using a heap give performance $O(1)$ to insert an element into Q. *Extract-Min* operation takes $O(\log m)$. For all endpoints, our output sensitive algorithm takes $O(m \log m + mk)$ where k is the average number of overlapping intervals of one interval.

4.6 Using Switching Graph

4.6.1 Unconstrained Regrasp Sequence

A switching graph provides a tool for planning a regrasp sequence. A path connecting the node containing the initial grasping position and the node containing the required grasping position indicates a sequence of edges that a finger switching should be performed. However, a path in a switching graph does not directly indicate which contact points on grasping edges are to be used in each step. For a pair of nodes having an edge connecting them, a switching graph tells us that we can switch between two grasps from two grasp sets but it does not tell which grasping points that we can perform a finger switching. This section describe a method of transforming a path in a switching graph to an actual regrasp sequence.

First, let us consider a finger switching. Finger switching takes place when we move from one node to another node in a graph. An edge in the graph tells us that a finger switching is viable. We have to find two grasps on each node that have one non-switching contact point in common. We pick a point from the intersection of the projections described in section 4.3. That point indicates one actual non-switching point. The next step is to find a point forming a grasp of the first node and a point forming a grasp of the second node. Let us consider a polygon defined in section 4.3. Once a value of u_a or u'_a in the intersection of the projections of P_a and P_b is chosen, we can construct a set of feasible contact points for the other finger by intersecting P_a with the line passing u_a and parallel with the axis of u'_a or the line passing u'_a and parallel with the axis of u_a

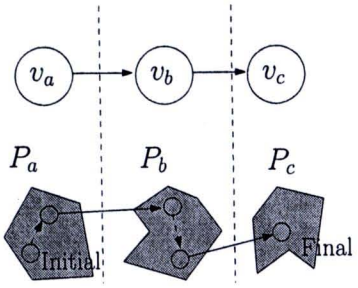

Figure 4.8: A corresponding between nodes and edges in a switching graph and a finger switching and a finger aligning. A dashed line in the bottom figure represents a finger aligning while a solid line represents a finger switching

Next, let us consider a finger aligning. Finger aligning may be required in-between two finger switchings, i.e., when we just traversed from node v_a to node v_b and about to move to the next node v_c. Let us assume that the first finger switching is just performed and we currently are in a grasp represented in v_b. In order to perform the next finger switching, i.e., to move to the node v_c, the grasping position must have one contact point in common with the final grasp. However, it might not be the current grasp. When an appropriate grasping configuration is computed as described earlier in this section, we have to change from the finishing grasp of the first switching to the a next switching. Since these two grasps are from the same connected set, we can change the current grasp to an appropriate grasp for the next switching by a finger aligning. Fig. 4.8 shows the corresponding between a switching graph and the actual action performed on a regrasp.

4.6.2 Optimal Regrasp Sequence

In this section, we will plan for a regrasp sequence that independent contact regions (ICR) are locally optimized for each finger switching using the principle of L_∞ Voronoi diagram (Papadopoulou and Lee, 2001). We propose to accomplish this task by computing the L_∞ Voronoi diagram of the edges of G. Considering only part of the resulting L_∞ Voronoi diagram that lies inside G, we claim that the largest square in G must be centered at a vertex of the diagram. Before proving this claim, let us briefly review the concept of L_∞ Voronoi diagram.

The L_∞ Voronoi diagram is conceptually defined in the same manner as the ordinary Voronoi diagram except that L_∞ distance metric is used instead of the more familiar L_2. Recall that the L_∞ distance $d(\boldsymbol{p}, \boldsymbol{q})$ between two points $\boldsymbol{p} = (p_x, p_y)$ and $\boldsymbol{q} = (q_x, q_y)$ is given by $d(\boldsymbol{p}, \boldsymbol{q}) = \max(|p_x - q_x|, |p_y - q_y|)$, i.e., the maximum of the differences in each coordinate, and the L_∞ distance $d(\boldsymbol{p}, e)$ between point $\boldsymbol{p}$ and segment e is defined to

be $\min_{q \in e} d(p, q)$, i.e., the shortest L_∞ distance between p and any point on e. Let S be a set of segments in the plane. A point lies on the L_∞ Voronoi diagram of S if, among all segments in S, at least two of them are equally of the lowest L_∞ distance to the point (in other words, they are the L_∞ nearest segments of the point). Equivalently speaking, considering the definition of the L_∞ distance, a point is on the L_∞ Voronoi diagram of S if it is the center of a square that touches at least two of the segments in S without having its interior intersect with any. An L_∞ Voronoi edge is defined to be the set of all points on the L_∞ Voronoi diagram that are on the same straight line segment. A Voronoi vertex is defined to be the point at which at least two Voronoi edges meet (implying that the point is equally L_∞ far from at least three segments). Since every point on the L_∞ Voronoi diagram lies at least on one Voronoi edge, the L_∞ Voronoi diagram is essentially a network of straight line segments. Specifically, the L_∞ Voronoi diagram of S divides the plane into polygonal regions called Voronoi regions. Each Voronoi region is associated with a unique segment in S such that the region entirely contains the segment, and any point in the interior of the region is L_∞ closer to this segment than to any other segment in S.

ICR are defined by a rectangle in G whose shorter side length is maximum. We extend to the problem of optimizing ICR for a finger switching which involves two grasps concurrently. The measure of goodness of two rectangles R_a with side lengths a_1 and a_2 and R_b with side lengths b_1 and b_2 is given by $f(R_a, R_b) = \min\{a_1, a_2, b_1, b_2\}$. Our goal is to maximize $f(R_a, R_b)$ such that the grasps represented by the centers of R_a and R_b can switch to each other.

To optimize the criterion, we use another representation of a square to describe ICR. Let v be a point in G and let $square(v)$ denote the largest square centered at v that is fully contained in G. The size of $square(v)$ is determined by $size(v) = min_{p \in \partial G}(d(v, p))$ where $d(v, p) = max(|u_v - u_p|, |u'_v - u'_p|)$. Therefore, any largest square is described by its center v and $size(v)$. Let $v_a \in P_a$ and $v_b \in P_b$, it is clear that maximizing the criterion is equivalent to maximizing $\min\{size(v_a), size(v_b)\}$. We denote by $\pi_u(v)$ and $\pi_{u'}(v)$ the projections of a point v on the axis of u and u' parameters respectively. The problem is transformed to locating the centers $v_a \in P_a$ and $v_b \in P_b$ of two squares such that $\pi_u(v_a) = \pi_u(v_b)$ or $\pi_{u'}(v_a) = \pi_{u'}(v_b)$ and $\min\{size(v_a), size(v_b)\}$ is maximal.

Our algorithm exploits an important characterization of Voronoi edges which allows us to search squares centered on them to optimize the criterion. We claim that the largest $square(v)$ must be centered on a Voronoi edge when one parameter of the point $v \in P$

is restrained. This is justified by the following argument:

We describe in the case that parameter u is restrained as shown by the dotted line in Fig. 4.9(a).

- If v is inside a Voronoi region whose upper and lower boundaries are a Voronoi edge and an edge of P (v_4 in Fig. 4.9(a)), $square(v)$ must have one corner on that polygonal edge. Moving v *away* from that polygonal edge will increase $size(v)$. We can move v in such direction until it reaches a Voronoi edge while $square(v)$ is growing.
- If v is on a Voronoi region whose upper and lower boundaries are both Voronoi edges (v_2 in Fig. 4.9(a)), moving v in one direction, $size(v)$ is increasing or decreasing along the way, and $size(v)$ is decreasing or increasing along the opposite way until it reaches a Voronoi edge (v_1 and v_3 in Fig. 4.9(a)).

This argument allows us to search two points $v_a \in VE_a$ and $v_b \in VE_b$ such that $\pi_u(v_a) = \pi_u(v_b)$ or $\pi_{u'}(v_a) = \pi_{u'}(v_b)$ for optimizing $\min\{size(v_a), size(v_b)\}$ where VE_a and VE_b are sets of Voronoi edges of P_a and P_b.

Searching procedure begins with identifying an interval for finger switching by the intersection between $\pi_u(P_a)$ and $\pi_u(P_b)$ or $\pi_{u'}(P_a)$ and $\pi_{u'}(P_b)$. We again describe in the case of an interval in the space of parameter u. Let $\pi_u(P_a) \cap \pi_u(P_b) \neq \emptyset$ be denoted by an interval $(l : h)$. Voronoi edges in VE_a and VE_b that intersect this interval are considered. It is possible that we obtain many branches of Voronoi edges. All combinations of pairs of Voronoi edge branches are investigated, a pair consists of one Voronoi edge branch from VE_a and another branch from VE_b. For each pair, we divide the two branches using subintervals defined by all Voronoi vertices in the branches as shown by the dotted lines in Fig. 4.9(b). Voronoi vertices are used to determine subdivisions of two Voronoi edge branches because sizes of largest squares centered at them are critical. Each pair of subsets of two Voronoi edges in a subinterval is then searched for local optimization of the criterion. Let the two subsets be represented by two segments whose endpoints are s_a, t_a and s_b, t_b. Since the size of a square centered on a Voronoi edge linearly increases or decreases and a Voronoi edge is also linear, therefore the size of a square can be parameterized by a parameter α. We define new size functions as

$$size_a(\alpha) = size(s_a) + \frac{\alpha}{r}(size(t_a) - size(s_a))$$

$$size_b(\alpha) = size(s_b) + \frac{\alpha}{r}(size(t_b) - size(s_b))$$

where r is the length of the associated subinterval and $\alpha \in [0, r]$. Clearly, α can be linearly inverted for positions on the segments. Let $\uparrow, \downarrow$ be increasing and decreasing. The locally optimum is obtained as follows (Fig. 4.10).

- If $\alpha \uparrow \Rightarrow size_a(\alpha) \downarrow$ and $size_b(\alpha) \downarrow$, $\alpha = 0$ induces a local optimum.
- If $\alpha \uparrow \Rightarrow size_a(\alpha) \uparrow$ and $size_b(\alpha) \uparrow$, $\alpha = r$ induces a local optimum.
- If $\alpha \uparrow \Rightarrow size_a(\alpha) \downarrow$ and $size_b(\alpha) \uparrow$ and $size_a(0) \leq size_b(0)$, $\alpha = 0$ induces a local optimum.
- If $\alpha \uparrow \Rightarrow size_a(\alpha) \downarrow$ and $size_b(\alpha) \uparrow$ and $size_a(r) \geq size_b(r)$, $\alpha = r$ induces a local optimum.
- If $\alpha \uparrow \Rightarrow size_a(\alpha) \downarrow$ and $size_b(\alpha) \uparrow$ and $size_a(0) > size_b(0)$ and $size_a(r) < size_b(r)$, α causing $size_a(\alpha) = size_b(\alpha)$ induces a local optimum.
- The remaining cases are replica of the last three cases.

All pairs of segments from all subintervals are queried for local optima. Our approach is done when all combinations of pairs of Voronoi edge branches have been explored. The best local optimum is the optimal solution for a finger switching.

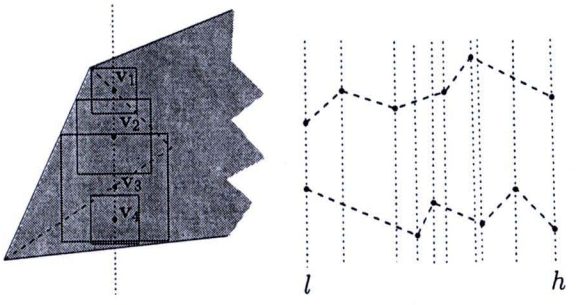

Figure 4.9: The largest square on Voronoi edges

4.7 Experimental Results

We have implemented the regrasp planning for a polygon with a large number of edges based on the switching graph concept. The program is written in C++ programming language. All run times are measured on a PC with a 2.4 GHz CPU.

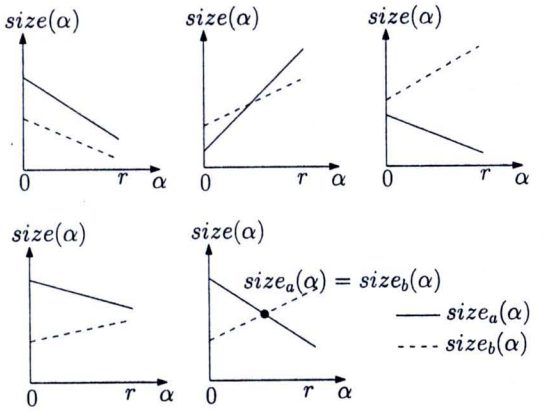

Figure 4.10: Determining a local optimum

Some test polygons with varying number of edges are shown in Fig. 4.11. We also vary the half-angle of the friction cone by 10°, 15° and 20°.

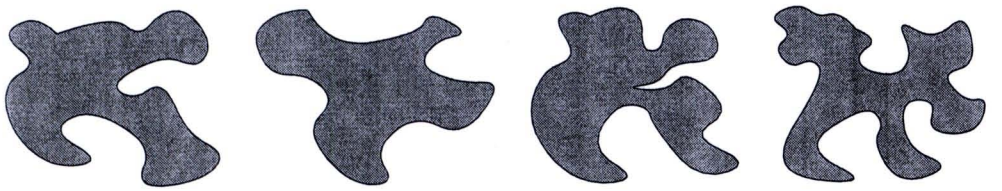

Figure 4.11: Test polygons with number of edges (a) 128 (b) 200 (c) 256 (d) 300

Table 4.1: Switching graph construction of 10° half-angle

Fig.	#node	#edge	#CC	time for nodes	time for edges
(a)	70	162	15	0.11	0.03
(b)	35	79	6	0.20	0.03
(c)	84	222	11	0.25	0.05
(d)	134	400	11	0.30	0.09

The results of switching graph constructions are shown in Table 4.1-4.3. They present for all test objects the number of connected polygons or nodes of switching graphs, edges of switching graphs, the number of connected components of the switching graphs, time spent in node and edge construction in second. The number of nodes of a switching graph depends on the object's shape. An object with more complexity produces more connected polygons. The number of connected components indicates probability to have a path joining any two nodes in the switching graph. The half-angle of the friction cone heavily effects the results. It's clear that larger friction cone induces larger feasible

Table 4.2: Switching graph construction of 15° half-angle

Fig.	#node	#edge	#CC	time for nodes	time for edges
(a)	65	194	5	0.16	0.03
(b)	41	121	4	0.22	0.02
(c)	85	323	3	0.31	0.06
(d)	150	630	4	0.39	0.19

Table 4.3: Switching graph construction of 20° half-angle

Fig.	#node	#edge	#CC	time for nodes	time for edges
(a)	58	230	3	0.17	0.03
(b)	41	156	2	0.27	0.03
(c)	84	424	2	0.38	0.06
(d)	143	782	2	0.45	0.08

grasp sets. This causes sets of force closure grasps to be merged more and connected polygons to be larger. As a result, the number of nodes decreases while the number of edges increases so that the number of connected components of a switching graph decreases. Run times of node constructions depend on areas of merged connected polygons which are inherited from the objects' shapes and the values of the half-angles. For an edge construction, a run time relates to the number of connected polygons.

An example of a regrasp sequence is presented in Fig. 4.12. The sequence is computed using the algorithm described in Section 4.6.1. The dashed lines are lines connecting two contact points which entirely lie in the two associated friction cones.

4.8 Summary

We have proposed a method for solving the regrasp planning problem for a polygon with a large number of edges. A hand using in this chapter is assumed three free-flying fingers. Our method provides complete solutions represented by a graph which allows us to plan a regrasp sequence by using a graph search. The experimental results show the efficiency of our algorithm which merges grasp sets that are adjacent to one another into one connected set. The obtained connected sets are used to construct a switching graph in realtime.

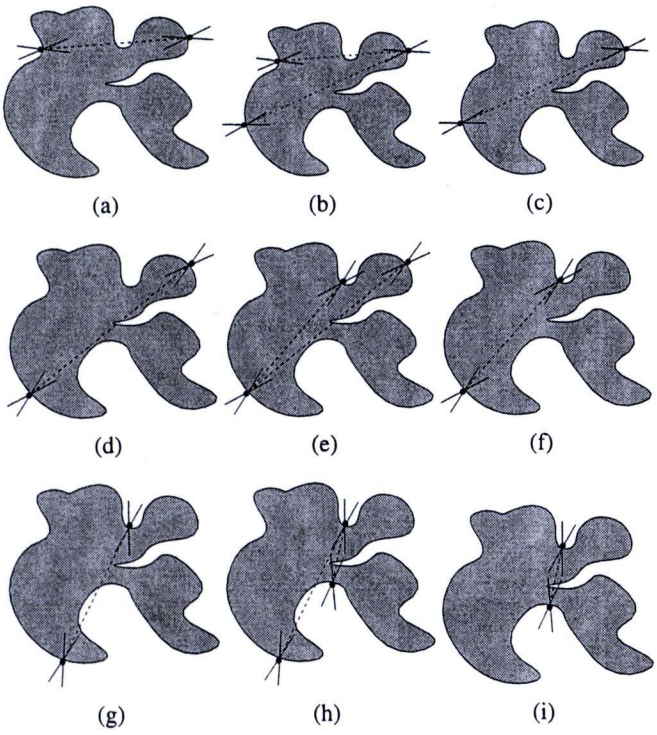

Figure 4.12: A regrasp sequence for the object in Fig. 3.15(c) when the half-angle is 15°.