CHAPTER VI

REGRASP PLANNING FOR A TRIANGULAR-MESH OBJECT

6.1 Introduction

As we have described in the previous chapters, regrasp planning can be formulated as a graph search problem where vertices of the graph represent force closure grasps while an edge connects two grasps that can be changed between each other by simple movement. To construct such graph, it is required to compute all force closure grasps and calculate all possible simple movements. This approach works well when the object is of low complexity, e.g. polyhedron with small number of facets. However, this is not the case for most real world objects, especially when the information of the object is acquired by sensor rather than being nicely modeled by human.

To achieve complete automation, it is vital for the robot to be able to perform sensing by itself. Acquisition of object spatial structure is usually done via range sensing devices such as a laser range finder or stereoscopic cameras. The data obtained through this method usually involve thousands of surface points. This poses a very challenging issue in grasp planning because of the combinatorial explosion of the search space. Given a thousand contact points, the number of all possible 4-finger grasps reach 10^{12}, beyond the address space of an ordinary computer. Considering all possible grasps would take too much time. To provide a trade-off between accuracy and computational simplicity, Goldfeder *et al.* (2007) applied superquadric fitting to parameterize object shapes from the point-cloud input. A grasp planning is performed in the hierarchy of superquadrics from the coarsest to finer approximations. In (Huebner et al., 2008), the minimum volume bounding box approach is used to fit input data by primitive box shapes. The result bounding boxes and data points are iteratively split to yield better box approximations. A grasp planner exploits the approximations as clues to synthesize grasps on arbitrary objects.

While the grasp planning problem for discrete contact points just recently began to receive more attention, the regrasp planning problem in the same setting remains mostly

unexplored. Most existing works in grasp planning for point data approximate the model of the object using only spatial information from the input points. Although this approach may be sufficient for computing a single grasp, it offers no clues on how a regrasp sequence can be obtained. The main contribution of this paper is to present a novel method for organizing the input contact points to facilitate the regrasp planning process. Our underlying idea is to partition the input points based on their wrenches. Contact points that can exert similar wrenches are grouped together. For each group, a representative is selected. Every set of representatives that can form a force closure grasp is then computed and referred to as a representative grasp. Our idea is motivated by a typical regrasping scenario when a supporting finger need to be placed on the object before some finger in a force closure grasp can be lifted off. Obviously, if the supporting finger can exert wrenches similar to those by the finger to be lifted off, it is likely that the object will still remain in force closure after the finger swap. The most important product of the proposed clustering strategy is the roadmap structure that leads to significant reduction of the search space. This structure is a graph that captures how representative grasps can switch among one another via finger swapping. Since each representative grasp roughly describes a different way wrenches can be aligned to form a force closure grasp, the roadmap somewhat approximates the global relationship that describes how force closure grasps can be switched among one another. Of course, an arbitrary force closure grasp may not be a member of the roadmap. Therefore, to utilize the roadmap for regrasp planing from an arbitrary initial grasp to a target grasp, we need regrasp sequences that bring the initial and the target grasps to some grasps in the roadmap. We will present methods for computing such sequences based on the clustering information. Preliminary experimental results show that most regrasp planning problems can be solved within a few seconds or a few minutes by our approach whereas exhaustive search takes about a day or longer.

6.2 Regrasp Planning on Discrete Point Set

In this chapter, we assume that the model of the input object is described by triangular meshes. This will result in a polyhedron with a large number of triangular facets. Each facet is usually very small relative to the entire object. We use the centroid of each facet as a possible contact point. This approach is also adopted in (Roa and Suarez, 2008). Since we consider discrete contact points, finger rolling and finger sliding which require continuous motion at a contact are not permitted. Only finger switching is considered in the planning.

6.2.1 Overview

The first step of regrasp planning is to compute a roadmap of all possible grasping configurations. Since we consider 4-finger grasps, a grasping configuration consists of four contact points. A grasping configuration that satisfies force closure induces a vertex in the roadmap and the configuration is stored in this vertex. An edge joining two vertices exists when two associated grasping configurations can switch to each other. This work assumes five fingers for regrasping. Four fingers are used to securely grasp the object. The remaining finger is used for finger switching. This means that two grasping configurations that have one distinct contact point can perform finger switching which implies that there exists an edge joining the two vertices corresponding to these two grasping configurations.

There are some drawbacks associated with this traditional approach. Suppose there are N contact points to consider. The possible grasping configurations are as many as N^4. For each grasping configuration, there can be as many as $4N$ other grasping configurations that can be reached directly by finger switching. Consequently, size of a roadmap constructed from a large number of contact points could easily become larger than the memory space of an ordinary computer.

To overcome these limitations, we propose to cluster the input contact points into groups. Instead of using all contact points, one contact point is picked from each group to be a representative for constructing the representative-level roadmap. The number of groups is a user-defined variable which indicates trade off between completeness and resource used in the computation. Of course, the representative-level roadmap does not cover grasping configurations consisting of some contact points that are not representatives. A local planner is required to compute a path from such grasping configurations to a grasp in the representative-level roadmap.

6.2.2 Spectral Clustering for Contact Point Set

Before constructing a representative-level roadmap, we apply spectral clustering algorithm to partition the input contact points into meaningful clusters. In spectral clustering, users are free to define how a cluster is meaningful according to their task at hand. However, the definition of meaningful clusters from existing works, mostly in the field of computer graphics, are not related to our regrasp planning problem. Our proposed idea is to define meaningful clusters from similarity of wrenches by means of grasping. Recall the finger switching operation: the remaining finger is placed on the object then one

finger is lifted to change grasping configuration. To maintain force closure, a reasonable heuristic is to ensure that the chosen contact point for the remaining finger and the contact point of the finger to be lifted can produce similar wrenches.

Spectral clustering takes a contact point set as input to compute an affinity matrix which embeds similarities of every pair of contact points. The matrix is solved for eigenvectors corresponding to the k largest eigenvalues. The eigenvectors are then used to determine the clustering of contact points.

6.2.2.1 Affinity Matrix

Affinity matrix contains information that reflects how contact points are grouped according to their applicable forces and torques. Each pair of contact points is measured for pairwise distance. The pairwise distances of all pairs form a matrix that encodes similarity between contact points. An affinity matrix is symmetric and denoted by $A \in \mathbb{R}^{N \times N}$, where $0 \leq a_{ij} \leq 1$ for all contact points i and j. Element a_{ij} encodes the likelihood that contact points i and j can be clustered into the same group. Let $\boldsymbol{f}_i, \boldsymbol{r}_i$ be a unit force perpendicular to the object's surface and the position of ith contact point w.r.t. a reference point $\boldsymbol{o}$. The associated torque is $\boldsymbol{t}_i = (\boldsymbol{r}_i - \boldsymbol{o}) \times \boldsymbol{f}_i$. The wrench generated by $\boldsymbol{f}_i$ at this contact point i is therefore $\boldsymbol{w}_i = (\boldsymbol{f}_i, \boldsymbol{t}_i)$. When friction is assumed, the friction cone at contact point i is approximated using an m-sided pyramid bounded by $\boldsymbol{f}_{i1}, \ldots, \boldsymbol{f}_{im}$. The associated wrench cone is defined by $\boldsymbol{w}_{i1}, \ldots, \boldsymbol{w}_{im}$ and called primitive wrenches.

Since a force closure test in the wrench space considers only the directions of wrenches, the distance function is formulated based on measuring the angle between two wrenches from two distinct contact points, which can be calculated from their inner product. However, the torque component of a wrench depends on the choice of the origin assumed in the torque calculation. We therefore use the centroid of the object as the reference origin. Since any vector $(\boldsymbol{r}_i - \boldsymbol{o})$ is a constant vector, the toque component of the associated wrench is not affected by any rigid transformation applied to the object, i.e., independent from the object's pose.

The proposed pairwise distance between two contact points i and j considers the difference between wrenches that the two contact points can exert. Roughly speaking, we compare the geometries of the two wrench cones. Instead of integrating all differences between all pairs of wrenches from the two cones, an approximation is taken by

comparing only the boundaries of the linearized wrench cones, i.e., angles between the primitive wrenches from the two wrench cones are measured. Each primitive wrench of i is compared with a primitive wrench of j. Obviously, there are many ways to match pairs of primitive wrenches for comparison. We have to select one correspondence that is appropriate for the distance function. Since the linearization of a friction cone is in either a clockwise or counterclockwise order, the result primitive wrenches are arranged in the same order and a reasonable correspondence of the primitive wrenches has to preserve this order. The starting index for a wrench cone in the comparison is however not necessary the first index obtained from the linearization. The indices of the primitive wrenches of a contact point can all be shifted by an integer x while preserving the order. Without loss of generality, we apply the shifting x for the primitive wrenches of the contact point j. The angles between $\boldsymbol{w}_{i1}, \ldots, \boldsymbol{w}_{im}$ and $\boldsymbol{w}_{j(1+x \mod m)}, \ldots, \boldsymbol{w}_{j(m+x \mod m)}$ are measured pair by pair in order (Fig. 6.1). The summation of these angles defines our geometrical difference between these two wrench cones. However, this value is varied by changing x. We imitate the principal of the shortest distance between two bodies in the Euclidean space: by varying x, the minimal summation of angles is used as the pairwise distance.

The difference between two wrench cones is measured as follows

$$M(i,j) = \min \sum_{k=1}^{m} \text{angle}(\boldsymbol{w}_{ik}, \boldsymbol{w}_{j(k+x)})$$

where $0 \leq x \leq m-1$ and $k + x \in \mathbf{Z}_m$. Since value of $M(i,j)$ depends on the number of pyramid's sides, the distance function is normalized as $d(i,j) = M(i,j)/m$.

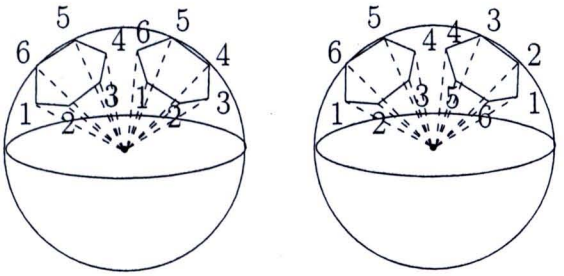

Figure 6.1: Transformation distance: An example of cones in 3D where the numbers show the order of comparison (a) $x = 0$ (b) $x = 2$

The Gaussian similarity function is applied to encode pairwise measures into the affinity matrix. It is formed by an exponential function as

$$a(i,j) = e^{-d(i,j)/2\sigma^2}.$$

Clearly, $0 \leq a_{ij} \leq 1$, and contact points of which their pairwise distance is smaller have larger affinities between them. The issue of choosing σ is neglected. We simply choose σ as the average of all measures, i.e., $\sigma = \frac{1}{N^2} \sum_{1 \leq i,j \leq N} d(i,j)$.

6.2.2.2 Spectral Clustering Algorithm

The spectral clustering algorithm in (Ng et al., 2001) is applied. The eigenvectors of the affinity matrix are used in clustering of contact points. The spectral clustering algorithm from (Ng et al., 2001) is as follows.

i. Compute the affinity matrix A.

ii. Define D to be the diagonal matrix whose (i,j)-element is the sum of A's i-th row, and construct the matrix $L = D^{-1/2}AD^{-1/2}$

iii. Compute the eigenvectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ of L associated with the k largest eigenvalues.

iv. Construct the matrix $V = [\boldsymbol{v}_1 \boldsymbol{v}_2 \ldots \boldsymbol{v}_k] \in \mathbb{R}^{N \times k}$ by stacking the eigenvectors in columns.

v. Form the matrix U from V by normalizing the row sums to have unit length, that is $u_{ij} = v_{ij}/(\sum_k v_{ij}^2)^{1/2}$

vi. Extracting each row of U as a point in $\mathbb{R}^k$, cluster them into clusters $K_1, \ldots, K_k$ by performing k-means algorithm

vii. Assign the original contact point p_i to cluster K_j if and only if row i of the matrix U is assigned to cluster K_j.

After running the clustering algorithm, we need to construct some structures for further uses in regrasp planning. Let the input contact points be stored in a table denoted by T_O. From the above clustering algorithm, Euclidean distances in the affinity matrix are used to partition the input contact points. We then apply the same distance measurement for the successor procedures instead of the pairwise distance measurement proposed in Section 6.2.2.1. The contact point associated with the row vector of U that is closest to the center of K_j is chosen to be the representative of the cluster K_j. We construct a

matrix E to store Euclidean distances between all row vectors of U and the centers of all clusters. A matrix F contains Euclidean distances among all row vectors. A table T_R is constructed to store all representatives.

6.2.3 Constructing Representative-Level Roadmap

We are now ready to compute a roadmap for the representatives. All vertices are constructed by checking force closure grasp for every four representatives in T_R. We call such grasping configurations *representative grasping configurations*. Each representative grasping configuration satisfying force closure is assigned to a vertex. The number of vertices in a switching graph is equal to the number of grasps found in force closure checking. In this chapter, we apply the algorithm of (Niparnan and Sudsang, 2007) for fast filtering grasps that do not satisfy the necessary condition. Grasps that pass the filter are then verified for force closure by applying the algorithm of (Zhu and Wang, 2003a). An edge joining two vertices of which corresponding representative grasping configurations can switch to each other can be easily computed by checking the number of common contact points between the two corresponding configurations. Since 4-finger force closure grasps are considered, two representative grasping configurations can switch to each other when they share three common contact points. The computed representative-level roadmap is denoted by R.

6.2.4 Planning Regrasp Sequence

The regrasp sequence planning process is splitted into two level search: in representative-level roadmap and in local roadmap. In the previous section, the construction of representative-level roadmap has been described. A regrasp sequence acquired from the representative-level roadmap contains only configurations consisting of contact points in T_R which is a subset of of the original contact point set T_O. This means that traversal in the representative-level roadmap does not cover grasping configurations that consist of some contact points in $T_O \backslash T_R$. Therefore, for arbitrary initial and goal grasps, the regrasp planner has to find regrasp sequences that link both grasps to some grasps in the representative-level roadmap.

An intuitive way is to find a path from the initial grasp or the goal grasp to their representative grasp, i.e., the grasp that consists of the representatives of all contact points forming the initial grasp or the goal grasp. The planner performs Algorithm 1 which exploits a heuristic of similarity in a cluster to determine a vertex in the representative-level

roadmap that the initial grasp or the goal grasp should be switched to. Let g be the initial grasping configuration or the goal grasping configuration that consists of contact points p_a, p_b, p_c and p_d. Let us denote by $p_{a'}, p_{b'}, p_{c'}$ and $p_{d'}$ the representatives of the clusters that respectively contain p_a, p_b, p_c and p_d. Also denote by g' the representative grasping configuration consisting of $p_{a'}, p_{b'}, p_{c'}$ and $p_{d'}$. If g' does not achieve force closure, i.e. the vertex defining g' is not in the representative-level roadmap R, the planner then performs Algorithm 2. Otherwise, a regrasp sequence from g to g' is planned as follows.

For each contact point $p_{i'}$ of g' ($i = a, \ldots, d$), all contact points in T_O that are in the same cluster of $p_{i'}$ are considered. We exploit nearness among $p_i, p_{i'}$ and contact points in the cluster to limit search space in local planning. The distance $F_{ii'}$ between p_i and $p_{i'}$ is queried from F. A contact point j in the cluster that induces $F_{ij} \leq F_{ii'}$ and $F_{i'j} \leq F_{ii'}$ is copied to a set S_i. We then compute all possible 4-finger force closure grasps such that the first, second, third and forth contact points are picked from $S_{a'}, S_{b'}, S_{c'}$ and $S_{d'}$, respectively. A local roadmap is then constructed such that each of its vertices represents each force closure grasp mentioned above, and each of its edges joins two vertices representing two grasping configurations with three common contact points (finger switching is possible). With a local roadmap, any graph search can be used to retrieve a path from the vertex representing g' to the vertex representing g. If no such path can be found, the planner invokes Algorithm 2.

Algorithm 1

```
1: Determine p_a', p_b', p_c', p_d' and g'
2: if g' ∉ R then
3:    Algorithm 2
4: else
5:    Determine S_a', S_b', S_c', S_d'
6:    L = ConstructRoadmap(S_a', S_b', S_c', S_d')
7:    if path = FindPath(L, g, g') then
8:       return path
9:    else
10:      Algorithm 2
11:   end if
12: end if
```

Algorithm 2 again exploits the information of clusters. It is invoked when g' does not achieve force closure or Algorithm 1 fails to find a regrasp sequence from g to g'. The underlying idea of this algorithm is to find another appropriate representative grasping configuration, which the given grasping configuration will change to, that satisfies force closure and has one different contact point from g'. Since the new representative grasping

configuration does not consist of all representatives of the given grasping configuration, to apply the same local planning strategy, we have to relocate one contact point of the given grasping configuration from its cluster to the cluster of the different representative. A local planning, is then performed among one changed cluster and three unchanged clusters, which guarantees that g and the new representative grasping configuration are force closure. However, this method perturbs the local properties of the changed clusters. To minimize the effect of this transfer and to maximize the use of local property, this algorithm aims to transfer one contact point in g to its second nearest cluster. The procedure *FindNearestCluster* begins with finding vertices in R of which the representative grasping configuration f' has one distinct representative from g'. Let $p_{x'}$ and $p_{y'}$ be the distinct representative of g' and f'. The associated contact point of $p_{x'}$ is denoted by p_x. We query the matrix E for the distance between p_x and the center of cluster $K_{y'}$ of $p_{y'}$. All representative grasping configurations in R having one distinct representative from g' are used to query the distances. The pair that induces the shortest distance between them is selected for locality reason. Now we redefine some notations to understand the pseudocode. Let the selected contact point be p_x, its cluster be $K_{x'}$, its second nearest cluster be $K_{y'}$ and the representative grasping configuration possessing the cluster $K_{y'}$ be f'. This procedure returns the variable X which consists of p_x, $K_{x'}$, $K_{y'}$ and f'. If X is determined, we then temporarily add p_x into $K_{y'}$. Let f' consist of p_q, p_r, p_s, p_t. We perform a local roadmap construction as applied in Algorithm 1 and find a path between g and f' which both are members of the local roadmap.

Algorithm 2

```
1: Determine p_{a'}, p_{b'}, p_{c'}, p_{d'}
2: X = FindNearestCluster(p_{a'}, p_{b'}, p_{c'}, p_{d'})
3: if X is not determined then
4:    Algorithm 3
5: else
6:    Add p_x into K_{y'}
7:    Determine S_q, S_r, S_s, S_t
8:    L = ConstructRoadmap(S_q, S_r, S_s, S_t)
9:    if path = FindPath(L, g, f') then
10:      Remove p_x from K_{y'}
11:      return path
12:   else
13:      Remove p_x from K_{y'}
14:      Algorithm 3
15:   end if
16: end if
```

If finding a path from g to f' still does not achieve, Algorithm 3 is applied by updat-

ing the representative-level roadmap. The contact points p_a, p_b, p_c, p_d are now considered as representatives. Let T_G be a set of the contact points $\{p_a, p_b, p_c, p_d\}$. All grasping configurations partially consisting of some contact points in T_G are verified for force closure condition and reported as new vertices in R. New edges are computed among the new vertices and between the new vertices and the recent vertices of R to complete updating R. Clearly, g is associated with a vertex in R. Therefore, a path from g to any grasp in R can be computed by a graph search.

Algorithm 3

1: $T_G = \{p_a, p_b, p_c, p_d\}$
2: $T_A = T_R \cup T_G$
3: $L = \text{ConstructRoadmap}(T_A, T_A, T_A, T_G)$
4: Update R by adding L and linking L to R

In conclusion, the overall algorithm firstly clusters the input contact points. Then a representative-level roadmap R is constructed from the representatives. To query a regrasp sequence from an initial grasping configuration g to a goal grasping configuration h, they have to be linked with R by using Algorithm 1 and 2. These algorithms find a path between g to a vertex in R and so on for h. If the algorithms fail to report a path, Algorithm 3 is performed by adding g or h into R. Finally, a graph search is then applied to find a path connecting two grasps in R.

6.3 Experiments and Results

The test objects are shown in Fig. 6.2. They are simplified and modeled with about 500 and 1000 triangular meshes. The half angle of friction cones is 10°. All objects are clustered into 30, 50 and 70 groups for regrasp planning. All run times are measured on a PC with a 2.4 GHz CPU. Examples of regrasping sequences are presented in Fig. 6.3 and 6.4.

Table 6.1-6.6 show the result from the construction of representative-level roadmaps. The result presents for each test object the number of force closure grasp vertices of the resulting representative-level roadmap, time spent in the construction and the number of connected components of the roadmap. The numbers of vertices of roadmaps depend heavily on the numbers of groups. Another factor is the object's shape. The objects in Fig. 6.2(a) and (f) result in more vertices than the others due to their sphere-like shapes which generally yield more force closure grasps. Note that the number of vertices is much smaller than the total number of force closure grasps that can be formed by the input contact points (up to millions for each test object). Small number of connected components

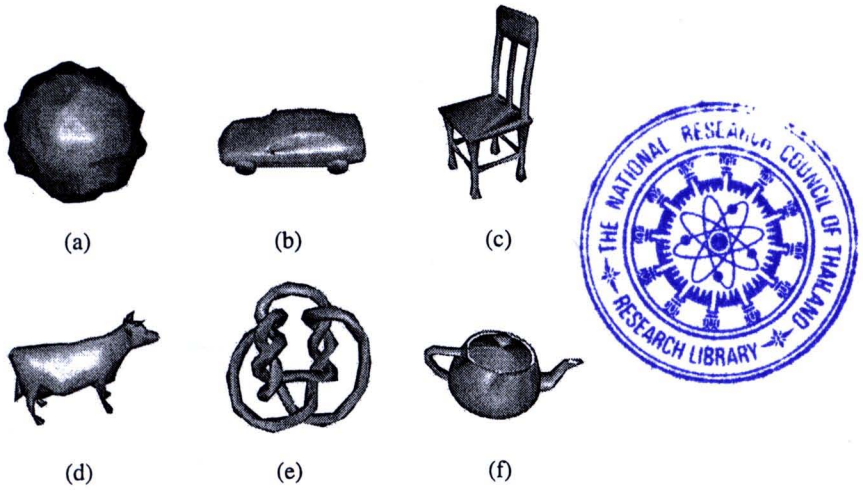

Figure 6.2: Test objects

indicates high probability to have a path joining any two vertices in the roadmap.

The tables also present the result from querying regrasping sequences. We randomly pick 30 pairs of force closure grasps to serve as the initial and target grasps for each query. Although the number of connected components of the representative-level roadmaps are quite large for some objects, successful regrasping sequences can be found for most pairs of grasps. This implies that the majority of grasps lie in the same connected component. The success rates tend to increase when the numbers of groups are increased as we can see from the situations when the numbers of groups are changed from 30 to 50. Minimum, maximum and average querying times are also shown in the tables. The numbers of vertices and the numbers of meshes affect computational times used to find a path connecting two grasps. The former effects when Algorithm 2, 3 and representative-level planner are executed whereas the later effects when all local planners are executed. From out experiments, minimum times are spent when both initial and target grasps can connect to representative-level roadmaps by Algorithm 1. Maximum times mostly occur when Algorithm 3 is provoked. However, when the number of groups is quite low such as 30 groups of 1,000 triangles, the average number of members in a group is greater than the number of groups so that computing a local roadmap by Algorithm 1 and 2 takes more computational time than provoking Algorithm 3. As discussed above, our approach provides trade off between completeness and resource used in the computation. From the results, the numbers of vertices and the success rates depend on the numbers of groups. Less number of groups provides faster computational times but variations of grasps and success rates decrease. This is reasonable for an ordinary personal computer to be used

in the regrasp planning problem.

For more insight of the local planners, the results of local planning are shown in Table 6.7 - 6.12. For each setting, we sample 10,000 force closure grasps. For Algorithm 1, each sampled grasp is verified whether the associated representatives form a force closure grasp. The numbers of force closure grasps found are shown in percent. For Algorithm 2, the procedure *FindNearestCluster* is executed for each sampled grasp. A grasp passes the verification if the variable X is determined. Then 30 grasps are randomly picked from the sampled grasps that pass the verification for each algorithm. Each grasp is planned for a regrasping sequence that joins it to the representative-level roadmap. The results list the numbers of grasps (out of 30) for which such sequence are found. To measure the efficiency of Algorithm 3, 30 sampled grasps that do not pass the verifications of Algorithm 1 and 2 are randomly picked. We then perform Algorithm 3 for each of them and verify whether it connects with part of the representative-level roadmap that exists before the update by Algorithm 3.

The results show the low passing rate of the verification by Algorithm 1 for some objects. Most grasps that pass the verification however can be connected to the representative-level roadmap. Algorithm 2 appears to be more effective than Algorithm 1. The passing rates for the verification of Algorithm 2 are significantly higher than those of Algorithm 1, (with slightly fewer regrasping sequences found on average). However, Algorithm 1 is still needed because of its considerably higher verification speed. By varying the numbers of groups, in most cases, the results show that the execution rates of Algorithm 1 and 2 increase and the numbers of achievements of path connection also tend to increase when the number of group is increased. Although applying Algorithm 3 can connect most grasps to the representative-level roadmap, it takes much longer to update the representative-level roadmap than to run both Algorithms 1 and 2 (the computational times are not shown here). However, computational time of Algorithm 3 will decrease when the number of groups is decreased because the number of vertices of the representative-level roadmap decreases and updating the representative-level roadmap takes less computational time.

6.4 Summary

We have proposed a new approach to solve the regrasp planning problem. Existing methods typically solve the problem only for a complete solution. However, using such approach on real world data such as highly complex contact points is next to impossible.

Table 6.1: Result of 500 mesh objects clustered into 30 clusters

Fig.	#Vert	#CC	time(s)	#Con	Search time(s)		
					min	max	avg
(a)	2156	1	3.04	30	0.21	3.46	0.94
(b)	1101	1	1.60	30	0.14	3.27	1.28
(c)	386	7	0.66	28	0.16	2.06	0.94
(d)	341	13	0.61	26	0.12	2.98	1.01
(e)	374	14	0.65	27	0.15	1.82	0.76
(f)	2071	1	2.84	30	0.31	7.16	1.80

Table 6.2: Result of 500 mesh objects clustered into 50 clusters

Fig.	#Vert	#CC	time(s)	#Con	Search time(s)		
					min	max	avg
(a)	15928	1	22.67	30	1.62	19.92	2.47
(b)	7752	2	10.89	30	0.79	17.32	3.12
(c)	2970	12	4.98	30	0.32	7.42	1.82
(d)	2281	12	4.25	29	0.23	6.16	1.92
(e)	2479	31	4.39	28	0.24	7.77	2.69
(f)	10993	2	15.54	30	1.13	15.85	3.25

This is due to combinatorial explosion of the search space. Our approach provides trade off between completeness and the resource used in the computation. It clusters the input using spectral clustering. The representatives from all clusters are used to constrain possible search space. The underlying idea is similar to that of the classical motion planning problem. We construct a partial solution, called a representative-level roadmap. This allows the original problem to be divided into three parts: planning regrasp sequence from the starting grasp to the roadmap, planning regrasp sequence in the roadmap and, finally, planning regrasp sequence from the roadmap to the target grasp. Since the set of representatives contains much fewer contact points, solving the problem on the roadmap is much less complex. Nevertheless, this is achieved at the cost of completeness.

Table 6.3: Result of 500 mesh objects clustered into 70 clusters

Fig.	#Vert	#CC	time(s)	#Con	Search time(s)		
					min	max	avg
(a)	69898	1	100.55	30	10.88	76.42	13.42
(b)	30434	1	44.58	30	4.02	23.48	4.84
(c)	8562	9	16.28	30	0.67	10.06	4.07
(d)	10322	10	19.03	30	0.73	22.41	3.79
(e)	7683	23	14.87	30	0.45	15.29	4.20
(f)	51675	3	73.17	30	8.49	160.31	14.02

Table 6.4: Result of 1000 mesh objects clustered into 30 clusters

Fig.	#Vert	#CC	time(s)	#Con	Search time(s)		
					min	max	avg
(a)	2813	1	3.91	30	1.17	19.50	7.26
(b)	754	1	1.13	29	0.89	25.13	7.34
(c)	846	2	1.29	30	0.16	11.45	3.33
(d)	321	14	0.59	24	0.42	6.49	2.68
(e)	278	12	0.51	28	0.20	7.45	1.78
(f)	1137	3	1.66	30	1.15	29.11	7.67

Table 6.5: Result of 1000 mesh objects clustered into 50 clusters

Fig.	#Vert	#CC	time(s)	#Con	Search time(s)		
					min	max	avg
(a)	17056	1	24.25	30	2.19	42.75	10.56
(b)	6249	1	9.02	30	0.85	11.95	3.28
(c)	3747	7	6.02	30	0.40	6.01	2.36
(d)	1969	11	3.75	29	0.20	7.40	2.29
(e)	3770	12	6.09	30	0.43	9.67	2.79
(f)	14580	2	20.37	30	2.04	21.25	5.65

Table 6.6: Result of 1000 mesh objects clustered into 70 clusters

Fig.	#Vert	#CC	time(s)	#Con	Search time(s)		
					min	max	avg
(a)	79444	1	110.41	30	15.82	18.16	16.83
(b)	25064	2	36.04	30	3.62	28.40	7.37
(c)	9288	19	17.08	30	0.89	12.99	4.19
(d)	11352	11	20.19	30	1.01	23.53	5.29
(e)	8258	23	15.33	29	0.76	10.12	4.28
(f)	46386	3	64.47	30	7.88	164.11	22.22

Table 6.7: Result of local planning of 500 mesh objects clustered into 30 clusters

Fig.	Algorithm 1		Algorithm 2		Algorithm 3
	%run	#connect	%run	#connect	#connect
(a)	24.58	30	87.75	29	30
(b)	25.94	29	81.23	30	30
(c)	8.90	22	72.47	24	29
(d)	7.57	26	70.49	26	29
(e)	6.06	25	63.01	23	28
(f)	37.88	30	84.77	30	30

Table 6.8: Result of local planning of 500 mesh objects clustered into 50 clusters

Fig.	Algorithm 1		Algorithm 2		Algorithm 3
	%run	#connect	%run	#connect	#connect
(a)	30.21	30	93.24	30	30
(b)	34.88	28	87.45	29	30
(c)	15.24	27	84.69	24	29
(d)	9.62	28	81.14	24	30
(e)	9.62	27	79.12	21	29
(f)	32.32	30	90.33	30	30

Table 6.9: Result of local planning of 500 mesh objects clustered into 70 clusters

Fig.	Algorithm 1		Algorithm 2		Algorithm 3
	%run	#connect	%run	#connect	#connect
(a)	40.59	30	95.08	28	30
(b)	47.41	29	92.49	24	30
(c)	14.65	27	89.13	20	30
(d)	14.90	26	91.72	23	30
(e)	10.25	27	87.35	19	30
(f)	47.54	30	93.64	30	30

Table 6.10: Result of local planning of 1000 mesh objects clustered into 30 clusters

Fig.	Algorithm 1		Algorithm 2		Algorithm 3
	%run	#connect	%run	#connect	#connect
(a)	30.28	30	86.91	30	30
(b)	24.49	30	80.42	28	30
(c)	15.92	26	79.84	24	29
(d)	6.46	28	68.67	28	30
(e)	4.01	25	60.50	27	29
(f)	25.27	30	83.20	30	30

Table 6.11: Result of local planning of 1000 mesh objects clustered into 50 clusters

Fig.	Algorithm 1		Algorithm 2		Algorithm 3
	%run	#connect	%run	#connect	#connect
(a)	30.01	30	92.79	30	30
(b)	38.01	30	89.28	30	30
(c)	17.18	25	84.08	22	30
(d)	9.05	28	78.80	30	30
(e)	10.59	27	85.08	27	30
(f)	43.17	30	92.03	29	30

Table 6.12: Result of local planning of 1000 mesh objects clustered into 70 clusters

Fig.	Algorithm 1		Algorithm 2		Algorithm 3
	%run	#connect	%run	#connect	#connect
(a)	38.39	30	95.07	30	30
(b)	40.72	29	91.80	28	30
(c)	12.89	28	89.44	23	30
(d)	16.06	28	90.74	27	30
(e)	9.23	27	86.34	28	30
(f)	49.28	29	93.77	30	30

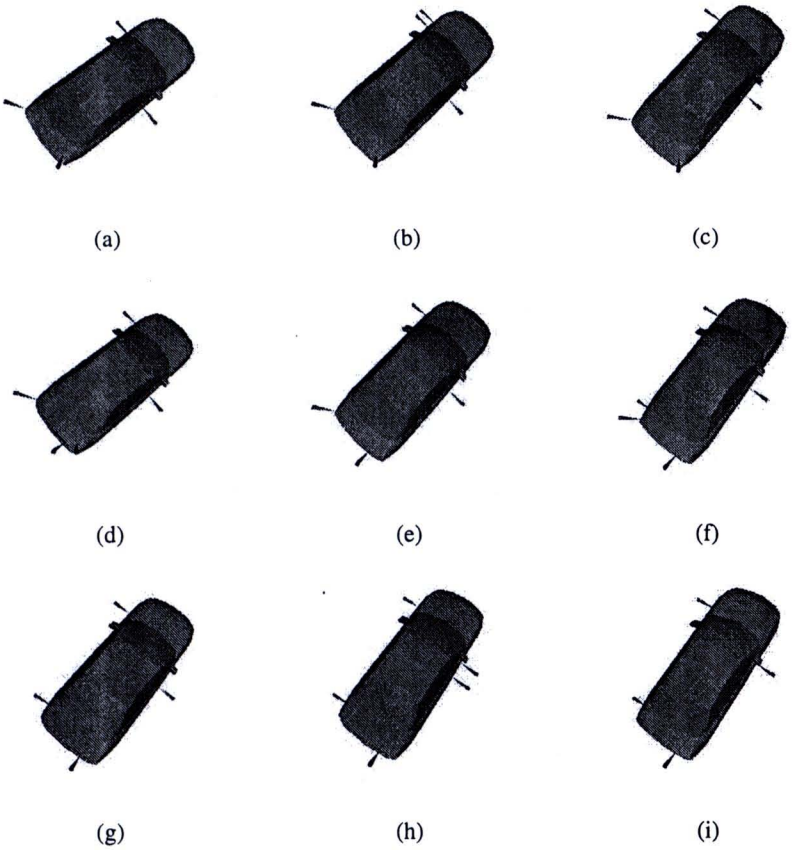

Figure 6.3: A regrasping sequence for the object in Fig. 6.2(b) with 500 triangles clustered into 50 groups

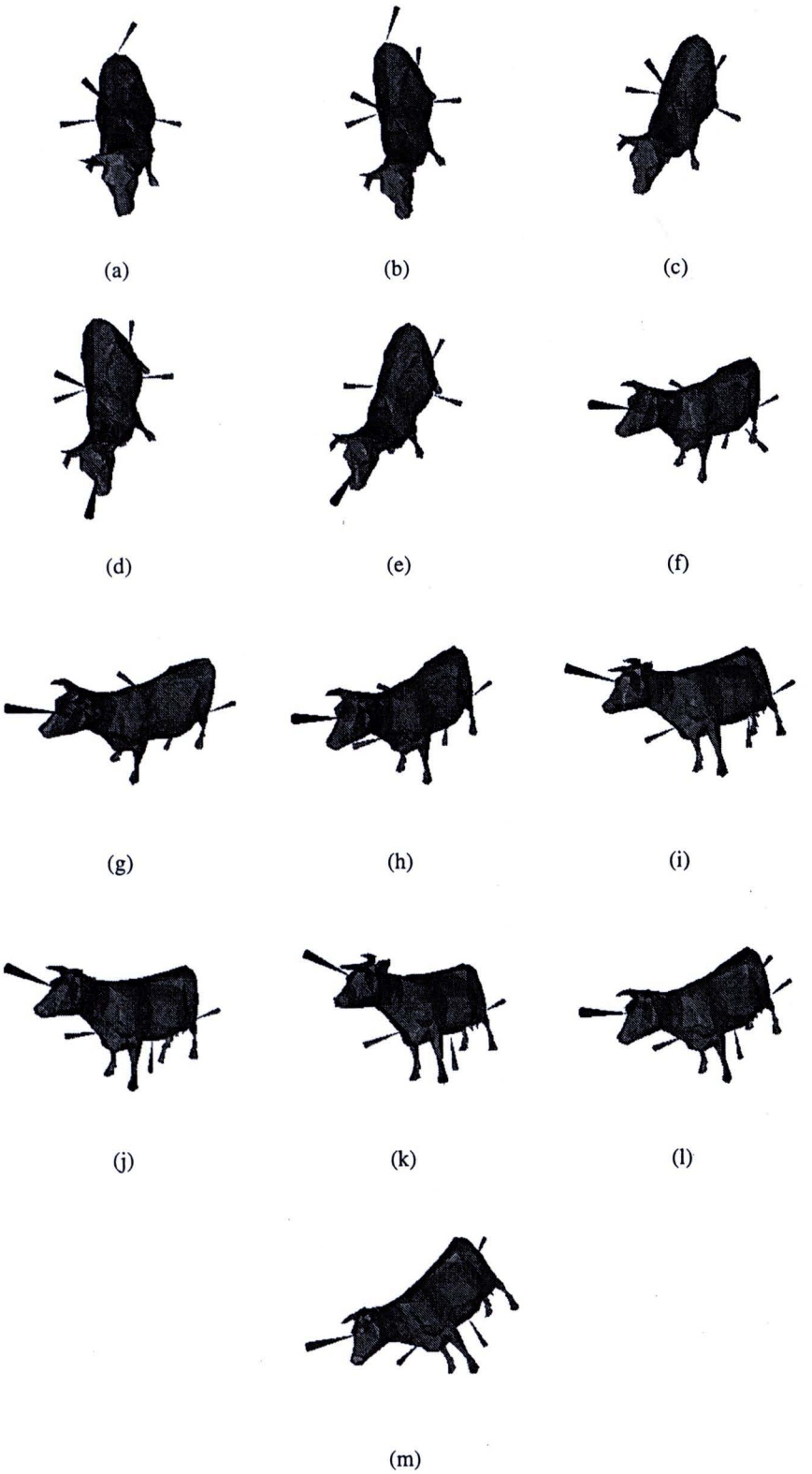

Figure 6.4: A regrasping sequence for the object in Fig. 6.2(d) with 1000 triangles clustered into 30 groups