CHAPTER III

ALGORITHM FOR THREE FINGER FORCE CLOSURE ASSERTION

3.1 Introduction

The main contribution of this chapter is an efficient algorithm for testing whether a three finger frictional grasp achieves force closure. The algorithm is based on a force closure condition that is derived from the graphical analysis of frictional contact forces presented over a decade ago by Brost and Mason (1989). Interestingly, the proposed algorithm leads to an efficient implementation, in speed and accuracy, when compared with other force closure tests recently proposed in the literature. Although the underlying idea for force closure test in (Brost and Mason, 1989) has been available for years, its primary use was mostly for intuition and working problems by hand. Its computational implementation and performance report has never been presented. Part of the problem comes from the lack of performance comparison to existing methods when new force closure tests are introduced. Our presented algorithm relies on representing wrenches associated with the input grasp using a 2D representation. With this representation, we transform force closure test into the problem of computing whether a line segment intersects a convex polygon of at most four vertices. Although the underlying idea is simple, the geometric nature of the problem requires careful consideration to the implementation detail which is often overlooked and sometimes leads to unsatisfactory performance. Since efficient implementation is our major concern, our results are presented along with a thorough comparison, in terms of computation speed and robustness, with existing methods.

We pay great attention to the accuracy of the results by comparing them to reference results computed using exact arithmetic. This comparison and the accompanied analysis reveal several interesting issues, including incompleteness and errors, regarding existing force closure tests. To the best of our knowledge, this presentation provides the first performance comparison on available force closure tests that has been conducted at this level of thoroughness. We hope that this comparative study would help stimulate development of more efficient methods and facilitate the readers in choosing a force closure test or implementing their own.

This chapter is organized as follows. Our proposed algorithm is presented in Section 3.2. Section 3.3 describes existing tests in brief and Section 3.3.2 presents the comparison between these tests and our method. Finally, Section 3.4 concludes the chapter.

3.2 The Algorithm

In this section, we present a detailed algorithm for three-fingered frictional force closure test. The algorithm demonstrates how the concept described in the previous section can be efficiently implemented.

Given a grasp defined by three contact points together with the corresponding contact normal vectors and the coefficient of friction, the following algorithm determines whether the grasp achieves force closure.

1. Choose one of the contact points as the origin and compute the two primitive wrenches of this contact point according to the chosen origin. Denote these two wrenches by ω and v.
2. According to the origin chosen in step 1, compute the four primitive wrenches of the remaining two contact points. If it is not true that some of these primitive wrenches have positive torques and some have negative torques, report that the grasp does not achieve force closure and halt.
3. Compute the positive and negative dual points on the plane $\tau = 1$ of the primitive wrenches obtained from step 2. The dual point of a wrench with zero torque, which is a point at infinity in the direction of the wrench, is regarded as a positive dual point.
4. Depending on the number of positive and negative dual points from step 3, apply one of the following steps.
    - Two positive and two negative dual points: let the dual points of ω and v be positive dual points.
    - One positive and three negative dual points, or one negative and three positive dual points: let the dual point of ω be a positive dual point and the dual point of $-v$ be a negative dual point.
5. It is guaranteed by step 4 that there must be four dual points of the same sign on the plane $\tau = 1$. Compute the convex hull of these four dual points and detect if the convex hull intersects the line segment formed by the remaining two dual points. If the intersection contains a point in the relative interior of the convex hull and the line segment, report that the grasp achieves force closure.

This algorithm is essentially the convex hull based method of Brost and Mason (1989) that is optimized for three-fingered grasps. The optimization stems from the choice of origin made in step 1. As the two primitive wrenches of the chosen contact point possess zero torque, their dual points which are points at infinity are obtained without any computation. Saving four out of twelve floating-point divisions in dual point computation practically yields considerable speedup. Additionally, the freedom to map the primitive wrenches ω and v to positive or negative dual points is exploited in step 4 to ensure that four dual points are of one sign (either positive or negative) and the remaining two dual points are of the other sign. This grouping allows us to apply our brute-force convex hull computation for four input points which is more efficient in this case than generic convex hull algorithms. The optimization described above, of course, comes at an increased complexity: dual points at infinity have to be explicitly dealt with. Fortunately, our convex hull computation and the intersection detection (step 5) are based on the cross product operation which can be easily extended to handle points at infinity. In the remainder of this section, this extension will be described along with our four-point convex hull algorithm and the intersection detection.

3.2.1 The Cross Product

Our convex hull algorithm for four input points and the detection of intersection between a line segment and a polygon (to be described shortly) rely critically on a test to determine for three points, say $\boldsymbol{a}, \boldsymbol{b}$, and $\boldsymbol{c}$, whether $\boldsymbol{a}$ lies on the directed line $\boldsymbol{bc}$, or on which side of it. The test can be performed by inspecting the sign of the cross product $\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}}$. A positive, negative, or zero outcome indicate whether $\boldsymbol{a}$ lies, respectively, on the left, on the right, or exactly on the directed line $\boldsymbol{bc}$. For convenience, let $\operatorname{sgn}(x)$ be $-1, 0$ or $+1$ when x is respectively negative, zero or positive. To describe how to compute $\operatorname{sgn}(\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}})$ when $\boldsymbol{a}, \boldsymbol{b}$, or $\boldsymbol{c}$ could be points at infinity, some properties of points at infinity are needed.

In Projective Geometry (Bix, 1994), a point at infinity is an intersection of parallel lines. Informally speaking, the farthest point of a ray is a point at infinity. Unlike an ordinary point which is defined by its position, a point at infinity $\boldsymbol{p}$ is defined by its direction vector, denoted $\boldsymbol{w}(\boldsymbol{p})$; rays pointing in the same direction meet at the same point at infinity (defined by the direction of the rays). We can therefore infer that a vector from an ordinary point to a point at infinity is in the same direction as the vector that defines the point at infinity. All points at infinity are defined to lie on the same line called the line at infinity. When traversing the line at infinity in the counterclockwise direction, all ordinary points lie on the left. With the properties described above, the following rules are added for computing $\operatorname{sgn}(\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}})$.

1. When only $\boldsymbol{a}$ is a point at infinity: Since $\overrightarrow{\boldsymbol{ba}}$ is parallel with $\boldsymbol{w}(\boldsymbol{a})$, $\operatorname{sgn}(\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}}) = \operatorname{sgn}(\overrightarrow{\boldsymbol{bc}} \times$

$w(a))$.

2. When c is at infinity but b is not: Since $\overrightarrow{bc}$ is parallel with $w(c)$, $\text{sgn}(\overrightarrow{bc} \times \overrightarrow{ba}) = \text{sgn}(w(c) \times \overrightarrow{ba})$. If a is also at infinity, $\text{sgn}(\overrightarrow{bc} \times \overrightarrow{ba}) = \text{sgn}(w(c) \times w(a))$.

3. When b is at infinity but c is not: Since $\text{sgn}(\overrightarrow{bc} \times \overrightarrow{ba}) = -\text{sgn}(\overrightarrow{cb} \times \overrightarrow{ca})$ and $\overrightarrow{cb}$ is parallel with $w(b)$, $\text{sgn}(\overrightarrow{bc} \times \overrightarrow{ba}) = -\text{sgn}(w(b) \times \overrightarrow{ca})$. If a is also at infinity, $\text{sgn}(\overrightarrow{bc} \times \overrightarrow{ba}) = -\text{sgn}(w(b) \times w(a))$.

4. When b and c are points at infinity: $\text{sgn}(\overrightarrow{bc} \times \overrightarrow{ba}) = \text{sgn}(w(b) \times w(c))$. If a is also at infinity, $\text{sgn}(\overrightarrow{bc} \times \overrightarrow{ba}) = 0$.

3.2.2 Computing the Convex Hull of Four Points

In fact, the Graham's scan (Goodman and O'Rourke, 1997) algorithm can be used here by replacing the standard cross product with the extended version described in Section 3.2.1. However, for the case of four input points, the following brute-force method demands less overhead and therefore more efficient.

Let the four input points be arbitrarily denoted by p_1, p_2, q_1 and q_2 and let us define segment $p = \overline{p_1 p_2}$ and segment $q = \overline{q_1 q_2}$. The underlying idea of our convex hull algorithm comes from observing how p and q lie with respect to each other. Note that the output convex hull is represented here as a sequence of points arranged counterclockwise. Such sequence can be shifted freely, i.e., the sequences $(c_1, c_2, c_3, c_4), (c_2, c_3, c_4, c_1)$ and (c_4, c_1, c_2, c_3) represent the same convex hull.

To characterize how segments p and q lie with respect to each other, let us define $t_i^p = \text{sgn}(\overrightarrow{p_1 p_2} \times \overrightarrow{p_1 q_i}), i = 1, 2$, and $t_i^q = \text{sgn}(\overrightarrow{q_1 q_2} \times \overrightarrow{q_1 p_i}), i = 1, 2$. The sign of cross product is computed as described in Section 3.2.1 so that input points at infinity can be handled. From the definition, t_i^p (resp. t_i^q) takes on the value of $0, +1$ or -1 when q_i (resp. p_i) is exactly on, on the left or on the right of segment p (resp. segment q). With this setting, arrangement of segments p and q can be classified into one of the following three cases.

3.2.2.1 When $t_1^p = t_2^p$ and $t_1^q = t_2^q$

This case is illustrated in Figure 3.1a. Obviously, the resulting convex hull is a concatenation of the endpoints of segments p and q, i.e., (p_k, p_l, q_m, q_n) where $k \neq l, m \neq n$ and $k, l, m, n \in \{1, 2\}$. Assignment of k, l, m and n has to be made such that (p_k, p_l, q_m, q_n) is in the counterclockwise order. It is easy to verify that this requirement can be satisfied by the fol-

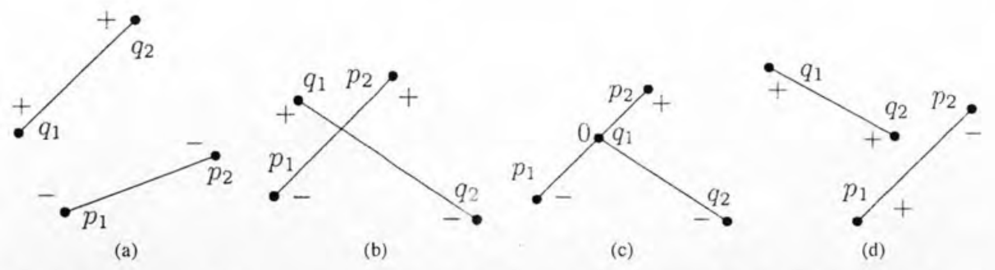

Figure 3.1: Arrangements of two segments forming a convex hull. The sign $+$ and $-$ indicates the value of t_i^p and t_i^q. The plus sign $(+)$ indicates that the point is on the left of the other segment while the minus sign means that the point is on the right.

lowing rule: When $t_1^q = t_2^q = +1$, we set $k = 1$ and $l = 2$, otherwise we set $k = 2$ and $l = 1$, and when $t_1^p = t_2^p = +1$, we set $m = 1$ and $n = 2$, otherwise we set $m = 2$ and $n = 1$.

3.2.2.2 When $t_1^p \neq t_2^p$ and $t_1^q \neq t_2^q$

An example of this case is shown in Figure 3.1b. Since the condition indicates that segments p and q cross each other, the resulting convex hull is an interwoven sequence of the endpoints from the two segments, i.e., $(\boldsymbol{p}_1, \boldsymbol{q}_m, \boldsymbol{p}_2, \boldsymbol{q}_n)$ where $m \neq n$ and $m, n \in \{1, 2\}$. Assignment of m and n need to cause the sequence to follow the counterclockwise order. This requires m and n to be chosen such that $t_m^p = -1$ *or* $t_n^p = +1$ (either t_m^p or t_n^p is allowed to be zero in case of three collinear points; see Figure 3.1c for example).

3.2.2.3 When $t_1^p = t_2^p$ and $t_1^q \neq t_2^q$, or when $t_1^p \neq t_2^p$ and $t_1^q = t_2^q$

As illustrated in Figure 3.1d, the resulting convex hull consists of only three points; one input point is discarded. Let us describe only the case in which $t_1^p = t_2^p$ and $t_1^q \neq t_2^q$ (the other case is treated likewise). In this case, either $\boldsymbol{q}_1$ or $\boldsymbol{q}_2$ (not both) has to be discarded because it lies inside the convex hull. If $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ are on the left side of segment p $(t_1^p = t_2^p = +1)$, the convex hull can be written in the form $(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{q}_m), m \in \{1, 2\}$. When no three points are collinear, it is easy to verify that the value of m must be chosen such that $t_m^q = -1$. However, when some three points are collinear, it is possible that $t_1^q \neq -1$ and $t_2^q \neq -1$. When this occurs, choose m such that $t_m^q = 0$ in order to correctly eliminate the redundant point. For the case that $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ are on the right side of segment p, we will have the resulting convex hull $(\boldsymbol{p}_2, \boldsymbol{p}_1, \boldsymbol{q}_m)$ where the value of m must be chosen such that $t_m^q = +1$ if possible, otherwise choose m such that $t_m^q = 0$.

A special case worth attention is when three of the input points, say $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ are at infinity. They are by definition collinear. As shown in Figure 3.2a, to safely discard $\boldsymbol{b}$, it is necessary that $\boldsymbol{a}$ and $\boldsymbol{c}$ are on different side of $\boldsymbol{b}$, i.e., $\operatorname{sgn}(\boldsymbol{w}(\boldsymbol{b}) \times \boldsymbol{w}(\boldsymbol{c})) = -\operatorname{sgn}(\boldsymbol{w}(\boldsymbol{b}) \times \boldsymbol{w}(\boldsymbol{a}))$. This condition

[illegible]

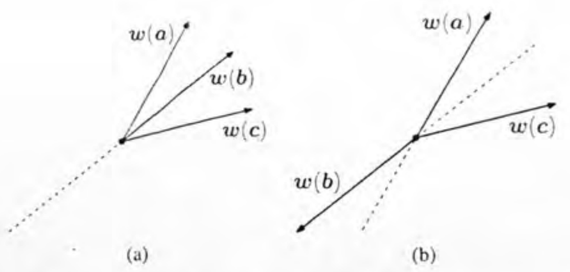

Figure 3.2: Arrangement of three points at infinity. (a) $w(b)$ is between $w(a)$ and $w(c)$. The vectors $w(a)$ and $w(c)$ lie on the opposite side with respect to $w(b)$. (b) Another case where $w(a)$ and $w(c)$ lie on the opposite side with respect to $w(b)$ but $w(b)$ does not lie between $w(a)$ and $w(c)$. The three vectors positively span the plane. It should be noted that $w(b)$ and $w(c)$ also lie on the opposite side with respect to $w(a)$.

is only necessary since b may not be discarded when $w(a), w(b)$ and $w(c)$ positively span the plane. In this case, b and c are also on different side of a (see Figure 3.2b) and the convex hull of these three points covers the entire plane. Any segment must intersect this convex hull, no detection is required.

3.2.3 Intersection Detection

A convex hull is represented by a sequence of extreme points $p_1, ..., p_k$. A pair of adjacent points (with the last point adjacent to the first) define a facet of the convex hull. Let $\text{RI}(\cdot)$ denotes the relative interior. Given a convex hull $\mathcal{C}$ whose facets are $c_1, c_2, \ldots, c_k$, a segment s can intersect with the interior of the convex hull in only three cases: 1) $\text{RI}(s)$ does not intersect with any c_i but it lies in $\text{RI}(\mathcal{C})$, 2) $\text{RI}(s)$ intersects with $\text{RI}(c_i)$ for some i such that s and c_i do not lie on the same line, and 3) $\text{RI}(s)$ intersects with the common boundary point of c_i and c_{i+1} when the other boundary points of c_i and c_{i+1} lie on different sides of the line supporting s. Figure 3.3 demonstrates examples of each case.

Case 1 can be detected by enumerating all facets of the convex hull and test whether a point in $\text{RI}(s)$, says the middle point of s, lies on the same side of all facets. For case 2, we enumerate every facet and check its intersection with s. The relative interior of two segments intersect and do not lie on the same line only when different endpoints of one segment lie on different sides of the other segment and vice versa. Case 3 occurs when an endpoint of facets of $\mathcal{C}$ lies on $\text{RI}(s)$ and some part of $\text{RI}(s)$ lies in $\text{RI}(\mathcal{C})$. Let p_i be the vertex of $\mathcal{C}$ that lies on $\text{RI}(s)$. We have to assert that p_{i-1} and p_{i+1} lie on different sides of s. Figure 3.3d and 3.3e show examples of case 3. In Figure 3.3f, case 3 does not occur although one vertex of $\mathcal{C}$ lies in $\text{RI}(s)$. Note that all the cross products to determine the side of p_{i-1}, p_i, p_{i+1} with respect to the line containing s are already computed in testing of case 2.

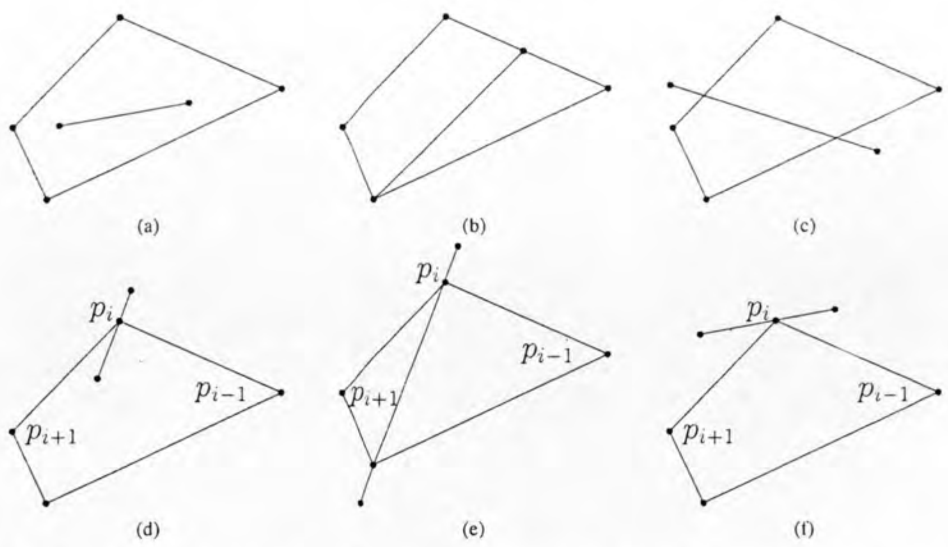

Figure 3.3: Examples of convex hulls and segments. (a) and (b) satisfy the first condition, (c) satisfies the second. The last three figures represent the case when the segment intersects the common boundary of the convex hull. (d) and (e) satisfies the last condition while (f) has no intersection.

To accommodate the situation in which extreme points of the convex hull $\mathcal{C}$ or endpoints of the segment s might be points at infinity, we exploit the modified cross product of Section 3.2.1 with the following special treatment. For the case 1 above, when s is a ray (one ordinary end point, and one end point at infinity), instead of the middle point, we take any point on the ray that is not the ordinary endpoint.

3.2.4 Extension to any Number of Fingers

The method described above is specially developed to benefit three finger cases. However, the general idea based on convex hull intersection introduced in Section 2.6 is applicable to any number of fingers. In particular, the two convex hulls can be constructed using algorithms such as Graham's Scan algorithm and detecting whether the two convex hulls intersect can be done by computing their Minkowski difference and checking whether the origin is contained inside the difference. For n fingers, computing the two convex hulls takes $O(n \lg n)$ and computing their Minkowski difference and testing origin containment takes $O(n)$ provided that the primitive wrenches are sorted by their angles when constructing the convex hulls. The standard Minkowski difference algorithm can be easily extended to handle points at infinity using the idea given in 3.2.1. An alternative is to make sure that no dual points are at infinity. This can be done by choosing the origin that does not lie on a boundary of any friction cone. A practical approach is to generate a new origin at random until the condition is satisfied. This generate-and-test scheme is more efficient than directly computing an appropriate origin since it is rare that a randomly chosen point lies exactly on a boundary of a friction cone. This test can even be omitted altogether in practice in preference of speed over completeness.

3.3 Numerical Comparison

Although several force closure tests were presented in the past decade, we can rarely find a comparative study regarding the performance of these tests. New methods were often proposed without comparing to any existing ones. Presented implementation results from different methods are usually obtained from different computational setting using different input. Obviously, it is difficult, if not impossible, for the reader to fairly compare and evaluate the strength and weakness across available methods. A solution is to establish a common framework for performance benchmark that allows new methods to be compared simply by publishing their results that are produced in compliance with the framework. We take a first step toward this direction. In particular, we implement selected methods in C++ programming language and test them under the same running environment using the same input. Validity of the results are assessed using reference obtained from computation with exact arithmetic. Interestingly, the comparison identifies many discrepancies never stated in the respective original works. This helps us analyze the source of errors exhibited by each method. Of course, performance depends strongly on the quality of implementation. We try our best in implementing the selected methods.

3.3.1 Selected Methods

Plethora of force closure assertion methods exist in the literature. Some methods pursue generality such that the method is applicable with any number of contact and/or dimension of work space. Examples include the ray shooting method by Liu (1999), the Q Distance (Zhu and Wang, 2003) and a quantitative test of Zhu and Ding (2006). who also suggest the use of GJK (Gilbert et al., 1988), a well known collision detection algorithm in computer graphics, as a general test of force closure. Han, Trinkle and Li (2000) proposes a method that can operate on a wide range of finger type using a convex optimization technique. The method of Brost and Mason (1989) is also applicable with any number of contact, though it cannot be used in a 3D grasp. Cornella and Suarez (2005) propose a condition for planar force closure grasps, which is applicable with any number of fingers, to identify independent contact region of a polygonal object. Related condition is also presented for a discretized object in (Cornella and Suarez, 2006).

Many researchers derive methods that are specialized to a particular number of contacts and/or dimension of workspace. These methods usually exploit a priori knowledge of the specialized situation to achieve better performance. Example includes the disposition algorithm (Li et al., 2003) which is very fast in assertion of force closure of three fingers. Tung and Kak (1996) propose a geometrical approach to compute two finger force closure grasp of a polygonal object. Ponce and his colleagues (Faverjon and Ponce, 1991; Ponce et al., 1993; Ponce and Faverjon, 1995) derive several conditions for two to four finger grasps. Cheong et al. (Cheong and van der Stappen, 2005; Cheong et al., 2006) uses a representation similar to force-dual representation to compute all 2D force closure grasp.

Among the aforementioned methods, many are presented as explicit tests while several are implicitly integrated in their grasp synthesis framework. We select six recent methods that distinguish themselves as separable tests of force closure. The first method is the method of Brost and Mason (1989) in which we provide an efficient implementation based on convex hull intersection. The next one is the disposition method (Li et al., 2003) which is included as a representative of specialized methods for three-finger grasps. The other four methods are general methods which can accommodate any number of fingers in any dimension. One of them is GJK (van den Bergen, 1999). The other two are based on linear programming which are the ray shooting method of Liu (1999) and the quantitative test of Zhu and Ding (2006). Finally, the standard convex hull routine, QHull (Barber et al., 1996) is chosen. In our comparison, only QHull is deployed with arbitrary precision arithmetic so that its results can be used as reference.

3.3.1.1 Dual Representation of Brost and Mason

The work of Brost and Mason (1989) provides the fundamental of a dual representation of force. The work suggests that force closure can be asserted in the force-dual representation. As originally described in (Brost and Mason, 1989), to confirm force closure, ones need to find in the plane $\tau = 1$ a pair of intersecting line segments, one joining positive dual points and the other joining negative dual points, or a triangle formed by three dual points of the same sign that contains another dual point of the opposite sign.

However, as noted toward the end of Section 2.6, we can instead compute two convex hulls, one for positive dual points and the other for negative dual points, and test whether they intersect. We use the Graham's scan algorithm for convex hull construction and apply the intersection detection described in Section 3.2.3. Our procedure avoids enumeration of triangles which results in a more efficient implementation. The advantage is more obvious when the number of fingers increases since for n fingers our procedure takes $O(n \lg n)$ while the original method presented in (Brost and Mason, 1989) obviously takes $O(n^3)$. As mentioned in the introduction, the original method in (Brost and Mason, 1989) is developed primarily for intuition and working problems by hand. Extra effort is needed to transform the underlying idea of the method into an efficient and robust computational implementation. Our preliminary experiment shows that this modified approach outperforms the original.

3.3.1.2 Disposition Method of Li, Liu and Cai

The method in (Li et al., 2003) distinguishes itself from the others because it specifically takes the nature of three finger grasps into account. It can quickly identify a certain class of non force-closure grasps. In short, this method removes unnecessary portion of the force cones and show that, with the trimmed force cone, to achieve force closure it is necessary and sufficient that there exists an intersection point of the boundary of two force cones that lies inside the third cone. The disposition of the cone requires only a few algebraic calculations.

3.3.1.3 GJK Algorithm

Invented by Gilbert et al. (1988), GJK algorithm is a well known collision detection algorithm. It determines the closest distance between two convex polyhedra. A variation of the GJK algorithm also supports a binary query; instead of reporting a distance between two objects, it rather decides whether the two objects collide with each other. Taking Minkowski difference of two convex polyhedra as an input, the algorithm determines whether the origin lies inside the convex hull of the Minkowski difference. As suggested in (Zhu et al., 2004), this query is suitable for a qualitative test of force closure. In this work, we use the GJK separating axis algorithm presented in (van den Bergen, 1999). The separating axis algorithm is an improved version of GJK which is designed for binary query. The algorithm can quickly determine whether the origin lies inside the convex hull of wrenches. However, it should be noted that the separating axis algorithm cannot distinguish whether the origin lies on the boundary or in the interior of the convex hull. GJK separating axis runs in $O(n)$ where n is the number of fingers.

3.3.1.4 Ray Shooting Algorithm of Liu

Liu (1999) proposes a simple test of force closure by drawing a ray from an arbitrary point P within the convex hull of wrenches to the origin and calculate the intersection X between the ray and the boundary of the convex hull. Force closure is achieved if and only if the distance between the point P and the origin is smaller than the distance between the point P and the intersection point X. This condition transforms the problem into the ray shooting problem which is extensively studied in the field of Computational Geometry and is solvable by linear programming. This test is used in many subsequent works of Liu such as (Ding et al., 2001a; Liu et al., 2004).

3.3.1.5 Q Distance of Zhu and Wang

Zhu and Wang (2003) propose the concept of Q distance for analysis and synthesis of force closure grasps. The Q distance is a grasp quality measure, calculated for the convex hull of primitive wrenches, denoted by A. Given a *ruler* convex polyhedron Q which contains the origin, the Q distance determines a positive scaling factor of Q (if exists) such that the scaled Q is entirely contained in A. The Q distance consists of two subfunctions, Q^+ distance and Q^- distance. The Q^+ distance determines whether there exists a positive scaling factor that causes a nonempty intersection between A and the scaled Q. This is required for the calculation of Q^- which actually determines the Q distance. It can be formulated as a set of linear programming problems. In our experiments, the ruler polyhedron Q is set to be a regular tetrahedron in order to achieve maximum computation speed.

3.3.1.6 Quick Hull with Arbitrary Precision Arithmetic

A straightforward approach to force closure testing is to directly compute the convex hull of primitive wrenches. The convex hull can be described by the intersection of its bounding half spaces. To test whether the origin lies strictly inside the convex hull, we test whether the origin lies in the interior of every bounding half space (described by the corresponding bounding facet). We use the Quick Hull algorithm (Barber et al., 1996) to compute the convex hull and its bounding facets. The algorithm is implemented using the CGAL library (Board, 2006). The internal computation of the library relies on an arbitrary precision number support from GNU Multiple Precision Arithmetic Library (Granlund, 2006). Employing the arbitrary precision arithmetic avoids the problem of round off errors encountered when using ordinary floating point operations. Of course, using exact arithmetic requires tremendous computing power. The method is therefore selected only as a reference method for validating the results.

Table 3.1 gives summary on the properties of each method. The first row, labeled NSC, describes our novel implementation. The following rows labeled BM, LLC, GJK, Liu, ZW and QHLLL represent the methods in Section 3.3.1.1, 3.3.1.2, 3.3.1.3, 3.3.1.4, 3.3.1.5 and 3.3.1.6, respectively. It should be noted that our new implementation and BM method is essentially the same. Their different lies especially in the case of three finger. The first column shows the time complexity of the method. For linear programming based method, this property depends on the underlying linear programming method. In our work, we use a routine from "Numerical Recipes in C" (Press et al., 1992) which employs Simplex method. Also note that LLC method works only when $N = 3$ and hence no time complexity is given. The next column shows the space where the algorithm performs the computation. Be noted that for linear programming based approach,

Table 3.1: Properties of selected methods

Methods	Time complexity	Working space	N finger	3D Grasp
NSC	$O(n \lg n)$	force-dual	yes	no
BM	$O(n \lg n)$	force-dual	yes	no
LLC	–	force	no	no
GJK	$O(n)$	wrench	yes	yes
Liu	LP	wrench	yes	yes
ZW	LP	wrench	yes	yes
QHULL	$O(n \lg n)$	wrench	yes	yes

this also includes its dual space. The last two columns indicates whether the method is capable of working with any number of finger and working in 3D grasps.

3.3.2 Numerical Examples and Results

We compare the running time of our method with the others. Nine test objects, shown in Figure 3.4, are used. For each object, 200 contact points are randomly sampled from its boundary to form a discrete point set. Using our method and others which are described in Section 3.3.1, every combination of three contact points from each set is tested whether it achieves force closure. The friction coefficient of 1120/6351 is used, which yield a friction cone with half cone angle at approximately 10.001 degrees. Note that the friction coefficient used in every experiment is a rational number such that the sine and cosine values of the corresponding half friction cone angle take on rational numbers. This is to ensure that the reference method is free from floating point operations and can generate true results. All methods are implemented in C++, running on a Pentium IV 3.0Ghz machine with 1GB of RAM. Of all methods, only the Quick Hull employs arbitrary precision arithmetic in order to guarantee the validity of the result. The other methods use primitive data type (64-bit floating point).

Table 3.2 presents the running time of each method to test all $C_{200,3} = 1{,}313{,}400$ grasps. Each row represents the result from each test object. The first column gives the actual running time of our algorithm, labeled "NSC". The remaining columns give the ratio (in percent) between the actual running time of a particular method and our method. The columns labeled BM, LLC, GJK, Liu and ZW represent the methods in Section 3.3.1.1, 3.3.1.2, 3.3.1.3, 3.3.1.4 and 3.3.1.5, respectively. Quick Hull method which is implemented with arbitrary precision arithmetic is not included since the calculation obviously requires tremendous amount of time (more than 1,000 folds of our method). The result of Quick Hull method is included in the experiment only for the purpose of result validation which will be discussed afterward. Note that the running time of the

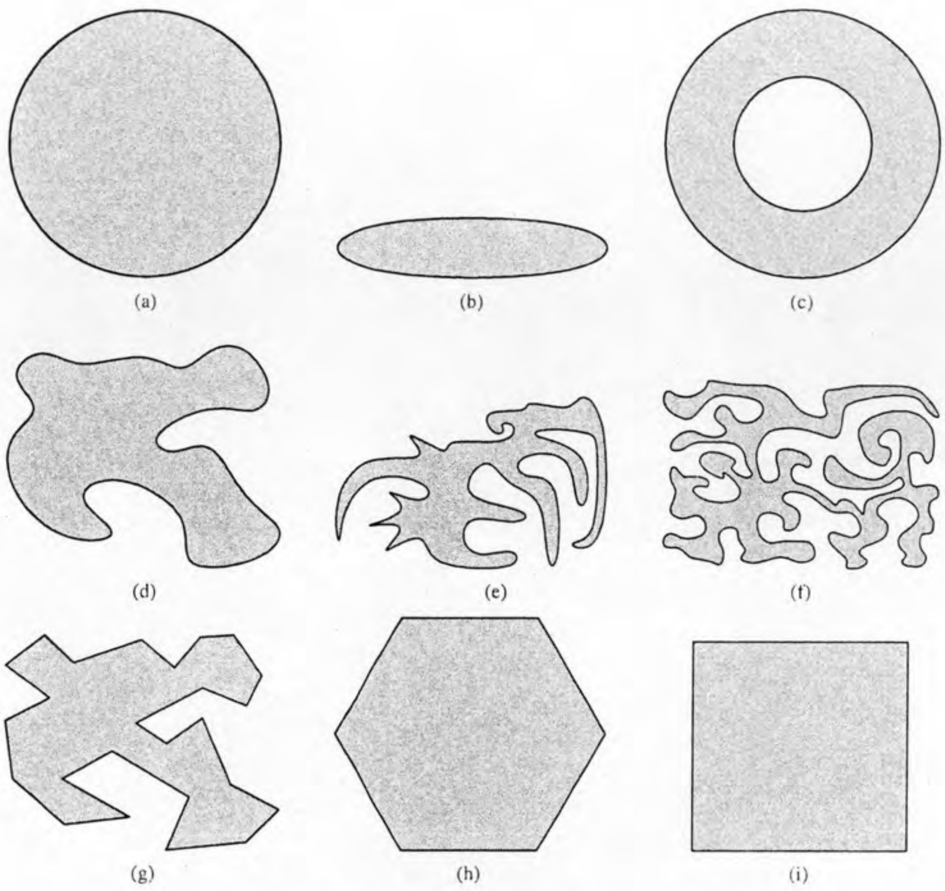

Figure 3.4: Test Objects

column BM could be increased approximately 10 percents if the original test in (Brost and Mason, 1989) was used instead of our modified version.

The result shows that our method is the fastest one, taking less than two third of the second fastest, the BM method. The results shows that there is a large gap between the first three methods (NSC, BM, and LLC) and the remaining methods. The major difference is that NSC, BM and LLC work in 2D space while the remaining work in 3D space. We analyze these faster three methods in terms of arithmetic operations they perform. In the case of three fingers, it is unnecessary to analyze the performance in terms of asymptotic complexity since the input size is fixed. We instead count the number of major operations performed by each method. The major operations considered here are divisions and cross product operations.

We first compare our method against BM method. Our method takes eight divisions to convert wrenches into force dual representation, comparing to twelve divisions required by BM method. From our preliminary experiment, division takes more than half of the total running time of BM method. Another speedup of our method over BM comes from the grouping that always

Table 3.2: Actual Running Time. The friction coefficient is $1120/6351$

Test Objs.	Time of NSC(s.)	Running Time Ratio (%)				
		BM	LLC	GJK	Liu	ZW
(a)	0.806	179.3	280.8	942.9	2,921.6	6,445.8
(b)	0.823	161.8	182.3	759.2	2,982.7	4,009.6
(c)	0.917	169.1	220.3	879.9	2,768.6	5,750.7
(d)	0.910	139.3	192.9	763.3	2,703.1	3,688.8
(e)	0.923	130.9	203.7	732.1	2,672.5	3,583.9
(f)	0.924	132.3	195.1	712.4	2,750.8	3,511.0
(g)	0.904	144.8	196.7	770.2	2,722.1	3,813.1
(h)	0.852	170.1	233.6	921.4	2,887.2	5,433.0
(i)	0.858	143.7	219.5	841.6	2,858.3	4,994.5
Avg.	0.880	152.4	213.9	813.7	2,807.4	4,581.1

yields four points to construct a convex hull. Convex hull of four points can be constructed by our brute force algorithm using four cross products. This is obviously more efficient than the BM method where convex hull is constructed using Graham's scan which also requires sorting. A four point convex hull also reduces the number of cross products required to detect the intersection. At worst, it takes $4 + 16$ cross products, where four of them is to check whether a middle point lies inside the convex hull while the remaining 16 comes from the intersection checking of a segment against the boundary of the convex hull. For BM method, it is possible that the dual representation results in two groups of three points each. Hence, we have to identify intersection of 3^2 segments which can be done in $9 \times 4 = 36$ cross products. Moreover, we have to test whether a point lies in the convex hull of each group, which takes 4 cross products. Hence, BM takes at most $4 + 4 + 36 = 44$ cross products. It is clear that our method use less number of divisions and cross products than BM.

BM Method outperforms LLC method by a noticeable fraction. LLC calculates the position of intersection between two lines. Determining the intersection point between two lines is an expensive operation comparable to six cross products and two divisions. There are at most six pairs of line to be taken into account which amounts to 12 divisions and 36 cross products in the worst case. Moreover, LLC also requires some additional calculation. In the worst case, it requires seven cross products to trim one boundary of friction cone and there are at most six boundaries to be trimmed. This amounts to 42 cross products. In conclusion, LLC uses at most $36 + 42 = 78$ cross products. This support the empirical result that LLC takes more running time than BM.

Table 3.2 represents only the running time, taking no account on the accuracy. Every selected method is a numerical method. Thus, unless the arbitrary precision arithmetic system is used in the implementation, the result is inevitably affected by roundoff error of the floating point

Table 3.3: Number of Errors. The friction coefficient is 1120/6351

Test Objs.	QHULL	Number of Fault					
		NSC	BM	LLC	GJK	Liu	ZW
(a)	538,395	-	-	-	-	-	-
(b)	137,598	-	-	-	2,386	-	-
(c)	463,537	-	-	-	56	-	-
(d)	146,353	-	-	8	1,676	-	-
(e)	140,013	-	-	16	10,673	-	-
(f)	122,081	-	-	21	6,021	10	-
(g)	163,627	-	-	-	777	-	-
(h)	369,807	-	-	-	4	10,762	-
(i)	315,531	-	-	-	-	62,266	-
Total	2,396,942	-	-	45	21,593	73,038	-

representation. In Table 3.3, we compare the validity of the result with the reference method, the Quick Hull algorithm using the arbitrary precision arithmetic system. For each method, we count the number of faults of the solution. The first column shows the number of force closure grasps reported by the reference method, labeled "QHULL". Each of the remaining columns shows the number of faults of the corresponding method.

Table 3.3 shows that only our method, the BM method and the Q Distance method report solutions that totally agree with the reference method. On the other hand, results from LLC, GJK and Liu dissent from the reference. We examine the cause of these errors for each method.

Firstly, we consider the disposition method (Li et al., 2003). Unlike the other cases, most of the conflicting results of the disposition method stem not from the floating point inaccuracy but from the input that is not covered by the case analysis in the proof of the algorithm. In brief, this method considers every permutation of the three contact points, using one contact point (at a time) as the origin and examining the sign of torques produced by the remaining two contact points. If one of the two contact points can produce only positive torque (resp. negative torque), part of the force cone of the other contact point that produces positive torque (resp. negative torque) is disposed because that portion of the cone obviously cannot help forming a force closure grasp. It is stated in Proposition 3 of (Li et al., 2003) that after all permutation of friction cones have undergone such disposition, existence of the intersection of the disposed double-side friction cones is a necessary and sufficient condition of force closure.

The problem arises when no disposition can take place, i.e., when each cone can produce both positive and negative torque no matter which contact point is chosen to be the origin. The proof of Proposition 3 in (Li et al., 2003) states in its second case analysis that this event can occur

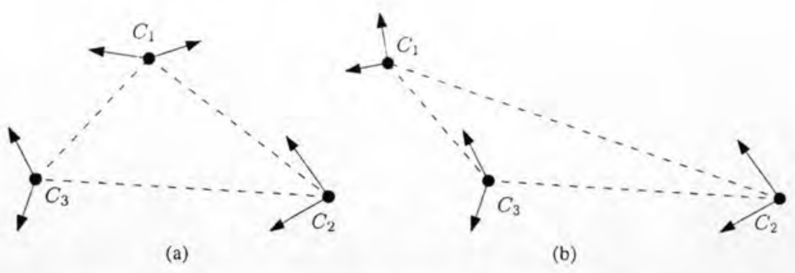

Figure 3.5: (a) The case which is covered by the disposition method. The contact point C_i and C_j both lie in the same side of the double-side friction cone of C_k, for any permutation of i, j, k. (b) The case that is not covered. C_1 lies in the positive friction cone of C_3 while C_2 lies in the negative friction cone. It is clear that all forces point leftward and it does not achieve force closure.

only when each pair of contact points lie in the same side of the double-side friction cone of the remaining contact point. The proof proceeds to derive that a two-finger force closure grasp can always be found when this event is encountered. The case analysis is unfortunately not complete: when the aforementioned event occurs, it is *not* necessary that each pair of contact points lie in the same side of the double-side friction cone of the remaining contact point. An example is shown in Figure 3.5b where each cone can produce both positive and negative torque but C_1 and C_2 lie in different side of the double side friction cone of C_3. In this example, it is obvious that no force closure grasp can be formed (all forces in one half plane), as opposed to the report in error by the algorithm that a two-finger force closure grasp is found. For our test objects, many relatively simple shapes do not exhibit this fallacy. This case is more likely to occur when there are three contact points that lie almost collinear and their normal directions point relatively toward the same direction. Such behavior appears in a noticeable amount in the two complex objects (e)-(f) in our test objects, which could be identified from the fault of the result.

Next, we consider the errors of GJK. These errors are due to the numerical nature of the algorithm. The GJK algorithm is sensitive to the shape of the convex hull of wrenches. It should be noted that even though wrench representation merges the notion of force and torque together, the unit of each component in a wrench is different. In theory, arbitrary change in the scale of each unit has no effect to the force closure property of the grasp. For example, changing the distance measurement of an object results in change of position scaling of contact points and subsequently changes the torque portion of the wrench while force portion remains intact. Scaling up the object results in a larger torque while scaling down the object reduces the torque. However, the direction of the force cone does not change. When the object is enlarged, the shape of the convex hull becomes flatter and thinner. This has a great impact to the GJK algorithms.

All of our test objects have the diameter in the range of several hundred units while the

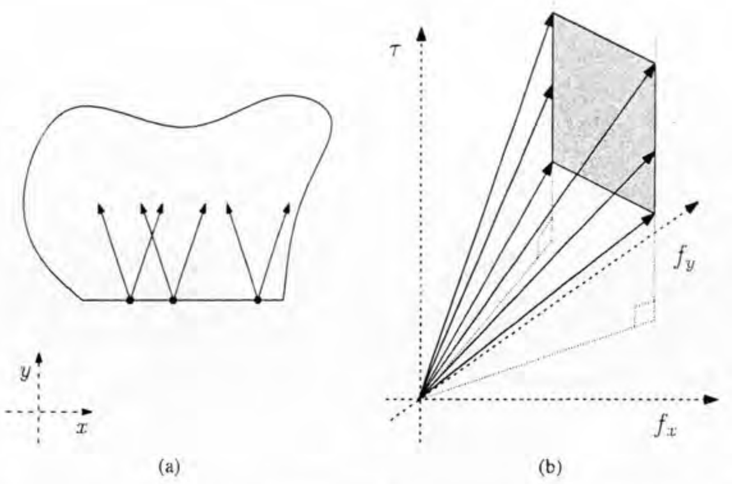

Figure 3.6: (a) Degeneration case commonly occurred in a linear face of an object. (b) The associated wrench, forming a 2D polygon not containing the origin

normal direction of a contact point is a unit vector. This exhibits the aforementioned problem of scaling. To verify this conjecture, we scale down each object by a hundred times, effectively rendering the convex hull to become more rounded. The result is that the number of faults significantly decreased.

Finally, we consider the ray shooting method of Liu. From all test data, it can be seen that the results from the ray shooting method mostly agree with the reference method, except for the hexagon and the rectangle cases (rows (h) and (i) in the table). This is because several special cases of the convex hull of wrenches can be found in these objects. Generally, the convex hull is a full rank polyhedron in 3D space. However, it is possible that the convex hull can be degenerate. A common degenerate case occurs when a convex hull is not at full rank and does not contain the origin. This case can occur very easily when all contact points lie in the same linear side of an object (see Figure 3.6a). In such case, all contact points have the same normal direction but have different positions. These contact points yield two groups of wrenches, each has the same force but has different torque (see Figure 3.6b). The convex hull of the wrenches is obviously degenerate.

In theory, the ray shooting method can distinguish this degeneration. Specifically, when the problem is transformed into a linear programming problem, the degenerate case yields an indeterminate result. However, this is always true only when the exact arithmetic is used. Under a limited precision floating point implementation, the linear programming could overlook this indeterminate case.

Many numerical errors encountered by each method stem from the boundary case where

Table 3.4: Actual Running Time. The friction coefficient is 3/4

Test Objs.	Time of NSC(s.)	Running Time Ratio (%)				
		BM	LLC	GJK	Liu	ZW
(a)	0.939	144.2	343.8	1,086.8	2,402.7	8,148.3
(b)	0.914	135.6	246.4	906.1	2,589.2	5,186.8
(c)	1.059	134.2	255.1	931.6	2,153.3	6,366.8
(d)	1.071	122.7	284.6	945.0	2,287.8	5,475.7
(e)	1.106	117.9	266.6	882.5	2,224.4	5,000.4
(f)	1.163	116.7	260.2	864.6	2,100.3	4,643.2
(g)	1.068	125.9	274.3	899.2	2,239.5	5,307.3
(h)	0.990	145.9	306.7	1,081.1	2,330.3	7,185.9
(i)	0.998	139.6	290.9	993.2	2,435.4	6,223.5
Avg.	1.034	131.4	280.9	954.5	2,307.0	5,948.6

the grasp is on the borderline of achieving or not achieving force closure. For several objects including the polygons with parallel edges such as the test objects (h) and (i), the boundary cases may be frequently encountered. This is because the object is likely to yield groups of three coplanar wrenches (three collinear points in the force-dual representation).

Our method, BM method and Q Distance do not suffer from such boundary cases because they are deliberately tailored to prevent the problem. For Q Distance, one entire instance of linear programming is derived to eliminate the case of equilibrium and non equilibrium. This proves to be useful from our experiment because the problematic case of objects (h) and (i) is easily detected. For our method and BM method, the issue is handled by the intersection detection describe in Section 3.2.3. Merely checking whether two line segments intersect cannot distinguish the case in Figure 3.3d and Figure 3.3f while checking whether endpoints lies inside the convex hull might miss the case in Figure 3.3b. These cases are more likely to occur when three points are collinear. Case (c) in Section 3.2.3 and the use of middle point instead of the boundary points reduce the chance on these boundary cases. From our preliminary experiment, using a simple checking instead of the method in Section 3.2.3 yields a significant number of errors. Careful attention to geometric intricacy of the problem is crucial to achieving a robust implementation.

We also set up another experiment to demonstrate the effect of larger friction coefficient on each method, especially the errors of LLC under larger friction coefficient. The experiment is repeated using a fairly large friction coefficient of 3/4, resulting in a friction cone with half angle at approximately 36.87 degrees. Table 3.4 shows the actual time used by each algorithm and Table 3.5 reports the number of faults of each algorithm in the same manner as Table 3.2 and Table 3.3, respectively.

Table 3.5: Number of Errors. The friction coefficient is $3/4$

Test Objs.	QHULL	Number of Fault					
		NS	BM	LLC	GJK	Liu	ZW
(a)	971,717	-	-	-	-	-	-
(b)	443,162	-	-	-	6,385	-	-
(c)	798,576	-	-	-	160	-	-
(d)	626,034	-	-	429	2,717	-	-
(e)	549,611	-	-	4,400	4,853	-	-
(f)	543,061	-	-	4,100	5,077	5	-
(g)	607,532	-	-	1,602	2,081	-	-
(h)	859,814	-	-	-	679	11,323	-
(i)	713,168	-	-	-	-	77,077	-
Total	6,112,675	-	-	10,531	21,952	88,405	-

A larger friction coefficient yields more force closure grasps. This reduces the benefit of the disposition method that can speedily rejects certain class of non force-closure grasps. Moreover, the errors of LLC increase significantly because it is more likely for its problematic cases to occur when friction coefficient gets considerably larger. The difference between the time used by our method and the disposition method is increased. Our method uses approximately half the time used by the disposition method and less than one third of the GJK method.

3.3.3 Numerical Examples and Results on Multiple Fingers

We also provide a running time comparison of the selected methods in the case of n fingers. Of all selected methods, our method (NSC) and LLC method are omitted from the comparison since they are designed specifically for three fingers. For BM method, the extension described in 3.2.4 is implemented.

Figure 3.7 plots the computation time with respect to the number of fingers. For each specific number of fingers m, 5,000 queries of m fingers are generated from the object in Figure 3.4e. The result shows that BM method outperforms all other methods while GJK is a very close second. Interestingly, Liu method initially outperforms ZW but when the number of fingers increases, the trend starts to show that ZW is faster in the long run.

3.4 Summary

We have introduced an efficient algorithm for force closure testing of three frictional contact points based on the condition derived from the graphical representation of frictional contacts originally given by Brost and Mason (1989). Our focus is on developing an efficient implementation that properly deals with the geometric intricacy of the problem. Performance of the test has

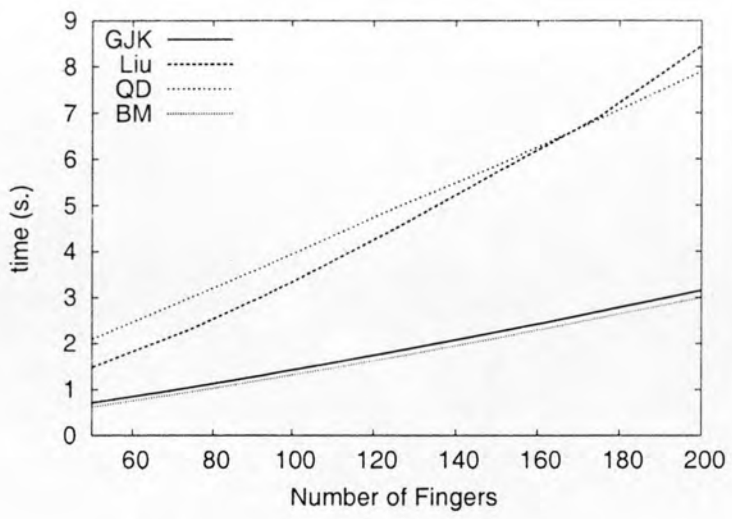

Figure 3.7: Plot between number of fingers and time.

been exhibited from both theoretical and empirical points of view. Besides computational speed, the comparison with existing methods takes into account validity of the results. This is necessary because implementation of each method using native data type of a modern computer, which is of limited precision by its nature, inevitably suffers from the problem of round-off errors. Our verification is based on reference results computed using exact arithmetic. This verification scheme has never been presented before in the literature. It identifies errors and reveals true robustness of force closure testing.

Although the method introduced in this chapter is described explicitly for planar grasps, it is worth noting that the method can easily be applied to 3D grasps as well. The key idea can be credited to (Li et al., 2003) where it is proved that three frictional 3D contact points achieve force closure if and only if there exists a planar force closure grasp on the plane through the three contact points such that the three contact forces are constrained to lie in the intersection between the 3D friction cones and the plane. With this idea, given any frictional 3D three finger grasp, we can compute the corresponding planar grasp through the three contact points and directly apply our method to test whether the planar grasp achieves force closure.