"Assessing the security & gravitational stability of a robot grasp"

4th Year Project Report

David Harker

(Supervisor: Bob Fisher)

Wed 1st June 1994

01 JUN 1994
(1.50 pm)

Departments of Artificial Intelligence and Computer Science

Abstract

This project is concerned with the analysis of the success or failure of object grasp configurations. It uses three systems of analysis to analyse the grasp for equilibrium, stability, and dynamic movement. The analysis assumes the grasp is made from fingers of one of three standard finger types : frictionless point fingers, frictional point fingers, or soft finger types. The result of this project was the following : success was achieved with the equilibrium analysis and stability analysis parts of the system, but not so much with the dynamic analysis of the system, for some technical reasons explained in this dissertation.

Acknowledgements

I would like to take this oppurtunity to thank my supervisors Dr. R.B. Fisher and David Wren for their invaluable help and assistance during the duration of this project.

Contents

List of Figures

Chapter 1

Introduction

Object grasping is an important subfield of robotics. Humans have a great ability to grasp numerous objects easily, even if the object is new to the person. Computer systems are available which can scan objects and determine potential grasping surfaces with which the object could be picked up, and systems are available to determine potential grasp configurations, although in many cases these are specialised systems, which limit themselves to a subset of potentially graspable objects.

The purpose of this project is to analyse the outcome of arbitrary grasps, and determine how stable the object grasp is with respect to slight perturbational forces on it.

1.1 The problem

The problem involves the analysis of an object grasp with arbitrary finger contacts. Figure 1-1 shows an example object along with the local contact regions which define surface patches graspable by finger contacts.

There are three main types of analysis, the purposes of which are to test the object grasp for different situations. The analyses are the following :

Equilibrium analysis : This analysis system decides, given information about the object and contacts whether a state of equilibrium can hold for the object grasp, or whether the object will slip from the grasp.

Stability analysis : If a state of static equilibrium holds for the object grasp, then this analysis system determines how stable the object is by changing the direction of the gravity vector incrementally, and testing if equilibrium still holds for the grasp under this new direction of gravity.

Dynamic analysis : If a state of equilibrium cannot exist for the object grasp, then the object will slip from the grasp. This analysis system simulates such an object slip, and

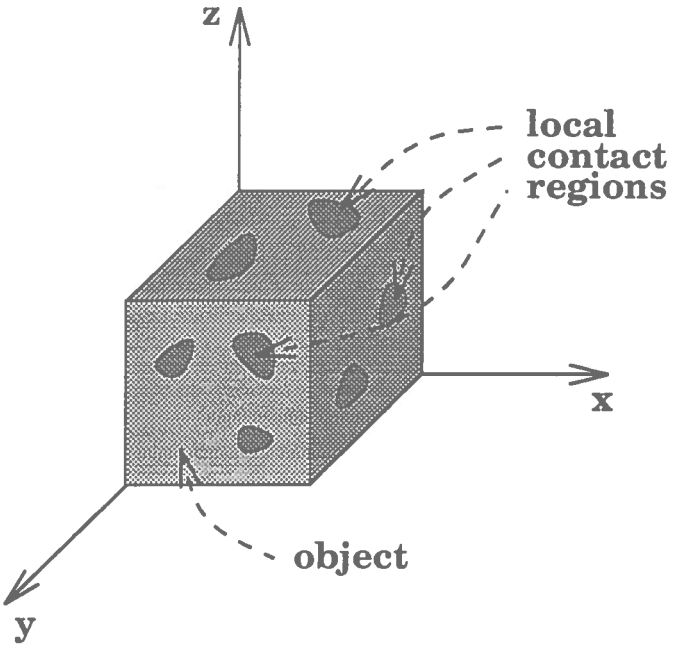


Figure 1.1: An example graspable object, complete with surface contact patches.

determines whether the object slips by a small amount and comes to rest in a state of static equilibrium, or whether the slip of the object is so severe that the finger contacts are no longer in range and the grasp fails completely.

two key words here are *equilibrium* and *stability*. It is important that these are explained and the differences between the two are established.

An object is said to be in *equilibrium* if the following two conditions hold :

- For all forces acting on the object the sum of their force convexes must equal zero.
- For all torques acting on the object the sum of their torque convexes must equal zero.

An object in equilibrium is not necessarily stable. The stability of an object depends on the objects ability to withstand small external perturbational forces acting against it.

1.2 The achievements

The equilibrium analysis and the stability analysis were successfully implemented. In the case of the dynamic analysis system the theory behind the system was well understood, but a combination of implementational problems and the limited time available resulted in no fully working version of the dynamic analysis system.

1.3 Overview of the Report

The report is decomposed into six chapters (including this one) which relate to the system as a whole. In Chapter 2 I describe the information needed by the system in order to perform its analyses on arbitrary object grasps, and how such information builds a world model for the system. Chapter 3 explains the implementation of the equilibrium analysis system, which determines whether or not an object grasp is in static equilibrium. In Chapter 4 I explain the theory behind the stability analysis system, and the results and deductions obtained from it. Chapter 5 is about the dynamic analysis system, which simulates an object as it slips with respect to the grasp. I also explain the problems that arose in the design and implementation of the dynamic analysis. Finally the conclusion reviews the finished system as a whole, and determines improvements and further work which could be done.

Chapter 2

A decomposition of the world model

In order for the system to perform an analysis of a potential grasp, it requires a sequence of input data in order to construct its model of the world. The input needed for the world model is decomposed into three main groups — information about the object, information about the finger contacts, and information about various control variables which influence the analysis.

In this chapter I will describe the list of input data in more detail, and explain how each piece relates to the development of a world model for the system.

2.1 Global information about the object to be grasped

As a result of the set of assumptions about the system as described in Chapter 1 (in particular considering local surface features of the object as opposed to a global object description, and considering the grasp as a set of disembodied fingers as opposed to the entire manipulator hand), the required information concerning the object to be grasped is scaled down from an entire volumetric description to three fundamental pieces of information described below.

2.1.1 Force and direction of gravity on the object

It would be an option just to assume that the situations concerning object and grasp will be analysed with respect to the earth's own gravitational field (ie. acceleration is $\vec{g} = 9.81ms^{-2}$ in a negative vertical direction). However, the system will be more diverse if arbitrary forces and directions of gravity can be considered. An additional benefit of considering arbitrary gravity directions is that it allows testing for equilibrium of a grasp in various specific orientations by simply altering the direction of gravity with respect to the grasp. This would have the same effect of altering the orientation of the whole object data with respect to gravity, but it

only involves the calculation of a new gravity vector. Generally, gravity can be viewed as the resultant external force on the object, so by changing the direction of gravity the system can test what happens when external forces act on the object. The ability to specify an arbitrary force and direction of gravity is used during stability analysis, which is explained in Chapter 4 of this report.

2.1.2 Centre of mass of the object

One part of the equilibrium analysis is concerned with the balancing of torque convexes acting on the object. The sum of all torques should equal zero. The resultant torque convexes from each finger contact are calculated by taking moments of the wrench force convexes of each finger contact. The centre of mass of the object gives the location at which gravity acts. It is because of this the centre of mass is part of the input to the system.

The centre of mass of an object can easily be computed, given a volumetric model such as a voxmap, if it is assumed that the object is of uniform density.

2.1.3 Moment of inertia of the object

The dynamic analysis of the object determines, in the case of the object slipping, whether it will slip into a position of equilibrium, or if it will slip to such a position that the finger contacts outside the range of the local contact patches. In performing the analysis it calculates the translational and rotational acceleration of the object. The moment of inertia of the object is required to compute the object's rotational acceleration about its centre of mass.

2.2 Information about the object surface contact points

For each finger contact in a potential grasp, the system must receive information about its contact patch, the type of contact, and the transformation of the contact into global coordinates. This section explains the use of such information in more depth.

2.2.1 Local contact patches

The analysis programs in this project assume the local surface features of an object to be represented by biquadratic equations. Figure 2-1 shows some example biquadratic meshes produced by MATLAB[3] of the main types of surface patches.

The biquadratic equation for a local contact patch is as follows :

$$z = ax^2 + by^2 + cxy + dx + ey + f$$

The coefficients for patch i are represented by vector $\vec{b_i}^T = (a, b, c, d, e, f)$.

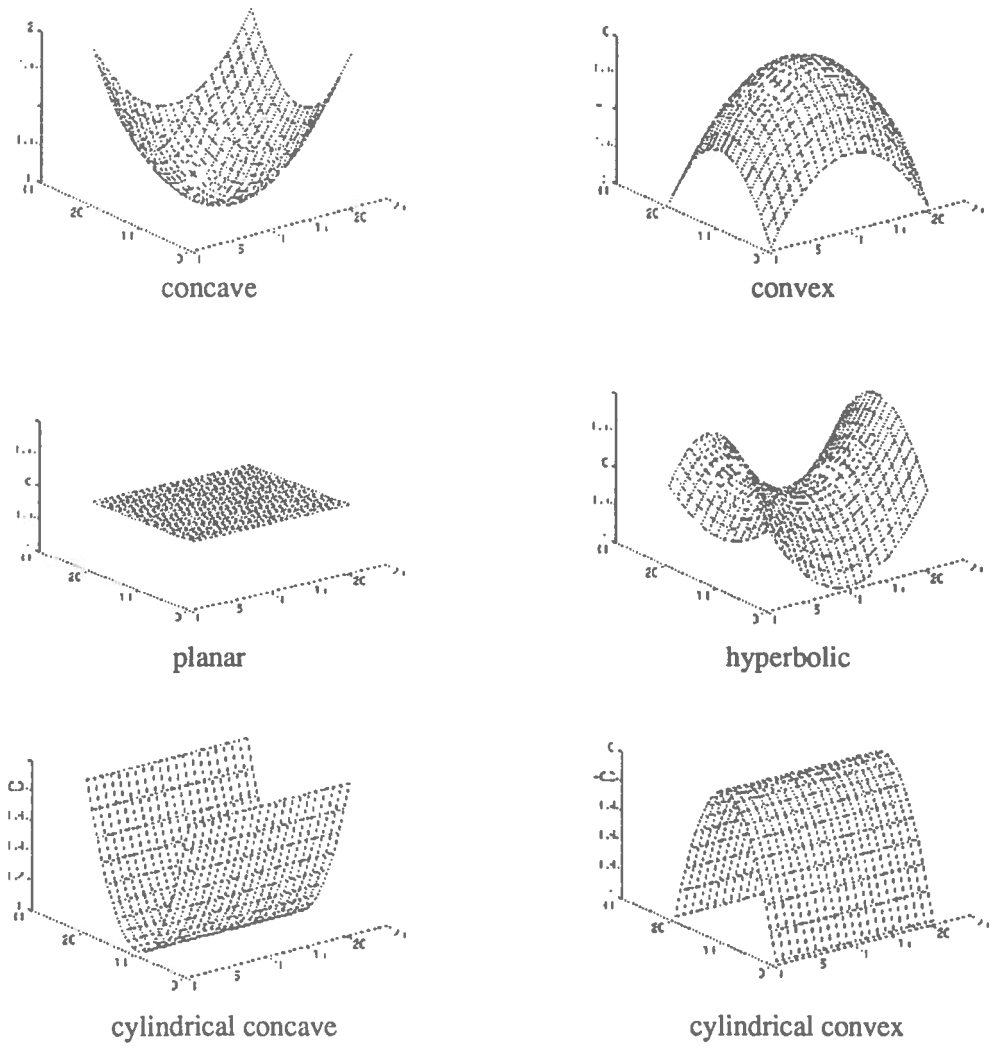


Figure 2.1: Example biquadratic surface patches.

There are a number of advantages in using biquadratic equations to describe local surface contacts :

- Surface normals of biquadratics are easy to calculate, which is a benefit as they are used extensively throughout the analysis.
- Biquadratics give a good approximation to the set of main surface features of an object (eg. convex, concave, planar, cylindrical, etc).
- Biquadratic equations return an unambiguous value of z for any given x & y.

Each biquadratic surface is defined in terms of its local coordinate system. Points on the patch and normals to the patch can be transformed to the global coordinate system of the object with the aid of a transformation matrix, as described later in this chapter.

2.2.2 Finger contact types

This project considers the analysis of three main finger contact types. They are described below in order of increasing complexity.

Frictionless point contacts :

This type of finger contact is the simplest to model, although it is the least impressive of the three in terms of the margin of stability it produces. In general to achieve stable grasps using frictionless point contacts, a greater number of finger contacts is needed in comparison to the other two finger contact types. In the case of frictionless finger contacts, up to six contacts are required for equilibrium, seven for stability.

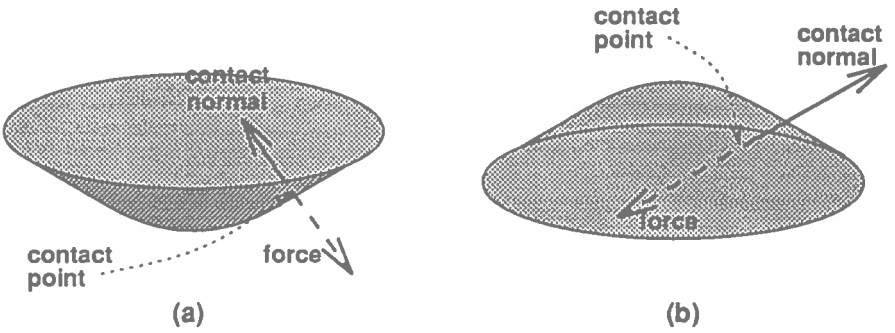


Figure 2.2: Example frictionless point forces for (a) concave patch, (b) convex patch.

The only useful force a frictionless point contact can exert into a surface patch lies in a direction parallel and opposite to the surface normal of the contact location on the patch. Some examples of such forces are shown in Figure 2-2.

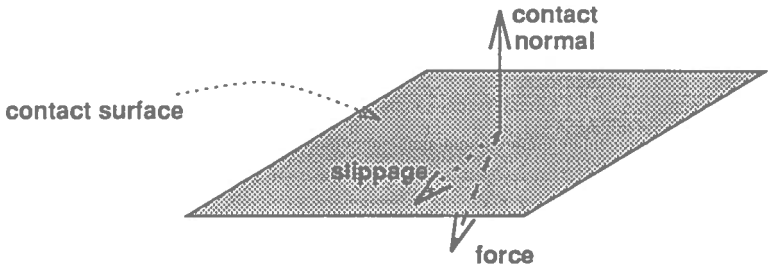


Figure 2.3: Force exerted in an unsuitable direction causes a slip.

Any attempt to exert a force in a direction other than the one described will result in the finger contact slipping relative to the surface patch. Figure 2-3 represents such a situation arising.

Frictional point contacts :

Frictional point contacts have an advantage over the preceding point contacts in the sense that they allow forces to be exerted in directions other than parallel to the surface normal, depending on the coefficient of friction between the finger and surface patch. The value of this coefficient defines a *friction cone* inside which, a non-slipping force can be exerted. Figure 2-4 shows an example friction cone on a concave contact patch for this type of contact.

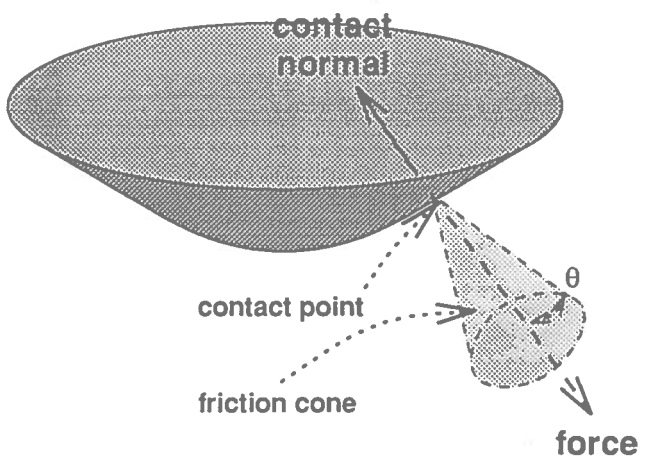


Figure 2.4: Example friction cone for frictional point contact.

The coefficient of friction, μ, between the finger and surface patch produces a friction cone of angle 2θ parallel and opposite to the contact normal, where:

$$\theta = tan^{-1}(\mu)$$

If a force is exerted by the finger contact outside the boundary specified by the friction cone, the result will be that the finger contact will slip with respect to the surface patch. Obviously it is preferable to have relatively high coefficients of friction between the finger and surface patch as this allows a greater range of force directions useable by the finger contact, thus a more useful finger contact overall.

Soft finger contacts :

This is the most powerful of the three finger contact types, so it is no surprise that it is the most complex to model. Soft finger contacts are compliant, so they possess the ability to mould themselves around their contact surface to some degree, resulting in larger friction cones than would be attained by the previous finger contact type in similar situations. Soft finger compliance is especially useful when considering the grasping of edges and vertices, as soft fingers can constrain an object around such features by deforming around the feature, the

deformation depending upon such criteria as the shape of the contact and the shape, size and compliance of the finger. This is a useful property of this finger type, although this research is concerned with the grasping of objects via surface contact points as opposed to edge and vertex contact points.

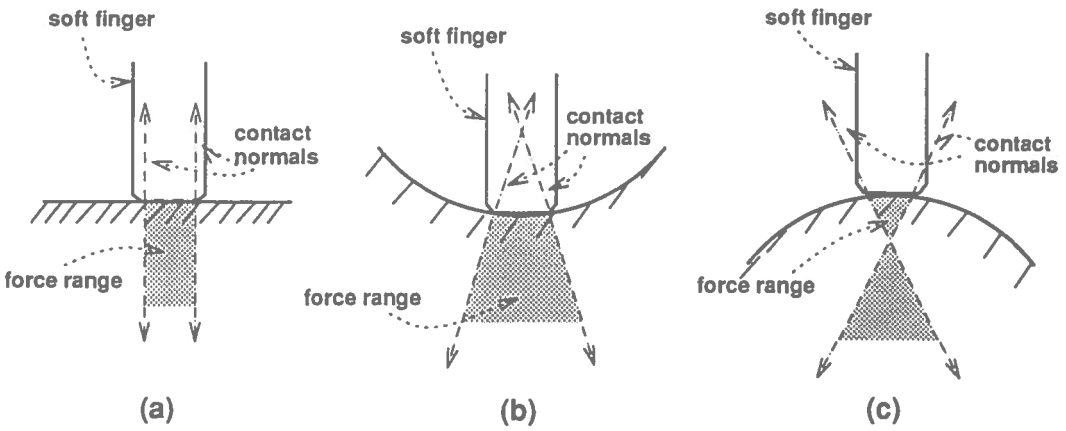


Figure 2.5: Example force exertion ranges for soft fingers (with zero friction) on (a) planar, (b) concave and (c) convex surface patches.

The radius of the finger is one factor determining the resultant area of contact between the finger and surface patch. The range of possible forces within which the soft finger can safely (ie. without slipping) exert a force, without considering friction for the moment, lies within the group of force directions parallel and opposite to the surface normals at the perimeter of the area of contact between the finger and its surface patch. Figure 2-5 illustrates this 2-dimensionally for three example surface patches.

Figure 2-5(c) reveals a potential problem which arises with this representation for convex patches. It suggests that in the situation of convex patches, any forces the soft finger can safely exert must lie in a direction such that they pass through the point where the range of possible forces crosses over. How this problem is overcome is discussed in more detail in Chapter 3.

The result of adding friction to this system is that each force direction on the perimeter of the range of contact forces is rotated outward by angle θ (where once again $\theta = tan^{-1}\mu$), as in Figure 2-6. The overall effect of this is a wider range within which the soft finger can exert forces without slipping.

An additional property of this type of finger contact is that it can produce a resistance to torque around the surface normal of the contact, as represented in Figure 2-7.

The modeling and implementation of soft finger types, along with the other two finger contact types, are explained in greater detail in the next chapter.

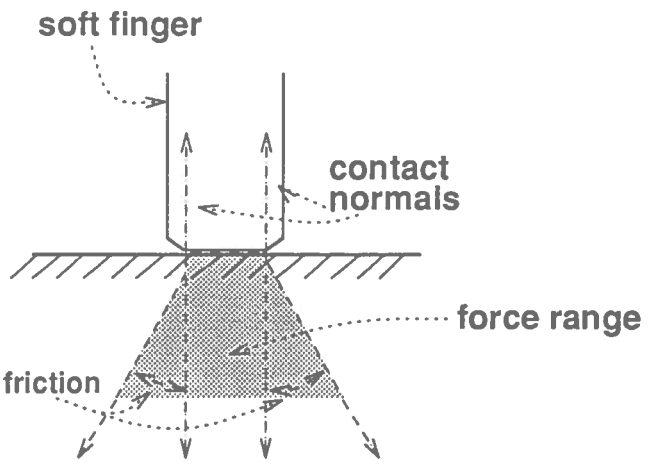


Figure 2.6: Example force exertion range (including friction) for a soft finger on a planar surface patch.

2.2.3 Local contact points and transformations

Analysis of a grasp involves the transformation of contact positions and force directions from their local coordinate systems to a global coordinate system. If the transformation matrix is represented in homogeneous coordinates then the transformation can be accomplished by the matrix multiplication :

$$\begin{pmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} U_x \\ U_y \\ U_z \\ 1 \end{pmatrix} = \begin{pmatrix} V_x \\ V_y \\ V_z \\ 1 \end{pmatrix}$$

where R_{ij} are rotation elements, T_j are translation elements, U is a 3-dimensional vector in local coordinates and V is a corresponding vector in the global coordinate system.

2.3 Miscellaneous control variables

Some of the inputs to the system exist purely for the function of setting control parameters to the three analysis processes of the system. The values of these parameters affects the results the system produces, as described below.

2.3.1 Upper and lower bounds on force values

Many grasps involve the use of counteracting parallel forces to constrain the object within certain dimensions; for example, given two force magnitudes f_1,f_2, one counteracting the

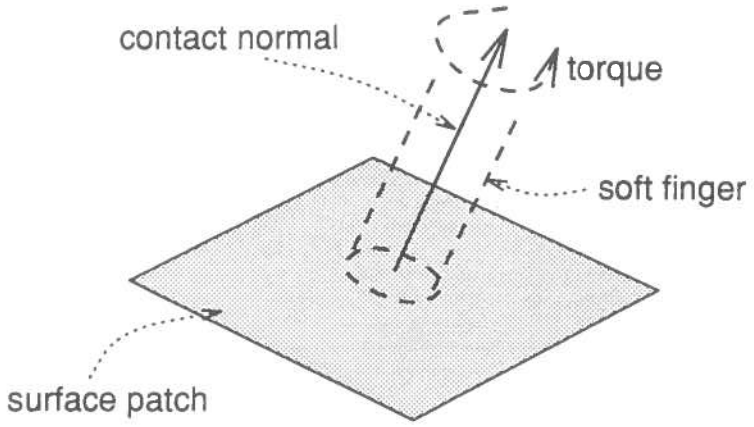


Figure 2.7: Soft fingers can resist torque about their surface normals.

other on the opposite side of an object such that for an equilibrium to exist :

$$f_1 = -f_2$$

If the two forces are unbounded then there is an infinite number of possible solutions for these force values. It is desired that the system produces a finite value for each force in a situation of equilibrium. A way of accomplishing this is to impose a maximum upper bound, u_f on each force value such that for all force values f_i, the following holds :

$$f_i \leq u_f$$

Similarly, a lower bound, l_f is introduced such that :

$$f_i \geq l_f$$

If this lower bound is set to zero, then it allows finger contacts to potentially exert zero force to the object grasp. This may be a desired situation, but usually if a finger contact is part of a grasp then it should exert some useful counteracting force, otherwise it is redundant, and introducing a lower bound greater than zero ensures that this happens.

2.3.2 Angle of gravitational stability

The stability analysis part of the program tests for equilibrium of the grasp during the perturbation of the gravity vector within a cone of stability. The size of the cone is determined by the input angle of gravitational stability.

- Maximum patch displacement range : d $(scalar)$
- Number of finger contacts : n $(integer)$

For each finger contact (i) :

- Finger contact type : t_i $(integer \in 1, 2, 3$ encoding frictionless, frictional, soft finger contact$)$
- Local coordinate of finger contact point : $\vec{p_i}$ $(3d\ vector)$
- Transformation matrix for contact point : T_i $(4 \times 4\ matrix)$
- Biquadratic surface patch at contact : $\vec{b_i}$ $(coefficient\ vector\ of biquadratic\ equation)$

For each frictional finger contact (i) :

- Coefficient of friction between finger & contact surface : μ_{f_i} $(scalar)$

For each soft finger contact i :

- Radius of finger : r_i $(scalar)$

2.3.3 Maximum patch displacement range

In the dynamic analysis of the system finger contacts can slip from their initial contact position on their local surface patches. The maximum distance they are allowed to slip before the grasp is deemed to have failed is determined by the maximum displacement range d. This is an input to the system which defines a circular area around each initial contact position within which slippages are analysed.

2.3.4 Tolerance values

The reason for introducing tolerance values into the system is to handle certain errors which can occur in calculating force and wrench convexes. The errors, ϵ, are typically small, and can be the result of rounding errors in floating point multiplication, or more dominantly from approximating a real surface with a mathematical equation.

The tolerance values are used in the equilibrium analysis such that instead of testing for equality :

$$x_1 = x_2$$

we test for equality within a certain tolerance range :

$$x_2 - \epsilon \leq x_1 \leq x_2 + \epsilon$$

The tolerance values input to the system determine the size of this range.

2.4 Summary

This chapter has been concerned with the input requirements of the system and the relevance of such data in the construction of the system's world model.

The following is a comprehensive list of required input data :

- Direction & magnitude of gravitational acceleration : $\vec{g}$ $(3d\ vector)$
- Centre of mass of the object : $\vec{c}$ $(3d\ vector)$
- Moment of inertia of the object : I $(3 \times 3\ matrix)$
- Tolerance value for errors : ϵ $(scalar)$
- Upper & lower bounds on force values : u_f,l_f $(scalar)$
- Maximum angle for gravitational stability tests : θ_s $(scalar)$

Chapter 3

Equilibrium Analysis

The equilibrium analysis system is used to determine whether or not a solution is possible for the object grasp to be in a state of static equilibrium, and if so finds a minimal set of values and directions for the finger contact forces.

3.1 Mapping the problem onto equations

In Chapter 2 we saw a selection of input data required by the system. In this section we will see how some of this data is mapped onto an equation form so it can be solved by the system.

3.1.1 Calculating surface normals

Surface normals are used extensively in the modelling of the three finger contact types. This section explains how they are computed from the finger contact and surface patch data.

Given the following two pieces of information :

- Equation of the surface patch of the form : $z = ax^2 + by^2 + cxy + dx + ey + f$
- Local contact position of finger on the patch : $\vec{x_i} = (x, y, z)$

the resulting surface normal, $\vec{n_i}$ is :

$$\pm \begin{pmatrix} \frac{\delta z}{\delta x} \\ \frac{\delta z}{\delta y} \\ -1 \end{pmatrix}$$

where :

$$\frac{\delta z}{\delta x} = 2ax + cy + d$$

$$\frac{\delta z}{\delta y} = 2by + cx + e$$

The $\pm$ sign of the contact normal determines whether the normal is into the surface (+ sign) or out of the surface (− sign). This is demonstrated by figure 3-1. The normals into the surface are used for calculating force directions.

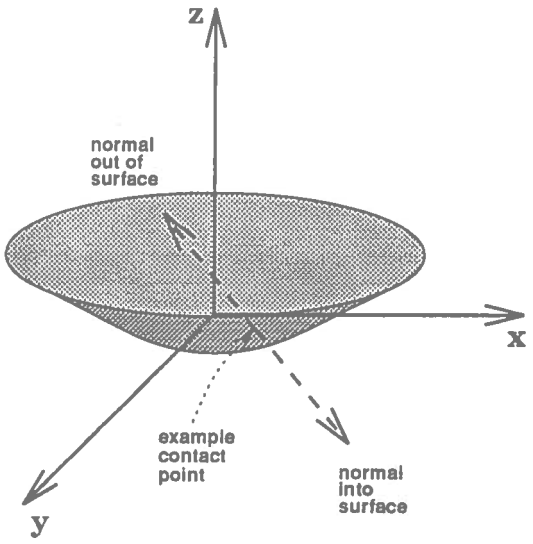


Figure 3.1: Contact normal for an example surface and contact point.

3.1.2 Calculating torques from forces

Any finger contact which exerts a force on the object has the potential of exerting a torque on the object. The centre of mass of the object can be used as a reference point about which the torques are exerted. This section deals with calculating such torques in a 3-dimensional system.

Given the following information :

- Direction of force i on the object, $\vec{v_i}$
- Origin of this force, $\vec{p_i}$
- Centre of mass of the object, $\vec{c}$

The resultant component of torque, $\vec{t_i}$ can be calculated with the aid of the *vector cross-product* formula (eg. Plastock & Kalley [4]) as follows :

$$\vec{t_i} = (\vec{c} - \vec{p_i}) \times \vec{v_i} \qquad - \; f_i \quad \text{magnitude of force?}$$

where $\times$ = *vector cross-product*. The result is a vector perpendicular to the plane defined by $\vec{c}$, $\vec{p_i}$ and $\vec{v_i}$, and the resultant component of torque acts about this vector, as depicted in Figure 3-2.

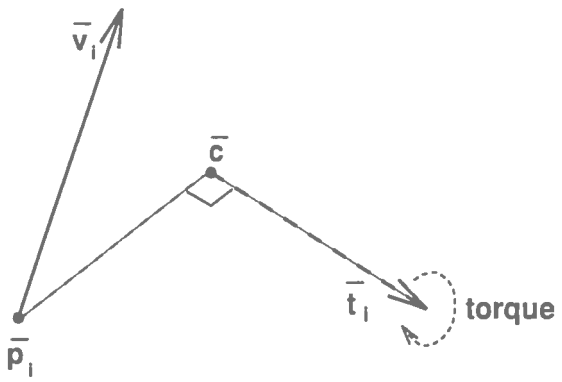


Figure 3.2: Example of a resultant torque in a 3-dimensional system.

3.1.3 Implementation of the finger models

The three finger models were described theoretically in Chapter 2. In order for the equilibrium analysis program to use such models they first have to be converted into a computational representation.

3.1.3.1 Frictionless point finger contacts

Frictionless point finger contacts can exert a force in one direction only — the direction parallel to the surface normal between the point contact and surface patch. As a result the implementation of a frictionless point finger model is straightforward, it simply involves constraining the range of possible force directions exertable by this type of contact to the single direction defined by the surface normal.

Similarly, this type of finger contact is constrained to exerting a torque in one direction only. The reason for this is that the direction of torque exerted by a finger contact is a function of the position of the finger contact, the centre of mass of the object, and the direction of force exerted by the finger. The finger contact position and the centre of mass of the object are constant values, so as the force direction is constrained to a single value, then so is the direction of torque exerted by this finger contact type.

If a particular grasp contains N_i frictionless point contacts, then the resultant force and torque convex for these contacts (in the global coordinate system) will be :

$$\begin{pmatrix} \sum_{i=1}^{N_i} f_i \vec{n_i} \\ \sum_{i=1}^{N_i} f_i \vec{n_i} \times (\vec{x_i} - \vec{c}) \end{pmatrix} \tag{3.1}$$

where f_i is the force exerted by finger i, $\vec{n_i}$ is the direction of force, $\vec{c}$ is the position vector of the centre of mass of the object, and $\vec{x_i}$ is the position vector of finger contact i.

3.1.3.2 Frictional point finger contacts

This type of finger contact requires more consideration than its predecessor, as there is a range of directions within which it can exert a useful force. This range is defined by a friction cone model (as described in Chapter 2), so in order to constrain the possible force directions to lie inside the friction cone, a computational representation of the cone is needed.

A key idea in the implementation is to approximate the theories and equations linearly if possible, the reason being that linear programming can then be used to solve the equations, which is theoretically simpler and typically computationally less expensive than would be the case if higher order methods were resorted to in solving the equations.

In Chapter 2, figure 2-4 showed an example friction cone for a frictional point finger contact. This cone can be thought of as a vector displaced by angle θ from the direction vector of the contact normal, rotated by 360° around the direction vector of the contact normal.

Such a friction cone can be approximated linearly by considering the convex sum of a set of N vectors, which lie on the boundary of the cone. Figure 3-3 (b) shows an example for $N = 4$. The convex sum of the vectors defines a range which approximates to the range of the friction cone model. Using this approximation a useful force direction (one which lies within the boundary specified by the friction cone) can be defined as any vector which is a linear combination of the set of N boundary vectors.

There is a trade off here in choosing the number N of boundary vectors — the greater the N, the close the approximation to the friction cone model, but the greater the computational cost.

It was decided that using $N = 4$ boundary vectors gives an accurate enough approximation to the friction cone model. Obviously there are possible force directions on or near the boundary of the friction cone model which the linear approximation would not allow (Figure 3-3 (c)). This can be justified by considering the fact that force directions that lie towards the centre of the friction cone model are typically safer, and force directions on or near the boundary of the friction cone are dangerously close to slipping, so there is no great loss if the approximation eliminates the possibility of certain boundary force directions.

The friction cone approximations were implemented with the aid of some 3-dimensional transformations (for example, Plastock & Kalley [4]). The basic method is as follows :

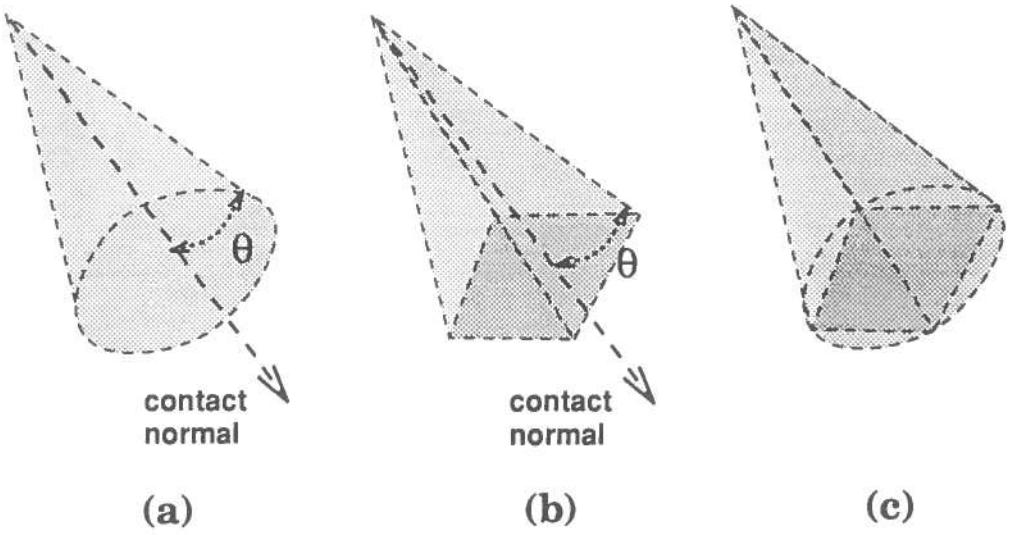


Figure 3.3: Example friction cones — a) model, b) linear approximation, c) both together.

1. Rotate the contact normal (in global coordinates) such that it lies on the positive z-axis.
2. Rotate the result about the x and y axes by $\pm\theta$, (where $\theta = tan^{-1}\mu$, μ = coefficient of friction), to give four vectors.
3. Rotate these four vectors inversely to the rotation of (1.) to give the boundary vectors in their global coordinate orientation.

This method can be seen diagrammatically in Figure 3-4 for an example contact normal and coefficient of friction.

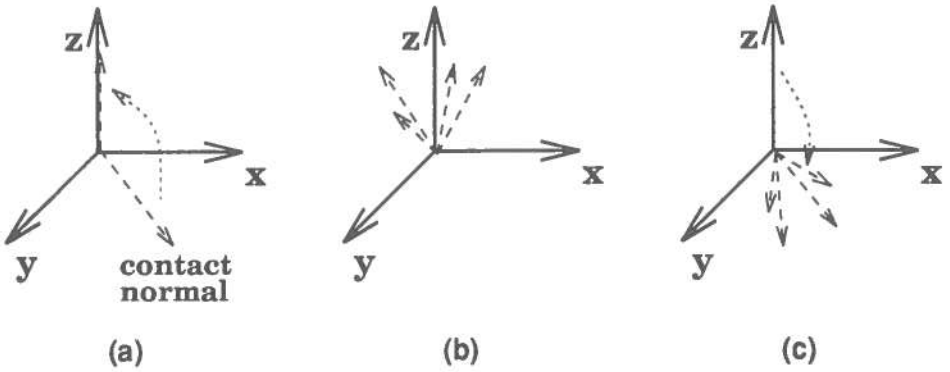


Figure 3.4: Method of producing friction cone approximation from contact normal : (a) rotate normal to positive z axis, (b) rotate by $\pm\theta$ about x and y axes, (c) rotate results inversely to (a).

As a frictional point contact is capable of exerting a force in a number of possible directions,

so too is it capable of exerting a torque in a number of directions. The range within which valid torques can be exerted is found by taking the convex sum of the torques of the boundary force vectors.

If a particular grasp contains N_j frictional point contacts, then the resultant force and torque convex for these contacts (in the global coordinate system) will be :

$$\begin{pmatrix} \sum_{j=1}^{N_j} \sum_{b=1}^{4} f_{jb} \vec{v_{jb}} \\ \sum_{j=1}^{N_j} \sum_{b=1}^{4} f_{jb} \vec{v_{jb}} \times (\vec{x_j} - \vec{c}) \end{pmatrix} \tag{3.2}$$

where f_{jb} is the component of force exerted by finger j in direction of boundary vector $\vec{v_{jb}}$, $\vec{c}$ is the position vector of the centre of mass of the object, and $\vec{x_j}$ is the position vector of finger contact j.

3.1.3.3 Soft finger contacts

Once again it is preferable to approximate the finger contact model with linear components if possible, although it is slightly more complicated for this type of finger contact than it was for the previous one. The two key differences between this type of finger contact and the previous type are :

- Soft finger contacts occupy an area of surface patch rather than a single point on the patch.
- Soft finger contacts can resist torque about their surface normals, to some extent.

Before discussing details concerning the implementation of a soft finger model I will take this opportunity to point out the main problems in computationally modelling the two properties of soft finger contacts listed above. The first of these properties describes an area of contact between finger and surface, but the size of this area is determined by a variety of factors — the size of the finger, the compliance of the finger, the shape of the surface patch, the amount of force exerted by the finger on the surface, to name but a few. The second of the above properties mention the soft fingers' resistance to torque about their surface normals, but to what extent can they resist torque? Yet again this is dependent on a number of contributing factors, for example the coefficient of friction between finger and surface, the are of contact between finger and surface, the amount of force exerted by the finger on the surface, etc.

In order to make the modelling of soft finger contacts feasible for this system some simplifying assumptions are added to the model :

- The area of surface patch occupied by the finger will be circular and of constant size defined by r, the radius of the circle, ie. assume a circular fingertip.

- The soft finger will be able to resist torque about its surface normal up to a maximum value defined by the upper bound value u_f (section 3.3.1).

With these assumptions in mind, the implementation of a computational model for soft finger contacts is made significantly easier.

In a similar way to frictional point finger contacts it would be preferable to approximate the range within which a force can exert a force by considering the convex sum of a set of four linear vectors. The difference here is that the vectors will not all share the same position on the contact surface as they did for the previous finger contact. To give a relatively close approximation to the soft finger contact, the four vectors should all lie somewhere on the perimeter of the circular area defining the region of contact between the finger and surface. They should represent the minimum and maximum principle components of curvature of the contact, (ie. one of the vectors should lie at a position on the perimeter where the normal has the greatest displacement from the normal at the centre of the patch, and one should lie at a position where the normal has the least displacement). The other two vectors should lie in positions directly opposite to the above positions in order that as much area of surface contact as possible is approximated. Figure 3-5 shows an example of this for boundary vectors $\vec{v_1}$ to $\vec{v_4}$.

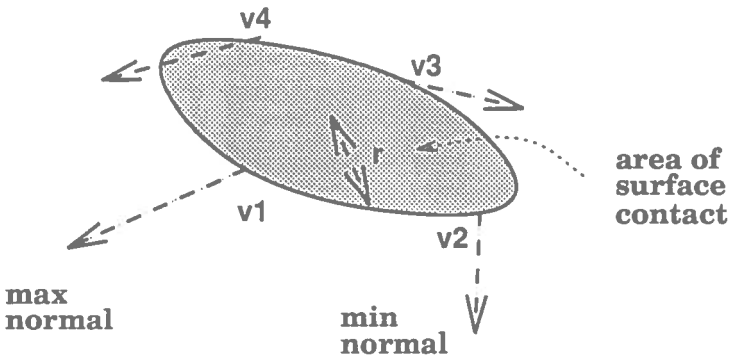


Figure 3.5: Approximating a soft finger force range linearly.

Regarding the problems associated with convex surface patches (as described in section 3.2.2), a solution is to treat the surface contacts as co-planar, thus alleviating the problem of surface normal ranges crossing over. This justification of this is that for convex patches of relatively low components of curvature (ie. the x^2 or y^2 coefficients are small and negative), then approximating them to planar patches only introduces a relatively small distortion into the approximation, which is diluted further by the simplifying assumptions already made about the system. Conversely, convex patches of relatively high components of curvature (ie. the x^2 or y^2 coefficients are large and negative) are more likely to represent edge or vertex contacts than facial surface contacts, which the system does not analyse in any case.

Adding friction to the four boundary vectors involves rotating each boundary vector by angle θ, ($\theta = tan^{-1}\mu$), about an axis perpendicular to a line joining the centre of the circular patch and the position of the boundary vector on the perimeter of the circle. Once again this can be implemented with such 3-dimensional transformations as described by Plastock & Kalley[4]. Figure 3-6 reveals the addition of friction to a boundary vector in an example situation. The range within which a soft finger can exert a force is defined as the convex sum of the four boundary vectors.

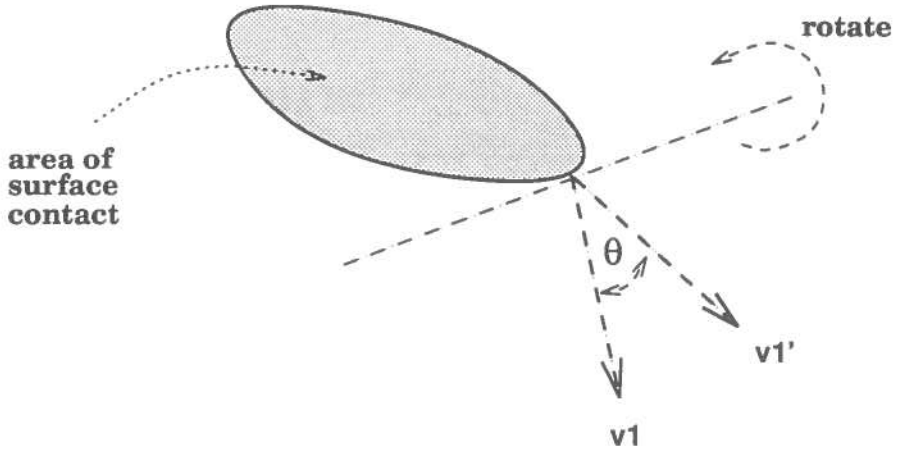


Figure 3.6: Adding friction to a boundary vector.

As with frictional point contacts, soft finger contacts are capable of exerting a force in a variety of possible directions, and so too are capable of exerting torques in numerous directions. Unfortunately the use of extended point contacts to approximate the model allows a certain property of the approximation which would not be possessed by a real soft finger contact — that of allowing an arbritrary force and torque to be exerted from a distinct point within the surface contact, as shown in Figure 3-7 (a). In a real soft finger contact the force and torque would be spread more evenly throughout the surface contact of the finger as in Figure 3-7 (b).

In the vast majority of cases the linear approximation is an adequate representation of the soft finger model, so I will leave this undesireable property as a potential for further refinement, as the amount of work required to eliminate this problem far exceeds any benefits gained from such an elimination.

Given this implementation of soft finger contacts by linear approximation, the resultant force and torque convex for N_k soft finger contacts (in the global coordinate system) will be the following:

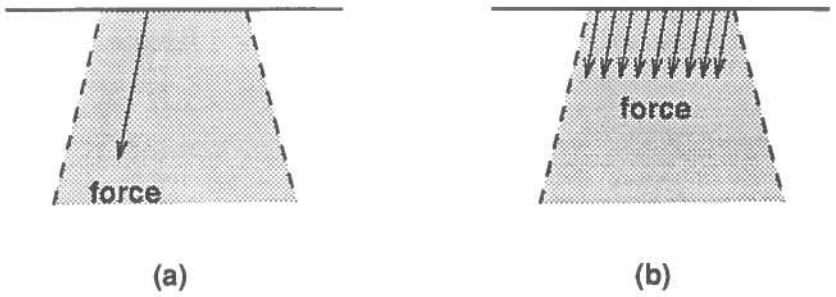


Figure 3.7: An example force & torque exerted by (a) linear approximation of soft finger model, (b) soft finger model.

$$\begin{pmatrix} \sum_{k=1}^{N_k} \sum_{b=1}^{4} f_{kb} \vec{v_{kb}} \\ \pm f_k \vec{n_k} + \sum_{k=1}^{N_k} \sum_{b=1}^{4} f_{kb} \vec{v_{kb}} \times (\vec{x_{kb}} - \vec{c}) \end{pmatrix} \tag{3.3}$$

where f_{kb} is the component of force exerted by finger k in direction of boundary vector $\vec{v_{kb}}$, $\vec{c}$ is the position vector of the centre of mass of the object, $\vec{x_{kb}}$ is the position vector of boundary vector $\vec{v_{kb}}$, and $f_k \vec{n_k}$ is the torque about the normal of the finger contact.

3.2 Solving the equations by linear programming

paragraph here

3.2.1 Conditions for equilibrium to exist

For a state of equilibrium to exist on an object grasp, all forces acting on the object must balance, and all torques acting on the object must balance.

Using equations (4.1), (4.2) and (4.3) and the additional gravitational force vector $m\vec{g}$, the condition for equilibrium can be described by the following equation :

$$m\vec{g} + \begin{pmatrix} \sum_{i=1}^{N_i} f_i \vec{n_i} \\ \sum_{i=1}^{N_i} f_i \vec{n_i} \times (\vec{x_i} - \vec{c}) \end{pmatrix} + \begin{pmatrix} \sum_{j=1}^{N_j} \sum_{b=1}^{4} f_{jb} \vec{v_{jb}} \\ \sum_{j=1}^{N_j} \sum_{b=1}^{4} f_{jb} \vec{v_{jb}} \times (\vec{x_j} - \vec{c}) \end{pmatrix}$$
$$+ \begin{pmatrix} \sum_{k=1}^{N_k} \sum_{b=1}^{4} f_{kb} \vec{v_{kb}} \\ \pm f_k \vec{n_k} + \sum_{k=1}^{N_k} \sum_{b=1}^{4} f_{kb} \vec{v_{kb}} \times (\vec{x_{kb}} - \vec{c}) \end{pmatrix} = 0 \tag{3.4}$$

The equation can be solved by finding a set of force and torque values such that :

$\sum (force\ values \times force\ directions) = 0$

$\sum (torque\ values \times torque\ directions) = 0$

where *force values* and *torque values* are scalar quantities, and are related to each other.

There are typically numerous solutions to this equation. The addition of upper and lower bound constraints on force and torque values limits the possible solutions to a subset of more sensible values.

3.2.2 Linear programming

The concept of *linear programming* refers to the maximisation of a linear function subject to a set of constraints. Using the theory and terminology from Press *et al* [5], the function to be maximised (the *objective function*) is of the form :

$$z = a_{01}x_1 + a_{02}x_2 + ... + a_{0N}x_N$$

where all the x_i's are non-negative (the primary constraint). The function can also be made subject to a set of additional constraints of the form :

$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{iN}x_N \leq b_i \geq 0$$

$$a_{j1}x_1 + a_{j2}x_2 + ... + a_{jN}x_N \geq b_j \geq 0$$

$$a_{k1}x_1 + a_{k2}x_2 + ... + a_{kN}x_N = b_k \geq 0$$

The process of maximising the function z can be thought of as the search for a solution coordinate in N-dimensional space, the boundaries of which are determined by the above constraints. Each inequality constraint separates the search space with a hyperplane such that valid solutions are restricted to one side of the plane (or on the plane itself). Each equality constraint defines a surface in the space such that valid solutions are restricted to lie on such a surface.

The constraints are used to define upper and lower bounds on the force values of the finger contacts, and to restrict the sum of force and torque values in the x, y and z planes to zero ($\pm$ tolerance value).

3.2.3 Restricting force values

For each frictionless finger contact i, the force value f_i of the finger contact is restricted with upper bound u_f and lower bound l_f such that :

$$f_i \leq u_f$$

$$f_i \geq l_f$$

For every frictional finger contact j, the force values f_{j1}, f_{j2}, f_{j3}, f_{j4} of the force range vectors are restricted such that :

$$f_{j1} + f_{j2} + f_{j3} + f_{j4} \leq u_f$$

$$f_{j1} + f_{j2} + f_{j3} + f_{j4} \geq l_f$$

ie. the sum of the force range values (which defines the total force value of the contact) is constrained by the upper bound on force values, u_f. Additionally the sum of force range values is constrained by the lower bound, l_f. This means that individual force values are allowed to be zero, which is fine because not all force range values need contribute to the total force value of the contact.

For each soft finger contact k, the force values f_{k1}, f_{k2}, f_{k3}, f_{k4}, f_k of the force range vectors are restricted such that :

$$f_{k1} + f_{k2} + f_{k3} + f_{k4} \leq u_f$$

$$f_{k1} + f_{k2} + f_{k3} + f_{k4} \geq l_f$$

$$f_k \leq u_f$$

The reasons for constraints on f_{k1}, f_{k2}, f_{k3}, f_{k4} are the same as for frictional point contacts. The third constraint limits f_k, which represents the resistance to torque about the surface normal of the finger.

3.2.4 Constraining the total force and torque values

The restriction of total force and torque values of all fingers, as defined by equation (4.4) is achieved by considering the component's total forces and total torques separately in their x, y and z planes. The result of this is the follwing twelve inequality constraints :

$$f_i \vec{n_{ix}} + f_{j1}\vec{v_{j1x}} + f_{j2}\vec{v_{j2x}} + f_{j3}\vec{v_{j3x}} + f_{j4}\vec{v_{j4x}} + f_{k1}\vec{v_{k1x}} + f_{k2}\vec{v_{k2x}} + f_{k3}\vec{v_{k3x}} + f_{k4}\vec{v_{k4x}} + \vec{g_x} \leq \epsilon$$

$$f_i \vec{n_{iy}} + f_{j1}\vec{v_{j1y}} + f_{j2}\vec{v_{j2y}} + f_{j3}\vec{v_{j3y}} + f_{j4}\vec{v_{j4y}} + f_{k1}\vec{v_{k1y}} + f_{k2}\vec{v_{k2y}} + f_{k3}\vec{v_{k3y}} + f_{k4}\vec{v_{k4y}} + \vec{g_y} \leq \epsilon$$

$$f_i \vec{n_{iz}} + f_{j1}\vec{v_{j1z}} + f_{j2}\vec{v_{j2z}} + f_{j3}\vec{v_{j3z}} + f_{j4}\vec{v_{j4z}} + f_{k1}\vec{v_{k1z}} + f_{k2}\vec{v_{k2z}} + f_{k3}\vec{v_{k3z}} + f_{k4}\vec{v_{k4z}} + \vec{g_z} \leq \epsilon$$

$$f_i \vec{n_{ix}} + f_{j1}\vec{t_{j1x}} + f_{j2}\vec{t_{j2x}} + f_{j3}\vec{t_{j3x}} + f_{j4}\vec{t_{j4x}} + f_{k1}\vec{t_{k1x}} + f_{k2}\vec{t_{k2x}} + f_{k3}\vec{t_{k3x}} + f_{k4}\vec{t_{k4x}} + f_k\vec{n_{kx}} \leq \epsilon$$

$$f_i \vec{n_{iy}} + f_{j1}\vec{t_{j1y}} + f_{j2}\vec{t_{j2y}} + f_{j3}\vec{t_{j3y}} + f_{j4}\vec{t_{j4y}} + f_{k1}\vec{t_{k1y}} + f_{k2}\vec{t_{k2y}} + f_{k3}\vec{t_{k3y}} + f_{k4}\vec{t_{k4y}} + f_k\vec{n_{ky}} \leq \epsilon$$

$$f_i\vec{n}_{iz} + f_{j1}\vec{t}_{j1z} + f_{j2}\vec{t}_{j2z} + f_{j3}\vec{t}_{j3z} + f_{j4}\vec{t}_{j4z} + f_{k1}\vec{t}_{k1z} + f_{k2}\vec{t}_{k2z} + f_{k3}\vec{t}_{k3z} + f_{k4}\vec{t}_{k4z} + f_k\vec{n}_{kz} \leq \epsilon$$

$$-f_i\vec{n}_{ix} - f_{j1}\vec{v}_{j1x} - f_{j2}\vec{v}_{j2x} - f_{j3}\vec{v}_{j3x} - f_{j4}\vec{v}_{j4x} - f_{k1}\vec{v}_{k1x} - f_{k2}\vec{v}_{k2x} - f_{k3}\vec{v}_{k3x} - f_{k4}\vec{v}_{k4x} - \vec{g}_x \leq \epsilon$$

$$-f_i\vec{n}_{iy} - f_{j1}\vec{v}_{j1y} - f_{j2}\vec{v}_{j2y} - f_{j3}\vec{v}_{j3y} - f_{j4}\vec{v}_{j4y} - f_{k1}\vec{v}_{k1y} - f_{k2}\vec{v}_{k2y} - f_{k3}\vec{v}_{k3y} - f_{k4}\vec{v}_{k4y} - \vec{g}_y \leq \epsilon$$

$$-f_i\vec{n}_{iz} - f_{j1}\vec{v}_{j1z} - f_{j2}\vec{v}_{j2z} - f_{j3}\vec{v}_{j3z} - f_{j4}\vec{v}_{j4z} - f_{k1}\vec{v}_{k1z} - f_{k2}\vec{v}_{k2z} - f_{k3}\vec{v}_{k3z} - f_{k4}\vec{v}_{k4z} - \vec{g}_z \leq \epsilon$$

$$-f_i\vec{n}_{ix} - f_{j1}\vec{t}_{j1x} - f_{j2}\vec{t}_{j2x} - f_{j3}\vec{t}_{j3x} - f_{j4}\vec{t}_{j4x} - f_{k1}\vec{t}_{k1x} - f_{k2}\vec{t}_{k2x} - f_{k3}\vec{t}_{k3x} - f_{k4}\vec{t}_{k4x} - f_k\vec{n}_{kx} \leq \epsilon$$

$$-f_i\vec{n}_{iy} - f_{j1}\vec{t}_{j1y} - f_{j2}\vec{t}_{j2y} - f_{j3}\vec{t}_{j3y} - f_{j4}\vec{t}_{j4y} - f_{k1}\vec{t}_{k1y} - f_{k2}\vec{t}_{k2y} - f_{k3}\vec{t}_{k3y} - f_{k4}\vec{t}_{k4y} - f_k\vec{n}_{ky} \leq \epsilon$$

$$-f_i\vec{n}_{iz} - f_{j1}\vec{t}_{j1z} - f_{j2}\vec{t}_{j2z} - f_{j3}\vec{t}_{j3z} - f_{j4}\vec{t}_{j4z} - f_{k1}\vec{t}_{k1z} - f_{k2}\vec{t}_{k2z} - f_{k3}\vec{t}_{k3z} - f_{k4}\vec{t}_{k4z} - f_k\vec{n}_{kz} \leq \epsilon$$

where the f_is correspond to frictionless point contacts, f_js to frictional point contacts, and f_ks to soft finger contacts.

The first three of these constraints restrict the sum of force values in their x, y and z directions to be less than or equal to ϵ, the tolerance value. The next three restrict the sum of torque values in their x, y and z directions to be less than or equal to ϵ. The remaining six constraints are similar to the first six, but restrict the sums to be greater than or equal to the negative of ϵ.

3.2.5 Defining a good objective function

The objective function is the one that linear programming tries to maximise. The objective function need not be linear; Hillier & Lieberman [2] describe how to implement a quadratic objective function by exploiting a property of logarithmic addition. The concern is this — what values should the coefficients of the function be allocated in order to achieve an equilibrium with optimal force values?

A sensible idea is to want to minimise the sum of all force magnitudes acting on the object. The justification of this is that if a grasp is in equilibrium, but unstable such that a slight peturbation of the object results in a slip, if the force magnitudes acting on the object were minimal the slip would be generally less erratic than would be the case if the force magnitudes were stronger.

A different idea is to want to keep the force directions of fingers with friction as near to the centre of their friction cones as possible. This is justified by the assumption that force directions near the centre of the friction cones are generally safer than force directions near

the boundary of the friction cones, which are on the verge of slipping. This can be achieved by ensuring that the coefficients of all force directions defining a friction cone are equal.

The two ideas above both make the assumption that all of the finger contacts play an equally important role in the outcome of a grasp. It could be the case that one finger contact is of more importance than the others in a particular grasp, and its attributes should be minimised (or maximised) at the expense of others. Deciding which contacts are of greater importance for arbitrary object grasps implies a need for empirical knowledge about each grasp.

The final decision was to live with the assumption that all finger contacts were of equal importance, and set the coefficients of all force values in the objective function to -1. This minimises the sum of all force values (as maximising negative coefficients = minimising positive coefficients), and attempts to centralise force directions within their friction cones.

3.2.6 Using the simplex algorithm

The equations defining a state of equilibrium can be solved using a method first published by Dantzig [1] called the *simplex method.* In order to use the simplex algorithm the equations need to be transformed into restricted normal form. The transformation is explained in detail in Press *et al* [5]; it basically involves transforming inequality constraints into equality constraints (normal form) by introducing artificial variables to the equations, and ensuring that each constraint contains a unique variable with a positive coefficient (restricted normal form).

The simplex code used in the analysis is taken directly from Press *et al* [5], and is based on the implementation of Kuenzi *et al* [3].

When the algrorithm is executed it returns with a flag whose value indicates one of three possibilities :

- A finite solution has been found.
- The objective function is unbounded.
- No solution satisfies the constraints.

As a result of the stringent constraints placed on the force values of each finger contact, the second of the three possibilities should never arise in the analysis.

3.2.7 Postprocessing to retrieve data

If a finite solution is found by the simplex algorithm for an object grasp, the magnitude and direction of force and torque values for each finger contact can be calculated from the resultant simplex data. For each variable in the original objective function, if the resultant coefficient is

non-zero, there is an index to it returned by the simplex algorithm. If the resultant coefficient is zero, it is in a different index and is not needed in the solution.

For each frictionless point finger contact i, the resultant force $\vec{F_i}$, and torque $\vec{T_i}$, is calculated by multiplying the indexed coefficient f_i by the force and torque directions :

$$f_i\vec{n_i} = \vec{F_i}$$

$$f_i\vec{t_i} = \vec{T_i}$$

For each frictional point finger contact j, the resultant force $\vec{F_j}$, and torque $\vec{T_j}$, is calculated by multiplying the indexed coefficients f_{j1}, f_{j2}, f_{j3}, f_{j4} by their respective force and torque directions, and summing the totals :

$$f_{j1}\vec{v_{j1}} + f_{j2}\vec{v_{j2}} + f_{j3}\vec{v_{j3}} + f_{j4}\vec{v_{j4}} = \vec{F_j}$$

$$f_{j1}\vec{t_{j1}} + f_{j2}\vec{t_{j2}} + f_{j3}\vec{t_{j3}} + f_{j4}\vec{t_{j4}} = \vec{T_j}$$

The resultant force $\vec{F_k}$, and torque $\vec{T_k}$ of soft finger contacts are calculated in the same way as frictional point contacts. The soft finger contact also posesses a resultant torque about its surface normal, $T\vec{n_k}$:

$$f_{k1}\vec{v_{k1}} + f_{k2}\vec{v_{k2}} + f_{k3}\vec{v_{k3}} + f_{k4}\vec{v_{k4}} = \vec{F_k}$$

$$f_{k1}\vec{t_{k1}} + f_{k2}\vec{t_{k2}} + f_{k3}\vec{t_{k3}} + f_{k4}\vec{t_{k4}} = \vec{T_k}$$

$$f_k\vec{n_k} = T\vec{n_k}$$

3.2.8 Verifying results produced by the simplex algorithm

The simplex algorithm returns from execution with either a finite solution for equilibrium or a flag signalling that no solution was found that satisfies the constraints.

The situation where a finite solution is found can be verified by summing the resultant force and torque magnitudes and directions, and checking that they equal zero ($\pm\epsilon$).

The second (no solution) situation is more difficult to verify. A possible verification would be to run the same problem on a mathematically proven linear programming system and check that no solution can be found on this system also.

3.3 Experimentation

The intention of the experiments for the equilbrium analysis system was to design a set of systematic test data to examine the effectiveness of the three types of finger contact in

producing an equilibrium grasp on various types of object data.

The variables used in the experimentation include :

- Frictionless fingers, frictional point fingers, soft fingers with friction.
- Varied coefficients of friction for frictional fingers.
- Variety of different contact positions.
- Varied numbers of fingers.

The example test object used in the experimentation was a planar cube. The main purpose of the test was to compare the performance of the three finger types over a number of contact configurations, ranging from single contact points opposing gravity, to parallel opposing contact positions in the same plane as the centre of mass of the object, to parallel opposing contact positions in planes other than the centre of mass of the object, to non-parallel opposing contact positions in planes other than the centre of mass of the object. Example diagrams of these configurations can be seen in Figure 3-8.

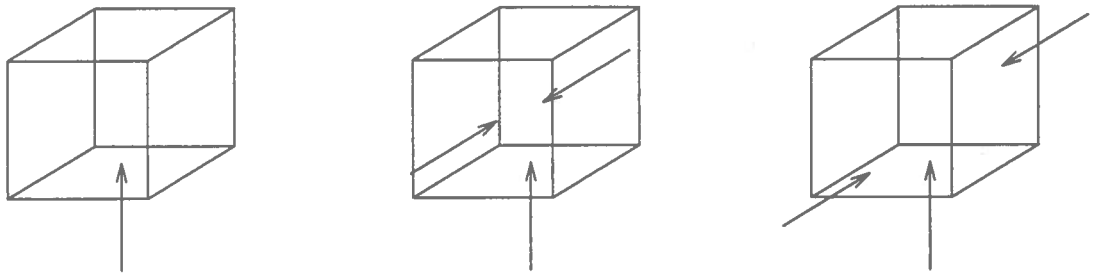


Figure 3.8: Example contact configurations — (a) single contact opposing gravity, (b) parallel opposing contacts in same plane as the centre of mass of the object & contact opposing gravity, (c) parallel opposing contacts in plane other than the centre of mass of the object & contact opposing gravity.

The results of the experimentation proved the following :

- For all three finger types, a minimum of one contact can be used to oppose gravity.
- The greater the coefficient of friction for frictional contacts and soft finger contacts, the better.
- Generally, frictionless point finger contact were the least effective of the three, frictional point contacts were much better, but soft finger contacts were the most impressive (for the same value of friction as frictional point contacts).

Results of the experimentation for equilibrium analysis can be found in Appendix A.

3.4 Discussion and summary

The equilibrium analysis system was implemented with a fair amount of success. The program achieves what is required of it in the sense that it states whether or not there is a possible equilibrium situation for arbitrary object grasps. It also returns a magnitude and direction of force and torque values for each finger contact in the analysis.

Possible improvements and further work could be done in the following areas of the equilibrium analysis system :

- The analysis system could be modified to model grasping of objects via edge and vertex contacts as well as the current grasping of surface contacts.
- The system could be updated from a linear one to a higher order one in order to remove the need for linear approximations.
- The soft finger model could be made more accurate by incorporating an elastic membrane model so deformations of the finger around surface contact patches could be modelled.

To summarise, in this chapter we have seen how the equilibrium analysis of a grasp involves the conversion of a world model into a sequence of linear equations, and through linear programming it can be determined whether the equations have a solution or not. If a solution is available, the object grasp is capable of static equilibrium, and an optimum solution can be searched for. If no solution is found then the grasp can not be capable of static equilibrium.

Chapter 4

Stability analysis

Chapter 3 explained how a grasp can be tested for equilibrium. An equilibrium grasp is satisfactory for an object to be held stationary with no external forces other than gravity acting on it, but testing for equilibrium is not enough if the object is to undergo even the slightest movements. If an object in the state of equilibrium is moved it may slip, as any movement introduces an accelerational force on the object. The purpose of stability analysis is to assess whether or not the object grasp is likely to remain in equilibrium despite small changes in external forces, ie. how *stable* the object is.

4.1 Using the gravity vector to test stability

The gravity vector $\vec{g}$ is viewed as an external force acting on the object. Other external forces can be modelled by altering the direction of the gravity vector. For example, if the gravity vector of a particular object grasp lies in a negative z direction such that $\vec{g} = (0, 0, -9.81)$, and it is desired to add a slight force to this object grasp of magnitude f_x in a positive x direction, it can be added to $\vec{g}$ to form $\vec{g'}$ such that the sum of external forces on the object is $\vec{g'} = (f_x, 0, -9.81)$. In this way any number of arbitrary external forces on an object by adding to the gravity vector the sum of such forces.

Assessing the gravitational stability of an object grasp involves the assessment of the security of the object grasp throughout a range of directions of gravity. The range of directions is defined by a *cone of stability* of angle θ_s, where θ_s is the maximum angle through which the gravitational stability of an object is analysed. Figure 4-1 shows an example cone of stability for initial gravity vector $\vec{g} = (0, 0, -9.81)$.

The level of stability of an object grasp can be analysed by performing numerous perturbations of $\vec{g}$ within the cone of stability and testing the equilibrium of the object grasp for each perturbation. For a stable grasp an equilibrium will hold for all perturbations of $\vec{g}$ within the stability cone. For an unstable grasp there are typically regions within the stability cone

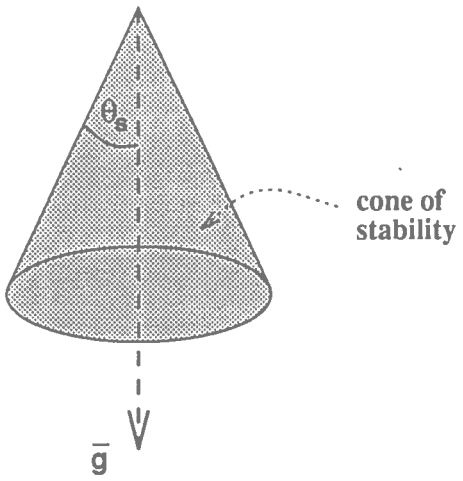


Figure 4.1: Example cone of stability for $\vec{g} = (0, 0, -9.81)$.

within which equilibrium cannot be satisfied. These regions describe directions of force that the object grasp is unable to counteract.

4.2 Implementation of stability analysis

Stability analysis basically involves the repetition of the following process :

- Perturb the original gravity vector $\vec{g}$ by a precise amount to obtain a new gravity vector $\vec{g'}$.
- Analyse the equilibrium of the grasp with the replacement of $\vec{g}$ by $\vec{g'}$.

The perturbation of $\vec{g}$ should systematically envelop the range of the cone of stability. There is a trade off here between the accuracy of the analysis and the time taken to achieve the analysis — the smaller the perturbations of $\vec{g}$ the greater the accuracy of the analysis, but also the greater the number of repetitions required to envelop the cone of stability.

The method of systematically perturbing the gravity vector is the following :

1. Rotate $\vec{g}$ by angle θ_1 about the centre of mass of the object $\vec{c}$ in an arbitrary plane to obtain a vector $\vec{g_2}$.
2. Rotate $\vec{g_2}$ by angle θ_2 about the original gravity vector $\vec{g}$ to obtain $\vec{g'}$.
3. Perform equilibrium analysis with new gravity vector $\vec{g'}$.

where (1.) is repeated for the range $\theta_1 = 0$ to θ_s in steps of c_1, (c_1 = a constant which divides θ_s exactly), and (2.) is repeated for the range $\theta_2 = c_2$ to 360° in steps of c_2, (c_2 = a constant

which divides 360 exactly). Figure 4-2 represents such perturbations for an example gravity vector $\vec{g}$.

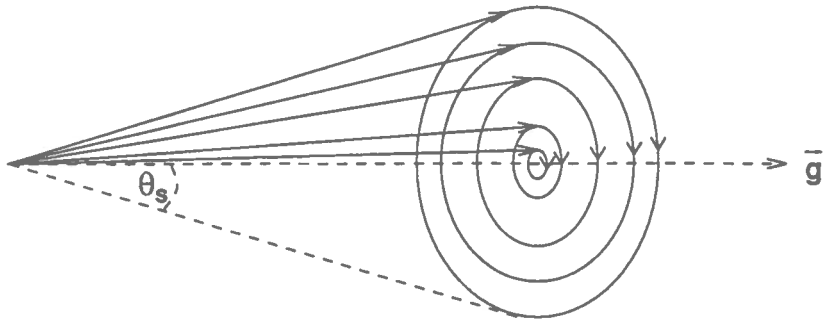


Figure 4.2: Testing for equilibrium within a cone of stability for an example $\vec{g}$.

To allow the visualisation of the stability of the object grasp the system produces a chart of the analysis. Each element in the chart is either a *o* or a *x* depending on the outcome of the equilibrium analysis for the element's respective gravity vector. The vertical axis of the chart represents the angle between the original gravity vector $\vec{g}$ and the perturbed gravity vector $\vec{g'}$ (which relates to the size of external force on the grasp), and the horizontal axis represents the rotation of $\vec{g'}$ about $\vec{g}$ (which relates to the direction of external force on the grasp). This assists the user in gaining a visual image of the stability of the grasp by inspecting the patterns of stable regions in the chart.

The stability analysis system also returns an absolute value of stability in terms of the percentage of equilibrium solutions during the analysis, which is useful for comparing the relative stability of object grasps.

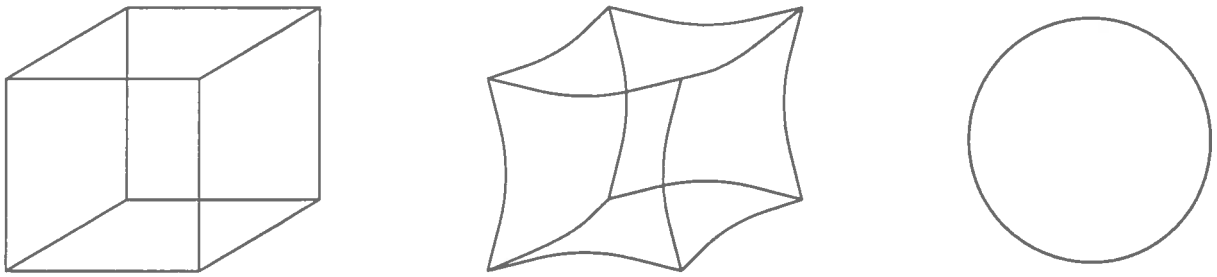
4.3 Experimentation

The intention of the experiments for the stability analysis system was to design a set of systematic test data to examine the effectiveness of the three types of finger contact in producing an stable grasps on various types of object data.

The variables used in the experimentation include :

- Frictionless fingers, frictional point fingers, soft fingers with friction.
- Varied coefficients of friction for frictional fingers.
- Variety of different contact positions.
- Varied numbers of fingers.

The example test objects used were a planar cube, a concave cube, and a sphere. Figure 4-3 shows examples of these three test objects. The main purpose of the tests were to establish the level of stability each grasp configuration could produce, and to test whether stable grasps of lower quality finger types (eg. frictionless point finger contacts) could be achieved by higher quality finger types (eg. soft finger contacts) with less finger contacts.



(a) (b) (c)

Figure 4.3: Example object types for stability analysis — (a) planar cube, (b) concave cube, (c) sphere.

The results of the experimentation proved the following :

- A small number of finger contacts (eg. 3) with high friction result in grasps as stable, if not more stable than a higher number (eg. 5 or 6) of frictionless finger contacts or finger contacts of relatively low friction.
- A good choice of contact positions makes a noticeable difference to the stability of the object grasp.
- In the case of soft finger contacts, concave surfaces patches generally resulted in more stable grasps than their planar counterparts did.

Results of the experimentation for equilibrium can be found in Appendix A.

4.4 Discussion and summary

The implementation of the stability analysis system was straightforward mainly because of the fact that it relies heavily on the success of the equilibrium analysis system. It allows the user to perceive a visual representation of the regions of stability and instability of an object grasp by displaying a chart, as well as displaying a percentage of the regions where static equilibrium is found during the analysis.

There are a few ways in which the system could be improved :

- A graphical interface could be written to display the state of the perturbation of $\vec{g}$ with respect to the object on a standard graphics package. Together with the stability chart already produced by the system this would allow the user to gain a clear idea about the stability of different configurations of object grasp.

- The perturbation of gravity vector $\vec{g}$ cannot produce external torques about the object's centre of mass, as it acts at the centre of mass of the object. This situation could be modified so that the stability of the object grasp is tested with external torques on the object as well as the existing external forces.

As a summary to this chapter, we have seen how the gravitational stability of the object can be assessed by performing numerous perturbations of the angle of the gravity vector $\vec{g}$ with respect to the object and assessing at each stage whether or not a static equilibrium is possible. The experimentation results showed the benefits of using soft finger contacts over the weaker finger contact types in terms of object grasp stability.

Chapter 5

Dynamic Analysis

If the equilibrium analysis of an object grasp signifies no solution for equilibrium, then the object will not remain stationary and will slip with respect to the finger contacts. It may be the case that the slip is such that the object only moves a small distance before coming to rest in an equilibrium position. Conversely the slip may be more severe and the object may move to such a position that finger contacts are no longer in range of their surface patches, in which case the grasp is deemed to have failed.

The purpose of the dynamic analysis system is to simulate the motion of the object as it slips and determine whether the contact geometries stop the slip or whether the object slips to a position or orientation out of range of the grasp. Figure 5-1 shows the operational structure of the dynamic analysis system.

5.1 Calculating the accelerational force and torque of the object

The accelerational force and torque of the object can be computed using a modified version of the equilibrium constraints used in the equilibrium analysis (see Section 4.2.4). Instead of constraining the force and torque values such that :

$$\sum (force\ values \times force\ directions) = 0$$

$$\sum (torque\ values \times torque\ directions) = 0$$

the constraints are modified such that :

$$\sum (force\ values \times force\ directions) = \vec{f_n}$$

$$\sum (torque\ values \times torque\ directions) = \vec{t_n}$$

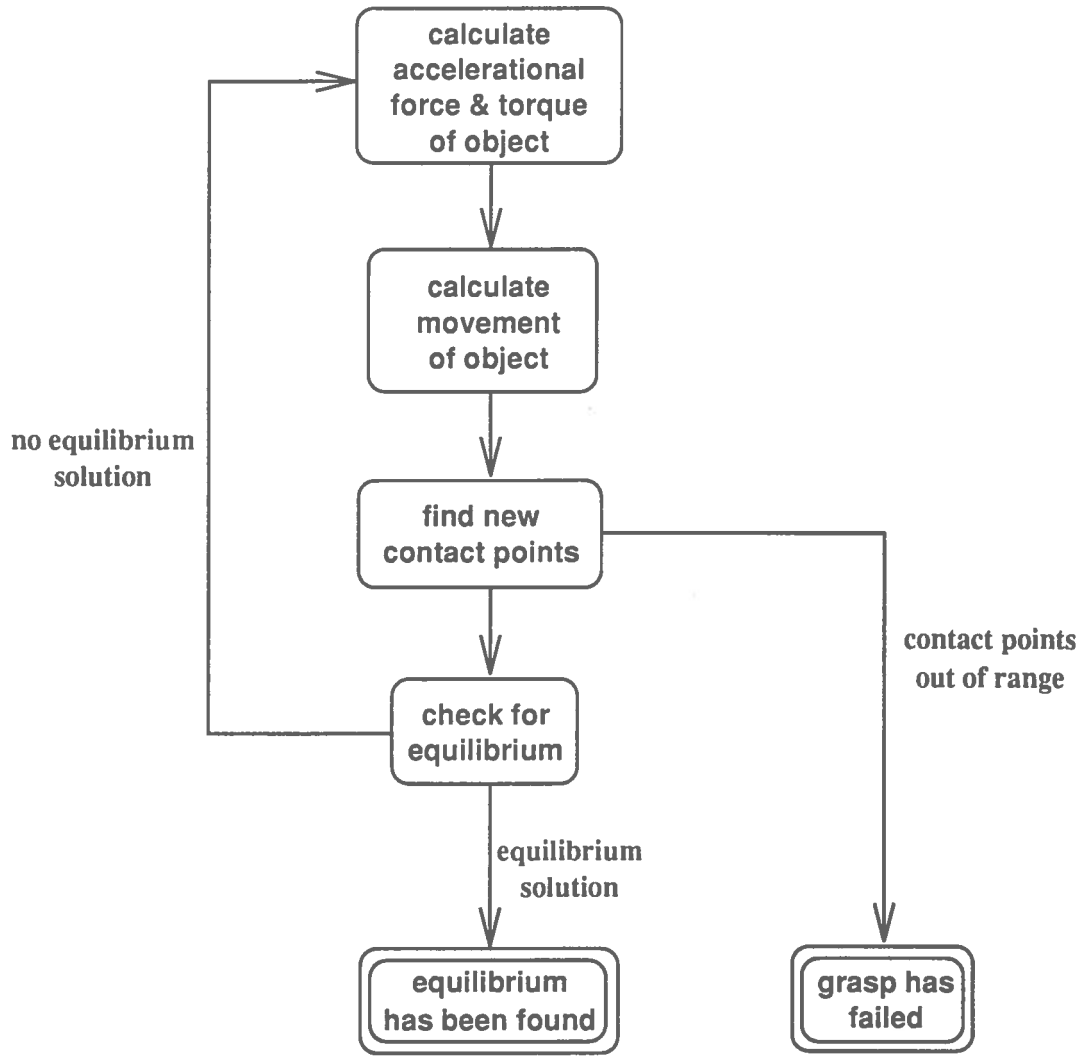


Figure 5.1: Structure of the dynamic analysis system.

where $\vec{f_n}$ is the resultant accelerational force and $\vec{t_n}$ is the resultant accelerational torque of the object at the n^{th} iteration of stability analysis.

The upper bound constraint u_f is used to restrict $\vec{f_n}$ and $\vec{t_n}$ to sensible values. As long as u_f is large enough the simplex routine will produce a finite solution for these new constraints. The reason for this is that instead of constraining the sum of force values to zero ($\pm\epsilon$), they are now constrained to lie within the upper bound limit such that :

$$-u_f \leq \sum force\ values \leq u_f$$

$$-u_f \leq \sum torque\ values \leq u_f$$

The objective function, z is also modified such that $\vec{f_n}$ and $\vec{t_n}$ are minimised, as opposed to the force values of the finger contacts which were minimised in the equilibrium analysis. The reason for wanting to minimise the resultant accelerational force and torque on the object is that in order to attempt to find an eventual equilibrium from the slip it is better to be optimistic and assume a minimal slip on the object rather than a slip with high acceleration.

5.2 Calculating the resultant object movement

The accelerational force and torque values acting on the object can be used to calculate the resultant movement of the object. The motion of the object can be decomposed into a translation (of the centre of mass) and a rotation (about the centre of mass) of the object.

5.2.1 Translational motion of the object

The translational motion of the object can be calculated with the use of a standard physical law of motion :

$$\Delta\vec{x_n} = \vec{\dot{x}_n}\Delta t + \frac{1}{2}\vec{\ddot{x}_n}(\Delta t)^2$$

where n is the current iteration of the analysis, $\Delta\vec{x_n}$ is the change in displacement since the previous iteration, $\vec{\dot{x}_n}$ is the velocity of the object at the current iteration, $\vec{\ddot{x}_n}$ is the acceleration of the object at the current iteration, and Δt is the increment of time since the previous iteration.

The velocity of the object at the current iteration, $\vec{\dot{x}_n}$ can be calculated by :

$$\vec{\dot{x}_n} = \vec{\dot{x}_{n-1}} + \vec{\ddot{x}_{n-1}}(\Delta t)$$

where $\vec{\dot{x}_{n-1}}$ is the velocity of the object at the previous iteration and $\vec{\ddot{x}_{n-1}}$ is the acceleration of the object at the previous iteration.

The acceleration of the object, $\vec{\ddot{x}_n}$ at the current iteration is calculated by :

$$\vec{\ddot{x}_n} = \vec{g} + \vec{f_n}/m$$

where $\vec{g}$ is the acceleration due to gravity, and $\vec{f_n}$ is the resultant accelerational force at the n^{th} iteration of the simplex routine.

Initially (at the 0^{th} iteration) the values are :-

$$\vec{x_0} = \vec{c}$$

$$\vec{x_0} = 0$$

$$\vec{\ddot{x}_0} = \vec{g} + \vec{f}_0 / m$$

which means that initially the centre of mass of the object is at its origin, its velocity is zero, and its acceleration is the sum of the acceleration due to gravity and the resultant force value from the first execution of the simplex routine in the analysis.

5.2.2 Rotational motion of the object

The rotation of the object about the centre of mass of the object can be calculated with the aid of the following formula :

$$\vec{t_n} = I\vec{\dot{\Omega}_n}$$

where $\vec{t_n}$ is the resultant torque about the centre of mass of the object at the n^{th} iteration of the simplex routine, I is the moment of inertia matrix of the object (see Section 3.1.3), and $\vec{\dot{\Omega}_n}$ is the angular acceleration about the centre of mass of the object at the n^{th} iteration of the analysis.

The moment of inertia matrix of the object I, assuming a unitary mass of the object (Chapter 1) is the following :

$$I = \frac{1}{\sum_j 1} \begin{pmatrix} \sum({r_j}^2 - {x_j}^2) & -\sum x_j y_j & -\sum x_j z_j \\ -\sum x_j y_j & \sum({r_j}^2 - {y_j}^2) & -\sum y_j z_j \\ -\sum x_j z_j & -\sum y_j z_j & \sum({r_j}^2 - {z_j}^2) \end{pmatrix}$$

where ${r_j}^2 = {x_j}^2 + {y_j}^2 + {z_j}^2$, and j is a segmentation of the object in the x, y, and z planes.

In order to calculate the angular acceleration of the object, $\vec{\dot{\Omega}_n}$, where :

$$\vec{\dot{\Omega}_n} = I^{-1}\vec{t_n}$$

the inverse matrix of the inertia matrix, I^{-1} is needed. The inverse matrix can be calculated if inertia matrix I is *non-singular* (ie. has a none zero determinant). Generally I^{-1} can be calculated successfully, there are only a minority of cases (for example if the object to be grasped was an infinitesimally thin disc) where I^{-1} can not be computed, so for the sake of the analysis, the assumption is made that the shape of the object is such that the inverse of its moment of inertia matrix can be computed.

The change in angular rotation of the object about its centre of mass at the n^{th} iteration of the analysis, $\Delta\vec{\Omega_n}$ can be calculated with the use of the following formula :

$$\Delta\vec{\Omega_n} = \dot{\vec{\Omega}}_n \Delta t + \frac{1}{2}\ddot{\vec{\Omega}}_n (\Delta t)^2$$

where $\dot{\vec{\Omega}}_n$ is the angular velocity of the object at the current iteration and Δt is the increment of time since the previous iteration.

The angular velocity about the centre of mass of the object at the current iteration, $\dot{\vec{\Omega}}_n$ is calculated by :

$$\Delta\dot{\vec{\Omega}}_n = \dot{\vec{\Omega}}_{n-1} + \ddot{\vec{\Omega}}_{n-1}(\Delta t)$$

ie. the current angular velocity is the previous angular velocity with the addition of the angular velocity gain from the angular acceleration between the previous iteration and the current iteration.

Initially (at the 0^{th} iteration) the rotational motion variables are such that :

$$\vec{\Omega_0} = 0$$

$$\dot{\vec{\Omega}}_0 = 0$$

$$\ddot{\vec{\Omega}}_0 = I^{-1}\vec{t_0}$$

Once $\Delta\vec{x_n}$ and $\Delta\vec{\Omega_n}$ have been established a transformation matrix, M_n can be formed with the use of *canonical rotations* (ie. about x, y and z axes) which premultiply the previous transformation matrix, M_{n-1} to give a transformation of the object from its original position to its new position. Initially the object is in its original position the initial transformation matrix, M_0 is the identity matrix.

5.3 Determining the positions of the new contact points

Now that the transformation matrix has been calculated, the new positions of the finger contact points on there patches can be calculated.

There are numerous possible ways in which the finger contacts may move in relation the movement of the object. For example, the finger joints may be flexible and if the movement of the object produces sufficient force the finger may move in a similar direction to its contact patch. Conversely the finger joints may be rigid, and the object patch drags along the finger contact as it slips as the finger remains stationary. As the project considers a grasp as a set of disembodied finger contacts, no information is known about the geometries of the finger joints, only the fingertips are considered. As a result, the assumption is made that each finger

can only move along the line of force that it exerts, so if the contact patch slips away from the finger joint, the finger joint moves in a forward direction along its line of exerted force to come into contact with the patch. If the patch slips into the finger joint, the finger joint moves in a backward direction along its line of force until the finger is in contact with the surface of the patch.

A problem arises in the use of the transformation matrix M_n to reorientate the object in its new position. The problem is due to the use of local contact patches and transformations to a global coordinate position. The transformation matrix M_n is specified in global coordinates, whereas the contact patches $\vec{b_i}$ are specified in local coordinates in terms of a biquadratic equation. The biquadratic patch cannot be transformed in its local coordinate system, as it is not an (x, y, z) vector, only points on the patch can be transformed using M_n by transforming them with T_i, then transforming them by the object movement matrix M_n to give the new position of the point in global coordinates, then transforming back to local coordinates by using the inverse transformation matrix T^{-1}.

The solution to this problem is not to transform the biquadratic patch at all, but to move the finger contacts and the gravity vector inversely to the transformation of the patch. Figure 5-2 shows an example where instead of transforming the object with respect to the contacts, the contacts (and the gravity vector, $\vec{g}$) can be transformed with repect to the object and the result is the same.

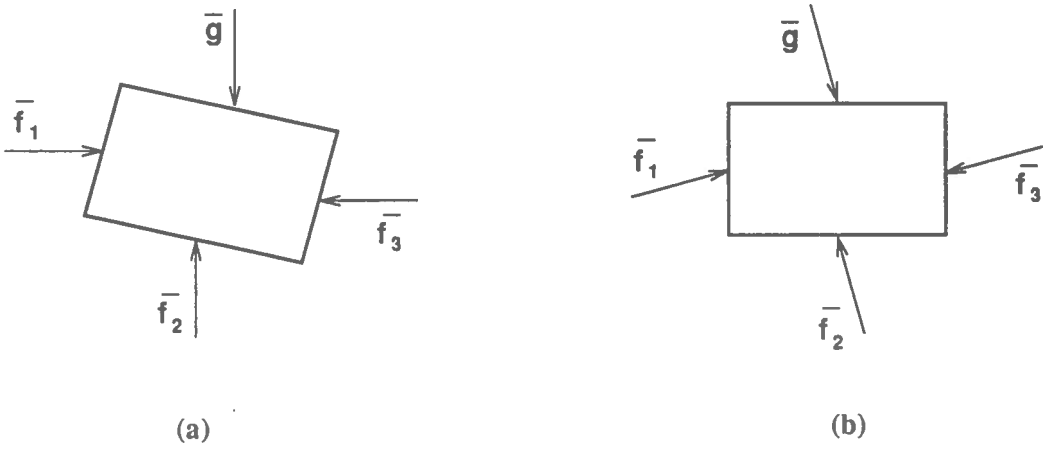


Figure 5.2: Equivalent object grasps with fingers $\vec{f_1}$, $\vec{f_2}$, $\vec{f_3}$ and gravity $\vec{g}$ for (a) object transformed, contacts and gravity static, (b) contacts and gravity transformed, object static.

This involves the use of inverse transformation matrices, as instead of transforming the biquadratic equation in global coordinates by matrix M_n, the finger points and gravity are transformed by the inverse matrix $M_n{}^{-}1$, (although gravity is not translated, only rotated). So the resultant transformation for a finger contact point $\vec{p_i} = (x, y, z)$ in its local coordinate

frame can be achieved by performing the following transformations :

$$\vec{q_i} = T^{-1} {M_n}^{-1} T(\vec{p_i})$$

where $\vec{q_i}$ is the resultant contact point in local coordinates.

The problem with the use of the above transformations is that the inverse transformation matrices T^{-1} and ${M_n}^{-1}$ might not exist. A possible solution to this problem is to define all contact patches and contact points in terms of a global coordinate system as opposed to a local coordinate system with transformation to global coordinates, although this raises more problems in the sense that the reason for representing contact patches in their individual local coordinate systems in the first place was that patches could be modelled relatively accurately using biquadratic equations in local coordinates, whereas if global coordinates were used, the equations would be significantly more complicated. The severity of this problem was such that, due to a limited time period within which the problem had to be solved, an operational version of the dynamic analysis system was not implemented. Despite this fact I will explain the remaining theory behind the dynamic analysis system, with the assumption that the contact points were successfully translated.

Once the position of the object patches with respect to the finger contacts points have been established, the next step is to move the position of the contact points such that they lie on the patch. This is achieved by performing the following operation for each finger contact :

1. Find the resultant line of force exerted by each finger contact.
2. Intersect this line of force with the surface patch to find the new contact point.
3. If the new contact point lies out of range of the patch then the object is deemed to have slipped.

5.4 Checking for equilibrium

If the contact points still lie within range of their contact patches, defined by d (see Chapter 2), then the current object grasp orientation can be tested for equilibrium. This can be achieved with the aid of the equilibrium analysis system (see Chapter 3).

5.5 Expermimentation

Unfortunately, because the dynamic analysis system was not operational, no experimentation could be performed on it. However, if the system was working, the main set of experimentation would be to test situations where a slip would eventually result in a state of static equilibrium, and where a slip would result in the local contact points being out of range, ie. a failed grasp.

These situations would be tested on a variety of objects (ie. concave, convex, planar, etc.) to determine which objects seem to be most robust in the case of slippage. Also the situations would be tested on different finger types, and different contact configurations in order to see the effect such differences have on slipping.

5.6 Discussion and summary

The theory behind the dynamic analysis system was relatively straightforward. The implementation issues were somewhat more complicated however. If a global coordinate system was implemented for equations of surface patches as opposed to a translation from local to global coordinates these implementational issuses would be eased somewhat. Due to the limited time available during the commencement of the programming of the dynamic analysis system a new design was not implemented, one of the reasons for this being that if the coordinate systems were changed, they would have to be changed in the equilibrium analysis sytem too.

To summarise, Chapter 6 has been concerned with the dynamic analysis system. This system attempted to simulate the slip of the object with respect to its grasp, and report whether the object grasp reached a position of static equilibrium or whether the grasp failed completely. The basic ingredients of the system were to iteratively simulate the motion of the object over a discrete time step, calculate the new contact positions at this time step, and check for static equilibrium or grasp failure.

Chapter 6

Conclusions and discussion

The end result of this project was a partially completed system. Two of the analysis programs were complete, and one was not functional due to reasons explained in Chapter 5 of this report. Despite the setback of the dynamic analysis, the system as it stands still performs a useful part in the analysis of object grasps.

There is a multitude of possible further work in the extension of this system. Here are a few possibilities :

- Improve the accuracy of the finger model approximations (for example, soft finger modelling could be improved by the use of elastic membrane simulations).
- Integrate the analysis systems into other robot related systems, for example, a system that scans objects and produces a set of possible grasp configurations (eg. Wren [6]).
- Introduce a system that analyses the relative importance of each finger contact in an object grasp by using a modification of the stability analysis program, which can then decide which contacts should be given preference in maximisation of the objective function used in equilibrium analysis.
- Add a graphical interface for the system so that object grasps, stability analysis, and dynamic analysis are displayed visually in 3-dimensions.

Bibliography

[1] Dantzig, G.B. Linear Programming and Extensions. Princeton NJ : Princeton University Press, 1963.

[2] Hillier, F.S, Lieberman, G.J, Operations Research (Second Edition), Holden-Day inc. pages 725-727, 1974.

[3] Kuenzi, H.P, Tzschach, H.G, and Zehnder, C.A. Numerical methods of Mathematical optimisation, (New York : Academic Press).

[4] Plastock, R.A, Kalley, G. Theory and problems of computer graphics, (Schaum's outline series), 1986.

[5] William H. Press, Saul A. Teukolsky, William T Vetterling, Brian P. Flannery. Numerical recipes in C (Second Edition), *The art of scientific computing*, Cambridge University Press, pages 430-443, 1992.

[6] Wren D. Identifying robot finger grasping points from range data. Master's thesis, Dept. of A.I., Edinburgh University, 1992.

Cite Nguyen ?
on theory of grasp

A Experimentation Results

```
Enter Object Data File name : mod

Reading Object Data from file : mod


Tolerance value : 0.000001

Upper bound on a force value : 1000.000000

Lower bound on a force value : 0.010000

Force of gravity on object  : 0.000000 0.000000 -9.810000

Gravitational angle of stability : 30.000000

Centre of mass of object    : 50.000000 50.000000 50.000000

Moment of inertia of object :

196.23    1.29   -4.28
  1.29  187.26    1.03
 -4.28    1.03  229.16

Enter Contact Data File name : cod

Reading Contact Data from file : cod


Total number of contact points : 3


Contact type of contact point 1 : frictionless point
```



```
Local co-ordinate of contact point 1 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 1 :
  0.00    0.00   -1.00    0.00
  0.00    1.00    0.00   50.00
  1.00    0.00    0.00   50.00


Global co-ordinate of contact point 1 :

0.000000 50.000000 50.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 1 :

0.000000 0.000000 -1.000000

Global normal of biquad 1 :

1.000000 0.000000 0.000000


Contact type of contact point 2 : soft fingered

Local co-ordinate of contact point 2 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 2 :
  0.00    0.00    1.00  100.00
  0.00    1.00    0.00   50.00
 -1.00    0.00    0.00   50.00
```

```
Global co-ordinate of contact point 2 :

100.000000 50.000000 50.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 2 :

0.000000 0.000000 -1.000000

Global normal of biquad 2 :

-1.000000 0.000000 0.000000

Coefficient of friction at contact point 2 : 0.320000

Finger radius at contact point 2 : 0.800000


Contact type of contact point 3 : soft fingered

Local co-ordinate of contact point 3 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 3 :
 -1.00    0.00    0.00   50.00
  0.00    1.00    0.00   30.00
  0.00    0.00   -1.00    0.00


Global co-ordinate of contact point 3 :

50.000000 30.000000 0.000000
```



```
Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 3 :

0.000000 0.000000 -1.000000

Global normal of biquad 3 :

0.000000 0.000000 1.000000

Coefficient of friction at contact point 3 : 0.020000

Finger radius at contact point 3 : 0.500000
```

STATIC (1)

```
Attempting static analysis for equilibrium ...


A state of equilibrium has been found :


Here are the force/torque values of each finger ...

 Finger 1 (frictionless point) :
  force(x,y,z)  = (0.011227, 0.000000, 0.000000)
  torque(x,y,z) = (0.000000, 0.000000, 0.000000)

 Finger 2 (soft fingered) :
  force(x,y,z)  = (-0.009492, 0.000662, 0.002483)
  torque(x,y,z) = (0.000000, 0.130166, -0.034711)
  tnorm(x,y,z)  = (-196.199905, 0.000000, 0.000000)

 Finger 3 (soft fingered) :
```

```
force(x,y,z)  = (-0.001736, -0.000661, 9.807516)
torque(x,y,z) = (196.199921, -0.130165, 0.034712)
tnorm(x,y,z)  = (0.000000, 0.000000, 0.000000)
```

STABILITY ①

```
Attempting stability analysis ...

oooooooooooooooooooooooooooooooooooooooo  ang =  0.00  G = (0.000000 0.000000 -9.810000)
xxxxoooooooooooooooooooooooooooooooooxxxx  ang =  1.50  G = (-0.040171 0.253634 -9.806639)
xxxxxxxxoooooooooooooooooooooooooooxxxxxxxx  ang =  3.00  G = (-0.080315 0.507095 -9.796556)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang =  4.50  G = (-0.120404 0.760208 -9.779759)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang =  6.00  G = (-0.160410 1.012800 -9.756260)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang =  7.50  G = (-0.200306 1.264698 -9.726074)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang =  9.00  G = (-0.240065 1.515729 -9.689223)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang = 10.50  G = (-0.279659 1.765721 -9.645731)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang = 12.00  G = (-0.319062 2.014503 -9.595629)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang = 13.50  G = (-0.358246 2.261905 -9.538949)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang = 15.00  G = (-0.397184 2.507756 -9.475733)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang = 16.50  G = (-0.435851 2.751889 -9.406022)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang = 18.00  G = (-0.474218 2.994136 -9.329865)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang = 19.50  G = (-0.512261 3.234330 -9.247313)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang = 21.00  G = (-0.549952 3.472308 -9.158424)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang = 22.50  G = (-0.587267 3.707906 -9.063258)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang = 24.00  G = (-0.624179 3.940964 -8.961882)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang = 25.50  G = (-0.660663 4.171319 -8.854362)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang = 27.00  G = (-0.696695 4.398817 -8.740774)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang = 28.50  G = (-0.732249 4.623299 -8.621196)
xxxxxxxxxxoooooooooooooooooooooooxxxxxxxxxx  ang = 30.00  G = (-0.767301 4.844613 -8.495709)


Grasp was in equilibrium for for 478 out of 820 perturbations of G, (=58.29 percent)
```

(2)

MOD

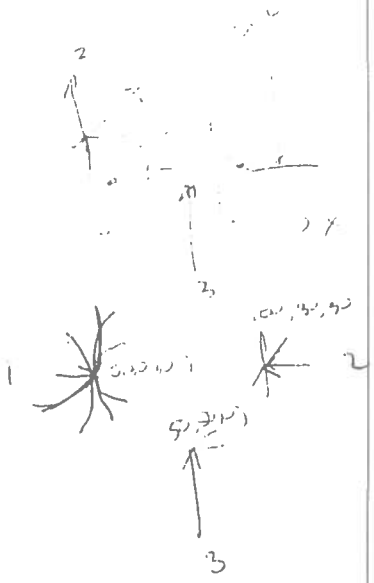
```
Enter Object Data File name : mod
```



```
Reading Object Data from file : mod


Tolerance value : 0.000001

Upper bound on a force value : 1000.000000

Lower bound on a force value : 0.010000

Force of gravity on object  : 0.000000 0.000000 -9.810000

Gravitational angle of stability : 30.000000

Centre of mass of object    : 50.000000 50.000000 50.000000

Moment of inertia of object :

196.23    1.29   -4.28
  1.29  187.26    1.03
 -4.28    1.03  229.16
```

FOD

```
Enter Contact Data File name : fod

Reading Contact Data from file : fod


Total number of contact points : 3


Contact type of contact point 1 : frictionless point

Local co-ordinate of contact point 1 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 1 :
```



```
 0.00    0.00   -1.00    0.00
 0.00    1.00    0.00   30.00
 1.00    0.00    0.00   30.00


Global co-ordinate of contact point 1 :

0.000000 30.000000 30.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 1.000000 1.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 1 :

0.000000 0.000000 -1.000000

Global normal of biquad 1 :

1.000000 0.000000 0.000000


Contact type of contact point 2 : frictionless point

Local co-ordinate of contact point 2 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 2 :
  0.00    0.00    1.00  100.00
  0.00    1.00    0.00   30.00
 -1.00    0.00    0.00   30.00


Global co-ordinate of contact point 2 :

100.000000 30.000000 30.000000
```



```
Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 2 :

0.000000 0.000000 -1.000000

Global normal of biquad 2 :

-1.000000 0.000000 0.000000


Contact type of contact point 3 : frictionless point

Local co-ordinate of contact point 3 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 3 :
 -1.00    0.00    0.00   50.00
  0.00    1.00    0.00   50.00
  0.00    0.00   -1.00    0.00


Global co-ordinate of contact point 3 :

50.000000 50.000000 0.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 3 :

0.000000 0.000000 -1.000000

Global normal of biquad 3 :
```

```
0.000000 0.000000 1.000000


Attempting static analysis for equilibrium ...


A state of equilibrium has been found :


Here are the force/torque values of each finger ...

 Finger 1 (frictionless point) :
  force(x,y,z)  = (0.010000, 0.000000, 0.000000)
  torque(x,y,z) = (0.000000, 0.200000, -0.200000)

 Finger 2 (frictionless point) :
  force(x,y,z)  = (-0.010000, 0.000000, 0.000000)
  torque(x,y,z) = (0.000000, -0.200000, 0.200000)

 Finger 3 (frictionless point) :
  force(x,y,z)  = (0.000000, 0.000000, 9.809999)
  torque(x,y,z) = (0.000000, 0.000000, 0.000000)


Attempting stability analysis ...

oooooooooooooooooooooooooooooooooooooooo  ang =  0.00  G = (0.000000 0.000000 -9.810000)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang =  1.50  G = (-0.040171 0.253634 -9.806639)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang =  3.00  G = (-0.080315 0.507095 -9.796556)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang =  4.50  G = (-0.120404 0.760208 -9.779759)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang =  6.00  G = (-0.160410 1.012800 -9.756260)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang =  7.50  G = (-0.200306 1.264698 -9.726074)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang =  9.00  G = (-0.240065 1.515729 -9.689223)
```



```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 10.50  G = (-0.279659 1.765721 -9.645731)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 12.00  G = (-0.319062 2.014503 -9.595629)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 13.50  G = (-0.358246 2.261905 -9.538949)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 15.00  G = (-0.397184 2.507756 -9.475733)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 16.50  G = (-0.435851 2.751889 -9.406022)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 18.00  G = (-0.474218 2.994136 -9.329865)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 19.50  G = (-0.512261 3.234330 -9.247313)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 21.00  G = (-0.549952 3.472308 -9.158424)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 22.50  G = (-0.587267 3.707906 -9.063258)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 24.00  G = (-0.624179 3.940964 -8.961882)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 25.50  G = (-0.660663 4.171319 -8.854362)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 27.00  G = (-0.696695 4.398817 -8.740774)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 28.50  G = (-0.732249 4.623299 -8.621196)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 30.00  G = (-0.767301 4.844613 -8.495709)


Grasp was in equilibrium for for 40 out of 820 perturbations of G, (=4.88 percent)


Enter Object Data File name : mod

Reading Object Data from file : mod


Tolerance value : 0.000001

Upper bound on a force value : 1000.000000

Lower bound on a force value : 0.010000

Force of gravity on object  : 0.000000 0.000000 -9.810000

Gravitational angle of stability : 30.000000

Centre of mass of object    : 50.000000 50.000000 50.000000
```

Moment of inertia of object :

196.23 1.29 -4.28
1.29 187.26 1.03
-4.28 1.03 229.16

Enter Contact Data File name : zod

Reading Contact Data from file : zod

Total number of contact points : 6

Contact type of contact point 1 : frictionless point

Local co-ordinate of contact point 1 :

0.000000 0.000000 0.000000

Transformation matrix for contact point 1 :

0.00 0.00 0.00 50.00
0.00 -1.00 -1.00 0.00
1.00 0.00 0.00 50.00

Global co-ordinate of contact point 1 :

50.000000 0.000000 50.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 1.000000 1.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 1 :

0.000000 0.000000 -1.000000

Global normal of biquad 1 :

0.000000 1.000000 0.000000

Contact type of contact point 2 : frictionless point

Local co-ordinate of contact point 2 :

0.000000 0.000000 0.000000

Transformation matrix for contact point 2 :

0.00 0.00 0.00 50.00
0.00 1.00 1.00 0.00
1.00 0.00 0.00 50.00

Global co-ordinate of contact point 2 :

50.000000 0.000000 50.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 1.000000 1.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 2 :

0.000000 0.000000 -1.000000

Global normal of biquad 2 :

0.000000 -1.000000 0.000000

Contact type of contact point 3 : frictionless point

Local co-ordinate of contact point 3 :

```
0.000000 0.000000 0.000000


Transformation matrix for contact point 3 :
  0.00    0.00   -1.00    0.00
  0.00    1.00    0.00   50.00
  1.00    0.00    0.00   50.00


Global co-ordinate of contact point 3 :

0.000000 50.000000 50.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 1.000000 1.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 3 :

0.000000 0.000000 -1.000000

Global normal of biquad 3 :

1.000000 0.000000 0.000000


Contact type of contact point 4 : frictionless point

Local co-ordinate of contact point 4 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 4 :
  0.00    0.00    1.00  100.00
  0.00    1.00    0.00   50.00
 -1.00    0.00    0.00   50.00
```



```
Global co-ordinate of contact point 4 :

100.000000 50.000000 50.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 1.000000 1.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 4 :

0.000000 0.000000 -1.000000

Global normal of biquad 4 :

-1.000000 0.000000 0.000000


Contact type of contact point 5 : frictionless point

Local co-ordinate of contact point 5 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 5 :
 -1.00    0.00    0.00   50.00
  0.00    1.00    0.00   50.00
  0.00    0.00   -1.00    0.00


Global co-ordinate of contact point 5 :

50.000000 50.000000 0.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 5 :
```

```
0.000000 0.000000 -1.000000

Global normal of biquad 5 :

0.000000 0.000000 1.000000


Attempting static analysis for equilibrium ...


A state of equilibrium has been found :


Here are the force/torque values of each finger ...

 Finger 1 (frictionless point) :
  force(x,y,z)  = (0.000000, 0.010000, 0.000000)
  torque(x,y,z) = (0.000000, 0.000000, 0.000000)

 Finger 2 (frictionless point) :
  force(x,y,z)  = (0.000000, -0.010000, 0.000000)
  torque(x,y,z) = (0.000000, 0.000000, -0.000000)

 Finger 3 (frictionless point) :
  force(x,y,z)  = (0.010000, 0.000000, 0.000000)
  torque(x,y,z) = (0.000000, 0.000000, 0.000000)

 Finger 4 (frictionless point) :
  force(x,y,z)  = (-0.010000, 0.000000, 0.000000)
  torque(x,y,z) = (0.000000, 0.000000, 0.000000)

 Finger 5 (frictionless point) :
  force(x,y,z)  = (0.000000, 0.000000, 9.809999)
  torque(x,y,z) = (0.000000, 0.000000, 0.000000)
```



```
Attempting stability analysis ...

oooooooooooooooooooooooooooooooooooooooo  ang =  0.00  G = (0.000000 0.000000 -9.810000)
oooooooooooooooooooooooooooooooooooooooo  ang =  1.50  G = (-0.040171 0.253634 -9.806639)
oooooooooooooooooooooooooooooooooooooooo  ang =  3.00  G = (-0.080315 0.507095 -9.796556)
oooooooooooooooooooooooooooooooooooooooo  ang =  4.50  G = (-0.120404 0.760208 -9.779759)
oooooooooooooooooooooooooooooooooooooooo  ang =  6.00  G = (-0.160410 1.012800 -9.756260)
oooooooooooooooooooooooooooooooooooooooo  ang =  7.50  G = (-0.200306 1.264698 -9.726074)
oooooooooooooooooooooooooooooooooooooooo  ang =  9.00  G = (-0.240065 1.515729 -9.689223)
oooooooooooooooooooooooooooooooooooooooo  ang = 10.50  G = (-0.279659 1.765721 -9.645731)
oooooooooooooooooooooooooooooooooooooooo  ang = 12.00  G = (-0.319062 2.014503 -9.595629)
oooooooooooooooooooooooooooooooooooooooo  ang = 13.50  G = (-0.358246 2.261905 -9.538949)
oooooooooooooooooooooooooooooooooooooooo  ang = 15.00  G = (-0.397184 2.507756 -9.475733)
oooooooooooooooooooooooooooooooooooooooo  ang = 16.50  G = (-0.435851 2.751889 -9.406022)
oooooooooooooooooooooooooooooooooooooooo  ang = 18.00  G = (-0.474218 2.994136 -9.329865)
oooooooooooooooooooooooooooooooooooooooo  ang = 19.50  G = (-0.512261 3.234330 -9.247313)
oooooooooooooooooooooooooooooooooooooooo  ang = 21.00  G = (-0.549952 3.472308 -9.158424)
oooooooooooooooooooooooooooooooooooooooo  ang = 22.50  G = (-0.587267 3.707906 -9.063258)
oooooooooooooooooooooooooooooooooooooooo  ang = 24.00  G = (-0.624179 3.940964 -8.961882)
oooooooooooooooooooooooooooooooooooooooo  ang = 25.50  G = (-0.660663 4.171319 -8.854362)
oooooooooooooooooooooooooooooooooooooooo  ang = 27.00  G = (-0.696695 4.398817 -8.740774)
oooooooooooooooooooooooooooooooooooooooo  ang = 28.50  G = (-0.732249 4.623299 -8.621196)
oooooooooooooooooooooooooooooooooooooooo  ang = 30.00  G = (-0.767301 4.844613 -8.495709)


Grasp was in equilibrium for for 820 out of 820 perturbations of G, (=100.00 percent)


Enter Object Data File name : mod

Reading Object Data from file : mod
```

```
Tolerance value : 0.000001

Upper bound on a force value : 1000.000000

Lower bound on a force value : 0.010000

Force of gravity on object  : 0.000000 0.000000 -9.810000

Gravitational angle of stability : 30.000000

Centre of mass of object    : 50.000000 50.000000 50.000000

Moment of inertia of object :

196.23    1.29   -4.28
  1.29  187.28    1.03
 -4.28    1.03  229.16

Enter Contact Data File name : zod2

Reading Contact Data from file : zod2


Total number of contact points : 3


Contact type of contact point 1 : soft fingered

Local co-ordinate of contact point 1 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 1 :
  0.00    0.00    0.00   30.00
  0.00   -1.00   -1.00    0.00
  1.00    0.00    0.00   30.00
```



```
Global co-ordinate of contact point 1 :

30.000000 0.000000 30.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 1.000000 1.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 1 :

0.000000 0.000000 -1.000000

Global normal of biquad 1 :

0.000000 1.000000 0.000000

Coefficient of friction at contact point 1 : 0.120000

Finger radius at contact point 1 : 1.200000


Contact type of contact point 2 : soft fingered

Local co-ordinate of contact point 2 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 2 :
  0.00    0.00    0.00   30.00
  0.00    1.00    1.00    0.00
  1.00    0.00    0.00   30.00


Global co-ordinate of contact point 2 :

30.000000 0.000000 30.000000
```

```
Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 1.000000 1.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 2 :

0.000000 0.000000 -1.000000

Global normal of biquad 2 :

0.000000 -1.000000 0.000000

Coefficient of friction at contact point 2 : 0.120000

Finger radius at contact point 2 : 2.200000


Contact type of contact point 3 : soft fingered

Local co-ordinate of contact point 3 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 3 :
 -1.00    0.00    0.00   50.00
  0.00    1.00    0.00   50.00
  0.00    0.00   -1.00    0.00


Global co-ordinate of contact point 3 :

50.000000 50.000000 0.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 3 :
```



```
0.000000 0.000000 -1.000000

Global normal of biquad 3 :

0.000000 0.000000 1.000000

Coefficient of friction at contact point 3 : 0.120000

Finger radius at contact point 3 : 1.200000


Attempting static analysis for equilibrium ...


A state of equilibrium has been found :


Here are the force/torque values of each finger ...

 Finger 1 (soft fingered) :
  force(x,y,z)  = (0.000000, 0.007681, -0.000011)
  torque(x,y,z) = (-0.154187, 0.000216, 0.153628)
  tnorm(x,y,z)  = (0.000000, 0.000000, 0.000000)

 Finger 2 (soft fingered) :
  force(x,y,z)  = (0.000000, -0.007681, 0.000014)
  torque(x,y,z) = (0.154251, -0.000277, -0.153629)
  tnorm(x,y,z)  = (0.000000, 0.000000, 0.000000)

 Finger 3 (soft fingered) :
  force(x,y,z)  = (0.000001, 0.000001, 9.809998)
  torque(x,y,z) = (-0.000061, 0.000060, 0.000000)
  tnorm(x,y,z)  = (0.000000, 0.000000, 0.000000)
```

```
Attempting stability analysis ...

oooooooooooooooooooooooooooooooooooooooo  ang =  0.00  G = (0.000000 0.000000 -9.810000)
oooooooooooooooooooooooooooooooooooooooo  ang =  1.50  G = (-0.040171 0.253634 -9.806639)
oooooooooooooooooooooooooooooooooooooooo  ang =  3.00  G = (-0.080315 0.507095 -9.796556)
oooooooooooooooooooooooooooooooooooooooo  ang =  4.50  G = (-0.120404 0.760208 -9.779759)
oooooooooooooooooooooooooooooooooooooooo  ang =  6.00  G = (-0.160410 1.012800 -9.756260)
oooooooooooooooooooooooooooooooooooooooo  ang =  7.50  G = (-0.200306 1.264698 -9.726074)
oooooooooooooooooooooooooooooooooooooooo  ang =  9.00  G = (-0.240065 1.515729 -9.689223)
oooooooooooooooooooooooooooooooooooooooo  ang = 10.50  G = (-0.279659 1.765721 -9.645731)
oooooooooooooooooooooooooooooooooooooooo  ang = 12.00  G = (-0.319062 2.014503 -9.595629)
oooooooooooooooooooooooooooooooooooooooo  ang = 13.50  G = (-0.358246 2.261905 -9.538949)
oooooooooooooooooooooooooooooooooooooooo  ang = 15.00  G = (-0.397184 2.507756 -9.475733)
oooooooooooooooooooooooooooooooooooooooo  ang = 16.50  G = (-0.435851 2.751889 -9.406022)
oooooooooooooooooooooooooooooooooooooooo  ang = 18.00  G = (-0.474218 2.994136 -9.329865)
oooooooooooooooooooooooooooooooooooooooo  ang = 19.50  G = (-0.512261 3.234330 -9.247313)
oooooooooooooooooooooooooooooooooooooooo  ang = 21.00  G = (-0.549952 3.472308 -9.158424)
oooooooooooooooooooooooooooooooooooooooo  ang = 22.50  G = (-0.587287 3.707906 -9.063258)
oooooooooooooooooooooooooooooooooooooooo  ang = 24.00  G = (-0.624179 3.940964 -8.961882)
oooooooooooooooooooooooooxoooooooooooooo  ang = 25.50  G = (-0.660683 4.171319 -8.854362)
oooooooooooooooooooooooooxxxxxoooooooooo  ang = 27.00  G = (-0.696695 4.398817 -8.740774)
oooooooooooooooooooooooooxxxxxoooooooooo  ang = 28.50  G = (-0.732249 4.623299 -8.621196)
oooooooooooooooooooooooooxxxxxxxoooooooo  ang = 30.00  G = (-0.767301 4.844613 -8.495709)


Grasp was in equilibrium for for 802 out of 820 perturbations of G, (=97.80 percent)


Enter Object Data File name : m2n

Reading Object Data from file : m2n


Tolerance value : 0.000001
```



```
Upper bound on a force value : 100.000000

Lower bound on a force value : 0.010000

Force of gravity on object  : -1.000000 0.000000 -9.810000

Gravitational angle of stability : 30.000000

Centre of mass of object    : 50.000000 50.000000 50.000000

Moment of inertia of object :

196.23    1.29   -4.28
  1.29  187.26    1.03
 -4.28    1.03  229.16

Enter Contact Data File name : zod2

Reading Contact Data from file : zod2


Total number of contact points : 3


Contact type of contact point 1 : soft fingered

Local co-ordinate of contact point 1 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 1 :
  0.00    0.00    0.00   30.00
  0.00   -1.00   -1.00    0.00
  1.00    0.00    0.00   30.00
```

Global co-ordinate of contact point 1 :

30.000000 0.000000 30.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 1.000000 1.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 1 :

0.000000 0.000000 -1.000000

Global normal of biquad 1 :

0.000000 1.000000 0.000000

Coefficient of friction at contact point 1 : 0.120000

Finger radius at contact point 1 : 1.200000

Contact type of contact point 2 : soft fingered

Local co-ordinate of contact point 2 :

0.000000 0.000000 0.000000

Transformation matrix for contact point 2 :

0.00	0.00	0.00	30.00
0.00	1.00	1.00	0.00
1.00	0.00	0.00	30.00

Global co-ordinate of contact point 2 :

30.000000 0.000000 30.000000

Biquad = axx + byy + cxy + dx + ey + f

Coeffs = 1.000000 1.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 2 :

0.000000 0.000000 -1.000000

Global normal of biquad 2 :

0.000000 -1.000000 0.000000

Coefficient of friction at contact point 2 : 0.120000

Finger radius at contact point 2 : 2.200000

Contact type of contact point 3 : soft fingered

Local co-ordinate of contact point 3 :

0.000000 0.000000 0.000000

Transformation matrix for contact point 3 :

-1.00	0.00	0.00	50.00
0.00	1.00	0.00	50.00
0.00	0.00	-1.00	0.00

Global co-ordinate of contact point 3 :

50.000000 50.000000 0.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 3 :

0.000000 0.000000 -1.000000

```
Global normal of biquad 3 :

0.000000 0.000000 1.000000

Coefficient of friction at contact point 3 : 0.120000

Finger radius at contact point 3 : 1.200000


Attempting static analysis for equilibrium ...


A state of equilibrium has been found :


Here are the force/torque values of each finger ...

 Finger 1 (soft fingered) :
  force(x,y,z)  = (0.000000, 0.002713, -0.009625)
  torque(x,y,z) = (-0.552626, 0.192496, 0.054269)
  tnorm(x,y,z)  = (0.000000, 0.000000, 0.000000)

 Finger 2 (soft fingered) :
  force(x,y,z)  = (0.000000, -0.002714, 0.011093)
  torque(x,y,z) = (0.552690, -0.221857, -0.054270)
  tnorm(x,y,z)  = (0.000000, -59.922535, 0.000000)

 Finger 3 (soft fingered) :
  force(x,y,z)  = (0.999999, 0.000001, 9.808532)
  torque(x,y,z) = (-0.000083, 59.951900, 0.000000)
  tnorm(x,y,z)  = (0.000000, 0.000000, 0.000000)
```



```
Attempting stability analysis ...

oooooooooooooooooooooooooooooooooooooooo  ang =  0.00  G = (-1.000000 -0.000000 -9.810000)
oooooooooooooooooooooooooxxxoooooooooooo  ang =  1.50  G = (-1.039829 0.254949 -9.802544)
ooooooooooooooooooxxxxxxxxxxxxxxxxxxxooo  ang =  3.00  G = (-1.078945 0.509723 -9.788369)
oooooooooooooooooxxxxxxxxxxxxxxxxxxxxxoo  ang =  4.50  G = (-1.117321 0.764147 -9.767486)
oooooooooooooooooxxxxxxxxxxxxxxxxxxxxxxo  ang =  6.00  G = (-1.154932 1.018048 -9.739908)
oooooooooooooooooxxxxxxxxxxxxxxxxxxxxxxo  ang =  7.50  G = (-1.191751 1.271252 -9.705656)
oooooooooooooooooxxxxxxxxxxxxxxxxxxxxxxo  ang =  9.00  G = (-1.227753 1.523584 -9.664751)
oooooooooooooooooxxxxxxxxxxxxxxxxxxxxxxo  ang = 10.50  G = (-1.262914 1.774871 -9.617224)
oooooooooooooooooxxxxxxxxxxxxxxxxxxxxxxx  ang = 12.00  G = (-1.297210 2.024943 -9.563105)
ooooooooooooooxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 13.50  G = (-1.330616 2.273628 -9.502431)
ooooooooooooooxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 15.00  G = (-1.363110 2.520752 -9.435246)
ooooooooooooooxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 16.50  G = (-1.394670 2.766150 -9.361593)
ooooooooooooxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 18.00  G = (-1.425275 3.009652 -9.281525)
ooooooooooooxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 19.50  G = (-1.454902 3.251091 -9.195095)
ooooooooooooxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 21.00  G = (-1.483533 3.490302 -9.102364)
oooooooooooxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 22.50  G = (-1.511147 3.727121 -9.003395)
ooooooooxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 24.00  G = (-1.537725 3.961386 -8.898254)
ooooooooxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 25.50  G = (-1.563249 4.192935 -8.787017)
ooooooxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 27.00  G = (-1.587702 4.421612 -8.669755)
ooooooxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 28.50  G = (-1.611066 4.647258 -8.546554)
ooooooxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 30.00  G = (-1.633327 4.869719 -8.417493)


Grasp was in equilibrium for for 307 out of 820 perturbations of G, (=37.44 percent)


Enter Object Data File name : m2p

Reading Object Data from file : m2p


Tolerance value : 0.000001
```

```
Upper bound on a force value : 100.000000

Lower bound on a force value : 0.010000

Force of gravity on object  : 2.000000 0.000000 -9.810000

Gravitational angle of stability : 10.000000

Centre of mass of object    : 50.000000 50.000000 50.000000

Moment of inertia of object :

196.23    1.29   -4.28
  1.29  187.26    1.03
 -4.28    1.03  229.16

Enter Contact Data File name : zod2

Reading Contact Data from file : zod2


Total number of contact points : 3


Contact type of contact point 1 : soft fingered

Local co-ordinate of contact point 1 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 1 :
  0.00    0.00    0.00   30.00
  0.00   -1.00   -1.00    0.00
  1.00    0.00    0.00   30.00


Global co-ordinate of contact point 1 :
```



```
30.000000 0.000000 30.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 1.000000 1.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 1 :

0.000000 0.000000 -1.000000

Global normal of biquad 1 :

0.000000 1.000000 0.000000

Coefficient of friction at contact point 1 : 0.120000

Finger radius at contact point 1 : 1.200000


Contact type of contact point 2 : soft fingered

Local co-ordinate of contact point 2 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 2 :
  0.00    0.00    0.00   30.00
  0.00    1.00    1.00    0.00
  1.00    0.00    0.00   30.00


Global co-ordinate of contact point 2 :

30.000000 0.000000 30.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 1.000000 1.000000 0.000000 0.000000 0.000000 0.000000
```

```
Local normal of biquad 2 :

0.000000 0.000000 -1.000000

Global normal of biquad 2 :

0.000000 -1.000000 0.000000

Coefficient of friction at contact point 2 : 0.120000

Finger radius at contact point 2 : 2.200000


Contact type of contact point 3 : soft fingered

Local co-ordinate of contact point 3 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 3 :
 -1.00    0.00    0.00   50.00
  0.00    1.00    0.00   50.00
  0.00    0.00   -1.00    0.00


Global co-ordinate of contact point 3 :

50.000000 50.000000 0.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 3 :

0.000000 0.000000 -1.000000
```



```
Global normal of biquad 3 :

0.000000 0.000000 1.000000

Coefficient of friction at contact point 3 : 0.120000

Finger radius at contact point 3 : 1.200000
```

STATIC 6

```
Attempting static analysis for equilibrium ...


A state of equilibrium has been found :


Here are the force/torque values of each finger ...

 Finger 1 (soft fingered) :
  force(x,y,z)  = (0.000000, 12.537384, 44.470936)
  torque(x,y,z) = (2051.881836, -889.418701, 250.747665)
  tnorm(x,y,z)  = (0.000000, 0.000000, 0.000000)

 Finger 2 (soft fingered) :
  force(x,y,z)  = (0.000000, -12.537388, -51.247513)
  torque(x,y,z) = (-2051.881836, 1024.950195, -250.747681)
  tnorm(x,y,z)  = (0.000000, -15.627781, 0.000000)

 Finger 3 (soft fingered) :
  force(x,y,z)  = (-1.999999, 0.000000, 16.586582)
  torque(x,y,z) = (0.000000, -119.903854, 0.000000)
  tnorm(x,y,z)  = (0.000000, 0.000000, 0.000000)
```

STABLE (6)

```
Attempting stability analysis ...

oooooooooooooooooooooooooooooooooooooooo  ang =  0.00  G = (2.000000 0.000000 -9.809999)
oooooooooooooooooooooooooooooooooooooooo  ang =  0.50  G = (1.986532 0.086293 -9.812356)
oooooooooooooooooooooooooooooooooooooooo  ang =  1.00  G = (1.972913 0.172579 -9.813965)
oooooooooooooooooooooooooooooooooooooooo  ang =  1.50  G = (1.959143 0.258852 -9.814828)
oooooooooooooooooooooooooooooooooooooooo  ang =  2.00  G = (1.945225 0.345105 -9.814942)
oooooooooooooooxxxoooooooooooooooooooooo  ang =  2.50  G = (1.931158 0.431332 -9.814309)
oooooooooooooxxxxxxxoooooooooooooooooooo  ang =  3.00  G = (1.916944 0.517528 -9.812929)
oooooooooooooxxxxxxxxxoooooooooooooooooo  ang =  3.50  G = (1.902584 0.603681 -9.810802)
oooooooooooxxxxxxxxxxxxoooooooooooooooooo  ang =  4.00  G = (1.888079 0.689790 -9.807926)
oooooooooooxxxxxxxxxxxxxoooooooooooooooooo  ang =  4.50  G = (1.873431 0.775846 -9.804305)
ooooooooxxxxxxxxxxxxxxxxoooooooooooooooooo  ang =  5.00  G = (1.858640 0.861843 -9.799937)
ooooooxxxxxxxxxxxxxxxxxxoooooooooooooooooo  ang =  5.50  G = (1.843707 0.947775 -9.794823)
ooooooxxxxxxxxxxxxxxxxxxoooooooooooooooooo  ang =  6.00  G = (1.828634 1.033634 -9.788962)
oooooxxxxxxxxxxxxxxxxxxxxoooooooooooooooooo  ang =  6.50  G = (1.813421 1.119414 -9.782356)
oooooxxxxxxxxxxxxxxxxxxxxoooooooooooooooooo  ang =  7.00  G = (1.798071 1.205110 -9.775006)
oooooxxxxxxxxxxxxxxxxxxxxoooooooooooooooooo  ang =  7.50  G = (1.782584 1.290714 -9.766911)
ooooxxxxxxxxxxxxxxxxxxxxxxoooooooooooooooooo  ang =  8.00  G = (1.766961 1.376219 -9.758072)
ooooxxxxxxxxxxxxxxxxxxxxxxoooooooooooooooooo  ang =  8.50  G = (1.751203 1.461619 -9.748489)
ooooxxxxxxxxxxxxxxxxxxxxxxoooooooooooooooooo  ang =  9.00  G = (1.735312 1.546908 -9.738165)
ooooxxxxxxxxxxxxxxxxxxxxxxoooooooooooooooooo  ang =  9.50  G = (1.719288 1.632080 -9.727098)
ooxxxxxxxxxxxxxxxxxxxxxxxxoooooooooooooooooo  ang = 10.00  G = (1.703134 1.717127 -9.715291)


Grasp was in equilibrium for for 558 out of 820 perturbations of G, (=68.05 percent)
```

(7)

M2N (7)

```
Enter Object Data File name : m2n

Reading Object Data from file : m2n


Tolerance value : 0.000001

Upper bound on a force value : 100.000000
```



```
Lower bound on a force value : 0.010000

Force of gravity on object  : -1.000000 0.000000 -9.810000

Gravitational angle of stability : 30.000000

Centre of mass of object    : 50.000000 50.000000 50.000000

Moment of inertia of object :

196.23    1.29   -4.28
  1.29  187.26    1.03
 -4.28    1.03  229.16
```

BOD (7)

```
Enter Contact Data File name : bod

Reading Contact Data from file : bod


Total number of contact points : 3


Contact type of contact point 1 : frictionless point

Local co-ordinate of contact point 1 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 1 :
  0.00    0.00   -1.00    0.00
  0.00    1.00    0.00   30.00
  1.00    0.00    0.00   30.00


Global co-ordinate of contact point 1 :
```

```
0.000000 30.000000 30.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 1 :

0.000000 0.000000 -1.000000

Global normal of biquad 1 :

1.000000 0.000000 0.000000


Contact type of contact point 2 : frictional point

Local co-ordinate of contact point 2 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 2 :
  0.00     0.00     1.00   100.00
  0.00     1.00     0.00    30.00
 -1.00     0.00     0.00    30.00


Global co-ordinate of contact point 2 :

100.000000 30.000000 30.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 2 :

0.000000 0.000000 -1.000000
```



```
Global normal of biquad 2 :

-1.000000 0.000000 0.000000

Coefficient of friction at contact point 2 : 0.320000


Contact type of contact point 3 : frictionless point

Local co-ordinate of contact point 3 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 3 :
 -1.00     0.00     0.00    50.00
  0.00     1.00     0.00    50.00
  0.00     0.00    -1.00     0.00


Global co-ordinate of contact point 3 :

50.000000 50.000000 0.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 3 :

0.000000 0.000000 -1.000000

Global normal of biquad 3 :

0.000000 0.000000 1.000000
```

STATIC (7)

```
Attempting static analysis for equilibrium ...
```

```
no equilibrium solution

Enter Object Data File name : mod

Reading Object Data from file : mod

Tolerance value : 0.000001

Upper bound on a force value : 1000.000000

Lower bound on a force value : 0.010000

Force of gravity on object  : 0.000000 0.000000 -9.810000

Gravitational angle of stability : 30.000000

Centre of mass of object    : 50.000000 50.000000 50.000000

Moment of inertia of object :

196.23    1.29   -4.28
  1.29  187.26    1.03
 -4.28    1.03  229.16

Enter Contact Data File name : bod

Reading Contact Data from file : bod

Total number of contact points : 3

Contact type of contact point 1 : frictionless point

Local co-ordinate of contact point 1 :
```



```
0.000000 0.000000 0.000000

Transformation matrix for contact point 1 :
  0.00    0.00   -1.00    0.00
  0.00    1.00    0.00   30.00
  1.00    0.00    0.00   30.00

Global co-ordinate of contact point 1 :

0.000000 30.000000 30.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 1 :

0.000000 0.000000 -1.000000

Global normal of biquad 1 :

1.000000 0.000000 0.000000

Contact type of contact point 2 : frictional point

Local co-ordinate of contact point 2 :

0.000000 0.000000 0.000000

Transformation matrix for contact point 2 :
  0.00    0.00    1.00  100.00
  0.00    1.00    0.00   30.00
 -1.00    0.00    0.00   30.00
```

```
Global co-ordinate of contact point 2 :

100.000000 30.000000 30.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Local normal of biquad 2 :

0.000000 0.000000 -1.000000

Global normal of biquad 2 :

-1.000000 0.000000 0.000000

Coefficient of friction at contact point 2 : 0.320000


Contact type of contact point 3 : frictionless point

Local co-ordinate of contact point 3 :

0.000000 0.000000 0.000000


Transformation matrix for contact point 3 :
 -1.00    0.00    0.00   50.00
  0.00    1.00    0.00   50.00
  0.00    0.00   -1.00    0.00


Global co-ordinate of contact point 3 :

50.000000 50.000000 0.000000

Biquad = axx + byy + cxy + dx + ey + f
Coeffs = 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
```



```
Local normal of biquad 3 :

0.000000 0.000000 -1.000000

Global normal of biquad 3 :

0.000000 0.000000 1.000000


Attempting static analysis for equilibrium ...


A state of equilibrium has been found :


Here are the force/torque values of each finger ...

 Finger 1 (frictionless point) :
  force(x,y,z)  = (0.010000, 0.000000, 0.000000)
  torque(x,y,z) = (0.000000, 0.200000, -0.200000)

 Finger 2 (frictional point) :
  force(x,y,z)  = (-0.009999, -0.000000, -0.000000)
  torque(x,y,z) = (0.000001, -0.199999, 0.200001)

 Finger 3 (frictionless point) :
  force(x,y,z)  = (0.000000, 0.000000, 9.809999)
  torque(x,y,z) = (0.000000, 0.000000, 0.000000)


Attempting stability analysis ...
```

```
oooooooooooooooooooooooooooooooooooooooo  ang =  0.00  G = (0.000000 0.000000 -9.810000)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang =  1.50  G = (-0.040171 0.253634 -9.806639)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang =  3.00  G = (-0.080315 0.507095 -9.796556)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang =  4.50  G = (-0.120404 0.760208 -9.779759)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang =  6.00  G = (-0.160410 1.012800 -9.756260)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang =  7.50  G = (-0.200306 1.264698 -9.726074)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang =  9.00  G = (-0.240065 1.515729 -9.689223)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 10.50  G = (-0.279659 1.765721 -9.645731)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 12.00  G = (-0.319062 2.014503 -9.595629)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 13.50  G = (-0.358246 2.261905 -9.538949)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 15.00  G = (-0.397184 2.507756 -9.475733)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 16.50  G = (-0.435851 2.751889 -9.406022)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 18.00  G = (-0.474218 2.994136 -9.329865)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 19.50  G = (-0.512261 3.234330 -9.247313)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 21.00  G = (-0.549952 3.472308 -9.158424)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 22.50  G = (-0.587267 3.707906 -9.063258)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 24.00  G = (-0.624179 3.940964 -8.961882)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 25.50  G = (-0.660663 4.171319 -8.854362)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 27.00  G = (-0.696695 4.398817 -8.740774)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 28.50  G = (-0.732249 4.623299 -8.621198)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ang = 30.00  G = (-0.767301 4.844613 -8.495709)
```

```
Grasp was in equilibrium for for 40 out of 820 perturbations of G, (=4.88 percent)
```

A Program code

```
1 /*
2   File : Toplevel.c
3 */
4
5 #include <stdio.h>
6 #include <stdlib.h>
```



```
7 #include <math.h>
8 #include "allstruct.h"
9
10 main()
11 { int N,icase,ip[100],pc,fp,sf;
12   vector G,Com;
13   contact point[100];
14   float inertia[4][4],tol,ub,lb,disp;
15   double **a;
16
17   read_object(&G,&Com,inertia,&tol,&ub,&lb,&disp);
18
19   read_contacts(&N,point);
20
21   process_contacts(N,&pc,&fp,&sf,point);
22
23   a = nr_matrix(1,(2*pc)+(5*fp)+(6*sf)+17,1,(3*pc)+(9*fp)+(11*sf)+16);
24
25   printf("\n\n\n Attempting static analysis for equilibrium ... \n\n");
26
27   static_analysis(a,N,G,Com,pc,fp,sf,point,tol,ub,lb,ip,&icase);
28
29   if (icase==0)
30     {
31       printf("\n\n A state of equilibrium has been found : \n");
32       Display_forces(a,point,ip,pc,fp,sf,Com);
33       printf("\n\n\n Attempting stability analysis ... \n\n");
34       stability_analysis(a,N,G,Com,pc,fp,sf,point,tol,ub,lb,ip,disp);
35     }
36   else if (icase==1)
37     {
38       printf("\n\n Equilibrium with unbounded force values has been found \n");
39     }
40   else if (icase==-1)
41     {
42       printf("bummer man, no equilibrium");
43     };
44
```

```
45    exit(0);
46
47 }
48
49
50
51 /*
52    File : Readobj.c
53 */
54
55 #include <stdio.h>
56 #include <stdlib.h>
57
58 void read_object(G,Com,inertia,tol,ub,lb,ag)
59    vector *G,*Com;
60    float inertia[][4],*tol,*ub,*lb,*ag;
61 {
62    int c1,c2;
63    char Objectfile[100];
64    float tolerance,upb,lob,ang;
65    FILE *ObjFile;
66
67    printf("\n Enter Object Data File name : ");
68
69    scanf("%s",&Objectfile);
70
71    if ((ObjFile = fopen(Objectfile,"r")) == 0) {
72      fprintf(stderr,"Can't open %s for input.\n",Objectfile);
73      exit(0);
74      }
75
76    printf("\n Reading Object Data from file : %s \n",Objectfile);
77
78    fscanf(ObjFile,"%f",&tolerance);
79    *tol = tolerance;
80
81    fscanf(ObjFile,"%f",&upb);
82    *ub = upb;
```



```
83
84    fscanf(ObjFile,"%f",&lob);
85    *lb = lob;
86
87    fscanf(ObjFile,"%f%f%f",&G->x,&G->y,&G->z);
88
89    fscanf(ObjFile,"%f",&ang);
90    *ag = ang;
91
92    fscanf(ObjFile,"%f%f%f",&Com->x,&Com->y,&Com->z);
93
94    for (c2=1;c2<=3;c2++)
95      for (c1=1;c1<=3;c1++)
96          fscanf(ObjFile,"%f",&inertia[c1][c2]);
97
98    printf("\n\n Tolerance value : %f \n",tolerance);
99    printf("\n Upper bound on a force value : %f \n",upb);
100   printf("\n Lower bound on a force value : %f \n",lob);
101   printf("\n Force of gravity on object  : %f %f %f \n",G->x,G->y,G->z);
102   printf("\n Gravitational angle of stability : %f \n",ang);
103   printf("\n Centre of mass of object    : %f %f %f \n",Com->x,Com->y,Com->z);
104   printf("\n Moment of inertia of object : \n\n");
105
106   for (c2=1;c2<=3;c2++)
107     {
108       for (c1=1;c1<=3;c1++) printf("%7.2f ",inertia[c1][c2]);
109       printf("\n");
110     };
111
112
113   fclose(ObjFile);
114
115 }
116
117
118
119
120
```

```
121
122
123 /*
124   File : Readcon.c
125 */
126
127 #include <stdio.h>
128 #include <stdlib.h>
129
130 void read_contacts(N,point)
131   int *N;
132   contact point[];
133 {
134   int Cons,Count,c1,c2;
135   char Contactfile[100];
136   FILE *ConFile;
137
138   printf("\n Enter Contact Data File name : ");
139
140   scanf("%s",&Contactfile);
141
142   if ((ConFile = fopen(Contactfile,"r")) == 0) {
143     fprintf(stderr,"Can't open %s for input.\n",Contactfile);
144     exit(0);
145     }
146
147   printf("\n Reading Contact Data from file : %s \n",Contactfile);
148
149   fscanf(ConFile,"%d",&Cons);
150
151   *N = Cons;
152
153   for (Count = 0; Count < Cons; Count++)
154     {
155       fscanf(ConFile,"%d",&point[Count].ftype);
156
157       fscanf(ConFile,"%f%f%f",&point[Count].localcp.x,
158                       &point[Count].localcp.y,
```



```
159                       &point[Count].localcp.z);
160
161       for (c2=0;c2<3;c2++)
162 for (c1=0;c1<4;c1++)
163   fscanf(ConFile,"%f",&point[Count].convmatrix[c1][c2]);
164
165       fscanf(ConFile,"%f%f%f%f%f%f",&point[Count].contactregion.xx,
166                               &point[Count].contactregion.yy,
167                               &point[Count].contactregion.xy,
168                               &point[Count].contactregion.x,
169                               &point[Count].contactregion.y,
170                               &point[Count].contactregion.c);
171
172       if (point[Count].ftype>1) {
173 fscanf(ConFile,"%f",&point[Count].friction);
174       }
175
176       if (point[Count].ftype==3) {
177 fscanf(ConFile,"%f",&point[Count].frad);
178       }
179
180     }
181
182   fclose(ConFile);
183
184 }
185
186
187
188
189
190
191
192
193
194
195 /*
196   File : Pcontacts.c
```

```
197 */
198
199 #include <stdio.h>
200 #include <stdlib.h>
201
202
203 void process_contacts(N,pc,fp,sf,point)
204   int N,*pc,*fp,*sf;
205   contact point[];
206 {
207   int C,c1,c2;
208
209   *pc = 0;
210   *fp = 0;
211   *sf = 0;
212
213   printf("\n\n Total number of contact points : %d \n",N);
214
215   for (C = 0; C < N; C++)
216     {
217       printf("\n\n Contact type of contact point %d : ",C+1);
218
219       if (point[C].ftype == 1) {
220 printf("frictionless point"); *pc += 1;}
221       else
222 if (point[C].ftype == 2) {
223   printf("frictional point"); *fp += 1;}
224 else
225   if (point[C].ftype == 3) {
226     printf("soft fingered"); *sf += 1;};
227
228       printf("\n\n Local co-ordinate of contact point %d : \n",C+1);
229       printf("\n %f %f %f \n",point[C].localcp.x,
230                               point[C].localcp.y,
231                               point[C].localcp.z);
232
233       printf("\n\n Transformation matrix for contact point %d : \n",C+1);
234
```



```
235       for (c2=0;c2<3;c2++)
236         {
237           for (c1=0;c1<4;c1++)
238             printf("%7.2f ",point[C].convmatrix[c1][c2]);
239           printf("\n");
240         };
241
242       printf("\n\n Global co-ordinate of contact point %d : \n",C+1);
243       point[C].globalcp =
244         Transformation(point[C], point[C].localcp);
245       printf("\n %f %f %f \n",point[C].globalcp.x,
246                               point[C].globalcp.y,
247                               point[C].globalcp.z);
248
249       printf("\n Biquad = axx + byy + cxy + dx + ey + f ");
250       printf("\n Coeffs = %f %f %f %f %f %f \n",point[C].contactregion.xx,
251                                                 point[C].contactregion.yy,
252                                                 point[C].contactregion.xy,
253                                                 point[C].contactregion.x,
254                                                 point[C].contactregion.y,
255                                                 point[C].contactregion.c);
256
257       printf("\n Local normal of biquad %d : \n",C+1);
258       point[C].normal = Normaltobiquad(point[C].contactregion,
259   point[C].localcp);
260       printf("\n %f %f %f \n",point[C].normal.x,
261                               point[C].normal.y,
262                               point[C].normal.z);
263
264       printf("\n Global normal of biquad %d : \n",C+1);
265       point[C].globaln = Rotation(point[C], point[C].normal);
266       printf("\n %f %f %f \n",point[C].globaln.x,
267                               point[C].globaln.y,
268                               point[C].globaln.z);
269
270       if (point[C].ftype>1) {
271         printf("\n Coefficient of friction at contact point %d : %f \n",
272                 C+1,point[C].friction);
```

```
273       }
274
275       if (point[C].ftype==2)
276 {
277   fpCone(point[C].globaln,point[C].friction,point[C].cone);
278 }
279
280       if (point[C].ftype==3)
281 {
282   printf("\n Finger radius at contact point %d : %f \n",C+1,
283  point[C].frad);
284   sfCone(point[C],point[C].cone,point[C].points);
285 }
286     }
287
288 }
289 /*
290   File : Static.c
291 */
292
293 #include <stdio.h>
294 #include <stdlib.h>
295
296
297 void add_pcs(a,point,pc,fp,sf,N,Com,UBound,LBound)
298   int N,pc,fp,sf;
299   float UBound,LBound;
300   double **a;
301   vector Com;
302   contact point[];
303 {
304   int pcs,C;
305   vector T;
306
307   pcs = 0;
308
309   for (C=0; C < N; C++)
310     if (point[C].ftype==1)
```



```
311       {
312 pcs++;
313
314 a[1][pcs+1] = -1;
315
316 a[pcs+1][1] = UBound;
317 a[pcs+1][pcs+1] = -1;
318 a[pcs+1][pcs+pc+(4*fp)+(5*sf)+4] = -1;
319
320 a[pcs+pc+(4*fp)+(5*sf)+13][1] = LBound;
321 a[pcs+pc+(4*fp)+(5*sf)+13][pcs+1] = -1;
322 a[pcs+pc+(4*fp)+(5*sf)+13][pcs+(2*pc)+(8*fp)+(10*sf)+16] = 1;
323
324 T = Torque(Com,point[C].globalcp,point[C].globaln);
325
326 a[pc+(4*fp)+(5*sf)+2][pcs+1] = -point[C].globaln.z;
327 a[pc+(4*fp)+(5*sf)+3][pcs+1] = -point[C].globaln.y;
328 a[pc+(4*fp)+(5*sf)+4][pcs+1] = -point[C].globaln.x;
329
330 a[pc+(4*fp)+(5*sf)+8][pcs+1] = point[C].globaln.z;
331 a[pc+(4*fp)+(5*sf)+9][pcs+1] = point[C].globaln.y;
332 a[pc+(4*fp)+(5*sf)+10][pcs+1] = point[C].globaln.x;
333
334 a[pc+(4*fp)+(5*sf)+5][pcs+1] = -T.z;
335 a[pc+(4*fp)+(5*sf)+6][pcs+1] = -T.y;
336 a[pc+(4*fp)+(5*sf)+7][pcs+1] = -T.x;
337
338 a[pc+(4*fp)+(5*sf)+11][pcs+1] = T.z;
339 a[pc+(4*fp)+(5*sf)+12][pcs+1] = T.y;
340 a[pc+(4*fp)+(5*sf)+13][pcs+1] = T.x;
341       }
342 }
343
344 void add_fps(a,point,pc,fp,sf,N,Com,UBound,LBound)
345   int N,pc,fp,sf;
346   float UBound,LBound;
347   double **a;
348   vector Com;
```

```
349   contact point[];
350 {
351   int fps,C,c1;
352   vector T;
353
354   fps = 0;
355
356   for (C=0; C < N; C++)
357     if (point[C].ftype==2)
358       {
359 fps++;
360
361 for(c1=0;c1<4;c1++)
362   {
363     a[1][pc+(4*(fps-1))+c1+2] = -1;
364
365     a[pc+(4*(fps-1))+c1+2][1] = UBound;
366     a[pc+(4*(fps-1))+c1+2][pc+(4*(fps-1))+c1+2] = -1;
367     a[pc+(4*(fps-1))+c1+2][(2*pc)+(4*fp)+(5*sf)+(4*(fps-1))+c1+5] = -1;
368
369     a[(2*pc)+(4*fp)+(5*sf)+fps+13][pc+(4*(fps-1))+c1+2] = -1;
370
371     a[pc+(4*fp)+(5*sf)+2][pc+(4*(fps-1))+c1+2] = -point[C].cone[c1].z;
372     a[pc+(4*fp)+(5*sf)+3][pc+(4*(fps-1))+c1+2] = -point[C].cone[c1].y;
373     a[pc+(4*fp)+(5*sf)+4][pc+(4*(fps-1))+c1+2] = -point[C].cone[c1].x;
374
375     a[pc+(4*fp)+(5*sf)+8][pc+(4*(fps-1))+c1+2] = point[C].cone[c1].z;
376     a[pc+(4*fp)+(5*sf)+9][pc+(4*(fps-1))+c1+2] = point[C].cone[c1].y;
377     a[pc+(4*fp)+(5*sf)+10][pc+(4*(fps-1))+c1+2] = point[C].cone[c1].x;
378
379     T = Torque(Com,point[C].globalcp,point[C].cone[c1]);
380
381     a[pc+(4*fp)+(5*sf)+5][pc+(4*(fps-1))+c1+2] = -T.z;
382     a[pc+(4*fp)+(5*sf)+6][pc+(4*(fps-1))+c1+2] = -T.y;
383     a[pc+(4*fp)+(5*sf)+7][pc+(4*(fps-1))+c1+2] = -T.x;
384
385     a[pc+(4*fp)+(5*sf)+11][pc+(4*(fps-1))+c1+2] = T.z;
386     a[pc+(4*fp)+(5*sf)+12][pc+(4*(fps-1))+c1+2] = T.y;
```



```
387     a[pc+(4*fp)+(5*sf)+13][pc+(4*(fps-1))+c1+2] = T.x;
388   }
389
390 a[(2*pc)+(4*fp)+(5*sf)+fps+13][1] = LBound;
391 a[(2*pc)+(4*fp)+(5*sf)+fps+13][(3*pc)+(8*fp)+(10*sf)+fps+16] = 1;
392
393       }
394 }
395
396 void add_sfs(a,point,pc,fp,sf,N,Com,UBound,LBound)
397   int N,pc,fp,sf;
398   float UBound,LBound;
399   double **a;
400   vector Com;
401   contact point[];
402 {
403   int sfs,C,c1;
404   vector T;
405
406   sfs = 0;
407
408   for (C=0; C < N; C++)
409     if (point[C].ftype==3)
410       {
411 sfs++;
412
413 for(c1=0;c1<4;c1++)
414   {
415     a[1][pc+(4*fp)+(5*(sfs-1))+c1+2] = -1;
416
417     a[pc+(4*fp)+(5*(sfs-1))+c1+2][1] = UBound;
418     a[pc+(4*fp)+(5*(sfs-1))+c1+2][pc+(4*fp)+(5*(sfs-1))+c1+2] = -1;
419     a[pc+(4*fp)+(5*(sfs-1))+c1+2][(2*pc)+(8*fp)
420   +(5*sf)+(5*(sfs-1))+c1+5] = -1;
421
422     a[(2*pc)+(5*fp)+(5*sf)+sfs+13][pc+(4*fp)+(5*(sfs-1))+c1+2] = -1;
423
424     a[pc+(4*fp)+(5*sf)+2][pc+(4*fp)+(5*(sfs-1))+c1+2] =
```

```
425       -point[C].cone[c1].z;
426     a[pc+(4*fp)+(5*sf)+3][pc+(4*fp)+(5*(sfs-1))+c1+2] =
427       -point[C].cone[c1].y;
428     a[pc+(4*fp)+(5*sf)+4][pc+(4*fp)+(5*(sfs-1))+c1+2] =
429       -point[C].cone[c1].x;
430
431     a[pc+(4*fp)+(5*sf)+8][pc+(4*fp)+(5*(sfs-1))+c1+2] =
432       point[C].cone[c1].z;
433     a[pc+(4*fp)+(5*sf)+9][pc+(4*fp)+(5*(sfs-1))+c1+2] =
434       point[C].cone[c1].y;
435     a[pc+(4*fp)+(5*sf)+10][pc+(4*fp)+(5*(sfs-1))+c1+2] =
436       point[C].cone[c1].x;
437
438     T = Torque(Com,point[C].points[c1],point[C].cone[c1]);
439
440     a[pc+(4*fp)+(5*sf)+5][pc+(4*fp)+(5*(sfs-1))+c1+2] = -T.z;
441     a[pc+(4*fp)+(5*sf)+6][pc+(4*fp)+(5*(sfs-1))+c1+2] = -T.y;
442     a[pc+(4*fp)+(5*sf)+7][pc+(4*fp)+(5*(sfs-1))+c1+2] = -T.x;
443
444     a[pc+(4*fp)+(5*sf)+11][pc+(4*fp)+(5*(sfs-1))+c1+2] = T.z;
445     a[pc+(4*fp)+(5*sf)+12][pc+(4*fp)+(5*(sfs-1))+c1+2] = T.y;
446     a[pc+(4*fp)+(5*sf)+13][pc+(4*fp)+(5*(sfs-1))+c1+2] = T.x;
447   }
448
449 a[(2*pc)+(5*fp)+(5*sf)+sfs+13][1] = LBound;
450 a[(2*pc)+(5*fp)+(5*sf)+sfs+13][(3*pc)+(9*fp)+(10*sf)+sfs+16] = 1;
451
452 a[1][pc+(4*fp)+(5*sfs)+1] = -1;
453 a[pc+(4*fp)+(5*sfs)+1][1] = UBound;
454 a[pc+(4*fp)+(5*sfs)+1][pc+(4*fp)+(5*sfs)+1] = -1;
455 a[pc+(4*fp)+(5*sfs)+1][(2*pc)+(8*fp)+(5*sf)+(5*sfs)+4] = -1;
456
457 a[pc+(4*fp)+(5*sf)+5][pc+(4*fp)+(5*(sfs-1))+6] = -point[C].globaln.z;
458 a[pc+(4*fp)+(5*sf)+6][pc+(4*fp)+(5*(sfs-1))+6] = -point[C].globaln.y;
459 a[pc+(4*fp)+(5*sf)+7][pc+(4*fp)+(5*(sfs-1))+6] = -point[C].globaln.x;
460
461 a[pc+(4*fp)+(5*sf)+11][pc+(4*fp)+(5*(sfs-1))+6] = point[C].globaln.z;
462 a[pc+(4*fp)+(5*sf)+12][pc+(4*fp)+(5*(sfs-1))+6] = point[C].globaln.y;
```



```
463 a[pc+(4*fp)+(5*sf)+13][pc+(4*fp)+(5*(sfs-1))+6] = point[C].globaln.x;
464       }
465 }
466
467 void add_gravity(a,pc,fp,sf,G)
468   int pc,fp,sf;
469   vector G;
470   double **a;
471
472 {
473   a[(2*pc)+(5*fp)+(6*sf)+14][pc+(4*fp)+(5*sf)+2] = -1;
474   a[(2*pc)+(5*fp)+(6*sf)+15][pc+(4*fp)+(5*sf)+3] = -1;
475   a[(2*pc)+(5*fp)+(6*sf)+16][pc+(4*fp)+(5*sf)+4] = -1;
476
477   if (G.z >= 0) {
478     a[(2*pc)+(5*fp)+(6*sf)+14][1] = G.z;
479     a[pc+(4*fp)+(5*sf)+2][pc+(4*fp)+(5*sf)+2] = -1;
480     a[pc+(4*fp)+(5*sf)+8][pc+(4*fp)+(5*sf)+2] = 1; }
481   else {
482     a[(2*pc)+(5*fp)+(6*sf)+14][1] = -G.z;
483     a[pc+(4*fp)+(5*sf)+2][pc+(4*fp)+(5*sf)+2] = 1;
484     a[pc+(4*fp)+(5*sf)+8][pc+(4*fp)+(5*sf)+2] = -1; }
485
486   if (G.y >= 0) {
487     a[(2*pc)+(5*fp)+(6*sf)+15][1] = G.y;
488     a[pc+(4*fp)+(5*sf)+3][pc+(4*fp)+(5*sf)+3] = -1;
489     a[pc+(4*fp)+(5*sf)+9][pc+(4*fp)+(5*sf)+3] = 1; }
490   else {
491     a[(2*pc)+(5*fp)+(6*sf)+15][1] = -G.y;
492     a[pc+(4*fp)+(5*sf)+3][pc+(4*fp)+(5*sf)+3] = 1;
493     a[pc+(4*fp)+(5*sf)+9][pc+(4*fp)+(5*sf)+3] = -1; }
494
495   if (G.x >= 0) {
496     a[(2*pc)+(5*fp)+(6*sf)+16][1] = G.x;
497     a[pc+(4*fp)+(5*sf)+4][pc+(4*fp)+(5*sf)+4] = -1;
498     a[pc+(4*fp)+(5*sf)+10][pc+(4*fp)+(5*sf)+4] = 1; }
499   else {
500     a[(2*pc)+(5*fp)+(6*sf)+16][1] = -G.x;
```

```
501     a[pc+(4*fp)+(5*sf)+4][pc+(4*fp)+(5*sf)+4] = 1;
502     a[pc+(4*fp)+(5*sf)+10][pc+(4*fp)+(5*sf)+4] = -1; }
503
504 }
505
506 void add_tolerances(a,pc,fp,sf,tol)
507   int pc,fp,sf;
508   float tol;
509   double **a;
510 { int C;
511
512   for (C=0;C<12;C++)
513     {
514       a[pc+(4*fp)+(5*sf)+2+C][1] = tol;
515       a[pc+(4*fp)+(5*sf)+2+C][(2*pc)+(8*fp)+(10*sf)+5+C] = -1;
516     };
517
518 }
519
520 void static_analysis(a,N,G,Com,pc,fp,sf,point,tol,ub,lb,ip,icase)
521   contact point[];
522   int N,pc,fp,sf,*icase,ip[];
523   vector G,Com;
524   float tol,ub,lb;
525   double **a;
526 { int ic,iz[100],C,c1,c2;
527
528   for (c1=0; c1 < 100; c1++)
529     {
530       ip[c1] = 0;
531       iz[c1] = 0;
532     }
533
534   add_pcs(a,point,pc,fp,sf,N,Com,ub,lb);
535   add_fps(a,point,pc,fp,sf,N,Com,ub,lb);
536   add_sfs(a,point,pc,fp,sf,N,Com,ub,lb);
537   add_gravity(a,pc,fp,sf,G);
538   add_tolerances(a,pc,fp,sf,tol);
```



```
539
540   for (c1=1; c1 <= (3*pc)+(9*fp)+(11*sf)+16; c1++)
541     for (c2=1; c2 <= (2*pc)+(5*fp)+(6*sf)+16; c2++)
542       a[(2*pc)+(5*fp)+(6*sf)+17][c1] -= a[c2+1][c1];
543
544   /* Displaymatrix(a,(2*pc)+(5*fp)+(6*sf)+17,(3*pc)+(9*fp)+(11*sf)+16); */
545
546   simplx(a,N+pc+(4*fp)+(5*sf)+15,pc+(4*fp)+(5*sf)+3,pc+(4*fp)+(5*sf)+12,N,3,
547  &ic,iz,ip);
548
549   *icase = ic;
550
551
552   /* printf("\n Value of ic : %d",ic);
553   for (c1=1;c1<=pc+(4*fp)+(5*sf)+3;c1++) printf("\n iz[%d] : %d",c1,iz[c1]);
554   for (c1=1;c1<=N+pc+(4*fp)+(5*sf)+15;c1++) printf("\n ip[%d] : %d",c1,ip[c1]);
555
556   Displaymatrix(a,(2*pc)+(5*fp)+(6*sf)+17,(3*pc)+(9*fp)+(11*sf)+16); */
557
558 }
559
560
561
562
563
564
565
566 /*
567   File : Stability.c
568 */
569
570 #include <stdio.h>
571 #include <stdlib.h>
572
573
574 void stability_analysis(a,N,G,Com,pc,fp,sf,point,tol,ub,lb,ip,disp)
575   contact point[];
576   int N,pc,fp,sf,ip[];
```

```
577   vector G,Com;
578   float tol,ub,lb,disp;
579   double **a;
580
581 {int icase,c1;
582  float ang,ang2,pi,per;
583  vector v1,v2,v3,v4;
584
585  pi = 3.14159265;
586
587  disp *= pi/180;
588
589  if (Length(G)==0)
590    {
591      printf("\n\n\n Cannot perform stability analysis : G = (0,0,0) \n\n\n");
592      exit(0);
593    };
594
595
596  c1 = 1; v1 = Rottoz(G);
597
598  for(ang=0; ang <= disp; ang += disp/20)
599    {
600      v2 = Rotaboutx(v1,ang);
601      for(ang2=pi/20; ang2 <= 2*pi; ang2 += pi/20)
602       {
603  v3 = Rotaboutz(v2,ang2);
604  v4 = Rotfromz(G,v3);
605
606  Clearmatrix(a,(2*pc)+(5*fp)+(8*sf)+17,(3*pc)+(9*fp)+(11*sf)+16);
607
608  static_analysis(a,N,v4,Com,pc,fp,sf,point,tol,ub,lb,ip,&icase);
609
610  if (icase==0)
611    {
612      printf("o"); c1++;
613    }
614  else
```



```
615    printf("x");
616        }
617      printf("  ang = %5.2f",ang*180/pi);
618      printf("  G = (%f %f %f) \n",v4.x,v4.y,v4.z);
619    }
620
621  per = c1/8.2;
622
623  printf("\n\n\n Grasp was in equilibrium for for %d out of 820 ",c1);
624  printf("perturbations of G, (=%4.2f percent)\n\n\n",per);
625
626 }
627 /*
628   File : Frict.c
629 */
630
631 #include <stdio.h>
632 #include <stdlib.h>
633
634 void fpCone(vector V, float ang, vector cone[])
635 {vector Temp, TempV, RotV;
636  int c1;
637
638  if (V.y != 0)
639    (TempV = V);
640  else {
641    TempV = Trans(V);
642  };
643
644  RotV = Rottoz(TempV);
645
646  cone[0] = Rotaboutx(RotV, ang);
647  cone[1] = Rotaboutx(RotV, -ang);
648  cone[2] = Rotabouty(RotV, ang);
649  cone[3] = Rotabouty(RotV, -ang);
650
651
652 for (c1=0; c1 < 4; c1++)
```

```
653   {
654     Temp = Rotfromz(TempV, cone[c1]);
655
656     if (V.y != 0)
657       (cone[c1] = Temp);
658     else {
659       cone[c1] = Trans(Temp);
660     };
661
662   };
663
664 }
665
666 /*
667   File : Soft.c
668 */
669
670 #include <stdio.h>
671 #include <stdlib.h>
672
673
674 void sfCone(contact ct, vector cone[], vector Points[])
675 { float x,y,maxx,maxy,minx,miny;
676   vector norm,maxnorm,minnorm,point[4],Temp;
677   int c1,sign;
678
679   maxx = ct.frad; maxy = 0;
680
681   Temp.x = maxx+ct.localcp.x;
682   Temp.y = maxy+ct.localcp.y;
683   Temp.z = zValue(ct.contactregion, Temp.x, Temp.y);
684
685   maxnorm = Normaltobiquad(ct.contactregion, Temp);
686
687   minx = 0; miny = ct.frad;
688
689   Temp.x = minx+ct.localcp.x;
690   Temp.y = miny+ct.localcp.y;
```



```
691   Temp.z = zValue(ct.contactregion, Temp.x, Temp.y);
692
693   minnorm = Normaltobiquad(ct.contactregion, Temp);
694
695   for(sign=-1;sign<=1;sign+=2)
696     {
697       for(x=-ct.frad;x<=ct.frad;x+=(ct.frad/100))
698 {
699   y = sign*(sqrt((ct.frad*ct.frad)-(x*x)));
700
701   Temp.x = x+ct.localcp.x;
702   Temp.y = y+ct.localcp.y;
703   Temp.z = zValue(ct.contactregion, Temp.x, Temp.y);
704
705   norm = Normaltobiquad(ct.contactregion, Temp);
706
707   if ((Length(norm) > Length(maxnorm))
708       && (x*norm.x >= 0)
709       && (y*norm.y >= 0))
710     { maxnorm = norm;
711       maxx = x;
712       maxy = y;
713     }
714
715   if ((Length(norm) < Length(minnorm))
716       && (x*norm.x >= 0)
717       && (y*norm.y >= 0))
718     { minnorm = norm;
719       minx = x;
720       miny = y;
721     }
722 }
723     }
724
725   point[0].x = maxx+ct.localcp.x;  point[0].y = maxy+ct.localcp.y;
726   point[1].x = -maxx+ct.localcp.x; point[1].y = -maxy+ct.localcp.y;
727   point[2].x = minx+ct.localcp.x;  point[2].y = miny+ct.localcp.y;
728   point[3].x = -minx+ct.localcp.x; point[3].y = -miny+ct.localcp.y;
```

```
729
730   for(c1=0;c1<4;c1++)
731     {
732       point[c1].z = zValue(ct.contactregion, point[c1].x, point[c1].y);
733       Points[c1] = Transformation(ct, point[c1]);
734
735       cone[c1] = Normaltobiquad(ct.contactregion, point[c1]);
736
737       if (point[c1].x*cone[c1].x < 0)
738 cone[c1].x = 0;
739       if (point[c1].y*cone[c1].y < 0)
740 cone[c1].y = 0;
741
742       Temp = Rottox(point[c1],cone[c1]);
743       Temp = Rotaboutx(Temp,ct.friction);
744       Temp = Rotfromx(point[c1],Temp);
745       Temp = Normalise(Temp);
746
747       cone[c1] = Rotation(ct,Temp);
748
749     }
750
751 }
752 /*
753   File : Dispforce.c
754 */
755
756 #include <stdio.h>
757 #include <stdlib.h>
758
759 void Display_forces(a,point,ip,pc,fp,sf,Com)
760   double **a;
761   contact point[];
762   int ip[],pc,fp,sf;
763   vector Com;
764 {int c1,c2,pcs,fps,sfs,tmp1;
765  vector f,t,t2,tmp2;
766
```



```
767 pcs = fps = sfs = 0;
768
769 printf("\n\n\n Here are the force/torque values of each finger ... \n\n");
770
771 for (c1 = 0; c1 < pc+fp+sf; c1++)
772   {
773     f.x = f.y = f.z = t.x = t.y = t.z = t2.x = t2.y = t2.z = 0;
774
775     printf("  Finger %d ",c1+1);
776
777     if (point[c1].ftype == 1)
778       {
779 printf("(frictionless point) : "); pcs++;
780 for (c2=1;c2<=(2*pc)+(5*fp)+(6*sf)+15;c2++)
781   if (ip[c2] == pcs)
782     {
783       f = multV(a[c2+1][1], point[c1].globaln);
784       t = Torque(Com, point[c1].globalcp,point[c1].globaln);
785       t = multV(a[c2+1][1],t);
786     }
787 printf("\n   force(x,y,z)  = (%f, %f, %f)",f.x,f.y,f.z);
788 printf("\n   torque(x,y,z) = (%f, %f, %f)",t.x,t.y,t.z);
789       }
790
791     if (point[c1].ftype == 2)
792       {
793 printf("(frictional point) :   "); fps++;
794 for (c2=1;c2<=(2*pc)+(5*fp)+(6*sf)+15;c2++)
795   if (ip[c2] > pc+(4*(fps-1)) && ip[c2] <= pc+(4*fps))
796     {
797       tmp1 = ip[c2]-pc-(4*(fps-1))-1;
798       tmp2 = multV(a[c2+1][1], point[c1].cone[tmp1]);
799       f = addV(f,tmp2);
800       tmp2 = Torque(Com, point[c1].globalcp, point[c1].cone[tmp1]);
801       tmp2 = multV(a[c2+1][1],tmp2);
802       t = addV(t,tmp2);
803     }
804 printf("\n   force(x,y,z)  = (%f, %f, %f)",f.x,f.y,f.z);
```

```
805  printf("\n   torque(x,y,z) = (%f, %f, %f)",t.x,t.y,t.z);
806        }
807
808      if (point[c1].ftype == 3)
809        {
810  printf("(soft fingered) :      "); sfs++;
811  for (c2=1;c2<=(2*pc)+(5*fp)+(6*sf)+16;c2++)
812    if (ip[c2] > pc+(4*fp)+(5*(sfs-1)) && ip[c2] <= pc+(4*fp)+(5*sfs))
813      {
814        tmp1 = ip[c2]-pc-(4*fp)-(5*(sfs-1))-1;
815        if (tmp1 < 4) {
816  tmp2 = multV(a[c2+1][1], point[c1].cone[tmp1]);
817  f = addV(f,tmp2);
818  tmp2 = Torque(Com, point[c1].points[tmp1],
819        point[c1].cone[tmp1]);
820  tmp2 = multV(a[c2+1][1],tmp2);
821  t = addV(t,tmp2);
822        }
823        else {
824  t2 = multV(a[c2+1][1], point[c1].globaln);
825        }
826      }
827  printf("\n   force(x,y,z)  = (%f, %f, %f)",f.x,f.y,f.z);
828  printf("\n   torque(x,y,z) = (%f, %f, %f)",t.x,t.y,t.z);
829  printf("\n   tnorm(x,y,z)  = (%f, %f, %f)",t2.x,t2.y,t2.z);
830        }
831
832      printf("\n\n");
833
834    }
835 }
836 /*
837   File : Misc.c
838 */
839
840 #include <stdio.h>
841 #include <stdlib.h>
842
```



```
843 vector addV(vector v1, vector v2)
844 {vector Result;
845
846  Result.x = v1.x + v2.x;
847  Result.y = v1.y + v2.y;
848  Result.z = v1.z + v2.z;
849
850  return Result;
851 }
852
853 vector multV(float f, vector v)
854 {vector Result;
855
856  Result.x = f * v.x;
857  Result.y = f * v.y;
858  Result.z = f * v.z;
859
860  return Result;
861 }
862
863 vector Normalise(vector Vec)
864 {float Nval;
865  vector Grav;
866
867  Grav = Vec;
868  Nval = sqrt((Vec.x * Vec.x) +
869              (Vec.y * Vec.y) +
870              (Vec.z * Vec.z));
871
872  Grav.x /= Nval;
873  Grav.y /= Nval;
874  Grav.z /= Nval;
875
876  return Grav;
877 }
878
879 vector Transformation(contact Ct, vector Vec)
880 {vector Result;
```

```
881
882  Result.x = ((Ct.convmatrix[0][0] * Vec.x) +
883              (Ct.convmatrix[1][0] * Vec.y) +
884              (Ct.convmatrix[2][0] * Vec.z) +
886               Ct.convmatrix[3][0]);
886
887  Result.y = ((Ct.convmatrix[0][1] * Vec.x) +
888              (Ct.convmatrix[1][1] * Vec.y) +
889              (Ct.convmatrix[2][1] * Vec.z) +
890               Ct.convmatrix[3][1]);
891
892  Result.z = ((Ct.convmatrix[0][2] * Vec.x) +
893              (Ct.convmatrix[1][2] * Vec.y) +
894              (Ct.convmatrix[2][2] * Vec.z) +
895               Ct.convmatrix[3][2]);
896
897  return Result;
898 }
899
900 vector Rotation(contact Ct, vector Vec)
901 {vector Result;
902
903  Result.x = ((Ct.convmatrix[0][0] * Vec.x) +
904              (Ct.convmatrix[1][0] * Vec.y) +
905              (Ct.convmatrix[2][0] * Vec.z));
906
907  Result.y = ((Ct.convmatrix[0][1] * Vec.x) +
908              (Ct.convmatrix[1][1] * Vec.y) +
909              (Ct.convmatrix[2][1] * Vec.z));
910
911  Result.z = ((Ct.convmatrix[0][2] * Vec.x) +
912              (Ct.convmatrix[1][2] * Vec.y) +
913              (Ct.convmatrix[2][2] * Vec.z));
914
915  return Result;
916 }
917
918 vector Normaltobiquad(fit region, vector point)
```



```
919 {vector Result;
920
921  Result.x = (2 * region.xx * point.x) +
922        (region.xy * point.y) +
923        region.x;
924
925  Result.y = (2 * region.yy * point.y) +
926        (region.xy * point.x) +
927        region.y;
928
929  Result.z = -1;
930
931  return Result;
932 }
933
934 vector Torque(vector com, vector cp, vector norm)
935 {
936   vector Result,v1;
937
938   v1.x = com.x - cp.x;
939   v1.y = com.y - cp.y;
940   v1.z = com.z - cp.z;
941
942   Result.x = v1.y*norm.z - v1.z*norm.y;
943   Result.y = v1.z*norm.x - v1.x*norm.z;
944   Result.z = v1.x*norm.y - v1.y*norm.x;
945
946   return(Result);
947
948 }
949
950 float Length(vector V)
951 {float Result;
952
953  Result = sqrt((V.x * V.x) +
954                (V.y * V.y) +
955                (V.z * V.z));
956
```

```
 return Result;

}

vector Trans(vector V)
{vector Result;

 Result.x = V.y;
 Result.y = V.x;
 Result.z = V.z;

 return Result;

}

vector Rottoz(vector V)
{vector Result;
 float c1,c2;

 c1 = sqrt((V.y * V.y) + (V.z * V.z));
 c2 = sqrt((V.x * V.x) + (V.y * V.y) + (V.z * V.z));

 Result.x = ((c1/c2) * V.x) + ((-V.x*V.y/(c1*c2)) * V.y)
          + ((-V.x*V.z/(c1*c2)) * V.z);
 Result.y = ((V.z/c1) * V.y) + ((-V.y/c1) * V.z);
 Result.z = ((V.x/c2) * V.x) + ((V.y/c2) * V.y)
          + ((V.z/c2) * V.z);

 return Result;

}

vector Rotfromz(vector V, vector V2)
{vector Result;
 float c1,c2;

 c1 = sqrt((V.y * V.y) + (V.z * V.z));
 c2 = sqrt((V.x * V.x) + (V.y * V.y) + (V.z * V.z));
```



```

 Result.x = ((c1/c2) * V2.x) + ((V.x/c2) * V2.z);
 Result.y = ((-V.x*V.y/(c1*c2)) * V2.x) + ((V.z/c1) * V2.y)
          + ((V.y/c2) * V2.z);
 Result.z = ((-V.x*V.z/(c1*c2)) * V2.x) + ((-V.y/c1) * V2.y)
          + ((V.z/c2) * V2.z);

 return Result;

}

vector Rottox(vector V, vector V2)
{vector Result;
 float c1,c2;

 c1 = (V.x / sqrt((V.x * V.x) + (V.y * V.y)));
 c2 = (V.y / sqrt((V.x * V.x) + (V.y * V.y)));

 Result.x = (c2 * V2.x) - (c1 * V2.y);
 Result.y = (c1 * V2.x) + (c2 * V2.y);
 Result.z = V2.z;

 return Result;

}

vector Rotfromx(vector V, vector V2)
{vector Result;
 float c1,c2;

 c1 = (V.x / sqrt((V.x * V.x) + (V.y * V.y)));
 c2 = (V.y / sqrt((V.x * V.x) + (V.y * V.y)));

 Result.x = (c1 * V2.y) + (c2 * V2.x);
 Result.y = (-c1 * V2.x) + (c2 * V2.y);
 Result.z = V2.z;

 return Result;
```

```
1033
1034 }
1035
1036 vector Rotaboutx(vector V, float ang)
1037 {vector Result;
1038
1039  Result.x = V.x;
1040  Result.y = (cos(ang)*V.y) - (sin(ang)*V.z);
1041  Result.z = (sin(ang)*V.y) + (cos(ang)*V.z);
1042
1043  return Result;
1044
1045 }
1046
1047 vector Rotabouty(vector V, float ang)
1048 {vector Result;
1049
1050  Result.x = (cos(ang)*V.x) + (sin(ang)*V.z);
1051  Result.y = V.y;
1052  Result.z = (-sin(ang)*V.x) + (cos(ang)*V.z);
1053
1054  return Result;
1055
1056 }
1057
1058 vector Rotaboutz(vector V, float ang)
1059 {vector Result;
1060
1061  Result.x = (cos(ang)*V.x) - (sin(ang)*V.y);
1062  Result.y = (sin(ang)*V.x) + (cos(ang)*V.y);
1063  Result.z = V.z;
1064
1065  return Result;
1066
1067 }
1068
1069 float zValue(fit Biquad, float x, float y)
1070 {float Result;
```



```
1071
1072  Result =   (Biquad.xx*x*x) + (Biquad.yy*y*y)
1073           + (Biquad.xy*x*y) + (Biquad.x*x)
1074   + (Biquad.y*y)    + (Biquad.c);
1075
1076  return Result;
1077
1078 }
1079
1080 void Displaymatrix(double **a, int row, int col)
1081 {int c1,c2;
1082
1083  printf("\n\n State of Simplex matrix : \n\n");
1084
1085  for (c1=1;c1<=row;c1++)
1086    {
1087      for (c2=1;c2<=col;c2++) printf("%8.2f ",a[c1][c2]);
1088      printf("\n");
1089    };
1090
1091 }
1092
1093 void Clearmatrix(double **a, int row, int col)
1094 {int c1,c2;
1095
1096  for (c1=1;c1<=row;c1++)
1097      for (c2=1;c2<=col;c2++) a[c1][c2] = 0;
1098
1099 }
```