Proceedings of the 2006 IEEE International Conference on Robotics and Automation
Orlando, Florida - May 2006

A Fast and Adaptive Test of Static Equilibrium for Legged Robots

Timothy Bretl
Department of Computer Science
Stanford University
Stanford, California 94305
Email: tbretl@stanford.edu

Sanjay Lall
Department of Aeronautics and Astronautics
Stanford University
Stanford, California 94305
Email: lall@stanford.edu

Abstract— **A legged robot walking on uneven terrain can avoid falling only by applying contact forces with its feet on the ground that compensate for gravity without causing slip. To plan safe motions, it is necessary to test this constraint at every posture explored at each set of foot placements. Since a huge number of postures may be explored, this test must be as fast as possible. Existing approaches either search explicitly for contact forces at each posture, or precompute the support polygon and check that the robot's center of mass lies above it. This paper presents a new algorithm that is faster than either existing approach. This algorithm is an incremental method of projection, that computes only enough of the support polygon to decide whether static equilibrium is possible at each posture. It takes advantage of information gained testing previous postures in order to test subsequent postures more quickly.**

I. Introduction

The primary concern of a legged robot walking on uneven terrain is to avoid falling. Assuming its motion is slow enough to neglect inertia, the robot must always be able to achieve *static equilibrium*—that is, to apply contact forces with its feet on the ground that compensate for gravity without causing slip. In some situations, specific gaits can guarantee the satisfaction of this constraint (for example, walking on level ground or climbing parallel-sided pipes). However, recent work considers legged locomotion on irregular and steep terrain [1]–[6], even free-climbing on vertical rock [7]. In these situations, motion is explicitly planned and non-gaited, so it is necessary to check that the robot is in equilibrium at every posture. Because many postures may be considered, this test must be as fast as possible. In particular, it is important to quickly invalidate postures at which there exists no set of contact forces achieving static equilibrium. In this paper, we present a fast and adaptive algorithm that tests necessary conditions for equilibrium. With some additional assumptions (controllability, joint torque limits) these conditions are also sufficient.

A. Support polygons

Gravity acts at the robot's center of mass (CM), the position of which varies as the robot moves. The properties of each foot placement determine the range of contact forces possible without slip. So, for fixed foot placements, static equilibrium jointly constrains both the contact forces and the CM position.

On flat terrain with point contacts, we have an intuitive notion of what this constraint means. Simply, the robot's CM must lie above the base of its supporting feet. The region between these supports is called the *support polygon*. The vertical prism having this polygon as cross-section is the set of all CM positions at which static equilibrium is possible. In other words, if the robot's posture places its CM over the support polygon, then we know contact forces exist that achieve equilibrium without actually having to compute them.

On uneven terrain, however, the shape of the support polygon depends on the properties of each foot-ground contact, and does not necessarily coincide with the base of the supports. In particular, for fixed foot placements it may be more costly to precompute the support polygon than to search explicitly for a set of contact forces at each CM position.

B. Linear programming and linear projection

A basic model of foot-ground interaction assumes that the terrain is rigid and that contact occurs at frictional points. Under this assumption, "no slip" means that each contact force is restricted to a second-order friction cone, the shape of which is often approximated by a set of linear inequalities. Likewise, "compensate for gravity" means that the contact forces and CM positions satisfy linear force and moment balance equations. So for fixed foot placements, subject to the above approximation, the set of jointly feasible contact forces and CM positions is a polyhedron. The support polygon is the projection of this polyhedron onto CM-space.

Using linear programming, we can search explicitly for a set of contact forces that place a particular CM position in equilibrium without computing the support polygon. A wide variety of fast algorithms are available, such as the simplex method, the ellipsoid method, and interior-point methods [8], [9]. Similarly, using linear projection, we can precompute the support polygon and thus determine whether it is possible to place a particular CM position in equilibrium without computing contact forces. In other words, we can eliminate forces from the equilibrium constraint. This problem has also received considerable attention [10], [11].

There is a clear tradeoff between these two approaches when they are used to test static equilibrium. Linear programming is very fast, but only answers the question for a single CM position. Linear projection is slower, but answers the question for every CM position. So, we might use the former to test a small number of CM positions at a fixed set of foot placements, and the latter to test a large number.

C. Incremental projection

Unfortunately, the number of CM positions that will be tested at each set of foot placements is initially unknown. For example, the planner of [7] searches for multi-step free-climbing motions by randomly sampling robot postures. As the search evolves, the planner may distribute many samples at some sets of foot placements, and few at others. So it is difficult to decide which approach—linear programming or linear projection—will be most efficient.

In fact, neither approach directly addresses the problem we want to solve. In particular, we want to determine whether static equilibrium is possible while computing neither the contact forces (as in linear programming) nor the support polygon (as in linear projection). In this paper, we present a new algorithm that is often faster than either existing approach. This algorithm is an incremental method of projection, that computes only enough of the support polygon to determine the membership of each CM position.

The position of the CM is determined by the robot's configuration, but in this paper we are not concerned with configuration space variables. Indeed, although the set of jointly feasible contact forces and CM positions is a polyhedron, the set of jointly feasible contact forces and robot configurations is not. Our algorithm is designed to test equilibrium at particular CM positions, not to compute the region of configurations mapping to CM positions in equilibrium.

II. Modeling the constraint of static equilibrium

A. Configurations and contact

Consider a legged robot walking on uneven terrain in a 3-D workspace. Suppose the robot contains p revolute joints, so its *configuration space* is $\mathcal{Q} = \mathbb{R}^3 \times SO(3) \times (S^1)^p$ and any placement of the robot in its workspace can be specified by a *configuration* $q \in \mathcal{Q}$. Some configurations may place the robot in contact with the terrain. Indeed, we allow the robot to contact the terrain anywhere with any part of its body. However, we assume there is a finite set of points $r_1, \dots, r_n$ on the surface of its body with which it can apply a force. For instance, if the robot has n legs, each r_i might be a point at the end of leg i. We define functions $r_i \colon \mathcal{Q} \to \mathbb{R}^3$ that map configurations q to points $r_i(q)$ in the workspace. We also assume there is a finite set of points $\mathcal{T} \subset \mathbb{R}^3$ on the terrain to which a force can be applied. For instance, these might correspond to the holds used by a free-climbing robot [7]. For now, we assume there is a unique outward (unit) normal vector ν at each point in $\mathcal{T}$, given by a mapping $\nu \colon \mathcal{T} \to \mathbb{R}^3$.

B. Stances and static equilibrium

We call a *stance* an association of points on the robot with points on the terrain. We define it by a map $s \colon \mathcal{A} \to \mathcal{T}$, where $\mathcal{A} \subset \{1, \dots, n\}$. At stance s, the point r_i on the robot must remain in contact with the point $s(i)$ on the terrain, for each index $i \in \operatorname{dom}(s)$. We allow any other point r_i to contact the terrain even if $i \notin \operatorname{dom}(s)$, but neither restrict this contact nor use it to avoid falling. The set of all possible stances is

$$\mathcal{S} = \{\, s \colon \mathcal{A} \to \mathcal{T} \mid \mathcal{A} \subset \{1, \dots, n\} \,\}.$$

The robot can still change its configuration while maintaining a fixed stance $s \in \mathcal{S}$. However, its configuration must remain within the following subset of $\mathcal{Q}$:

$$\mathcal{Q}_s = \{\, q \in \mathcal{Q} \mid r_i(q) = s(i) \text{ for all } i \in \operatorname{dom}(s) \,\}.$$

Further, while s is maintained, the robot may only apply forces at points on the terrain in $\operatorname{range}(s)$. For convenience of notation, we reorder these points as $r_1, \dots, r_N$ and denote the corresponding normal vectors by $\nu_i = \nu(r_i)$. Let $x_i \in \mathbb{R}^3$ be the reaction force acting on the robot at each point r_i for $i = 1, \dots, N$. This force can be decomposed into a component $\nu_i^T x_i \nu_i$ normal to the surface of the terrain (in the direction ν_i) and an orthogonal component $(I - \nu_i \nu_i^T) x_i$ tangential to the surface. Define a function $c \colon \mathcal{Q} \to \mathbb{R}^3$ mapping the robot's configuration q to the position of its CM $c(q)$ (in general, many different q might map to the same position c). Assume the robot has mass m and the acceleration due to gravity is $g \in \mathbb{R}^3$. All of these vectors are defined with respect to a global coordinate system, where $g = -\|g\| e_3$. Then we say that the robot is in *static equilibrium* at (s, q) if

$$\sum_{i=1}^{N} x_i + mg = 0 \tag{1}$$

$$\sum_{i=1}^{N} r_i \times x_i + c(q) \times mg = 0 \tag{2}$$

$$\|(I - \nu_i \nu_i^T) x_i\|_2 \leq \mu \nu_i^T x_i \text{ for all } i = 1, \dots, N. \tag{3}$$

These are convex constraints on the reaction forces $x_1, \dots, x_N$ and the CM position c. Constraints (1) and (2) ensure force and torque balance. Constraint (3) restricts each reaction force to lie in a friction cone of half-angle $\phi = \tan^{-1} \mu$, where μ is the static coefficient of friction. Finally, notice that these constraints depend neither on the particular mapping s, nor explicitly on q. Rather, they depend only on the set $\operatorname{range}(s)$ and the CM position $c(q)$.

C. Paths and planning

Static equilibrium imposes a distinct structure on configuration space—in particular, a decomposition according to stance. Define the *feasible space* at a stance as the subset of configurations $\mathcal{F}_s \subset \mathcal{Q}_s$ at which $x_1, \dots, x_N$ exist such that (1)-(3) are satisfied. Feasible spaces $\mathcal{F}_s$ and $\mathcal{F}_{s'}$ at two different stances s and s' might intersect. If the robot moves to a configuration $q \in \mathcal{F}_s \cap \mathcal{F}_{s'}$, it can take a *step* from one stance to the other. To walk across uneven terrain, the robot must take a sequence of steps, which we call a *multi-step motion*. To plan such a motion, we would like to construct a graph with vertex set

$$\mathcal{V} = \left\{ \mathcal{U} \subset \mathcal{Q} \;\middle|\; \begin{array}{l} \mathcal{U} \text{ is a connected component of } \mathcal{F}_s \\ \text{for some } s \in \mathcal{S} \end{array} \right\}$$

where there is an edge between $\mathcal{U}_1, \mathcal{U}_2 \in \mathcal{V}$ if $\mathcal{U}_1 \cap \mathcal{U}_2 \neq \emptyset$. However, constructing this graph is hard. Even computing the exact shape of a single connected component $\mathcal{U} \subset \mathcal{V}$ may take prohibitive time. For this reason, the Probabilistic-Roadmap (PRM) approach [12], or one of its variants [13], is often used to represent each $\mathcal{F}_s$. A PRM planner captures connectivity by a network of local paths joining sampled configurations. Its speed derives from the assumption that checking whether $q \in \mathcal{F}_s$ can be done quickly, even if computing $\mathcal{F}_s$ would take prohibitive time. So, it is important that the test of static equilibrium be as fast as possible.

The robot may need to satisfy constraints other than static equilibrium as well: for example, collision and self-collision, joint-angle limits, joint-torque limits, and controllability. These constraints further restrict configurations in $\mathcal{F}_s$. However, some of them (such as joint-torque limits) are costly to check. By testing static equilibrium first, we can quickly eliminate many infeasible configurations $q \notin \mathcal{F}_s$.

D. Approximating the friction cone

The constraints (1)-(3) are jointly convex in $x_1, \ldots, x_N$ and c. In particular, (1)-(2) are linear and (3) is a second-order cone constraint. Hence, it is possible to test static equilibrium (given c, to search for $x_1, \ldots, x_N$ that satisfy (1)-(3)) by solving a *second-order cone program* (SOCP) [14]–[16]. However, in practice (3) is often approximated by a polyhedral cone (that is, by linear constraints). This formulation has three advantages: first, it can be evaluated more quickly; second, it avoids representational problems on non-smooth terrain (we can easily relax the assumption of unique ν_i); and third, it allows the variables x_i to be eliminated (quantifier elimination is difficult for second-order cone constraints, since the result may not be a second-order cone). We will use the polyhedral approximation for now, returning to this issue in Section VI.

We construct a full QR factorization of each ν_i as

$$\nu_i = Q_i \begin{bmatrix} R_i \\ 0 \\ 0 \end{bmatrix}$$

where $Q_i \in \mathbb{R}^{3\times 3}$ is orthogonal and $R_i \in \mathbb{R}$ is $R_i = \|\nu_i\| = 1$. For each i, we approximate the second-order cone

$$\mathcal{C}_{\text{icecream}} = \{\, z \in \mathbb{R}^3 \mid \|(I - \nu_i \nu_i^T) z\|_2 \leq \mu \nu_i^T z \,\}$$

by the four-sided polyhedral cone

$$\mathcal{C}_{\text{poly}} = \{\, z \in \mathbb{R}^3 \mid W Q_i^T z \preceq 0 \,\}$$

where

$$W = \begin{bmatrix} -\mu & -1 & 0 \\ -\mu & 1 & 0 \\ -\mu & 0 & -1 \\ -\mu & 0 & 1 \end{bmatrix}.$$

To see that this is a reasonable approximation, notice that

$$\mathcal{C}_{\text{icecream}} = \{\, Q_i z \in \mathbb{R}^3 \mid z_2^2 + z_3^2 \leq \mu z_1^2 \text{ and } z_1 \geq 0 \,\}$$

and that

$$\mathcal{C}_{\text{poly}} = \{\, Q_i z \in \mathbb{R}^3 \mid |z_2| \leq \mu z_1 \text{ and } |z_3| \leq \mu z_1 \,\}.$$

As defined, $\mathcal{C}_{\text{poly}}$ is an outer approximation to $\mathcal{C}_{\text{icecream}}$. By adjusting the parameter μ, we could construct an inner approximation instead. Similarly, we could make the approximation more accurate by increasing the number of facets of $\mathcal{C}_{\text{poly}}$ (equivalently, by increasing the number of rows of W). The end result is to replace (3) by

$$W Q_i^T x_i \preceq 0 \text{ for all } i = 1, \ldots, N. \tag{4}$$

E. Defining a coordinate system

Here, we put (1), (2), and (4) in a form suitable for computation. First, notice that

$$c \times mg = m\|g\| \begin{bmatrix} -c_2 \\ c_1 \\ 0 \end{bmatrix}$$

so constraints (1)-(2) do not depend on c_3. Hence, we define

$$y = Pc$$

where

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

For each $z \in \mathbb{R}^3$ we define a linear map $T(z) \colon \mathbb{R}^3 \to \mathbb{R}^3$ such that $T(z)x = z \times x$ for all $x \in \mathbb{R}^3$. So (1), (2), and (4) become

$$\sum_{i=1}^{N} x_i + mg = 0 \tag{5}$$

$$\sum_{i=1}^{N} T(r_i) x_i + T(mg) P^T y = 0 \tag{6}$$

$$W Q_i^T x_i \preceq 0 \text{ for all } i = 1, \ldots, N \tag{7}$$

We now have the following simple conditions: $x_1, \ldots, x_N$ and c satisfy (1), (2), and (4) if and only if

$$\begin{aligned} A_1 x + A_2 y &\preceq t \\ B_1 x + B_2 y &= u \end{aligned} \tag{8}$$

where

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}, \quad y = Pc,$$

$$A_1 = \operatorname{diag}(W_1 Q_1^T, \ldots, W_N Q_N^T), \quad A_2 = 0, \quad t = 0,$$

$$B_1 = \begin{bmatrix} I & \ldots & I \\ T(r_1) & \ldots & T(r_N) \end{bmatrix}, \; B_2 = \begin{bmatrix} 0 \\ T(mg) P^T \end{bmatrix}, \; u = \begin{bmatrix} -mg \\ 0 \end{bmatrix}.$$

F. Problem statement

From (8), the set of all feasible reaction forces x and CM positions y at a fixed stance is the *polyhedron*

$$\mathcal{X} = \{\, x \in \mathbb{R}^{3N}, y \in \mathbb{R}^2 \mid A_1 x + A_2 y \preceq t, B_1 x + B_2 y = u \,\}.$$

Likewise, the *support polygon* (that is, the set of all feasible CM positions y at a fixed stance) is the *projection* $\mathcal{Y}$ of the polyhedron $\mathcal{X}$ onto y-space, namely

$$\mathcal{Y} = \{\, y \in \mathbb{R}^2 \mid \exists x \in \mathbb{R}^{3N} \text{ such that } (x, y) \in \mathcal{X} \,\}.$$

The problem of testing static equilibrium at a given CM position y is the problem of determining whether or not there exists an x such that $(x, y) \in \mathcal{X}$. Equivalently, it is the problem of determining whether $y \in \mathcal{Y}$.

During multi-step search, an unknown number of configurations q_i will be sampled at each stance $s \in \mathcal{S}$. We want to check that the robot is in static equilibrium at each (s, q_i). The stance s defines the polyhedron $\mathcal{X}$ and its projection $\mathcal{Y}$, and each q_i defines a point $y_i = Pc(q_i)$. So our problem is to test the membership of an unknown number of points in the projection of a polyhedron.

III. Linear Programming and Linear Projection

A. Linear programming

Finding a point x at a given y_i such that $(x, y_i) \in \mathcal{X}$ can be done by solving the *linear program* (LP) feasibility problem

$$\begin{aligned} \text{find} \quad & x \\ \text{subject to} \quad & A_1 x + A_2 y \preceq t \\ & B_1 x + B_2 y = u \\ & y = y_i. \end{aligned} \tag{9}$$

A variety of algorithms are available to solve this LP [8]. In general, these can take time polynomial in the dimension of x and the number of constraints.

B. Linear projection

Finding the set $\mathcal{Y}$ is a problem of linear projection. Since $\mathcal{X}$ is a polyhedron, its projection $\mathcal{Y}$ will also be a polyhedron. If we precompute a new set of linear inequalities such that

$$\mathcal{Y} = \{ y \in \mathbb{R}^2 \mid A_{\text{proj}} y \preceq b_{\text{proj}} \}$$

then to test whether $y_i \in \mathcal{Y}$ we simply verify $A_{\text{proj}} y_i \preceq b_{\text{proj}}$. But there is an inherent computational complexity associated with linear projection, since the number of inequalities defining the projection $\mathcal{Y}$ might be exponential in the dimensionality $3N + 2$ of the polyhedron $\mathcal{X}$. Numerical stability is also an issue, and it is difficult to know when linear projection will be easy or hard. So in general, precomputing $\mathcal{Y}$ is slower than solving (9), although checking afterward if $y_i \in \mathcal{Y}$ is faster.

Most available algorithms for linear projection are variants of *elimination* or *enumeration*. Elimination methods construct $\mathcal{Y}$ directly by computing *valid inequalities* (positive combinations of the linear inequalities defining $\mathcal{X}$) that "eliminate" the variables x one by one (see [17] for a survey). Enumeration methods construct $\mathcal{Y}$ indirectly by first "enumerating" all vertices of $\mathcal{X}$, which are then easily projected onto y-space [18]–[22]. In fact, due to a correspondence between linear inequalities and vertices (one can be represented as the other in a dual space), algorithms for elimination and enumeration are in some sense mathematically equivalent.

C. Application to multi-step planning for legged robots

The configuration space of a legged robot has a distinct structure (Section II). Each stance $s \in \mathcal{S}$ is associated with a different support polygon and a different feasible space $\mathcal{F}_s$, which is often explored using a sample-based planner (such as PRM). It is common to sample over a hundred thousand configurations during a multi-step search. Further, we might generate many samples at some stances, and few at others. Either linear programming or linear projection can be used to test static equilibrium, but the time taken by each method depends entirely on this unknown number of samples.

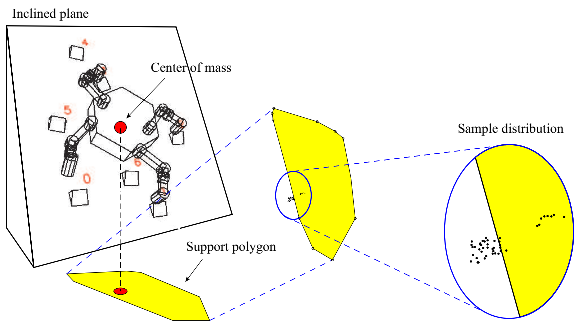


Fig. 1. Example distribution of CM positions at a fixed stance for a free-climbing robot. Each CM position corresponds to a sampled posture.

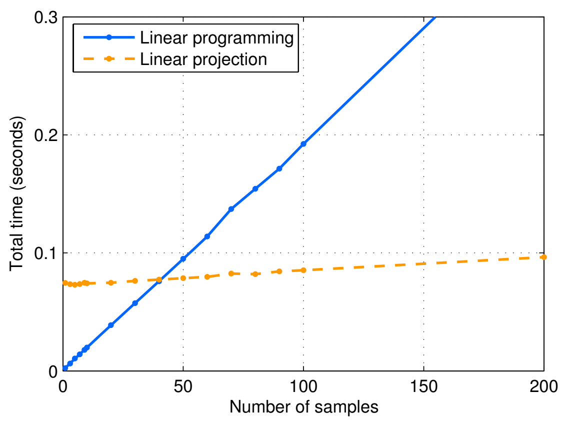


Fig. 2. The total amount of time required to test static equilibrium at a variable number of sampled postures at the stance shown in Fig. 1.

For example, consider the four-limbed, free-climbing robot of [7] at the stance s shown in Fig. 1. It needs to determine which holds it can reach, so it can decide where to step next. Some number of configurations in $\mathcal{Q}_s$ are being sampled by a PRM algorithm, in this case about seventy. The CM position is computed for each configuration—some points lie inside, and some outside, the support polygon (although the planner does not yet know which). As is typical, the distribution of CM positions is quite narrow, although the distribution of sampled configurations in joint space might have been wide.

In this example 70 configurations are sampled, but this number may vary considerably. Fig. 2 shows the time it takes to test a variable number of configurations for static equilibrium (that is, to test whether a variable number of CM positions lie over the support polygon). Linear programming is fast for a small number of samples but slow for a large number. Linear projection is the opposite. Both methods perform about the same—relatively slowly—for the number of samples in Fig. 1. These results do not depend on the sample distribution.

D. Implications

As the above example indicates, neither linear programming nor linear projection directly addresses the problem of testing static equilibrium—that is, of testing the membership of points y_i in the projection of a polyhedron $\mathcal{X}$. To do so, the former finds x such that $(x, y_i) \in \mathcal{X}$, while the latter finds the projection $\mathcal{Y}$ of $\mathcal{X}$ onto y-space and checks that $y_i \in \mathcal{Y}$. Both approaches generate useless information—neither a particular x nor the exact shape of $\mathcal{Y}$ are needed.

More importantly, both approaches discard useful information. For example, if a point $y_i \notin \mathcal{Y}$, most LP algorithms generate a separating hyperplane as proof. This hyperplane separates many other points from $\mathcal{Y}$ as well—in fact, it defines a halfspace that contains $\mathcal{Y}$ (where μ and λ are dual variables):

$$\{\, y \in \mathbb{R}^2 \mid \left(\lambda^T A_2 + \mu^T B_2\right) y \leq \left(\lambda^T b + \mu^T d\right) \,\} \supset \mathcal{Y}.$$

If any other point y_j is outside this halfspace, we know immediately that it is outside $\mathcal{Y}$, without solving another LP. Similarly, linear projection discards information about the sample distribution. Rather than compute all of $\mathcal{Y}$, it is only necessary to compute enough of $\mathcal{Y}$ to distinguish between feasible and infeasible sampled points. The evolving sample distribution suggests which part of $\mathcal{Y}$ should be computed.

IV. Incremental projection

Here, we present a fast algorithm to test static equilibrium. It is an incremental method of projection that computes only enough of the support polygon to determine the membership of each CM position.

A. Preliminaries

Let $\mathcal{Y} \subset \mathbb{R}^n$ be a closed and bounded polyhedron. A *face* of $\mathcal{Y}$ is any set of the form $\mathcal{Y} \cap \{\, z \in \mathbb{R}^n \mid a^T z = b \,\}$ for $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$ such that $a^T z \leq b$ for all $z \in \mathcal{Y}$. If such a face is nonempty and $a \neq 0$, we call the set $\{\, z \in \mathbb{R}^n \mid a^T z \leq b \,\}$ a *supporting halfspace*. A face is called *proper* if it is not identically $\mathcal{Y}$ (in 2-D, for example, vertices and edges are proper faces). It is clear that $\mathcal{Y}$ has a finite number of proper faces. We define the *support function* $S_{\mathcal{Y}} \colon \mathbb{R}^n \to \mathbb{R}$ of $\mathcal{Y}$ by

$$S_{\mathcal{Y}}(a) = \sup\{\, a^T z \mid z \in \mathcal{Y} \,\}.$$

Using the support function, we define a map $f_{\mathcal{Y}} \colon \mathbb{R}^n \to \mathbb{R}^n$ from normal vectors to nonempty faces of $\mathcal{Y}$ by

$$f_{\mathcal{Y}}(a) = \{\, z \in \mathcal{Y} \mid a^T z = S_{\mathcal{Y}}(a) \,\}.$$

The face $f_{\mathcal{Y}}(a)$ is proper for any $a \neq 0$. We also define a map $h_{\mathcal{Y}} \colon \mathbb{R}^n \to \mathbb{R}^n$ from normal vectors to supporting halfspaces:

$$h_{\mathcal{Y}}(a) = \{\, z \in \mathbb{R}^n \mid a^T z \leq S_{\mathcal{Y}}(a) \,\}.$$

Note that $f_{\mathcal{Y}}(a) \subset h_{\mathcal{Y}}(a)$ for any $a \in \mathbb{R}^n$.

B. Algorithm

Rather than precompute the projection $\mathcal{Y}$, our incremental projection algorithm TEST-SAMPLE (Fig. 3) maintains inner and outer polyhedral approximations $\mathcal{Y}_{\text{inner}} \subset \mathcal{Y}$ and $\mathcal{Y}_{\text{outer}} \supset \mathcal{Y}$, respectively. Rather than test explicitly whether $y \in \mathcal{Y}$, the algorithm loops until either $y \in \mathcal{Y}_{\text{inner}}$ (implying $y \in \mathcal{Y}$) or $y \notin \mathcal{Y}_{\text{outer}}$ (implying $y \notin \mathcal{Y}$).

```
TEST-SAMPLE($y$)
1  while $y \notin \mathcal{Y}_{\text{inner}}$ and $y \in \mathcal{Y}_{\text{outer}}$ do
2      pick $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$ such that $a^T y > b$
           and $a^T z \leq b$ for all $z \in \mathcal{Y}_{\text{inner}}$
3      $\mathcal{Y}_{\text{outer}} \leftarrow \mathcal{Y}_{\text{outer}} \cap h_{\mathcal{Y}}(a)$
4      pick $v \in f_{\mathcal{Y}}(a)$
5      $\mathcal{Y}_{\text{inner}} \leftarrow \text{conv}(\mathcal{Y}_{\text{inner}} \cup \{v\})$
6  if $y \in \mathcal{Y}_{\text{inner}}$ then
7      return TRUE
8  else
9      return FALSE
```

Fig. 3. The incremental projection algorithm.

Initially, $\mathcal{Y}_{\text{inner}} = \emptyset$ and $\mathcal{Y}_{\text{outer}} = \mathbb{R}^n$. At each iteration, we pick a hyperplane strictly separating the query point y from the inner approximation $\mathcal{Y}_{\text{inner}}$ (Line 2). This hyperplane has some normal vector a. We shrink $\mathcal{Y}_{\text{outer}}$ by intersecting it with the supporting halfspace defined by a (Line 3). We grow $\mathcal{Y}_{\text{inner}}$ by taking its convex hull with some point in the face defined by a (Lines 4-5). After TEST-SAMPLE terminates, we store $\mathcal{Y}_{\text{inner}}$ and $\mathcal{Y}_{\text{outer}}$ to make subsequent evaluations faster. In particular, the algorithm may terminate on Line 1 without iteration.

Fig. 5 shows one iteration of TEST-SAMPLE for our implementation of the algorithm in 2-D. However, before addressing issues of computation (see Section IV-E), we will first show that TEST-SAMPLE converges for arbitrary n.

C. Proof of convergence

Theorem 1: Suppose $\mathcal{Y}$ and $\mathcal{Y}_{\text{inner}}$ are closed and bounded polyhedra such that $\mathcal{Y}_{\text{inner}} \subset \mathcal{Y}$. Let $y \notin \mathcal{Y}_{\text{inner}}$. Then there exist $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$ such that $a^T y > b$ and $a^T z \leq b$ for all $z \in \mathcal{Y}_{\text{inner}}$. Also, for any such a, the following are true:

(i) there exists $v \in f_{\mathcal{Y}}(a)$,
(ii) either $f_{\mathcal{Y}}(a) \cap \mathcal{Y}_{\text{inner}} = \emptyset$ or $y \notin h_{\mathcal{Y}}(a)$ or both.

Proof: The existence of a and b follows from the Hahn-Banach Theorem. Statement (i) holds since $f_{\mathcal{Y}}(a) \neq \emptyset$ for any $a \in \mathbb{R}^n$. To show (ii), we will suppose $f_{\mathcal{Y}}(a) \cap \mathcal{Y}_{\text{inner}} \neq \emptyset$ and prove that $y \notin h_{\mathcal{Y}}(a)$. Assume there exists $w \in \mathcal{Y}_{\text{inner}} \cap f_{\mathcal{Y}}(a)$. Then $a^T w = a^T z$ for all $z \in f_{\mathcal{Y}}(a)$ (because $w \in f_{\mathcal{Y}}(a)$) and $a^T w \leq b$ (because $w \in \mathcal{Y}_{\text{inner}}$). Therefore $a^T z \leq b$ for all $z \in f_{\mathcal{Y}}(a)$. Also, for all $x \in h_{\mathcal{Y}}(a)$ and $z \in f_{\mathcal{Y}}(a)$ we know by definition that $a^T x \leq a^T z$. So $a^T x \leq b$ for all $x \in h_{\mathcal{Y}}(a)$ while $a^T y > b$, and thus $y \notin h_{\mathcal{Y}}(a)$. ■

Theorem 2: Suppose $\mathcal{Y}$ is a closed and bounded polyhedron. Then for any sequence of choices of a, b, and v, TEST-SAMPLE terminates after a finite number of iterations.

Proof: Let $\mathcal{I}$ be the set of all nonempty proper faces of $\mathcal{Y}$ that intersect $\mathcal{Y}_{\text{inner}}$. Theorem 1 implies that at each iteration (where necessarily $y \notin \mathcal{Y}_{\text{inner}}$), for any choice of a and b either TEST-SAMPLE will terminate (because $y \notin h_{\mathcal{Y}}(a)$ and hence $y \notin \mathcal{Y}_{\text{outer}} \cap h_{\mathcal{Y}}(a)$) or a new proper face will be added to $\mathcal{I}$ (because $\mathcal{Y}_{\text{inner}} \cap f_{\mathcal{Y}}(a) = \emptyset$ but

$\mathrm{conv}(\mathcal{Y}_{\text{inner}} \cup \{v\}) \cap f_{\mathcal{Y}}(a) \neq \emptyset$ for any choice of $v \in f_{\mathcal{Y}}(a)$). Since $\mathcal{Y}$ has a finite number of proper faces, TEST-SAMPLE must terminate after a finite number of iterations. ■

D. Implementation

Although we have shown that TEST-SAMPLE converges, we have not specified how each line of the algorithm is computed. One method proceeds as follows. At each iteration, we store $\mathcal{Y}_{\text{outer}}$ as a list of halfspaces and $\mathcal{Y}_{\text{inner}}$ as a list of vertices. To test whether $y \in \mathcal{Y}_{\text{outer}}$ we check that y is contained in every halfspace of $\mathcal{Y}_{\text{outer}}$. To test whether $y \notin \mathcal{Y}_{\text{inner}}$ we check that the following LP is feasible:

$$\begin{array}{ll} \text{find} & a, b \\ \text{subject to} & y^T a > b \\ & z^T a \leq b \text{ for every vertex } z \in \mathcal{Y}_{\text{inner}}. \end{array} \tag{10}$$

If so, we pick a and b as any solution to (10). Then, when $\mathcal{Y}$ is a polyhedron, we solve the following LP to pick v (recall that we only have an explicit description of $\mathcal{X}$, not $\mathcal{Y}$):

$$\begin{array}{ll} \text{maximize} & a^T z \\ \text{subject to} & A_1 x + A_2 z \preceq t \\ & B_1 x + B_2 z = u. \end{array} \tag{11}$$

Any optimal solution $x_{\text{opt}}, z_{\text{opt}}$ of (11) satisfies $z_{\text{opt}} \in f_{\mathcal{Y}}(a)$, so we take $v = z_{\text{opt}}$. Also, notice that $S_{\mathcal{Y}}(a) = a^T v$ for any $v \in f_{\mathcal{Y}}(a)$, so having solved (11) we simply take $h_{\mathcal{Y}}(a) = \{ z \in \mathbb{R}^n \mid a^T z \leq a^T v \}$. We update $\mathcal{Y}_{\text{inner}}$ by adding v to the list of vertices, and update $\mathcal{Y}_{\text{outer}}$ by adding $h_{\mathcal{Y}}(a)$ to the list of halfspaces.

E. Refinements for implementation in 2-D

The performance of TEST-SAMPLE depends on the data structure used to store $\mathcal{Y}_{\text{inner}}$ and $\mathcal{Y}_{\text{outer}}$. These sets must be represented in a form that makes it easy to test the membership of query points, or it would not be useful to construct them. The challenge with $\mathcal{Y}_{\text{inner}}$ is to convert its nominal representation as the convex hull of points to a more convenient representation as the intersection of halfspaces (testing membership in the former requires the solution of (10), an LP, while in the latter requires only the evaluation of inequalities). The challenge with $\mathcal{Y}_{\text{outer}}$ is to limit or remove redundant halfspaces (otherwise, testing membership can become slow after many iterations of TEST-SAMPLE). In general, both of these operations are computationally expensive.

In 2-D, however, we can make several refinements to our implementation of TEST-SAMPLE. First, we modify Lines 1-2:

- *It is easy to test $y \notin \mathcal{Y}_{inner}$ and to pick a and b.* In 2-D it is straightforward to convert between vertex and halfspace representations. Assuming that $\mathcal{Y}_{\text{inner}}$ already contains at least three non-collinear vertices, we construct a halfspace representation by sorting these vertices in counterclockwise order and computing the edge between each consecutive pair (using the subroutine HALFSPACE, in Fig. 4). Instead of solving (10), we check if there is a violated inequality in this halfspace representation. If one exists, it separates y from $\mathcal{Y}_{\text{inner}}$, both proving $y \notin \mathcal{Y}_{\text{inner}}$ and defining a valid choice of a and b.
- *It is easy to test $y \in \mathcal{Y}_{outer}$.* The violated inequality defined by a and b corresponds to an edge of $\mathcal{Y}_{\text{inner}}$ containing two consecutive vertices. By construction, each vertex is on the boundary of $\mathcal{Y}$ and is incident to at least one halfspace of $\mathcal{Y}_{\text{outer}}$. Let $\mathcal{H}$ be the intersection of all such halfspaces. Then it is clear that $y \in \mathcal{Y}_{\text{outer}}$ if and only if $y \in \mathcal{H}$ (for example, see Fig. 5). So it is only necessary to test the membership of y in a subset of the halfspaces defining $\mathcal{Y}_{\text{outer}}$.

```
TEST-SAMPLE-2D($y$)
 1  while $i \leq$ length($v$) do
 2      if $\underline{a}_i^T y \leq \underline{b}_i$ then
 3          $i \leftarrow i + 1$
 4      else
 5          if (($\kappa_i$ = TRUE) or ($\overline{a}_i^T y > \overline{b}_i$)
                    or ($\overline{a}_{i+1}^T y > \overline{b}_{i+1}$)) then
 6              return FALSE
 7          else
 8              $v_0 \leftarrow$ SOLVE-LP($\underline{a}_i$)
 9              $(\overline{a}_0, \overline{b}_0) \leftarrow (\underline{a}_i, \underline{a}_i^T v_0)$
10              if $\overline{b}_0 = \underline{b}_i$ then
11                  $\kappa_i \leftarrow$ TRUE
12              else
13                  $(\underline{a}_i, \underline{b}_i) \leftarrow$ HALFSPACE($v_i, v_0$)
14                  $(\underline{a}_0, \underline{b}_0) \leftarrow$ HALFSPACE($v_0, v_{i+1}$)
15                  INSERT($v_0, \underline{a}_0, \underline{b}_0, \overline{a}_0, \overline{b}_0$, FALSE, $i + 1$)
16  return TRUE
SOLVE-LP($a$)
1  solve the following linear program:
       maximize    $a^T z$
       subject to  $A_1 x + A_2 z \preceq t$
                   $B_1 x + B_2 z = u$
2  return the optimal solution $z_0$
HALFSPACE($z_1, z_2$)
1  $a \leftarrow \begin{bmatrix} -e_2^T (z_1 - z_2) \\ e_1^T (z_1 - z_2) \end{bmatrix}$
2  $b \leftarrow a^T z_1$
3  return $(a, b)$
INSERT($v_0, \underline{a}_0, \underline{b}_0, \overline{a}_0, \overline{b}_0, \kappa_0, i$)
1  insert $v_0$, $\underline{a}_0$, $\underline{b}_0$, $\overline{a}_0$, $\overline{b}_0$, and $\kappa_0$ into lists
       $v$, $\underline{a}$, $\underline{b}$, $\overline{a}$, $\overline{b}$, and $\kappa$, respectively, at index $i$
```

Fig. 4. The incremental projection algorithm for 2-D.

Of course, even in 2-D we need to solve the LP (11), in $3N + 2$ dimensions, to pick $v \in f_{\mathcal{Y}}(a)$ in Line 4. But we can modify Lines 3 and 5 to make it easier to update $\mathcal{Y}_{\text{inner}}$ and $\mathcal{Y}_{\text{outer}}$:

- *It is easy to update $\mathcal{Y}_{inner}$.* We would like to maintain our halfspace representation of $\mathcal{Y}_{\text{inner}}$, rather than recompute it at every iteration. Notice that for any $z \in \mathbb{R}^2$, if $z \notin \mathcal{Y}_{\text{inner}}$ and $z \in \mathcal{Y}_{\text{outer}}$ then z violates exactly one inequality in the halfspace representation. In particular, if we pick $v \in f_{\mathcal{Y}}(a)$ (where necessarily $v \in \mathcal{Y}_{\text{outer}}$) such that $v \notin \mathcal{Y}_{\text{inner}}$, then v violates only the inequality defined by a and b. So we update $\mathcal{Y}_{\text{inner}}$ quickly by subdividing the corresponding edge. In fact, any further iteration of TEST-SAMPLE results only in further subdivision of this edge, and so is implemented as a recursion.

- *It is easy to limit the size of $\mathcal{Y}_{outer}$.* At every iteration of TEST-SAMPLE, we generate a halfspace $h_{\mathcal{Y}}(a)$ of $\mathcal{Y}_{\text{outer}}$ that is incident to a vertex $v \in f_{\mathcal{Y}}(a)$ of $\mathcal{Y}_{\text{inner}}$. In general, every time we pick $v \in f_{\mathcal{Y}}(a)$ such that $v \in \mathcal{Y}_{\text{inner}}$, then we add another halfspace incident to the same vertex. We showed in Section IV-C that this can only happen once per query, but over many queries we might generate many halfspaces incident to the same vertex, slowing the test of $y \in \mathcal{Y}_{\text{outer}}$. But because of our choice of a and b, if $v \in \mathcal{Y}_{\text{inner}}$ then $h_{\mathcal{Y}}(a) = \{ z \in \mathbb{R}^2 \mid a^T z \leq b \}$. Hence, we know the edge of $\mathcal{Y}_{\text{inner}}$ defined by a and b is also an edge of $\mathcal{Y}$. By marking this edge as "exact," we can update $\mathcal{Y}_{\text{outer}}$ implicitly, so each vertex in $\mathcal{Y}_{\text{inner}}$ remains associated with only one supporting halfspace in $\mathcal{Y}_{\text{outer}}$.

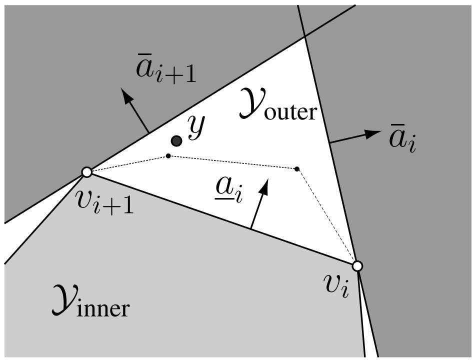
Based on the above refinements, we can use a simple data structure to store $\mathcal{Y}_{\text{inner}}$ and $\mathcal{Y}_{\text{outer}}$. We represent $\mathcal{Y}_{\text{inner}}$ by a list of points v_i ordered counter-clockwise and by a list of halfspaces $\{ z \in \mathbb{R}^2 \mid \underline{a}_i^T z \leq \underline{b}_i \}$, where the halfspace i is incident to the points v_i and v_{i+1}. We represent $\mathcal{Y}_{\text{outer}}$ by a list of halfspaces $\{ z \in \mathbb{R}^2 \mid \bar{a}_i^T z \leq \bar{b}_i \}$, where the halfspace i is incident to the point v_i. We also define the following set of boolean variables to indicate when edges of $\mathcal{Y}_{\text{inner}}$ are exact:

$$\kappa_i = \begin{cases} \text{TRUE} & \text{if } h_{\mathcal{Y}}(\underline{a}_i) = \{ z \in \mathbb{R}^2 \mid \underline{a}_i^T z \leq \underline{b}_i \}, \\ \text{FALSE} & \text{otherwise.} \end{cases}$$

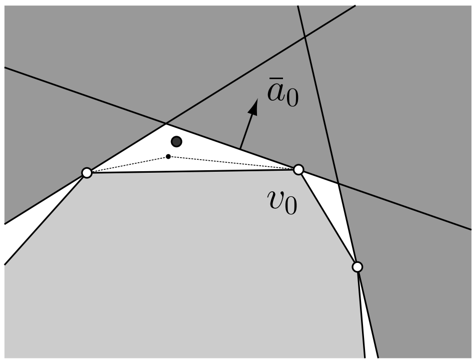
Our implementation is described by the algorithm TEST-SAMPLE-2D (Fig. 4). A single iteration is shown in Fig. 5.

V. EXPERIMENTAL RESULTS

Our incremental projection algorithm reduces the time required to test static equilibrium at sampled configurations for a fixed stance, because it takes advantage of the distribution of previously tested configurations to guide the construction of inner and outer approximations to the support polygon. To see this, we first compare incremental projection to the methods of linear programming and linear projection for the example in Fig. 1 (Section III-C). Fig. 6 shows the time taken by each method to test a variable number of randomly distributed samples (averaged over many runs). Linear programming is fast for small numbers of samples but slow for large numbers; linear projection is the opposite. Incremental projection performs better than either approach for reasonable numbers of samples. In particular, it performs much better for 70 samples, seen in practice. Of course, if the sample distribution requires it, incremental projection may eventually have to compute the entire support polygon—then, its performance would be the same as linear projection. This would occur, for example, if samples are distributed at every vertex of the support polygon, or if samples are distributed uniformly. Although it is intuitive



(a)



(b)

Fig. 5. One iteration of incremental projection in 2-D.

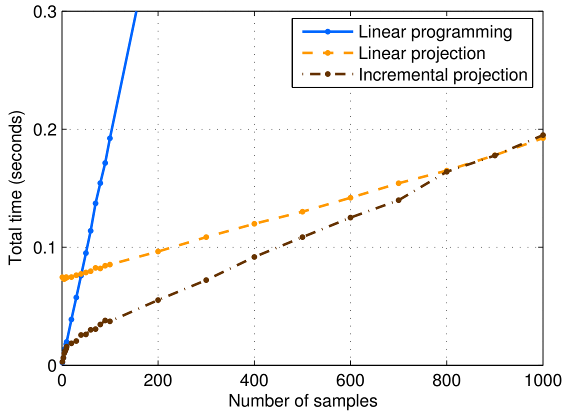


Fig. 6. The total amount of time required to test a variable number of randomly distributed samples.

to compare these algorithms with respect to the number of samples, it is really the sample distribution that is important.

Incremental projection also allows a greater amount of complexity in the projection $\mathcal{Y}$ (likewise, in the polyhedron $\mathcal{X}$) at little additional computational cost. This capability might be used to model other more complicated types of contact with additional constraints, or to model frictional point contact with greater accuracy (for example, increasing the fidelity of the polyhedral friction cone approximation). Fig. 7 shows the total time required to test 100 randomly distributed samples in projections of increasing complexity. Incremental projection

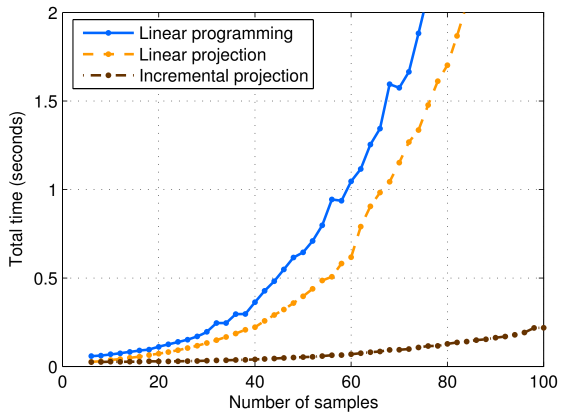


Fig. 7. The total amount of time required to test 100 randomly distributed samples as the complexity of the projection $\mathcal{Y}$ increases.

TABLE I

TIME (SECONDS) TO TEST STATIC EQUILIBRIUM WHILE PLANNING

	Size of problem		
Method	1-step	30-step	2000-step
Linear programming	0.093	9.48	287
Linear projection	0.061	2.94	167
Incremental projection	0.040	1.82	64

remains fast even as complexity increases.

Finally, incremental projection makes multi-step planning faster. Table I shows results for three example planning problems. The first is the single step considered in Figs. 1 and 6. Incremental projection is about 33% faster than linear projection, and more than 50% faster than linear programming. The second is a multi-step search in a terrain with 7 potential foot placements, requiring the exploration of 30 steps. Again, the incremental projection algorithm is about 33% faster than linear projection (saving about a second of computation time), and is much faster than linear programming. The third is a multi-step search in terrain with 100 potential foot placements, requiring the exploration of more than 2000 steps. Here, incremental projection is over 50% faster than linear projection (saving almost two minutes). The advantage of using this algorithm grows with the complexity of the planning problem because most steps are explored only briefly (that is, with a relatively small number of samples). In each case, the amount of time saved represents about 5-10% of total planning time.

VI. Conclusion

In this paper we presented a new algorithm to test static equilibrium, in particular to test the membership of points in the projection of a polyhedron. Incremental projection is often faster than both linear programming and linear projection because it uses information (acquired after testing multiple sample points) that both previous approaches discard. For example, if a sample point is outside the projection, most LP algorithms generate a separating hyperplane as proof. Incremental projection uses such hyperplanes to define pieces of an outer approximation that might allow other infeasible sample points to be eliminated without further computation. Likewise, linear projection disregards future information about the distribution of sample points. Rather than compute the entire projection, our incremental algorithm computes only enough to distinguish between feasible and infeasible samples.

Acknowledgments

This work was partially funded by NSF grant IIS-0412884.

References

[1] R. B. McGhee and G. I. Iswandhi, “Adaptive locomotion of a multi-legged robot over rough terrain,” *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-9, no. 4, pp. 176–182, 1979.
[2] D. Wettergreen, “Robotic walking in natural terrain,” Ph.D. dissertation, Carnegie Mellon University, 1995.
[3] S.-M. Song and K. J. Waldron, *Machines that walk: the adaptive suspension vehicle*. Cambridge, MA: The MIT Press, 1989.
[4] J. Chestnutt, J. Kuffner, K. Nishiwaki, and S. Kagami, “Planning biped navigation strategies in complex environments,” in *IEEE Int. Conf. Hum. Rob.*, Munich, Germany, 2003.
[5] J. J. Kuffner, Jr., K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, “Motion planning for humanoid robots,” in *Int. Symp. Rob. Res.*, Siena, Italy, 2003.
[6] K. Harada, S. Kajita, H. Saito, F. Kanehiro, and H. Hirukawa, “Integration of manipulation and locomotion by a humanoid robot,” in *Int. Symp. Exp. Rob.*, Singapore, 2004.
[7] T. Bretl, “Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem,” to appear in *Int. J. Rob. Res.*, 2006.
[8] M. J. Todd, “The many facets of linear programming,” *Mathematical Programming*, vol. 91, no. 3, pp. 417–436, 2002.
[9] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
[10] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, ser. Monographs in Theoretical Computer Science. New York, NY: Springer-Verlag, 1987, vol. 10.
[11] G. M. Ziegler, *Lectures on Polytopes*, ser. Graduate texts in mathematics. Springer-Verlag, 1995, vol. 152, revised first edition 1998.
[12] L. E. Kavraki, P. Svetska, J.-C. Latombe, and M. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, 1996.
[13] H. Choset, K. Lynch, S. Hutchinson, G. Kanto, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.
[14] M. Buss, H. Hashimoto, and J. B. Moore, “Dextrous hand grasping force optimization,” *IEEE Trans. Robot. Automat.*, vol. 12, no. 3, pp. 406–418, 1996.
[15] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, “Applications of second-order cone programming,” *Linear Algebra and its Applications*, vol. 284, pp. 193–228, 1998, special Issue on Linear Algebra in Control, Signals, and Image Processing.
[16] L. Han, J. C. Trinkle, and Z. X. Li, “Grasp analysis as linear matrix inequality problems,” *IEEE Trans. Robot. Automat.*, vol. 16, no. 6, pp. 663–674, 2000.
[17] G. B. Dantzig and B. C. Eaves, “Fourier-Motzkin elimination and its dual,” *J. Combinatorial Theory*, vol. 14, pp. 288–297, 1973.
[18] K. Fukuda and A. Prodon, “Double description method revisited,” *LNCS*, vol. 1120, pp. 91–111, 1996.
[19] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, “The quickhull algorithm for convex hulls,” *ACM Transactions on Mathematical Software*, vol. 22, no. 4, pp. 469–483, 1996.
[20] D. Bremner, K. Fukuda, and A. Marzetta, “Primal-dual methods for vertex and facet enumeration,” in *ACM Sympos. Comput. Geom.*, Nice, France, 1997, pp. 49–56.
[21] J. Ponce, S. Sullivan, A. Sudsang, J.-D. Boissonnat, and J.-P. Merlet, “On computing four-finger equilibrium and force-closure grasps of polyhedral objects,” *Int. J. Rob. Res.*, vol. 16, no. 1, pp. 11–35, 1997.
[22] M. Joswig, “Beneath-and-beyond revisited,” in *Algebra, Geometry, and Software Systems*, M. Joswig and N. Takayama, Eds. Springer, 2003, pp. 1–21.