16-741 Mechanics of Manipulation
Fall term, 2018

Assignment 6, Out: 16 November 2018 Due: 30 November 2018

Introduction

In this assignment, you will look into the computational aspect of force closure, while we focused on intuitive graphical methods in class. The goal is to implement a program that can figure out whether a certain grasp of an object can provide force closure or not, and then to use this program to plan force closure grasps.

Recall the definition of force closure and the related theorem in the text:

DEFINITION 5.4: Force closure means that the set of possible wrenches exhausts all of wrench space.
THEOREM 5.5: A set of vectors $\{v_i\}$ positively spans the entire space $\mathbb{R}^n$ if and only if the origin is in the interior of the convex hull:

$$\text{pos}(\{v_i\}) = \mathbb{R}^n \iff \mathbf{0} \in \text{int}(\text{conv}(\{v_i\})) \tag{1}$$

One important issue in force closure algorithms is how to efficiently check if the origin of the wrench space is inside the convex hull. In this assignment, this problem will be formulated as a ray-shooting problem which can be solved by linear programming. The approach was developed in [1], and refined and simplified in [2] and [3].

Guidelines

Here are some guidelines for you.

- You will be following the process described in [1], [2], and [3]. If you don't have access to IEEE Xplore, please contact your TA.
- You can write code in MATLAB or GNU Octave utilizing the functions provided in the handout. If you wish to use other languages such as Python or C++, you should write your own version of code for the provided functions.
- You can use external toolboxes/libraries such as Optimization Toolbox as long as they are not used to answer to a force closure test directly.
- Empty MATLAB functions in the handout are just to give you hints for your implementation. You don't have to follow their format, provided that you can answer to each questions and show the results in the writeup.
- Your submission will be graded based on the numerical results of the test cases in the handout. Section 1, 2, and 3 describe what functions you should implement, and section 4 explains how you can run test cases and what results you should report. In section 5, you will use your functions to find a force closure grasp for a given object.

1 Contact Screw in Wrench Space

For given contact normals and corresponding contact points, compute normalized contact screws, that is, normalized Plücker coordinates of contact normals in wrench space.

```
[W] = contactScrew(CP,CN)
```

- `CP`: a set of contact point positions $\{(p_{i,x}\ p_{i,y}\ p_{i,z})^T\}$; $3 \times N$ matrix
- `CN`: a set of inward-pointing directions of contact normals $\{(n_{i,x}\ n_{i,y}\ n_{i,z})^T\}$; $3 \times N$ matrix
- `W`: a set of normalized contact screws $\{(c_{i,x}\ c_{i,y}\ c_{i,z}\ c_{0i,x}\ c_{0i,y}\ c_{0i,z})^T\}$ such that $\|(c_{i,x}\ c_{i,y}\ c_{i,z})^T\| = 1$; $6 \times N$ matrix

where N is the number of contact points. Note that a contact normal is an inward-pointing vector at its contact point and the contact is yet assumed to be frictionless unilateral point contact.

2 Force Closure Test by Ray-shooting Algorithm

For given contact screws, check whether this grasp is force closure or not.

```
[bFC, zmax] = isForceClosure(W)
```

- `W`: a set of normalized contact screws $\{(c_{i,x}\ c_{i,y}\ c_{i,z}\ c_{0i,x}\ c_{0i,y}\ c_{0i,z})^T\}$; $6 \times N$ matrix
- `bFC`: a flag which is true if the grasp is force closure; boolean
- `zmax`: the value of the maximizing objective function at the optimal point; scalar

In this function, you should check if `W` satisfies the condition of force closure as described in (1). You can formulate this problem as a ray-shooting problem [1], [2], [3]. The basic idea is quite simple as shown in Figure 1, and its strong point is that it can be solved by linear programming.

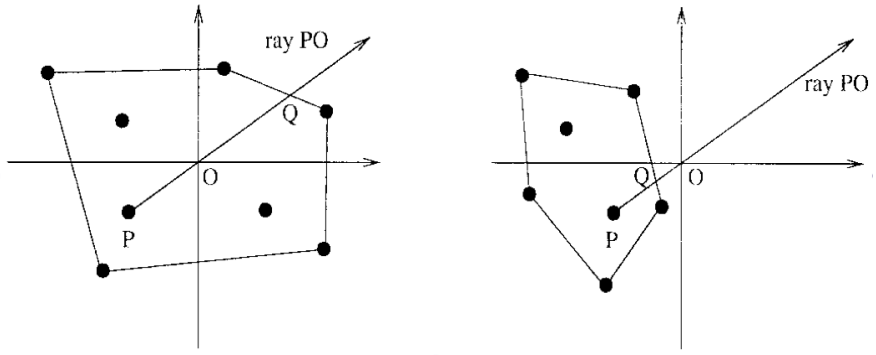


Figure 1: A ray-shooting algorithm can be used to check whether the origin is inside the convex hull [1]. From an internal point P, such as a centroid of vertices, shoot a ray toward the origin, and compare the distances to the origin O and the intersection point with the convex hull boundary Q. If $\overline{PO}$ is shorter than $\overline{PQ}$, the origin is within the convex hull, and thus, the grasp is force closure.

A linear programming problem can be written in canonical form as follows:

$$\begin{aligned} &\text{maximize } && c^T x \\ &\text{subject to } && Ax \leq b \\ & && x \geq 0 \end{aligned} \tag{2}$$

where x is the variable vector, c and b are given coefficient vectors, and A are a given coefficient matrix. $c^T x$ is called the objective function, and the inequalities $Ax \leq b$ and $x \geq 0$ are called constraints. Note that all the objective function and constraints are in linear form.

A MATLAB function `linprog()` in Optimization Toolbox which you might want to use is in a slightly different form:

$$\begin{aligned} &\text{minimize } && f^T x \\ &\text{subject to } && Ax \leq b \\ & && A_{eq} x = b_{eq} \\ & && x_{lb} \leq x \leq x_{ub} \end{aligned} \tag{3}$$

However, you can notice that some linear transformation tricks can convert one form to the other. For example, let $f = -c$ to change a maximization problem to a minimization problem.

The objective function and constraints for a force closure test are introduced in [1] based on dual transformation of a convex hull to a convex polytope (see (11) and (12) of [1]). Note that the dual transformation is slightly different from the one we have been using in class due to a sign change. A simplified version of distance comparison between $\overline{PO}$ and $\overline{PQ}$ is proposed in [2] (see Theorem 2). From their results, a force closure test can be done by a few lines of code with an optimization library.

However, there is a bug in formalizing a force closure test as a linear programming problem. If the convex hull does not have any 6-dimensional volume (think of a flat ball in 3D space), there would be no internal points. Therefore, the point P will lie on the boundary of the convex hull, and the matrix A in (3) will lose its rank. This means that the number of constraints gets smaller than the number of variables. Even in this case, a linear programming solver can find a solution if the origin is on the *flat* convex hull, which may lead to a *false positive*. Otherwise, it should end up with an infinite value of the objective function. This issue is addressed in [3] in detail, and you should look at Fig. 5 and Proposition 4.

3 Friction Cone Approximation

For given contact points, contact normals, and a friction coefficient, generate contact points and contact normals that correspond to `M` edges of a linearized polyhedral friction cone.

```
[CPF, CNF] = frictionCone(CP, CN, mu, M)
```

- `CP`: a set of contact point positions $\{(p_{i,x}\ p_{i,y}\ p_{i,z})^T\}$; $3 \times N$ matrix
- `CN`: a set of inward-pointing directions of contact normals $\{(n_{i,x}\ n_{i,y}\ n_{i,z})^T\}$; $3 \times N$ matrix
- `mu`: a coefficient of (static) friction; scalar
- `M`: the number of side facets of a linearized polyhedral friction cone; scalar
- `CPF`: a set of contact point positions of edges of polyhedral friction cones $\{(p_{ij,x}\ p_{ij,y}\ p_{ij,z})^T\}$; $3 \times (NM)$ matrix
- `CNF`: a set of inward-pointing directions of edges of polyhedral friction cones $\{(s_{ij,x}\ s_{ij,y}\ s_{ij,z})^T\}$; $3 \times (NM)$ matrix

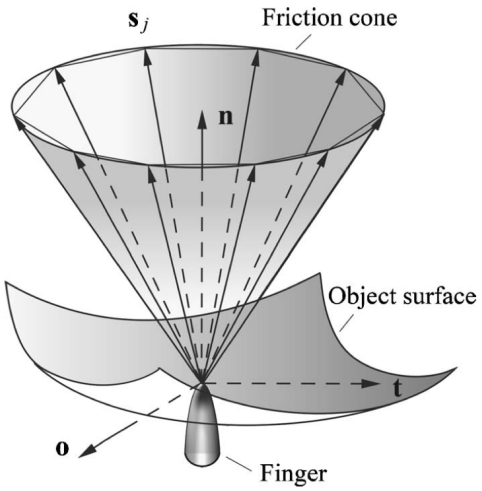


Figure 2: Linearization of a friction cone as a polyhedral convex cone [3]. Any contact forces within a linearized friction cone are in the positive linear span of the edges of the polyhedral convex cone, i.e., $f \in \text{pos}(\{s_j\})$.

Thus far, we have assumed frictionless point contact condition. However, the functions `contactScrew()` and `isForceClosure()` can also be used for frictional point contact by approximating a friction cone as a polyhedral convex cone (Figure 2). Since each edge of the polyhedral convex cone can take part in making a wrench, it can be regarded as an independent contact normal in our framework, and this is what this function is supposed to do.

If you want to use (7) and (8) in [2] to get s_j, a function in the following format might come in handy.

```
[R] = computeRotMat(n)
```

- `n`: inward-pointing direction of a contact normal $\{(n_x\ n_y\ n_z)^T\}$; 3×1 vector
- `R`: a rotation matrix with positive x-axis aligned with `n`; 3×3 matrix

where `R` represents the relative orientation of the local coordinate frame at a contact point with respect to the object frame. Note that [2] uses a convention that `n` is applied along the positive direction of x-axis of `R` which is the first column vector of a rotation matrix. Also note that there is one redundant degree of freedom in the determination of `R` as only one direction vector is specified.

4 Evaluation: Test Cases [60 pts]

For quantitative evaluation of your implementation, two test cases are provided in a function `part1(TEST)` where `TEST` is an index for test case selection. This function calls your implemented functions to check force closure for frictionless and frictional point contact conditions, respectively. It also calls a provided function `drawContactScrew(CP, W, M)` to visualize your computed contact screws.

You should run the following commands and report the output and the figures in the writeup. More specifically, we are interested in your `bFC` and `zmax` for both of frictionless and frictional contact conditions. An example presented in [1] and [2] can also be tested with a command `part1(0)` to debug your code.

```
>> part1(1)
>> part1(2)
```

The following are the given grasp and friction parameters for each test case.

- TEST = 1

$$\texttt{CP} = \begin{bmatrix} \texttt{p1} & \texttt{p2} & \texttt{p3} \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 1 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$\texttt{CN} = \begin{bmatrix} \texttt{n1} & \texttt{n2} & \texttt{n3} \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ -1 & -1 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\texttt{mu} = 0.5$$

$$\texttt{M} = 100$$

- TEST = 2

$$\texttt{CP} = \begin{bmatrix} \texttt{p1} & \texttt{p2} & \texttt{p3} & \texttt{p4} & \texttt{p5} \end{bmatrix} = \begin{bmatrix} -0.81 & -1.75 & 1.49 & 0.49 & 1.23 \\ 1.26 & -1.37 & 0.85 & -2.57 & -1.04 \\ 1.23 & 0.74 & -1.39 & -0.39 & 1.20 \end{bmatrix}$$

$$\texttt{CN} = \begin{bmatrix} \texttt{n1} & \texttt{n2} & \texttt{n3} & \texttt{n4} & \texttt{n5} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$$\texttt{mu} = 0.2$$

$$\texttt{M} = 100$$

5 Application: Grasp a Soccer Ball [40 pts]

In this problem, you are asked to find a force closure grasp for a given object which an icosahedron as shown in Figure 3(a). You should use your implemented functions to check force closure of a grasp that you generated or selected.

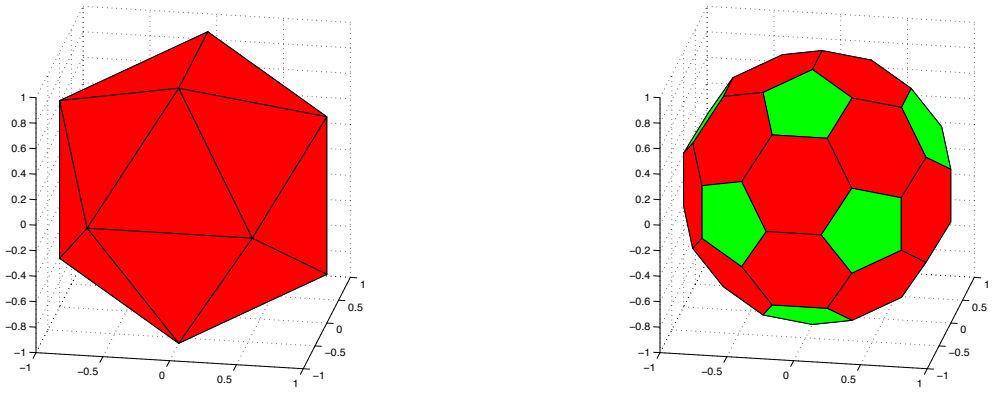
The object's geometric properties and some utility functions are implemented as a MATLAB class called `SoccerBall`. Once you create an instance of `SoccerBall`, you can draw an icosahedron by `drawBall()` and get a contact normal at an interpolated point on a facet by `getContactNormal(iv, ratio)`. See the provided MATLAB code for more detail.

You would implement an algorithm to find a force closure grasp in a function `part2(FRIC)` where `FRIC` is a flag which is false (or 0) for frictionless contact and true (or 1) for frictional contact. For frictional contact, set the parameters as `m` = 0.3 and `M` = 10.

```
>> part2(false)
>> part2(true)
```

A simplistic way to implement this would be a random search that repeats generating a random N contacts and checking for force closure until a solution is found. Note that we assume no contact is at any edges or corners, i.e., there is no undetermined contact normals.

From your implementation, you should be able to answer in the writeup to the following questions for frictionless and frictional point contact conditions, respectively:



(a) Icosahedron **(b)** Truncated icosahedron

Figure 3: A regular icosahedron is a polyhedron with 20 equilateral triangle faces. If each edge is truncated by one third of its length at all 12 vertices, the typical shape of a soccer ball (a.k.a. Telstar) can be constructed.

- N: How many fingers *at least* do you need to achieve force closure?
- `CP`: Where should the fingers be located?
- `zmax`: How far is the origin from the convex hull centroid of the normalized contact screws?
- Also include a figure of contact normals applied to the icosahedron in 3D space.

Closing Remarks

From this assignment, you have implemented algorithms that can check force closure for a *given grasp* and (naively) find a force closure grasp for a *given object*. There are two more interesting questions to think of from this end.

You might have noticed that there can be many possible force closure grasps for a *given object*. If you have to pick only one grasp among them, how would you do this? This question gives rise to the need for a quantitative metric which is called *grasp quality*. This is usually based on analysis of contact screws in wrench space. You might find [4] useful as a starting point.

Once you are *given a force closure grasp*, you are able to generate any wrenches for the object, but how much force should be applied at each contact point? Note that this problem is inderterminate if the number of contacts is larger than the number of constraints which is usually six in 3D space. Then we can formulate it as an optimization problem where minimal grasping forces are typically encouraged. This is called a *grasping force optimization* problem. If you are interested in this, see [5] for more detail.

What to submit

Please archive everything into a zip file named **andrewid_ps6.zip**, and send it to `matt.mason@cs.cmu.edu` and `jonathanking@cmu.edu`. Below are what you should submit:

- a folder named **code** containing all the .m files you were asked to write
  (if you used Python, C++, or else, please provide all the required files to run your code)
- a pdf named **andrewid_ps6.pdf** containing the results and explanations

References

[1] Yun-Hui Liu, "Qualitative Test and Force Optimization of 3-D Frictional Form-Closure Grasps Using Linear Programming", *IEEE Transactions on Robotics and Automation*, vol. 15, no. 1, pp. 163–173, 1999. http://ieeexplore.ieee.org/arnumber=744611

[2] Yu Zheng and Wen-Han Qian, "Simplification of the Ray-Shooting Based Algorithm for 3-D Force-Closure Test", *IEEE Transactions on Robotics and Automation*, vol. 21, no. 3, pp. 470–473, 2005. http://ieeexplore.ieee.org/arnumber=1435492

[3] Yu Zheng and Wen-Han Qian, "An Enhanced Ray-shooting Approach to Force-Closure Problems", *Journal of Manufacturing Science and Engineering*, vol 128, no. 4, pp. 960–968, 2006. http://guppy.mpe.nus.edu.sg/~legged_group/nusbip/members/yuzheng/An%20Enhanced%20Ray-Shooting%20Approach%20to%20Force-Closure%20Problems.pdf

[4] Zexiang Li and Shankar Sastry, "Task-Oriented Optimal Grasping by Multifingered Robot Hands" *IEEE Journal of Robotics and Automation*, vol. 4, no. 1, pp. 32–44, 1988. http://ieeexplore.ieee.org/arnumber=769

[5] Martin Buss, Hideki Hashimoto, and John B Moore, "Dextrous hand grasping force optimization", *IEEE Transactions on Robotics and Automation*, vol. 12, no. 3, pp. 406–418, 1996. http://users.rsise.anu.edu.au/~john/papers/JOUR/144.PDF