On Random Sampling in Contact Configuration Space*

Xuerong Ji and Jing Xiao
Computer Science Department
University of North Carolina - Charlotte
Charlotte, NC 28223, USA
xji@uncc.edu and xiao@uncc.edu

Abstract

Random sampling strategies play critical roles in randomized motion planners, which are promising and practical for motion planning problems with many degrees of freedom (dofs). In this paper, we address random sampling in a constrained configuration space – the contact configuration space between two polyhedra, motivated by the need for generating contact motion plans. Given a contact formation (CF) between two polyhedra A and B, our approach is to randomly generate configurations of A satisfying the contact constraints of the CF. Key to the approach is to guarantee that sampling happens only in the constrained space to be efficient, which has not been addressed in the literature. We first describe our random sampling strategy for configurations constrained by CFs consisting of one or two principal contacts (PCs) and then present implementation results.

1 Introduction

Contact motions are important in automatic assembly processes, not only because they happen frequently when clearance between objects is tight, but also because they reduce degrees of freedom (dofs), thus reduce uncertainties [14, 16]. Contact motion occurs on the boundary of configuration space obstacles (C-obstacles) [13], but computing C-obstacles remains a formidable task to date. While there were exact descriptions of C-obstacles for polygons [2, 4], there were only approximations for polyhedra [5, 11]. Contact motions, however, require exactness of contact configurations. Hence, some researchers started exploring methodologies on contact motion planning without explicitly computing C-obstacles (see for example, [7]).

Recently the authors introduced a general divide-and-merge approach for automatically generating a contact state graph between arbitrary polyhedra [9, 20]. Each node in the graph denotes a contact state, indicating by a *contact formation* (CF) [17] and a representative configuration of the CF, and each edge denotes the neighboring relationship between the two nodes connected by the edge. With this approach, the problem of contact motion planning is effectively simplified as graph search at high-level for state transitions and motion planning at low-level within the set of contact configurations constrained by the *same* contact state[1]. However, even for such reduced-dimension and reduced-scope motion planning, the dimensionality or dofs can still be quite high for less-constrained contact states, such as those consisting of only a single *principal contact* (PC) [17] (see Fig. 1 for PCs between two polyhedra). Thus, randomized motion planning is desirable for planning contact motions, which requires random sampling of *contact* configurations.

There are promising randomized planners for collision-free motions, such as those based on probabilistic roadmaps (PRM) [12, 15], to which random sampling is to simply generate arbitrary configurations of the considered object/robot. The sampled configurations may or may not be collision-free. To make sampling more efficient, several researchers introduced

*This research has been supported by the National Science Foundation under grants IIS-9700412 and CDC-9726424.

[1] Note that a general contact motion crossing several contact states consists of segments of motion in each contact state.

different methods targeting at producing more samples in certain critical areas that tend to be close to C-obstacles and sparse samples in other areas[1, 3, 8]. However, sampling *exactly* on the C-obstacle surface, or in the *contact* configuration space, has not been addressed in the literature.

In this paper, we extends the random sampling strategy for a single PC reported in [10] to contact formations (CFs) of two PCs. Our approach takes advantage of our work on automatic generation of contact state graphs by building sampling on the knowledge of a contact formation and a representative contact configuration under the contact formation (obtainable from such a graph). Particularly, given a CF and a *seed* configuration satisfying the CF, the goal is to randomly sample configurations satisfying the CF.

The paper is outlined as follows. In Section 2, we review the notion of CFs and analyze the dofs for each kind of single-PC or two-PC CF between two arbitrary polyhedra. In Section 3, we present the random sampling algorithms. In Section 4, we provide some experimental results of the sampling strategy. Section 5 concludes the paper.

2 Contact Formations and Degrees of Freedom (dofs)

In this section, we first introduce notations related to contact formations used throughout the paper and then analyze the dofs for different types of contact formations. Knowledge of the dofs is needed for our random sampling strategy.

2.1 Notations

Consider two arbitrary polyhedra A and B. Assume that A is moveable and B is static. The faces, edges, and vertices of each object are the object's topological *elements*. The *boundary elements* of a face are its edges and vertices, and the boundary elements of an edge are its vertices.

As defined in [17], a *principal contact* (PC) between A and B describes a single contact between a pair of contacting elements which are *not* the boundary elements of other contacting elements. There are 10 types of PCs (Fig. 1): face-face (f-f), face-edge(f-e)/edge-face(e-f), face-vertex (f-v)/vertex-face(v-f), edge-edge-cross (e-e-c), edge-edge-touch (e-e-t), edge-vertex (e-v)/vertex-edge (v-e), and vertex-vertex (v-v), of which e-e-t, e-v/v-e and v-v PCs between convex elements are *degenerate* PCs. We denote a PC as u_A-u_B, where u_A and u_B denote the contacting elements of A and B respectively. With the notion of PCs, an arbitrary contact between A and B can be characterized by the set of PCs formed, called a *contact formation* (CF), denoted as $\{PC_1,...,PC_n\}$. As degenerate PCs rarely happen in practice, in this paper, we only consider CFs formed by *non-degenerate* PCs, referred to simply as PCs.

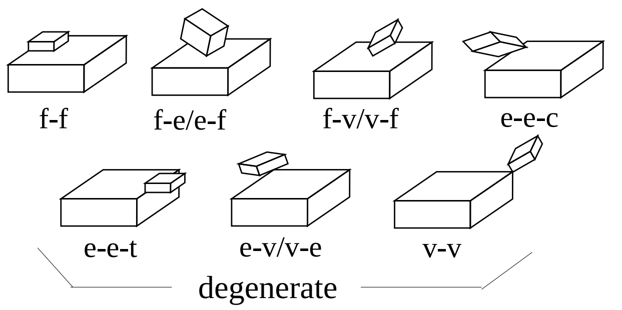


Figure 1: *Principal Contacts (PCs)*

The two contacting elements of a non-degenerate PC uniquely determine a plane, which we call a *contact plane* (CP). Based on the region of contact on the contact plane, we can further classify the PCs into the following three types:

- **plane PC**: f-f, where the contacting elements intersect at a planar region on the contact plane,
- **line PC**: e-f and f-e, where the contacting elements intersect at a line, called a *contact line*.
- **point PC**: v-f, f-v, and e-e-c, where the contacting elements intersect at a point, called a *contact point*.

A *contact configuration* is defined as the configuration of A relative to that of B when A and B are in contact. Thus, the geometric interpretation of a PC is the region of contact configurations (on the contact plane) where the PC holds, and that of a CF is the intersection of the regions of contact configurations of the participating PCs.

2.2 Degrees of Freedom

The dofs of a PC is expressed by the constraints it imposes on contact configurations. To represent such contact constraints, for an arbitrary polyhedron P, we attach coordinate system (or frame) to it. Moreover, for each element (vertex, edge or face) of the object P, we attach a coordinate system as follows:

- vertex: the coordinate system v has its origin at the vertex, and the orientation is the same as that of P.
- edge: the coordinate system e has its origin at one of its bounding vertices, the direction of $+X$ is along the edge pointing to the other bounding vertex, the direction of $+Z$ is defined as the outward normal of the edge [2], and the direction of $+Y$ is determined by the right-hand rule.
- face: the coordinate system f has its origin at one of its bounding vertices, the direction of $+Z$ is defined as the outward normal of the face, the direction of $+X$ is along one of the bounding edges of the face, and the direction of $+Y$ is determined by the right-hand rule.

Fig. 2 illustrates the coordinate systems of object P and some of its elements.

We can present the contact constraint equations for each type of PC between A and B in terms of expressions for contact configurations ${}^BT_A$ (homogeneous transformation matrix) (extending [19]), where for A, B and their elements, we attach coordinate systems by the above definitions. In each expression, the underlined symbols are independent variables, of which the Greek symbols are rotational variables in Euler angles.

- *v-f*:
  ${}^BT_A = {}^BT_{f^B} \cdot T_{trans}(\underline{x}, \underline{y}, 0) \cdot T_{rotzyx}(\underline{\alpha}, \underline{\beta}, \underline{\gamma}) \cdot {}^AT_{v^A}^{-1}$
- *f-v*:
  ${}^BT_A = {}^BT_{v^B} \cdot T_{rotzyx}(\underline{\alpha}, \underline{\beta}, \underline{\gamma}) \cdot T_{trans}(\underline{x}, \underline{y}, 0) \cdot {}^AT_{f^A}^{-1}$

[2]The outward normal of an edge (or a vertex) is defined as the sum of the outward normals of the faces forming the edge (or vertex).

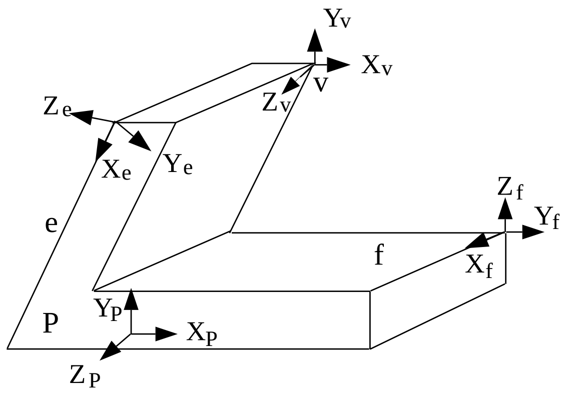


Figure 2: *Coordinate systems for an object P and for its vertex v, edge e and face f*

- *e-e-c*:
  ${}^BT_A = {}^BT_{e^B} \cdot T_{trans}(\underline{x_1}, 0, 0) \cdot T_{rotzyx}(\underline{\alpha}, \underline{\beta}, \underline{\gamma}) \cdot T_{trans}(\underline{x_2}, 0, 0) \cdot {}^AT_{e^A}^{-1}$
- *e-f*:
  ${}^BT_A = {}^BT_{f^B} \cdot T_{trans}(\underline{x}, \underline{y}, 0) \cdot T_{rotz}(\underline{\alpha}) \cdot T_{rotx}(\underline{\gamma}) \cdot {}^AT_{e^A}^{-1}$
- *f-e*:
  ${}^BT_A = {}^BT_{e^B} \cdot T_{rotx}(\underline{\gamma}) \cdot T_{rotz}(\underline{\alpha}) \cdot T_{trans}(\underline{x}, \underline{y}, 0) \cdot {}^AT_{f^A}^{-1}$
- *f-f*:
  ${}^BT_A = {}^BT_{f^B} \cdot T_{trans}(\underline{x}, \underline{y}, 0) \cdot T_{rotx}(\pi) \cdot T_{rotz}(\underline{\alpha}) \cdot {}^AT_{f^A}^{-1}$

where ${}^AT_{v^A}$, ${}^AT_{e^A}$ and ${}^AT_{f^A}$ (or ${}^BT_{v^B}$, ${}^BT_{e^B}$ and ${}^BT_{f^B}$) represent the transformation matrices from the frame of a vertex, edge and face of A (or B) to the frame of A (or B) respectively. $T_{trans}(*, *, *)$ is the 4×4 translational matrix. $T_{rotz}(*)$, $T_{roty}(*)$, and $T_{rotx}(*)$ are the 4×4 rotational matrix about z, y and x axis respectively, and $T_{rotzyx}(*, *, *)$ is the combination of the three rotational matrices.

The dofs for each type of PC equals to its number of independent variables. Using 't' to indicate translational dofs and 'r' to indicate rotational dofs, we summarize the dofs for single-PC CFs below:

- **plane PC**: **dofs** = 3, (2t and 1r)
- **line PC**: **dofs** = 4, (2t and 2r)
- **point PC**: **dofs** = 5, (2t and 3r)

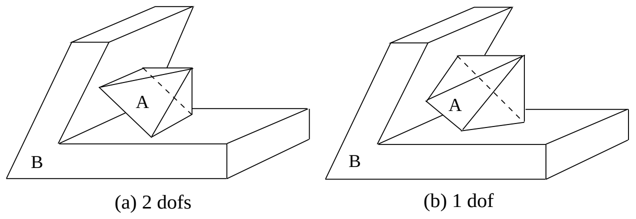


Figure 3: *CFs with two line PCs: (a) the contact lines are parallel (2 dofs), (b) the contact lines are not parallel (1 dof)*

Since a free-flying polyhedron has 6 dofs, it is easy to see that a point PC reduces the dof of the object by 1, a line PC by 2, and a plane PC by 3.

The constraint equations for a two-PC CF are the set of equations for the PCs involved. The dofs of a two-PC CF depends not only on the topological types of the PCs but also on the geometrical relation between the PCs and their corresponding contact regions. We summarize them below:

CFs where the two contact planes are not parallel:

- **two plane PCs {f-f, f-f}**:
  dofs = 1, (1t)
- **line PC and plane PC {e-f/f-e, f-f}**[3]:
  dofs = 1, (1t)
- **point PC and plane PC {v-f/f-v/e-e-c, f-f}**:
  dofs = 2, (1t and 1r)
- **two line PCs {e-f/f-e, e-f/f-e}** (see Fig. 3):

$$\textbf{dofs} = \begin{cases} 2,\ \text{(1t and 1r)} \\ \quad \text{if the two contact lines are parallel} \\ 1,\ \text{(1t)} \\ \quad \text{otherwise} \end{cases}$$

- **point PC and line PC {v-f/f-v/e-e-c, e-f/f-e}**:
  dofs = 3, (1t and 2r)
- **two point PCs {v-f/f-v/e-e-c, v-f/f-v/e-e-c}**:
  dofs = 4, (1t and 3r)

[3]The notion "e-f/f-e" means either e-f or f-e. The same explanation applies to all "/" used in the paper.

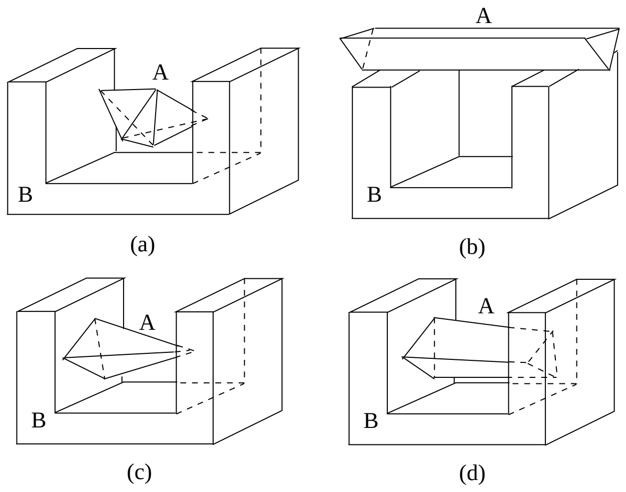


Figure 4: *CFs with parallel contact planes: (a) two point PCs (4 dofs), (b) two line PCs with collinear contact lines (4 dofs), (c) one point and one plane PCs (3 dofs), (d) two plane PCs (3 dofs)*

For CFs where the contact planes are parallel (see Fig. 4):

$$\textbf{dofs} = \begin{cases} 4,\ \text{(2t and 2r)} \\ \quad \text{if two point PCs, else} \\ 4,\ \text{(2t and 2r)} \\ \quad \text{if contact point(s)/line(s) are collinear} \\ 3,\ \text{(2t and 1r)} \\ \quad \text{otherwise} \end{cases}$$

3 Random Sampling Strategy

As mentioned in Section 1, the authors introduced a general divide-and-merge approach [9, 20] to generate contact state graphs automatically. This approach reduces the contact motion planning problem to a graph search problem at the high level and the problem of planning of contact motions within the *same* contact formation at the low level, which we call *CF-compliant* motion planning. Our random sampling strategy aims at CF-compliant motion planning and takes advantage of the known information provided by the divide-and-merge approach: a CF and a (seed) contact configuration satisfying the CF.

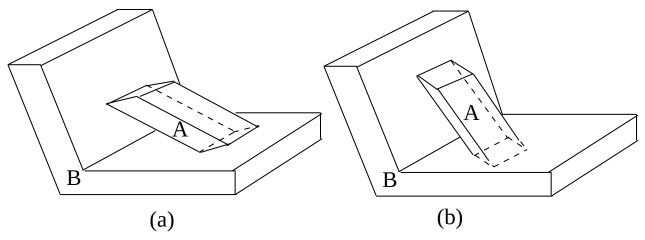


Figure 5: *Two configurations of the same {e-f, e-f} CF: (a) valid with no local penetration, (b) invalid with local penetration*

For each CF, our strategy generates contact configurations which satisfy the CF and are guaranteed no *local penetration*, that is, *no penetration between the two objects through elements u_A or u_B of each PC or through an element directly connected to u_A (or u_B) and u_B (or u_A).* We call such configurations *valid* contact configurations for the CF. Fig. 5 shows two configurations with the same {e-f, e-f} CF, where (a) is a valid configuration, and (b) is invalid because it has a local penetration between an adjacent face of the edge and the face of B in the bottom PC.

We use two general methods to randomly generate a valid configuration:

- **Direct Calculation:** this method first calculates the valid range for the values of each independent variable[4] and then randomly selects a value within the range for the variable. In this way, all sampled configurations are valid ones. For single-PC CFs and some two-PC CFs (see Section 3.1 and 3.2 for details), it is a good method for sampling since in those cases, the value ranges of all independent variables can be efficiently calculated.
- **Hybrid Method:** this method is for sampling regarding the other two-PC CFs where the range of valid values for certain variables are hard or nearly impossible to calculate. It first uses **Direct Calculation** to obtain valid samples for variables whose ranges can be efficiently computed. Then, if there are still other variables, it uses the following two-step procedure to obtain a valid random sample for each such variable without calculating the range of valid values with respect to the two-PC CF:

  Step 1: Use **Direct Calculation** to randomly find a value of the variable *satisfying one PC only.*

  Step 2: If the value sampled does not result in a configuration satisfying the other PC as well, simply discard the value and repeat Step 1, which we call **resampling**. Alternatively, a **convergent iteration** strategy can be used to modify the invalid value iteratively until a valid value is resulted (i.e. it leads to a valid configuration satisfying both PCs).

[4]A valid range refers to the range of values for a variable which satisfy the contact constraints of the CF and do not cause local penetration between the two objects.

3.1 Sampling for single-PC CFs

Given a CF={PC_1}, and a valid configuration C_{seed} under the CF, the following function randomly generates a new valid configuration under the CF:

```
func $random\_sample\_1PC(C_{seed}, PC_1)$
begin
  // randomly translate $A$ from $C_{seed}$ to get $C$
  $C \leftarrow trans\_1PC(C_{seed}, PC_1)$;
  // randomly rotate $A$ from $C$ to get new $C$
  $C \leftarrow rotate\_1PC(C, PC_1)$;
  return $C$;
end.
```

The above function calls two subfunctions, which we explain in turn now.

Function $trans\_1PC()$ is used to translate A randomly along the contact plane of PC_1 to new valid configurations. Note that finding explicitly the valid ranges for the translational variables needs to calculate the Minkowski sum of the two contacting elements of the PC. In this function, we use a simple and efficient method to achieve the same sampling effects without calculating the Minkowski sum. The function randomly picks two points on the two contacting elements u_A and u_B of PC_1 and then translates A until the two points meet. In this way, the translation always leads to a valid configuration (i.e., maintaining the PC), and the configurations are evenly sampled due to the uniform randomness.

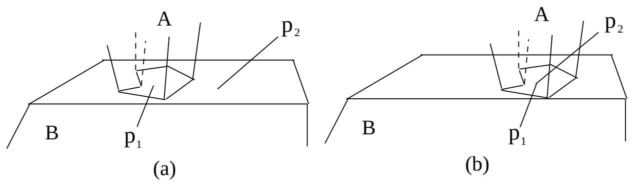


Figure 6: *Example {f-f} CF and the sampling procedure of the translational variables: (a) pick p_1 and p_2 randomly on the two contacting faces of A and B respectively, and (b) translate A so that p_1 and p_2 meet.*

The function $trans\_1PC()$ is outlined below:

```
func $trans\_1PC(C_t, PC_1)$
begin
  // $u_A$ is the contacting element of A in $PC_1$
  randomly sample a point $p_1$ on $u_A$;
  // $u_B$ is the contacting element of B in $PC_1$
  randomly sample a point $p_2$ on $u_B$;
  $C \leftarrow$ translate $A$ from $C_t$ so that $p_1 = p_2$;
  return $C$;
end.
```

Note that if any of u_A and u_B is a vertex, the randomly sampled point on it is the vertex itself; if any of them is an edge or a face, the randomly sampled point is a random point *inside* the bounded edge or face. It is easy to pick a point randomly inside a bounded edge or a bounded convex face: the point is some convex combination of the boundary vertices. If the face is concave, it is first decomposed into several convex parts (e.g., triangles), and then the point is sampled inside the convex parts.

Function $rotate\_1PC()$ generates a random rotation (of A) *compliant to* the contact constraints of the CF to achieve a new valid contact configuration. It produces the random or arbitrary rotation through a sequence of rotations with respect to each independent rotational variable. For each such variable, it first calls a function $get\_axis()$ to get its axis r and then calls $find\_angle\_range()$ to determine the valid range of values for the variable (to satisfy the CF and cause no local penetration). Next it randomly picks an angle inside the range and makes a rotation about r by the angle. The function is outlined below.

```
func $rotate\_1PC(C_r, PC_1)$
begin
  $C \leftarrow C_r$;
  $d \leftarrow$ rotational dofs of the CF;
  for $l = 1$ to $d$ do begin
    $r \leftarrow get\_axis(C, l, PC_1)$;
    $(\theta_1, \theta_2) \leftarrow find\_angle\_range(C, r, PC_1)$;
    $\theta \leftarrow$ randomly sampled angle in $(\theta_1, \theta_2)$;
    $C \leftarrow$ rotate $A$ by $\theta$ about $r$ from $C$;
  end;
  return $C$;
end.
```

The function $get\_axis()$ works based on the type of the CF. The axes are determined to facilitate the calculation of valid value ranges for the rotations: the first axis is along the normal of the contact plane of the PC, the second axis (if exists) is either along the contact line (for line PCs) or any line on the contact plane of the PC and passing through the contact point (for point PCs), and the third axis (if exists) is determined by the right-hand rule from the other two. The angle range for the first rotation is $(-\pi, \pi]$, and the ranges for the other two (if exist) are returned by function $find\_angle\_range()$, which uses the algorithm described in [18] to calculate angle ranges.

3.2 Sampling for two-PC CFs

For two-PC CFs with only one translational degree of freedom, **Direct Calculation** is sufficient for sampling valid configurations, while for other two-PC CFs, the **Hybrid Method** introduced earlier (in the beginning of Section 3) is used to sample valid configurations. Nevertheless, we can combine the sampling processes for all two-PC CFs in a general function as described below.

From a given seed configuration C_{seed}, function $random\_sample\_2PC()$ randomly generates a valid configuration for an arbitrary two-PC CF. Without losing generality, we designate *PC_1 to be the PC with fewer dofs if PC_1 and PC_2 have different dofs.* Let

CP_1 and CP_2 denote the contact planes of PC_1 and PC_2 respectively, CL denote the intersecting line of CP_1 and CP_2 if they are not parallel, and $\vec{cl}$ denote a unit vector along either direction of CL. The function is outlined as follows:

```
func $random\_sample\_2PC(C_{seed}, PC_1, PC_2)$
begin
  if (CF has 1 translational dof) then
    $C \leftarrow trans\_2PC(C_{seed}, PC_1, PC_2, \vec{cl})$;
  else begin
    $\vec{v_0} \leftarrow$ random unit vector on $CP_1$;
    $C \leftarrow trans\_2PC(C_{seed}, PC_1, PC_2, \vec{v_0})$;
  end;
  for $l = 1$ to 3 do
    if (CF has $l$-th rotational dof) then
      $C \leftarrow rotate\_1dof(l, C, PC_1, PC_2)$;
  return $C$;
end.
```

In the above function, independent translational variables are sampled by **Direct Calculation** with guaranteed valid values. Function $trans\_2PC()$ implements a random translation satisfying the two-PC CF. With the axis $\vec{v}$ as an input, the function starts from a given valid configuration C_t and calls a procedure $find\_trans\_range()$ to calculate the valid range of translations along $\vec{v}$ *relative to the given configuration* C_t, and then it randomly generates an increment of translation for object A within the valid range and obtains a new valid configuration, as outlined below.

```
func $trans\_2PC(C_t, PC_1, PC_2, \vec{v})$
begin
  // find translational ranges along $-\vec{v}$ and $\vec{v}$
  // while maintaining $PC_1$ and $PC_2$. $d_1, d_2 \geq 0$
  $[-d_1, d_2] \leftarrow find\_trans\_range(C_t, \vec{v}, PC_1, PC_2)$;
  $d \leftarrow$ randomly sampled value in $[-d_1, d_2]$;
  //if $d < 0$, translate along $-\vec{v}$ by $|d|$
  $C \leftarrow$ translate $A$ along $\vec{v}$ by $d$ from $C_t$;
  return $C$;
end.
```

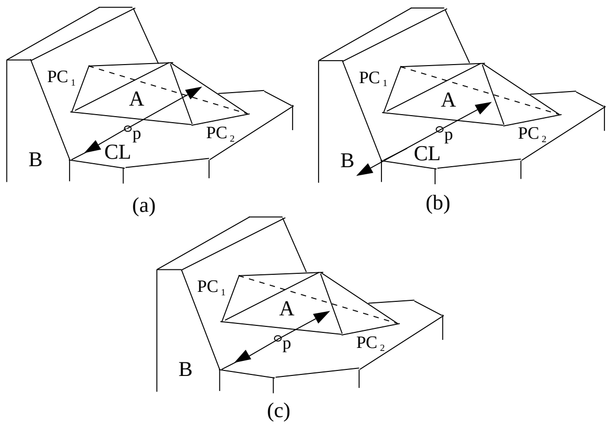


Figure 7: *The calculation of valid range of the translation variable along CL for a CF with two non-parallel line PCs: (a) calculate the range for PC_1 along CL, (b) calculate the range for PC_2 along CL, (c) find intersection of the ranges.*

The implementation of function $find\_trans\_range()$ involves computing the shortest separation distance between the two contacting elements of the PC involved, which can be two faces, a face and an edge, a face and a vertex, or two edges, along a given tangential direction of the contact plane. Fig. 7 shows how $find\_trans\_range()$ works in the case where CP_1 and CP_2 are non-parallel by an example. The function first projects an arbitrary point of A to CL, denoted as p, and then calculates the valid ranges to translate A along $\vec{cl}$ and $-\vec{cl}$ without breaking PC_1 and PC_2 respectively. Next it finds the intersection of the two ranges and returns it as the final range. In the case of a given random unit vector, the function works similarly.

To sample rotational variables, we need to first determine rotation axes. From analyzing all kinds of two-PC CFs, we discover the following three general characterizations of rotation axes, each corresponding to one rotational variable (if it exists), which are sufficient for all possible rotations constrained by two-PC CFs. We simply index them by $l = 1, 2, 3$ in $random\_sample\_2PC()$:

- $l = 1$: the rotation axis is denoted by X and defined as passing through one point on PC_2 (i.e., one point on both contacting elements of PC_2) along the normal of CP_1.

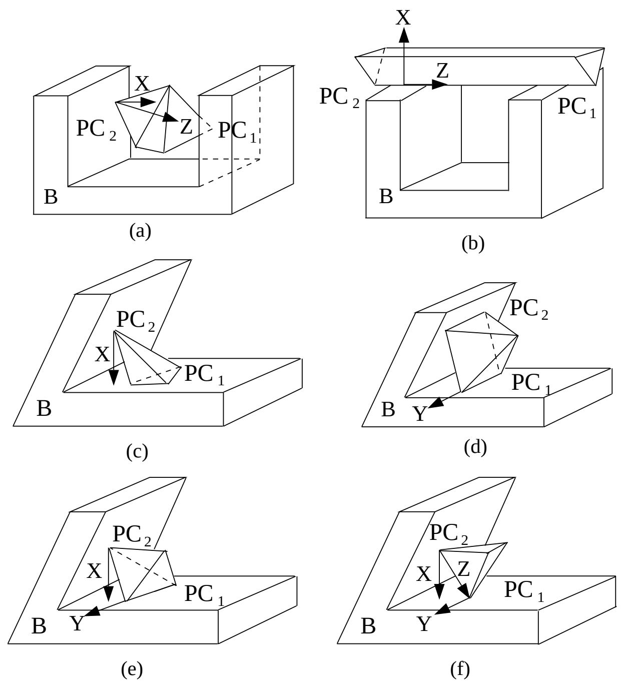


Figure 8: *The axes of the three rotational variables for various CFs, (a) two point PCs with parallel CPs, (b) two line PCs with parallel CPs and collinear contact lines, (c) one point and one plane PCs, (d) two lines PCs with parallel contact lines, (e) one point and one line PCs, (f) two point PCs.*

- $l = 2$: the rotation axis is denoted by Y and defined as along the contact line of PC_1 (if PC_1 is a line PC), or passing through the contact point of PC_1 and parallel to CL (if PC_1 is a point PC).
- $l = 3$: the rotation axis is denoted by Z and defined as passing through the two contact points (for two point PCs) or along one contact line (for two collinear line PCs, or one point and one line PCs with the point on the line).

Fig. 8 illustrates these three kinds of rotation axes in various examples. Note that in all cases, the rotation axis Y is actually for a combined rotation and translation in order to maintain the CF. As the rotation and translation are mutually dependent, there is only one independent variable. We will explain its sampling later.

Clearly, depending on the rotational dofs, not all rotations about these axes are always possible. Table 1 summarizes all kinds of two-PC CFs, their rotational dofs and corresponding rotational axes.

Now we explain the sampling strategies regarding rotational variables about X, Y, and Z respectively in more detail.

About X, i.e., $l = 1$:

We use the **Hybrid Method** introduced earlier. The function $rotate\_1dof(1, C, PC_1, PC_2)$ first calculates the angle range about X which satisfies PC_2 by calling $find\_angle\_range()$ (see Section 3.1). It then samples an angle θ inside the range, and next rotates A about X by θ to get a new configuration C. If C also satisfies PC_1, i.e., forms a valid configuration, it is returned; otherwise, either **resampling** or **convergent iteration** can be used (as introduced before Section 3.1). Here convergent iteration is to modify θ by $k\theta$, i.e., $\theta \leftarrow k\theta$, where $0 < k < 1$, repeatedly until θ results in a valid configuration, i.e., a convergence to the valid value range is achieved.

About Y, i.e., $l = 2$:

Sampling again uses the **Hybrid Method**. As mentioned earlier, the motion here is a combined translation and rotation with one independent variable. In all cases where this motion is possible (Fig. 8 gives some examples), CP_1 and CP_2 are not parallel, and they intersect at line CL.

The function $rotate\_1dof(2, C, PC_1, PC_2)$ uses a translational variable d along an axis $\vec{v}$ on CP_1 and perpendicular to CL as the independent variable for the combined motion. It first randomly samples d with a value satisfying PC_1 by **Direct Calculation** (in a procedure similar to but simpler than $trans\_2PC()$, since only one PC needs to be satisfied). Next it checks whether PC_2 can also be satisfied by a *guarded* rotation about Y, with the angle calculated, which depends on the value of d. If so, the function returns a valid configuration C; otherwise, again, either **resampling** or **convergent iteration** on the value d (i.e., $d \leftarrow kd$, where $0 < k < 1$, repeatedly until d results in a valid configuration) can be used. Fig. 9 shows how a rotation

CF type		rot. dofs	X	Y	Z
$CP_1 \parallel CP_2$	two point PCs	2	$\surd$	x	$\surd$
	collinear contact points/lines	2	$\surd$	x	$\surd$
	others	1	$\surd$	x	x
$CP_1 \neg \parallel CP_2$	2 plane PCs	0	x	x	x
	1 line, 1 plane PCs	0	x	x	x
	two non-parallel line PCs	0	x	x	x
	1 point, 1 plane PCs	1	$\surd$	x	x
	two parallel line PCs	1	x	$\surd$	x
	1 point, 1 line PCs	2	$\surd$	$\surd$	x
	2 point PCs	3	$\surd$	$\surd$	$\surd$

Table 1: *Two-PC CFs, their rotational dofs, and corresponding rotational axes*

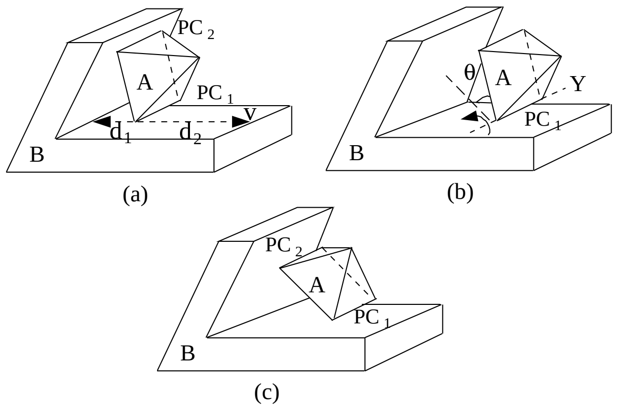


Figure 9: *Sampling a combined motion with rotation about Y for a CF with two line PCs: (a) seed configuration and translational range $[-d_1, d_2]$ along $\vec{v}$, (b) configuration after translation along $\vec{v}$ by a sampled d (not shown) and the guarded rotation angle θ, (c) configuration after the guarded rotation about Y by θ to meet PC_2.*

about Y is sampled.

About Z, i.e., $l = 3$:

Unlike the two previous cases about X and Y, here the rotational variable about Z is sampled using **Direct Calculation**. The function $rotate\_1dof(3, C, PC_1, PC_2)$ first calculates the rotational angle ranges $(\theta_{11}, \theta_{12})$ and $(\theta_{21}, \theta_{22})$ about Z for PC_1 and PC_2 respectively by calling $find\_angle\_range()$ (see Section 3.1), and then finds the intersection of the ranges as the valid range of the rotational variable. Finally an angle θ is sampled randomly inside the range, and A is rotated about Z by θ to generate a valid configuration.

In summary, we have presented above a general function $random\_sample\_2PC()$ to produce random configurations satisfying any given two-PC CFs. Note that only in two steps regarding rotations about X and Y (i.e., $l = 1, 2$), the **Hybrid Method** is used, and **Direct Calculation** is used in sampling all the other variables to maximize efficiency.

In the **Hybrid Method**, the alternative to **Direct Calculation** is either **resampling** or **convergent iteration**. Resampling for a single variable is simply to repeat the sampling process for that variable until a value which results in a valid configuration is found or after some pre-determined number of tries.

Convergent iteration, on the other hand, guarantees to find a valid value for the variable or makes it converge to the valid value range. If there are more than one connected valid value range, which may happen in the cases where the two-PC CF has multiple connected regions of contact configurations, caution is needed to make the convergence to each valid range equally likely in order to ensure the even distribution of samples. This, however, can be achieved by always using the newly randomly sampled configuration as the seed configuration for the next sample.

4 Experimental Results

The random sampling strategy for single-PC and two-PC CFs has been implemented in *C*. The program runs on SUN Ultra10 workstation. The machine is rated at 12.1 SPECint95 and 12.9 SPECfp95. The input to the program is a CF and a valid contact configuration satisfying the CF. The output are random configurations of the same CF which are guaranteed no local penetration. We use VRML as the output format.

Fig. 10 shows the sampling results for several single-PC CFs between a cube A and an L-shape B. Fig. 11 shows the sampling results for two-PC CFs between different shapes of objects. The running time (in seconds) for generating 1000 samples of the examples in Fig. 10 and Fig. 11 are summarized in Table 2.

From Table 2, clearly it takes much shorter time to generate samples for single-PC CFs. This is because, though usually single-PC CFs have higher dofs, sampling is done by **Direct Calculation**. For two-PC CFs of the same objects, usually the higher dofs the CF has, the more time is needed to sample the same number of configurations, although the time also depends on the geometry of the objects. For the same CF, the running time of the algorithm is nearly proportional to the number of samples generated.

In the last two rows of Table 2, we show the running times of the examples using **convergent iteration** and **resampling** respectively. It seems that **convergent iteration** runs faster in most cases. Our experiments show that for **convergent iteration** with $k = 0.5$, after at most 13 iterations, a valid configuration can be obtained for all the four examples. For **resampling**, on the other hand, after at most 73 resampling iterations, a valid configuration can be obtained.

Our experiments also show that among the three rotational variables, the sampling of the second one about Y with dependent translation is the most expensive, followed by that of the first variable, and sampling for the third variable is the fastest because of **Direct Calculation**.

5 Conclusion

This paper addresses random sampling of configurations constrained by contact, which is not only necessary for planning contact motions with certain randomized planners but also useful for planning collision-free motions since the sampled contact configurations probabilistically characterize the C-obstacles. An efficient random sampling strategy is implemented for sampling configurations constrained by single-PC or two-PC CFs, which satisfy contact constraints of the CF without causing local penetration. The strategy is characterized by directly computing valid samples wherever possible to maximize efficiency. In the next step, we intend to apply such a strategy to randomized contact motion planning.

Acknowledgement The authors appreciate early inspiration from discussions with Jean-Paul Laumond, Jean-Claude latombe, and David Hsu.

References

[1] N. Amato, O. Bayazit, L. Dale, C. Jones, D. Vallejo, "OBPRM: An Obstacle-Based PRM for 3D Workspaces", *Workshop Algor. Found. of Robotics*, pp. 155-168, Mar. 1998.

[2] F. Avnaim, J. D. Boissonnat, and B. Faverjon, "A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles", *IEEE Int. Conf. Robotics & Automation*, pp. 1656-1661, Apr. 1988.

[3] V. Boor, M. H. Overmars, and A. F. Stappen, "The Gaussian Sampling Strategy for Probabilistic Roadmap Planners", *IEEE Int. Conf. Robotics & Automation*, May 1999.

[4] R. Brost, "Computing Metric and Topological Properties of C-space Obstacles", *IEEE Int. Conf. Robotics & Automation*, pp. 170-176, May 1989.

[5] B. Donald, "A Search Algorithm for Motion Planning with Six Degrees of Freedom", *Artificial Intelligence*, pp. 295-353, 31(3), 1987.

[6] A. Farahat, P. Stiller, and J. Trinkle, "On the Algebraic Geometry of Contact Formation Cells for Systems of Polygons", *IEEE Trans. Robotics & Automation*, 11(4), pp. 522-536, Aug. 1995.

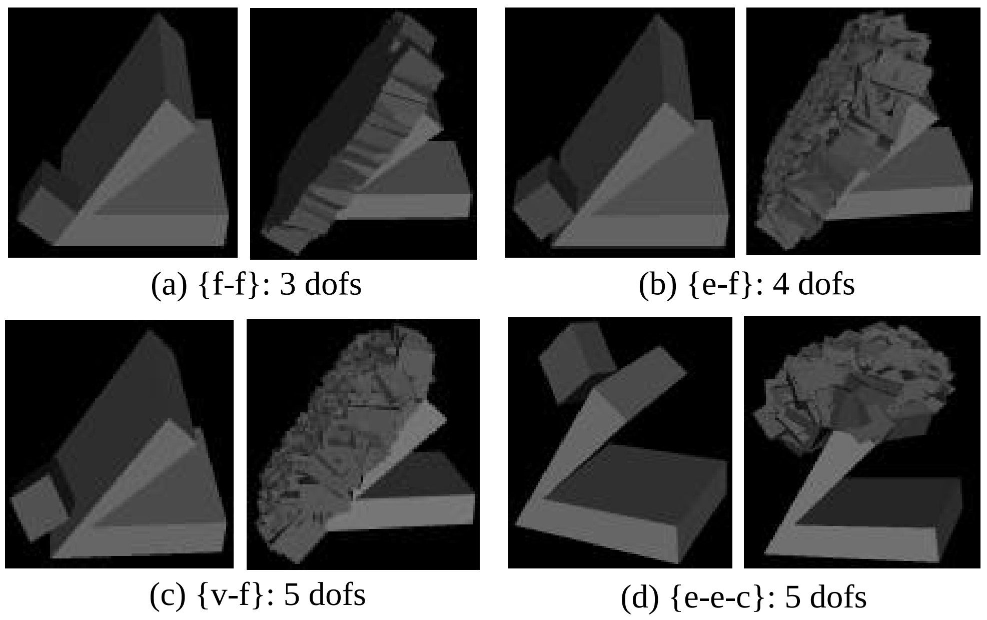


Figure 10: *Examples for single-PC CFs: seed configurations and the results for 1000 samples*

[7] H. Hirukawa, "On Motion Planning of Polyhedra in Contact", *Workshop Algor. Found. of Robotics*, 1996.

[8] D. Hsu, L. Kavraki, J.C. Latombe, R. Motwani, and S. Sorkin, "On Finding Narrow Passages with Probabilistic Roadmap Planners", *Workshop Algor. Found. of Robotics*, pp. 141-153, Mar. 1998.

[9] X. Ji and J. Xiao, "Automatic Generation of High-Level Contact State Space," *IEEE Int. Conf. Robotics & Automation*, pp. 238-244, May 1999.

[10] X. Ji and J. Xiao, "Towards Random Sampling with Contact Constraints", 2000 *ICRA*.

[11] L. Joskowicz, R. H. Taylor, "Interference-Free Insertion of a Solid Body Into a Cavity: An Algorithm and a Medical Application", *Int. J. Robotics Res.*, 15(3):211-229, June 1996.

[12] L.E. Kavraki, P. Svestka, J.C. Latombe, and M. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces", *IEEE Trans. Robotics & Automation*, 12(4):566-580, 1996.

[13] T. Lozano-Pérez, "Spatial Planning: A Configuration Space Approach", *IEEE Trans. Comput.*, C-32(2):108-120, 1983.

[14] M. T. Mason, "Compliant Motion", *Robot Motion: Planning and Control*, MIT Press, 1982.

[15] C. Nissoux, T. Simeon, and J. P. Laumond, "Visibility based probabilistic roadmaps", *IEEE Int. Conf. Intell. Robots and Systems*, 1999.

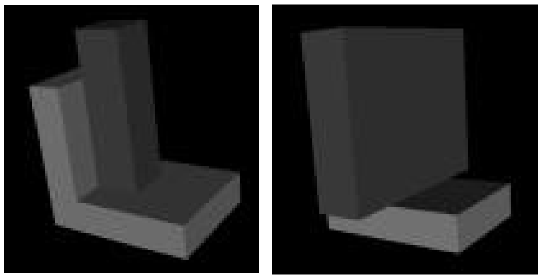
[16] D. E. Whitney, "Historical Perspective and State of the Art in Robot Force Control", *IEEE Int. Conf. Robotics & Automation*, pp. 262-268, 1985.

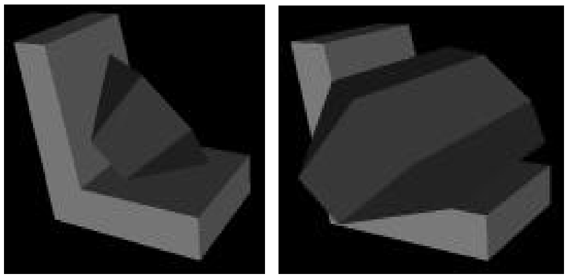
[17] J. Xiao, "Automatic Determination of Topological Contacts in the Presence of Sensing Uncertainties," *IEEE Int. Conf. Robotics & Automation*, pp. 65-70, May 1993.

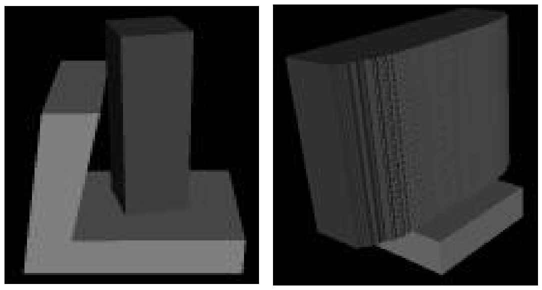
[18] J. Xiao and L. Zhang, "Computing Rotation Distance between Contacting Polytopes," *IEEE Int. Conf. Robotics & Automation*, pp. 791-797, Apr. 1996.

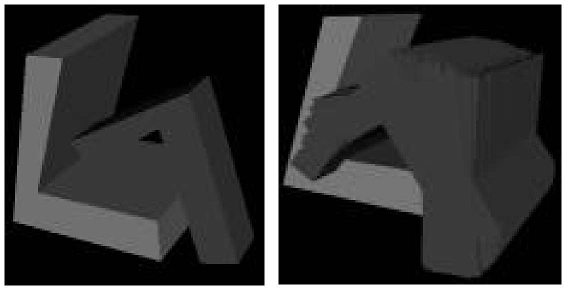
[19] J. Xiao and L. Zhang, "Contact Constraint Analysis and Determination of Geometrically Valid Contact Formations from Possible Contact Primitives", *IEEE Trans. Robotics and Automation*, 456-466, Jun. 1997.

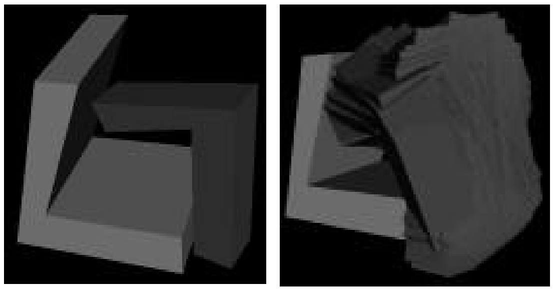
[20] J. Xiao and X. Ji, "A Divide-and-Merge Approach to Automatic Generation of Contact States and Planning of Contact Motion", accepted to 2000 *ICRA*.


(a) {f-f, f-f}: 1 dof


(b) {e-f, e-f}: 1 dof


(c) {e-e-c, f-f}: 2 dofs


(d) {e-f, f-e}: 2 dofs


(e) {v-f, e-f}: 3 dofs

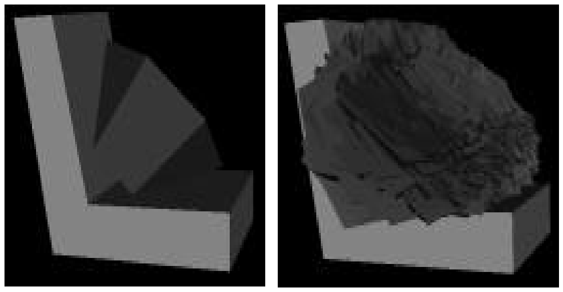

(f) {v-f, v-f}: 4 dofs

Figure 11: *Examples for two-PC CFs: seed configurations and the results for 1000 samples*

Method	CF	dofs	time(s)	CF	dofs	time(s)
Direct	{f-f}, Fig. 10(a)	3	0.23	{e-f}, Fig. 10(b)	4	0.25
Direct	{v-f}, Fig. 10(c)	5	0.45	{e-e-c}, Fig. 10(d)	5	0.45
Direct	{f-f, f-f}, Fig. 11(a)	1	6.7	{e-f, e-f}, Fig. 11(b)	1	1.5
Hybrid	{e-e-c, f-f}, Fig. 11(c)	2	3.7 or 3.8	{e-f, f-e}, Fig. 11(d)	2	61.1 or 56.1
Hybrid	{v-f, e-f}, Fig. 11(e)	3	50.4 or 50.9	{v-f, v-f}, Fig. 11(f)	4	34.7 or 40.4

Table 2: *Examples in Fig. 10 and Fig. 11 and their running times for 1000 samples. In the last two rows, the two numbers given for each case under* **times(s)** *correspond to the running times using* **convergent iteration** *and* **resampling** *respectively.*