グラフィクスとCAD 31－3
（1988. 2. 26）

シミュレーションによる移動物体間の衝突検出

嶋田憲司　　岡野彰　　川辺真嗣

日本アイ・ビー・エム(株)　東京基礎研究所

本報告では、シミュレーションによって移動物体間の衝突を検出するための二つの方法を提案する。ひとつの方法は従来の交点計算法の考え方をもとに、さらに形状要素の3次元運動を時間の関数として一般的に表現できるように拡張したものである。これによってロボットの複雑な動作に対しても見逃しなく衝突が検出でき、その時刻と状態が算出できる。また、もうひとつの方法は、ある軌道における衝突発生の可能性を判断できる判別条件を導入し、この条件の判別と軌道の2分割を再帰的に繰り返すことによって衝突を検出するものである。この方法では衝突の可能性が小さい場合に非常に少ない計算量で処理が終わるので、平均的な計算量は従来のサンプリング法をnとしたとき$\log_2(n)$に改善される。これら二つの方法は、筆者らが開発中のオフラインロボットプログランミングシステムのために考案されインプリメントされたものである。

Collision Detection among Moving Objects in Simulation

Kenji SHIMADA,　Akira OKANO,　and　Shinji KAWABE
IBM JAPAN, Tokyo Research Laboratory
5-19 Sanbancho, Chiyodaku, Tokyo, 102, Japan

This paper describes two algorithms of different types for detecting collisions between moving objects in simulation. In the first, we propose a general way of expressing the 3D trajectory of a geometric element as a function of time; the basic idea of the algorithm is the same as that of the conventional intersection calculation method. By using the algorithm we can detect a collision with absolute certainty, and know the time and status of the collision even in the complex 3D motions of robots. In the second algorithm, we have introduced a condition which can indicate whether or not any collisions may occur in a trajectory. If any collisions may occur, the trajectory is divided into two parts. By recursively repeating this process, we can detect a collision. The average computational cost is reduced to $\log_2(n)$, where n expresses the cost of the conventional sampling method. Both of the algorithms were proposed and implemented for our off-line robot motion programming system.

1．はじめに

製品の多様化と開発製造サイクルの短縮化に対応するために生産ラインへの産業用ロボットの導入が進んでいる。従来このようなロボットの動作は実機を用いて生産現場でプログランミングされていたが、この方法では生産ラインを長時間にわたって停止しなければならない。この問題を避けるために実機を用いずにロボットの動作データを生成するオフラインプログランミングの方法が求められている。この場合には生成されたデータを実機にダウンロードして動かす前に、そのデータが正しいことを計算機シミュレーションによってあらかじめ検証しておくことが望まれる。特にアームが他のアームや作業環境と不都合な衝突を起こさないことを確認することが大切である。さらに、このような衝突検出の機能はロボットのシミュレーションだけでなく、製造プロセス全般にわたっても広く利用される技術であると思われる。そこで、本稿では物体の3次元形状データと動作軌道が与えられたときに、衝突の有無をチェックし、衝突が発生する場合にはその時刻と発生時の位置姿勢を算出するために以下の二方法を提案し評価を行なう。

・従来の交点計算法を一般化した衝突検出方法
・衝突に関する判別条件を用いる衝突検出方法

これらはお互いに全く独立した手法であるが、筆者らが開発しているオフラインロボットプログランミングシステム[1,2]のために考案され実際にインプリメントされたものである。

2．移動物体間の衝突検出方法

最初に従来の衝突検出方法について述べ、いくつかの問題点を指摘してみる。これまでに提案された方法[3,4,5,6,7]は以下の3種類に大別できる。

・サンプリング法：　充分に小さい一定時間間隔Δt ごとに移動物体の位置、姿勢を補間して静止状態での干渉問題として解く方法
・包絡体生成法：　移動物体が通過する空間をひとつの立体として生成して、この立体どうしの干渉を調べる方法
・交点計算法：　移動物体の面、稜線、頂点などの軌跡を時間の関数として数式表現し、これらの間の方程式を解いて交点を求める方法

サンプリング法はインプリメントが容易であるために多く用いられているがサンプリングの時間間隔Δtが大き過ぎると衝突を見逃す可能性が大きくなる。また、包絡体生成法では衝突の有無を知ることはできるが発生時刻や発生状態が得られない。このようにサンプリング法と包絡体生成法の問題点は衝突を静止状態での干渉問題に帰着させて解いていることに起因する。これに対して交点計算法は運動を時間の関数として陽に表現するので見逃しがなく発生の時刻や状態も正確に知ることができるという点で他の方法よりも優れている。しかし、これまでに提案された文献4では交点が解析的に計算できる純粋な回転運動や並進運動に関して定式化されているだけで一般的な運動の扱いは示されていない。また、文献5では一般的な計算式が示されているが処理が複雑でインプリメントが難しい。そこで本稿の前半では文献4の考え方をもとに一般的な運動を扱えるように機能を拡張した方法を提案する[8]。この方法は文献5に比較してインプリメントも容易である。

さて、以上の方法においては物体の形状を表現するために計算機内の3次元形状モデルのデータが使われる。一般に3次元形状モデルを用いる計算において常に指摘される問題点は計算量の膨大さであるが、衝突検出の場合も例外ではない。衝突検出における計算量増大の原因は形状に関するものと軌道に関するものが考えられる。形状に関しては従来から考慮されてきた。この最も単純な方法は正確な形状を含むような粗い形状、例えば、包絡球、包絡直方体、包絡円柱などを用いたラフチェックであり、その有効性はよく知られている。しかし、もう一方の軌道に関連して計算量の増加を抑えることは、これまでの方法では考慮されていない。そこで、本稿の後半では、軌道に着目して導いた衝突に関する判別条件を利用して少ない計算量で衝突を検出する方法を提案する[9]。この方法の特長は軌道の複雑さにかかわらず衝突の可能性が小さい場合には計算量が非常に少なくなるという点である。また、見逃しがなく衝突時刻と発生状態が得られるという要件も満たしている。

3．交点計算法による衝突検出方法

物体の3次元形状は多面体で近似され、物体の表面は面、稜線、頂点の3種類の形状要素で構成されているものとする[10]。また、運動の軌道は一定時間間隔ごとの位置姿勢を表す一連の 4×4 の同次座標行列として与えられるものとする。このとき形状要素の軌跡を時間tの関数として数式表現してこれらの間の交点を計算する方法を考える。一般に多面体で表現された移動物体どうしの衝突には次の2通りの状況が考えられる[4]。

・頂点と面が衝突する。（図1）
・稜線と稜線が衝突する。（図2）

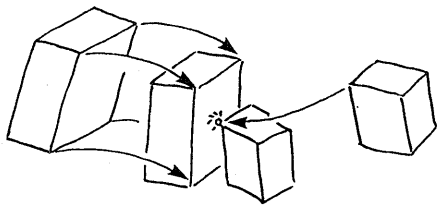


Fig. 1 Collision between a plane and a vertex

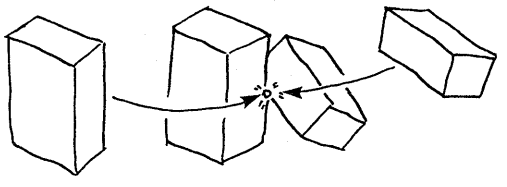


Fig. 2 Collision between an edge and another edge

したがって、頂点と面、稜線どうしの全組合せについて交点計算を行なう必要がある。

まず軌道の与え方を定義しておく。物体の位置姿勢を時刻順に４個の同次座標行列で表現することとし、これを順にT_{i1},T_{i2},T_{i3},T_{i4}とする。添字のｉは物体の名前である。T_{i1}は移動開始時刻、T_{i4}は移動終了時刻での位置姿勢である。T_{i2},T_{i3}は各々、移動開始時刻から1/3,2/3の時間が経過した時点での位置姿勢を示す。この４つの行列を用いて物体の軌道を表す同次座標行列の各要素を時間tの３次関数で近似し、以後これを $T_i(t)$と表現することにする。もし全体として軌道が複雑でひとつの３次関数で近似できない場合は、3n+1 個の行列 T_{i1},T_{i2},....T_{i3n+1}を与えて軌道をn個に分けた上で同様に３次関数で近似するものとする。

次に、形状要素の表現法を定義しておく。各頂点の位置に関しては物体の基準座標系で表現された座標値 (x,y,z,1) が与えられている。また、稜線については

$$x = a_{ij}u + b_{ij}, \quad 0 \leq u \leq 1$$

という形で表した時の a_{ij} と b_{ij} が定義されている。ここで a_{ij} は物体の基準座標系で表現された稜線の方向ベクトルであり、 b_{ij} は稜線の一端点の位置ベクトルである。さらに面については

$$x = a_{ij}u + b_{ij}v + c_{ij}$$

という形で表した時の a_{ij}, b_{ij}, c_{ij} が定義される。 a_{ij}, b_{ij} は物体の基準座標系で表現された面上の二つのベクトルであり、 c_{ij} は面上の一点の位置ベクトルである。

以上のように移動物体の位置姿勢を $T_i(t)$ 、その物体の基準座標系で表現した頂点の座標を p_{ij} とすると、世界座標系で表現した頂点の座標は

$$x = T_i(t)p_{ij}$$

となる。また世界座標系で表現された稜線の式は

$$x = T_i(t)\ (\ a_{ij}u + b_{ij}\ )$$

となる。同様に世界座標系で表現された面の式は

$$x = T_i(t)\ (\ a_{ij}u + b_{ij}v + c_{ij}\ )$$

となる。これで形状要素の軌道が時間の関数として表現されたので二つの衝突の状況にわけて交点の計算法を以下に示す。

頂点と面が衝突する場合

移動物体を ｉ, ｋ とする。物体ｉの面と物体ｋの頂点の式は以下のように表現される。

物体ｉの面　：$x = T_i(t)\ (\ a_{ij}u + b_{ij}v + c_{ij}\ )$

物体ｋの頂点：$x = T_k(t)p_{kl}$

交点は二つの式の右辺を等しくおいた次の方程式の解として算出できる。

$$T_i(t)\ (\ a_{ij}u + b_{ij}v + c_{ij}\ ) = T_k(t)p_{kl}$$

$$\therefore\ T_i(t)a_{ij}u + T_i(t)b_{ij}v + T_i(t)c_{ij} - T_k(t)p_{kl} = 0$$

$$\therefore \begin{bmatrix} T_i(t)a_{ij} & T_i(t)b_{ij} & T_i(t)c_{ij}-T_k(t)p_{kl} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = 0$$

$$\therefore \left| T_i(t)a_{ij} \quad T_i(t)b_{ij} \quad T_i(t)c_{ij}-T_k(t)p_{kl} \right| = 0$$

このように行列式が0となるような t, u, v が解の候補となるので、全ての j, l の組合せ、すなわち全ての面と頂点の組合せについて解けばよい。$T_i(t)$と$T_k(t)$の各要素は時間tの３次関数として表されているので、この方程式はtの９次方程式となる。一般に実数係数n次代数方程式のn個の複素数解を求める方法は知られているので[11,12]この方程式から９個の解が得られる。実数解のうち、移動開始時刻と移動終了時刻との間にあるものに対し、その時刻において頂点が面の領域の内側かどうかを調べる。内側であれば最終的に衝突時刻の候補となる。

稜線と稜線が衝突する場合

物体ｉの稜線と物体ｋの稜線の式は以下のように表現される。

物体ｉの稜線：$x = T_i(t)\ (\ a_{ij}u + b_{ij}\ )$

物体ｋの稜線：$x = T_k(t)\ (\ c_{kl}v + d_{kl}\ )$

交点は二つの式の右辺を等しくおいた次の方程式の解として算出できる。

$$T_i(t)\ (\ a_{ij}u + b_{ij}\ ) = T_k(t)\ (\ c_{kl}v + d_{kl}\ )$$

$$\therefore\ T_i(t)a_{ij}u + T_i(t)b_{ij} - T_k(t)c_{kl}v - T_k(t)d_{kl} = 0$$

$$\therefore \begin{bmatrix} T_i(t)a_{ij} & -T_k(t)c_{kl} & (T_i(t)b_{ij}-T_k(t)d_{kl}) \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = 0$$

$$\therefore \left| T_i(t)a_{ij} \quad -T_k(t)c_{kl} \quad (T_i(t)b_{ij}-T_k(t)d_{kl}) \right| = 0$$

行列式が0となるような t, u, v が解の候補となるので、全ての j, l の組合せ、すなわち全ての稜線どうしの組合せについて解けばよい。この方程式もtの９次方程式である。得られた実数解のうち、移動開始時刻と移動終了時刻との間にあるものに対し、その時刻において交点が両方の稜線に含まれているかを調べる。含まれていれば最終的な解の候補となる。

頂点と面、稜線と稜線の全組合せを調べて残った最終的な解の候補の中で最も小さなtが最初の衝突発生の時刻である。このように３次元の運動を３次関数で補間して時間の関数として一般的に表現しておけば、これらの間で交点計算を行なうことによって衝突が検出でき、衝突の発生時刻とその時の状態も知ることができる。

４．衝突に関する判別条件を用いる衝突検出法

4.1 衝突に関する判別条件

最初に、移動物体間に衝突が発生するための必要条件を示し、さらにこの条件に基づいて衝突が発生しないための十分条件を導出する。

図３に示すように、時間 $t_s \leqq t \leqq t_e$ において物体Aと物体Bが移動したとき、

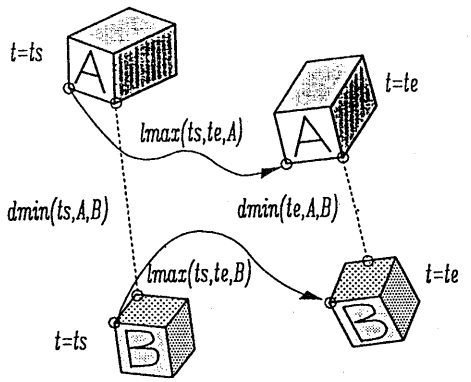
$d_{min}(t_s, A, B)$: 時刻t_sにおける物体間の最小距離

$d_{min}(t_e, A, B)$: 時刻t_eにおける物体間の最小距離

$l_{max}(t_s, t_e, A)$: $t_s \leqq t \leqq t_e$ における物体Aの最大移動長

$l_{max}(t_s, t_e, B)$: $t_s \leqq t \leqq t_e$ における物体Bの最大移動長

とする。ただし、「物体の最大移動長」とは「物体に属す



Feg. 3 Minimum distance and maximum trajectory length

る全ての点の移動長のうち最も大きいもの」のことである。

時刻 $t_s \leqq t \leqq t_e$ において物体Aと物体Bが衝突する場合には必ず以下の二つの条件が成立する。

条件P $d_{min}(t_s,A,B) \leqq l_{max}(t_s,t_e,A)+l_{max}(t_s,t_e,B)$

「移動開始時刻t_sにおける最小距離より $t_s \leqq t \leqq t_e$ における物体A,Bの最大移動長の和が大きい」

条件Q $d_{min}(t_e,A,B) \leqq l_{max}(t_s,t_e,A)+l_{max}(t_s,t_e,B)$

「移動終了時刻t_eにおける最小距離より $t_s \leqq t \leqq t_e$ における物体A,Bの最大移動長の和が大きい」

すなわち、（衝突が起きる） → （P ∧ Q）

であり、（P∧Q）は衝突が起きるための必要条件である。この命題の対偶をとると次のように衝突が起きないための十分条件が導かれる。

（P ∧ Q） → （衝突が起きる）

∴ （P ∨ Q） → （衝突が起きない）

すなわち、以下に示す判別条件が得られる。

```
/* 衝突に関する判別条件 */
if  dmin(ts,A,B) > lmax(ts,te,A)+lmax(ts,te,B)
        or
    dmin(te,A,B) > lmax(ts,te,A)+lmax(ts,te,B)
    /*  移動開始時または終了時における物体間の
        最小距離が最大移動長より大きい  */
then
    a collision should not happen at ts≦t≦te
```

4.2 衝突検出アルゴリズム

ここでは二つの物体が十分に小さい距離 D_{col} 以内に近づくことを衝突と定義する。4.1で導出した条件の判定と軌道の２分割を再帰的に行なうことがこのアルゴリズムにおいて最も重要な点である。

二つの移動物体A,Bが時間 $T_s \leqq t \leqq T_e$ において３次元空間内を姿勢の変化を伴いながら独立に移動する場合を考える。まず、$T_s \leqq t \leqq T_e$ に含まれる任意の時間 $t_s \leqq t \leqq t_e$ について 図４ のフローチャートに従って衝突に関する状況を調べる関数 STATUS(t_s,t_e) を定義する。この関数は 図５の４通りの状況のうちの一つを結果の値として返す。

状況 A : $t=t_e$ で衝突が検出された

状況 B : $t_s \leqq t \leqq t_e$ において衝突は起きない

状況 C : $t_s \leqq t \leqq t_e$ において衝突の可能性あり

状況 D : $t_s \leqq t \leqq t_e$ において必ず衝突が起きている

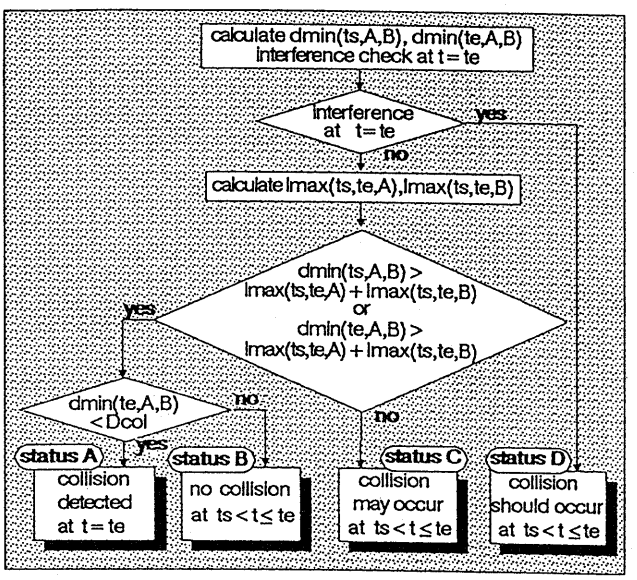


Fig. 4 STATUS(ts, te) checks a trajectory and returns the status A, B, C, or D

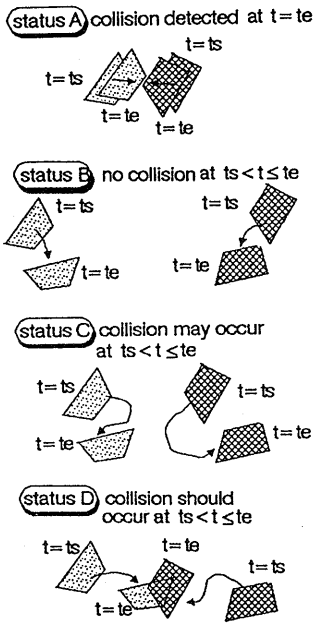


Fig. 5 Four collision status

この関数 STATUS(t_s,t_e) を用いると、以下の手続き COLLISION(t_s,t_e) を $t_s=T_s$, $t_e=T_e$ として呼び出すによって $T_s \leqq t \leqq T_e$ における衝突の有無と発生時刻を得ることができる。ただし、初期状態 $t=T_s$ において物体A,Bは干渉していないものとする。

```
procedure COLLISION(ts,te) :
begin
 ST←STATUS(ts,te);
 if ST=A then return( collision at tcol=te )
 else
  if ST=B then return( no collision )
  else
   /* in case of C or D, recursive calling */
    if COLLISION(ts,(ts+te)/2) =
       collision at tcol
    then return( collision at tcol )
    else return( COLLISION((ts+te)/2,te) );
end
```

STATUS(t_s,t_e) の結果が状況C、または状況Dである場合には、時間 $t_s \leqq t \leqq t_e$ において衝突が発生する可能性が残されるので、この時間を

$$t_s \leqq t \leqq (t_s+t_e)/2 \ , \ (t_s+t_e)/2 \leqq t \leqq t_e$$

に２分割して各々の時間について再帰的に調べる。最終的にCOLLISION(T_s,T_e) の処理が終了するのはある時間においてSTATUS(t_s,t_e) が状況Aを返すか、全ての時間において状況Bを返したときである。また、一連の動作中に複数の衝突が発生することもあるが、分割された二つの時間について常に前半、後半の順序で処理することにより、最も早い時刻に発生する衝突が検出できる。

図６に示すような例では、実際の処理の流れは以下のようになる。ただし、$T_s=0, T_e=1$ とする。

１．$0 \leqq t \leqq 1$　　状況C　　衝突可能性あり、
　　$0 \leqq t \leqq 4/8$, $4/8 \leqq t \leqq 1$ に２分割
２．$0 \leqq t \leqq 4/8$　状況D　　必ず衝突が起きる、
　　$0 \leqq t \leqq 2/8$, $2/8 \leqq t \leqq 4/8$ に２分割
３．$0 \leqq t \leqq 2/8$　状況B　　衝突可能性なし、
４．$2/8 \leqq t \leqq 4/8$ 状況D　　必ず衝突が起きる、
　　$2/8 \leqq t \leqq 3/8$, $3/8 \leqq t \leqq 4/8$ に２分割
５．$2/8 \leqq t \leqq 3/8$ 状況A
　　$t_{col}=3/8$ で衝突が検出された

4.3. アルゴリズムの評価

ここでは、これまで多く用いられてきたサンプリング法と比較しながら、本アルゴリズムの計算量について論じる。一般に、衝突検出では計算量に関して次の２点を満たすアルゴリズムが望ましい。

- 「解の精度」に応じた計算量である
- 「衝突の可能性」に応じた計算量である

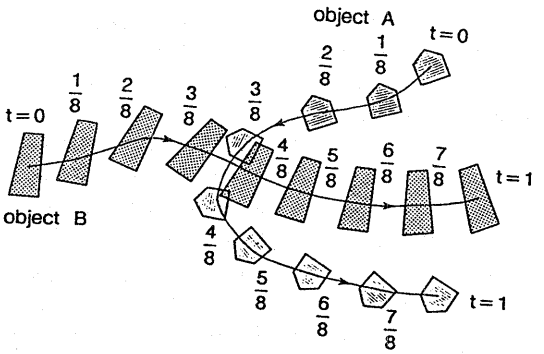


Fig. 6 Trajectories of two moving objects

ただし、ここでは、「衝突の可能性」を次のように直感的に考えることにする。

可能性が小さい：「物体が互いに離れた場所で
　　比較的小さな動きをする」
可能性が大きい：「物体が近接した位置関係を
　　保って比較的大きな動きをする」

最初に「解の精度」について考える。サンプリング法では、「解の精度」はサンプリングの時間間隔Δｔに依存する。良い精度を得るためにはΔｔを小さくすればよいが、計算量が多くなる。それほど精度が必要でない場合は、逆にΔｔを大きくして計算量を減らすことはできるが、衝突を見逃す可能性が生じる。

これに対して本アルゴリズムでは、「解の精度」に関係なく見逃しがないことが保証されている。距離 D_{col} 以内に物体どうしが近づいたときを衝突として検出するので、D_{col} の値そのものが「解の精度」であると考えてよい。サンプリング法と同様に D_{col} を小さく設定すれば計算量が多くなり、D_{col} を大きくすれば計算量が減少する。ただし、D_{col} を大きくして「解の精度」を悪くしても衝突を見逃すことはない。

次に、「衝突の可能性」と計算量の関係について考える。サンプリング法と本アルゴリズムの計算量を比較した結果を図７に示す。ここでは、単位となる処理の繰返し回数I,Lを以下のように定義する。

I:　サンプリング法において、単位となる処理である
　　「物体間の静止状態での干渉チェック」回数
L:　本アルゴリズムにおいて、単位となる処理である
　　「物体間の最小距離と最大移動長の算出」回数

単位となる処理１回に要する計算時間を各々T_i,T_lとすると、全体の計算量は本来、 $T_i \cdot I$, $T_l \cdot L$ として見積もるべきであるが、ここでは単に I,L を全体の計算量の測度として扱うことにする。一般にT_lのほうがT_iより大きいが、同じオーダーと考えてよいからである。

サンプリング法において、見逃しなく十分な精度が得られるような軌道の分割数を n とする。衝突が発生しない例題に対しては軌道を最後まで調べなければならないので「衝突の可能性」と関係なく常に　I=n　となる。また、衝

突が発生する例題では、検出した時点で処理を終了するので I は 1≦I≦n の範囲で変化し、平均的には I=n/2 である。ただし、この場合は「衝突の可能性」に関連して I の値が変わるわけではない。

これに対して、本アルゴリズムでは L の値は「衝突の可能性」に対応している。図7の (C), (C') のような衝突の可能性が大きい例題については、最悪の場合にサンプリング法の分割数と同じ L=n の計算量が必要となるが、(A), (A') のような可能性が小さい例題については最小の場合に L=1 で処理が終わる。一般に平均的な場合の L は、軌道の位置関係、物体の形状、衝突発生状態などの複数の要因の組合せに依存するので、単純に見積もることはできない。

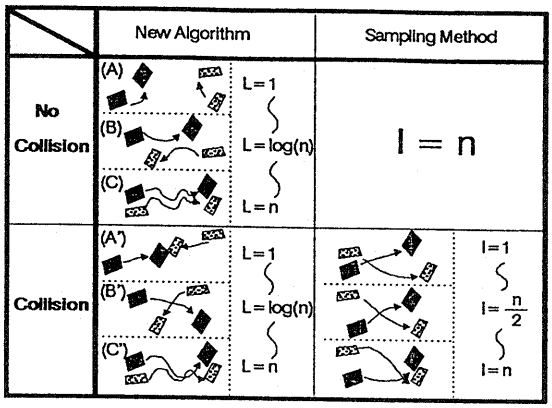


Fig. 7 Estimation of computational cost

しかし、平均的な難しさの例題の場合に軌道の2分割が確率 1/2で起きると仮定すれば $L=\log_2(n)$ と見積もることができる。衝突の可能性が小さい例題になればなるほど軌道の分割がより少ない回数しか起きないので L は小さくなる。このように本アルゴリズムは「衝突の可能性」に応じて処理の詳細さが決まるので必要以上に詳しい計算をしないという点でサンプリング法よりも優れている。

4.3 実験結果

本アルゴリズムを、現在開発中のオフラインロボット教示システム[1,2]の衝突検出機能に適用して実験を行った。図8に示すような3つの軌道について実験を行い、実際に衝突が検出できることを確認した。本システムでは詳細な物体の3次元形状は B-reps を用いたソリッドモデルとして表現しているが、精度がそれほど要求されない場合には、複数の球の集合として表現した近似形状を使っている。球を用いることによって最小距離と最大移動長を非常に少ない計算量で求めることができるからである。

衝突検出に要した計算時間を図9に示す。この実験ではサンプリング法の軌道分割数を n = 128 とし、本アルゴリズムにおける $D_{c\circ l}$ も軌道長を n 等分した値に等しくしているので、両者は同じ精度の解を求めていることになる。本アルゴリズムでは軌道1、軌道2、軌道3の順に衝突の可能性が大きくなるにしたがって計算時間も長くなることがわかる。

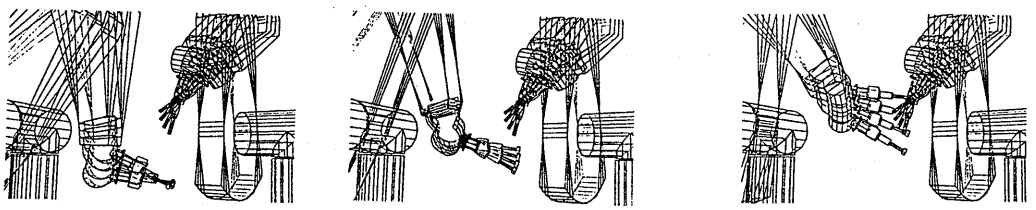
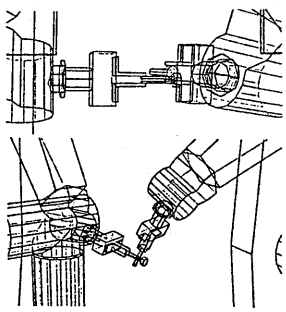


Fig. 8 Trajectories and a detected collision

	posibility of collision	computational cost			
		New Algorithm (L)		Sampling Method (I)	
		precise shape	rough shape	precise shape	rough shape
trajectory1	low	2	2	128	128
trajectory2	↓	5	5	128	128
trajectory3	high	15	19	58	53

Fig. 9 Computational cost depends on the possibility of collision



５．おわりに

本稿では計算機シミュレーションによって移動物体間の衝突を検出するための二つの方法を提案した。前半で示した方法は従来の交点計算法の考え方をもとに、さらに形状要素の３次元運動を時間の関数として一般的に表現することによって交点計算を行なうものである。これによってロボットの複雑な動作に対しても見逃しなく衝突が検出でき、衝突の発生時刻とその時の状態が算出できる。また、後半では衝突に関する条件の判別と軌道の２分割を再帰的に繰り返すことによって効率よく衝突を検出する方法を提案した。この方法では計算量は衝突の可能性に応じたものであり、必要以上に詳しい計算をしないので、平均的な計算量は軌道の分割数をnとしたとき、サンプリング法の I=n に対して本方法では $L=\log_2(n)$ と改善された。

本稿ではお互いに独立した二つの方法を提案してきたが、これらを適応の場に応じて使い分ける工夫も必要であろう。この点に関してはまだ詳しく検討していないので今後の課題として考えてゆきたい。

謝辞

本研究を進めるにあたり有益なご助言をいただいた、日本アイ・ビー・エム東京基礎研究所のDr. C.K.Chowと松家英雄担当に深く感謝いたします。

参考文献

1)Kawabe, Ishikawa, Okano, and Matsuka: Interactive Graphic Programming for Insustrial Robots, Proc. of 15th ISIR, pp699~706, 1985

2)Okano, Kawabe, and Yoshida: Task Planning System - Total System Approach, Proc. of CAPE86, pp.81~90, 1986

3)小沢邦昭，熊本健二郎，明石吉三，中田英樹：オフラインロボット教示における高速干渉チェックの一方法，日本ロボット学会誌4巻2号，pp.5-13, 1986

4)J.W.Boyse: Interference Detection Among Solids and Surfaces, Communication of the ACM, Vol.22, No.1, pp3-9, 1979

5)J.Canny, "Collision Detection for Moving Polyhedra," IEEE Transaction on Pattern Analysys and Machine Intelligence, Vol. PAMI-8, No. 2, pp 200-209, 1986

6)S.Cameron, "A Study of the Clash Detection Problem In Robotics", in Proc. of International Conference on Robotics and Automation, IEEE, 1985, pp 488-493

7)D.M.Esterling, J. Van Rosendale, "An Intersection Algorithm for Moving Parts", NASA Synposium on Computer Aided Geometric Modelling, 1984, pp. 119-123

8)S.Kawabe, A.Okano, K.Shimada: Collision Detection among Moving Objects in Simulation, The 4th. International Symposium on Robotics Research, 1987

9)嶋田憲司，岡野彰，川辺真嗣：ジトメトリックシミュレータのための衝突検出アルゴリズム，昭和６２年度精密工学会秋季大会学術講演会論文集，pp.273-204, 1987

10)Special Issue on Solid Modelling, IEEE CG&A, Vol.2, No.2, 1982

11)伊理正雄，藤野和建，"数値計算の常識"，共立出版，1985

12)IBM System/360 and System/370 IBM 1130 and IBM 1800 Subroutine Library-Mathmatics User's Guide, SH12-5300-1