制約論理プログラミングによるロボット動作計画*

3A−2

向井 理朗†　大和田勇人†　溝口文雄†

東京理科大学 理工学部†

1 はじめに

本稿で扱うロボット動作計画とは作業環境内に存在する障害物を回避しつつどのように初期状態から目標状態へ移動するかという障害物回避問題について扱う.

従来の障害物回避のアプローチとしてポテンシャル場を用いる方法とコンフィギュレーション空間を用いる方法[1]がある.ポテンシャル場を用いる方法では衝突を避けるためのポテンシャルと目標状態に達するためのポテンシャルを用いるが,ポテンシャルの極小点に迷い込む可能性がある.コンフィギュレーション空間(以下C-Space)を用いる方法では物体の姿勢を一意に定めるパラメータによって張られるC-Spaceを定義し,この上で経路探索を行なう.インプリメントが簡単でマニピュレータ全体の経路が求まる反面,C-Spaceの生成に膨大な計算量を要するという問題点がある.

ここでは,制約論理プログラミングの宣言性,モジュール性,数理的処理という特徴に着目し,コンフィギュレーション空間法に応用することにより経路を導出する方法を述べる.ここで使用した制約論理型言語は,CLP(ℜ)であり,対象としたロボットは回転型のマニピュレータである.

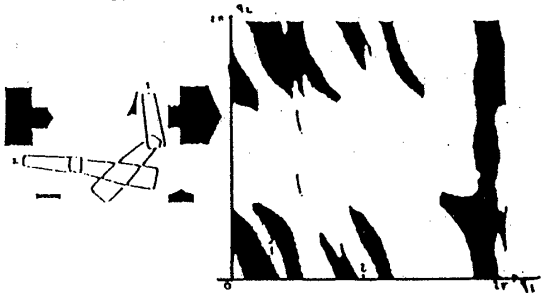

図1: コンフィギュレーション空間

2 本研究のアプローチ

関節角度をパラメータとするC-Spaceを定義する.C-Spaceを用いる経路計画の場合には必要最小限の自由空間が存在すればよい.そこでここでは簡易コンフィギュレーション空間を生成し,この上で経路を探索する.簡易コンフィギュレーション空間では干渉チェックの際に四分木を応用することにより,自由空間となる領域を導出する.マニピュレータの形状を定めるパラメータとしては関節の回転角度$[J_1, J_2]$を用いる.

2.1 簡易コンフィギュレーション空間

C-Space上に2つの空間を定義する.**自由空間**はマニピュレータが障害物と干渉をおこさない関節角度領域であり,**障害物空間**は干渉をおこす関節角度の組合せが含まれている関節角度領域である.ここで対象としているマニピュレータの性能上,動作領域を$-30 \leq J_1 \leq 110, -90 \leq J_2 \leq 0$と限定する.データベースの初期内容として,

```
obs([[400,400],[400,200],[600,400],[600,200]]).
free([]).
c_obs([[-30,-90,110,0]]).
```

があり,それぞれ作業空間内の障害物,C-Space上の自由空間,障害物空間を示す.

干渉チェックの効率化をはかるためそれぞれの関節角度を分割,動作領域を4つの領域に分けて干渉チェックを行なう.明らかに自由空間とわかる部分は分割しない.ところでC-Space上のセルはアームの移動する領域(非線形領域)を表す.制約論理プログラミングでは非線形制約については扱うことができない.そこで凸多角形に近似し,線形制約で表現する.

```
obs(X,Y,L) :- length(L,N),space(N,X,Y,L).
arm(X,Y,L,fail) :- length(L,N),space(N,X,Y,L).
arm(_,_,_,true).
space(N,X,Y,[[X1,Y1],…,[Xn,Yn]]) :-
        X=L1 · X1+…+Ln · Xn,
        Y=L1 · Y1+…+Ln · Yn,
        1=L1+…+Ln,
        L1>=0,1>=L1,…Ln>=0,1>=Ln.
```

obs/3とarm/4ではそれぞれ制約化され,その共通領域の有無によってarm/4にfail/trueを返す.space/4は凸多角形を表す線形制約であり,obs3, arm/4から呼び出される.

```
?- obs(List),obs(X,Y,List),arm(X,Y,List,Ans).
Ans = fail
*** yes
```

ここでのアルゴリズムは以下のとおりである.

1. 障害物空間(c_obs)を四分割しそれぞれについて自由空間であるかをチェックする.*cross_check*は分割された領域が自由空間であるのか障害物空間であるのかをチェックする述語である.この中で*cross_check*1では前述のobs/4, arm/4により障害物とマニピュレータを制約化し,その制約に共通部分がある(干渉する)時にfailを返す.このように*cross_check*は分割したそれぞれの領域が障害物空間となるか自由空間になるか判別するモジュールとなっている.

```
cross_check(Obs,[],[],[]) :- !.
cross_check(Obs,[[S1,S2,G1,G2]|Ps],F,C) :-
  cross_check1(Obs,[S1,S2],[G1,G2],fail),
```

*Motion Planning using Constraint Logic Programing.
†Toshirou MUKAI,Hayato OHWADA,Fumio MIZOGUCHI
†Science University of Tokyo

```
    C = [[S1,S2,G1,G2]|Cs],
    cross_check(Obs,Ps,F,Cs).
cross_check(Obs,[[S1,S2,G1,G2]|Ps],F,C) :- !,
    F = [[S1,S2,G1,G2]|Fs],
    cross_check(Obs,Ps,Fs,C).
```

2. 初期状態・目標状態が自由空間内に存在し，かつ自由空間の連続で結ぶことができれば，データベースを更新し経路探索を行なう．

3. 1で分割した領域のうち自由空間の部分は *free*/1 にそれ以外の部分は *c_obs*/1 に加えて，新しい障害物空間 (*c_obs*) について1を繰り返す．

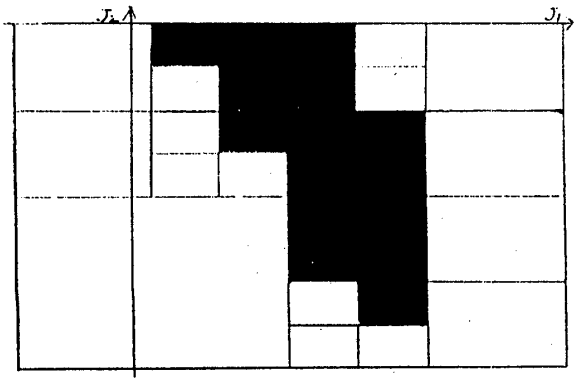

図 2: 簡易コンフィギュレーション空間

図2はこの方法によって形成された簡易コンフィギュレーション空間である．白い部分が導出された自由空間であり黒い部分が障害物空間である．

2.2 経路探索

簡易コンフィギュレーション空間上の自由空間と障害物空間の境界は少なくとも自由空間である．そこで，境界上にノードを設定し，経路探索を行なう．check_node はノード間を結ぶパスが障害物領域を通っていないかをチェックするものである．*inspace*/4 は凸多角形の内部を表す制約であり，干渉チェックの場合と同様のチェックを行なっている．

```
check_node(_,A,[],A).
check_node([X1,Y1],[X2,Y2],[Obs|Ob],A) :-
    inspace(4,X,Y,Obs),
    arm(X,Y,[[X1,Y1],[X2,Y2]],Ans),
    (Ans = true,check_node([X1,Y1],[X2,Y2],Ob,A);
     Ans = fail,A = []).
```

3 実行例

次のような初期状態，目標状態で実行例を示す．リストはそれぞれの状態の関節角度である．作業空間，障害物空間については前述のデータベースのものとする．

[初期状態] init([0,0]). [目標状態] goal([60,0]).

search/1 ではまずデータベースの内容を読み込み，次に make_cspace/6 において，干渉チェックを行なう．ここでは前述の cross_check を使って干渉チェックを行なっている．最後に，経路探索として check_node1/3) により，障害物領域と交わらないような経路が決定される．

```
search(Plan) : −free(F),c_obs(C),obs(Obs),
    make_cspace(Obs,C,FREE,F,Cs,[]),
    check_node1(FREE,Cs,Plan).
? − search([0,0],[60,0],Plan).
Plan = [[0,0],[5,−11.25],[40,−67.5],
    [57.5,−78.75],[75,−78.75],[75,−22.5],[60,0]]
* * *yes
```

search/3 という問い合わせに対し，まず簡易コンフィギュレーション空間を生成し，この上で初期状態，目標状態を結ぶ経路探索を行なう．出力として Plan を得る．リストの内容が，それぞれ途中姿勢を示すコンフィギュレーションである．

4 評価

C-Space を用いた動作計画の場合，最も効率に関係してくるのが干渉チェックである．制約論理プログラミングを使用した最大のメリットとしてその宣言性，数理処理能力が挙げられる．対象となる Object を宣言的にかつ線形制約で表現できるので，干渉チェックが容易である．本稿では C-Space 上の障害物領域を四分割してそれぞれを線形制約化し，干渉チェックを行なうことにより，明らかに自由空間とわかる領域の干渉チェックを最小限に抑えることができる．この点で Tomas の方法と比較して効率が良くなっている．

また，ここでは必要最小限の自由空間を持つ簡易コンフィギュレーション空間を採用している．C-Space 上の最短経路が必ずしも作業空間での最適経路となるとは限らない．障害物回避動作計画で必要なのは障害物と衝突を起こさない経路を導出することであり，そのためには簡易コンフィギュレーション空間でも十分であると考えられる．

5 おわりに

本稿では制約論理プログラミングの宣言性と数理処理による Object の線形制約化ということに着目し，従来のアプローチでは膨大な計算量を必要としていた干渉チェック・経路探索に応用し，生成した簡易コンフィギュレーション空間上での経路探索を行なうという方法を示した．

この方法は Tomas のコンフィギュレーション空間法のインプリメントが容易であり，マニピュレータ全体の経路が求まるという利点に加え，従来の問題点であった計算量の爆発を防いでいる．

参考文献

[1] Tomas Lozano-Pérez : A simple motion planning algorithm for general robot manipulators. AAAI '86

[2] Koichi Kondo : Motion Planning with Six Degrees of Freedom by Multistrategic Bidirectional Heuristic Free-Space Enumeration. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL7, NO.3, 1991*