〔解 説〕

ロボット言語のデータ構造*

橋 野 賢

1. はじめに

多自由度リンク機構から構成されるマニピュレータに作業を行わせる場合、いかに効率よくソフトウェア開発を行うかは研究者・技術者の大きな関心事である。基礎研究、システム研究を重視するアメリカは、ソフトウェアの蓄積において日本に数年先行しているが、ロボット言語においても1日どころか数日の長があることは認めざるを得ない。

ロボット研究者間の競争が今日のように熾烈になると、ソフトウェア開発の効率化には、とりわけ実動人員の少ない国立研究所においては必須の条件となってくる。世の中には実に多くのソフトウェア言語が存在するが、どれがロボット制御に適しているか、すなわちアクチュエータへ指令を送り、センサ信号によって外界を知り、座標変換等の数式処理を1秒間に数十回処理できる言語は何かを見極めなければならない。またコンピュータの処理速度が高速化しても、制御への要求はますます複雑になり、その速さを帳消しにするので、今後とも言語の開発とユーザにおける言語選択の悩みは続くであろう。

ロボット用言語に必要な機能を開発と保存に分けて考えると、まず開発時に必要な機能は高速処理が可能であること、開発効率が高いこと、すなわち単位時間にデバッグを含めて何行書けるかということである。一方保存においては、ドキュメンテーションの良さが絶対必要である。我々は、このような言語として FORTH, C, PASCAL を使用してきた。FORTH はドキュメンテーションに多少難はあるが、インタプリタ（最近コンパイラの FORTH もある）であるためデバッグをしながらその独特の辞書を構築することができる。そのため実に効率の良いソフトウェア開発を行うことができる。FORTH の基本概念は、新井らによってロボット中間言語²⁾として採用されている。PASCAL は構造化プログラミングに適していること、RECORD というフレキシブ

ルなデータ構造をもつこと、POINTER をドキュメンテーション良く使うことができることによって、ロボットの制御に用いられている。C は外見は PASCAL に非常に良く似ているが、より細かな作業を行うことができる。ミニコンピュータ VAX の基本 OS である UNIX が C で書かれていることから、アメリカのソフトウェア界のすう勢は C に傾いている。しかし UNIX はオーバーヘッドが大きいので、クリティカルな実時間制御には適していないといわれている。

AL, AUTOPASS, LAMA 等については、分り易い解説があるのでそれを参考にしていきたい³⁻⁹⁾。ここでは、ロボット言語そのものではないが、高級言語を使用したデータ構造の例として PASCAL をとりあげる。また最も新しいロボット言語として IBM の AML について解説する。

2. PASCAL によるデータ構造

Purdue 大学の Paul らは、マニピュレータの制御言語として PASCAL を使用してきた⁹⁻¹¹⁾。現在は PASCAL から C へ書き直されているが、基本的には同じ内容であるので、ここではドキュメンテーションの良い PASCAL を用いて、主に座標変換のデータ構造について見てみよう。

図1 ベルトコンベア上のピンをブロックの穴に挿入する作業における手先の位置¹¹⁾

原稿受付 1984 年 1 月 19 日

* Data Structure of the Robot Language

ここで想定する作業は、ベルトコンベア上のピンをマニピュレータでつかみ、作業台の上にあるブロックの穴に挿入するものである。一連の作業における手先の位置を、図1に示すようにP1からP8で表す。位置 P_i ($i=1\sim 8$)への移動と指部の作業は次のプログラムで表される。

```
MOVE P1: ピンへ接近
MOVE P2: ピン上へ移動
GRASP: ピンを把握
MOVE P3: 垂直に持ち上げる
MOVE P4: 穴へ接近
MOVE P5: 穴に接触して止まる
MOVE P6: ピンを穴のZ軸に一致させる
MOVE P7: ピンを挿入する
RELEASE: ピンをはなす
MOVE P8: 退却
```

図2 Transformの構造

ここではP1への接近についてのみ述べる。Zを固定座標系から見たマニピュレータ基底部の変換行列、T6をZ座標系から見た手先の変換行列、Eを手先座標系から見たツールの変換行列、Pを固定座標系から見たピンの変換行列、PAをP座標系から見たピンにアプローチすべき点の変換行列とすると、P1への移動は次に示す行列の関係式で表現される。

$$P1: _ _ Z \quad T6 \ E = P \quad PA \quad (1)$$

2.1 データ構造の定義

ここで2つのデータの型を定義する。一つはベクトルであり、今一つは変換行列である。ベクトルはPASCALを用いると次のように型宣言により3つの実数の集まりとして定義される。

```
type vector=record x,y,z: real end;
```

変換行列は4つの vector, すなわち方向を示す3つの vector と位置を示す1つの vector によって同様に定義される。

```
type transform=record n,o,a,p: vector end;
```

さらに、逆行列を指すポインタと実際に行列の値を計算するファンクション名も、この transform の中に入れると次のようになる。

```
type transpointer=↑transform;
transform=record n,o,a,p: vector;
inverse: transpointer;
fn: functionname;
```

```
end; {transform}
```

これをメモリ上に展開すると図2になる。

先に述べた変換行列 Z, T6, E, P, PA はこの transform を用いて定義される。

```
var Z, T6, E, P, PA: transform;
```

var は変数であることを示す。

2.2 変換行列式のデータ構造

関係式(1)は手先がピンのアプローチ点にあるときの幾何学的関係を示している。この関係をコンピュータのメモリ上で展開するためにポインタのレコードを用意する。

```
type termpointer=↑term;
term=record
  nxt, inv: termpointer;
  trans: transpointer
end; {term}
```

各 term という record は各変換行列に対応しており、nxt は関係式(1)の両辺において、右側にある変換行列のレコードの位置を指す。両辺の右端のE及びPAのそれはそれぞれ PA^{-1} , E^{-1} を指すレコードの位置を指す。両辺の左端の逆行列 Z^{-1} , P^{-1} においては、同様にP,Zを指すレコードの位置を指す。invはそれぞれ逆行列に対応するレコードの位置を指す。このようにして結合したのが図3である。図中Z, T6等は図2に示すデータ構造の位置を指す。また図3では、図を見易くするため inv と trans の位置が逆になっている。各 P_i ($i=1\sim 8$) に対してこのようなデータ構造が対応するからそれを識別するポインタが必要になる。

```
type positionpointer=↑position;
position=record
  t6ptr, tolptr: termpointer;
  righty, flip: boolean
end;
```

このようにして図3に示すP1に対する完全なデータ構造ができて上がる。

2.3 データ構造を用いての変換行列の解法

視覚装置またはティーチングでPが求まると、PAは自動的にきまる。マニピュレータの設置場所とツールの位置姿勢が決まれば、関係式(1)はT6について解くことができる。T6が計算できればヤコビアンを用いて各関節角を求めることができ、それをマニピュレータに出力することによって、手先位置をP1に持ってゆることができる。ここでT6を先に述べた構造を利用して計算してみる。

```
procedure solvet6 (var sl: solvelist;
  var n: ntrans; pp: positionpointer);
```

図 3 変換行列式の完全なデータ構造¹¹⁾

```
var p : termpointer ;
begin {solvet 6}
  p:=pp↑·t6 ptr↑·inv↑·nxt ; {the tail of t6}
  n:=0 ;
  while p↑·inv↑·trans < > t6 do begin
    n:=n+1
    sl[n]:=p↑·trans ;
    p:=p↑·nxt end
  end ; {solvet 6}
```

手続き名の次のカッコ内は引数であり、var は出力変数、var がないのは入力変数である。P はローカル変数である。{ } 内はコメント文である。pp↑·t6 ptr↑·inv↑·nxt は図 4 の㉑である。p↑·inv↑·trans は図 4 の㉒である。これは T6 に等しくないで n:=1, sl[1]:=Z⁻¹, P は更新されて㉑となり、while 文の式が t6 と等しくなるまで n と sl[n] の値は更新される。結局 solvet 6 (sl,n,p1) の解は n=4, sl[1]=Z⁻¹, sl[2]=P, sl[3]=PA, sl[4]=E⁻¹ となる。移動に必要な加速度、時間等を指示する mode と呼ばれるレコードを加えて、実際のプログラムにおいては move(p1,m1) と書かれる。ピンを 2 個ブロックの穴に挿入する作業は次のように書かれる。

```
for i:=1,2 do begin {ピンは 2 個}
  read(camera,pc) ; {ピン位置をカメラで入力}
  matrixmultiply(p,cam,pc) ; {ピン位置をセット}
  move(p1,m1) ; {ピンへ接近}
```

図 5 AML で制御される組立用
マニピュレータ RS-1¹²⁾

ロッピーディスク各1台を装備している。ティーチングは携帯用ペンダントを用いて行うことができる。

3.2 AML の特徴

AML は^{12,13)}ドキュメンテーションの良い、また構造化プログラミングが可能な Pascal-like な言語である。またインタプリタであるため容易にデバッグを行うことができる。他の高級言語の特徴のうち、ロボット言語に有用な機能は積極的に取り入れられている。AML にはデータ構造に適した aggregate と呼ばれる型がある。これは PASCAL の record と同じく、どんな型のデータも混在できるものである。また aggregate の要素として aggregate も指定でき、原理的には無限次元のデータ構造をとることが可能である。次に aggregate のいくつかの例を示す。

```
<1, 2, 3, 4>
<'FEEDER-3', 455, <5.25, 14.75, 2.375>>
<'A', 'B', 'C'>
<> ...空 aggregate
```

いま A が次に示す値をもつ aggregate とする。

```
<1, <2, 3>, <4, 5, 6>, <7, <8, 9>>>
```

各要素は次のように表される。

```
A(1)=1
A(2)=<2, 3>
A(4)(2)(1)=8
A(<1>)=<1>
A(<>)=<>
```

他の高級言語の機能の借用の例は aggregate に関したものが多く、次に例をいくつか示す。INTA(n) は $\langle 1, 2, \dots, n \rangle$ という aggregate を作り出す。これは駆動すべき関節数を増減するとき等に有効である。また SELECT という命令は $\text{SELECT}(\langle 1, 0, 1 \rangle, \langle 2, 3, 4 \rangle) \rightarrow \langle 2, 4 \rangle$ のように 2 番目の引数の要素を 1 番目の引数で取捨選択して新しい aggregate を作ることができる。これはある関節だけ停止したい時等に使用できる。以上は APL からの借用であるが、LISP のリスト処理に似た機能もある。

```
<1, 2, 3> # <4, 5> → <1, 2, 3, 4, 5>
```

文字については、

```
'THIS_IS' | ' _AML' → 'THIS_IS_AML'
```

これはエラーメッセージの出力に有効であろう。

最後に aggregate の演算と代入に関する規則を示す。aggregate の演算則

1. $\langle U_1, \dots, U_n \rangle \circ \langle V_1, \dots, V_n \rangle = \langle U_1 + V_1, \dots, U_n + V_n \rangle$
2. $\langle U_1, \dots, U_n \rangle \circ S = \langle U_1 + S, \dots, U_n + S \rangle$
3. $S \circ \langle V_1, \dots, V_n \rangle = \langle S + V_1, \dots, S + V_n \rangle$

ここで $\circ$ は演算子, S はスカラーである。

aggregate の代入則

1. $S = \langle V_1, \dots, V_n \rangle$ if $n=1$ then $S = V_1$
else error
2. $\langle V_1, \dots, V_m \rangle = S$ $\langle V_1 = S, \dots, V_m = S \rangle$
3. $\langle U_1, \dots, U_m \rangle = \langle V_1, \dots, V_n \rangle$ if $m \leq n$
then $\langle U_1 = V_1, \dots, U_m = V_m \rangle$
else $\langle U_1 = V_1, \dots, U_n = V_n, U_{n+1}, \dots, U_m \rangle$

他の言語に比べて柔軟性に富んでいることがわかる。

3.3 サブルーチンの特徴

AML の今一つの大きな特徴は、コマンドを含めた全てのサブルーチンが数または aggregate でリターンすることである。また、仮引数と実引数の型を合わせる必要がない。以上のことが、プログラムを書き易くまた柔軟性に富んだ強力なものにしている。MUL というサブルーチンを次のように定義する。

```
MUL: SUBR(A, B);
      RETURN(A*B);
      END;
```

実引数が整数の時 $\text{MUL}(1, 2) = 2$

実引数が aggregate の時, $\text{MUL}(\langle 1, 2 \rangle, \langle 3, 4 \rangle) = \langle 3, 8 \rangle$ となる。これらの特徴はベクトル、行列の計算の多いロボット制御に適している。

AML において変換行列は aggregate を用いて次のように表現される。

```
<p<n, o, a>
```

p は位置ベクトル, n, o, a の方向ベクトルである。RS-1 の手先位置は次のようにして求めることができる。

```
XYZ: STATIC <1, 2, 3>; 肩部の関節番号
RPW: STATIC <4, 5, 6>; 手首関節番号
HANDLEN: STATIC 8.0; 指の長さ
HANDFRAME: SUBR;
      R: NEW EULERROT(QPOSITION(RPW));
      RETURN(<<QPOSITION(XYZ) - HANDLEN *
              R(3), R>>);
      END;
```

初めの3行は、サブルーチン HANDFRAME で使用

図 6 RS-1 の手先位置計算¹²⁾

する定数の定義である。Rはローカルな変数であり、その型は NEW と；の間の式によって決まる。QPOSITION は実引数 RPW 即ち手先の 3 関節角の現在値を度単位で求めている。この数値は EULERROT の実引数であり、Rは 3 次元ベクトルを要素とする 3×3 行列になる。Rの各要素は良く知られた直交、方向、アプローチベクトルである。RETURN はカッコ内の値をサブルーチンの値としてコールしたルーチンに返すものである。QPOSITION (XYZ) は図 6 に示した 3 次元空間座標を表す。サブルーチン HANDFRAME の値は図 6 の位置と方向を表す 4×4 行列となり、 $\langle P \langle R_x, R_y, R_z \rangle \rangle$ と表現される。

3.4 AML を使用した作業例

ナットのパレタイジングを例にとってプログラムの実際を見てみよう。パレットには 30 本のロッドが立っており、各ロッドに約 8 個のナットをさし込むことができる。ナットは、幾何学的にゆるく拘束されており、振動による位置ずれを防止している。一つのパレットからナットを 1 個つまみ上げ、別のパレットのロッドに挿入する一連の作業を図 7 に示す。図では手先位置の動きを模式的に示した。POS(1) でロッドの真上に接近し、次にナットの位置 POS(2) へ接近する。指の力感覚を利用してナットの正確な位置がわかれば POS(3) で把握し、POS(4) に移動する。POS(4) と POS(1) は同一地点である。POS(5) は別のパレットの挿入すべきロッドの

図 7 パレタイジング作業における手先の動き

真上を指す。POS(6) はロッドの上端よりやや下の位置であり、POS(7) で力感覚を利用してナットをロッドに密着させ、POS(8) までロッドに沿って移動する。ナットがおさまれば POS(9) で指を開き、ロッドの真上 POS(10) に移動し、次のナットを把握するために更新された POS(1) に移動する。

AML に用意されている移動命令は MOVE, AMOVE, DMOVE の 3 種類があり、次のように表される。

MOVE (joints, goals, monits, $\langle \text{spd, accel, decel, settle} \rangle$)

他の移動命令も同じ引数をもつ。joints は駆動すべき関節番号の aggregate, goals は目標点で joints と同じ次元数の aggregate, monits は移動停止のための力感覚等のモニタの指定, spd, accel, decel は速度、加減速の指示, settle は目標点での位置精度の指示を行うものである。DMOVE における goals は現在値からの差分である。MOVE は目標点に達した後、次のステップに進むが、AMOVE はサーボに指示を出力後、直ちに次のステップに進み、WAITMOVE で移動の終了を待つ。

パレタイジング作業ソフトウェアの核の部分を図 8 に示す。POS(1) から POS(4) までの前半と、POS(5) から POS(10) までの後半に大別できる。パレット間の大きな移動 (POS(1), POS(5)) を利用して、前半では後半の、後半では前半の目標位置を計算する (DESTPOS (J), STATPOS(I+1))。そのため POS(1), POS(5) への移動は AMOVE 命令を使用する。力の強い油圧マニピュレータにおいては、異常時の移動停止のためにモニタの設定が重要である。本プログラムでは移動の前後に FOREOP, MONSET, AFTEROP というモニタに関するコマンドテーブルを作成し、適宜 EVAL 命令によってストリングで書かれたコマンドを実行する形式をとっている。POS(1) から更新された POS(1) までの一連の作業は約 8 秒の実行時間を必要とする。POS(1), POS(4), POS(5), POS(10) など精度を必要としない通過点に対して、別途通過点 MOVE 命令が加われば、更に高速になる。

4. お わ り に

PASCAL と AML についてデータ構造を中心に解説した。ロボット言語が研究者の手持ちのコンピュータシステムに載り、それによってロボットが動作すれば、ロボット言語そのものの研究も飛躍的に進歩するであろう。マイコンの代表的 OS CP/M と同じように、入出力部にユーザ定義が可能な柔軟性をもったロボット言語用 OS の開発が切望される。

```

EVAL(MONSET(1));
AMOVE((JX,JY,JZ,JW,JG), POS(1), MONABN, SPD(1));
DESTPOS(J);
N= 2;
WAITMOVE;
REPEAT
  BEGIN
    EVAL(FOREOP(N));
OMRTRY:
    EVAL(MONSET(N));
    DMOVE((JX,JY,JZ,JW,JG), POS(N), MON, SPD(N));
    EVAL(AFTEROP(N));
    END
  UNTIL (N= N+1) GT 4;
--
EVAL(MONSET(5));
AMOVE((JX,JY,JZ,JW,JG), SELECT((1,1,1,1,0),POS(5)), MONABN, SPD(5));
STATPOS(I+1);
N=6;
WAITMOVE;
REPEAT
  BEGIN
    EVAL(FOREOP(N));
OMRTRY1:
    OMRTRY= OMRTRY1;
    EVAL(MONSET(N));
    DMOVE((JX,JY,JZ,JW,JG), POS(N), MON, SPD(N));
    EVAL(AFTEROP(N));
    END
  UNTIL (N= N+1) GT 10;
ENDFORCE= 0;
END;

```

POS(1)へ移動
 POS(2)
 POS(4)へ移動
 POS(5)へ移動
 POS(6)
 ~
 POS(10)
 へ移動

図 8 バレタイジングプログラムの核の部分

参 考 文 献

- 1) 前田, “マイクロコンピュータソフトウェア開発の実践的技法・ノウハウの全て. ソフトウェア編” pp.238, 日本システミックス
- 2) 新井, 松元, “ロボット制御システムにおける中間言語の形式”, pp.721-722, 昭和 58 年度精機学会秋季大会
- 3) 井上, “環境の理解と行動”, 情報処理, pp.978-985, 19 (10), 1978
- 4) S.Bonner, K.G.Shin, “A Comparative Study of Robot Languages”, Computer, pp.82-96, 1982(DEC)
- 5) 新井, “ロボット言語—記述形式と知的レベル—”, 計測と制御, pp.14-19, 21(12), 1982
- 6) “特集—ロボット言語”, ロボット, No.39, 1983 (May)
- 7) “CAM-I Proposes Standards in software”, The Industrial Robot, pp.252-253, 1982 (Dec)
- 8) 新井, “ヨーロッパにおけるロボット言語開発の動向”, オートメーション, pp.27-31, 27(4)
- 9) K.Takase, R.P.C.Paul, “A Structured. Approach to Robot Programming and Teaching”, IE³ Trans. on Systems, Man, Cybernetics, pp.274-289, SMC-11(4), 1981
- 10) R.P.C.Paul, “WAVE-A model based language for manipulator control”, The Industrial Robot, pp. 10-17, 4(1), 1977
- 11) R.P.C.Paul, “Robot Manipulators”, MIT Press, 1982
- 12) R.H.Taylor, D.D.Grossman, “An Integrated Robot System Architecture”, Proceedings of the IE³, pp. 842-856, 71(7), 1983
- 13) R.H.Taylor, P.D.Summers, AML: A Manufacturing Language”, The International Journal of Robotics Research, pp.19-41, 1(3), 1982

橋野 賢 (Satoshi HASHINO)

昭和 22 年 4 月 5 日 福岡県生れ. 昭和 48 年九州大学理学部 物理系大学院修士修了. 同年通産省機械技術研究所入所. 生産工学部ロボット工学課に在籍, 現在に至る. 昭 57~58 米国バデュー大学客員研究員.

日本機械学会, バイオメカニズム学会の会員.

(日本ロボット学会正会員)