解　説

環境の理解と行動*

井上博允**

1. まえがき

人間のように巧みに動く知的な機械を作ることは，長い間，技術者達が持ち続けてきたひとつの夢であった．コンピュータの発達は，このような動機に有力な道具を提供した．三次元の環境を理解する視覚，巧妙に行動する機械の手足，そして知的に考える頭脳，これらをコンピュータを使って実現してみようという試みが具体的に検討され始めたのは 1960 年頃である．

コンピュータ制御の機械の腕は 1962 年に，テレビカメラで見たつみ木をコンピュータで認識させる研究は 1963 年に，はじめて報告された．1960 年代末から 1970 年にかけて，世界のいくつかのグループが，テレビカメラで物体を見て，それをハンドリングする，ハンド・アイ・システムを開発した．これらの初期の知能ロボットの環境は，つみ木の世界に限られていたが，数々の興味深い試みが行われていた．

インスタント・インサニティ，つまり，各面が赤青緑白の 4 色のどれかで塗ってある立方体 4 個をうまく並べて柱を作り，柱の各面に 4 色が 1 度ずつ現れるようにするパズルを実行したロボット（スタンフォード大学），つみ木の構造物を見て，そのコピーをつみ木で作った例 (MIT)，三面図を読取ってつみ木を行った例(日立中研)，視覚によるフィードバックを使って小さな四角の箱に 4 角柱をぴったり組合わせた例（電総研），等々，当時のロボットがテレビカメラの眼と機械の腕で行った仕事の例である．

米国における知能ロボット開発の背景には，人間のもつ知的能力を機械的に実現し，コンピュータをよりスマートにしようという人工知能研究の大きな流れがあった．チェス等のゲーム，数式処理，定理の証明，自然言語の解釈，画像の解釈などへの，コンピュータを用いた挑戦は，1950 年代末に始まっていた．知能ロボットは，これらの広汎な人工知能研究から派生したひとつのトピックだったのである．

1970 年頃から，産業用ロボットの開発と相前後して，我国でもロボットの研究は増加した．コンピュータ技術の歴史の差，使用できる設備の規模と様式の違いもあって，我国のロボットは，一般に応用志向の色彩が強く，人工知能との関連が薄かった．しかし，我国における産業的応用を志向したロボットの研究・開発の急激な進展は，逆に，米国や西欧を刺激することとなり，1972, 3 年ごろから，アドバンスト・オートメーションの研究にも力が注がれるようになったのである．

知能ロボットの多くの部分は，視覚の研究に関連が深い．視覚については，本特集号に，物体・光景の認識と理解という別の解説記事が予定されている．本稿では，環境の理解と行動といういただいた題のもとに，ロボットの行動，とりわけ，ロボットの作業を記述するための高水準言語に関する最近の研究に焦点を絞って解説し，あわせて今後の課題について考えてみることにしたい．

2. AL[1]

スタンフォード大学 AI ラボでは，知能ロボットのプロトタイプ開発以来，腕の解法の研究，腕の設計，ハンド・アイ・システムによるインスタント・インサニティ・パズルの実行，腕の軌道計画とその制御，ビジュアル・フィードバック，ポンプの組立，アセンブリ言語タイプのプログラミング・システム WAVE[2]，2 本の腕による蝶番の組立，等々，多くの研究が蓄積されていた．AL は，これらの研究の蓄積の上にたって，プログラマブル・オートメーションの将来の姿を研究するために設計され開発されたプログラミング・システムである．

2.1 AL の設計思想

スタンフォード大学 AI ラボでは，Algol をベース

* Intelligent Robots that Work in Real World by Hirochika INOUE (Faculty of Engineering, University of Tokyo).

** 東京大学工学部

とし，連想3つ組の機能 Leap を組込んで拡張した言語 SAIL (Stanford AI Language) を開発し，主に SAIL を用いて研究を行ってきた．AL のコンパイラも SAIL で書かれており，Algol タイプのソース言語となっている．AL は，次のような考え方に基づいて設計されている．

（1） Algol タイプのプログラム制御構造を基本とし，プログラムの構造化を行いやすいように配慮されている．また，汎用のテキストマクロ機能をコンパイラに組込んで，一般的かつフレキシブルなプログラミングを可能としている．

（2） データのタイプには，通常のスカラー量だけでなく，マニピュレーションが3次元の世界を扱うという特殊性を考慮して，ベクトル，回転変換，座標系などの3次元の性質を組込み，算術演算子もこれらを扱い易いように拡張する．

（3） 運動の制御には，Paul の軌道計画法を採用する．感覚条件等のモニタリング，運動中の力の指定等は節形式で記述する．作業の計画と実行は分離し，前者はコンパイル時に行い，後者はランタイム・システムで行う．なお，前者は大型の TS 上で走り，後者は専用のミニコンピュータで実行される．

（4） 作業計画時に参照する環境を表す変数は，作業計画の進行と共に構造的に変化していく．環境を表すデータベースの管理を簡単かつ矛盾なく行うために，セマンティックな AFFIX, UNFIX という概念を導入し，結合部品のひとつを動かした場合に，他の部品のデータも正しく更新され管理されるようにする．

（5） 高水準の動作記述への拡張性を重視する．このため，条件付マクロ展開，反復マクロ展開等の機能を組込み，一般的なマクロまたはライブラリとして書かれたプログラムから，環境データベースを使って，それが使われる場面に適した効率的なコードを生成できるようにする．

（6） 2本以上の手の動作の同時実行を記述するための COBEGIN, COEND ペア，前提条件 PREREQUISITE の記述を利用したサブタスクの順序の自動割付け，プロセス間の同期をとるための EVENT, SIGNAL, WAIT 等を導入して複雑な動作の記述を可能にする．

（7） ランタイム・システムでは，コンパイラで生成されたコードを実行する．腕の各軸のソフトウェアサーボ，および，条件のモニタリング等，実時間で多くのプロセスを同時に実行させるために，CPU 時間をスライスして使用する．計画された軌道のある程度の修正はランタイム・システムで行わせる．

（8） 全体として，プログラミング，デバッギングを行い易いように，インタラクティブなシステム構成とする．

2.2 腕の運動の計画と実行

AL は，Paul の軌道計画によるマニピュレータ制御法[3] に基づいて，運動の記述を行う方針で設計されている．この方法は，マニピュレータを，なめらかに，速く，精度よく動かすよう工夫された方法で，運動の始点と終点の間に，必要に応じていくつかの経由点を設定し，これをつなぐなめらかな軌道を計画し，各種の補償を行ってこの軌道を正確に実現しようというものである．

運動の始点や終点は，普通，テーブルや他の物体に近いので，不用意にマニピュレータを動かすと，テーブル等にぶつかることがある．このような衝突を避けるために，**図-1** のように，始点(I)と終点(F)の近くに，安全な departure 点(D)と approach 点(A)を設定し，ここを必ず通過するように軌道を計画する．対象物に接近したり，離れたりする場合，これらの点は，対象物に相対的に付加された点を考えた方がすっきりする．これを deproach (**de**parture, ap**proach**) 点と呼び，対象物のひとつの属性と考え，対象物に固定された点とみなす．

始点，departure 点，approach 点，終点等が決まると，まず，これらの点（位置・姿勢）をマニピュレータの関節角に変換する．それを各関節別にプロットすれば，図-1 右のようなグラフが6枚得られる．各関節とも，これらのプロットを時間 t の多項式を用いてなめらかにむすぶ．すなわち，始点と終点で速度と加速度が0，途中の点では，速度と加速度が連続になるように，これらの点を通過する軌道を計画するわけである．

このようにして計画された軌道を正確に制御するた

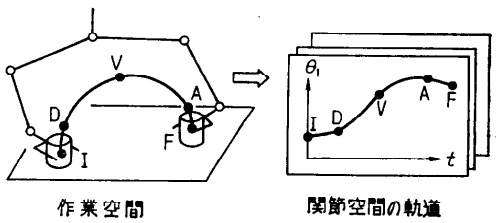

図-1 腕の運動軌道の計画

めには，重力の影響，各サーボの等価慣性モーメントの変化の影響，加速力の効果などを補償することが必要である．Paul は，これらの補償量を，マニピュレータの数学モデルに基づいて，前以って計算で求めておくことを考えた．軌道の計算と補償量の計算は，かなり大量の計算を要し，計算時間もかかるから実時間では無理である．AL では，軌道と補償量の計算はコンパイラの最終段階で行い，そのデータに基づいてランタイムシステムで，サーボ動作を実行させるという方法をとったわけである．

2.3 腕の運動の記述

AL のソース言語では，運動は，基本的に次のような形式で記述される．

MOVE YELLOW TO finalpoint VIA int1, int2

この記述をもとに，コンパイラは，YELLOW という腕の現在位置から，departure 点を通り，さらに中間点 int1, int2 を通過し，approach 点を経て finalpoint へ動く軌道を計画する．departure 点，approach 点が陽に指定されていない場合には，環境のデータベースを探して，deproach 点を決定する．もちろん指定されている場合にはこの限りではない．また，中間点では，そこを通過するときの時間や速度を指定することもでき，それらのデータは軌道計画の際に使われる．

```
MOVE YELLOW TO finalpoint
  VIA int1 WHERE VELOCITY=v1
      WTH DURATION=2*SEC
  VIA int2 WHERE VELOCITY=v2
      WITH DURATION≧8*SEC
```

動作中にモニタすべき条件は ON 節で指定する．

```
MOVE YELLOW TO ypark
  ON DURATION≧3*SEC DO warning←1
  ON FORCE (v1)≧18*OZ DO STOP
```

第1の ON 節は経過時間をモニタし，3秒以上たったら warning という変数に1をセットする．第2の ON 節は力をモニタし，v1 方向の力が 18 オンス以上になったらアームを停止させる．

動作中の力や全動作時間は WITH 節で指定する．

```
MOVE YELLOW TO @
  WITH DURATION=10*SEC
  WITH FORCE=−18*OZ ALONG Z OF
       station
  WITH FORCE=0 ALONG X, Y OF station
```

このプログラムは，テーブルの X, Y 軸方向の力を0に保ち，Z方向に −18 オンスの力で 10 秒間押えつける動作を意味する．なお@は，腕の現在位置を示す記号である．

このほかにも AL にはいろいろなオプションが組込まれ，こみ入った動作をも記述できるように設計されている．詳細については文献1) を参照されたい．

2.4 環境の記述

具体的に作業を計画し，腕の軌道の計算を行うためには，環境を表現したデータベースが必要である．AL では，3次元環境の記述を容易にするために，VECTOR（3次元ベクトル），ROT（回転マトリクス），FRAME（座標系を表わす．原点の位置ベクトルと回転マトリクスからなる．），PLANE（平面を表わす），TRANS（座標変換マトリクス）などのデータタイプが導入され，それらのデータを作るための同名の関数が用意されている．

図-3 のような環境を AL で記述した例を図-2 に示す．作業の計画が進行するにつれて，環境は変化していく．それに伴って，対象物を表す FRAME の値も変化する．そのとき，関連のある FRAME の値に矛盾が起らぬように，環境を表すデータベースは正しく

```
FRAME beam, beam_hole;
FRAME bracket, bracket_hole, bracket_grasp;
FRAME bolt;
beam ← FRAME(ROT(Z,90*DEG),VECTOR(10,6,0));
beam_hole ← beam*TRANS(ROT(X,-90*DEG),VECTOR(3,0,7));
AFFIX beam_hole TO beam;
ASSERT FORM(DEPROACH,beam_hole,TRANS(NILROT,VECTOR(0,0,-3));
bracket ← FRAME(ROT(Z,45*DEG),VECTOR(20,14,0));
bracket_hole ← bracket*TRANS(ROT(180*DEG),VECTOR(3,3,0));
AFFIX bracket_hole TO bracket;
bracket_grasp ← bracket*TRANS(ROT(X,180*DEG],VECTOR(0,3,3));
AFFIX bracket_grasp TO bracket RIGIDLY;
bolt ← FRAME(ROT(Z,90*DEG)*ROT(X,180*DEG),VECTOR(16,30,5));
```

図-2 AL における環境の記述例

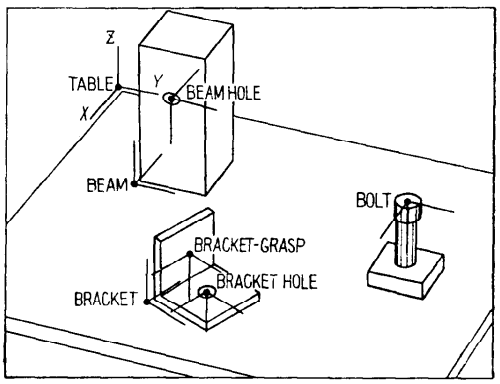

図-3 作業環境の初期状態

管理されなければならない．例えば，柱の FRAME を動かした場合には，柱の穴の FRAME の値も正しく変化しなければならない．図-2 の環境記述の中の 6, 10, 12 行目の AFFIX 文はこのような意味上の固定関係を指定する．FRAME のひとつが変更されたとき，他の FRAME の値も自動的に正しく変更されるというわけである．

2.5 作業プログラムの例（その1）

AL のソースプログラムで，作業は一体どんな具合に記述されるのだろうか，簡単な作業のプログラムの例を示そう．図-3 に示すように，テーブルの上に，柱とL型金具とボルトが置かれている．柱の穴とL型金具の穴を合わせ，そこにボルトをさしこむという作業を例にとる（ただし，プログラムにはねじ込む動作は含まない．ねじ込みは，後で，電動ドライバ実行されるから）．

作業の手順を直接書き下したプログラムの例を**図-4** に示す．このプログラムでは，すべての対象物は図-2 で記述された通りの正しい位置にあり，腕は十分正確に制御されるものとして書かれている．動作ミスが生じたときの対策は一切準備されていない最も単純なプログラムである．

腕は黄色のものと青色のものと2本ある．それぞれを YELLOW, BLUE と呼ぶことにする．

（イ） YFINGER すなわち YELLOW の指を 3 cm 開く．

（ロ） YELLOW を bracket-grasp という位置へ動かす．

（ハ） CENTER とは指を閉じて対象物をつかむことである．触覚を利用して指は対象物の位置に適応し，開閉幅の中央でつかむので，この動作を CENTER と呼ぶ．

（ニ） YELLOW は bracket をつかんだ．以後 YELLOW が動けば bracket も共に動く．従って，bracket を YELLOW に AFFIX して，FRAME の値を正しく管理させる．

（ホ） WRT は with respect to の省略形．

（ヘ） BLUE は bolt をつかんでいる．bolt を beam-hole に 5 cm 挿入する．このとき，BLUE の X, Y 方向の力を 0 にして穴の位置に適応させる．Z 方向の力が 60 オンス以上になったら，BLUE を停止させる．

（ト） YELLOW は指を開いて bracket を離す．bracket と YELLOW は互いに自由になるから（ニ）で固定した AFFIX の関係を解く必要がある．また bracket は beam に組付けられた．だから，bracket を YELLOW から UNFIX し，beam に AFFIX する．

```
example:BEGIN
  "world model description here"
  OPERATE YFINGERS WITH OPENING=3*CM;   ……(イ)
  MOVE YELLOW TO bracket_grasp;        ……(ロ)
  CENTER YELLOW;                       ……(ハ)
  bracket_grasp ← YELLOW;   }
  AFFIX bracket TO YELLOW;  }……(ニ)
  MOVE bracket_hole TO beam_hole;
  OPERATE BFINGERS WITH OPENING=3*CM;
  MOVE BLUE TO bolt;
  CENTER BLUE;                                          (ホ)
  bolt ← BLUE;                                        ……
  AFFIX bolt TO BLUE;                                 ……
  MOVE bolt TO beam_hole + VECTOR(0,0,-5.3) WRT beam_hole;
     WITH FORCE=0 ALONG X,Y OF BLUE                 }
     ON FORCE(Z WRT BLUE) > 60*OZ DO STOP BLUE;     }…(ヘ)
  OPERATE YFINGERS WITH OPENING=3*CM;   }
  UNFIX bracket FROM YELLOW;            }
  AFFIX bracket TO beam;                }……(ト)
  MOVE YELLOW TO YPARK;                 }
  OPERATE BFINGERS WITH OPENING=3*CM;
  UNFIX bolt FROM BLUE;
  AFFIX bolt TO beam;
  MOVE BLUE TO BPARK;
  WRITE("Finished");
END example;
```

図-4 AL による作業プログラムの例（その1）

2.6 ライブラリ・マクロルーチン

図-4 に示した例は，すべてがうまくいくという仮定の上にたった，エラー対策のない単純なプログラムの例であった．しかし，現実には，失敗やアクシデントはつきものである．エラー対策を考慮に入れると，同じ仕事でもプログラムは大変複雑になる．プログラムの手間を省くために，度々使用される基本的な動作は，汎用のライブラリとして登録しておくのが得策であろう．AL では，高度のテキストマクロの機能をコンパイラに組込む方法で，より高水準の記述へと発展させ得るように配慮されている．ライブラリと言っても，単にサブルーチンを呼んでくるというのではなく，種々の状況を考慮に入れて書いてある一般的なマクロルーチンを呼びだして，そのときの環境状態と呼出し条件により，適切なマクロ展開を行って，効果的なプログラムを生成しようというものである．少し長くなるが，その例を**図-5**(次頁参照)に示す．

#(…) は，計画時変数の値を示す．←←は，計画時変数への代入を意味する記法である．図-5 に示すライブラリ grasp が，**図-6**(次頁参照)の形で呼び出されたときを考えてみる．変数 the_arm の値は YELLOW である．すなわち，#(the_arm)≒BLUE だから，(イ)の文が展開され，the_finger には YFINGER が代入される．これ以降に現れるプログラム中の the_finger はすべて YFINGER という値をもつ．opening_before_departure は指定されていないから（ロ）は何も展開し

```
ROUTINE grasp(TRANS special_departure,special_approach;
          FRAME ATOM the_arm(DEFAULT YELLOW);
          FRAME object,grasping_point,thing_object_affixed_to;
          DISTANCE SCALAR opening_before_departure,thickness(DEFAULT 0.3*CM),
                          opening_for_approach(DEFAULT 15*CM));
grasping:BEGIN
  ATOM the_fingers;
  CLAUSE t,u;
  PLAN IF #(the_arm)=BLUE
       THEN the_fingers ←← BFINGERS
       ELSE the_fingers ←← YFINGERS     ----- (イ)
  PLAN IF SPECIFIED(opening_before_departure)  ------(ロ)
       THEN OPERATE #(the_fingers) WITH OPENING=opening_before_departure;
  PLAN IF SPECIFIED(special_approach)
       THEN u ←← CLAUSE(WITH APPROACH=special_approach)  }--- (ハ)
       ELSE u ←← NILCLAUSE;
  PLAN IF SPECIFIED(special_departure)  --------(ニ)
       THEN MOVE #(the_arm) TO grasp_point
            WITH DEPARTURE=NILDEPROACH
            VIA #(the_arm)*special_departure THEN
              BEGIN
                OPERATE #(the_fingers) WITH OPENING=opening_for_approach
              END
            #(u) ------(ホ)
       ELSE MOVE #(the_arm) TO grasp_point
            WITH DEPARTURE=NILDEPROACH
            VIA #(the_arm)*DEPROACH(grasp_point) THEN
              BEGIN
                OPERATE #(the_fingers) WITH OPENING=opening_for_approach
              END
            #(u); -------(ヘ)
  CENTER #(the_arm)
    ON OPENING < (thickness-0.2*CM) DO    ------ (ト)
       missed:BEGIN  ------- (チ)
         STOP #(the_arm);
         SCALAR flag;
         OPERATE the_fingers WITH OPENING=opening_for_approach;
         PLAN IF SPECIFIED(special_approach)
              THEN BEGIN
                     MOVE #(the_arm) TO #(the_arm)*special_approach DIRECTLY
                   END
              ELSE BEGIN
                     MOVE #(the_arm) TO #(the_arm)*DEPROACH(grasp_point) DIRECTLY
                   END;
         WRITE("Grasp failed; Type a '1' to retry");
         READ(flag);
         IF flag#1 THEN ABORT;
         MOVE #(the_arm) TO grasp_point DIRECTLY;
         CENTER #(the_arm)
           ON OPENING  (thickness-0.2*CM) DO ABORT("Closed on air");
       END missed;
  grasp_point ← #(the_arm);
  PLAN IF SPECIFIED(thing_object_affixed_to)                    }---- (リ)
       THEN UNFIX object FROM thing_object_affixed_to;
  AFFIX object TO #(the_arm);
END grasping;
```

図-5 AL におけるライブラリ・マクロルーチンの例

ない．次に，(ハ)の文では special_approach が指定されているから，uという変数には

WITH APPROACH=FRAME (ROT(Z, 90*DEG), VECTOR(0, 0, −3))

というテキストが代入され，(ホ)(ヘ)の節変数uの値として，このテキストが展開される．(ニ)の PLAN IF 文では，呼出し時に special_departure が指定されていないから ELSE 部が展開される．ここまでのマクロ展開によって，腕は対象物をつかむ場所まで動いているはずだから，つぎに，つかみ動作 CENTER を展開する．このプログラムでは対象物をつかみ損なった場合の対策が考えられている．そのために指の開き幅が対象物の幅より 0.2cm 狭くなったか否かをモニタする(ト)．指幅が対象物の幅より 0.2cm 狭くなったということは，対象物をつかみ損ねたことを意味する．この場合には，つかみ直すための missed という

```
grasp(the_arm=YELLOW, object=bracket, grasp_point=bracket_grasp,
      special_approach=FRAME(ROT(Z,90*DEG),VECTOR(0,0,-3)),
      opening_for_approach=3*CM);
```

図-6 ライブラリ・マクロルーチン呼出しの形式

動作手順を用意しておく．対象物を正しくつかんだ場合には，(リ)のように FRAME 間の結合関係を正しく変更し，grasping という手続きは終了する．

このようにして，図-5 に示した grasp というライブラリ・マクロルーチンが，図-6 のように呼出されると結局，**図-7**(次頁参照)のようなプログラムに展開されるわけである．

2.7 作業プログラムの例（その2）

前節で例示したようなライブラリ・マクロルーチンがいろいろ用意されていれば，それを呼出して効果的なプログラムを生成させることが容易になる．**図-8**(次頁参照)は grasp, normal_search, release というラ

```
MOVE YELLOW TO bracket_grasp
  WITH DEPARTURE=NILDEPROACH
  VIA YELLOW*FRAME(NILROT,10*7) THEN
    BEGIN
      OPERATE YFINGERS WITH OPENING=3*CM
    END
  WITH APPROACH=FRAME(ROT(Z,90*DEG),VECTOR(0,0,-3));
CENTER YELLOW
  ON OPENING < 0.2*CM DO
     missed:BEGIN
       STOP YELLOW;
       SCALAR flag;
       OPERATE YFINGERS WITH OPENING=3*CM;
       MOVE YELLOW
         TO YELLOW*FRAME(ROT(Z,90*DEG),VECTOR(0,0,-3))
         DIRECTLY;
       WRITE("Grasp failed; Type a '1' to retry");
       READ(flag);
       IF flag≠1 THEN ABORT;
       MOVE YELLOW TO bracket_grasp DIRECTLY;
       CENTER YELLOW
         ON OPENING < 0.2*CM DO ABORT("Closed on air");
     END missed;
bracket_grasp ← YELLOW;
AFFIX bracket TO YELLOW;
```

(図-5 で定義したライブラリ・マクロルーチン grasp を図-6 のように呼びだすと本図のプログラムが展開される)

図-7 ライブラリ・マクロルーチンの展開例

イブラリを利用して書いたプログラムの例である．仕事は，**2.5** で述べたものと同じであるが，エラー対策等を考慮した，より豊富な内容の動作が，高い水準でコンパクトに記述されている．なお，図-8 中，grasp, normal_search, release の各文は，**2.6** で述べたように，ライブラリ・マクロルーチンを呼びだして展開され，環境の状態に応じて適切なプログラムに変換されるわけである．

3. AUTOPASS[4] (AUTOmated Parts ASsembly System)

IBM トーマス・ワトソン研究所では，数年前からロボットにより組立作業の研究を，小さなグループが

```
 9  1. ASM SUPPORT BRACKET
10  P/U AND POSITION THE NUT IN THE NEST OF THE FIXTURE.
11  1090037 NUT, CAR RET TAB        QTY 01
12  P/U, ORIENT AND POSITION THE BRACKET INTO THE FIXTURE
    WITH ITS TAB OVER THE NUT.
13  1115191 BRKT ASM, RAIL SUPPORT      QTY 01
14  P/U SCREW AND LOAD DRIVER.
15  1107379 STUD, CR TAB INTLK      QTY 01
16  P/U, ORIENT AND POSITION THE INTERLOCK OVER
    THE BRACKER HOLE, WITH THE NOTCHED LUG UP.
17  1117637 INTERLOCK, CR + TAB      QTY 01
18  P/U AIR DRIVER.
19  DRIVE SCREW TIGHT.
20  TORQUE 12.0 IN/LBS.
21  ASIDE AIR GUN.
```

図-9 組立工に対する組立指示書の例

行っており，既に，マニピューレータを試作し，それを制御するための，機械の制御と密接したレベルのプログラミングシステム ML[5] (Manipulator Language) を発表している．このような低水準の言語で，複雑な組立作業を記述するのは大変煩雑な仕事である．そこで，プログラミングの労力を低減し，より大きな作業をより容易に記述するために，IBM のグループでも高水準言語が検討され，その結果定義された言語が，AUTOPASS である．

AUTOPASS では，人工知能流の高度の計画作成技法は採用せず，ユーザが，組立作業の全体計画を作って作業を記述し，それをコンパイルしてロボットを動かすコードを生成するという方針をとっている．人手による組立作業の場合にも，**図-9** に例示したような組立指示書は，どうしても書かざるを得ない．ちょうど，このようなレベルで，ロボットに対するプログラムを書けるようにすることを狙っている．ユーザは，どの部品をどれと組立てるのか，そのときどの工具を使うのか，また，対象となる部品は作業環境のどこに

```
example2:BEGIN(

  "world model description here"

  COBEGIN
    ypickup:BEGIN
      grasp(object=bracket,grasp_point=bracket_grasp,opening_for_approach=3*CM);
      MOVE bracket_hole TO beam_hole + VECTOR(0,0,-3) WRT beam_hole;
      MOVE YELLOW TO @ + VECTOR(0,0,6) WRT beam_hole
        ON FORCE (Z WRT beam_hole) > 50*OZ DO STOP YELLOW
        ON ARRIVAL DO ABORT("!ERROR! bracket went too far");
    END ypickup;
    bpickup:BEGIN
      grasp(the_arm=BLUE,the_object=bolt,grasp_point=bolt,opening_for_approach=3*CM);
    END bpickup;
  COEND;
  MOVE bolt TO beam_hole + VECTOR(0,0,-5.3) WRT beam_hole;
  normal_search(BLUE,0.2*CM,1.6*CM,60*OZ,9);
  MOVE BLUE TO @ * FRAME(ROT(Z,90*DEG),VECTOR(0,0,4))
    ON FORCE (Z WRT BLUE) > 60*OZ DO STOP BLUE;
  COBEGIN
    parky:BEGIN
      release(the_object=bracket,the_opening=3*CM,the_new_parent=beam);
      MOVE YELLOW TO YPARK;
    END parky;
    parkb:BEGIN
      release(the_arm=BLUE,the_object=bolt,the_opening=3*CM,the_new_parent=beam);
      MOVE BLUE TO BPARK;
    END parkb;
  COEND;
END example2;
```

図-8 AL による作業プログラムの例 (その 2)

置かれているかを指示する．AUTOPASS コンパイラは，環境モデルと呼ばれる，データベースを用いて，ラン・タイムの環境変化をシミュレートしつつ，ロボットに対する動作指令とパラメータを生成する．指令書や環境モデルに不明確な点がみつかった場合には，ユーザとの対話のもとで，コード生成を行うという，考え方をとっている．

AUTOPASS がどんなスタイルの言語であるかを示すために，図-9 に示した手作業向けの組立指示書と同じ内容の作業を，同言語でプログラムした例を，**図-10** に示す．AUTOPASS は，まだ，ようやく定義された段階であり，内部の詳細については，不明な点も多いが，かなり AL の影響を受けているようである．なお，AUTOPASS は，PL/I に組み込まれる形で研究が進められる予定のようである．参考までに，現在発表されている AUTOPASS の高水準ステートメントの構文を，**図-11** にまとめておく．構文中の大文字は，予約語であり，筆記体の部分はオプションである．

4. LAMA[6] (Language for Automatic Mechanical Assembly)

LAMA は，MIT で検討されている，機械の組立作業を対象とした高水準のプログラミングシステムである．LAMA は LISP で書かれており，作業も例えば**図-12** のような形式で記述される．LAMA は，このようなプログラムと，別途作成される環境のモデルから，マニピュレータが実行しうるプログラムへ変換する．環境のモデルは，直方体と円筒を基礎とした幾何学モデルとして構成される．このモデルを用いて，作業中に他の物体と衝突しないような運動のパスを決定し，また，組立作業を実行するための細かい手続きや，必要なパラメータを決定する．

LAMA は，まだ研究の初期の段階にあるせいか，環境の表現法も単純であり，AL や AUTOPASS に比べて，現実の組立作業との関連づけが弱く，実際的な高水準言語へ発展するには，今しばらく時間がかかりそうに思われる．

5. 作業記述言語に関する今後の課題

本稿では，ロボットの組立作業を記述するための高水準言語に関する最近の研究について紹介し

```
1. OPERATE nutfeeder WITH car-ret-tab-nut AT fixture.nest
2. PLACE bracket IN fixture SUCH THAT bracket.bottom
       CONTACTS car-ret-tab-nut.top
       AND bracket.hole IS ALIGNED WITH fixture.nest
3. PLACE interlock ON bracket SUCH THAT
       interlock.hole IS ALIGNED WITH bracket.hole
       AND interlock.base CONTACTS bracket.top
4. DRIVE IN car-ret-intlk-stud INTO car-ret-tab-nut
       AT interlock.hole
       SUCH THAT TORQUE IS EQ 12.0 IN-LBS USING air-driver
       ATTACHING bracket AND interlock
5. NAME bracket interlock car-ret-intlk-stud car-ret-tab-nut
       ASSEMBLY support-bracket
```

図-10 AUTOPASS による作業プログラムの例

```
State change statement

  PLACE object1 ON object2 grasping final-conditions contains then-hold
  INSERT object IN receptor position sensor then-hold
  EXTRACT object distance sensor
  LIFT object distance
  LOWER object ONTO surface sensor then-hold
  LOWER object distance sensor then-hold
  SLIDE object ON surface slide-termination then-hold
  PUSH object direction UNTIL final-condition then-hold
  ORIENT object SUCH THAT positional-condition sensor then-hold
  TURN rotor turning-condition rotation-axis then-hold
  GRASP object grasp-position hand-position grasping-force
  MOVE spatial-feature final-condition
  MOVE spatial-feature TO position final-condition
  MOVE spatial-feature motion-specification final-condition
  RELEASE

Tool statement

  OPERATE tool load-list target-position attachment
          final-condition tool-parameter-list then-hold
  CLAMP locking-device SUCH THAT final-condition
  UNCLAMP locking-device SUCH THAT final-condition
  LOAD tool load-list
  UNLOAD tool load-list
  FETCH tool from-holder
  REPLACE tool to-holder
  SWITCH tool ON/OFF
  LOCK locking-device attachment
  UNLOCK locking-device release

Fastener statement

  ATTACH fastener second-fastener TO target-position side-attachment
         final-condition
  DRIVE IN drive-fastener target-position final-condition
           using-driver attachment driver-parameter-list
  RIVET object-list target-position side-attachment
  FASTEN object1 TO object2 more-objects WITH fastener target-position
         final-condition
  UNFASTEN fastener-list source-position release target-position

Note: Any statements in the above three classes may be preceded by
a qualifying hand specification.      WITH hand-name.....

Miscellaneous statement (partial listing)

  VERIFY inspection-condition inspection-action-list
  OPEN STATE OF locking-device IS final-condition-list
  NAME object-list ASSEMBLY assembly-name
  END
```

図-11 AUTOPASS のシンタックス

```
(GRASP OBJ: [PISTON-PIN])
(PLACE-IN-VISE OBJ: [PISTON-PIN]
               SUCH-THAT: (PARALLEL [PISTON-PIN] [TABLE]))
(UNGRASP OBJ: [PISTON-PIN])
(GRASP OBJ: [PISTON]
       SUCH-THAT: (FACING+ ([PISTON] TOP) DOWN))
(INSERT OBJ1: [PISTON-PIN]
        OBJ2: [PISTON PIN-HOLE]
        SUCH-THAT: (PARTLY (FITS-IN OBJ1 OBJ2) 0.25))
(UNGRASP OBJ: [PISTON])
(GRASP OBJ: [PISTON-ROD]
       SUCH-THAT: (FACING+ ([ROD-BAR] TOP) UP))
(INSERT OBJ1: [PISTON-PIN]
        OBJ2: [PISTON-ROD SMALL-END-HOLE])
(UNGRASP OBJ: [PISTON-ROD])
(GRASP OBJ: [PISTON])
(REMOVE-FROM-VISE OBJ: [PISTON])
(PUSH-INTO OBJ: [PISTON-PIN]
       SUCH-THAT: (AND (FITS-IN [PISTON-PIN] [PISTON PIN-HOLE])
                       (FITS-IN [PISTON-PIN] [PISTON-ROD SMALL-END])))
(UNGRASP OBJ: [PISTON])
```

図-12 LAMA による作業プログラムの例

た．AL, AUTOPASS, LAMA ともに，環境としては，機械の組立作業をとり上げている．この3つのうち，最も研究が進んでいるものがALであり，現在のところ唯一の組立作業記述用の高水準言語といってもよい位である．AL は SAIL に埋込まれており，AUTOPASS は PL/I に，そして LAMA は MACLISP で書かれている．いずれも自前の言語に組込んで発展させていこうとしている点は見習うべきであろう．

アドバンスド・オートメーションの将来の姿を追求していくとき，高水準言語によるプログラミングシステム開発へと行きつくのは，ごく自然な歩みである．現在は，まだ，構文の検討やコンパイラの構成法に重点がおかれているが，次第に他の問題も顕在化しつつあるように思われる[7)~9)]．以下に，いくつかの今後の課題について考えてみることにする．

（1） 作業環境の記述について　ロボットに組立作業等を実行させる際には，動作の記述と共に，作業環境の記述が必要である．ロボットに対する動作の指示は，作業プログラムの作製である．これは，言語の高水準化に伴い，次第に簡略化され，コンパクトな記述が可能になっていくものと思われる．一方，環境や対象物に対する教示は，いつまでも，図-2 に示したような，宣言的記述だけに頼るわけにはいかない．なぜなら，言語の高水準化が進んで，動作記述がコンパクトになればなるほど，プログラム全体の中で環境記述文の占める割合が増大し，遂には，プログラムの大半は環境記述に費されるということになりかねないからである．従って，言語水準の高度化に伴って，効果的な環境教示方式の研究が大切な課題として顕在化してくるものと思われる．

筆者等は，環境教示のひとつの手段として，ローカルな視覚と，グラフィクスを結合して，環境のモデルを効率的に作製する方法を検討してきた．環境を通常のモニタテレビに映し，その画面に，グラフィクスを重畳できる装置[10)]を用い，グラフィクスの分野における幾何学モデル作製技法と，局所的視覚，更に，小さなスポット光を用いる距離測定法をインタラクティブに作用させれば，図-2 のような環境記述が効果的に実現できる可能性があると考えられる．

（2） 視覚構成言語について　周知のごとく，実際に作業を実行する際に，視覚の果す役割は極めて大きい．AL は，将来，視覚構成言語と組合わせることが予定されており，そのはしりとして，Verification Vision の研究[8)]が報告されている．組立作業の環境では，見るべき環境のモデルが，データベースとして予め準備されている．すなわち，どの対象物が，大体，どの位置に，どんな姿勢で置かれているかということはわかっているわけである．視覚に要求されることは，これらの知識を活用して，実際の環境を検証し，正確な位置や姿勢を求めることである．この処理を実時間で行えばビジュアルフィードバックに直接つながるわけである．視覚の研究は広く行われ，多くの成果が報告されている．既に確立されたローカルな処理を一般的なプログラミングシステムとしてまとめることは，必要に応じ，必要な視覚を作り出すことを容易にする．このような視覚構成言語の開発は，ロボットに多様な環境理解能力を付与することに大変役立つものと思われる．

（3） ランタイム・システムの問題について　AL では，コンパイル時に，環境のモデルを使って，実行状態をシミュレートしながら，実行プログラムを生成する．そして，ランタイム・システムでそれを実行する．ロボットがプログラムどうりに順調に行動すれば実世界とモデルの対応は一応保たれる．しかし，失敗すれば，その対応が大きくくずれる可能性がある．実世界と，そのモデルの対応を保持するためには，特にロボットが行動し，環境を変えていく過程，すなわちランタイムのデータベース管理技法をもっとつめておく必要があると考えられる．また，ランタイム・システムでは，多くのプロセスが同時に走る．感覚系と行動系が同時に走る場合の，プログラム技法は，大変興味深い場を提供している．

6. むすび

1970年頃，世界のいくつかのグループが，知能ロボットの開発を競った．その後，知能ロボット開発熱は鎮静化したように見える．しかし，それは，研究者達がロボットに対する関心を失ったためではないと筆者は思う．人間に似た巧妙に動く知的な機械を作りたいという，長い間人々が持ち続けた夢が，わずか数年で消えてしまうとは考えにくいからである．実際，つみ木の世界で行動した初期の知能ロボットはその後オートメーションの世界に住みつき，そこで着実に生長し続けてきた．視覚も，手の動作記述言語も，また，人工知能の研究も，いまそれぞれに力をためつつある．その間，LSI やマイクロコンピュータなどハードウェアの技術も急進展し，価格も低下した．このようにし

て，新しい世代の知能ロボットを開発するための素地は整いつつある．

新しい世代の知能ロボットには，つみ木やオートメーションの世界に代る，新鮮な環境が必要である．筆者は，その第1候補としてアドバンスド・テレオペレーションの環境を提起したい．アドバンスド・テレオペレータとは，原子力発電所や再処理工場の補守，点検，修理作業，深海における作業，宇宙空間での作業など，人間にとって危険な環境での作業を，高度に情報化，コンピュータ化された遠隔操作で行うロボットである．オートメーションの環境は，必要ならばロボットの能力に合わせて多少変更することが許される．しかし，テレオペレーションの環境は，変えることはできない．また作業は通常一回だけであり，ゆう長なプログラミングを行っている暇はない．このように，テレオペレーションの世界は，ロボットにとって，実世界性，実時間性ともに厳しい条件にある．新しい世代の知能ロボットでなければ，理解し，行動することのできない環境なのである．

参考文献

1) R. Finkel et al.: AL, A Programming System for Automation, Stanford AI Lab., AIM-243 (1974).
2) R. Paul: WAVE, A Model Based Language for Manipulator Control, Industrial Robot, p. 10, Vol. 4, No. 1 (1974).
3) R. Paul: Modelling, Trajectory Calculation and Servoing of a Computer Controlled Arm, Stanford AI Lab. AIM-177 (1972).
4) L. I. Lieberman and M. A. Wesley, AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly, IBM J. of R & D, p. 321, Vol. 21, No. 4 (1977).
5) P. W. Will and D. Grossman: An Experimental System for Computer Controlled Mechanical Assembly, IEEE Trans. Vol. C-24, No. 9, p. 895 (1975).
6) T. Lozano-Pérez: The Design of a Mechanical Assembly System, MIT AI Lab. AI-TR-397 (1976).
7) T. O. Binford et al.: Exploratory Study of Computer Integrated Assembly Systems, Stanford AI Lab. AIM-285.4 (1977).
8) R. C. Bolles: Verification Vision Within a Programmable Assembly System, Stanford AI Lab. AIM-295 (1976).
9) R. A. Finkel; Constructing and Debugging Manipulator Programs, Stanford AI Lab. AIM-284 (1976).
10) 長谷川：ロボットの作業データを表示するモニター装置，昭和53年電気学会全国大会，No. 1062 (1978).

(昭和53年6月7日受付)