〔総合論文〕

知能ロボット・プログラミングシステム COSMOS

小笠原　司*　井上博允**

本論文は，知能ロボット研究用に開発したプログラミングシステム COSMOS について述べたものである．COSMOS は，Cognitive Sensor Motor Operations Study の略称である．高水準のロボット言語，ロボットアームとそのコントローラ，3次元視覚機能，敏感な触覚センサ，および，ユーザインターフェイスより構成される．ホストのミニコンピュータとマイクロコンピュータは GPIB で結合され，基本ロボットコマンドによって交信している．システムは階層構造をとっており，ハードウエア，ソフトウエアともにモジュール化および拡張性が重視された設計になっている．逐次，人工知能の手法を組み込んでゆくため，システムのソフトウエアの主な部分は Lisp で記述されている．

本論文ではロボット言語 AL/L (Assembly Language in Lisp) とその環境モデル管理システム，ロボットソフトウエアの構成，およびタブレットを用いた教示システムについて述べる．ロボット言語の処理系では，環境モデルの管理が重要な役割を占めていることが明らかになった．また，システムを用いた実験を通して，タブレットが教示用デバイスとして優れていること，および COSMOS のロボットプログラミングシステムとしての有効性が確認された．

1. はじめに

知能ロボットは，感覚，思考，行動機能を持った機械である．できる限り人間の能力に近い機械を作ろうとロボットの研究は続けられてきた．しかし，これまでに開発されてきたものは，扱う対象物が限られたものである，特定の環境でしか動作しない，ルーチンワークにしか適用できない，動作が非常に遅い，等，知能ロボットというには程遠いものであった．その原因の一つとして，要素技術が不十分であることがあげられるが，もう一つの大きな原因は，ロボットのシステム化に関する技術が確立されていないことである．現在，知能ロボットと呼ばれているものを見ると，感覚機能と行動機能を単に結合しただけのものが多い．しかし，今後は構成要素の複雑化，多種化によって，より複雑で大規模なロボットシステムの開発が行われるであろう．その時にネックとなるのが，システム化の技法，ソフトウエアの構成法である．このことをふまえ，知能ロボットのシステム構成法について明らかにするとともに，以後の知能ロボット研究の基礎となるべきロボットシステムを作成するのが本研究の目的である．

ロボットのシステム化に関する研究は，ほとんどなされていない．Stanford 大学では，ロボット言語 AL[1] をベースにプログラミングシステムを作ろうと試みている．AL は初期の動作レベルのロボット言語として優れたものであり，その後のロボット言語の研究は AL を目標に行われてきた．しかし，AL では環境モデルの管理が十分でなく，また人工知能の手法の導入など拡張性については問題が残っている．一方，IBM では AML[2] をベースにロボットシステムの統合を図っているが，AML はロボット言語としては高水準とはいい難く，特に環境モデルの機能は備えておらず，産業用への応用が主な目的である．また，NBS では階層構造のロボットシステムの研究を行っており，マルチプロセッサ構成のシステムを開発している[3]．しかし，ロボット言語などの上位のソフトウエアに関する研究は遅れており，問題解決機能などの導入はまだなされていない．前者二つの研究は，いずれも独自のロボット言語を開発し，その上にシステムの統合を行うという手法をとっている．しかし，専用のロボット言語では機能の拡張，特に知能の高度化といった面でどうしても障害が多く，実験システムには不向

原稿受付　1984 年 6 月 21 日
* 電子技術総合研究所制御部
** 東京大学工学部機械工学科

きである．

本研究ではシステムのモジュール化，拡張性を重視し，Lisp[4] をベースにしてロボット制御用の関数を組込み，システムの統合を図った．モジュールを独立したプロセスと考え，システムを複数のプロセスに分割し，階層構造に結合し，プロセス間でデータの通信を行いながら処理をすすめてゆくというシステム構成をとった．これにより，ミニコンピュータ1台とマイクロコンピュータ3台という比較的小規模なハードウエア上に，デバイスの制御からロボット言語処理系までの全てのソフトウエアを組込み，比較的短時間で，使い易く拡張性のあるシステムを構成することに成功した．以下では，本研究で開発した知能ロボットシステム COSMOS (Cognitive Sensor Motor Operation Study の略) について，基本方針，ロボット言語と環境モデル，システム構成，応用例について述べる．

2. 基本思想

COSMOS は，知能ロボット実験システムであり，具体的な応用分野としては，原子力プラント，宇宙，海底など，危険な環境下での人間の作業を代行し得る遠隔操作型の知能ロボットを想定している．作業内容としては，機械装置の分解・修理・組立てを考える．

COSMOS は，次の2点を基本方針としている．

(1) 感覚機能，行動機能，思考機能，人間との対話手法などを一つのシステムとして統合し，操作しやすい知能ロボットのプロトタイプを構築する．

(2) このシステムをツールとして種々の研究を行い，それらの成果をシステム内に使用可能なソフトウエアとして蓄積することにより，知能ロボットの機能より高度なものへと育てていく．

この2点に従い，システムは拡張性，モジュール化を重視した設計がなされている．

研究の成果を以後の研究ツールとして使用可能なソフトウエアとして蓄積することを試みる際には，まず使用するプログラム言語を定めておく必要がある．本研究では，上位のソフトウエアには，逐次人工知能の手法を組込んでゆく方針なので，上位のソフトウエアは Lisp を用いてインプリメントすることにした．Lisp は会話型言語であり，ロボットのような会話形式のシステムに適している．なお，数値計算を多用する部分は，Pascal で記述する．アセンブラの使用はできるだけ避け，最下位レベルの機械に依存する部分と，特に高速性が要求される部分のみに止め，将来のハードウエア変更時の労力を減らすよう配慮しておく．

知能ロボットシステムは，いくつかのサブシステムから構成される．個々のサブシステムの研究は，独立した完結性をもつように進められるが，成果はシステム内に統合され，その後はツールとして使っていくことができれば理想的である．そのため各ソフトウエアとも，できる限りモジュール化を考えて設計し，かつ読みやすく書くという努力を重ねる．機能上またはレベル毎に分割される大きなまとまりは，一つのモジュールとし，モジュール間はメッセージの受け渡しによってインターフェイスし，成果を蓄積する方針をとる．

上位のソフトウエアは全てホスト計算機上に乗せ，マルチプロセス方式の OS のもとに走らせる．デバイスを直接制御したり，センサのデータを入力する機能は，複数のマイクロコンピュータに分散させる．マイクロコンピュータとホスト計算機は，原則として GPIB でインターフェイスし，新しいデバイスの追加，拡張を容易にする．キーボード，タブレット，音声入力装置といった，ユーザの入力装置は，互いに置き換えが自由に行えるよう仮想端末の考え方で取扱う．

新しいシステムの開発研究の際には，いろいろの試行錯誤が避けられない．計算機の能力との関係で，実行時間をとるか，一般性をとるか判断をせまられることがあるが，本システムではできるだけ後者をとることとし，結果的にシステムのレスポンスが遅くなってもやむをえないと考えている．

3. ロボット言語 AL/L と環境モデル

ロボット言語の処理系は，記述されたプログラムをよりレベルの低い言語体系へ展開するものと考えればよい．そのレベルは，作業レベル，対象物レベル，動作レベルに分類され[5]．前者ほど高水準なものである．そして，言語のレベルアップを行うには，より高いレベルで記述された動作を，既に開発済の言語に展開する処理系を作ることで実現できる．本研究では，動作レベルのさらに下に，プリミティブレベルのコマンド体系を設定し，動作レベルの記述からコマンド体系を生成する処理系を開発した．これらのレベルは，計算機のプログラム言語と次のように対応づけることができる．

プリミティブコマンド——機械語
動作レベル言語——アセンブラ
対象物レベル言語——高級言語
作業レベル言語——より高度なプログラミング言語

3.1 AL/L[6]

AL/L (Assembly Language in Lisp) は，Lisp に埋めこまれた動作レベルのマニピュレーション言語である．AL/L の構文は Stanford 大学の AL に似ているが，設計方針は異なっている．筆者らは，全く新しい言語設計

```
<program> ::= ( PROGRAM <program identifier> <body> )
<body> ::= {<statement>} ...
<declare statement> ::= ( DECLARE <variable> <data type> )
<data type> ::= SCALAR | VECTOR | ROTATION | COORDINATES | TRANS
<assignment statement> ::= ( SETQ <variable> <expression> )
<if statement> ::= ( IF <condition> <then part> [<else part>] )
<loop statement> ::= ( LOOP {<statement>} ... )
<exit statement> ::= ( EXIT <condition> )
<parallel statement> ::= ( PARALLEL {<process>} ... )
<process> ::= ( {<statement>} ... )
<signal statement> ::= ( SIGNAL <event> )
<wait statement> ::= ( WAIT <event> )
<pause statement> ::= ( PAUSE <expression> )
<affix statement> ::= ( AFFIX <coordinates> <coordinates>
                                    [<relation> [<expression>] )
<relation> ::= RIGIDLY | NONRIGIDLY
<unfix statement> ::= ( UNFIX <coordinates> <coordinates> )
<assert statement> ::= ( ASSERT <name> <slot> <datum> )
<move statement> ::= ( MOVE <coordinates> <destination> {<clause>} ... )
<clause> ::= <via clause> | <with clause> | <condition monitor>
<via clause> ::= ( VIA {<coordinates>} ... [ ( VELOCITY <expression> ) ]
                                           [ ( DURATION <expression> ) ] )
<with caluse> ::= ( WITH APPROACH <expression> )
                | ( WITH DEPARTURE <expression> )
                | ( WITH <feedback expression> )
                | ( WITH DURATION <expression> )
<open statement> ::= ( OPEN <hand> <width> )
<close statement> ::= ( CLOSE <hand> <force> )
<center statement> ::= ( CENTER <arm> )
<stop statement> ::= ( STOP <arm> )
<condition monitor> ::= ( ON [<label>] <condition> ( {<statement>} ... ))
<enable statement> ::= ( ENABLE <label> )
<disable statement> ::= ( DISABLE <label> )
<macro declaration> ::= ( MACRO <macro name> ( <parameters> )
                                    ( <local variables> ) <body> )
<macro call> ::= ( <macro name> {<actual parameter>} ... )
<plan assignment statement> ::= ( %SETQ <variable> <expression> )
<plan loop statement> ::= ( %LOOP {<statement>} ... )
<plan exit statement> ::= ( %EXIT <condition> )
```

Fig.1 AL/L language syntax

するよりも，現在存在するロボット言語の中で，高水準と考えられる AL のデータ構造および構文に準拠した言語を作成することから研究を開始した．しかし，システムの拡張性，プログラムの生産性などを考え，Lisp 埋めこみ型の言語とした．Lisp を用いた理由は，以下の点である．

（1） ユーザが関数を定義することにより容易に言語のレベルを拡張することが可能である．

（2） 人工知能の手法を容易に取り入れることができる．

（3） 会話形式のプログラミング環境に適している．

一方，Lisp を用いる場合の欠点として，

（1） ガーベジコレクションにより動作が中断するおそれがある．

（2） カッコが多くて読みずらい．

等が上げられるが，（1）については，並列ガーベジコレクションを採用するなどの方法により解決できる．また（2）については，ロボット言語の水準を高めてゆけばよい．さらに，プリプロセッサで他の構文から Lisp のプログラムに変換することで解決できる．Lisp のフレキシビリティ，プログラミングシステムとしての強力さを考えると，そのような欠点は補って余りあるものである．

Fig.1 に AL/L の構文を示す．以下に，AL/L の概要について述べる．

3.1.1 データタイプ

対象物の 3 次元表現のために，SCALAR（スカラー），VECTOR（ベクトル），ROT（回転），COORDINATES（座標系），TRANS（変換）のデータ型を持っている．SCALAR は，距離，力，時間，等を表す量である．VECTOR は 3 次元位置ベクトルを表す．ROTATION は回転を表す．COORDIANTES は，局所座標系を表すのに用いられる．TRANS は，座標系，あるいはベクトル間の変換を表す．各データタイプは，同次座標表現に基づき，リスト構造を用いて **Table 1** のように表現される．これらのデータ間の演算については，**Table 2** のような関数を定義する．

3.1.2. 環境の宣言

Table 1 Data type of AL/L

SCALOR	integer or real
VECTOR	(VECT (x) (y) (z) (1))
ROTATION	(ROT (n_x o_x a_x 0) (n_y o_y a_y 0) (n_z o_z a_z 0) (0 0 0 1))
TRANS	(TRANS (n_x o_x a_x p_x) (n_y o_y a_y p_y) (n_z o_z a_z p_z) (0 0 0 1))
COORDINATES	(COORDINATES (n_x o_x a_x p_x) (n_y o_y a_y p_y) (n_z o_z a_z p_z) (0 0 0 1))

環境モデルに新しいデータを付加するには，ASSERT 文を用いる．この文は宣言部中で用いられ，環境の初期状態を宣言する．プログラムで用いる変数については，DECLARE 文であらかじめ型宣言をする．例えば，BOX という座標系が (100, 100, 0) の位置にあることを宣言するためには，次のように記述すればよい．

```
(DECLARE BOX COORDINATES)
(ASSERT BOX LOCATE-AT
                (COORDINATES NILROT
                        (VECTOR 100 100 0)))
```

ここで，NILROT は BOX の座標系が絶対座標系と同じ向きであることを表す定数である．

3.1.3. 結合関係

AFFIX 文により物体間の結合関係を記述する．次の文は，OBJECT 1 が OBJECT 2 に密 (rigid) に結合されていることを宣言している．

```
(AFFIX OBJECT 1 OBJECT 2
 RIGIDLY)
```

密な結合 (rigid affixment) とは，座標系が互いに対象な関係になっていることをいう．すなわち，OBJECT 1 はいつも OBJECT 2 と一緒に動く．これに対し，粗な結合 (nonrigid affixment) は非対象な関係である．すなわち，OBJECT 2 が動けば OBJECT 1 も動くが，OBJECT 1 が動いても OBJECT 2 は動かない．結合関係のデフォルト値は，密な結合である．また，結合関係は UNFIX 文により解除される．

```
(UNFIX OBJECT 1 OBJECT 2)
```

3.1.4 動作記述文

ロボットアームの動きは MOVE 文および STOP 文で記述され，指先の動作は OPEN, CLOSE および CENTER 文で記述される．

MOVE 文の構文は次のようになる．

(MOVE 〈物体〉〈目的地〉{〈修飾節〉…})

〈物体〉は，動かす物体の座標系の名前，またはアームの名前である．〈目的地〉は，アームの動作の目標点の座標系または式である．〈修飾節〉により動作の修飾をすることができる．〈修飾節〉には，VIA 節，WITH 節，そして ON 文の条件のモニタが書ける．VIA 節は中継点を表し，WITH 節は，接近点，力，および速度を表す．さらに，条件モニタではモニタリングすべき条件（式）と，それが真になったときに実行すべき動作を記述する．次の例では，アームは VIAPOINT を通って BRICK へ動く．そして，Z 軸方向の力が 100 g 以上になればアームは停止する．

```
(MOVE ARM BRICK (VIA VIAPOINT)
                        (ON(>(FORCE ZHAT)100)
                                ((STOP ARM))))
```

3.1.5 プログラム・コントロール

AL/L は，Lisp 埋め込み型の言語である．したがって，条件分岐，繰返しの制御については Lisp の関数に委ねている．並行プロセスの記述には PARALLEL 文を用いる．これは，ロボットの協調動作の記述に有効で

Table 2 Arithmetic functions

(!+ 'arg1 'arg2 ...)	sum of arguments s+s=s,v+v=v,t+v=v,c+v=c
(!- 'arg1 'arg2 ...)	difference of arguments s-s=s,v-v=v,c-v=c
(!* 'arg1 'arg2 ...)	product of arguments s*s=s,s*v=v,r*v=v,r*c=c,t*v=v
(!/ 'arg1 'arg2 ...)	quotient of arguments s/s=s,v/s=v
(!DOT 'v1 'v2)	inner product
(NORM 'v)	norm of vector
(UNIT 'v)	normalization of vector
(VECTOR 's1 's2 's3)	construct vector
(ROT 'v 's)	construct rotation
(TRANS 'r 'v)	construct transform
(COORDINATES 'r 'v)	construct coordinates
(POS 'c)	position vector of coordinates
(ORIENT $\{{}^{'c}_{'t}\}$)	orientatin of coordinates/transform
(WRT 'v 'c)	position vector in coordinate system
(EULER 'ϕ 'θ 'ψ)	construct rotation from Euler's angle
(REV_ROT 'r)	rotation axis and angle
(REV_EULER 'r)	Euler's angle of rotation
(REV_TRANS 't)	rotation and position vector of transform
(REV_COORDINATES 'c)	rotation and position vector of coordinates

s=scalar,v=vector,r=rotation,t=trans,c=coordinates

ある．その構文は次のようになる．

(PARALLEL　{〈プロセス〉}…)

〈プロセス〉は互いに独立でなければならない．同期はイベントにより行われる．SIGNAL 文と WAIT 文によりプロセスの同期をとる．

(SIGNAL　〈イベント〉)

(WAIT　〈イベント〉)

3.1.6　マクロ機能

マクロ手続きは，プログラムの記述を簡略化するのに有効である．MACRO 文は，引数付きマクロを定義する．

(MACRO 〈マクロ名〉 (〈引数〉)
(〈局所変数〉) 〈本体〉)

コンパイル時に，システムは環境モデルマネージャを用いてマクロ展開を行う．その時に仮引数は実引数に置換えられ，また環境モデルが参照される．% IF 文は条件付マクロ展開を行う．% LOOP 文は繰返し展開に用いられる．% EXIT 文により展開ループから脱出する．

3.1.7　プログラム例

Fig. 2 に AL/L のプログラム例を示す．このプログラムは，模型のエンジンの分解作業の一部を記述したものである．動作の記述の前に，教示装置あるいはプログラムによって環境を教示しなければならない．ここではプログラムにより記述した．PARALLEL 文を用いて2本の腕を使ってシリンダからコンロッドを抜き出す作業を記述している．

プログラム中で GRASP, PULL-OUT, PUT-AT はマクロで定義できる．例えば，PULL-OUT, PUT-AT は次のように定義される．

```
(MACRO PULL-OUT ( ) ( )
  (MOVE RARM (!+RARM (VECTOR 50 0 0))
             (WITH (=(FORCE Y) 0))
             (WITH (=(FORCE Z) 0))))
(MACRO PUT-AT (OBJECT DEST) ( )
  (MOVE OBJECT DEST)
  (OPEN RHAND 60)
  (UNFIX OBJECT RARM)
```

また，LARM, RARM は左右の腕の名前であり，CYLINDER, CYLINDER-PASS, CONROD 等は COORDINATES 型変数である．CYLINDER-READY, LHAND-READY, CYLINDER-LEFT はイベント名である．

```
(PROGRAM DISASSEMBLY
   (DECLARE CYLINDER COORDINATES)
   (DECLARE CYLINDER-GRASO COORDINATES)
   (ASSERT CYLINDER LOCATE_AT (COORDINATES ..... ))
   (AFFIX CYLINDER-GRASP CYLINDER RIGIDLY
                         (TRANS ..... ))
        .
        .
   ; declaration of environment
        .
        .
   ; motions for disassemble
   (MOVE RARM RPARK)              ;goto parking position
        .
        .
   (PARALLEL
       ((GRASP RARM CYLINDER)     ;motion of rigth arm
        (MOVE RARM CYLINDER-GRASP)
        (SIGNAL CYLINDER-READY)
        (WAIT LHAND-READY)
        (PULL-OUT)
        (SIGNAL CYLINDER-LEFT)
        (PUT-AT CYLINDER CYLINDER-PLACE))
       ((OPEN LHAND 60)           ;motion of left arm
        (MOVE LARM CONROD-PASS)
        (WAIT CYLINDER-READY)
        (CLOSE LHAND)
        (AFFIX CONROD LARM RIGIDLY)
        (SIGNAL LHAND-READY)
        (WAIT CYLINDER-LEFT)
        (MOVE CONROD TOP-OF-VISE)))
        .
        .
```

Fig. 2　An example program of AL/L

3.2 AL/L の環境モデル管理システム

3.2.1　環境モデルの機能

ロボット言語の処理系にとって，環境モデルは非常に大きな役割を負っている．ロボット言語処理系は，高水準の動作プログラムをプリミティブレベルの動作コマンド列に展開する．この処理を行うためには，処理系は内部に環境モデルを持ち，動作をシミュレートして，各時点における物体の位置関係を正確に把握していなければならない．

環境モデルの処理は全て環境モデル管理システムが行い，言語の処理系は環境モデル管理システムに対してコマンドを発し，それに対して答える形式が，ソフトウエアのモジュール化・拡張性を考えた場合望ましい．ロボット言語のレベルが高くなるほど，環境モデルに格納すべきデータの種類は増加し，その管理システムは高度なものになる．環境モデル管理システム内では次のような処理を行う．

（1）　座標系型データの管理

各物体の基準座標系に名前を付け，その値を記憶する．さらに，動作によって物体が移動した場合には，それらの値を自動的に変更する．汎用のプログラミング言語の変数の管理と似ている．動作レベルのロボット言語で用いられる．

（2）　物体間の結合関係の管理

物体の結合関係を表すために物体間をリンクで接続し，結合関係のネットワークを作り上げる．このリンクには，結合関係，相対位置等を記入しておく．動作ごとに物体間の関係を変更し，結合リンクをたどれば，物体の位置，他の物体との関係がわかるようにする．対象物レベルのロボット言語に必要である．

（3） 物体に関する情報の管理

物体間の関係はネットワークで表現できる．各ノードが各物体に相当し，ノードにその物体に関する情報を記録する．例えば，物体の大きさ，重さ，形状，色，把握点，近接点などである．これらの情報は，把握点の計算，衝突回避や問題解決の際の知識として利用される．

（4） 作業記述に関する情報の管理

ユーザ定義の動作手続きや物体の組立方法など，作業に関する知識を環境モデル内に記述しておく．作業レベルのロボット言語の実現に必要であり，これによって，動作の展開を環境モデル管理システム内で行うことができる．

3.2.2 環境モデルの基本機能の実現

物体間の結合関係や物体に関するデータを表現するにはフレーム[7]を用いると都合がよい．そこで，COSMOSではLisp上にFRL (Frame Representation Language)[8]をインプリメントし，その上で環境モデル処理システム[9]を作成した．

環境モデル内には，**Fig. 3** に示すような概念フレームが存在する．実際の物体やアーム，およびそれらの関係を表すインスタンスフレームは，AKO リンクによって概念フレームに接続されている．各概念に対する操作は，処理内容がわかりやすく変更も容易なことから，FRLの副作用（$IF-ADDED ファセット，$IF-REMOVED ファセットなど）を用いて記述した．

THING は，フレームの木構造のルートにあたるフレームであり，全てのフレームから継承メカニズム (inheritance) を実現している AKO リンクをたどって参照することができる．THING フレームには，PROTOTYPE スロットに対する副作用の処理，および拘束条件が記述されている．PROTOTYPE スロットは，実際のフレームの分類に用いている．SCALAR, VECTOR といったデータタイプの場合には，APVAL スロットに値が代入され，型のチェックを行う関数が $REQUIRE ファセットに記述されている．COORDINATES フレームには，座標系の値を計算するための手続きが記述されている．物体は，OBJECT, ASSEMBLY, PART, SUBPART, BORE, STATION, HAND に分類している．物体は，全て COORDINATES 型の値をもつため，COORDINATES フレームと AKO リンクで結ばれている．

```
(FASSERT
    THING
    (PROTOTYPE
        ($IF-ADDED ((FPUT :FRAME 'AKO '$VALUE :VALUE)))
        ($IF-REMOVED ((FREMOVE :FRAME 'AKO '$VALUE :VALUE)))
        ($REQUIRE
            ((FREQUIRE
                '(LAMBDA (VALUE)
                    (MEMBER
                        VALUE
                        '(SCALAR
                            VECTOR
                            ROTATION
                            TRANSFORM
                            COORDINATES
                            ASSEMBLY
                            PART
                            OBJECT
                            SUBPART
                            BORE
                            STATION
                            HAND
                            LINK
                            SHAPE))))))))
(FASSERT SCALAR (APVAL ($REQUIRE ((NUMP :VALUE)))))
(FASSERT VECTOR (APVAL ($REQUIRE ((IS_VECT :VALUE)))))
(FASSERT ROTATION (APVAL ($REQUIRE ((IS_ROT :VALUE)))))
(FASSERT TRANSFORM (APVAL ($REQUIRE ((IS_TRANS :VALUE)))))
(FASSERT
    COORDINATES
    (APVAL ($VALUE ((% (GET_TRANS :FRAME 'WORLD))))))
(FASSERT OBJECT (AKO ($VALUE (COORDINATES))))
(FASSERT ASSEMBLY (AKO ($VALUE (OBJECT))))
(FASSERT PART (AKO ($VALUE (OBJECT))))
(FASSERT SUBPART (AKO ($VALUE (COORDINATES))))
(FASSERT BORE (AKO ($VALUE (COORDINATES))))
(FASSERT STATION (AKO ($VALUE (COORDINATES))))
(FASSERT HAND (AKO ($VALUE (COORDINATES))))
```

Fig. 3 Definition of primitive frames for the model manager

3.2.3 結合関係の実現

LINK フレームで結合関係を実現している．LINK 型フレームの PARENT スロットに COORDINATES 型の親フレームを記述し，CHILD スロットに子の COORDINATES 型フレームを記述する．HOW-AFFIXED スロットで結合の種類を表し，TRANSFORM スロットで相対位置関係を表す．また，COORDINATES 型フレームから，LINK 型のフレームをアクセスするために，PARENT-LINK と CHILD-LINK の2つのスロットを追加する．LINK 型のフレームは自動的に管理できるように，いろいろな副作用を定義する必要があり，**Fig. 4** のようなフレームが定義されている．

例えば，アームが箱を持っている場合には，HAND 1, BOX, LINK-

OF-BOX-TO-HAND 1 の3つのフレームが作られる．LINK-OF-BOX-TO-HAND 1 は HAND 1 と BOX の結合関係を記述しているフレームであり，それらは **Fig. 5** に示すように表現されている．

3.2.4 把握点・近接点に関する処理の実現

把握点は物体に特有のものである．したがって，OBJECT フレームに GRASP スロットを設け，把握点に関する副作用を $IF-ADDED ファセットに記述する．ここでは，実際の把握点の値が OBJECT 型のフレームの GASP-POINT スロットに代入されること，および1つの物体に対しては，把握点は1つしかないという仮定のもとに，古い値は消去されること，という2つの副作用が記述されている．

```
(FASSERT
    COORDINATES
    (PARENT_LINK
        ($IF-ADDED
            ((PUT_VAL :VALUE 'PROTOTYPE 'LINK))
            ((PUT_VAL :VALUE 'CHILD :FRAME))
            ((MAPCAR
                 '(LAMBDA (LINK)
                      (PUT_VAL
                          :VALUE
                          'TRANSFORM
                          (GET_TRANS
                              :FRAME
                              (CAR (GET_VAL :VALUE 'PARENT))))
                      (REMOVE_VAL :FRAME :SLOT LINK))
                 (DELQ :VALUE (COPY (GET_VAL :FRAME :SLOT))))))
        ($IF-REMOVED ((REMOVE_VAL :VALUE 'CHILD :FRAME))))
    (CHILD_LINK
        ($IF-ADDED
            ((PUT_VAL :VALUE 'PROTOTYPE 'LINK))
            ((PUT_VAL :VALUE 'PARENT :FRAME)))
        ($IF-REMOVED ((FREMOVE :VALUE 'PARENT '$VALUE)))))

(FASSERT
    LINK
    (PARENT
        ($IF-ADDED ((PUT_VAL :VALUE 'CHILD_LINK :FRAME)))
        ($IF-REMOVED
            ((REMOVE_VAL :VALUE 'CHILD_LINK :FRAME))
            ((FREMOVE :FRAME 'CHILD '$VALUE))))
    (CHILD
        ($IF-ADDED ((PUT_VAL :VALUE 'PARENT_LINK :FRAME)))
        ($IF-REMOVED
            ((REMOVE_VAL :VALUE 'PARENT_LINK :FRAME))
            ((FREMOVE :FRAME 'PARENT '$VALUE))))
    (HOW_AFFIXED ($DEFAULT (NONRIGID))))
```

Fig. 4 Implementation of affixment mechanism

近接点（アプローチ点とデパーチャ点がある）は，座標系型フレームの APPROACH, DEPARTURE, および DEPROACH スロットに処理を記述する．近接点に関する処理は把握点と同じであるため，副作用手続きは OBJECT フレームの GRASP スロットと共有している．ただし，関数 PUT-SPECIAL-POINT によって，APPROACH-POINT, DEPARTURE-POINT, および DEPROACH-POINT スロットに値が代入される．さらに，DEPROACH-POINT スロットに代入された値は，APPROACH-POINT, DEPARTURE-POINT スロッ

```
(BOX (PROTOTYPE ($VALUE (PART)))
     (AKO ($VALUE (PART)))
     (PARENT_LINK ($VALUE (LINK_OF_ARM1_TO_BOX)))
     (CHILD_LINK ($VALUE (LINK_OF_BOX_TO_BOX_GRASP))))

(ARM1
    (PROTOTYPE ($VALUE (HAND)))
    (AKO ($VALUE (HAND)))
    (PARENT_LINK ($VALUE (LINK_OF_WORLD_TO_ARM1)))
    (CHILD_LINK ($VALUE (LINK_OF_ARM1_TO_BOX))))

(LINK_OF_ARM1_TO_BOX
    (PROTOTYPE ($VALUE (LINK)))
    (AKO ($VALUE (LINK)))
    (PARENT ($VALUE (ARM1)))
    (CHILD ($VALUE (BOX)))
    (TRANSFORM
        ($VALUE
            ((TRANS
                (-1.000000E+00 0.000000E+00 -4.680949E-06 1.000002E+01)
                (0.000000E+00 1.000000E+00 0.000000E+00 -1.000000E+01)
                (4.680949E-06 0.000000E+00 -1.000000E+00 4.999955E+00)
                (0.000000E+00 0.000000E+00 0.000000E+00 1.000000E+00)))))
    (HOW_AFFIXED ($VALUE (RIGID))))
```

Fig. 5 Example frames of affixment mechanism

トのデフォルト値として定義される．

以上の処理を行うために，OBJECT および COORDINATES フレームに **Fig. 6** のようなスロットを追加した．

```
(FASSERT
    OBJECT
    (GRASP
        ($IF-ADDED
            ((PUT_SPECIAL_POINT :FRAME :SLOT :VALUE))
            ((REMOVE_VAL :FRAME :SLOT :VALUE)))))

(FASSERT
    COORDINATES
    (DEPROACH ($IF-ADDED ((@ OBJECT GRASP))))
    (APPROACH ($IF-ADDED ((@ OBJECT GRASP))))
    (DEPARTURE ($IF-ADDED ((@ OBJECT GRASP))))
    (DEPROACH_POINT
        ($IF-ADDED
            ((FPUT :FRAME 'APPROACH_POINT '$DEFAULT :VALUE))
            ((FPUT :FRAME 'DEPARTURE_POINT '$DEFAULT :VALUE)))))
```

Fig. 6 Impremetation of grasp point and departure point

3.3 ロボット言語の処理系の構成

3.3.1 AL/Lの処理系の構成

AL/L では，会話形式の処理を行い，1ステートメントごとに解釈・実行を繰り返す形式をとっている．AL/L ロボット言語処理系は動作の記述をロボットのプリミティブコマンド系列に展開する．以後の処理は実行システムが行う．

Fig. 7 に AL/L 処理系の構成を示す．AL/L が Lisp 埋めこみであるため，その言語処理系の処理形体も関数型をとり，ソースステートメントを渡しオブジェクトコードが値として返る．ソースステートメントはパーザに渡され，構文解析が行われる．次に，macro expander によりライブラリ化した手続きがマクロ展開される．マクロ展開されたステートメントを用いて，シミュレータにより環境モデルのデータの変更が行われるとともに，コードジェネレータを通してオブジェクトコードが生成される．オブジェクトコードは4章で述べる基本コマンドに対応するものである．環境モデルの処理についてはサブシステム化し，QA システムの形式で利用している．

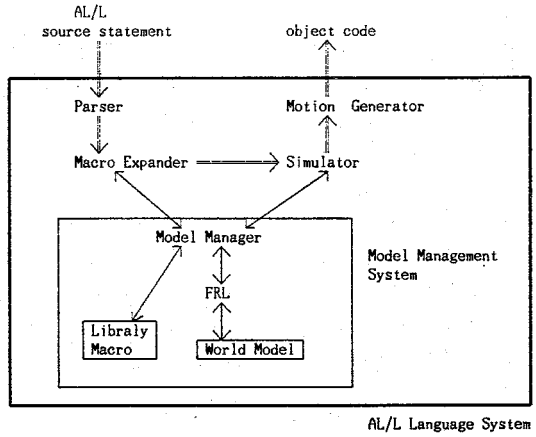

Fig. 7 Configuration of AL/L language system

環境モデルの構成は 3.2 節で述べたようになっており，これらの関数を実行すると，継承メカニズムや付加手続きを用いて処理が行われる．以下では MOVE OPEN, CLOSE, AFFIX, UNFIX の各ステートメントについて処理アルゴリズムを示す．実際のプログラムは Lisp で記述されているが，ここでは Pascal 形式の記述を行っている．get-および put-で始まる手続きが環境モデルを操作する関数である．これらの関数は次の4つの環境モデル管理システムへのアクセス関数を用いて定義されている．

(1) (GET-VAL 'frame 'slot)

フレームのスロットの VALUE ファセットの値を得る．

(2) (PUT-VAL 'frame 'slot 'value)

フレームのスロットの $VALUE ファセットに値を付加する．

(3) (REPLAVE-VAL 'frame 'slot 'value)

フレームのスロットの $VALUE ファセットの値を置き換える．

(4) (REMOVE-VAL 'frame 'slot 'value)

フレームのスロットの $VALUE ファセットの値 value を消去する．

3.3.2 MOVE 文の処理 (**Fig. 8**)

(1) パーザによって構文のチェックが行われ，OBJECT, FINAL, CLUASES の各引数に値が代入される．

(2) FINAL 自身あるいはその結合している物体を探し，その値を LAST-POSITION に代入する．この値は，CLOSE 文での処理に用いられる．なお結合している物体とは，結合しているフレームのうち，その PROTOTYPE が PART あるいは ASSEMBLY のものである．

(3) OBJECT あるいは，その結合している HAND をチェックする．もしあれば，すでに HAND が物体を持っているのであるから何もしない．HAND に結合されている物体がなければ OBJECT をつかみにゆく．すなわち次の動作に対応するオブジェクトコードを生成する．

```
(OPEN HAND 60)
(MOVE HAND OBJECT (GET-GRASP OBJECT)
                  (GET-APPROACH OBJECT))
(CLOSE HAND 0)
```

ここで，GET-GRASPにより環境モデルから把握点を求める．また，GET-APPROACHによってOBJECTのAPPROAGH点の位置を計算する必要がある．

（4） OBJECTからHANDへの変換行列XFORMを計算する．

（5） MOVE文のオブジェクトコードを生成する．次のようなオブジェクトコードが生成される．

(@MOVE 〈アーム名〉
〈departure 点〉
〈中継点〉…
〈approach 点〉)

各点はCOORDINATES型の値で記述される．AL/Lのソースステートメントでは，各点の値は対象物の通る位置で指定されるが，オブジェクトコードでは，アームの通る位置で記述しなければならない．そこで，コード生成時に各点の値を環境モデルから計算した後，XFORMを掛けてアームの位置に変換する．

（6） FORCE，TORQUEについては，CLAUSE中に存在すれば，指定値を計算した後，オブジェクトコードを次の形式で生成する．

(@FORCE 〈向き〉 〈値〉)
(@TORQUE 〈向き〉 〈値〉)

条件モニタについては，次のようなモニタリングプロセスを起動するコードを生成する．

(@ON 〈モニタ名〉 〈動作〉)
(@ENABLE 〈モニタ名〉)
(@MOVE…)
(@DISABLE 〈モニタ名〉)

@ONよりモニタリングプロセスの定義を行う．@ENABLEにより，そのプロセスをスタートした後，@MOVEを実行する．@MOVEが終了すれば，モニタリングプロセスを停止させる．FORCE，TORQUE，DURATION節，ON節については，現在のところ実行システムがその実現機能を持っていない．

（7） HANDの現在位置をFINAL !* XFORMに変更する．次のAL/Lステートメントと同じ機能である．

(ASSERT HAND LOCATE-AT
(!* FINAL XFORM))

これにより，環境モデル中のアームの位置が変更される．物体についてはアームに結合されているため，位置データの変更の必要はない．

（8） 生成されたオブジェクトコードが値として返さ

```
statement move(OBJECT,FINAL,CLAUSES);
  local XFORM, HAND;
  begin
    parse_statement();                                          (1)
    replace_val(LAST_POSITION,
                get_affixed(FINAL,[PART,ASSEMBLY]));            (2)
    if is_hand(OBJECT) then HAND := OBJECT;
       else HAND := get_affixed(OBJECT,[HAND]);
    if (HAND == NIL) then begin                                 (3)
      open_hand(HAND);
      move(HAND,get_grasppoint(OBJECT),
                get_approach(OBJECT),
                get_departure(OBJECT));
      close_hand(HAND);
      end;
    XFORM := get_xform(OBJECT,HAND);                            (4)
    code_generate(@MOVE,HAND,
                        get_departure(OBJECT) !* XFORM,
                        get_viapoints(CLAUSES) !* XFORM,
                        get_approach(FINAL) !* XFORM,
                        get_trans(FINAL) !* XFORM);             (5)
    code_generate(force_code(CLAUSES));
    code_generate(torque_code(CLAUSES));
    code_generate(conditon_code(CLAUSES));                      (6)
    put_location(HAND,FINAL !* XFORM);                          (7)
    return(CODE);                                               (8)
  end;
```

Fig. 8 An algorithm of MOVE statement

```
statement open(HAND,WIDTH);
  begin
    parse_statement();                                          (1)
    OBJECT := get_affixed(HAND,[PART,ASSEMBLY]);
    if (OBJECT != NIL) then  unfix(OBJECT,HAND);                (2)
    code_generate(@OPEN,HAND,WIDTH);                            (3)
    return(CODE);                                               (4)
  end;
```

Fig. 9 An algorithm of OPEN statement

```
statement close(HAND,WIDTH,FORCE);
  local OBJECT;
  begin
    parse_statement();                                          (1)
    OBJECT := get_value(LAST_POSITION);
    if (OBJECT != NIL) then affix(OBJECT,HAND,rigidly);         (2)
    code_generate(@CLOSE,HAND,WIDTH,FORCE);                     (3)
    return(CODE);                                               (4)
  end;
```

Fig. 10 An algorithm of CLOSE statement

れる.

3.3.3 OPEN 文の処理 (**Fig. 9**)

(1) パーザにより構文チェックを行い, HAND, WIDTH に引数を代入する.

(2) 物体をつかんだ状態で OPEN を行ったならば, その物体に対する結合関係を解除しなければならない. そこで, HAND に結合されている物体 (PART または ASSEBMLY) を探し, あればその物体に対して UNFIX を行う. UNFIX の処理は 3.3.5 項に記述する. この処理を OPEN 内で自動的に行うことにより, OPEN 文の後に UNFIX 文を挿入する必要がなくなる.

(3) WIDTH を計算してオブジェクトコードを生成する.

(@OPEN〈ハンド〉〈幅〉)

(4) 生成したオブジェクトコードを値として返す.

3.3.4 CLOSE 文の処理 (**Fig. 10**)

(1) パーザにより構文チェックを行い, HAND, WIDTH, FORCE に引数を代入する.

(2) CLOSE は, 物体をつかむ時に実行する文である. そのときには, AFFIX 文を用いて環境モデルに対して物体の結合関係を変更する必要がある. この処理を自動化するために, LAST-POSITION の値が NIL でなければ, その物体に対して次の AL/L ステートメントを実行する.

(AFFIX OBJECT HAND RIGIDLY)

(3) WIDTH, FORCE の値を計算し, オブジェクトコードを生成する.

(4) 生成したオブジェクトコードを値として返す.

```
statement affix(FRAME,PARENT,TYPE,TRANS);
  local LINK;
  begin
    parse_statement();                                          (1)
    if (get_link(FRAME,PARENT) == NIL) then
       generate_link(FRAME,PARENT);                             (2)
    LINK := get_link(FRAME,PARENT);
    TRANS := calculate_transform(FRAME,PARENT,TYPE,TRANS);
    replace_transform(LINK,TRANS);
    replace_affixtype(LINK,TYPE);                               (3)
    return(FRAME);                                              (4)
  end;
```

Fig. 11 An algorithm of AFFIX statement

```
statement unfix(FRAME,PARENT);
  begin
    parse_statement();                                          (1)
    LINK := get_link(FRAME,PARENT);                             (2)
    if (LINK != NIL) then begin
      remove_val(FRAME,parent_link,LINK);                       (3)
     affix(FRAME,WORLD);                                        (4)
      end;
    return(LINK);                                               (5)
  end;
```

Fig. 12 An algorithm of UNFIX statement

```
(PROGRAM EX2
    (DECLARE INITIAL_POINT COORDINATES)
    (DECLARE FINAL_POINT COORDINATES)
    (DECLARE BOX PART)
    (DECLARE BOX_GRASP COORDINATES)
    (ASSERT
        INITIAL_POINT
        LOCATE_AT
        (COORDINATES NILROT (VECTOR 300 300 100)))
    (ASSERT
        FINAL_POINT
        LOCATE_AT
        (COORDINATES NILROT (VECTOR 300 -300 100)))
    (AFFIX BOX INITIAL_POINT NILTRANS NONRIGIDLY)
    (AFFIX
        BOX_GRASP
        BOX
        (TRANS (ROT YHAT 180) (VECTOR 10 10 5))
        RIGIDLY)

    (OPEN_HAND ARM1 50)
    (MOVE ARM1 BOX_GRASP (APPROACH (VECTOR 0 0 -10)))
    (CLOSE_HAND ARM1 20)
    (MOVE
        BOX
        FINAL_POINT
        (DEPARTURE (VECTOR 0 0 10))
        (APPROACH (VECTOR 0 0 10)))
    (OPEN_HAND ARM1 50)
    (MOVE ARM1 BPARK (DEPARTURE (VECTOR 0 0 -10))))
```

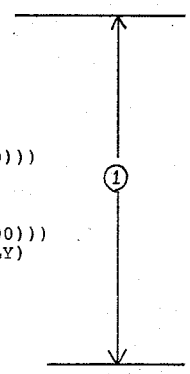

Fig. 13 AL/L program to move block

3.3.5 AFFIX 文の処理 (**Fig. 11**)

(1) パーザにより構文チェックを行い, FORCE, PARENT, TYPE, TRANS の各引数に値を代入する.

(2) FRAME, PARENT 間の関係を表すリンクフレームがあるかどうかを get-link で調べ, なければ generate-link によりリンクフレームを生成する. この手続きにより, Fig. 5 のような結合関係を表すフレームが定義される.

```
WORLD

  *BPARK
   (TRANS (EULER 0 90 0) (VECTOR 550 0 600))
     +ARM1
      (TRANS (EULER 0 0 0) (VECTOR 0 0 0))
  +INITIAL_POINT
   (TRANS (EULER 0 0 0) (VECTOR 300 300 100))
     +BOX
      (TRANS (EULER 0 0 0) (VECTOR 0 0 0))
        *BOX_GRASP
         (TRANS (EULER 0 180 0) (VECTOR 10 10 5))
  +FINAL_POINT
   (TRANS (EULER 0 0 0) (VECTOR 300 -300 100))


 (a) initial state




WORLD

  *BPARK
   (TRANS (EULER 0 90 0) (VECTOR 550 0 600))
  +INITIAL_POINT
   (TRANS (EULER 0 0 0) (VECTOR 300 300 100))
  +FINAL_POINT
   (TRANS (EULER 0 0 0) (VECTOR 300 -300 100))
  +ARM1
   (TRANS (EULER 0 90 0) (VECTOR 550 0 600))
  +BOX
   (TRANS (EULER 0 0 0) (VECTOR 300 -300 100))
     *BOX_GRASP
      (TRANS (EULER 0 180 0) (VECTOR 10 10 5))


 (b) final state
```

Fig. 14 Transition of environment model in the block move program

* represents rigid affixment

+ represents nonrigid affixment

```
(@OPEN ARM1 50)
(@MOVE
    ARM1
    (COORDINATES
        (-1 0 0 310) (0 1 0 310) (0 0 -1 115) (0 0 0 1))
    (COORDINATES
        (-1 0 0 310) (0 1 0 310) (0 0 -1 105) (0 0 0 1)))
(@CLOSE ARM1 20)
(@MOVE
    ARM1
    (COORDINATES
        (-1 0 0 310) (0 1 0 310) (0 0 -1 115) (0 0 0 1))
    (COORDINATES
        (-1 0 0 310) (0 1 0 -290) (0 0 -1 115) (0 0 0 1))
    (COORDINATES
        (-1 0 0 310) (0 1 0 -290) (0 0 -1 105) (0 0 0 1)))
(@OPEN ARM1 50)
(@MOVE
    ARM1
    (COORDINATES
        (-1 0 0 310) (0 1 0 -290) (0 0 -1 115) (0 0 0 1))
    (COORDINATES
        (0 0 1 550) (0 1 0 0) (-1 0 0 600) (0 0 0 1)))
```

Fig. 15 Generated object code

（3） リンクフレームの名前を LINK に代入しておく．AFFIX 文の引数のうち，TYPE，TRANS はオプションである．もし，これらが指定されていれば，リンクフレーム中の HOW-AFFIXED および TRANSFORM スロットの $VALUE ファセットを，これらの引数の値を計算したもので置き換える．

（4） AFFIX 文は環境モデルに作用するのみであり，オブジェクトコードは生成しない．そこで，FRAME を値として返している．

3.3.6 UNFIX の文の処理 (**Fig. 12**)

（1） パーザにより構文チェックを行い，FRAME，PARENT の各引数に値を代入する．

（2） FRAME，PARENT 間のリンクフレームを求める．

（3） もし，リンクフレームが存在すれば，そのリンクフレームを消去する．その副作用で，FRAME の PARENT-LINK および PARENT の CHILD-LINK も消去される．

（4） また，AL/L では全ての物体は必ず，直接あるいは間接的にフレーム WORLD に結合されている．そこで FRAME を WORLD に結合する．

（5） UNFIX も AFFIX 同様，環境モデルに作用するのみである．UNFIX 文は LINK を値として返す．

3.4 実験および考察

Fig. 13 に箱の移動プログラムを示す．①の部分は環境の宣言部であり，この部分の処理を行った後の環境モデル内の状態は，**Fig. 14**（a）のようになる．ここでは物体に関するフレームの座標値のみを表している．このプログラムを実行後の環境モデル内の状態は，Fig. 14（b）のように変化する．そして，**Fig. 15** に示すオブジェクトコードが生成される．この処理にミニコン上の Lisp インタプリタを用いて約 200 秒かかった．AL/L のソースステートメント 1 行当り 10 秒程度の処理時間を要することになる．ちなみに，VAX-780 の Franz Lisp でも約 200 秒，コンパイルすると約 20 秒になった．

AL/L 処理系の処理速度はかなり遅い．これは，FRL に基づく環境モデル管理システムを採用しているからで

ある．データを更新するたびに，継承メカニズムをたどって，概念フレーム中の手続きを探し，それを実行する．実際問題としては，ロボット言語処理系における環境モデルの管理を行うには，FRL を用いる必然性はなく，また，FRL の仕様が全て必要というものではない．必要十分な機能の抽出と，それを実現する高速の環境モデル管理システムの実現が必要である．

AL/L の環境モデルは，結合関係におけるループのチェック以外，モデル内の論理構造についてはほとんどチェックしていない．それらはプログラマに責任が委ねられている．しかし，環境モデル管理の大きな目的は，むしろモデルの首尾一貫性のチェックにある．エラー発生時の処理方法も含めて，環境モデルの管理手法についてさらに研究が必要である．

4. 総合プログラミングシステム

前章では，高位レベルのロボット言語 AL/L について述べた．本章では，COSMOS のシステム構成について述べる．ロボット言語により記述されたプログラムは，言語処理系により低レベルの記述に展開される．展開された記述は，実行システムに送られ，その記述を実行するために必要なサブシステムのコマンド列に展開される．さらに，そのコマンドを受け取ったサブシステムは，各モジュールに対してコマンドを送る．このように考えると，ロボットのプログラミングシステムの構成は，階層構造にすることが自然であり，COSMOS もその方針をとった．

Table 3 Primitive commands

Arm Functions	status	CUR-COORD,CUR-POS,CUR-JOINT,WIDTH, ERROR-CHECK
	motion mode	VELOCITY,LOADING,FORCE,SERVO,MOTION, ALLOW-ERR,INALLOWERR
	motion	MOVE,MOVE-X,MOVE-Y,MOVE-Z,REACH,SWEEP, LIFT,ROLL,PITCH,YAW,J-MOVE,OPEN-HAND, CLOSE-HAND
	miscellanious	DATASAVE,DATASAVEEND,DATAMOVE,ARM-RESET, ZEROING
Tactile Sensor Functions		SENSOR,STOP-SENSOR,SENSORSTAT,SENSORVALUE, THRESH
Simulate Functions	arm simulate	SIMULATE,DISP-COORD,PLOT-COORD, MANIPULATOR,CUBE
	3D graphics	INITG,VIEWPORT,VIEW,SCREENSIZE,INITTRANS, CLEAR,WIREFRAME,MOVETO,DRAWTO,TRANSLATE, ROTX,ROTY,ROTZ,SCALE,INTENSITY, WORLDTOVIEW,WORLDTOSCREEN,SAVETRANS, SETTRANS,DRAWTEXT
Vision Functions	data acquisition	STILL,MEMORY,WINDOWDUMP,WINDOWLOAD, IMAGEDUMP,IMAGELOAD,WINDOW
	visual processing	DIMINISH,EDGE,THRESHOLD,THINNING, FIND-BLOCK,FIND-LINES

4.1 コマンド体系

ロボットシステムは，いくつかのサブシステムより構成されるトータルシステムである．そして，システムを使って応用プログラムを作る場合には，システム全体の細かな構成を気にしなくてもいいようなものにしておかなければならない．そこで，トップレベルで各サブシステムを動作させるための基本コマンドを用意しておき，サブシステムの内容はブラックボックス化する方針をとった．

COSMOS では，ロボットのハードウエアとして，物体操作機能を実現するためのロボットアーム，視覚情報処理用のテレビカメラ，感覚機能として触覚センサを装備した．これらについて基本コマンドを設定した．それは，**Table 3** のように分類される．これ以外のデバイスについても，同様なコマンドを設定することによりシステムへの組込みは容易にできる．

トップレベルは Lisp インタプリタであり，これらの基本コマンドは，それぞれ Lisp の関数として定義されている．したがって，ユーザはこれらの関数を用いて新しく関数を定義してゆけば，独自のロボットシステムを容易に構築することができる．例えば物体を動かすことなく，手の位置を物体の中心に移動させて把握する動作 CENTER は，基本コマンドを使って **Fig. 16** のように定義できる．このようにして定義した関数を用いて，さらに別の関数を定義してゆくことにより，システムを容易に構成できるわけである．

4.2 ハードウエア構成

Fig. 17 にシステムのハードウエア構成，**Fig. 18** にシステム全景を示す．行動機能として，物体操作機能の実現を考え，6+1 自由度の関節型汎用アーム[10]を使用している．DC サーボモータ駆動であり，可搬重量は約 1 kg である（**Fig. 19**）．

視覚情報の入力装置としては，テレビカメラに接続されたイメージフレーム・メモリ（768×512×8）を用い，ホストのミニコンピュータ（データゼネラル社 Eclipse S/140）にインターフェイスされている．

イメージ・フレーム・メモリの内容はモニタテレビに表示されるので，この装置はグラフィックディスプレイ装置としても使用される．さらに，テレビカメラと併用して，3次元距離測定用のレーザスポット・スキャナが接続されている．

触覚センサは，ストレインゲージを使ったもので，分解能 0.3 g 以下と敏感であり，2本の指の内側に1個ずつ取り付けられている[11]．センサの処理はテキサスインスツルメント社の 16 ビットマイクロプロセッサ TMS 9900 を用いており，このプロセッサはアームコントロール用の TMS 9900 とパラレルポートで接続されている．ホストのミニコンとこれらのマイコンは，GPIB でインターフェイスされている．

ユーザの入出力装置としては，グラフィックターミナル，タブレット，音声入力装置が接続されている．仮想端末の処理を行うため TMS 9900 を用いており，RS 232 C を介して Eclipse に接続されている．

```
(DEFUN CENTER NIL
    (LET ((L NIL) (VSAVE (VELOCITY)))
         (VELOCITY 20)
         (STOP-SENSOR)
         (OPEN 60)
         (THRESH 0 (+ (SENSORVALUE 0) *THRESH*))
         (THRESH 1 (+ (SENSORVALUE 1) *THRESH*))
         (SENSOR 0 'ON)
         (SENSOR 1 'ON)
         (OPEN 0)
         (SETQ L (SENSORSTAT))
         (COND
             ((AND (CAR L) (NOT (CADR L))) (CENTER0 1))
             ((AND (NOT (CAR L)) (CADR L)) (CENTER0 0)))
         (VELOCITY VSAVE)))

(DEFUN CENTER0 (N)
    (LET ((D (READ-POS)))
         (STOP-SENSOR)
         (THRESH N (+ (SENSORVALUE N) *THRESH*))
         (SENSOR N 'ON)
         (SWEEP (COND ((EQUAL N 0) 70) (T -70)))
         (SETQ D (DISTANCE D (READ-POS)))
         (STOP-SENSOR)
         (SWEEP
             (COND
                 ((EQUAL N 0) (- (/ D 2)))
                 ((EQUAL N 1) (/ D 2))))))
```

Fig. 16　CENTER motion written in primitive commands

4.3　ソフトウエア体系

COSMOS ソフトウエアは，次の7つのサブシステムから構成されている．

（1）　トップレベル
（2）　ロボット言語 AL/L
（3）　アームコントロール・システム
（4）　アームシミュレータ
（5）　視覚システム

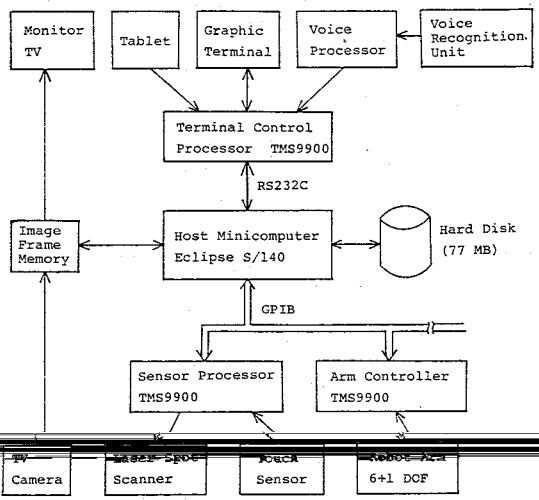

Fig. 17　System configuration of COSMOS

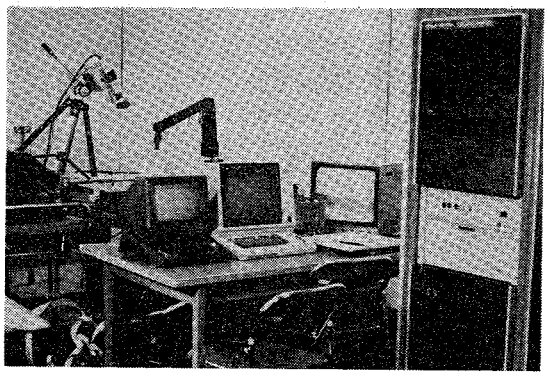

Fig. 18　General view of COSMOS

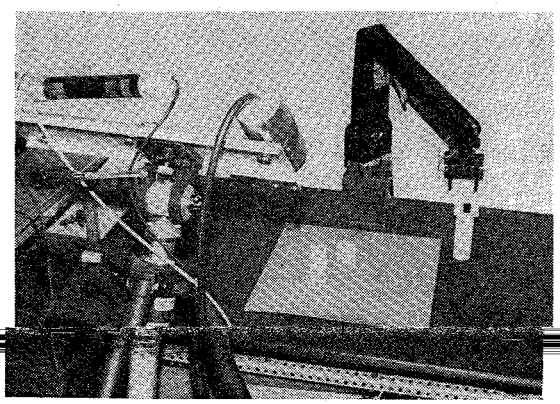

Fig. 19　Hand-Eye device

（6） アームコントローラ

（7） センサプロセッサ

（1）から（5）までがミニコン上で走り，（6）と（7）はそれぞれ別のマイコン上で走る．ミニコン上のOSは，マルチユーザ，マルチプロセスをサポートするDG社のAOS (Advanced Operating System) であり，その下で，システム開発用ソフトウエアおよびCOSMOSのシステムソフトウエアが走っている．各プロセスはプロセス間通信機能 IPC (Interprocess Communication) を用いて他のプロセスと交信を行うことができる．COSMOSソフトウエアは膨大なものになるが，Eclipse S/140の論理アドレス空間は64 kbyteであり，とても一つのプログラムにはおさまらない．そこで，ソフトウエアをいくつかのプロセスに分割し，データのやりとりはIPCを用いて行うことにした．

上位のソフトウエアはLispで書かれている．サブシステム毎に別のLispインタプリタが走り，サブシステム間はコマンドの受け渡しで交信している．下位の数値

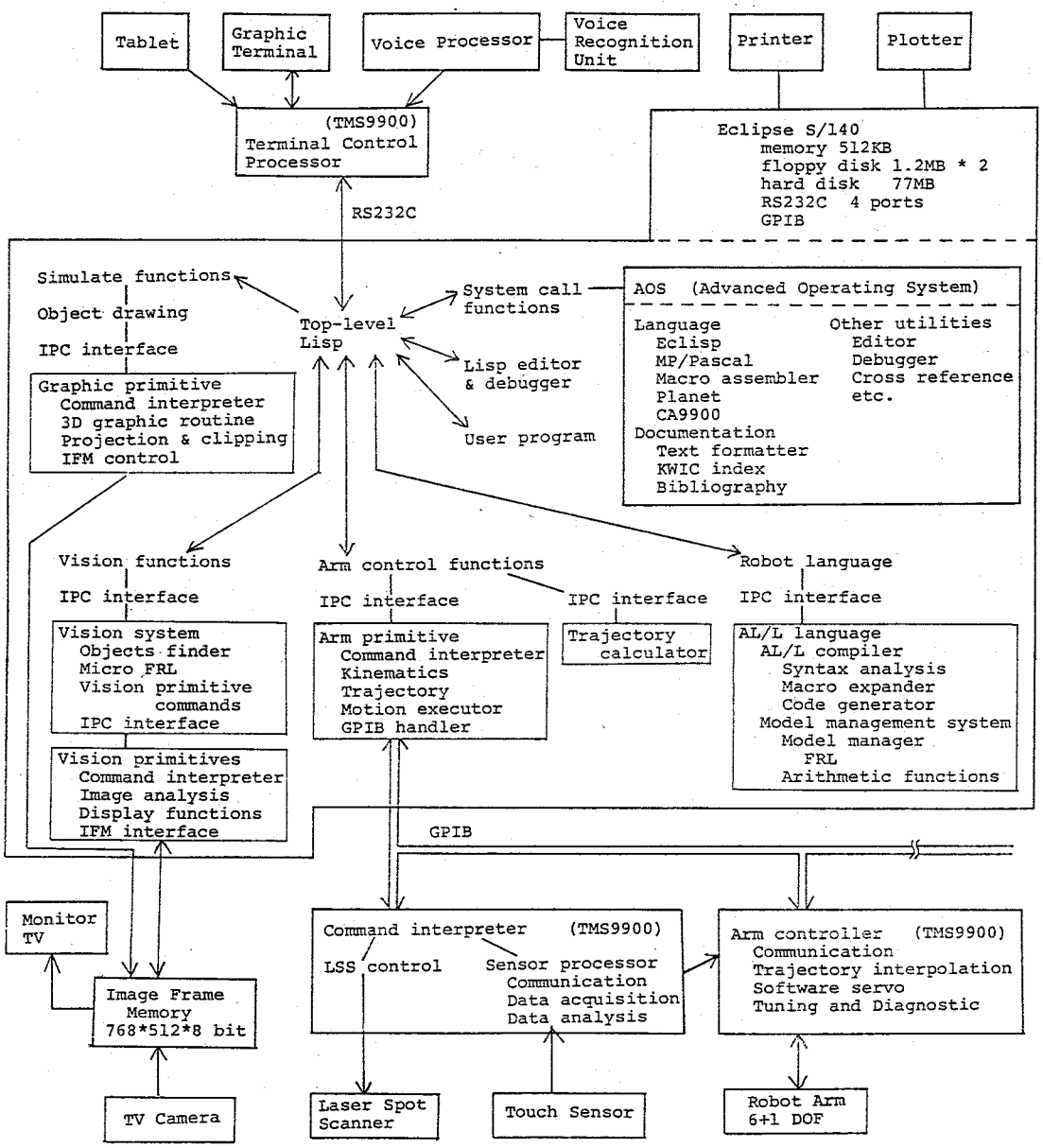

Fig. 20 Construction of COSMOS software

演算を多用する部分は Pascal を用いている．また，マイクロコンピュータ上のソフトウエアは，Pascal 風の高級言語 Planet[12] および，アセンブラで記述されている．Lisp インタープリタ，Planet クロスコンパイラ，およびクロスアセンブラは全て自前のものであり，ミニコン上で走る．COSMOS のソフトウエア体系は，**Fig. 20** のようにまとめられる．

（1） トップレベル

ユーザとのインターフェース部分である．Lisp インタープリタであり，他のサブシステムを起動するためのプリミティブコマンドが関数として定義されており，ユーザは，それを用いて会話形式でロボットの動作を定義し，実行する．

Lisp インタープリタは本システムのために開発したものである[13]．Eclipse の論理アドレスが 64 kbyte と狭いため，ひとつの Lisp インタープリタのフリーセルは 10 k セルであり，2000 行程度のプログラムしか実行できない．そこでプロセス間通信機能を用いて，他の Lisp プロセスと交信しながら処理を行う方式にした．プロセスを分割することは，プログラムのモジュール化を推し進めることになり，システムを構成する上ではむしろよかったと考えている．プログラミング環境として，エディタ，デバッガ，ファイル操作等が組込まれている．

（2） ロボット言語 AL/L

ロボットの作業記述言語の処理系であり，動作レベルで記述されたプログラムを，プリミティブコマンドに展開する．ロボットの作業プログラムを処理する場合には，動作の進行に伴う作業環境の変化を矛盾なく管理する必要がある．そのため，本システムでは FRL を用いた環境モデル処理システムが組込まれている．このモジュールは，パーザ部，環境モデル管理部，マトリックスやベクトルの数値演算部の 3 つのプロセスに分割されている．プログラムは Lisp で記述されている．

（3） アームコントロール・システム

ロボットアームの物体操作機能を制御するサブシステムである．AL/L 言語処理系は基本コマンド列を生成する．この基本動作コマンドを受けとり，アームの幾何学的計算，運動軌道の計画を行ってアーム駆動用コマンドを発生する．基本動作コマンドでは，直交座標系に基づいた目標値が指定される．これをロボットアームの関節ごとの目標値列に変換するとともに軌道の計算，補償量の計算などを行う．軌道計算は，Paul の多項式近似[14]，3 次式近似，直線近似の 3 つのアルゴリズムが選択できるようになっている．この部分はほとんど Pascal で記述されている．コマンドは，GPIB バスを経由してアームコントローラ，およびセンサプロセッサへ転送される．

（4） アームシミュレータ

直接ロボットを動かす代わりに，グラフィックディスプレイ上で動作をシミュレートするものであり，ロボットの動作プログラミングのデバッグの補助手段として用いる．3 次元グラフィック処理が行われる．3 次元グラフィック処理部は Pascal で記述されたサブプロセスになっている．シミュレート部は Lisp で記述されている．

（5） 視覚システム[15]

ロボットの視覚に関する処理を行う．濃淡レベルの画像に対する基本処理部と 3 次元の物体認識の部分に大別される．前者は，役割別に分類したプログラムモジュールの集合である．各モジュールは Pascal で記述されており，上位のシステムと IPC により通信する．後者は，基本処理部へのコマンド群を用いて Lisp で記述された視覚情報の処理プログラム群を含んでいる．線画の抽出，両眼立体視，視覚フィードバック機能等が組込まれている．

（6） アームコントローラ

直接ロボットアームの制御を行うサブシステムであり，TMS 9900 上にインプリメントされている．ホストの Eclipse とは GPIB で接続されており，コマンドを受取って，軌道の補間，ソフトウェア・サーボを行う．またアームの診断および調整機能も備えている．アセンブラで記述されている．

サーボの間隔は 5.2 msec であり，データの転送量を減らすため，ホストからは 1/6 秒ごとの目標値を送り，コントローラ内で補間している．重力補償，加速度補償等のパラメータはホストから設定できるようになっている．また，センサプロセッサとはパラレルポートで接続されており，感覚器官からの高速フィードバックが利用できるようになっている．

（7） センサプロセッサ

歪ゲージを用いた触覚センサの信号処理を行う．左右の指先に 1 個ずつ触覚センサが付いており，それらのモニタリングを行う．センサの分解能は 8 bit である．センサのモニタリングは，閾値との比較によって行われ，閾値より大きな値あるいは小さな値になれば，パラレルポートを通して信号がアームコントローラに送られる．ステータスのチェックおよび閾値の設定はホストから行える．センサを利用した関数の具体例は，Fig. 16 の CENTER 動作に示されている．

このほか，レーザ・スポット・スキャナの制御も行う．TMS 9900 上で走り，Planet で記述されている．

5. システム応用例
——タブレットを用いた教示——

本システムを用いて，各種の知能ロボットに関する研究が行われた．ここでは，本システムを使った応用例としてタブレットを用いた教示システム[16]について述べ，システム全体の動きについて解説する．

COSMOS では，ユーザがロボット言語を介してシステムに指令を与える．その場合には，ロボットの置かれている環境を全て教示しなければならない．しかし，環境をプログラム言語で記述するのは非常に煩雑であり，わずか数行で記述できる動作を実行するために，その数倍の環境情報を入力しなければならない．この傾向は，ロボット言語が高水準になるほど顕著になる．この問題を解決するためには，使い勝手のよい環境教示システムが不可欠となる．そこで，COSMOS ではタブレットを入力装置として用いた教示システムを構築した．

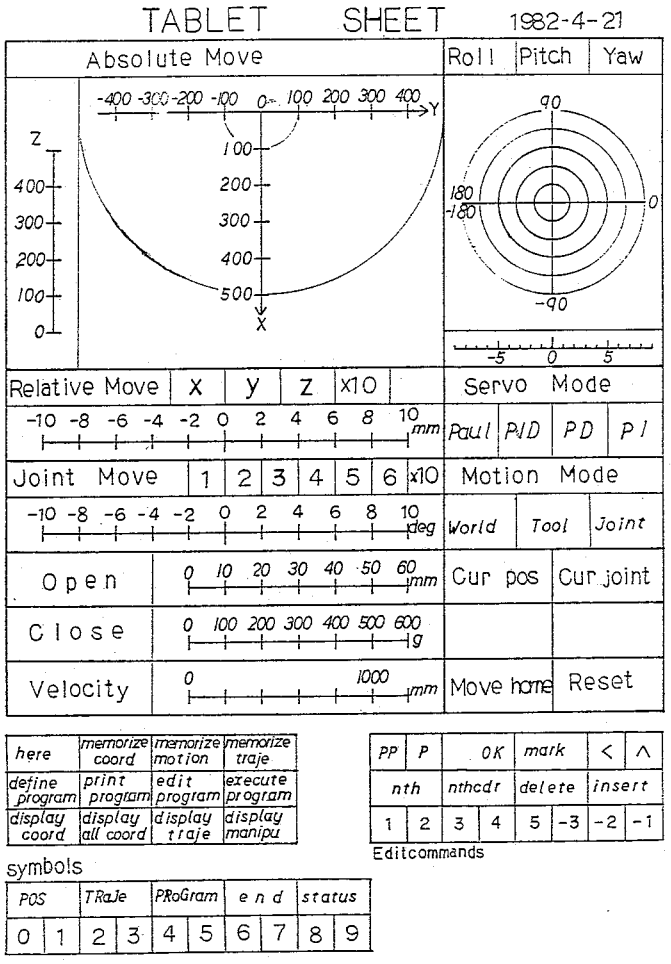

Fig. 21 A tablet menu sheet

タブレットを用いる場合の長所は，

（1） ストローク数が少なくてすみ操作が簡単である．

（2） 多くのコマンドがメニュー方式で使用可能である．

（3） 他の入力装置は不要である．

（4） シートにメニューやその解説を書けばマニュアルの代用になる．

等の点があげられる．

Fig. 21 にタブレットシートの例を示す．このシートのメニューは，（1）アームの動作，（2）動作モードの設定，（3）位置，軌道の定義，（4）表示，（5）動作のプログラミング，より成っている．シートの上部3分の2がアームの動作および動作モードメニューである．動作メニューを用いてアームを教示した点にもってゆき，その点を HERE コマンドで定義する．これをくりかえすことにより，プログラムに必要な点の位置・姿勢を簡単に定義することができる．動作メニューでは，タブレットの特徴を生かして2次元的な位置の指定を可能にしている．また，相対移動についてはワールド座標系あるいはハンド座標系に基づいた動きが指定できるとともに，スケールファクターを用いて，目盛の単位を 10 倍にすることが可能である．

左下のメニュー群は，点や軌道の定義および表示，さらにプログラミングのメニューである．その下のシンボルテーブルを使って，点，軌道，およびプログラム名を入力する．ここでは，それぞれ 10 個ずつ定義できるようになっている．MEMORIZE コマンドで動作をプログラム中に記憶する．このコマンドで生成される動作列には，

（1） 絶対位置による動作方式

```
(MOVE '(COORDINATES
                (1 0 0 100)
                (0 1 0 200)
                (0 0 1 300)
                (0 0 0 1)))
```

（2） 点の名前による動作形式

```
(MOVE COORD:1 COORD:2
COORD : 3)
```

（3） 軌道の名前による動作形式

```
(MOVE-T TRAJE:1)
```

の3種類がある．

右下のメニュー群は，プログラムのエディティング用のものである．Lisp の構造化エディタを利用することができる．

ユーザはタブレットを使って，対話形式でロボットの環境・動作の教示を行なうことができる．コマンドはタブレットからメニューで入力され，ターミナル・コントロール・プロセッサにより次のような仮想端末の入力形式に変換されてホストに送られる．

(TABLET-CMD x y flag)

x, y はタブレット上の位置，flag はペンのスイッチの on/off を表す．

トップレベルでは TABLET-CMD という関数が用意されている．この関数では，引数を解釈してメニューを判断し，実行に必要なサブシステムを起動する．アームを動かす場合には，アームコントロール・システムに対してコマンドが送られ，各モジュールを経由してアームコントローラへ目標値が送られる．アームコントローラは，与えられた目標値を補間しつつサーボを行い，ロボットを制御する．このときのデータの流れは，**Fig. 22** のようになる．動作をシミュレートする場合には，アームシミュレータに対してコマンドを送ると，そこでアームの姿勢，あるいは定義されている点のデータをもとに，3次元プロット用のコマンドを発生し，プロットルーチンがグラフィックディスプレイ上に作図する．

この教示システムは非常に短期間のうちに実現でき，COSMOS システムの使い易さ，拡張性が確認された．ここで示したタブレットシートはあくまで実験的なものであり，また問題点が多く改良の余地がある．メニューをさらに豊富に，また系統化する必要がある．また2次元入力装置という特徴を生かして，軌道の入力や物体の形状の入力にも拡張できる．

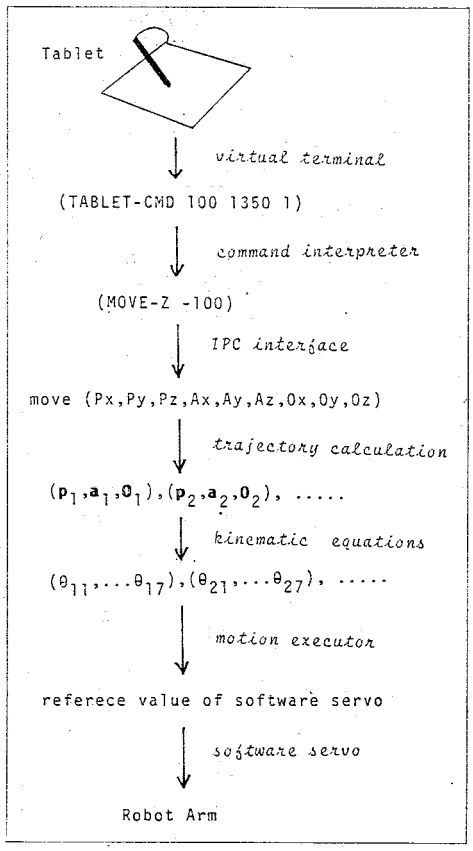

Fig. 22 Data stream from tablet to robot arm

6. 考察および結論

COSMOS を用いて各種の知能ロボットに関する研究[17,18]が行われ，その成果はシステム内に取り入れられ，COSMOS は着々と進化している．その過程を通じ，COSMOS のロボット・プログラミングシステムとしての価値が認識されるとともに，いくつかの問題点が明らかになった．

以下に考察および今後の課題を要約する．

（1） COSMOS は，トップレベル Lisp のもとに，ロボット言語，アームの制御，アームシミュレータ，視覚システム等を統合した形で，使い易い，ロボットのプログラミング環境を提供している．知能ロボットについてのいろいろな実験は，基本コマンドを使って Lisp で容易に，かつ短時間のうちにプログラム可能である，COSMOS は柔軟性に富む研究ツールになったといえる．

（2） ロボット言語 AL/L は，現在のところ動作レベルの言語であるが，今後さらに高水準なレベルへ拡張する予定である．このためには，必然的に問題解決の能力が必要となり，人工知能の手法の組み込みが今後の課題である．上位のソフトウエアが Lisp で記述されているため，この方向への発展は比較的容易に実現できるはずである．

（3） ロボット言語の研究を通して，環境モデルの管理システムが極めて重要な役割を占めていることが明らかになった．今後は，環境モデルを中心として，センサとの高速のインタラクションをとるようなシステムの開発が重要な課題である．それと関連して，応答の速いロボット用ランタイム OS の研究が必要である．

（4） 作業および環境の教示については，タブレットのほかに，6自由度ジョイスティック[19]，音声入力方式[20]などについて実験を行ったが，タブレットが使い易い．タブレットだと，細かいところまで指定できること，トップレベルに送るデータが少量ですむためである．メニューについてはまだまだ改善の余地がある．また，他の入力装置に関しては，細かな動作を指定できないのが欠点であるが，これを改善するにはロボット言語が対象

物レベルあるいは作業レベルに高められていることが前提となる．

（5） COSMOS システムの現在の最大の問題は，計算機の能力不足である．Eclipse は，ひとつのプロセスに対して，64 K バイトしかメモリを割付けない．このため，Lisp のセルは 10 K しかとれず，大きなプログラムは走らない状態にある．これを解決するために，いくつかのプロセスに分割し，プロセス間で交信を行う方法を採用したが，この方法だと，どうしてもレスポンスが遅くなってしまう．これを解決するためには，アドレス空間の大きな計算機を用いたシステムの開発が早急の課題である．

7. むすび

知能ロボット・プログラミングシステム COSMOS について，その構成を述べた．CSOMOS は，あくまでもプロトタイプシステムである．システム開発および COSMOS を利用した実験を通じて明らかになった問題点を解決し，より高度なシステムを開発することが次の課題である．

〔謝 辞〕 CSOMOS の研究に協力してたださった東京大学機械工学科情報システム工学研究室の卒業生ならびに現メンバーの城下修，内藤理，川越誠司，松井俊浩，溝口博，稲葉雅幸，藤田正弘，松原仁，岡野彰，藤田裕二，多川明の諸氏に深謝する．

参考文献

1) S. Mujtaba, R. Goldman, "AL Users' Manual", Stanford University Artificial Intelligence Laboratory, January 1979
2) R. Taylor, P. Summers, and J. Meyer, "AML : A Manufacturing Language", Robotics Research, Vol. 1, No. 3, pp. 19-41, 1982
3) J. Albus, "Brains, Behavior, and Robotics", Hc Graw-Hill, 1981
4) P. Winston and B. Horn, "Lisp", Addison-Wesley, 1981
5) D. Falek and M. Parent, "An Evolutive Language for an Intelligent Robot", The Industrial Robot, pp. 168-171, 1980
6) H. Inoue, T. Ogasawara, O. Shiroshita, and O. Naito, "Design and Implementation of High Level Robot Language", Proceesings of 11 th ISIR, pp. 675-682, 1981
7) M. Minsky, "A Framework for Representing Knowledge", in P. Winston ed., "The Psychology of Computer Vision", McGraw-Hill, 1975
8) R. B. Roberts, I. P. Goldstein, "The FRL Manual", AI Memo 409, MIT, September 1977
9) 城下 修，小笠原 司，井上博允，"ロボット言語における作業環境モデル処理システム"，情報処理学会第 23 回全国大会講演論文集，pp. 711-712，1981
10) 井上博允，岡本憲治，"研究用標準型汎用ロボットアームの開発"，第 19 回 SICE 学術講演会予稿集，No. 3717，pp. 585-586，1980
11) 溝口 博，井上博允，"歪ゲージを用いた高感度触覚センサ"，第 22 回 SICE 学術講演会予稿集，pp. 597-598，1983
12) 松井俊浩，"マイクロコンピュータ用プログラム言語 Planet の開発"，東京大学機械工学科井上研究室，1980
13) T. Ogasawara and T. Matsui, "ECLISP User's Manual", Inoue Lab., University of Tokyo, 1981
14) R. Paul, "Modeling, Trajectory Calculation, and Servoing of a Computer Controlled Arm", Stanford Univ. AI Lab. Rep. AIM 177, 1972
15) 稲葉雅幸，井上博允，"知能ロボットプロトタイプ COSMOS の視覚システム"，第 1 回日本ロボット学会学術講演会予稿集，pp. 31-32，1983
16) 小笠原 司，井上博允，"タブレットを用いたロボットの作業教示システム"，第 21 回計測自動制御学会学術講演会予稿集，pp. 355-356，1982
17) H. Inoue and M. Inaba, "Hand Eye Coordination in Rope Handling", Proc. of ISRR, 1983
18) 岡野 彰，井上博允，"対象物レベルのロボット言語の研究"，第 1 回日本ロボット学会学術講演会予稿集，pp. 39-40，1983
19) 川越誠司，"ロボットアームの作業教示システムの研究"，東京大学大学院工学系研究科修士論文，1982
20) 佐川和幸，井上博允，"音声を用いたロボット操作システム"，第 1 回日本ロボット学会学術講演会予稿集，pp. 145-146，1983
21) 井上博允，小笠原 司，"知能ロボットのシステム構成法"，昭和 57 年電気四学会連合大会，1982
22) 小笠原 司，井上博允，"知能ロボット実験システムのソフトウェア構成"，第 25 回自動制御連合講演会予稿集，pp. 297-298，1982

小笠原 司
(Tsukasa OGASAWARA)

昭和 30 年 10 月 16 日生れ．53 年 3 月東京大学工学部計数工学科卒業．58年 3 月同大学院工学系研究科情報工学専門課程博士課程修了．同年 4 月電子技術総合研究所入所．知能ロボットの研究に従事．計測自動制御学会，情報処理学会の会員．工学博士．

（日本ロボット学会正会員）

井上博允 (Hirochika INOUE)

昭和 17 年 7 月 5 日生れ．40 年東京大学工学部産業機械工学科卒業．45年同大学院博士課程修了，工学博士．45年電子技術総合研究所入所．53年 4 月より東京大学助教授工学部機械工学科．現在，同教授．知能ロボット総合システムの研究中．日本機械学会，情報処理学会，電子通信学会，計測自動制御学会，IEEE，ACM の会員．

（日本ロボット学会正会員）

COSMOS : A Total Programming System for Integrated Intelligent Robot*

Tsukasa OGASAWARA** Hirochika INOUE***

ABSTRACT

This paper describes robot language and software constructs of total intelligent robot, that is designed and implemented as general purpose research tool for robot system studies. The system is called COSMOS which is an abbreviation of Congnitive Sensor Motor Operations Study. It consists of high level robot language, arm and its control system, three dimensional vision, sensitive tactile sensor, and user interfaces. A host minicomputer and microcomputers are connected with the GPIB bus, and they communicate using primitive robot commands. The system consists hierachical control structure, and has modularity and extensibility in both hardware and software. For convenience of future extensions toward artificial intelligence technique, major parts of the system are implemeted in Lisp. Description of this paper includes robot language AL/L (Assembly Language in Lisp) and it's environment model manager, structure of integrated robot software, and programming through tablet menu. First, design principle is described. Then, the language syntax of AL/L and construction of it's language processor is explained. We recognize that an environment model manager plays an important role in the compilation of manipulation programs written in robot language. Through our development of teaching programs, it became clear that tablet is a good teaching device, and the system is effective for robot programming. This paper also discusses technical problems for future study.

Keywords : Robot programming, Robot software, Intelligent robot, Robot language, Environment model

* Received June 21 1984

** Automatic Control Division, Electrotechnical Laboratory
*** Faculty of Engineering, Tokyo University