SCARA 형 로보트의 프로그램형 언어 구성에 관한 연구

이 기 동*, 고 명 삼, 하 인 중, 이 범 희
서울대학교 제어계측공학과 로보트·지능시스템 연구실

A Study on a programming Language of SCARA type Robot

Ki Dong Lee, Myoung Sam Ko, In Joong Ha, Bum Hee Lee
Dept. of Control & Instrumentation Eng., Robotics & Intelligent Systems Lab.
Seoul National University

ABSTRACT

In this paper, the design method, design techniques and structure of a language for a SCARA type industrial robot, are presented. The proposed new language is modular and expandable using the C programming language and the 8086 assembly language. It is composed of monitor mode, editor mode, execution mode, I/O mode and teach mode.
The developed language is implemented on the robot controller to verify its performance.

1. 서론[2),7),8),9)]

로보트 프로그래밍 언어(Robot Programming Language)는 로보트의 종류에 따라 서로 다르며 그언어들은 마이크로 컴퓨터 단계(Micro-computer level)에서 인간지능단계(Human Intelligence level)까지 여러 단계가 있다. 그림 1은 여러가지 로보트 언어의 진화과정을 나타낸다.[1),11),12),13),14)] 일반적으로 1960년대의 산업용 로보트는 교과서적인 프로그래밍 언어(Textual Programming Language)를 사용하지 않고 단순한 교시(teach)와 반복(repeat)의 방법에 의한 도장(Painting)이나 스폿트 용접(Spot welding) 등에 사용되어 왔다.[2)] 그러나 1960년대 후반부터는 교과서적인 프로그래밍 언어를 사용하게 되었는데 그 이유는 팰레타이징(Palletizing) 같은 경우는 교시(teach)하는 것보다 계산하는 것이 훨씬 편리하며, 정확하고, 오프라인 프로그래밍 언어(off-line Programming Language)의 필요성이 대두되었고, 센서(sensor) 정보를 이용할 수 있다는 장점이 있기 때문이다. 이것은 MIT의 Hand-eye 프로젝트에서 최초로 시도하였다.[6)] 그 이후 1973년 미국의 Stanford 인공 지능 연구소에서 로보트의 작업 수행능력 보다 로보트의 이론적인 한계사항을 발견하기 위해서 WAVE란 언어를 개발하였고, 1974년에는 ALGON을 기초로 하여 여러개의 로보트를 평행으로 두고 공동 작업을 제어하기 위한 AL을 발표하였다. 이것은 조립작업용으로 만든 최초의 언어이다. 그이후 IBM의 T.J. Watson 연구소에서 직각좌표형 메니퓰레이터(Cartesian Manipulator)를 제어하기 위해서 EMILY, ML, AUTOPASS, AML 등의 언어를 발표하였다. 또한 1976년에는 Unimation 사에서 최초로 상업화된 로보트 프로그램밍 언어인 VAL을 개발했는데 이것은 Stanford의 AL과 흡사하며 Basic 언어를 기초로 하여 개발했다. 현재는 좀더 확장된 VAL-II가 발표되었다.[16),17)] 그밖에 Cincinnati Milacron의 T_3, McDonnell Douglas의 MCL[4)], Automation 사의 RAIL, General Electric의 HELP 등이 개발되었다.[16)]

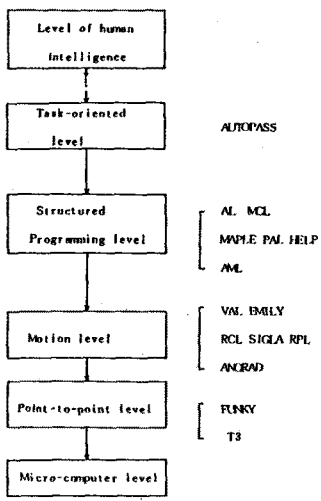

그림 1. 로보트 언어의 분류
Fig.1 Classification of Robot Language

2. 로보트 언어의 구성 및 개발

2.1 로보트 언어의 일반적인 구성

일반적으로 로보트 언어는 모니터 모우드(Monitor Mode), 에디팅 모우드(Editing Mode), 수행 모우드(Execution Mode)등이 있다. 여기에 교시 모우드(Teaching Mode), I/O 모우드 등이 있고, 공통적으로 사용하는 알고리즘(Algorithm)이 있을 수 있다. 이것의 구성을 간단한 블럭선도로 나타내면 그림 2 와 같다.

모니터 모우드는 전체 시스템의 주 흐름이 되며 제어기 운

전의 시작과 완료 과정을 관장한다. 또한 각 모우드를 관할한다. 에디팅 모우드는 매니퓰레이션 프로그램을 새로이 만들 뿐만 아니라, 이미 만들어 놓은 프로그램을 수정하여, 편집하는 기능과 동시에 프로그램 생성 및 편집시에 작성된 프로그램이 명령어 형식 및 문법(Syntax)에 맞는 것인가를 점검하는 파싱(Parsing)도 수행하게 된다. 물론 이 에디팅모우드에 인터프리터가 선택되었을 경우에는 중간코드(Intermediate Code)까지 작성된다. 수행 모우드는 사용자가 에디팅 모우드에서 작성한 프로그램을 원하는 실행 횟수만큼 실행으로 옮기는 기능과 작성한 프로그램을 한 스텝씩 실행시키는 디버깅기능 등을 수행한다. 교시 모우드는, 교시상자(teaching pendant)를 이용한 제어에 필요한 프로그램이며, I/O 모우드는 외부의 기계나 장치등과 연결해서 사용할 수 있게 한 부분이고 알고리즘은 Kinematics, Inverse Kinematics 및 Path Planning등에 관련된다.

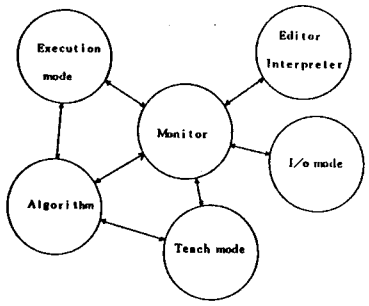

그림 2. 로보트 언어의 구성도

Fig.2 Block Diagram of general Robot Language Configuration

2.2 개발된 로보트 언어의 구성[9)]

본 논문에서 제의된 로보트 언어는 모니터 모우드, 에디팅 모우드 및 수행모우드의 세가지 모드로 구분된다. 한편 프로그램별로 구별한다면 모니터 모우드, 에디팅모우드, 수행 모우드, 중간코드 작성모우드, 교시모우드등으로 구별할 수 있다. 표 1은 현재까지 작성된 프로그램의 기능과 그것의 명칭과의 관계를 나타내며 모든 프로그램은 C-언어로 작성되었다.

표 1. 작성된 프로그램의 내용

프로그램 이름	기능
MAIN	모니터 모우드 프로그램
EDITOR	에디터 모우드 프로그램
INTER-COM	중간코드 작성 프로그램
EXEC	수행 모우드 프로그램
RUN-COM	명령어 수행 프로그램
MLIB-C	알고리즘 및 산술적계산 프로그램
MCOM-10	모니터 모우드 명령어 수행 프로그램
MICOM-20	모니터 모우드 명령어 수행 프로그램 및 데이터 초기화
MICOM-30	데모(Demo) 프로그램 및 Path-planning 교시 모우드와 인터럽터(interrupt) 프로그램

2.3 개발된 로보트 언어의 명령어

여기서 명령어는 프로그램 자체에서 이용하는 모우드 명령어와 로보트의 작업을 규정하는데 이용하는 매니퓰레이션 명령어로 구별한다.

2.3.1 모우드 명령어

모우드 명령어는 다시 모니터 모우드 명령어와 에디터 모우드 명령어로 구별된다. 모니터 모우드 명령어는 모두 14개이며 그 종류는 아래와 같다.

EDIT, INITIAL, SPEED, RUN, HERE, TEACH, LDIR, LSET, LLIMIT, ULIMIT, LDELETE, ZERO-RETURN, FDIR, FDEL

다음 에디터 모우드 명령어는 모두 7개이며 그 종류는 아래와 같다.

D(delete), L(list), I(insert), Q(quit), E(end), T, TS

2.3.2 매니퓰레이션 명령어[3),4)]

실제 로보트 몸체의 운동을 제어하는 매니퓰레이션 명령어들은 로보트의 종류에 따라 서로 다르나 일반적으로 그 기능을 분류하면 다음과 같다.

i) 로보트의 손을 제어하는 명령어
ii) 로보트의 팔의 운동을 제어하는 명령어
iii) 외부 출력을 제어하는 명령어
iv) 프로그램을 제어하는 명령어
v) 기타의 명령어

명령어들의 문법구조라는 것은 각각의 명령어들에 따라 미리 정해진 형식을 의미한다. 이것은 프로그램을 작성할때 직접 조사하여 잘못된 입력을 가려내는데 사용하며 잘못이 없을 경우에만 프로그램이 저장된다. 또한 중간코드 구조라는 것은 명령어들이 문법구조에 틀린점이 없을 경우 이것을 나중에 실제로 수행할때 수행의 편리를 위하여 미리 정해진 토큰(Token)을 이용하여 프로그램을 개조하여 저장하는 구조를 말하며 중간코드와 원래의 입력파일을 동시에 저장해 두는 것은 중간코드는 수행시에 사용하고 입력파일은 화면에 표시할때 유용하게 사용한다.

예를들면 FOR-TO-STEP의 명령어 같은 경우 그 문법구조는 다음과 같다.

FOR <변수1> = { <변수2> / <정수> } TO { <변수3> / <정수> } [STEP { <변수4> / <정수> }]

여기서 문법구조를 조사한다는 것은 위의 형태로 입력 데이타가 들어 왔는지의 여부를 조사하는 것을 의미하며 로 표시한 부분은 생략가능한 부분이다. <변수2><변수3><변수4>는 프로그램 수행이전에 그값이 선언되어야 한다. 그러나 변수1은 그럴 필요가 없다. FOR 명령어를 중간코드로 바꾸는 구조는 아래의 그림 3과 같다.

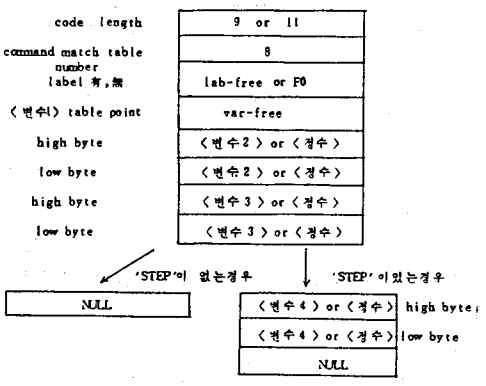

그림 3. FOR 명령어의 중간코드 구조
Fig.3 Record Format for FOR Command

제일 윗단의 9 또는 11은 코드의 길이를 나타내며 여기서는 두가지의 값을 가질 수 있다. 즉 'STEP' 이 없을 경우에는 9이고 'STEP'이 있을 경우에는 '11'이 된다. 그 밑에 있는 command match table number 는 있는데 이것은 각명령어마다 미리 정해진 토큰(Token) 값을 의미한다. 그 토큰값은 표 2에 정리되어 있다. 여기서는 FOR 명령어이므로 command match table number 값이 8이 된다. 그 아래에는 라벨(label)의 유무를 나타낸다. 즉 라벨이 없을 경우에는 F0 (HEX)의 값을 저장하고 라벨이 있을 경우에는 그 라벨이 저장된 테이블의 위치를 나타내는 'lab-free'의 값을 기록해 놓는다. 이렇게 하여 나중에 프로그램 수행시에 라벨의 위치를 쉽게 찾을 수 있어 시간을 단축한다. 'lab-free'의 값은 한 프로그램이 가질 수 있는 라벨의 갯수만큼 가질 수 있는데 본 프로그램에는 0에서 49까지 가질 수 있게 하였다. 물론 확장할 수도 있다. 그 다음은 FOR 의 문법구조에 따라 <변수1>에 관한 정보를 기록하는데 <변수1>은 미리 그값이 정해진 변수일 필요가 없으므로 만일 <변수1>이 변수 테이블에 등록되지 않은 변수일 경우에는 변수 테이블에 등록하고 그값은 NULL로 저장하고 변수 테이블에 등록되어 있는 변수일 경우에는 이전 값을 없애고 그곳에 NULL을 저장한다. <변수1>테이블 포인터에는 'var-free' 값을 기록해 놓는다. 'var-free'의 값은 'lab-free'와 마찬가지로 본 프로그램에서는 0에서 49까지 가질수 있게 하였다.

그 다음은 <변수2> 또는 <정수>에 관한 정보가 입력되는데 <변수2>는 <변수 1>과 같이 'var-free' 값이 0에서 49까지 가질수 있으나 <정수> 값과 구별하기 위하여 65486에서 65535(Decimal) 값으로 변환시키고 <정수> 값은 0에서 65485(Decimal)까지만 유효한 것으로 정의한다. 데이타 값이 16 bit 이므로 상위 바이트를 먼저 저장하고 그다음 하위 바이트를 저장한다. 마찬가지로 <변수3> 또는< 정수> 에 관한 정보도 상위, 하위 바이트로 나누어서 저장한다. 'FOR' 명령어에서 'STEP'은 생략가능하므로 'STEP'이 없을 경우에는 중간코드 마지막에 NULL을 저장하고 'STEP'이 있을 경우에는 <변수4>와 <정수> 에 관한 정보를 저장해야 하지만 정수 또는 변수4 도 앞의 경우와 마찬가지로 상위, 하위 바이트로 나누어서 저장한다. 그리고 마지막에는 역시 NULL을 저장한다. 여기서 본 프로그램상 주의할 사항은 변수는 숫자를 제외한 문자 5개까지이며 숫자와 문자를 같이 사용한 변수는 허용되지 않고 정수는 0에서 65485(Decimal)까지만 유효하다는 것이다.

표 2. 명령어들의 토큰값 및 중간코드발생프로그램과 수행프로그램

Table 2. Command Match Table Number & Code Generate Program & Execution Program

Command	Match Number	Code Generate	Execution Program
CLOSE	1	inter-cls( )	cls-ser( )
CLOSEI	2	inter-cli( )	cli-ser( )
DELAY	4	inter-dly( )	dly-ser( )
DOWN	6	inter-dwn( )	dwn-ser( )
FOR	8	inter-for( )	for-ser( )
GAIN	10	inter-gan( )	gan-ser( )
GETP	11	inter-get( )	get-ser( )
GOSUB	12	inter-gos( )	gos-ser( )
GOTO	13	inter-got( )	got-ser( )
GOTO(A)	14	inter-goa( )	goa-ser( )
GOTO(I)	15	inter-goi( )	goi-ser( )
IF	17	inter-if( )	if -ser( )
IFSIG	18	inter-ifs( )	ifs-ser( )
MOVE	20	inter-mov( )	mov-ser( )
MOVEF	21	inter-mvf( )	mvf-ser( )
MOVEP	22	inter-mvp( )	mvp-ser( )
MOVES	23	inter-mvs( )	mvs-ser( )
NEXT	25	inter-nxt( )	nxt-ser( )
OPEN	27	inter-opn( )	opn-ser( )
OPENI	28	inter-opi( )	opi-ser( )
PAUSE	30	inter-pau( )	pau-ser( )
READY	31	inter-rdy( )	rdy-ser( )
REMARK	33	inter-rem( )	rem-ser( )
RETURN	34	inter-rtn( )	rtn-ser( )
SET	36	inter-set( )	set-ser( )
SETI	37	inter-sei( )	sei-ser( )
SHIFT	38	inter-sht( )	sht-ser( )
SIGNAL	39	inter-sig( )	sig-ser( )
SPEED	40	inter-spd( )	spd-ser( )
STOP	41	inter-stp( )	stp-ser( )
UP	42	inter-up( )	up-ser( )
VPNT	43	inter-vpt( )	vp-ser( )

3. 실험 및 결과 검토

로보트의 프로그래밍 언어를 연구하는데 사용된 장비는 그림에서 나타낸 바와 같이 HP64000 LDS (Locol Develop-

ment System)과 IBM-PC, 8086, 8085 Emulator 등인데 실험의 개요는 64000 LDS 에서 C-언어로 프로그램을 작성하여 컴파일러(compiler)와 링커(linker)를 거친후 생성된 목적프로그램을 8086 Emulator 에 실어서 프로그램을 수행시켜보고 수정하는 과정을 반복하였다. 마찬가지로 서어보에서도 8085 Emulator를 IBM-PC와 연결해서 프로그램을 수행시켰다.
이러한 개발 시스템은 그림 4에 나타나 있다.

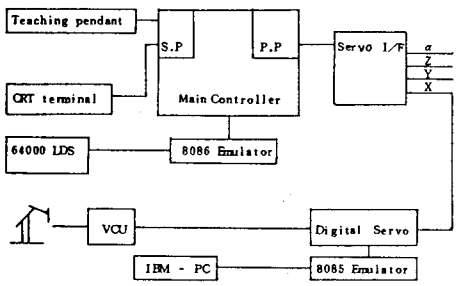

그림 4. 로보트 제어 시스템 개발 블럭선도
Fig.4 Block diagram of Robot Control Development System

본 논문에서 제시된 방법으로 작성된 로보트 프로그래밍 시스템으로 TOSHIBA에서 생산된 SCARA 형 로보트를 실제 구동하는 실험을 한 결과 기존의 내장된 로보트 프로그래밍 시스템인 SCOL-1의 기능을 그대로 수행했으며 본 실험실에서 직접 제작한 로보트 제어기에 연결하여 원하는 성능을 확인하였다.

4. 결론

지금까지 설명한 로보트 언어는 OS (Operating System)을 작성하기 편리한 C-언어로 설계되었으며 입력 출력부분은 8086 어셈블리 (Assembly) 언어로 작성되었다.[5),10)] 본 논문에서 설계한 로보트의 언어는 실제 SCARA 형 로보트인 TOSHIBA 로보트 몸체에 자체로 제작한 로보트 제어부를 연결하여 동작을 시키는 부분 즉 하드웨어 및 소프트웨어의 실제적 구현에 중점을 두었다. 따라서 다른 로보트 언어들에서 고려한 사항들을 다 수록하지 못한점도 있다. 그러나 일반적으로 SCARA형 로보트가 지니는 특성은 모두 가지고 있으며 특히 원시프로그램을 직접 구성해서 때문에 명령어를 확장하거나 수정을 쉽게 할 수 있으며, 기존의 SCARA형 TOSHIBA 로보트는 프로그램 입력을 키이보드로 하지 않고 함수키이(function key)를 이용하기 때문에 명령어를 확장 및 수정했을 경우 변경이 필요하다. 또한 원시프로그램이 공개되지 않았기 때문에 TOSHIBA 로보트 이외의 로보트에는 직접 적용할 수 없으나 본프로그램은 약간의 수정을 거치면 쉽게 타기종에 적용시킬 수 있다. 그다음 본 프로그램의 특징은 높은 단계의 언어인 C-언어를 사용해서 설계했기 때문에 쉽게 수정할 수 있으며 될 수 있는한 모든 프로그램들을 세분화하여 계층구조를 이루게 하였으며 수정 및 편집을 단위 프로그램별로 할 수 있게 하였다. 이러한 방식은 프로그램 기법의 현재 추세이다.

그러나 개발된 로보트 프로그램은 보충해야할 부분이 있으며 나아가서는 확장되어야 할 부분들이 있다. 즉, 보충해야할 부분은 각각의 모우드에서 새로운 기능의 명령어를 첨가시키는 것이며 또한 본 논문에서 제시한 프로그램에서도 중간중간 자기진단 기능이 있지만 좀더 체계적인 자기진단 기능을 첨가해야 한다. 자기진단기능은 주위환경 및 로보트의 상태 사용자의 상태등 모든 점을 고려하여 발생할 수 있는 상황을 처리할수 있어야 함으로 그 프로그램이 단기간에 나올 수 있는 것이 아니다. 그러므로 현재와 같이 로보트 언어의 원시프로그램이 공개되지 않은 상황에서는 그 프로그램의 확장이 힘들다. 그러나 국내에서도 계속 로보트 언어를 설계 및 공개하고 있으므로 조만간 이에 대한 연구가 체계적으로 될수 있으리라 생각한다. 또한 알고리즘에 있어서도 좀더 적합한 그리고 시간을 줄일수 있는 형태로 개선하고 새로운 알고리즘을 적용 실험할수 있도록 해야 하며, 나아가서 확장해야할 부분은 로보트 개발의 추세에 따라 비전(vision) 시스템을 적용할 수 있게 프로그램을 확장하고 토오크 및 힘 센서(Force/Torque Senso)를 사용하는데 편리한 구조로 확장하며 그 동작 오차를 줄여나가야 할 것이다. 이러한 문제들이 있음에도 불구하고 본 논문에서 제시한 프로그램은 로보트 언어설계 및 개발에 많은 도움을 줄수 있을 것이다.

참 고 문 헌

[1] Larry Heath, "Fundamentals of Robotics theory and Applications", Reston Publishing Company, pp.145-165, 1985.
[2] W.A. Gruver et.al., "Evaluation of Commereially Available Robot Programming Languages", Conf. Preceedings of 13th ISIR, 1983.
[3] John J. Donovan, "System Programming", McGraw-Hill, pp.279-314, 1982.
[4] Unimation Inc., "User's Guide to VAL", Ver.12, June, 1980.
[5] Kernigham, "The C Programming Language", Prentice-Hall, 1978.
[6] B.O.Wood, "MCL, The Manufacturing Control Language", Conf. Preceedings of 13th ISIR, 1983.
[7] S.Bonner et al., "A Comparative Study of Robot Language", IEEE, Computer, Dec., 1982.
[8] 김대원, "디버깅 특성을 고려한 로보트 매니퓰레이터의 언어에 관한 연구", 서울대학교 공학석사 학위논문, 1985.
[9] 김경환, " VAL-컴파일러를 이용한 로보트 프로그램 시스템의 설계", 서울대학교 공학석사 학위논문, 1986.
[10] 打越浩幸," C プログラムブック" アスキー出版局 pp.82-112, 1984.
[11] R.P. Paul, "WAVE: A Model-Based Language for Manipulator Control," Technical Paper MR. 76-615, SME, 1976.
[12] R. Finkel et al., "An Overiew of AL, A Programming System for Automation," Proceedings of the 4th International Joint Conference on AI, 1975.
[13] L.I. Liebeman et al., "AUTOPASS: An Automatic Programming System for Computer Controlled

Mechanical Assembly," IBM J. of Research and Development, Vol. 21, No.4, July, 1977.
[14] R.H. Taylor et al., "AML: A Manufacturing Language," Int. J. of Robotics Research, Vol. 1, No.3, Fall, 1982.
[15] William Thomas Park, "The SRI Robot Programming System", Proceedings of the 13th ISIR, 1983.
[16] B.E. Shimano et al. "VAL-II: A Robot Programming Languages and Control System", Robotics Research, pp.917-940, MIT Press, 1984.
[17] B. Shimano, "VAL: A Versatile Robot Programming and Control", Proceedings of COMPAC 79, 1979.
[18] Daniel H. Marcellus, "Systems Programming for small Computer," Prentice-Hall, pp.231-260, 1984.