'90 韓國自動制御學術會議論文集 1990. 10.26~27

사용에 편리한 ROBOT 언어 (SAITEL)의 개발

o
이영우, 이관형

삼성종합기술원

SAITEL: AN EASY ROBOT LANGUAGE TO USE FOR SCARA TYPE ROBOTS

Young Woo Lee, Gwan Hyung Lee

Samsung Advanced Institute of Technology

ABSTRACT

The robot operation by teach playback is easy and was widely used for simple jobs performed by a simple robot manipulator. However, as robots and their control systems and tasks become more and more sophisticated, such a simple robot operation is no longer adequate and programming languages capable for the complicated systems and tasks are greatly needed. In this paper, a high level robot specific programming language, SAITEL, is presented. It is an interpreter, based on Assembly, and has form similar to BASIC. SAITEL is easy to use for people who are not skilled programmers, and provides the capability to define robot task very conveniently. SAITEL was implemented on a direct drive SCARA robot developed in the Samsung Advanced Institute of Technology, and proved to be very useful for the operation of SCARA-type robots. It can be used also for other types of robots by slight modification.

1. 서론

60년대초 미국 General Motors사에서 Die Casting 작업에 로보트가 처음으로 사용된 이래 산업용 로보트의 이용은 급속도로 확산되고 있다. 단순한 운반 및 Spot Welding 작업에서 부터 복잡한 물체의 조립이나 검사 작업 등에도 로보트가 사용되고 있으며 로보트의 기능도 점차 각종 센서에 의해 지능화되고 확대되어 가고 있는 추세이다. 또한 로보트 관련기술의 발달로 로보트의 성능이 점차 좋아지고 가격이 저렴해지면서 현재는 경제적 채산성 및 생산성 향상을 위한 로보트 이용이 증가하고 있다.

산업용 로보트는 특정한 생산작업을 수행하기 위해 여러가지의 프로그램된 운동을 통해 부품의 운반, 조립이나 도구 또는 특수한 생산기구를 조작하기 위해 설계된 프로그램이 가능한 기계장치이다. NC Machine을 통한 기존의 자동화에 비해 산업용 로보트를 사용한 자동화의 주요한 장점중의 하나는 Programmability에 있다. 이러한 로보트를 이용함에 있어서 무엇보다 중요한 것은 "주어진 작업을 수행하기 위해서 어떻게 로보트를 효과적으로 교시할 것인가?" 하는 것이다. 이제까지 가장 많이 사용되어 온 방식은 Teaching Playback 방식으로 작업자가 로보트팔을 잡고 다니며 교시하거나 Teach Pendant를 가지고 로보트를 수동 조작에 의해 움직여 로보트를 교시하는 방법이다. 그러나 작업 내용이 복잡해지고 주변 장치가 다양해짐에 따라 단순히 경로만을 지시하는 것 이외에 복잡한 산술계산을 하거나, 센서로 부터 입력된 신호의 처리 및 그 결과에 의한 판단, 다른 컴퓨터 및 로보트들과의 정보 교환이 중요해짐에 따라 단순한 Teaching Playback 방식에 의한 교시 방법은 한계에 달하여 컴퓨터 언어와 유사한 Textual Language를 이용한 교시 방법이 점차 그 중요성을 더해가고 있다.

1. 기존 로보트 언어의 고찰

지금까지 발표된 대부분의 로보트 대략 100 여종에 이르고 있는데, 이와 같이 많은 로보트 언어가 생기게 된 데는 기존의 로보트 언어들이 특정 시스템에만 적합하도록 되어 있어 다른 로보트 시스템을 위해서는 부득이 그에 적합한 새로운 언어가 필요하기 때문이다. 이는 로보트 언어가 다른 컴퓨터 언어와 달리 로보트가 가지는 환경 즉, Control 하드웨어와 기구부의 구조, 사용 목적 및 요구 기능 등에 크게 의존하기 때문이다. 따라서 이러한 중복 투자를 막기 위하여 구미 여러나라와 일본에서는 로보트 언어의 표준화를 위해 노력하고 있다. 예를 들면, 독일에서는 IRDATA (Industrial Robot DATA)를 표준언어로 지정하여 사용하고 있고, 일본에서도 '일본 로보트 공업회'에서 '로보트 표준언어 단체 시안'을 작성하여 ISO/TC184/SC2로 제안하고 있다.

로보트 언어의 발달 과정을 살펴보면, 1960년대 후반부터 프로그래밍 언어를 사용하게 된 후 1973년 미국의 Stanford 대학교 인공 지능 연구소에서 로보트의 작업 수행 능력의 향상보다는 로보트의 이론적인 한계 사항을 발견할 목적으로 WAVE란 언어를 개발하였다. 1974년에는 동 연구소에서 Algol을 기초로 하여 여러개의 로보트를 평행으로 두고 공동 작업을 수행하기 위한 AL을 발표하였는데, 이것이 조립작업용으로 만든 최초의 로보트 언어이다. 그 이후 IBM사의 T. J. Waston 연구소에서 직각좌표형 매니퓰레이터를 제어하기 위해서 EMILY, ML, AUTOPASS, AML 등의 언어를 발표하였다. 또한 1979년에는 Unimation사에서 최초로 상업화된 로보트 프로그래밍 언어인 VAL을 개발했는데 이것은 Stanford대의 AL과 흡사하며 BASIC 언어를 기초로 하여 개발했으며 현재는 좀더 확장된 VAL-II가 발표되었다. 그 밖에 Cincinnati Milacron사의 T3, McDonnel Douglas사의 MCL, Automation 사의 RAIL, General Electric사의 HELP 등이 개발되었다.

2. SAITEL (SAIT Easy Language)의 설계 개념

로보트 언어는 일반적으로 사용자의 입장에서 볼 때 다른 컴퓨터 언어와 비슷한 형태로 되어 있다. 기존의 컴퓨터 언어에서 사용하는 명령어에 로보트를 구동하는데 필요한 명령어를 합한 형태를 취하고 있으나 실제 내부에서 처리하는 방법은 현지히 다르다. 로보트를 구동하면서 생기는 이러가지 에러(Error)의 처리 및 복구, 작업의 변경 등을 수행하기 위해서는 수행되는 로보트 프로그램의 위치 및 상태 등을 계속적으로 내부에 지장시키고 수시로 점검하고 있어야 하며 로보트를 구동하거나 주변장치를 원하는 대로 작동시키기 위해서는 실시간 (Real-Time) 시스템이 되어야 한다.

SAIT에서 개발한 직접구동 (DD) 방식의 SCARA 로보트는 주된 용도가 부품조립용이므로, SAITEL은 조립에 필요한 기능을 업무에 두고 Teaching의 편리성, 작업

프로그램 작성 및 디버깅 (Debugging)의 용이성에 중점을 두고 개발되었다. 또한 차후의 기능 확장 및 타 로보트에도 이식이 가능하도록 하기 위하여 하드웨어의 큰 변화가 없이 기능의 변화가 가능하도록 하고 프로그램을 모듈 (Module)별로 작성하여 하드웨어의 변화시에 그 변경이 쉽도록 하였다. 그리고 컴퓨터와의 연결을 통해 작업장 외에서도 로보트를 동작시키기 위한 작업 프로그램의 작성이 가능하도록 하였다. 로보트 언어를 수행하는데 필요한 하드웨어의 최소 구성은 ROM 64 KByte, RAM 32 KByte, RS232C Port 1개, Timer Interrupt 1개이다. Arm을 구동하는 Servo System과의 인터페이스(Interface)는 공통 메모리 (Common Ram)를 통하여 이루어진다.

II. 로보트 언어 시스템 구조

일반적인 로보트 프로그래밍 시스템은 로보트 몸체, 대상물, 작업 지시와 동작 지시를 하는 언어와 이 언어를 해독해서 로보트를 제어하는 처리계, 그리고 외부 세계와 관련된 정보를 갖는 환경 모델 등 여러 요소로 구성된다. 이러한 일반적인 로보트 프로그래밍 시스템을 블럭 선도로 나타내면 그림 1과 같다.

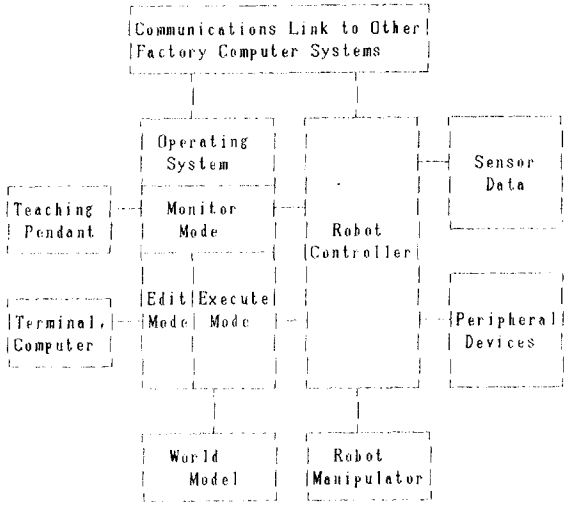

그림 1 로보트 프로그래밍 시스템

1. 제어기 시스템

제어기 (Controller) 하드웨어는 3개의 CPU를 주축으로 하며 각 축의 위치를 저장하고 계산하는 Position Board 및 각 축 Motor Driver로 구성되었다. 3개의 CPU는 각각 로보트 전시스템을 관장하는 로보트 언어 처리부, Point To Point (PTP) 및 Continuous Path (CP) 등의 동작을 제어하는 Trajectory Planning (TP)부, 각 축의 Motor를 제어하는 Servo Control부로 나뉘어진다. 이외에 주변장치로는 2개의 터미날 (Terminal)과 1개의 Teach Pendant가 부착될 수 있도록 하였다. SAIT에서 개발한 DD 로보트를 구동하는 콘트롤러의 중요한 Block만을 표시하면 그림 2와 같다. CPU는 16 Bits 인 Motorola 68000이며 모두 3개의 CPU를 사용한다.

Language Processor는 로보트의 Supervisor 기능과 함께 작업자가 사용할 수 있는 로보트 언어 기능을 제공한다. 로보트 언어는 Interpreter로 되어 있으며 문장의 편집은 일반 BASIC 언어와 같이 줄번호를 이용하여 이루어진다. 작업 프로그램의 수행시에 로보트 동작에 관한 명령은 동작에 필요한 각종 정보와 함께 TP Processor로 넘기게 로보트가 동작하도록 한다. System Processor는 로보트 언어 처리부와 병렬로 동작하는 소프트웨어로 로보트 언어 처리시에 사용하는 Timer 기능 외에도 로보트의 자기 진단 기능, I/O 상태의 계속적인 Check, 시분할 병렬 처리를 위한 기능 등을 갖고 있다.

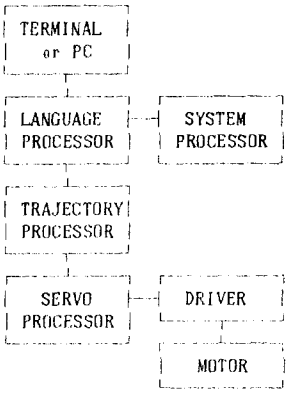

그림 2 콘트롤러 구성도

TP Processor는 로보트의 궤적 계획 (Trajectory Planning)을 위한 소프트웨어 시스템이며 여기에서 로보트의 동작을 위한 로보트 팔의 이동 경로, 속도, 가감속도 등의 연산처리가 이루어진다. 즉 로보트의 PTP, CP와 직선 및 원호 보간이 실행된다. 여기에서 연산된 결과는 Servo Processor로 넘겨진다.

Servo Processor는 TP Processor에서 넘어 온 데이타를 가지고 위치 및 속도 제어 등을 통하여 각 축을 구동하기 위한 Processor이다. TP에서는 연산 시간이 오래 걸리므로 (16 ms) Servo Processor에서는 로보트의 진동을 없애기 위해 좀더 빠른 Sampling Time (2 ms)을 갖고 모터를 구동한다.

2. SAITEL 언어 처리의 개요 및 특징

SAITEL은 서론에서도 언급한 바와 같이 언어처리를 로보트 콘트롤러 내부에서 모두 처리하기 위해 모든 Operating System을 독자적으로 설계했으며 주어진 하드웨어의 용량내에서 구동될 수 있도록 하였다. SAITEL의 개발에는 Motorola 68000 Assembler와 C 언어를 사용하였다. 로보트 언어 처리부는 그림 3에 나타난 것과 같이 TP를 제외한 7개의 Block으로 나뉘어져 있다. 그림 3에 나타나지 않은 System Processor는 그 기능이 독립적으로 이루어진다. System Processor가 하는 일은 주로 Timer의 역할과 로보트 전체의 자기 진단, 작업 프로그램 수행시의 입출력 등이다. 이 외에도 모니터 처리를 위한 시분할 병렬 처리를 수행한다.

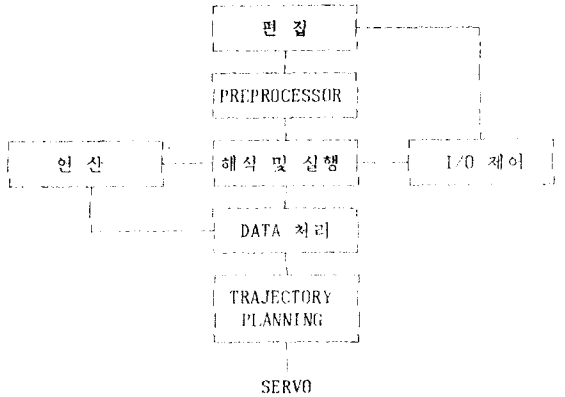

그림 3 로보트 언어 처리부의 구성도

로보트 언어 처리부를 크게 분류하면 편집과 해석 및 실행부로 나뉘어진다. 편집은 한 문장이 입력될 때마다 명령에 해당되는 단어는 즉시로 인터프리터의 내부 형태인 토큰열로 바뀌고 줄번호 순으로 정렬되어 주기억 장치인

내부 메모리에 저장된다. 사용자는 편집 모드와 실행 모드의 구분이 없이 편집과 실행을 자유로이 할 수 있으며 Job 번호를 지정함에 따라 각기 다른 10 개의 프로그램을 독립적으로 편집 혹은 실행할 수 있다. 편집은 터미널에서 직접 수행할 수도 있으며 컴퓨터에서 작성된 작업 프로그램을 XON/XOFF Protocol을 사용하여 로보트 콘트롤러로 전송하거나 이미 작성된 작업 프로그램을 컴퓨터로 전송할 수도 있다.

해석 및 실행은 줄번호가 없는 문장이 입력되면 명령으로 간주하여 실행을 하게 되며 프로그램으로 실행하는 명령과 단독 명령과의 차이는 없다. 이러한 방법은 실시간 처리가 요구되는 곳에 자주 사용되는 방법이다. 실행 부분은 문장의 앞에 있는 명령에 해당하는 토큰열을 해석하고 해석된 명령의 처리 함수가 있는 곳으로 분기한다. 각 명령을 처리하는 함수는 프로그램의 규정된 문법에 따라 한 명령 혹은 한 데이타씩 읽어내어 처리하고 문장이 끝나면 다시 다음 문장을 읽어 명령을 처리한다. 실행은 프로그램 명령의 실행과 직접 명령의 실행으로 나뉘어지는데 내부 구조상 특별한 구분이 없다. 즉 사용자는 프로그램을 실행시키기 위해 'RUN' 외에도 'GOTO', 'GOSUB' 등의 분기 명령을 사용할 수 있다.

'RUN' 명령에 의해 프로그램이 수행되면 먼저 프로그램을 수행하기 위한 변수 및 사용영역이 할당되고 프로그램의 수행 상태를 Update 하기 위한 내부 테이블, 스택 등이 초기화된다. 한 명령을 실행하고 다음 명령을 해석하기 전에 사용자가 지정한 스텝수, Break Point 등에 대한 Check와 함께 System Processor에서의 자기진단 결과에 대한 에러 유무에 대한 판단을 수행한다. 명령중 가장 긴 시간을 소비하는 것은 로보트의 동작과 관계되는 명령이며, 위치값을 읽어 내부에서 'FRAME', 'BASE' 및 'TOOL'에 관한 좌표변환을 한 후 TP Processor로 전송하는데 걸리는 시간은 약 5 msec이다.

변수명에 대한 테이블 및 값은 수행중에 발생하는 대로 처리를 한다. 문장의 해석중에 변수를 만나면 변수 테이블에서 찾아 값을 불러오게 되며 변수명이 등록이 되어 있지 않으면 등록을 한 후 값을 불러온다. 프로그램의 분기시에는 줄번호를 사용하며 사용자가 Label을 설정하고 분기 위치를 Label로 지정하여도 내부에서 설정된 Label에 해당되는 줄번호로 처리한다.

앞에서 설명한 바와 같이 SAITEL은 SAIT에서 개발한 SCARA형 DD 로보트에서 수행되도록 설계된 로보트 전용 언어이다. SAITEL은 로보트 언어 처리부와 모니터, 시스템 소프트웨어 등 3개의 소프트웨어로 구성되어 있으며 3개가 시분할 병렬 수행에 의해 구동된다. 이기에서 모니터는 실제로 작업자에게는 제공되지 않으며 개발과정에서 필요에 의해 설계된 것이나 작업 프로그램의 디버깅시에도 매우 편리하므로 기능을 보완 및 확장하여 로보트 소프트웨어 시스템에 포함시켰다. 모니터는 로보트가 동작중에도 로보트 및 콘트롤러의 동작 상태, 프로그램의 위치, 로보트의 구동 위치, 작업 프로그램 수행시의 각종 사용 변수 등을 로보트의 동작과 상관없이 점검할 수 있게 하여준다. 또 System Processor는 로보트 언어 처리부와 병렬로 동작하는 소프트웨어로 로보트 언어 처리시에 사용하는 Timer 기능 외에도 계속적으로 수행해야 하는 로보트의 자기 진단 기능, I/O 상태의 계속적인 Check 등과 시분할 병렬 처리를 위한 기능을 가지고 있다. 이러한 구성은 로보트 언어처리부를 콘트롤러 내부에 갖고 있도록 하여 사용자의 입장에서 볼 때 매우 편리하며 또한 가격을 낮추어 생산 현장에 적용시에 경제성을 높히기 위함이다.

III. SAITEL 언어의 기능

일반적으로 로보트 언어는 기존의 컴퓨터 언어에서 사용하는 명령어에 로보트의 구동, 외부 주변 장치와의 정보 교환, 각종 센서의 처리 등을 위한 명령어가 추가되어 있는 형태로 되어 있으며, 일부 로보트 언어에는 여러 개의 로보트를 제어하기 위한 기능과 병렬처리를 위한 기능을 가진 것도 있다. 로보트 언어가 범용 컴퓨터 언어와 마찬가지로 제어 구조, 자료형, 연산자 등을 갖기 때문에 로보트 언어를 개발하는데에 있어서도 이러한 기능에 중점을 두어 기존의 컴퓨터 언어에 로보트 프로그램밍에 필요한 함수나 자료구조 등을 첨가하여 사용하는 경우도 있다.

SAITEL은 인터프리터 방식을 사용한 새로운 로보트 언어이다. OS는 대부분 Assembler로 작성되었으며, 연산처리 등은 Coprocessor를 이용한 C 언어로 작성하여 프로그램의 수행 속도를 높힘과 동시에 개발 기간을 절약하였다.

1. 로보트 제어 명령

SAITEL은 로보트를 움직이기 위한 다양한 명령어를 갖고 있다. 동작 방식은 직선, 원호 및 PTP 이며, 경유점을 가지는 CP 기능도 있다. 또한 동시제어 및 각 축만을 독립적으로 구동할 수도 있다. 로보트 동작 방식의 지정은 주어진 좌표값에 대해 각 축별로 연산을 수행할 수가 있어 직선 및 원호 등의 특정한 형태뿐만 아니라 포물선, Sine 곡선, 타원 등 수식으로 나타낼 수 있는 모든 형태의 동작 방식으로 움직일 수 있다.

작업시의 Cycle Time을 줄이기 위해서 정확한 위치 제어가 필요하지 않은 위치로의 이동은 'COARSE' 라는 명령을 이용하여 동작 시간을 단축할 수 있다. 이 명령은 주어진 동작점에서의 위치 제어 범위를 크게하여 다음 동작점으로의 이동을 빠르게 한다. 동작 위치는 직접 4 축에 해당되는 위치값을 입력할 수도 있고 Teach Pendant 또는 터미널을 이용하여 로보트를 원하는 위치로 움직이도록 하고 그 위치를 지정할 수도 있다. 로보트를 이용한 작업의 종류에 따라 또는 로보트가 취급하는 물체의 무게에 따라 로보트의 동작이 원활이 이루어지도록 로보트의 속도 및 가속도, 감속도를 변경할 수도 있다. 작업대의 위치에 따라서 로보트의 자세를 지정할 수 있으며 직선 보간의 경우에는 직선 운동시의 속도를 mm/sec 단위로 설정할 수 있다.

2. 좌표변환 기능

로보트 언어에서의 주요한 기능중의 하나는 위치 관련 정보를 어떻게 작업자가 효과적으로 사용하고 또 최소한의 교시 작업만으로 다양하고 복잡한 작업을 로보트가 수행하도록 하느냐 하는데에 있다. 이를 위하여 SAITEL 에는 다양한 좌표 변환 기능이 있다. 좌표 변환은 로보트가 수행하는 작업을 변경하거나, 다른 작업에 기존에 작성된 작업 프로그램을 이용할 수 있도록 함으로써 작업의 효율을 높이는데 기여한다. 실제로 대부분의 로보트를 이용한 작업에서는 로보트를 직접 움직여서 작업 위치를 교시하게 된다. 그러나 작업 환경이 복잡해지고 교시해야 할 위치가 많은 경우에는 새로운 작업을 시작하거나 작업을 변경할 때 소요되는 시간이 많이 걸리게 된다.

SAITEL에서는 로보트 좌표계에서 작업 환경에 맞도록 새로운 좌표계를 설정하고, 또 작업 도중에 수시로 바꿀 수 있는 작업 테이블 및 부품 Feeder에 대한 좌표계를 설정할 수 있도록 하였다. 각각의 좌표계를 'FRAME' 과 'BASE' 라는 명령을 사용하여 좌표계를 설정할 수 있도록 하였으며 그 외에도 작업 Tool에 대한 좌표계와 이미 교시된 위치값에 대한 Shift, Rotation 그리고 크기에 대한 Scale을 지정할 수 있도록 하였다.

'FRAME' 명령은 로보트의 좌표계와 작업대의 좌표계의 관계를 지정하는 명령이다. 실제로 로보트를 이용한 작업시에 주변 장치들의 정확한 위치를 수치적으로 알아내기는 매우 어렵다. 그러므로 로보트를 직접 움직여서 로보트에 대한 상대적인 좌표계를 찾아낸 후 작업대와 로보트간의 좌표 변환 Matrix를 구하게 된다. 이때 'FRAME' 좌표값을 구하는 방법은 로보트를 움직여 작업 공간의 원점과 X축상의 한점 그리고 Y축상의 한점 등 3점을 교시하여 내부에서 로보트에 대한 작업대 좌표계의 변환 Matrix를 계산한다. 계산된 좌표계는 'FRAME' 명령에 의해 새로이 지정될 때까지 계속적으로 유효하다.

'BASE' 명령은 Z축과 직각인 평면상에서 이루어지는 좌표변환 명령이다. 'FRAME' 명령과 달리 'BASE' 명령은 작업 프로그램 상에서 어느 때이고 필요에 따라 새로운 좌표를 설정할 수 있다. 그러므로 여러 개의 작업대가 있을 경우에 사용자는 작업대마다 'BASE' 명령에 의해 각각의 새로운 좌표계를 지정하여 작업할 위치를 교시할 수 있다. 'BASE' 좌표계는 사용자가 직접적으로 값을 넣을 수도 있지만 'FRAME' 좌표계와 마찬가지로 로보트를 직접 움직여서 'BASE' 좌표계를 지정할 수도 있다. 이때는 원점과 X축 방향의 한점을 교시한 후 'BASE' 명령을 이용한다.

'TOOL' 좌표계는 로보트를 이용한 작업시에 로보트의 Tool에 따른 프로그램의 변경을 용이하게 하기 위해 사용된다. 이는 Hand에 부착되는 Tool만 바꿔 할 수 있는 일에 대해 매우 유용한 명령이다. Tool의 모양에 따라 작업자가 직접 'TOOL' 좌표계를 설정할 수도 있으며 로보트를 직접 움직여서 Hand에 Tool이 부착된 상태에서의 로보트 위치와 부착되지 않은 상태에서의 로보트 위치를 각각 교시하여 로보트 내부에서 'TOOL' 좌표계를 계산하여 지정할 수도 있다.

3. 프로그램의 편집 및 수행 제어

SAITEL은 프로그램 언어중 가장 사용하기 쉬운 BASIC의 형태를 취하고 있으며 문법도 BASIC 언어와 거의 유사하다. 이러한 BASIC 형태는 컴퓨터 언어에 대한 지식 수준이 낮은 현장 작업자에게 가장 이해되기 쉬우며 줄번호를 가지기 때문에 어느 터미널을 사용하더라도 쉽게 프로그램의 편집을 수행할 수 있다. 또 작업 프로그램의 편집시에 줄번호순으로 정리되어 저장되며 편집중에라도 언제든지 로보트를 움직일 수 있다.

로보트 프로그램은 자체 메모리 영역에서 모두 10개의 서로 다른 작업 프로그램을 가질 수 있다. 각각의 작업 프로그램은 서로 독립적으로 편집, 저장 및 삭제될 수 있으나 주기억장치 안에서의 교시된 좌표값 및 위치값은 서로 공유하고 있다. 그러나 작업 프로그램과 위치값들은 각각 하나의 파일로 간주되어 콘트롤러 안에 내장된 보조 메모리에 저장된다. 보조 메모리에 저장되는 작업 프로그램 파일은 각각의 파일 이름을 갖고 있으며 Indexed Sequential File의 형태로 저장된다. 작업 프로그램 파일의 저장은 Directory 영역과 File 영역으로 구분되어 저장되며 각각의 프로그램 파일은 File Directory 영역에 프로그램의 시작 번지와 끝 번지에 대한 정보가 들어 있어 이를 갖고 파일의 삽입과 삭제를 한다. 이때 앞에 있는 파일의 끝 번지는 뒤에 있는 파일의 시작 번지를 가리키는 Link 구조를 갖고 있다.

4. 연산 기능

BASIC 형태의 언어 구조를 가지기 때문에 연산을 위한 수치 및 변수는 특별한 선언이 없이 사용된다. 변수명은 4글자까지 유효하며 모든 값은 내부에서 실수로 변환되어 처리되며 특정한 기능 즉, Bit 단위의 논리 연산시에는 내부에서 정수로 변환되어 처리된다. 로보트의 좌표값에도 수치뿐만 아니라 변수도 사용될 수 있으며 연산도 각 좌표 및 각 축에 대해서 수행될 수 있어 로보트의 동작을 나타내는 수식을 그대로 로보트의 좌표값으로 처리하여 로보트를 구동할 수 있다. 또한 'FRAME', 'BASE', 'TOOL' 등의 좌표변환 함수도 변수 및 수식으로 처리될 수가 있다.

연산은 모든 처리가 1 Pass로 이루어지며 우선 순위에 따라 처리가 이루어진다. 연산을 1 Pass로 처리하기 위해서 프로그램이 순환적(Recursive)으로 처리되며 구현을 쉽게 하기 위해 C 언어로 작성되었다.

5. 입출력 기능

입출력에는 두 가지가 있다. 하나는 PLC 등과 같은 외부 주변장치와의 상호 접속을 위한 Bit 단위 혹은 Byte 단위의 I/O 입출력이고 다른 하나는 RS232C 등과 같은 Serial 입출력이다. 로보트의 작업 환경이 복잡해지고 주변 장치와의 정보 전달 및 외부의 상태에 따른 작업 변경을 위한 기능이 필요하게 됨에 따라 SAITEL에서도 여러가지의 입출력 기능을 제공하고 있다. 특히, 사용자는 필요한 Serial Port를 지정하여 연산 결과 및 로보트의 위치, 문자 등을 출력할 수 있고 또 입력되는 글자를 처리할 수가 있다. 이때 Serial 입출력의 경우 로보트와 접속되는 주변 기기와의 전송 속도 차이를 고려해 XON/XOFF Protocol을 갖고 입출력을 처리한다.

Bit 및 Byte 단위의 I/O 입출력시에도 여러 가지가 가능하다. 입출력 함수는 연산 함수로 처리되어 입력 결과에 따른 연산, 판단 등을 쉽게 할 수 있으며 출력시에도 연산 결과에 따른 조건부 출력이 가능하다. 또한 로보트를 이용한 작업 시스템의 구성시에 전체 작업 시스템의 Master로 로보트가 아닌 다른 중·대형 PLC를 사용하는 경우 Slave가 되는 로보트를 동작시키기 쉽도록 I/O 입출력시에 각 Bit 별로 입력 신호가 들어올 때 사용자가 마치 터미널 혹은 Teach Pendant에서의 Key 입력과 같은 효과를 갖도록 할 수 있다. 즉 지정한 입력 단자에 High 혹은 Low 신호가 입력되면 Keyboard에서 'RUN', 'STOP', 'HOME' 혹은 'MOVE POS(1)' 등과 같은 문자가 입력된 것으로 간주하고 로보트를 동작시킬 수 있다. 입력 신호에 따르는 기능 설정은 사용자가 언제든지 변경 가능하고 또 프로그램에 의해 수정될 수 있다. 이때의 입출력은 Language Processor가 아닌 System Processor에서 처리된다.

6. 디버깅 및 에러 처리 기능

SAITEL에서 제공하고 있는 디버깅 기능은 BASIC 언어에서와 같은 Trace 명령을 갖고 수행 과정을 터미널에 표시할 수가 있다. 또 작업 프로그램의 수행도 'RUN 10, 200'과 같은 형태로 일정 영역을 지정하여 수행하면서 각 변수들을 확인해 볼 수 있으며 각 문장 순으로 한 스텝 혹은 여러 스텝씩 수행할 수 있다. 특히 로보트를 직접적으로 구동하지 않고도 프로그램을 수행시킬 수 있도록 하는 가상 구동 기능을 가지고 있어 잘못된 동작 프로그램의 수행시에 발생할 수 있는 위치 에러 및 다른 주변장치와의 충돌 등에 대해서 확인할 수 있도록 하였다.

로보트 언어에서 에러의 처리는 매우 중요한 부분을 차지한다. 작업을 수행하다보면 여러 원인에 의해 로보트가 다른 작업물과 충돌하거나 주변장치의 고장 및 작업 물체의 불량으로 인한 로보트의 정지 등이 발생할 수가 있다. 다른 프로그램 언어와는 달리 로보트에서는 작업시에 에러가 발생했을 경우에 작업을 항상 처음부터 다시 수행할 수가 없는 경우가 있으며 이러한 경우에는 에러를 복구한 후에 에러가 발생한 곳부터 프로그램을 계속적으로 수행해야 한다. 이를 위해서는 프로그램 수행에 필요한 변수들이 항상 Update되면서 일정한 곳에 기억되고 있어야 한다. 또한 에러가 발생했을 경우의 상황을 내부 메모리에 저장되고 있다가 에러의 복구후 동작을 재개할 때 참고하여 프로그램을 다시 수행시켜야 한다. 이와 같이 작업시의 여러 원인에 의한 동작 정지 등의 각종 에러시에도 작업 프로그램을 처음부터 다시 수행하지 않고도 에러의 원인을 제거한 후 정지된 위치부터 다시 수행할 수 있도록 하여 작업정지 시간을 줄이고 조립 과정에 있는 부품의 손실을 최소화하도록 하였다.

로보트 및 주변 환경과는 관계가 없는 프로그램만의 잘못에 의한 에러시에도 작업에 영향을 주지 않기 위한 조치가 필요하다. 이러한 경우에 프로그램을 수정하고 수정된 부분에 대해서도 앞에 언급한 Trace, 스텝 기능, 가상 구동 기능 등의 디버깅을 수행할 수가 있다.

7. 모니터 기능

모니터는 로보트 사용자에게 제공되는 것이 아니고 로보트 시스템 소프트웨어 설계자가 필요에 의해 로보트의 동작을 점검하기 위해 작성된 것이다. 그러나 이는 로보트를 이용한 작업 프로그램의 디버깅에도 사용될 수 있으며 동작중 잘못 교시된 위치를 수정할 수도 있다. 모니터는 로보트 Language Processor와 동시에 수행되며 이는 시분할 병렬프로세싱에 의해 구현된다. 모니터는 로보트가 동작중에도 로보트 및 콘트롤러의 동작 상태, 프로그램의 위치, 로보트의 구동 위치, 작업 프로그램 수행시의 각종 사용 변수 등을 로보트의 동작과 관계없이 점검할 수 있게 해주며 로보트 내부 메모리에 정상적인 수치가 실리는지 혹은 Trajectory Processor나 Servo Processor로의 데이타 전달이 정상적으로 이루어지고 있는지를 수시로 점검할 수 있게 해 준다. 즉 모니터는 로보트 사용과는 관계없이 다른 터미널을 사용하여 로보트의 동작을 점검할 수 있게 한다.

모니터는 모두 C 언어로 작성되었으며 명령의 해석 및 실행 과정은 로보트 언어 처리에 비해 매우 간단하다. 구현 방법으로는 Timer에 의해 일정 시간마다 수행되는 System Processor가 Language Processor와 모니터를 번갈아 수행시킨다. 이러한 모니터는 사용자에 의해 수행시킬 수도 있고 정지시킬 수도 있다. 모니터와 Language Processor를 동시에 구동시키면 Language Processor만 단독으로 구동시킬때 보다 한 명령을 수행하는데 걸리는 시간은 2배가 된다. 이와 같이 일정 시간 마다의 시분할에 의해 각각 수행되는 두 개의 프로그램은 서로 다른 메모리 영역에서 동작되나 공통 입출력을 가지고 있고 함수도 서로 공유하고 있다. 그러므로 두 개의 프로그램을 동시에

수행할 때에 고려할 점은 상호간의 데이타 입출력시에 충돌에 의한 Deadlock이 발생되지 않아야 한다. 이를 위해 각각의 프로그램은 다른 Stack 영역에서 동작하고 정보의 교환시 및 입출력시에는 Dekker's Algorithm을 이용하여 두 프로그램간의 충돌을 회피하여 Deadlock을 방지하고 있다.

지금까지 설명된 SAITEL 로보트 언어의 주요 명령어를 요약하면 표 1과 같다.

표 1 SAITEL 로보트 언어 명령어의 종류

기 능 항 목	명 령 어 및 설 명
프로그램 수행	RUN, STEP, CONT, CLEAR
프로그램 수행 제어	GOTO, GOSUB, RETURN, FOR TO STEP NEXT, END, STOP
프로그램 편집	EDIT, PRINT, DELET, NEW, LIST, RENUM
FILE 관리	MLOAD, MSAVE, MDEL, MREN, COMPACK MDIR, MNEW
위치 DATA 관리	PLOAD, PSAVE, PDEL, PREN, PRECOVER PDIR,
I/O 제어	ON, OFF, OPEN, CLOSE, STAT, IN PULSE
DATA 입출력	PRINT, GET, INPUT, PENDANT, TAB
연산 함수	ABS, SIN, COS, TAN, ASIN, ACOS SQRT, INT, ATAN, LN, LOG
연산자	+, -, *, /, AND, OR, NOT, BAND, BOR <>, <=, >=, =, <, >, BNOT
아암 동작	MOVE, MOVES, MOVEC, APPRO, APPROS JUMP, DRIVE, HOME, DEPART, DEPARTS
동작 PARAMETER 지정	SPEED, ACCEL, DECEL, FINE, COARSE LEFTY, RIGHTY, CP ON/OFF, SPEEDS
위치 지정	HERE, POS(n), PX, PY, PZ, PS PLIST, DX, DY, DZ, DS
좌표 변환	FRAME, BASE, TOOL, INVERSE, SHIFT SCALE, RX, RY, RZ
기타	CLS, WHERE, ASC%(), CHR$(), DELAY SET, TIME, TIMER, TON, TOFF, LET HEX$(), LOCATE, DIM

IV. 작업 프로그램 예시

본 절에서는 SAITEL을 이용한 간단한 프로그램을 예시하고자 한다. 작업의 예로서 그림 4와 같은 컨베이어 라인에 흐르는 팔레트(Pallet)에 Feeder A에 있는 작업물을 올려놓고 그 위에 Feeder B에 있는 물체를 삽입하는 것이다. 여기서 작업물과 물체의 방향성은 없다고 가정한다. 이때 팔레트의 위에는 가로 세로가 각각 4개씩 모두 16개의 작업물을 올려 놓을수 있다고 하고 작업물을 놓는 간격은 가로 세로 모두 50 mm 라고 하자. 여기서 입력 신호인 팔레트의 도착 완료 센서 신호는 입력 3번으로, 작업 완료 신호는 출력 4번이라고 가정한다. 또 주변 장치인 PLC와의 연결을 위해 입력단자 0번과 1번을 할당하고 각각 'RUN'과 'STOP' 신호를 의미한다고 하자. Feeder A의 교시 위치는 POS(10), Feeder B의 교시 위치는 POS(11)이고 교시된 팔레트의 원점과 X축의 한점, Y축의 한점을 각각 POS(0), POS(1), POS(2)라고 한다면, 작업 프로그램은 다음과 같다.

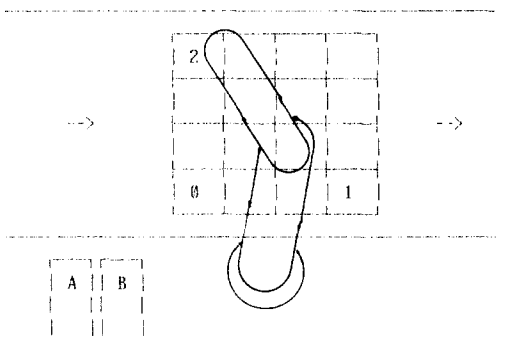

그림 4 Robot 이용 작업례의 개략도

```
100 REM ------------------------------------------
110 REM --- INITIALIZE WORK PARAMETER
120 REM ------------------------------------------
130 SET IN(0) = "RUN"
140 SET IN(1) = "STOP"
150 FRAME POS(0), POS(1), POS(2)
160 X = 50 : Y = 50
170 A = 10 : B = 11
180 PALLETTE = 5
190 OPEN
200 REM ------------------------------------------
210 REM --- PALLETIZE AND ASSEMBLY
220 REM ------------------------------------------
230 IF IN(3) = 0 THEN 230
240 FOR X = 0 TO 150 STEP 50
250     FOR Y = 0 TO 150 STEP 50
260         POS(PALLETTE) = X, Y, 0, 0
270         FEEDER = A
280         @PICK_PLACE
290         FEEDER = B
300         @PICK_PLACE
310         PULSE(4)
320     NEXT
330 NEXT
340 GOTO 230
350 REM ------------------------------------------
360 REM --- SUBROUTINE FOR PICK & PALCE
370 REM ------------------------------------------
380 LABEL "PICK_PLACE"
390     APPRO POS(FEEDER), 50
400     MOVE POS(FEEDER)
410     CLOSE
420     DEPART 50
430     APPRO POS(PALLETTE), 50
440     MOVE POS(PALLETTE)
450     OPEN
460     DEPART 50
470 RETURN
```

위 프로그램의 예에서 줄번호 100에서 190까지는 작업에 필요한 각 변수 및 'FRAME' 좌표계를 설정하기 위한 것이며 줄번호 200 에서 340까지는 실제 작업을 수행하는 부분이다. 여기서 @PICK_PLACE는 줄번호 350에서 470까지 'PICK_PLACE'라는 이름으로 설정된 LABEL 명을 의미한다.

V. 결론

본 논문에서는 SAIT에서 개발한 DD SCARA 로보트에 적용할 수 있는 로보트 언어인 SAITEL에 대해 설명하였다. SAITEL은 비록 특정한 로보트 시스템에만 적용되나 하드웨어에 종속되는 부분을 줄이고 구동할 수 있는 하드웨어도 최소한의 시스템만으로 가능하도록 하였다. SAITEL은 인터프리터 방식의 로보트 언어이며 가장 이해하기 쉬운 언어인 BASIC의 형태로 되어 있다. SAITEL의

기본은 Motorola 68000 Assembler와 C 언어로 되어 있으며 처리 속도를 높이기 위해 인산은 Coprocessor를 사용하였다. 타 로보트 언어와의 비교는 부록에 상세히 분석하여 수록하였다.

SAITEL은 모든 처리를 콘트롤러 내부에서 처리하며 일반적인 미니급 혹은 컴퓨터와 쉽게 연결되어 사용할 수 있다. 또한 주변장치와의 상호 접속을 고려하여 Serial 입출력 및 I/O 신호 처리에 다양한 기능을 부가함으로써 현장 공정에의 적용시 작업자가 쉽게 로보트가 갖는 기능을 변경할 수 있도록 하였다. 특히 로보트 구동과는 별도로 보니터 시스템을 지원한 병렬처리로 수행하여 작업 프로그램의 디버깅 및 수정을 용이하게 할 수 있고 작업시에 발생할지도 모를, 여러 원인에 의한 동작 정지 등의 각종 에러시에도 에러의 원인을 제거한 후 작업 프로그램을 처음부터 다시 수행하지 않고 정지된 위치부터 재개하여 작업 시간을 단축시킬 수 있도록 하였다.

본 SAITEL의 개발로 SCARA 형과 같은 조립용 로보트를 현장에서 취급할 때에 작업자들이 컴퓨터 프로그램에 대한 깊은 지식이 없이도 제반 작업에 필요한 복잡한 로보트 프로그램을 용이하게 작성할 수 있게 되었으며, 또한 이 SAITEL 로보트 언어는 다른 형태의 로보트에도 기본 하드웨어 조건만 만족하면 손쉽게 이식하여 사용할 수 있다. 앞으로 남은 과제로는 본 SAITEL을 시각 시스템(Vision System)과의 인터페이스 및 여러 로보트간의 협력 작업 등에도 손쉽게 사용할 수 있도록 보완 발전시키는 일이다.

참고 문헌

[1] John J. Craig. Introduction to Robotics, Addison-Wesley, 1986

[2] Brian O. Wood and Mark A. Fugelso, "MCL: The Manufacturing Control Language," 13th ISIR, 1983.

[3] Hitoshi Kubota. "Robot Language, PARL," National Technical Report Vol. 31, No. 4, Aug 1985.

[4] Richard P. Paul, Robot Manipulators: Mathematics, Programming, and Control, MIT Press, 1982.

[5] Masaru Nakano. "TL 10: A Programming Language for Assembly Robots," Jour. of Robotic Systems, 1985.

[6] 이기동, "SCARA형 로보트의 프로그래밍 언어 구성에 관한 연구", 서울대학교 대학원 석사학위 논문, 1987.

[7] SPRL 사용 설명서, 삼성항공, 1986.

[8] M. Selfridge and W. Vannoy, "A Natural Language Interface to a Robot Assembly System," IEEE Jour. Robotics & Automation, Vol. RA-2, No.3, Sep. 1986.

[9] Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice-Hall, 1978.

[10] I. J. Cox and N.H. Gehani, "Concurrent Programming and Robotics", The International Journal of Robotics Research, Vol.8, No.2, Apr. 1989.

[11] H.M. Deitel, An Introduction to Operation Systems, Addison Wesley, 1984.

[12] VAL-II USER'S MANUAL, Adept Technology, 1987.

[13] Richard D. Klafter, Robotic Engineering, Prentice-Hall, 1989.

[14] William Gruver et al., "Evaluation of Commercially Available Robot Programming Languages," 13th ISIR, Chicago, April 1983.

부록

타 로보트 언어와의 비교

	AL	HELP	MCL	RAIL	RPL	VAL	SAITEL
Language Type							
Subroutine					X		
Extension			X				
New language	X	X		X		X	X
Geometric Data Types							
Frame	X		X	X		X	X
Joint angle vector	X		X			X	X
Transform-ation	X		X			X	X
Rotation	X		X				X
Path				X			
Ability to Control Multiple Arms							
Single arm only				X		X	X
Multiple arms	X	X	X		X		
Control Modes							
Position	X	X	X	X	X	X	X
Guard moves	X	X					
Conveyor racking			X				
Motion Types							
Coordinated joint	X	X		X	X	X	X
Straight line	X		X	X	X	X	X
Spline	X	X		X	X	X	X
Circle				X			X
Implicit geo-metry pattern			X				X
Control Structures							
Statement labels		X	X		X	X	X
if-then	X	X	X	X	X	X	X
if-then-else	X	X	X	X	X		X
while do	X	X	X	X	X		
do-until	X			X	X		
case	X			X	X		
for	X			X	X		X
subroutine	X	X	X	X	X	X	X
Sensor Interface							
Vision	X	X	X	X	X	X	
Force	X	X					
Limit switch	X		X	X	X	X	X
Support Modules							
Text editor	X	X		X	X	X	X
File system	X	X		X	X	X	X
Interpreter	X	X		X	X	X	X
Complier	X		X		X		
Simulator	X		X				
Macro	X	X	X		X		
Debugging Features							
Single stepping		X		X	X		X
Breakpoints	X			X	X		X
Trace		X	X		X		X
Dump	X	X	X		X		X