'91 韓國自動制御學術會議論文集 1991. 10.22~24

실시간 운영 체제를 이용한 범용 로봇 제어 언어의 개발

°이 덕만 오 종한 이 진수

포항공대 전자전기과

The Development of General Purpose Robot Language Based on Real Time Operating System

DukMan Lee JongHan Oh Jin S. Lee
Dept. of Electron & Electric Eng.

Phohang Institute of Science and Technology

Abstract

We need general developing environment to contorl robot with effect but less energy. So, software and hardware tools are very important. In this paper, we present a general-purpose robot control language and its implementation on Real Time O/S and VME bus system.

The system runs on the VMEexec Real Time Operating System and robot program is written in the "C" language. The developed program is linked with the robot control C library to produce an executable image . Under the developed robot control environment, the user can write a general high-level control program leaving all the specific information about the robot in a robot specific file.

I. 서론

요즈음, 고속의 마이크로 프로세스가 급속히 개발되어 저렴한 가격으로 보급되고 이런 마이크로 프로세스를 채용한 workstation 및 PC 가 널리 보급되고 있다. 이러한 고속 하드웨어의 발전 가속화는 기술적으로 상당히 진보되고 복잡한 실시간 응용 프로그램이 손쉽게 처리될수 있는 하드웨어 환경이 점차 구축되어 가고 있음을 의미한다. 또한 이런 하드웨어의 발전과 더불어 고속의 프로세싱 (processing) 환경을 지원하기 위한 실시간 소프트웨어의 등장 또한 주목할 만 하다. 기존의 실시간 제어 소프트 환경은 UNIX 나 DOS 에 기반을 둔 소프트웨어가 대부분 이었다. 이는 UNIX 와 DOS 가 다양한 개발도구 (developing tool) 와 네트워킹 기능, 편리한 사용자 인터페이스, 강력한 디버깅 기능을 제공함으로써 소프트웨어 개발자에게 상당한 편이와 융통성을 주었기 때문일 것이다. 그러나 UNIX 나 DOS 의 경우 는제반 프로세싱 환경을 염두에 두고 개발된 운영체제이기 때문에 실시간 시스템에서 엄격히 요구되는 시간제약 조건을 만족하지 못하고 있다. 근간에는 이런 UNIX 나 DOS 의 단점을 보완한 신뢰성 있는 실시간 운영체제 (Real - Time operating system) 들이 상당수 등장하고 있다. 이러한 고도의 하드웨어 및 소프트웨어 환경의 생성은 개발자의 측면으로서는 상당히 부가가치가 크고 파급효과가 지대한 일이라 할 수 있다. 따라서, 본 논문에서는 하드웨어와 소프트웨어의 제약으로 말미암아 복잡하고도 정교한 알고리즘의 수행이나 여러 다양한 센서의 통합에 문제점이 많았던 현 컴퓨터 환경을 지양하고 미래 지향적인 로봇 제어 하드웨어 및 소프트웨어를 염두에 둔 범용 로봇 제어 환경을 제시하고자 한다. 이러한 로봇 제어 환경은 개방 하드웨어 구조여서 여러 시스템 I/O 들의 조합이 용이하고 실시간 운영체제의 신속한 작업 처리능력 또한 로봇에 요구되는 실시간 제한을 충실히 지켜주고 있어서 사용자에게는 상당한 유용성과 융통성을 제공해 준다. 본 논문에서는 세계적으로 널리 쓰이고 있는 VME 버스 시스템에 기반을 둔 모토로라 (Motorola) 델타 계열 (Delta series) VME 랙 (rack) 을 이용한 범용 로봇 제어 환경을 구축하였다. 이 랙은 보통 12 장의 보드를 장착 할 수 있는 슬롯 (slot) 이 있고 SCSI 인터페이스, ethernet, X-window 기능 등 컴퓨터 에서 제공하는 일반 기능을 모두 제공하면서 각 슬롯에는 사용자가 원하는 각종의 CPU 보드 및 I/O 보드를 장착할 수 있도록 하고 있다. 운영체제로는 호스트 (host) CPU 에서 sysV68 UNIX 를 타겟 (target) CPU 에서는 VMEexec 실시간 운영체제을 사용하였는데 이 실시간 운영체제는 멀티 태스킹 (multi-tasking), 태스크 우선순위 (task priority), 선점 (preemtion), 시간 함수 (time function), 동기화 매카니즘 (synchronization), 즉 메세지 (message), 큐 (queue), 세마포어 (semaphore) 등 각가지 운영체제의 기능을 제공한다.

일반 로봇 시스템은 모터, A/D 변환기, D/A 변환기, 시각 센서, 힘 센서, 제어 CPU, 통신, 제어 알고리즘 등 온갖 부분을 가지고 있다. 그러므로, 본 범용 로봇 제어환경에서는 이러한 부분을 체계적인 방법으로 해결하기 위하여 다음의 두 측면에서 접근하였다. 첫째, 아나로그 서보 (analog servo) 를 제외한 기존의 로봇 제어기 (controller) 를 모두 VME 버스 랙 안에 장착하고 모든 기능을 디지탈화 한다. 둘째, 실시간 운영체제에 개발된 범용 로봇 제어 환경을 구축한다.

따라서 본 논문에서는 먼저 시스템 구성의 측면을 먼저 간략히 기술하고 범용 로봇 제어 언어의 기능을 중점적으로 설명하고자 한다.

II. 본론

1. 범용 로봇 제어 시스템 구성

본 논문에서 제시한 하드웨어 환경은 그림 1. 에서 보는

바와 같이 VME 버스 환경의 개방구조 (open architecture) 이다. 슬롯 1 번은 MVME 147 SA 보드로서 8 Mb 램 메모리에 68030 CPU, 25 Mhz 의 성능을 가지고 있다. 이 보드 상에서는 sysV68 UNIX 운영체제가 돌아감으로 사용자는 UNIX 의 강력한 편집 기능, 사용자 인터페이스 기능, 네트워킹 기능 등 여러 유용한 자원을 이용할 수 있다. 슬롯 2 와 3 에는 슬롯 1 과 같은 성능의 보드가 2 장 장착되어있고 슬롯 1 에서 개발된 범용 로봇 제어 프로그램이 다운로드 (download) 된다. 이때, 이 응용 태스크들은 슬롯 2, 3 에 미리 다운 로드 되어있던 VMEexec 실시간 운영체제에 의해 적절히 스케쥴링 받는다. (그림 2.) 슬롯 2 에서는 일반적인 상위 로봇 제어 프로그램이 돌아가고 슬롯 3 에서는 하드웨어와 직접 관련된 제어 루틴이 돌아간다. 슬롯 4 는 A/D 보드 슬롯 5 는 전류 출력 보드 슬롯 6 는 시리얼 통신 보드 슬롯 7 은 디지탈 I/O 보드로서 이들 4, 5, 6, 7 슬롯 보드들은 로봇와 직접적인 하드웨어 인터페이스를 담당한다. 각각의 보드들은 VME 후면 (back plane) 버스로 연결되어 있어 데이타 교환의 용이성과 신속성이 보장된다.

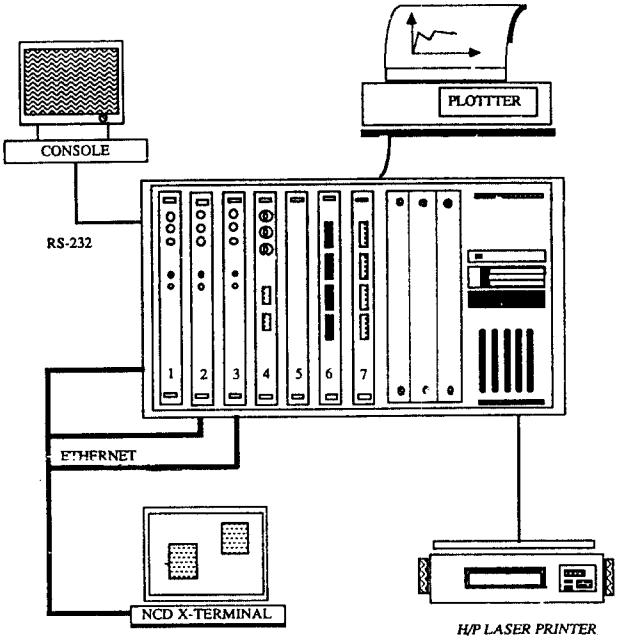

그림 1. 시스템 구조

2. 범용 로봇 제어 소프트 환경

앞에서 제시한 하드웨어 구조로 로봇을 제어하기 위한 소프트웨어는 크게 로봇 제어 언어부와 하위 I/O 인터페이스의 두 부분으로 구성되어 있다.

로봇 제어 언어부는 사용자가 로봇의 동작을 조절하고 I/O 들을 관할하기 위한 프로그램을 작성하는데 용이한 도구를 제공한다. 이 부분에는 로봇의 경로를 사용자가 오프라인 (off-line) 에서 프로그램 할 수 있는 환경이 제공되며 작성된 프로그램을 토대로 로봇의 목표치를 계산하고 그 값을 하위 제어 태스크에 전달해 주는 궤적 생성기 (tragectory generator) 가 지원된다. 이들에 대한 개략도는 그림 3. 에 나타나 있다. 궤적 생성 태스크는 매 20 mSec 마다 자신의 루프를 돌면서 사용자 명령을 기다린다. 사용자 명령이 있으면 이 태스크는 로봇이 움직여야 할 경로 계산을 하고 매 샘플 타임 마다 그 값을 디지탈 서보에게 넘겨준다.

하위 I/O 인터페이스부는 각종의 I/O 보드와의 통신 및 디지탈 서보를 관장하는 프로그램이 독립된 태스크로서 존재하고 있다. 디지탈 서보는 매 5 mSec 마다 PID 알고리즘을 실행하고 그 출력을 아나로그 서보에게 보낸다. 궤적 생성기와 디지탈 서보는 각기 다른 보드상에서 실행되므로 두 소프트웨어의 통신을 위해서 메세지 전달 방식을 사용하였다.

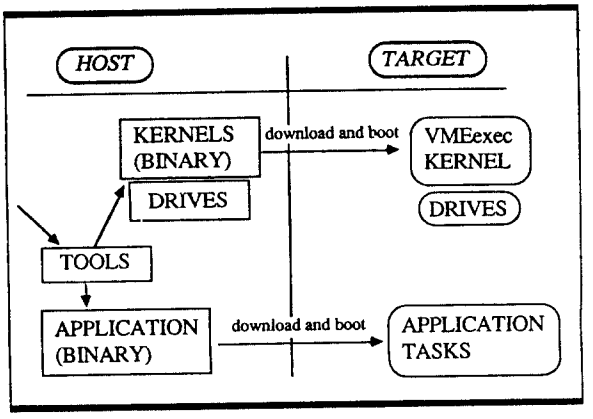

그림 2. 응용 소프트웨어 실행 환경

3. 범용 로봇 제어 언어의 구성 및 기능

사용자는 일반적으로 하드웨어나 로봇의 구체적인 내부 알고리즘을 모르고 구체적으로 알 필요가 없는 경우가 대부분이다. 따라서 미래의 로봇 제어 언어는 다음의 몇 가지 조건을 만족시키는 형태가 요구된다.

첫째, 앞으로는 고속, 대용량, 저가격 컴퓨터가 일반화 될 것이므로 일반 컴퓨터의 사용이 필요하다. 종래의 로봇 제어 언어들은 일반 Workstation 이나 PC 같은 일반 컴퓨터를 이용하지 않고 자체의 하드웨어 상에서 제어 언어를 처리하였으므로 일반 컴퓨터에서 얻을 수 있는 여러 다양한 기능들이 제약을 받았다. 예로 언어 내부에서 부동 소숫점 처리, 서브루틴 (subroutine) 에 인자 (argument) 전달, 문자 스트링 사용 등에 관한 면 외에 주변 하드웨어와의 접속 및 그래픽 시뮬레이션과 같은 다양한 작업을 수행하는데 대한 사용자의 융통성이 제한되었다.

둘째, 로봇 종속적인 (robot dependant) 세부 사항을 몰라도 작업 지시를 내릴 수 있는 형태가 바람직 하며 고급 사용자들을 위해서는 보다 진보적인 형태의 작업을 수행할 수 있는 환경을 제공하여야 한다.

세째, 로봇의 종류에 따라 프로그램의 전반적인 내용이

가변적이여서는 안된다.

이런 맥락에서 본 논문에서 제시하는 범용 로봇 제어언어 (General Purpose Robot Language) 는 위의 사항을 충실히 만족하도록 구성하였다. GPRL 에서는 로봇 종속적인 내용을 로봇의 기종에 관계없이 범용적으로 쓰일 수 있도록 하기 위하여 하드웨어적 종속적인 부분과 그렇지 않은 부분을 분리하여 C 라이브러리 (library) 화 함으로써 범용성을 보장하였고 C 언어에서 얻을 수 있는 갖가지 장점들을 수용하였다. 갖가지 로봇 종속적인 내용은 함수화 되었고 함수의 인자들을 통해 사용자들은 자신의 목적을 달성할 수 있으므로 객체지향적인 (object oriented) 형태가 되었다. 고급 사용자들은 자신이 원하는 형태의 기능을 손수 함수화 할 수 있고 이를 로봇 라이브러리에 추가하여 자신의 목적에 맞추어 사용 할 수 도 있다. 또한 모든 프로그램이 실 시간 운영체제의 환경속에서 실행되므로 본 라이브러리에서 제공 되지 않는 실시간 환경 함수 지원을 받아 작업을 수행 할 수 있는 잇점이 있다.

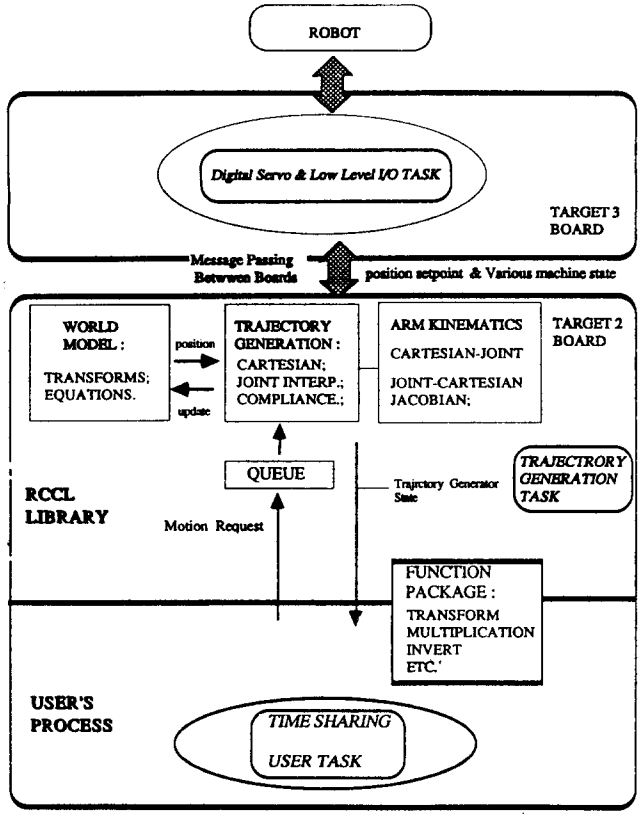

그림 3. 범용 로보트 제어 소프트 환경

3.1 공간 좌표 규정 (World Modeling)

로봇 프로그램은 그 자체가 어떠한 대상을 3 차원 공간에서 움직이는 것을 전제로 하고 있기 때문에 이러한 동작을 표시할 어떤 도구가 필요하다. 공간 상에서 입체를 완벽히 수학적으로 기술 하기 위해서는 위치 (position) 과 방위 (orientation) 의 정보가 필요하다. 따라서 이것들을 표시할 가장 기본적인 요소로 새로운 데이타 형 (data type) 을 정의 할 필요가 있다. 이렇게 새로 정의된 형을 이용하면 joint angle sets, Cartesian position, orientation 등을 표시할 수가 있으며 또한 이에 사용되는 연산도 정의 할 수가 있다. 3 차원 공간에서 의 동작을 표시하는 가장 일반적인 데이터 형 으로 homogeneous transform 이 있다. HT(homogeneous transform) 는 4 x 4 matrix 로써 3 차원의 position 과 orientation 을 함께 표시하는 가장 적절한 데이타 형으로 많은 제어언어에 사용되고 있다. 물론 HT 에는 물체의 크기나 형태, 무게나 부피등의 정보는 포함되어 있지않지만 물체를 한 곳에서 다른 곳으로 움직이는 기본적인 경우는 HT로 실현 시킬 수가 있다. 그러나, 작업장에 여러가지 복잡한 물체나 기기들이 많이 있거나 어떤 물체는 상당히 부피가 클 경우 로봇이 작업을 할 때 충돌 문제도 고려해야 하므로 보다 완전한 world modeling 을 위해서는 물체의 크기나 모양도 고려한 CAD 형 (CAD type) 의 world modeling 을 해야한다. 이러한 데이타가 주어진다면 프로그래밍도 자연스럽게 task-level programming system 을 이루어 갈 수가 있는 것이다. 예를 들면 위에서 언급한 CAD model 이 있을 경우 자동 충돌 감지나 path plan 등은 사용자가 일일이 입력시킬 것 없이 프로그래밍 시스템이 알아서 해 줄 수가 있는 것이다. 물론 현재로서는 이러한 복잡한 modeling 을 해 주기에는 컴퓨터의 계산속도나 memory 에는 한계가 있지만 현 컴퓨터의 추세로 보아 CAD modeling 이 로봇제어 언어의 기본으로 들어갈 날이 멀지않은것 같다. 그러면 이절에서 소개한 world modeling 의 응용예를 살펴보자.

3.1.1 위치 방정식 (Position Equation)

로봇 태스크 (robot task) 란 공간에 도달할 점을 결정하고 그 점에 로봇이 도달하도록 매니퓨레이터 (manipulator) 를 움직이는것 이라고 할 수 가 있다. GPRL 에서는 이를 수학적으로 계산하기 위해서 Homogeneous Transform 을 이용한 위치 방정식을 세우고 있다. 즉, 각 매니퓨레이터의 관절에 고정된 축을 정하고 축과 축의 상대적인 위치를 Homogeneous Matrix 로 구성한다. 그런다음, 각 메트릭스들을 가고자 하는 위치에 대한 닫힌 폐쇠기구 경로 (closed kinematics chain) 식을 만들고 그 방정식이 만족되도록 로봇의 매니퓨레이터를 움직이는 방식을 쓰고있다. GPRL 에서 쓰는 일반적인 폐쇠 기구 경로식은 다음과 같다.

$$BASE \cdot T_6 \cdot TOOL = OBJ \cdot GRASP \cdot DRIVE$$

여기서 각 변환의 정의는 다음과 같다.

BASE : 기준 좌표에서 로봇의 어깨까지 변환.

T_6: 어깨에서 로봇의 선단부 (end effector) 까지 변환.

TOOL : 선단부에서 도구까지의 변환.

OBJ : 기준 좌표에서 작업 대상 물체의 기준 좌표 까지 변환.

GRASP : 도구에서 작업 대상 물체 까지 파지 위치를 나타내는

변환.

이렇게 식이 적힐때 변환 T_6는 DRIVE 변환이 원하는 위치에서 identity 가 되도록 변화된다. 즉, 그 T_6의 변환이 로봇의 선단부 (end effector) 동작이 되는 셈이다. GPRL 에서는 위치 방정식의 각 항을 구성하는 Homogeneous 변환 Matrix 가 아래의 역할을 할 수 있도록 하는 융통성을 준다.

상수 변환 (constant transform) : 이 변환은 궤적 수행시 항상 matirx 의 값이 정적 (static) 하게 보존된다. 이는 일반 로봇 제어 언어에서 흔히 볼 수 있는 상수 matirx 형이며 실제 프로그램에서는 일정한 값이므로 각각의 matirx 를 일일이 달고다니지 않기 위해 미리 곱해진다.

홀드 변환 (Hold Transformations) : 이 변환은 사용자 프로세스의 실행 전 구간에 걸쳐 쓰고 읽을 수 있다. 그러므로, 이 변환식이 바뀌면 궤적도 즉시 그 변화를 수용하게 된다. 이런 형의 변환은 교시 장치 (teach pendant) 나 조이 스틱 (joy stick) 을 이용한 경로 온 라인 수정시 두움이 된다.

함수화 변환 (Functionally defined transformations) : 이 변환은 함수형의 포인터를 이용하여 궤적 계산시 반영되는데 변환식은 "C" 함수안에 포함되고 해당 궤적 계산 동안 에 이 "C" 함수 내부의 변환식이 매 샘플 레이트로 계산 된다. 함수를 궤적 계산의 위치방정식에 끼워 넣을 수 있으므로 변환식 외에 부가 기능에 대한 사항을 함수 내에 포함 시킬 수 있다. 물론 이 함수의 실행 시간은 충분히 짧아야 한다. 예로서 계산될 궤적이 어떤 함수 인자의 값이라면 parametrized 궤적을 얻을 수 있고, 또 그 값이 센서 값의 함수이면 매니퓰레이터를 제어하기 위한 동기화된 센서 피드백 값을 얻을 수 있다.

3.1.2 GPRL implementation

(1) Homogeneous Transform 의 구현

GPRL 함수는 사용자가 다이내믹하게 homogeneous 변환을 생성케 해 준다. 기본적인

함수 호출은 다음과 같다.

```
t = newtrans ( type ) ;
after call of newtrans  4 x 4 matrix t is
```

$$t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

여기서 t 는 생성된 homogeneous 변환의 포인터이다. 그리고, type 은 원하는 형태의 변환을 규정한다. 즉, 앞에서 기술한 const, hold, valb 형을 말한다. 여기서 생성된 변환식의 값은 항등변환식인데 이 기능을 원하면서 초기화 과정을 원한다면 다음 함수 호출 형태를 취할 수 있다. 예를 들어,

```
t= gentr_rot( px, py, pz, v, a );
```

여기서 px, py, pz 는 병진 운동부분을 나타내고 v, a 는 v 벡터를 중심으로 a 각도 만큼 회전을 뜻한다. 위와 더불어 오일러 각도, roll pitch yaw 각도 등을 다루는 함수가 GPRL 에 있다.

기타

```
gentr_trsl(px, py, pz);
gentr_eul(px, py, pz, phi, the, psi);
gentr_rpy(px, py, pz, phi, the, psi);
```

(2) 위치 변환식 (position equation) 의 구현

앞에서 생성된 변환 자료를 바탕으로 GPRL 에서는 위치 방정식을 makeposition 이라는 함수를 호출하여 만든다. makeposition 에서 기술된 위치 방정식은 궤적생성기(trajectory generator)에서 자동적으로 인식되고 이 인식된 자료를 근간으로 DRIVE 변환을 계산하도록 되어있다.
makeposition 은 가변 인수 (variable argument) 를 인자로 쓰고있는데 이 방식은 위치방정식을 생성하는 GPRL 고유의 기능이다. 예를 들어, base, tool, obj, grasp이라는 변환이 위의 gentr_trsl(), 이나 gentr_rot(), gentr_rpy(), 등을 통해 생성 되었다고 하자. 이는 각각이 homogeneous 변환을 나타냄을 이미 이야기 한 바가 있다. makeposition 에 대한 "C" 의 정의는 다음과 같다.

```
POS_PTR makeposition( lhs[ , lhs ] ··· ,EQ, rhs, [ , rhs ], ··· ,TL, tl );

TRSF_PTR lhs, ···,rhs, ···,tl ;
```

위 예제에 해당하는 함수 호출은 다음과 같다.

```
POS_PTR p;
p = makeposition( base, t6, tool, EQ, obj, grasp, TL, tool );
```

여기서 EQ 는 lhs 와 rhs 를 구별하기 위한 구별자이고 TL 은 tool 이 무엇인가를 알려주는 구별자이다. makeposition 함수는 변환 그래프를 만들어주고 그 ring data 구조의 포인터를 반환하는 역할을 한다. 사용자에게 position 은 "C" 구조체로 구현되어 있는데 그 정의는 다음과 같다.

```
struct position {
        char *name;
        int   code;
        float scale;
        event end;
```

여기서 code 는 운동(motion)의 종료시 그 이유를 표시하고 scale 은 운동(motion)시 0 에서 1 로 변화하는데 DRIVE 변환의 인자로써 쓰이며, 궤적 경로의 중간점에서 사용자 프로세스를 동기화시키는 작용을 한다. 마지막으로 end 는 해당 운동(motion)의 종료를 알리는 event count 이다. (그림 4.)

MAKEPOSITION EXAMPLE

p1 = makeposition(Z, T6, E, EQ, P, PA, TOOL, E)

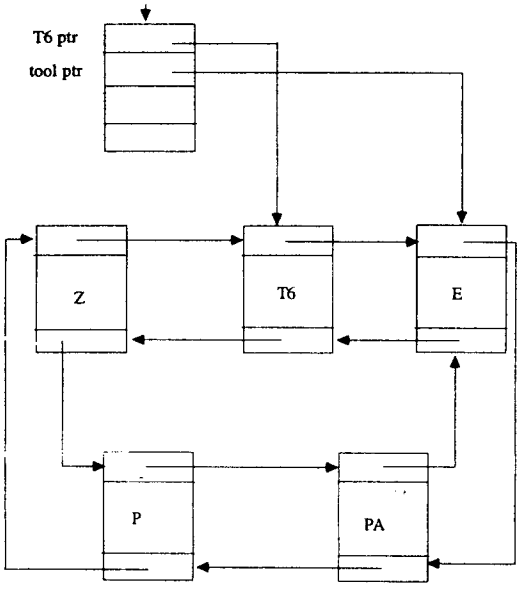

그림 4. MAKEPOSITION 의 원리도

(3) 매니퓨레이터 운동 교시 (manipulator motion specification)

다음 함수 호출은 궤적 생성기(trajectory generator)에 운동요구를 전달하는 작용을 한다.

```
move ( p );
```

이 함수를 호출하면 실제 앞에서 만든 위치방정식에 대한 정보와 아래에 기술될 운동에 관한 정보를 다음의 테이블로 구성하여 큐(queue)에 저장시키는 기능을 한다.

```
typedef struct {
        PST *goalposition;
        EFT *eft;
        char defaltInterModSet;
        int  defaltInterMode;
        char transitionTimeSet;
        int transitionTime;
        char segmentTimeSet;
        int segmentTme;
        char velocitySet;
        int translationalVelocity;
        int rotationalVelocity;
        char configurationSet;
        int configuration;
        int (*monitorFunction) ();

      TRSF *updateTransform;
        PST *updatePosition;

        char sampleRateSet;
        int sampleRate;
        float objectLoad;

        int numHoldXforms;

        TRSF tC0;
} MOTION_REQUEST;
```

3.2 궤적 생성 (Trajectory Generation)

GPRL 은 2 가지 형태의 궤적으로공간 상에서움직일 수 있도록 하고 있는데 그 하나는 joint mode 이고 다른 하나는 cartesian mode 이다.

3.2.1 JOINT MODE

각 운동요구에 대해 최종 매니퓨레이터의 위치 식이 정해지면 joint setpoint 가 inverse kinematics 로 정해지고 그 셋포인트를 매 샘플링 시간마다 선형 보간한다. 이런류의 운동은 아주 효율적이고 단지 조인트 가속도와 최대 조인트 속도에만 제한을 받는다. 이 운동의 궤적은 직선이나 예측가능한 경로는 아니지만 각 셋 포인트 간을 움직일때 최소시간이라는 관점이 중요시 될때 즉, 큰 운동 궤적이 요구될 때는 아주 용이하다.

2.3.2 CARTESIAN MODE

매니퓨레이터의 경로를 매 샘플 시간 마다 DRIVE 함수를 써서계산하고 그것의 inverse kinematics 식을 풀면 joint setpoint 가 주어진다. 이때 그 경로는 직선 경로이고 예측가능경로이다. 단 항상 싱귤래러티(singurality)의 가능성이 있는것이 좀 흠이라 하겠다. 이 운동에 있어 아주 중요한 역활을 하는 것이 DRIVE(s) 함수인데 이는 PAUL(1981) 의 논문에 상세히 기록되어 있다.

3.3 동기화 (Synchronization)

동기화는 프로그램이 센서나, 운동 종료 조건시, 또는 가변 변환 (variable transformation) 이 매니퓨레이터의 위치를 변화시키는 것과 같이 외부정보에 의존할 때 필요하다. GPRL 은 실시

간 운영체제 환경 속에서 실행되므로 외부 사건과의 동기나 내부 태스크간의 동기를 위한 프로그램은 작성하지 않고 VMEexec 실시간 운영체제에서 제공하는 시스템 콜 (system call) (Event , Sepaphore, Queue, Message, Signal 등) 을 이용한다.

보통 로봇 프로그래밍 언어에서는 MOVE 라는 문장이 매니퓰레이터의 운동과 동기화 되어있다. 그러나, 가장 좋은 경우라면 그 선택이 사용자에게 맡겨져 있어야 할 것이다. 즉, MOVE 실행 후 프로그램이 계속 될 것인지 아니면 그 운동이 끝날 때까지 프로그램을 정지시켜야 할지를 사용자가 결정하도록 해야한다는 뜻이다. GPRL 에서도 move 라는 명령어가 있기는 하나 move 명령어와 동기되어 로봇이 움직이는 것은 아니다. GPRL 에서는 특이하게 큐잉(queueing) 구조와 동기화 프리미티브(primitive)를 사용해 다소 복잡하기는 하나 아주 유연성있는 환경을 제공하고있다. 몇 개의 운동요구사항이 먼저 프로그램 되고 move 명령어에 의해서 운동 (motion) 이 실행되는 것이 아니라 호출 순서 대로 큐 (queue) 속으로 들어간다. 그 각각은 큐에서 자신의 운동(motion)이 수행되기를 기다리고 있는데 이들을 어떻게 서비스하느냐는 오로지 사용자의 권한이다. 즉, 하이레벨 (high level) 프로그램에서 사용자가 선택을 한다. 사용자의 선택에 의해 각 운동은 동기화 혹은 비동기화 될 수 있다.

4. 결 론

본 논문에서는 새로운 컴퓨터 환경을 이용한 로봇 제어 시스템을 보였다. 이러한 시스템 구성은 종래의 하드웨어 및 소프트웨어로서는 어려움이 많았던 부분을 많이 극복할 수 있고 또한 미래 지향적인 시스템 구성임을 믿어 의심치 않는다. 본 시스템은 로봇 제어 환경 조성에 쓰였으나 로봇이 가지고 있는 종합적 성질을 감안할 때 일반 제어 시스템에도 표준적인 구조가 될 수 있을 것이다. 앞으로의 과제로는 이런 개발환경을 바탕으로 진보된 알고리즘의 적용과 현장 적용 그리고 다중 로봇에의 확장등이 남아 있다.

참고 문헌

[1] Jin S. Lee, Samad Hayati, Vincent Hayward and Jhon Lloyd, "Implementation of RCCL, a robot control C library, on the micro VAX II", SPIE Conference on the Intelligence Robotic Systems, Cambrigdge, MA, oct. 1986

[2] Hayward V., "RCCL Version 1.0 User's Manual.", Purdue University Technical Report , TR-EE 83-46. oct. 1986

[3] Hayward v., Paul R. P., "Introduction to RCCL: A Robot Control C Library." IEEE First International Conference on Robotics, Atlanta., June 1984

[4] Hayward v., Paul, R. P., "Robot Manipulator Control Under Unix: RCCL A Robot Control C Library.", International Journal of Robotic Research, 5(3). fall 1986

[5] Paul, R. P. , "Manipulator Catesian Path Control", IEEE Transanctions on Systems, Man and Cybernetics. Vol. SMC-9, Number 11, Nov. 1979, pages 702-711

[6] Paul, R. P., "Robot Manipulators: Mathematics, Programming, and Control", Cambridge: MIT Press 1981