다중로봇을 위한 관리제어 시스템의 설계

論 文
45~1~15

A Design of Supervisory Control System for a Multi-Robot System

徐 一 弘* · 呂 熙 珠** · 金 載 顯** · 柳 宗 蔥*** · 吳 尙 錄§
(Il Hong Suh · Hee-Joo Yeo · Jae-Hyun Kim · Jong-Seock Ryoo · Sang-Rok Oh)

Abstract - This paper presents a design experience of a control language for coordination of a multi-robot system. To effectively program job commands, a *Petrinet-type Graphical Robot Language*(PGRL) is proposed, where some functions, such as *concurrency* and *synchronization*, for *coordination* among tasks can be easily programmed. In our system, the proposed task commands of PGRL are implemented by employing formal model languages, which are composed of three modules, *sensory*, *data handling*, and *action module*. It is expected that by using our proposed PGRL and formal languages, one can easily describe a job or task, and hence can effectively operate a complex real-time and concurrent system. The control system is being implemented by using VME-based 32-bit microprocessor boards for supervisory, each module controller(arm, hand, leg, sensor data processing module) and a real time multi-tasking operating system(VxWorks).

Key Words : PGRL(Petrinet-type Graphical Robot Language), Formal Model Language

1. 서 론

자동화된 제조 분야에서 복잡하거나 정교한 작업들을 수행하기 위해 work cell안에서 여러대의 로봇 디바이스들을 제어할 수 있는 지능적인 콘트롤러를 개발해야 할 필요성이 증대되고 있다. 이를 위해서, 일반적인 task나 job을 프로그래밍할 언어를 설계해야 할 필요가 생겼고, 이에 따른 인터프리터와 컴파일러도 로봇 제어기와 함께 개발되어야만 한다. 그러나, 지금까지 대부분의 기존 로봇 제어 언어들은 특정한 동작들을 나타내는 의미 명령어로 구성된 작업기술형태로써 대부분 한대 또는 두대의 로봇을 제어하도록 개발되었다. 그러나 그러한 언어들은 기능적인 제약 때문에 광범위한 응용 면에서 한계가 있으며 특히, 그것의 내부적인 속성이 단위 동작들 간의 동기성(Synchronization) 및 동시성(Parallelism)을 표현하는데는 적합하지 않다[1,2,6,8]. 이는 여러대의 로봇이 수행할 작업을 한대의 제어기에서 프로그램 할때 기존의 Text-based 언어로는 각 로봇이 동시에 수행해야 할 작업을 표현한다거나 각 로봇이 수행할 작업들 간의 상호관계(동기성, 선행조건,..)를 표현할 수 없거나 있다 하여도 초보적인 단계에 머물러 사용에 불편함이 있었다. 본 연구에서는 이러한 문제를 해결하기 위하여 다중로봇 작업의 지시를 각 로봇이 수행해야할 단위 동작을 프로그램하는 부분과 이러한 단위동작들을 이용하여 여러대의 로봇이 함께 수행해야할 전체적인 작업을 지시하는 두개의 부분으로 구분하고 이에 대한 새로운 형태의 제어언어를 개발하였다.

* 正 會 員 : 漢陽大 工大 電子工學科 敎授 · 工博
서울大 新技術硏究센터 硏究員
** 正 會 員 : 漢陽大 大學院 電子工學科 博士課程
*** 正 會 員 : LG産電 硏究員
§ 正 會 員 : KIST 情報電子硏究部 責任硏究員 · 工博
接受日字 : 1995年 7月 24日
1次 修正 : 1995年 10月 16日
2次 修正 : 1995年 12月 13日

첫번째로 로봇의 단위동작을 프로그램 하는 부분은 기존 로봇 제어기에서 Primitive motion을 개발하는 과정에 해당하는 것으로 로봇동작의 구조적 특성을 잘 반영하며 Task단위 동작을 개발하기에 적합한 Formal Model 구조를 갖는다[4]. 여기서 로봇 동작의 특성은 그 응용범위의 확대 및 센서 사용의 증가와 더불어 센서의 입력이 로봇의 동작에 영향을 끼쳐 주변환경과의 상호작용을 일으키게 되며, 또한 로봇 프로그램은 계층적(hierarchical)이고 유동적(flexible)이며 재귀적으로(recursively) 정의될 수 있는 특성을 갖는다. 이러한 특성들에 바탕을 두어 Sensor-based Robot Program에 유용하도록 설계된 Formal Model은 단위 Task를 수행하기 위해 필요한 여러개의 computing agent들이 동시에 상호 작용을 일으키며 분산처리 방식으로 처리되는 계산구조를 말한다. 본 연구에서 단위 task를 수행하기 위해 필요한 computing agent는 Module이라 정의하며, input port 및 output port를 갖는 port automata형태를 갖는다. Module은 각각 자신만의 고유한 작업을 담당하여 처리하도록 정의되며, 이러한 module들 간의 연결을 정의하여 원하는 로봇 동작을 프로그램하게 된다.

로봇 작업 프로그램의 두번째 부분으로는 Formal Model로 정의된 각각의 단위동작들을 이용하여 실제 로봇이 수행해야 할 일의 흐름을 지시하는 것으로써 지금까지 대부분의 로봇 제어기에서는 Text로 표현된 작업명령을 일의 순서에 맞게 나열함으로써 로봇에게 작업을 지시하였다. 이러한 작업지시 방법은 한대의 로봇에 대한 작업을 프로그램 할 경우 실제 로봇작업의 흐름과 text로 표현된 로봇프로그램의 흐름이 동일하므로 매우 유용하게 사용되며 이해하기에도 편리하였다. 그러나, 이러한 text-based 제어언어를 이용하여 두대 이상의 로봇을 제어할 경우 로봇 프로그램의 흐름은 하나인데 반해 실제 작업의 흐름은 두가지 이상이 되므로 사용이 어려웠다. 이러한 문제를 해결하기 위한 한가지 방법으로 기존 text-based 제어언어에 두대 로봇을 위한 동시동작 명령어들을 추가한 제어언어 들이 개발되고 있다[12,13].

본 논문에서는 여러대의 로봇을 동시에 제어하기 위한 제어언어로써 지금까지의 접근방법과는 전혀 다른 Petri net을 이용한 Graphical Robot Language를 제안하고 구현하였다. Petri net은 기계, 전기, 전자, 컴퓨터 등 여러 분야에서 주어진 시스템을 모델링하여 해석하거나 설계하는 수단으로 유용하게 사용되는 방법이다. 로보틱스 분야에서도 로봇이 사용되는 작업공정을 Petri net으로 모델링하여 공정 설계의 최적화 및 작업공정의 생산성등의 성능을 평가하는 수단으로 이용되고 있다[9,10,11]. 특히 [9]에서는 하나의 작업대에서 여러대의 로봇이 동시에 작업을 수행하는 공정에서 각각의 로봇이 수행해야 할 작업을 Petri net으로 모델링하고 이들 단위 작업간의 상호관계를 정의함으로써 작업효율을 높이는 연구가 수행되었다. 그러나 본 연구에서는 주어진 시스템 혹은 작업을 모델링하고 이를 평가, 해석하는 기존의 응용방식과는 달리 개발된 Graphic Editor상의 Petri net을 이용하여 로봇이 수행할 작업을 프로그램하는 다중로봇을 위한 작업기술 수단으로 Petri net을 이용하였다. 본 논문에서는 로봇의 작업을 기술하는데 동시성의 표현이 뛰어난 Petri net을 이용함으로써 다중로봇의 작업을 효율적으로 프로그램 할 수 있게 되었으며 Petri net으로 기술된 작업프로그램을 해석하여 로봇 작업들간의 교착상태(Deadlock) 및 자원공유 문제(Resource Allocation)등을 사전에 검색하여 논리오류(Logical Error)를 방지할 수 있게 되었다.

이와같은 두가지의 새로운 제어언어를 사용하는 다중로봇 제어시스템의 전체 시스템 구조는 그림 1과 같으며 다음과 같은 세가지 주요 제어유닛으로 구성된다.

(1) PGCU(Petrinet-type Graphical Compiler Unit)

PGRL Graphic Editor에서 사용자가 작성한 program을 compile하여 job program에 대한 data file을 생성하는 기능을 담당한다. 사용자가 작성한 program은 PGRL을 이용한 graphical symbolic high-level program과 Formal Model language program으로 구성된다. 이러한 program은 PGCU의 compiler에 의해 task간의 관계를 표현하는 TRM(Task Relation Matrix)과 사용자가 정의한 task를 수행하는데 필요한 세부 정보를 포함하는 data file로 컴파일 된다.

(2) TCU(Task Coordination Unit)

TCU는 PGCU에서 생성된 job program에 대한 data file을 이용하여 전체 job을 구성하는 task들간의 제어를 담당한다. TCU는 크게 PGRL program의 실행을 담당하는 PGRL controller와 Formal Model Language로 표현된 task의 실행을 담당하는 Formal Model controller로 구성된다.

(3) DSPU(Device Control and Sensor Signal Processing Unit)

DSPU는 로봇의 각 축을 구동하는 독립된 device control unit과 sensor signal processing unit으로 구성된다. 그 중에서, device control unit은 inverse kinematics를 풀어 actuator를 구동함으로써 동시에 multi-servo actuator를 제어하며 이들 controller는 필요하다면 hybrid position/force control 또는 impedance control과 같은 position/force를 제어하는 더 향상된 알고리즘을 사용할 수 있도록 설계되었다[12,13]. 한편, sensor signal processing unit은 각 로봇에 부착된 각종 센서신호의 전 처리 기능을 담당한다.

제안된 개념을 입증하기 위해서, VME-bus방식인 32비트 마

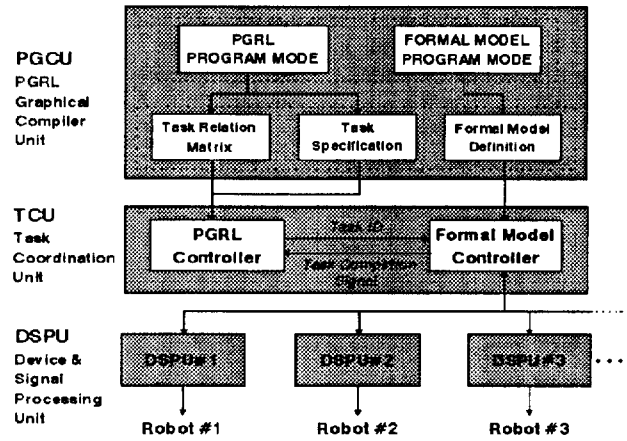


그림 1 다중로봇 제어시스템의 전체 시스템 구조

Fig. 1 Overall system architecture of multi-robot control system

이크로프로세서인 MC 68030/ 6882-based CPU[16]와 실시간 다중처리 운영체제인 VxWorks[17]를 이용하여 제어 시스템을 구현하였으며 SCARA 수평다관절 로봇과 PT-200 5축 수직다관절형 로봇을 이용하였다[14-17].

2. Formal Model Language for PGRL

2.1 Formal Model Language의 개요

로봇 제어언어의 개발에 있어서 일반적인 접근 방법은 범용 프로그래밍 언어에 로봇 제어를 위한 몇 개의 특별한 primitive들을 추가하여 로봇 프로그램에 사용하는 것이며, 또 다른 하나의 방법은 로봇 프로그래밍을 위한 고유한 computation model을 개발하는 것이다[4]. 각종 센서의 사용 및 로봇의 고기능화 그리고 그 응용범위의 확대로 인하여 로봇의 작업 범위가 매우 다양해지고 작업에 필요한 로봇 동작의 종류 또한 매우 여러 가지가 요구되고 있다. 이러한 고기능 로봇의 task를 표현하기 위해서는 기존 제공되는 로봇 동작제어 명령만을 이용하여 로봇의 작업을 지시하기에는 많은 어려움이 예상된다. 따라서 본 연구에서는 전체 job을 수행하는데 필요한 단위 task를 표현하는 제어언어로써 sensor based robot programming이 용이하도록 설계된 Formal Model Language를 개발하였다.

Formal Model Language란 real-time & multi-tasking O.S상의 독립된 processor로 동작하는 여러 개의 module을 이용하여 로봇의 task command를 기술하는 것을 말한다. 여기에 사용되는 module들은 각각 자신만의 고유한 기능을 담당하며 그림 2와 같이 input port와 output port 및 argument를 갖는 port automata로 구성된다. input port와 output port는 integer와 double형 port를 10개씩 갖고 있으며 매 sampling time 마다 값을 받아들이고 결과를 내보내는 장소이다. 또한 arguments는 module의 실행에 필요한 초기 값으로 작업 프로그램 시 사용자가 정의하게 될 parameter들을 말한다.

여기서 PM이란 이름의 module을 PM(i1,i2,...)(o1,o2,...)(a1,a2,...)으로 표기하며 첫번째와 두번째 괄호 안의 변수는 각각 input port와 output port의 port name을 의미하고, 세번째 괄호 안의 변수는 module의 동작을 위하여 초기에 정의되어야 하는 arguments를 의미한다. 각 module의 정의 형식은 다음과 같다.

```
Module Name  : Adder(i_input,d_input)(i_output,d_output)(offset,end_con)
Input Ports  : int i_input[10] ,     double d_input[10];
Output Ports : int i_output[10] ,    double d_output[10];
Arguments    : int offset,           end_con;
Preprocessing : int i=0;
Behavior     :
{    for(i=0;i<10;i++)
          o_port[i] = i_port[i] + offset;
     if(o_port[i]>end_con) End_Condition = TRUE;
}
```

위의 정의 형식에서 module의 특성 및 기능은 Preprocessing 정의부와 Behavior 정의부에서 나타나며 Preprocessing부는 module의 Behavior 부분이 동작하는데 필요한 전처리 부분으로써 내부변수정의 및 module이 사용할 resource의 초기화 등이 포함된다. Behavior 부분은 module이 1 sampling time 동안에 수행해야 하는 일을 의미하는 C Language로 표현된 computing behavior 정의 부분이며 End_Condition이 만족될 때까지 수행된다.

Formal Model Language에서 사용하는 module은 그 특성에 따라 sensory, processing, action module로 구분된다. Sensory module은 외부 환경을 인식하여 그 값을 processing module로 전달하는 output port만을 갖는 port automata이고 action module은 processing module로 부터 값을 받아 로봇을 움직여 주는 input port만을 가지는 port automata이다. Processing module은 sensory module과 action module을 연결하여 필요한 동작을 생성하기 위한 data handling 및 algorithm을 포함하며 input port와 output port를 모두 갖는다. 이렇게 정의된 각각의 module은 다른 module들과 서로 연결되어 새로운 module을 정의할 수 있게 된다. 새로운 module을 구성하는 module들은

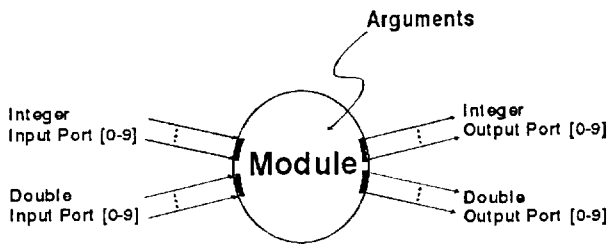
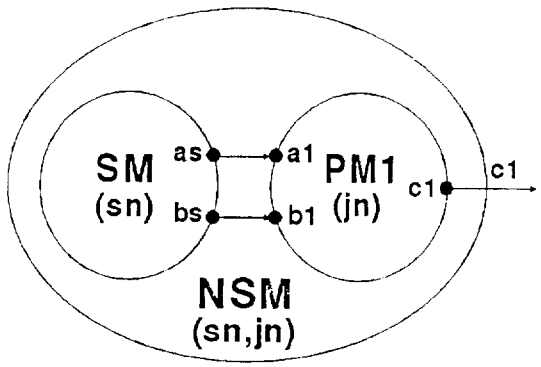
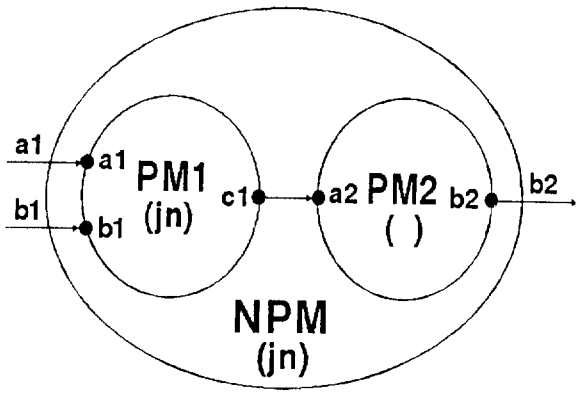


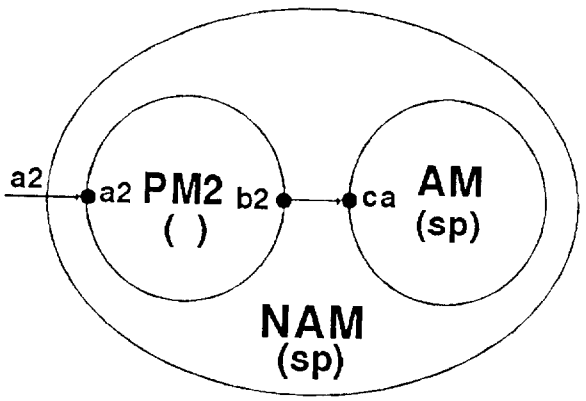
그림 2 Module의 기본 구조
Fig. 2 Basic module structure



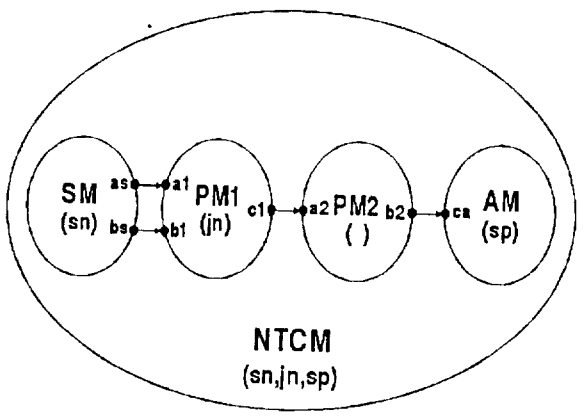
(a) sensory module 생성



(b) processing module 생성



(c) action module 생성



(d) task command module 생성

(a) NSM( )(c1)(sn,jn) = [SM( )(as,bs)(sn) || PM1(a1,b1)(c1)(jn) ||]
(b) NPM(a1,b1)(b2)(jn) = [|| PM1(a1,b1)(c1)(jn) ; PM2(a2)(b2)( ) ||]
(c) NAM(a2)( )(sp) = [|| PM2(a2)(b2)( ) || AM(ca)( )(sp)]
(d) NTCM( )( )(sn,jn,sp) = [SM( )(as,bs)(sn) || PM1(a1,b1)(c1)(jn) ; PM2(a2)(b2)( ) || AM(ca)( )(sp)]

그림 3 Module의 4가지 생성규칙 예
Fig. 3 An example for 4 types of rule production

다음과 같이 "[ ]"로 묶여지게 되며 "||"에 의해 sensory, processing 및 action module의 영역이 구분된다.

New Module( )( )( ) = [Sensory Module( )( )( ) || Processing Module( )( )( ) || Action Module( )( )( )]

여기서 Sensory Module의 output port는 Processing Module의 input port로 연결되며 Processing Module의 output port는 Action module의 input port로 연결되고 New Module의 arguments는 각 module로 나누어 연결된다. 또한, 각 module의 영역에 두개 이상의 module이 포함될 경우 이들의 관계를 ";"와 ":"로 표시하며 각각 sequential 및 parallel 수행을 의미한다.

여러 개의 module을 연결하여 새로운 module을 정의하는데는 그림 3과 같은 4가지 종류의 생성규칙이 있으며, 여기서 SM은 sensory module이고 PM1과 PM2는 processing module이며 AM은 action module 이다.

2.2 Centered Grasp Motion에 대한 Formal Model Language Example

하나의 예로써 두개의 finger가 접촉이 없으면 finger를 close하고 하나의 finger가 object에 접촉하면 wrist를 접촉 방향으로 일정량만큼 움직여 주는 작업을 반복하다가 두개의 finger가 모두 접촉하면 작업을 끝내는 그림 4(a)와 같은 centered grasp motion[4] task level command, CENTERED_GRASP에 대한 Formal Model 구조는 그림 4(b)와 같으며 다음과 같이 정의된다.

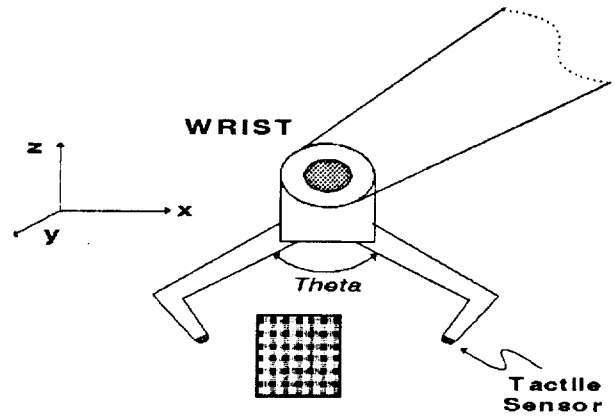
CENTERED_GRASP( )( )(theta_gain, xyz_gain)
= [TACT( )(left_tact,right_tact)( ) ||

CENTER(left_tact,right_tact)(d_theta,dx,dy,dz)(theta_gain, xyz_gain) ||
MOVE_HAND(d_theta,dx,dy,dz)( )( )]

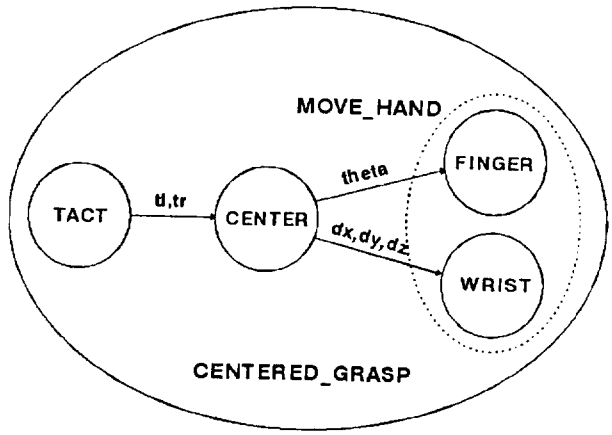
MOVE_HAND(d_theta,dx,dy,dz)( )( )
= [|| || FINGER(d_theta)( )( ) : WRIST(dx,dy,dz)( )( )]

여기서 sensory module TACT( )(left_tact, right_tact)( )는 finger에 부착된 왼쪽과 오른쪽의 tactile sensor값을 읽어 output port로 0 또는 1을 return하는 sensor처리 module을 말한다. 또한 action module로는 FINGER와 WRIST를 사용하는데 FINGER(d_theta)( )( )는 양쪽 finger의 각도가 d_theta가 되도록 움직여 주는 module이며 WRIST(dx,dy,dz)( )( )는 input port의 dx,dy,dz양만큼 arm의 wrist를 움직여 주는 module을 말한다. Processing module인 CENTER(left_tact, right_tact)(d_theta, dx, dy, dz)(theta_gain, xyz_gain)은 input module로부터 값을 받아 centered grasp task를 수행하기 위한 action module의 입력 값을 계산해 주는 기능을 갖으며 다음과 같이 정의된다.

```
Module Name : CENTER(i_input,d_input)(i_output,d_output)
                (theta_gain,xyz_gain)
Input Ports   : int i_input[10] ,          double d_input[10];
Output Ports  : int i_output[10] ,         double d_output[10];
Arguments     : double theta_gain;
                double xyz_gain;
Preprocessing : int    left_tact , right_tact;
                double d_theta, d_x, d_y, d_z;
Behavior      :
{
        left_tact  = i_input[1];
        right_tact = i_input[2];
                if ( left_tact==0 && right_tact==0 )
                        d_theta = -3. * theta_gain;
        else {
                d_theta=0;
                if( left_tact==0 &&  right_tact==1 )
                        d_x = 1. * xyz_gain;
                if( left_tact==1 &&  right_tact==0 )
                        d_x =-1. * xyz_gain;
        }
                if ( left_tact==1 && right_tact==1 )
                        End_Condition = TRUE;
        d_output[1] = d_theta;
        d_output[2] = d_x;
        d_output[3] = d_output[4] = .0;
}
End
```



(a) Centered Grasp Task Example



(b) Formal Model Structure

그림 4 Centered Grasp Motion에 대한 Formal Model Example
Fig. 4 An formal model example of centered grasp example

2.3 Formal Model Language Controller의 구성 및 구현

본 연구에서는 Formal Model Language로 표현된 task를 수행하기 위하여 그림 4와 같이 각 module들을 O.S상의 process로 생성시키고 port의 할당 및 이들간의 process switching을 담당하는 Formal Model Controller와 각 module이 실제 수행되는 장소인 DSPU를 구현하였다. Formal Model Language구현의 관점에서 module은 combination module과 single module로 구분된다. Single module은 앞에서 설명한 바와 같이 자신의 실행 code를 갖고 있으며 해당 DSPU에서 동작한다. 반면, combination module은 생성규칙에 의해 여러 개의 module로 묶여진 module을 말하며 실행코드는 갖지 않는다. 따라서, PGRL의 place가 의미하는 task는 nested된 여러 개의 combination module 및 single module로 구성되며, 본 논문에서는 이러한 task를 Formal Model로 정의된 구조에 따라 구동시키기 위하여 계층적(hierarchical)이고 재귀적(recursive)인 구조의 Formal Model Controller를 개발하였다.

(1) Single Module의 실행구조

먼저 single module은 DSPU에서 실행되며 Formal Model Controller로 부터 자신이 사용할 port의 address와 module을 control하는데 사용할 semaphore를 지정받아, real time multi-tasking O.S인 VxWorks상의 독립된 process로 동작한다. 이를 위하여 앞절에서 기술한 형식으로 정의된 module은 다음과 같이 상용 C-compiler로 컴파일 가능한 다음과 같은 code로 변환된다.

```
MOD_NAME(place_id,ps_sem,csem,i_input,d_input,i_output,d_output)
int     place_id;  /*data base에서 argument를 참조할 장소*/
SEM_ID ps_sem;     /*process switching에 사용할 semaphore의
id*/
SEM_ID csem;       /*전처리부 완료신호에 사용할 semaphore의 id*/
int     i_input[10];  /* integer input port address      */
double  d_input[10];  /* double input port address       */
int     i_output[10]; /* integer output port address     */
double  d_output[10]; /* double output port address      */
{
        내부변수정의 및
        PGRL data base에서 Argument 참조
        Module의 전처리부 ( Preprocessing )

semGive(csem);   /*module의 수행준비가 끝났음을 알리는 신호*/
while(End_Condition==FAL){
        semTake(ps_sem,WAIT_FOREVER);
        taskUnlock();

            1 Sampling Time 동안에 수행해야 할
            Module의  Behavior 정의부

        taskLock();
        semGive(ps_sem);
    }
}
```

여기서 각 module의 전처리부와 Behavior정의부는 앞절에서 정의한 것과 같으며 나머지 부분은 module을 control하기 위한 공통된 형식을 따른다. 하나의 task 수행명령이 Formal Model Controller에 전달되면 task를 구성하는 모든 module이 O.S상의 process로 등록되며 각 module은 process로 생성되자 마자 전처리부를 수행하고 나서 module의 수행준비가 끝났음을 알리는 semaphore(csem)를 Formal Model Controller로 보내게 된다. Formal Model Controller는 task를 구성하는 모든 single module로 부터 semaphore를 받게되면 Formal Model로 정의된 module의 연결 구조에 맞도록 각 module에 대한 process switching을 시작함으로써 task를 수행하게 된다. Process switching은 semaphore를 이용하여 구현하였으며 각 module은 자신을 생성시킨 상위 process로 부터 semaphore(ps_sem)을 전달받으면 Behavior부를 수행하고 나서 다시 semaphore를 반납하게 된다.

(2) Combination Module의 실행구조

Combination module은 실행코드를 갖지 않으며 여러개 module의 조합으로 연결된 구조를 갖는다. 따라서 combination module은 자신을 구성하는 module(이후 편의상 sub module이라 지칭)을 process로 생성시키고 이들의 process switching을 제어하는 기능을 수행해야 한다.

본 논문에서는 이러한 combination module의 관리기능을 Formal Model Controller에서 실행되는 CMCM(Combination Module Control Module)이 담당하도록 하였으며 다른 module과 마찬가지로 전처리부와 Behavior부를 갖는다. CMCM의 전처리부에서는 해당 combination module을 구성하는 모든 sub module이 사용할 input/output port의 address를 할당하여 O.S상의 process로 생성시키는 기능을 담당하며 Behavior부분에서는 생성된 sub module을 제어하기 위한 semaphore operation을 담당한다. 이러한 CMCM의 전처리부에 대한 pseudo code는 다음과 같다.

```
<Memory에 sub module들간에 I/O port로 사용할 내부연결
 port를 sub module의 갯수만큼 할당한다. >
FOR (첫번째 sub module) TO (마지막 sub module)
        IF (sub module중 첫번째 module) THEN
            input port로 CMCM의 input port를 사용
            output port로 첫번째 내부연결 port를 사용
        IF (sub module중 마지막 module) THEN
            output port로 CMCM의 output port를 사용
        IF (sub module이 single module) THEN
            지정된 input port와 output port를 이용하여
            해당 sub module을 DSPU에 process로 생성
        IF (sub module이 combination module) THEN
            지정된 input port와 output port를 이용하여
            해당 combination module을 담당하는 CMCM을 생성
        IF (이전에 생성한 sub module과의 관계가 sequence
            ) THEN
            input port로 이전 module의 output port를 사용
            output port로 새로운 내부연결 port를 사용
        IF (이전에 생성한 sub module과의 관계가 parallel)
            THEN
```

nput port로 이전 module과 동일한 output port를 사용

output port로 이전 module과 동일한 output port를 사용

NEXT

예를 들어, 그림 4(b)의 combination module인 CENTERED_GRASP을 담당하는 CMCM의 전처리부에서는 'submodule을 process로 생성시키게 되는데 submodule중 single module인 TACT와 CENTER는 직접 DSPU의 process로 생성시키며 combination module인 MOVE_HAND는 이를 담당하는 또 하나의 CMCM으로 생성하게 된다. 따라서, CENTERED_GRASP task를 수행하기 위한 module의 구조는 그림 5와 같으며 Formal Model Controller는 가장 상위 module인 CMCM#1만을 생성하고 제어함으로써 하부의 module구조는 계층적으로 제어되도록 설계하였다.

본 연구에서는 이러한 Formal Model Controller를 MC68030 CPU board와 VxWorks를 이용하여 구현하였고, performance test를 위하여 33개의 single module과 14개의 combination module로 구성된 task를 정의하여 실험을 하였으며, 실험을 통하여 총 47개의 module을 control하는데 걸리는 시간, 즉 Behavior 부분 수행시간을 제외한 Process switching 시의 time delay가 23msec로 측정되었다. 따라서, 1개의 module에서 소요되는 예상 time delay는 약 0.5msec로 예상된다.

2.4 실험 및 결과

2.4.1 Formal Model Language를 이용한 Visual Servoing

본 연구에서는 camera를 통한 시각 feedback 정보를 이용하여 SCARA 로봇이 2차원 상에서 움직이는 물체를 추적하는 그림 6과 같은 visual servoing 작업을 Formal Model Language를 이용하여 정의하고 이에 대한 실험을 수행하였다. Visual servoing motion을 위한 Formal Model에 사용되는 module은 다음과 같다.

FEATURES( )(df_x,df_y,df_z)( )

Sensory module로써 camera를 통해 object 특징량의 변화를 추출하여 output port인 df_x,df_y,df_z로 출력하는 기능을 갖는다.

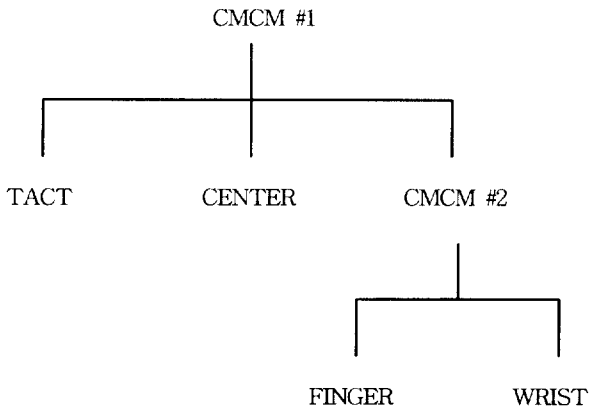


그림 5 centered grasp task에 대한 module의 구조
Fig. 5 Module structure for centered grasp task

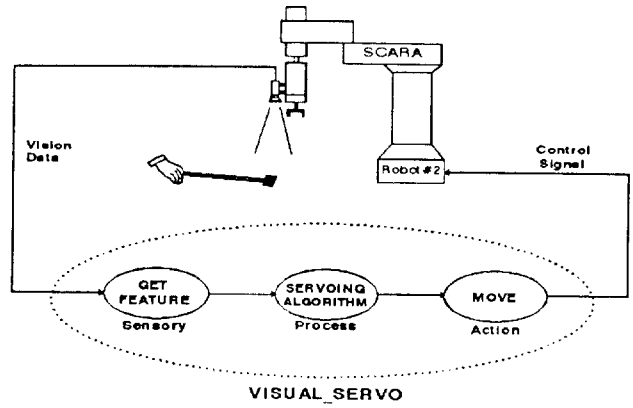


그림 6 Visual Servoing Motion의 실험 환경
Fig. 6 Experimental environment of visual servoing motion

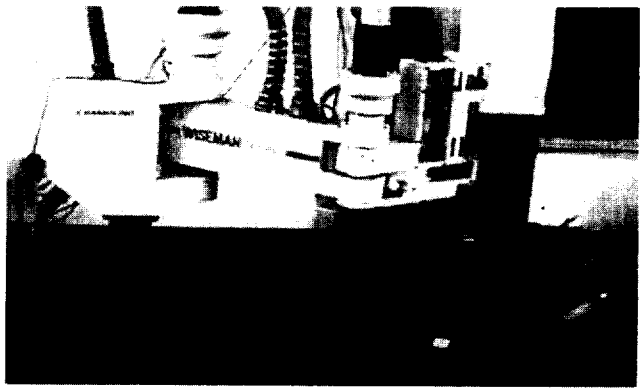


그림 7 Visual Servoing Motion의 실험 사진
Fig. 7 Experiment photograph of visual servoing motion

이러한 세개의 module을 이용한 Formal Model 정의는 다음과 같으며 이에 대한 실험 사진은 그림 7과 같다.

SERVO_ALGORITHM(df_x,df_y,df_z)(dx,dy,dz)(Time_limit, gain)

Processing module로써 input port로 부터 받은 특징량 변화에 따라 이를 추적하기 위해 로봇이움직여야 할 양을 계산하여 output port인 dx,dy,dz로 출력하는 기능을 갖는다.

MOVE1(dx,dy,dz)( )( )

Action module로써 input port로 부터 받은 직교좌표계 상에서 로봇이 움직여야 할 양을 inverse kinematics과정을 통하여 실제 로봇의 각 joint가 움직여야 할 변위를 계산하여 motor를 구동시키는 기능을 갖는다.

VISUAL_SERVO( )( )(Time_limit, gain) =
[FEATURES( )(df_x,df_y,df_z)( ) ||
SERVO_ALGORITHM(df_x,df_y,df_z)(dx,dy,dz)(Time_limit, gain) ||
MOVE1(dx,dy,dz)( )( )]

2.4.2 Formal Model Language를 이용한 Surface Tracking

손목에 부착된 force/torque sensor를 통한 힘정보를 이용하

여 PT200 로봇이 임의의 곡면을 따라 이동하는 그림 8.과 같은 surface tracking 작업을 Formal Model Language를 이용하여 정의하고 이에 대한 실험을 수행하였다. Surface tracking motion을 위한 Formal Model에 사용되는 module은 다음과 같다.

FORCE( )(fx,fy,fz,tx,ty,tz)( )
Sensory module로써 F/T sensor를 통해 로봇의 end effector에 전달되는 힘의 양을 추출하여 output port인 fx,fy,fz,tx,ty,tz로 출력하는 기능을 갖는다.

TRACK_ALG(fx,fy,fz,tx,ty,tz)($d\theta_1$,$d\theta_2$,$d\theta_3$,$d\theta_4$,$d\theta_5$,$d\theta_6$)(f_dir, f_N)
Processing module로써 input port로 부터 받은 힘의 변화량을 참조하여 곡면을 따라가기 위해 로봇의 joint가 움직여야 할 양을 계산하여 output port인 $d\theta_1$,$d\theta_2$, $d\theta_3$,$d\theta_4$,$d\theta_5$,$d\theta_6$로 출력하는 기능을 갖는다.

MOVE2($d\theta_1$,$d\theta_2$,$d\theta_3$,$d\theta_4$,$d\theta_5$,$d\theta_6$)( )(move_dir, move_dist)
Action module로써 input port로 부터 전달받은 실제 로봇의 각 joint가 움직여야 할 변이만큼 motor를 구동시키는 기능을 갖는다.

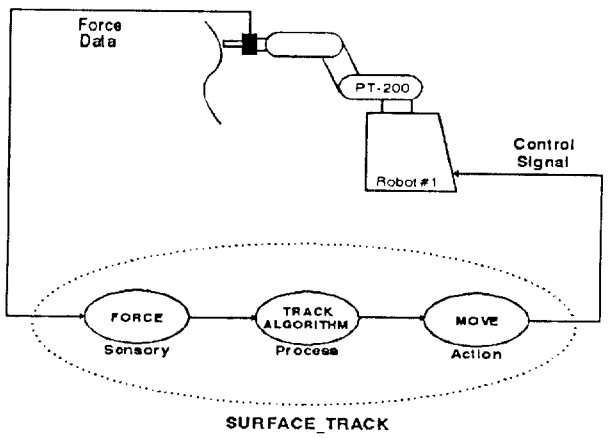


그림 8 Surface Tracking Motion의 실험 환경
Fig. 8 Experimental setup for surface tracking motion

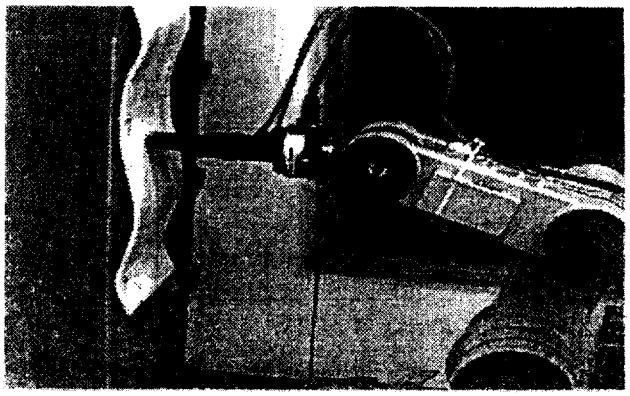


그림 9 Surface Tracking Motion의 실험 사진
Fig. 9 Experiment Photograph of surface tracking motion

이러한 세개의 module을 이용한 Formal Model 정의는 다음과 같으며 이에 대한 실험 사진은 그림 9.와 같다.

SURFACE_TRACK( )( )(f_dir, f_N, move_dir, move_dist) =
[FORCE( )(fx,fy,fz,tx,ty,tz)( ) ||
TRACK_ALG(fx,fy,fz,tx,ty,tz)($d\theta_1$,$d\theta_2$,$d\theta_3$,$d\theta_4$,$d\theta_5$,$d\theta_6$)(f_dir, f_N) ||
MOVE2($d\theta_1$,$d\theta_2$,$d\theta_3$,$d\theta_4$,$d\theta_5$,$d\theta_6$)( )(move_dir, move_dist)
]

3. Petrinet-type Graphical Robot Language (PGRL)의 개발

3.1 Petri Net Type Graphical Robot Language(PGRL)

Petri net을 이용하여 시스템을 표현하게 되면 사건(Event)과 조건(Condition)의 두가지 기본적인 개념으로 축약된다. 사건은 시스템에서 일어나는 action을 뜻하며 사건의 발생(occurrence)은 시스템의 상태에 따라 결정된다. 또한 시스템의 상태는 조건으로 표현되며 참 또는 거짓의 값을 가진다. 따라서 시스템의 조건을 Petri net의 place로 표현하고 사건을 transition으로 표현하여 시스템의 특성을 쉽게 모델링할 수 있다[11]. 본 연구에서는 Petri net을 이용하여 다중로봇의 작업을 프로그램하도록 하였다. 각 로봇이 전체 작업을 완수하기 위해 필요한 Formal Model로 정의된 단위 task를 PGRL의 각 place에 할당하며, 각 place는 해당 로봇이 task를 수행 중인 상태를 나타내고 task가 완료되면 place의 조건이 만족된다. Transition은 출력 place가 의미하는 task의 수행 시작을 나타내는 사건을 말하며, 사건이 발생하기 위해서는 해당 transition의 선행조건이 만족되어야 한다. 즉, 입력 place가 의미하는 task의 수행이 모두 완료된 상태이어야 transition이 enable상태가 된다.

이러한 PGRL을 이용하여 사용자가 다중로봇의 작업을 프로그램할 수 있도록 본 연구에서는 그림 10과 같은 Windows를 이용한 Graphic Editor를 구현하였다. 이러한 Graphic Editor에서는 PGRL에서 사용하는 place, transition 그리고 arc 등을 graphical symbol인 아이콘(icon)으로 제공하며 이들을 이용하여 사용자는 Windows상에서 graphical하게 다중로봇의 작업을 프로그램할 수 있다. 특히 Graphic Editor에서 마우스를 이용하

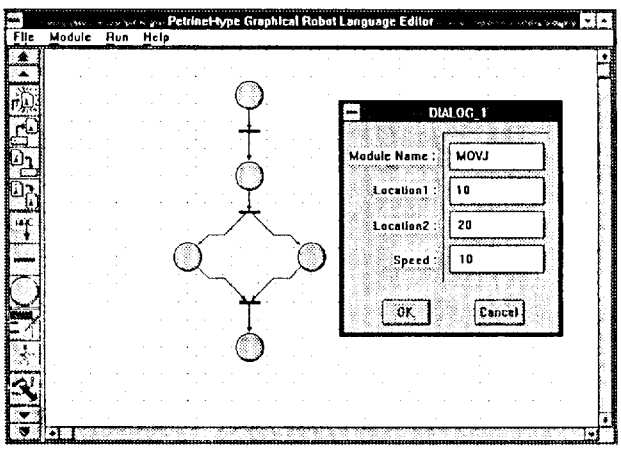


그림 10 PGRL 프로그램을 위한 Graphic Editor 화면
Fig. 10 Graphic editor screen for PGRL program

여 place icon을 double click하면 생성되는 대화상자에서 동작의 종류, 목표점, 동작속도 및 place가 갖는 초기 token의 수 등과 같이 로봇이 해당 task를 수행하는데 필요한 초기 parameter값을 사용자로부터 입력받도록 함으로써 PGRL의 place가 로봇이 수행해야 할 task의 모든 정보를 포함하도록 구성하였다.

3.2 PGRL을 이용한 조립 작업의 Program Example

앞에서 설명한 바와 같이 PGRL의 place는 각각 로봇의 task로써 수행되게 되며 다중로봇 시스템의 작업을 기술하는데 있어서 PGRL의 편리성과 효율성을 보이기 위하여 두대 로봇과 두대의 부품 공급 장치를 이용한 조립 작업을 PGRL을 이용하여 기술하였다. 여기서 기술한 조립 작업은 두대의 로봇 Arm1과 Arm2가 각각 bolt 공급장치와 box 공급장치로 부터 부품을 집어 bolt를 box에 삽입한 후 조립된 제품을 conveyor 위에 옮겨 놓는 작업을 PGRL을 이용하여 기술하였다. 여기서 조립작업시 필요한 선행 조건에는 각 부품 공급 장치가 부품을 공급한 후에 로봇이 부품을 grasp하여야 하는 조건과, 또한 Arm2에 의해 box를 작업대 위에 올려놓은 후에 Arm1이 bolt를 삽입해야 하는 조건이 있다. 이러한 job의 선행 조건들 및 동시 동작이 PGRL의 Fork 및 Join을 이용하여 표현되었다. 여기서 주의할 것은 첫 번째 loop가 수행되기 위해서는 p_4, p_8, p_{11}은 초기에 token을 보유하고 있어야 하며, 또한 start를 나타내는 p_1은 선행 조건이 없는 place 이며 초기 token의 개수를 N개로 지정함으로써 전체 조립 작업을 N번 반복하도록 하였다. 이에 대한 PGRL Program은 그림 11과 같이 나타나며, 각 place의 의미는 표 1과 같다.

표 1 Box 조립 예에서 각 Place의 의미
Table 1 Meaning of each place in a box assembly task

PLACE	PLACE의 의미
P_1	Start Place
P_2	Arm1이 Bolt Feeder로 이동 중인 상태
P3	Arm1이 Bolt를 grasp한 상태
P4	Feeder에 Bolt가 도착한 상태
P5	Arm1이 Bolt를 Box에 삽입하는 동작
P6	Arm2가 Box Feeder로 이동 중인 상태
P7	Arm2가 Box를 grasp한 상태
P8	Feeder에 Box가 도착한 상태
P9	Arm2가 Box를 잡고 작업대로 옮기는 동작
P10	Arm2가 조립된 Box를 Conveyor로 옮기는 동작
P11	Arm2가 조립된 Box를 release한 상태

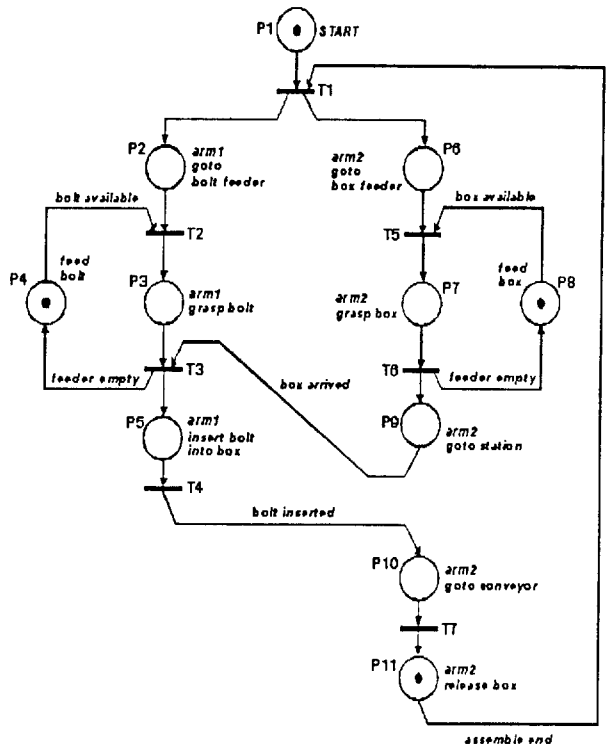


그림 11 PGRL을 이용한 Box 조립 작업 Program 예제
Fig. 11 An example program of box assembly task using PGRL

3.3 PGRL Program의 Compile과 Data File의 생성

PGRL에서 place의 집합을 $P=\{p_1,p_2,...,p_n\}$, transition의 집합을 $T=\{t_1,t_2,...,t_m\}$로 표현하고 place와 transition사이의 arc를 (p_i,t_j) 또는 (t_j,p_i)로 표현할 수 있다. PGRL program을 기술하는 더욱 편리한 표현으로 transition t_j에 대한 입력 place의 집합을 $I(t_j)$로 표현하고 마찬가지로 t_j에 대한 출력 place의 집합을 $O(t_j)$로 다음과 같이 표현할 수 있으며 A는 arc의 집합을 나타낸다.

$I(t_j) = \{ p_i : (p_i,t_j) \in A \}$, $O(t_j) = \{ p_i : (t_j,p_i) \in A \}$

Place p_i에 대해서도 역시 $I(p_i)$ 와 $O(p_i)$의 표현이 적용되며 place p_i로 부터 transition t_j로 향한 arc는 $p_i \in I(t_j)$ 및 $t_j \in O(p_i)$를 의미한다.

Graphic Editor에서 PGRL program이 작성되면 Graphic Compiler는 arc에 의한 연결 관계를 인식하여 m개의 place와 n개의 transition간의 m×n 관계정보행렬, TRM(Task Relation Matrix)을 생성한다. TRM에서 i번째 행은 p_i에 대한 transition과의 관계를 나타내고 j번째 열은 t_j에 대한 place의 관계를 나타낸다. 여기서 1은 $(p_i,t_j) \in A$ 즉 p_i에서 t_j로 arc가 있는 것을 의미하고, -1은 $(t_j,p_i) \in A$ 즉, t_j에서 p_i로 arc가 있는 것을 의미하며, 0은 p_i와 t_j사이에 arc가 없는 것을 의미한다. 따라서 다음과 같이 place 또는 transition의 input과 output을 TRM에서 참조할 수 있다. 3.2절의 PGRL 프로그램에 대한 TRM은 표2와 같다.

$I(p_i)$: TRM의 i번째 행에서 1로 표시된 transition의 집합
$O(p_i)$: TRM의 i번째 행에서 -1로 표시된 transition의 집합

$I(t_j)$: TRM의 j번째 열에서 -1로 표시된 place의 집합
$O(t_j)$: TRM의 j번째 열에서 1로 표시된 place의 집합

PGRL program의 정보에는 앞에서 설명한 task간의 관계 정보뿐만 아니라 task수행에 필요한 정보도 포함된다. 이러한 정보에는 수행해야 할 task의 이름과 task를 수행하는데 필요한 parameter 값들이 있으며 이에 대한 데이터의 구조는 다음과 같다.

표 2 그림 11의 PGRL Program에 대한 Task Relation Map

Table 2. Task relation map for PGRL program in Fig.11

Place \ Transition	t_1	t_2	t_3	t_4	t_5	t_6	t_7
p_1	-1	0	0	0	0	0	0
p_2	1	-1	0	0	0	0	0
p_3	0	1	-1	0	0	0	0
p_4	0	-1	1	0	0	0	0
p_5	0	0	1	-1	0	0	0
p_6	1	0	0	0	-1	0	0
p_7	0	0	0	0	1	-1	0
p_8	0	0	0	0	-1	1	0
p_9	0	0	-1	0	0	1	0
p_{10}	0	0	0	1	0	0	-1
p_{11}	-1	0	0	0	0	0	1

```
struct TASK {
short int   task_id ;    /* Task id                    */
short int   i_n[5] ;     /* input node id              */
short int   o_n[5] ;     /* output node id             */
short int   n_token ;    /* number of initial token    */
char   task_name[10] ;   /* Task Command Name          */
int    ivar[5] ;         /* integer variable list      */
double dvar[5] ;         /* double variable list       */
char   cvar[5] ;         /* character variable list    */
}
```

3.4 PGRL의 Language Controller의 구조 및 구현

본 연구에서 제안한 다중로봇 제어언어의 실행 구조는 제어언어 자체의 구조와 같이 두 가지로 구성된다. 각각의 task를 이용하여 구성한 다중로봇의 전체 작업을 표현한 PGRL 프로그램, 즉 작업의 흐름을 제어하는 부분과 Formal Model Language로 표현된 각 task를 제어하는 부분이 있다. 이들은 서론의 그림 1에서와 같이 각각 TCU(Task Coordination Unit) 내의 PGRL Controller와 Formal Model Controller에서 담당한다. Graphic Editor에서 사용자가 작성한 프로그램은 compile되어 data file로 전달되며 PGRL Controller와 Formal Model Controller는 이를 참조하여 CENTAUR의 job program을 실행하게 된다. PGRL Controller는 TRM을 참조하여 현재 수행하여야 할 task를 결정해 Formal Model Controller로 해당 task의 실행 명령을 전달한다. Formal Model Controller에서는 PGRL Controller로 부터 실행 명령을 받은 task를 수행하고 수행이 끝난 후 task완료 신호를 다시 PGRL Controller로 보낸 뒤 다음 명령을 기다리며, Task완료 신호를 받은 PGRL Controller는 각 place의 현재 상태를 판단하여 다음에 수행해야 할 task를 결정하게 된다. 이러한 PGRL Controller에서 다음에 수행해야 할 task를 결정하는 algorithm은 다음과 같다. 여기서 trm[m][n]은 m개의 place와 n개의 transition에 대한 m×n 관계정보행렬, TRM을 저장하는 배열을 말한다.

```
< 가정 : i번째 place pi의 task완료신호를 수신 >
FOR x=1 TO n
    IF (trm[i][x] = -1) THEN
    /* n개의 모든 transition에 대해 pi를 입력으로 갖는지
    검사 */
        IF (x번째 transition tx = enable) THEN
        /* pi를 입력으로 갖는 transition, tx가 enable한지
        검사 */
            FOR y=1 TO m
                IF (trm[y][x]=1) THEN
                /* enable한 transition, tx의 출력
                place를 검사*/
                    y번째 place, py의 task 수
                    행명령을
                     Formal Model Controller
                     로 전송
            NEXT y
NEXT x
```

3.5 PGRL Program의 해석

3.5.1 자원공유(Resource Allocation) 문제의 해결

Petri net을 이용하여 작업을 표현 또는 기술하는데 있어 고려해야할 사항으로써 Resource Allocation문제가 있다. 이러한 문제는 공유할 수 없는 동일한 자원(Resource)를 사용하는 두 개이상의 Task가 동시에 수행될때 발생되며, 이때 서로 사용하고자 하는 자원을 어떤 Task에게 할애할 것인가를 결정하는 문제를 말한다. 예를들어 그림 12(a)와 같은 PGRL Program에서 Task A와 Task B가 동일한 Vision Camara를 사용하는 경우 이 두개의 Task가 동시에 수행될때 충돌(Conflict)이 발생하게 된다. 본 연구에서는 이러한 문제를 해결하기 위하여 Semaphore역할을 담당하는 Place를 이용하여 Conflict가 예상되는 Task간의 Mutual Exclusion 표현을 가능하도록 구성하였다. 즉 그림 12(b)와 같이 Conflict가 예상되는 두 Task를 실행시키는 각각의 Transition이 선행조건으로 Semaphore를 갖도록 하여 먼저 작업을 시작하는 Task가 작업을 마칠때 까지 다른 Task의 수행을 금지시키므로써 Conflict를 예방하였다.

3.5.2 PGRL 프로그램의 해석

본 논문에서는 Petri net의 해석에 있어서 고려되는 여러가지 성질중 로봇작업의 해석에 필요한 Safeness, Liveness, Deadlock의 해석에 대하여 연구하였다. Safeness는 Petri net을 이용하여 Hardware device를 모델링할 경우 고려해야할 성질로써 Place의 Token의 수가 1을 넘지 않을때 Safe 하다고 한다. 본 연구에서 이러한 성질은 로봇이 해당 Place의 작업을 수행중 다시 실행명령이 내려지는 오류를 예측할 수 있다. 또한 Liveness와 Deadlock은 컴퓨터 시스템의 Resource Allocation에 있어서 제기되는 문제점으로 Liveness는 Transition이 Fire될 수 있는지 여부를 검사하는 것이며, Deadlock은 동시에 수행되는 두개 이상의 프로세스가 서로 상대방이 사용하고 있는 Resource를 기다리고 있어 작업이 더이상 진행될 수 없는 교착상태를 말한다.

Petri net을 해석하는데 사용하는 일반적인 방법으로는 특정

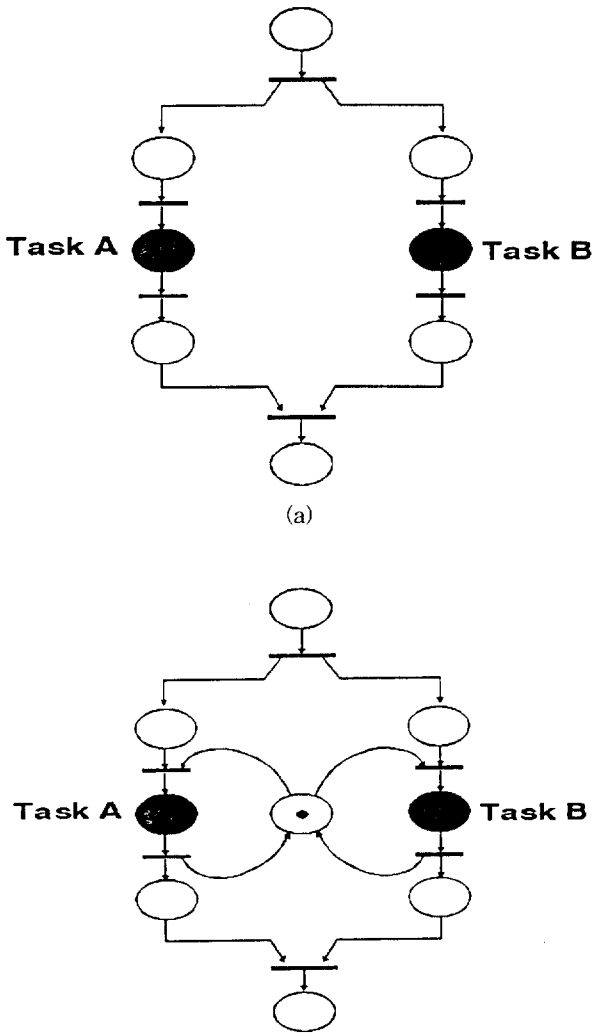


그림 12 동일한 Resource를 공유하는 두개의 Task가 동시에 수행되는 Petri net 예

Fig. 12 An example of petri net which two tasks sharing same resource is accomplished concurrently

Petri net에 대한 Reachability Tree를 이용하는 방법과 Matrix equation을 이용하는 방법이 있다. 먼저 Matrix equation을 사용하는 방법은 Petri net의 Place와 Transition간의 연결관계 정보를 갖는 Incident Matrix를 이용하여 특정 마킹상태(Petri net의 각 Place가 갖는 Token의 수를 나타냄)에 도달하기 위한 각 Transition이 Fire되는 횟수를 계산하는 방법이다. 이 방법은 Petri net이 현재의 마킹상태에서 특정 마킹상태로 도달가능한지 여부를 수식적으로 간단히 알아낼 수 있지만 그 해(Solution)가 각 Transition이 Fire되는 횟수만을 나타낼 뿐 순서의 정보는 갖지 않아 해 자체가 충분조건이 되지 않는다. 또한 이러한 방법으로 Safeness, Liveness, Deadlock등의 성질을 검사하기 위해서는 위의 문제가 발생되는 모든 경우의 마킹상태를 구하고 이에 대한 도달가능 여부를 모두 계산해야 하므로 실제 응용에는 적합하지 않다.

Reachability Tree를 이용하는 방법은 Petri net의 초기 마킹상태로 부터 가능한 모든 마킹상태를 그려나감으로써 Tree를 구성하는 것이다. 이렇게 그려진 Tree에서는 가능한 모든 마킹상태와 이에 이르기 위해 필요한 Transition이 Fire되는 경로가 나타나므로 위에서 언급한 세가지 문제점이 일어나는지 여부를 알 수 있다. 즉 Tree에 나타난 모든 마킹중에 Place가 2개이상의 Token을 갖는 경우가 발생하는지를 검사하여 Safeness를 알 수 있으며, Transition이 Fire되는 경로중 한번도 나타나지 않는 Transition은 Live하지 않음을 알 수 있다. 또한 Tree의 구조가 특정 마킹상태에서 더이상 진행하지 못하게되면 그 부분에서 Deadlock이 발생할 수 있음을 예측할 수 있다. 예를 들어 그림 13과 같은 두개의 프로세스로 구성되고 P3와 P4를 상호 교차되어 공유함으로써 Deadlock이 예상되는 Petri net에서 초기 마킹상태 M(0)는 [1,0,0,1,1,1,0,0]과 같다. 여기서 [ ]안의 요소들은 순서대로 P0, P1, . . . P7이 초기에 갖는 Token의 수를 나타낸다. 이러한 초기상태에서 Enable한 Transition T8과 T11이 Fire되었을 경우의 마킹을 구해나가는 방법으로 Tree를 구성하면 그림 14와 같다.

위와 같은 Tree에서 각 마킹을 연결하는 화살표에 쓰여진 Transition은 화살표가 가리키는 Place에 도달하기 위해 Fire되어야 하는 Transition을 나타낸다. 이와 같이 작성된 Tree에서 M(2) 및 M(5)에서 각각 T10과 T13이 Fire되면 초기 마킹상태

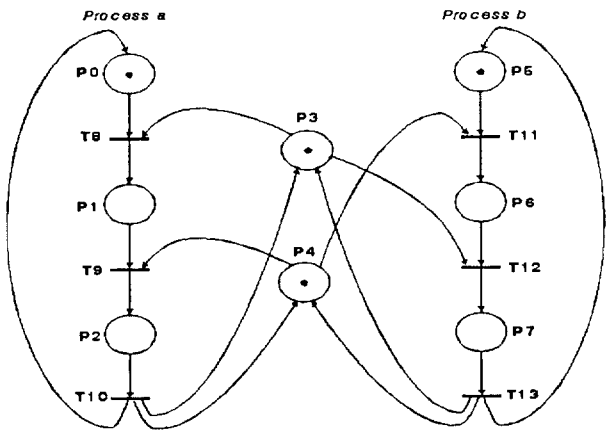


그림 13 동시에 수행되는 두개의 프로세스로 구성된 Petri net

Fig. 13 Petri net composed of two processors executed in concurrently

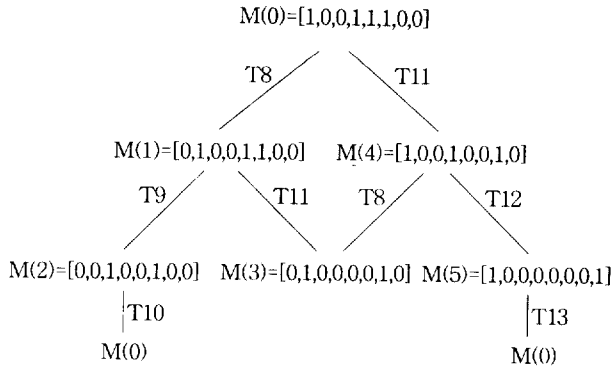


그림 14 그림13의 Petri net에 대한 Reachability Tree
Fig. 14 Reachability tree for petri net in Fig. 13

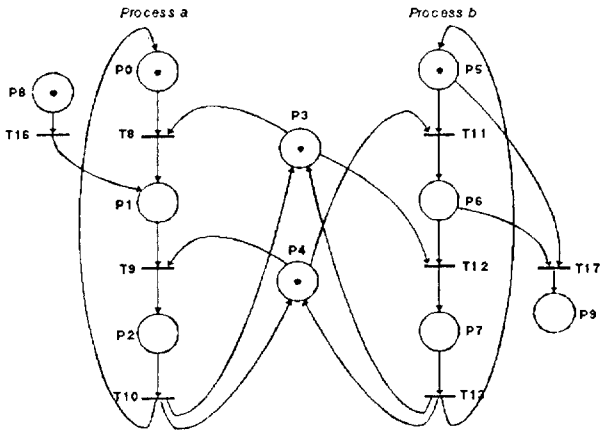


그림 15 Safeness, Liveness, Deadlock의 문제가 예상되는 Petri net
Fig. 15 Petri net expected safeness, liveness, deadlock problem

M(0)로 돌아오지만 M(3)에서는 더이상 Enable한 Transition이 없는 Deadlock이 발생됨을 알 수 있다.

이와같이 Reachability Tree를 구성하는 방법은 주로 주어진 시스템을 모델링한 특정 Petri net에 대해 직관적으로 Tree를 구성하고 이를 바탕으로 Petri net의 Safeness, Liveness, Deadlock등을 판단하는데 매우 유용하게 사용될 수 있다. 본 연구에서는 주어진 Petri net이 아니라 PGRL Graphic Editor에서 로봇 프로그래머에 의해 작성된 어떤 형태의 Petri net에 대해서도 해석이 가능하도록 Reachability Tree의 자동생성기와 이를 바탕으로 Safeness, Liveness, Deadlock을 검사하는 PGRL Analyzer를 개발하였다. Tree의 자동생성 방법은 초기 마킹상태인 Tree의 Root로 부터 시작해서 왼쪽에서 오른쪽의 순서로 각 Node의 Child Node를 재귀적(Recursively)으로 생성해 가는 Depth-First Traversal Algorithm을 이용하여 Tree를 생성하게 되며, 이러한 PGRL Analyzer의 Flowchart는 그림 16과 같다.

PGRL Analyzer의 입력은 Graphic Editor에서 작성된 로봇 프로그램에 대한 TRM(Task Relation Matrix)이 되며, 출력은 생성된 Reachability Tree의 정보와 Safeness, Liveness,

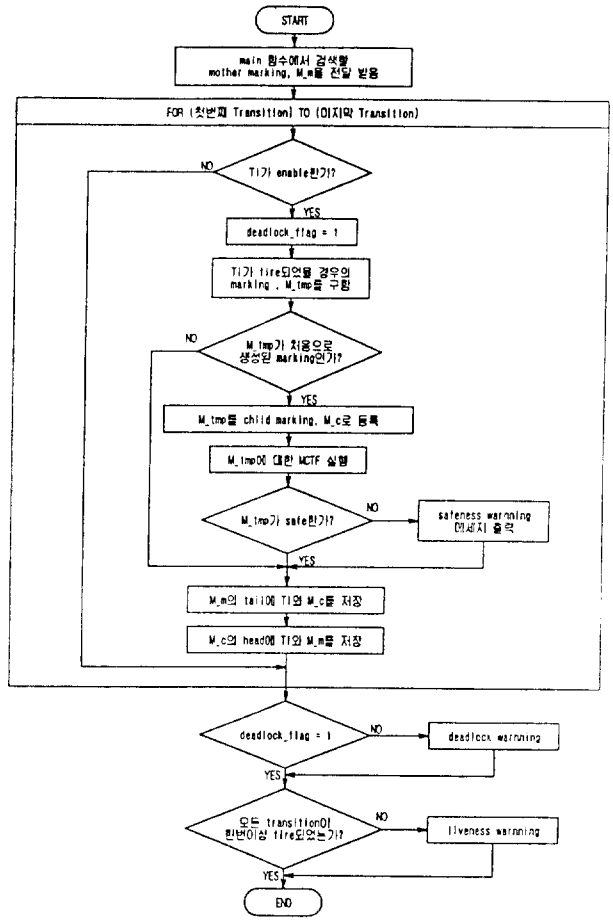


그림 16 PGRL Analyzer의 Tree생성 및 해석 Algorithm
Fig. 16 PGRL analyzer for tree generation and analysis algorithm

Deadlock의 검사결과 이다. Tree의 정보는 Tree의 Node, 즉 각 마킹상태와 이에 대한 Head와 Tail로 구성된다. Head와 Tail은 각 마킹상태에서 각각 위쪽과 아래쪽으로 연결된 마킹의 번호와 Transition의 번호를 표현 함으로써 전체적인 Tree의 구조를 나타내는 정보는 Doubly Linked List 형태를 갖도록 하였다. 이에 대한 한 예로써 그림 13의 Petri net에 대해 생성된 Tree의 정보는 다음과 같다.

M(0)=[1,0,0,1,1,1,0,0,]	tail={(1,8)(4,11)}	head={(2,10)(5,13)}
M(1)=[0,1,0,0,1,1,0,0,]	tail={(2,9)(3,11)}	head={(0,8)}
M(2)=[0,0,1,0,0,1,0,0,]	tail={(0,10)}	head={(1,9)}
M(3)=[0,1,0,0,0,0,1,0,]	tail={}	head={(1,11)(4,8)}
M(4)=[1,0,0,1,0,0,1,0,]	tail={(3,8)(5,12)}	head={(0,11)}
M(5)=[1,0,0,0,0,0,0,1,]	tail={(0,13)}	head={(4,12)}

본 연구에서 개발한 PGRL Analyzer의 검증을 위해 Deadlock이 발생되는 그림 13의 Petri net에 Place, P8과 P9 그리고 Transition, T16과 T17을 추가함으로써 Safeness 및 Liveness의 문제가 예상되는 그림 15와 같은 Petri net에 대한 해석을 수행하였다. 이에 대한 검사 결과로써 생성된 Reachability Tree의 정보와 Safeness, Liveness, Deadlock의 검사결과는 다음과

같다. 여기서 5, 6, 8, 9, 12번째의 마킹에서 Token의 수가 1이 넘는 Place가 발생되어 Safeness 문제가 일어나고, 6번째 마킹에서는 Deadlock이 발생되며, 17번째 Transition, T17은 한번도 Fire되지 않아 Live하지 않음을 알 수 있다.

```
************************************************
        <<< Data of Reachability Tree >>>
************************************************

M( 0)=[1,0,0,1,1,1,0,0,1,0,]    tail={(1,10)(14,13)(4,16)}    head={(2,12)(15,15)}
M( 1)=[0,1,0,0,1,1,0,0,1,0,]    tail={(2,11)(13,13)(5,16)}    head={(0,10)}
M( 2)=[0,0,1,0,0,1,0,0,1,0,]    tail={(0,12)(3,16)}           head={(1,11)}
M( 3)=[0,1,1,0,0,1,0,0,0,0,]    tail={(4,12)}                 head={(2,16)(5,11)(7,10)}
M( 4)=[1,1,0,1,1,1,0,0,0,0,]    tail={(5,10)(7,11)(10,13)}    head={(3,12)(8,10)(11,15)(0,16)}
M( 5)=[0,2,0,0,1,1,0,0,0,0,]    tail={(3,11)(6,13)}           head={(4,10)(1,16)}
M( 6)=[0,2,0,0,0,0,1,0,0,0,]    tail={}                       head={(5,13)(10,10)(13,16)}
M( 7)=[1,0,1,1,0,1,0,0,0,0,]    tail={(3,10)(8,12)}           head={(4,11)}
M( 8)=[2,0,0,2,1,1,0,0,0,0,]    tail={(4,10)(9,13)}           head={(7,12)(12,15)}
M( 9)=[2,0,0,2,0,0,1,0,0,0,]    tail={(10,10)(12,14)}         head={(8,13)}
M(10)=[1,1,0,1,0,0,1,0,0,0,]    tail={(6,10)(11,14)}          head={(9,10)(4,13)(14,16)}
M(11)=[1,1,0,0,0,0,0,1,0,0,]    tail={(4,15)}                 head={(10,14)(12,10)(15,16)}
M(12)=[2,0,0,1,0,0,0,1,0,0,]    tail={(11,10)(8,15)}          head={(9,14)}
M(13)=[0,1,0,0,0,0,1,0,1,0,]    tail={(6,16)}                 head={(1,13)(14,10)}
M(14)=[1,0,0,1,0,0,1,0,1,0,]    tail={(13,10)(15,14)(10,16)}  head={(0,13)}
M(15)=[1,0,0,0,0,0,0,1,1,0,]    tail={(0,15)(11,16)}          head={(14,14)}

************* Safeness Check ***************
Warning  :  Marking[5]={0,2,0,0,1,1,0,0,0,0,} is not Safe !!!
Warning  :  Marking[6]={0,2,0,0,0,0,1,0,0,0,} is not Safe !!!
Warning  :  Marking[8]={2,0,0,2,1,1,0,0,0,0,} is not Safe !!!
Warning  :  Marking[9]={2,0,0,2,0,0,1,0,0,0,} is not Safe !!!
Warning  :  Marking[12]={2,0,0,1,0,0,0,1,0,0,} is not Safe !!!

************* Deadlock Check ***************
Warning  :  Marking[6]={0,2,0,0,0,0,1,0,0,0,} is Deadlock !!!

************* Liveness Check ***************
Warning  :  Transition[17] can not be enable !!!
```

4. 결 론

본 연구에서 개발한 PGCU는 Petri net을 이용하여 로봇의 작업을 프로그램하는 로봇 제어언어에 있어서 기존의 방법과는 전혀 다른 새로운 시도이며, 그 유용성은 서론에서도 언급하였다시피 일반적인 다중 로봇 작업의 표현에 있어서 기존의 text-based language보다 좀 더 표현하기 쉽고 이해하기 쉬운 장점이 있다. 또한, PGRL의 각 place가 의미하는 로봇의 단위 동작 task는 sensor-based robot motion의 개발 및 표현이 용이하도록 설계된 Formal Model에 의해 표현되고, Formal Model의 각 module은 real-time O.S의 독립된 process로 구현하였다. 또한 본 연구에서는 Petri net의 해석기법을 도입하여 설계한 PGRL Analyzer를 개발함으로써 PGRL로 표현된 다중 로봇 작업에서 일어날 수 있는 논리오류를 사전에 검색하여 방지할 수 있게 되었다.

본 연구에서 제시한 다중로봇을 위한 제어시스템에서 PGCU, TCU 및 DSPU로 구성되는 task 수행 및 계획 설정에 있어서의 체계적인 방법론과 정형화된 체제는 고 정밀도의 실행과 상황 판단을 요구하는 자동화 시스템의 구성에 있어서 참고할 만한 체제로 활용될 수 있을 것이다.

참 고 문 헌

[1] J. W. Roach and M.N.Boaz, "Coordinating the Motion of Robot Arms in a Common Workspace," *IEEE Journal of Robotics and Automation*, RA-3(5), 437-444 (Oct. 1987).

[2] M. A. Arbib, R. O. Eason and R. C. Gonzalez, "Autonomous Robotics Inspection and Mani- pulation Using Multisensor Feedback," *IEEE Trans. on Computer*, 24 (4), 17-31 (April 1991).

[3] C. J. Paul *et. al.*, "An Intelligent Reactive Monitoring and Scheduling System," IEEE *Control Systems*, 12(3), 78-86 (June 1992).

[4] D. M. Lyons and M. A. Arbib, "A Formal Model of Computation for Sensor-Based Robotics," *IEEE Trans. on Robotics and Automation*, 2(3), 280-293 (June 1989).

[5] Gerard Berry, Georges Gonthier, "The Esterel Synchronous Programming Language : Design, Semantics, Implementation," *Research funded by the French Coordinated Research Program C3.*

[6] H. Chu and H. A. Elmaraghy, "Integration of Task Planning and Motion Control in a Multi-Robot Assembly Workcell," *J. Robotics and Computer Integrated Manufacturing*, 10(3) (June 1993).

[7] Pascal Maigret, "Reactive Planning and Control with Mobile Robots," IEEE *Control Systems*, 12(3), 95-100 (June 1992).

[8] E. Rutten, *et. al.*, "A Task-Level Robot Programming Language and its Reactive Execution," *Proc. IEEE Int. Conf. on Robotics and Automation*, 3, 2751-2756 (May 1992).

[9] E. D. Adamides and D. Bonvin, "Obtaining Synergetic Behavior by Exploiting Relations in Distributed Robot Plans," *Proc. IEEE Int. Conf. on Robotics and Automation*, 2, 1706-1712 (May 1994).

[10] F. Y. Wang, *et. al.*, "A Petri-Net Coordination Model for an Intelligent Mobile Robot," *IEEE Trans. on Systems, Man, and Cybernetics*, 21(4), 777-789 (July/Aug. 1991).

[11] R. Zurawski and M. C. Zhou, "Petri Nets and Industrial Applications," *IEEE Trans. on Industrial Electronics*, 41(6), 567-583 (Dec. 1991).

[12] I. H. Suh *et. al.*, "Design and Implementation of a Dual-Arm Robot Control System with Multi-Sensor Integrating Capability," *Proc. of IECON'91*, 2, 898-903 (Oct. 1991).

[13] I. H. Suh *et. al.*, "A Control System for Multiple-Robot Manipulators ; Design and Implementation," *Proc. of ISRAM'94*, 5, 279-285 (August 1994).

[14] *Operation Manual of Universal Force-Moment Sensor System*, Nitta Ind., Ltd. (1988).
[15] *User Manual for DT1451*, Data Translation, Inc. (1988).
[16] *CPU-30 User's Manual*, FORCE COMPUTERS, Inc. (Nov. 1990).
[17] *VxWorks Programmer's Guide*, Wind River System, Inc. (1990).

저 자 소 개

서 일 홍 (徐 一 弘)

1955년 4월 16일생. 1977년 서울대 공대 전자공학과 졸업. 1982년 한국과학기술원 전기 및 전자공학과 졸업(공박). 1982년~1985년 3월 대우중공업 기술 연구소 근무. 1987년~1988년 미국 미시간대 객원연구원. 현재 한양대 공대 전 자공학과 교수

여 희 주 (呂 熙 珠)

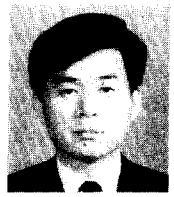
1965년 5월 2일생. 1988년 한양대학교 전자공학과 졸업. 1990년 동 대학원 전자공학과 졸업(석사). 현재 한양대 대학원 전자공학과 박사과정

김 재 현 (金 載 顯)

1969년 1월 16일생. 1991년 한양대학교 전자공학과 졸업. 1993년 동 대학원 전자공학과 졸업(석사). 현재 한양대 대학원 전자공학과 박사과정

유 종 석 (柳 宗 奭)

1969년 5월 4일생. 1993년 한양대학교 전자공학과 졸업. 1993년 동 대학원 기전공학과 졸업(석사). 현재 LG산전 연구원



오 상 록 (吳 尙 錄)

1958년 6월 7일생. 1980년 서울대학교 전자공학과 졸업. 1982년 한국과학기술원 전기 및 전자공학과 졸업(석사). 1987년 동 졸업(공박). 현재 한국과학기술연구원 정보전자연구부 책임연구원