論 文
38~9~8

로보트 자동 프로그래밍 시스템의 개발에 관한 연구

A Study on the Development of an Automatic Robot Programming System

趙 惠 卿*·李 範 熙**·高 明 三***
(Hye-Kyung Cho · Bum-Hee Lee · Myoung-Sam Ko)

요 약

본 논문에서는 로보트 언어를 이용하여 로보트에게 기능을 부여하기 위한 한가지 방법으로서, 작업단계언어(task-level language)의 개념을 이용한 로보트 자동 프로그래밍 시스템을 소개한다. 새로운 로보트 언어라기 보다는, 기존의 로보트 제어기 및 언어 시스템을 그대로 이용하되 주어진 작업을 수행하기 위한 로보트 프로그램을 작성하는 도구인 본 시스템은, 작업 단계의 명령어를 입력받아 기존의 운동 단계(motion level) 로보트 언어로 작성된 프로그램을 출력하는데, 제한된 분야의 작업 단계 명령만으로는 사용자의 의사가 충분히 반영되기 어려우므로 로보트의 동작을 위주로 지시하는 명령어도 마련하여 작업단계언어와 운동단계언어의 장점을 살리고, 비록 운동 단계의 명령어로 작업을 지정한다 할지라도 사용자가 특정 로보트 언어의 문법을 익혀야 할 필요없이 프로그램 할 수 있게 한다. 고급(high-level)로보트 언어 개발의 기초 연구라 할 수 있는 본 논문에서는, 이를 실현할 수 있는 기본적인 기능을 구현하였을 뿐만 아니라, 실용화를 제한하는 요인들을 교시 데이타등을 이용, 보완하여 실제 이용 가능성을 높힌다.

Abstract- Many works have been reported in various fields on the subject of controlling a robot with high-level robot languages. This paper presents one such effort and explains the development of an automatic robot programming system which utilizes the concept of the task level language. This system is expected to act as an intelligent supporting tool in robot programming and be put into practical use. Emphasis is placed on the role of the programming system as a tool that generates the executable robot program according to the user specified tasks. Several task level commands are used in the developed system, and the resulting inflexibility is complemented by the motion level commands of the motion level robot languages. Thus, the advantages of both task and motion level languages are utilized, and no knowledge of specific language grammer is needed even when using motion level commands. To increase the usability of the developed system, various methods are provided for supplementing the programming system using taught data.

1. 서 론

날로 그 필요성이 강조되고 있는 공정자동화 과정은 좀 더 다양하고 복잡한 로보트의 기능을 요구하고 있다.[1,2] 로보트 언어는 로보트 구조의 변화 없이 이러한 기능을 부여할 수 있는 도구로서 계속적으로 발전해 왔다. 그런데 교시(teach)를 기본으로 하는 온라인(on-line) 프로그래밍 시스템은 단순, 반복 작업에는 유용하게 이용되어 왔으나, 교육기간중 로보트가 생산에 참여할 수 없다는 점과, 센서정보의 반영이 곤란하여 복잡한 작업에 응용할 수 없다는 점등이 큰 문제점으로 지적되게 되었으며, 이와 같은 문제점의 해결을 위해 개발된 오프라인(off-line) 프로그래밍 시스템 역시, 컴퓨터 프로그래밍에 관한 전문적인 지식을 요구하여 다양하고 완벽한 기능을 구현하는 언어일수록 간단한 작업조차도 프로그램하기가 복잡해진다는 모순에 빠지게 되었다[3,4].

이러한 로보트 구동방법의 모순을 해결하기 위한 방법의 하나로, 교과서적인 언어(textual language)의 형태를 유지하되 프로그램 과정을 용이하게 하고자 하는 작업단계언어(task-level language)의 개발을 들 수 있는데, 작업단계언어는 로보트에게 부여할 작업을 로보트의 동작을 위주로 기술하기 보다는, 로보트의 동작에 의한 작업 공간 내 각 물체의 상태변화로 기술하므로, 사용자가 물체 중심으로 작업을 계획하고 명령할 수 있다는 장점을 취한 언어이다.[5] 작업 단계의 언어는 작업환경 모델링 기능과 원하는 작업 사양을 효과적으로 표현할 수 있는 표현 형식을 갖추고 있어야 하며, 이 사양을 분석하여 실제로 로보트를 제어하기 위해서는 충돌이 발생하지 않는 경로를 선택할 수 있어야 하며, 잡는 위치의 선정능력 및 제대로 수행되지 못한 작업에 대처하는 능력등 전문적인 의사결정 능력이 요구된다.

작업단계 언어의 기원이라 할 수 있는 AUTOPASS[6]는 사용자로 부터 조립 단계를 입력 받아 이미 만들어진 모델을 이용하여 이를 로보트 구동이 가능한 형태의 명령어로 바꾸어 준다. 작업 단계 언어의 각 분야에 대한 기본적인 구조는 마련되었으나, 로보트의 동작으로 인해 발생한 작업 공간 내 물체들의 위치 관계의 변화를 자동적으로 인식하지 못하므로 사용자가 이를 일일이 입력하여야 하며, 또한 논리적인 오류의 확인이 불가능하므로, 사용자는 실현 가능한 작업만을 입력하도록 유의하여야 한다.

기계적 조립 작업을 대상으로 하는 LAMA[7]는 간단한 프로그램 형태의 조립 작업 계획을 받아들여, 세부적인 작업으로 확장해 나가는 구조로 되어 있다. 다면체(polyhedral) 형태의 물체를 고려한 경로 계획 및 충돌 감지가 가능하며, 조립을 위해 실현해야 할 동작은 내부적으로 각종 제한조건(constraints)에 의해 표현되는데, 여기에는 제한평면(constraint plane)의 개념을 이용하였다. 그러나 한 물체에 대한 제한 조건들 조차도 서로 간섭이 없어야 한다는 가정이 만족되어야 한다는 문제점이 있다.

Edinburgh에서 개발된 RAPT[8,9]는 작업지정방법(task specification)을 중점적으로 다룬 언어로, 작업을 물체들의 상호 위치 관계로 기술하고 이로부터 필요한 위치데이타를 추론하는 방법을 제시하였다. 'against', 'aligned'등 공간 상의 위치 관계는 수학적 방정식으로 변환되며, 이같은 비선형 방정식들을 풀어 각관절의 각도등 로보트 구동에 필요한 값들을 계산해 낸다. 이 방법은 전형적인 문제들에 대해서는 풀이가 가능하였으나, 기하학적인 모델 및 위치 관계들의 정의가 불완전하여 경로 계획이나 충돌 확인등에 이용될 수 없다.

비젼 시스템을 이용하여 작업의 실현 가능성을 많이 고려한 Handey[10]에서는 'MOVE object TO destination'이라는 간단한 명령어, 즉 pick & Place동작을 위한 충돌회피 및 잡는 위치 선정(grasping) 문제를 다루었다. 그러나 컴플라이언트 운동(compliant motion)이나 오차 보상에 관한 기능은 마련되지 못하였다.

위에서 볼 수 있듯이 작업단계 언어에 있어 필수적인 의사결정 능력은 많은 데이타와 계산능력 및 아직 완전히 해결되지 않은 문제들의 풀이까지 요구하여 이들 언어의 실용화를 제한하므로, 현재까지 개발된 작업단계 언어는 연구 단계에 머무르고 있다.

본 연구는 전술한 작업단계 언어의 개념을 이용하여, 현실적으로 이용가능한 프로그래밍 도구로서 지적인 보조자의 역할을 하는 로보트 자동프로그래밍 시스템(이하, SUN-ARPS : Seoul National University Automatic Robot Programming System이라 칭함)을 개발하여, 프로그래밍에 익숙하지 않은 사용자들도 쉽게 로보트를 이용할 수 있게 한다. 새로운 로보트 언어라기보다는, 기존

*正 會 員 : 서울大 大學院 制御計測工學科 博士過程
**正 會 員 : 서울大 工大 制御計測工學科 助教授·工博
***正 會 員 : 서울大 工大 制御計測工學科 教授·工博
接受日字 : 1989年 5月 10日
審査完了 : 1989年 8月 23日

의 로보트 제어기 및 언어시스템을 그대로 이용하되 대화를 통해 주어진 작업을 수행하기 위한 로보트 프로그램을 작성해 주는 도구인 SUN-ARPS는, 작업 단계의 명령어를 입력받아 운동 단계의 로보트 언어로 작성된 프로그램을 출력하기 위하여 작업공간 모델링, 충돌 회피 경로의 선택등을 포함한 작업단계언어의 기본적인 문제들을 간단한 형태로 구현한다. 또한, 일률적인 작업의 정의로 인하여 사용자의 의사를 세부적으로 전달하기 어렵다는 문제를 보완하기 위하여 운동단계의 명령 및 도구(tool) 조작을 위한 명령어도 마련하여, 두 단계 언어의 장점을 살리는 구조로 되어 있다.

다양한 종류의 로보트와 로보트의 언어를 적용 대상으로 하는 SUN-ARPS는 PUMA구동을 위한 VAL[11], SCARA형 로보트 구동을 위한 SUNL-I[12]을 합성하여 실험하며, IBM/AT에서 인공지능 시스템 개발용 언어로서 융통성있는 문장처리가 편리한 GCLISP[13]으로 실현된다.

본 논문의 구성은 다음과 같다. 먼저 2장에서 SUN-ARPS의 구조 및 적용 범위에 대하여 논하고, 3장에서는 프로그램을 합성하기 위한 SUN-ARPS작업 계획 시스템의 구조를 설명하며, 4장에는 SUN-ARPS를 이용하여 실제 작업을 위한 프로그램을 작성하는 과정 및 실험 결과가 제시된다. 5장에서는 SUN-ARPS의 특징과 앞으로의 연구방향을 제시하고 결론 맺는다.

2. 로보트 자동 프로그래밍 시스템의 설계

SUN-ARPS는 다음과 같은 가정하에 구성된다.

· 주요 작업 대상은 Pick & Place 및 간단한 조립 작업으로 한다.

· 환경 모델링을 위한 자료의 위치 및 방향 정보 오차는 없다.

· 로보트의 동작에 의한 오차가 없어 실제 환경과 모델링 된 환경이 항상 같다.

위와 같은 가정하에 제한된 분야의 작업에 대한 로보트 언어를 작성한다 할지라도, 작업 단계의 명령어로 작업을 지정하기 위해서는 작업 단계언어의 중요한 몇 가지 기능이 구현되어야 하므로, SUN-ARPS는 그림 2.1에 보인 것과 같이 전문적인 의사 결정을 담당하는 몇 개의 모듈로 구성된다. 사용자가 사용자 인터페이스(User Interface)를 통해 원하는 작업 및 작업 환경에 대한 데이타를 입력하면, 경로 계획기(Path Planner), 작업 관리자(Task Manager), 속도 관리자(Speed Manger), 작업 공간 관리자(Work-spce Manager),

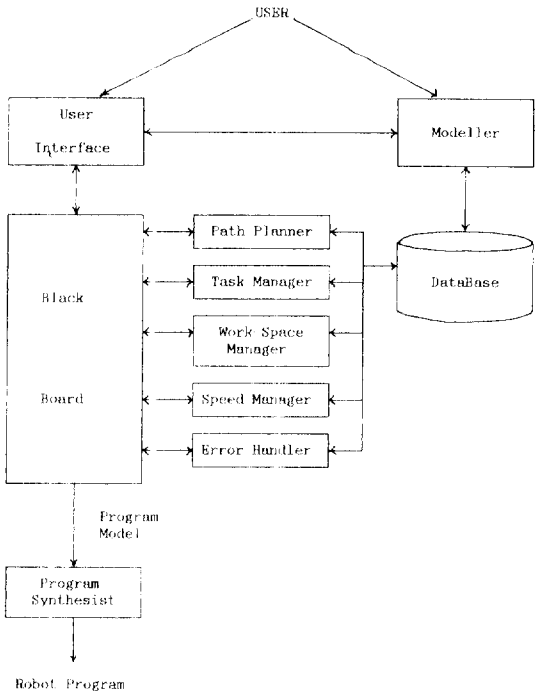

그림 2.1 SUN-ARPS의 구조
Fig. 2.1 Structure of the SUN-ARPS

오류 확인기(Error Handler)는 환경 모델러(World Modeller)가 제공하는 정보를 바탕으로 블랙보오드(black board)를 통해 서로 정보를 교환하여, 매니퓰레이터 구동을 위한 프로그램 모델을 만들어 낸다. 이 결과는 프로그램 합성기(Program Synthesist)에 의해 원하는 로보트 언어로 변환되어 진다. 각 모듈의 역할 및 구조는 3장에서 자세히 설명한다.

다음은 위의 모듈이 구현하는 SUN-ARPS의 각종 명령어를 통해, 이를 이용하여 실현할 수 있는 작업의 범위를 알아본다. SUN-ARPS 명령어의 풀다운 메뉴(pull-down menu)는 그림 2.2와 같은 계층적 구조를 이루는데 그 성격에 따라 크게 다음의 다섯가지 부류로 구분된다.

· 시스템 자체에 관한 명령어——SYSTEM
· 파일(file) 처리를 위한 명령어——FILE
· 로보트에게 작업을 지정하기 위한 명령어——TASK
· 로보트동작 이외의 기능을 부여하기 위한 명령어——EDIT
· 작업 환경에 대한 데이타 관리를 위한 명령어——DATABASE

SUN-ARPS의 주 작업 대상은 Pick & Place 및

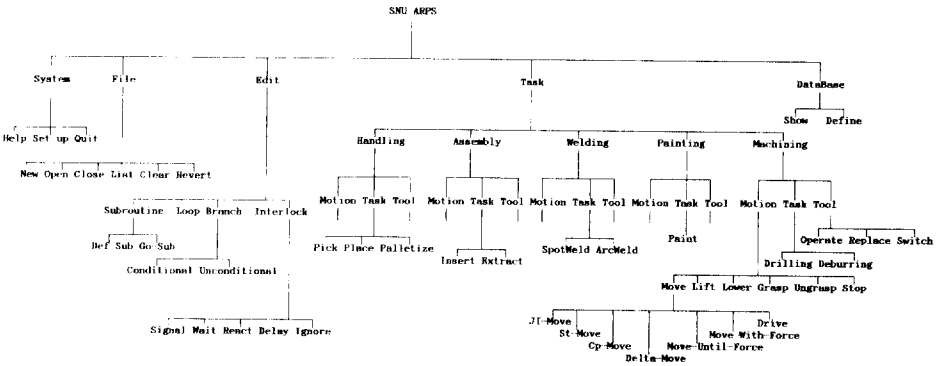

그림 2.2 SUN-ARPS의 명령어 구조
Fig. 2.2 Command Structure of the SUN-ARPS

간단한 조립작업이나, 다양한 작업에 응용할 수 있는 확장성을 고려하여 용접(welding), 페인팅(painting), 머쉬닝(machining)등을 위한 작업단계 명령어도 마련한다. 현재는 이들 작업을 위한 운동경로 명령만이 생성되지만, SUN-ARPS의 대상언어를 추가함으로써 이들 작업을 완전히 실현할 수 있는 프로그램을 작성할 수 있을 것으로 사료된다.

또한 각 명령어는 또 다른 명령어들과 나무 구조(tree-structure)로 연결되어 있어, 명령의 단계가 낮아질수록 좀 더 세부적인 지정이 가능해 진다. TASK에 관한 명령어들을 예로 보면, 사용자는 처음에 작업에 따른 명령어군을 선택하고, TASK모드에서 작업 단계의 명령어로 작업을 입력하다가 좀 더 세부적인 지정이 필요하면, 로보트의 운동을 위주로 작업을 지정할 수 있는 MOTION모드를 택하여, 이동시의 보간 형태(interpolation type)까지 조정할 수 있다. 이와 같은 명령어 체계를 통해 로보트 명령어의 수준을 한 단계로 고정하는 데에서 기인하는 단점을 보완하고, 각 단계 언어의 장점을 살려, 간단한 작업은 간단하게, 어려운 작업은 세부적인 지정을 통해 실현될 수 있게 한다.

3. 작업 계획 시스템의 설계

이 장에서는 그림 2.1에서 보인 SUN-ARPS의 작업계획 시스템의 각 부분에 대해 설명한다.

3.1 작업공간 모델러(World Modeller)

작업공간 모델링은 로보트 뿐만 아니라 작업공간 내의 센서, 주변기 및 각종 물체들에 공통된 하나의 좌표계를 두어 이들 상호간의 결합 관계를 나타내는 작업으로, 작업환경에 통일성을 부여하고 보다 지능적인 작업을 가능하게 한다.

일반적으로 조립 작업시의 위치 데이타는 운반시 이용되는 위치 데이타와 세부적인 조립 작업시에 이용되는 위치 데이타로 구분할 수 있는데, 전자는 교시를 통하여 보다 정확히 얻어지나 후자는 오히려 부정확해지는 특징이 있다. 이러한 특징을 고려하여, SUN-ARPS의 모델러는 CAD의 출력을 분석하여 위치정보를 계산하는 것을 원칙으로 하지만 각 물체의 기준좌표가 되는 잡는 위치(grasp position)는 교시 데이타(teach data)를 받아들여 정확하게 보완한다.

AUTOLISP[14]을 이용하여 구현된 모델러의 각 명령어는 AUTOCAD[15]의 사용자 메뉴 기능을 이용하여 편리하게 이용되는데, 작업공간 내의 다양한 물체를 체계적으로 표현하는 규칙으로는 원기둥과 사각기둥을 기본으로 하는 일종의 CSG (Constructive Solid Geometry)를 택하여, 모든 물체를 원기둥과 사각 기둥을 조합한 나무 구조(tree structure)로 표현한다. 모델링 시스템에서 허용하는 물체 유형은 다음과 같다.

- BOX : 구성원소 사각기둥
- CYLINDER : 구성원소 원기둥
- RING : 중앙이 비어있는 원기둥
- BOX-WITH-HOLES : BOX가 기본이된 사각기둥과 원기둥의 CSG
- CYLINER-WITH-HOLES : CYLINDER가 기본이된 사각기둥과 원기둥의 CSG

CAD모델러를 이용하여 작업공간을 정의하는 예

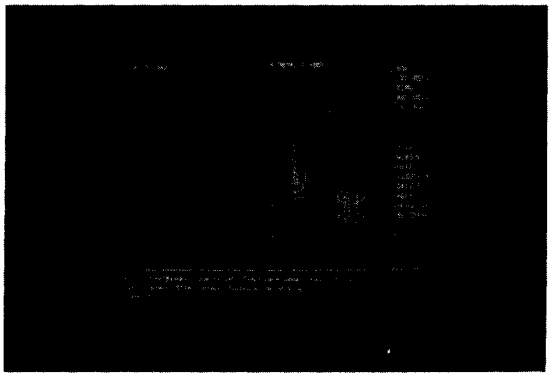

그림 3.1 CAD를 이용한 모델링
Fig. 3.1 Modelling by CAD System

를 아래 그림 3.1에 보인다. 데이터베이스에는 물체의 위치 및 구조에 대한 정보 뿐만 아니라, 다른 물체들과의 위치관계나 무게, 크기등도 함께 골조구조(frame)[16]에 저장되는데, 그림 3.1의 상자를 예로 들면 아래와 같다.

[BOX
type : cubic
size : 20 20 20
weight : 550
grasp-pos : p1
base : (0 0 -15 0 0 0)
concave-box : (hole 1 (5 5 15)
(5 5 8 0 0 0))
concave-cyl : (hole 2 (2 15)
(-5 -5 8 0 0 0))
approach : (0 0 50 0 0 0)
supported-by : conveyor
supports : nil]

3.2 경로 계획기(Path Planner)

경로 계획기는 작업공간의 상태로부터 접근위치, 후퇴위치 및 충돌이 일어나지 않기 위한 중간 경위점을 선택한다. 충돌회피를 위한 알고리즘에 관하여는 현재까지 많은 연구 결과가 발표되었는데, 회전 관절(revolute joint)을 갖는 로보트에 대한 것으로는 크게, 장애물을 로보트의 관절좌표(configuration space)에서 근사시키는 방법[17]과, 장애물 경계선의 정확한 수식을 이용하는 접근방법[18, 19]으로 양분된다.

본 연구는 충돌회피를 위한 알고리즘의 개발이라는 측면보다는 작업단계언어의 기본적인 골격을 구현하는 데에 중점을 두고 있으므로, 다음과 같

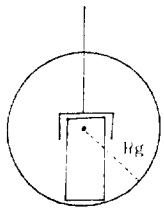

그림 3.2 로보트 손 끝의 근사 과정
Fig. 3.2 Approximation of the Robotic Hnad

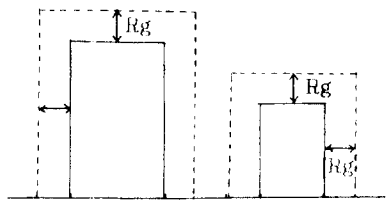

그림 3.3 장애물 근사 과정
Fig. 3.3 Approximation of the Obstacles

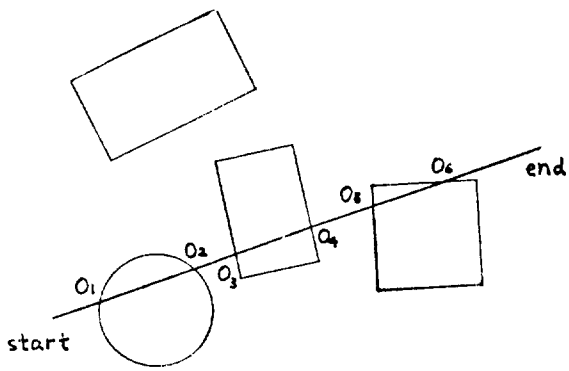

그림 3.4 수평면 확인 과정
Fig. 3.4 Horizontal Testing Process

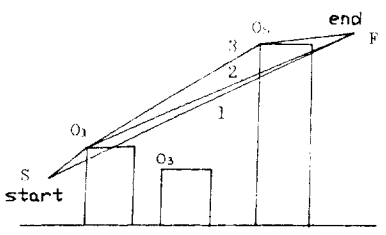

그림 3.5 수직면 확인 과정
Fig. 3.5 Vertical Testing Process

은 가정하에 매우 간단한 알고리즘을 이용한다. [18, 19]

—로보트 손 끝과 물체의 충돌만을 고려한다.

—Z방향의 경로 변환만을 허용할 때의 최단 거리 경로를 찾는다.

—이동시 로보트가 잡고 있는 물체의 방향은 변경하지 않는다.

SUN-ARPS에서 채택한 4 DOF의 모델은 x—y 평면에 수직인 어떤 평면으로 절단하든지 모든 물체가 직사각형으로 표현되는 특징이 있다. 이와 같은 특징은 3차원의 문제를 두 개의 2차원 문제로 분할하여 고려할 수 있게 하므로 본 연구의 경로선택 방법도 두 개의 2차원 문제로 분할하여 접근한다. 이 충돌 회피 알고리즘은, 2차원 평면에서 다각형의 물체를 피하여 이동하는 최단 경로는 이 다각형의 꼭지점 중 몇 개를 경위하게 된다는 특징에 근거를 두고 있다[19].

충돌 회피를 위한 중간 경로를 선택하는 방법은 다음과 같다.

a) 장애물 모델의 근사

그림 3.2과 같이 로보트의 손끝을 물체의 방향변화에 따른 영향이 무시될만큼 충분히 큰 반지름을 구로 근사시킨다. 그리고 그림 3.3에서와 같이, 모든 장애물의 크기를 이 구의 반지름 만큼 증가시켜 조정하여, 로보트의 손끝을 한 점으로 다룰 수 있게 한다.

b) 수평면 확인

이동 경로의 시작점과 끝점으로 이루어진 직선과 작업공간 내의 모든 장애물을 x—y평면상에 투영(projection) 시켜 2차원에서 이들의 교점을 구함으로써 충돌 가능한 물체들을 찾아낸다(그림 3.4 참조).

c) 수직면 확인

b)에서 구한 물체들을 이동 경로가 이루는 직선이 x—y평면상에 투영될 때 형성되는 면으로 절단하면, 충돌이 발생하지 않도록 Z축 방향으로 경로를 변환하는 문제는 2차원 공간에서 다각형의 물체와의 충돌을 피하는 경로를 찾는 문제로 귀결되며, 이 경우 최단 거리 경로는 이들 다각형의 꼭지점을 경위하게 된다는 특징[19]을 이용할 수 있다. 그런데 SUN-ARPS의 모델의 특징상 이들 다각형은 모두 직사각형으로 표시되므로, 이동 경로의 시점과 종점이 이루는 직선의 기울기와, 시점 또는 종점과 b) 에서 구한 교점이 이루는 직선의 기울기를 비교함으로써, 충돌이 일어나지 않기 위해 경위해야 할 꼭지점들을 찾는다.

그림 3.5에 c) 의 과정을 보였는데, $\overline{SF}$의 기울기가 $\overline{SO_1}$의 기울기 보다 작으므로 점 0_1을 경위하도록 경로를 2로 변경하며, $\overline{O_1F}$의 기울기가 $\overline{O_1O_5}$의 기울기보다 작으므로 점 0_5를 지나도록 변경한 경로 3을 얻는다. 시스템은 중간 경위점 0_i의 역기구학을 풀어 작업공간 내에 있는가를 확인하고 작업공간에서 벗어난 경우 에러 메시지를 출력한다.

3.3 작업 공간 관리자(Wok-spece Manger)

이 모듈은 작업 공간 내에서의 각 물체들의 위치를 관리하는 부분이다. 주 기능으로는 각 로보트에 알맞은 역기구학(inverse kinematics)을 풀어 그 결과를 작업공간 모델과 비교하여 도달 불가능한 지점이 프로그램되는 것을 방지하는 기능, 기구학을 풀어 새로운 위치 데이타를 계산해 내는 기능등이 있으며, 접촉 확인(touch test) 알고리즘을 이용하여, 주어진 위치를 이미 다른 물체가 점유하고 있지는 않은가, 물체를 놓도록 지정된 점의 공간은 충분한가등 작업 공간 관리를 위한 기본적인 조건을 확인하는 기능이 구현되어 있다.

앞절에서 설명하였듯이 SUN-ARPS의 물체모델은 3차원의 문제를 두개의 2차원 문제로 분할하여 풀기에 적합하므로 물체들간의 접촉 확인(touch

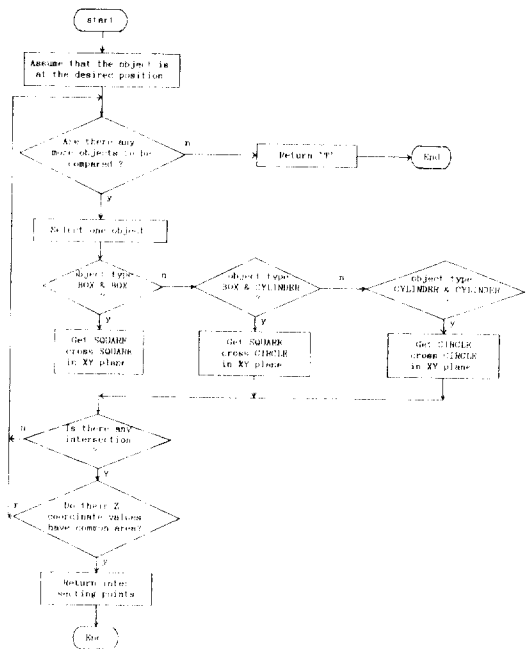

그림 3.6 접촉확인 알고리즘의 흐름도
Fig. 3.6 Flow Chart for the Touch Test Algorithm

test)도 2차원의 문제로 귀결되는데, CSG의 기본이 되는 형태가 원기둥과 사각 기둥이므로, 접촉확인은 사각형과 사각형, 원과원, 원과 사각형간의 교점 풀이를 응용하는 방법으로 해결한다. 그림 3.6에 접촉확인 알고리즘의 흐름도를 보인다.

3.4 작업 관리자(Task Manager)

2장에서 설명한 SUN-ARPS의 각 명령어를 수행하는 과정은 그림 3.7과 같이 나타낼 수 있다. 사용자가 명령어를 선택하면, 시스템은 현재 상태에 그 명령어를 적용할 수 있는가를 확인하여, 적용 가능할 경우 그 명령어 수행에 필요한 변수를 받아들이고, 주어진 조건하에 실행 가능한가를 확인한다. 만족된 작업만이 시행되며, 그렇지 못한 작업은 에러 메시지와 함께 취소된다.

작업관리자에는 각 명령어의 위와 같은 수행과정이 규칙으로 정의되어 있어, 이 규칙에서 제시하는 바를 따라 프로그램을 발전시켜 나간다. 명령어 'PCIK'에 대한 규칙을 예로 들면 아래와 같다.

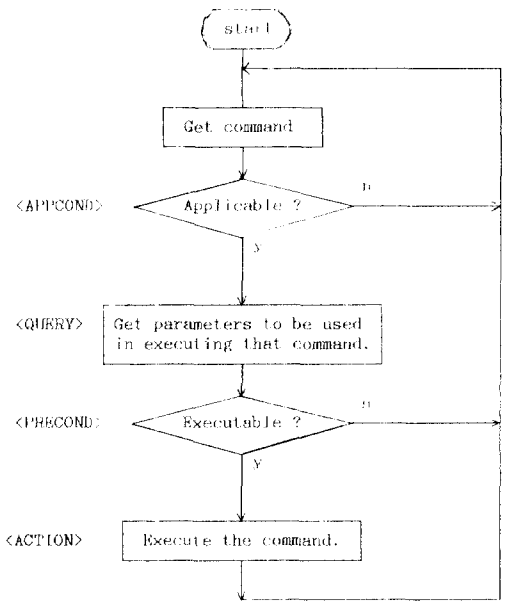

그림 3.7 SUN-ARPS의 명령어 실행 과정
Fig. 3.7 Command Excution Process of the SUNARPS

```
(RULE PICK
    (APPCOND (hand-empty))
    (QUERY target-obj
        time)
    (PRECOND (no-supports check)
        (in-workspace object check)
        (weightbound check)
        (sizebound check)
        (path plan)
    (ACTION (move-to approach)
        (move-to object)
        (grasp)
        (move-to depart)
        (update)))
```

이와 같이 규칙의 형태로 명령어를 표현하면 명령어를 표현하면 명령여 구조가 이들을 실현하기 위한 제반 추론 시스템과 독립되므로, 필요에 의해 새로운 명령어를 추가하거나 기존 명령어의 수행 내용을 변경하는 작업이 손쉬워진다.

3.5 오류 확인기(Error Handler)

로보트 프로그램에 익숙하지 않은 사용자들은 비논리적인 즉, 현재 환경에서는 실행할 수 없는 명령어를 입력하기도 한다. 이 모듈은 이러한 오류를 방지하기 위해 작업 상황을 이해하고 있다가 적절한 조언을 하여 사용자의 프로그래밍을 돕는 역할을 한다.

3.4절에서 설명한 각 작업에 대한 규칙의 APPCOND 혹은 PRECOND에 의해 선택적으로 호출되어 주어진 조건을 확인하고, 상황에 따라 'T' 혹은 'F'를 되돌려 주는 오류 확인기는 이 장에 소개되는 대부분의 모듈이 자신이 담당한 분야에서의 오류 확인 과정을 수행하므로, 주로 대상물 간의 위치 관계상의 오류를 확인한다. 오류 확

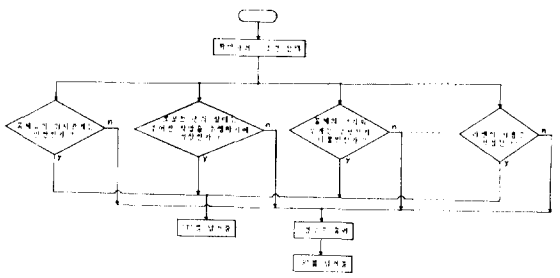

그림 3.8 오류 확인기의 동작
Fig. 3.8 Operation Flow for the Error Handle

인기의 동작을 도식적으로 나타내면 아래 그림 3.8과 같다. 이러한 과정을 통해 기본적인 동작이 확인되므로 비합리적인 명령어는 에러 메시지를 출력하고 자동적으로 취소된다.

3.6 속도 관리자(Speed Manager)

많은 로보트 언어들이 속도를 지정함에 있어 각 관절 모터의 최대속력에 대한 상대속도를 이용하나, 초보자는 이러한 상대 속도의 지정에 익숙하지 않으므로 이 모듈은 작업을 수행하는 데에 소요되는 개략적인 시간을 추정하여 사용자가 의도하는 작업속도를 결정하도록 돕는다. 명령어 'PICK'의 경우를 예로 들어 보면, 이 명령어의 실행 과정은 물체의 접근 위치까지의 이동, 물체를 잡기 위한 이동, 물체를 잡고 물러나오는 이동등 몇 개의 경로로 나뉘어진다. 이와 같이 보통 한 명령어를 수행하기 위해 이동해야 할 경로는 몇 개의 부분으로 나누어 볼 수 있는데, 작업의 상황에 따라 각 부분 구간을 이동하는 속도도 적절하게 변경되어야 한다. 다음에 이와 같은 사실을 반영한 속도계산 방법에 대해 살펴본다.

· 직선 보간 운동시의 속도 계산

직선 보간 운동시 속도를 계산하는 방법은 다음과 같다.

a) 각 부분 구간의 이동속도에, 작업에 따라 0부터 1까지의 가중치를 할당한다.

b) 각 부분 구간의 이동 거리에 이 가중치를 곱하여 제일 빠른 속도(가중치 1)로 이동해야 하는 구간의 속도를 결정하기 위한 가상의 이동 거리를 구한다.

$$D_i' = D_i / W_i \qquad (3.1)$$

D_i : 각 부분 구간의 거리
D_i' : 각 부분 구간의 가상의 이동거리
W_i : 각 부분 구간 속도의 가중치 $(0 < W_i \leq 1)$

c) 사용자로부터 한 작업을 마치는데 할당된 시간을 입력받아 중간경위점에 머무르는 시간, 로보트 손조작에 필요한 시간등을 빼고 이동에만 소요할 수 있는 시간을 계산한다.

$$Tr = T - (h * \text{hand-operation-time}) \qquad (3.2)$$

h : 핸드의 on/off 횟수
T : 전체 동작을 마치도록 요구된 시간
Tr : 이동에 이용할 수 있는 시간

d) 각 구간의 가상 이동 거리를 모두 합하여 이동에 소요되는 시간으로 나누어 제일 빠른 속도로 이동하는 구간의 속도를 구한다.

$$Vm = \frac{1}{Tr} \sum_{i=1}^{n} Di' \qquad (3.3)$$

n : 부분 구간의 수

e) d)에서 구한 속도가 제한값을 초과하면 사용자에게 이동시간을 증가시킬 것을 요구하고 c)로 돌아간다.

f) 최대 속도에 각 구간에 할당된 가중치를 곱하여, 그 구간을 이동하는 데 적합한 속도를 구한다.

$$V_i = W_i * V_m \qquad (3.4)$$

g) 구해진 ㎜/s 단위의 속도를 실제 로보트 언어에서 이용하는 %속도 변환한다.

· 관절 보간 운동시의 속도 계산

관절 보간 방법에서는 상대 속도가 각 관절을 구동하는 써보오 모터의 단위 시간당 엔코더 값의 증분과 대응 되므로 이를 이용하여 속도를 결정한다. 보통 관절보간 운동에는 사다리꼴 속도 프로파일(pro-file)이 많이 이용되는데, 같은 사다리꼴 속도 프로파일을 쓰더라도 가속구간을 고정하는 방법, 가속도를 고정하는 방법등 이를 실제로 실현하는 방법은 로보트 언어마다 차이가 있다. 그러나 이 모듈은 대강의 이동 시간만을 필요로 하므로 실제 로보트 언어에서 이용하는 방법을 쓰지 않고, 그림 3.9(a)에서와 같이 가감속 구간을 각각 이동 시간의 1/5로 고정하고, 그때의 가속도는 제한값을 넘지 않는다는 가정하에 이동 속도를 결정한다. 이와 같이 가감속 구간을 고정하면 이동 속도는 그림 3.9(b)에서와 같이 사용자가 요구한 이동 시간의 4/5동안 평균적으로 출력해야 할 엔코더의 증분을 계산하는 문제로 귀결된다.

속도를 구하는 단계는 아래와 같다.

a) 이동 구간 시점, 종점의 역기구학을 풀어 주어진 위치로 이동하기 위한 각 관절의 각도 증분을 계산한다.

$$\Delta\theta_i = \theta_{ei} - \theta_{si} \quad (\text{deg}) \qquad (3.5)$$

b) 각 관절마다 a)에서 구한 각도 증분값을 엔코더 펄스당 관절의 회전비를 이용하여 이동에 필요한 엔코더 값으로 변환한다.

$$\mu_i = \frac{\Delta\theta_i}{\Delta \text{Enc}_i} = \frac{360 \text{ deg/gear ratio}}{\text{pulses per revolution}} \qquad (3.6.a)$$

$$\Delta \text{Enc}_i = \Delta\theta_i / \mu_i \quad (\text{pulses}) \qquad (3.6.a)$$

표 3.1 프로그램 모델에 따른 목적 언어 명령어의 비교

Table 3.1 Comparison of two target language commands according to the variation of program model.

Program Model Syntax	VAL Syntax	SNUL 1 Syntax
(ready)	READY	READY
(config configuration)	configuration	
(approach m-type position distance)	m-type : straight APPROS position, distance	SET positon-a = position SHIFT position-a BY distance MOVES position-a
	m-type : joint APPRO position, distance	SET position-a = position SHIFT position-a BY distance MOVE position-a
(depart m-type position distance)	m-type : straight DEPARTS distance	SET position = position SHIFT position-d BY distance MOVES position-d
	m-type : joint DEPART distance	SET position-d = position SHIFT position-d BY distance MOVE posotion-d
(position motion-type p-name)	m-type : straight MOVES position	MOVES position
	m-type : joint MOVEposition	MOVE position
(drive jt-num delta-angle speed)	DRIVE jt-num, delta-angle, speed	
(speed sp-value)	SPEED sp-value	SPEED sp-value
(close-hand)	CLOSEI	CLOSEI
(open-hand)	OPENI	OPENI
(labelling label)	label	label
(loop-start var value label)	SETI var = value label	FOR var = 1 TO value STEP 1
(loop-end var label)	SETI var = var—1 If var GT 0 GOTO label	NEXT

c) 사용자가 의도한 시간을 샘플링 주기로 나누어 이동에 소요할 수 있는 샘플링 갯수를 구한다.

$$N_f = \mathrm{INT}((4/5) * \mathrm{Tr}/\text{Sampling period})$$

d) 이동에 필요한 엔코더 증분을 이동시 소요할 수 있는 샘플링 갯수로 나누어, 매 샘플링시 평균적으로 출력해야 할 엔코어 값을 구한다.

$$E_i = \Delta \mathrm{Enc}_i / N_f \quad \text{(pulses/sample)} \qquad (3.7)$$

e) 샘플링 주기당 엔코더 증분량과 상대 속도와의 관계를 이용하여 샘플링 때마다 주어진 펄스수를 출력하는 %속도를 찾는다.

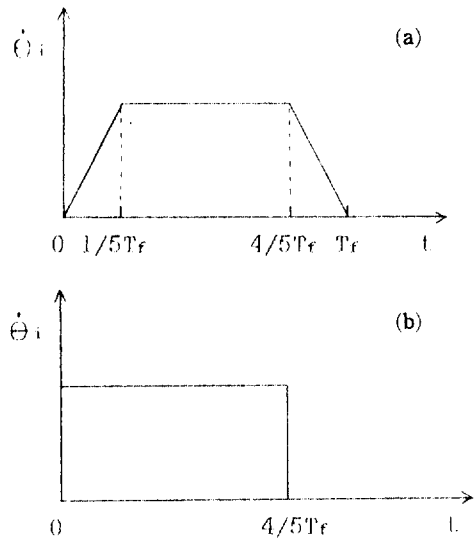

그림 3.9.a SUN-ARPS의 속도 프로파일
Fig. 3.9.a Velocity Profile of the SUN-ARPS
그림 3.9.b 등속구간의 속도를 얻기 위한 속도 프로파일
Fig. 3.9.b Velocity Profile for Obtaining Constant Speed

$$V_i = f(E_i) \tag{3.9}$$

f) 각 관절에 대해 구한 %속도중 최대값을 그 구간 이동시의 속도로 할당한다.

$$V = \max_i V_i \qquad \begin{matrix} i = 1, \cdots, n \\ n : \text{number of joints} \end{matrix} \tag{3.10}$$

g) f)에서 구한 속도가 제한값을 넘으면,사용자에게 이동시간을 증가시키도록 요구하고 c)로 돌아간다.

3.7 프로그램 합성기(Program Synthesist)

지금까지 프로그래밍시 필요한 전문적인 의사결정을 담당하는 SUN-ARPS의 각 모듈의 기능에 대해 살펴 보았다. 그러나 이와 같은 과정을 통하여 생성되는 것은 실제 로보트 프로그램이 아니라 명령어와 이에 필요한 변수들을 포함한 프로그램 모델이다. 실제 로보트 프로그램은 프로그램 모델을 로보트 언어로 바꾸어 주는 프로그램 합성기에 의해 작성된다. SUN-ARPS의 목적 언어가 운동 단계의 언어들이므로 이 프로그램 모델은 운동 단계의 언어와 유사한 문법으로 로보트의 동작 및 이에 필요한 인자를 표현한다.

시스템이 이러한 구조를 갖는 것은 여러 가지 목적 언어를 생성하기에 유용하도록 목적 언어와 비교적 무관한 부분과 목적 언어와 관계깊은 부분을 분리하였기 때문이다. 표 3.1에 프로그램 모델의 문법에 대한 VAL의 문법과 SNUL-I 문법의 일부를 보인다. 이로부터 프로그램 합성기의 입력에 따른 각 언어 출력의 대응 관계를 알 수 있다.

3.8 사용자 인터페이스(User Interface)

사용자 인터페이스는 시스템과 사용자와의 의사전달을 담당하는 부분으로 SUN-ARPS의 사용자층이 프로그램의 경험이 없는 초보자임을 고려할 때, 그 중요성이 강조되는 부분이다. 사용자 인터페이스로는 용자와의 대화를 담당하는 프로그램 상담자, 메뉴 및 기타 화면관리를 위한 윈도우 관리자, SUN-ARPS에서 사용자에게 제공하는 프로그램 파일 및 각종 내부 파일을 관리하는 파일 관리자등이 구현되어 있다.

4. 실험 및 결과

이 장에서는 SUN-ARPS를 이용하여 Pick & Place작업을 위한 프로그램을 작성하는 과정을 통해 그 기능을 실험한 결과를 보인다. 아래에 모델링 과정을 통해 골고구조의 데이타 베이스를 만든 후, 지붕(roof)을 들어 올려 담(wall)위에 놓은 작업을 입력하는 대화 및 이 과정의 디스플레이 화면에 대하여 설명한다. 시스템과 사용자와의 대화는 메뉴위주로 진행되나 편의상 시스템의 질문을 'Q'로, 사용자의 대답을 'A'로 표기한다.

Q : COMMAND?
A : set-up
Q : What is the robot to be used?
A : puma
Q : What is the language to be generated?
A : val
Q : what is the Workcell name to work in?
A : world
Q : COMMAND?
A : pick
Q : Which object?
A : roof
Q : In how many seconds do you want to complete the operation?
A : 7
Q : COMMAND?
A : show
Q : I am holding ROOF.

I am at position P3-A.
3 objects are in workspace.
The position to grasp PILLAR is P1.
The position to grasp WALL is P2.
The posititon to grasp ROOF is P3-A.
Position P1, P2, P3, P4, P5, P3-A are defined.
No sensor is used.
No I/O port is used.
Do you want more information?

A : n
Q : COMMAND?
A : Place
Q : On some Object or At some position?
A : on
Q : On which Object ?
A : wall
Q : What do you want to call the position on WALL?
A : onwall
Q : In how many seconds do you want to complete the operation?
A : 8

그림 4.1은 명령어 PICK을 선택한 경우의 화면을 나타낸다. 화면 좌측 상단에는 명령어 선택을 위한 COMMAND 윈도우가 나타나며, 우측 하단에는 명령어를 편리하게 선택하기 위한 기능키(function key)가 설명되어 있다. 또한, 화면의 좌측 하단에는 지금 선택한 명령어에 대한 과정(history)이 나타나 현재의 명령어 PICK이 TASK모드 중 HANDLING에 속함을 알려 주며, STATUS 윈도우에는 작업공간 'WORLD'에서 PUMA로보트를 이용해 작업하며, 이 작업은 VAL언어로 실현되기 위해 'TEST'라는 파일에 저장되게 된다는 내용이 나타나 있다. 시스템은 QUERY 윈도우를 통해 명령어 PICK의 효과를 보여 주며, 이 명령어를 선택할 것인가를 묻고 있다.

그림 4.2는 앞의 대화 중 'show'에서 보였듯이 지붕을 잡으라는 명령이 입력된 후, 작업 공간의 상태를 설명한다. LIST윈도우에는 지금까지 입력된 명령어 'PICK ROOF'이 나타나며, 이때 작업공간의 상태를 설명하는 명령어인 SHOW를 선택하면, 로보트 손의 상태, 현재위치, 작업 공간 내의 물체 들에 관한 간단한 정보가 나타난다. 이 명령어는 기타 작업 공간의 위치 데이타, 센서 및 I/O에 대한 정의 내용을 설명하고, 사용자가 지정하는 특정한 물체에 대한 자세히 정보를 전달하기

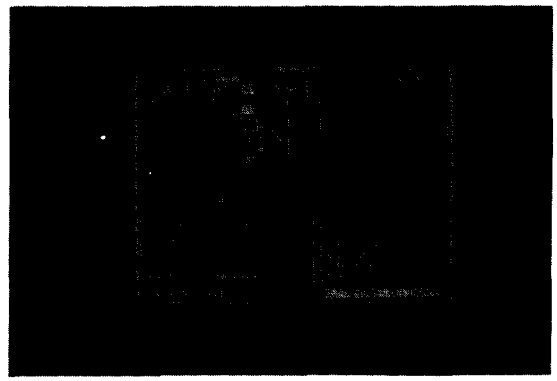

그림 4.1 PICK동작 효과 설명
Fig. 4.1 Description of PICK Operation

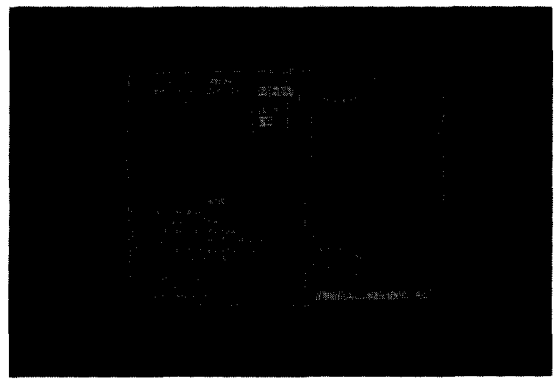

그림 4.2 작업공간의 설명
Fig. 4.2 Description of Work Space

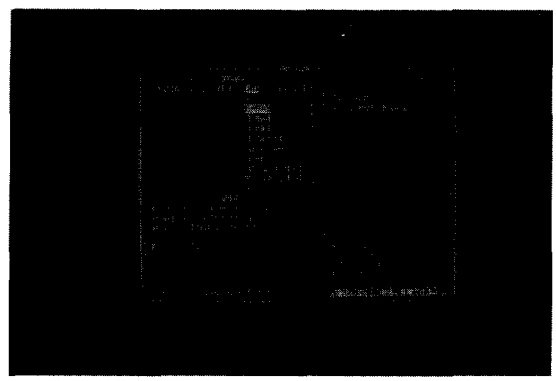

그림 4.3 부적절한 명령에 대한 경고
Fig. 4.3 Error Message for an Improper Command

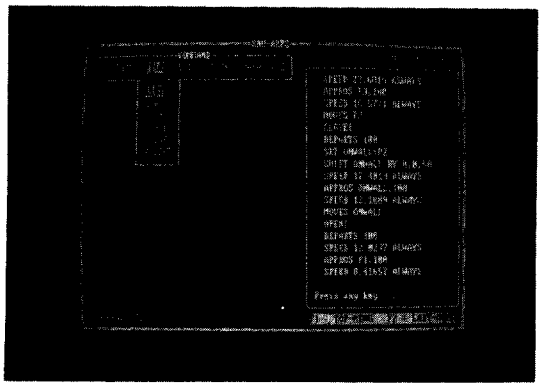

그림 4.4 작성된 프로그램 리스트
Fig. 4.4 Generated Program List

도 한다.

그림 4.3은 앞의 대화에서 보인대로 지붕(ROOF)을 담(WALL)위에 내려 놓은 후, 작업 단계가 아니라, 운동 단계로 명령하는 과정을 보인다. 화면 좌측 하단의 메뉴 윈도우는 이를 반영하여 현재의 명령어군은 'HANDLING'작업의 'MOTION'단계로, 'MOVE'에 관련된 명령어들이 입력될 수 있음을 나타내고 있다. 그림 4.3은 기둥이 놓여 있는 위치 P1에 관절보간 방법으로 이동하려 하자 이것이 위험한 일이라는 메시지를 보이는 상태이다.

위와 같은 과정을 통하여 원하는 작업이 입력되면, 사용자는 명령어 CLOSE를 통하여 입력된 작업을 실제의 로보트 언어로 바꾸고 이를 파일에 저장한다. 여기서 작성된 VAL프로그램 리스트는 아래와 같다.

```
Program 'test. val'
    SPEED 23.68 ALWAYS
    APPROS P3, 100
    SPEED 16.58 ALWAYS
    MOVES P3
    CLOSEI
    DEPARTS 100
    SET ONWALL=P2
    SHIFT ONWALL BY 0, 0, 50
    SPEED 20.72 ALWAYS
    APPROS ONWALL, 100
    SPEED 14.50 ALWAYS
    MOVES ONWALL
    OPENI
    DEPARTS 100
    SPEED 12.02 ALWAYS
    APPROS P1, 100
    SPEED 8.41 ALWAYS
    MOVES P1
    STOP
```

그림 4.4는 위의 VAL프로그램을 보여주는 화면이다. 이 프로그램을 이용하여 실제로 로보트를 구동하는 과정에서는 CAD에서 얻은 데이타가 실제로 로보트를 구동할 만큼 정확하지 못하였으므로, 교시를 통한 몇몇 위치 데이타의 보정과정을 통해 주어진 작업의 실현을 확인할 수 있었으며, 속도의 지정도 만족할 만한 수준이었다.

이와 같은 실험을 통해 SUN-ARPS의 프로그램 합성 기능은 잘 구현 되었으며, 환경 모델의 정보도 프로그램의 진행과 함께 연속적으로 변화하여 사용자의 논리적인 작업 계획을 지원할 수 있음을 확인하였다. 그러나 오프 라인으로 정확한 위치 정보를 구하는 과정은 매우 어려운 일이며, 이러한 과정에 관한 연구가 뒷받침 되어야만 이들 데이타를 기준으로 작업하는 SUN-ARPS의 다른 부분들이 주어진 역할을 제대로 수행할 수 있음을 확인하였으며, 또한 오프라인으로 얻은 위치 및 방향 정보를 이용하기 위해서는 로보트 언어 자체에도 이들 데이타를 받아 들이는 명령어가 새로이 마련되어야 함을 알 수 있었다.

이러한 문제를 교시정보를 이용하여 보완하도록 하였으므로, 주어진 작업에 적합한 프로그램을 작성하는 SUN-ARPS는 실제 로보트 언어를 모르는 많은 사용자들을 위한 훌륭한 보조자의 역할을 할 수 있을 것으로 기대된다.

5. 결 론

본 논문은 작업단계언어의 개념을 이용하여 현실적으로 이용 가능한 프로그래밍 도구로서, 지적인 보조자의 역할을 하는 로보트 자동 프로그래밍 시스템 SUN-ARPS의 개발에 관해 연구하였다. 새로운 로보트 언어라기 보다는 주어진 작업을 수행할 수 있는 실행가능한 로보트 프로그램을 작성하는 도구인 SUN-ARPS에서는 기초적인 작업 단계 명령을 실현할 수 있는 기본적인 기능을 구현하고, 다양한 로보트 언어 시스템에 적용할 수 있는 가능성을 확인하기 위해, PUMA구동을 위한 VAL, SCARA형 로보트 구동을 위한 SNUL-I을 합성하여 실험하고 보완하였다.

이 과정을 통해 SUN-ARPS는 로보트 프로그램

을 용이하게 하고자 했던 기존의 시도에 비해 다음과 같은 장점을 지님을 확인하였다.

- 로보트 프로그램이 일정한 규칙에 의해서만 생성되는 것이 아니라 사용자가 여러 가능성 중에서 선택할 수 있으므로 사용자의 의사가 좀 더 세부적으로 반영된다.
- 사용자가 작업단계언어 수준의 문법조차 익힐 필요가 없다.
- 프로그램시 문법상의 에러 이상의 논리적 에러도 확인되므로 프로그래밍에 관한 지식이 없는 사용자도 쉽게 이용할 수 있다.
- 다양한 설명기능으로 사용자 교육의 효과가 있다.
- 여러 종류의 로보트 및 로보트 언어에 대해 쉽게 확장할 수 있다.

그러나, 현재 SUN-ARPS는 전문적인 의사결정—충돌 회피 경로 선택, 잡는 위치의 선정등—을 매우 단순한 알고리즘으로 처리하고 있으므로 이 부분들에대한 연구 결과를 흡수하여 좀더 지능적으로 처리하도록 보완하고, 그래픽스 인터페이스[20]로써 작성된 프로그램의 실행과정을 시각적으로 확인할 수 있는 기능이 추가되어야 할 것으로 사료된다.

*본 연구는 학술진흥재단(자유과제)의 부분적인 재정지원으로 수행되었음.

참 고 문 헌

[1] Yoram Koren, Robotics for Engineers, McGraw-Hill, 1985.

[2] M.P. Groover et all., Industrial Robotics-Technology, Programming, and applications, McGraw-Hill, 1986.

[3] T. Lozano-Perez, "Robert Programming," Proceedings of the IEEE, vol. 71, no. 7, July, pp. 821-841, 1983.

[4] S. Bonner and K.G. Shin, "A Comparative Study of Robot Languages," IEEE Computer, pp. 82-96. 1982.

[5] C.S.G. Lee et al., Tutorial on Robotics, IEEE Computer Society, 1986.

[6] L.I. Lieberman et al., "AUTOPASS : An Automatic Programming System for Computer Controlled Mechanical Sssembly," IBM J. of Research & Development, vol. 21, no. 4, July, 1977.

[7] T.Lozano-Perez and P.H. Winson, "LAMA : A Language for Auto-Matic Mechanical Assembly," Proc. 5th IJCAI, pp. 710-716, 19778.

[8] R.J. Poppelstone et al., "RAPT-A Language for Describing Assemblies," The Industrial Robot, vol. 5, no. 3, pp. 131-137, 1978.

[9] R.J. Popplestone et al., "An Interpreter for a Language Describing Assemblies," Artificial Intelligence, vol. 14, no. 1, pp. 79-107, 1980.

[10] T. Lozano-Perez et al., "HANDEY : A Robot System that Recognizes, Plans, and Manipulates," Proc. IEEE Int'l Conf. on Robotics & Automation, pp. 843-849, 1987.

[11] User's Guide to VAL, Unimation Inc., June, 1980.

[12] 이기동, "SCRA형 로보트의 프로그래밍 언어 구성에 관한 연구," 서울대학교 공학석사 학위 논문, 1985.

[13] Golden Common LISP, Gold Hill Computers Inc., 1985.

[14] AutoLisp Programmer's Reference, AutoDesk Inc., 1986.

[15] D. Raker and H. Rice, Inide SutoCad, New Riders Publishing, 1985.

[16] Knowledge Craft Reference Manual, Carnegie Grop Inc., 1986.

[17] T. Lozano-Perez, -A Simple Motion Planning Algorithm for General Robot Manipulators," Proc. 5th AAAI, pp. 626-631, 1986.

[18] J. Khouri, "An Efficient Alorithm for Shortest Path in Three Dimensions with Polyhedral Obstacles," Proc. American Control Conference, pp. 161-165, 1985.

[19] T. Lozano-Perez et al., "An Algorithum for Planning Collision free Paths Among Polyhedral Obstacles," Com. of the ACM, vol. 22, no. 10, pp. 560-570, 1979.

[20] A. Naylor et al., "PROGRESS-A Graphical Robot Programming System," Proc. IEEE Int'l Conf. Robotics and Automation, pp. 1282-1291, 1987.