조립작업을 위한 로봇의 지능적 작업 수준 계획 시스템
An Intelligent Task-level Planning System for Robotic Assembly

서일홍*, 정재훈*, 조상규*, 김형욱*, 황병훈*, 김진오†

*한양대학교 전자공학과 지능제어및 로보틱스 연구실
†삼성전자 생산기술센타 자동화연구소

Abstract

In this paper, we propose a task-level assembly planning system. In practical assembly task, there are a lot of problems to cope with. One of those is selecting mid-points via which robot moves to a goal position. To solve the problem, conventional approaches utilize several methods composed of teaching every points to pass through and generating optimal points to avoid obstacles. But they have several drawbacks as like time-consuming property, difficulty to control. For this reason, more efficient assembly system is developed on the basis of on-line task-level command interpreter. Thus, a real-time assembly task can be successfully performed by the proposed task-level planning system.

1. 서론

로봇을 이용한 생산공정의 자동화는 현재 산업 현장에서 핵심 기술로 급부상되고 있다. 일반적으로 컴퓨터 제어에 의한 조립 자동화 시스템의 경우 완성품의 구조가 복잡해짐에 따라 조립 자동화 시스템에 대한 필요성이 점차 증대되고 있다. 현존하는 대부분의 로봇제어언어는 로봇을 중심으로 작업을 교시하기에 편리하게 만들어져 있다. 그러므로 전문 시스템 개발자가 아닌 일선 작업자가 쉽게 프로그래밍하기 위해서 작업 대상물을 중심으로 작업을 기술, 프로그램할 수 있는 Task-level 제어언어가 필요하다. 즉 이는 로봇 작업의 시작과 종료상태, 그리고 작업대상물의 중간상태를 연속적으로 프로그래밍하므로써 로봇의 기구학이나 위치와 경로를 사용자가 따로 지정할 필요가 없는 새로운 형태의 언어이다. 1977 년에 AUTOPASS 를 시초로해서 Task-level 제어언어에 대한 끊임없는 연구가 진행되어 왔으나 언어 자체의 복잡성과 적용할 시스템의 언어와의 비호환성 등으로 인해 Task-level 제어언어의 구현을 위해서는 아직도 많은 어려움이 남아 있다[1]-[6].

본 논문에서는 조립 자동화를 위해 필요한 Task-level 제어명령어들을 정의하고 이를 SCARA 로봇제어명령어로 변환하여 실시간 수행할 수 있도록 Task-level 조립 계획 시스템을 제안하고자 한다.

2. Task-level 조립 계획 시스템의 설계

2.1 Task-level 제어언어의 정의

Task-level 명령어는 부품간의 결합관계를 내재하여야 한다. Ko 와 Lee 는 부품간의 관계를 Against, Fits, Tight-fits, Contact 등의 네 가지 형태로 분리하였고 또 Hommem de Mello 와 Sanderson 은 부품을 나타내는 정보와 연결을 나타내는 정보의 집합으로 구성되는 "관계 모델 그래프"를 사용하였다.[7]-[9] 본 논문에서는 이러한 기존의 연결 관계 표현방법과 물체의 기본적인 이동방법을 토대로 아래와 같은 Task-level 제어언어들을 정의하였다. 위치 정보를 이용한 Pick, Place 와 위치와 힘 정보를 이용한 Attach, Insert, Screw 로 정의한다. 이렇게 정의한 Task-level 제어언어들은 실질적인 적용을 위해 SCARA 로봇 환경에 맞도록 다음과 같이 정의할 수 있다.

pick <object>
: SCARA 로봇의 특성상 End-effector 를 최고점까지 끌어 올리므로써 미리 갖다놓은 object 와 충돌을 회피할 수 있다고 가정한다.

place <object> to <location>
: <location> 까지 z 축의 위치는 바꾸지 않고 이동해서 End-effector 를 수직으로 내린다.

attach <object1> to <object2>
: <object2> 방향으로의 접촉힘을 일정한 범위로 유지하면서 목표점을 만족하면 정지하고 <object1>을 놓는다.

insert <object1> into <object2>
: 삽입방향으로의 접촉힘을 일정한 범위로 유지하면서 목표지점에 도달하면 정지하고 <object1>을 놓는다.

위에서 정의한 place, attach, insert 는 <object>를 pick 한 상태를 전제로 한다. 공통적으로 Pick 상태로 계산된 경로를 따라 이동하고 각각의 Task 를 수행한 후 계산된 경로의 반대순서로 그 Task 전의 위치로 돌아온다. 이와 같이 정의된 Task-level 제어언어를 이용하여 조립작업을 기술할 때에는 각각의 조립부품에 대해서 Pick 과 나머지 3 개의 결합 Task 중 하나에 의해서 표현되어 진다.

2.2 전체 시스템의 구성

제안하고자 하는 Task-level 조립 계획 시스템의 구조는 그림 1 에서와 같이 동작 특성에 따라 크게 두 부분으로 나눌 수 있다. 첫번째는 On-line Task-level 제어언어 번역기 부분으로, 주어진 Task-level 명령어를 해석하여 실시간으로 로봇 제어 명령어들로 바꾸어 로봇 시스템에 전달하는 역할을 수행한다. 두번째는 Off-line 완성품 데이타베이스(Target Data Base) 생성기 부분으로, 사용자가 특정완성품에 대해 완성품 데이타베이스 생성기로 조립순서와 완성품을 이루기 위한 Task-level 명령어를 Graphical 한 방법으로 지정하고 각 Task-level 명령어에 대해서 구체적인 조립동작을 만들고 이를 완성품 데이타베이스에 저장하게 된다. 여기서 조립순서는 특정 완성품에 대해 여러 개가 존재할 수 있다.

전체 시스템의 구성도와 각 부분의 역할은 다음과 같다.

① TLP (Task-Level Program) : 사용자가 Programming 하거나 자동생성한다..

② TDB (Target Data Base) : 완성품에 대한 가능한 조립순서별 Task-level 제어명령 및 이를 수행할 로봇제어명령들이 저장된다.

③ DBB(Data Base Browser) : 주어진 Task-level 프로그램으로부터 작업순서를 추출해 내고, 주어진 작업 순서가 TDB 에 있는지를 검색하여 해당되는 조립 완성품의 이름을 TLI 에 넘겨준다.

④ TLI (Task-Level Command Interpreter) : 주어진 Task-level 프로그램의 각 명령어에 대한 로봇제어 명령어들을 DBB 와 TDB 를 참조하여 생성한다.

⑤ ODB (Object DataBase) : 조립부품을 이루는 기본 부품들에 대한 3 차원 형상정보와 기본부품들간의 연결관계를 포함한다.

⑥ EDB (Element DataBase) : 각 부품들을 구성하는 기하학적인 모양들의 형상정보들이 저장되어 있다.

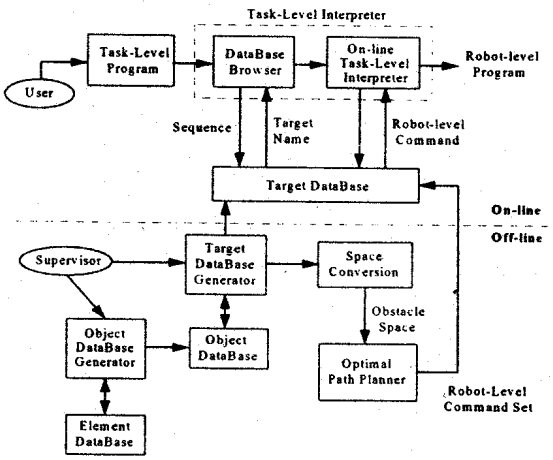


그림 1. 전체 구성도

⑦ 공간 변환(Space Conversion) : ODB 에서 표현된 각 부품의 3 차원 형상을 바탕으로 하여 Configuration Space 에서 다른 부품과 겹치게 되는 부분을 장애물로 간주한다.

⑧ 최적 경로 계획(Optimal Path Planner) : SCARA 로봇에서 제공하는 로봇제어명령을 이용하여 초기 위치에서 목표지점까지 가기 위한 경로를 생성한다.

3. On-line Task-level 제어언어 번역기의 설계

Task-level 제어 언어 번역기의 전체흐름은 그림 2 과 같다. 사용자가 조립작업을 Task-level 제어언어로 기술하면 DBB 에서 입력된 프로그램으로 부터 조립순서를 추출하고, TDB 에 현재의 조립순서에 해당하는 정보가 있는지를 검색한다. TLI 는 DBB 로부터 받은 완성품의 이름과 작업순서에 해당하는 로봇제어 명령어들을 TDB 로부터 가져와서 로봇제어 프로그램을 생성한다.

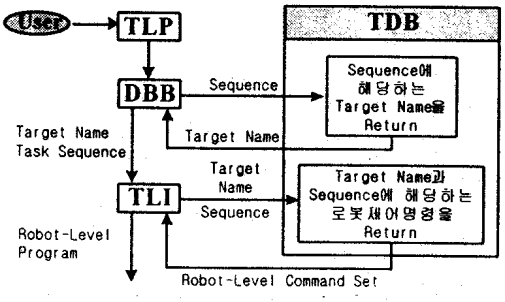


그림 2. 제어언어 번역기의 전체 흐름

Task-level 제어언어로 작성한 프로그램을 줄단위로 번역하여 이를 수행하기 위한 C 언어 프로그램을 출력하고, 실제 컴파일은 C 컴파일러에 의해 수행하는 인터프리터의 형태를 취했다. 그림 3 와 같이 LEX, YACC 을 이용하여 Task-level 제어언어 번역기를 구현하기 위한 어휘분석기와 구문분석기를 생성하였다.

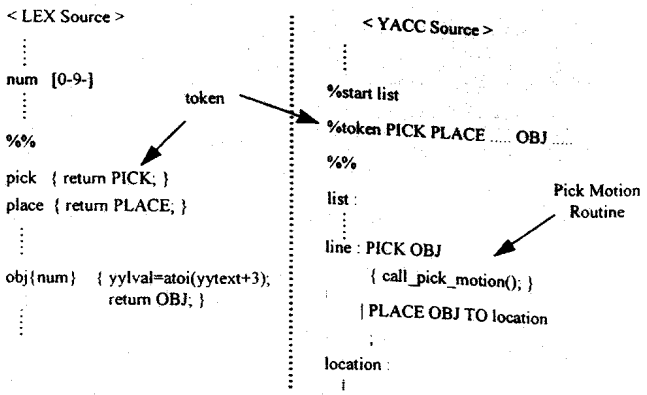


그림 3. LEX 와 YACC 을 이용한 Interpreter

4. 완성품 데이타베이스의 Off-Line 구성

4.1 조립부품의 모델링 및 데이타베이스의 구성

본 논문에서는 조립품에 대한 3 차원 형상정보를 표현하기 위하여 그림 4 과 같이 각 조립부품을 이루는 기본부품의 Parameter 와 연결관계를 데이타 베이스로 저장하기로 한다.

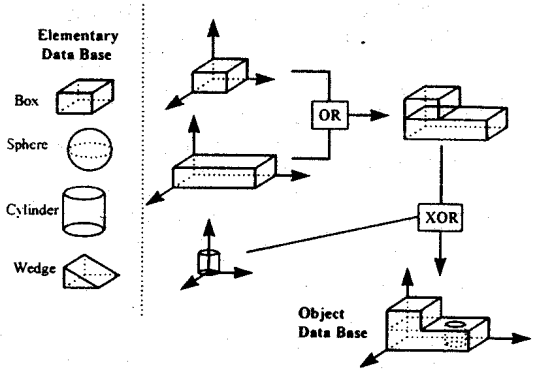


그림 4. 조립물체의 3 차원 형상정보 표현

위와 같이 조립부품의 형상정보가 표현되면 조립부품의 좌표계를 기준으로 조립부품의 형상정보와 조립부품을 이루는 기본부품 간의 관계정보 및 Grasp 위치, 자세 등을 저장한다. 조립부품의 정보를 저장하는 데이타베이스의 구조는 다음과 같다.

```
struct data_base {
    char name[ ];              /* 물체의 이름 */
    char cur_loc[ ];           /* 물체의 현재위치 */
    double grasp_point[3];     /* 물체를 잡기 위한 위치 */
    double grasp_angle;        /* 4 축의 각도 */
      struct item {
          int current_object;  /* 기본부품의 종류 */
          double length;       /* 물체의 길이 */
          double width;        /* 물체의 폭 */
          double height;       /* 물체의 높이 */
          double radius;       /* 물체의 반지름 */
          double dx, dy, dz;   /* 좌표계 원점의 offset */
          int connect;         /* 연결방법 */
      } feature[ ];            /* 물체의 형상정보 */
} object[ ];
```

여기서 현재위치는 조립부품이 공급되는 Feeder 의 위치를 나타낸다. 또한, 물체를 잡기 위한 자세는 End-effector 의 중심위치와 4 축의 각도로써 표시하였고, 형상정보를 나타내는 feature 는 조립부품을 이루는 각 기본부품의 형상정보 및 그들간의 연결관계(OR, XOR)를 표현하도록 구성되어 있다.

4.2 완성품 데이타베이스의 생성

조립부품 데이타베이스가 만들어지면 각 조립부품의 형상정보를 이용하여 완성품 데이타베이스(Target DataBase)를 생성한다. 사용자는 완성품 데이타베이스 생성기를 이용하여 각 조립작업의 순서와 두 부품의 결합위치를 지정하고, 조립작업을 위해 필요한 경로들을 구하여 로봇제어 명령어들로 표현한다. 이러한 정보들은 완성품 데이타베이스에 저장되어서 Task-level 제어언어 번역기에서 참조된다. 완성품의 정보를 저장하는 데이타베이스의 구조는 다음과 같다.

```
struct target_data {
    char target_name[ ];          /* 완성품의 이름 */
    int connect_num;              /* 연결횟수 */
    char *sequence[ ];            /* 조립순서 */
        struct con {
            char task[ ];         /* Task 명령의 이름 */
            char grasped[ ];      /* 이동할 부품의 이름 */
            int command_num;      /* 로봇제어명령의 갯수 */
            char *command[ ];     /* 로봇제어명령 */
        } connect[ ];             /* 완성품 연결정보 */
} target[ ];
```

연결횟수는 조립작업의 결합의 횟수를 나타낸다. 조립순서에는 이동할 대상부품의 이름들을 명시하였다.

4.3 Configuration Space 상에서의 경로계획

4.3.1 장애물 공간의 생성

로보트가 조립부품을 잡고 움직일 때 다른 부품과 부딪치지 않도록 경로를 계획하려면, 조립부품 데이타베이스에 저장된 각 부품의 3 차원 정보를 토대로 장애물 공간을 구해서 이 영역에 걸리지 않도록 경로계획을 해야 한다. 또한, 로보트의 End-effector 도 부품과 걸릴 수 있기 때문에 로봇의 End-effector 와 잡고 있는 부품을 하나의 물체로 간주하여 장애물 공간을 구해야 한다.

SCARA 로봇은 1 축의 각도 θ_1 과 2 축의 각도 θ_2 에 의해서 X 축과 Y 축이 결정된다는 점을 착안하면, 모든 관절각에 대해서 검색하지 않고도 다음과 같이 장애물 공간을 구할 수 있다.

< 장애물 공간을 구하는 알고리즘>

STEP1 : 두 조립부품이 결합될 때의 자세에 의해서 4 축각도 θ_4 의 값을 결정하여 고정한다.

STEP2 : 각 조립부품의 형상정보를 이용하여 각 조립부품을 포함하는 최소육면체를 구한다.

STEP3 : STEP2 에서 얻어진 최소육면체의 높이에 의해서 3 축각도 θ_3 의 범위를 정한다.

STEP4 : 각 부품의 최소육면체를 2 차원 X-Y 평면에 사영한다.

STEP5 : 1 축각도 θ_1 과 2 축각도 θ_2 를 미소단위량씩 변화시켜서 로봇이 잡고 있는 물체의 사영된 도형이 다른 부품의 사영된 도형에 걸리면 Z 축의 범위 내에서 이동시켜서 다른 부품에 걸리게 되는 경우 그 때의 Joint Angle 을 장애물 공간에 저장한다.

4.3.2 최적 경로 계획

Pick 에 의해서 조립부품을 집어 올렸을 때의 로보트의 관절각을 초기위치로, 조립이 이루어졌을 때의 로보트의 관절각을 목표지점으로 하여 이를 연결하는 최적의 경로를 계산한다. 최적의 경로를 구하기 위해서 장애물에 걸리지 않는 것을 Constraint 로 하고 Cost Function 으로는 가감속 횟수와 거리를 두었다. 매번 가속과 감속을 하면 부드러운 로봇 Motion 을 생성하지 못하고 시간이 많이 걸리므로 가감속의 횟수를 줄이도록 하였고, 또한 목표지점에 도달할 때까지의 거리를 최소가 되도록 경로 계획을 하였다. 본 논문에서는 비교적 적은 계산으로 원하는 경로를 얻을 수 있는 알고리즘을 제안한다.

1) 초기위치(Ps)에서 목표지점(Pg)까지 연결하는 직선이 장애물에 걸리는 지를 검사한다.
2) 장애물에 걸릴 경우에는 그림 5 와 같이 장애물까지 가장 먼 거리(d_{max})만큼 미리 정해 놓은 기본 방향을 따라 검색하여 장애물에 걸리는지를 조사한다.
3) 장애물에 걸리지 않는 방향 중에서 목표지점까지의 거리가 가장 작은 쪽을 선택한다.
4) 현재 위치에서 정해진 방향을 연결하는 직선상의 지점 중에 목표지점까지의 거리가 가장 작은 지점을 다음 경유점으로 선택한다.

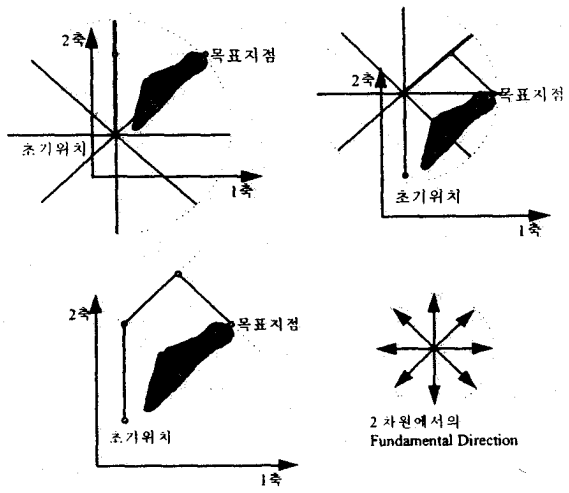


그림 5. 가감속 횟수의 최적화

여기서, 경유점 주변에 Transient 구간을 두어서 이 구간이 장애물에 걸리지 않도록 경유점을 선택해야 한다. 이렇게 목표지점까지의 경로를 구하면 기본방향들의 조합으로 경로가 이루어지게 된다. 거리의 최적화에서도 경유점 주변에서의 감속 구간이 장애물에 걸리지 않도록 경유점을 잡아야 한다.

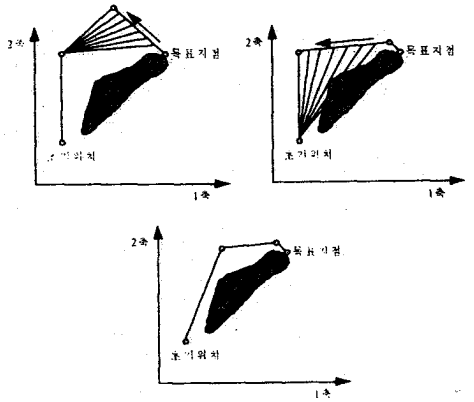


그림 6. 거리의 최적화

4.4 로봇제어명령으로의 변환

이상과 같은 과정을 거쳐서 경로를 구하면 가감속의 횟수가 적으면서 거리가 최소인 경로를 얻을 수 있다. 경로계획시에 Joint Space에서 직선 경로가 장애물에 걸리지 않도록 하였기 때문에 Joint Space에서 직선으로 움직여야 한다. 로봇의 기본 Motion에는 Joint와 Linear Motion이 있는데 Joint Motion은 속도가 빠르다는 장점이 있지만 로봇의 End-effector의 중간경로를 지정하지 않기 때문에 장애물과의 충돌 여부를 예측할 수 없다. Linear Motion은 End-effector의 경로를 직선으로 보장할 수 있지만 계산량이 많기 때문에 속도에 제한이 있다. 조립작업은 빠른 반복작업을 요구하기 때문에 Joint Motion을 사용하여 시간이 적게 걸리도록 하면서 이동경로가 직선을 이루도록 다음과 같은 방법을 제시한다.

각 축이 현재 위치에서 목표지점까지 최고속도로 움직일 때 걸리는 시간이 같으면 각축의 Motion이 동시에 끝나므로 경로를 직선으로 보장할 수 있다. 하지만 Motion이 동시에 끝나지 않는 경우에는 그림 7(b)와 같이 Joint Space 상에서 중간경로를 직선으로 보장할 수 없다.

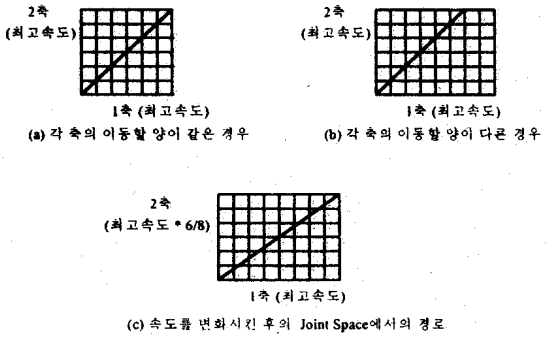


그림 7. Joint Space 에서의 경로

그러므로 이동할양이 적은 축의 속도를 줄여서 모든 축이 동시에 끝나도록 해주면 그림 7(c)와 같이 직선경로를 형성할 수 있다. 따라서, 기존의 Joint Motion을 부분 수정하여 drive (θ 1, v1) (θ 2, v2) (θ 3, v3) (θ 4, v4)와 같이 각 축에 대해서 변위각도와 속도를 지정하여 주면 Joint Space에서 직선경로를 형성할 수 있다.

5. 실험 및 결과

본 논문에서 제안한 Task-level 조립 계획 시스템을 구현하기 위하여 다음과 같은 실험환경을 구성하였다. 실험에 사용된 로봇은 4축 SCARA 로봇이며 로봇 제어기의 하드웨어 구조는 VME-bus를 기본으로 두 대의 32-bit CPU Board (CPU30)를 사용하였다. 또한 실시간 운영체제인 VxWorks를 사용하였고 Off-Line 데이타베이스 구축을 위하여 UNIX 상의 X-window와 MOTIF를 사용하였다. 자동조립시스템의 성능분석을 위해 그림 8와 같은 조립부품을 사용하여 간단한 조립작업을 실험하였다. [10]-[11]

시스템의 실시간 특성을 위해서는 조립하려는 완성품을 구성하는 각 부품에 대한 모델링과 각각의 Task-level 명령어를 수행하기 위해 필요한 최적의 경로 생성 및 구해진 경로를 여러 개의 로봇제어명령으로 표현하는 과정을 통해 완성품에 관련된 정보로서 완성품 데이타베이스를 미리 구축하여야 한다.

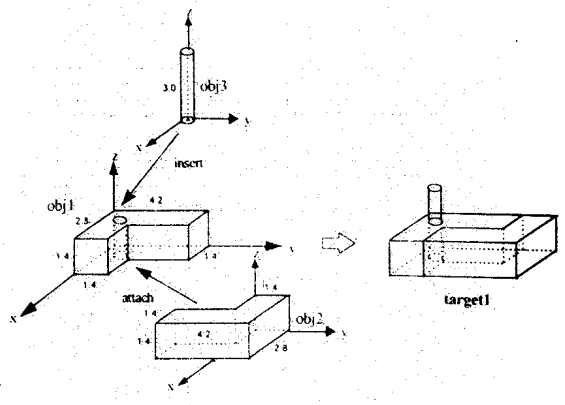


그림 8. 조립 대상 부품 및 조립작업

Target을 이루는 그림 8의 부품들중 obj1에 대한 CAD Model은 그림 9와 같다. 각 부품의 모델링에서는 부품의 3차원 형상, 물체를 잡을 때의 위치와 자세를 조립부품 데이타베이스로 구축한다.

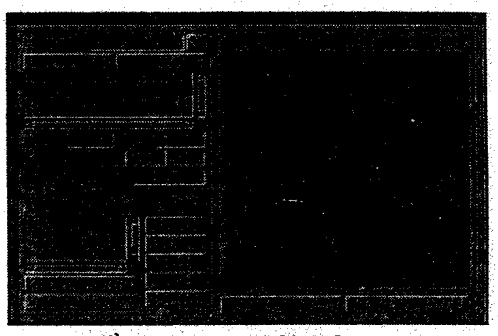


그림 9. Graphical ODB Generator

그림 10에서처럼 완성품 데이타베이스에는 각 완성품을 이루는 조립부품들간의 관계를 표현하는 Task-level 제어명령 및 이에 해당하는 로봇제어명령들이 저장되어 있다.

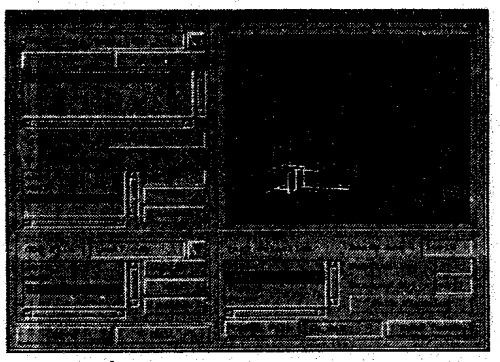
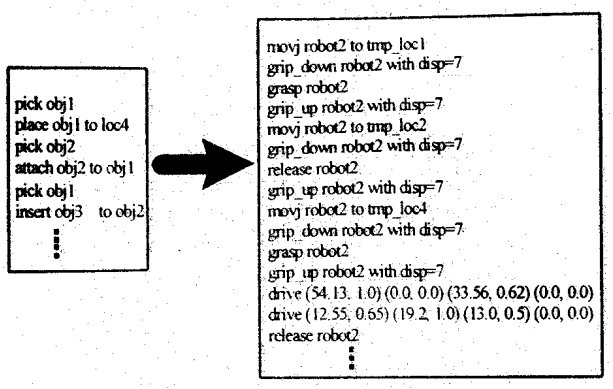


그림 10. Graphical TDB Generator

여기서 완성품 데이타베이스를 구성하기 위한 각 조립부품 사이의 결합경로는 앞에서 제안한 알고리즘에 의해 자동생성되어 로봇제어명령으로 표현된다.

완성품 데이타베이스를 만들때 Graphical하게 조립 순서를 지정하므로 이를 바탕으로 Task-level 프로그램을 자동생성할 수 있으며 따로 그 순서를 기억할 필요가 없고 사용자가 일일이 프로그램을 할 필요가 없다. 그 Task-level 프로그램과 추출된 조립순서를 바탕으로 변환된 로보트제어 명령어들은 아래와 같다.



< Task-level Program > < Robot-level Program >

그림 11에서는 사용자가 조립완성품에 관한 정보를 미리 완성품 데이타베이스에 저장하고 원하는 조립작업을 Task-

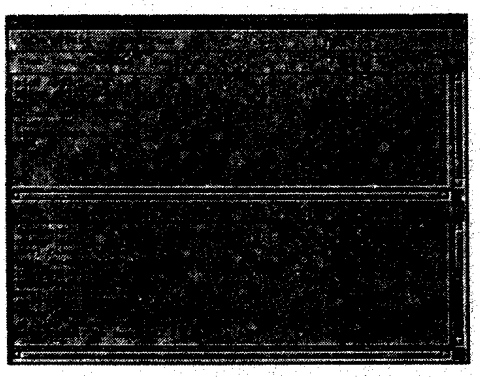


그림 11. Graphical TLI Generator

level 명령어를 이용하여 기술하면 Task-level 제어언어 번역기에서는 조립순서를 추출하고 그 순서대로 조립완성품에 대한 로봇 제어명령어들을 가져옴으로써 주어진 작업을 실시간으로 수행할 수 있도록 하였다.

6. 결론 및 추후 과제

본 연구에서는 기존의 로봇 중심의 언어가 아닌 물체 중심으로 작업을 기술할 수 있는 Task-level 제어언어 개발에 중점을 두었고, 또한 이를 효과적으로 로봇 제어언어로 번역하기 위한 Task-level 조립계획 시스템의 구현방법을 제안하였다. 특히 각각의 완성품을 이루기 위한 로봇제어언어를 완성품 데이타 베이스로 구축하여 제안된 시스템이 실시간 특성을 가지도록 하였다. 또한 물체의 3 차원 기하학적 특성의 데이타베이스화, 로봇의 동역학을 고려한 최적의 경로계획 등의 연구가 수반되었다. 본 연구를 통하여 기존의 로봇 중심 언어로 기술하기 불편했던 작업을 비교적 간단한 몇 가지 Task-level 제어언어를 통해 구현할 수 있었으며 실질적인 실험을 통해 제안된 시스템의 효율성을 입증하였다. 추후과제로는 각 조립물체에 대한 정보와 완성품에 대한 정보로 부터 완성품 데이타베이스를 자동으로 생성하는 방법과 Sensor-Based Task-level Language의 정의에 대한 연구가 필요하다.

Reference

[1] L.I.Lieberman and M.A.Wesley, "AUTOPASS : An Automatic Programming System for Computer Controlled Mechanical Assembly," *IBM J Res Develop*, Vol. 21, No. 4, pp. 321-333, 1977.

[2] A. Stentz, "Optimal and Efficient Path Planning for Partially-Known Environments," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 3310-3317, May, 1994.

[3] E. Coste-Maniere, B. Espiau and E. Rutten, "A Task-Level Robot Programming Language and its Reactive Execution," *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2751-2756, July, 1992.

[4] E. Cervera, Angel P. del Pobil, E. Marta and M. A. Serna, "A Sensor-Based Approach for Motion in Contact in Task Planning," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 468-473, May, 1995.

[5] J.P Thoms and P.N Nissanke, "A Graph-based formalism for Modelling Assembly Tasks," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 1296-1301, May, 1995.

[6] I. Kamon and E. Rivlin, "Sensory Based Motion Planning with Global Proofs," *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 435-440, August, 1995

[7] D. Halperin and R. H. Wilson, "Assembly Partitioning along Simple Paths: the Case of Multiple Translations," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 1585-1592, May, 1995.

[8] 한국과학재단, "지능형 조립시스템을 위한 기초연구", 1993.

[9] J. Liu, "Assembly Planning Based on a Task Grammar Augmented with Qualitative Heuristic Knowledge," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 962-969, May, 1995.

[10] Wind River Systems, "VxWorks Programmer.s Guide," 1990.

[11] SUN Microsystems, "Programmer Utilities and Libraries," pp. 205-262, 1988.