자율이동로봇을 위한 반사층의 실시간 주행제어구조

김형진* · 전성용* · 손원종* · 홍금식†

(2006 년 1 월 3 일 접수, 2006 년 9 월 7 일 심사완료)

Navigation Control Architecture of the Reactive Layer for Autonomous Mobile Robots

Hyung-Jin Kim, Sung-Yong Jeon, Won-Jong Sohn and Keum-Shik Hong

Key Words : Mobile Robot(이동로봇), Navigation Control Architecture(주행제어구조), Action(단위행위), Real-time Operating System(실시간 운영체제), RTAI

Abstract

In a hybrid three-layer control architecture(deliberative, sequencing, and reflexive), the lowest reflexive layer consists of resources, actions, an action coordinator, and motion controllers. Because the execution of individual components in the reflexive layer should be done in real-time, each component has to be simple and, due to this reason, the Linux-RTAI(Real-Time Application Interface for Linux) has been used as an operating system. In this paper, a navigation control architecture, which combines the components in the reflexive layer and the navigation-related modules in the sequencing layer, is proposed. And then, as basic components, four actions(*Goto, Avoid, Move*, and *EmergencyStop*) are designed. Experimental results confirm the effectiveness of the proposed architecture and the performance of individual associated actions.

1. 서 론

로봇의 자동화 성능에 대한 인간의 기대가 날로 증가함에 따라 로봇의 하드웨어는 점점 복잡해지며 다양한 센서들의 사용이 요구되고 있다. 또한 로봇에 자율기능을 부여하기 위해 사용되는 제어 알고리즘이 복잡해짐에 따라, 로봇의 자율주행을 위한 새로운 개념의 제어구조 도입이 필요하다.

제어구조에 관한 연구는 80 년대 말부터 개발이 시작된 계층제어구조와 Brooks의 subsumption architect-ture[6~8]와 Arkin의 motor schema[3~4]로 대표되는 행위기반제어구조(behavior-based architecture)로 시작되었다. 계층제어구조는 미리 정해진 계획에 따라 순서대로 로봇의 작업을 수행하도록 설계된 제어구조로서 산업용 로봇의 자동화에 널리 적용되고 있다.[9] 행위기반제어구조는 센서의 특정 정보를 자극으로 보고 이에 반응하는 로봇의 행위들의 조합으로 로봇의 움직임을 생성하는 제어구조이다. 여러 행위를 효율적으로 조합하기 위해서 Arkin의 경쟁조정자(competitive coordinator)와 Brooks의 협조조정자(cooperative coordinator)가 사용된다. 경쟁조정자는 여러 개의 후보 행위 중에서 가장 적절한 하나의 단위행위를 선택하여 로봇을 구동시키는 것이고, 협조조정자는 필요한 행위들을 조합하여 새로운 단위행위로서 로봇을 구동시키는 것을 말한다.

하드웨어가 복잡해지고 요구성능이 많아짐에 따라 계층제어구조와 행위기반제어구조, 경쟁조정자와 협조조정자가 혼합된 형태의 제어구조가 연구되고 있다. 로봇의 작업계획이나 환경지도작성 등 로봇을 직접 구동시키는 명령은 아니지만 복잡한 계산이 필요한 부분은 상위계층에서 계층제어구조로 구현되고(deliberative control), 주행, 장애물회피, 벽면추종, 목표물 추적 등 실제로 로봇을 움직이

† 책임저자, 회원, 부산대학교 기계공학부
E-mail : kshong@pusan.ac.kr
TEL : (051)510-2454 FAX : (051)514-0685
* 회원, 부산대학교 지능기계공학과

는데 사용되는 제어알고리즘은 행위기반제어로 구현되어 하위계층에 둔다(reactive control).

군사용 무인차량에 적용할 목적으로 NIST에서 개발한 4D/RCS(real-time control system)[2]와 화성탐사로봇에 적용하기 위해 NASA와 California Institute of Technology, Carnegie Mellon University가 공동으로 개발한 제어구조인 CLARAty(Coupled Layer Architecture for Robotics Autonomy)[18]는 대표적인 혼합형구조라 할 수 있다. 그러나, 이들은 실외용으로서 안정성 확보가 주된 목적으로 설계되었기 때문에 구조가 복잡하고 데이터베이스의 성능에 크게 의존하고 있는 형태이다.

독일 Fraunhofer IPA에서 노인 및 장애인을 위한 Care-O-bot에 적용된 제어구조[10]는 주행과 로봇팔의 조작을 동시에 제어하기 때문에 계산량이 많고 로봇의 움직임이 느린 단점이 있다.

스웨덴 왕립기술원에서 개발하여 박물관 안내로봇에 적용된 제어구조인 BERRA(Behavior-based Robot Research Architecture)[16]는 현재 개발되는 대부분의 주행로봇이 이 제어구조를 따를 만큼 보편적인 형태이나 각 계층의 역할이 모호하고 행위기반제어구조의 특성을 잘 구현하지 못하고 있다.

한국에서는 한국과학기술연구원(KIST)에서 개발한 서비스로봇에 적용된 Tripodal Schematic Architecture[13]와 한국과학기술원(KAIST)에서 반자율주행 휠체어의 제어구조[14]를 개발한 사례가 있다. Tripodal Schematic Architecture는 전체적 구조와 정보의 흐름을 표현하는 layered functionality diagram과 컴포넌트의 구현을 위해 기능별로 프로그래밍하는 class diagram, petri-net을 이용하여 명령 실행 및 오류복구를 위한 configuration diagram으로 구성되어 있다. 이는 재사용성 및 확장성 부분에서 향상되었으나 단순한 작업을 위해 지나치게 복잡한 구조로 설계되어 개발시간이 길고, 정보의 흐름이 복잡하여 시간지연이 많은 단점이 있다. 자동휠체어에 적용된 제어구조는 반자율주행을 위해 설계되었기 때문에 제어알고리즘을 적용하기에 불편하고 운영체제의 성능에 많이 의존하는 경향이 있다.

이 외에도 해저탐사용 로봇[18]에 적용되는 제어구조 등 목적에 따라 다양하게 연구되고 있으나, 대부분의 경우 로봇이 제어구조 하에서 자율적으로 주행하기 위해서는 동작하면서 일어나는 모든 상황을 미리 예측하여 데이터베이스를 구성해 놓아야 한다. 그렇지 않으면 모든 상황에 대해 적절한 행위를 취할 수 없게 되어 있다.

위에 제시된 다수의 제어구조에서 반사층의 제어구조를 살펴보면 각각의 요소들을 객체지향 접근법으로 모델링하여 재사용성과 유연성 측면에서는 우수하나 데이터의 흐름관계가 불명확하고, 다양한 센서시스템을 사용한 알고리즘 구현에 초점이 맞춰져 있어 반사층의 특성을 제대로 살리지 못할 뿐 아니라 실시간 구현에 대한 언급도 부족하다.

따라서 본 논문에서는 위의 단점들의 보완 및 이동로봇이 자율기능을 갖추기 위한 기반으로서, 자율이동로봇의 실시간 성능을 향상시켜 이동성 및 제어 안정성을 확보하기 위한 주행제어구조의 메커니즘을 제안한다. 또한 주행제어구조의 핵심으로서 자율주행을 위한 다양한 제어알고리즘이 적용될 반사층의 단위행위를 기능별로 설계한다.

주행제어구조는 제어구조에서 주행부분의 제어알고리즘을 실시간으로 구현하기 위한 목적으로 실시간 운영체제인 Linux-RTAI환경에서 실행되도록 설계한다.

단위행위는 위치제어에 필요한 *Goto*, 움직이는 장애물의 인식과 회피 기능을 위한 *Avoid*, 충돌과 같은 긴급상황에 대처할 *EmergencyStop*과, 속도제어를 위한 *Move*로 구성되며, 로봇의 임무수행과 주행제어구조의 이동 효율성을 증대시키기 위해 가능한 단순한 알고리즘으로 적용된다. 또한 각 단위행위들을 효율적으로 조합, 조정하기 위해 설계된 행위조정자의 메커니즘에 관하여 설명한다.

2. 자율이동로봇

2.1 하드웨어 구성

본 연구에서 사용된 로봇의 규격은 가로, 세로 길이가 각각 680 mm, 높이는 850 mm 이고 총 중량은 약 60 kg이고 외관은 Fig. 1과 같다. 센서 및 주변장치로는 LRF(Laser Ranger Finder), 적외선 스캐너, 초음파센서, 범퍼, 엔코더, 모니터, 터치스크린, 스피커 등이 사용된다.

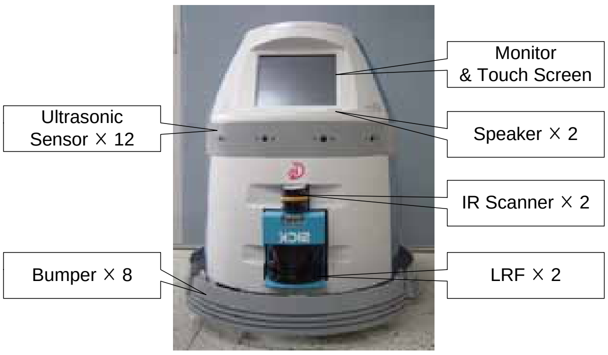


Fig. 1 The used mobile robot developed by the Center for Intelligent Robotics, KIST

2.2 제어구조

이동로봇이 자율기능을 지니기 위해서는 다양한 하드웨어와 소프트웨어를 통합하여 제어할 수 있는 틀이 있어야 한다. 또한 적절한 운영체제를 고려해야 하는데, 이러한 틀로서 제공되는 개념이 바로 제어구조이다.

Fig. 2는 현재 연구가 진행되고 있는 제어구조를 일반화한 그림이다. 이는 인정층(認定層, deliberative layer)과 결순층(決順層, sequencing layer), 반사층(反射層, reflexive layer)의 세 개의 계층으로 나뉘어져 있으며 반사제어와 예정제어가 다양한 제어알고리즘과 함께 혼합된 혼합 인정/반사 제어구조라 한다.

인정층은 인간으로부터 주어지는 자연어 명령을 인지하여 로봇이 실행할 수 있는 기계어 명령으로 바꾸는 역할을 담당하며, 로봇이 감당해야 할 임무를 주행부분과 조작부분으로 나누어 작업을 계획하는 역할을 한다.

결순층은 인정층에서 생성된 작업계획을 실행하기 위해 센서로부터 획득된 정보를 가공하여 순차적인 정보를 생성하는 계층이다. 자율주행모듈(Navigation_Module)에서는 자기위치를 인식하고 환경지도를 작성하여 이동해야 할 경로를 생성하는 역할을 한다. 반사층에서 실행할 수 있도록 Process_Supervisor가 명령을 전달한다.

마지막으로 반사층은 상위계층에서 내려오는 명령을 실행하고 임의의 환경에서 장애물이 나타났을 때 로봇의 안전을 보장할 뿐만 아니라, 인간에게 미치는 위험요소를 줄이기 위해 실시간으로 대응한다. 자연스런 움직임을 생성하기 위한 다수의 단위행위(Action)들과 행위조정자(Action_Coordinator)가 사용된다.

2.3 운영체제

제안된 주행제어구조를 위한 운영체제는 실시간 운영체제로서 RedHat Linux 9.0에 RTAI 3.1을 패치하여 사용한다. 실시간 운영체제로서의 RTAI는 interrupt handler를 가지고 있다. 또한 모든 Linux 응용 프로그램 및 기능의 사용이 가능하고 표준 리눅스 드라이버를 그대로 사용할 수 있다는 것이 특징이다.[9]

3. 반사층의 주행제어구조

주행제어구조는 이동로봇의 제어구조에서 결순층과 반사층의 컴포넌트 중에서 주행에 관련된 컴포넌트들을 하나의 제어구조로 요약된 것을 말한다.

본 절에서는 자율주행을 위한 시스템의 틀로서 “주행제어구조”를 설계하고 반사층의 각 컴포넌트의 역할을 기술한다. 주행제어구조에서 실시간성이 필요한 컴포넌트는 kernel space, 그렇지 않은 컴포넌트는 user space의 두 개의 계층으로 분리하여 설계한다.

Fig. 3은 본 논문에서 제안되는 주행제어구조의 개요도이다. 결순층의 Navigation_Module은 많은 시간을 요하는 정교한 알고리즘을 사용하여 센서 데이터로부터 고차원적인 정보를 생성하고 반사층에서는 실시간으로 간단한 계산을 주기적으로, 혹은 비주기적으로 수행하며 로봇의 동작을 제어한다. 반사층의 컴포넌트로서 Resource, Action, Action_Coordinator와 Motion_Controller가 있다.

3.1 Resource

Resource는 여러 센서에 관여하는 컴포넌트이다. 각 센서마다 별도의 Resource 컴포넌트가 존재하며 각각의 센서로부터 정보를 읽고 이를 필요로 하는 여러 Action 컴포넌트에 제공한다.

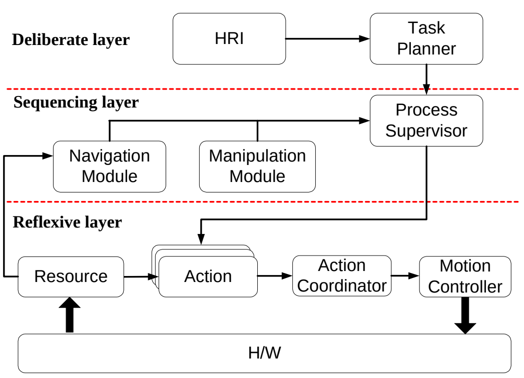


Fig. 2 The adopted control architecture

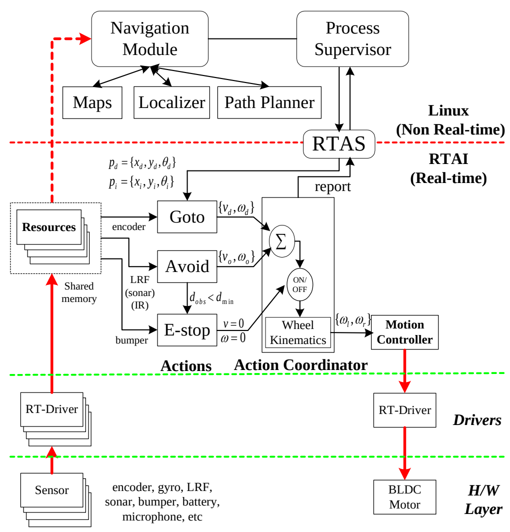


Fig. 3 The proposed navigation control architecture

Resource는 센서의 데이터 전송방식에 따라 폴링 방식[Fig. 4(a)]과 인터럽트방식[Fig. 4(b)]으로 분류할 수 있다. 폴링방식의 경우 주기적으로 데이터를 생성하기 위해서 RTAI에서 제공하는 주기적인 스레드(periodic thread)로 Resource 컴포넌트를 구현한다. *Sonar, LRF(front/rear), IR(front/rear), Encoder, Gyro, Battery* 등 대부분의 Resource가 이 방식에 해당한다. 인터럽트방식은 센서 인터페이스에서 생성되는 인터럽트에 반응하여 센서 데이터를 읽는다. RT_driver는 인터럽트를 통해 데이터를 읽고, 이를 RT_thread에 알리기 위한 방법으로 callback 기능을 제공한다. *Bumper*와 긴급정비 버튼에 이 방식이 사용되었다.

3.2 Actions

Action이란 로봇이 자율주행 시 조합하여 사용할 수 있도록 특정기능을 구현한 단위이동방법을 말한다. 여러 Resource 중에서 필요한 몇 개의 정보를 받아서 로봇을 움직이기 위한 병진속도와 각속도를 생성하며, Action은 RTAI에서 제공하는 함수를 이용하여 RT_thread로 구현된다. 반사계층에 존재하는 Action은 *Goto, Avoid, Move, EmergencyStop*으로 구성되어 있다.

Action들은 그 구성이 최대한 단순해야 한다. 기본적인 흐름은 Fig. 5와 같다. Resource 컴포넌트에 위해 수행이 재개(rt_task_resume)되면 센서의 정보를 읽고 연산 작업을 수행하며 그 결과 정보를 저장한 후, 다시 정지상태(rt_task_suspend)가 된다.

3.3 Action_Coordinator

로봇은 임의의 시각에 하나의 단위행위로만 동작될 수 있고, 혹은 여러 개의 단위행위들의 조합으로 동작될 수도 있다. 각각의 단위행위에서 계산된 로봇의 이동정보를 효과적으로 통합하여 Motion_Controller에 이동명령을 전달하여 다양하고 부드러운 움직임을 생성하는 것이 Action_Coordinator이다.

Action_Coordinator를 설계할 때 고려되어야 할 점에 따른 해결책은 아래와 같다.

(1) Action_Coordinator는 상위계층으로부터 현재 실행되어야 할 단위행위들의 동작형태를 결정해야 한다. *Goto*와 같이 일반적으로 로봇의 움직임은 위치정보에 따라 계산된다. 그러나, "주어진 속도로 이동물체를 추적하라"와 같은 속도정보에 따른 로봇의 움직임을 고려하여야 한다는 것이다. 즉, 로봇은 필요에 따라 위치제어 개념이 아닌 속도제어 개념으로 이동해야 할 때가 있는데, 로봇이 어떤 동작형태를 지녀야 할지를 먼저 인지시켜주어야 하기 때문이다.

→ RTAS에서 동작형태를 변경할 때 get_op_mode 함수를 이용하여 쉽게 접근할 수 있도록 하였다.

(2) 간단한 몇 개의 단위행위를 가지고 모든 상황에 따라 효율적으로 대처할 수 있는 로봇의 동작을 생성하기 위해서는 경쟁조정방법과 협조조정방법을 잘 사용하여야 한다.

→ 알고리즘 부분을 하나의 함수로 하여 쉽게 변경이 가능하도록 설계하였다. 현재는 성능평가 단계이므로 복잡한 알고리즘을 추가하기 보다 80 cm 내에 장애물이 접근하면 *Avoid*로 변환되도록 설계되었다. Fig. 6은 Action_Coorinator의 흐름도로서 설계시 요구사항을 만족시키는 구조로 설계되었다. Action_Coorinator는 로봇을 구동시키기 위한 최종 정보로 각 바퀴의 회전속도를 계산한다.

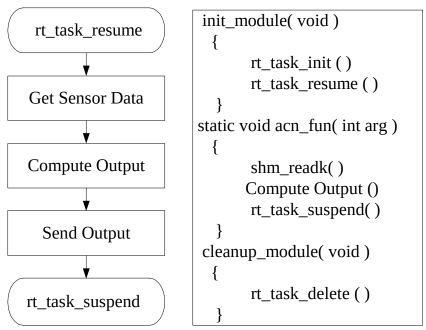


Fig. 5 Data flow in Actions

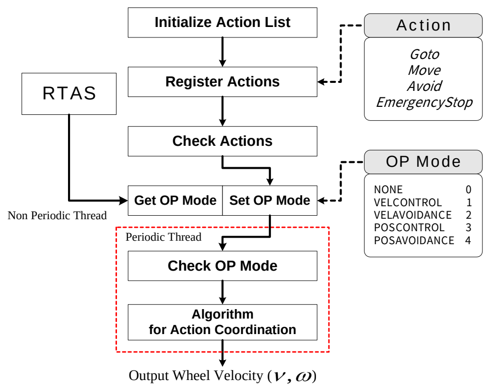


Fig. 6 Data flow of the Action_Coordinator

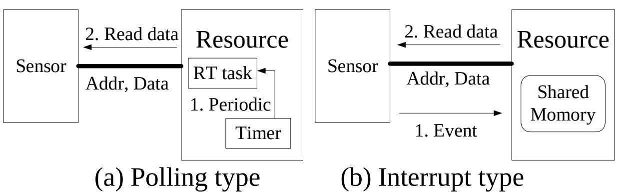


Fig. 4 Data processing types of resources

3.4 Motion_Controller

Motion_Controller는 실제로 모션보드에 제어신호를 전달하는 컴포넌트이다. Motion_Controller는 Action_Coordinator로부터 지정된 프로토콜을 통해 양쪽 바퀴의 속도정보를 받아 바퀴를 구동시킨다.

3.5 RTAS

RTAS(Real-Time Action Supervisor)는 user space(결순층)와 kernel space(반사층)를 연결하는 부분으로서, Action_Coordinator에 단위행위의 동작모드를 지정하고 로봇 이동을 위한 위치정보나 속도정보를 제공한다. 또한 RT_FIFO를 통해 kernel space에서 생성되는 데이터 및 동작상태를 전송 받는 역할을 한다. 본 연구에서는 GUI 환경에서 구현하여 주행제어구조의 동작여부와 성능을 평가하는 용도로 사용된다.

4. 단위행위

로봇이 복잡한 환경에서 안정적인 움직임을 보이기 위해서는 체계적인 단위행위가 설계되어야 하며, 주어진 센서의 정보를 잘 활용할 수 있어야 한다. Action은 여러 Resource 중에서 필요한 몇 개의 정보를 받아서 로봇을 움직이기 위한 병진속도와 회전속도를 생성한다. 또한, Action은 RTAI에서 제공하는 함수를 이용하여 RT_thread로 구현된다.

본 논문에서는 위치정보에 따른 로봇의 움직임 명령을 생성하는 *Goto*와 장애물 회피를 위한 *Avoid*, 속도기반 명령을 받는 *Move*, 긴급정지를 위한 *EmergencyStop*의 알고리즘을 소개한다.

4.1 *Goto*

*Goto*는 임무수행층에 위치한 Navigation_Module의 Localizer와 Path_Planner를 통해 계산된 로봇의 현재위치나 Encoder로부터 계산된 현재위치와, Process_Supervisor로부터 각 경로들의 위치정보를 얻어 최종 목적지까지 로봇을 이동시키는 단위행위이다.

*Goto*의 실행 주기는 Localizer에서 로봇의 현재위치에 대한 업데이트 주기보다 빠르게 실행되기 때문에 Localizer에서 정확한 로봇위치 정보를 받기 전까지는 Encoder로부터 받은 주행정보를 이용하여 현재위치를 계산한다.

상위 계층에서 현재위치에 대한 정보를 제공함에도 불구하고 Encoder을 이용하여 현재 위치정보를 계산하는 이유는 반응계층에서는 상위계층의 계산주기(200 msec)보다 훨씬 계산주기가 빠르므로(20 msec) hard real-time 특성이 우수하기 때문이다.

Aicardi et al.[1]의 결과를 채택하여 로봇의 병진속도 v와 회전속도 ω를 아래와 같이 설계한다.

$$v = k_3 \cos\rho,\ \ k_3 > 0, \tag{1}$$

$$\omega = k_4\rho + k_3 \frac{\cos\rho \sin\rho}{\rho}(\rho + k_2\theta_d), \quad k_2,\ k_3,\ k_4 > 0. \tag{2}$$

그러나, 실제로 로봇은 최종목표로 가기 위해서 여러 개의 경유점을 거쳐야하는데 이런 방법으로는 모든 경유점마다 속도가 0이 되기 때문에, 임의의 시각 t보다 2 ~ 3 구간 앞선 경유점을 목표점으로 하여 로봇의 병진속도와 회전속도 명령을 생성한다.

4.2 *Avoid*

반응계층의 *Avoid*는 반사적, 본능적 회피라 일컬을 수 있다. 본 논문에서 제안한 *Avoid*의 알고리즘은 센서로부터 측정된 장애물의 현재 떨어진 거리와 속도, 방향에 반응하여 로봇의 회피 동작을 위한 병진속도와 회전속도를 생성한다. 장애물을 인식할 때에 다양한 Resource가 사용될 수 있으나 현 단계에서는 LRF만 적용하였으며 계산식은 Minguez and Montano[15]의 nearness diagram 을 적용하였다. Fig. 7과 같이 로봇의 장애물 회피를 위한 경계값 d_{safe} 안에 장애물이 접근했을 때, 로봇의 속도 v_o와 회전속도 ω_o를 아래와 같이 계산한다.

$$v_o = K_{obs} \times \frac{d_{obs}}{d_{safe}} \times \left(\frac{\pi/2 - |\phi|}{\pi/2}\right) \times v_{\max}, \tag{3}$$

$$\omega_o = \frac{\phi}{\pi/2} \times \omega_{\max}. \tag{4}$$

여기서 k_{obs}은 로봇의 속도조절을 위한 게인값이

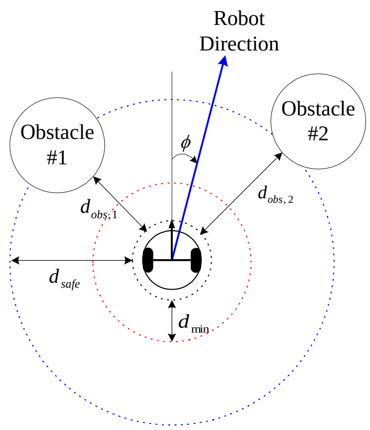


Fig. 7 Concept of obstacle avoidance

고 d_{obs} 는 로봇과 장애물과의 최소거리, ϕ 는 로봇진행방향에서 본 장애물의 방향을 나타내고, 낸다.

4.3 *Move*

일반적으로 로봇은 위치기반(point-to-point)으로 이동하도록 제어하는데, 물체추적 및 벽면추종과 같은 작업을 수행할 때 로봇의 위치제어방법이 모호해 진다. 이런 문제를 해결하기 위해 *Move*가 도입된다. *Move*는 RTAS에서 받은 명령정보를 속도로 받아 Action_Coordinator에 제공한다.

4.4 *EmergencyStop*

*EmergencyStop*은 Motion_Controller에서 하드웨어적으로 로봇의 이동속도를 모터 브레이크로 잡아 멈추는 것을 말한다. 일정속도로 이동중인 로봇이 멈추려면 관성에 의해서 약간 밀리는데, 이로 인해 안정상 위험을 초래할 수 있기 때문에 *EmergencyStop*이 필요하다. 이는 두 가지 경우에 동작하게 되는데, 첫 번째 경우는 로봇의 범퍼가 장애물과 충돌했을 때 동작하며, 두 번째 경우는 로봇이 장애물을 회피하는 중에 장애물이 $d_{\min}$ 안으로 들어왔을 때 동작한다.

5. 실 험

본 연구에서 제안한 주행제어구조의 성능평가는 두 가지의 경우로 나누어 수행된다. 첫 번째는 위치제어모드에서 수행되는데, 목표위치로 갈 때에 장애물이 없는 상황, 장애물이 있는 상황, 인간과의 충돌과 같은 긴급상황에 따라 *Goto*, *Avoid*, *EmergencyStop*의 동작여부를 평가한다. 다음으로, 속도제어 모드에서 등속주행 성능을 평가하고, 일정속도로 로봇이 진행할 때 회전속도를 변화시켜가며 얼마나 정확한 주행이 가능한지를 확인한다.

5.1 위치제어기반 주행실험

5.1.1 *Goto* 실험

본 실험은 위치제어기반 동작모드(op mode #3: *Goto*)에서 자율주행이 성공적으로 이루어지는가에 대한 실험이다. 우선, 조정자가 GUI 환경에 그려진 로봇의 환경을 보고 로봇을 이동시킬 목표점에 클릭을 하면 로봇이 이동하게 된다. 이렇게 GUI를 따로 구성한 이유는 실제 자율주행을 위해서는 Process_Supervisor에서 생성된 이동정보를 RTAS가 *Goto*에 정보를 줄 때, 목표위치로 이동하게 되지만 현재에는 Process_Supervisor 없이 반사층의 자율주행 실험을 하기 때문이다. Fig. 8(a)는 GUI 환경에서 명령을 주는 화면이며, Fig. 8(b)는 로봇이 주어진 명령에 따라 목표지점까지 정확하게 주행했음을 보여준다. 실제로 원하는 위치와 약간의 오차가 있는 것은 엔코더에 의한 누적오차이며 프로그램 시 설정한 offset 오차범위는 ±10 cm이다.

5.1.2 *Goto + Avoid* 실험

다음은 로봇이 원하는 목표로 가는 중에 만나는 장애물에 대해 *Goto*모드(op mode #3)에서 *Goto+Avoid* 모드(op mode #4)로 전환이 원활이 이루어지는가에 대한 실험이다. Fig. 9는 이동 중에 지역국소점을 만났을 때, 장애물을 회피하며 목표점으로 가고 있음을 보여주는 사진이다. 장애물이 없을 때는 op mode #3에 의해 목표위치를 향해 가다가 장애물이 인식되면 op mode #4로 전환하여 회피동작을 수행한다. 이 때 지역국소점이 나타나면, 벽면과의 공간이 넓은 쪽으로 이동하고 장애물 영역을 벗어나면 op mode #3에 의해 최종위치에서 멈추었다.

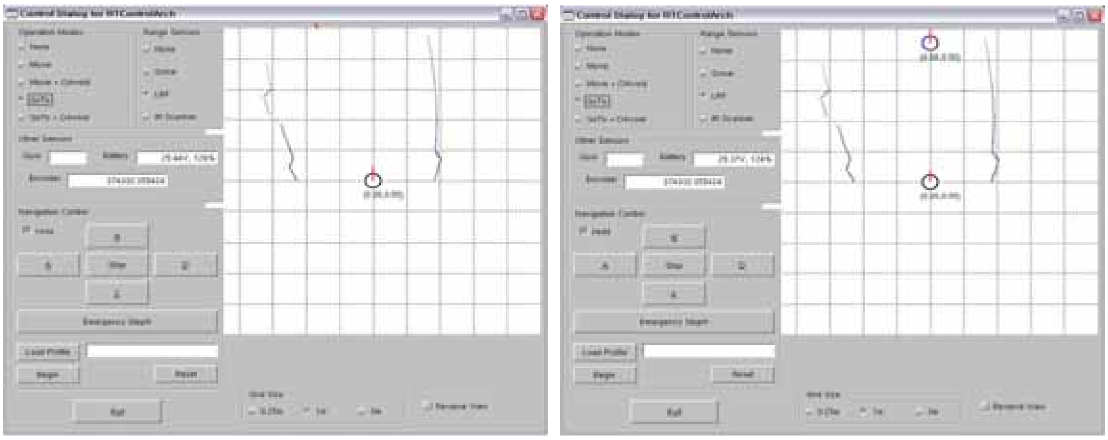
5.1.3 *EmergencyStop* 실험

Fig. 10은 로봇이 목표점으로 이동 중에 갑작스런 충돌이 발생했을 때, 어떻게 대처하는지에 대한 실험을 하는 장면으로 Fig. 10(a)는 등속으로 이동하다가 장애물이 범퍼에 부딪힐 때 하드웨어적으로, Fig. 10(b)는 장애물이 갑자기 너무 로봇 가까이 왔을 때 소프트웨어적으로 *EmergencyStop*이 동작하여 충돌한 장애물이 사라질 때까지 기다리는 것을 보여주고 있다. Fig. 11(a),(b)는 실험에 대한 로봇의 속도 출력 명령값과 센서값의 관계 그래프로써 센서에서 임계값 이하의 값이나 범퍼의 데이터 값이 들어오면 로봇의 속도 출력명령이 0이 됨을 보여준다.

5.2 속도제어기반 주행실험

5.2.1 *Move* 실험

Fig. 12는 로봇이 멈춘 상태에서 5초 동안 10, 20, 30 cm/sec의 속도로 이동하라는 명령을 각각 주었을 때, 로봇이 출력한 속도결과를 나타낸다. 처음 1초 동안과 마지막 1초는 속도에 관계없이 정속주행토크를 발생하기까지 필요한 시간으로서, 제작사에서 제공한 제원에 기인한 것이다. 정속구간에서는 정확한 속도로 이동했음을 확인할 수 있었으며, 속도가 빨라질수록 잡음이 발생하지만 무시할 수 있을 정도이다. Fig. 13은 직진속도가 30 cm/sec일 때, 회전속도를 -15 ~ 15 cm/sec로 변화시켜가면서 주행한 거리결과를 엔코더를 통해서 0.1 sec 단위로 측정한 결과를 그래프로 나타낸 것이다. 명령에 따라 정확히 주행하였음을 알 수 있다.



(a) Goal position setting for *Goto* test

(b) Snap shots

Fig. 8 *Goto* test

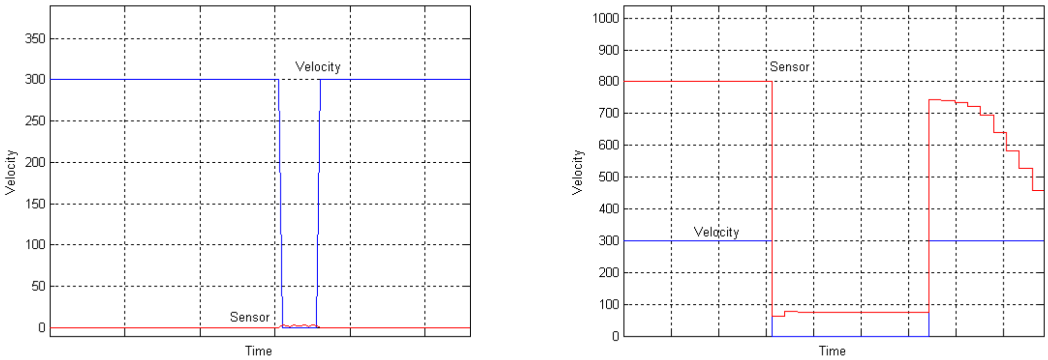
(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 9 Snap shots of the combined *Goto* + *Avoid* test

(a) Hard *EmergencyStop* under a bumper input (b) Soft *EmergencyStop* upon a critical range

Fig. 10 Snap shots of *EmergencyStop* tests



(a) Velocity curve of the hard *Emegencystop* (b) Velocity curve of the soft *Emegencystop*

Fig. 11 Comparison of two *EmergencyStops* : hard and soft

Table 1 Comparison of control architectures

	Saphira(11)	TeamBot(5)	BERRA(14)	The proposed Architecture
OS				
Linux	Yes	Yes	Yes	Yes
MS Windows	Yes	Yes	No	No
Real-time OS	No	No	No	Yes (RTAI 3.1)
Main prog. Language	C	Java	C++	C/C++
Software Tools Req. (a)	gcc	Java 1.2	gcc 2.95	gcc 3.2
Graphics				
GUI	Yes	Yes (simulation)	Yes	Yes
Graphics Prog. (b)	Yes (limited)	Yes	No	Yes
HRI				
Text	Yes	Yes	Yes	Yes (other layer)
Speech	No	No	Yes	Yes (other layer)
GUI	Yes	No	Yes	Yes (other layer)
Palm	No	No	Yes	Yes (other layer)
Multi Agent Support	No	Yes	No	Yes
Multi Host	No	No	Yes	Yes
Multi Process	Threads	Thread	Multi-process	Multi-process/ Thread
Data Flow Paradigm				
Push	Yes	No	Yes	Yes
Pull	Yes	Yes	Yes	Yes
Platform Portability				
Hardware Abstraction	Good	Very good	Very good	Very good
Sensor Extension Cap. (c)	No	Good	Very good	Very good
Sensor Support				
Sonar	Yes	Yes	Yes	Yes
LRF	Yes	No	Yes	Yes
IR	No	No	Yes	Yes
Camera	Yes	Yes	Yes	Yes (other layer)
Bumper	Yes	Yes	Yes	Yes
Microphone	No	No	No	Yes (other layer)
Gyro	No	No	No	Yes
Touchscreen	No	No	No	Yes
Behavior Coordinator (Action)	Fuzzy logic	Various	VFH(d)/VFF(e)	Various
Timing Aspect				
Real-time Support	No	No	No	Yes
Sensor Actuator Latency	0.6 sec	0.4 sec	0.17 sec	0.02/0.2 sec
Bandwidth				
Sensor to Behavior (Action)	4 KB/s	OS limitation	OS limitation	OS limitation
Code Size				
Complete System	11 MB	45 MB	36 MB	5.82 MB

(a) Prog.: Program, (b) Req.: Requirements, (c) Cap.: Capability, (d) VFH: Vector Field Histogram,
(e) VFF: Vector Field Force.

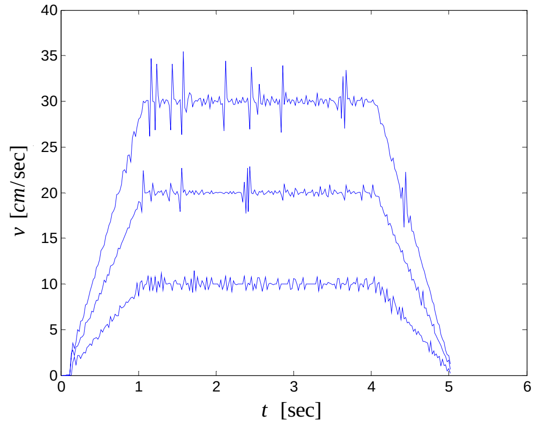


Fig. 12 Constant-velocity travels by *Move*

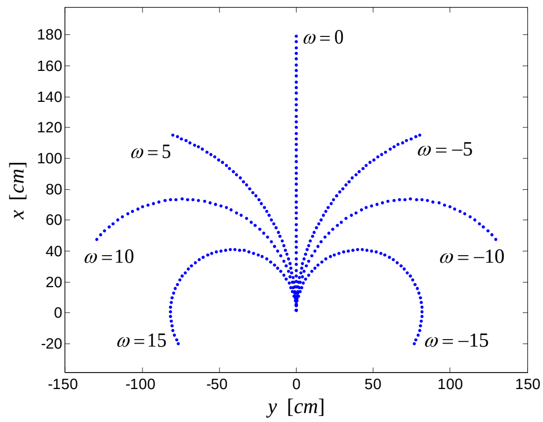


Fig. 13 Constant-angular-velocity moves with $v = 30$ cm/sec

5.3 주행제어구조 성능비교

앞에서 제안한 주행제어구조의 성능을 평가하기 위해 과거에 성공적으로 개발되어진 Saphira[12], TeamBots,[5] BERRA[16]들과 비교하여 Table 1로 나타내었다.

평가기준은 Orebäck and Christensen[19]이 제안한 평가방법을 따랐으며 다른 제어구조에 비해 제안된 주행제어구조는 실시간 운영체제를 탑재하였고, 알고리즘 계산 및 센서정보의 처리에 있어 실시간성이 확보되었음을 보여준다. 또한 다양한 센서를 가지고 있고, 센서의 탈부착이 용이하게 개발되었으며, 프로그램 크기 또한 작음을 확인 할 수 있다. 제안된 주행제어구조는 아직 자율이동로봇의 전체 제어구조가 완성되지 않고 주행부분으로만 구성되어 있지만 하드웨어 및 소프트웨어, 데이터 처리면에서 상대적으로 우수함을 볼 수 있다.

6. 결 론

본 논문에서는 기존에 개발된 제어구조를 분석하여 단점 및 문제점들을 파악하였다. 그리고 이러한 단점들 및 문제점을 해결할 뿐만 아니라 이동로봇에 자율기능을 부여하기 위한 틀로서 로봇에 가장 적합하다고 판단되는 three-layer의 구조를 선택하였고 실시간성을 향상시키기 위한 방안으로 실시간 O/S를 선정하였다. 또한 실시간성이 향상된 반사층의 주행 제어구조를 제안하였으며, 이동성을 향상시키기 위한 방법으로서 단위행위들을 설계하였다. 주행성능 실험을 통해 이동성 및 안정성을 검증하였으며, 다른 제어구조와 비교해 봄으로서 그 성능이 세계 유수의 연구결과에 견줄만 한 결과임을 알 수 있었다. 추후, 자기위치를 보정할 수 있는 *Calibrate* 등 다양한 단위행위가 추가되면 보다 강인하고 안정적인 주행이 가능해질 것이다.

후 기

이 논문은 부산대학교 학술연구비(2 년)에 의하여 연구되었습니다.

참고문헌

(1) Aicardi, M., Casalino, G., Bicchi, A. and Balestrino, A., 1995, “Closed Loop Steering of Unicycle-like Vehicles via Lyapunov Techniques,” *IEEE Robotics and Automation Magazine*, Vol. 2, No. 1, pp. 27~35.
(2) Albus, J. S., 2002, “4D/RCS A Reference Model Architecture for Intelligent Unmanned Ground Vehicles,” *Proc. of the SPIE Annual International Symposium on Aerospace/Defence Sensing, Simulation and Controls*.
(3) Arkin, R. C., 1998, *Behavior-Based Robotics*, The MIT Press, Cambridge.
(4) Arkin, R. C., 1998, “Motor Schema-based Mobile Robot Navigation,” *Int. J. of Robotics Research*, Vol. 8, No. 4, pp. 92~112.
(5) Balch, T., 2000, *TeamBots*, www.teambots.org.
(6) Brooks, R. A., 1986, “A Robust Layered Control System for a Mobile Robot,” *IEEE J. of Robotics and Automation*, Vol. 2, No. 1, pp. 14~23.
(7) Brooks, R. A., 1996, “Behavior-based Humanoid Robotics,” *Proc. of the IEEE/RSJ Int. Conf. in Intelligent Robots and Systems’96*, Vol. 1, pp. 1~8.
(8) Choi, C. H., Lee, J. S., Song, J. B., Chung, W. J., Choi, J. S. and Kim, M. S., 2002, “Topological Map Building for Mobile Robot Navigation,” *J. of Control, Automation and Systems Engineering*, Vol.8, No.6, pp. 373~380.

(9) Dozio, L. and Mantegazza, P., 2003, "Linux Real Time Application Interface (RTAI) in Low Cost High Performance Motion Control," *Proc. of the Motion Control 2003, Conf. of ANIPLA*.
(10) Gat, E., 1998, "On Three-Layer Architecture," *Artificial Intelligence and Mobile Robots*, AAAI Press.
(11) Hans, M. and Baum, W., 2001, "Concept of a Hybrid Architecture for Care-O-bot," *Proc. of ROMAN-2001*, pp. 407~411.
(12) Konolidge, K., and Myers, K., 1996, *The Saphira Architecture for Autonomous Mobile Robots*, SRI International.
(13) Kim, G. H., Chung, W. J., Kim, M. S. and Lee, C. W., 2004, "Implementation of Multi-Functional Service Robots Using Tripodal Schematic Control Architecture," *Proc. of the Int. Conf. on Robotics and Automation*, pp. 4005~4010.
(14) Kim, S. J. and Kim, B. K., 2004, "Development of Real-Time Control Architecture for Autonomous Navigation of Powered Wheelchair," *J. of Control, Automation, and Systems Engineering*, Vol. 10, No. 10, pp. 940~946.
(15) Lee, S. Y. and Song, J. B., 2004, "Generalized Voronoi Diagram Based Path Planning for a Mobile Robot in Dynamic Environment," *Proc. of the 4th Int. Conf. on the Advanced Mechatronics ICAM*, pp. 114~119.
(16) Lindström, M., Orebäck, A. and Christensen, H. I., 2000, "BERRA: A Research Architecture for Service Robots," *Proc. of the IEEE Conf. on Robotics and Automation*, pp. 3278~3283.
(17) Minguez, J. and Montano, L., 2004, "Nearness Diagram (ND) Navigation: Collision Avoidance in Troublesome Scenarios," *IEEE Trans. on Robotics and Automation*, Vol. 20, No. 1, pp. 45~59.
(18) Nesnas, I., Wright, A., Bajracharya, M., Simmons, R., Estlin, T., and Kim, W. S., 2003, "CLARAty: An Architecture for Reusable Robotic Software," *SPIE Aerospace Conf.*.
(19) Orebäck, A. and Christensen, H. I., 2003, "Evaluation of Architecture for Mobile Robotics," *Autonomous Robots*, Vol. 14, pp. 33~49.
(20) Ridao, P., Batlle, J. and Carreras, M., 2002, "O^2CA^2, A New Object Oriented Control Architecture for Autonomy: The Reactive Layer," *Control Eng. Practice*, Vol. 10, pp. 857~873.
(21) Song, J. B. and Byun, K. S., 2004, "Design and Control of a Four-Wheeled Omnidirectional Mobile Robot with Steerable Omnidirectional Wheels," *Journal of Robotic Systems*, Vol. 21, No. 4, pp. 193~208.