Journal of Institute of Control, Robotics and Systems (2013) 19(11):1036-1042
http://dx.doi.org/10.5302/J.ICROS.2013.13.1922 ISSN:1976-5622 eISSN:2233-4335

실시간 임베디드 리눅스 기반 노약자 지원 로봇 개발

Elderly Assistance System Development based on Real-time Embedded Linux

고 재 환, 양 길 진, 최 병 욱*
(Jae-Hwan Koh[1], Gil-Jin Yang[2], and Byoung-Wook Choi[2])
[1]HA Control R&D Lab., LG Electronics
[2]Dept. of Electrical and Information Engineering, Seoul National University of Science and Technology

Abstract: In this paper, an elderly assistance system is developed based on Xenomai, a real-time development framework cooperating with the Linux kernel. A Kinect sensor is used to recognize the behavior of the elderly and A-star search algorithm is implemented to find the shortest path to the person. The mobile robot also generates a trajectory using a digital convolution operator which is based on a Bezier curve for smooth driving. In order to follow the generated trajectory within the control period, we developed real-time tasks and compared the performance of the tracking trajectory with that of non real-time tasks. The real-time task has a better result on following the trajectory within the physical constraints which means that it is more appropriate to apply to an elderly assistant system.

Keywords: xenomai, real-time operating system, Kinect, A-star search algorithm, Bezier curve

I. 서론

지능형 모바일 로봇은 사람과 유사한 다양한 센서를 통하여 외부 환경을 인식(sensing)하며, 얻어진 정보를 통하여 자신과 외부의 상태를 지각(perception)하고 판단을 수행하는 인지(cognition)와 지능이 결합되어 운동기능(manipulation)의 역할을 수행 한다. 이와 같은 전체 소프트웨어 계층에서 확실성, 지능성, 적응성과 함께 실시간성이 요구되며, 이와 같은 요구 사항을 만족하는 응용 프로그램으로 서비스를 안정적이면서 체계적으로 지원하기 위해서는 운영체제의 도움을 받아야 한다. 일반적으로 범용 운영체제로 많이 쓰이는 운영체제에는 윈도우와 리눅스가 있지만 이와 같은 운영체제는 실시간성을 보장하지 못함으로써 예상치 못한 행위를 제어하는데 어려움을 겪게 된다. 지능형 모바일 로봇은 인간과 유기적으로 관계하기 때문에 예상치 못한 행위는 인간에게 치명적인 위험이 될 수 있다. 또한 실시간성을 보장 받지 못함으로써 사용자가 원하는 성능을 발휘하지 못하게 될 수 있다. 이러한 단점을 보완하기 위해 실시간 운영체제(RTOS: Real-Time Operating System)를 적용하는 추세이다[1].

실시간 운영체제는 임베디드 운영체제 중에서 시간을 제어할 수 있는 기능이 포함된 운영체제로 상용 RTOS와 오픈소스인 실시간 임베디드 리눅스로 구분 지을 수 있다[2-5]. 상용 RTOS로는 VxWorks, Nucleus PLUS, QNX Neutrino 등이 있고, 리눅스를 활용하여 실시간성을 지원하는 실시간 임베디드 리눅스로는 RT-Linux, RTAI 프로젝트, Xenomai 프로젝트 등이 있다. RT-Linux의 경우에는 상용화가 이루어지면서 상용버전 RT-Linux/Pro와 오픈 소스인 RT-Linux로 나누어져 있다.

상용 RTOS를 이용한 지능형 모바일 로봇 개발은 초기 개발 비용이 많이 들고 특정 운영체제로의 기술 종속이 될 수 있다[6]. 따라서 본 논문에서는 상용 RTOS의 대안이 될 수 있는 실시간 임베디드 리눅스를 이용한 지능형 모바일 로봇 개발을 제안한다. 실시간 임베디드 리눅스는 저렴한 비용으로 개발이 가능할 뿐만 아니라 성능 면에서도 상용 RTOS에 비해 뒤떨어지지 않는다[7].

본 연구에서는 선행 연구에서 수행된 실시간 임베디드 리눅스 성능 비교에서 매우 안정적인 특성을 보이는 Xenomai를 지능형 모바일 로봇에 적용한다[8]. 또한 이를 응용한 어플리케이션으로 노약자 지원 시스템을 개발한다.

본 논문의 구성은 다음과 같다. 먼저 II 장에서는 본 논문에서 제안한 실시간 임베디드 리눅스 기반 노약자 지원 시스템에 대해 설명한다. III 장에서는 실시간 기반과 비실시간 기반에서의 성능 차이에 대해 논의하고, IV 장에서는 노약자 지원 시스템 개발 과정을 서술한다. 마지막으로 V 장에서는 본 논문의 결론을 도출한다.

II. 시스템 구조

그림 1은 본 논문에서 구현한 실시간 임베디드 리눅스 기반 노약자 지원 시스템의 구성을 보여주고 있다. 주요 시나리오는 지능형 모바일 로봇의 주 제어장치의 운영체제로

* 책임저자(Corresponding Author)
Manuscript received May 22, 2013 / revised July 29, 2013 / accepted August 22, 2013
고재환: LG전자 주식회사(jaehwan.ko@lge.com)
양길진: 서울과학기술대학 전기공학과(yang6495@gmail.com)
최병욱: 서울과학기술대학 전기정보공학과(bwchoi@seoultech.ac.kr)
※ 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단 기초연구사업의 지원을 받아 수행된 연구임(2012-006057).

Xenomai를 이용하여 실시간 시스템으로 구성하고, 노약자의 활동 인식을 위하여 키넥트(Kinect) 센서를 이용한다. 키넥트 센서의 영상 이미지를 이용하여 노약자의 활동 인식을 위한 알고리즘 개발하였으며, 행동 이상을 확인하고 노약자에게로 자율 주행하는 시스템을 구현하였다. 자율 주행은 최단거리 주행을 위해 A*(A-star) 알고리즘을 이용하였고, 경로는 부드러운 곡선 주행을 위하여 베지어 곡선을 이용하였다. 또한 경로 추종시 물리적 제한을 만족하는 디지털 컨볼루션 방법으로 궤적 생성을 하는 알고리즘을 개발하였다[9,10-12]. 마지막으로 모니터링 시스템은 별도의 프로그램을 개발하지 않고 기존의 동부 로봇에서 제공된 모니터링 프로그램을 사용하였다.

1. 하드웨어 구성

본 연구에서는 실험을 위해 동부로봇에서 제작한 TETRA-DS III을 이용한다. TETRA는 2개의 AC 서보 모터를 가지고 있으며, 주행 최대 속도는 2.0m/s이다. 또한 범퍼 센서, 초음파 센서, LRF (Laser Range Finder)를 탑재하고 있으며 제어보드로 VIA 임베디드 보드를 사용한다. 표 1은 연구에서 사용한 TETRA의 하드웨어 사양이다.

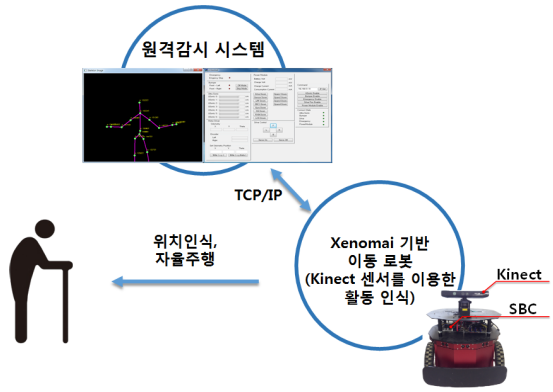


그림 1. 시스템 구성도.

Fig. 1. System overview.

표 1. 모바일 로봇의 하드웨어 사양.

Table 1. The hardware specification of the mobile robot.

본체	외형 사이즈	L522×W456×H295mm
	중량	20kg 이하
이동부	주행방식	2-Wheel Differential Drive (AC Servo Motor)
	주행속도	max. 2.0m/s
	감속비	15:1
	payload	40kg
	바퀴간 거리	406mm
	Clearance	50mm
바퀴부	바퀴 직경	240mm
	바퀴 폭	50mm
센서부	범퍼 센서	3방향 감지, 전방 180도 감지
	초음파 센서	7개, Ring Array
	LRF	Hokuyo URG-04LX
	Kinect 센서	3D Depth Sensors, RGB Camera, Multi-Array MIC, Motorized Tilt
Embedded Board	SBC	VIA EPIA-N700 or 800, PCM-9362D-S6A1E
	HDD	32GB SSD
	Serial	RS-232 3Port, RS-232/485 1Port
	OS	Ubuntu(Xenomai) /Windows7

본 연구에서는 그림 1과 같은 노약자 지원 어플리케이션 개발을 위해 TETRA에 Microsoft사의 키넥트 센서를 추가하였다. TETRA에 내장된 VIA 임베디드 보드는 사양이 낮기 때문에 키넥트 센서를 구동하기에는 어려움이 존재한다. 이러한 이유로 그림 3과 같이 키넥트 센서 사용을 위해 별도의 SBC (Single Board Computer)를 추가하였다.

VIA 임베디드 보드는 운영체제로 Ubuntu를 사용하며, 실시간성 확보를 위해 리눅스 커널 2.6.32.11에 Xenomai(2.5.3)를 적용하였다. 추가된 SBC의 경우 키넥트 센서의 권장 운영체제인 Windows7을 사용하였으며, 개발 툴로 Visual Studio 2010을 이용하였다. 또한 노약자의 스켈레톤 이미지를 얻기 위해 Kinect for Windows SDK v1.5와 OpenCV2.3.1을 이용하였다.

그림 2는 본 연구에서 사용한 모바일 로봇의 실제 모습이다. 키넥트 센서를 이용한 노약자 검출을 위해 로봇의 상단에 키넥트 센서 및 SBC를 추가하였으며, 장애물 검출 및 충돌 감지를 위해 초음파 센서, 범퍼 센서, LRF를 장착하였다.

그림 3은 본 연구에서 구현한 실시간 H/W 제어 구조이다. 모바일 로봇의 제어 및 구동을 위한 VIA 임베디드 보드와 키넥트 센서 사용을 위한 SBC 사이의 통신은 TCP/IP를 통해 이루어진다. 또한 Power, Sensor, Drive 보드와 VIA 임베디드 보드는 시리얼 통신을 통해 데이터를 주고받으며, VIA 임베디드 보드 내의 다양한 서비스는 실시간 태스크로 구현되었다.

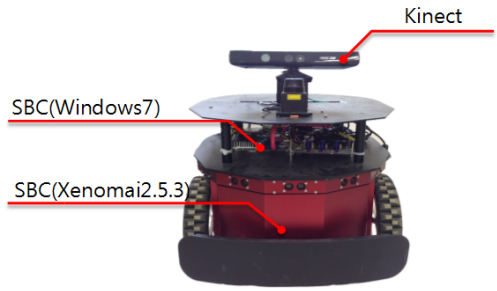


그림 2. TETRA-DS III 모바일 로봇.

Fig. 2. The mobile robot TETRA-DS III.

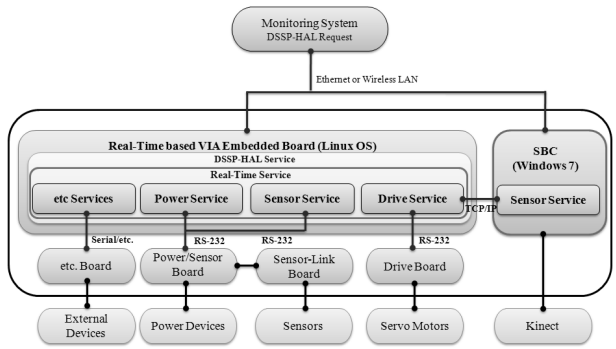


그림 3. 실시간 H/W 제어구조.

Fig. 3. The real-time hardware control structure.

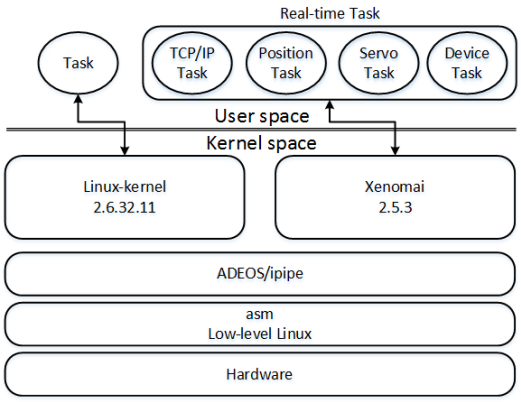


그림 4. 실시간 시스템 소프트웨어 구성.
Fig. 4. Architecture of the real-time software.

2. 실시간 소프트웨어 구성

VIA 임베디드 보드 내에서 작동하는 실시간 태스크는 그림 4와 같이 구조를 보인다[8]. 응용 프로그램은 사용자 공간에서 Xenomai 실시간 타스크로 구현되었다. 모터 구동 태스크 서보 타스크(servo task)와 추가된 SBC로부터 노약자의 관절 데이터를 가져오는 태스크 통신 타스크(TCP/IP task), 로봇의 위치를 추정하는 태스크 위치 타스크(position task), 센서 및 기타 장치들을 관리하는 태스크 디바이스 타스크(device task)들로 구성되었다.

키텍트 동작을 위하여서는 별도의 SBC를 이용하여 윈도우즈 기반으로 다음 장에서 설명한 방법으로 다양한 응용 프로그램이 구현되어 있다. 키넥트 센서를 이용하여 RGB 영상 추출, Depth 영상으로부터 거리 정보를 얻는다. 그리고 Skeleton 이미지를 얻어오는 부분과 이를 이용하여 노약자의 활동성을 판단하는 프로그램으로 구성된다. 또한 실시간 제어 장치인 VIA 임베디드 보드와의 TCP/IP 통신을 담당하는 부분으로 이루어져 있다.

3. 키넥트 센서 활용

본 연구에서는 키넥트 센서를 이용하여 노약자의 관절데이터를 얻어오기 위하여 RGB 영상, Depth 영상, Skeleton 이미지를 얻어올 수 있는 NUI API를 이용 하며, NUI API 사용을 위해서는 NuiInitialize 함수를 이용해야 한다. NuiInitialize 함수는 원하는 기능에 따라 5가지의 플래그를 인자로 사용하며, 본 논문에서는 실험에 필요한 다음과 같은 플래그를 OR연산 형태로 사용하였다.

- NUI_INITIALIZE_FLAG_USES_COLOR
- NUI_INITIALIZE_FLAG_USES_DEPTH_AND_PLAYER_INDEX
- NUI_INITIALIZE_FLAG_USES_SKELETON

NuiInitialize 함수를 이용한 초기화가 끝나면 Depth 이미지와 Color 이미지 출력을 위해 NuiImageStreamOpen 함수를 이용하여 스트림을 개방한다. 스트림 개방을 위해서는 해상도를 인자로 넘겨주어야 하는데 NUI_INITIALIZE_FLAG_USES_COLOR를 이용할 경우 NUI_IMAGE_RESOLUTION_1280x1024, NUI_IMAGE_RESOLUTION_640x480 두 가지 해상도를 지원하며, NUI_INITIALIZE_FLAG_USES_DEPTH_AND_ PLAYER_INDEX의 경우 NUI_IMAGE_RESOLUTION_320x240과 NUI_IMAGE_RESOLUTION_80x60 두 가지의 해상도를 지원한다. 본 연구에서는 Color 이미지의 경우 NUI_IMAGE_RESOLUTION_640x480, Depth 이미지의 경우 NUI_IMAGE_RESOLUTION_320x240을 사용하였다.

이미지 스트림 개방 외에 노약자의 골격 추적을 위해 NuiSkeletonTrackingEnable 함수 호출을 통해 SkeletonTracking을 활성화 시켜야한다. SkeletonTrancking이 활성화되면 Image, Depth 데이터와 함께 Skeleton 데이터가 함께 전달되며, 스켈레톤 추적 상태에 따라 다음 세 가지로 구분된다.

- NUI_SKELETON_TRACKED
- NUI_SKELETON_NOT_TRACKED
- NUI_SKELETON_POSITION_ONLY

본 논문에서는 노약자의 상태를 지속적으로 관찰해야 되기 때문에 NUI_SKELETON_TRACKED 상태에서만 Skeleton이 검출되었다고 판단하며, 이를 바탕으로 Skeleton 이미지를 출력하였다. 또한 SkeletonTracking이 활성화 되면 프레임내의 관절 정보를 매끄럽게 만들기 위해 NuiTransformSmooth 함수를 이용하였다.

마지막으로 이미지를 지속적으로 가져오기 위해 NuiImageStreamGetNextFrame 함수를 이용하며, 본 연구에서는 Color, Depth, Skeleton 이미지를 100ms마다 가져올 수 있도록 설정하였다. 그리고 스트림이 지속적으로 개방되면 메모리 누적이 발생하기 때문에 NuiImageStreamReleaseFrame 함수를 이용하여 메모리 누적을 방지하였다.

III. 실험

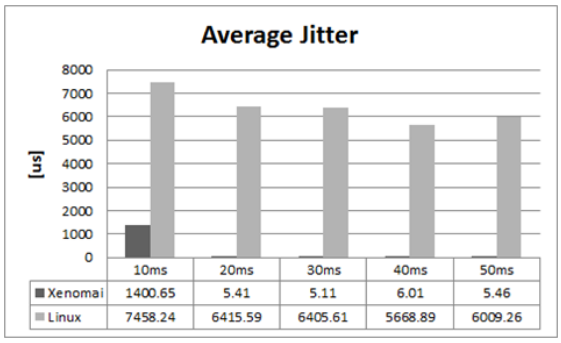
실시간 임베디드 리눅스 기반 지능형 모바일 로봇의 성능을 검증하기 위해 두 가지 측면에서 범용 리눅스와 실시간 임베디드 리눅스의 성능을 비교하였다. 첫 번째로는 속도 명령이 입력되는 모터 구동 태스크의 주기성을 비교하였으며, 두 번째로는 주어진 궤적을 추종 시 발생하는 거리 오차를 비교하였다.

격자 맵의 크기가 4x4인 환경에서 실험을 위해 로봇의 출발 좌표는 (0, 0, 45°), 노약자의 좌표는 (4, 4, 45°)으로 가정하였다. 격자 맵의 한 칸의 실제 크기는 50cm이고, 로봇의 최대 속도는 0.25m/s로 설정하였다.

1. 태스크의 주기성

디지털 컨볼루션과 베지어 곡선을 통해 얻어진 속도 프로파일이 정해진 시간에 입력되지 않을 경우 모바일 로봇이 주어진 궤적을 추종하는데 어려움을 겪게 된다. 따라서 정해진 시간에 정확한 속도 명령이 입력되어야 한다. 이러한 이유로 본 논문에서는 Xenomai가 적용된 리눅스와 범용 리눅스에서 모터 구동 태스크의 주기성을 측정하여, 정확한 주기에 속도 명령이 입력되는지를 분석하였다. 태스크의 주기는 10ms부터 50ms까지 10ms씩 증가시키며 실험하였다. 결과는 그림 5와 같다.

결과를 보면 10ms의 주기에서 Xenomai는 1400us, 범용 Linux는 7458us의 오차가 발생하였다. 실시간 임베디드 리



	10ms	20ms	30ms	40ms	50ms
Xenomai	1400.65	5.41	5.11	6.01	5.46
Linux	7458.24	6415.59	6405.61	5668.89	6009.26

그림 5. 태스크의 주기성 비교.

Fig. 5. Comparison of task periodicity.

눅스인 Xenomai와 범용 리눅스 모두 많은 오차를 나타내고 있으며, 이는 시리얼 통신의 지연으로 인하여 정해진 시간 안에 모든 일을 끝내지 못하고 다음 주기에 남은 일을 처리하는 경우가 존재하기 때문이다. 10ms 외의 주기에서는 Xenomai의 경우 5~6[us]의 오차를 나타내며, 비교적 정확한 주기에 태스크가 실행되었다. 반면 범용 리눅스의 경우 10ms에서 7.5[ms]의 오차를 나타내었으며, 그 외의 주기에서는 5.6~6.4[ms]의 오차를 나타냄으로써 Xenomai에 비해 많은 오차를 보였다.

2. 주행거리

그림 6와 같은 속도 프로파일 입력 시 모바일 로봇의 실제 주행 거리를 측정하여 비실시간 기반 모바일 로봇과 실시간 기반 모바일 로봇의 성능을 비교하였다. 속도 입력은 그림 6와 같은 속도 프로파일을 10ms, 30ms, 50ms의 주기로 샘플링 하여 속도 명령을 입력하였다.

실시간 및 비실시간 기반 모바일 로봇에서 그림 7와 같은 속도 프로파일 입력 시 로봇이 이동한 궤적을 모터의 엔코더 값을 읽어 그림 7과 같이 나타내었다.

실시간 기반의 경우 10ms에서 전술했던 이유 때문에 다른 주기에서보다 실제 주행거리가 늘어나는 모습을 보였다. 그 외의 주기에서는 원하는 좌표에 근접하게 접근하였고, 로봇의 다이나믹스, 샘플링 오차, 슬립으로 인해 약간의 오차가 발생하였다.

비실시간인 범용 리눅스의 경우 전체 주기에서 주어진 시간 내에 모든 작업을 수행하지 못해 태스크의 주기가 지

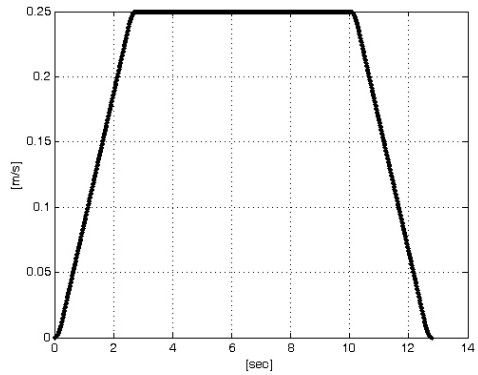


그림 6. 물리적 제한을 만족하는 속도 프로파일.

Fig. 6. The velocity profile that satisfies the physical limit.

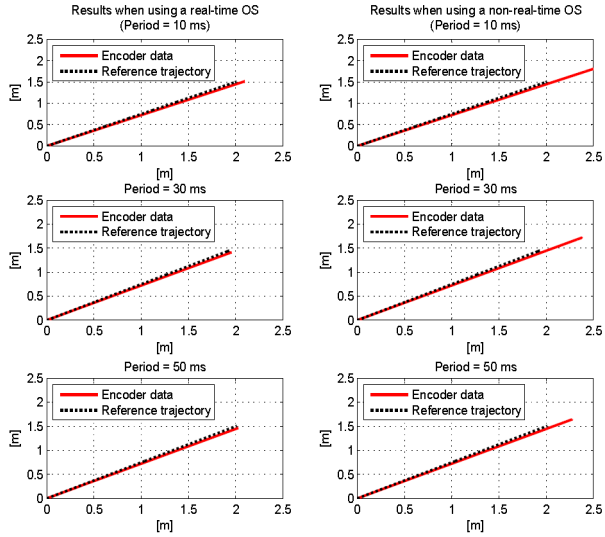


그림 7. 시스템에 따른 주행 실험 결과.

Fig. 7. The result of the driving experiment according to the system.

연되었다. 이러한 영향으로 원하는 목표보다 많은 거리를 이동하였다. 또한 태스크의 주기가 길어질수록 주기 내에 원하는 작업 수행이 가능하게 되어 오차가 점점 줄어드는 모습을 보였다.

IV. 노약자 지원 시스템 개발

본 연구에서는 그림 1과 같은 노약자 지원 시스템을 개발하기 위해 그림 8과 같은 순서도를 이용하였다.

모바일 로봇은 시작과 동시에 H/W 및 S/W 초기화가 이루어지며 로봇에 탑재된 키넥트 센서를 이용하여 노약자를 검출하고 위급한 상황이 발생하기 전까지 지속적으로 추적 및 관찰을 수행한다. 키넥트 센서를 통해 얻어지는 노약자의 관절 데이터를 바탕으로 위급 상황을 판단하며, 위급 상황이 발생했을 경우 현재 로봇의 위치를 기준으로 A*알고

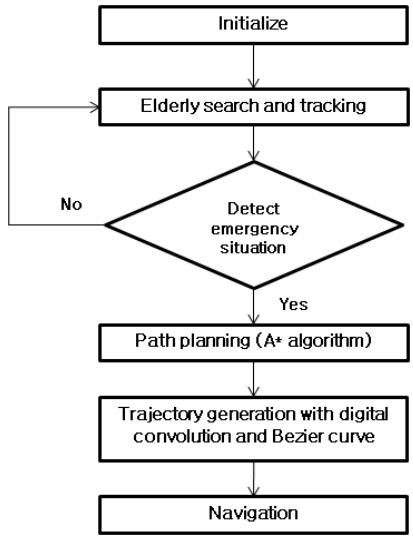


그림 8. 노약자 지원 시스템 순서도.

Fig. 8. Flow chart of the elderly assistance system.

리즘을 이용해 노약자까지의 최단 경로 검색을 수행한다. 또한 디지털 컨볼루션과 베지어 곡선을 이용하여 궤적을 생성하며 얻어진 궤적을 바탕으로 주행을 한다.

1. 노약자 행동 인식

노약자의 생활을 지원하기 위해서는 노약자를 인식(sensing)하고, 위급한 상황이 발생했을 때 이를 지각(perception)하고 인지(cognition)할 수 있어야 한다. 본 연구에서는 노약자를 인식하기 위해 키넥트 센서를 사용하였다. 키넥트 센서를 통해 노약자의 행동을 관찰하고, 얻어진 정보를 통해 노약자의 상태를 판단하였다. 키넥트 센서를 통해 얻는 정보는 그림 9와 같이 RGB 영상, Depth 영상, 인식된 사람의 관절 데이터이다.

본 연구에서는 실험을 위해 노약자가 쓰러지는 상황을 가정하였다. 로봇에 탑재된 키넥트 센서를 이용해 노약자를 관찰하다 노약자가 쓰러지는 상황을 인식하고, 로봇의 현재 위치 및 각도와 거리 데이터를 이용해 노약자의 위치를 파악한다. 파악된 노약자의 위치를 TCP/IP 통신을 통해 VIA 임베디드 보드에 전달한다. 그림 9와 그림 10은 키넥트 센서를 통해 노약자가 쓰러졌을 때 비상상태 발생을 판단하는 모습이다. 실험의 편의를 위해 노약자의 관절이 한계 값 밑으로 내려갈 경우 쓰러졌다고 판단하였다.

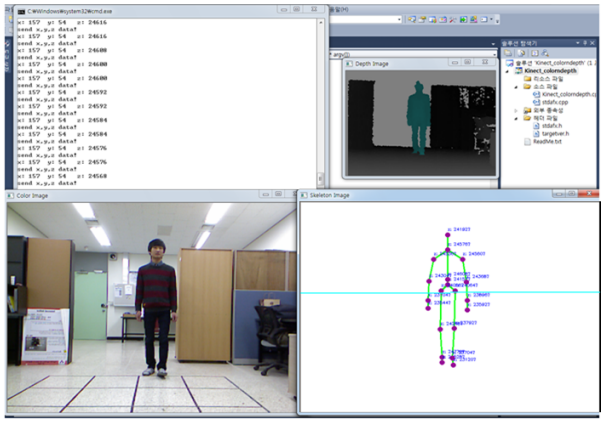


그림 9. 비상 상황 감시(Normal Status).

Fig. 9. Emergency monitoring (Normal Status).

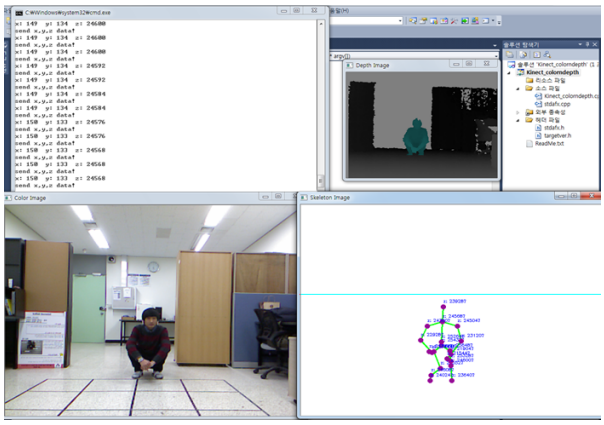


그림 10. 비상 상황 감시 (Emergency Status).

Fig. 10. Emergency monitoring (Emergency Status).

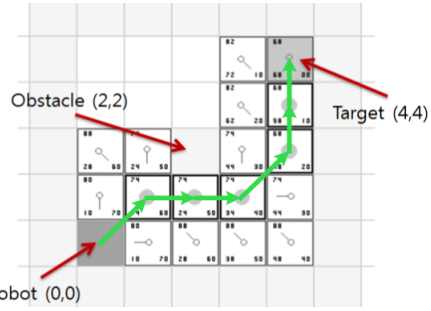


그림 11. A* 알고리즘을 이용한 경로 계획.

Fig. 11. Path planning using the A* algorithm.

2. 경로 계획

노약자가 쓰러졌다고 판단되면 현재 로봇과 노약자 사이의 최단 경로를 A* 알고리즘을 이용해 검색한다. 그림 11은 노약자가 (4, 4)에 존재하고 로봇의 현재 위치가 (0, 0)일 때 최단 경로를 검색하는 경우를 나타낸 것이다.

구현된 노약자 지원 시스템은 SLAM (Simultaneous Localization and Mapping)이 구현되어 있지 않기 때문에 주변 맵 정보는 알고 있다고 가정하였고, (2, 2)에 장애물이 존재한다고 가정하였다. 여기서 격자 맵의 한 칸의 실제 크기는 50cm로 로봇의 크기를 고려하여 설정하였다.

A* 알고리즘에 의해 구해진 경로는 (0, 0) → (1, 1) → (2, 1) → (3, 1) → (4, 2) → (4, 3) → (4, 4)로 각 경로점을 따라 이동로봇이 주행하게 된다.

3. 주행

A* 알고리즘을 통해 경로가 구해지면 베지어 곡선과 디지털 컨볼루션을 이용해 물리적 제한을 만족하며 얻어진 경로를 부드럽게 움직일 수 있는 궤적이 생성된다. 그림 12는 베지어 곡선을 이용해 얻어진 궤적이다. 최종 위치인 (4, 4)에서는 노약자를 정면으로 바라보며 로봇이 도착하도록 궤적을 계획했다.

그림 12에서 점의 간격이 조밀한 부분은 로봇이 천천히 움직이는 구간이며, 넓은 부분은 로봇이 빠르게 움직이는 구간이다. 그림 12와 같이 움직이기 위해 디지털 컨볼루션을 이용하여 그림 13과 같은 로봇의 중심에 대한 속도 프로파일을 생성하였다. TETRA의 최대 속도는 0.25m/s로 가

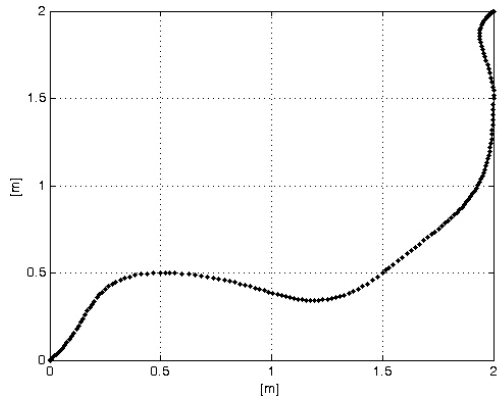


그림 12. 베지어 곡선을 이용한 궤적 계획.

Fig. 12. Trajectory planning using a Bezier curve.

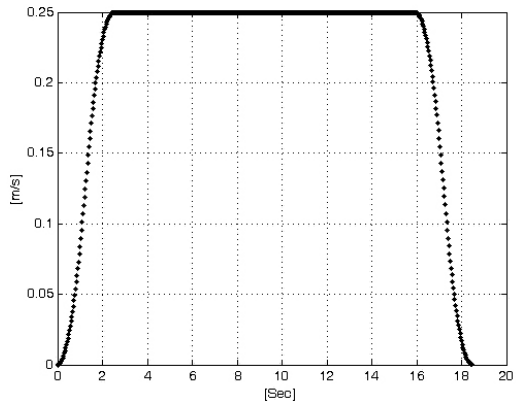


그림 13. 로봇 중심에 대한 속도 프로파일.
Fig. 13. Velocity profile at the center of the robot.

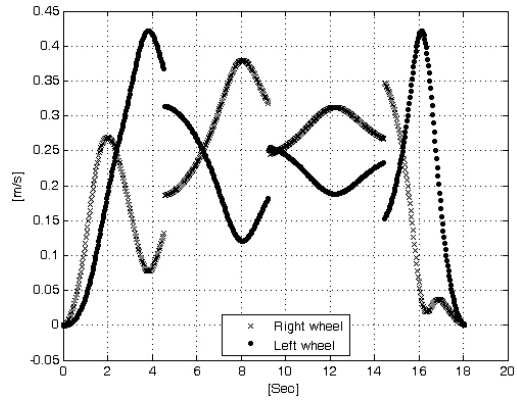


그림 14. 양 바퀴에 대한 속도 프로파일.
Fig. 14. Velocity profiles for the two wheels.

정하였다.

TETRA는 두 바퀴 차분구동형 이동로봇이기 때문에 중심에 대한 속도 프로파일을 양 바퀴로 나누어 속도 입력을 넣어주어야 한다[10-11]. 그림 13의 속도 프로파일을 역기구학을 이용해 두 바퀴로 나누었다. 두 바퀴에 대한 속도 프로파일은 그림 14와 같다.

그림 14를 보면 로봇의 최대 속도인 0.25m/s가 넘는 속도 명령이 생성 되며 불연속한 경로가 생성된다. 따라서 관절 공간이 두 바퀴의 속도 명령제한을 고려하고 연속된 경로 생성이 필요하며 이에 대한 논문을 별도로 발표하였다[11]. 본 연구에서는 실제 TETRA의 최대 속도는 2m/s이기 때문에 양 바퀴에 대해 0.25m/s가 넘는 속도 입력이 들어오더라도 추종이 가능하며 불연속적인 속도명령은 인터폴레이션하여 전체 시스템의 성능을 관찰하는 것을 목표로 하였다.

4. 시뮬레이션

그림 14와 태스크의 주기성 실험 결과를 바탕으로 시뮬레이션을 구성하였으며 시뮬레이터는 anyKode사의 Marilou를 사용하였다[13]. 실시간 기반과 비실시간 기반의 차이는 속도 명령을 입력할 때 태스크의 주기성 실험 결과에서 나온 오차 시간만큼 속도 명령이 지연되게 만들었다. 또한 모터 구동 태스크의 주기는 30ms로 설정하였다.

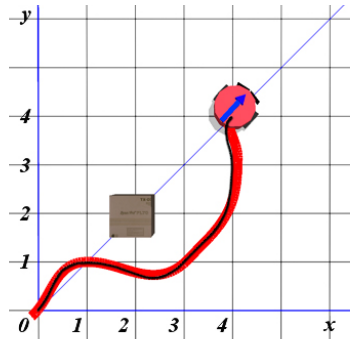


그림 15. 시스템에 따른 주행 시뮬레이션 (Xenomai).
Fig. 15. The driving simulation according to system (Xenomai).

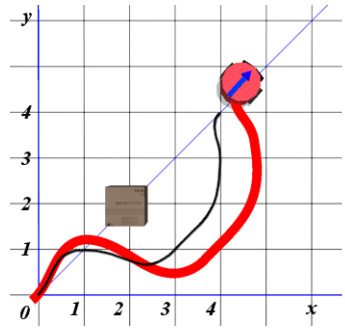


그림 16. 시스템에 따른 주행 시뮬레이션 (Linux).
Fig. 16. The driving simulation according to system (Linux).

그림 15는 Xenomai 기반에서의 시뮬레이션 결과이다. (2, 2)에 존재하는 장애물을 회피하며 주어진 궤적을 따라 목표 좌표인 (4, 4)에 도착하였다. 실선은 명령을 나타낸다.

그림 16은 범용 리눅스기반에서의 시뮬레이션 결과이다. Xenomai 기반에서와는 달리 태스크의 주기성 오차로 인해 정해진 궤적을 벗어나며 움직이는 모습을 볼 수 있다. 따라서 목표좌표인 (4, 4)에 멈추지 못하고 더 많은 거리를 주행하였다.

5. 실험

실제 주행 테스트를 위해 50cm 간격으로 맵을 구성하였다. 로봇의 초기 위치는 시뮬레이션과 동일하게 (0, 0)에 위치하며, 노약자의 위치는 (4, 4) 근처에 존재한다. 따라서 로봇은 노약자가 (4, 4) 근처에 존재하므로 (2, 2)에 있는 장애물을 회피하며 최종 목적지인 (4, 4)에 도착한다. 그림 17과 그림 18은 실시간 기반에서와 비실시간 기반에서의 실험 결과이다.

실시간 기반인 그림 17에서는 장애물을 회피하며 주어진 궤적을 따라 최종 목표에 도달하는 것을 볼 수 있지만, 비실시간 기반인 그림 18에서는 주어진 궤적을 벗어나 주행함으로써 장애물과 충돌하고 최종 목표보다 더 많은 거리를 움직여 노약자와 충돌 위험이 발생 하는 모습을 보였다.

주행 실험을 통하여 얻은 모터의 엔코더 값을 이용하여 이동로봇의 실제 주행 궤적을 그림 19에 나타내었다. Xenomai의 경우 Command와 유사한 궤적을 그리며 주행했지만, 비실시간 기반인 리눅스의 경우 태스크 주기성의 오차 때문에 Command를 많이 벗어난 궤적으로 움직였다.

그림 17. 실시간 시스템에서 모바일 로봇의 주행 실험.

Fig. 17. The driving experiment of the mobile robot at the real-time system.

그림 18. 비실시간 시스템에서 모바일 로봇의 주행실험.

Fig. 18. The driving experiment of the mobile robot at the non-real-time system.

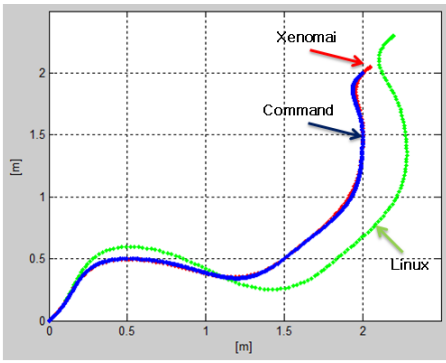


그림 19. 시스템에 따른 주행궤적 비교.

Fig. 19. Driving trajectory comparison according to system.

V. 결론

본 연구에서는 실시간 임베디드 리눅스 중 하나인 Xenomai를 지능형 모바일 로봇에 적용하여 비실시간 기반 지능형 모바일 로봇과 실시간 기반 지능형 모바일 로봇의 성능을 분석, 비교함으로써 실시간 시스템의 강점을 부각시켰다. 그리고 응용 어플리케이션으로 노약자 지원 시스템을 개발함으로써, 실시간 기반 지능형 모바일 로봇의 적용 모델의 예를 제시하였다. 이러한 연구 결과를 바탕으로 향후 다양한 지능형 모바일 로봇에 실시간 임베디드 리눅스를 적용시킨다면 시스템의 성능 및 신뢰도 향상을 기대할 수 있을 것이다.

REFERENCES

[1] B. W. Choi, D. G. Shin, J. H. Park, S. Y. Yi, and S. Gerald, "Real-time control arhcitecture using xenomai for intelligent service robot in USN environment," *Journal of Intelligent Service Robotics*, vol. 2, no. 2, pp. 139-151, 2009.

[2] T. Bird, Comparing Two Approaches to Real-time Linux, www.linuxdevice.com, 2002.

[3] I. Ripoll, RTLinux versus RTAI, www.linuxdevice.com, 2002.

[4] K. Dankwardt, Comparing Real-time Linux Alternatives, www.linuxdevice.com 2002.

[5] W. S. Liu, *Real-Time System*, Prentice Hall, 2000.

[6] D. Abbout, *Linux for Embedded and Real-time Applications*, Elsevier, 2006.

[7] A. Barbalace, A. Lunchetta, G Manduchi, M. Moro, A. Soppelsa, and C. Taliercio, "Performance Comparison of VxWorks, Linux, RTAI and XENOMAI in a Hard Real-time Application," *Proc. of Real-Time Conference 2007 15th IEEE-NPSS*, pp. 1-5, May 2007.

[8] J. H, Koh and B. W. Choi, "Real-time performance of real-time mechanisms for RTAI and xenomai in various running conditions," *Internal Journal of Control and Automation*, vol. 6, no. 2, pp. 139-151, 2013.

[9] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, pp. 100-107, 1968.

[10] G. Lee, J. Kim, and Y. Choi, "Convolution-based trajectory generation methods using physical system limits," *J. Dynamic Systems, Measurement, and Control, ASME*, vol. 135, pp. 1-8, 2013.

[11] G. J. Yang and B. W. Choi, "Joint space trajectory planning considering physical limits for two-wheeled mobile robots," *Journal of Institute of Control, Robotics and Systems (in Korean)*, vol. 19, no. 6, pp. 540-546, Jun. 2013.

[12] K. G. Jolly, R. S. Kumar, and R. Vijayakumar, "A bezier curve based path planning in a multi-agent robot soccer system without violating the acceleration limits," *Robotics and Automation Systems*, vol. 57, pp. 23-33, Jan. 2009.

[13] anyKode, Marilou Robotics Studio www.anykode.com

고 재 환

제어 · 로봇 · 시스템학회 논문지 제18권 제4호 참조.

양 길 진

제어 · 로봇 · 시스템학회 논문지 제19권 제6호 참조.

최 병 욱

제어 · 로봇 · 시스템학회 논문지 제19권 제6호 참조.