지능형 로봇을 위한 이중 커널 구조의 제어 시스템 구현 및 실시간 제어 성능 분석

Implementation of Dual-Kernel based Control System and Evaluation of Real-time Control Performance for Intelligent Robots

박 정 호, 이 수 영, 최 병 욱*
(Jeong-Ho Park, Soo-Yeong Yi and Byoung-Wook Choi)

Abstract : This paper implements dual-kernel system using standard Linux and real-time embedded Linux for the real-time control of intelligent robot systems. Such system provides more useful services including standard Linux thread that is easy to implement complicated tasks and real-time tasks for the deterministic response to velocity control. Here, an open source real-time embedded Linux, XENOMAI, is ported on embedded target board. And for interfacing with motor controller we adopted a real-time serial device driver. The real-time task was implemented with a priority to keep the cyclic control command for trajectory control. In order to validate deterministic response of the proposed system, the performance measurement of the delay in performing trajectory control with feedback loop is evaluated with non real-time standard Linux. The proposed software architecture is anticipated to take advantage of features in both standard Linux and real-time operating systems for the intelligent robot systems.

Keywords : embedded Linux, real-time embedded Linux, robot control architecture, XENOMAI

I. 서론

지능형 로봇이란 시각, 청각 등 감각 센서를 통해 외부 정보를 받아 스스로 판단하여 적절한 행동을 하는 로봇을 말한다. 지능형 로봇은 인공지능과, 생체공학, 신경회로망 이론, 퍼지 이론, 음성인식, 화상인식 기술, 센서 기술 등 여러 분야의 기술을 필요로 한다. 따라서 지능형 로봇은 기계 메커니즘만 중시되던 과거와는 달리 다양한 센서를 통해 정보를 얻는 채널과 이를 분석하여 스스로 행동하기 위한 처리 알고리즘, 그리고 처리된 결과에 따라 적절한 동작 명령을 전달하는 구동 시스템의 종합이다[1-2].

환경 인식과 정보 획득, 지능적 판단에 기초한 자율적인 행동을 특징으로 하는 지능 로봇은 인간을 지원하고, 어려운 상황에서 인간을 대신하거나 특수한 작업을 수행하는 등, 응용 분야의 다양성에 따라 로봇 제어 소프트웨어도 점점 복잡해지고 있다. 이러한 다양한 센서 데이터의 실시간 처리와 이와 관련된 복잡한 알고리즘을 처리하기 위해서 운영체제의 사용이 필수 요소가 되고 있다. 기존 로봇에 많이 사용되는 운영체제로는 윈도우즈, 또는 리눅스가 있으나, 이러한 운영체제는 원하는 작업의 실시간성을 보장하지 못하므로 필요한 행위를 원하는 시간에 수행하지 못하는 경우가 발생할 수 있다[1]. 지능형 서비스 로봇은 인간과 유기적으로 관계하기 때문에 이러한 예상치 못한 상황에 처하게 되면 인간에게 심각한 위협을 줄 수도 있으며, 원하는 성능을 얻지 못하는 상황이 발생한다. 이러한 문제점을 보완하기 위해서 실시간 운영체제(RTOS: Real-Time Operating System)를 적용하여 다중 작업 처리에 의하여 복잡한 작업을 소 작업 단위로 단순화하고, 각 소 작업의 우선 순위에 의하여 주어진 작업을 원하는 시간 내에 처리함으로써 성능을 개선하고 외부 반응에 적절하게 반응하고자 하는 연구가 많이 이루어지고 있다[3-4].

상용 실시간 운영체제는 검증된 성능, 체계화된 개발 환경, 그리고 체계적 기술 지원 등의 장점이 있지만, 기술이 종속될 수 있으며, 초기 개발비용이 많이 들고 개발의 유연성이 떨어지는 단점이 있다. 이에 비하여 실시간 임베디드 리눅스는 오픈 소스 프로젝트로서 많은 개발자들의 지식을 이용할 수 있으며, 특히 성능에서도 뒤떨어지지 아니하면서 저렴한 비용으로 실시간 시스템 구현이 가능하다는 장점이 있다[5]. 이에 비하여 체계적 기술 지원과 자료의 부족으로 인하여 개발자의 노력이 많이 필요하다는 단점이 있다. 실시간 임베디드 리눅스로는 RT-LINUX 프로젝트, RTAI(Real-Time Application Interface) 프로젝트, XENOMAI 프로젝트 등이 있다. 현재, RT-LINUX는 상용화를 이루었으며, RT-LINUX의 상용화에 대응하여 유연한 기능을 구현한 RTAI 프로젝트가 진행되었고, 최근에는 RTAI 프로젝트가 XENOMAI 프로젝트로 발전하면서 다양한 실시간 운영체제를 하나의 하드웨어 플랫폼에 포용하려는 노력이 진행되고 있다[6,7].

실시간 임베디드 리눅스를 지능형 로봇의 운영체제로 채택한 사례로는 RTAI 3.0와 실시간 디바이스 드라이버를 이용하여 이동로봇 제어시스템을 구현한 결과가 있다[4]. 이 연구에서는 RTAI의 실시간 디바이스 드라이버인 RT_COM을 이용해서 속도 프로파일 제어 명령을 전달하여 엔코더 값을 읽음으로써 리눅스와 RTAI의 성능 차이를 분석하였으며 RTAI를 이용하여 수동으로 조종하는 이동로봇을 구현 하였다.

최신 실시간 임베디드 리눅스인 XENOMAI를 적용하여 휴머노이드 로봇을 구현한 사례 연구도 발표되었다. 이 연구에서는 XENOMAI의 실시간 디바이스 드라이버인 RT_IEEE 1394를 이용하여 휴머노이드 로봇의 관절 구동을 담당하는 모터와 주 제어장치 사이에 통신을 함으로써 XENOMAI의

* 책임저자(Corresponding Author)
논문접수 : 2008. 2. 19., 채택확정 : 2008. 7. 22.
박정호 : ㈜다사로봇(jhpark@dasarobot.com)
이수영, 최병욱 : 서울산업대학교 전기공학과
(suylee@snut.ac.kr/bwchoi@snut.ac.kr)

경성 실시간성(hard real-time)을 검증하였고 실시간 제어를 통하여 동적 보행을 구현하였다. 또한 상용화 제품인 RT-Linux Pro와 성능을 비교/분석 하였다[8].

본 논문에서는 가장 최신의 실시간 임베디드 리눅스인 XENOMAI 2.3을 제어 장치에 포팅하고, 지능형 로봇의 주행 알고리즘 중에 위치 추종 알고리즘을 적용함으로써 향후 다양한 지능형 로봇에 적용하기 위한 기초적인 연구를 수행하고자 한다. 또한 실시간 임베디드 리눅스와 비실시간 임베디드 리눅스에서 다수의 부하 작업에 따른 주행 작업의 성능을 비교하였다. 주기적 특성을 필요로 하는 속도 제어기에서 실시간성에 따른 성능 분석도 함께 실시하였으며, 이러한 주기성의 차이는 실제 주행 거리의 오차로 나타나는데 부하 작업에 따른 성능 분석을 실시하였다. 그리고 마지막으로 실시간 임베디드 리눅스를 적용하여 원하는 궤적을 유지하며, 목표점에 도달하는 작업을 수행하였다.

논문의 구성은 다음과 같다: 먼저 II 장에서 기존의 실시간 운영체제와 XENOMAI 오픈 소스 프로젝트의 차이 그리고 XENOMAI 포팅 과정을 논하였다. 그리고 지능형 로봇에 적용하기 위한 실시간 디바이스 드라이버 구현 과정을 III 장에서 논하며, 다양한 관점에서 실시간 임베디드 리눅스 적용의 장점을 비교하도록 한다. 그리고 IV 장에서 포팅된 임베디드 리눅스 상에서 실시간 디바이스 드라이버를 적용한 지능형 로봇의 위치추정 알고리즘에 적용한 결과를 기술하도록 한다. 이와 같은 적용 결과는 향후 실시간 임베디드 리눅스를 이용하려는 개발자에게 아키텍처의 구성과 포팅 방법을 설명하며, 성능적인 관점에서 비교 결과를 이용하여 실시간 임베디드 리눅스 적용의 판단 기준을 제공할 것이다. 또한 지능형 로봇에 적용함으로써 비실시간 임베디드 리눅스와 동시에 실시간 임베디드 리눅스를 이용하여 지능형 서비스 로봇을 구현하고자 하는 개발자에게 개발 방법과 성능 그리고 장단점에 대한 판단 기준을 제공하리라 기대한다.

II. XENOMAI 기반의 이중 커널 구조 구현

실시간 시스템은 기존의 시스템과 달리 시스템의 동작이 논리적으로 완벽해야 하고, 또한 동작이 정해진 시간 안에 정확히 처리되어야 하는 시스템이다. 이것은 운영체제가 무조건 빨리 작업을 수행한다는 것을 의미하지는 않는다. 주어진 시간에 예측 가능한 시스템 함수를 이용하여 원하는 출력을 제공하는 시스템을 말한다. 또한 다중 작업을 지원함으로써 복잡한 작업을 작업 단위로 단순화할 수 있으며, 개발자는 작업에 우선순위를 부여하여 중요한 작업의 실시간성을 보장함으로써 전체 성능을 보장하는 시스템을 말한다. 즉, 주어진 특정 시간 내에 작업을 처리하도록 보장하는 운영체제로서 임베디드 시스템과 같이 제한시간에 영향을 받는 프로그램이나 로봇과 같이 다양한 외부 입력의 처리와 이에 반응하는 원하는 제어 성능을 보장하여야 하는 시스템에 적합한 운영체제이다.

XENOMAI 프로젝트는 2001년 8월에 프로젝트 그룹을 생성하고 활동을 시작하였다. RTAI/fusion으로 이름을 정하여 GNU/Linux 기반의 산업 등급에서 사용 가능한 실시간 공개 소프트웨어 플랫폼을 만들기 위한 것으로 2003년에는 기존의 또 다른 실시간 임베디드 리눅스 프로젝트인 RTAI와 융합되었다. 하지만 이 두 RTOS는 코어 레벨에서 많은 관점의 차이점을 보임에 따라 XENOMAI는 2005년에 다시 RTAI로부터 분리하여 독자적으로 프로젝트를 진행하였다[6].

XENOMAI는 또 다른 오픈 소스 프로젝트인 ADEOS (Adaptive Domain Environment for Operating Systems)를 기반으로한 운영체제이다[9]. 기본적으로 ADEOS는 하나의 하드웨어 플랫폼에 여러 개의 운영체제를 이용하기 위한 프로젝트로서 실시간 임베디드 리눅스인 XENOMAI와 비실시간 임베디드 리눅스인 Linux 2.6.17이 동시 사용가능하다. XENOMAI에서는 XENOMAI nucleus라고 불리는 추상적인 커널 코어를 기반으로 하여 다양한 RTOS 서비스를 제공하고 있다. XENOMAI nucleus 위에 여러 종류의 실시간 인터페이스를 적재함으로써 사용할 수 있으며, 인터페이스라 불리는 "skins"은 각각의 RTOS를 의미한다. 현재 적용 가능한 "skin"의 종류로는 Native API, POSIX API, RTAI, pSOS+, VRTX, VxWorks 등이 있으며, 이러한 "skin"을 이용하여 각각의 RTOS들을 에뮬레이션 할 수 있다. 어떠한 "skin"도 다른 API에 대하여 특권을 가지고 있지 않으며, 본 논문에서는 Native API를 적용하여 경성 실시간을 보장하였다.

본 논문에서 적용한 XENOMAI 성능을 커널과 API 입장에서 정리하면 다음과 같다.

XENOMAI nucleus의 주요 기능

- Mutil-threading support
- Basic synchronization support
- Timer and clock management
- Basic memory allocation

XENOMAI Native API의 주요 기능

- Task management
- Timing services
- Synchronization
- Messaging and communication
- Device I/O Handling
- Registry support

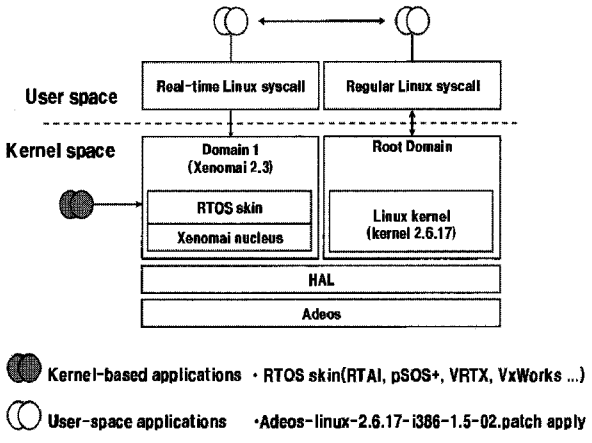


그림 1. XENOMAI 기반의 이중 커널의 제어 구조.
Fig. 1. Dual-kernel control architecture based on XENOMAI.

그림 1은 본 논문에서 구현한 이중 커널 구조의 제어 시스템 아키텍처로서 비실시간 임베디드 리눅스 Kernel 2.6.17 버전과 실시간 임베디드 리눅스 XENOMAI 2.3 버전을 동시에 사용 가능하도록 구현한 아키텍처이다. 본 논문에서는 i386 보드에 포팅하였으며, 가장 최신의 커널을 패치하였다. 따라서 사용자는 실시간 성을 요하는 작업은 실시간 임베디드 리눅스 시스템 함수를 이용하여 구현 가능하며, 그 이외의 작업은 일반적인 리눅스 함수를 이용하여 구현함으로써 개발의 유용성과 확장성을 보장하고 있다. 실시간 작업은 커널 영역과 사용자 영역에서 모두 가능하며 작업의 편리성과 성능의 보장이라는 측면을 고려하여 작업을 작성하여야 한다.

III. 실시간 디바이스 드라이버

본 논문에서 적용하고자 하는 이동 로봇은 주 제어장치와 속도 제어장치간의 직렬 통신에 의하여 정보를 주고 받는다. 따라서 이와 같은 직렬 통신에서 실시간성을 보장하기 위하여서는 실시간 디바이스 드라이버를 구현하여야 한다. 현재 XENOMAI 프로젝트에서는 RTDM(Real-Time Driver Model) 코어를 적용하여 실시간 디바이스 드라이버를 지원한다. 그림 2는 RTDM의 계층도를 나타내고 있다. RTDM은 추상화된 계층으로서, HAL(Hardware Abstract Layer)과 응용프로그램 사이에서 인터페이스 역할을 한다[10].

XENOMAI에서는 직렬 통신을 위한 RT_COM, 실시간 리눅스를 위한 네트워크 성능을 제공하는 RT_NET, 소켓 기반의 CAN(Controller Area Network)을 위한 RT_CAN, IEEE1394를 위한 RT_FireWire 등과 같은 실시간 디바이스 드라이버가 오픈 소스 프로젝트 형태로 지원되고 있다. 이와 같은 디바이스 드라이버는 실시간성 보장을 위하여 함수의 지연(blocking) 기능이 배제되어 있으며, 각 툴 체인과 버전간의 호환성을 확보하는 작업이 필요하게 된다.

본 논문에서는 실시간 디바이스 드라이버 중 실시간 직렬 드라이버인 RT_COM을 적용하였으며, native skin에는 RTDM이 포함되어 있다.

구현된 실시간 디바이스 드라이버와 비실시간 디바이스 드라이버의 성능을 비교, 분석하기 위하여 직렬 통신을 사용하는 모터 제어 장치에 속도제어 실험을 하였다. 모터 제어 장치는 주제어 장치와 57,600bps의 속도로 직렬 통신을 하며,

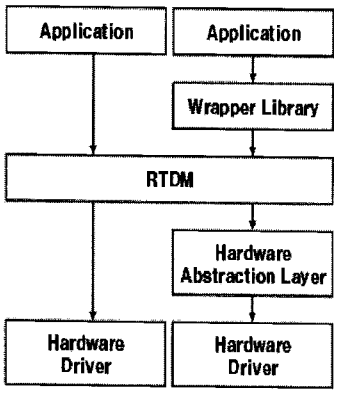


그림 2. RTDM 계층.
Fig. 2. RTDM layers.

엔코더는 한 바퀴에 4,000펄스가 출력된다. 한 바퀴를 회전하였을 때의 이동거리는 약 308mm이며, 모터의 최대속도는 300mm/s이다. 속도 제어를 위한 작업의 주기는 100ms이다. 작업량(workload)의 영향을 위하여 가상적인 실수 연산의 작업을 30ms의 주기로 동작하였으며, 실시간 임베디드 리눅스의 경우는 우선순위 조정을 통하여 속도 제어 타스크의 실시간성을 보장하면서 실험하였다.

속도 명령에 따른 거리 명령 궤적은 그림 3과 같다. 이상적인 속도 제어기는 일정한 주기성에 따른 속도 명령을 추적하여 결과적으로 그림 3과 같은 이상적인 주행을 수행하여야 한다.

그림 4는 범용 임베디드 리눅스에서 일반적인 직렬 통신을 이용하여 주 제어장치와 속도 제어장치의 명령을 수행한

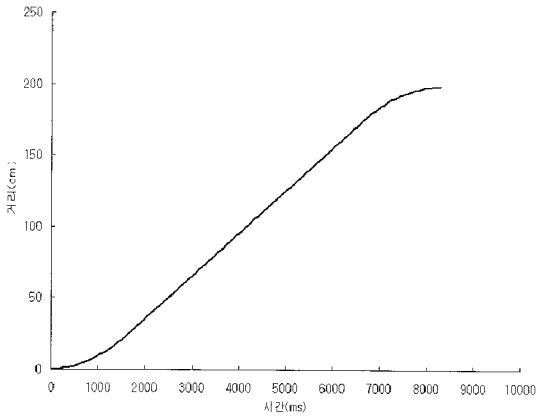
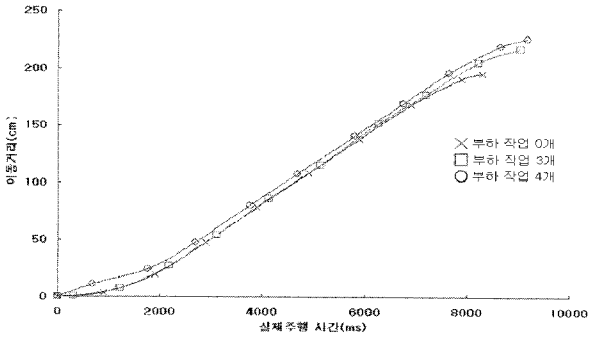
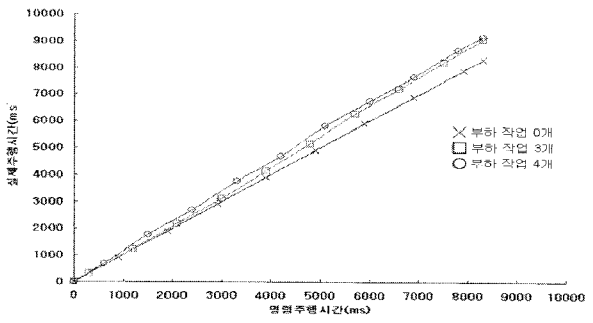


그림 3. 시간에 따른 명령 거리 궤적.
Fig. 3. Command trajectory according to time.



(a) 실제주행시간 - 이동거리.



(b) 명령주행시간 - 실제주행시간.

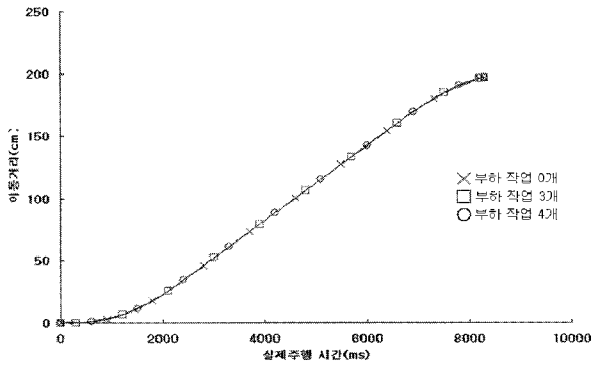
그림 4. 비실시간 리눅스에서의 성능.
Fig. 4. Performance evaluation of non real-time Linux kernel.

결과이다. 그림 4(a)는 연산 부하 량에 따른 모터 이동거리 실제 궤적이고, 그림 4(b)는 이상적 속도 명령 시간에 대한 실제 수행 명령 시간에 대한 결과이다. 그림에서 나타난 바와 같이 연산 부하 작업의 수가 증가함에 따라 주기 작업의 명령이 지연되어서 원하는 명령 값에서 벗어남을 볼 수 있다. 이는 범용 임베디드 리눅스에서 모든 작업에 균등한 우선순위에 의한 스케쥴링 기법에 기인한 것으로, 다른 연산 부하 작업이 증가함에 따라 모터제어 작업에 스케쥴링이 원하는 시간이 이루지지 못하게 된다. 따라서 정해진 샘플링 주기가 유지되지 못하고 지연되기 때문이다. 결과적으로 속도 명령 작업 주기성의 지연에 따라 이동거리가 명령에 충실히 따르지 못하고 지연됨으로써 제어 성능에 심각한 에러를 초래하게 된다.

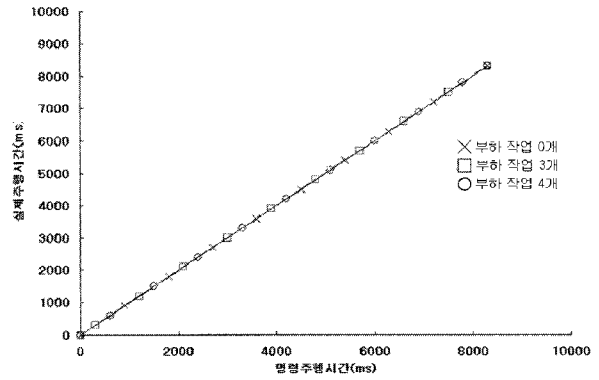
표 1. 비실시간 리눅스의 부하작업의 수에 따른 모터제어 성능.

Table 1. Motor control performance of non real-time Linux according to the number of workloads.

타스크(개)	실제 이동거리 /명령거리 (cm)	실제 작업시간 /명령시간 (ms)
0	197/197	8,300/8,300
3	217/197	9,039/8,300
4	226/197	9,171/8,300



(a) 실제주행시간 – 이동거리.



(b) 명령주행시간 – 실제주행시간.

그림 5. 실시간 임베디드 리눅스에서의 성능.

Fig. 5. Performance evaluation of real-time embedded Linux.

부하 량에 따른 주기성의 오류로 인한 거리 오차를 자세히 나타내기 위하여 비실시간 리눅스의 모터 제어 성능을 표 1에 정리하였다. 제어 명령 거리는 197cm 이나 실제 작업시간이 부하 작업에 따라 지연됨에 따라 이동거리 역시 지연되고 있음을 알 수 있다. 이와 같은 성능은 위치 제어가 절대적인 지능형 이동 로봇의 경우 치명적 결함을 가져올 수 있다.

이에 비하여 그림 5에서 보인 실시간 리눅스 제어 시스템의 경우는 부하 작업의 수에 상관없이 속도 명령의 주기성이 변하지 않고 작업을 수행하고 있으며, 이에 따른 거리도 원하는 명령을 충실히 따르는 모습을 확인할 수 있다. 작업 구성은 실시간 직렬 디바이스 드라이버를 이용하였고, 주기적으로 작동하는 속도 제어 타스크는 높은 우선순위를 가지도록 구성하였다. 그리고 부하 작업은 그림 4와 동일하게 구성하였다. 이와 같은 경우 그림과 같이 연산 부하작업의 수와 무관하게 이동거리 및 총 작업시간이 명령 값을 충실히 수행함을 알 수 있다. 이와 같은 결과는 실시간 타스크가 우선순위에 의하여 다른 부하에 우선하기 때문이며, 따라서 실시간 커널과 실시간 디바이스 드라이버의 장점을 확인할 수 있다. 또한 정밀한 위치제어를 위하여서는 실시간 운영체제를 사용하여야 함을 실험적으로 확인하였다.

IV. 주행 실험 및 검토

본 절에서는 III 장에서 기술한 이중 커널 구조를 이용하여 이동 로봇의 주행제어 시스템을 구현하였다. 실시간 임베디드 리눅스는 XENOMAI를 이용하고 있으며 직렬 통신을 위하여 실시간 디바이스 드라이버를 이용하고 있다. 또한 기타 이동 로봇을 위한 작업은 비실시간 임베디드 리눅스에서 표준 쓰레드로 구성하였다.

앞선 III 장에서의 실험은 오픈 루프에서 직렬 통신에 따른 성능을 비교한 것이며, 본 장에서의 결과는 폐 루프에서 위치 피드백을 통하여 정밀 제어 알고리즘을 구현한 결과이다. 비록 오픈 루프에서도 실시간 임베디드 리눅스로 구현한 제어 성능을 비교하였지만, 이중 커널 구조의 유용성을 확인하기 위하여서는 범용 비실시간 임베디드 리눅스에서의 작업

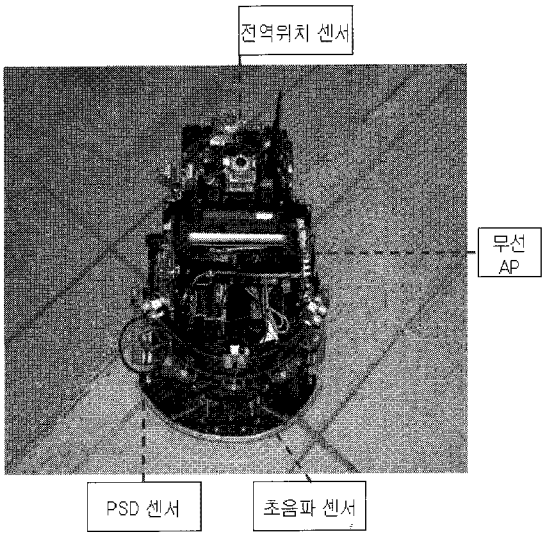


그림 6. 이동로봇.

Fig. 6. Mobile robot.

과 실시간 임베디드 리눅스에서 위치 피드백 제어를 동시에 수행함으로써 제안된 구조의 유용성을 확인할 수 있다.

그림 6은 실제 실험에 적용한 이동로봇으로, 장애물 감지를 위한 초음파 센서와 PSD(Position Sensitive Device) 센서는 전방에 배치하고, 전역위치 센서는 로봇의 구동 중심축에 위치하게 되며, 구동 축은 로봇의 중심축에 위치하게 되며 원격제어장치와 통신을 위한 무선 AP(Access Point)는 로봇의 상단에 위치하게 된다. 전역위치 센서로는 (주)하기소닉의 스타게이저를 채택하였다[11].

그림 7은 리눅스 커널 2.6.17에 XENOMAI 2.3 기반으로 본 논문에서 적용한 이동 로봇의 시스템 구성도이다. 모터 모듈과 전역위치 센서 모듈은 실시간 직렬 통신을 통하여 메인 프로세서와 통신을 하게 되고 기타 위치 센서는 일반적인 직렬 통신을 통하여 통신하여 작업도 비실시간 임베디드 리눅스에서 구현하였다. 또한 외부 모니터링 시스템을 위한 연결은 무선 인터넷을 이용하며 비실시간 임베디드 리눅스에서

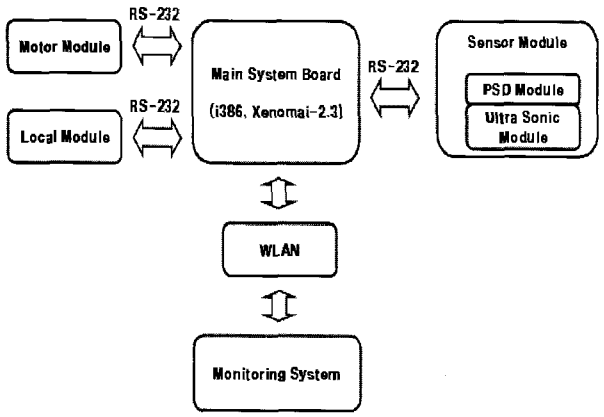


그림 7. 이동 로봇 시스템 구성도.

Fig. 7. Control system configuration of the mobile robot.

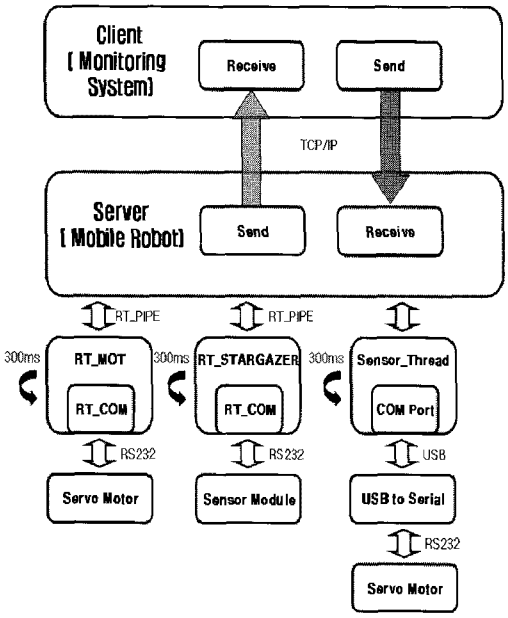


그림 8. 이동 로봇 소프트웨어 구조.

Fig. 8. Software structure of the proposed system.

쓰레드로 구현하였다. 이와 같이 실시간성을 위한 제어는 실시간 타스크로 구현하고, 다양한 응용 프로그램은 확장성이 용이한 비실시간 임베디드 리눅스에서 작업을 수행한다.

그림 8은 본 논문에서 사용된 로봇 소프트웨어 구조로 3계층 구조를 이루고 있다. 원격 모니터링 시스템과 주 제어장치는 무선 LAN을 이용한 TCP/IP 통신을 하며, 주 제어장치에 구현된 소프트웨어는 일반 비실시간 임베디드 리눅스 기반의 소프트웨어와 실시간 임베디드 리눅스 기반의 작업으로 구성되어 있다. 일반 리눅스 작업은 원격 모니터링 시스템과의 통신, 일반적인 이동 로봇의 주행 알고리즘 그리고 그림 7에 나타낸 바와 같이 모터 보드에 장착되어 있는 일반적인 센서 처리를 수행하고 있다.

이에 비하여 실시간 임베디드 리눅스는 실시간성을 요하는 작업을 중심으로 구현하였다. 즉 모터 제어 장치와 로봇의 위치 추정을 위한 센서 처리를 실시간 타스크로 구현하였다. 구현된 실시간 작업은 RT_MOT와 RT_STARGAZER로서 실시간 디바이스 드라이버인 RT_COM을 이용해서 전역위치 센서의 값을 읽어 들이고 이 값을 가공해서 RT_MOT 작업으로 위치 좌표를 전송한다. RT_MOT는 RT_STARGAZER로부터 전역 좌표를 받아 주행 알고리즘을 수행 한 후 모터의 속도 값을 실시간 디바이스 드라이버인 RT_COM을 이용해서 모터의 제어 명령 전송 및 모터 모듈로부터 확인 값을 전송받는다. 수행 주기는 300mm로 로봇의 주행 속도를 감안하여 설계되었다.

비실시간 임베디드 리눅스에서 구현된 알고리즘은 다음과 같이 세 개의 쓰레드로 구성 하였다.

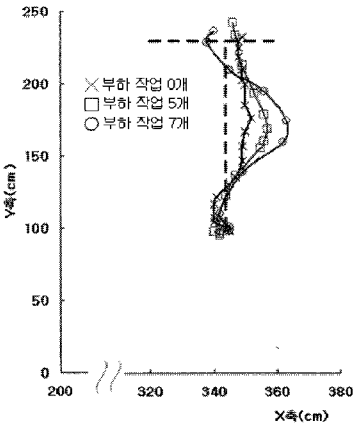
-Sensor_Thread: USB포트에 USB-직렬(USB to Serial) 변환기를 이용하여 초음파센서와 PSD센서 모듈로부터 센서 값을 전송 받는다. 전송 받은 데이터는 가공과정을 거쳐 RT_PIPE를 통해 RT_MOT TASK로 전송된다.

-Server_Receive_Thread: 무선 LAN을 통해 전송 받은 데이터를 가공하여 RT_PIPE를 통해 RT_MOT 작업으로 전송한다.

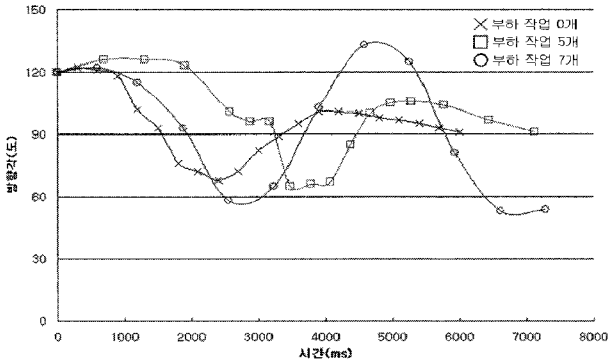
-Server_Send_Thread: 모니터링 시스템에 로봇의 현재위치, 장애물 상태 정보 그리고 작업의 종료 여부 정보를 무선 LAN을 통해 전달한다.

다음 실험은 그림 8과 같은 구성에서 모든 타스크를 비실시간 임베디드 리눅스에서 구현하여 주행 실험을 수행한 예이다. 이동 로봇의 직진성을 검증하기 위한 것으로 이동로봇은 x축 344cm, y축 100cm에서 출발하여 x축 344cm, y축 230츠 지점에서 정지 하여야 한다. 로봇의 시작 각도가 90도인 경우 피드백 제어에 의해서 부하 작업에 따른 차이점을 볼 수가 없어, 이동로봇이 출발할 때 로봇의 각도를 120도 지점에서 출발 시켰다.

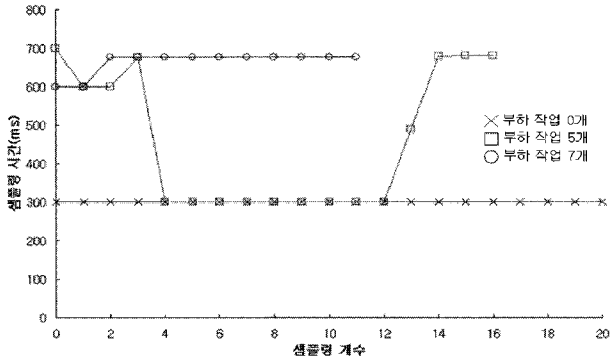
본 실험에서는 그림 8에 나타난 타스크외에 컴퓨터 부하를 늘리기 위한 가상의 실수 연산 쓰레드를 추가하여 영향을 분석하였다. 그림 9(c)는 300ms로 설계된 속도 제어 쓰레드의 실제 수행된 샘플링 시간으로 원하는 주기작업에 비하여 작업이 지연되고 있음을 알 수 있다. 이와 같이 다른 작업에 의한 지연의 결과는 그림 9(a) 및 9(b)와 같이 주행 성능에서 원하는 위치와 각도로 이동하지 못하는 결과로 나타나고 있다. 적절한 피드백이 원하는 시간에 이루어지지 못함으로 인



(a) 직교 좌표 $(x-u)$.



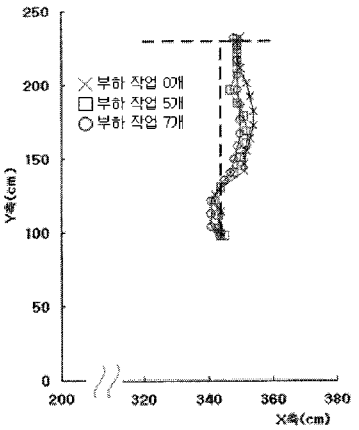
(b) 방향각 – 시간.



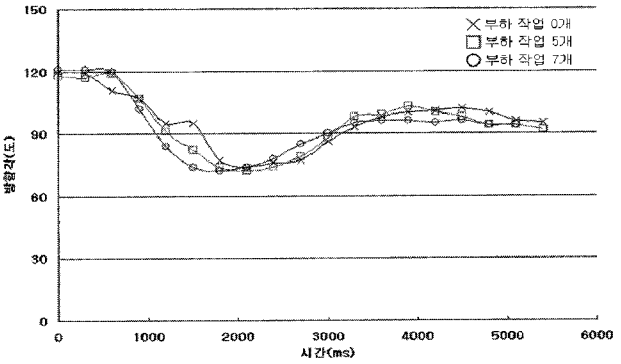
(c) 샘플링 시간 - 샘플링 개수.

그림 9. 비실시간 리눅스에서의 주행 성능 분석.

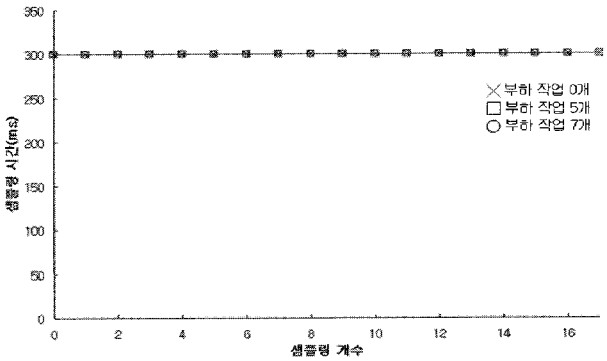
Fig. 9. Performance evaluation of trajectory control in non real-time Linux kernel.



(a) 직교 좌표 $(x-u)$.



(b) 방향각 – 시간.



(c) 명령주행시간 – 실제주행시간.

그림 10. 실시간 임베디드 리눅스에서의 주행 성능 분석.

Fig. 10. Performance evaluation of trajectory control in real-time Linux kernel.

하여 원하는 궤적과 상당이 다른 주행 성능을 보이고 있으며, 이는 속도 명령을 수행하는 작업이 다른 작업으로 인하여 스케쥴링이 지연되어서 나타난 결과이다.

이에 비하여 실시간 임베디드 리눅스 기반에서 실시간 타스크로 구현된 속도 제어 타스크의 경우는 그림 10(a)에서와 같이 부하 작업의 개수에 무관하게 유사한 궤적을 그리며 목표점에 추종하는 모습을 보이고 있다. 그림 10(b)에서와 같이 로봇의 방향각이 부드럽게 90도 부근에서 추종하는 모습을 보이고 있다. 이러한 결과는 그림 10(c)에서와 같이 속도 제어 타스크의 주기가 300ms로 일정함을 알 수 있다. 그리고 총 주행 시간은 5,400ms로 일정하다. 이와 같은 결과는 우선순위에 의해 속도 제어 타스크가 실시간성을 유지하면서 수행된 결과로 제어 성능에 직접적인 영향을 가져오고 있다. 그림에서 부하 작업에 따라 약간씩 성능 변화를 보이는 것은 위치 추정 센서의 오류와 바닥의 미끌림에 의한 결과이며, 실시간성의 지연에 의한 것은 아니다.

V. 결론

지능형 서비스 로봇은 다양한 센서를 이용하고 복잡한 알고리즘 수행 등으로 인하여 편리한 프로그래밍 환경과 확장성이 필요하다. 이와 같은 요구 사항에 리눅스가 장점을 가지고 있어서 로봇의 운영체제로 많이 사용되고 있다. 그러나

속도 제어와 같이 실시간성을 요구하는 작업의 필요로 인하여 실시간 운영체제 또한 많이 사용되고 있다. 그러나 대부분의 상용 실시간 운영체제는 개발비용 증가와 로열티 비용의 증가, 기술 종속성 등으로 인하여 개발의 유연성이 떨어지는 단점이 있다.

본 논문은 이와 같은 두 가지 요구를 만족하기 위하여, 이중 커널 구조의 제어 구조를 구현하고 성능 분석을 통하여 지능형 제어 로봇을 위한 제어 플랫폼으로서의 유용성을 검증하였다. 실시간 임베디드 리눅스로는 XENOMAI를 포팅하고, 실제 이동 로봇의 인터페이스를 위하여 실시간 직렬 통신 디바이스 드라이버를 구현하였다. 또한 계층적 구조의 지능형 서비스 프로그램을 수행하여 비실시간 임베디드 리눅스의 경우와 제안된 시스템의 성능 분석을 통하여 유용성을 검증하였다.

결론적으로 XENOMAI를 사용하여 실시간 성능을 보장함과 동시에 유용한 일반 리눅스 함수를 동시에 이용함으로써 시스템의 성능을 향상 시켰다. 또한 기존 연구에서 보고된 바와 같이 상용 실시간 운영체제의 평균지연 시간에 뒤지지 않는 성능을 보장하면서 우선순위에 의하여 많은 일반 리눅스 작업에 대해서도 실시간성을 보장함을 보였다[5]. 또한, 실시간 디바이스 드라이버와 비실시간 디바이스 드라이버를 비교함으로서 실시간 시스템의 장점을 확인하였고, 실시간 작업과 비실시간 리눅스 프로그램을 분리 적용하여 지능형 로봇의 주행 알고리즘에 적용함으로서 지능형 로봇을 위한 유용성을 검증하였다.

참고문헌

[1] 김문상외 35인. "차세대 지능로봇 핵심 기술," Jinhan M&B, 2006.

[2] 김진오, "지능형 로봇과 차세대 성장동력," 무선관리단 전파지, 5. 2004.

[3] G. Bruzzone, , M. Caccia, G. Ravera and A. Bertone, "Standard Linux for embedded real-time robotics and manufacturing control systems," *Robotics and Computer-Integrated Manufacturing*, In Press, Corrected Proof, Available online 15 January 2008

[4] 신은철, 최병욱, "실시간 임베디드 리눅스를 이용한 이동 로봇 플랫폼 구현," 제어 및 자동화 시스템 공학회, 제 12권 2호, pp. 194-200, 2006.

[5] A. Barbalace, A. Luchetta, 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

[6] XENOMAI project homepage: http:// www.XENOMAI.org

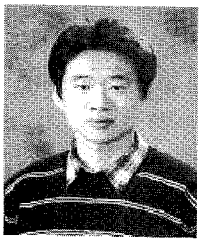
[7] RTAI project homepage: http://www.rtai.org

[8] M.O.F Sarker, C. H. Kim, J. S. Cho and B. J. You, "Development of a Network-based Real-Time Robot Control System over IEEE 1394: Using Open Source Software Platform," *Proc. of IEEE International Conference on Mechatronics*, pp. 563-568, July 2006.

[9] Adeos project homepage: http://home.gna.org/adeos

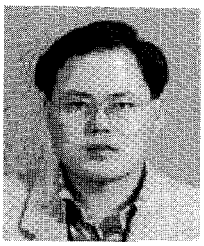
[10] J. Kiszka, "The Real-Time Driver Model and First Applications," *Proc. of 7th Real-Time Linux Workshop*, 2005.

[11] 하기소닉 홈페이지: http://www.hagisonic.com/



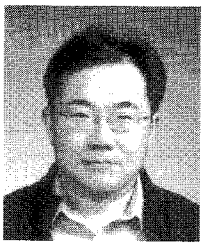
박 정 호

2006년 선문대학교 기계 및 제어공학부(공학사). 2008년 서울산업대학교 전기공학과 석사(공학석사). 2008~현재 ㈜다사로봇 연구원. 관심분야는 유비쿼터스 컴퓨팅, 지능형 서비스 로봇.



최 병 욱

1963년 2월 13일생. 1986년 한국항공대학교 항공전자공학과 졸업(공학사). 1988년 한국과학기술원 전기 및 전자공학과 졸업(공학석사). 1992년 동 대학 대학원 박사과정 졸업(공학박사). 1992년~2000년 LG산전 연구소 엘리베이터 연구실장. 임베디드 시스템 연구팀장. 2001년~2003년 ㈜임베디드웹 대표이사. 2000년~2005년 선문대학교 제어계측공학과 부교수. 2005년~현재 국립서울산업대학교 부교수. 2007-2008 Nanyang Technological University, Senior Fellow. 관심분야는 소트프웨어 엔지니어링, 내장형 시스템, 실시간 운영체제, 임베디드 리눅스, 필드버스 및 분산 제어 시스템, 지능제어로봇.



이 수 영

1988년 연세대학교 공과대학 전자공학과(공학사). 1990년 한국과학기술원 전기 및 전자공학과(공학석사). 1994년 한국과학기술원 전기 및 전자공학과(공학박사). 1995~1999년 한국과학기술연구원 휴먼로봇연구센터 선임연구원. 1999~2007년 전북대학교 전자정보공학부 부교수. 2007~현재 서울산업대학교, 전기공학과 부교수. 관심분야는 보행로봇 시스템, 보행설계, 로봇비젼 등임.