실시간 임베디드 리눅스를 이용한 이동 로봇 플랫폼 구현

Implementation of a Mobile Robot Control Platform using Real-Time Embedded Linux

신 은 철*, 최 병 욱
(Eun-Cheol Shin and Byoung-Wook Choi)

Abstract : The SoC and digital technology development recently enabled the emergence of information devices and control devices because the SoC presents many advantages such like lower power consumption, greater reliability, and lower cost. However, it is nearly impossible to use the SoC without operating systems because the SoC is included with many peripherals and complex architecture. It is required to use embedded operating systems and real-time operating systems may be used as an embedded operating system. So far, real-time operating systems are widely used to implement a Real-Time system since it meets developer's requirements. However, real-time operating systems have disadvantages including a lack of standards, expensive development, and license. Embedded Linux is able to overcome their disadvantages. In this paper, the implementation of control system platform for a mobile robot using real-time Embedded Linux is described. As a control hardware system platform, XScale board is used. As the real-time Embedded Linux, RTAI is adopted which is open source and royalty free, and supports various architectures and real-time devices, such like real-time CAN and real-time COM. This paper shows the implementation of RTAI on XScale board that means the porting procedure. We also applied the control system platform to the mobile robot and compared the Real-Time serial driver with non real-time serial driver. Experimental results show that that using RTAI is useful to build real-time control system with powerful functionalities of Linux.

Keywords : real-time operating system, real-time serial driver, real-time embedded linux, mobile robot, control system platform

I. 서론

최근 정보기기 및 제어기기와 같은 임베디드 시스템(embedded system)이 소형화, 경량화되면서도 고성능의 다양한 기능을 제공하고 있다. 이러한 기기에는 주로 MPU (Micro Processing Unit), 메모리, DSP (Digital Signal Processor), 주변 장치 등이 하나의 칩으로 만들어진 SoC (System on Chip)가 사용되고 있다. SoC는 하나의 반도체 칩으로 만들어지기 때문에 소비 전력이 적고, 검증된 IP (Intellectual Property)를 사용하여 개발되므로 신뢰성이 높으며, 개발비용이 절감되는 이점이 있다. 그러나 SoC는 다양한 기능이 하나의 칩으로 구현되었기 때문에 개발에 많은 노력과 시간이 필요한 단점을 가지고 있다.[1] SoC의 단점을 보완할 수 있는 방법으로 임베디드 운영체제가 적용되고 있다. 임베디드 운영체제는 높은 신뢰성과 강력한 기능 가지고 다양한 하드웨어를 지원함으로써 최근 SoC 기술과 더불어 주요한 기술로 부각되고 있다[2-4].

임베디드 시스템에는 임베디드 운영체제 외에 실시간 운영체제가 사용되고 있다. 실시간 운영체제는 시간의 제약을 만족하는 운영체제로 군사장비 분야에서 많이 사용되고 있으며, 최근에는 로봇, 자동화 시스템 등의 여러 분야에서 사용되고 있는 추세이다. 그러나 실시간 운영체제는 대부분이 상용 운영체제로 기술이 종속될 수 있으며, 초기 개발비용이 많이 들고 개발의 유연성이 떨어지는 단점이 있다[5].

본 논문에서는 제어시스템 플랫폼을 구현하기 위해 SoC 기반의 하드웨어에 실시간 임베디드 리눅스를 적용하고 이를 이용하여 이동 로봇을 구현한다. 제어시스템 플랫폼에는 최근 정보 단말기에서 가장 많이 사용되는 SoC 중 하나인 인텔의 PXA255가 사용되었으며, 실시간 임베디드 리눅스로는 RTAI(Real-Time Application Interface)를 이용한다. RTAI는 실시간성을 보장하는 임베디드 리눅스로서 공개된 소프트웨어이다. 공개된 RTAI를 개발된 하드웨어에 적용하기 위해서는 부트로더(boot loader)를 구현하고 공개된 커널에서 개발된 하드웨어 의존적인 부분을 수정한다. 그리고 이더넷과 같은 주변 장치를 위한 디바이스 드라이버와 실시간 시리얼 드라이버를 작성하고, 파일 시스템을 구축한다[6]. 그리고 실시간 임베디드 리눅스를 적용한 제어시스템 플랫폼의 유용성을 테스트하기 위해 실시간 디바이스 드라이버와 비실시간 디바이스 드라이버의 성능을 비교하고, 이동 로봇에 적용하였다.

II. 실시간 운영체제와 RTAI

실시간 운영체제란 정해진 시간에 주어진 작업을 수행하도록 지원하는 소프트웨어이다. 이것은 운영체제가 무조건 빨리 작업을 수행한다는 것을 의미하지 않는다. 주어진 시간에 예측 가능한 시스템 함수를 이용하여 원하는 출력을 제공하는 시스템을 말한다. 이러한 실시간 운영체제는 시간을 제어하여 다중 태스크(multi task) 구조의 프로그램을 지원하고, 실시간 시스템을 구현하기 위해 태스크들간의 통신, 동기화 그리고 스케줄링(scheduling) 등의 메커니즘(mechanism)을 제공한다.

그러나 실시간 운영체제는 많은 장점에도 불구하고 사용하기 어렵고 고가이며, 또한 대부분의 경우 소스가 공개되지 않을 뿐만 아니라 프로세서를 변경할 경우 커널과 개발 환경을 재구성하여야 하는 단점을 내포하고 있다. 이러한 문제점을 해결할 목적으로 임베디드 리눅스가 사용되고 있으나, 임베디드 리눅스는 실시간 시스템을 지원하지 않는 단점을 가

* 책임저자(Corresponding Author)
논문접수 : 2005. 4. 18., 채택확정 : 2005. 9. 28.
신은철 : 한국생산기술연구원 로봇기술개발본부(unchol@kitech.re.kr)
최병욱 : 서울산업대학교 전기공학과(bwchoi@snut.ac.kr)

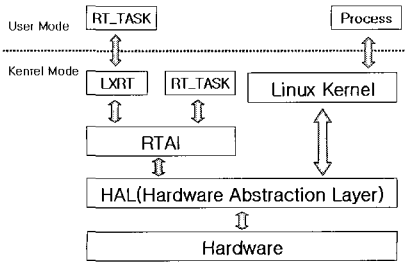


그림 1. RTAI의 구조.
Fig. 1. The structure of RTAI.

지고 있다. 이와 같은 이유로 임베디드 리눅스의 장점을 확보하면서 실시간성을 확보하려는 연구가 많은 개발자들에게서 이루어져왔으며, 특히 RTLinux와 RTAI 는 많은 사용자들을 확보하고 있다.

RTAI는 1996년 이탈리아의 DIAPM의 Paolo Mantegazza에 의해 RTLinux를 기본 개념으로 개발되어 현재는 RTLinux와 별개로 공동 커뮤니티의 프로젝트로써 개발의 진행되고 있다. RTAI는 사실 실시간 운영체제가 아니라 실시간 태스크를 위한 인터페이스이다. 즉 RTAI를 사용하기 위해서는 운영체제가 필요하다. 본 논문에서는 RTAI와 인터페이스 할 수 운영체제로 리눅스를 사용한다[7,8].

그림 1은 RTAI의 기본 구조이다. 리눅스 커널은 하나의 타스크로 취급된다. RTAI와 리눅스 커널은 HAL (Hardware Abstract Layer)에 의해서 하드웨어와 인터페이스 할 수 있다. HAL은 RTAI와 리눅스 커널을 실행시킬 때 RTAI에서 실행되는 실시간 태스크(RT_TASK)를 우선적으로 실행시키며, 리눅스 커널을 가장 낮은 우선순위로 동작시킨다. 즉, 실시간으로 동작해야 하는 실시간 태스크를 우선적으로 실행한 후 리눅스 커널과 리눅스 커널에서 동작하는 프로세스들을 낮은 우선 순위 타스크로써 실행한다[9,10].

리눅스는 성능을 우선시하는 운영체제로써 라운드 로빈 방식으로 스케줄러가 동작한다. 즉 각각의 프로세스 들이 일정 시간마다 돌아가면서 실행이 되기 때문에 실시간성이 보장되지 않는다. 최근 커널 버전이 2.6.x로 발전하면서 선점형 기능이 추가되긴 하였지만 경성 실시간 시스템(hard-real time system)을 지원하지 않기 때문에 제어 시스템에서 사용하기 무리가 있다. 하지만 RTAI는 여타 다른 실시간 운영체제와 마찬가지로 태스크 선점 및 우선순위 설정/상속 등의 기능을 제공함으로써 경성 실시간 시스템을 요구하는 제어 시스템에 적합하다. 리눅스 기반의 실시간 운영체제로써 RTLinux가 있다. 하지만 RTAI는 무료/오픈 소스인 반면 이 운영체제는 상용이기 때문에 제품 단가가 올라갈 뿐만 아니라 지원되는 프로세서 또한 RTAI에 비해 한정되어 있다.

III. 실시간 임베디드 리눅스 포팅 및 RTAI 설치

1. 실시간 임베디드 리눅스 포팅

RTAI 는 실시간 시스템을 구현하기 위한 인터페이스이기 때문에 운영체제가 필요하다. 본 논문에서는 RTAI를 위한 운영체제로 임베디드 리눅스를 사용하였다. 따라서 개발과정은 일반적인 임베디드 리눅스의 포팅과정과 유사하다[2-5]. 그림 2는 RTAI 포팅 및 설치 과정을 나타낸다.

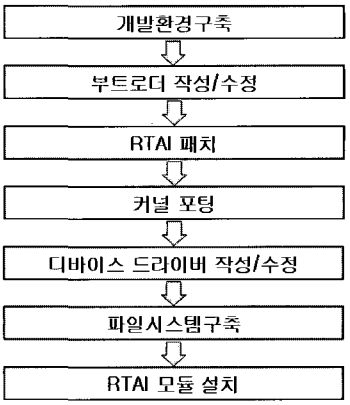


그림 2. RTAI 포팅 및 설치 과정.
Fig. 2. A process of porting and installing the RTAI.

임베디드 시스템은 PC와 달리 개발환경과 실행환경이 서로 다르기 때문에 교차 개발환경을 구축해야 한다. 교차 개발환경에는 임베디드 시스템의 프로세서를 위한 교차개발 툴 체인과 시리얼 포트를 이용한 터미널 프로그램, 부트로더 및 커널 그리고 파일시스템을 다운로드하기 위한 JTAG 및 TFTP 서버 등이 포함된다.

부트로더란 하드웨어를 초기화하는 프로그램으로써 인터럽트 벡터(interrupt vector)와 메모리 맵(memory map) 설정 등의 작업을 수행한다. 본 논문에서는 Blob (Boot Loader Object) 부트로더를 사용하였다. Blob은 StrongARM과 XScale(PXA250과 PXA255)을 지원하는 부트로더로써 하드웨어 초기화외에 커널과 파일시스템 이미지를 이더넷을 통해 다운받아서 플래시 메모리에 저장하는 기능이 추가되어 있다. Blob은 인텔에서 개발한 XScale과 StrongARM의 평가보드를 위한 부트로더이기 때문에 프로세서에서 제공하는 모든 기능이 포함되어 있다. 따라서, 이를 개발된 하드웨어에 적용하기 위해서는 사용하지 않는 주변장치와 관련된 부분을 삭제하여야 하며 메모리 맵과 시스템 레지스터, 이더넷 드라이버를 수정해야 한다. 이는 각각의 시스템 마다 메모리 맵과 주변장치가 다르기 때문이다. 수정된 부트로더는 크로스 컴파일러를 이용하여 이미지 파일로 만든 후 JTAG 인터페이스를 통해 다운로드한 후 플래시에 저장된다.

RTAI를 리눅스에서 사용하기 위해서는 리눅스 커널에 RTAI 패치를 적용해야 한다. RTAI는 주로 타이머와 인터럽트 관련 소스가 수정되었다. 리눅스 커널은 하드웨어 계층 위에 존재하며 크게 하드웨어와 관련된 부분과 관련이 없는 부분으로 나누어지며, 일반적으로 리눅스 포팅이라하면 개발된 하드웨어와 관련된 부분을 수정하는 작업을 말하며, 포괄적으로는 부트로더 작성, 메모리 맵 설정, 시스템 레지스터 설정, 디바이스 드라이버 작성 등이 해당된다[11].

리눅스는 대부분의 실시간 운영체제와 달리 커널과 파일시스템이 분리되어 있기 때문에 파일 시스템을 구축해주어야 한다. 파일 시스템이란 운영체제가 데이터를 관리할 수 있도록 구성한 물리적 저장 공간을 말하며, 파일 시스템에는 쉘(shell)과 라이브러리, 응용 프로그램등이 포함된다. 그림 3은 임베디드 리눅스에서 사용되는 파일시스템의 종류와 구축과정을 나타낸다. 그림과 같이 원하는 응용 프로그램을 하

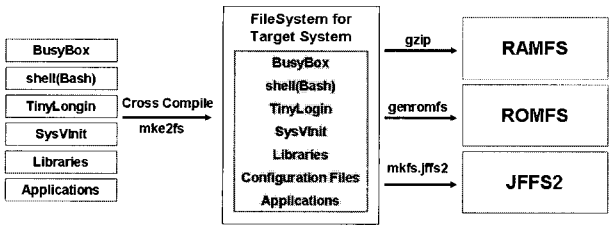


그림 3. 파일시스템의 구축과정.

Fig. 3. Implementation sequence of file system.

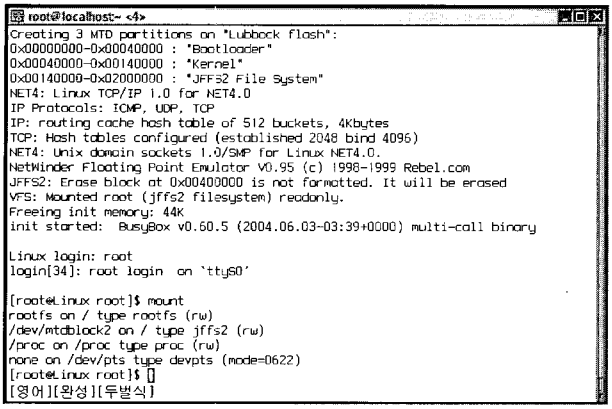


그림 4. 파일시스템과 디바이스 드라이버가 작성된 커널의 부팅 결과.

Fig. 4. The booting sequence of embedded linux kernel.

드웨어에서 실행가능한 형태로 컴파일한 후 원하는 파일시스템 포맷으로 변환 후 플래시에 저장하게 된다.

임베디드 리눅스에서는 RAMFS(RAM File System), ROMFS (ROM File System), JFFS2(Journaling Flash File System Version 2)가 주로 사용된다. 본 논문에서는 BusyBox, Tinylogin, Bash, 라이브러리 등을 포함시켜 JFFS2 파일시스템을 구축하였다. JFFS2는 RAMFS와 마찬가지로 읽기 및 쓰기가 가능할 뿐만 아니라, 전원이 없어도 데이터가 지워지지 않는 장점이 있다.

이상과 같은 단계별 개발 과정, 즉 부트로더 구현, 커널 포팅, 디바이스 드라이버 구현 그리고 파일시스템 구현을 통하여 개발된 하드웨어에서 동작하는 임베디드 리눅스가 완성되어 진다. 그림 4는 RTAI가 적용된 리눅스 커널의 부팅 결과를 나타낸다.

2. RTAI 설치 및 테스트

임베디드 리눅스 포팅을 완료한 후 실시간 임베디드 리눅스인 RTAI를 사용하기 위해서는 추가적인 작업이 필요하다. RTAI에서 제공되는 HAL과 실시간 메커니즘을 사용하기 위해서는 각각의 메커니즘에 대한 모듈 소스를 컴파일하여 생성된 오브젝트 파일을 커널에 추가해야 한다. 그러나 RTAI는 기본적으로 x86 호환 PC를 대상으로 컴파일 및 설치 옵션이 설정되어 있기 때문에 이 옵션을 수정해 주어야 한다.

실시간 태스크를 실행하기 위해서는 rtai_hal.o와 rtai_ksched.o 모듈이 기본적으로 커널에 추가되어야 한다. 또한 rtai_hal.o과 rtai_ksched.o는 HAL과 스케줄러를 위한 모듈이다. 그 밖에 모듈들은 사용하는 메커니즘에 따라 커널에 추가하면 된다.

그림 5는 RTAI로 구현된 시스템 보드에서 주기와 우선순

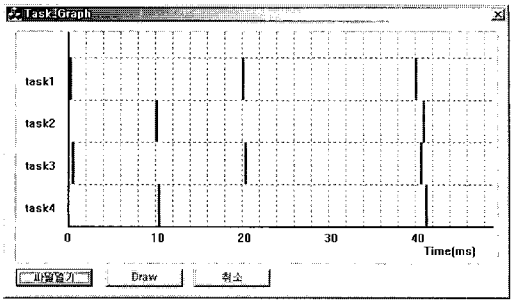


그림 5. 실시간 태스크의 실행 결과.

Fig. 5. The result of real-time tasks processing.

위, 실행순서가 서로 다른 4개의 태스크들을 실행한 결과이다. task1과 task2는 20ms의 주기로 동작하며, task3와 task4는 30ms의 주기로 동작한다. 실행순서는 task1->task2->task3->task4 순이다. 그리고 task1과 task3는 task2와 task4보다 우선순위가 높다. task1과 task2에서 task1의 우선순위가 높기 때문에 먼저 실행됨을 확인할 수 있으며, task3과 task4에서도 마찬가지로 task3가 우선적으로 실행되고 있다. task1과 task3는 태스크의 실행순서에 따라 task1이 먼저 실행되었으며, task2는 task3보다 먼저 실행되었지만 우선순위가 낮기 때문에 task3가 먼저 실행되었다.

IV. 실시간 시리얼 드라이버

1. 실시간 시리얼 드라이버의 구조

RTAI에는 시리얼 통신을 위한 RT_COM, 이더넷 네트워킹을 위한 RT_NET, CAN(Controller Area Network)을 위한 RT_CAN 등과 같은 실시간 디바이스 드라이버가 지원된다. 실시간 디바이스 드라이버는 리눅스 디바이스 드라이버와 달리 블록킹(blocking)이 생기는 기능이 배제되어 있으며, IRQ와 인터럽트 핸들러(interrupt handler)가 HAL에 등록된다. 예를 들어, RT_NET의 경우 데이터 전송 여부를 확인하는 TCP 통신을 지원하지 않고, 전송 여부를 확인하지 않는 UDP통신만을 지원한다[12-14].

본 논문에서는 실시간 디바이스 드라이버 중 실시간 시리얼 드라이버인 RT_COM을 적용하였다. RT_COM은 RTLinux와 RTAI를 위한 실시간 시리얼 드라이버로써 SourceForge.net에서 공개적으로 진행중인 프로젝트 중에 하나이다.

리눅스 시리얼 프로그램은 그림 6과 같은 구조로 되어 있기 때문에 다른 프로세스에 의해서 지연이 생길 수 있을 뿐만 아니라 다른 디바이스 드라이버의 ISR (Interrupt Service Routine) 또는 시리얼 드라이버 소스에서 사용된 블록킹 함수에 의해서 지연이 생길 수 있다.

반면 RT_COM과 실시간 태스크를 이용한 경우는 그림 7과 같은 구조로 되어 있기 때문에 태스크의 우선순위를 높게 함으로써 다른 태스크에 의한 지연을 없앨 수 있으며, RT_COM 소스 내에는 블록킹 함수가 포함되어 있지 않기 때문에 실시간성이 보장된다.

RT_COM은 모듈 구조로 되어 있기 때문에 필요시 커널에 추가되며, 이 때 IRQ와 ISR이 HAL에 등록되고, 인터럽트가 활성화된다. HAL에서는 해당 IRQ에 대한 인터럽트가 발생하면 등록된 ISR을 호출한다[15].

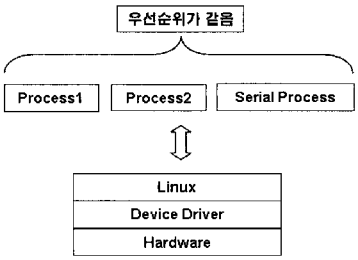


그림 6. 리눅스 시리얼 드라이버을 이용한 시리얼 통신 프로그램의 구조.
Fig. 6. The structure of serial program using linux serial driver.

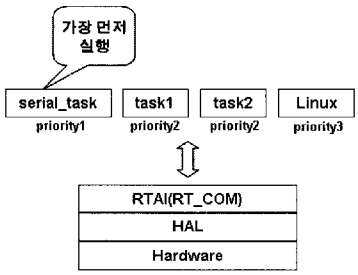


그림 7. RT_COM을 이용한 시리얼 통신 프로그램의 구조.
Fig. 7. The structure of serial program using RT_COM.

2. 실시간 시리얼 드라이버 작성

RT_COM은 PC16550A UART 컨트롤러가 내장된 x86 계열의 프로세서를 적용 대상으로 개발되었다. 따라서 PC16550A 호환 UART 컨트롤러가 내장된 프로세서에서는 모두 사용 가능하다. 본 논문에서 사용한 PXA255 역시 PC16550A 호환 UART 컨트롤러를 내장하고 있기 때문에 소스 코드의 수정 작업을 통하여 RT_COM을 사용할 수 있다.

RT_COM을 PXA255에서 사용하기 위해 수정해야 하는 부분은 UART 컨트롤러의 시작 주소와 프로세서에 할당된 IRQ 그리고 UART 컨트롤러의 내부 레지스터이다. 본 논문에서는 UART 컨트롤러의 시작 주소를 하드웨어 구성에 맞추어 0xf8200000으로 IRQ는 14번으로 수정하였다. 또한 UART 컨트롤러의 입력되는 클럭의 분주와 FIFO 트리거 레벨, 인터럽트 관련 레지스터를 수정하였다.

3. 실시간 시리얼 드라이버와 비실시간 시리얼 드라이버 비교

RT_COM을 이용하는 경우와 리눅스 시리얼 드라이버를 이용하는 경우를 비교하는 실험을 하기 위해 시리얼 통신으로 제어되는 모터 모듈의 속도 제어 실험을 하였다.

모터 모듈은 57600bps로 제어장치와 통신으로 하며, 엔코더는 한 바퀴에 4000펄스가 출력된다. 한 바퀴 회전을 했을 때의 이동거리는 약 308mm이며, 모터의 최대 속도는 500mm/s이다. 모터의 속도는 20ms 마다 제어되며, 실험방법을 다음과 같다.

1) 0 ~ 2초 동안 모터를 $0.2m/s^2$의 가속도로 속도를 증가시킨다.

2) 2 ~ 4초 동안 모터를 등가속도로 유지한다.

3) 4 ~ 6초 동안 모터를 $-0.2m/s^2$의 가속도로 속도를 감소시킨다.

그림 8은 RT_COM을 이용한 경우의 결과이고, 그림 9는

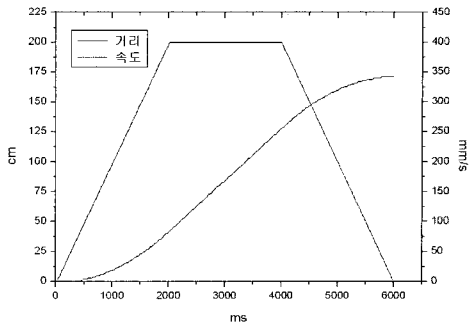


그림 8. RT_COM을 이용한 모터 제어.
Fig. 8. The control of motor using RT_COM.

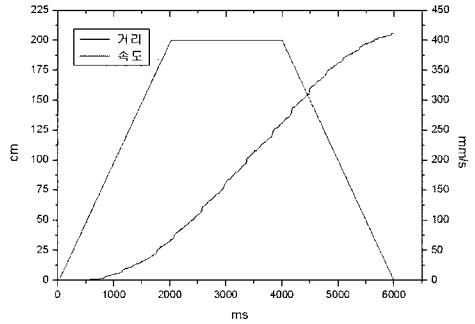


그림 9. 시리얼 드라이버를 이용한 모터제어.
Fig. 9. The control of motor using linux serial driver.

리눅스 시리얼 드라이버를 이용한 경우의 결과이다. 그림 8에서의 이동거리는 1.71m이고, 그림 9에서는 2.06m이다.

그림 8과 9의 결과에서 볼 수 있듯이 RT_COM을 이용한 경우는 속도가 일정하게 감/가속 되었지만 리눅스 시리얼 드라이버를 이용한 경우에는 속도가 불규칙하게 감/가속되었다. 결과적으로 제어 시간이 길어지면서 이동 거리가 RT_COM을 이용한 경우보다 0.35m를 더 이동하였다. 이러한 결과가 나타난 이유는 리눅스 시리얼 드라이버를 이용한 경우 다른 ISR이나 프로세스에 의해서 정확한 주기로 속도를 감/가속시키지 못 했기 때문이다.

V. 모바일 로봇 구현

모바일 로봇에서 위치 추정(localization)과 경로 계획(path planning)은 주행을 위해 꼭 필요한 기능들이다. 그러나 이러한 기능을 구현하기 위해서 Vision을 이용한 외각선 축출 또는 거리 센서(초음파센서, PSD, 레이저 스캔너 등)를 이용한 Markov, Montecarlo localization 알고리즘 등이 사용된다. 하지만 이러한 알고리즘들은 많은 연산을 필요로 하기 때문에 비실시간 운영체제를 이용할 경우 연산 중 장애물 등이 나타났을 때 빨리 반응을 할 수 없다. 그러나 실시간 운영체제를 이용한 경우에는 장애물 감지 기능을 다른 기능에 비해 높은 우선 순위로 할당함으로서 이러한 문제점을 극복할 수 있다. 물론 실시간 운영체제를 이용하지 않고 DSP (Digital Signal Porcessor)와 같은 MPU을 이용하여 이러한 문제를 해결할 수 있지만, 이러한 프로세서들은 메모리가 한정되어 있을 뿐만 아니라 최적의 성능을 발휘하기 위해 각 프로세서 만의 고유 명령을 이용해야 하며, 디버깅이 어려운 단점이 있다.

본 논문에서는 앞서 구현한 실시간 임베디드 리눅스 기반의 제어시스템 플랫폼의 특징을 이용하여 모바일 로봇(mobile robot)에 적용하여 실험하였으며, DMA를 이용하지 않고 메모리에 저장된 영상 데이터를 전송하는 코드를 많은 연산을 필요로 하는 코드 대신에 추가하여 실험하였다.

모바일 로봇에는 2개의 센서 모듈과 모터 모듈, 카메라 모듈 그리고 이것들을 통합관리하기 위한 시스템보드가 포함되어 있다. 시스템보드는 앞서 구현된 실시간 임베디드 리눅스 기반의 제어시스템 플랫폼이다.

센서 모듈과 모터 모듈은 시리얼(RS-232C) 통신을 통하여 메인 시스템보드와 데이터를 주고 받는다. 센서 모듈과의 통신을 위해 USB-to-Serial 컨버터를 이용하여 연결하였는데 이는 메인 시스템보드에서 UART를 2개 밖에 지원하지 않기 때문이다. 카메라 모듈은 CIF 해상도의 CMOS 카메라로 EPLD를 통해서 저장된 영상 데이터를 받는다. 모바일 로봇과 클라이언트(PC)는 무선랜을 통하여 TCP/IP로 데이터를 주고받는다.

모바일 로봇의 소프트웨어는 RT_TASK와 서버, 클라이언트로 구성된다. RT_TASK는 커널 모드에서 실행되는 실시간 프로그램으로 서보모터를 제어하기 위해 사용된다. 서버는 사용자 모드에서 실행되는 리눅스 프로세스로써 RT_TASK와 RTF_FIFO를 통하여 데이터를 주고받고, 클라이언트와는 TCP/IP를 통하여 데이터를 주고 받는다. 그리고 센서 모듈과 카메라 모듈을 제어하여 여기서 받은 데이터를 버퍼에 저장한다. 클라이언트는 TCP/IP 네트워크 상에 연결된 PC에서 실행되는 프로그램으로써 모바일 로봇을 제어하고 모니터링한다.

그림 10은 모바일 로봇에 구현된 소프트웨어 구조를 나타낸다. 서버 프로그램은 4개의 쓰레드(Thread)로 구성되어 있으며, 각각의 쓰레드는 메시지 큐(Message Queue)를 통해서 데이터를 전달한다. recv와 send 쓰레드는 TCP/IP를 통해서 클라이언트와 통신하는 쓰레드이다. recv 쓰레드는 클라이언트로부터 데이터를 받을 때까지 기다리고 있다가 데이터를 받으면 그것을 해석하여 해당 쓰레드로 데이터를 전달하는 기능이 구현되어 있으며, send 쓰레드는 servo와 sensor 쓰레드로 부터 메시지를 받으면 클라이언트로 데이터를 전달하는 쓰레드이다. servo 쓰레드는 RT_TASK와의 데이터 전달을 위한 쓰레드로 RTF_FIFO를 통해서 데이터를 전달하는 쓰레드이다. sensor 쓰레드는 시리얼 통신을 통해서 센서 모듈(초음파센서, PSD센서)와 카메라 모듈을 제어하는 쓰레드이다.

그림 11은 RT_TASK의 구조를 나타낸다. RT_TASK는 rtf_handler로부터 메시지를 받아서 모터를 제어하는 servo_ctrl 태스크와 20ms 주기로 모터의 상태를 체크하는 servo_status 태스크로 구성되어 있다. 세마포어(Semaphore)는 servo_ctrol 태스크와 servo_status 태스크가 동시에 RT_COM을 사용하는 것을 방지하기 위해 사용되었다.

그림 12는 RT_TASK의 실행 결과를 나타낸다. status는 servo_status 태스크의 실행결과이며, ctrl은 servo_ctrl 태스크의 실행결과이다. status는 20ms의 주기로 동작하고 있으며, ctrl는 RTF_FIFO를 통해 데이터를 받았을 때 실행되고 있다. 그림 12의 아래 그림을 보면 servo_ctrl 태스크가 servo_status 태스크에 의해서 멈춤을 확인할 수 있다. 이것은 servo_status 태스크가 servo_ctrl 태스크보다 우선순위가 높기 때문이다. 그리고 servo_status 태스크가 블록킹이 되고 servto_ctrl 태스크가 다시 실행되었는데, 이것은 servo_status 태스크가 세마포어에 의해서 블록킹 되었기 때문이다. servo_ctrl 태스크의 실행이 끝나면 servo_status태스크가 다시 실행됨을 확인할 수 있다.

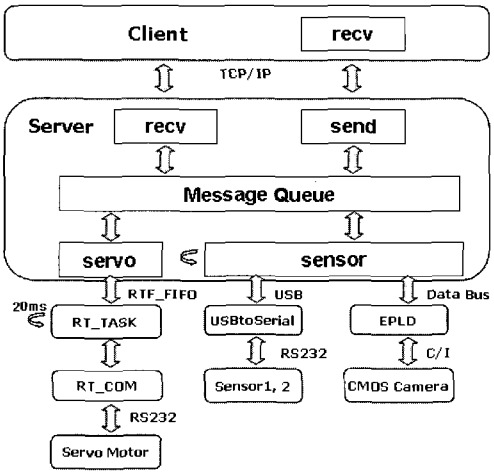


그림 10. 모바일 로봇 S/W의 구조.
Fig. 10. The structure of mobile robot S/W.

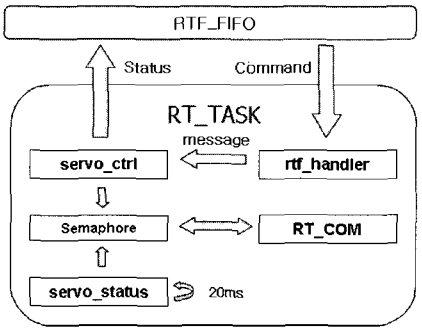


그림 11. RT_TASK의 구조.
Fig. 11. The structure of RT_TASK.

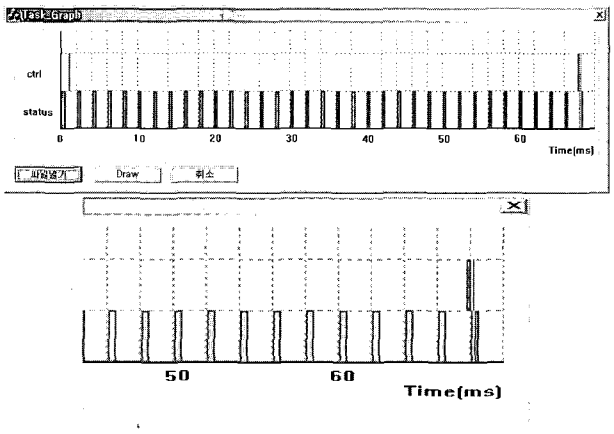


그림 12. RT_TASK의 실행 결과.
Fig. 12. The result of RT_TASK processing.

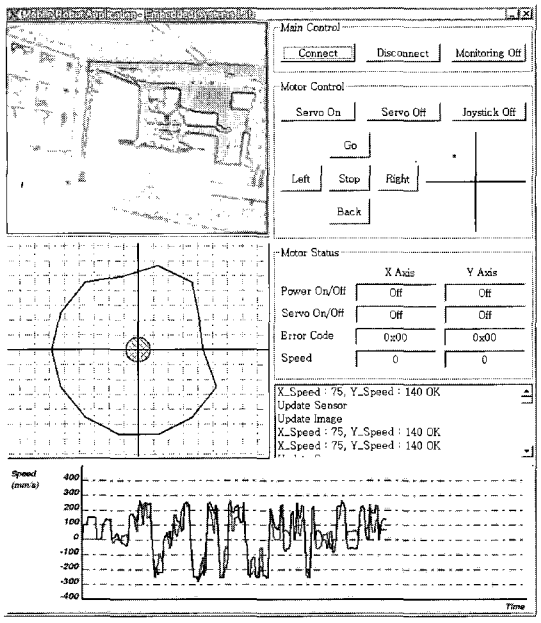


그림 13. 클라이언트 프로그램.
Fig. 13. The client program.

클라이언트 프로그램은 서버로부터 받은 영상 데이터와 센서 데이터 값, 모터의 속도를 화면에 출력하고 모터제어명령을 조이스틱과 마우스를 통해 사용자로부터 입력 받아 서버로 전달하는 프로그램이다. 본 논문에서는 리눅스가 설치된 PC에서 QT를 이용하여 클라이언트 프로그램을 구현하였다. 클라이언트 프로그램은 서버에 접속하면 쓰레드를 생성한다. 이 쓰레드는 서버로부터 데이터를 받아 화면에 출력한다.

QT에서는 소켓 통신을 위한 QSocket 클래스와 쓰레드를 위한 QThread 클래스를 지원한다. QSocket과 QThread를 이용할 경우 리눅스에서 제공되는 소켓과 쓰레드 라이브러리를 사용할 때 보다 데이터 전송 시간이 3배 정도 더 걸리기 때문에 사용하지 않고 리눅스에서 제공하는 라이브러리를 사용하였다. 그림 13은 구현된 클라이언트 프로그램의 실행 화면이다.

VI. 결론

임베디드 시스템이 소형화, 저전력화, 고성능화되면서 다양한 주변 장치를 하나의 칩으로 만든 SoC를 이용한 시스템이 주류를 이루고 있다. SoC는 다양한 개발의 이점을 제공하고만 복잡한 기능으로 인해 운영체제 없이 개발하기에는 한계가 있다. 때문에 임베디드 시스템에 임베디드 운영체제와 실시간 운영체제가 적용되고 있다. 그러나 임베디드 운영체제는 실시간성을 제공하기 못 하며, 실시간 운영체제는 대부분이 상용이기 때문에 기술이 종속될 수 있으며, 초기 개발 비용이 증가하는 단점이 있다.

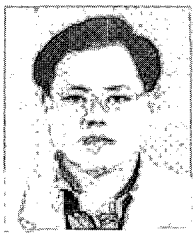
본 논문에서는 상용 실시간 운영체제가 아닌 오픈 소스이며 무료로 제공되는 RTAI를 임베디드 시스템에서 가장 많이 사용되는 SoC 중 하나인 PXA255 기반의 시스템보드에 적용하였다. RTAI는 리눅스를 기본 운영체제로 실시간 태스크를 위한 인터페이스를 제공하는 방법을 채택함으로써 커널의 수정을 최소화하였기 때문에 임베디드 리눅스가 적용된 기존의 시스템에 간단한 수정 작업만으로 실시간성을 제공할 수 있는 장점을 제공할 뿐만 아니라 평균 Latency 시간이 3~4 us정도로 상용 운영체제(1 ~ 20us) 보다 결코 떨어지지 않으면서 개발비용을 절감할 수 있는 이점을 제공한다. 그리고 실험을 통해 리눅스 시리얼 드라이버와 실시간 시리얼 드라이버를 비교함으로써 실시간 시스템의 장점을 확인하였다. 결과적으로 본 논문에서 구현한 시스템은 저렴한 비용으로 실시간 시스템을 구현할 수 있는 하나의 방안이 될 수 있을 것이다.

참고문헌

[1] Electronic Tread Publications, System on chip Market and Trends, April, 2003.
[2] 조덕연, 최병욱, "임베디드 리눅스를 이용한 산업용 인버터의 웹기반 원격관리," 제어 및 자동화 시스템 공학회, 제9권 4호, 340-346, 2003.
[3] 최병욱, 신은철, 이수영, "임베디드 리눅스를 이용한 웹기반 빌딩자동화시스템," 제어 및 자동화 시스템 공학회, 제 10권 4호, 335-341, 2004.
[4] 최병욱, 김현기, 신은철, "임베디드 리눅스 기반의 다중 프로토콜 제어기 개발 및 빌딩자동화시스템과의 연동 적용," 제어 및 자동화 시스템 공학회, 제 10권 5호, 428-433, 2004.
[5] 최병욱, "Embedded operating system and it's applications in development for networked robot," 한국로봇공학회, 제1회 워크샵, 2004.
[6] ㈜휴인스 기술연구소, "Intel PXA255와 임베디드 리눅스 응용," 2004, 홍능과학출판사.
[7] Ismael Ripoll, "RTLinux versus RTAI", www.linuxdevices.com, 2002.
[8] Kevin Dankwardt, "Comparing real-time linux alternatives," www.linuxdevices.com, 2000.
[9] L. Dozio, P. Mantegazza, "Linux real time application interface(RTAI) in low cost high performance motion control," *Motion Control 2003*, a conference of ANIPLA, Associazione Nazionale Italiana per I'Automazione, Milano, Italy, 27-28, March, 2003.
[10] Herman Bruyninckx, "Real time and embedded guide," December, 2002.
[11] Alessandro저/김인성,류태중 역, "리눅스 디바이스 드라이버," 2000년 2월, 한빛미디어.
[12] Hard Real-Time Networking for Linux/RTAI, www.rts.uni-hannover.de/rtnet
[13] Real-Time CAN on Linux, sourceforge.net/project/rtcan
[14] Real-Time Linux drvier for the serial port, rt-com.sourceforge.net
[15] P.N. Daly and Kupper, "The serial port driver of real-time linux," *Real Time Documentation Project*, vol 1, 2000.

신 은 철

1977년 6월 15일생. 2003년 선문대학교 전자공학과, 제어계측공학과 졸업(공학사). 2005년 동 대학교 대학원 제어계측과 석사과정 졸업(공학석사). 2005년~현재 한국생산기술연구원 연구원. 관심분야는 임베디드 시스템, 임베디드 리눅스, 실시간 운영체제.



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. 관심분야는 소트프웨어 엔지니어링, 내장형 시스템, 실시간 운영체제, 임베디드 리눅스, 필드버스 및 분산 제어 시스템, 지능제어로봇.