공학석사 학위논문

임베디드 시스템 기반의 보행로봇 제어기 개발

Development of a Biped Walking Robot Controller Based on a Embedded System

지도교수 유 삼 상

2006년 2월

한국해양대학교 대학원

기 계 공 학 과

손 정 호

本 論文을 孫廷虎의 工學碩士 學位論文으로 認准함.

委員員 工學博士 鄭 在 鉉 (印)

委 長 工學博士 柳 三 相 (印)

委 員 工學博士 崔 炯 植 (印)

2006 年 2 月

韓國海洋大學校 大學院

機 械 工 學 科

孫 廷 虎

목 차

Development of a Biped Walking Robot Controller Based on a Embedded System

Son, Joung Ho

Department of Mechanical Engineering Graduate School, Korea Maritime University

Abstract

In this paper, we have made a new motion controller and the embedded ARM board to control the biped walking robot. As a control system for the robot, a distributed control system composed of the main controllers and the motor controller for the robot using the one-chip microprocessor were developed. The main controller and the motor controller composed as a distributed control system were developed using the embedded ARM board and the TMS320c2407 processor, respectively. A new trajectory tracking algorithm for the motor controller appropriate for the one-chip microprocessor was devised employing pre-generated off-line trajectory data. Also, the robot composed of an operation system of the ARM board based on the real-time LINUX was developed. In addition to these, a CAN communication system was developed to process the complex control system. Some tests and their results were presented to validate superior performance of the developed system.

기 호 설 명

A_1	각 관절의 동차 변환 행렬
θ_i	각 관절의 각도
d_i	각 관절의 오프셋
a_i	각 관절의 링크 길이
α_i	각 관절의 비틀림
$R_{z,\theta}$	z 축을 기준으로 θ 만큼 회전한 행렬
$T_{z,d}$	z 축을 따라 d 만큼 회전한 행렬
$T_{x,a}$	x 축을 따라 a 만큼 이동한 행렬
$R_{x,\alpha}$	x 축을 기준으로 α 만큼 회전한 행렬
A_i	각 관절의 동차 변환 행렬
l_i	각 관절의 링크 길이
$\vec{p}$	말단 장치의 위치 벡터
ϕ	최종링크가 기저면과 이루는 각

그 림 목 차

표 목 차

1. 서 론

1.1 연구배경

최근 위험한 환경에서 인간을 대신하여 작업하는 로봇이나, 노인이나 장애자 수족 역할을 하는 가정용 로봇, 박물관이나 우체국과 같은 공공기관에서 사람들을 돕는 로봇에 이족보행 로봇의 활용이 예상되고 있다. 그런데, 이족보행 로봇의 제어시스템은 매우 복잡하여 고성능의 프로세서가 요구되지만 부피와 중량이 무거운 기존의 PC형태의 제어시스템을 적용하는 것은 로봇의 경량화에 걸림돌이 된다. 특히, 자율보행을 위해서는 모든 배터리, 제어시스템 등을 자체에 탑재하여 이동하므로 제어시스템은 소형이 되어야 하는 반면 고성능 프로세서를 적용하여야 하는 매우 어려운 문제를 안고 있다. 따라서, 여러 형태의 중소형 프로세서들 중에서 가장 적합한 것을 선택하여 효율적인 체계로 제어시스템을 구성해야 한다.

현재 개발된 이족보행로봇들인 아시모[1], HRP II[2] KHR 등은 자체적으로 제어시스템을 개발하여 적용되고 있으나 개발내용에 대한 구체적인 자료의 공개가 되어 있지 않다. 따라서 이동로봇용 로봇의 개발에는 각각 새로운 프로세서를 사용하여 독자적인 제어시스템을 구성해야 하는 과제를 안고 있다. 즉, 로봇의 설계 및 중소형 One-chip 마이크로프로세서를 사용하여 새로운 각 관절축 모터의 위치 및 속도를 제어하는 축 제어시스템을 포함하는 로봇 제어시스템의 연구와 더불어 원활한 작업을 위한 총체적인 경로제어 알고리즘 개발도 필수적이다. 따라서 모터를 제어하고 자율보행을 위해서는 모터와 모터 드라이버 모션보드가 하나의 모듈화가 필요하고 모션보드는 모터의 원활한 경로제어를 위해 빠른 연산이 가능한 시스템이 필요하다. 또한 각각의 모션보드 제어와 로봇의 지능을 부여하기 위해서는 로봇에 탑제되어 적은 전력 으로도 구동이 되고 진동과 노이즈에 강인함을 지님과 동시에 하드웨어와 소프트웨어의 최적화와 특정한 작업에 있어서는 소형의 마이크로 프로세서 보다는 강력한 컴퓨팅 파워를 낼수있는 제어기가 필요로 하게 된다. 이에 더하여 새로운 구조의 로봇을 실제 작업과 제어를 위해 기구학 모델을 세우는 것도 매우 중요한 과제이다.

1.2 연구 목적 및 내용

본 연구에서는 제어시스템에서 이족보행 로봇에 적합한 제어 시스템을 설계 제작하고 로봇의 실제 작업과 제어를 위해 다리부의 기구학 모델을 구하였다. 제어시스템으로는 최근 각광을 받는 저용량, 저가인 반면 고성능DSP Chip인 TMS2407 마이크로프로세서[3,4]를 이용하여 로봇의 관절 제어 시스템을 모듈화된 분산 제어시스템으로 구성하고, 이들을 총괄적으로 제어하기 위한 주제어기로 ARM CPU[6]가 탑재된 Embeded 보드를 이용하여 로봇에 내장되는 Embeded 형태의 제어기를 구성하였다. 제어시스템의 주 프로세서가 One-chip 마이크로프로세서이므로 로봇의 경로 제어를 위해 이를 이용한 각 관절축 모터의 위치 및 속도 제어 알고리즘을 개발하였다. 또한 전용 관절축 제어기와 주제어기간의 프로토콜을 포함하는 CAN통신시스템을 구축하였다.

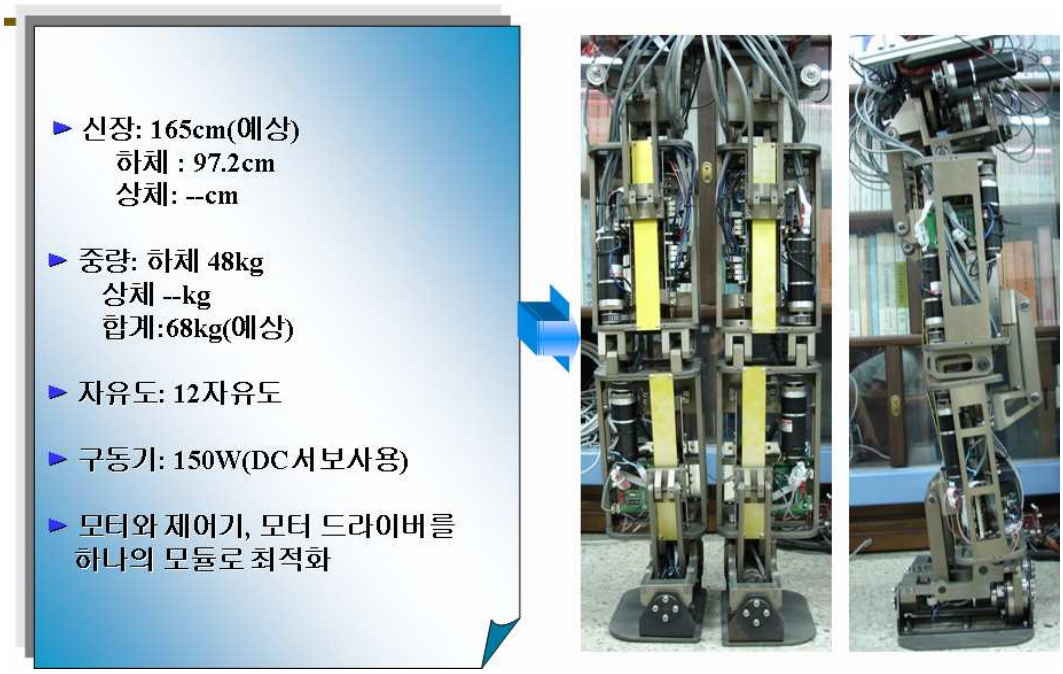


Fig 1.1 12 d.o.f rovolute robot

2. 이족보행로봇의 구조 및 해석

2.1 로봇의 기구구조

인간의 보행과 흡사한 형태를 하기 위해서는 하체에 최소한 12개의 자유도를 가져야 한다고 많은 연구결과들이 나와있다. 그것은 허리에 Yaw, Roll, Pitch 3축을 가지고 무릎의 Pitch축, 발목의 Roll과 Pitch축의 형태인데 이는 현재 이족보행로봇을 개발하기 위한 기본적인 조건으로 제시되고 있다. 본 연구를 수행하기 전 실험되었던 로봇은 하체에 8자유도를 가진 형태로써 허리 부분의 Yaw축과 Roll축이 없어서 양쪽 다리 총 4자유도가 부족한 형태였다.

본 연구를 위해 개발된 이족 보행로봇은 전체 25개의 모터로 구동하는 시스템으로써 하체에만 12개의 모터가 장착되어있다. 앞에서도 언급한 바와 같이 허리부분의 Yaw축과 Roll축은 RV감속기와 타이밍 벨트(timing belt), 타이밍 풀리(timing pulley)로 연결되어 있으며 발목부분의 roll축은 RV감속기와 모터가 직결되어 구동된다. 반면 허리의 Pitch축과 무릎의 Pitch축, 발목의 Pitch축은 4절링크 구조에 볼스크루를 응용한 형태로 강성과 토크 전달력을 크게 하고 제어정밀도를 높이는 구조로 되어있다.

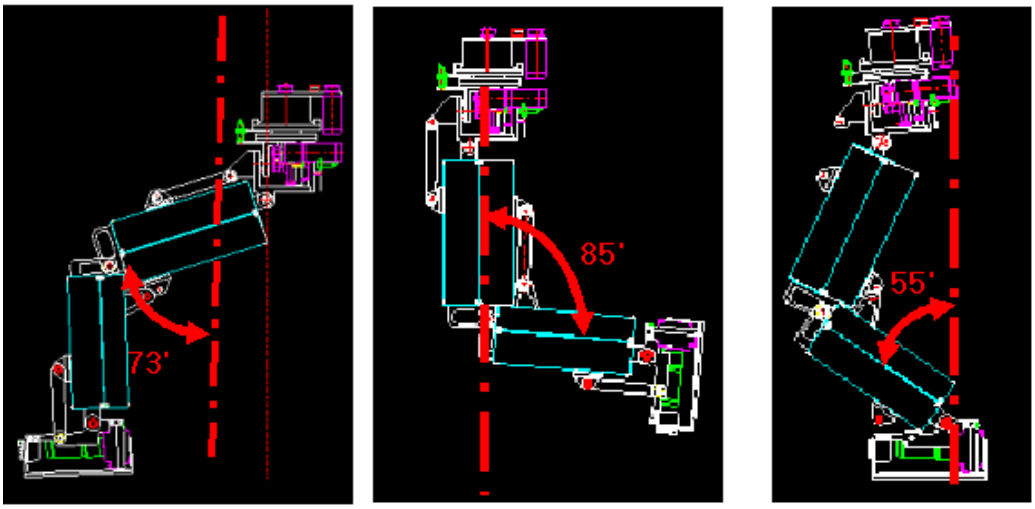


Fig 2.1 Motion of robot

2.1.1 구동 액츄에이터

본 논문에서는 사절 기구의 한 변을 볼나사의 직선운동으로 변환하여 볼나사의 직선운동이 이족 로봇의 회전관절에 큰 모멘트를 발생시켜서 회전운동으로 변환하는 구조의 구동 액츄에이터를 이용 하였다. 4절 링크 중에서 볼나사로 구성되는 링크가 로봇의 다리 링크가 되고, 여기에 가이드가 부착되거나 가이드를 링크로 사용하여 관절 구동 액츄에이터의 중량을 최소화하는 구조로 구성되어 있다. 볼나사는 강성과 정도가 매우 높으므로 구동용 DC서보 모터를 볼나사에 직결하고 원하는 관절각의 회전에 따른 볼나사의 변위를 이동시키는 구조로 구성되어 있다. 볼나사 변위와 각 관절 간의 관계식은 Closed-form으로 표현되므로 제어기에서 관계식을 적용하여 관절각을 용이하게 제어하였다. 볼나사와 다양한 용량의 구동모터 간의 동력전달은 타임벨트를 이용하여 Parallel하게 구성되어 액츄에이터의 부피가 적은 형태로 되어있다.

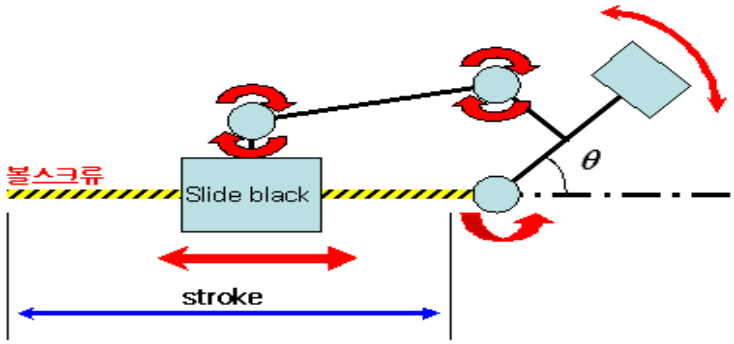


Fig 2.2 Structure of four bar link

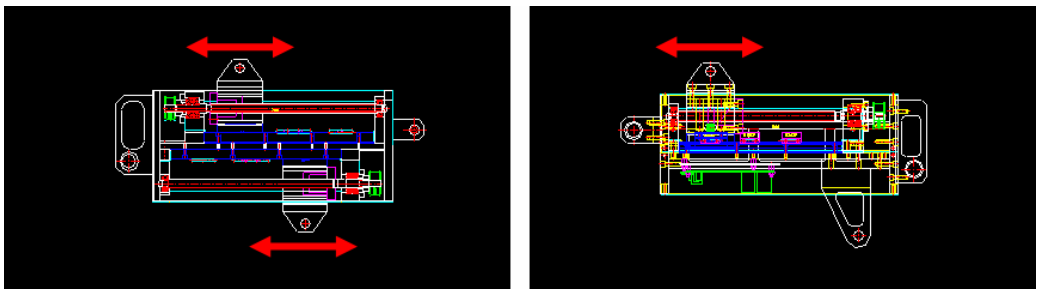


Fig 2.3 Four bar link application

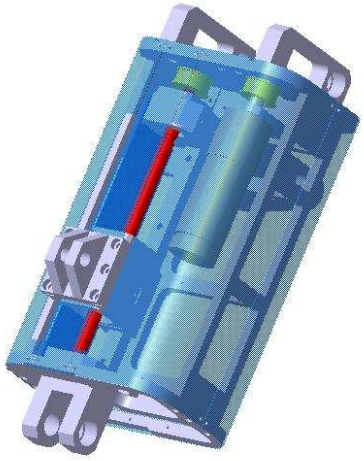


Fig 2.4 Four bar link 3D model

2.2 로봇의 관절구조

본 연구에 쓰인 이족보행로봇 하체의 관절구조는 허리 부분의 Yaw축과 Roll축 그리고 발목 부분의 Roll축은 감속기와 직결되거나 타이밍 풀리, 벨트 시스템으로 구성되어 있으나 허리, 무릎, 발목의 Pitch축은 토크의 전달력과 강성을 위하여 자체 제작된 4절링크 구조로 되어있다. 허리, 무릎, 발목의 pitch축의 각 관절각들은 각각의 4절링크 중 직선변위를 가지는 하나의 변과 상관관계를 가지고 있는데 이 논문에서 제시하는 관계식으로써 변환될 수 있다. 이 절에서는 허리, 무릎, 발목 관절에 해당하는 4절링크를 정의하고 각 관절각과 4절링크 변위와의 관계식을 구하고 속도 및 가속도의 관계식도 함께 구한다. 4절링크에서 3개의 변의 길이는 고정이고 나머지 한 변은 볼스크루의 변위로 만들어 볼 스크루의 지지부와 다리가 이루는 각을 고정각으로 하면 볼 스크루변위와 관절각의 관계를 설명할 수 있다.

2.2.1 허리관절의 구조

허리관절의 볼나사 시스템에 존재하는 3개의 링크를 기준으로 사절링크를 만든다. 이때 고정부는 실제 볼나사와 사절링크 변위가 평행하도록하고 고정각이 90도가 되도록 위치를 정한다. 허리 관절의 사절링크 모델링은 Fig 2.5 과 같다.

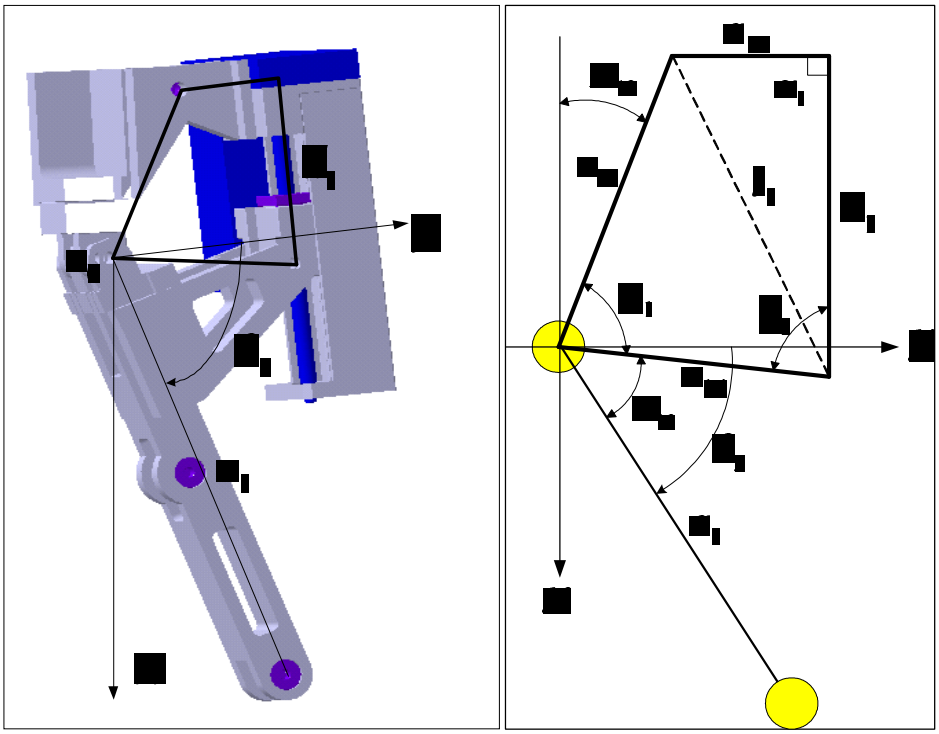


Fig 2.5 The pelvise joint

볼나사의 변위 d_1과 허리 관절의 관절각 θ_3의 관계를 유도한다. Fig 2.5 의 사절링크에서 l_1를 이용하여 cos 법칙을 적용하면 다음과 같다.

$$l_1 = a_{12}^{\ 2} + d_1^{\ 2} - 2a_{12}d_1\cos\alpha_1 = a_{13}^{\ 2} + a_{14}^{\ 2} - 2a_{13}a_{14}\cos\beta_1$$

$$d_1 = \frac{C_1 + [C_1^{\,2} + 4(A_1 + B_1 \cos\beta_1)]^{\frac{1}{2}}}{2} \tag{2.2.1}$$

식 (2.2.1) 에서

$$A_1 = a_{13}^{\,2} + a_{14}^{\,2} - a_{12}^{\,2}$$
$$B_1 = -2a_{13}a_{14}$$
$$C_1 = 2a_{12}\cos\alpha_1$$

여기서 링크 길이 a_{12}, a_{13}, a_{14} 와 링크 각 α_1 는 고정된 값이다. 관절각 θ_3 와 사절링크에서 변위 d_1 는 마주보는 각 β_1 의 관계는 다음식으로 표현된다.

$$\beta_1 + N_{12} + N_{13} + (\frac{\pi}{2} - \theta_3) = \pi$$
$$\beta_1 = \frac{\pi}{2} - N_{12} - N_{13} + \theta_3 \tag{2.2.2}$$

여기서 N_{12}, N_{13} 는 고정각이다. 식 (2.2.1) 의 미끄럼 변위를 시간에 관해 미분하면 다음과 같이 속도 및 가속도를 구할 수 있다.

$$\dot{d}_1 = -[C_1^{\,2} + 4(A_1 + B_1\cos\beta_1)]^{-\frac{1}{2}} B_1 \sin\beta_1\dot{\beta}_1 \tag{2.2.3}$$

$$\ddot{d}_1 = -2[C_1^{\,2} + 4(A_1 + B_1\cos\beta_1)]^{-\frac{3}{2}} B_1^{\,2}\sin^2\beta_1\dot{\beta}_1^{\,2}$$
$$-[C_1^{\,2} + 4(A_1 + B_1\cos\beta_1)]^{-\frac{1}{2}}(B_1\cos\beta_1\dot{\beta}_1^{\,2} + B_1\sin\beta_1\ddot{\beta}_1) \tag{2.2.4}$$

관절각 θ_3 와 d_1 의 속도 및 가속도 관계식을 구하면 다음과 같다.

$$\beta_1 = \arccos\left[\frac{d_1^{\,2} - A_1 - C_1 d_1}{B_1}\right] \tag{2.2.5}$$

$$\dot{\beta}_1 = R_{11}\dot{d}_1 \tag{2.2.6}$$

$$\ddot{\beta}_1 = R_{12}\dot{d}_1^{\,2} + R_{13}\ddot{d}_1 \tag{2.2.7}$$

식 (2.2.6) 와 (2.2.7) 에서

$$R_{11} = \frac{[C_1^{\,2} + 4(A_1 + B_1\cos\beta_1)]^{\frac{1}{2}}}{B_1\sin\beta_1}$$

$$R_{12} = -2[C_1^{\,2} + 4(A_1 + B_1\cos\beta_1)]^{-1} B_1\sin\beta_1 R_{11}^{\,2} + \frac{\cos\beta_1}{\sin\beta_1} R_{11}^{\,2}$$

$$R_{13} = -\frac{[C_1 + 4(A_1 + B_1\cos\beta_1)]^{\frac{1}{2}}}{B_1\sin\beta_1}$$

2.2.2 무릎관절의 구조

무릎관절에 적용된 사절링크의 구조는 허리관절과 동일한 형태를 가진다.

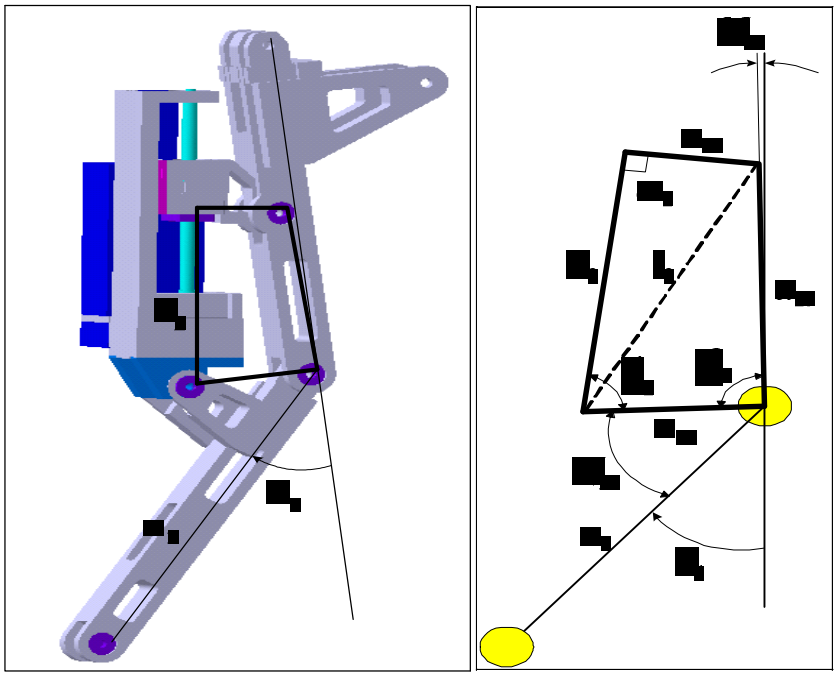


Fig 2.6 The knee joint

볼나사의 변위 d_2 과 허리 관절의 관절각 θ_4 의 관계를 유도한다. Fig 2.6 의 사절 링크에서 l_2 를 이용하여 cos 법칙을 적용하면 다음과 같다.

$$l_2 = {a_{22}}^2 + {d_2}^2 - 2a_{22}d_2\cos\alpha_2 = {a_{23}}^2 + {a_{24}}^2 - 2a_{23}a_{24}\cos\beta_2$$

$$d_2 = \frac{C_2 + [{C_2}^2 + 4(A_2 + B_2\cos\beta_2)]^{\frac{1}{2}}}{2} \qquad (2.2.8)$$

식 (2.2.8) 에서

$$A_2 = {a_{23}}^2 + {a_{24}}^2 - {a_{22}}^2$$
$$B_2 = -2a_{23}a_{24}$$
$$C_2 = 2a_{22}\cos\alpha_2$$

여기서 링크 길이 a_{22}, a_{23}, a_{24} 와 링크 각 α_2 는 고정된 값이다. 관절각 θ_4 와 사절링크에서 변위 d_2 와 마주보는 각 β_2 의 관계는 다음식으로 표현된다.

$$\beta_2 + N_{22} + N_{23} + \theta_4 = \pi$$
$$\beta_2 = \pi - N_{22} - N_{23} - \theta_4 \qquad (2.2.9)$$

여기서 N_{22}, N_{23} 는 고정각이다. 식 (2.2.8) 의 미끄럼 변위를 시간에 관해 미분하면 다음과 같이 속도 및 가속도를 구할 수 있다.

$$\dot{d}_2 = -[C_2^{\,2} + 4(A_2 + B_2 \cos\beta_2)]^{-\frac{1}{2}} B_2 \sin\beta_2 \dot{\beta}_2 \qquad (2.2.10)$$

$$\ddot{d}_2 = -2[C_2^{\,2} + 4(A_2 + B_2 \cos\beta_2)]^{-\frac{3}{2}} B_2^{\,2} \sin^2\beta_2 \dot{\beta}_2^{\,2}$$
$$-[C_2^{\,2} + 4(A_2 + B_2 \cos\beta_2)]^{-\frac{1}{2}} (B_2 \cos\beta_2 \dot{\beta}_2^{\,2} + B_2 \sin\beta_2 \ddot{\beta}_2) \qquad (2.2.11)$$

관절각 θ_4 와 d_2 의 속도 및 가속도 관계식을 구하면 다음과 같다.

$$\beta_2 = \arccos\left[\frac{d_2^{\,2} - A_2 - C_2 d_2}{B_2}\right] \qquad (2.2.12)$$

$$\dot{\beta}_2 = R_{21}\dot{d}_2 \qquad (2.2.13)$$

$$\ddot{\beta}_2 = R_{22}\dot{d}_2^{\,2} + R_{23}\ddot{d}_2 \qquad (2.2.14)$$

식 (2.2.13) 와 (2.2.14) 에서

$$R_{21} = \frac{[C_2^{\,2} + 4(A_2 + B_2 \cos\beta_2)]^{\frac{1}{2}}}{B_2 \sin\beta_2}$$

$$R_{22} = -2[C_2^{\,2} + 4(A_2 + B_2 \cos\beta_2)]^{-1} B_2 \sin\beta_2 R_{21}^{\,2} + \frac{\cos\beta_2}{\sin\beta_2} R_{21}^{\,2}$$

$$R_{23} = -\frac{[C_2 + 4(A_2 + B_2 \cos\beta_2)]^{\frac{1}{2}}}{B_2 \sin\beta_2}$$

2.2.3 발목관절의 구조

발목관절의 사절링크는 앞서 설명한 허리와 무릎관절의 구조와 조금 다른 형태이다. 그림 2.7을 보면 변위 d_3 위치가 앞의 사절링크들과 다르다. 그러므로 관절각 θ_5 와 사절링크에서 변위와 마주보는 각 β_3 의 관계에 대한 해석은 조금 다른 형태를 취한다.

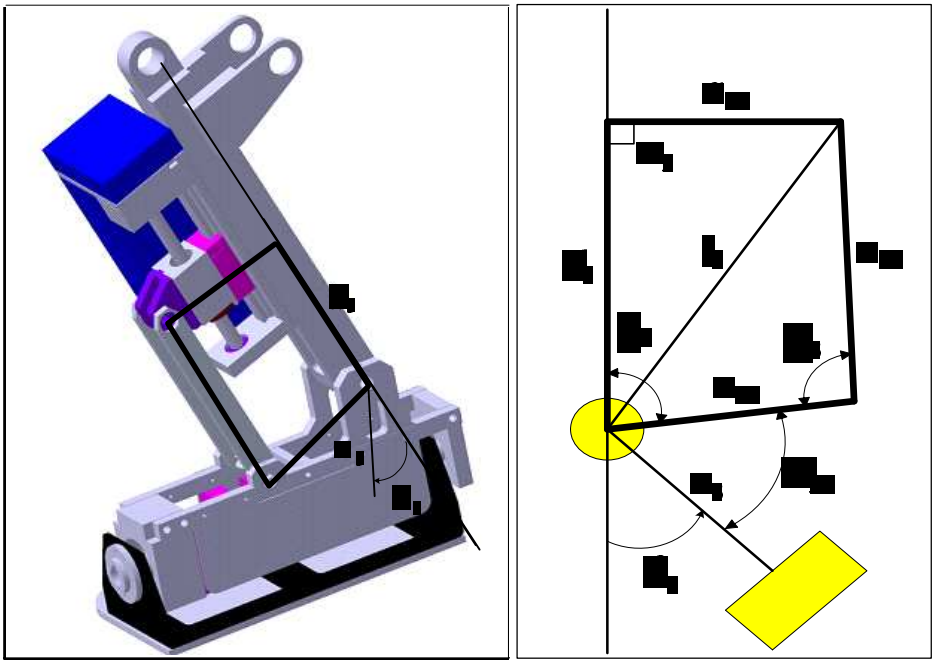


Fig 2.7 The ankle joint (1)

볼나사의 변위 d_3과 허리 관절의 관절각 θ_5의 관계를 유도한다. Fig 2.7 의 사절링크에서 l_3를 이용하여 cos 법칙을 적용하면 다음과 같다.

$$l_3 = {a_{32}}^2 + {d_3}^2 - 2a_{32}d_3\cos\alpha_3 = {a_{33}}^2 + {a_{34}}^2 - 2a_{33}a_{34}\cos\beta_3$$

$$d_3 = \frac{C_3 + [{C_3}^2 + 4(A_3 + B_3\cos\beta_3)]^{\frac{1}{2}}}{2} \qquad (2.2.15)$$

식 (2.2.15) 에서

$$A_3 = {a_{33}}^2 + {a_{34}}^2 - {a_{32}}^2$$
$$B_3 = -2a_{33}a_{34}$$
$$C_3 = 2a_{32}\cos\alpha_3$$

여기서 링크 길이 a_{32}, a_{33}, a_{34}와 링크 각 α_3는 고정된 값이다.

관절각 θ_5와 사절링크에서 변위 d_3와 마주보는 각 β_3의 관계식을 구한다.

Fig 2.8 에서 σ_3는 o_2을 지나고 x축에 수직한 직선이 고정변위 a_{34}와 이루는 각이며 방향성을 지닌다. σ_3와의 관계식을 구하면 다음과 같다.

$$N_{32} - \theta_5 + \sigma_3 = \frac{\pi}{2}$$
$$\sigma_3 = \frac{\pi}{2} - N_{32} + \theta_5 \qquad (2.2.16)$$

여기서 N_{32}는 고정각이다.

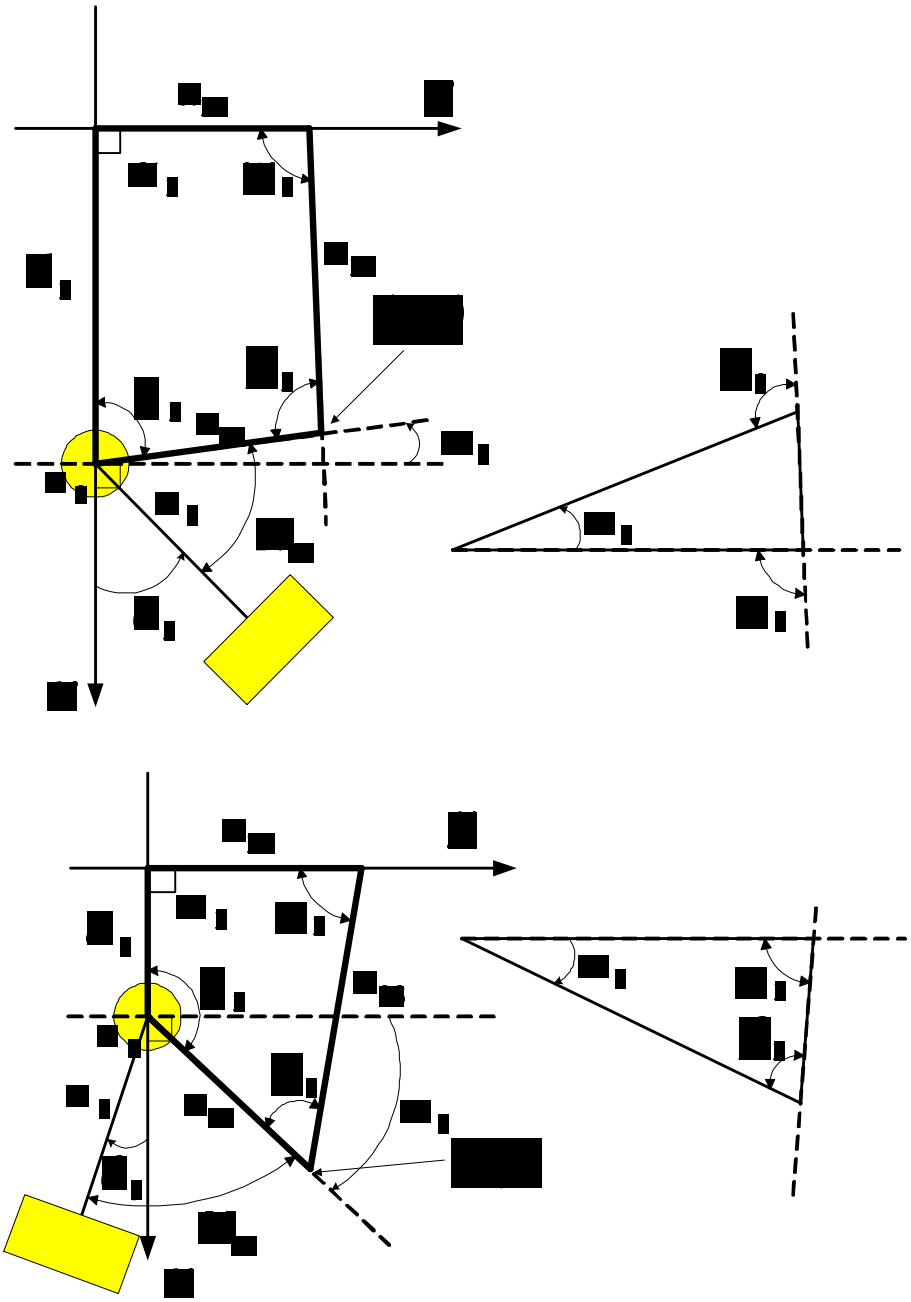


Fig 2.8 The ankle joint (2)

Fig 2.7에서 a_{33}과 a_{34}가 만나는 점 (x, y)을 표현하여 σ_3와 ψ_3의 관계식을 유도하

면 다음과 같다.

$$x = d_3 + a_{34}\sin\sigma_3 = a_{33}\sin\psi_3$$
$$y = a_{34}\cos\sigma_3 = a_{32} - a_{33}\cos\psi_3 \qquad (2.2.17)$$

$$\psi_3 = \arccos\left(\frac{a_{32} - a_{34}\cos\sigma_3}{a_{33}}\right) \qquad (2.2.18)$$

Fig 2.8 에 표현된 삼각형을 이용하여, σ_3, ψ_3의 관계식을 유도하면 다음과 같다.

$$\beta_3 + \psi_3 + \sigma_3 = \pi$$
$$\beta_3 = \pi - \psi_3 - \sigma_3 \qquad (2.2.19)$$

식 (2.2.16), (2.2.18) 이용하여 β_3와 θ_5의 관계식으로 정리하면

$$\beta_3 = \pi - \arccos\left(\frac{a_{32} - a_{34}\cos\left(\frac{\pi}{2} - N_{32} + \theta_5\right)}{a_{33}}\right) - \left(\frac{\pi}{2} - N_{32} + \theta_5\right)$$

$$= \frac{\pi}{2} + N_{32} - \theta_5 - \arccos\left(\frac{a_{32} - a_{34}\cos\left(\frac{\pi}{2} - N_{32} + \theta_5\right)}{a_{33}}\right) \qquad (2.2.20)$$

식 (3.2.15) 의 미끄럼 변위를 시간에 관해 미분하면 다음과 같이 속도 및 가속도를 구할 수 있다.

$$\dot{d}_3 = -[C_3^{\,2} + 4(A_3 + B_3\cos\beta_3)]^{-\frac{1}{2}} B_3\sin\beta_3\dot{\beta}_3 \qquad (2.2.21)$$

$$\ddot{d}_3 = -2[C_3{}^2 + 4(A_3 + B_3\cos\beta_3)]^{-\frac{3}{2}} B_3{}^2 \sin^2\beta_3 \dot{\beta}_3{}^2$$
$$-[C_3{}^2 + 4(A_3 + B_3\cos\beta_3)]^{-\frac{1}{2}} (B_3\cos\beta_3 \dot{\beta}_3{}^2 + B_3\sin\beta_3 \ddot{\beta}_3) \quad (2.2.22)$$

관절각 θ_5 와 d_3 의 속도 및 가속도 관계식을 구하면 다음과 같다.

$$\beta_3 = \arccos\left[\frac{d_3{}^2 - A_3 - C_3 d_3}{B_3}\right] \quad (2.2.23)$$

$$\dot{\beta}_3 = R_{31}\dot{d}_3 \quad (2.2.24)$$

$$\ddot{\beta}_3 = R_{32}\dot{d}_3{}^2 + R_{33}\ddot{d}_3 \quad (2.2.25)$$

식 (2.2.24) 와 (2.2.25) 에서

$$R_{31} = \frac{[C_3{}^2 + 4(A_3 + B_3\cos\beta_3)]^{\frac{1}{2}}}{B_3\sin\beta_3}$$

$$R_{32} = -2[C_3{}^2 + 4(A_3 + B_3\cos\beta_3)]^{-1} B_3\sin\beta_3 R_{31}{}^2 + \frac{\cos\beta_3}{\sin\beta_3} R_{31}{}^2$$

$$R_{33} = -\frac{[C_3 + 4(A_3 + B_3\cos\beta_3)]^{\frac{1}{2}}}{B_3\sin\beta_3}$$

3. 모션 제어기의 설계

3.1 관절 구동모터 제어기의 hardware 구성

이족 로봇을 제어하기 위해 설계한 모션제어기의 전체 시스템 구성도는 그림 3.1과 같다. 호스트 PC에서 로봇의 동작에 대한 명령을 보내고, CAN 통신을 통하여 하위 제어기에 명령을 내린다. 호스트 PC와 모션 제어기 사이에는 ARM Board가 있다. 모션제어기는 ARM Board로부터 모터 증폭부로 보내는 PWM과 회전방향 신호 명령에 따라 미리 정해진 경로 점들을 따라간다. 본 논문에서는 기존의 대용량 마이크로 프로세서를 이용한 경로 제어가 아닌 소용량의 One-chip 마이크로프로세서를 이용하여 로봇의 제어시스템을 설계하는 것이다.

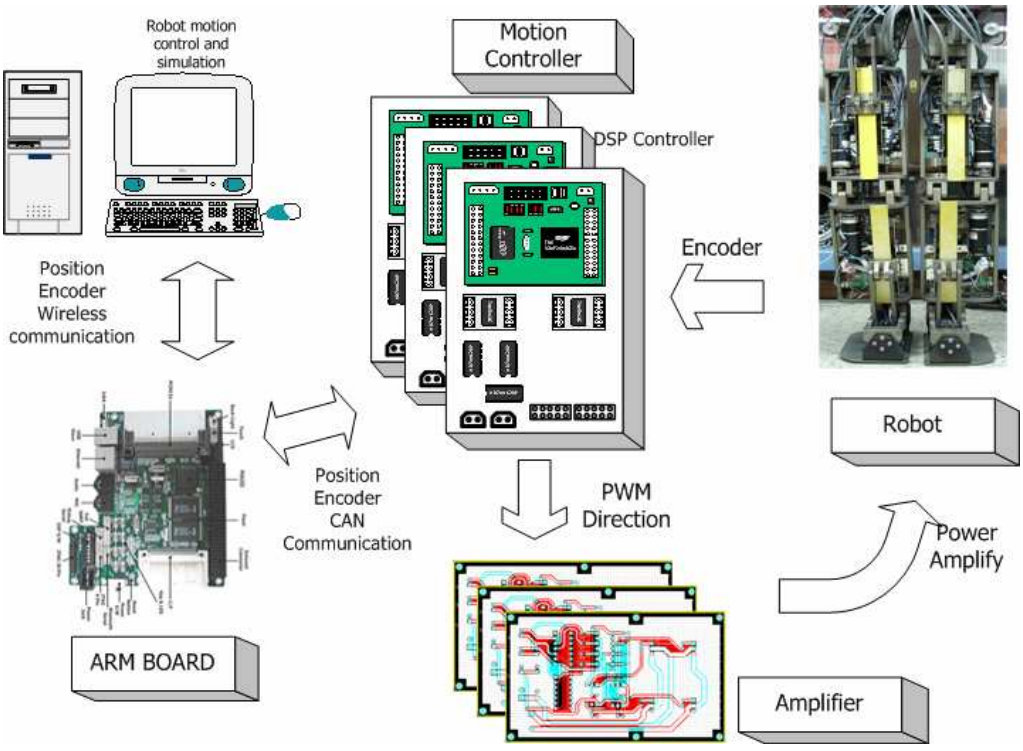


Fig 3.1 Composition of motion controller

전체 제어시스템은 크게 Host PC, 주 제어기, 관절 축 제어기 및 서보모터 드라이버로 구성된다. Host PC는 무선 이더넷 통신을 이용하여 주 제어기를 구성하는 ARM Board에 로봇의 작업명령을 내리거나 로봇 팔의 관절 구동모터의 상태를 모니터링하는 것과 제어 프로그램의 편집과 입출력을 관장하는 전체 제어시스템의 관리를 담당한다.

주 제어기인 ARM Board는 Host PC로부터 받은 로봇 팔의 경로계획에 따라 관절 구동모터 제어기 들에 순차적 혹은 동시적 동작에 대한 명령을 보내고, 각 관절 구동모터로부터 받은 엔코더 신호를 처리하여 무선통신을 통해 PC로 전송하도록 구성하였다.

관절 구동모터 제어기는 Fig 3.2과 같이 TMS320LF2407A모터 제어용 마이크로프로세서를 기반으로 두 대의 서보모터를 제어할 수 있도록 엔코더 카운터 두 개가 내장된 전용 모터 제어보드를 구성하여 한 개의 제어보드가 두 축의 모터를 동시에 제어할 수 있도록 구성하였다. 관절 구동모터 제어기의 제원은 Table3.1와 같다.

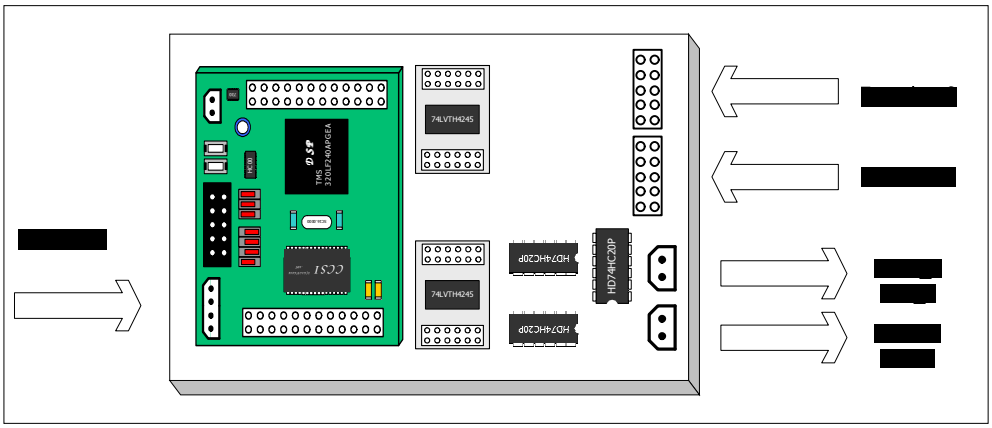
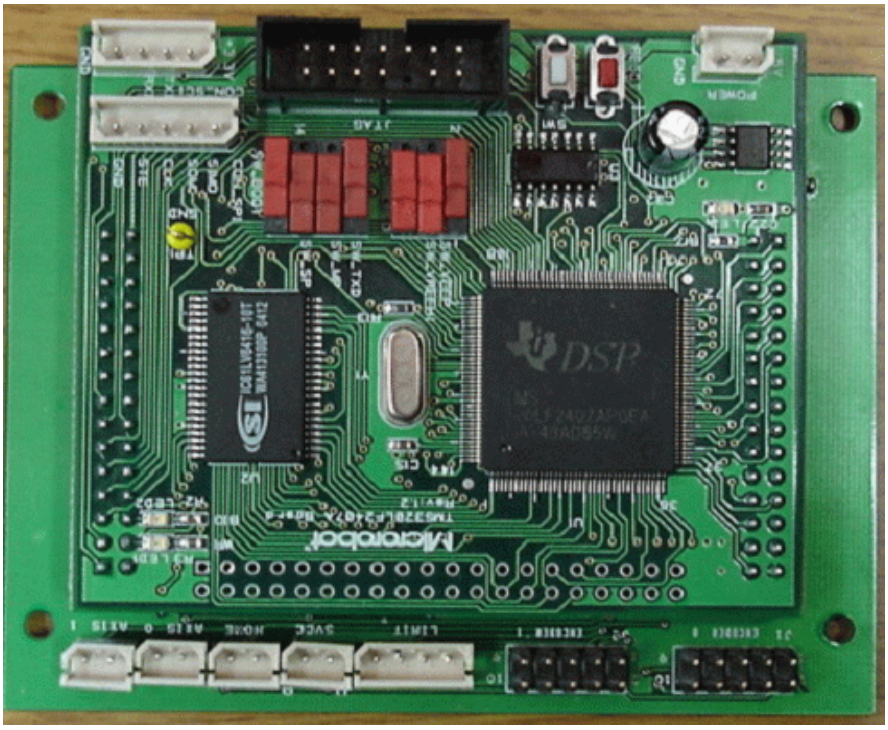


Fig 3.2 Composition of motion controller



Pic 3.1 Outlook of motion controller

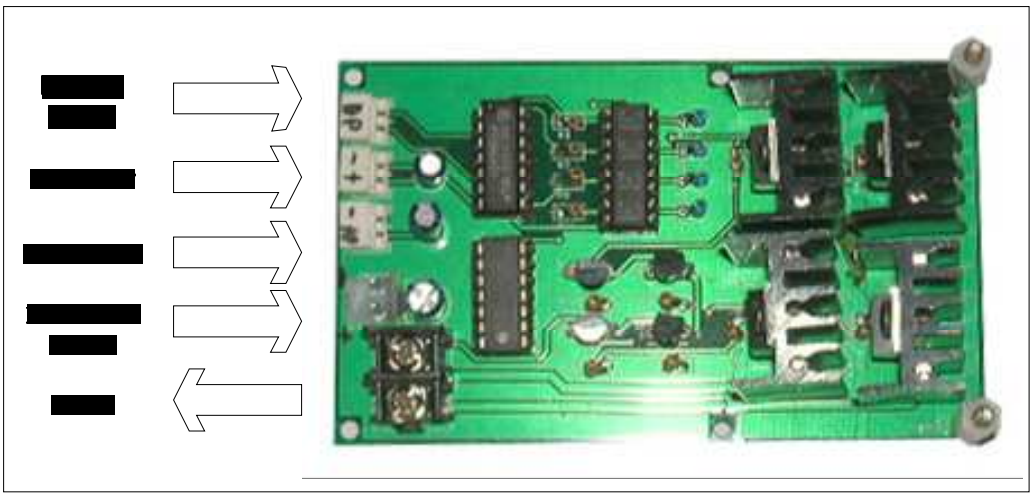
모션 제어기의 제원은 아래와 같다.

Table 3.1 property of motion controller

부 품	부 품 명	기 능
MCU	TMS320LF2407A	PWM, Encoder counter,UART, Timer, GPIO
Level Shifter	74LVHT4245	3.3V <-> 5V
DeadTime Generator	CD4538	Dual Precision Monostable

모터 드라이버는 구동모터 제어기에서 제어알고리즘에 따라 출력되는

Pulse width modulation(PWM)과 모터회전 방향 신호에 따라 PWM 듀티비에 비례 증폭되어 모터에 전압을 공급한다. 본 연구에서는 Power Transister와 브릿지 회로를 구성하여 모터 드라이버를 직접 제작하였고 이의 실제 사진은 Fig3.2과 같다.



Pic 3.2 Outlook of the motor driver

Fig3.3는 모터 드라이버의 회로도이다.

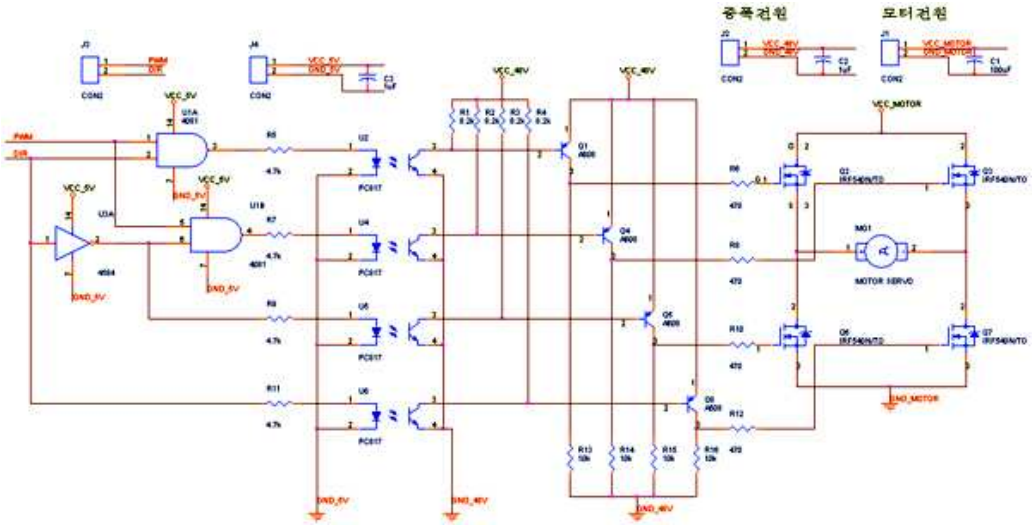


Fig 3.3 Outlook of the motor driver circuit

모터 드라이버 회로도에서 효율을 높이기 위해서는 FET의 Drain 보다 Gate 에 전압이 1.5배 이상 높아야 한다. Gate 에 공급되는 전압원으로 Pic 3.3 과 같은 전원부를 제작하였다.

Pic 3.3 Outlook of the power sources amplifier

3.2 관절 구동모터 경로 및 제어알고리즘 설계

로봇을 원하는 경로를 따라 원하는 속도로 제어하기 위해서는 적합한 경로 궤적을 설계하고 이에 따라서 관절 구동모터의 위치와 속도를 제어하는 것이 매우 중요하다. 본 연구에서는 TMS320LF2407A 모터 제어용 마이크로프로세서를 이용하여 관절 구동모터의 위치와 속도를 제어하기 위한 경로 궤적을 설계하였다. 이는 메모리 용량이 제한적이므로 궤적을 추종하는 경로점의 설정이 1000이내의 제한적 문제를 갖고 있다. 모터의 경로 추종을 위한 입력 데이터를 생성하기 위해서는 우선적으로 제어의 목표값으로 이용되는 목표궤적을 설정해야 하며, 궤적 중에는 연속적인 많은 경로점들의 집합으로 불리는 “구동프로파일”을 생성해야 한다. 이를 이용하여 실제 모터의 상태와 추종해야 하는 구동프로파일 상의 상태와 비교해서 발생하는 오차를 제거하는 PID제어 알고리즘을 적용한다.

본 논문에서는 Fig 3.4 과 같이 보편적인 사다리꼴 속도와 이에 상응하는 가속도 및 위치궤적을 적용하였다.

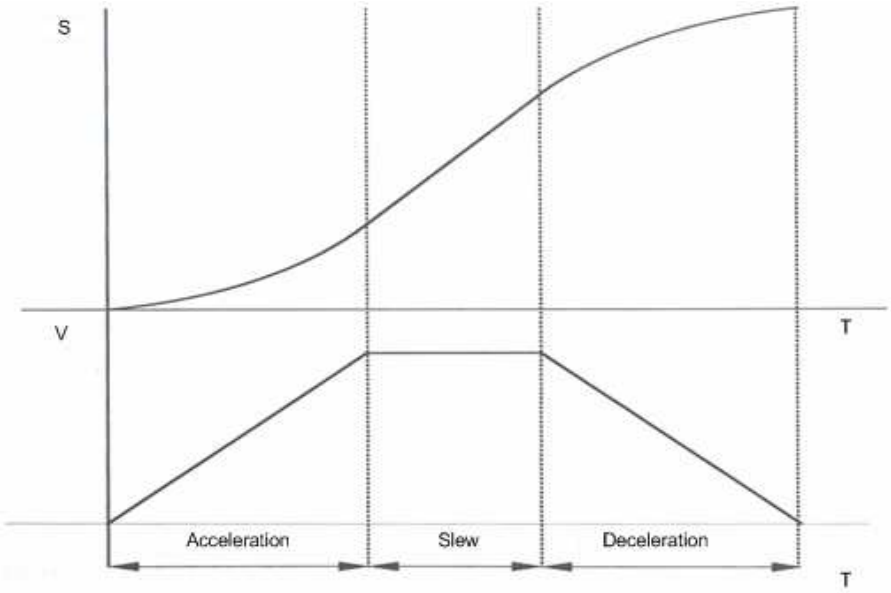


Fig 3.4 Reference profile for motors

관절 구동 모터들이 추종해야 할 궤적은 위치 궤적과 이의 도함수인 속도 및 가속도 궤적으로 구성된다. 경로 궤적 상의 1000개 정도의 경유점을 Off-line 상에서 Table을 구성하여 설정하고 모터가 이들을 추종하게 제어한다. 인터럽트 주파수가 빨라 질수록 구동 프로파일 Table에 많은 점들을 생성하여 이들을 경유하게 되고 속도가 부드러워 지는 반면 CPU에 많은 부하가 걸리게 되므로 고속의 CPU를 필요로 하게 된다. 따라서 본 연구에서는 500HZ 의 인터럽트 주파수를 생성하여 계산이 비교적 간단하고 저가의 DSP에서도 사용이 가능한 사다리꼴 속도 궤적을 다음과 같이 설계하였다.

$$S_t = S_a + S_c + S_r \qquad (3.2.1)$$

여기서 S_t는 총이동거리, S_a와 S_r는 각각 가감속시 이동거리, 그리고 S_c는 등속시 이동거리이다. 여기서 S_t가 역기구학 해석에 의해 결정되면 S_a와 S_r를 임의로 결정

하여 S_c 를 결정할 수 있다. 여기서 일반적으로 $V_{\max}$ 와 T_a 의 관계는 다음과 같다.

$$S_a = S_r = V_{\max} * T_a / 2 \quad (3.2.2)$$

로봇다리가 목적지까지 도착하는 시간은 모든 구동모터가 도착하는 시간과 일치하도록 설계된다. 따라서 각 구동 모터의 가감속 시간과 등속시간의 합으로 다음과 같이 표현된다.

$$T_t = T_a + T_r + S_c / V_{\max} \quad (3.2.3)$$

여기서 T_t 은 총 소요시간, T_a 는 가속시간, I_p 는 인터럽트 주기가 된다. 가감속과 등속시의 인터럽트 발생횟수는 각각 T_r / I_p, T_a / I_p, 그리고 $S_c / (V_{\max} I_p)$ 이고 총 인터럽터 발생횟수는 이들의 합으로 알 수 있다. 따라서 본 연구에서는 먼저 인터럽터 주기를 결정하고 이동하고자 하는 거리를 받아서 총 인터럽트 발생횟수를 계산하고 이를 가감속과 등속 구간으로 나누어 속도 프로파일을 생성하였다.

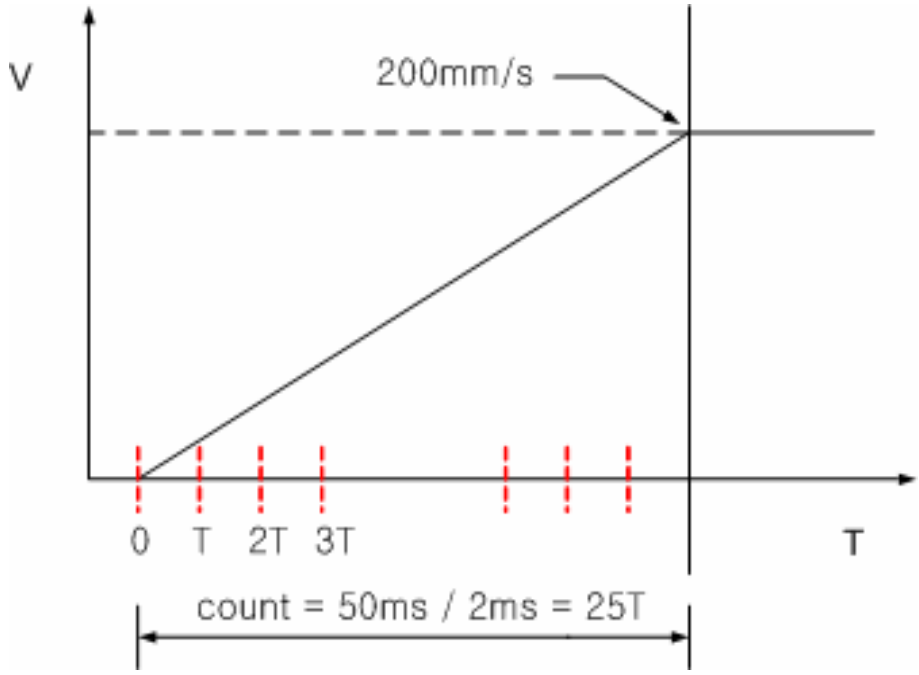


Fig 3.5 velocity change in acceleration profile

가속구간과 감속구간은 궤적 추종을 잘 할 수 있도록 신속한 제어를 위해 미리 Off-line에서 자동 연산한 모터의 기준속도를 테이블로 만들었다. 여기서 위치테이블은 설계한 속도 테이블을 매 인터럽트마다 적분하여 생성한다. 선택한 Fig 3.5 의 가감속 구간에 속하는 위치궤적은 S자형의 위치프로파일로 다음의 식 (3.2.4)들로 표현된다. 여기서 마이크로프로세서에 적용하기 위해서 본 논문에서는 512개의 위치경로점을 생성하였고, 속도는 매 인터럽트 시간인 1ms 내에서 미분하여 적용하였다.

$$y = a + bt + ct^2 + dt^3 \tag{3.2.4a}$$

$$y' = b + 2ct + 3dt^2 \tag{3.2.4b}$$

$$y'' = 2c + 6dt \tag{3.2.4c}$$

여기서 식 (3.2.4a), (3.2.4b) 및 (3.2.4c) 는 각각 위치, 속도 및 가속도 함수를 나타낸다. 본 논문에서는 연산시간이 적게 걸리고 실용성이 높은 로봇에 일반적으로 사용되는 PID 제어기를 적용하였다. 이 제어기를 매우 짧은 1ms 인터럽트 시간 동안

연산하여 모터에 적용하였다. 인터럽트 시간 안에서 수행되는 기본 동작은 Fig 3.6 와 같다

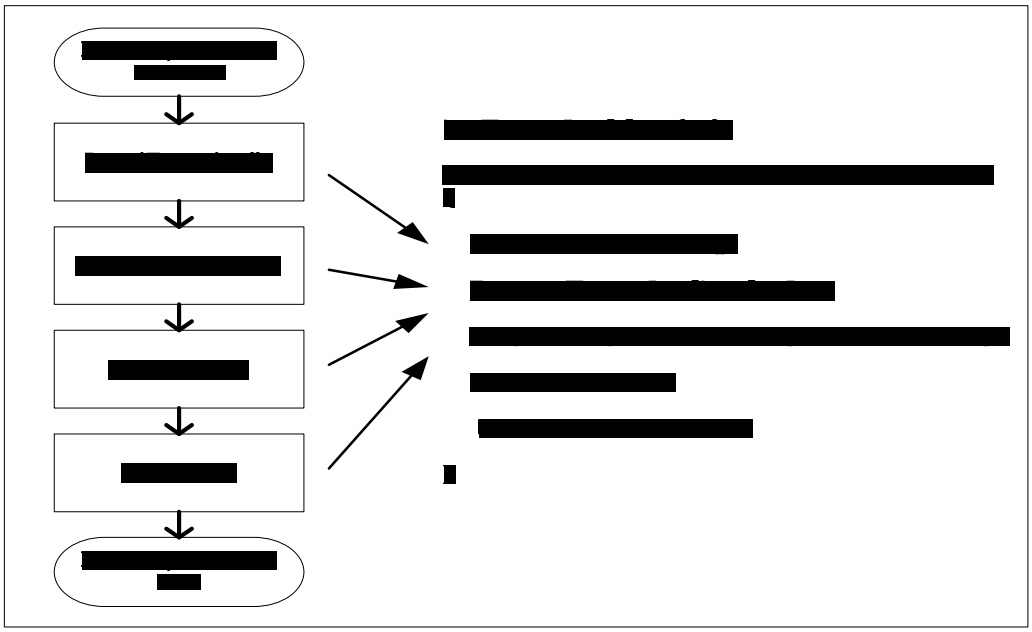


Fig 3.6 block diagram of interrupt routine

본 연구에서는 Fig.11와 같은 feed-forward 항을 갖는 PID 제어기를 적용하였다.

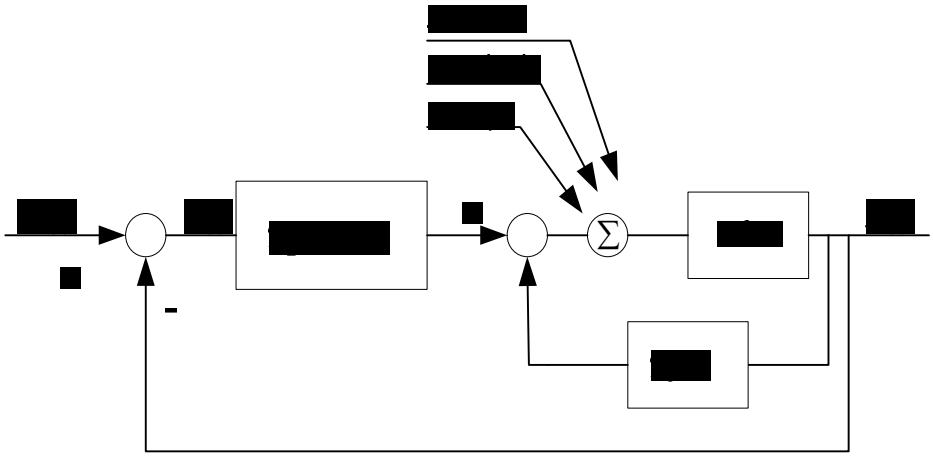


Fig 3.7 PID controller with Feedforward term

4. 로봇의 상위 제어시스템 설계 및 CAN통신 구현

4.1 임베디드 시스템

임베디드 시스템(embedded system)이란 미리 정해진 특정한 기능들을 수행하기 위하여 컴퓨터의 하드웨어와 소프트웨어가 서로 결합된 고기능의 전자 제어 시스템을 말한다. 즉, 마이크로프로세스를 구동하여 특정한 기능을 수행하도록 프로그램이 입력되어 있는 내장시스템을 가진다.

4.1.1 임베디드 시스템 개요

최근 들어와서 “Embedded System(독립장비)”이 많이 거론이 되고 있다. 이는 바로 PC시장의 포화현상, 개인용 휴대장비의 다양화, 기술의 표준화 및 고도화가 급속도로 이루어 지고 있기 때문이다. 그리고 PC시장의 발전은 CPU발전이 주도를 하고 있고 전체 시장의 크기와 팽창은 주춤하고 사무환경의 변화와 소비자의 다양한 욕구를 모두 수용하기에는 어려운 점이 많다.

이는 기존의 산업전자와 공장자동화 분야를 위주로 발전을 하였던 Embedded 분야가 기능의 다양화 및 지능화, 여러가지 인터페이스의 지원과 GUI를 이용한 Visual한 표현, 네트웍 기능의 수용으로 차츰 산업용의 목적을 충실할 뿐만 아니라 일반 사용자의 요구도 수용을 해나가고 있다. 대표적인 분야는 핸드폰, 전자수첩, 게임기, 가전제품 등이 있다.

Fig 4.1 Embedded system application

4.1.2 임베디드 운영체제와 실시간 임베디드 리눅스

예전에는 System이 비교적 간단해서 OS의 개념을 적용하지 않고 순차적인 Program을 작성해 왔기 때문에 Embedded System에서 OS가 필요하지 않았다.하지만 시스템이 복잡하고 해야 할 일이 많아 짐에 따라 순차적인 Program으로 작성하기에는 매우 어렵게 되었다 따라서 임베디드 시스템에서도 멀티 기능을 지원하는 OS의 필요성이 대두됨과 동시에 임베디드 시스템의 특성상 실시간 (real time)이라는 요소를 만족해야 하였으므로 실시간 운영체제가 임베디드 시스템에 도입되었다. 본 연구에서도 휴머노이드 로봇이라는 복잡한 시스템을 정확하고 효율적으로 운용하기 위하여 실시간 리눅스 시스템을 구성하여 사용하였다.

(1) 전경/배경 시스템

소형이면서 복잡하지 않은 시스템은 일반적으로 Fig 4.2 처럼 디자인 한다. 이런 시스템을 전경/배경 시스템 혹은 수퍼-루프(super-loop)라 한다. 응용프로그램은 요구

되는 동작을 수행하기 위해서 특정한 모듈이나 함수를 호출하는 무한루프로 구성된다(배경 프로세스). 인터럽트 서비스 루틴(interrupt Service Routine 또는 ISR)은 비동기적으로 발생하는 이벤트를 처리한다(전경 프로세스). 일정한 시간 내에 수행해야 할 중요한 동작은 반드시 ISR이 처리해야 한다. 이 때문에 ISR은 원래의 수행시간보다 길어지는 경향이 있다. 또한 ISR이 배경 프로세스로 전달하고자 하는 정보는 배경 프로세스가 다시 실행되기 전까지는 처리되지 않는다. 이것은 태스크 레벨 응답이라고 한다. 최악의 태스크 레벨 응답시간은 배경 루프를 한 바퀴 실행하는데 걸리는 시간에 좌우된다. 일반적인 코드는 실행시간이 일정하지 않기 때문에 루프를 수행하는데 걸리는 시간을 예측하기 어렵다.

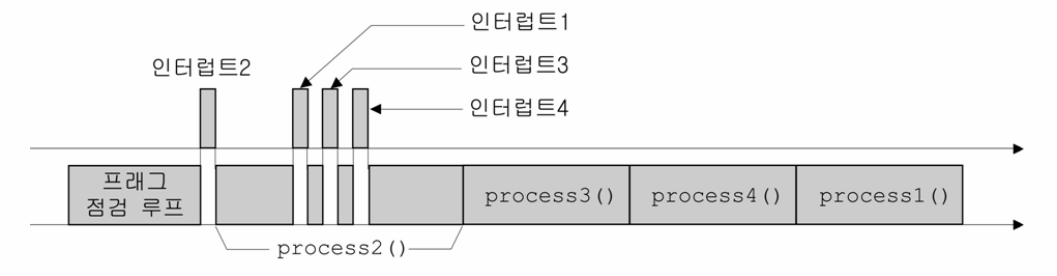


Fig 4.2 Composition of the super-loop system

(2) 임베디드 운영체제

OS란 Operating System의 약자로서 우리말로 운영체제라고 번역한다. 우리가 쓰고 있는 모든 컴퓨터에는 OS가 들어가 있다. PC에서 흔히 볼 수 있는 DOS, Windows 95, Windows NT, OS/2들이 모두 OS들이고 Workstation이나 그 이상의 컴퓨터에서는 UNIX나 그의 변종인 OS들이 수행되고 있다.

임베디드 Real-time OS는 일반적인 OS와 마찬가지로 여러 가지 task를 동시에(가상적인 의미에서) 수행하는 것이다. 대부분의 Real-time OS는 다음과 같이 Task Scheduling, Task Communication, Task Synchronization, Memory Management, Interrupt Service, I/O Driver, Timer 등의 기능을 하며 Real-time이기 때문에 Real-time을 처리하는 OS로 효율적인것 외에 그 속도에 중요성이 있다.

(3) 임베디드 리눅스

리눅스는 리누스 토발즈가 미닉스를 수정하여 인터넷에 공개한 데서부터 시작 된

것으로 특정 어플리케이션에 맞도록 리눅스 커널을 최적화한 것이다. 라이센스비가 무료이며 커널 소스 또한 공개되며 현재 전 세계의 수많은 개발자들에 의해 지속적으로 업데이트 되는 장점에 더한 다음과 같은 장정이 있다.
임베디드 리눅스의 장점은 다음과 같다.

• Linux는 역사가 깊고 많은 사람이 사용한다.
• Open Source, Open Architecture이다.
(전세계에 수많은 news group와 디버깅 FAQ 및 기술자들이 산재해 있다.)
• 소규모 모듈단위로 설계가 되어있다. (POSIX를 지원한다.)
• Real Time운영을 지원한다.
(유망분야 : 휴먼 로봇, 원자력 작업장비, 무인 탐사선, 네트웍 장비)

단점은 다음과 같다.
• 임베디드 리눅스 위에서 실행될 수 있는 어플리케이션이 상당히 부족하다.
• 임베디드 리눅스를 이용하여 장비개발시 개발환경에 접근하는데 상당히 어렵다.

본 연구에서는 리눅스 커널 버전 2.4.18을 이용하여 암 보드에 포팅하여 사용을 하였다. 다수의 모션 제어기를 구동 시키고 효율적인시스템의 운영을 위해 임베디드 ARM Board 에 LINUX를 OS로 구성하고 이에 기반을 둔 CAN 통신시스템을 구성하였다.

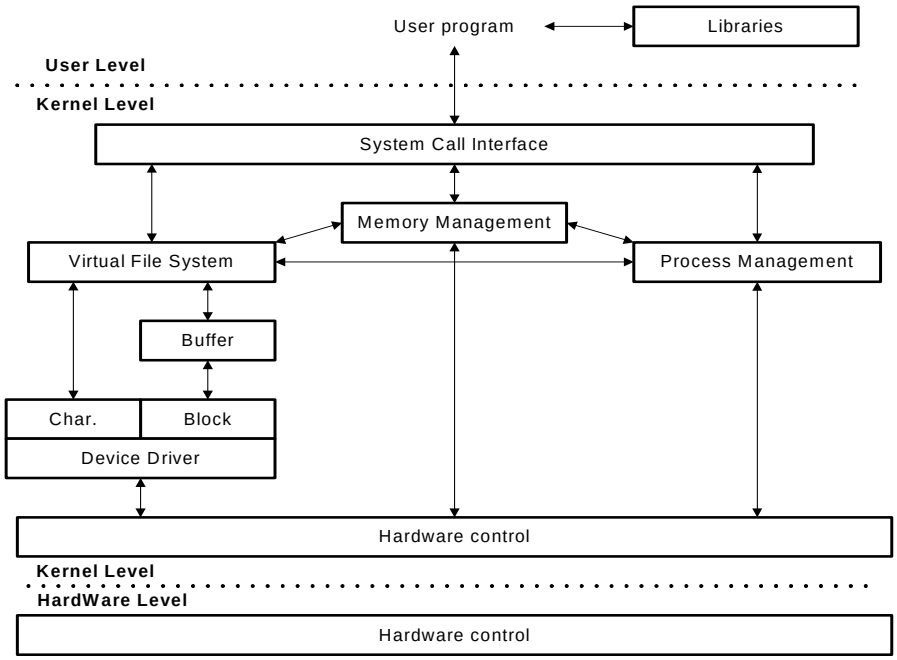


Fig 4.3 Composition of linux

4.2 ARM보드를 이용한 제어시스템 설계

PC의 프로세서보다는 저성능이고 소 용량의 메모리를 가지나 일반적인 one-chip 마이크로프로세서보다는 고성능이고 큰 용량을 가지고 실시간 처리 기능을 가진 ARM 프로세서가 탑제된 보드를 이용하여 상위 제어 시스템을 설계 하였다. 이는 로봇에 특화된 시스템으로 구성을 할 경우 PC보다 더욱 뛰어난 기능구현이 가능하고 일반 one-chip마이크로프로세서로 구현이 어려운 로봇의 멀티 프로세싱과 멀티 미디어 및 네트워크 기능을 구현이 가능하다.

임베디드 리눅스를 임베디드 실시간 리눅스(embedded realtime linux)라 하는데, 본 연구에서는 이를 이용하여 제어시스템의 OS를 구성하였다. LINUX를 이용하여 리눅스 시스템을 운영하기 위해서는 하드웨어가 지원해줘야 하는 부분이 몇 가지 있다. 먼저 메모리 관리 유닛(MMU, Memory Management Unit)이 있는 최소 32비트 CPU가 있어야 하고, 시스템을 효과적으로 운영하려면 충분한 양의 메모리도 필요하다. 그리고 개발을 할 때 타겟에서 제대로 디버깅을 하고자 한다면 최소한의 입출력 기능도

필요한데, 이는 실제 개발 과정에서 어려운 문제를 해결하고자 할 때 매우 중요하다. 마지막으로, 커널이 영구적인 스토리지나 네트워크 스토리지를 통해 루트 파일시스템을 로드 하거나 루트 파일 시스템에 접근할 수 있어야 한다. 이런 이유로 인해 X-hyper255B ARM Board를 이용하여 제어기를 구성하고 개발환경을 구축, 커널포팅, 디바이스 드라이버 제작, 어플리케이션 작성을 하였다.

(1)X-hyper255B ARM Board 특징.

- Intel PXA255 Processor가 탑재되어 저전력, 고성능(400MHz)의 특징을 가지고 있어서 Mobile 관련 제품군에 뛰어난 성능을 발휘할 수 있다.
- 최고의 안정성을 자랑하는 Linux Kernel 최신 버전(2.4.18)이 탑재되어 사용자가 원하는 프로그램을 안정성있게 운용할 수 있고 리눅스에서 네트웍의 강점을 가져갈 수가 있다.
- 하드디스크 등 별도의 저장 매체가 필요없는 환경에서 응용프로그램이 실행될 수 있고 MTD를 통한 Flash Filesystem(JFFS2)가 구현되어 있어 용량 활용의 극대화와 안정적인 실행이 가능하다.
- PXA255의 확장 Interface를 제공하여 개발의 유연성 및 자유로운 확장이 가능하다.

(2)Hardware specification

하드웨어에 대한 스펙은 다음과 같다. Intel PXA255 CPU 400MHz 의 고성능 CPU가 탑재되었고 메모리도 64M 대용량으로 장착되어 사이즈가 큰 프로그램도 무난히 동작할 수 있다. 이외에 각종 인터페이스와 확장 핀이 제공되어 PXA255가 제공하는 성능을 자유롭게 확장하여 사용할 수 있다.

Table 4.1 Hardware specification

Item	Description
Processor	Intel PXA255 400MHz
SDRAM	Samsung 64Mbyte
Flash	Intel strata flash 32MByte
Ethernet	CS8900A 10BaseT
Audio	AC'97 Stereo audio

Display	LG TFT LCD 6.4" (640 * 480)
Touch	ADS7843 (Touch screen)
USB	USB Slave
Serial	2 Port
JTAG	1 Port
PCMCIA	1 Slot
RTC	RTC4513(Real Time Clock)
IrDA	HDSL3600
CF	1 Slot
MMC	1 Slot
Jack	Mic. Speaker Jack
Keypad	8 Key Button
Connector	120PIN Connetor for GPIO, Address and Data BUS

(3)Software specification

소프트웨어에 대한 스펙은 다음과 같다. 임베디드 리눅스 버전2.4.18 커널이 탑재되었고 Tiny X Server를 통한 GUI 환경이 제공되어 사용자의 편이성 및 개발에 대한 시험 범위가 넓다. 각종 H/W에 대한 디바이스 드라이버 일체가 제공되어서 디바이스 드라이버 개발에 용이하다.

Table 4.2 Software specification

Item	Description
O/S	Linux 2.4.18 kernel
Device Driver	CS8900 Ethernet
	AC97 streo audio
	Frame buffer
	ADS7843 (Touch screen)
	USB Slave
	PCMCIA, CF

	MMC
	RTC4513(Real Time Clock)
	IrDA
File System	JFFS2, Ramdisk
GUI	Tiny X Server
	MP3 Player
	MPEG Player
	Game

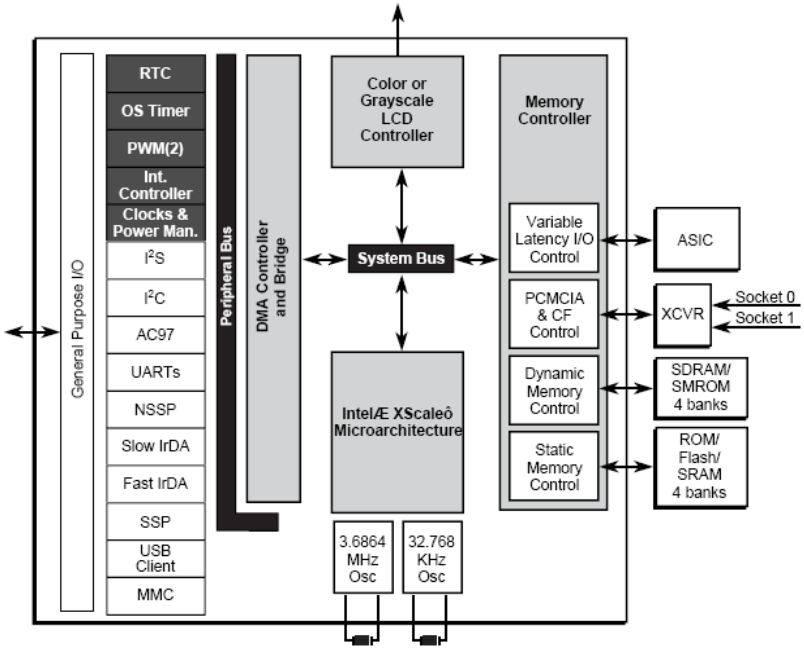
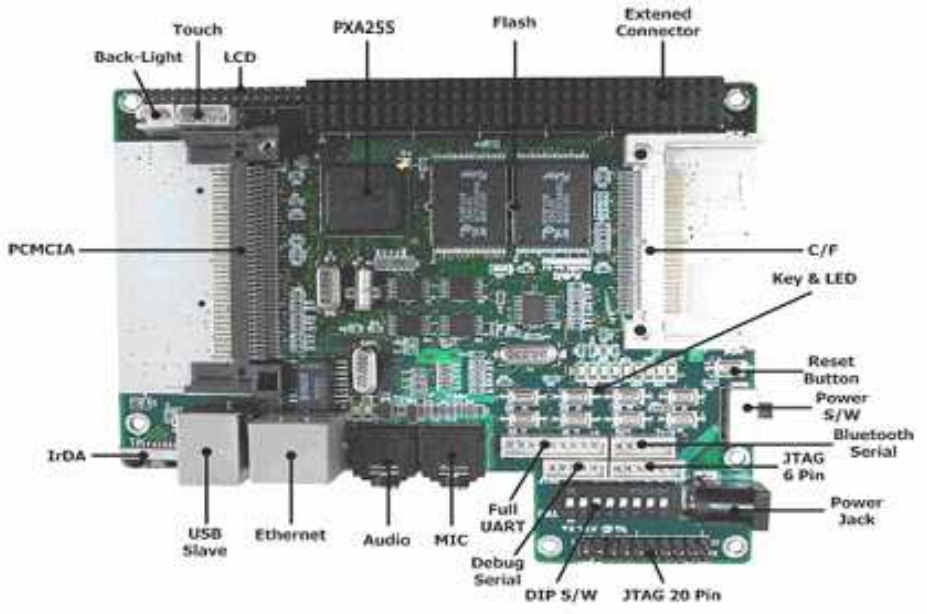
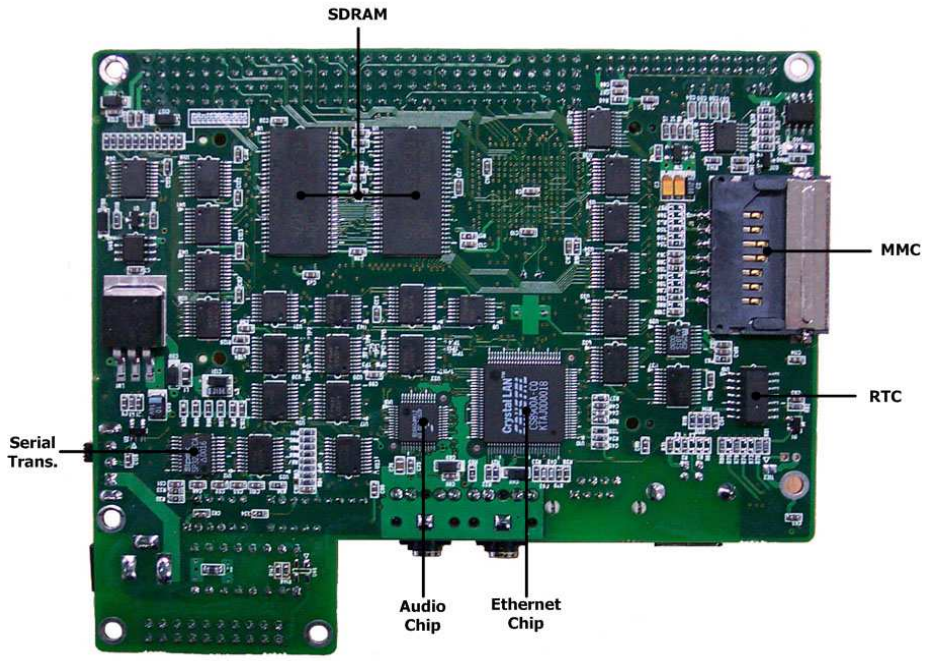


Fig 4.4 Composition of arm-core



Pic 4.1 X-Hyper 255B front



Pic 4.2 X-Hyper255B back

4.2.1 포팅

(1)개발환경 설정

임베디드 리눅스의 개발과 포팅을 위한 하드웨어 구성은 Fig 4.5 과 같은 형태를 취하고 호스트 시스템으로는 리눅스가 설치된 PC를 사용한다. 여타 다른 OS를 사용할 경우 호환성 문제로 인해서 사용이 많이 어려워 질 수가 있다. 본 연구에서 리눅스 배포 판인 레드햇 9.0이 설치된 PC를 호스트로 사용을 하였다.

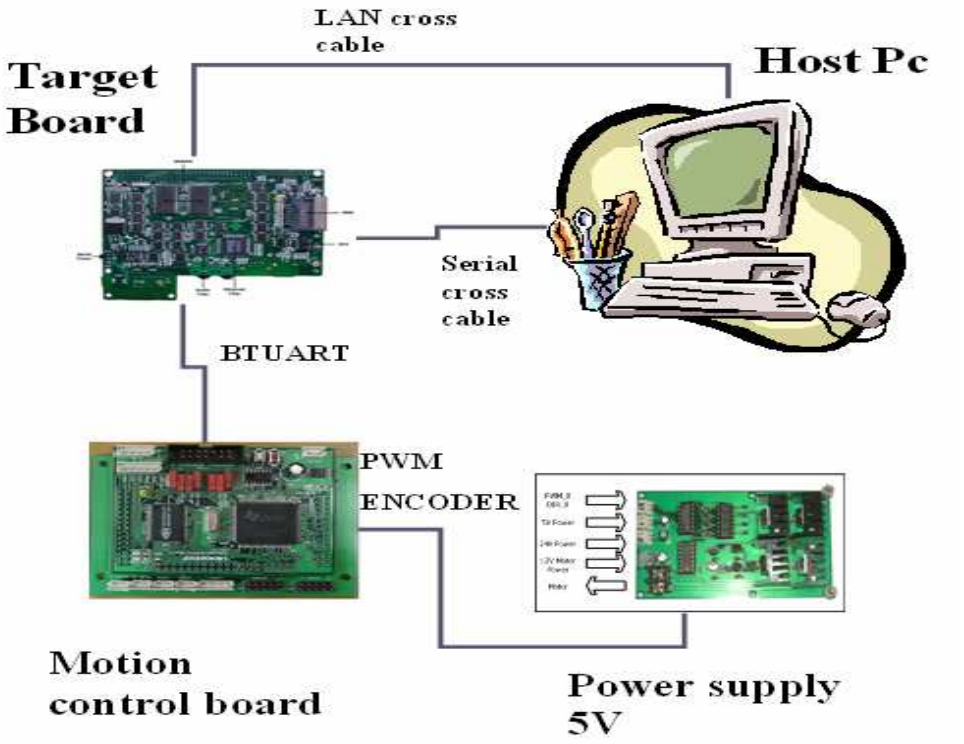


Fig 4.5 Embedded system of development environment

리눅스 포팅을 위한 순서는 다음과 같다.

1 크로스 컴파일러 설치

- 커로스 컴파일러 개발 환경이란 호스트 시스템에 구현하고자 하는 임베디드 시스템용 리눅스를 개발하기 위한 환경을 말한다. 먼저 해당 프로세서에 맞는 툴

체인(tool chain)환경을 구축해야 한다. 툴 체인은 각종 프로그램 소스들을 컴파일하고 빌드하여 실행 가능한 바이너리(binary)를 생성하는데 필요한 각종 유틸리티 및 라이브러리의 모음이다. 기본적으로 어셈블러, 링커, C컴파일러, C라이브러리 등으로 구성된다. 보드 차체적으로 지원되는 크로스 컴파일러를 설치하여 암 보드용 파일들을 컴파일 하여 사용 하였다.

l JTAG케이블을 이용 Bootload 설치

- JTAG가 제공하는 기능을 살펴보면 프로세서의 상태와는 상관없이 디바이스의 모든 외부 핀을 구동시키거나 값을 읽어 들일 수 있다. 기능을 보면 디바이스 내에서 모든 왜부와의 연결점을 가로챈다. 그리고 회로의 배선과 소자의 전기적 연결상태를 테스트하거나 디바이스간의 연결상태를 테스트하거나 플래시 메모리에 데이터를 읽거나 쓰기 위함이다.

타겟 보드의 전원이 들어옴과 동시에 시작이 되어야 하는 부트로더는 일반 PC의 ROM BOIS를 대신하여 여러 하드웨어 설정을 담당하고, 커널의 로딩으로 이어지는 중요한 역할을 담당하게 된다. 한번 설정된 하드웨어 설정은 다시 바꾸어 주지 않는 한 커널 로딩 후에도 계속 유효하므로 부트로더 포팅시 주의를 기울여 작성하여야 한다.

본 연구에서는 사용된 암 보드의 JTAG기능을 이용하여 호스트 PC에서 크로스 컴파일러로 컴파일을 하여 플레쉬롬 부트로드 영역에 다운로드 하여 사용을 하였다.

(2)호스트와 임베디드 시스템 사이의 통신

타겟 보드를 좀 더 효율적으로 운용하고 개발시간 단축을 위해서는 각종 하드웨어 적인 통신이 설치되어 있어야 한다. 타겟 보드 운용에 필요한 통신은 다음과 같다.

l Minicom과 하이퍼 터미널

- Minicom과 하이퍼 터미널의 두 프로그램은 호스트 시스템 쪽에서 직렬 통신을 위한 프로그램들로 Minicom은 리눅스에서, 하이퍼터미널은 마이크로 소프트 윈도우에서 사용되는 프로그램이다. 본 연구에서 직렬 통신을 이용해서 암 보드의 모니터 역할을 하고 있기 때문에 이 두 프로그램을 사용해서 부트로더의 명령 프롬프트 및 임베디드 시스템어서 실행되고 있는 임베디드 리눅스의 쉘 프롬프트에

명령을 전달하였다.

l Bootp

- Bootp는 TCP/IP상에서 자동 부팅을 위한 최초의 표준으로, 디스크 장치가 없는 클라이언트를 위한 프로토콜이며 Udp와 Tftp프로토콜을 사용한다. 호스트 시스템과 임베디드 시스템간의 접속을 연결하고 각종 정보들을 가져오기 위한 준비 절차라고 이다. 이는 Bootp를 통해서 호스트 PC로부터 IP를 할당 받아서 타겟보드로 Tftp를 통해 데이터 전송이 이루어지게 하였다.

l Tftp

- Tftp(trivial file protocol)가 Ftp와 비슷한 점은 네트워크를 통한 파일 전송 서비스 이다.그러나 ftp는 신뢰성을 보장하는 TCP전송 방식을 사용하지만 tftp는 단방향 핸드셰이킹 방법인 UDP를 통한 전송 방식을 사용한다. UDP 는 단지 메시지를 브로드케스트하게 되어 수신 측에서 수신하는지에 상관없이 목적지를 향해 데이터를 보낸다. 그러므로 라우팅이 복장하거나 네트워크가 혼잡할 경우 패킷이 유실될 수 있다. 하지만 개발 시에 본 연구에서는 타겟보드와 호스트 간에 1:1로 연결이 되어있어 패킷 유실의 위험이 매우 적었고 커널과 파일 시스템의 용량이 많았기 때문에 tftp를 통해서 커널과 파일시스템을 포팅 하였다.

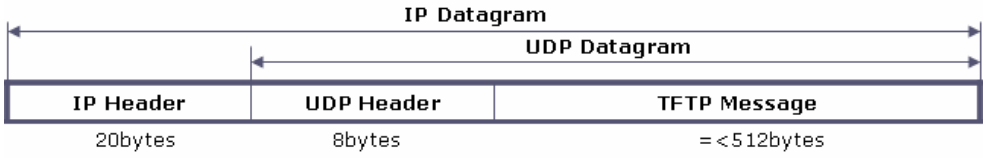


Fig 4.6 Composition of tftp packet

l Nfs

- NFS(Network File System)는 선 마이크로시스템즈가 개발하여 RFC 1094 에서 정의된 프로토콜로서 컴퓨터 시스템이 네트워크의 다른 부분과 네트워크 주소를 공유하고 서브넷 번호에 의해 구별되는 네트워크 세그먼트에 관계없이 데이터를 액세스할 수 있게 지원한다. NFS는 컴퓨터 사용자가 원격지의 컴퓨터에 있는 파일을 마치 자신의 컴퓨터에 있는 것처럼 검색하고, 마음대로 저장하거나 수정할 수 있게 해주는 클라이언트/서버형 응용프로그램이다. 사용자 시스템에는

NFS 클라인언트가 있어야 하며, 다른 컴퓨터에는 NFS서버가 설치되어 있어야 한다. 또한, 양쪽에는 모두 TCP/IP플토콜이 설치되어 있어야 하는데, 왜냐하면 NFS 서버와 클라이언트가 파일을 보내거나 수정하는 프로그램으로 TCP/IP를 사용하기 때문이다. 따라서 NFS기능을 임베디드 시스템 개발에 이용하면 호스트 시스템의 특정 디렉토리를 임베디드 시스템에서 공유할 수 있게되어 적은 자원을 가지고 있는 임베디드 보드를 매유 유용하게 이용을 할 수 있고 개발 시 애플리케이션 프로그램을 타겟에 다운로드 하지 않고도 바로 테스트가 가능하기 때문에 많은 시간 절약도 가져오게 된다. 이번 연구에서도 호스트에 NFS서버를 구동시키고 따로 폴더를 만들어서 응용 프로그램 개발을 하였다.

(3)커널과 파일 시스템 적재

l 커널

- 커널이란 운영체제를 구성하고 있는 핵심(core)로써 타겟보드의 DRAM에 상주하여, 시스템의 구동에 필요한 환경 설정과 수행되는 프로세스들을 스케줄링 하는 소프트웨어이다. 커널은 크게 마이크로 커널(Micro Kernel)과 모놀리틱 커널(Monolithic)로 나눌 수 있으며, 마이크로 커널은 커널이 가져야 하는 핵심적인 기능만을 구현한 최소 커널로써 나머지는 서비스 프로세스로 이루어 진다. 모놀리틱 커널이란 커널 내부에 시스템 운영에 필요한 많은 서비스 루틴들을 포함한 구조를 가지고 있다. 임베디드 시스템 에서는 커널의 용량이 중요하게 작용을 하기 때문에 대부분 마이크로 커널을 이용을 하고 있다. 제작된 시스템 에서 리눅스 마이크로 커널 버전 2.4.18을 사용하여 암 칩과 보드 패치를 적용하고 설치된 크로스 컴파일러로 zImage를 생성하여 보드에 포팅 하였다.

l 파일 시스템

- 파일 시스템이란 운영체제가 파티션이나 디스크에 파일들이 연속되게 하기 위해 사용하는 방법들이고 자료구조이다. 파일시스템이라는 말은 파일을 저장하는데 사용되는 파티션이나 디스크를 가리킬 때나, 파일 시스템의 형식을 가리킬 때 사용되기도 한다. 사용된 보드는 효율적인 메모리 사용이 가능한 jffs2 파일 시스템을 사용하여 별도의 물리적 저장 장치가 존재하지 않는 임베디드 보드의 메모리를 컴퓨터의 하드 디스크처럼 사용가능 하도록 하는 ramdisk 이미지를 생성

하여 보드에 다운로드 하여 사용하였다.

4.2.2 디바이스 드라이버 설치

디바이스(device)란 하드 디스크(Hard Disk), 플라피 디스크(Floppy Disk), 프린터(Printer), 단말기(Terminal), 스캐너(Scanner), 네트워크 어댑터(Network Adaptor), PCMCIA, Touch Screen, LCD 디스플레이, Audio 등과 같이 컴퓨터 시스템 이외의 다른 주변 장치를 말한다. 디바이스 드라이버 프로그램은 리눅스 OS가 자체적으로 지원 하는 것도 있지만, 대부분 디바이스에 맞는 드라이버 프로그램을 찾아서 인스톨 시켜야 한다. 디바이스 드라이버의 기능을 요약 하면 다음과 같다.

- 디바이스(device)와 시스템 사이에 데이터를 주고받기 위한 인터페이스(interface)
- 표준적으로 동일한 서비스 제공을 목적
- 커널(kernel)의 일부분으로 내장
- 서브루틴과 데이터의 집합체
- 디바이스의 고유한 특성을 내포

임베디드 리눅스 시스템에서 디바이스를 다루는 방법은 디바이스에 대해서 하나의 파일처럼 파일을 통해 액세스가 가능하도록 하고 있다. 하나의 디바이스는 임베디드 리눅스 시스템에서 메이져 번호(major number)와 마이너 번호(minor number)로서 표현된다. 그리고 리눅스에서 디바이스 드라이버의 종류는 크게 문자 디바이스(character device), 블록 디바이스(block device) 그리고 네트워크 디바이스(network device)로 나눈다.

본 연구에 사용된 보드는 BT-UART를 직렬통신 형태로 바꾸어서 모션 제어기의 MAST보드와 통신을 하였고 무선 이더넷 드라이버를 사용하여 호스트 PC와 통신을 하였다.

(1)BT-UART

사용된 보드는 FFUART를 디버깅용으로 사용하기 때문에 BTUART를 모션보드 와의 직렬 통신에 이용 하였다. 직렬포트를 위한 디바이스 드라이버는 리눅스 커널에 가장 기본적으로 포함되어야 할 내용이므로 일반적으로 모듈 프로그램으로 작성하지

는 않는다. 임베디드 시스템 개발을 위해서도 콘솔을 꼭 사용해야 하는 관계로 임베디드 리눅스의 커널을 빌드할 때 직렬포트를 사용가능 상태로 수정하고 커널을 포팅하여 사용하였다.

(2)PCMCIA 무선이더넷

네트워크 디바이스 드라이버와의 인터페이스는 다른 문자나 블록 드라이버와는 다르다. 즉 기존 드라이버가 사용하는 파일 시스템이 아니라 프로토콜 스택을 통해 인터페이스가 이루어진다. 리눅스 에서 네트워크 구조는 다음과 같은 형태를 가지게 된다.

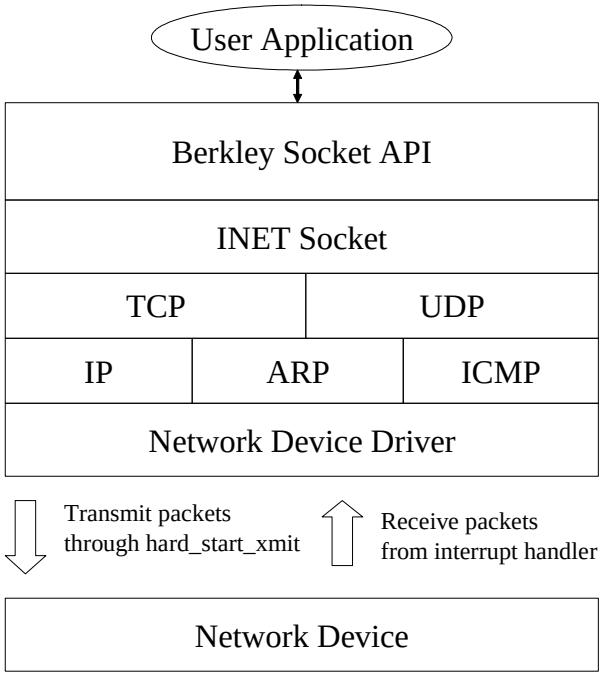


Fig 4.7 Composition of network device drive

리눅스에서 네트워크는 소켓으로 관리한다. 네트워크를 초기화하여 다양한 통신 수단을 사용할 수 있으며 네트워크의 초기화란 통신을 파일 시스템의 체제로 초기화하는 것이고, 하나의 연결이란 하나의 파일을 등록하고 초기화하여 파일 기술자(file descriptor)에 연결한 후 파일 오퍼레션을 규정하는 행위와 같다. 실제 사용 함수는

init_module()과 open()의 과정이 된다.

본 연구에 사용된 무선 이더넷 카드는 16-bit PC CARD interface를 지원한다. Soket0은 PCMCIA slot, socket1은 CF slot로 사용된다. 따라서 사용가능 하도록 커널의 소켓 드라이버 부분을 활성화 시키고 사용 무선 이더넷 카드에 적합하게 커널소스 수정을 해주었다.

4.2.3 애플리 케이션 개발

로봇 구동을 위해서는 PC와의 통신 및 각종 센서 값을 입력 받아서 로봇이 해야 할 행동을 판단하고 각각의 모션보드에 모터의 위치 데이터를 주게 된다. 이는 상위 제어기가 해야 할 일이 많아 짐과 동시에 각각의 입력 처리도 실시간으로 처리를 해야 한다. 그래서 제작된 프로그램은 실시간 및 멀티 기능 구현을 위해 독립된 다중 스레드 및 다중 프로세서를 사용하였다. 각각의 프로세서는 완전히 독립된 동작을 함과 동시에 독립된 메모리 공간을 사용한다. 하지만 프로세서들 간에도 데이터를 주고 받아야 할 경우가 생기게 된다. 그래서 프로세간 통신을 위한 세마포어를 사용하여 센서 데이터 와 PC와의 통신 데이터 및 모션 보드에서 보내는 모터의 상태 값을 서로 주고 받을 수 있도록 하였다.

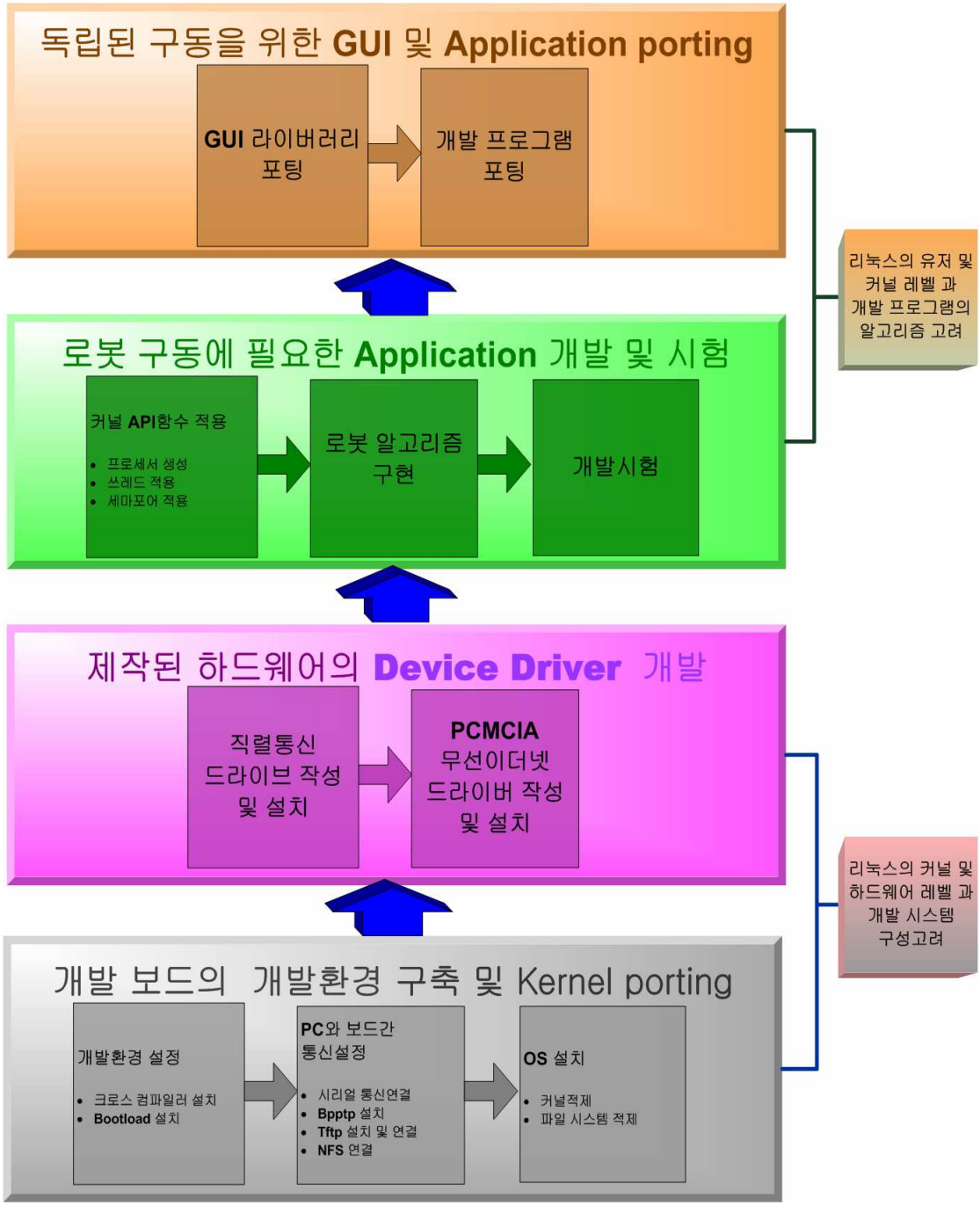


Fig 4.8 Block diagram of the arm-board

(1) 쓰레드(thread)

프로그램 이란 정해진 일들을 실행하는 준비된 명령어 열을 말한다. 운영체제가 커널 자료 구조에 적절한 정보를 추가하고 프로그램 실행을 위한 필요한 자원을 할당할 때, 프로그램은 프로세서가 된다. 프로세스는 주소공간(사용메모리 영역)과 적어도 하나의 쓰레드라고 불리는 제어의 흐름을 가지고 있다.

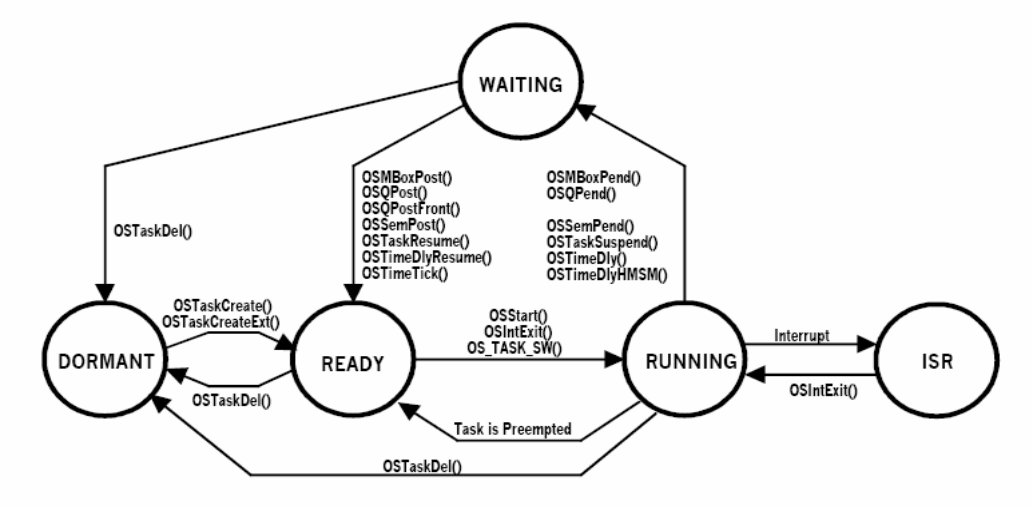


Fig 4.9 State of processor switch

한 프로세서가 여러 소스로부터 입력을 받게 되면 블록 상태는 여러가지 문제점을 야기한다. 왜냐하면, 프로세스는 어떤 소스에서 다음 번의 입력이 들어올지 알 수 없기 때문이다. 본 연구에서 직렬통신과 이더넷통신 그리고 다수의 모션드라이버 운용을 하고 있기 때문에 프로그램이 하나의 입력이 오기 전까지 블록 상태가 되는 것을 방지하기 위해서 여러 쓰레드가 각각의 다른 입력에 대해 병렬처리를 할 수 있도록 구성을 하였다. 다중 쓰레드 프로그래밍은 모든 쓰레드가 같은 메모리 공간을 (그리고 파일 디스크립터와 같은 일부 시스템 자원들을) 공유하는 유닉스 스타일의 다중 프로세싱과는 다르다. 대신에 유닉스의 프로세스와 같이 자신만의 고유 메모리상에 동작한다. 그래서 한 프로세스의 두 쓰레드 사이의 문맥 교환(context switch)는 두 프로세스 사이의 문맥 교환보다 굉장한 수월 하다. 쓰레드 구현 함수의 예는 다음과 같다.

(2) 세마포어

세마포어는 1960년 중반에 Edgser Dijkstra가 발명했다. 이것은 대부분의 멀티태스

킹 커널이 제공하는 프로토콜 메커니즘이다. 세마포어는 코드를 계속 실행하기 위해서 획득해야 하는 열쇠이다. 다른 프로세서가 세마포어를 이미 사용하고 있다면 세마포어를 요구한 태스크는 현재 소유자가 세마포어를 양도할 때까지 대기 상태가 된다. 다시 말하면 각각의 프로세서는 세마포어에 의한 이벤트가 발생하기 전까지는 동작을 하지 않는 것이다. 리눅스 에서는 네임드 세마포어(named semaphore)와 언네임드 세마포어(unnamed semaphore)의 두 가지 종류의 세마포어가 존재한다. 언네임드 세마포어는 한 프로세서의 쓰레드에 의해서만 사용이 가능하고 네임드 세마포어는 다수의 프로세서에서 공유자원을 사용할 수 있다. 본 연구 에서는 다수의 쓰레드만 사용하였기 때문에 언네임드 세마포를 사용하여 PC의 동작 명령이 오면 필요한 동작에 대한 쓰레드를 구동하도록 설계하였다.

4.3 CAN통신

여러 개의 모션보드와 하나의 암 보드간의 효율적인 통신을 위해서는 네트웍 통신이 구현되어야 한다. 하지만 일반적인 직렬통신을 이용하여 네트워크를 구성 할 경우 통신의 신뢰성이 떨어지게 되고 모션보드에 잦은 인터럽터 루틴의 발생으로 인해서 구동 시에 오 동작의 원인으로 작용 할 수가 있다. 그래서 정교한 오류검출 및 처리로 신뢰성이 뛰어나고 소규모 네트웨크 구성에 많이 이용되는CAN(Controller Area Network) 통신을 이용하여 다수의 모션보드와 암보드간의 네트워크 통신을 구현 하였다[11,12].

이의 구성은 두 개의 와이어를 가진 빠른 시리얼 버스를 이용하고 저가의 프로토콜 디바이스를 이용하기 때문에 시스템 구성이 경제적이다. 따라서 원칩 컨트롤러를 이용한 저가의 분산-제어 시스템에 가장 유용한 통신이다. 제작된 시스템의 CAN 통신 구현은 DSP2407포함되어 있는 CAN 컨트롤를 사용하였고 송수신 인터립트를 이용하여CPU 로드를 최소화 하였다. 모체로봇에의 적용 구성은 Fig4.10과 같다.

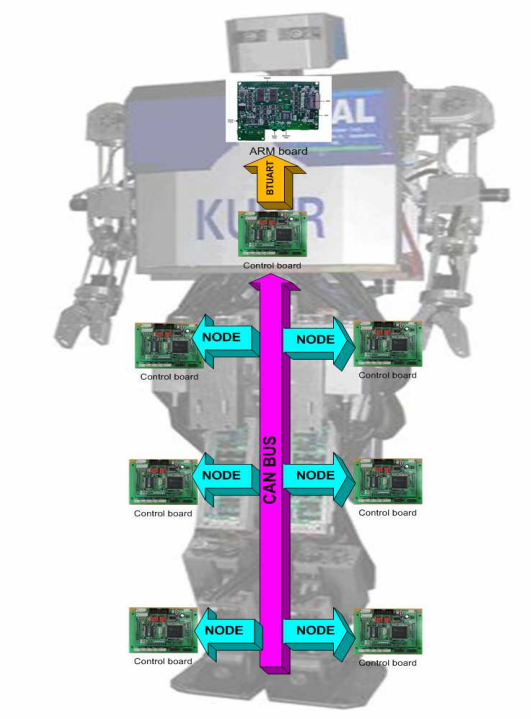


Fig 4.10 Composition of robot network

4.3.1 시스템 개요

CAN통신은 BOSCH(독일)의 자동차 전장 품 개발 업체 (Injector, ABS 등)에 의해서 자동차용 네트워크 통신을 위해서 개발되었다. 현재는 자동차뿐만 아니라 다수의 ECU를 사용하는 분산제어 시스템에 가장 효율적인 통신방식으로 사용되고 있다.

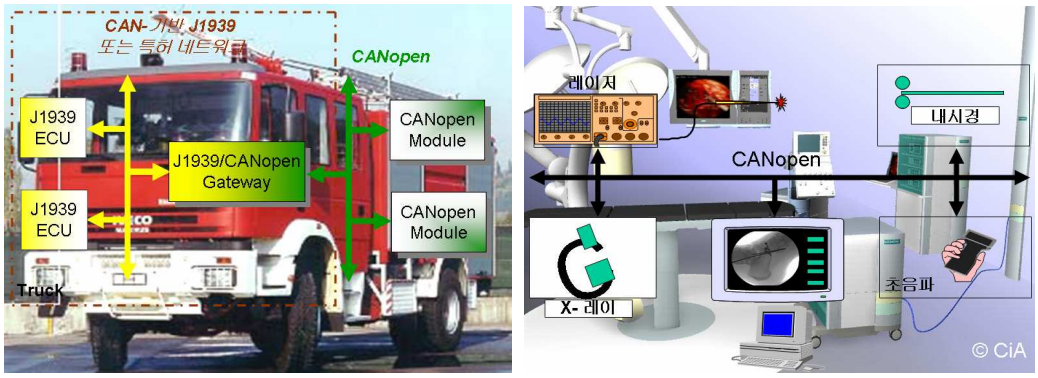


Fig 4.11 CAN communication application

4.3.2 CAN 프로토콜 계층모델

CAN 프로토콜 구조는 OSI Reference Model에 의해 물리 계층 (Physical Layer)와 데이터 링크 계층 (Data Link layer) 으로 나누어 질 수 있고, 데이터 링크 계층은 MAC 서브레이어와 LLC 서브레이어로 나눠진다.

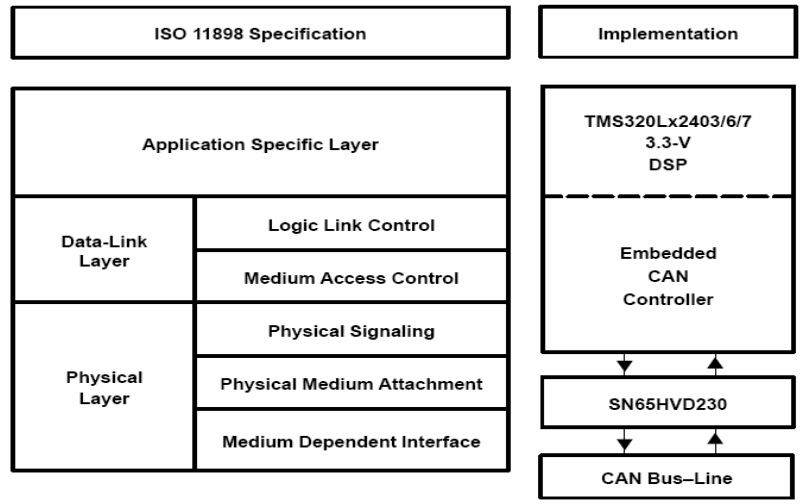


Fig 4.12 The Layered ISO 11898 Standard Architecture

(1)물리계층

물리계층은 신호가 실제로 전송되는 방식을 정의하는 곳으로 비트 타이밍(bit

timing), 비트 인코딩(bit encoding), 동기화(Synchronization) 등의작업들의 성격을 정한다. 버스 라인들은 1Mbaud 의 최고 속도에서 최대 40 m (130 ft)까지 길어질 수 있으며 120 Ohms의 말단처리 레지스터들로 종결된다. 버스 라인들은 보오율을 감소시킬 때 더 길어질 수 있습니다. 표준에 따르면 최대 30개의 노드들이 CAN 드라이버에 연결될 수 있다. 더욱 많은 노드들의 연결을 위해서는, 더욱 강한 드라이버 또는 반복기(repeaters) 가 사용되어야 하고 반사(reflection)를 피하기 위해 버스 라인들로부터 노드에 이르는 연결은 1Mbps 에서 0.3 m (1 ft)를 초과하지 말아야 한다.

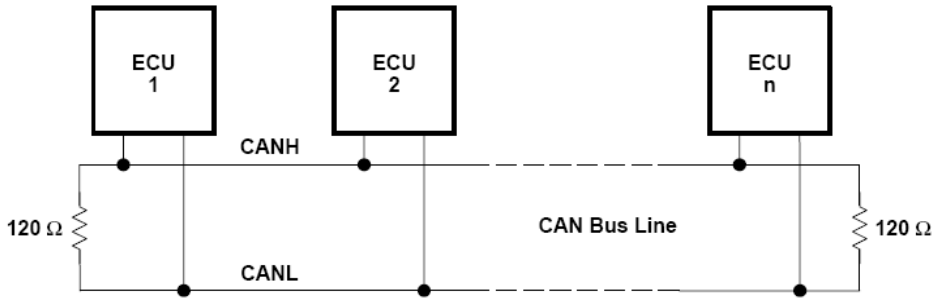


Fig 4.13 Details of a Typical CAN Node

(3) 네트워크 구성

CANopen의 표준적인 물리적 배치는 버스이다 최대 127 개 노드들을 사용할 수 있다. Drop off line (junctions) 들은 1000kbit 또는 그 이하의 통신 속도를 사용가능하고 적어도 한 개의 네트워크 노드는 어떤 최소한의 마스터 기능을 이행해야만 한다.

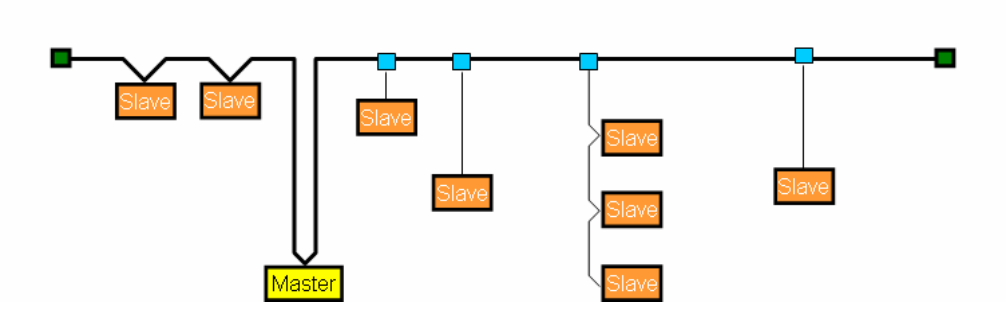


Fig 4.14 Typical CAN Network

(4) CAN 프레임 구조

CAN에서 데이터는 데이터 프레임을 통하여 전송된다. 데이터 프레임은 전송 노드에서 수신 노드로 데이터를 운반하는 프레임으로 최대 8bytes까지의 데이터를 전송할 수 있다. 중재 필드는 11 비트의 식별자와 1 비트의 RTR비트로 구성되는데 노드에서 동시에 메시지를 전송할 때 발생하는 메시지간의 충돌은 식별자 비트를 비교함으로써 해결한다. 제어 필드는 6개의 비트로 구성되는데 IDE 비트는 표준 프레임과 확장 프레임을 구별한다. 0 비트는 예비로 두고, 나머지 4비트는 DLC (Data Length Code) 로서 데이터 필드의 데이터 길이를 알려주는 코드이다. CRC 필드는 15 bits의 CRC와 1 bit의 CRC 구획문자로 구성된다. 수신 노드에서 데이터 프레임을 수신하면 먼저 스터핑 비트를 없애고 난뒤에 CRC를 통하여 SOF에서 데이터 필드까지의 에러 유무를 체크하게 된다. ACK 필드는 1 bit의 ACK 슬롯과 1 bit의 ACK 구획문자로 구성되는데,메시지를 올바르게 받은 수신 노드가 ACK 필드를 받는 바로 그 순간에ACK 슬롯의 비트 값을 셋팅하게 된다. 마지막으로 EOF (End of Frame) 는 7 bits로 구성되고, 메시지의 전송이 끝났음을 알린다.

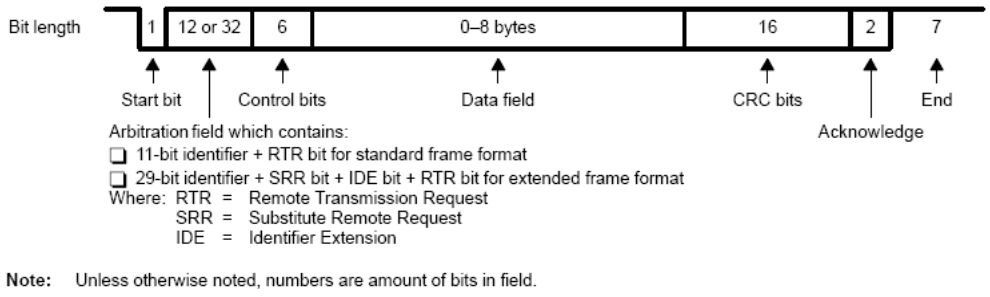


Fig 4.15 Standard CAN message frame

4.3.3 DSP2007을 이용한 CAN통신 구현

DSP2407에는 CPU 로드를 최소화 하기 위한 FULL CAN 컨트롤러가 탑재되어 있다. 그래서 다수의 네트워크를 구성하여도 실제 CPU에 걸리는 로드는 그리 많지가 않아 다른 속도 및 위치제어에 대한 영향을 최소화 하였다.

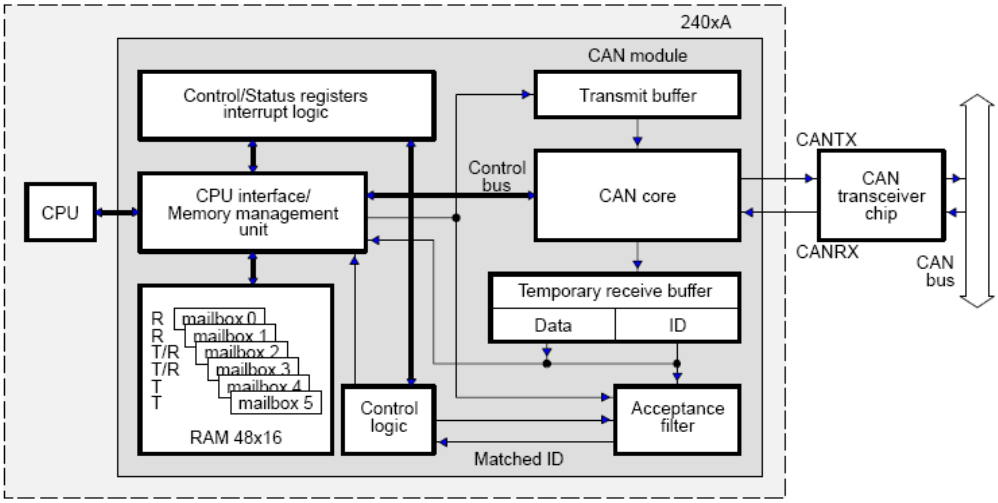


Fig 4.16 CAN Module Block Diagram

5. 실험 및 고찰

이족보행로봇의 제어용으로 개발한 ARM 보드의 제어실험을 수행하였다. 제어시스템의 성능을 확인하기 위해서 우선적으로 서보모터의 경로제어시험를 하였다. 시험은 기본적인 위치제어와 3장에서 설계한 사다리꼴 모양의 속도 궤적을 사용하여 시험하였다. 또 한 로봇전체 시스템의 성능 및 효율을 확인하기 위하여 로봇의 보행 실험을 하였다.

5.1 모터의 경로제어 시험

축에 적용되는 모터를 다음의 조건에서 실험을 행하였다.
(1) 제어주기 = 500Hz, 기어비 = 1:51, Kp = 1, Kd = 6, 목표위치 = 3000 펄스

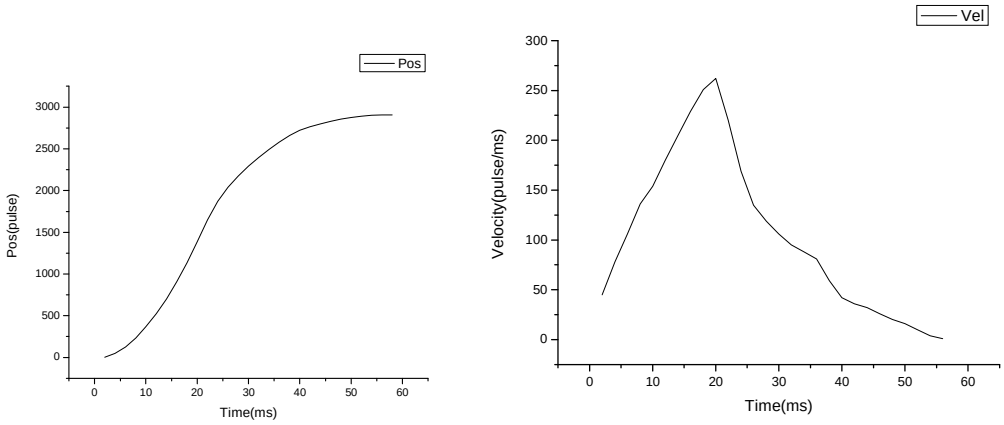


Fig 5.1 Position control result

(2) 제어주기 = 1kHz, 기어비 = 1:51, Kp = 1, Kd = 6, 목표위치 = 10000 펄스

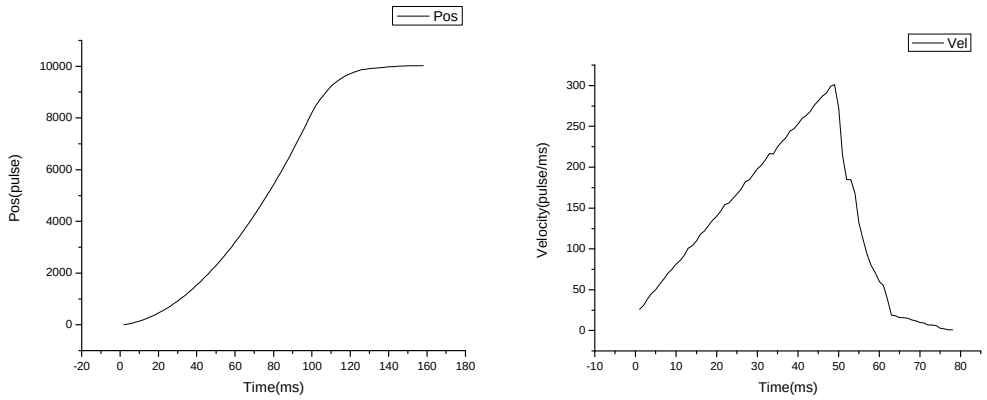


Fig 5.2 Position control result

사다리꼴 모양의 속도 프로파일에 따른 모터의 위치, 속도 그래프이다.

(3) 제어주기 = 1kHz, 기어비 = 1:51, Kp = 1, Kd = 6,
목표위치 = 45000 펄스, 평균속도 = 90펄스/ms

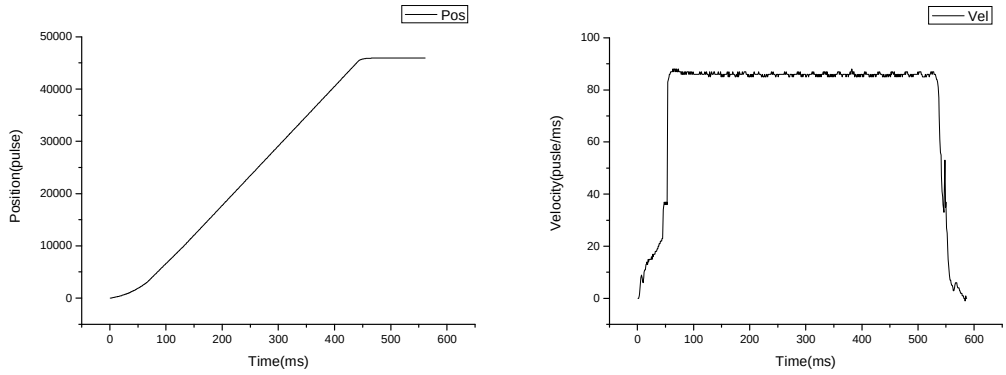


Fig 5.3 Position and velocity control result

(4) 제어주기 = 1kHz, 기어비 = 1:51, Kp = 1, Kd = 6,
목표위치 = 45000 펄스 평균속도 = 45펄스/ms

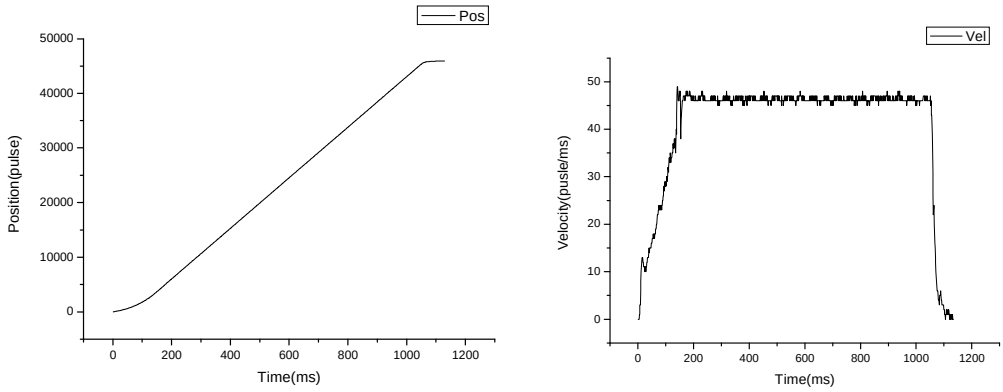
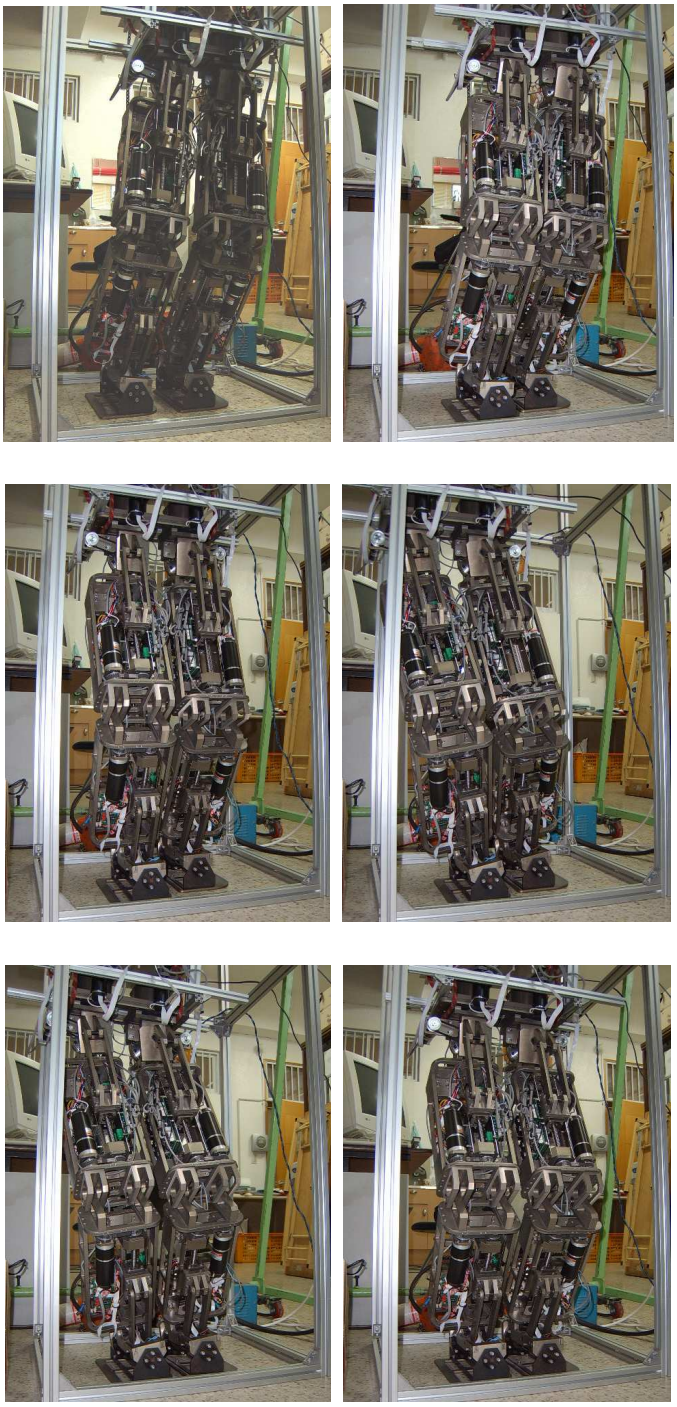


Fig 5.4 Position and velocity control result

결과 짧은 구간을 이동할 때 500Hz 보다는 1kHz 로 제어주기가 짧아 졌을 때, 보다 안정한 구동을 하였다. 위치 그래프는 모든 실험에서 비슷한 모양을 가지나 속도 그래프는 구동 프로파일에 따라 많이 달라짐을 볼 수 있다. 구동 프로파일을 사용한 실험에서 나타난 속도 그래프를 보면 속도의 진동이 많이 생겨 보다 섬세한 이득튜닝과 궤적상의 점들의 수를 결정하는 것이 필요하다.

5.2 로봇의 보행실험

적인 로봇제어 시스템의 동작 성능과 효률등을 알아보기 위해서 이족보행로봇으로 보행 실험을 하였다. 보행로봇에는 12개의 모터와 6개의 2축 제어기가 구성되어 있다. 보행실험에 있어 관절각들과 로봇구동의 명령은 ARM보드에 미리 경로를 입력하여 Teaching에 의해 동작하도록 하였다. 로봇의 원점동작에 대한 명령은 센서의 입력을 받아 자동으로 이루어 지도록 프로그램을 되어 있다. 실험결과의 데이터는 로봇 보행에 대한 사진과 동영상 자료로 획득하였다. Pic 5.1 는 로봇의 단계적 보행에 대한 사진이다.



Pic 5.1 Motion control of robot

모터 실험결과를 Pic5.1과 같이 12자유도 보행로봇 다리에 적용하여 시험한 결과 연속적이고 부드러운 동작을 확인할 수 있었다.

6. 결 론

본 논문에서는 로봇의 움직임을 제어하기 위한 리눅스 ARM보드와 One-chip 마이크로프로세서를 이용한 소형화된 임베디드 제어시스템을 개발하였다. 기존의 범용 PC 기반의 제어기와는 달리 범용적 동작을 위해서가 아니라 자체 제작된 로봇의 효율적 구동을 위해 제작하였다. 그래서 ARM보드에 포팅된 리눅스의 커널과 시스템의 전체적인 구성을 로봇에 맞게 특화 시켜 제작을 하였고 로봇 구동을 위한 설치 및 설정을 매우 간편하게 할 수 있도록 하였다. 따라서 상용제어기를 사용할 때 가장 문제점인 내부 구동 프로그램 자체를 상황에 따라 신속한 변경이 가능하고 직접 제작을 하였기 때문에 구동상의 문제점이 발생하면 빠른 복구 및 보수가 가능하도록 하였다. 또한 로봇 구동에 필요한 모든 축제어기가 하나의 ARM보드에 부착이 가능하게 되어 시스템이 로봇 내부에 모두 적제가 되도록 하였고 임베디드 ARM보드의 특성상 적은 전력으로도 사용이 가능하기 때문에 부가 전원 장치의 규모도 줄어 들게 된다.

제어 시스템은 12개의 모터를 PID제어 알고리즘을 구성하여 동시에 궤환제어를 할 수 있도록 하였다. DSP One-chip 마이크로프로세서 기반의 모듈형 축제어기와 12개의 모터를 동시에 제어할 수 있는 ARM chip 기반의 상위제어기와, 이 시스템의 동작 명령과 각 관절의 상태를 무선통신을 통하여 On-line으로 PC에서 모니터링할 수 있는 임베디드 기반의 주제어 시스템을 개발하였다. 개발 시스템의 축제어기의 경로 프로파일을 구성하여 모터 제어실험을 한 결과 경로 제어는 양호한 결과를 얻었으나 속도제어에서는 약간의 개선 여지가 있다. 이렇게 제작된 모션보드와 상위제어 보드인 ARM보드를 이용하여 CAN통신으로 모든 제어기들이 네트워크를 구성하도록 하여 로봇의 전체적인 제어 시스템을 제작 하였다. 이는 새로운 형태의 이족보행 로봇에 최적화된 시스템을 구성하였고 자체적인 제어 시스템 제작에 따른 기술확보 및 로봇보행의 효률성도 증대 되었다. 향후, 모션 제어기의 속도성능 개선 및 사용상의 유연성, ARM보드를 이용한 로봇의 지능 알고리즘 개발 등의 과제를 남겨놓고 있다.

참고문헌

[1] http://asimo.honda.com/

[2] J.H Kim, S.W Park, I.W Park, and J. H. Oh, “Development of a Humanoid Biped Walking Robot Platform KHR-1 - Initial Design and Its Performance Evaluation,” in Proceedings of 3rd IARP Int. Workshop on Humanoid and Human Friendly Robotics, pp. 14-21, Tsukuba, Japan, Dec. 2003.

[3]TMS320LF2407A User Guide, Texas Instrument. Inc.

[4] TMS320C24계열을 이용한 DSP 하드웨어 설계, SyncWorks

[5]http://www.samsung.com/Products/Semiconductor/SystemLSI/DevelopmentTools/32bitscores01.htm

[6] Robot Dynamics Control, Spong Vidyasagar

[7] Embedded Systems Building Block, Jean J. Labrosse

[8] MicroC/OS-II Real Time Kernel, Jean J. Labrosse 11-12, 2002

[9]임베디드 리눅스 프로그래밍, 이연조

[10]LINUX DEVICE DRIVERS, Alessandro Rubini

[11]CAN System Engineering, Wolfhard Lawrenz

[12]Data Communication, Computer Networks and Open Systems, Fred Halsall

부록

1 쓰레드

```
//스레드 관련
void *thread_put_slaveID(void *arg);
void *thread_get_slaveID(void *arg);
void *thread_run_kubir(void *arg);
char *message="end";

        //스레드 생성

        put_thread_res                                                  =
pthread_create(&uart1_put_thread,NULL,thread_put_slaveID,(void *)message);
        get_thread_res                                                  =
pthread_create(&uart1_get_thread,NULL,thread_get_slaveID,(void *)message);
        run_kubir_res                                                   =
pthread_create(&run_kubir_thread,NULL,thread_run_kubir,(void *)message);
        if((put_thread_res|get_thread_res|run_kubir_res) !=0)
        {
                perror("Thread creation failed₩n");
                exit(0);
        }

//스레드 실행 함수
void * thread_put_slaveID (void *arg)
{
        char Buff1[256];
        int RxCount1;
        int i;

        while(!flag)
        {
```

```
                // 한 문자를 수신한다.
                RxCount1 = read( handle1, Buff1, 1 );

                if( RxCount1 < 0)
                {
                        printf( "Read Error₩n₩r" );
                        break;
                }
                write( handle0, Buff1, RxCount1 );
        }
        pthread_exit(0);
}
```

1 세마포어

```
//세마포어 관련
int MoveMotor_res, EndPos_res, OnHome_res;
sem_t MoveMotor_sem, EndPos_sem, OnHome_sem;

//세마포어 초기화(사용전 반드시 해야한다.)
        MoveMotor_res = sem_init(&MoveMotor_sem,0,0);  //언네임 세마포어 스레드들에 의해서 사용가능 자식 프로세서는 상속불가.
        EndPos_res = sem_init(&EndPos_sem,0,0);  //언네임 세마포어 스레드들에 의해서 사용가능 자식 프로세서는 상속불가.
        OnHome_res = sem_init(&OnHome_sem,0,0);
        if((MoveMotor_res|EndPos_res|OnHome_res) < 0)
          {
          perror("Semaphore creation failed₩n");
          exit(0);
          }
//사용함수
sem_post(&EndPos_sem);
```

```
sem_wait(&EndPos_sem);
```

1 시리얼

```
//시리얼 헨들
int handle0,handle1;
        // 화일을 연다.
handle = open( "/dev/ttyS1", O_RDWR | O_NOCTTY );

if( handle < 0 )
{
        //화일 열기 실패
        printf( "Serial Open Fail [/dev/ttyS00]₩r₩n " );
        exit(0);
}

tcgetattr( handle, &oldtio ); // 현재 설정을 oldtio에 저장
memset( &newtio, 0, sizeof(newtio) );
newtio.c_cflag = B9600 | CS8 | CLOCAL | CREAD ;
newtio.c_iflag = IGNPAR;
newtio.c_oflag = 0;

//set input mode (non-canonical, no echo,.....)
newtio.c_lflag = 0;
newtio.c_cc[VTIME] = 30; // time-out 값으로 사용된다. time-out 값은 TIME*0.1
초 이다.
newtio.c_cc[VMIN] = 0; // MIN은 read가 리턴되기 위한 최소한의 문자 개수
tcflush( handle, TCIFLUSH );
tcsetattr( handle, TCSANOW, &newtio );
```

1 무선 이더넷

```
//TCP/IP 소켓변수
int socket_fd;

//TCP/IP 구조체 선언
char data_string[100];
int string_length, NetRxCount;
struct sockaddr_in server_address;

// 클라이언트 소켓 생성
if((socket_fd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
{
        perror("sorket error");
        exit(1);
}

//통신을 위해 서버의 IP와 코트 번호 등을 정의
bzero((char *)&server_address, sizeof(server_address));
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = inet_addr("192.168.1.200");    //서버 IP
server_address.sin_port = htons(2200);

//서버와 연결
if(connect(socket_fd,(struct sockaddr *) &server_address,sizeof(server_address))
< 0)
{
        perror("connect error");
        exit(1);
}  // end if
```

1 CAN 통신

```
/*********************************************
                CAN interrupt setting
*********************************************/
void CAN_INTE_Init( void )
void CAN_INTE1_Init( void )
/*************************************************************/
/* Initializtion and Configuration for CAN                   */
/*************************************************************/
void CAN_Init( void )

/*************************************************************/
/* ID setting for MailBoxes                                  */
/*************************************************************/
void MailBox0_IDSet( int MBX0_ID )      /* MBX0  은 only 수신용 */

void MailBox1_IDSet( int MBX1_ID )      /* MBX0  은 only 수신용 */

void MailBox5_IDSet( int MBX5_ID )  /* MBX5 은 only  송신용 */

void MailBox4_IDSet( int MBX4_ID )  /* MBX5 은 only  송신용 */

/*************************************************************/
/* MailBox Setting for Transmission-Receive mode             */
/*************************************************************/
void Trans_Setting( int MBX2, int MBX3, int MBX4, int MBX5)       /* MailBoxes
Setting for Transmission MBOX4 5 */

void Receive_Setting( int MBX0, int MBX1, int MBX2, int MBX3)          /*
MailBoxes Setting for Receive */
```

```
/*************************************************************/
/* Transmission Functions                                    */
/*************************************************************/
void Kubir_CAN_Trans_5(int CAN_mID , char MOTOR_ID, long POS, char STATE )

void RSA_CAN_Inte_Trans_4(int CAN_mID , int STATE, int data1, int data2, int data3 )
/*************************************************************/
/* Receive Functions                                         */
/*************************************************************/
char *Kubir_CAN_Receive( void )
```

지금까지 대학원 생활을 함께한 실험실 여러분과 교수님 그리고 부모님께 감사 드리고 앞으로 더욱 멋진 모습 보여 드리겠습니다!