공학석사 학위논문

Stereo Vision을 이용한 휴머노이드 로봇의 팔 제어 시스템 연구

Development of Stereo Vision Assisted Control System for Humanoid Robot

지도교수 유 삼 상

2008년 2월

한국해양대학교 대학원

기 계 공 학 과

김 태 형

本 論文을 金泰亨의 工學碩士 學位論文으로 認准함.

위원장 최 형 식 (인)

위 원 유 삼 상 (인)

위 원 정 재 현 (인)

2008년 2월

한국해양대학교 대학원

목 차

Abstract
기호설명
그림목차
표목차

제 1 장 서 론 1

제 2 장 로봇의 전체시스템 구성 3
2.1 로봇의 전체시스템 구성 3
2.2 KUBIR III의 머리 기구부 구성 8
2.3 KUBIR III의 허리 기구부 구성 10
2.4 KUBIR III의 팔 기구부 구성 11
2.5 Four-bar-link의 구성 15
2.6 KUBIR III의 머리부의 기구학 해석 16
2.7 KUBIR III의 팔의 기구학 해석 19
2.8 Four-bar-link의 기구학적 해석 25
2.9 역기구학 26

제 3 장 로봇의 제어부 구성 30
3.1 Embedded System의 개요 30
3.2 Embedded System의 개발 환경 31
3.3 KUBIR III의 전체 제어부 구성 34
3.4 KUBIR III의 주 제어기 구조 37

제 4 장 로봇 보행의 3D 시뮬레이션 41
4.1 3D 시뮬레이터 개발 환경 41
4.2 3D 시뮬레이터 변환 과정 42
4.3 3D 시뮬레이터 43
4.4 3D 시뮬레이터 검증 44

제 5 장 Stereo Vision System을 이용한 영상처리 48
5.1 Image Frame Grabber의 구조 및 구성 48
5.2 KUBIR III의 CCD 카메라의 구조와 기능 50
5.3 Stereo Vision에서의 거리측정 52

제 6 장 영상처리를 이용한 물체 인식 실험 55
6.1 비주얼 서보잉 55
6.2 거리 측정 실험 57

제 7 장 결 론 62

참고문헌 63

부 록 65

Development of Stereo Vision Assisted Control System for Humanoid Robot

Kim Tae Hyung

Department of Mechanical Engineering
Graduate School, Korea Maritime University

Abstract

We developed a new type of human-sized biped walking robot(BWR), KUBIR III, driven by the closed-chain type of joint actuator. The lower part of the robot is composed of twelve joints; body is 1 d.o.f., head is 2 d.o.f., and arms are 12 d.o.f. In all, 27 degree-of-freedom biped robot has been developed in this study. A unique actuation structure for the leg joints of the BWR is designed using the four-bar-link mechanism driven by the ball screw, which has a high strength and high gear ratio. The weight of the robot is 70kg with 170cm tall, and it is designed to be capable of supporting 120kg.

CCD camera is attached to the developed robot, each of eyes to be capable of supporting stereo vision system. We have provided the closed-form solution of the inverse kinematics for the end-point of two arms with 12 d.o.f. of the robot in order to recognize and grab the object in the workspace. If the left and right cameras successfully find the same target object, then the implemented active vision system with two cameras focuses on a landmark and can detect the depth and the direction information.

Also, we have developed 3D graphic simulator before performing experiments for prevention of accident. As a result of experiments, the gripper of the arm of robot can successfully catch the object in workspace.

기 호 설 명

$l_1,\ l_2,\ l_3$	4절링크의 링크 길이
A_i	각 관절의 동차 변환 행렬
T_i^j	기구의 동차 변환 행렬
a_i	x_i 축을 따라서 o 에서 x_i 축과 z_{i-1} 축의 교점까지 거리
d_i	z_{i-1} 축을 따라서 o_{i-1} 에서 x_i 축과 z_{i-1} 축의 교점까지 거리
α_i	x_i 축을 중심으로 측정된 z_{i-1} 과 z_i 사이의 각도
θ_i^*	z_{i-1} 축을 중심으로 측정된 x_{i-1} 과 x_i 사이의 각도
$p_{i,j}$	기구의 j의 말단장치 위치벡터
B	두 카메라 렌즈의 중심과 중심을 연결한 선분 거리
W	물체의 한 점
C	$\cos\theta$
S	$\sin\theta$

그 림 목 차

Fig. 2.1 A front of specification of KUBIR III 4
Fig. 2.2 A back of specification of KUBIR III 5
Fig. 2.3 The head of KUBIR III 8
Fig. 2.4 Rotation range of the head 9
Fig. 2.5 3D model of the waist joint actuator 10
Fig. 2.6 Gravity compensator of the waist joint 10
Fig. 2.7 Rotation range of the shoulder pitch joint 11
Fig. 2.8 Shoulder joint 11
Fig. 2.9 Rotation range of the elbow pitch joint 12
Fig. 2.10 Elbow joint 12
Fig. 2.11 Rotation range of the wrist pitch and roll joint 13
Fig. 2.12 Wrist joint 13
Fig. 2.13 Structure of four-bar-link 15
Fig. 2.14 D-H coordinate for the head 16
Fig. 2.15 D-H coordinate for the arm 19
Fig. 2.16 Four-bar link modeling 25
Fig. 2.17 Spherical configuration manipulator 28
Fig. 3.1 Embedded system application 31
Fig. 3.2 Embedded system development environment 32
Fig. 3.3 Motor driver 35
Fig. 3.4 Motion board 35
Fig. 3.5 Motor controller 35
Fig. 3.6 Total control system of KUBIR III 36
Fig. 3.7 Board diagram of PCM-9582 38
Fig. 3.8 Main controller of KUBIR III 38
Fig. 3.9 Loaded hard disk and image grabber on embedded computer 39
Fig. 3.10 Loaded hard disk and image grabber on embedded computer 39
Fig. 3.11 Power supply on the back of KUBIR III 40
Fig. 4.1 Public application program of OpenGL 41
Fig. 4.2 File conversion process for 3D simulator 42
Fig. 4.3 Design of simulator of KUBIR III 43
Fig. 4.4 Walking simulator of KUBIR III 44
Fig. 4.5 Simulator of catch the target 45
Fig. 4.6 Simulator of catch the target 45
Fig. 4.7 Verification of simulator at 0 degree 46
Fig. 4.8 Verification of simulator at 30 degree 46

Fig. 4.9 Comparison between 3D simulator and experimental test 4 7
Fig. 5.1 Internal structure of the image grabber 'Morphis' 4 9
Fig. 5.2 Photo of stereo vision system..... 5 0
Fig. 5.3 Photo of CCD camera..... 5 1
Fig. 5.4 Model of stereo vision processing 5 2
Fig. 5.5 Section of stereo vision processing model..... 5 3
Fig. 6.1 Eye to hand in the visual servoing..... 5 5
Fig. 6.2 Eye in hand in the visual servoing 5 6
Fig. 6.3 Image processing of the vision system 5 8
Fig. 6.4 Experiment of distance measurement 5 9
Fig. 6.5 Result of distance using program..... 6 0
Fig. 6.6 Distance between camera and object 6 0
Fig. 6.7 Result of computing distance using the stereo vision system 6 1

표 목 차

Table 2.1 Specification of KUBIR III
Table 2.2 Degree of freedom of KUBIR III
Table 2.3 Specification of the joint actuator for head part
Table 2.4 Specification of the joint actuator for arm part
Table 2.5 Link parameter of the head of KUBIR-III
Table 2.6 Link parameter of the arm of KUBIR-III
Table 5.1 Specification for TCC-3232H

제 1 장 서 론

로봇에 대한 정확한 정의는 없지만 일반적으로 “인격을 갖고 있지 않은 기계로서 사람에 의해 프로그램 된 후 명령에 따라 스스로 동작하는 기계” 라고 정의 되어있다. 로봇의 어원인 ‘Robota’ 라는 단어는 체코어로 ‘힘든 노동, 노예’를 의미하는 것으로 로봇을 ‘힘든 노동을 하는 노예 같은 존재’로 보았음을 알 수 있다.

‘로봇’ 이라는 단어가 기계의 이름으로 쓰이기 시작한 것은 1920년대부터였지만, 스스로 움직이는 것이 가능한 로봇은 그보다 훨씬 전 이였다. 1900년대에 와서 유럽의 정밀 기계 기술의 발달은 거의 완숙의 경지에 이르렀다. 초기의 로봇은 모두 복잡한 톱니바퀴와 지레로 움직였다. 전자 제어 로봇은 컴퓨터를 이용한 제어 계측 기술의 개발과 마이크로 프로세서를 이용한 집적 회로의 설계가 가능해지면서 성능이 더욱 뛰어나면서도 정교한 작업을 할 수 있는 로봇이 생산되기 시작했다. 특히 정교한 작업을 요구하는 수술이나 인간이 하기 위험한 작업 등이 제어 계측 로봇의 역할이 됨에 따라 인간이 로봇에 의존하는 정도는 더욱 증가하고 있다.

1940년대 후반에 Oak Ridge와 Argonne National Laboratory에서 방사성 물질을 처리하기 위해 Master-Slave 형의 로봇을 개발한 것에 기초하여 1950년대 중반엔 George C. Devol는 프로그래밍에 의하여 운전되는 매니퓰레이터로 발전 시켰다. 1960년대에 ‘Unimates’ 라는 산업용 로봇이 처음으로 만들어지면서 로봇시대가 열리게 되었고 1980년대부터 고도의 기능을 가진 로봇이 실용화되기에 이르렀다[1].

현재까지 세계 각국에서 많은 로봇이 개발되면서 로봇의 활용분야 또한 광범위해지고 사람들에게 친숙하게 다가오고 있다. 위험한 산업현장에서의 작업은 물론, 우주 행성 관찰용 로봇, 심해 탐사용 로봇, 군사용 로봇, 안내 로봇, 엔터테인먼트용 로봇, 의료 검진 수술용 로봇 등 사람들이 할 수 있고, 못하는 범위까지 이제는 인간의 삶에서 없어서는 안 되는 문명의 발전이라고 할 수 있다. 본 연구에서 개발하고 있는 로봇은 사람의 힘든 일을 도와주는 친 인간적인 형태이다. 인간에 친숙한 형태의 로봇을 만들기 위하여 인간의 모습과 유사하게 설계하였고, 관절의 움직임 또한 동일하도록 고안하였다.

현재 이족보행로봇으로는 Honda의 ‘ASIMO’, Sony의 ‘QURIO’, 한국의 ‘HUBO’, ‘MARU’ 그리고 본 논문에서 다루는 한국해양대학교의 ‘KUBIR’이 개발되고 있다[2]. 한국해양대학교의 이족보행로봇 ‘KUBIR’은 다른 이족보행로봇과는 다른 메커니즘으로 무거운 물건을 들고 이송할 수 있도록 제작되어 보다 인간생활에서의 편리함을 더 해 주도록 개발하였다[3].

본 논문에서는 기존의 KUBIR I과 KUBIR II의 기능을 보완한 KUBIR III에 대해 기재하였다. KUBIR III 역시 기존의 KUBIR과 마찬가지로 27 자유도를 가지며 모터의 위치 역시 같다. 하지만, 기존의 KUBIR에서 문제가 되었던 보행의 안정성을 위해 다리의 기구부의 형태가 바뀌었고, 무거운 물건을 들고 이송 할 수 있도록 팔의 기구부 형태가 보완되었다. 또한, 휴머노이드 로봇에 필수적인 카메라를 장착하였다. 본 논문에서 사용된 카메라는 소형 CCD 카메라로 물체의 거리 측정을 할 수 있게 stereo vision system을 구현하였으며[4], 기구학 해석을 통하여 말단 장치 좌표를 구하였고[5], OpenCV로 영상처리된 화면에서 로봇의 팔이 움직일 수 있도록 visual servoing을 구현하였다[6]. 그리고, 기존의 KUBIR과는 달리 embedded mini computer를 로봇의 등에 부착하여 stereo vision system과 각 각의 모터 제어, ethernet 통신, serial 통신 등 주 제어 시스템으로 사용하였다.

로봇의 하드웨어적 손상과 실험에서의 소요 시간을 단축하기 위해서 OpenGL을 이용한 microsoft사의 visual .net 기반에서 3D 시뮬레이터를 개발 하였고, 이를 통하여 로봇의 동작을 예측하고, 실제 로봇의 개발 기간을 단축 할 수 있게 되었다.

본 논문의 2장에서는 로봇의 전체 시스템 구성과 로봇의 기구부 구성, 로봇의 기구학적 해석을 논하였고, 3장은 로봇의 제어부 구성을 설명하였다. 4장에서는 보행 로봇의 3D 시뮬레이션과 그 결과를, 5장에서는 stereo vision system을 이용하여 영상처리 구현 내용을 기재하였다. 6장에서는 영상처리를 이용한 물체 인식 거리 측정 실험을 기재하였고, 마지막으로 7장에서는 위 논문의 결론과 향후 과제에 대해 논하였다.

제 2 장 로봇의 전체시스템 구성

2.1 로봇의 전체시스템 구성

개발된 휴머노이드 로봇은 외장케이스를 제외하고 전체 키 160cm, 중량 72kg이다. 다리 105cm, 상체에서 머리 55cm, 팔 길이 80cm 등 이다. 모터의 개수는 한 다리당 6개, 한 팔당 6개, 머리 2개, 허리 하나를 가지며 총 27개의 모터를 사용하고 있다. 로봇의 머리에는 눈에 해당하는 CCD 카메라 2대가 장착되어 있으며 머리는 yaw와 pitch운동이 가능하다. 팔의 자유도는 6 자유도(degree of freedom)를 가지며 어깨에서부터 yaw(45도)-pitch-pitch-pitch-roll의 구조로 되어 있어 사람과 유사한 움직임을 할 수 있도록 되어있다. 팔의 끝 부분은 사람의 손 역할을 하는 그리퍼(gripper)가 장착되어 있어 물건을 잡을 수 있는 구조로 되어 있다. 다리의 자유도는 6 자유도 이며 yaw-roll-pitch-pitch-pitch-roll의 구조로 되어 있어 팔과 마찬가지로 인간과 유사한 보행을 할 수 있는 구조로 설계되어 있으며 또한 팔 부분과 다리 부분을 연결하는 허리 관절은 로봇의 보행 시 무게 중심을 조절하는 역할을 하게 된다.

현재 KUBIR III의 앞서 KUBIR I과 KUBIR II가 제작되어 실험을 하였는데 이전의 실험에서의 문제점인 관절 작동의 저속과 사절 링크의 복잡한 구조, 운동 반경의 제한 또한 무거운 중량을 해결하였다. KUBIR II에서부터 높은 구동기(actuator)를 장착하였고, 사절 링크를 모듈화 하였다. 또한, 기구학적 해석을 이용한 설계로 무게를 줄임으로도 튼튼한 내구성을 가지고 있으며, 기구부는 알루미늄을 열처리하여 가공하였다. 외형 케이스 또한 알루미늄 재질이며, 가볍고 튼튼하게 가공하였다.

로봇의 안정된 보행을 위하여 허리 부위에는 중력보상기(gravity compensator)를 장착하여 보다 안정된 보행을 가능하게 하였고, 11개의 하모닉 드라이브(harmonic driver)를 장착하여 모터의 효율을 높이고, 정밀한 제어를 가능하게 되었다. 또한, 사절링크구조(four-bar-link)를 12곳에 적용하였다. 사절링크는 속도가 느려지는 단점을 가지고 있지만 보다 높은 토크를 발생 함으로 무거운 물건을 들 수가 있으며, 고 부하가 예상되는 부분에서도 안정된 움직임을 가질 수 있다.

아래의 그림 Fig. 2.1은 KUBIR III의 전체 구성을 보여주고 있다.

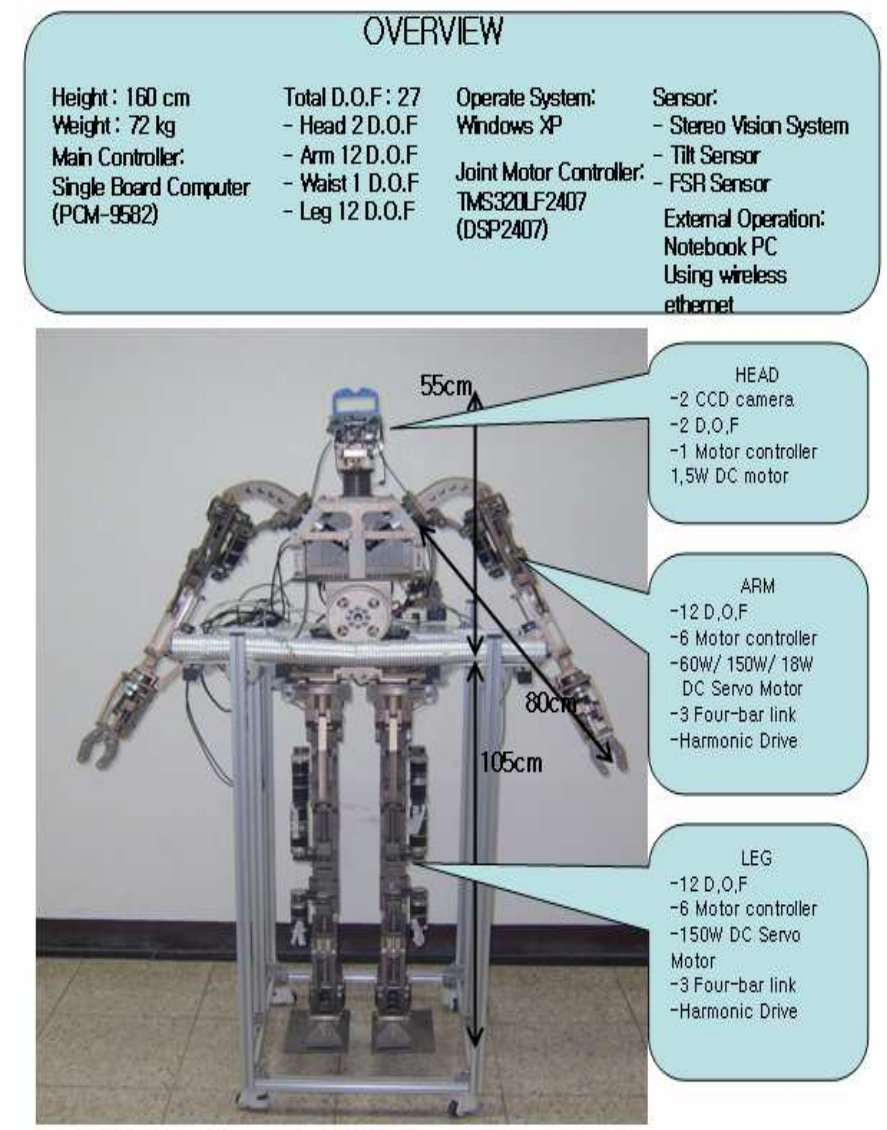


Fig. 2.1 A front of specification of KUBIR III

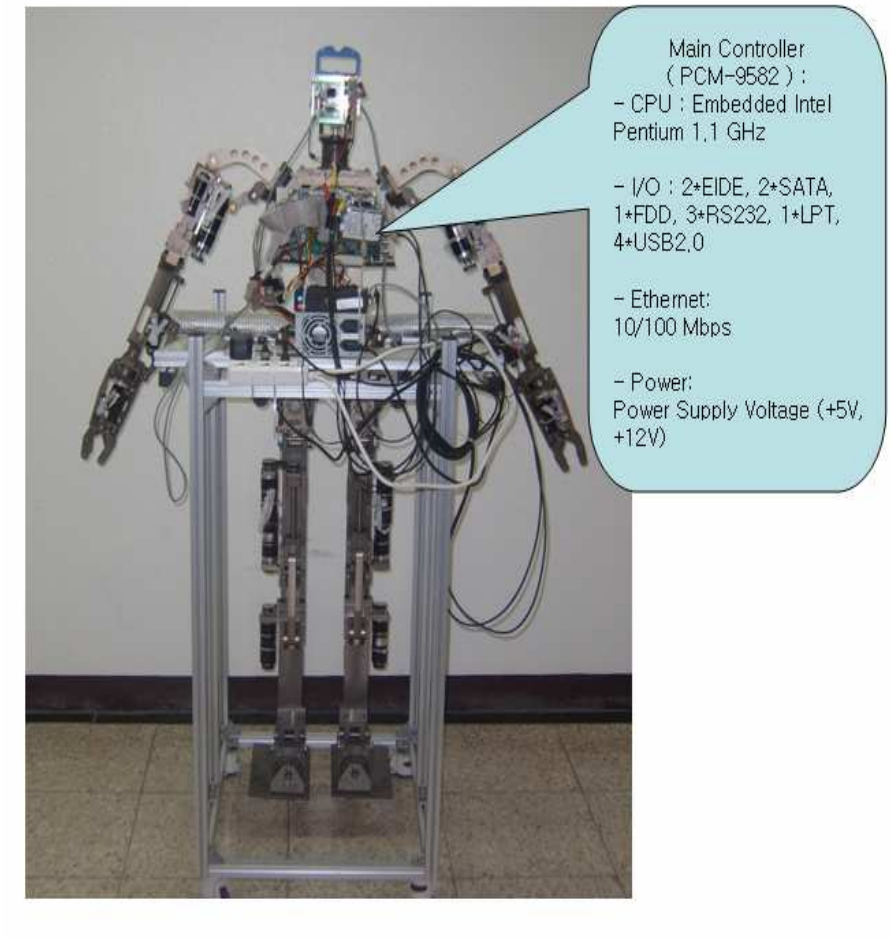


Fig. 2.2 A back of specification of KUBIR III

아래의 Table 2.1과 2.2는 KUBIR III의 각 부분의 제원을 나타내었다.

Table 2.1 Specification of KUBIR III

Height	160 [㎝]	
Weight	72 [㎏]	
Actuator	Head	DC motor
	Arm	DC Servo motor + Harmonic speed reducer + Ball screw
	Waist	DC Servo motor + Harmonic speed reducer + Ball screw + Gravity compensator
	Leg	DC Servo motor + Harmonic speed reducer + Ball screw
Control unit	Main controller	Embedded Single-board Computer (PCM-9582)
	Joint controller	TMS320LF2407 + Motor Driver + Motion board
Power capacity	24V / 50AH	

Sensory device	2 CCD camera / Image grabber Tilt sensor, FSR sensor Magnetic sensor, Proximity sensor
Operation device	Notebook PC with wireless LAN

Table 2.2 Degree of freedom of KUBIR III

Head	2 DOF	
Arm	Left	6 DOF (shoulder 2 + elbow 1 + wrist 1 + hand 1)
	Right	6 DOF (shoulder 2 + elbow 1 + wrist 1 + hand 1)
Waist	1 DOF	
Leg	Left	6 DOF (pelvis 2 + thigh 2 + knee 1 + ankle 1)
	Right	6 DOF (pelvis 2 + thigh 2 + knee 1 + ankle 1)
Total	27 DOF	

2.2 KUBIR III의 머리 기구부 구성

KUBIR III의 머리는 1.1kgf로 Fig.2.3 과 같이 2자유도를 가지고 있으며, 인간의 눈과 같은 역할을 하는 2대의 CCD 카메라가 장착되어 있어 로봇 전방의 물체를 인식하는 역할을 한다. Table 2.3 과 같이 2개의 모터가 감속기와 풀리로 직결로 연결되어 관절을 구성하고 있으며 Fig. 2.4 와 같이 카메라가 바라보는 각도를 기준으로 상하 90°와 좌우 90°의 운동범위를 가지고 있어 카메라의 입사각을 고려하면 카메라가 볼 수 있는 시야는 로봇 전방 전체이므로 머리부의 상하운동을 생략하여도 무방하다. 머리부의 상하 운동을 생략함으로써 기구부의 설계와 기구학적 해석이 용이하게 되었다.

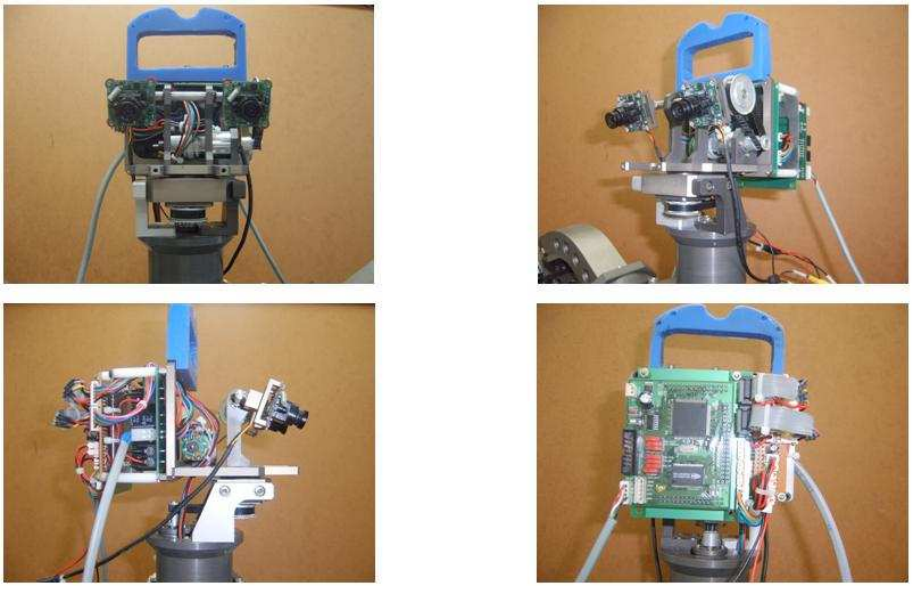


Fig. 2.3 The head of KUBIR III

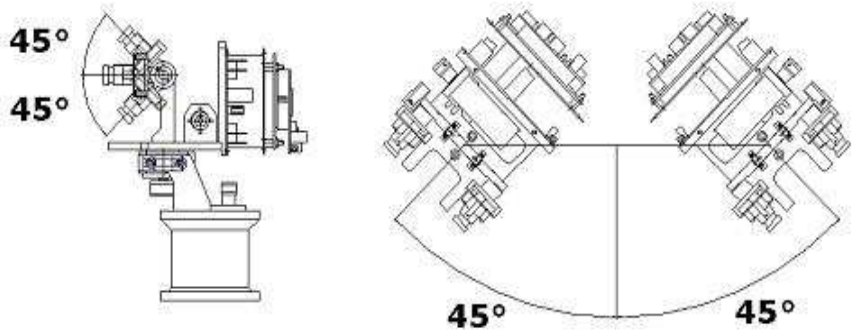


Fig. 2.4 Rotation range of the head

Table 2.3 Specification of the joint actuator for head part

축	운동	모터출력 [W]	풀리비	감속비
0	Yaw	1.5	14:40	1:104
1	pitch	1.5	14:40	1:104

2.3 KUBIR III의 허리 기구부 구성

KUBIR III의 14.4kgf 허리는 모터와 감속기가 1:150 비로 직결 연결된 구조로 1자유도를 가진다. 그리고 Fig.2.5와 같이 감속기 앞에 중력보상기(gravity compensator)가 장착되어 스프링의 탄성력으로 로봇의 상체가 중력에 의해 기울어지는 현상을 사전에 방지할 수 있으며 로봇보행 시 안정적인 움직임을 가능하게 한다. Fig.2.6은 KUBIR III의 허리에 부착되어 있는 중력보상기의 실제 모습이다.

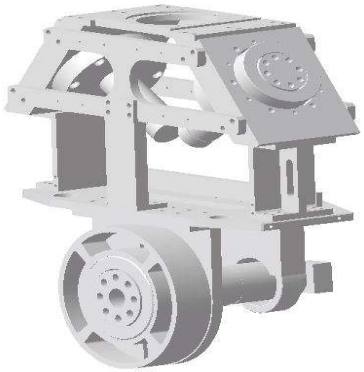


Fig. 2.5 3D model of the waist joint actuator

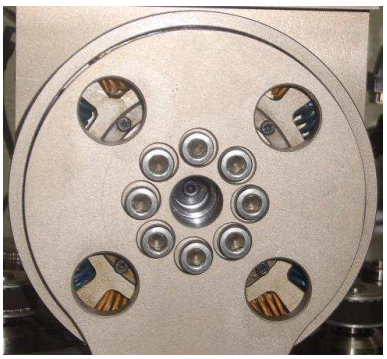


Fig. 2.6 Gravity compensator of the waist joint

2.4 KUBIR III의 팔 기구부 구성

KUBIR III의 양팔은 총 12자유도를 가지고 있으며 2개의 직결구조 yaw축과 6개의 4절 링크 구조 pitch축, 그리고 말단장치인 그리퍼(gripper) 2축으로 구성되어 있다.

Fig.2.7은 2.9kgf의 어깨부분의 관절구조로써 직결구조의 yaw축이 팔 전체를 회전할 수 있도록 되어 있으며 첫 번째 pitch축이 위치하고 있다. 어깨부분의 yaw운동은 몸체와 35°로 기울어져 있어 회전축으로 180° 운동이 가능하다. pitch축은 Fig.2.7과 같이 로봇 몸체로부터 안쪽으로 6°, 바깥쪽으로 30°의 운동범위를 갖는다. Fig.2.8은 어깨 부위의 기구부의 모습이다.

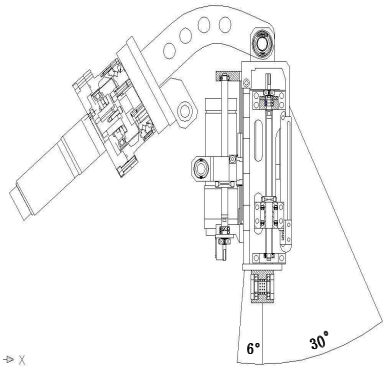


Fig. 2.7 Rotation range of the shoulder pitch joint

Fig. 2.8 Shoulder joint

1.4kgf의 팔꿈치관절은 Fig.2.9와 같은 구조이며 두 번째 pitch축이다. Fig.2.9와 같이 몸체의 앞쪽으로 70°의 운동범위를 갖는다. Fig.2.10은 팔꿈치 부위의 실제 모습이다.

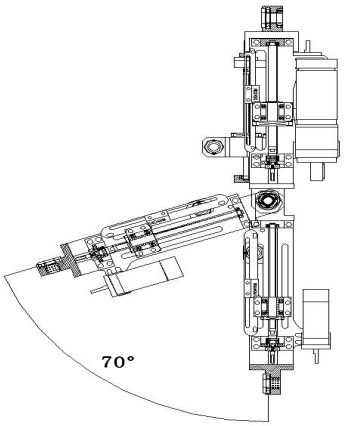


Fig. 2.9 Rotation range of the elbow pitch joint

Fig. 2.10 Elbow joint

1.9kgf의 손목부분은 Fig.2.11 과 같이 3번째 pitch축과 roll축, 그리고 말단장치인 그리퍼 이렇게 3개의 관절로 구성된다. yaw축은

180°의 운동범위를 가지며 pitch축은 Fig.2.11 과 같이 안쪽 35°의 범위에서 운동이 이루어진다. Fig.2.12는 손목부위와 그리퍼(gripper)부위의 실제 사진 모습이다.

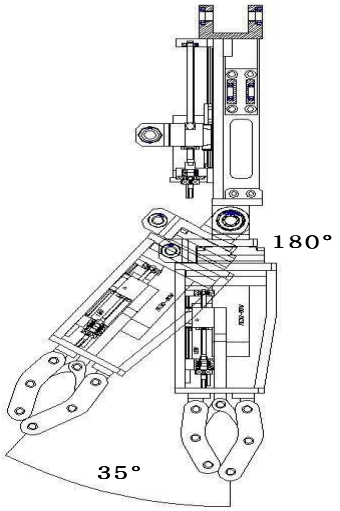


Fig. 2.11 Rotation range of the wrist pitch and roll joint

Fig. 2.12 Wrist joint

Table 2.4 는 양팔에 사용된 관절구동기의 사양을 나타낸다.

Table 2.4 Specification of the joint actuator for arm part

축	관절구조	운동	모터출력 [W]	풀리비	감속비	볼스크류 리드[㎜]
0	직결	yaw	60	-	1 : 160	-
1	4 절링크	pitch	150	1 : 1	1 : 4.3	2
2	4 절링크	pitch	60	1 : 1.75	-	1
3	4 절링크	pitch	60	1 : 1.75	-	1
4	직결	yaw	60	-	1 : 100	-
5	4 절링크	gripper	18	1 : 1.75	-	1

2.5 Four-bar-link의 구성

4절 링크는 관절 구동기에서 사용되고 있으며, 골반관절의 pitch축, 무릎관절, 발목의 pitch축에 사용되고 있다. 4절 링크의 한 변을 강성이 높은 관절 볼 나사를 적용하여 모멘트를 발생 시키는 방법으로 상대적으로 높은 관절 구동력을 발생 할 수 있는 장점과 볼 나사의 용이한 교체로 관절 회전 속도를 상황에 맞게 다양하게 구성 할 수 있는 장점이 있다. 이러한 기구부에 모터와 모터 드라이브 모듈을 구성함으로써 보행 로봇에 용이하게 탈 부착하여 사용할 쉬 있게 설계하였다. 볼 나사는 강성과 정도가 매우 높으므로 DC모터와 직결 연결하여 사용되며, 구조에 따라 타이밍 벨트를 이용하여 융통성 있게 구성 될 수 있다.

Fig.2.13은 4절 링크의 구조를 나타내었다.

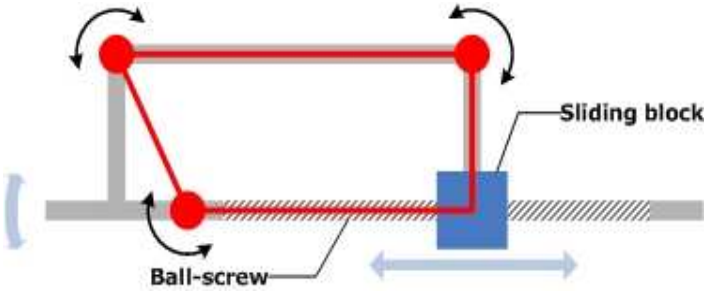


Fig. 2.13 Structure of four-bar-link

2.6 KUBIR III의 머리부의 기구학 해석

KUBIR III는 강체 로봇(rigid robot)이다. KUBIR III에 대한 순기구학 방정식 또는 형상 기구학 방정식은 로봇의 관절 변수가 주어졌을 때, 말단 장치의 위치와 방향을 결정한다. 회전 관절일 경우에 관절 변수는 링크 사이의 각도이고, 직선 관절일 경우에 관절 변수는 링크의 늘어난 길이이다. 순기구학은 로봇이 움직이는 동작을 알아내기 위한 첫 번째 방법이다[7].

본 논문에서 기구학 해석은 로봇 머리의 각 관절의 각과 스테레오비젼 시스템에 의해 산출되는 특정 물체의 거리에 대한 말단 좌표를 구하기 위하여 이용되었다.

로봇 머리 부분의 D-H 좌표계를 자세히 나타내면 Fig.2.14 와 같다.

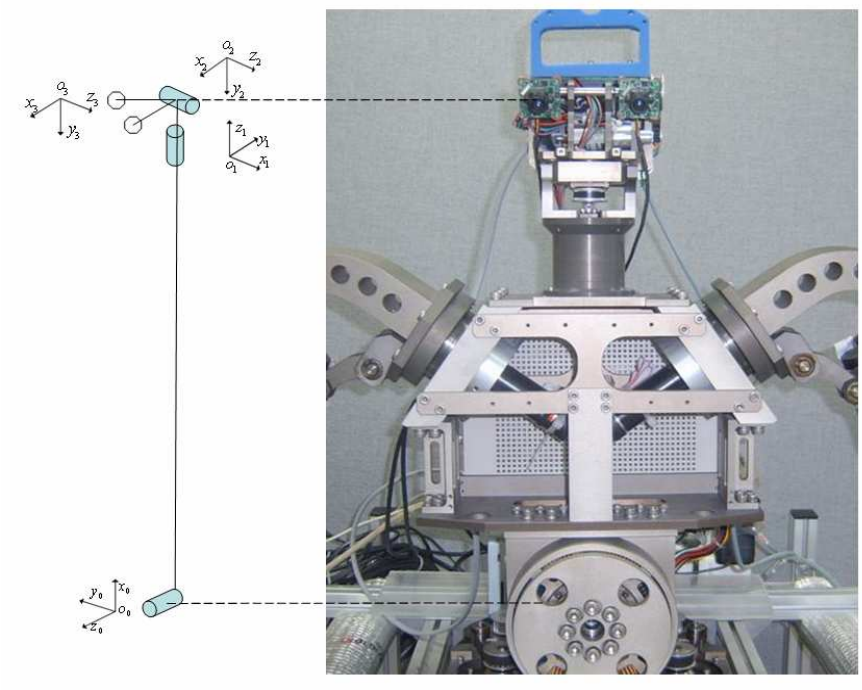


Fig. 2.14 D-H coordinate for the head

Table 2.5 Link parameter of the head of KUBIR-III

link	a_i	α_i	d_i	θ_i
1	0	90°	0	θ_1^*
2	0	-90°	d_2	θ_2^*
3	a_3^*	0	0	θ_3^*

여기서 a_i는 각 관절의 링크길이(length), α_i는 비틀림(twist), d_i는 오프셋(offset), θ_i는 각도(angle)이다.

d_2는 목에서 머리까지의 길이이며, 이 길이는 약 17.15cm정도이다.

A 행렬들은 Table 2.5 의 매개변수들로부터 다음과 같이 얻어진다.

$$A_1 = \begin{bmatrix} C\theta_1 & 0 & S\theta_1 & 0 \\ S\theta_1 & 0 & -C\theta_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} C\theta_2 & 0 & -S\theta_2 & 0 \\ S\theta_2 & 0 & C\theta_2 & 0 \\ 0 & -1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} C\theta_3 & -S\theta_3 & 0 & a_3 C\theta_3 \\ S\theta_3 & C\theta_3 & 0 & a_3 S\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.1)$$

따라서 변환행렬 T는 다음과 같이 주어진다.

$$T_0^3 = A_1 A_2 A_3 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & P_x \\ r_{21} & r_{22} & r_{23} & P_y \\ r_{31} & r_{32} & r_{33} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$$r_{11} = c\theta_1 c\theta_2 c\theta_3 - s\theta_1 s\theta_3$$
$$r_{12} = -c\theta_1 c\theta_2 s\theta_3 - s\theta_1 c\theta_3$$
$$r_{13} = -c\theta_1 s\theta_2$$
$$P_x = c\theta_1 c\theta_2 a_3 c\theta_3 - s\theta_1 a_3 s\theta_3 + s\theta_1 d_2$$

$$r_{21} = s\theta_1 c\theta_2 c\theta_3 + c\theta_1 s\theta_3$$
$$r_{22} = -s\theta_1 c\theta_2 s\theta_3 + c\theta_1 c\theta_3$$
$$r_{23} = -s\theta_1 s\theta_2$$
$$P_y = s\theta_1 c\theta_2 a_3 c\theta_3 + c\theta_1 a_3 s\theta_3 - c\theta_1 d_2$$

$$r_{31} = s\theta_2 c\theta_3$$
$$r_{32} = -s\theta_2 s\theta_3$$
$$r_{33} = c\theta_2$$
$$P_z = s\theta_2 a_3 c\theta_3$$

그러므로 로봇 머리의 말단 위치벡터 P_{head}는 다음과 같다.

$$P_{head}=\begin{bmatrix}P_x\\P_y\\P_z\end{bmatrix}=\begin{bmatrix}c\theta_1c\theta_2a_3c\theta_3-s\theta_1a_3s\theta_3+s\theta_1d_2\\s\theta_1c\theta_2a_3c\theta_3+c\theta_1a_3s\theta_3-c\theta_1d_2\\s\theta_2a_3c\theta_3\end{bmatrix} \quad (2.3)$$

2.7 KUBIR III의 팔의 기구학 해석

로봇 팔 부분의 D-H 좌표계를 자세히 나타내면 Fig.2.15 와 같다.

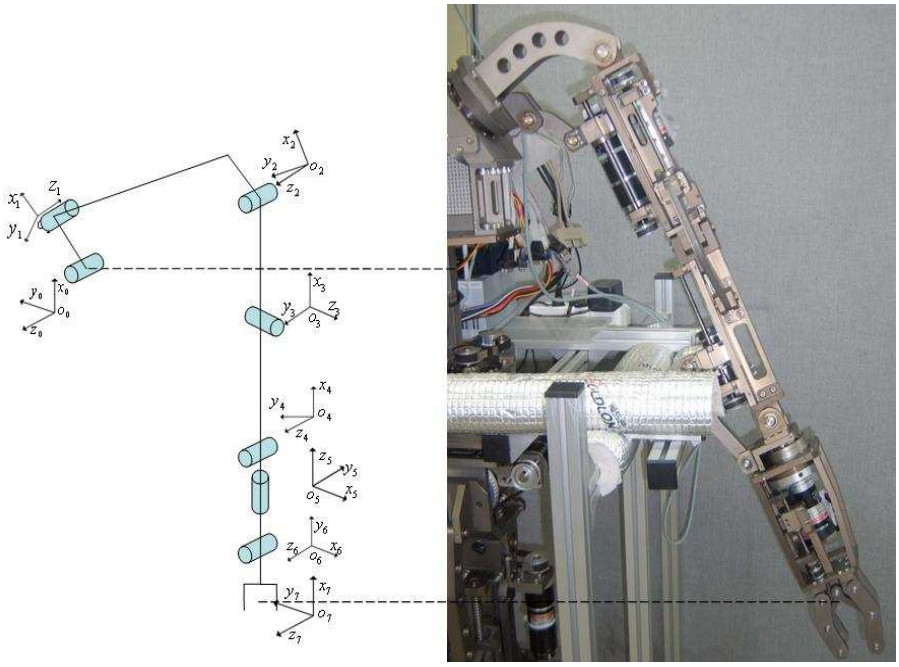


Fig. 2.15 D-H coordinate for the arm

Table 2.6 Link parameter of the arm of KUBIR-III

link	a_i	α_i	d_i	θ_i
1	a_1	90°	0	θ_1^*
2	0	-90°	d_2	θ_2^*
3	a_3	90°	0	θ_3^*
4	a_4	-90°	0	θ_4^*
5	0	90°	0	θ_5^*
6	0	-90°	d_6	θ_6^*
7	a_7^*	0	0	θ_7^*

여기서 a_1 은 허리 중심에서 어깨 시작까지의 거리이며, 약 26cm이다. d_2 는 어깨 시작점부터 어깨 끝까지의 거리이며, 약 18cm이다. a_3 는 어깨 끝부터 팔꿈치까지의 거리이며, 약 22cm이다. 또한, a_4 는 팔꿈치부터 손목 윗부분까지이며, 약 23cm이고, d_6 는 손목 윗부분부터 손목까지의 거리이며, 거리는 약 20cm 이다.

A 행렬들은 Table 2.6의 매개변수들로부터 다음과 같이 얻어진다.

$$A_1 = \begin{bmatrix} C\theta_1 & 0 & S\theta_1 & a_1 C\theta_1 \\ S\theta_1 & 0 & -C\theta_1 & a_1 S\theta_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} C\theta_2 & 0 & -S\theta_2 & 0 \\ S\theta_2 & 0 & C\theta_2 & 0 \\ 0 & -1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} C\theta_3 & 0 & S\theta_3 & a_3C\theta_3 \\ S\theta_3 & 0 & -C\theta_3 & a_3S\theta_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} C\theta_4 & 0 & -S\theta_1 & a_4C\theta_4 \\ S\theta_4 & 0 & C\theta_1 & a_4S\theta_4 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} C\theta_5 & 0 & S\theta_5 & 0 \\ S\theta_5 & 0 & -C\theta_5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_6 = \begin{bmatrix} C\theta_6 & 0 & -S\theta_6 & 0 \\ S\theta_6 & 0 & C\theta_6 & 0 \\ 0 & -1 & 0 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_7 = \begin{bmatrix} C\theta_7 & -S\theta_7 & 0 & a_7 C\theta_7 \\ S\theta_7 & C\theta_7 & 0 & a_7 S\theta_7 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

따라서 변환행렬 T 는 다음과 같이 주어진다.

$$T_0^7 = A_1 A_2 A_3 A_4 A_5 A_6 A_7 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & P_x \\ r_{21} & r_{22} & r_{23} & P_y \\ r_{31} & r_{32} & r_{33} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$r_{11} = ((((c\theta_1 c\theta_2 c\theta_3 - s\theta_1 s\theta_3)c\theta_4 - c\theta_1 s\theta_2 s\theta_4)c\theta_5 + (-c\theta_1 c\theta_2 s\theta_3 - s\theta_1 c\theta_3)s\theta_5)c\theta_6 + (-(c\theta_1 c\theta_2 c\theta_3 - s\theta_1 s\theta_3)s\theta_4 - c\theta_1 s\theta_2 c\theta_4)s\theta_6)c\theta_7 + (-((c\theta_1 c\theta_2 c\theta_3 - s\theta_1 s\theta_3)c\theta_4 - c\theta_1 s\theta_2 s\theta_4)s\theta_5 + (-c\theta_1 c\theta_2 s\theta_3 - s\theta_1 c\theta_3)c\theta_5)s\theta_7$

$r_{12} = -((((c\theta_1 c\theta_2 c\theta_3 - s\theta_1 s\theta_3)c\theta_4 - c\theta_1 s\theta_2 s\theta_4)c\theta_5 + (-c\theta_1 c\theta_2 s\theta_3 - s\theta_1 c\theta_3)s\theta_5)c\theta_6 + (-(c\theta_1 c\theta_2 c\theta_3 - s\theta_1 s\theta_3)s\theta_4 - c\theta_1 s\theta_2 c\theta_4)s\theta_6)s\theta_7 + (-((c\theta_1 c\theta_2 c\theta_3 - s\theta_1 s\theta_3)c\theta_4 - c\theta_1 s\theta_2 s\theta_4)s\theta_5 + (-c\theta_1 c\theta_2 s\theta_3 - s\theta_1 c\theta_3)c\theta_5)c\theta_7$

$r_{13} = -(((c\theta_1 c\theta_2 c\theta_3 - s\theta_1 s\theta_3)c\theta_4 - c\theta_1 s\theta_2 s\theta_4)c\theta_5 + (-c\theta_1 c\theta_2 s\theta_3 - s\theta_1 c\theta_3)s\theta_5)s\theta_6 + (-(c\theta_1 c\theta_2 c\theta_3 - s\theta_1 s\theta_3)s\theta_4 - c\theta_1 s\theta_2 c\theta_4)c\theta_6$

$$P_x = ((((c\theta_1 c\theta_2 c\theta_3 - s\theta_1 s\theta_3)c\theta_4 - c\theta_1 s\theta_2 s\theta_4)c\theta_5 + (-c\theta_1 c\theta_2 s\theta_3 - s\theta_1 c\theta_3)s\theta_5)c\theta_6 + (-(c\theta_1 c\theta_2 c\theta_3 - s\theta_1 s\theta_3)s\theta_4 - c\theta_1 s\theta_2 c\theta_4)s\theta_6)a_7 c\theta_7 + (-((c\theta_1 c\theta_2 c\theta_3 - s\theta_1 s\theta_3)c\theta_4 - c\theta_1 s\theta_2 s\theta_4)s\theta_5 + (-c\theta_1 c\theta_2 s\theta_3 - s\theta_1 c\theta_3)c\theta_5)a_7 c\theta_7 + (((c\theta_1 c\theta_2 c\theta_3 - s\theta_1 s\theta_3)c\theta_4 - c\theta_1 s\theta_2 s\theta_4)s\theta_5 - (-c\theta_1 c\theta_2 s\theta_3 - s\theta_1 c\theta_3)c\theta_5)d_6 + (c\theta_1 c\theta_2 c\theta_3 - s\theta_1 s\theta_3)a_4 c\theta_4 - c\theta_1 s\theta_2 a_4 s\theta_4 - s\theta_1 a_3 c\theta_3 + s\theta_1 d_2 + a_1 c\theta_1$$

$$r_{21} = ((((s\theta_1 c\theta_2 c\theta_3 - c\theta_1 s\theta_3)c\theta_4 - s\theta_1 s\theta_2 s\theta_4)c\theta_5 + (-s\theta_1 c\theta_2 s\theta_3 + c\theta_1 c\theta_3)s\theta_5)c\theta_6 + (-(s\theta_1 c\theta_2 c\theta_3 - c\theta_1 s\theta_3)s\theta_4 - s\theta_1 s\theta_2 c\theta_4)s\theta_6)c\theta_7 + (-((s\theta_1 c\theta_2 c\theta_3 + c\theta_1 s\theta_3)c\theta_4 - s\theta_1 s\theta_2 s\theta_4)s\theta_5 + (-s\theta_1 c\theta_2 s\theta_3 + c\theta_1 c\theta_3)c\theta_5)s\theta_7$$

$$r_{22} = -((((s\theta_1 c\theta_2 c\theta_3 + c\theta_1 s\theta_3)c\theta_4 - s\theta_1 s\theta_2 s\theta_4)c\theta_5 + (-s\theta_1 c\theta_2 s\theta_3 + c\theta_1 c\theta_3)s\theta_5)c\theta_6 + (-(s\theta_1 c\theta_2 c\theta_3 + c\theta_1 s\theta_3)s\theta_4 - s\theta_1 s\theta_2 c\theta_4)s\theta_6)s\theta_7 + (-((s\theta_1 c\theta_2 c\theta_3 + c\theta_1 s\theta_3)c\theta_4 - s\theta_1 s\theta_2 s\theta_4)s\theta_5 + (-s\theta_1 c\theta_2 s\theta_3 + c\theta_1 c\theta_3)c\theta_5)c\theta_7$$

$$r_{23} = -(((s\theta_1 c\theta_2 c\theta_3 + c\theta_1 s\theta_3)c\theta_4 - s\theta_1 s\theta_2 s\theta_4)c\theta_5 + (-s\theta_1 c\theta_2 s\theta_3 - c\theta_1 c\theta_3)s\theta_5)s\theta_6 + (-(s\theta_1 c\theta_2 c\theta_3 + c\theta_1 s\theta_3)s\theta_4 - s\theta_1 s\theta_2 c\theta_4)c\theta_6$$

$$P_y = ((((s\theta_1 c\theta_2 c\theta_3 + c\theta_1 s\theta_3)c\theta_4 - s\theta_1 s\theta_2 s\theta_4)c\theta_5 + (-s\theta_1 c\theta_2 s\theta_3 + c\theta_1 c\theta_3)s\theta_5)c\theta_6 + (-(s\theta_1 c\theta_2 c\theta_3 + c\theta_1 s\theta_3)s\theta_4 - s\theta_1 s\theta_2 c\theta_4)s\theta_6)a_7 c\theta_7 + (-((s\theta_1 c\theta_2 c\theta_3 + c\theta_1 s\theta_3)c\theta_4 - s\theta_1 s\theta_2 s\theta_4)s\theta_5 + (-s\theta_1 c\theta_2 s\theta_3 + c\theta_1 c\theta_3)c\theta_5)a_7 c\theta_7 + (((s\theta_1 c\theta_2 c\theta_3 + c\theta_1 s\theta_3)c\theta_4 - s\theta_1 s\theta_2 s\theta_4)s\theta_5 - (-s\theta_1 c\theta_2 s\theta_3 + c\theta_1 c\theta_3)c\theta_5)d_6 + (s\theta_1 c\theta_2 c\theta_3 + c\theta_1 s\theta_3)a_4 c\theta_4 - s\theta_1 s\theta_2 a_4 s\theta_4 + c\theta_1 a\theta_3 c\theta_3 - c\theta_1 d_2 + a_1 s\theta_1$$

$$r_{31} = (((s\theta_2 c\theta_3 c\theta_4 + c\theta_2 s\theta_4)c\theta_5 - s\theta_2 s\theta_3 s\theta_5)c\theta_6 + (-s\theta_2 c\theta_3 s\theta_4 + c\theta_2 c\theta_4)s\theta_6)c\theta_7 + (-(s\theta_2 c\theta_3 c\theta_4 + c\theta_2 s\theta_4)s\theta_5 - s\theta_2 s\theta_3 c\theta_5)s\theta_7$$

$$r_{32} = -(((s\theta_2 c\theta_3 c\theta_4 + c\theta_2 s\theta_4)c\theta_5 - s\theta_2 s\theta_3 s\theta_5)c\theta_6 + (-s\theta_2 c\theta_3 s\theta_4 + c\theta_2 c\theta_4)s\theta_6)s\theta_7 + (-(s\theta_2 c\theta_3 c\theta_4 + c\theta_2 s\theta_4)s\theta_5 - s\theta_2 s\theta_3 c\theta_5)c\theta_7$$

$$r_{33} = -((s\theta_2 c\theta_3 c\theta_4 + c\theta_2 s\theta_4)c\theta_5 - s\theta_2 s\theta_3 s\theta_5)s\theta_6 + (-s\theta_2 c\theta_3 s\theta_4 + c\theta_2 c\theta_4)c\theta_6$$

$$P_z = (((s\theta_2 c\theta_3 c\theta_4 + c\theta_2 s\theta_4)c\theta_5 - s\theta_2 s\theta_3 s\theta_5)c\theta_6 + (-s\theta_2 c\theta_3 s\theta_4 + c\theta_2 c\theta_4)s\theta_6)a_7 c\theta_7 + (-(s\theta_2 c\theta_3 c\theta_4 + c\theta_2 s\theta_4)s\theta_5 - s\theta_2 s\theta_3 c\theta_5)a_7 c\theta_7 + ((s\theta_2 c\theta_3 c\theta_4 + c\theta_2 s\theta_4)s\theta_5 + s\theta_2 s\theta_3 c\theta_5)d_6 + s\theta_2 c\theta_3 a_4 c\theta_4 + c\theta_2 a_4 s\theta_4$$

그러므로 로봇 팔의 말단 위치벡터 P_{arm}는 다음과 같다.

$$P_{arm} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}$$

2.8 Four-bar-link의 기구학적 해석

KUBIR III의 양팔에 사용된 4절링크 관절을 제어하기 위해, Fig.2.16과 같이 네 변 중 세 변의 길이는 고정이고 나머지 한 변은 볼 스크류의 변위로 만들어 볼 스크류의 지지부와 관절 각을 고정 각으로 하여 관절 각과 볼 스크류의 변위와의 관계를 구하였다.

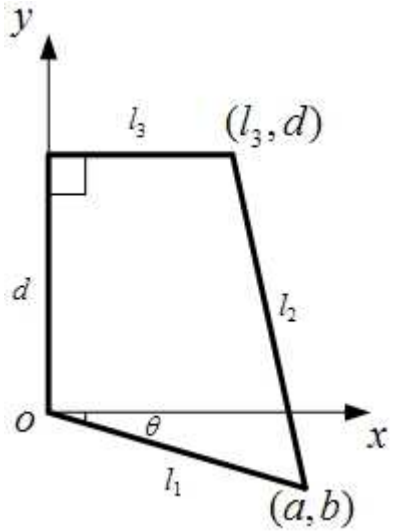


Fig. 2.16 Four-bar link modeling

첫 번째 링크 l_1에 의해 결정되는 임의의 점을 (a,b)라 하면

$$a = l_1 \cos\theta$$
$$b = l_1 \sin\theta \qquad (2.6)$$

이고 중심이 (a,b)이고 반지름이 l_2인 원의 방정식은 식 (2.7) 이다.

$$(x-a)^2 + (y-b)^2 = l_2^2 \qquad (2.7)$$

점 (l_3, d)을 위 식에 대입하여 d에 관하여 정리하면

$$d^2 - 2bd + a^2 + b^2 + l_3^2 - l_2^2 - 2al_3 = 0 \qquad (2.8)$$

위의 2차 방정식을 풀면

$$d = b \pm \sqrt{(l_2 + a - l_3)(l_2 - a + l_3)} \qquad (2.9)$$

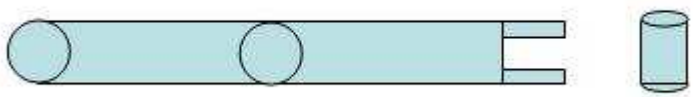
변위 d 는 위와 같이 구할 수 있다.

2.9 역기구학

기구학은 연결구조 끝의 공간내의 위치를 계산하는 과정이다. 연결구조의 모든 관절에는 각이 존재한다. 이 계산은 쉽고 또 해결 방법도 한가지밖에 없다. 역기구학(inverse kinematics)은 기구학과 역순으로 하는 것이다. 구조의 종점(end point)이 주어졌을 때, 어떠한 각은 종점까지 다를 수 있는 관절을 만들 수 있다. 이 과정은 로봇공학에서 아주 유용하다. 로봇 팔이 어떤 물체를 들어올린다고 할 때, 소프트웨어가 어깨로부터 상대적으로 어느 위치에 물체가 있는지 안다고 한다면, 간단히 물체에 다다르기 위한 관절의 각을 계산하면 된다[8].

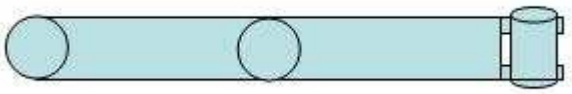
역기구학으로 물체의 위치까지의 종점을 찾아내기 위한 방법은 아래와 같다.

(1) 해결방법이 없는 경우



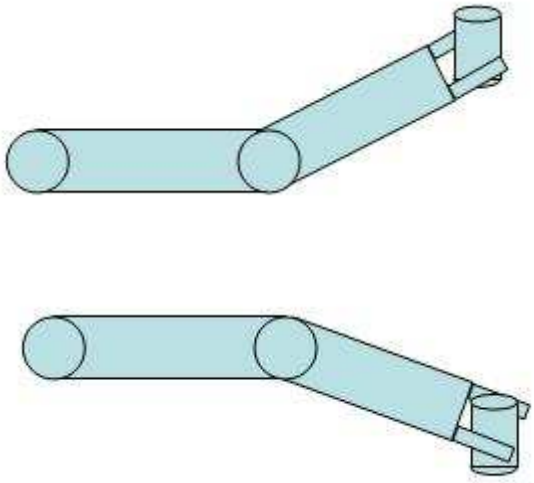
많은 경우들 중에, 연결구조가 목표물에 다다를 수 없는 경우이다. 예를 들어, 자신의 팔꿈치를 자기 손으로 건드릴 수 없는 것처럼 시스템이 매우 불안정하게 될 것이다. 다시 말하면, 타겟이 연결구조의 영역 밖에 있는 경우이다.

(2) 해결방법이 하나인 경우



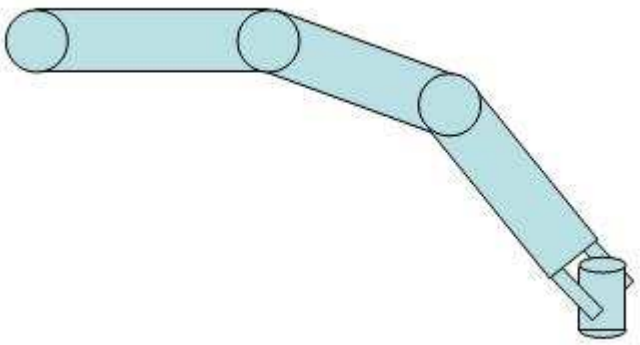
해결방법이 하나인 경우에는 연결구조가 타겟의 위치에 종점이 하나인 경우이다.

(3) 해결방법이 두 개인 경우



해결방법이 두 가지인 경우, 이 기술은 현 상태의 연결구조에서 어느 것이 가장 가까울 것인지 찾아낸다.

(4) 해결방법이 많은 경우



관절이 두 개 이상이 되면 해결방법이 무한에 가깝게 존재 할 수도 있다. 하지만 몇몇 해결방법은 다른 해결방법보다 나은 것이 있다. 예로 연결구조가 팔을 나타내고 있다면, 어떤 해결책은 더 안정적으로 보일 것이고, 다른 것들은 뭔가 굉장히 무리가 있어 보일 것이다. 이런 해결방법이 대부분 최적의 해결방법이다.

(5) 역기구학 해석

아래의 Fig. 2.17에서의 3자유도의 구형 머니퓰레이터(manipulator)의 역위치기구학을 구하고자 한다면, 팔꿈치형 머니퓰레이터처럼 첫 관절변수는 기저(base)의 회전이 되고, 그 답은 p_x, p_y 가 동시에 0이 아니면

θ_1 = Atan(p_x, p_y)

로 주어진다. p_x와 p_y가 모두 0 이면 특이형상이 되고, θ_1은 어떤 값도 가질 수 있다.

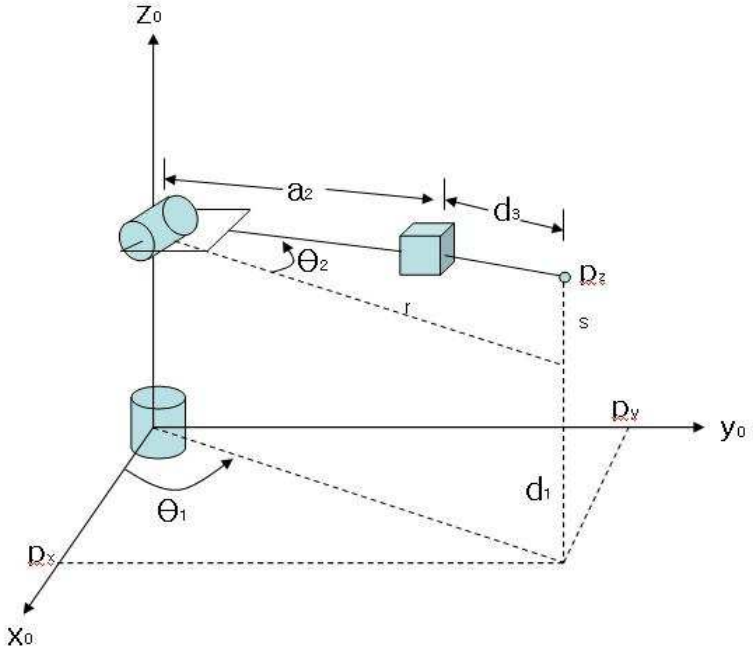


Fig. 2.17 Spherical configuration manipulator

Fig. 2.17에서 각도 θ_2는

θ_2 = Atan(r , s)

와 같고, 여기서 $r^2 = p_x^2 + p_y^2$, s = $p_z - d_1$ 이다. 팔꿈치형 머니퓰레이터 처럼 θ_1, θ_2의 두 번째 답은 다음과 같다.

$\theta_1 = \pi$ + Atan(p_x , p_y) ; $\theta_2 = \pi$ - Atan(r , s)

직선거리 d_3는

$(d_3 + a_2)^2 = r^2 + s^2$

에서

$$d_3 = \sqrt{r^2 + s^2} - a_2 = \sqrt{p_x^2 + p_y^2 + (p_z - d_1)^2} - a_2$$

와 같이 구해진다.

직선거리 d_3의 음의 제곱근을 버리면, 손목중심이 z_0와 만나지 않는 한 역위치기구학의 답 둘을 얻는다. 오프셋이 있으면 팔꿈치형 머니퓰레이터와 같이 왼팔 형상과 오른팔 형상을 가진다[7].

제 3 장 로봇의 제어부 구성

3.1 Embedded System의 개요

임베디드 시스템이란 기계 또는 전자 장치에 두뇌 역할을 하는 마이크로 프로세서(microprocessor)를 장착해 설계함으로써 효과적인 제어를 할 수 있도록 하는 시스템을 의미한다[9].

일반적으로 임베디드 시스템은 하드웨어 구성이 고정되어 있으며, 소프트웨어는 하드웨어를 재구성하거나 바꾸는 일은 할 수 없고, 응용 프로그램도 하드웨어에 내장된 상태로만 사용할 수 있는데, 이것을 한마디로 정의하면 임베디드 시스템은 정해진 용도에 맞게 최적화한 것이라고 할 수 있다.

다시 말하면 PC는 다목적 시스템인데 비해 임베디드 시스템은 대부분 특정 목적 한 가지 용도를 위해 제작되어 있으며, 이러한 예로 우리 생활에서 사용되는 각종 가전제품, 전자기기, 제어장치들도 단순히 전자 적인 회로들만 구성된 것이 아니라 마이크로 프로세서가 내장되어 있으며, 이러한 마이크로 프로세서에는 특정한 기능을 수행하도록 프로그램이 내장 되어 있는 것을 볼 수 있다.

이렇게 임베디드 시스템은 산업분야, 가전분야, 사무분야, 군사분야 등 다양한 응용 분야를 가지고 있으며, 일상의 적용 예로는 핸드폰, PDA, 사이버 아파트의 홈 관리 시스템, 홈 네트워크 게이트웨이 장치, 교통관리 시스템, 주차관리 시스템, 엘리베이터 시스템, 현금지급기(ATM), 항공 관제 시스템, 우주선 제어 장치, 군사용 제어 장치 등을 들 수 있다.

임베디드 소프트웨어는 여러가지로 나뉘어지는데, 그 중에서도 특히 '임베디드 OS(operating system)'가 중요하게 부각되고 있다. 임베디드 시스템 용도에 맞는 기능을 얼마나 빨리 잘 구현해 주느냐가 이 운영체제의 핵심인데, 윈도CE, 팜 OS 등이 대표적이고 리눅스 채택도 늘고 있다. 하지만 본 연구에서는 영상처리, 모터 제어, 센서 입력.출력 등을 용이하게 제어 할 수 있는 Windows XP를 운영체제로 사용하였다.

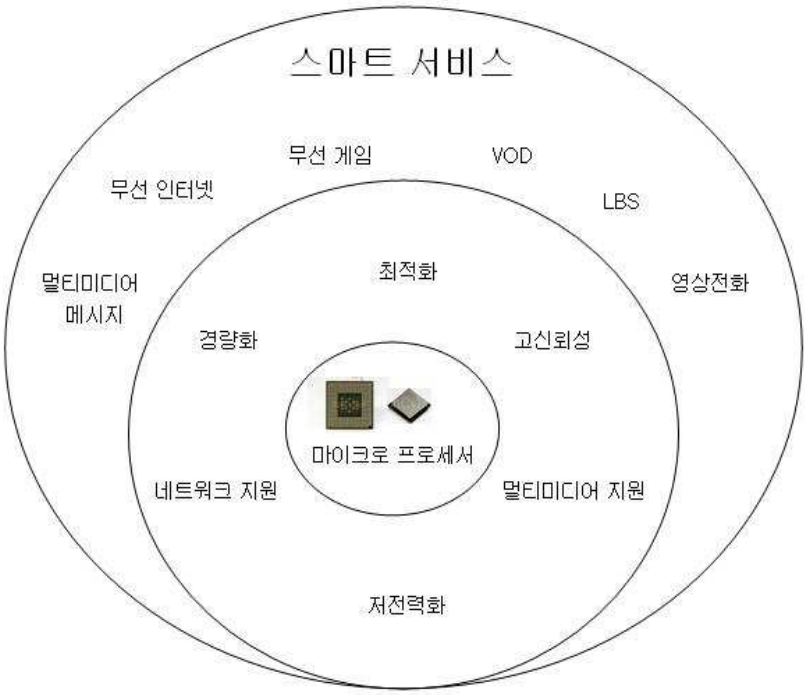


Fig. 3.1 Embedded system application

3.2 Embedded System의 개발 환경

개발환경을 구축하기 전에 일반적으로 임베디드 시스템을 개발하기 위해서 어떻게 환경을 구축하여 사용하고 있는지 알아보았다. 임베디드 소프트웨어 개발자가 아닌 일반 프로그래머들은 PC환경에서 개발하며 그렇게 개발한 프로그램 역시 PC 환경에서 실행하기 때문에 모니터와 키보드, 마우스와 같은 표준 입. 출력 하드웨어들을 모두 구비하고 있는 PC환경하에서 개발 할 때는 이러한 개발 환경 때문에 고민하는 일은 보통 하지 않는다.

그러나, 임베디드 소프트웨어를 처음으로 접하는 개발자들이 가장 먼저 닥치는 난관이 바로 개발환경 구축에 관련된 것이다. 임베디드 소프트웨어란 자체가 PC와는 달리 표준 입. 출력 하드웨어가 전혀 없고 보드(board)만 있는 플랫폼에서 개발해야 하기 때문에 익숙하지 않은 개발자들에게는 매우 난감한 문제가 아닐 수 없다.

일반적인 임베디드 시스템 개발 환경은 아래 그림 Fig.3.2와 같다.

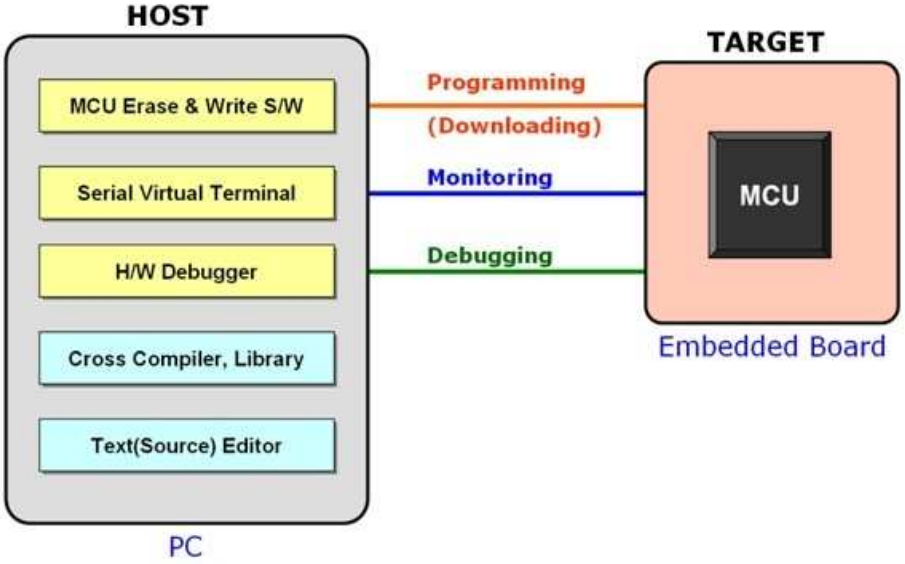


Fig. 3.2 Embedded system development environment

먼저, 우리가 프로그램 source 코드를 개발하는 PC를 일반적으로 호스트(host)라고 하며, 개발의 목적이 되는 임베디드 보드를 일반적으로 타겟 보드(target board)라고 말한다.

임베디드 시스템 개발 환경을 구축한다는 의미는 바로 위의 그림 Fig.3.2와 같이 호스트 환경을 구축한다는 의미이다[10]. 이러한 호스트 환경은 크게 하드웨어 환경과 소프트웨어 환경으로 나눌 수 있는데, 먼저 하드웨어 환경은 호스트와 타겟 보드 간의 데이터를 주고 받기 위해 케이블로 연결한다는 의미이며, 소프트웨어 환경은 이렇게 연결된 케이블에 데이터를 주고 받기 위해 호스트 상에 해당 소프트웨어들을 설치하고 설정하는 것을 의미한다.

이러한 하드웨어, 소프트웨어 환경은 호스트와 타겟 보드 간에 주고 받는 데이터에 따라 위의 그림 Fig.3.2처럼 크게 세 부분으로 나뉘고 추가로 호스트에 cross compiler와 library가 있다.

1) Programming (downloading)

타겟 보드를 구동하기 위해 필요한 프로그램 코드들을 타겟 보드에 내장된 메모리나 혹은 해당 프로세서 내부에 내장된 프로그램 메모리에 저장하기 위한 환경을 말하며, 이를 별도의 케이블과 별도의 전용 소프트웨어가 필요하다.

일반적으로 사용하는 32bit의 고 사양 프로세서의 경우에는 프로세서 내부에 프로그램 코드용 메모리가 따로 없으므로 이 경우에는 보통 타겟 보드에 장착된 외부 Flash 메모리로 해당 프로그램 코드를 저장하며, ATmega128의 경우에는 프로세서 내부에 프로그램 메모리가 따로 있으므로 여기에 저장 할 수 있다. 또한, 32bit 프로세서에 돌아가는 일부 운영체제의 경우에는 타겟 보드 구동에 필요한 프로그램 코드 사이즈가 비교적 크기 때문에 병렬 케이블과 같은 저속의 케이블 대신 이더넷과 같은 고속의 케이블을 별도로 연결하는 경우도 있다.

2) Monitoring

타겟 보드는 호스트와 달리 모니터와 키보드, 마우스 등의 표준 입, 출력 하드웨어가 없는 경우가 대부분이기 때문에 타겟 보드에 명령을 입력하거나 타겟 보드의 상태를 독립적으로 확인해 볼 수 없기 때문에 시리얼 연결을 통한 터미널 응용 프로그램을 통해 호스트에서 확인해야 한다. 이를 위한 별도의 케이블 연결과 소프트웨어 설치 및 설정이 필요하다.

3) Debugging

앞서 언급했듯이 타겟 보드는 별도의 입. 출력 하드웨어가 없기 때문에 내부 상태를 호스트의 가상 터미널을 통해 소프트웨어적으로 확인해 보거나, 혹은 하드웨어 전용 장비를 통해 하드웨어적으로 확인해 볼 수 있는데 이러한 하드웨어 디버깅을 위해서는 별도의 케이블과 에뮬레이터와 같은 전용 하드웨어 장비 또한 그에 따른 소프트웨어를 구축하여야 한다.

4) Cross Compiler & Library

Programming 과 Monitoring, Debugging 환경은 이미 타겟 보드용 프로그램 코드를 개발한 상태에서 호스트와 타겟 보드 간 연결을 위해 필요한 환경이라면, tool chain은 타겟 보드의 프로그램 코드를 개발하기 위한 소프트웨어 환경을 말한다. 임베디드 소프트웨어란 결국 타겟 보드를 구동하기 위해 필요한 소프트웨어라고 할 수 있다. 일반적으로 PC상에서 실행되는 소프트웨어는 PC상에서 개발하는 것과는 달리 타겟 보드는 타겟 보드에서 직접 소프트웨어를 개발하기가 불가능하기 때문에 타겟 보드용 소프트웨어는 결국 호스트에서 개발하여야 한다.

그러기 위해 필요한 컴파일러를 호스트에서 타겟 보드용 소스를 컴파일 한다고 해서 크로스 컴파일러라 하며, 이 때 필요한 라이브

러리와 관련 유틸리티들을 일컬어 tool chain이라 하며 임베디드 개발 환경 구축 시 가장 먼저 구축해야 하는 부분이다.

3.3 KUBIR III의 전체 제어부 구성

KUBIR III의 주 제어기인 embedded mini computer를 등에 부착시켜 2개의 CCD 카메라와 27개의 motor, motor driver, motion board, TMS320LF2407을 구동시킨다. 그리고 각각의 관절 제어기는 CAN(controller area network)으로 연결되어 있으며 관절 제어기로 제어 명령을 내리게 된다. 또한, mini computer가 로봇에 장착되어 있으므로 미리 저장된 입력 값에 따라 외부 명령 없이 동작이 가능하게 되었고, 여러 가지 센서를 입력 받아 자동 제어가 가능하게 되었다.

개발에 사용된 컴퓨터에는 무선 랜카드가 장착되어 있어 외부에서 로봇으로 명령을 내릴 수 있으며 반대로 관절 각 및 에러 등과 같은 로봇의 현재 상태를 외부로 알릴 수 있다. 사용된 무선 이더넷 네트워크는 애드혹(ad-hoc)으로써 애드혹은 네트워크의 구성 및 유지를 위해 기지국이나 액세스포인트(access point)와 같은 기반 네트워크 장치가 필요하지 않다는 장점이 있다. 따라서 조종기 역할을 하는 노트북의 무선 랜과 로봇 주제어기의 무선 랜이 중계기 없이 바로 연결되어 통신이 가능하다[11].

또한 로봇의 머리 부분에 설치된 2대의 CCD 카메라로부터 들어오는 아날로그 영상신호를 획득하여 이진화하기 위해 주 제어기에는 이미지그래버(image grabber)가 설치되어 있으며 로봇은 영상처리를 통해 입력된 영상에서 특정 물체를 검출하여 관절 제어에 적용할 수 있도록 되어 있다.

KUBIR III에서는 부피의 최소화와 시스템에 알맞게 사용하기 위하여 자체적으로 컨트롤러와 모터 드라이브를 개발하였다. 개발된 컨트롤러는 하나의 드라이버로 두 개의 모터를 제어할 수 있다는 장점이 있다. Fig.3.3은 motor driver, Fig.3.4는 motion board, Fig.3.5는 motor controller의 사진이다.

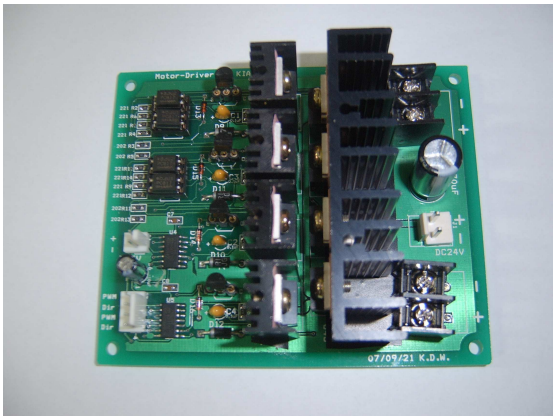


Fig. 3.3 Motor driver

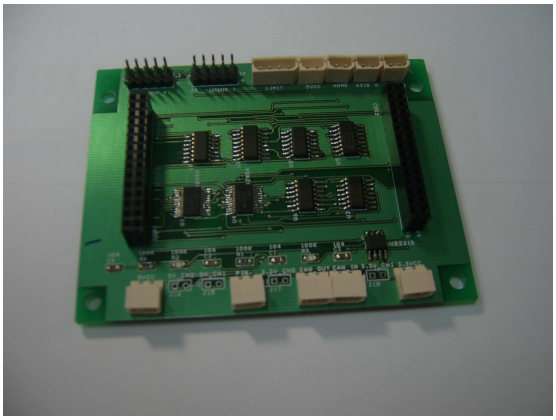


Fig. 3.4 Motion board

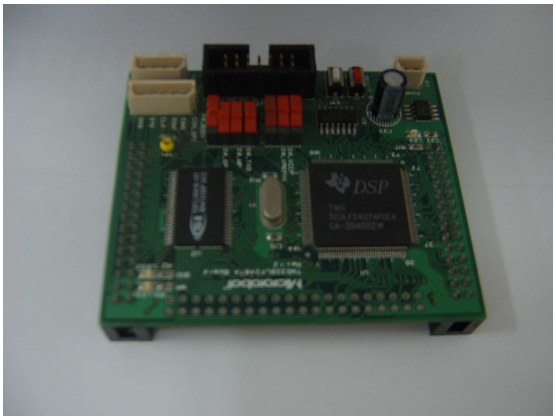


Fig. 3.5 Motor controller

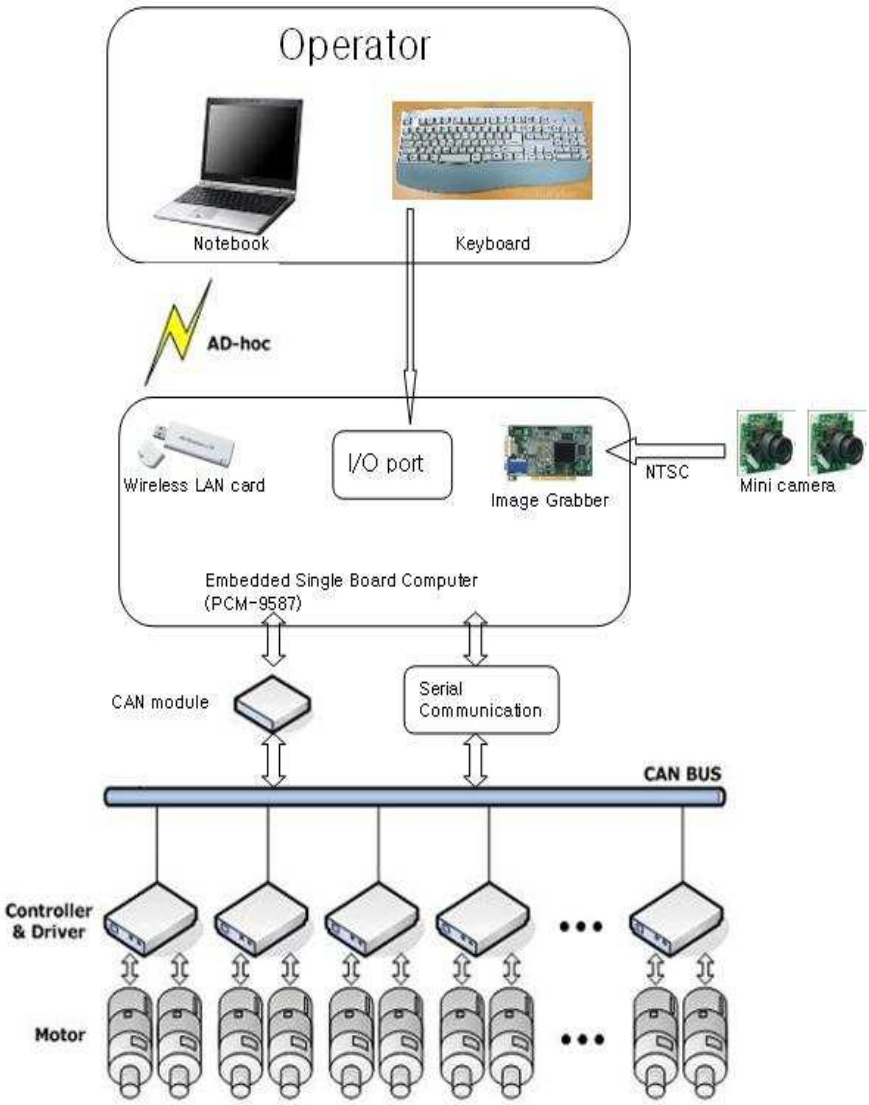


Fig. 3.6 Total control system of KUBIR III

3.4 KUBIR III의 주 제어기 구조

본 연구에서는 KUBIR III의 복잡한 구조와 많은 센서의 입력과 stereo vision system의 적용 그리고 영상처리를 위해서는 주 제어기는 computer의 기능을 가져야 한다고 생각되었다. 그리하여 로봇에 탑재할 수 있는 소형 embedded single board computer를 선정하였다.

KUBIR III의 주 제어기인 embedded computer에는 이미지 그래버(image grabber)가 장착되어 있어 CCD 카메라로부터 입력 받은 영상 정보를 획득 하고 영상 처리(digital image processing) 할 수 있으며, 내장된 시리얼 통신 포트와 CAN 통신 모듈을 이용하여 관절 구동기나 센서로부터 여러 가지 정보를 얻을 수 있다. 또한, 무선 랜 카드(wireless LAN card)를 이용하여 KUBIR III의 현재 정보를 외부로 전송 할 수 있으며, 반대로 외부로부터 제어 명령을 수신하여 명령에 따른 동작을 할 수 있다.

KUBIR III의 주 제어기의 운영체제(operating system)로는 외부에 추가 될 하드웨어의 인식률과 소프트웨어 호환성, 디버깅이 편리하고, 내부 프로그램의 코딩(coding)에 용이한 Microsoft 사의 'Windows XP'를 사용하였다.

사용된 주 제어기는 Advantech사의 PCM-9582를 사용하였다. Fig.3.7은 PCM-9582의 다이어그램이다[12].

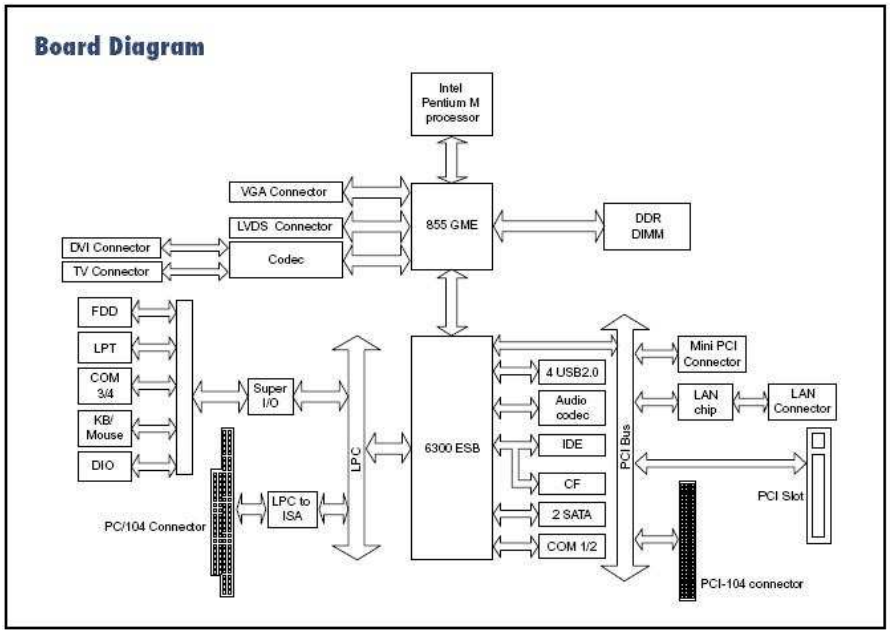


Fig. 3.7 Board diagram of PCM-9582

Fig. 3.8은 개발중인 KUBIR III의 등에 부착된 embedded computer의 모습이다.

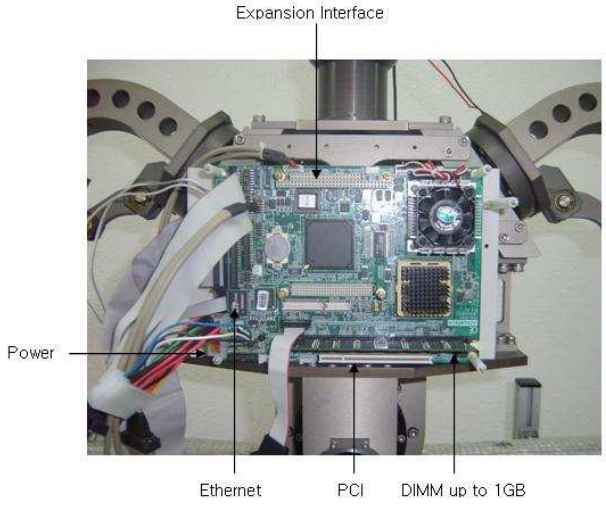


Fig. 3.8 Main controller of KUBIR III

Fig.3.9와 Fig.3.10은 embedded computer에 이미지 그래버 (image grabber)와
하드 디스크를 장착시킨 모습이다.

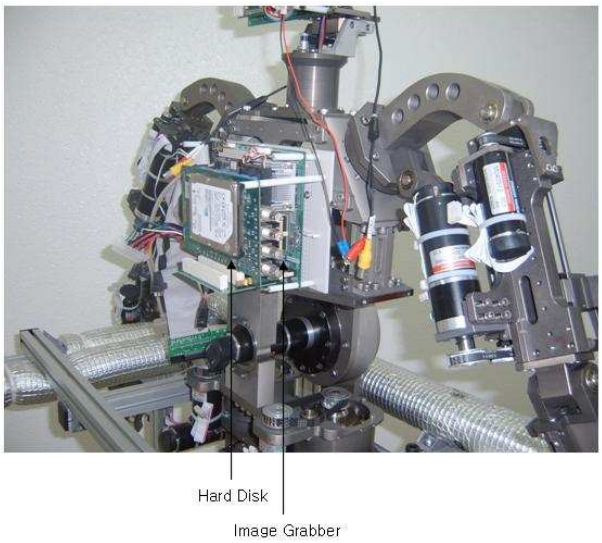


Fig. 3.9 Loaded hard disk and image grabber on embedded computer

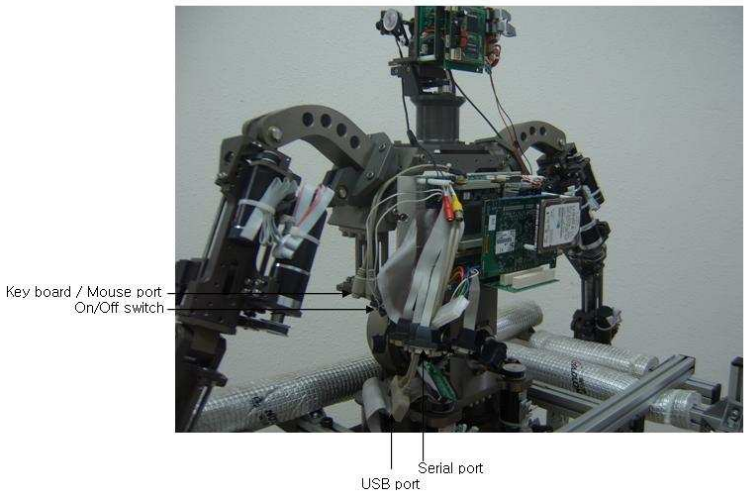


Fig. 3.10 Loaded hard disk and image grabber on embedded computer

또한, KUBIR III는 제어부와 모터 전원을 분리 함으로써 모터의 비정상적인 동작과 과 부하의 상황에서 발생하는 cut off 상태에서 제어부의 안전을 유지하였다.

Fig.3.11은 주 제어기와 카메라, KUBIR III의 머리부의 컨트롤러 구동을 위하여 베터리를 연결시킨 모습이다.

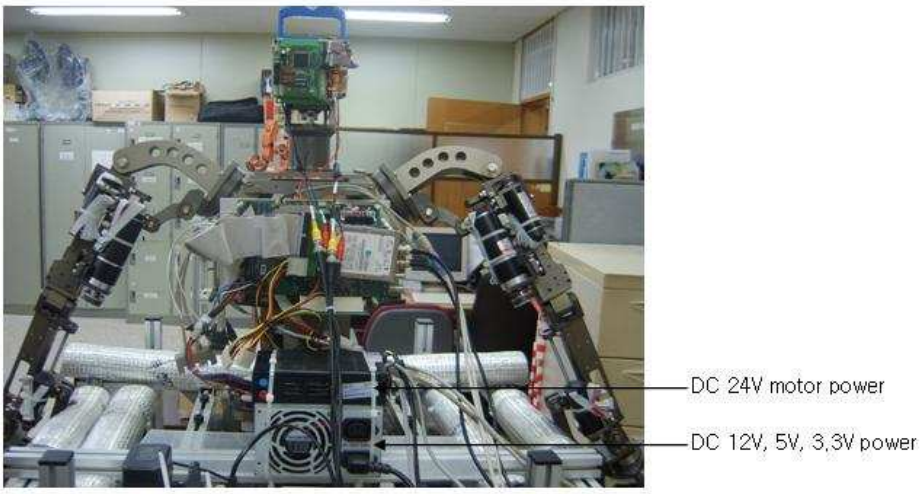


Fig. 3.11 Power supply on the back of KUBIR III

제 4 장 로봇 보행의 3D 시뮬레이션

4.1 3D 시뮬레이터 개발 환경

본 연구에서 개발된 보행로봇의 전체 하중이 100kg 정도이다. 그리고 로봇의 움직임에 있어서 고도의 안정성이 요구되고 전체 시스템의 손실을 막기 위해서는 시뮬레이션이 우선적으로 필요하다.

실제와 똑 같은 결론을 도출하기 위하여 시뮬레이션에서도 실제와 같이 디자인 하였다. 개발된 시뮬레이션 프로그램은 Microsoft Visual C++ .NET 컴파일러를 사용하였고, 3D 그래픽환경을 만들기 위하여 Silicon Graphics 사(SGI)의 OpenGL(open source graphics library)를 사용하였다. OpenGL이란 그래픽 하드웨어 제어를 위한 소프트웨어 인터페이스라고 할 수 있으며, 높은 이식성과 빠른 실행 속도를 가진 3D 그래픽 & 모델링 라이브러리이다[13].

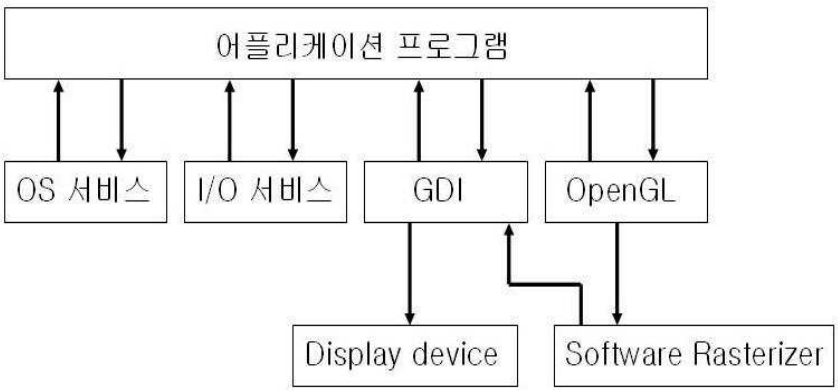


Fig. 4.1 Public application program of OpenGL

위의 Fig. 4.1은 윈도우 프로그램에서 3D 디스플레이 되기 위한 일반적인 OpenGL의 구현을 설명한다.

4.2 3D 시뮬레이터 변환 과정

본 연구의 보행 로봇 시뮬레이션은 CATIA 3D 프로그램의 설계 파일을 3D MAX 프로그램으로 import 하여 AutoCAD 파일로 변환하였다. AutoCAD 파일로 변환한 파일을 다시 MilkShape3D 파일로 export하여 최종적으로 MilkShape3D 프로그램으로 .txt 파일로 변환하였다.

CATIA 3D 설계 파일의 각 부분들을 변환하고 OpenGL의 라이브러리 함수를 이용하여 관절들을 연결하였다. 위와 같이 함으로써 3D display가 가능하게 되었고 마지막으로 Visual C++ .NET 컴파일러를 이용하여 시뮬레이션을 구현하였다. 아래의 Fig. 4.2는 CATIA 설계 파일을 Visual C++.NET으로 변환한 과정을 나타내고 있다.

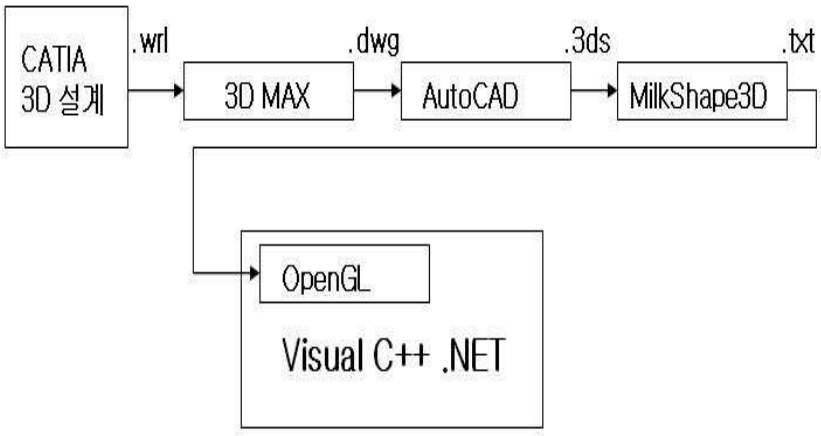


Fig. 4.2 File conversion process for 3D simulator

4.3 3D 시뮬레이터

아래의 Fig. 4.3은 머리 2 자유도, 상체 13 자유도, 하체 12 자유도를 표현 가능한 실제 로봇의 모델을 3D 프로그램으로 나타내었다.

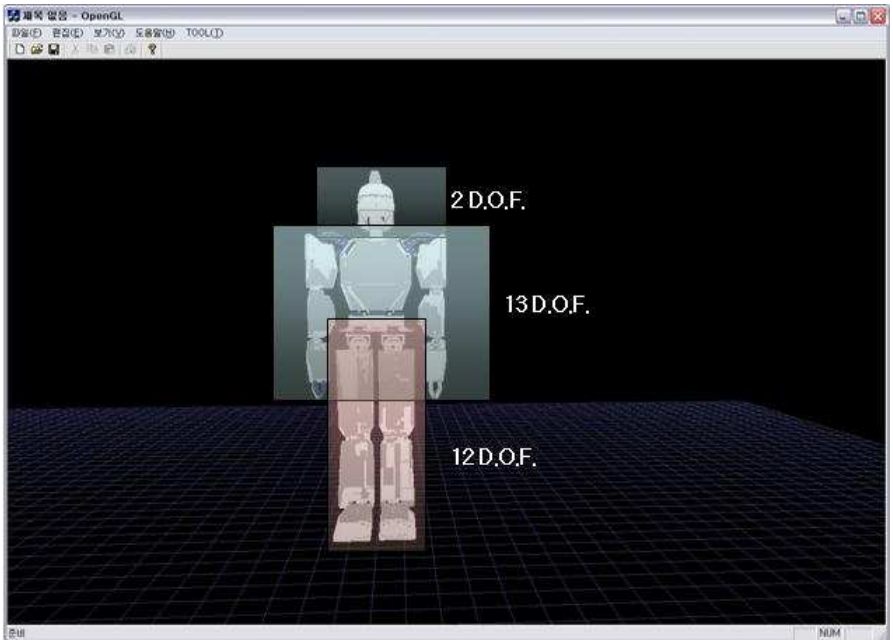


Fig. 4.3 Design of simulator of KUBIR III

아래의 Fig. 4.4는 상체의 모터 값과 다리부분의 모터 값을 이용하여 로봇의 움직임을 형상화 하여 보행동작을 예측한 시뮬레이터이다.

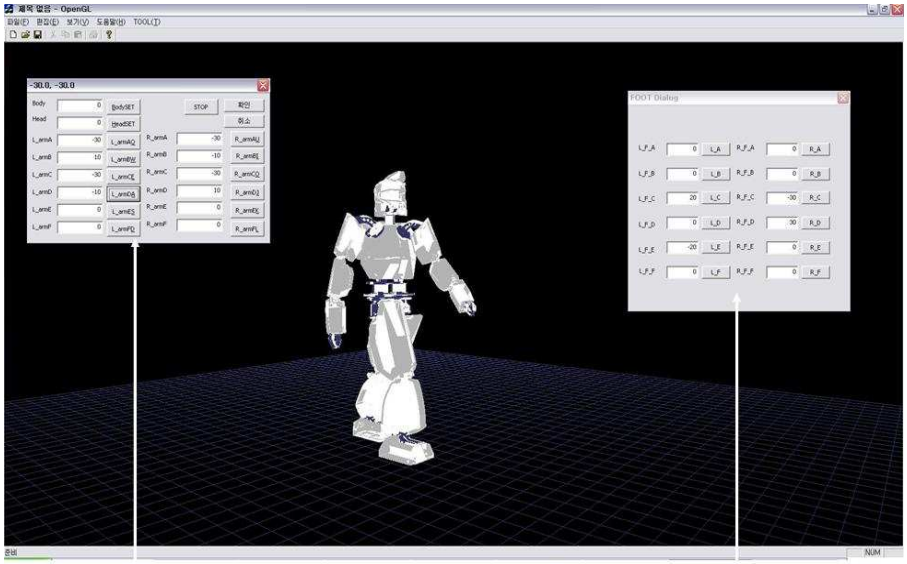


Fig. 4.4 Walking simulator of KUBIR III

4.4 3D 시뮬레이터 검증

본 연구에서 개발된 시뮬레이터는 로봇에서부터 떨어진 타겟까지의 거리를 로봇의 팔이 도달 할 수 있는지에 대한 시뮬레이터이다. 실제로는 로봇의 팔이 다다를 수 없는 경우에 있어서는 로봇이 이동하여 스테레오 비젼 시스템으로부터 거리를 측정한 값이 주 제어기에서 실시간으로 트래킹하여 팔이 타겟까지 다다를 수 있는 가능성을 확인, 반복 작업 해야 한다. 본 시뮬레이터는 순기구학과 역기구학의 검증에 대해 확인 하였고, 각 관절의 각도를 측정하여 실제의 로봇의 동작을 예측 가능하게 되었다.

아래의 Fig. 4.5와 4.6은 로봇의 팔이 타겟을 잡는 그림이다.

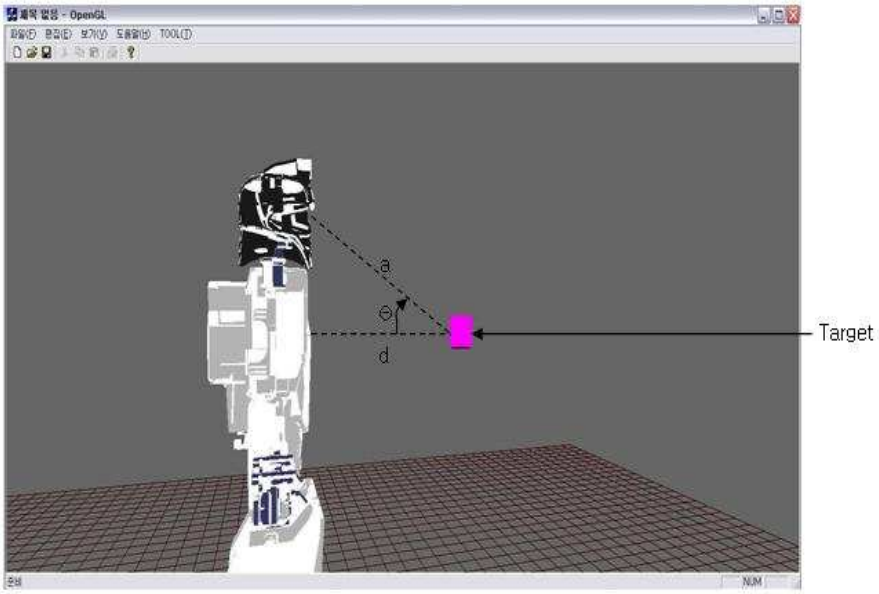


Fig. 4.5 Simulator of catch the target

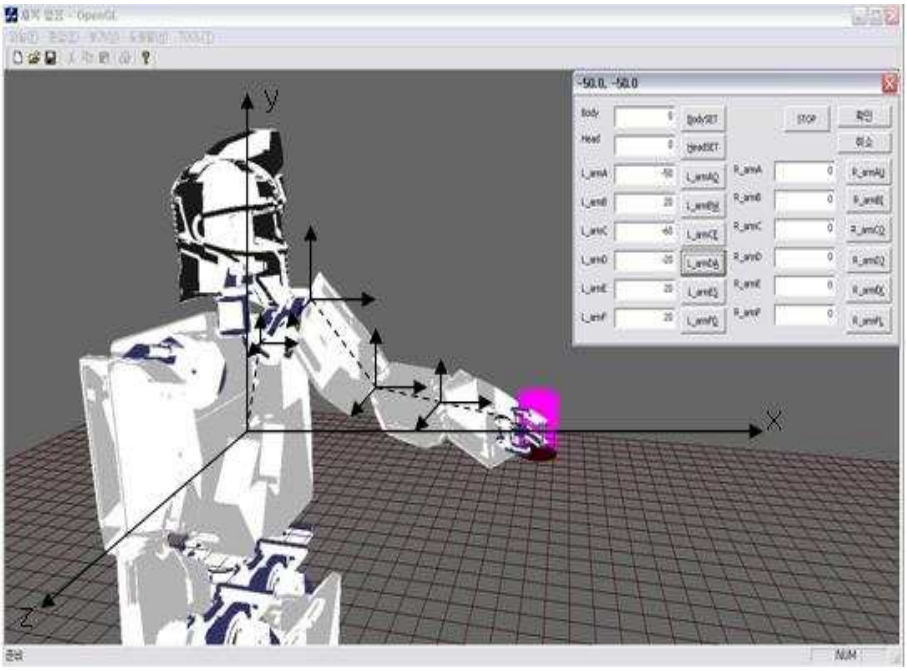


Fig. 4.6 Simulator of catch the target

본 연구에서의 시뮬레이터는 각 관절의 각도에 따라 위치 값이 정해지게 된다. 개발한 시뮬레이터의 검증을 위해서 실제 로봇의 움직임과 비교해 보았다.

먼저 Fig. 4.7와 Fig. 4.8은 시뮬레이터의 각도에 따른 움직임을 보여주고 있다.

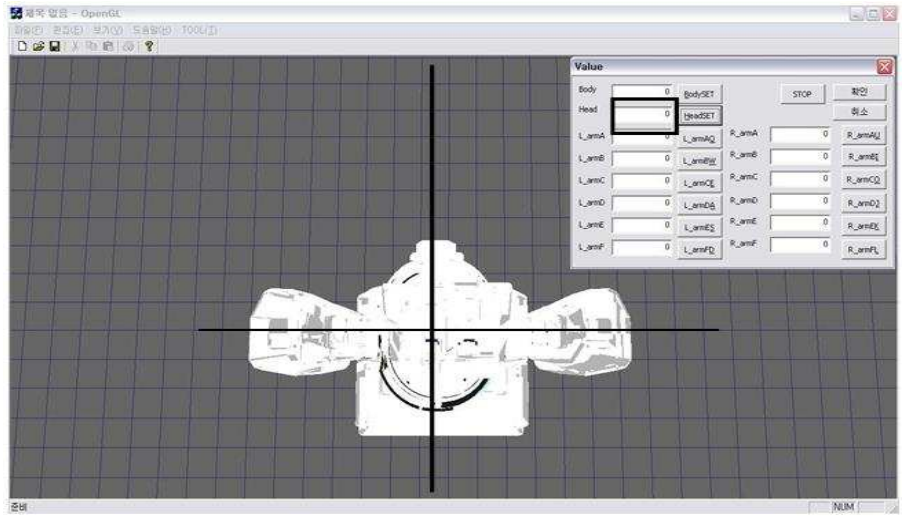


Fig. 4.7 Verification of simulator at 0 degree

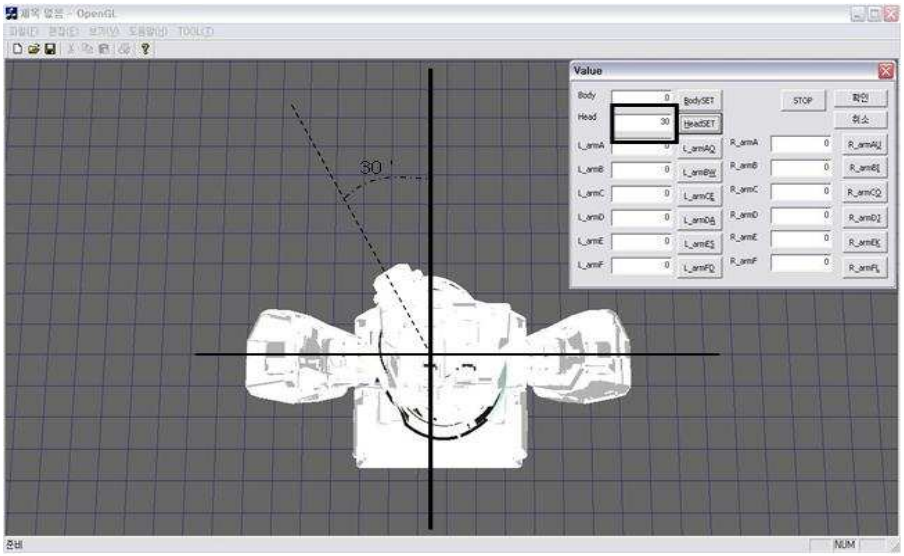


Fig. 4.8 Verification of simulator at 30 degree

아래의 Fig. 4.9은 시뮬레이터와 실제 KUBIR III의 움직임을 측정한 것이다.

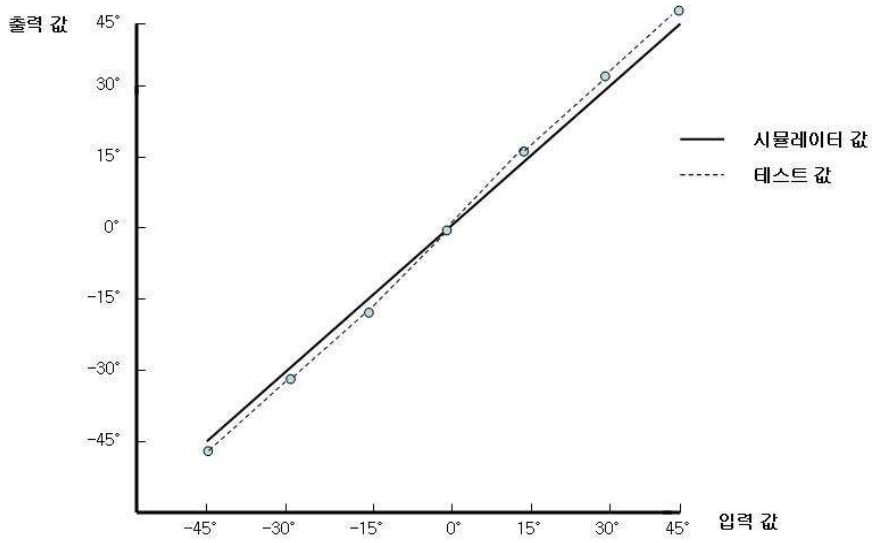


Fig. 4.9 Comparison between 3D simulator and experimental test

위의 그래프와 같이 시뮬레이터의 값과 테스트의 값이 약 2~3도 정도 오차가 난다는 걸 알 수 있다. 실제 테스트를 할 때에는 엔코더 값을 각도로 변환하여 입력하였기 때문에 엔코더, 감속기, 볼스크류, PID 알고리즘 등의 프로그램에서의 복잡한 계산식의 오차로 인하여 시뮬레이터와는 약간의 오차가 발생하였다.

이러한 문제점을 해결하기 위하여 안정된 통신 체계와 기구학에서의 정확한 계산 결과, 데이터 등의 확보가 필요하다.

제 5 장 Stereo Vision System을 이용한 영상처리

일반적인 로봇의 센서로는 광(optical), 자기(magnetic), 온도(thermal), 압력(pressure), 진동(vibration), 영상(vision) 등이 있다. 위의 센서들은 각 사용분야에 장 단점이 있으며, 이중에서도 본 연구에 개발된 KUBIR III의 눈에 해당하는 센서로는 vision이 가장 잘 어울릴 것이다. 본 연구에 사용된 stereo vision system은 사람의 눈으로 바라 보는 것처럼 인식되기 때문에 친 인간적이라고 말 할 수 있으나, 이를 사용함에 있어서는 어떠한 센서보다 어렵다고 할 수 있다[14].

사물의 인식을 위한 디지털 영상처리(digital image processing)는 현재까지도 개발의 과정이 느린 편이며, 광 범위 한 이미지의 메모리사용, 연산 속도에 따른 프로세서의 반응, 영상 처리와 모터 움직임간의 실시간성을 맞추기가 상당히 어렵다. 하지만, 신뢰 할 수 있는 프로그램의 개발과 하드웨어의 향상으로 많은 발전이 있었으며, 이를 KUBIR III에 접목 시키고자 하였다[15].

stereo vision system을 이용하기 위하여 본 연구에서는 CCD(charge-coupled device camera) 카메라와 이미지 프레임 그래버(image frame grabber)가 필요하다.

5.1 Image Frame Grabber의 구조 및 구성

본 연구에서 사용한 이미지 프레임 그래버는 Matrox사에서 개발한 Morphis라는 제품을 사용하였다. 아래의 Fig. 5.1은 Morphis의 내부 구조이다[16].

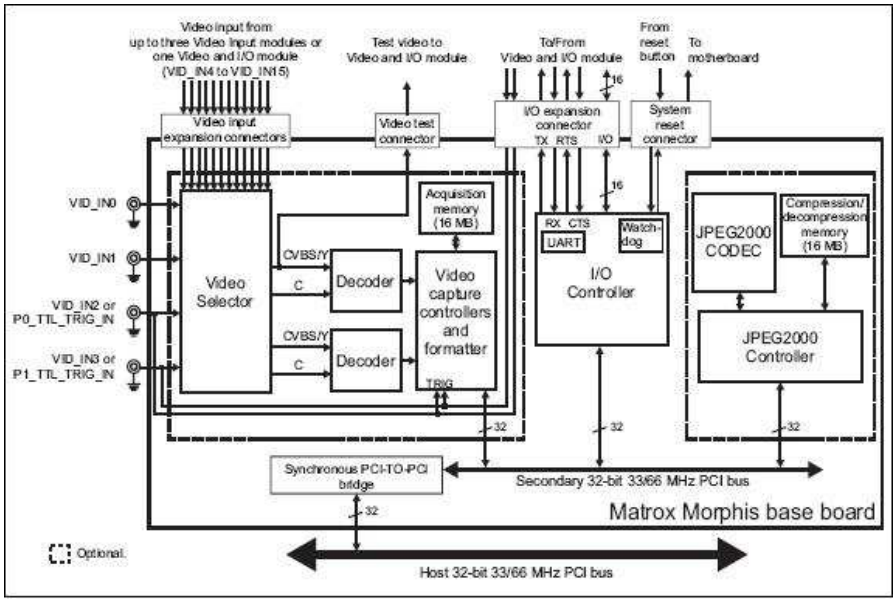


Fig. 5.1 Internal structure of the image grabber 'Morphis'

이미지 프레임 그래버는 무엇보다도 동기분리(sync separation) 회로가 입력된 영상신호(video signal)로부터 동기신호(synchronization pulses)를 분리한다. 영상신호는 그래버로 연입되기 시작한 시점으로부터 3 frame time 정도 지난 시점부터 안정적인 그래버의 동작을 보장할 수 있다. 따라서 영상 신호가 하나의 신호원으로부터 다른 신호원으로 외부에서 교체 되었을 때, 즉, 여러 개의 카메라로부터 영상을 입력 받을 때 그래버에 문제를 야기 시킬 수 있다. 이러한 신호원 교체에 따른 시간적 문제는 수평 동기신호(H-Sync), 수직 동기 신호(V-Sync)를 외부에서 각 카메라로 공급하여 여러 개의 카메라와 그래버가 동기화 됨으로 해결된다.

프래임 그래버와 카메라는 최고의 화음을 이룰 수 있어야 한다. 머신 비젼용 그래버와 멀티미디어용 그래버는 그 활용을 위하여 설계단계에서부터 다르다.

낮은 속도의 BUS(ISA)에서는 그래버내에 이미지 메모리를 필요로 하지만, PCI와 같은 빠른 속도의 BUS에서는 수 KByte의 작은 FIFO(first-in first-out) 버퍼만 있으면 된다. 영상신호가 그래버에 입력된 후 3 이미지가 초기화 하는데 필요하다. 이러한 불필요한 시간을 단축하기 위해서는 각 카메라등의 동기화가 필요하다.

RGB용 그래버에는 3개의 독립적인 A/D기능을 보유하고 있으므로 스테레오 비젼이나 고속의 듀얼 채널 인풋(dual channel input)에

활용할 수 있다. Fig.5.2는 본 연구에 사용된 stereo vision system 이다[17].

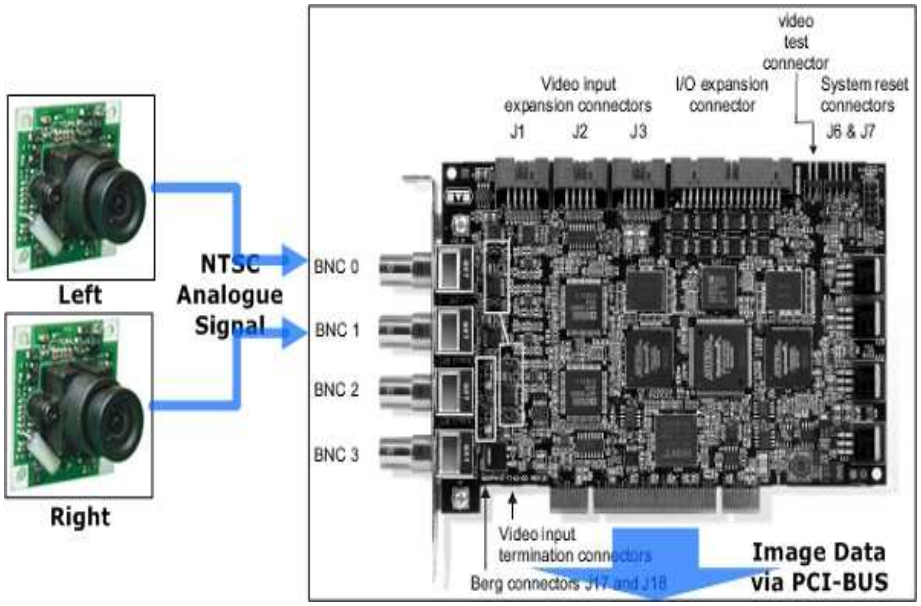


Fig. 5.2 Photo of stereo vision system

5.2 KUBIR III의 CCD 카메라의 구조와 기능

CCD는 빛을 전기 신호로 변환하는 6mm * 4mm 크기의 광전변환 센서이다. 구조는 빛을 전기적 신호로 변환하는 '광전변환 부분'과 변환된 전하를 저장했다 전송하는 '전송 부분'으로 나뉜다. 현재 비디오 카메라, 팩스, 스캐너 등의 사무기기에 널리 이용되고 있다.

최근 필름 없이 영상이 기록되는 제 3 세대 카메라로 각광받고 있는 디지털 카메라에서도 CCD가 핵심적인 역할을 담당하고 있다. 광학 카메라의 필름에 해당하는 것이 바로 CCD와 메모리 카드이기 때문이다. 렌즈로부터 들어온 빛의 세기는 먼저 CCD에 기록된다. 이 때 촬영된 영상의 빛은 CCD에 붙어 있는 RGB색 필터에 의해 각기 다른 색으로 분리된다. 분리된 색은 CCD를 구성하는 수 십만 개의 광 센서에서 전기적 신호로 바뀐다. CCD에서 나온 아날로그 신호는 0과 1의 디지털 신호로 변환되어 영상 신호가 만들어지고, 액정 화면을 통해 나타난다. 신호 처리기에서 압축된 영상 신호는 메모리 카드에 기록되어 출력이 가능하게 된다[18]. 아래의 Fig.5.3

은 본 연구에서 사용된 카메라의 사진이고 사양은 Table 5.1과 같다.

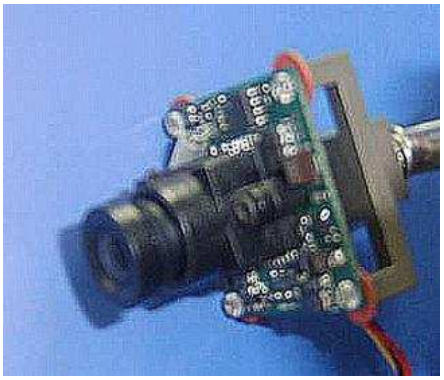


Fig. 5.3 Photo of CCD camera

Table 5.1 Specification for TCC-3232H

Specifications	TCC-3232H
Image Sensor	1/3" SONY Hi-Resolution Super HAD CCD
Dimensions	32 * 32mm Size
Resolution	420TV Lines
Minimum Illumination	0.3Lux / F2.0

5.3 Stereo Vision에서의 거리측정

영상 면에서 물체 표면까지의 거리 w를 깊이(depth)라 부른다. 한 개의 영상 점은 깊이를 단일하게 결정지어 줄 수 없다. 깊이는 스테레오 영상에 의하여 산출될 수 있다. 이것은 물체의 한 점에 대한 영상을 두 개 얻어서 이것으로부터 공간 상의 한 점을 정확히 산출하는 기법이다[19].

Fig.5.4에 도시한 바와 같이 한 개의 물체의 점을 두 개의 카메라를 이용하여 두 개의 영상 면 위에 따로따로 얻는 것을 스테레오 영상이라 한다. 두 카메라 렌즈의 중심과 중심을 연결한 선분을 기저선이라 하고 그 거리를 B라 하자. 두 카메라의 초점 거리를 모두 a라 하자. 두 카메라는 동일하다고 가정하고 각 카메라의 x, y축과 물체의 x, y축은 평행 하다고 하자.

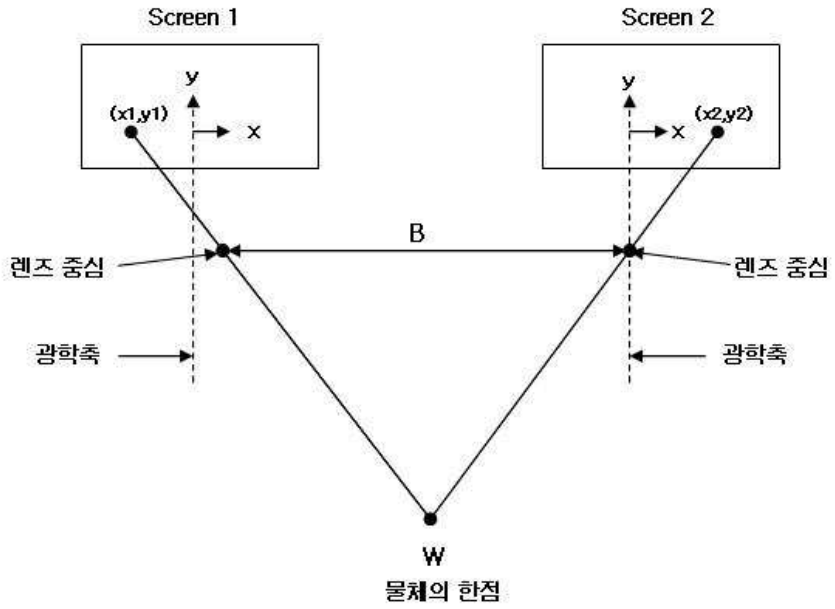


Fig. 5.4 Model of stereo vision processing

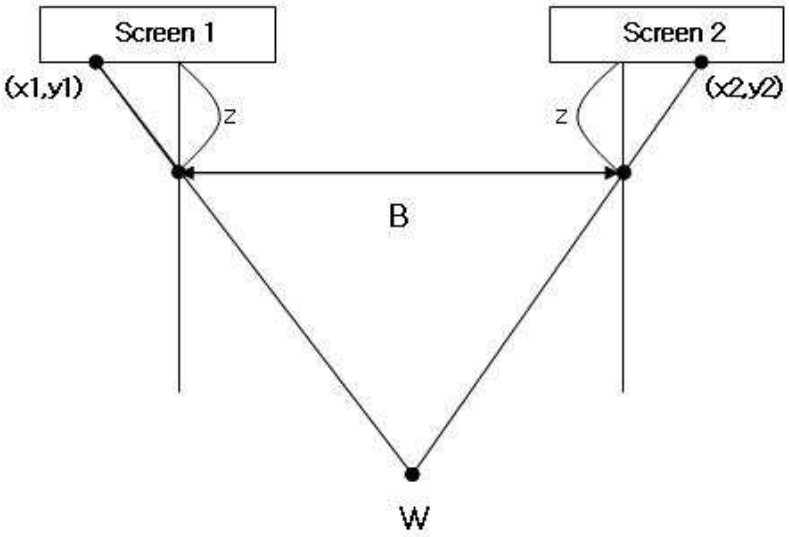


Fig. 5.5 Section of stereo vision processing model

Fig.5.5에서 보는 바와 같이 물체의 한 점에 대한 첫째 영상 면의 한 점을 (x1, y1, 0)이라 하자. 그리고 물체의 좌표계를 첫번째 영상의 좌표계와 동일한 것을 사용했을 때 그 점의 좌표를 (a1, b1, c1)이라 하자. 그러면 투사 변환을 이용하여

$$x_1 = \frac{z a_1}{z - c_1}$$

$$y_1 = \frac{z b_1}{z - c_1}$$

이를 (a1, b1)에 대해 다시 쓰면

$$a_1 = \frac{x_1 (z - c_1)}{z}$$

$$b_1 = \frac{y_1 (z - c_1)}{z}$$

을 얻을 수 있다.

같은 식으로 같은 물체의 점에 대한 둘째 영상 좌표계의 영상 점의 좌표를 (x2, y2)이라 하고, 둘째 영상 좌표계에서의 물체 점의 좌표를 (a2, b2, c2)라 하면

$$a_2 = \frac{x_2(z - c_2)}{z}$$

이다. 첫 영상 면의 x – y 좌표계는 둘째 영상 면의 x – y 좌표계와 평행하지만 x 축상으로 렌즈의 거리 B 만큼 떨어져 있으므로

$$a_2 = a_1 + B$$

$$c_2 = c_1 = c$$

이다. 이를 사용하면

$$a_1 = \frac{x_1}{z}(z - c)$$

$$a_1 + B = \frac{x_2}{z}(z - c)$$

를 얻는다. 양변을 각각 감산하여

$$c = z - \frac{zB}{x_2 - x_1}$$

를 얻는다. 이것은 같은 물체의 점에 대한 두 영상의 x좌표 x1, x2를 알면 깊이 c를 구할 수 있음을 시사한다.

그러나 이것을 얻는데 있어서 가장 어려운 문제는 두 영상으로부터 어떤 점이 동일한 물체점의 영상 점인지를 알아내는 것이다. 이것을 대응점 문제(correspondence problem)라고 부른다.

이러한 두 점은 매우 가깝게 놓여 있는 것이 보통이므로, 한 영상의 국소부분과 다른 영상의 국소부분을 분리하여 상관계수법(correlation technique)에 의하여 적합성 여부를 판별하는 것도 그 한 방법이다.

제 6 장　영상처리를 이용한 물체 인식 실험

6.1 비주얼 서보잉

비주얼 서보잉(visual servoing)이란 카메라 영상에서 얻은 비젼 정보를 피드백 받아 close-loop 제어를 하는 분야를 말한다. 이 비주얼 서보잉 이라는 말은 1979년 Hill and Park에 의해 처음 소개되어 현재까지 로봇 비젼 등 여러 분야에서 널리 사용되고 있다[20].

비주얼 서보잉은 카메라 위치에 따라, 그리고 제어 신호에 따라 분류가 되어진다.

1) 카메라 위치에 따른 분류

비주얼 서보잉은 카메라의 위치에 따라 eye to hand 방식과 eye in hand 방식으로 분류된다[21].

가) Eye to hand 방식

eye to hand 방식은 Fig. 6.1과 같이 카메라가 고정되어 있고 스테이지나 로봇의 end effector가 움직이는 방식이다.

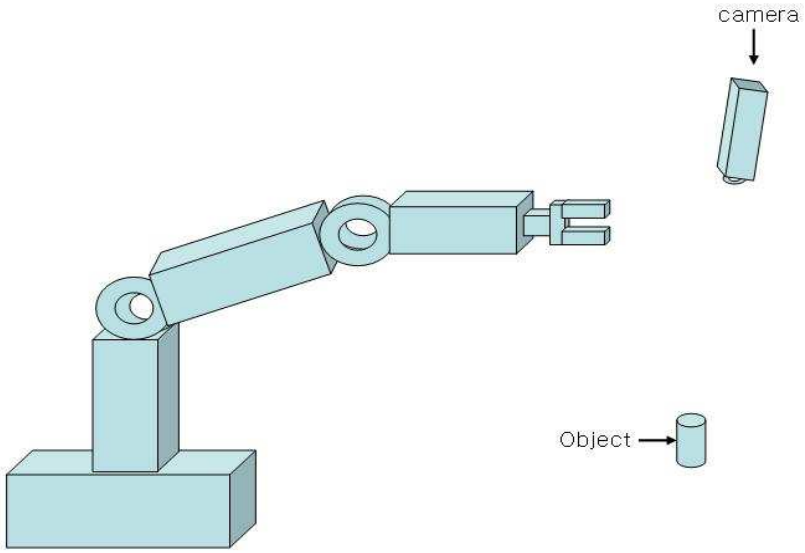


Fig. 6.1 Eye to hand in the visual servoing

나) Eye in hand 방식

eye in hand 방식은 Fig. 6.2와 같이 카메라가 로봇의 end effector에 고정 되어 원하는 위치로 카메라가 직접 이동하는 방식이다.

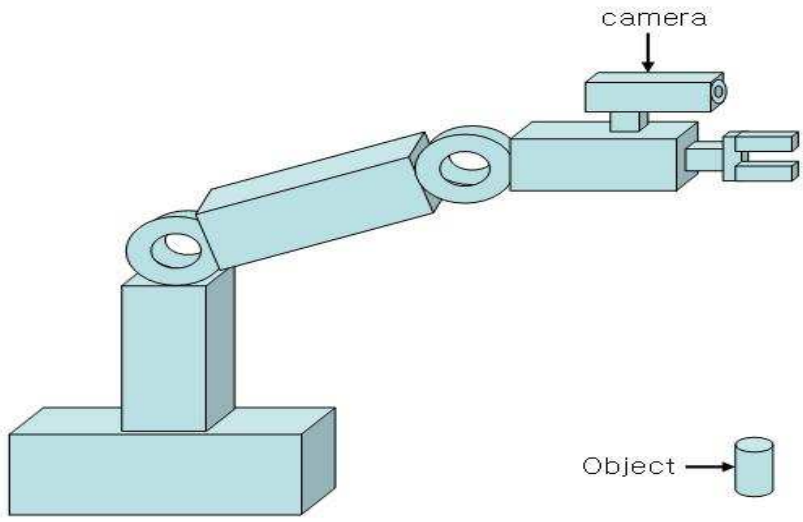


Fig. 6.2 Eye in hand in the visual servoing

2) 제어 신호에 따른 분류

비주얼 서보잉은 제어 신호에 따라 위치 기반 비주얼 서보잉(position – based visual servoing)과 영상기반 비주얼 서보잉(image-based visual servoing)으로 분류된다.

가) 위치기반 비주얼 서보잉

위치기반 비주얼 서보잉은 카메라에서 얻은 영상에서 특징치를 추출하여 그 특징치로 물체의 위치을 예측하여, 여기서 얻어진 물체의 위치와 원하는 물체의 위치와의 에러를 제어 신호로 하여 카메라나 물체를 이동하는 방법이다.

위치기반 비주얼 서보잉은 영상의 노이즈에 강건하다는 장점이 있지만 물체의 위치를 예측하기 위해서는 물체에 대한 정보가 있어야 하고 카메라 calibration 에러에 민감하다는 단점이 있다.

나) 영상기반 비주얼 서보잉

영상기반 비주얼 서보잉은 카메라에서 얻은 영상에서 특징치를 추출하고 그 특징치에서 위치를 예측하지 않고 이미지 평면상의 특

징치 점이 비주얼 서보잉의 제어 신호로 입력되는 방법이다.

영상기반 비주얼 서보잉은 이미지 평면상의 특징치의 에러가 제어 신호이므로 물체에 대한 CAD 정보가 필요 없고 카메라나 스테이지의 calibration 에러에 강간하다는 장점이 있지만, 광 축 방향으로의 거리를 예측해야 하고 local minima에 빠질 수 있다는 단점이 있다.

6.2 거리 측정 실험

본 연구에서의 실험은 원하는 대상에 대한 좌, 우 카메라에 나타난 화면의 위치 오차를 이용하여 거리를 측정하였다. 특정한 물체뿐만 아니라 원하는 대상을 선정하여 거리를 추출 가능하게 프로그램하였다. 스테레오 카메라를 사용하지 않고 각 각의 CCD 카메라를 이용하여 스테레오 비젼시스템을 구현하였으므로 약 간의 오차가 발생된 것을 확인 할 수 있었으며, 3D 영상을 획득하기가 쉽지 않음을 알게 되었다.

아래의 Fig. 6.3은 본 연구에서 개발된 스테레오 비젼 영상 처리 화면이다. 아래의 사진은 하나의 카메라에서 640 * 480의 픽셀로 영상을 출력하였고, 사진에서 보는 것과 같이 물체의 중심이 오른쪽 카메라 영상의 출력화면과 왼쪽 카메라 영상 출력화면이 다르다는 것을 알 수 있다. 각각의 화면에 출력된 화면의 중심과 물체의 중심간의 거리를 측정하여 카메라와 물체간의 거리를 측정 할 수 있었다.

또한, 하나의 GUI 프로그램 내에서 영상처리와 모터를 구동 가능하게 프로그램 되어 있다. KUBIR III의 상체의 모습을 본 떠 만든 GUI 프로그램은 각 각의 관절의 모터의 엔코더 값을 입력하여 수동적으로도 움직일 수 있게 만들었으며, 향후 KUBIR III의 하체에 대해서도 추가적으로 프로그램 할 계획이다.

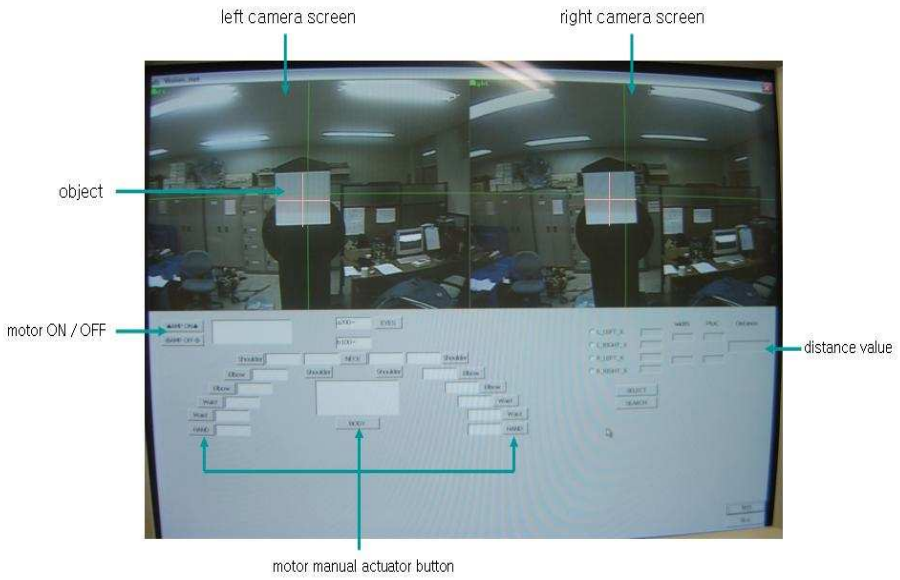
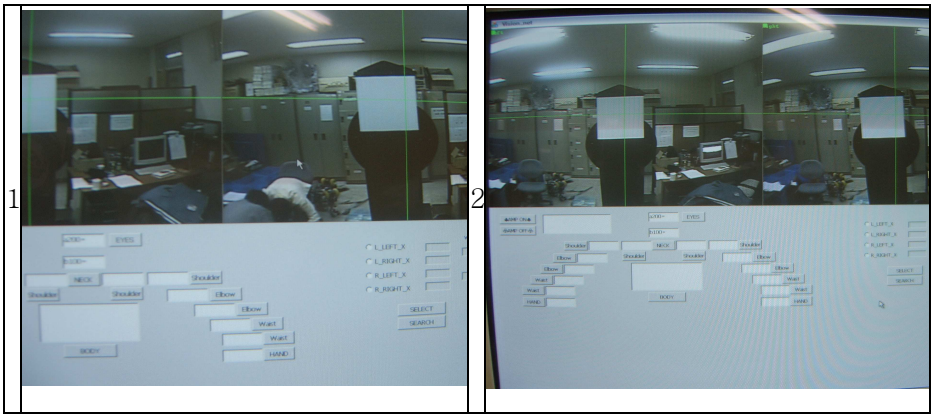


Fig. 6.3 Image processing of the vision system

아래의 Fig.6.4는 거리 측정 실험 사진이다. 오른쪽 카메라에 대한 물체의 좌표를 알 수 있으며, 왼쪽 카메라에 대한 물체의 좌표를 구한 다음, 거리 측정 비례식을 이용하여 최종적으로 물체와 카메라 사이의 거리를 알 수 있다.



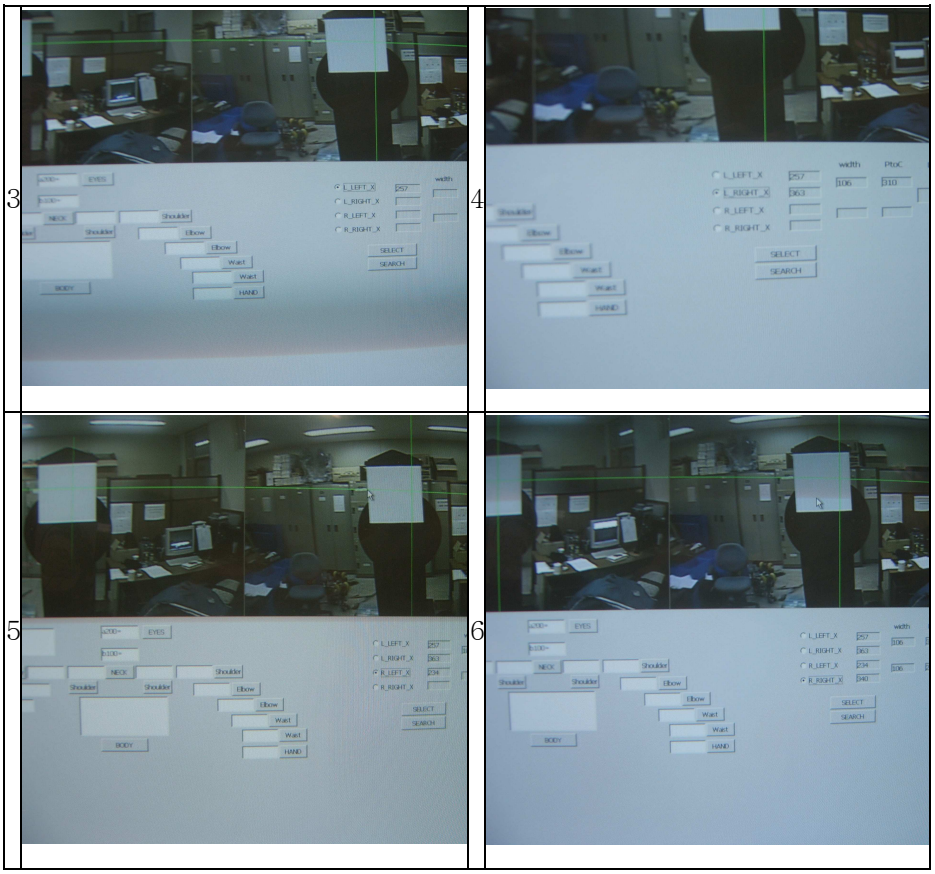


Fig. 6.4 Experiment of distance measurement

아래의 사진 Fig.6.5는 계산된 결과와 Fig.6.6은 실제 거리 측정한 사진이다.

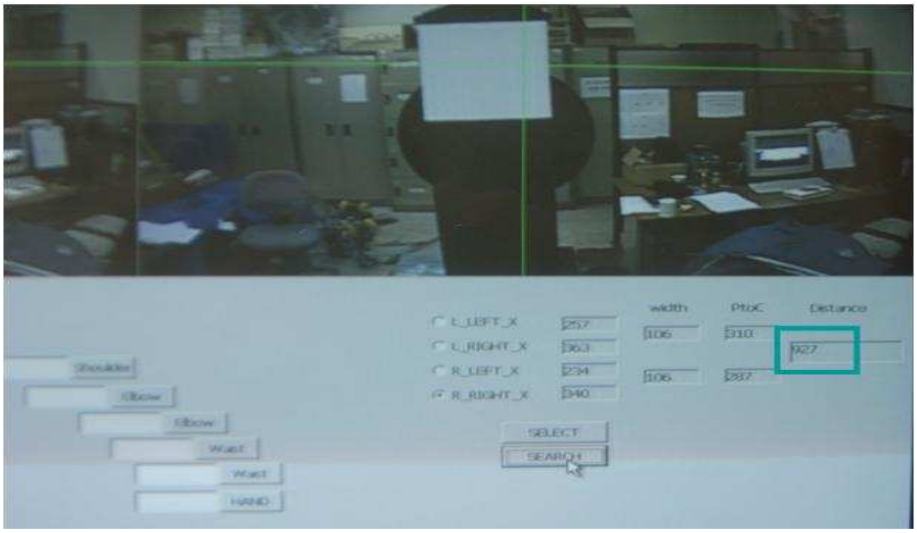


Fig. 6.5 Result of distance using program

Fig. 6.6 Distance between camera and object

위의 실험에서 프로그램에서의 거리와 실제 거리의 오차는 약 0.3cm 정도 나타났으며, 이 오차는 카메라의 렌즈의 초점과 특성에 따라 약간의 차이가 있는 것으로 나타났다.

본 연구에서 KUBIR III는 움직이는 이족 보행 로봇이므로 처음의 오차는 움직이면서 그 물체에 도달할 때까지 계속해서 에러를 보정하는 알고리즘을 이용하여 실제 그 물체를 잡을 때에는 에러가 점차 줄어들 것으로 예상한다.

이와 같은 실험을 반복하여 물체의 거리를 검출하였는데, 또 하나의 오차의 경우로는 카메라 렌즈의 길이에 따른 오차이다. 아래의 Fig.6.7은 카메라 렌즈의 길이를 5cm와 5.5cm로 설정하고 물체 거리 산출 방정식에 대입하였다. 그 결과를 나타내고 있으며, 두 그래프의 비교 결과 렌즈의 길이를 5.5cm로 하였을 때 보다 정상치에 가까운 결과를 나타내고 있음을 확인하였다.

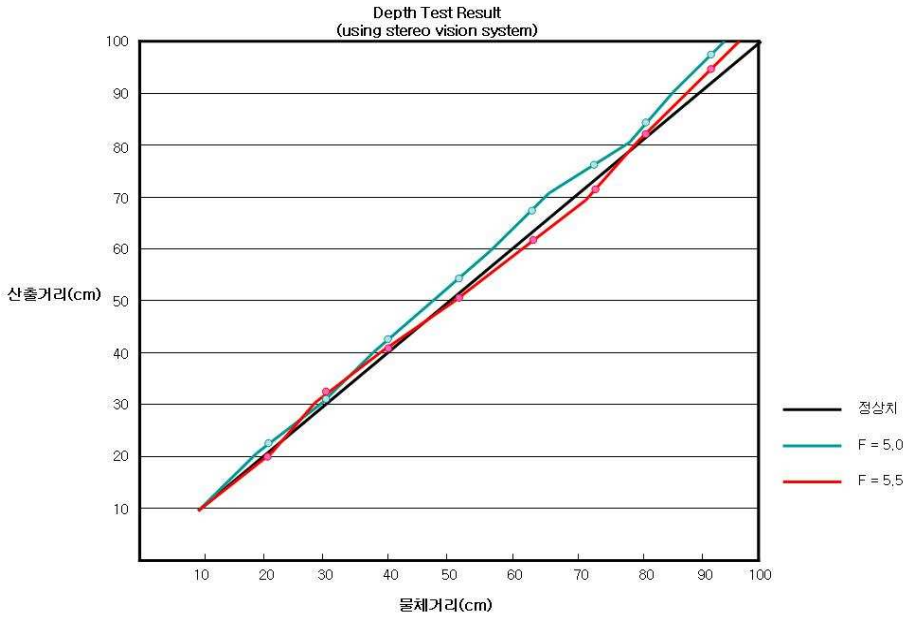


Fig. 6.7 Result of computing distance using the stereo vision system

제 7 장 결 론

본 논문에서는 한국해양대학교 휴머노이드 로봇 KUBIRIII (Korea Maritime University Biped Robot III)의 주 제어 시스템으로 PCM-9582 embedded system을 이용하여 안정된 성능과 센서의 활용, 모터 제어, 유, 무선 통신 제어 시스템을 연구하였다.

KUBIR III의 머리부분과 상체부분의 기구학 해석으로 각 각의 관절 변화의 해를 구하였으며, 2대의 소형 CCD 카메라와 이미지 그래버를 이용하여 스테레오 비젼 시스템을 구축하였다. 스테레오 비젼 시스템을 이용하여 물체와의 거리를 측정하여 기구학 해석을 이용한 관절 각의 해를 적용시켜 KUBIR III의 팔이 물체의 위치까지 도달 할 수 있도록 개발하고 있는 상태이며, 추후에 로봇의 팔에 모터 컨트롤러와 하드웨어 기반을 완성하여 테스트를 착수할 예정이다.

KUBIR III의 안전과 손실을 예방하기 위한 3D 시뮬레이터를 개발하여 로봇의 움직임을 예측 가능하게 되었고, 앞으로의 개발과정에서의 시행착오를 방지 할 수 있게 되었다.

향후 KUBIR III의 성능 개선을 위하여 주 제어기에서 처리하는 영상처리와 모터 제어, 각 각의 센서 신호등을 실시간으로 제어하기 위한 리얼타임 운영체제(real-time operation system)나 RTX 등과 같은 실시간 라이브러리의 사용이 필수적이며, 스테레오 비젼 시스템의 향상을 위하여 모듈화된 스테레오 비젼을 사용하는 것이 보다 효율적일 것이다. 또한 embedded system의 성능 업데이트와 제어 신호의 신뢰성 보완이 필요하다.

참고문헌

[1] Edited by B. K. GHOSH, NING XI, T. J. TARN, “Control in Robotics and Automation Sensor-Based Integration” Academic Press chapter 10.

[2] A.Takanishi, Y. Egusa, M. Tochizawa, M. Takeya, and I. Kato, "Realization of Dynamic Biped Walking Stabilized with Trunk Motion" RoManSy 7: Proc. Seventh CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators, A.Morecki, G.Bianchi and KoJaworek, Eds.Hermes, Paris, pp.68~79, 1990.

[3] 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.
http://asimo.honda.com

[4] Hernsoo Hahn, Youngjoon Han, “Visual Tracking of a Moving Target Using Active Contour based SSD Algorithm”, Robotics and Autonomous System SCIE(53, 265-281), Volume 54, Issue 9, Pages 719-778(30 September 2006)

[5] H. Koyasu, J. Miura , and Y. Shirai, "Recognizing Moving Obstacles for Robot Navigation Using Real-time Omnidirectional Stereo Vision", *J. of Robotics and Mechatronics*, Vol. 14, No. 2, pp. 147-156, 2002

[6] Bradley J.Nelson, Pradeep K. Khosla, "An Extendable Framework for Expectation-Based Visual Servoing Using Environment Models", IEEE trans. on Robotics and Automation, Vol. 1, pp. 184~189, 1995.

[7] Spong, Vidyasagar, "Robot Dynamics and Control", WILEY.

[8] http://freespace.virgin.net/hugo.elias/models/m_ik.htm

[9] JongMoo Choi, “Kernel Aware Module Verification for Robust Reconfigurable Operating System”, JISE, Vol. 23, No. 5, Sep. 2007.

[10] Cal Erickson, “Memory Leak Detection in Embedded System”, http://www.linuxjournal.com

[11] D. K. Kim, “A New Mobile Environment: Mobile AdHoc Networks(MANET),” IEEE vehic. Tech Soc. News, August 2003, pp. 29-35.

[12] http://www.advantech.co.kr

[13] Richard S. Wright, Jr. Benjamin Lipchak 외 저, 최현호 역, “OpenGL SUPERBIBLE 제 3 판, 정보문화사

[14] 조강현, “비젼 지능 시스템”, UUP

[15] B. K. Kim, “Development of Control System for a Humanoid Robot Arm with 12 D.O.F using the Stereo Vision System”, p.32-p.36

[16] http://www.matrox.com

[17] 죽촌유부, 정차근 역, “디지털 CCD 카메라 기술”, 미래컴

[18] http://www.newtonkorea.co.kr/

[19] 김희승, “영상인식 – 영상 처리, 컴퓨터 비젼, 패턴인식, 신경망 –“, 생능출판사

[20] Hill, J. & Park, W. (1979), Real time control of a robot with a mobile camera,*in* ‘Proceedings of the 9th International Symposium on Industrial Robots’

[21] Kyung Nam Jang, “Microassemble using Visual Servoing”, Korea Advanced Institute of Science and Technology.

부 록

OpenGL을 이용한 Visual .net 기반의 3D 시뮬레이터

헤더파일

```
// OpenGLView.h : interface of the COpenGLView class
////////////////////////////////////////////////////////////////////////
////////

#if !defined(AFX_OPENGLVIEW_H__04E132DE_F520_449C_84A9_909EEF44749D__
INCLUDED_)
#define
AFX_OPENGLVIEW_H__04E132DE_F520_449C_84A9_909EEF44749D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "ValueDlg.h"
#include "OpenGLDoc.h"
#include "model.h"
#include "FootDlg.h"

class COpenGLView : public CView
{
protected: // create from serialization only
        COpenGLView();
        DECLARE_DYNCREATE(COpenGLView)

// Attributes
public:
        COpenGLDoc* GetDocument();
        CValueDlg m_pValueDlg;
        CFootDlg m_pFootDlg;

        Model* m_test[39];
        HGLRC m_hRC; //Rendering Context
        CDC* m_pDC;  //Device Context

        BOOL InitializeOpenGL();
```

```
BOOL SetupPixelFormat();
void RenderScene();
void LeftFoot();
void RightFoot();
void LeftArm();
void RightArm();
void Body();
void Head();
void DrawGrid();

static UINT RenderProc(void* lParam);

//Position and Direction
GLfloat eye_x, eye_y, eye_z;
GLfloat center_x, center_y, center_z;

GLfloat m_xRotateAng;
GLfloat m_yRotateAng;

//Increment for keyboard control
GLfloat m_PosIncr;
GLfloat m_AngIncr;

CPoint m_MousePoint;
BOOL m_MouseCheck;

BOOL m_Draw;

GLfloat m_xPosition, m_yPosition, m_zPosition;

BOOL m_ThreadOff;
CWinThread* hRenderThread;

int L_footA;
int R_footA;
int L_footB;
int R_footB;
int L_footC;
int R_footC;
int L_footD;
int R_footD;
```

```
        int L_footE;
        int R_footE;
        int L_footF;
        int R_footF;


// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(COpenGLView)
        public:
        virtual void OnDraw(CDC* pDC);  // overridden to draw this
view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        protected:
        //}}AFX_VIRTUAL
// Implementation
public:
        virtual ~COpenGLView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
protected:

// Generated message map functions
protected:
        //{{AFX_MSG(COpenGLView)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnDestroy();
        afx_msg void OnSize(UINT nType, int cx, int cy);
        afx_msg BOOL OnEraseBkgnd(CDC* pDC);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
public:
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
        afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnRButtonUp(UINT nFlags, CPoint point);
```

```
        afx_msg void OnMouseMove(UINT nFlags, CPoint point);
        afx_msg BOOL OnMouseWheel(UINT nFlags, short zDelta, CPoint
pt);
        afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT
nFlags);
        afx_msg void OnMenuValue();
        afx_msg void OnToolFootdlg();
};
#ifndef _DEBUG  // debug version in OpenGLView.cpp
inline COpenGLDoc* COpenGLView::GetDocument()
   { return (COpenGLDoc*)m_pDocument; }
#endif

////////////////////////////////////////////////////////////////////
////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif
// !defined(AFX_OPENGLVIEW_H__04E132DE_F520_449C_84A9_909EEF44749D__I
NCLUDED_)
```

Draw 함수 구조

```
void COpenGLView::OnDraw(CDC* pDC)
{
        COpenGLDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        // TODO: add draw code for native data here
        //Clear out the color & depth buffers
        ::glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
                RenderScene();
        glPopMatrix();
        //Tell OpenGL to flush its pipeline
        ::glFinish();
        //Now Swap the buffers
        ::SwapBuffers(m_pDC->GetSafeHdc());
}
```

Rendering 함수 구조

```
void COpenGLView::RenderScene()
{
	glPushMatrix();
		glTranslatef(m_xPosition, -500.0f + m_yPosition, -
		4000.0f - m_zPosition-500);
		glRotatef(m_xRotateAng, 1.0, 0.0, 0.0);
		glRotatef(m_yRotateAng, 0.0, 1.0, 0.0);

		//between A and B frame
		glPushMatrix();
			glTranslatef(0.0, 980.0, 10.0);
			glRotatef(180, 0, 0, 1);
			glTranslatef(250, -40, -120);
			glColor3f(1.0, 1.0, 2.0);
			glScalef(800.0, 800.0, 800.0);
			//glutSolidCube(50);
			m_test[24]->render();
		glPopMatrix();
		glPushMatrix();
			glTranslatef(0, 230, 0);
			DrawGrid();
		glPopMatrix();
		glPushMatrix();
			glPushMatrix();
				Body();
			glPopMatrix();
			glPushMatrix();
				LeftArm();
			glPopMatrix();
			glPushMatrix();
				RightArm();
			glPopMatrix();
			glPushMatrix();
				Head();
			glPopMatrix();
			LeftFoot();
			RightFoot();
		glPopMatrix();
	glPopMatrix();
}
```

Create 함수구조

```
int COpenGLView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CView::OnCreate(lpCreateStruct) == -1)
		return -1;
	// TODO: Add your specialized creation code here
	//Initialize OpenGL Here
	hRenderThread = AfxBeginThread(RenderProc, this,
	THREAD_PRIORITY_LOWEST, 0, CREATE_SUSPENDED);
	hRenderThread->ResumeThread();
	m_pValueDlg.Create(IDD_VALUE, NULL);
	m_pFootDlg.Create(IDD_FOOTDLG, NULL);
	InitializeOpenGL();
	m_test[4]  = new Model("MS3D₩₩LINK_FORW.txt");
	m_test[5]  = new Model("MS3D₩₩LINK_FORW.txt");
	m_test[10] = new Model("MS3D₩₩LINK_BACK.txt");
	m_test[11] = new Model("MS3D₩₩LINK_BACK.txt");
	m_test[12] = new Model("MS3D₩₩LINK_FORW.txt");
	m_test[13] = new Model("MS3D₩₩LINK_FORW.txt");
	m_test[14] = new Model("MS3D₩₩SLD_BL_BOTTOM.txt");
	m_test[15] = new Model("MS3D₩₩SLD_BL_BOTTOM.txt");
	m_test[16] = new Model("MS3D₩₩SLD_BL_TOP.txt");
	m_test[17] = new Model("MS3D₩₩SLD_BL_TOP.txt");
	m_test[18] = new Model("MS3D₩₩SLD_BL_TOP.txt");
	m_test[19] = new Model("MS3D₩₩SLD_BL_TOP.txt");
	m_test[25] = new Model("MS3D₩₩HEAD.txt");
	m_test[26] = new Model("MS3D₩₩NECK.txt");
	m_test[27] = new Model("MS3D₩₩BODY.txt");
	m_test[28] = new Model("MS3D₩₩R_Shoulder.txt");
	m_test[29] = new Model("MS3D₩₩L_Shoulder.txt");
	m_test[30] = new Model("MS3D₩₩L_Arm_1.txt");
	m_test[31] = new Model("MS3D₩₩R_Arm_1.txt");
	m_test[32] = new Model("MS3D₩₩L_Arm_2.txt");
	m_test[33] = new Model("MS3D₩₩R_Arm_2.txt");
	m_test[34] = new Model("MS3D₩₩ELBOW.txt");
	m_test[35] = new Model("MS3D₩₩L_Arm_3.txt");
	m_test[36] = new Model("MS3D₩₩Grab_1.txt");
	m_test[37] = new Model("MS3D₩₩Grab_2.txt");
	m_test[38] = new Model("MS3D₩₩R_Arm_3.txt");
	m_test[24] = new Model("MS3D₩₩Heap.txt");
	m_test[22] = new Model("MS3D₩₩Under_Heap.txt");
```

```
        m_test[23] = new Model("MS3D₩₩Under_Heap.txt");
        m_test[21] = new Model("MS3D₩₩Upper_Thigh.txt");
        m_test[20] = new Model("MS3D₩₩Upper_Thigh.txt");
        m_test[8]  = new Model("MS3D₩₩L_Thigh.txt");
        m_test[9]  = new Model("MS3D₩₩R_Thigh.txt");
        m_test[7]  = new Model("MS3D₩₩R_Calf.txt");
        m_test[6]  = new Model("MS3D₩₩L_Calf.txt");
        m_test[2]  = new Model("MS3D₩₩Foot.txt");
        m_test[0]  = new Model("MS3D₩₩L_FootCase.txt");
        m_test[3]  = new Model("MS3D₩₩Foot.txt");
        m_test[1]  = new Model("MS3D₩₩R_FootCase.txt");
        return 0;
}
```

Destroy 함수구조

```
void COpenGLView::OnDestroy()
{
        CView::OnDestroy();
        int i;
        // TODO: Add your message handler code here
        //Make the RC non-current
        if(::wglMakeCurrent(0,0) == FALSE)
        {
                MessageBox("Could not make RC non-current");
        }
        //Delete the rendering context
        if(::wglDeleteContext(m_hRC)==FALSE)
        {
                MessageBox("Could not delete RC");
        }
        //Delete the DC
        if(m_pDC)
        {
                delete m_pDC;
        }
        //Set it to NULL
        m_pDC = NULL;
        m_ThreadOff = TRUE;
        for(i = 0; i <= 25; i++)
        {
                if(m_test[i] != NULL)
```

```
                {
                        delete m_test[i];
                        m_test[i] = NULL;
                }
        }
}
```

Size 함수구조

```
void COpenGLView::OnSize(UINT nType, int cx, int cy)
{
        CView::OnSize(nType, cx, cy);
        // TODO: Add your message handler code here
        GLdouble aspect_ratio;          //width/height ratio
        if( 0>=cx || 0>=cy)
        {
                return;
        }
        //select the full client area
        ::glViewport(0, 0, cx, cy);
        //compute the aspect ratio
        //this will keep all dimension scales equal
        aspect_ratio = (GLdouble)cx/(GLdouble)cy;
        //select the projection matrix and clear it
        //::glMatrixMode(GL_PROJECTION);
        ::glLoadIdentity();
        //select the viewing volume
        ::gluPerspective(45.0f, aspect_ratio, .01f, 10000.0f);
        //switch back to the model view matrix and clear it
        ::glMatrixMode(GL_MODELVIEW);
        //::glLoadIdentity();
}
```

Initialize 함수구조

```
BOOL COpenGLView::InitializeOpenGL()
{
        //Get a DC for the Client Area
        m_pDC = new CClientDC(this);
        //Failure to Get DC
        if(m_pDC == NULL)
        {
                MessageBox("Error Obtaining DC");
```

```
		return FALSE;
	}
	//Failure to set the pixel format
	if(!SetupPixelFormat())
		return FALSE;
	//Create Rendering Context
	m_hRC =::wglCreateContext(m_pDC->GetSafeHdc());
	//Failure to Create Rendering Context
	if(m_hRC == 0)
	{
		MessageBox("Error Creating RC");
		return FALSE;
	}
	//Make the RC Current
	if(::wglMakeCurrent(m_pDC->GetSafeHdc(), m_hRC)==FALSE)
	{
		MessageBox("Error making RC Current");
		return FALSE;
	}
	//Specify Black as the clear color
	::glClearColor(0.4f, 0.4f, 0.4f, 0.0f);
	// Display
	//Specify the back of the buffer as clear depth
	//::glClearDepth(1.0f);
	//Enable Depth Testing
	//glutInitDisplayMode(GLUT_DEPTH);
	//glEnable(GL_DEPTH_TEST);
	glShadeModel(GL_SMOOTH);
	glEnable(GL_LIGHT0);
	glEnable(GL_LIGHTING);
	glEnable(GL_COLOR_MATERIAL);
	//glMaterialf(GL_FRONT, GL_DIFFUSE, 1.0);
	return TRUE;
}
```

OpengGL setting 함수

```
BOOL COpenGLView::SetupPixelFormat()
{
	static PIXELFORMATDESCRIPTOR pfd =
	{
		sizeof(PIXELFORMATDESCRIPTOR),// size of this pfd
		1,                            // version number
		PFD_DRAW_TO_WINDOW|           // support window
		PFD_SUPPORT_OPENGL|           // support OpenGL
		PFD_DOUBLEBUFFER,             // double buffered
		PFD_TYPE_RGBA,                // RGBA type
		24,                           // 24-bit color depth
		0,0,0,0,0,0,                  // color bits ignored
		0,                            // no alpha buffer
		0,                            // shift bit ignored
		0,                          // no accumulation buffer
		0,0,0,0,                  // accumulation bits ignored
		16,                       // 16-bit z-buffer
		0,                        // no stencil buffer
		0,                        // no auxiliary buffer
		PFD_MAIN_PLANE,                         // main layer
		0,                                      // reserved
		0,0,0                     // layer masks ignored
	};
	int m_nPixelFormat = ::ChoosePixelFormat(m_pDC->GetSafeHdc(),
&pfd);
	if(m_nPixelFormat == 0)
	{
		return FALSE;
	}
	if(::SetPixelFormat(m_pDC->GetSafeHdc(), m_nPixelFormat,
&pfd)==FALSE)
	{
		return FALSE;
	}
	return TRUE;
}
```

감사의 글

어느덧 3년이라는 시간이 흘러 여기까지 오게 되었습니다. 3년 전 선택의 기로에서 대학원의 진학은 조금의 망설임도 없었습니다. 저의 목표는 오직 로봇이 좋아서 배움 하나 뿐 이였습니다. 취직, 돈, 학벌 등이 아닌 내가 하고 싶은 것을 할 수 있다는 것이 무엇보다 좋았습니다. 그리하여 3년이라는 시간이 너무 빨리만 가는 것 같고, 더 많이 못 배운 것에 대한 많은 아쉬움이 남습니다. 저에게 공부뿐만 아닌 인생을 가르쳐주신 최형식 교수님께 감사 드립니다. 또한, 저를 항상 걱정해주시고 격려해주신 유삼상 교수님께 감사 드립니다. 그리고, 논문 지도에 많은 도움을 주신 정재현 교수님께도 큰 감사 드립니다.

3년이란 시간 동안 동거동락하며 살아 온 선, 후배, 동기들..항상 옆에서 힘이 되어주었고, 웃으면서 재미있게 지내온 날들은 절대 잊혀지지 않을 것입니다. 앞으로도 항상 번창하면서 꾸준히 연락하여 돈독한 지능로봇 실험실의 후배들을 위해 이끌어 갑시다.

그리고, 개인적인 힘든 일들이 있을 때마다 위로해주고, 아껴준 친구들에게 사랑한다고 말하고 싶습니다.

마지막으로 부모님, 가족 분들 진심으로 사랑합니다.

감사합니다.

목표는 도전하는 자의 것이다! 함부로 목표를 만들지 말라.
사람은 나의 것이 아니다. 나의 생각대로 말하지 말라.
거짓은 1프로의 진실이 담겨있다. 그것을 찾아라.
믿음은 진실된 사람의 것이고, 진실된 사람은 두려울 것이 없다.
희망을 가져라. 신은 결코 나를 버리지 않을 것이다.

\- 나의 다섯 이야기 -

과거와 현재, 미래의 나의 모습을 그려보고 도전하는 사람은 아름답다.