최 종
연구보고서

# 포도수확기 개발

Development of Grape Harvester

연 구 기 관

성 균 관 대 학 교

농 림 부


대한민국
납본
기증등록
2001. 2. 12
855200
국회도서관

# 제 출 문

농림부 장관 귀하

본 보고서를 "포도수확기 개발" 과제의 최종보고서로 제출합니다.

2000 . 10 . 8 .

주관연구기관명 : 성균관대학교
총괄연구책임자 : 이 용 국
연 구 원 : 이 대 원

# 요 약 문

Ⅰ. 제 목

포도수확기 개발

Ⅱ. 연구개발의 목적 및 중요성

대부분의 과일은 수확시기에 노동력 투하시간이 가장 많다. 포도도 마찬가지로 적기 수확을 위해서 수확시기에 많은 노동력이 집중된다. 현재까지의 포도 수확은 대부분 사람의 수작업에 의존되고 있다. 외국의 경우에는 가공용 포도수확을 위한 수확기가 프랑스에서 개발되었지만, 생과용 수확기 개발은 아직 전무하다. 따라서 본 연구에서는 생과용 포도 수확을 위한 자동화 시스템의 개발을 위해 수행하였다.

농업생산액 중에서 원예산업이 차지하는 비중이 계속 증가하고 있으며, '95 연말 농업생산액의 38.9%를 점하고 있어 원예산업의 중요성이 커지고 있다. 이중 채소가 25.2%, 과수가 11.2%, 화훼가 2.0%를 차지하고 있고, 시설채소도 '90년 3.6%에서 '95년에는 8.5%로 급격히 성장하고 있다. 이러한 현상은 계속 이어질 전망이며, 적기에 이들을 수확하기 위한 수확기 개발이 필요한 실정이다. 따라서 농업 분야에서도 단순 자동제어 시스템으로 해결될 수 없거나 복잡하고 지루한 반복작업이 지속되는 분야에서 농업용 엔드이펙터 개발이 필요한 실정이다.

과채류 수확에 필요한 시스템의 개발 시 중요하게 고려할 요소는 수확하고자 하는 대상물의 물리적 특성이다. 수확하고자 하는 대부분의 농작물은 수확시에도 생명체로서 존재하고 있다. 그러므로 일반적으로 공장에서 사용하는 산업용 로봇의 몇 가지 기능만을 변화시켜 농업용으로 이용하는 것은 여러 가지 측면에서 한계를 가지고 있다. 즉, 공업용 대상물은 대부분 정밀 작업이 요구

되는 견고한 강체이지만, 농작물은 이동하거나 충격을 가하면 상처를 쉽게 받는 유연한 생명체이다. 따라서 보다 효율적인 과채류의 수확에 필요한 시스템의 개발에는 보다 감각적인 기술의 도입이 요구된다.

현재 포도의 수확에는 많은 노동력이 소요되는 수(手)작업으로 이루어지고 있다. 포도는 재배수형과 종류에 따라 결과의 높이에 차이는 있지만, 대부분 지상(地上)에서 50cm-150cm내외에서 결과(結果)한다. 또한 수확시 과경의 경질화에 따라 수확시 많은 노동력을 필요로 한다. 특히 포도 과육은 높은 함수율로 인해서 수확시 상품가치의 하락을 방지하기 위해서 많은 주위가 필요하다. 따라서 각각의 독립된 수확을 하지 않으면 안 된다. 이에 따른 연속된 단순 반복 작업으로 피로도는 더욱 증가될 수밖에 없는 실정으로 재배에 많은 어려움을 낳고 있다. 수작업을 통한 포도의 수확작업에는 한 손으로 과육을 잡고, 과경의 절단 위치를 눈으로 보거나, 감지하여 다른 한 손으로 도구를 이용하여 절단한다. 이러한 작업을 자동적으로 절단할 수 있는 시스템의 개발이 필요하다.

따라서 본 연구에서는 포도의 홍수 출하시 노동력을 절감할 수 있는 포도 수확기 개발을 위해 가장 핵심 기술인 영상처리 시스템과 엔드이펙터 시스템 개발을 목적으로 수행하였다.

## Ⅲ. 연구개발 내용 및 범위

포도수확기 개발을 위해서는 크게 두 가지 기술이 선행되어야 한다. 먼저 포도의 산물 정보를 인식할 수 있는 시스템의 개발이며, 본 연구에서는 영상을 통한 포도인식을 수행하였다. 그리고 다양한 형태의 포도 과경을 절단할 수 있는 엔드이펙터 시스템의 개발이다.

포도인식을 위한 영상처리시스템은 우선 수확적기의 포도를 선별할 수 있어야 한다. 또한 수확하고자 하는 포도의 정확한 좌표 결정을 위한 영상처리 시스템이어야 한다. 그리고 포도수확을 위한 영상처리 시스템은 포도와 다른

줄기나 잎을 판별할 수 있어야 한다. 그리고 포도 수확을 위한 과경 절단용 엔드이펙터 시스템은 여러 가지 형태에 과경에 대해서도 능동적으로 절단 가능하여야 하며, 포도의 상품성을 높이기 위해서 적절한 파지능력을 구비하여야 한다. 따라서 본 연구의 범위는 포도수확기에 적합한 엔드이펙터 시스템 개발과 영상처리시스템 개발이다.

따라서 본 연구에서는 다음과 같은 방법으로 연구를 수행하였다.

1) 엔드이펙터 시스템

먼저 엔드이펙터 시스템의 개발을 위해서 포도 주산지역을 방문하여 포도의 과경 길이 및 형태를 조사하여 엔드이펙터 설계의 기초 자료로 활용하였으며, 또한 재배양식이나 결과 높이 등도 포도수확기 개발에 중요한 요소가 될 것으로 생각하여, 지역별, 지형별 및 품종별로 영향을 받지 않는 수확기의 개발이 필요하다. 또한 포도 수확시 과경 길이와 형태는 매우 다양하다. 이러한 과경의 형태에 대해서 능동적으로 수확 가능한 엔드이펙터를 개발할 필요가 있다. 이를 위해서 우선 과경까지의 접근성이 좋아야 한다. 따라서 가능하면 소형이어야 하며, 간단한 시스템으로 구성하였다. 이러한 점을 고려하여 엔드이펙터 시스템은 단순화 시켰으며, 재료는 드랄미늄을 이용하여 무게를 최소화 시켰다. 또한 과경이 외형상 나타나지 않은 포도의 수확을 위해 일정 강도로 포도과육을 밀면서 과경의 위치를 파악하고자 설계되었다.

현장조사 통해서 설계 제작된 엔드이펙터를 현장 실험을 통하여 수확기의 접목에 적절한지를 분석하였다. 개발한 엔드이펙터의 기초실험을 통해서 수확효율을 검증한 후, 자동화를 위한 제어 보오드를 설계 및 제작과 동시에 엔드이펙터 구동을 위한 모터제어 프로그램 등을 개발하였다.

2) 영상처리 시스템

일반적으로 포도의 수확은 품종에 따라 차이가 있지만, 대부분 수확 시 과일의 색이 변한다. 따라서 포도수확기 개발에 있어서 포도 인식을 위한 영

상처리 시스템은 포도인식을 위한 칼라이미지 프로세싱을 이용하였다. 2차원 내의 포도인식은 RGB차이에 따른 분석이 가능하다. 그러나 3차원공간상의 포도위치 점을 찾기 위해서는 두 대의 카메라, 또는 다른 2가지 형태의 입력 영상을 분석하지 않으면 안 된다. 따라서 본 연구에서는 여러 가지 3차원 영상분석 방법 중에 2대의 카메라를 이용한 스테레오 영상처리시스템(stereo vision system)을 설계, 제작하였다.

제작된 영상처리시스템은 두 개의 카메라가 동시에 180 °회전이 가능하며 상하, 좌우로 각각 이동이 가능하도록 설계, 제작하여 가능하면 여러 방향의 포도를 인식할 수 있도록 하였다. 현장에서의 포도 패턴인식 위해서 신경회로망의 연상메모리 알고리즘을 이용하였다. 또한 포도영상을 위한 알고리즘에 포도의 기하학적인 형상도 고려하였다.

## Ⅳ. 연구개발결과 및 활용에 대한 건의

본 연구에서는 포도수확을 위한 엔드이펙터 시스템과 영상처리시스템을 개발하였다. 현장실험 결과 비교적 양호한 결과를 얻었다. 그러나 개발한 두 가지 시스템만으로는 현장 적용 가능한 포도수확기의 현장보급이나 상품화에는 미흡하다. 따라서 본 연구에서는 완전한 포도수확기 개발을 위해서 기초 자료의 성격으로 수행되었으며, 본 연구에서 개발한 내용을 기초로 포도 수확기의 전체적인 시스템 개발에 활용 가능할 것으로 판단된다.

# SUMMARY

## (영문요약문)

I. TITLE

DEVELOPMENT OF GRAPE HARVESTER

II. Objectives and importance of development

Most of fruit needs laboring greatly to harvest fruit in the period of harvest season. Grape also needs requiring a lot of labor to harvest in time, since harvesting of grape is cut and graped by hand. At abroad, especially, a harvester of grape, which is not eat freshly raw grape fruit but is processed to be made wine, was developed in France. However, a harvester of grape which can eat fresh raw grape fruit haven't not been developed. Therefore, this study was designed and constructed to develope the computer vision and end effector of the harvesting system.

Grape grippers need to be developed with compliance and tactile feedback. Compliance means the gripper takes the shape of the grape fruit when cut and gripped. In doing so compliance can reduce the amount of force required to pick up a grape. Tactile feedback means the gripper and its associated computer can determine the force applied, an extremely important feature for preventing damage to sensitive agricultural products. Computer vision plays an important role in the application of harvester to agriculture. harvesting systems, incorporating computer vision, appear to be an alternative with a potential for sucess.

Currently, a lot of labor needs to harvest a bunch of grape with simple, repetitious and fatigue operation by hand. A bunch of grapes was located at different height within 50cm ~ 150cm from ground. This height is one of

the most important factors to develop the computer vision system of fruit harvester. Physical prosperity of grapes was measured and analyzed for its gripper to cut and grip well a grape fruit,

## Ⅲ. Contents of development

Harvesting system of grape proved to be reliable system for recognizing the position of a bunch of grape at the Cartesian coordinate and cutting and gripping a fresh grape fruit in grape creeper. Its development involved the integration of a computer vision system and end effecters along with an PC computer. Software, written in visual C++, combined the functions of image capture, image processing, and control into programs Two separate programs, one for the end effector, and one for image processing, were included in the software.

### 1) End-effecter system

In order to find average and standard deviation of size, shape of a bunch grape, the grape farm was visited and investigated to find out the important factors for end effector to be designed and built and to catch and analyze images. The efficient end-effecter should reach very well a bunch of grape without touching obstacle, such as leaf and stem in the grape vine. Also, it was made as simple, small and light as possible. Therefore, It was made duralumin, which is lighter material than iron, to reach easily fresh grape fruit and reduce a torque.

1) The end effector, which designed and constructed for this project, was an economical choice for cutting and gripping grape fruits growing in a grape field environment.

2) After finding the optimum initial speed, slew speed, ramp speed, the end effector proved to be a smooth operating end effector.

3) For verifying efficiency of the end-effecter, the control board was

designed and manufactured, and also developed the software to control the motors in the end effector.

2) Image processing system

Grape fruit's color is changed generally for harvesting grapes, which have different colors many as different kinds. An color image processing was carried out by using computer vision system for grape harvester. Grape recognition in two dimensions could be to find out only object in the background. However, in order to recognize the accurate position of three dimensions, two different images were caught and analyzed simultaneously from two cameras.

1) The computer vision system was designed and built by using two cameras, which can rotate 180° and move to up and down, right and left respectively to recognize the grapes at different direction.

2) Associative memory algorithm of neural network was developed to recognize a pattern of fresh grape.

3) The image processing techniques were used to enhance geometrical shape of grape in the grape field.

## Ⅳ. The result and utilization of development

This study was conducted to develop the end-effecter system and image processing system for harvesting grape. The harvest system proved to find out the position of grape fruit and to cut and grip it.

Based on the results of research the following recommendations are made for further study. First, an automatic manipulator system needed to make the fruit out of the working zone. Second, a harvest system could be developed for field harvesting operations. In this case, the system would be mounted on a field machine for making skillful harvest of grape fruits.

# CONTENTS

## (영 문 목 차)

# 목 차

# 제 1 장 : 엔드이펙터 시스템

## 제 1 절 서 론

농산물 수입개방으로 외국산 농산물의 대량 유입은 우리 농업의 기반을 흔들고 있는 실정이다. 따라서 농산물의 고품질화 및 생산비 절감으로 대외 경쟁력을 높일 수밖에 없는 실정이다. 이를 위해서 농산물의 적기 수확은 재배기술 못지 않게 중요하다.

농업생산액 중에서 원예산업이 차지하는 비중이 계속 증가하고 있으며, '95 연말 농업생산액의 38.9%를 점하고 있어 원예산업의 중요성이 커지고 있다. 이중 채소가 25.2%, 과수가 11.2%, 화훼가 2.0%를 차지하고 있고, 시설채소도 '90년 3.6%에서 '95년에는 8.5%로 급격히 성장하고 있다. 이러한 현상은 계속 이어질 전망이며, 적기에 이들을 수확하기 위한 수확기 개발이 필요한 실정이다. 수확기 중에서 가장 중요한 부분 중에 하나인 엔드이펙터(End-effector)의 개발이 요구되고 있다. 현재 대부분의 과채류 생산에 엔드이펙터를 이용한 수확기 개발을 시도하고 있으며, 부분적으로 상당한 수준에 도달한 분야도 있다. 농업 분야에도 단순 자동제어 시스템으로 해결될 수 없거나 복잡하고 지루한 반복작업이 지속되는 분야에서 농업용 엔드이펙터 개발이 필요한 실정이다.

과채류 수확에 필요한 엔드이펙터 개발 시에 중요하게 고려할 요소는 수확하고자하는 대상물의 물리적 특성이다. 수확하고자하는 대부분의 농작물은 수확 시에도 생명체로서 존재하고 있다. 그러므로 일반적으로 공장에서 사용하는 산업용 로봇의 엔드이펙터의 몇 가지 기능만을 변화시켜 농업용 엔드이펙터로 이용하는 것은 여러 가지 측면에서 한계를 가지고 있다. 즉, 공업용 대상물은 대부분 정밀 작업이 요구되는 견고한 강체이지만, 농작물은 이동하거나 충격을 가하면 상처를 쉽게 받는 유연한 생명체이다. 따라서 보다 효율적인 과채류의 수확에 필요한 엔드이펙터의 개발에는 보다 감각적인 기술의 도입이 요구된다.

현재 포도의 수확에는 많은 노동력이 소요되는 수(手)작업으로 이루어지고 있다. 포도는 재배수형과 종류에 따라 결과의 높이에 차이는 있지만, 대부분 지상(地上)에서 50cm-150cm내외에서 결과(結果)한다. 또한 수확시 과병의 경질화에 따라 수확시 많은 노동력을 필요로 한다. 특히 포도 과육은 높은 함수율로 인해서 수확시 상품가치의 하락을 방지하기 위해서 많은 주위가 필요하다. 따라서 각각의 독립된 수확을 하지 않으면 안 된다. 이에 따른 연속된 단순 반복 작업으로 피로도는 더욱 증가될 수밖에 없는 실정으로 재배에 많은 어려움을 낳고 있다. 수작업을 통한 포도의 수확작업에는 한 손으로 과육을 잡고, 과병의 절단 위치를 눈으로 보거나, 감지하여 다른 한 손으로 도구를 이용하여 절단한다. 이러한 작업을 자동적으로 절단할 수 있는 엔드이펙터의 개발이 필요하다. 엔드이펙터를 개발하기 위하여 기초 연구로서 시도된 본 연구의 구체적인 연구 목적은 다음과 같다.

가. 엔드이펙터의 파지장치의 개발을 위하여 포도 결과의 물리적 특성 분석.

나. 엔드이펙터 기술을 확립하여 과실의 파지와 과경(果柄)을 절단 할 수 있는 엔드이펙터를 개발.

다. 다양한 형태의 과경에 효율적으로 수확이 가능한 엔드이펙터 개발에 있다.

## 제 2 절 연구동향 및 문헌조사

농산물의 수확을 위한 엔드이펙터의 개발은 과실과 과경의 형태에 적합하고 현장 상황에 알맞은 엔드이펙터의 개발이 필요하여, 현재까지 많은 연구가 진행되고 있지만 실용적으로 사용되고 있는 경우는 거의 없는 실정이다. 그러나 많은 분야에서 꾸준히 연구되어 가까운 미래에 실용화가 가능한 영역도 많이 있다. 특히 접목이나, 이식에는 현재 현장에서 많이 이용되고 있지만, 수확에 필요한 기술은 아직 미흡한 점이 많다.

엔드이펙터와 관련된 국내와 연구를 살펴보면 다음과 같다. Hoy는 감자수확을 위해 3개의 오목한 손가락을 가진 그리퍼를 설계했다. 손가락 내부에 공기가 들어있다. 촉각은 압력센서를 이용하여 피드백(Feedback)시키므로 감자와 꼭 맞게 만들어진 금속의 지주가 잡는다. 오목한 손가락의 그리퍼는 감자 지름의 0.01inch까지 분해능을 가지고 있다.

묘목을 이식하기 위한 엔드이펙터는 Hwang 등(1985), Kutz 등(1987), Ting 등(1988), 이(1990), 류 등(1997)이 수행하였다. Hwang 등은 후추 묘목과 할라파(Jalapeno)의 줄기를 삽처럼 생긴 손가락으로 2개를 이식시킨다. Kutz 등은 2개의 평행한 집게를 가진 엔드이펙터를 가지고 있다. 손가락은 높이가 30~50mm인 토마토와 금잔화(Marigold)의 묘목을 이식하는 실험을 위하여 설계되었다. Ting 등은 바늘 형태의 엔드이펙터에 근접센서를 사용하여 이식하도록 설계하였다.

류 등은 육모용 이식을 목적으로 엔드이펙터를 개발하였는데, 무게를 줄이기 위하여 알루미늄을 사용하여 제작하였다. 모종을 잡을 때에는 스테핑모터가 잎의 방향에 따라 회전한 후 공압실린더와 공압척을 순차적으로 작동하여 모종을 잡도록 설계되었다.

D'Esnon (1985)와 Chai 등(1989)은 포도 수확용, Harrell 등(1990)은 오렌지용 엔드이펙터, 近藤 直(1995)은 감귤과 토마토용, D'Esnon (1985)은 생식용 사과용 엔드이펙터를 개발하였다.

D'Esnon은 생식용 사과 수확을 목적으로 로봇을 개발하였는데, TV카메라를 이용해서 과실을 검출하고 핸드부는 진공을 사용하여 과실을 흡착한 후 비틀어서 수확은 방법을 사용하였다. 광조건이 좋은 경우 울타리형으로 재배된 수형의 사과나무에서 50% 이상의 과일을 4초에 1개꼴로 수확가능하고 상처가 거의 없으며 75%는 사과 과병이 붙어 있었다. 유연한 손가락 8개가 전후로 180 °회전 할 수 있도록 되어 있어 과병의 방향에 관계없이 대부분의 과실을 메니퓰레이터(Manipulator) 내로 들어오게 할 수 있도록 되어있다. 줄기와 잎이 팔의 이동에 장해가 되거나, 작동영역 외에 과실이 존재하는 경우는 자동적

으로 팔을 끌어들인다. 실험 결과 1개를 수확하는데 4초의 시간이 소요되었다.

A.Sitticharcon chai 등은 포도 수확을 목적으로 계발하였으며, 그리퍼(Gripper)부의 파지력은 수확시 과방의 질량이 약 500g이므로, 수축의 마찰저항을 고려하여 10N으로 한다. 또 수확시 수축의 단면적은 약 30 $mm^2$이며, 수축의 절단저항을 고려하여 그리퍼(Gripper)부의 절단력은 100N으로 했다.

파지, 절단부의 아래쪽으로 과방을 밀어내는 기능을 부과하고 있다. 파지부와 절단부는 하나의 DC모터와 용수철로, 밀어내는 부분은 모터의 회전운동을 래크와 피니언에 의해 직선운동으로 바꾸어 구동하고 있다.

Harrell 등은 오렌지수확로봇을 개발하였다. 메니퓰레이터 선단에 카메라가 있고 12개의 엔드이펙터을 가지고 초당 6개의 과실을 수확하도록 고안하였으며, 하루에 4시간의 점검정비시간을 제외하고 20시간의 작업이 가능하였다.

近藤 直 등은 오이 수확작업의 자동화를 목적으로 로봇수확에 적합한 재배양식을 구명하고 그 재배양식에 맞는 메니퓰레이터를 개발하였다. 특징은 로봇수확에 적합하도록 과실과 경엽 등의 분리가 용이한 경사재배양식에서 수확실험을 하였으며, 메니퓰레이터는 직선운동의 1자유도와 회전 및 관절운동의 5자유도로 구성하였다. 재배된 오이의 붕(棚)의 각도가 크면 수확이 용이한 것으로 나타났다. 수확은 과실을 그리퍼 내의 접촉센서로 검출하면 파지부에서 과실을 약 70N의 힘으로 파지, 유지함과 동시에 센서, 절단부가 과실을 약 3N의 힘으로 잡고, 오이를 따라 상방향으로 미끄러짐으로써 지름을 검지 할 수 있다. 과병에서는 과실 지름이 급속하게 작아지기 때문에 그 위치를 마이크로 스위치로 알아내어 과병을 절단하였다. 절단력은 8N~15N이다.

近藤 直은 하우스 안의 토마토 등의 과채류 수확을 목적으로 개발을 하였으며, 범용성을 갖도록 방제, 솎음 작업등도 메니퓰레이터와 엔드이펙터를 교환하면 가능하도록 제작하였다. 관절형 로봇으로서 그리퍼 끝이 플랜지에 시각부와 절단기를 가진 엔드이펙터를 부착한 것이다. 엔드이펙터의 경우 반원 링형의 절단기를 사용하여 수확하도록 되어 있다. 그리퍼는 완만히 구부러진 그리퍼가 좌우로 부착되어 있고 과실을 미끄러지지 않도록 그리고 가능한 손상

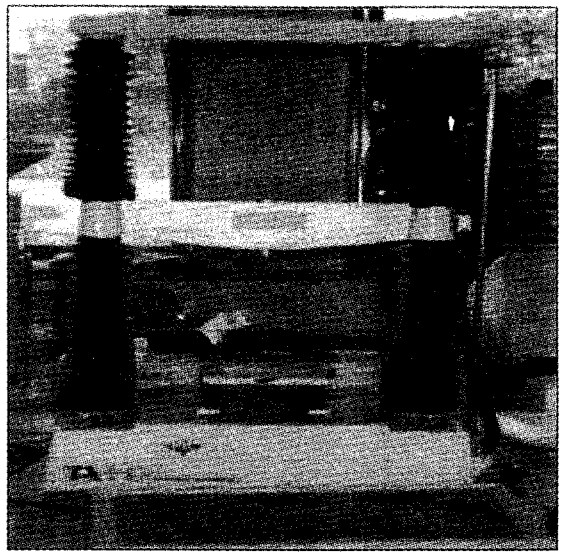
을 주지 않도록 손가락의 안쪽에 고무가 부착되어 있다. 소형 DC모터로 그리퍼를 개폐하고, 전류를 제어함으로써 과실을 일정한 힘으로 파지하며, 메니퓰레이터를 구동하여 당기거나 구부려서 수확을 한다.

近藤 直은 감귤 등과 같은 큰 수관을 가지는 과실을 수확하는 목적으로 개발하였으며, 과실인식 및 위치검출은 카메라를 이용하였고 핸드는 고무재질의 3개의 손가락으로 파지 하도록 하였다. 고무제의 3개의 손가락을 가지고 있지만, 그리퍼의 상부에 있는 가위로 결과지를 절단하도록 되어 있다. 럽액츄레이터(rubactuator)와 손가락 끝을 잇는 철사를 럽액츄레이터의 수축력으로 잡아당기면 손가락이 매끄럽게 휘어져 과실을 파지 한다. 손가락은 위에 2개, 아래에 1개가 있고, 2개의 위에 있는 손가락 사이에 결과지를 넣어 과실을 파지하여 수확하였고 엔드이펙터의 구동원은 유압을 이용하였다.

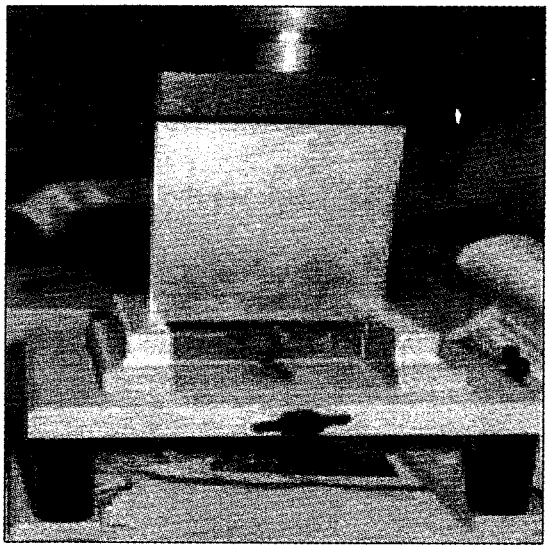
## 제 3 절 장치 및 방법

### 1. 물성측정

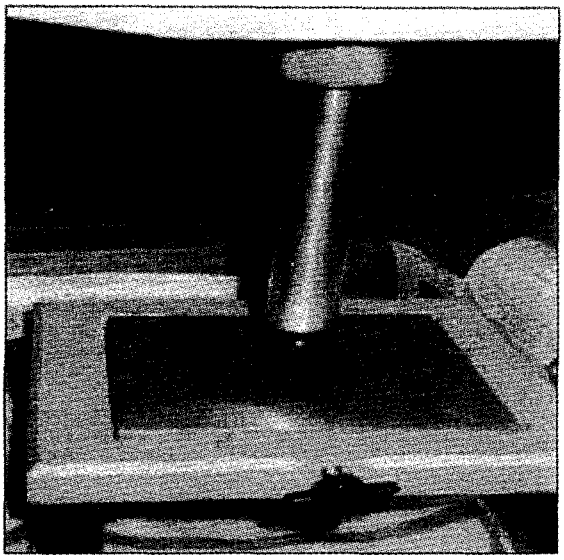
포도 수확기 개발을 위한 기초 자료로서 포도의 물리적 성질을 구명할 필요가 있다. 이는 엔드이펙터의 설계에 중요한 자료가 되기 때문이다. 따라서 물성측정은 <그림 1-1>과 같은 SMS(Stable Micro Systems)사의 Texture analysers-XT.RA을 이용하여 경도(Hardness), 깨짐성(Fracturability), 탄력성(Springness), 점성(Gumminess)을 측정하였다. <그림 1-2>, <그림 1-3>은 조직물성 측정기를 이용하여 포도과병과 과육의 물성을 측정하는 형태을 나타내고 있다. 물성측정시 포도는 <그림 1-4>와 같은 캠벨얼리를 수확하여 이용하였으며, 측정 위치는 과병부분과 과육부분에서 행하였다. 포도의 함수율은 75℃에서 1주일 동안 Dry-oven에서 건조한 후 생체무게와 비교하여 측정하였다.



<그림 1-1> 물성측정장치



<그림 1-2> 과경물성측정

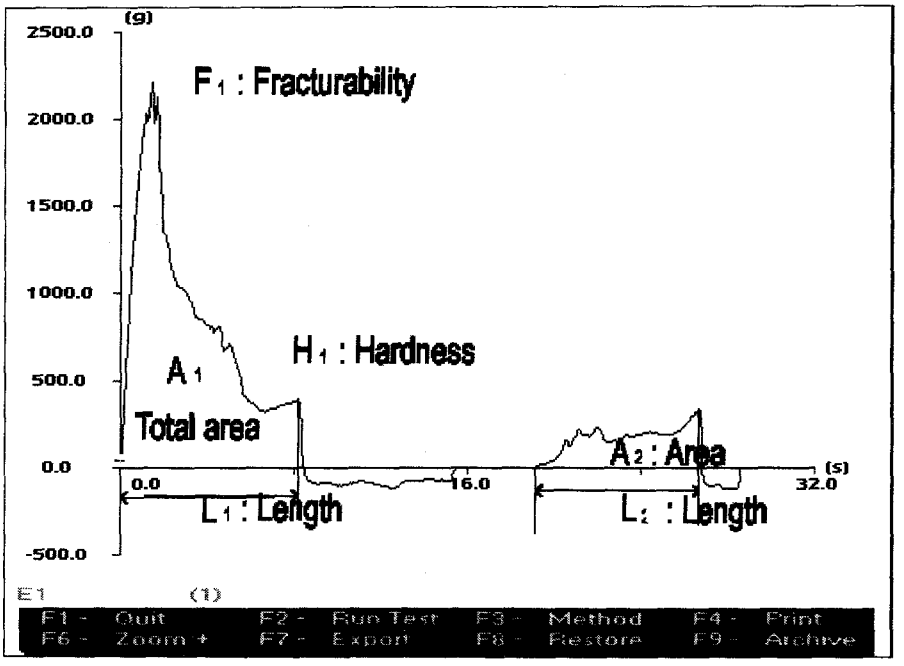


<그림 1-3> 과육물성측정

<그림 1-4> 실험용 포도

포도수확기 개발을 위한 가장 중요한 기술 중에 하나는 엔드이펙터 부분의 설계 및 제작이라 할 수 있다. 보다 효율적인 수확을 위해서 정확한 엔드이펙터의 설계를 위해 본 실험에서는 포도 과병과 과육의 깨짐성, 경도, 탄력성, 점성을 조직물성측정기 TPA(Texture Profile Analysis)을 이용하여 측정하였다.

<그림 1-5>는 조직물성측정기 TPA(Texture Profile Analysis)을 이용하여 깨짐성(Fracturability), 경도(Hardness), 탄력성(Springness), 점성(Gumminess)을 측정한 것이다. 여기서, F1은 깨짐성은 힘을 가했을때 처음으로 하강하는 곡선을 나타낸다. 경도는 압축중 0점으로 떨어지는 최고점으로 나타나고, 탄력성은 L2/L1을 이용하여 구할 수 있다. 점성은 경도 응집성(Cohesiveness)으로 나타난다. 여기서, 응집성은 A2/A1으로 구한다.




<그림 1-5> 과채류 물성측정 그래프

앞에서 언급하였듯이 포도의 물성측정은 깨짐성, 경도, 탄력성, 점성을 측정하였으며, 그 결과 경도의 경우 파지부 설계의 기초자료로서 의미가 있을 것으로 판단된다. 위의 물성값을 기초로 수확기 개발을 위한 기초연구자료로 이용가능할 것으로 판단된다.

## 2. 실험장치

### 가. 엔드이펙터의 설계기준

엔드이펙터는 과채류의 수확에 가장 중요한 요소이다. 특히 포도의 수확시에는 다양한 과병의 형태 및 길이에 대해서 충분한 작업성능을 발휘할 수 있어야 한다. 따라서 효율적인 수확 작업을 위한 엔드이펙터의 설계 및 제작을 위한 기준은 다음 사항을 충분히 고려하여야 한다.

첫째, 현장에서의 작업시 원활한 작동 및 작업의 효율을 높이기 위해서 경량이어야 하고

둘째, 포도 과병의 절단을 위한 접근이나 절단, 작업공간에서의 이동시 포도과육이나 나무에 상해를 주지 않아야 하는 소형이어야 하며,

셋째, 고온다습한 환경에서 오랫동안 작업을 하여야 하기 때문에 습기에 대한 내부식성이어야 하고,

넷째, 여러 가지 예상하기 못한 기계적인 문제의 발생시 원활한 복구를 위해서 구조가 간단하고, 작업 중 변형이 없어야 한다.

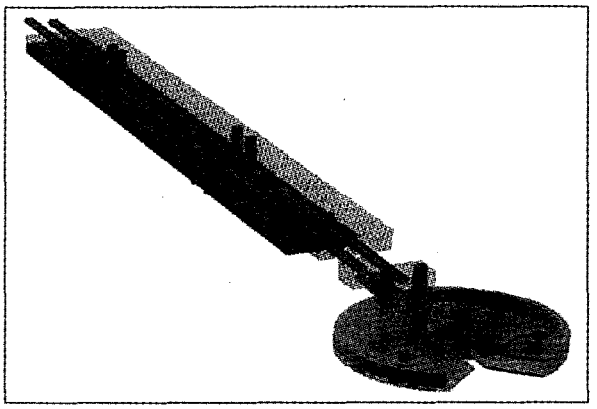
### 나. 엔드이펙터 제작

#### 1) 시스템 I

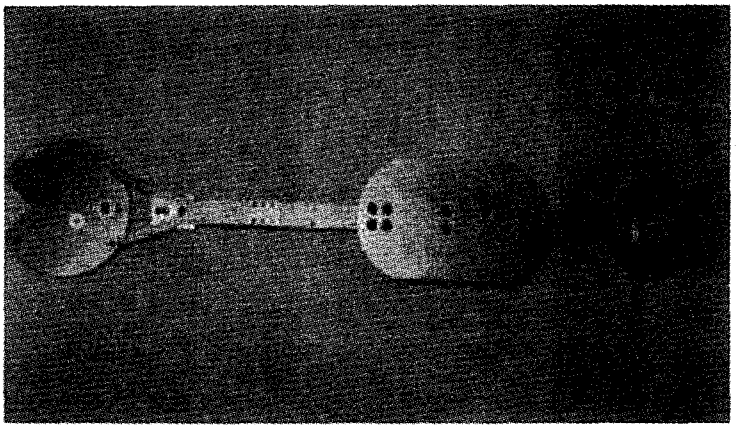
포도 수확을 위한 엔드이펙트는 3D-max을 이용하여 설계하였으며, 가공 및 제작은 CNC 조각기을 사용하였다. 설계한 엔드이펙터는 자동화 시 그리퍼의 Y축 상부에 부착할 수 있도록 설계되었다. 시스템의 비틀림을 방지하기 위하여 직선 가이드를 장착하였고, 파지시 저항력을 주기 위하여 5mm의 인장스

프링을 사용하였다. 또한, 보완 및 수정을 용이하게 하기 위해서 탈부착이 가능하도록 3mm의 볼트로 고정하였다. 각 기계 요소들의 재질은 무게를 줄이고, 부식되지 않도록 하기 위하여 드랄미늄과 1mm의 스테인레스을 사용하여 제작하였다.

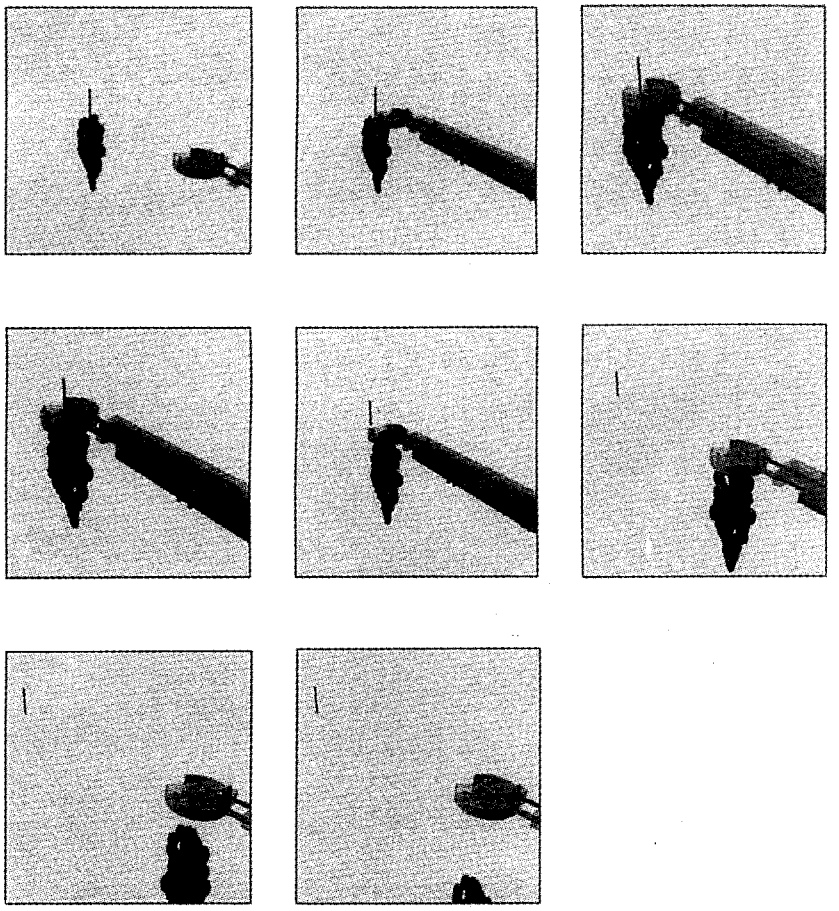
<그림 1-6>은 엔드이펙터시스템 I를 설계도를 나타낸 그림이며, <그림 1-7>은 구동 모터가 부착된 엔드이펙터의 제작된 실물사진이다. <그림 1-8>은 시스템 I을 통한 수확단계를 개략적으로 도식화한 그림이다.



<그림 1-6> 엔드이펙터 시스템 I의 입체도.



<그림 1-7> 엔드이펙터 시스템 I의 실상도.



<그림 1-8> 엔드이펙터 시스템 I의 작동과정.

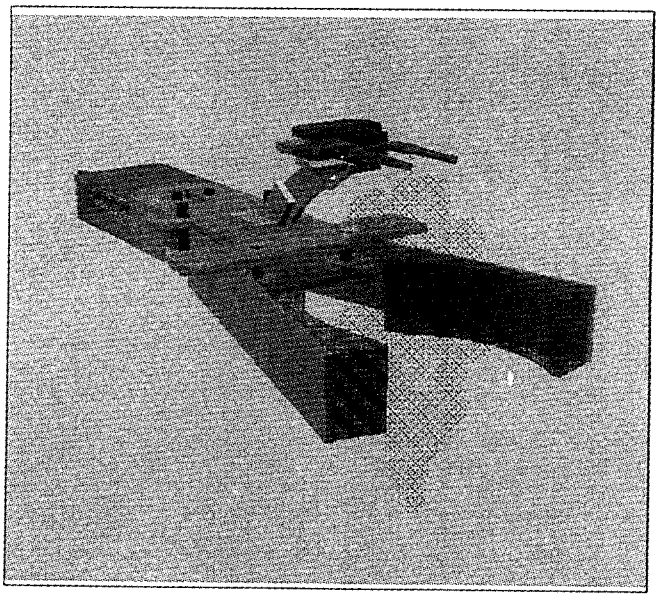
포도를 영상입력을 통하여 인식하고 과경 부분을 절단하기 위한 엔드이펙터는 공간상에서 원하는 위치로 신속히 이동하기 위해 가벼워야 하므로 주재료로 드랄니늄을 사용하였다.

엔드이펙트 시스템은 12V, 60rpm의 감속모터(Geared motor)로 조정축(Control axis)을 당기면 이동축(Moving axis)이 안쪽으로 당겨지면서 고정핀(Fixation a pin)으로 결합되어 있는 연결판(Connection tube)을 당기고 파지부(Attaching Part)에 고정되어 있는 고정판(Fixing plate)에 힘이 가해져서 파지

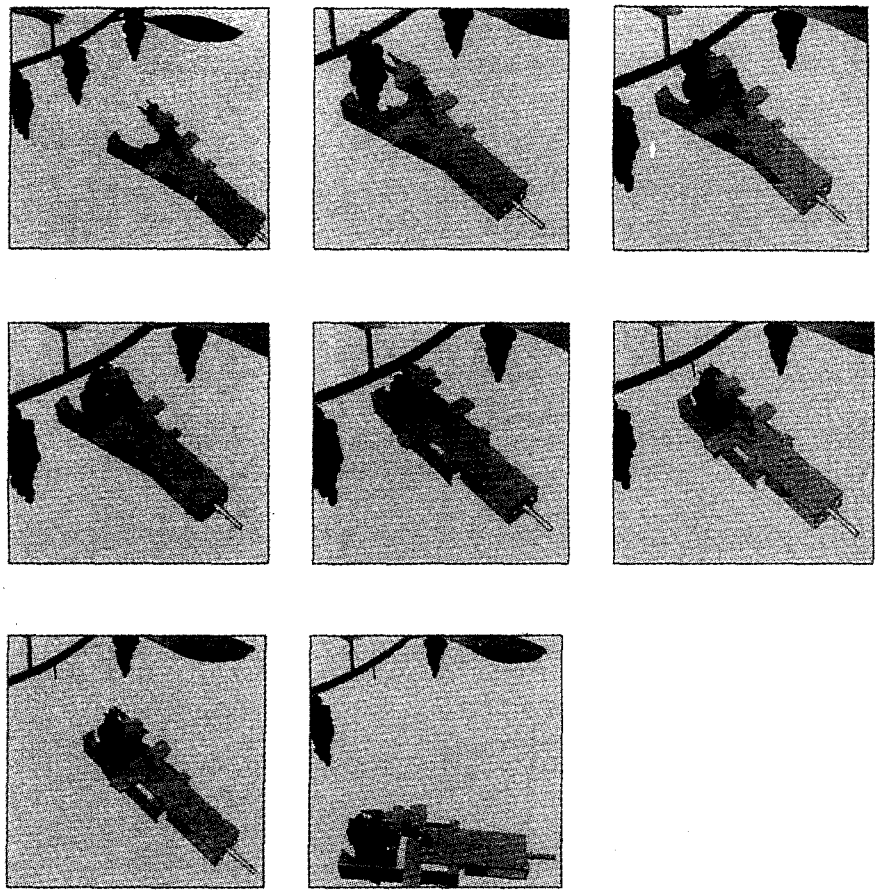
부가 조정축에 당기는 힘을 더 가하면 과병이 닿아서 절단되는 것이다. 절단후 인장력을 제거하면 인장스프링(Extension coil spring)에 의해 원점으로 이동하는 것이다.

2) 시스템 II

시스템 II은 포도의 과병 길이가 짧아서 접근이 불가능한 포도의 수확에 효과적으로 대응하기 위해서 시스템 I과는 다른 형태로 설계하였다. 전체적인 설계 기준은 시스템 II과 동일하며, 시스템 I과의 차이는 <그림 1-9>에서 보는 것과 같이 시스템을 수작업으로 수확하는 형태와 마찬가지로 일정한 힘을 과육부분에 가하여 과병이 나타나도록 한 후 절단장치를 이용하여 절단할 수 있도록 하였다. <그림 1-10>은 시스템 II를 이용한 포도의 수확과정을 나타낸 그림이다.



<그림 1-9> 엔드이펙터 시스템 II의 입체도.



<그림 1-10> 엔드이펙터 시스템 II의 작동과정.

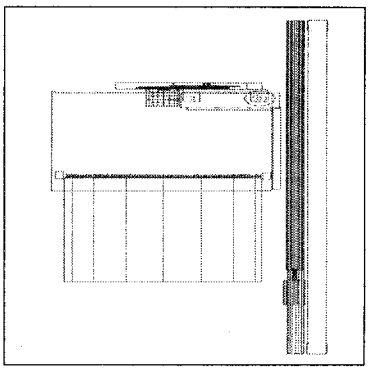
3) 시스템 III

가) 실험장치

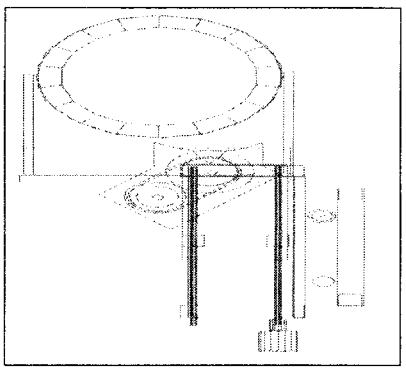
포도수확기의 설계에 있어 고려해야할 가장 중요한 것은 작업성능이다. 작업성능을 높이기 위해서는 상승부의 스크류를 신속하게 이동시켜 정확한 위치에 있게 하는 것이다. 따라서 수확작업을 수행하는데는 엔드이펙터 성능의 중요한 요소인 공간상에서 원활하게 작동하고 수확 중에 포도에 상처를 주어서는 않된다는 점과 습기에 대한 내부식성을 가지기 위하여 알루미늄 계열의 드랄리

늄과 ABS 수지를 이용하여 제작하였다.

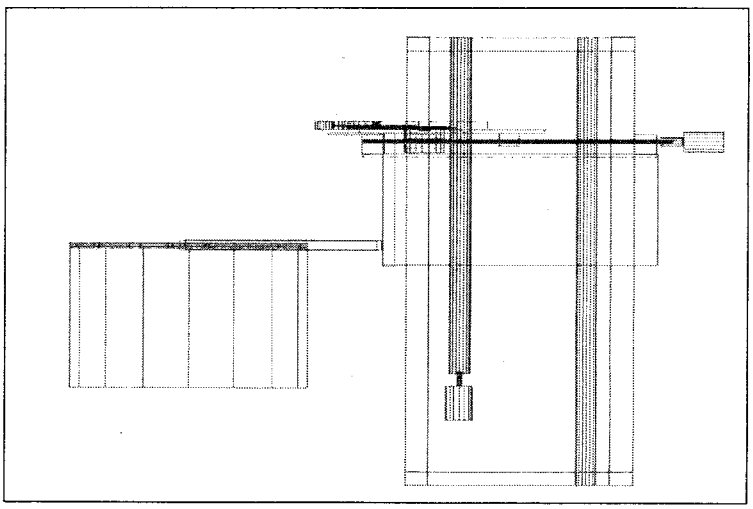
엔드이펙터의 설계는 Kinetix사의 AutoCAD2000과 3DS MAX를 이용하여 설계하였으며, CNC 조각기를 이용하여 제작하였다.



(a) 좌측면도

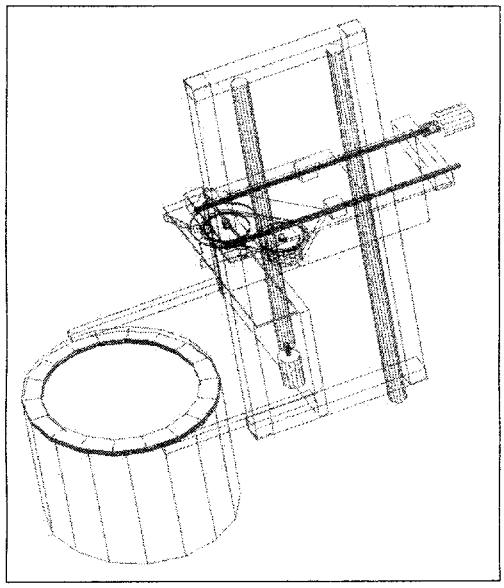


(b) 정면도



(c) 우측면도

<그림 1-11> (a), (b), (c) 엔드이펙터 시스템 Ⅲ 설계도



<그림 1-12> 시스템 Ⅲ의 상승부 설계도

(1) 상승부

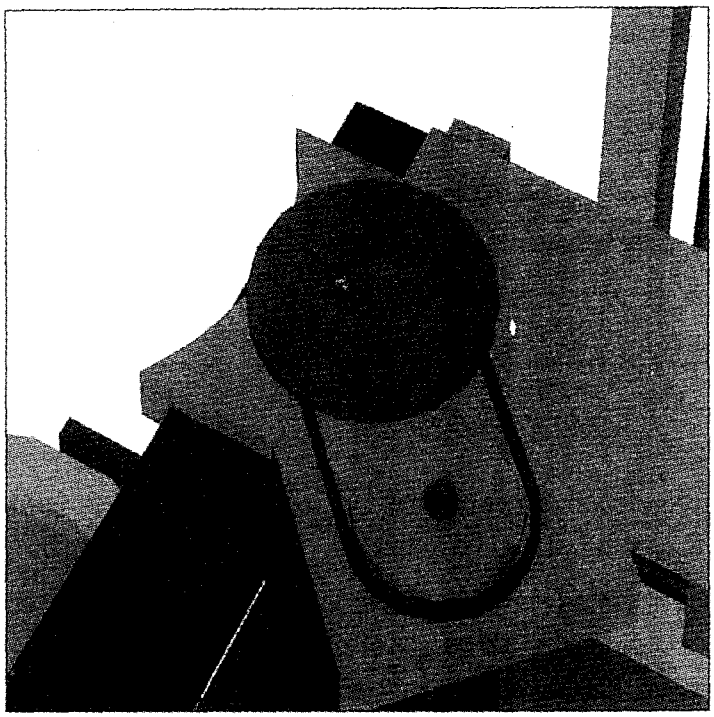
수확작업을 수행하는데 있어 상승부의 역할은 연약하고 상해를 입기 쉬운 포도에 엔드이펙터를 근접시키는 것이다. 상승부는 12Φ 스크류를 사용했으며 스큐류의 흔들림을 방지하기위해 12Φ 샤프트를 사용하였다. 모터는 모터뱅크사의 GM35 24V DC 모터를 사용하였다. 커플링은 모터쪽이 7Φ 스크류 쪽이 5Φ 짜리를 사용하였다.

(2) 절단부

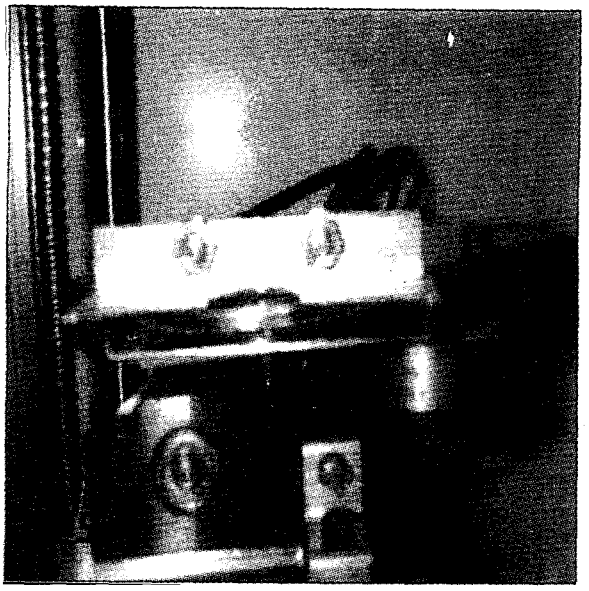
포도의 경우 과경이 질기기 때문에 순간적으로 큰 토크를 요구하므로 보다 쉽게 구연하기 위해 톱날을 사용하게 되었다. 톱날은 목공용 톱날로서 70Φ를 사용하였다. 톱날부분을 구동시키는 모터는 12V 1500RPM짜리 타이완 모터를 사용하였다. 절단부의 상승시 톱날집을 8Φ 스쿠류를 사용하여 상승시에는 톱날집이 들어갔다가 광센서로 인식하고 나면 톱날부위가 나오도록 설계하였다.

스크류는 모터뱅크사의 24V 토크 4.91(mN.m) 6000RPM GM-35 기어모터를 사용하여 구동시켰다. 스크류는 카스코 정밀주식회사의 SA 범용스스크류를 사용하였고 지지대는 BK8과 BF8를 사용하였다.

광센서는 오토닉스사의 FD-620-10모델로 수광부와 발광부가 같이 있는 광센서로 허용 휨 반경은 30R 최소 검출 물체는 ∅0.03 검출거리 200mm이다. 또 광화이버 센서 BF3RX를 사용하였다. 이 모델은 응답시간 1ms이하이고 전원전압 DC12 ~ 24V 이고 소비전류 40 mA이하 사용 광원은 적생 발광 다이오드 감도 조종은 VR내장 방식으로 2단 조정으로 강조정 및 미세 조정이 가능하다. 광센서의 위치는 너무 낮으면 톱날집이 포도 줄기를 상하게 할 우려가 있고 너무 높으면 포도를 상하게 할 우려가 있는바 이에 톱날위치와 최소의 거리가 되는 톱날집 바로 위에 설치를 하였다.



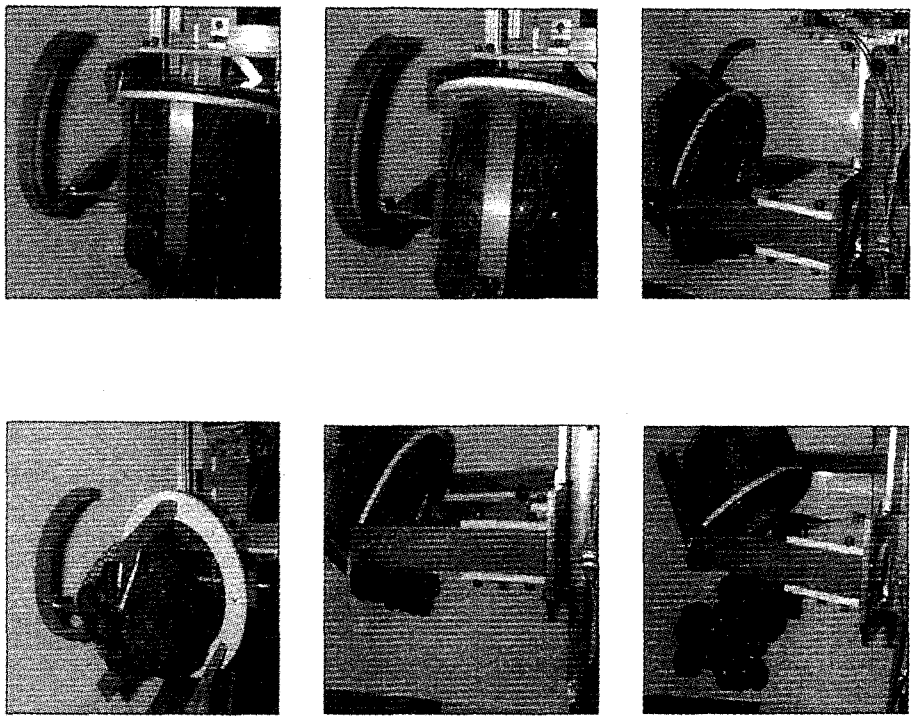
<그림 1-13> 절단부



<그림 1-14> 광센서

(3) 수납부

수납부는 떨어지는 포도를 가능하면 충격 없이 받아 이송부로 정확이 전달하는 역할을 한다. 포도에 손상 없이 받아내어야 함으로 가이드의 최대직경을 160Ø으로 설정하였다. 상승부 스크류가 하강시 가이드가 레일을 따라 회전하여 포도를 이송부로 전달할 수 있도록 설계하였다. 180도 회전할 수 있도록 하기 위해 바퀴를 사용하였으며 바퀴는 소형화를 위해 외경 22Ø인 베어링으로 대신하였다. 이를 마찰 줄여 회전시키기 위해 지지대에 내경 6Ø인 베어링을 사용하였다. 포도를 수납하기 위한 소재로는 충격을 완화할 수 있고 포도가 이송부로 부드럽게 전달될 수 있는 것으로 낚시에 쓰는 어망을 이용하였다. 또한 무게를 줄이기 위해서 ABS수지를 사용하였다. 수납부의 동작 과정은 <그림 1-15>와 같다.

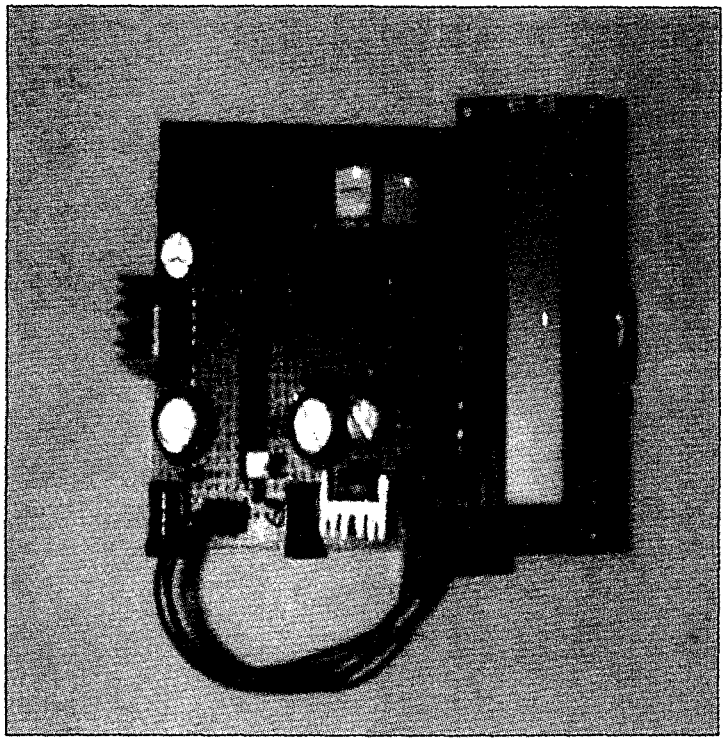


<그림 1-15> 수납부 작동 순서

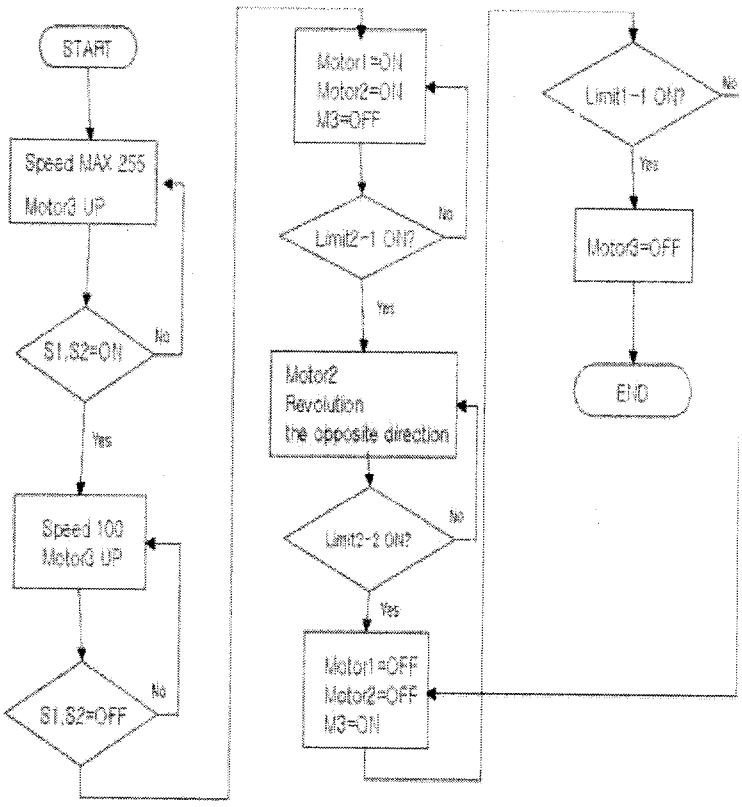
(4) 제어장치

전체적인 제어는 PIC16C73A-04/SO칩을 사용하였다. 또한 인터페이스장치로는 16 by 2라인의 LCD를 사용하였고, 모터드라이브는 L298칩을 사용하였다.

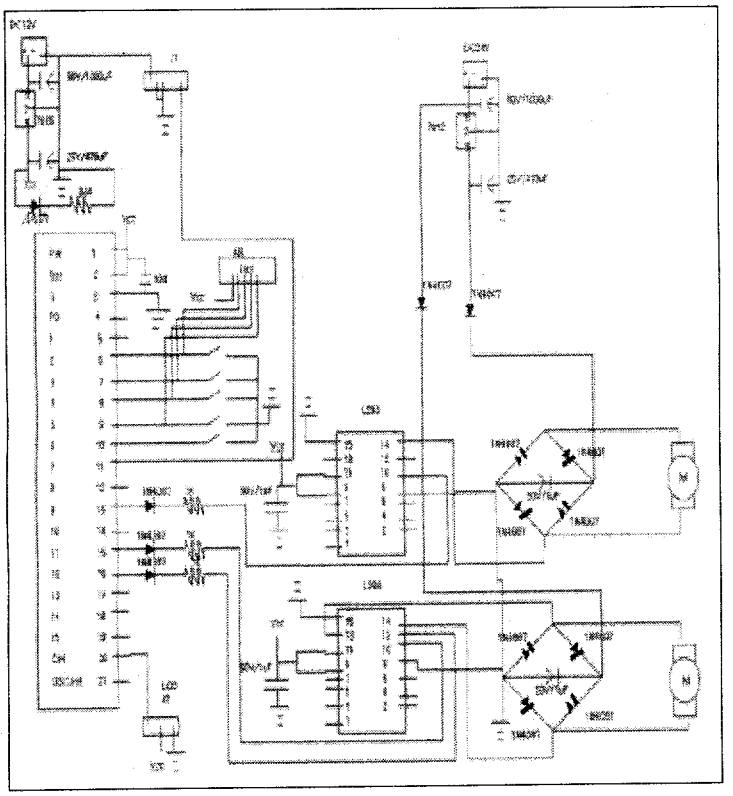
컨트롤 기판은 다음과 같다. 기판에 들어가는 전원은 24V아덥터 1개와 12V아덥터 1개를 사용하였다. 기판에서의 작동은 톱날의 회전 속도를 최고 1500rpm으로 이를 255분할하였으며, 이를 LCD에서 수치를 확인하여 실제 수치로 환산할 수 있도록 하였다.



<그림 1-16> 콘트롤 보드




<그림 1-17> 콘트롤러 흐름도



<그림 1-18> 콘트롤러 회로도

나) 작동방법

제작된 엔드이펙터가 일정한 속도로 상승하면서 광센서에 의해서 과경의 위치를 찾게된다. 광센서가 물체를 인식하면 "1"이 되고 하지 못하면 "0"으로 인식한다. 절단부를 구동시키는 절단모터는 광센서가 0 → 1 일때 off 상태가 되고, 1 → 0 일때 on 상태가 된다. 반면에 엔드이펙터의 상승·하강을 유도했던 수직이송모터는 광센서가 0 → 1 일때 on 상태가 되고, 1 → 0 일때 off 상태가 된다. 또한 절단모터의 유동거리는 120mm 정도이며 이 거리를 왕복한 후 off 상태로 돌아간다. 동시에 수직이송모터는 역회전하여 엔드이펙터의 하강을 유도하게 되고 따라서 레일의 영향으로 가이드가 뒤집어진다. 다시

상승하게될 때 가이드의 바퀴와 하중에 의하여 원상태로 돌아오게 된다. 그러면서 반복된 수확작업을 가능하게 하는 것이다 .

다) 실험방법

포도엔드이펙트의 성능을 판단하기 위한 실험장소는 안성군 대덕면 명당리 소재의 성민포도원으로 약 200ton 의 거봉(올림피아)을 생산하는 곳으로 높이 약 1.8m의 포도나무가 약 1.2m의 간격으로 재배되는 곳이다.

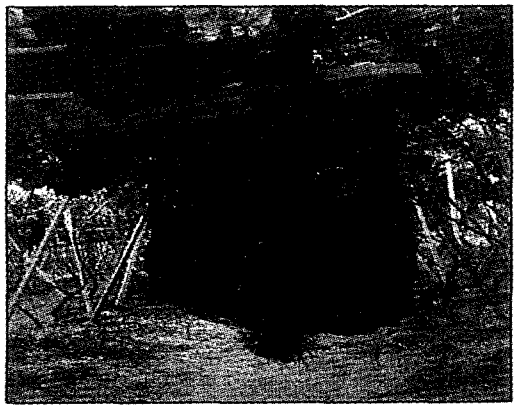
수확이 가능한 포도 아래 설계한 포도 엔드이펙트를 설치하여 광센서가 포도 과경을 감지하여 알맞은 위치에서 최적의 rpm으로 과경을 절단할 수 있는가를 실험하였다. 이를 위해 rpm을 조금씩 높여가면서 과경이 절단 될 때까지 여러 차례 실험을 반복하였다.

이 실험에서 버니어캘리퍼스를 이용하여 과경의 지름과 본 엔드이펙터로 절단 가능하다고 판단되는 과경의 길이를 측정하고 절단된 포도의 무게를 측정하여 톱날의 회전속도 조절을 통한 최적의 절단 속력을 추정하였다. 또 절단된 포도의 낙하상태를 살펴 정확히 수납부 속으로 떨어지는지 여부와 포도의 손상 상태 등을 확인하고 이송부로 잘 전달될 수 있는지 여부도 실험하였다.

<그림 1-19> 실험대상 포도

<그림 1-20> 실험 1

<그림 1-21> 실험 2



<그림 1-22> 실험 3

# 제 4 절 결과 및 고찰

## 1. 과경 절단의 최적 RPM

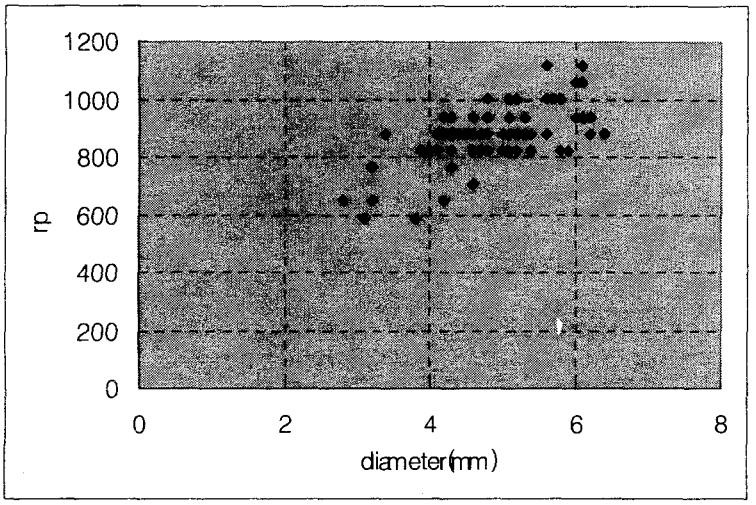
포도 과경을 최적 rpm으로 절단하기 위하여 rpm의 값을 일정 하게 증가시키고 과경의 위치를 임의로 조금씩 변화를 주어 실험하였다. 대부분의 포도 과경은 860rpm 정도에서 모두 절단되었으나 무게가 500g 이상이 되면서 훨씬 큰 rpm을 필요로 하고 있다. 또한 이를 수치상으로 알아보기 쉽게 하기 위하여 무게 따른 rpm의 평균값을 계산하여 보았다.

<표 1-1> 평균 작동 RPM

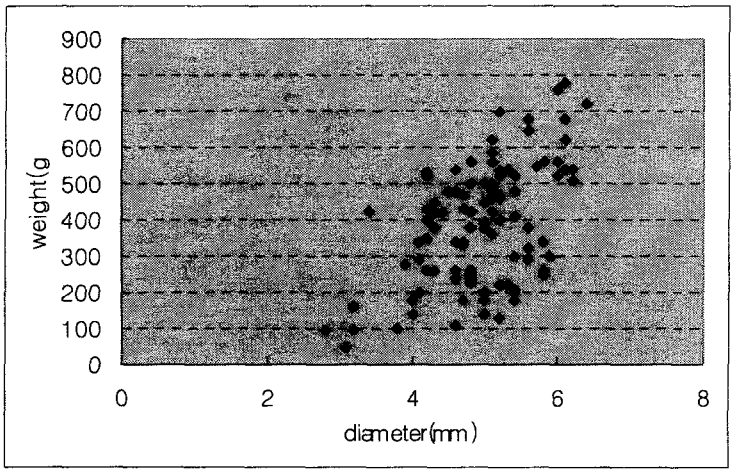
| Variable / Number | weight | rpm |
|---|---|---|
| 1 | 100 | 647.159 |
| 2 | 200 | 861.592 |
| 3 | 300 | 860.294 |
| 4 | 400 | 882.353 |
| 5 | 500 | 901.961 |
| 6 | 600 | 955.882 |
| 7 | 700 | 976.471 |

<표 1-1>에서 보여지는 것과 같이 rpm을 결정하는데 있어서 과실의 무게가 큰 영향을 주지는 못하였다. 이는 절단부의 절단부 이송모터가 일정한 속도로 움직였기 때문이라고 여겨진다. 절단모터의 높은 rpm에 의해서도 과경을 모두 절단되지 않는 경우가 종종 발생하였다. 따라서 과경이 두꺼운 경우, 보다 느린 속도로 움직여 과경이 절단될 여유 시간을 주어야 했다. 즉, 절단부 이송모터는 과경의 두께나 질긴 정도에 따라 속도가 제어될 필요가 있었다. 100g 이하의 과실에 있어서는 예상 밖으로 아주 낮은 rpm만을 요구하였다. 이는 일반 포도 줄기와 달리 가지가 굳어지기 전 아주 연약한 과경임을 의미한

다. 반면 500g 이상의 포도의 경우 훨씬 높은 rpm을 요구하는데 이는 무거운 과실을 지탱하기 위해서 과경이 더 질겨졌기 때문일 것이다.




<그림 1-23> 지름에 따른 rpm



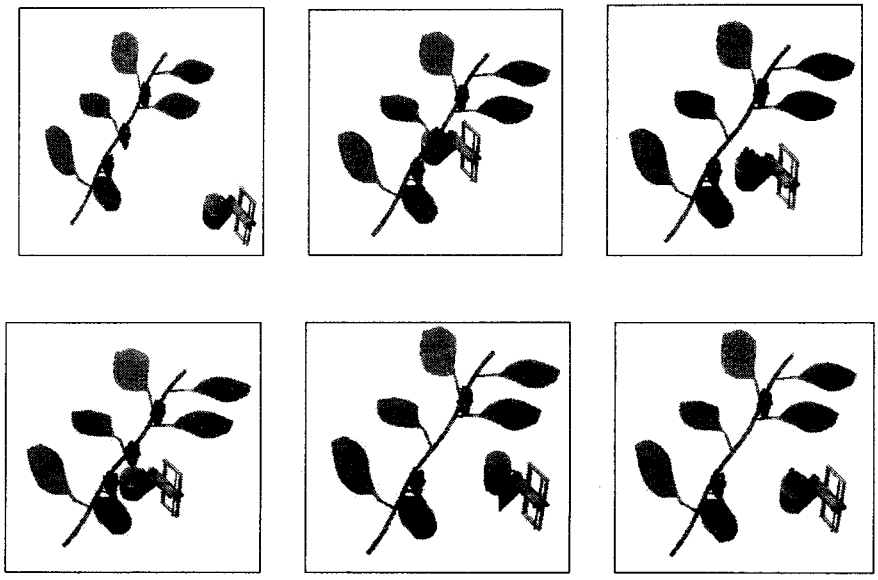

<그림 1-24> 지름과 무게의 관계

위 결과에서 과경의 지름이 4mm이상인 경우 800rpm이상의 속도가 필요하

다. 또한 과경의 직경이 두꺼워졌다고 해서 무게의 변화가 갑자기 변화한 것이 아니므로 과경 자체의 변화가 있는 것으로 여겨진다.

2. 수납부의 이송부로의 전달

수납부는 중점을 포도가 수납부로 정확히 수납부에 떨어지는가와 포도에 손상이 적게 떨어지는가에 중점을 두고 측정하였다. 수납부는 ABS수지를 이용하여 가이드를 대고 가이드에 가이드와 직접적인 접촉을 피해 망을 고정시켜 가이드의 휨방지와  망을 사용하여 충분한 충격흡수 작용을 할 수 있게 하였다. 또한 실험에 사용된 포도의 최대 길이가 대략 150mm인 점을 감안하여 첫 번째 수납부의 길이가 180mm인데 비해 두 번째 장치는 120mm로 짧게하여 보다 낙차가 적고 부드럽게 이송부에 전달되도록 만들었다. 두 번째 실험에서는 충격흡수와 이송부로의 전달면에서 손상없이 부드럽게 전송할 수 있었다.



<그림 1-25> 작동순서

3. 엔드이펙터의 수확작업 시 적합성

상품성이 있다고 여겨지는 거봉 포도의 경우 대략 400g ~ 800g 정도이며 과경의 지름은 400 ~ 600mm정도이다. 그리고 엔드이펙터가 작업할 수 있는 과경의 길이는 20mm이상이 되어야 포도와 줄기에 손상을 입히지 않고 작업이 가능하다. 모든 포도를 절단, 운반하기 위해서 수직이송모터는 최대 속도로 상승하여 포도의 과경 위치를 파악하고 절단모터는 적정 속도는 1200rpm이상 되어야 모든 과경를 무리 없이 절단할 수 있을 것이다. 톱날집 전진을 위한 속도는 6.25mm/sec였으나 이보다 조금 느린속도 즉, 4mm/sec의 속도로 줄여 과경이 잘릴 충분한 여유를 주어야 한다. 수납부의 경우 어망의 길이는 120mm 정도 인데 실험에 있어 모든 포도를 안전하게 받아냈으며 이송부 전달시 크기에 상관없이 모두를 전달하는데 성공하였다.

# 제 5 절 요약 및 결론

본 연구는 과경을 절단할 수 있는 절단부와 과실의 손상을 최소화하여 과실을 받고 운반할 수 있는 수납부로 이루어진 포도 수확용 로봇의 엔드이펙터를 개발하는 것이다.

본 연구에서 제작된 포도 수확용 로봇의 엔드이펙트는 제어를 쉽게 하기위해 DC모터를 사용하였으며 내구성과 무게를 고려하여 ABS수지와 드랄미늄을 사용하여 제작하였다. 또한 과경의 절단부위의 높이를 측정하기 위해 광센서를 사용하였다. 수납부는 수납망을 설치하고 가이드를 만들어 포도의 손상을 적게 하고 이송부로 부드럽게 전달되도록 하였다.

주요 연구결과를 요약하면 다음과 같다.

1. 절단부에서의 절단속도는 과실의 무게와 과경의 굵기를 감안하여 1100rpm에서 1200rpm이 가장 적당한 절단부 속도이다.

2. 절단부의 전진속도를 결정할 때 톱날이 과경에 낌을 방지하기 위해 과경의 굵기를 고려하여 속도를 조절하여야 한다.

3. 포도를 절단하여 절단된 포도를 수납 할 수 있는 수납부의 길이는 100mm에서 120mm가 적당하며 수납부의 설계시 가이드가 휨이 없고 포도에 충격이 없게 고려하여 설계하여야 한다.

# 제 2 장 : 영상처리 시스템

## 제 1 절 서 론

현재 포도의 수확에는 대부분이 많은 노동력이 소요되는 수(手)작업으로 이루어지고 있다. 포도는 재배 수형과 종류에 따라 결과의 높이에 차이는 있지만, 대부분 지상(地上)에서 50cm~150cm내외에서 결과(結果)한다. 또한 수확시 과경의 경질화에 따라 수확시 많은 노동력을 필요로 한다. 특히 포도 과육은 높은 함수율로 인해서 수확시 상품가치의 하락을 방지하기 위해서 많은 주위가 필요하다. 따라서 각각의 독립된 수확을 하지 않으면 안 된다. 이에 따른 연속된 단순 반복 작업으로 피로도는 더욱 증가될 수밖에 없는 실정으로 재배에 많은 어려움을 낳고 있다. 수작업을 통한 포도의 수확작업에는 한 손으로 과육을 잡고, 과경의 절단 위치를 눈으로 보거나, 감지하여 다른 한 손으로 도구를 이용하여 절단한다. 이러한 작업을 자동적으로 작업할 수 있는 포도 수확용 로봇의 개발이 필요하다.

포도 수확용 로봇을 개발하기 위해서는 선행적으로 포도의 형상과 위치를 정확하게 파악하는 것이 필요하다. 일반적으로 포도의 수확은 품종에 따라 차이가 있지만, 대부분 수확시 과일의 색이 변한다. 따라서 포도 수확용 로봇 개발에 있어서 포도 인식을 위한 영상처리 시스템은 포도인식을 위한 칼라 영상처리를 이용하였다. 또한 포도인식을 위한 알고리즘에 포도의 기하학적인 형상도 고려하였다. 2차원내의 포도인식은 입력된 영상의 색상차이에 따른 분석이 가능하다. 그러나 3차원공간상의 포도의 거리 정보를 찾기 위해서는 두 대의 카메라, 또는 다른 2가지 형태의 입력 영상을 분석하지 않으면 안 된다.

포도 수확용 로봇을 개발하기 위해서는 선행적으로 포도의 형상과 위치를 정확하게 파악하는 것이 필요하다. 따라서 카메라로부터 얻어진 영상을 해석하

여 포도형상 및 위치를 인식하는 영상처리 알고리즘을 개발하는데 그 목적이 있다.

본 연구목적은 3차원 공간상의 포도를 카메라로부터 얻어진 2차원 영상으로부터 포도를 정확히 인식하고 인식된 포도를 정적인 물체의 3차원 시각을 고려하여 신속하고 정확하게 거리정보를 획득할 수 있는 3차원 시각장치를 구현하는데 목적이 있다. 본 연구를 수행함에 있어 세부적인 목표는 다음과 같다.

첫째, 2차원공간으로 표현된 포도 영상으로부터 포도의 형상을 고려하여 포도 인식 알고리즘을 구현하는데 있다.

둘째, 인식된 포도의 거리 정보를 획득하기 위해 인간의 시각과 유사한 형태의 스테레오 시각시스템을 설계하고, 스테레오 영상을 이용하여 대상체의 거리정보를 삼각측량법을 이용하여 거리 정보를 추출하기 위한 알고리즘을 제시하고자 한다.

셋째, 개발된 실험장치 및 알고리즘을 포도 농가에 적용하여 현장실험을 통한 포도 인식 알고리즘을 검증하고자 한다.

## 제 2 절 연구동향 및 문헌조사

### 1. 연구동향

컴퓨터 시각장치에 의한 3차원 정보를 이용한 연구현황으로서는 최근에 많이 연구되어지고 있으며, 그 중에서 가장 일반적인 방법으로 사람의 눈과 비슷한 구조를 가진 스테레오 시각을 이용하는 방법이 있다. 이는 한 대의 카메라를 통해 얻어지는 3차원 형상 정보는 3차원 대상체가 2차원 평면상에 투사되어 얻어진 정보이므로 대상체가 갖는 3차원 공간 정보를 얻기 위한 접근법이 필요하다. 스테레오 시각(stereo vision)은 각각의 카메라를 통하여 2차원 영상을 얻고 이들간에 삼각측량법을 이용하여 3차원 정보를 산출하는 것이다.

그 방법으로서 두 단계가 있는데, 첫째 단계는 두 영상에서 선택된 점들의 대응관계(correspondence)를 가지는 가를 판단하고 그 점에서의 깊이정보를 계산하는 일이다. 둘째 단계는 보간법 등을 적용하여 영상의 모든 점에 대하여 깊이정보 맵을 구하고 또 그것을 이용하여 3차원 공간의 표면을 기술하는 것이다. 하지만 스테레오 방법의 단점으로서는 다음과 같은 단점을 내포하고 있는 실정이다. 1) 한쪽 카메라의 임의 점과 다른 카메라의 대응점 추출을 위한 영상좌표계상의 매핑 문제. 2) 촛점거리 및 촛점각을 계산하기 쉽지 않다. 3) 획득된 거리영상은 감지 대상의 기하학적인 특징 때문에 물체의 심도정보뿐만 아니라 많은 잡음을 포함하고 있다. 그러므로 전처리 과정이 필요하다. 4) 연산시간이 길며 대상체에 대한 정확한 사전 정보가 요구되기 때문에 유연성 및 일반성이 결여되어 있다.

이러한 어려운 문제를 해결하기 위해 최근에는 고성능의 프로세서가 개발되어 계산시간문제가 해결되고 강력한 고성능 스테레오 모델이 개발됨으로써 특수 목적을 위한 실용적인 스테레오 비젼 시스템이 등장하고 있다.

## 2 문헌개요

대상체에 대한 3차원 정보를 얻기 위한 방법 중 사람의 시각구성과 유사한 구조를 가진 스테레오 방법은 2개의 카메라 렌즈를 통하여 얻어진 영상을 이용하여 3차원 거리정보를 얻고, 표면의 재구성을 추출하는 것이다. 이는 각각의 영상에 존재하는 한 점의 거리와 서로 다른 촛점에서 얻은 두 영상에서의 대응되는 상대적 차이에 의해 계산하며, 특징점들의 변위로 부터 거리지도(depth map)를 얻어 3차원 표면을 재구상 한다. 또한, 인간의 시각시스템에서 사용하는 자유도를 따라, 능동적인 스테레오 시각장치를 구현하는 연구가 최근에, 유럽과 일본에서 진행중이며, 많은 연구단체에서 이 분야에 서로 다른 이론과 실험을 하고 있다. 액티브 비젼 연구는 지각시스템을 제어하여 스테레오 구성 배경에 나타내며, 각각의 서로 다른 영상에 물체의 거리를 계산할 수 있

도록 구현하여 대응배합이 초점절차에 의해 얻어질 때마다 거리를 구할 수 있도록 구현한다. 또한 카메라와 카메라렌즈의 컴퓨터 제어 사이에 각의 변화조절이 거리를 결정하는데 용이하도록 설계 및 알고리즘을 구현하여 지각 시스템을 고려한 내적 및 외적 매개변수의 변화에 용이하도록 연구가 진행중이다.

Enrico Grosso(1995)등은 능동적/동적 스테레오 시각(Active/Dynamic Stereo Vision)에 대해 로봇작동을 위한 적절한 시각정보의 특징점 문제에 대해 언급하였다. 그는 스테레오 이미지에서 변위는 배경에서 상대적 거리지도를 얻기 위해 광류를 결합시킨다. 배경에서 고정점으로 부터 카메라의 거리에 의한 깊이는 중심뿐만이 아니라 상대적으로 측정된다. Narendra, Lynn Abbott(1993)은 능동시각(Active stereo): 변위구성(Integrating Disparity), 한계(Vergence), 초점(Focus), 검색(Aperture), 표면추정을 위한 계산에 대해 연구하였다. 그는 깊이정보의 근원으로서 스테레오 변위(disparity)를 강조하였다. 카메라의 한계와 렌즈촛점은 깊이의 복구를 위해 이용되며 이에 대한 구성(Integration)방법은 배경 기술을 전개하는 것을 바탕으로 능동적 이미지의 변수선택(한계, 초점, 검색, 그리고 줌)에 의해 이루어진다. 접근하는 두 단계로서는 시각적인 추적선택(visual target)과 표면복구(surface reconstruction)가 있다. 이 방법의 구현에 의한 실험 결과로서는 정확한 표면이 충분하게 시각적으로 상세하게 나타났다. 실행한 분석은 거의 2m의 거리에서 0.15%의 평균에러를 나타내며, 불연속적인 깊이를 포함한 배경에 대한 표면의 재구성은 더 복잡하게 구현된다. John Aloimonos(1987)등은 능동시각(Active Vision)의 구현에 있어, 기본적인 몇 가지의 문제점을 나타내었다. 소위 능동(active)이라 불리는 여러 종류의 활동은 센서장치의 기하학 매개변수를 제어하는데 목적이 있다. 능동의 목적은 지각처리(perceptual)결과의 질을 향상시키기 위해 관찰적 현상을 조종하게 다루는 것이다. 본 연구에서는 기본적인 비젼문제에서 수동적(passive)보다 능동적(active)관찰이 더 효과적인 방법으로 해결할 수 있는 것을 증명하였다. 특히 영상내의 각 점에서의 밝기는 광원의 위치와 물체표면의 기울기로부터 얻어진다는 밝기 해석법을 지닌 명암의 형상복구(shape from

shading)와 거리계산(depth computation), 등고선의 형상복구(shape from contour), 무늬상태의 형상복구(shape from texture), 움직임으로부터 기하학구조(structure from motion)은 수동적보다 능동적 관찰이 더 간단함을 보여준다. Jorge Dias, Carlos Paredes 등은(1998) 이동로봇과 인공시각을 가진 시스템을 이용하여 물체를 추적하기 위한 시뮬레이션 추적실험을 연구하였다. 그는 이동로봇(mobile robot)과 능동 시각시스템(active vision system)을 이용해서 평면에서 물체를 따라 추적하는 방법을 연구하였다. 이 해결방법은 시각시스템을 가지고 로봇을 제어하는 비쥬얼 피드백(visual feedback)을 이용하여 각각의 제어 시스템의 상호작용에 의해 처리되고 이동로봇의 시각적 상호작용 방법에 의해 수행되며, 이러한 두 가지 시스템은 일정한 거리에 움직이는 물체를 따르는 것과 이동로봇에 관해 방향을 따르는 것으로 설계되어 있다. 능동시각시스템(active vision system)의 방향과 위치는 실시간으로 목표물(target)을 추적하기에 이용되는 제어에 대한 피드백 신호이다. 이 연구에서는 비쥬얼 피드백(visual feedback)과 시스템의 움직임을 위한 보정(compensation)을 이용하여 시각적 고정, 시각적 추적, 작동의 문제를 처리한다. 능동시각시스템(active vision system)으로 갖추어진 이동로봇은 목표물을 따라 움직이는 추적장치의 시스템 사이에서 문제를 처리한다. 이 시스템 구성은 두 관점을 가지고 있는데, 각각의 제어시스템 사이에서 상호작용과 시각시스템에 의해 제공된 일반적인 피드백 정보의 사용이다. 이 연구에서의 전체적인 해결방안은 이동로봇의 앞에 움직이는 목표물에 대한 추적 설계를 구현하기 위해 시각적 처리에 근거를 두는데 목적이 있다. 또한 시각시스템을 이용해서 물체의 추적을 실시간(real-time)으로 처리하도록 하고 있다. 몇몇의 작업은 카메라 이미지상의 변화와 카메라 자세 변화와의 관계인 Jacobian을 가지고 로봇을 제어하는 비쥬얼 서보기구(visual servoing)-간접조속장치-에 의해 처리된다. 이 방법은 이미지상의 변화로부터 카메라 자세의 변화를 구할 수 있는 특징을 가지고 있다. Coombs(1992)는 양안시 시각(binocular vision)을 이용하여 실시간으로 시각처리를 구현하도록 연구하였다. 이 시스템은 매니퓰레이터 위에 양안시 시스템을

장착하여 두 이미지의 정보를 시각적처리로 제어되도록 설계하였다. 이러한 연구는 지연시간을 없애기 위한 전조적인 제어설계를 포함하고 있다.

Papanikolopoulos(1993)는 궤도가 이미지 평면에 평행한 물체를 추적하기 위해 매니퓰레이터에 카메라를 장착하는 방법으로 추적처리를 제안하였다. 시각시스템에 의해 충족된 정보는 광류(optical flow)를 기본으로 하는 알고리즘에 의해 처리되는데, 이는 실시간으로 물체를 추적하기에 충분한 정확도를 지니고 있으며, 윈도우의 이미지에서 효과적인 사용은 이러한 방법을 더 향상시킬수 있다고 하였다. Allen(1993)은 3차원적으로 물체를 따라 추적하는 방법으로 시각시스템은 두대의 카메라를 사용하여, 물체에 대해 매니퓰레이터의 시스템으로부터 계산되어진 스테레오이미지를 이용함으로써 실시간으로 계산되어진다. 일단 추적하는 단계가 이루어지면, 매니퓰레이터에 제어되는 시스템은 움직이는 물체를 구분하여 그것을 집게 된다.

최근에는 능동적 시각 원칙(active vision principles)을 기반으로 하여 시스템의 개발을 위해 전 유럽적인 계획(Large European Project)-Vision as Process Project-에 관여하고 있다. 시스템의 발달을 위한 연구의 수행으로는, 카메라 제어, 시각 반사광(ocular reflexes), 실시간 영상처리, 이미지추적, 지각처리방식화(perceptual grouping), 능동적 3차원 모델링(active 3-D Modeling), 그리고 물체인식 들의 연구가 수행되고 있다. 국내에 의해 보고된 연구로서는 김등(1998)은 능동 시각을 위한 자동 초점 시스템을 개발하였다. 그의 연구는 능동 시각 시스템에서의 자동 초점 조절을 위하여 기존의 적외선을 사용한 초점 조절방식이나, 아날로그 신호의 처리방식과 달리 디지털 영상을 바탕으로 한 초점 조절을 시도하였다. 초점 조절에 필수적인 초점값 산출을 위해 잡영에 민감한 일반적인 고대역 통과 필터를 개선한 이동평균 필터로 산출된 초점값은 CCD카메라의 초점조절에 있어 잡영의 영향을 덜 받으며, 하드웨어 구현을 간소화하도록 각종 초점 알고리즘을 실현할수 있는 시스템을 구현하였다. 심등(1994)은 움직이는 물체의 파지를 위하여 스테레오 카메라를 사용한 로봇 매니퓰레이터의 위치 및 자세 제어를 시뮬레이션을 통해 유효성을 확인하였다. 작

업대상이 이동물체의 형상을 사전에 알고 있다는 가정하에서 스테레오 비젼과 이미지의 Jacobian을 이용하여 움직이는 물체의 파지를 위한 End-effector의 위치 및 방향제어를 시뮬레이션하였다. 시뮬레이션 결과에 의하면 이동하는 물체의 속도와 위치에 상관없이, 로봇 매니퓰레이터의 End-effector의 자세를 이동물체의 자세로 카메라 이미지상의 변화와 카메라 자세 변화와의 관계인 Jacobian을 가지고 로봇을 제어하는 Visual feedback servoing방법의 유효성을 확인하였다. 손등(1997)은 스테레오 영상처리를 이용한 토마토 위치검출 알고리즘을 개발하였는데. 이 연구에 의하면 토마토 수확로봇개발을 위한 시각구성으로서 스테레오 영상처리를 이용하여 토마토의 3차원 위치정보를 얻기 위해 먼저 토마토의 중심좌표를 계산하여 거리계산을 하도록 연구하였다. 또한 최적의 두 카메라간 거리 결정이 100mm로 구성할 때 카메라 공유면적이 가장 넓은 것으로 분석되었고, 조도 및 광원의 위치가 위치검출에 미치는 영향에서는 순광에 비해 역광에 대한 검출오차가 더 크고 조도에서는 100Lux이하에서 물체인식이 부정확하여 오차가 크게 나타난 것으로 분석되었다. 장등(1998)은 사과수확로봇의 개발을 위한 화상처리를 이용한 사과의 인식 알고리즘을 보고하였다. 대상물 인식을 위한 위치결정 알고리즘을 개발하였는데 색상함수를 구함으로써 대상물을 인식하고 장애물을 검출할 수 있도록 하였다. 사과 인식을 위한 변환된 결정함수 X=(R-G+256)/2 로 사과를 인식하고 상관계수를 사용한 템플레이트 매칭으로 사과의 인식과 좌표를 결정하여 장애물의 검출을 위해서 색상비율에 따라 가지와 잎의 장애물을 쉽게 검출할수 있다고 보고하였다. 특히 사과수확의 경우 사과의 색과 잎의 색은 확연히 다르기 때문에 사과가 잎에 가려져 있더라도 각각의 색상비율에 의해 사과를 인식할수 있을 것으로 사료되지만 오이수확의 경우에 있어서는 거의 색상비율의 차이가 없기 때문에 오이수확에는 적용이 어려울것으로 사료된다. 위와 같이 스테레오비젼에 의한 방법은 사람의 인공시각과 유사한 형태로 해석하기 위해 능동시각시스템(Active stereo vision)에 대한 연구가 많이 이루어지고 있으나, 카메라에 의한 외적요소 및 초점대응거리 등의 많은 문제를 해결하고 있는 실정이다. 또한

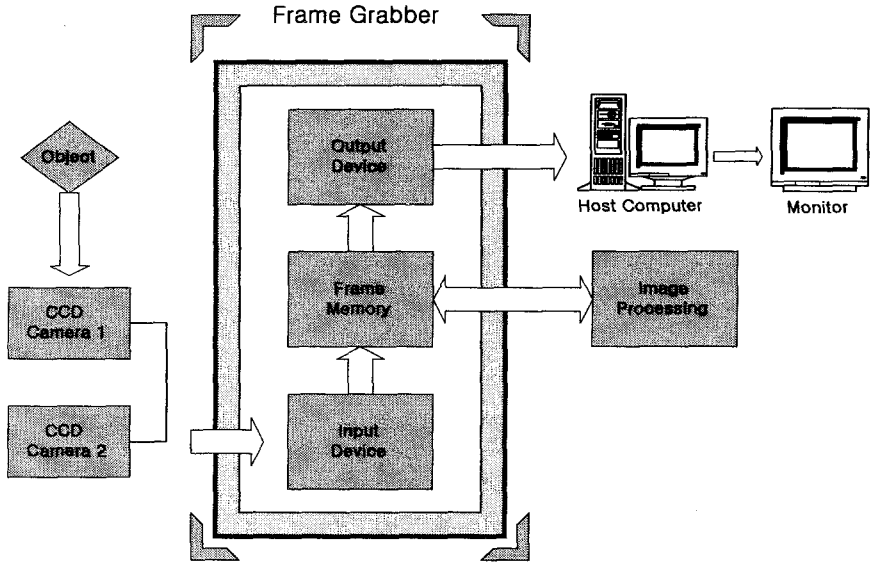
카메라와 다른 센서 즉 초음파,레이저 등을 이용하여 물체의 3차원 정보추출에 관해 많은 연구가 이루어지고 있다.

## 제 3 절 실험장치 및 방법

### 1. 영상처리 시스템

#### 가. 시스템 개요

일반적으로 사용된 영상처리 시스템의 기능 블록은 그림과 같이 영상신호 입력부, 영상처리부, 주 컴퓨터 및 영상출력부로 구성된다. 컬러 CCD카메라로부터 입력된 영상신호는 RS-170 형태의 아날로그 신호로 컴퓨터에 변환된다.




<그림 2-1> 영상처리시스템 Block Diagram

#### 나. 영상 입력부

입력 센서부에 해당하는 영상 입력장치로는 4.8mm렌즈를 부착한 Ikegami 사의 컬러 CCD 카메라(ICD-703) 2대를 사용한다. 본 연구에서 사용될 카메

라의 제원은 다음과 같다.

<표 2-1> CCD Camera 사양

| Item | Specification | |
|---|---|---|
| CCD camera (Model: ICD-703) | T V | NTSC standard 525Lines |
| | | 60 Field/sec 2:1 |
| | Image sensor | 1/3inch interline transfer CCD |
| | Pixel elements | 771(H)×492(V) |
| | Resolution | 460TV Lines |
| | S/N ratio | 50dB(AGC OFF) |

다. 신호 처리부

RS-170 12.28Mhz의 아날로그 신호 값으로 CCD 카메라에서 출력되는 영상 신호는 컴퓨터와의 인터페이스를 위하여 디지탈 신호값으로 변환되어야 한다. 영상변환은 영상처리보드의 디지타이져(digitizer)에 의해 디지탈 신호로 변환되는데 이 신호값을 이용하여 원하는 처리를 소프트웨어적으로 수행할 수 있게 된다. 컴퓨터 프로그램에 의해 수행된 영상처리 결과는 영상메모리 내부의 LUT(Look Up Table)를 조작함으로써 영상출력 전용의 모니터상에서 확인할 수 있다. 이를 위하여 본 연구에서는 Coreco사의 OCULSU TCI-SE PCI Frame Grabber(CORECO Inc., St. Laurent, CANADA) 를 사용한다. 이 장치는 A/D Convertor 가 내장된 Input Device, Processing Device, Frame Buffer Memory 가 포함된 Storage block, D/A Convertor가 내장된 Output Device 등으로 구성되어 있다. 본 연구에서 사용된 신호처리부의 제원은 <표 2-2>와 같다.

<표 2-2> OCULUS TCI-SE True Color Frame Grabber Specification

| | |
|---|---|
| o IMAGE MEMORY | 2MB |
| o IMAGE FRAME | Linear Memory |
| o DISPLAY | Uses Host Display |
| o OVERLAY MEMORY | None |
| o BIT DEPTH | 15 or 24 bit per pixel |
| o SOURCE | RS-179, CCIR, NTSC, PAL |
| o Inputs | 2 or 3 Composite inputs |
| o RESOLUTION | 640×480×30 Frame/sec |

라. 출력부

처리된 결과값과 영상 출력결과는 PC용 모니터로 출력한다. 이는 영상처리부를 통해 처리된 데이터를 R, G, B영상으로 출력한다. 본 연구에서 사용된 출력부의 제원은 다음과 같다.

<표 2-3> VGA Color Monitor Specification

| | |
|---|---|
| o Manufacturer | Samsung |
| o Model No. | SyncMaster 17GLi |
| o Resolution | 1024×768 |
| o Frequency | 60Hz |

2. 스테레오 비젼 시스템(Stereo Vision)

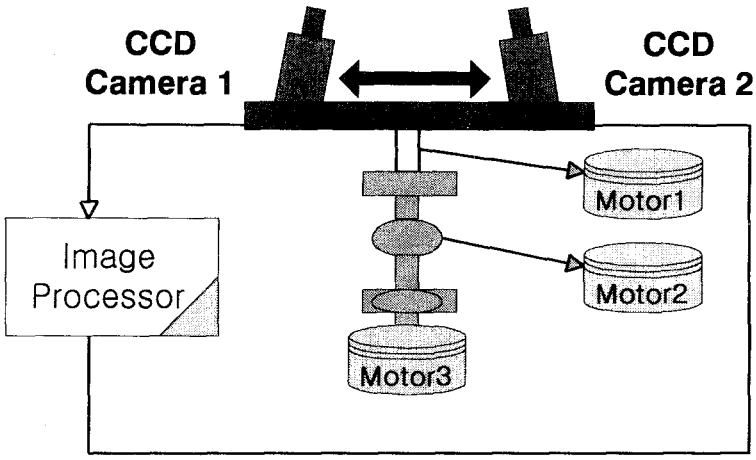
가. 이론적 배경

스테레오 비젼 연구는 지각시스템의 매개변수변화를 제어하는 것과 관련되며 이 연구는 초기에 Pennsylvania대학에서 R. Bajcsy에 의해 제안되었다. 이 접근은 후에 E. Krotkow가 스테레오 구성을 구현하여 각각의 서로 다른 영상에 배경물체의 거리를 계산할 수 있도록 함으로써, 각각의 이미지에 대한 대응매칭이 초점절차에 의해 얻어 질 때마다 거리를 구할 수 있도록 구현하였다.

Krotkow에 의해 제시된 시스템은 카메라와 카메라렌즈의 컴퓨터 제어 사이에 각의 변화조절이 거리를 결정하는데 용이하도록 되어 있으며, 후에 이 연구는 지각 매개변수(sensory parameters)의 변화로부터 알고리즘을 제시하였다. 최근에 이 연구는 지각 시스템을 고려한 내적 및 외적 매개변수의 변화를 용이하게 실행하도록 연구가 진행중이다. 내적 매개변수(촛점, 초점거리,)는 모터가 달린 렌즈를 사용함으로 제어할 수 있고 외적 매개변수제어는 인간의 시각 시스템에서 사용하는 자유도를 따라서 나누어진다. 인간은 두 눈이 수평선상 -시각범위(eye vergence), 수직적-시각 팬(eye panning)위치제어를 한다. 게다가 인간은 각각의 눈이 광학축(cyclotorsion)을 둘레로 회전할 수도 있다. 이러한 3 자유도는 시각구성의 조건으로 정의한다. 게다가 목(neck)의 움직임(pan and tilt)과 몸체 운동(3차원 공간 위치변화)은 시점 선택에서 사용될 수 있다.

나. 기하학 구조 (Vision system Geometry)

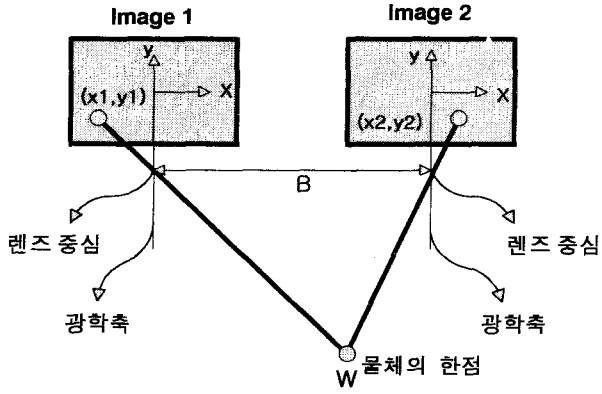
시각 시스템은 아래 그림과 같은 구조와 같이 카메라 2대를 장착하고 사람과 유사한 방식처럼 3자유도가 가능하게 설계되어진다. 능동 시각 시스템은 On-board컴퓨터에 연결하여 동적제어가 가능하게 설계되며 2대의 카메라로 이미지를 처리하도록 연결되어 진다. 본 연구의 능동시각시스템은 3대의 스테핑모터를 이용하였고 2대의 컬러CCD Camera는 각각의 카메라의 범위각제어(vergence control)가 가능하도록 설계하였으며, Head Tilt, Neck Pan이 작동되도록 구현하였다. 또한 이 시스템의 관리와 인터페이스는 컴퓨터에 의해 처리되고 주처리장치에 연결된다.



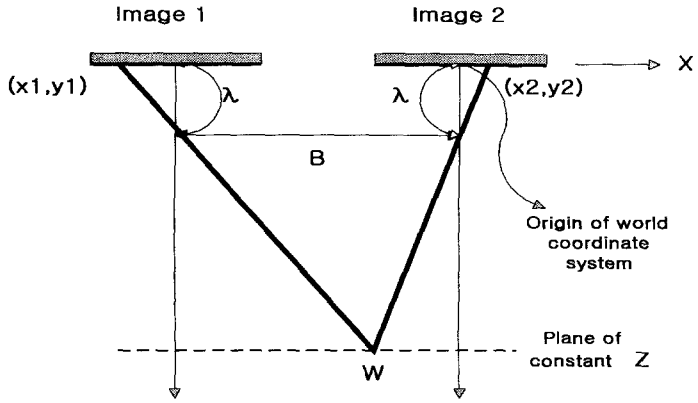

<그림 2-2> 스트레오 비젼 시스템 개략도

다. 거리정보 및 카메라의 범위각(Vergence angle)제어

1) 스테레오 영상의 거리정보추출




<그림 2-3> 스테레오 영상 처리의 모델




<그림 2-4> 스테레오 영상처리 모델의 단면도

스테레오 영상은 한 개의 물체의 점에 대하여 두 개의 카메라를 이용하여 영상을 따로 얻는 것을 말하는데, 두 카메라 렌즈의 중심과 중심을 연결한 선을 기저선(Baseline)이라 하고 그 거리는 B이다. 두 카메라간의 초점 거리는 모두 $\lambda$이다. 두 카메라는 동일하다고 가정하고 각 카메라의 x,y축과 물체의 x,y축은 평행하다고 하면 영상면의 한 점 $(x_1, y_1)$ 이고 물체의 첫 번째 영상의 좌표계와 동일한 것을 사용했을 때 그 점의 좌표를 $(u_1,\ v_1,\ w_1)$이라 하면

$$u_1 = \frac{x_1(\lambda - w_1)}{\lambda}$$

이다.

둘째 영상 좌표계의 영상 점의 좌표도 마찬가지로 $(x_2, y_2)$라 하고 물체점의 좌표를 $(u_2,\ v_2,\ w_2)$ 라 하면

$$u_2 = \frac{x_2(\lambda - w_2)}{\lambda}$$

이다.

첫 영상면의 x-y좌표계는 둘째 영상면의 x-y좌표계와 평행하지만 x축상으로 렌즈의 거리 B 만큼 떨어져 있으므로

$$u_2 = u_1 + B \quad \text{이고} \quad u_2 = w_1 = w$$

이다.

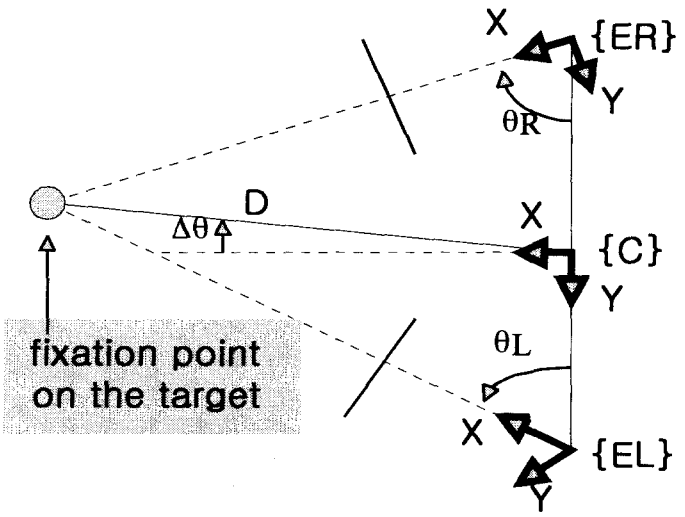
이를 사용하면 $u_1 = \frac{x_1}{\lambda}(\lambda - w)$

$$u_1 + B = \frac{x_2}{\lambda}(\lambda - w)$$

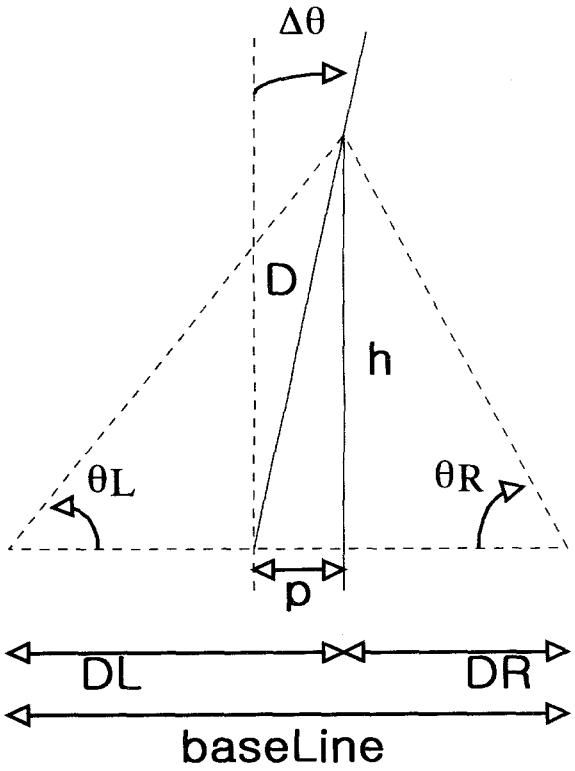
을 얻으며, 양변을 각각 감산하여

$$w = \lambda - \frac{\lambda B}{x_2 - x_1}$$

을 얻는다. 이것은 물체의 점에 대한 두 영상의 x좌표, $x_1$, $x_2$ 를 알면 깊이 w를 구할 수 있다.



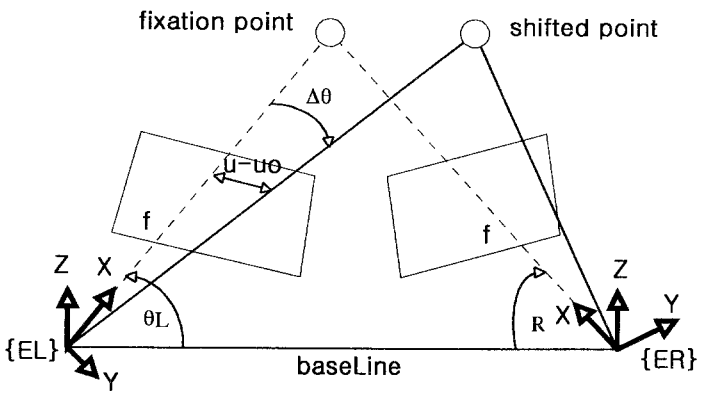

<그림 2-5> XY평면 C기준점에서의 물체의 거리와 각




<그림 2-6> 보조 매개변수

2) 카메라 범위각 제어

이미지에서 대상체의 위치정보는 비젼시스템의 위치와 방향제어에 이루어진다. 공간상에서 대상체의 위치변화는 이미지위치 역시 변화할 것이다. 이것은 두 개의 이미지 중심에서 대상체의 이미지투사를 하는 방법으로 제어하면 대상체의 거리를 구한다.




<그림 2-7> 카메라 범위각 제어

이미지 중심에서 대상체의 투사를 유지하는데에는 카메라각 $\theta L$ and $\theta R$ 이 변화할 것이다. 대상체의 변화시 수평적 부등(horizontal disparity) $\triangle u = (u - u0)$ 이며 x축의 기준점의 픽셀좌표를 u라 하면 각각의 카메라 각은 다음과 같이 변화할 것이다.

$$\triangle \theta = \arctan \frac{u - u0}{S_x f}$$

여기서 f는 초점길이(focus length)이고 $S_x, S_y$ 는 카메라의 본질적 매개변수라 불리우는 비례인자(scale factor)이다. 물체의 추적위치는 그림에서 보여주고 있는 기준점에 따른 고정점(두 개의 매개변수)을 이용해서 {C}기준점에 대해 거리 D와 각 $\triangle \theta$에 의해 알수 있다. 보조매개변수에 의한 관계에서 D와 $\triangle \theta$값의 방정식은 다음과 같이 얻을 수 있다.

$$h = \tan(\theta R)\ D_r$$

$$h = \tan(\theta L)\ D_r$$

$$B = DL + DR$$

$$P = DL - \frac{B}{2}$$

{C}기준에 관련한 고정점의 거리 D와 $\triangle\theta$ 는 다음의 방정식에 의해 얻을 수 있다.

$$\triangle\theta = 90^0 - \arctan(\frac{h}{p})$$

$$D = \sqrt{h^2 + p^2}$$

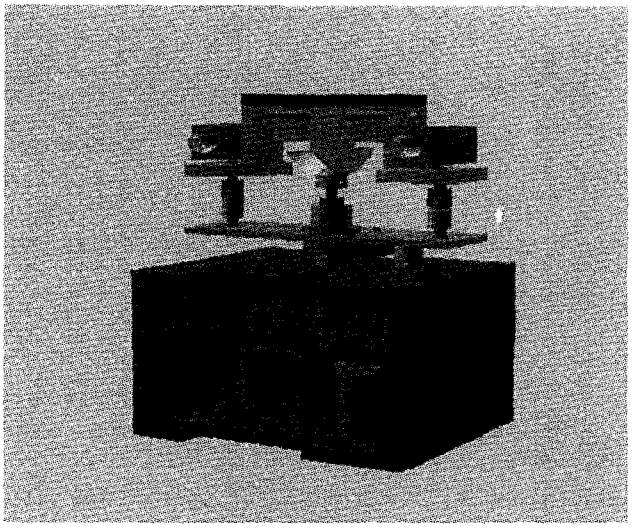
만약 $\theta L$ 이 $\theta R$과 같다면, 위의 관계들은 정확하지 않다. 이러한 경우에 $\triangle\theta$는 0이고, 거리 D는 다음과 같다.

$$D = \frac{B}{2}\ \tan(\theta L)$$

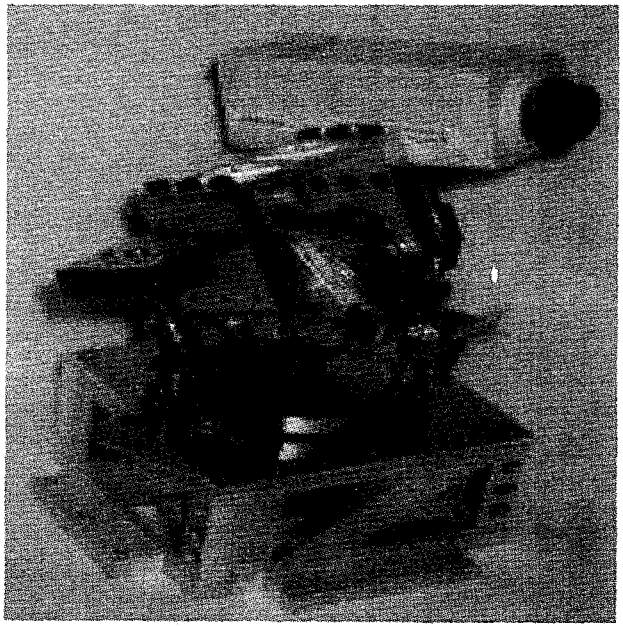
위의 서술된 수학식 표현에서, 두 카메라가 고정점에서 이동점으로의 움직임은 스테레오 방법에 의해 유도된 수학식에서 $\theta L$과 $\theta R$의 각을 가지고 대상체의 두 이미지에서 거리의 변화 D는 다음과 같이 접근된다.

$$D = \frac{B}{2}\ \tan(\theta L)$$

위의 <그림 2-2>에서 Motor1은 카메라의 범위각제어(Camera Vergence angle control)역활을 하며 Motor2는 카메라의 Head Tilt, Motor3는 카메라의 Neck Pan 역할을 한다. 각각의 모터의 제어는 one-cnip controller에 의해 제어되도록 설계되어 있다. 또한 두 카메라간의 거리는 135mm로 설계하여, 일정거리 만큼 떨어진 두 카메라 영상의 인식영역에 있어서 두 카메라간 거리가 멀때보다 100~150mm사이에서 넓은 인식영역을 나타냈다. 본 연구의 BaseLine은 135mm이며 3개의 스테핑모터를 이용하여 회전가능하게 설계하였다.



<그림 2-8> 스트레오 시스템의 입체도



<그림 2-9> 제작된 스트레오비전 시스템

3. 포도 영상처리

본 3차원 영상처리장치는 카메라의 PAN TILT 동작 이외에도 두 대 카메라의 포커스도 조절이 가능해 영상처리시 대상물체에 대한 DEEP IMFORMATION 추출은 물론 추출된 이미지에서 OBJECT외 배경의 삭제도 용이하도록 하였다. 제어장치 및 PIC 마이컴 제어용 프로그램 및 메인컴퓨터와의 통신프로그램도 완성되었으며, 앞으로 추출된 영상정보를 신속하게 처리할 수 있는 알고리즘을 개발이 필요할 것으로 판단된다.

가. 전처리 알고리즘

영상처리 알고리즘에서 전처리는 왜곡된 영상에서 보다 나은 영상데이타를 얻고자 하는데 있다. 전처리는 영상의 질을 개선하거나 영상을 특정한 응용목적에 알맞도록 변환시키는 등의 영상처리를 의미한다. 2대의 카메라에 의한 스테레오방법의 대상체 인식정보를 추출하기 위해 선행되어야 할 전처리 알고리즘으로서는 대상체와 배경을 분리함으로써 대상체에 대한 많은 정보를 얻기 위한 이치화 과정을 하고 경계검출 알고리즘을 사용할 것이다. 또한 두 카메라에 의한 스테레오 방법으로 대상체의 이미지를 그랩하여 이미지를 처리하기 위해, 배경으로부터 대상체 정보를 추출하는 알고리즘을 구현할 것이다 . 이는 두 대의 카메라로 추출하고자 하는 대상체를 그랩(Grab)하면 대상체 외의 배경의 이미지는 화소의 위치변화를 대상체보다 크게 나타난다. 두 대의 카메라에 의해 이미지의 대상체에서 큰 화소변화의 배경은 없애고 추출하고자 하는 특징점만을 형상정보로 얻기 위한 알고리즘을 구현하고자 한다.

1) 영상분할을 위한 이치화 방법

이치화를 보통 Binaruzation 또는 Thresholding 이라고 하며 크게 적용영역에 따라 두가지로 구분하면 Global thresholding 와 Local thresholding으로 나눌수 있다. 전자는 처리 범위가 전체 측정윈도우에 해당하며 후자는 부분적인 측정윈도우를 지정함으로써 수행된다. 특히 특정영상을 분석하고 측정하고

자 할때는 대상물체에 따라 효율적으로 물체를 분리할 수 있는 국부이치화 방법에 대한 연구가 중요하다.

입력영상을 f(x,y), 출력영상을 g(x,y) 라고 정의하면 화소의 좌표 x, y에 대한 이치화 기준은 경계값 T보다 클 경우를 1로 나타내고 반대의 경우는 0으로 정하게 된다. 여기서 0은 물체(또는 배경)을 1은 배경(또는 물체)을 나타낸다.

$$g(x,y) = \begin{cases} 1 : f(x,y) \geq T \\ 0 : f(x,y) < T \end{cases}$$

여기서 T는 Brightness level 로 배경부분과 추출할 패턴을 분리하는 이 변수를 Threshold value 라고 하는데 이 변수를 어떻게 화상의 히스토그램 분석에 활용하여 결정하는 것이 문제로 되어 있다. 이 변수는 Manual 조작에 의해 임의로 설정하여 줄 수도 있고 또는 2개의 Windows설정에 의해 경계치를 결정하는 Windows Thresholding Method 알고리즘과 가상의 모집단 분류에 의해 2모집단의 분산값을 최대로 하는 통계적 판별분석법을 이용하여 Threshold Value를 정할 수 있다. 본 연구에서는 정적 이미지를 처리하도록 bitmap함수로 구현하였고 동적 이미지에서는 그레버보드(Grabber Board)의 FrameBuffer 함수를 직접 이용하여 이미지 처리가 가능하게 이치화를 하도록 구현하였다.

2) 에지 중심(edge based)분할 방법

에지(Edge)는 서로 다른 명암도를 갖는 영역간의 경계에 위치하는 점이다. 그러므로 영상함수 f(x)에 대해, 에지에서의 기울기 df/dx 는 명암이 균일한 지역에서의 기울기와 현저히 비교된다. 그러므로, 도함수(Derivative)를 계산하고 임계치(Threshold)를 사용하여 에지점(edge point)들을 검출할 수 있다. 그러나 잡음(noise)이 존재하는 경우에, 이 에지검출자는 많은 오인된 에지들

을 검출하게 되는데 좀 더 나은 결과를 위해 화소단위로 에지크기를 측정하는 대신에 영역 단위로 에지크기를 측정하는 방법을 사용한다.

본 전처리 과정은 형상정보를 얻기 위해 이치화와 경계검출과정을 실행함으로써 좀더 물체정보를 정확히 검출하는데 목적이 있다. 특히 서로 다른 명암도를 갖는 영역간의 경계를 이치화 및 경계검출 알고리즘을 사용하면 더 나은 형상정보추출을 얻을 수 있다.

에지는 두 영역(region)의 경계에 위치하는 점들을 말하며, 영역간의 경계부분은 한 영상안에서 명암도의 불연속으로 나타난다. 이는 두 영역의 경계(boundary)에 의해 대상을 표현하기 위한 방법이다. 주로 영상에 있는 명암값(gray-level value)들로부터 직접 대상들의 경계를 찾아내기 위해 중간단계의 에지(edge)영상으로 변환(transform)하고나서 더 정확한 경계 영상을 구성하는 방법으로, 명암을 가지고 에지들을 찾아내는 것이 일반적인 방법이다.

에지 연산자들의 공통적인 특징은 연산자들이 최대 명암 변화의 방향(direction)과 이 변화의 정도를 나타내는 크기(magnitude)를 계산하는 것이다. 미분 연산자의 에지는 서로 다른 명암도를 갖는 영역간의 경계에 위치하는 점이다. 도함수를 계산하고 임계치(Threshold)를 사용하여 에지점들을 검출하게 되는데 좀 더 나은 결과를 위해 화소단위로 에지크기를 측정하는 대신에 영역 단위로 에지크기를 측정하는 방법을 사용한다. 또한 라플라시안은 선의 끝과 모퉁이에 특히 민감한 반응을 보이는데 경사진 곳의 견각(shoulder)에서 민감한 반응을 보이고, 그 곳에서 영점교차(zero-crossing)를 야기하며, 영점교차선의 경사도로서 에지크기를 나타낸다. Sobel 연산자는 해당 에지주변의 점을 에지로 검출하는 성질을 갖는다. 이러한 성질은 에지의 확대 및 축소를 야기시키며 Template Matching 알고리즘도 다소 유사한 특성을 갖는다. 또한 노이즈에 대해 상대적으로 강한 면을 볼 수 있다.

경계검출 방법은 우선 여러방향의 에지 마스크를 씌어 얻은 각 방향 에지 값들을 제곱의 합의 제곱근값이나 계산상 수월한 각 방향 에지 절대값의 합 또는 절대값의 최대값 등의 비선형 계산을 거친 후 임계값보다 큰지 작은지에

따라 그 점이 에지인지 아닌지를 나타내도록 2치(binary value)하여 검출되도록 한다. 대표적인 에지 연산자로는 Sobel 연산자, Laplacian 연산자, Roberts 연산자, Prewitt 연산자 등이 있으며 초기의 이미지를 Sobel, Laplacian, Roberts의 경계 추출알고리즘을 이용하여 구현해 보았다. 그림에서 보듯이 Sobel edge 알고리즘이 임계치 150에서 효율적으로 검출됨을 보여준다.

나. 칼라영상처리

수확기의 영상입력장치가 일반 산업용 로봇과 차이점은 대상체 개개의 형상, 크기, 색이 각기 다르고, 3차원적으로 불규칙하게 분포한다는 점, 자연 상태의 외부환경하에서 작업하기 때문에 조명조건(태양광의 직사, 그늘, 역광 등)이 불균일하고 가변적인 점 등을 들 수 있다.

대상물을 배경으로부터 분리해서 식별하고 고속으로 처리하기 위해 영상의 2 치화를 많이 사용한다. 흑백 카메라로 영상을 입력하는 경우, 2 치화 처리는 설정한 문턱값 밝기보다 더 밝은가 아닌가로 수행한다. 농업용 로봇을 이용한 작업의 경우, 실내에서 이미 거의 위치가 정해져 있는 묘에 대한 형상계측 등에서는 전술한 2치화 처리 방법도 가능하리라 생각된다. 그러나 야외 작업용의 수확용 로봇에서는 전술한 것처럼 조명 상태가 불규칙하고 가변적이어서 이 방법으로는 대상물의 식별이 곤란할 경우가 많다. 이와 같은 조건에서는 색정보를 이용한 식별이 유효하다. 특히 토마토, 귤 등은 수확시기에 배경을 이루는 줄기, 잎과는 명확한 색상의 차이가 있다.

컬러 영상신호는 적(R), 녹(G), 청(B)신호와 동기신호로 구성되어 있다. 이 컬러 영상신호로부터 대상물의 색 차이를 이용하여 작물의 2치영상을 얻는 방법으로서, 색신호끼리 직접 비교하거나 RGB색신호를 이용하여 연산한 값을 문턱값과 비교하는 등의 방법이 이용되고 있다.

RGB색 신호로부터 얻어 사용하는 변수량에는 휘도(Y), 색상, 채도, 색차(R-Y, B-Y) 그리고 RGB 3원색 중 R값이 차지하는 비율 등이 있다. 그리고 휘도 Y는

$$Y = 0.30R + 0.59G + 0.11B$$

의 식으로 구해진다. 색상은 적, 청, 황의 색조를 나타낸다. 채도는 포화도라고도 부르며, 색의 선명한 정도를 나타낸다. 예를 들면 같은 청색에서도 하야스름한 청색과 선명한 청색이 있는데 선명한 청색을 채도가 크다고 한다. 연산·비교의 방법에는, 컬러 TV카메라로부터 입력되는 아날로그 영상신호를 연산증폭기를 통한 후 비교기로 비교하는 방법과 컬러 영상 입력장치를 이용하여 RGB 각 신호를 각기 A-D변환하여 영상메모리에 입력해서 컴퓨터나 영상처리 프로세서로 행하는 방법이 있다.

이외에 컬러 영상신호로부터 디코더를 사용하여 색상과 채도를 취출하고, 색상과 채도를 취출하고, 색상과 채도를 비교기로써 문턱값과 비교하여 오렌지의 2치영상을 입력하는 방법도 있다. RGB 3원색을 각각 A-D변환하고, 영상메모리에 입력하여 디지털 영상처리를 하는 방법도 있다. 작물의 식별방법으로서는, 주로 다음의 방법들이 이용되고 있다.

(1) 3원색 중 2색의 차, 예를 들면 R-G를 문턱값과 비교

(2) 영상신호 중에 차지하고 있는 특정 색신호의 비율, 예를 들면 R/(R+G+B)를 문턱값과 비교

(3) 색상, 채도를 연산하여 구하고, 문턱값과 비교

(4) NTSC방식의 TV방송에서 사용하고 있는 클로미넌스(Q,I)와 색차신호(R-Y, B-Y)를 연산으로 구하고, 문턱값과 비교

이와 같은 A-D변환된 RGB신호를 이용하여 디지털적으로 식별하는 방법은 하드웨어로써 하는 방식이 많은데, 대상물과 배경의 색에 다수 존재하거나 겹쳐 보이는 것, 예를 들면 과수의 잎이나 솎아내기 전의 묘를 대상으로 하여 개개의 잎이나 묘를 식별하기 위해서는, 이것만으로는 불충분하여 고도의 영상처리 알고리즘이 필요하다. 이 경우, 2치화하고나서 메모리로 입력하기보다는 RGB 3원색 모두 A-D변환하는 편이 정보량이 많아 인식 가능성이 크다. 옥외작업의 경우 식물체는 다양하고 가변적인 자연환경하에서 복잡한 형상, 색, 그리고 크기 등을 가진다. 따라서 잎이나 가지 등과 같은 계통의 색을 띤 과실

을 식별할 경우에는 색신호를 이용하는 방법만으로는 충분하다고 할 수 있다 그러므로 일반적인 영상입력센서가 감도를 갖는 가시광영역에서부터 근적외영역까지의 분광반사특성을 조사하여 식별에 적합한 대상물 특유의 파장대역을 여러 개 취하여 식별하는 방법이 유효하다.

수확용 로봇의 경우 영상입력부 기능으로서 대상물 각 부위의 분광반사특성에 기초한 식별기능, 즉 잎의 형상, 생장상태, 영양상태, 종류, 줄기, 가지의 방향 그리고 병의 유무 등을 식별하는 것이 필요한 경우가 많다. 이들 정보의 대부분은 잎에 있고, 그 형상, 방향 등에 관한 정보를 여러 종류의 방법으로 간단히 수치화해서 기술할 수 있으면 인식능력은 커진다. 또한, 영상인식에 있어서는 3차원 공간의 대상체가 3차원으로 투영되어 나타나는 영상정보로부터 어떻게 거리에 관한 정보를 얻어 대상을 이해하는 데 결부시킬까 하는 것이 중요한 과제의 하나이다. 이때, 대상의 형상에 관한 특징을 미리 알고 있으면, 완전한 거리정보를 얻을 수 없는 경우도 그 형상의 특징과 규칙성을 이용한 인식이 가능하게 된다. 영상처리 및 인식의 주사순서에 있어서도 영상 전체를 거의 화소의 배열순으로 주사하는 것이 아니라, 대상물의 형상 특징으로부터 영상 중에서 대상이 존재하는 범위를 미리 추측해서 주사할 수 있으면, 불필요한 주사를 감소시키는 동시에 처리시간을 단축할 수 있으며 인식률도 높일 수 있다.

식물체의 잎이나 꽃 등의 배열은 규칙성을 가지는 경우가 많아서, 이것을 미리 지식 베이스로써 저장해 두면 신속하고 정확한 인식을 할 수 있다. 예를 들면 오이는 거의 2열 호생엽서(互生葉序)에 가깝고, 대상물고 시점의 상대적인 위치관계, 시점의 방향, 카메라의 시야각 등을 알고 있으면, 대상물의 영상을 대략 추측할 수 있다. 반대로, 이 잎차례의 규칙성을 이용하면 영상으로부터 대상물과 시점의 위치관계 및 시점의 위치관계 및 시점의 방향 등의 추측도 어느 정도는 가능하다. 이와 같이 투영모델은 시점의 이동 및 전장에 따라 변화한다. 이 모델과 영상을 대응시킴으로써, 대상물과 시점의 위치관계 추정 및 줄기와 잎을 주사하는 알고리즘을 작성할 수 있다.

작물은 이랑 등을 따라 다수의 열로 재배되는 경우가 많으므로 우선, 대상이 되는 작물과 그 이외의 것을 식별할 필요가 있다. 이때 작물 이외의 토양, 배경 등의 식별에는 주로 분광반사특성을 이용하고, 작물끼리의 식별에는 거리정보를 이용하는 것이 적당하다고 생각된다. 다음에 영상 데이터를 2치화 혹은 다치화 처리를 하여 모멘트, 페레장비 등을 이용하여 대상의 주줄기, 선단, 하단 등의 검출 및 크기의 계측을 행한다. 그리고 투영 모델, 페레장비, 복잡도 등을 이용하여 시점과 대상물의 위치관계 및 방향을 추정한 후, 추정된 방향의 투영모델을 기초로 각 줄기와 잎을 주사하여 위치, 방향 등을 검출한다.

## 제 4 절 연상메모리를 이용한 포도인식

### 1. 연상메모리 알고리즘

포도 수확기를 개발하기 위해서는 포도 형상과 위치를 정확하게 파악하는 것이 필요하다. 신경회로망(Neural network)의 연상메모리(Associative memory)를 이용하여 포도 형상 정보를 인식하고자 한다. 신경회로망을 이용한 연상메모리는 학습 패턴(Learning pattern)을 학습한 후에 입력 패턴(Input pattern)으로부터 출력패턴을 얻는다. 학습 패턴에 대한 입력 패턴의 연상능력과 검출조건을 분석한 후, 실제영상에서 포도 형상과 위치정보를 자동검출(Auto scanning)에 의한 출력패턴을 인식하고자 한다. 본 연구는 연상메모리를 통하여 포도를 정확하게 측정할 수 있는 인식기술을 구명하여 포도 수확기 개발에 활용하고자 한다.

영상은 칼라 CCD카메라로 부터 입력시키고, 영상신호는 아날로그 신호로서 프레임 그래버(Frame grabber)에 의해 컴퓨터에 변환되어 입력한다. 본 연구에서 사용된 컴퓨터 사양은 CPU Pentium Pro 200Mhz, RAM 64M, Video RAM 4M이며, 개발 환경은 GUI(Graphic User Interface) 기반의 C++로 개발하였다.

입력 센서부에 해당하는 영상입력 장치로는 16mm렌즈를 부착한 CCD 카메라를 사용하였다. 신호 처리부는 프레임 그레버에 의해 12.28Mhz의 아날로그 신호값으로 변환되고, CCD 카메라에서 출력되는 영상 신호는 컴퓨터와의 인터페이스(Interface)를 위하여 영상처리보드에 의해 디지털(Digital)신호 값으로 변환시킨다. 컴퓨터 프로그램에 의해 수행된 영상처리 결과는 영상메모리 내부의 LUT(Look Up Table)에 의해 모니터(Monitor)에서 영상을 출력시킨다. 영상처리보드는 OCULUS TCI-SE PCI 프레임 그레버(Coreco Inc., St. Laurent, Canada)를 사용하였다.

실험재료는 충북 영동군 포도밭에서 획득한 영상을 이용하였다. 이는 수확적기에 포도밭에서 찍은 영상이다. 아래 <그림 2-10>은 본 연구에 사용한 실제 영상을 나타낸다.

<그림 2-10> 실제 포도 영상

연상 메모리에 의한 형상인식은 일반적으로 학습시간이 오래 걸리는 단점이 있다. 그러므로 연상 메모리는 학습패턴 s와 출력패턴 t의 곱을 연결강도 w로 사용하고, 학습패턴과 출력패턴의 연관성을 기억시켜 둔다. 여기서, 학습패턴과 출력패턴은 다음과 같이, s는 1×n 벡터, t는 1×m 벡터로 배열한다.

$$\mathbf{s} = [\, s_1 \;\; s_2 \;\; s_3 \;\cdots\; s_n \,]$$

$$\mathbf{t} = [t_1 \;\; t_2 \;\; t_3 \cdots\cdots \;\; t_m]$$

학습패턴과 출력패턴간의 벡터는 식과 같이 $\mathbf{s}^T\mathbf{t}$를 계산하므로 n×m 매트릭스(Matrix)가 된다. 그러므로 연결강도 **w**는 식과 같이 나타난다.

$$\mathbf{s}^T\mathbf{t} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ s_n \end{bmatrix} [t_1 \; t_2 \; t_3 \;\; \cdots\cdots \; t_m]$$

$$= \begin{bmatrix} s_1t_1 & s_1t_2 & \cdots & s_1t_i & \cdots\cdots & s_1t_m \\ s_2t_1 & s_2t_2 & \cdots & s_2t_i & \cdots\cdots & s_2t_m \\ \cdot & \cdot & & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & & \cdot \\ s_nt_1 & s_nt_2 & \cdots & s_nt_i & \cdots\cdots & s_nt_m \end{bmatrix}$$

신경망의 학습에 있어서 새로운 학습패턴 쌍 [ **s**, **t**]에 의한 연결강도의 변화는 식과 같으며, P개의 학습패턴 쌍을 확장하여 연결강도 w를 구할 수 있다.

$$\mathbf{w} = \mathbf{w}_1 + \mathbf{w}_2 + \cdots + \mathbf{w}_p$$

$$= \mathbf{s}^T(1)\,\mathbf{t}(1) + \mathbf{s}^T(2)\,\mathbf{t}(2) + \cdots + \mathbf{s}^T(p)\,\mathbf{t}(p)$$

$$= \sum_{p=1}^{P} \mathbf{s}^T(p)\,\mathbf{t}(p)$$

이와 같이, 연상메모리에서는 연상하고자 하는 패턴 쌍이 주어지면, 이들의 연관성을 계산할 수 있다. 동질연상 메모리는 식과 같이 학습패턴 **s**와 연상될 출력패턴 **t**가 동일한 형태이다. 그러므로 연결강도는 **s** = **t** 인 연상메모리이다. 여기서 $p$는 기억시킬 연상패턴 쌍의 개수이다.

$$\mathbf{w} = \sum_{p=1}^{P} \mathbf{s}^T(p)\ \mathbf{t}(p)$$

$$= \sum_{p=1}^{P} \mathbf{s}^T(p)\ \mathbf{s}(p)$$

또한, 부분 입력이나 오류가 섞인 학습패턴은 원래의 학습패턴으로 복원하기 위해, 연결강도 **w**의 대각 요소를 제거할 필요가 있다. 대각요소 제거는 식과 같이 학습패턴 수만큼의 단위 메트릭스(Unit matrix) $p\mathbf{I}$ 를 이용한다. 이는 연상 효과를 향상시킬 수 있다.

$$\mathbf{w} = \sum_{p=1}^{P} \mathbf{s}^T(p)\ \mathbf{s}(p) - p\mathbf{I}$$

즉, 동질연상 메모리에서는 학습패턴과 출력패턴의 직교치(Orthogonal value)가 0일 때, 학습패턴으로 출력되는 모델이다. 그러나, 저장된 학습패턴들이 직교하는 경우는 대부분 없으므로, 인접 패턴에 의한 혼선(Cross-talk)이 존재한다. 이러한 문제를 해결하기 위해, 양극성 계단 함수(1과 -1)를 활성함수(Activation function)로 사용하여 어느 정도 학습패턴과 유사한 출력패턴을 연상해낼 수 있다.

그러므로, 출력패턴 **y**는 다음 식과 같이 연상될 샘플패턴 **x**와 연결강도 **w**를 곱하여 패턴의 결과를 나타낼 수 있다.

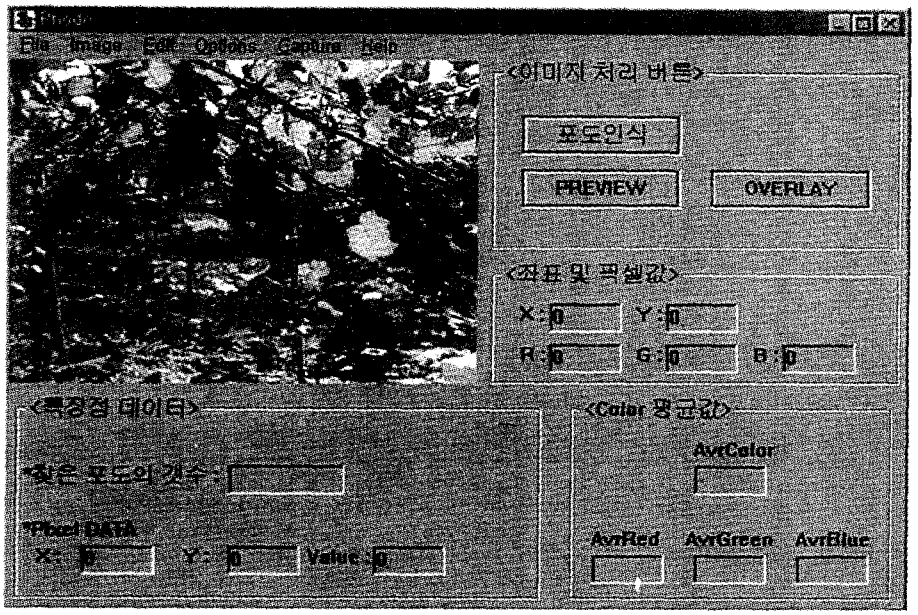
$$\mathbf{y} = \mathbf{xw}$$

$$= \mathbf{x}\ [\sum_{p=1}^{P} \mathbf{s}^T(p)\ \mathbf{s}(p) - p\mathbf{I}\ ]$$

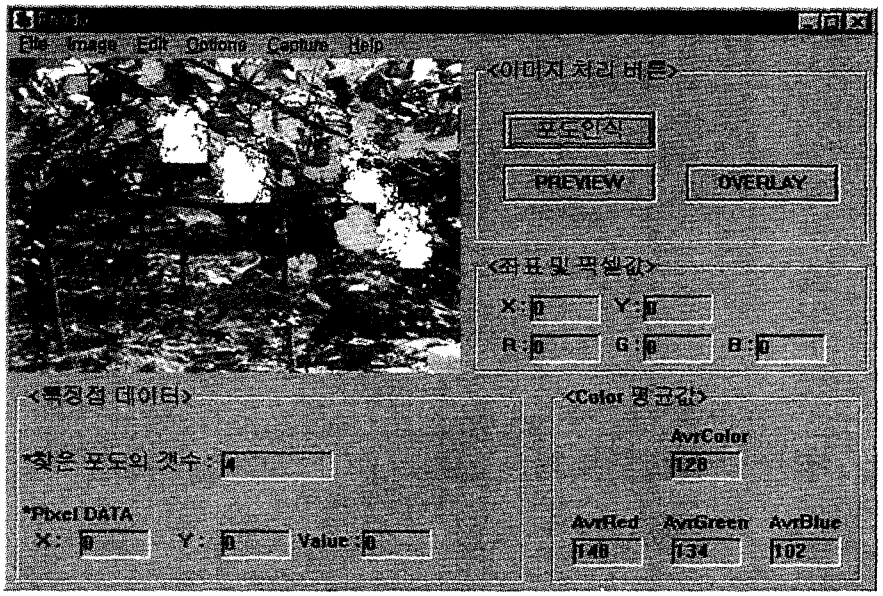
2. 결과 및 고찰

연상메모리 알고리즘을 이용하여 포도인식을 수행한 결과는 다음과 같다. 수학적기의 포도밭에서 얻은 실제 영상을 신경회로망의 연상메모리에 적용하여 포도의 형상 및 위치를 인식하고자 하였다. <그림 2-11>은 실제 영상에 적용한 것을 보여주며, 이때 픽셀의 크기는 320 x 240으로 하였다. <그림 2-12>에

서 학습 패턴을 2개로 정한 후 5개의 입력 패턴을 적용하여 5개의 출력 패턴을 얻었다. 이 중에서 4개는 학습패턴과 일치하였고, 1개는 일치하지 않았다. 일치하는 패턴은 실제로 포도송이가 있는 위치에서 검출되었으며, 반면에 포도의 잎과 덩굴이 있는 곳에서는 일치하지 않았다.




<그림 2-11> 연상메모리를 이용한 영상처리 결과(학습 패턴을 위한 영상)




<그림 2-12> 연상메모리를 이용한 영상처리 결과 (출력 패턴을 출력한 영상)

이 때 한 개의 포도 크기가 차지하는 픽셀의 크기는 40 x 50으로 하였다. 이 때문에 연결강도 **w**를 구하기 위해서는 데이터의 크기가 2000개나 되었다. 그래서 하나의 입력패턴을 출력 패턴으로 바꾸기 위해서는 2분 내외 소요되었다. 포도송이의 형상과 위치를 인식하기 위한 연상메모리 적용은 이미지 프로세싱 시간을 줄이기 위한 알고리즘 개발이 필요하다. 또한 학습 패턴의 가로 세로의 크기가 다른 경우에도 연상메모리 적용이 가능한 알고리즘이 필요하다.

## 3. 결 론

포도 수확기를 개발하기 위하여 포도의 과경을 인식하는 것이 주요하다고 할 수 있다. 포도의 잎, 덩굴 등의 배경으로부터 다양한 포도송이의 위치 및 형상을 정확하게 인식하기 위한 기초 연구로 시도되었다. 영상처리 시스템을 사용한다면 포도의 색깔을 이용하여 포도를 쉽게 발견 할 수는 있지만 포도

열매의 과경을 추적하는 것은 쉽지 않다. 이 문제를 해결하기 위하여 신경회로망의 연상메모리를 이용하였다.

이 연구에서는 포도밭에서 얻은 영상을 이용하여 포도송이의 형태와 위치를 인식하고자 하였다. 연상메모리를 이용하여 포도송이는 발견할 수 있었지만 포도의 과경은 찾을 수는 없었다. 앞으로 좀더 많은 연구를 통하여 과경의 형상과 위치를 정확하게 인식하는 것이 필요할 것이다.

# 제 5 절 스테레오 비젼(Stereo Vision)을 이용한 포도 검출

## 1. 서 론

포도 수확용 로봇을 개발하기 위해서는 선행적으로 포도의 형상과 위치를 정확하게 파악하는 것이 필요하다. 일반적으로 포도의 수확은 품종에 따라 차이가 있지만, 대부분 수확시 과일의 색이 변한다. 따라서 포도 수확용 로봇 개발에 있어서 포도 인식을 위한 영상처리 시스템은 포도인식을 위한 칼라 영상처리를 이용하였다. 또한 포도인식을 위한 알고리즘에 포도의 기하학적인 형상도 고려하였다. 2차원내의 포도인식은 입력된 영상의 색상차이에 따른 분석이 가능하다. 그러나 3차원공간상의 포도의 거리 정보를 찾기 위해서는 두 대의 카메라, 또는 다른 2가지 형태의 입력 영상을 분석하지 않으면 안 된다.

따라서 카메라로부터 얻어진 영상을 해석하여 포도형상 및 위치를 인식하는 영상처리 알고리즘을 개발하는데 그 목적이 있다.

본 연구목적은 3차원 공간상의 포도를 카메라로부터 얻어진 2차원 영상으로부터 포도를 정확히 인식하고 인식된 포도를 정적인 물체의 3차원 시각을 고려하여 신속하고 정확하게 거리정보를 획득할 수 있는 3차원 시각장치를 구현하는데 목적이 있다. 본 연구를 수행함에 있어 세부적인 목표는 다음과 같다.

첫째, 2차원공간으로 표현된 포도 영상으로부터 포도의 형상을 고려하여 포도 인식 알고리즘을 구현하는데 있으며,

둘째, 인식된 포도의 거리 정보를 획득하기 위해 인간의 시각과 유사한 형태의 스테레오 시각시스템을 설계하고, 스테레오 영상을 이용하여 대상체의 거리정보를 삼각측량법을 이용하여 거리 정보를 추출하기 위한 알고리즘을 제시하고자 한다.

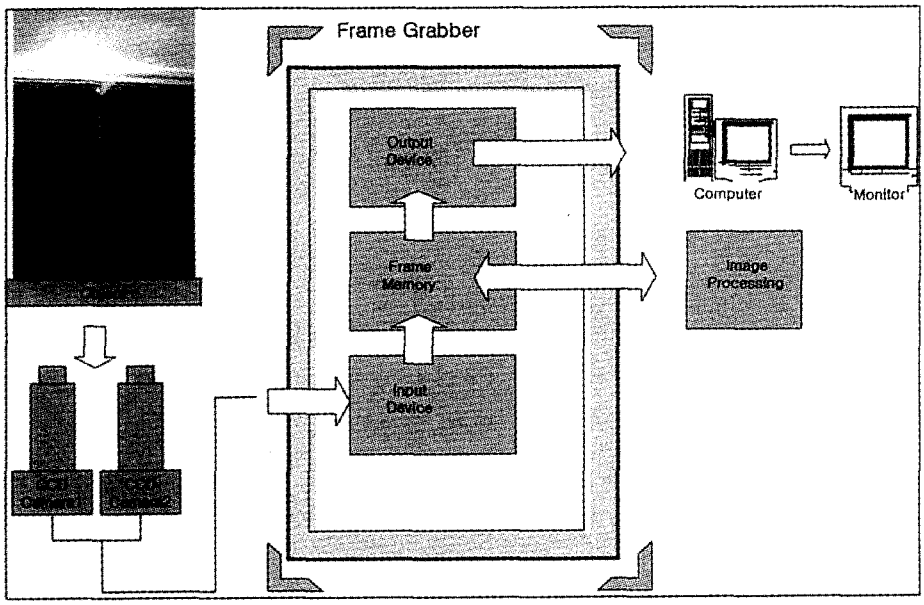
셋째, 개발된 실험장치 및 알고리즘을 포도 농가에 적용하여 현장실험을 통한 포도 인식 알고리즘을 검증하고자 한다.

2. 실험장치 및 재료

가. 실험장치

1) 실험장치개요

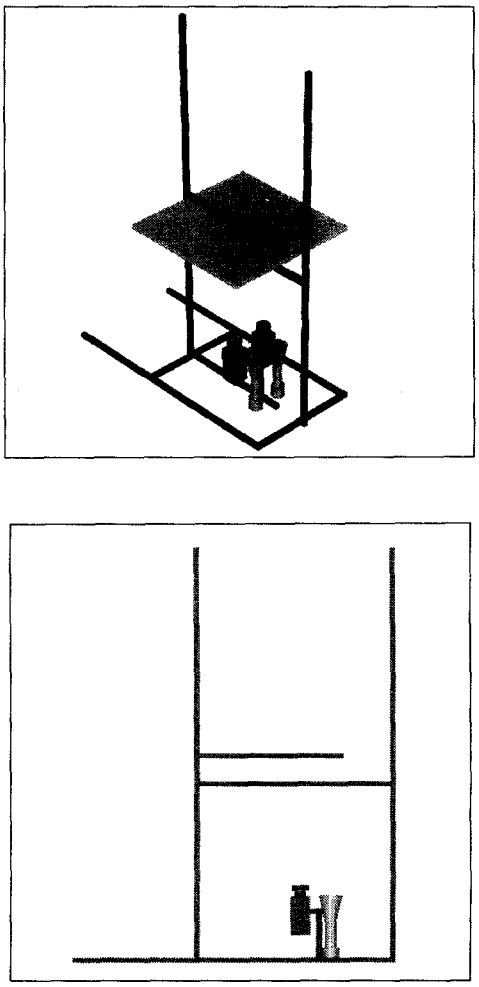
일반적으로 사용된 영상처리 시스템의 기능 블록은 <그림 2-13>과 같이 영상신호입력부, 영상처리부, 주 컴퓨터 및 영상출력부로 구성된다. 컬러 CCD 카메라로부터 입력된 영상신호는 RS-170 형태의 아날로그 신호로 컴퓨터에 변환된다.



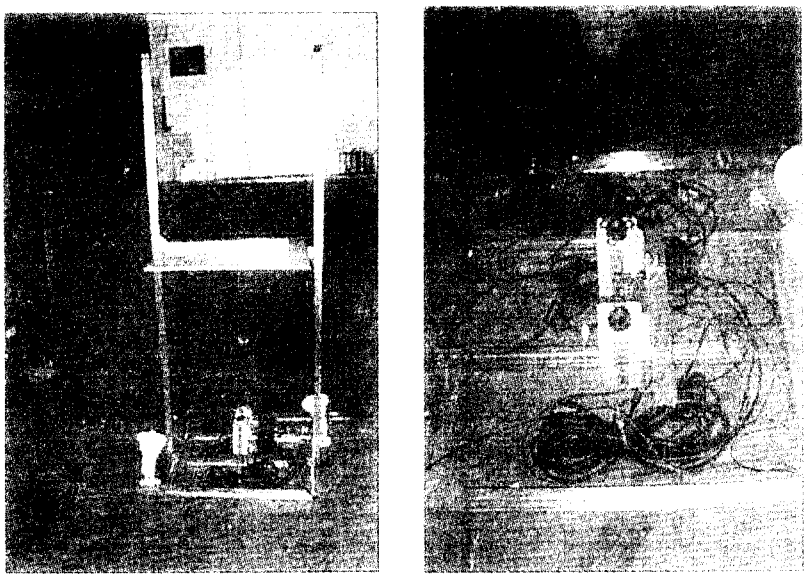

<그림 2-13> 영상처리 시스템

2) 실내실험장치

실내 실험을 하기 위해 영상처리 시스템을 설계·제작하였다. 실험 장치의 골격 재질은 20mm 두께의 프로파일을 사용하였고, 배경을 차단하기 위해 흰색 두께가 5mm이고 가로와 세로가 600mm인 우드락을 사용하였다.

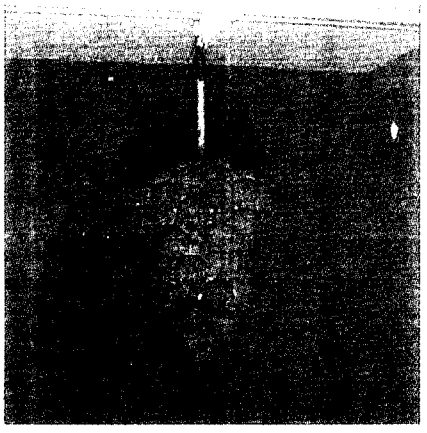


<그림 2-14> 실내실험을 위한 영상처리시스템의 입체도



<그림 2-15> 실내실험을 위해 실제 제작된 영상처리시스템

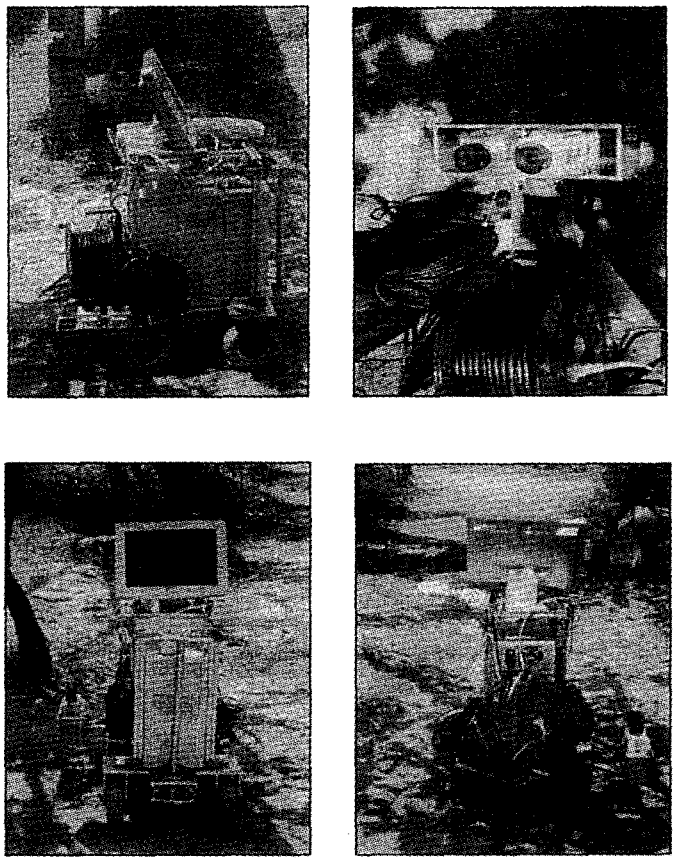
실내 실험을 위해 실축의 포도 모형을 제작하였다.



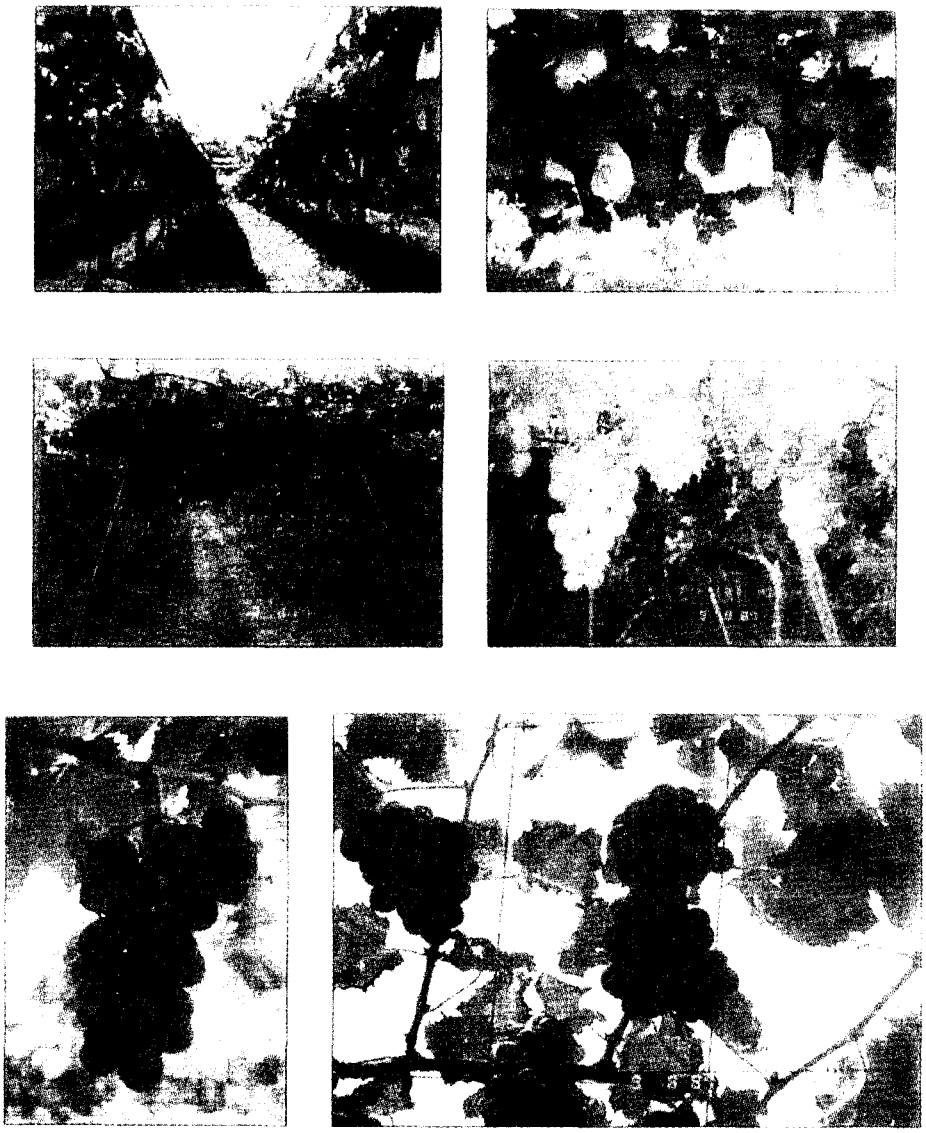
<그림 2-16> 실내실험을 위한 포도모형

3) 현장실험장치

실내 실험을 통해 얻은 영상처리 알고리즘을 현장실험에 적용하여 검증하기 위해 현장실험을 위한 영상처리 시스템을 설계·제작하였다. 실험 장치의 골격 재질은 30mm 두께의 프로파일을 사용하였고, 드랄미늄을 사용하여 고정하였다. 현장실험장치는 영상처리를 위해 두 대의 카메라를 고정할 수 있는 고정틀과 컴퓨터를 놓을 수 있는 컴퓨터 고정부, 그리고 현장의 포도밭을 이동할 수 있도록 4개의 바퀴와 조향장치로 구성되어 있다.



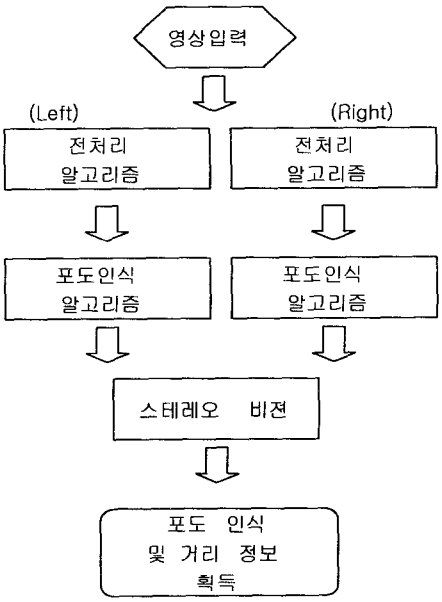
<그림 2-17> 현장실험을 위해 실제 제작된 영상처리시스템



<그림 2-18> 현장실험을 위한 포도 농가 현장

3. 실험방법

가. 포도 인식 알고리즘 개요




<그림 2-19> 영상처리 흐름도

나. 전처리 알고리즘

영상처리 알고리즘에서 전처리는 왜곡된 영상에서 보다 나은 영상데이터를 얻고자 하는데 있다. 전처리는 영상의 질을 개선하거나 영상을 특정한 응용 목적에 알맞도록 변환시키는 등의 영상처리를 의미한다. 2대의 카메라에 의한 스테레오방법의 대상체 인식정보를 추출하기 위해 선행되어야 할 전처리 알고리즘으로서는 대상체와 배경을 분리함으로써 대상체에 대한 많은 정보를 얻기 위한 이치화 과정을 하고 경계검출 알고리즘을 사용할 것이다. 또한 두 카메라에 의한 스테레오 방법으로 대상체의 영상을 그랩(Grab)하여 영상을 처리하기 위해, 배경으로부터 대상체 정보를 추출하는 알고리즘을 구현할 것이다 . 이는 두 대의 카메라로 추출하고자 하는 대상체를 그랩(Grab)하면 대상체 외의 배경의 영상은 화소의 위치변화를 대상체보다 크게 나타난다. 두 대의 카

메라에 의해 영상의 대상체에서 큰 화소변화의 배경은 없애고 추출하고자 하는 특징점만을 형상정보로 얻기 위한 알고리즘을 구현하고자 한다.

1) 영상분할을 위한 이치화 방법

이치화를 보통 Binaruzation 또는 Thresholding 이라고 하며 크게 적용영역에 따라 두가지로 구분하면 Global thresholding 와 Local thresholding으로 나눌 수 있다. 전자는 처리 범위가 전체 측정윈도우에 해당하며 후자는 부분적인 측정윈도우를 지정함으로써 수행된다. 특히 특정영상을 분석하고 측정하고자 할 때는 대상물체에 따라 효율적으로 물체를 분리할 수 있는 국부이치화 방법에 대한 연구가 중요하다.

입력영상을 f(x,y), 출력영상을 g(x,y) 라고 정의하면 화소의 좌표 x, y에 대한 이치화 기준은 경계값 T보다 클 경우를 1로 나타내고 반대의 경우는 0으로 정하게 된다. 여기서 0은 물체(또는 배경)을 1은 배경(또는 물체)을 나타낸다.

$$g(x,y) = \begin{cases} 1 & : f(x,y) \geq T \\ 0 & : f(x,y) < T \end{cases}$$

여기서 T는 Brightness level로 배경부분과 추출할 패턴을 분리하는 이 변수를 Threshold value 라고 하는데 이 변수를 어떻게 화상의 히스토그램 분석에 활용하여 결정하는 것이 문제로 되어 있다. 이 변수는 Manual 조작에 의해 임의로 설정하여 줄 수도 있고 또는 2개의 Windows설정에 의해 경계치를 결정하는 Windows Thresholding Method 알고리즘과 가상의 모집단 분류에 의해 2모집단의 분산값을 최대로 하는 통계적 판별분석법을 이용하여 Threshold Value를 정할 수 있다. 본 연구에서는 정적 영상을 처리하도록 bitmap함수로 구현하였고 동적 영상에서는 그레버보드(Grabber Board)의 FrameBuffer 함수를 직접 이용하여 영상 처리가 가능하게 이치화를 하도록 구현하였다.

2) 에지 중심(edge based)분할 방법

에지(Edge)는 서로 다른 명암도를 갖는 영역간의 경계에 위치하는 점이다.

그러므로 영상함수 f(x)에 대해, 에지(edge)에서의 기울기 df/dx 는 명암이 균일한 지역에서의 기울기와 현저히 비교된다. 그러므로, 도함수(Derivative)를 계산하고 임계치(Threshold)를 사용하여 에지점(edge point)들을 검출할 수 있다. 그러나 잡음(noise)이 존재하는 경우에, 이 에지검출자는 많은 오인된 에지들을 검출하게 되는데 좀 더 나은 결과를 위해 화소단위로 에지크기를 측정하는 대신에 영역 단위로 에지크기를 측정하는 방법을 사용한다.

본 전처리 과정은 형상정보를 얻기 위해 이치화와 경계검출과정을 실행함으로써 좀더 물체정보를 정확히 검출하는데 목적이 있다. 특히 서로 다른 명암도를 갖는 영역간의 경계를 이치화 및 경계검출 알고리즘을 사용하면 더 나은 형상정보추출을 얻을 수 있다.

에지는 두 영역(region)의 경계에 위치하는 점들을 말하며, 영역간의 경계부분은 한 영상안에서 명암도의 불연속으로 나타난다. 이는 두 영역의 경계(boundary)에 의해 대상을 표현하기 위한 방법이다. 주로 영상에 있는 명암값(gray-level value)들로부터 직접 대상들의 경계를 찾아내기 위해 중간단계의 에지(edge)영상으로 변환(transform)하고 나서 더 정확한 경계 영상을 구성하는 방법으로, 명암을 가지고 에지들을 찾아내는 것이 일반적인 방법이다.

에지 연산자들의 공통적인 특징은 연산자들이 최대 명암 변화의 방향(direction)과 이 변화의 정도를 나타내는 크기(magnitude)를 계산하는 것이다. 미분 연산자의 에지는 서로 다른 명암도를 갖는 영역간의 경계에 위치하는 점이다. 도함수를 계산하고 임계치(Threshold)를 사용하여 에지점들을 검출하게 되는데 좀 더 나은 결과를 위해 화소단위로 에지크기를 측정하는 대신에 영역 단위로 에지크기를 측정하는 방법을 사용한다. 또한 라플라시안은 선의 끝과 모퉁이에 특히 민감한 반응을 보이는데 경사진 곳의 견각(shoulder)에서 민감한 반응을 보이고, 그 곳에서 영점교차(zero-crossing)를 야기하며, 영점교차선의 경사도로서 에지크기를 나타낸다. Sobel 연산자는 해당 에지주변의 점을 에지로 검출하는 성질을 갖는다. 이러한 성질은 에지의 확대 및 축소를 야기시키며 Template Matching 알고리즘도 다소 유사한 특성을 갖는다. 또한 노이즈에 대해 상대적으로 강한 면을 볼 수 있다.

경계검출 방법은 우선 여러 방향의 에지 마스크를 씌어 얻은 각 방향 에지값들을 제곱의 합의 제곱근값이나 계산상 수월한 각 방향 에지 절대값의 합

또는 절대값의 최대값 등의 비선형 계산을 거친 후 임계값보다 큰지 작은지에 따라 그 점이 에지인지 아닌지를 나타내도록 2치(binary value)하여 검출되도록 한다. 대표적인 에지 연산자로는 Sobel 연산자, Laplacian 연산자, Roberts 연산자, Prewitt 연산자 등이 있다.

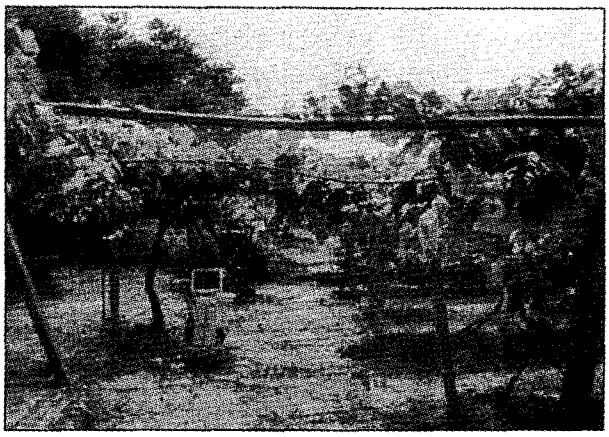
3) 세선화 방법

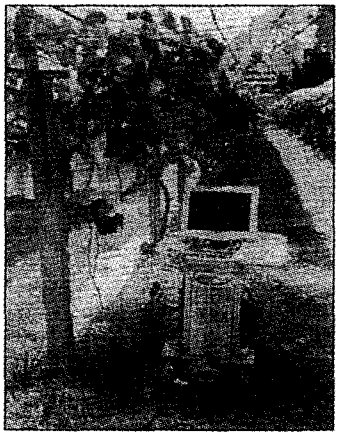
세선화 방법은 이진화된 영상으로부터 획득하고자하는 대상체의 특징형상을 얻는데 좀더 나은 효과를 얻기 위해 사용된다. 기본 개념은 두꺼운 선을 최외각으로부터 한겹씩 벗겨나가서 마지막에 남는 선 성분을 추출하는 것이다. 윤곽선 추출을 비교하여 보면 윤곽선 추출은 외각의 선 성분을 추출하는 반면 세선화는 가운데 선을 추출한다. 즉 영상의 본질적인 구조를 보존한 채, 선도형을 추출하는 조작이다.

세선화 방법은 원래 도형의 연결선은 변화시키지 않고, 선으로 변화시키는 것이 필요하다. 세선화 방법의 결과에서 도형의 결합 관계의 특징이 구해지므로, 글씨나 도면 등에서 선의 구조를 해석하는데 불가결한 처리로 되어 있다. 여러 세선화 알고리즘 중에서 3×3의 이웃 윈도우 안에서 조사하여 각 영역이 세선화 할 때까지 각 영역의 경계선을 한번에 한 픽셀 두께씩 벗겨내는 것이다. 이런 처리가 반복적으로 되풀이된다.

이 세선화 방법을 이용함으로써 이진화된 영상으로부터 대상체인 포도를 인식하는데 있어 포도 형상의 특징을 추출하여 포도 인식을 좀더 효과적으로 할 수 있을 것으로 사료된다.

4) 현장실험 과정





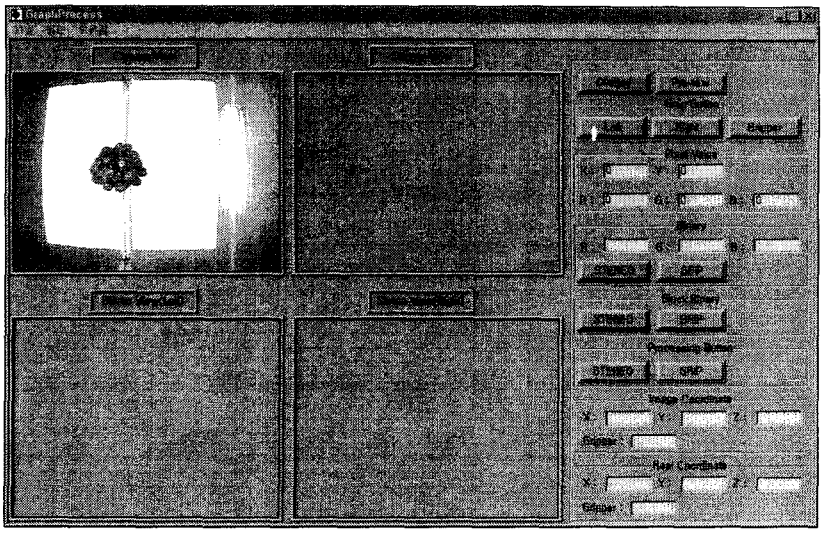
<그림 2-20> 현장실험에 적용된 영상 처리 시스템

4. 포도 인식 Main Program

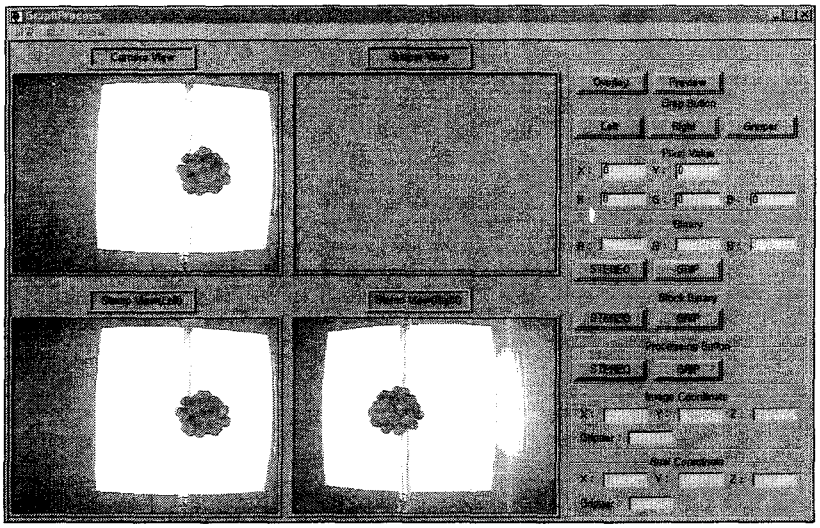
Main Program의 개발 환경은 GUI(Graphic User Interface) 기반의 Microsoft사의 Visual C++로 개발하였다.

프로그램 구성은 사용자가 모니터 할 수 있는 4개의 View와 영상처리 버튼 그리고 인식된 포도의 좌표 데이터 출력창으로 구성되어 있다.

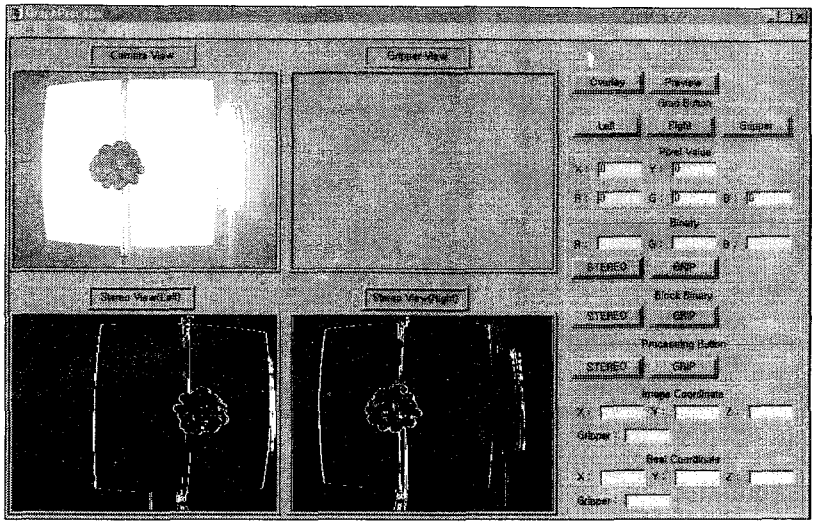
<그림 2-21>은 개발된 프로그램의 초기화면이며, <그림 2-22>~<그림 2-24>는 영상처리 알고리즘을 실행하는 과정을 나타낸 그림이다.



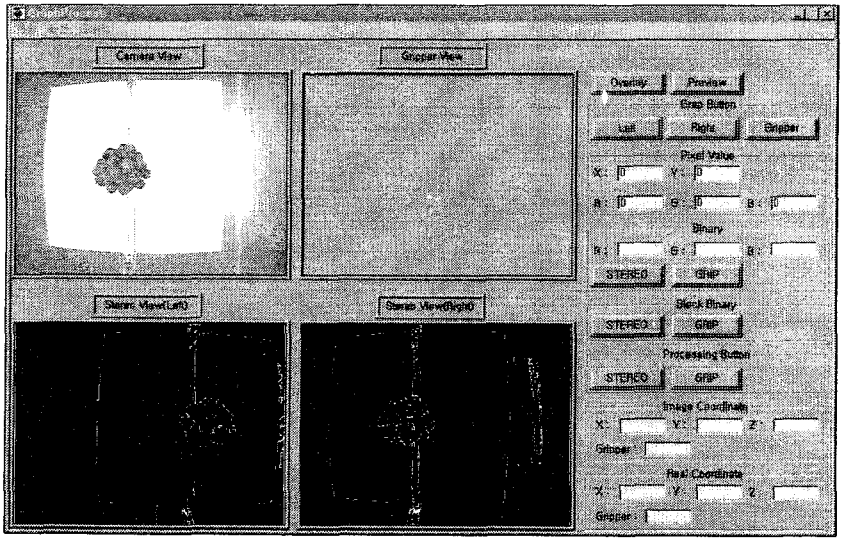
<그림 2-21> 프로그램 초기화면



<그림 2-22> 프로그램 그랩된 화면




<그림 2-23> 프로그램 이진화된 화면




<그림 2-24> 프로그램 세선화된 화면

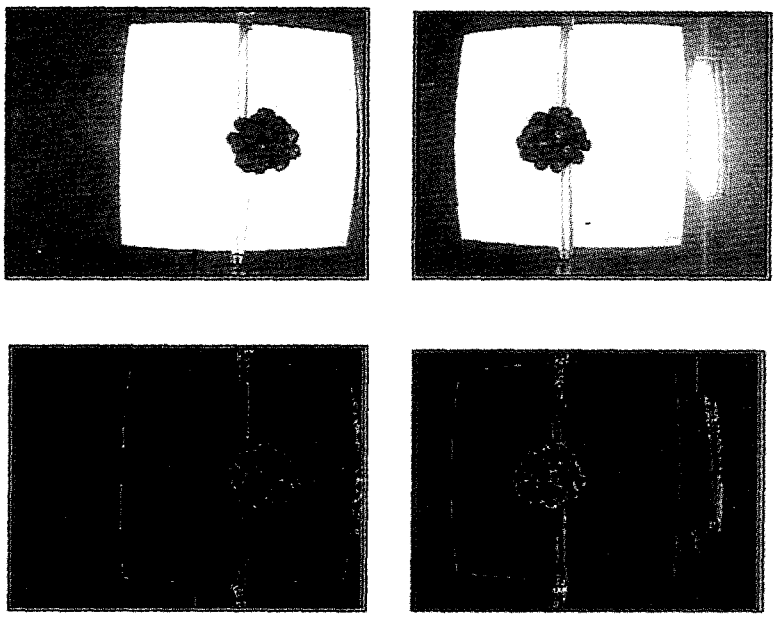
5. 결과 및 고찰

가. 모형장치를 이용한 영상처리 결과

3차원 공간상의 포도를 카메라로부터 얻어진 2차원 영상으로부터 포도를 정확히 인식하고 인식된 포도를 정적인 물체의 3차원 지각을 고려하여 신속하고 정확하게 거리정보를 획득할 수 있는 3차원 시각장치를 구현하였으며, 영상처리 알고리즘의 구성은 전처리 알고리즘, 포도 인식 알고리즘, 스테레오 비젼(Stereo Vision)으로 크게 세 부분으로 구성된다.

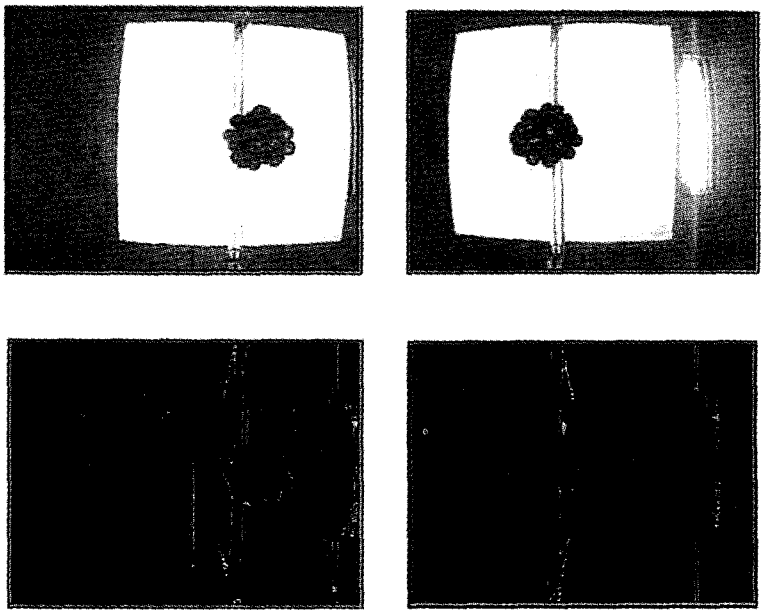
본 연구에서 개발한 프로그램에 영상처리 알고리즘을 구현하였다.

전처리 알고리즘을 적용하는 과정에서 전처리 알고리즘의 환경에 따른 영향에 대해 실험을 하였다. <그림 2-25>와 <그림 2-26>은 배경을 어느 정도 차단한 전처리 결과와 배경을 차단하지 않은 전처리 결과를 보여주고 있다.



(A) 좌 (B) 우

<그림 2-25> 배경을 차단한 영상처리 결과

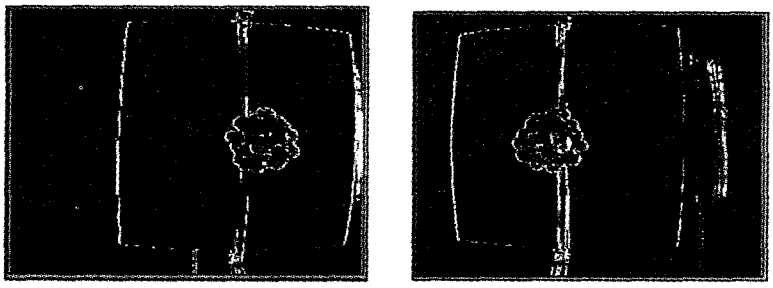


(A) 좌 (B) 우

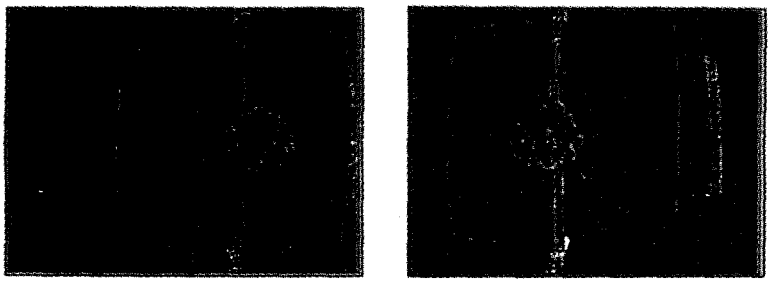
<그림 2-26> 배경을 차단하지 않은 영상처리 결과

<그림 2-25>와 <그림 2-26>에서 볼 수 있듯이 배경의 제약을 받지 않는 것으로 나타났으며, 현장 실험에서도 좋은 결과를 가져올 것으로 사료된다.

<그림 2-27>과 <그림 2-28>은 이진화 방법만 적용한 영상처리 결과와 이진화 후 세선화 방법을 적용한 영상처리 결과를 보여주고 있다.



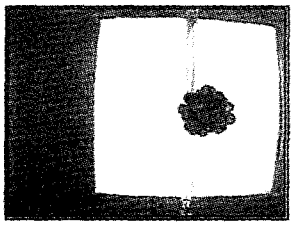
<그림 2-27> 이진화 방법만 적용한 영상처리 결과

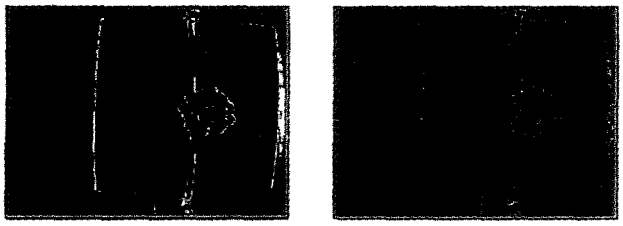


<그림 2-28> 이진화 후 세선화 방법을 적용한 영상처리 결과

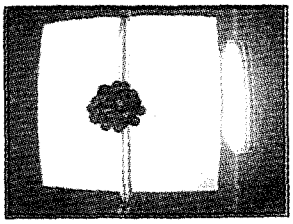
<그림 2-27>과 <그림 2-28>에서 볼 수 있듯이 이진화 방법만 적용한 영상처리 결과는 굵은 선으로 표시되어 포도를 인식하는 과정에서 오류를 범 할 수 있는 가능성이 있는 반면에, 이진화 후 세선화 방법을 적용한 영상처리 결과는 세선화를 통해 포도의 형상은 변화시키지 않고 선의 굵기를 가능한 얇게 하여 포도 형상을 고려한 포도인식알고리즘을 적용하는데 오류를 줄일 수 있을 것으로 나타났다.

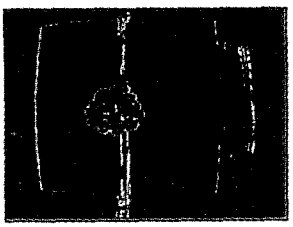
<그림 2-29>와 <그림 2-30>은 위의 과정을 통해 전처리 알고리즘을 일괄적으로 적용시켜 구현한 그림이다.

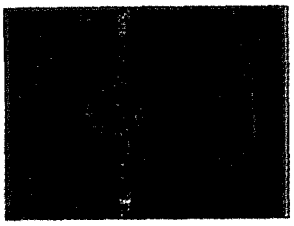




<그림 2-29> 좌측 영상의 영상처리 과정







<그림 2-30> 우측 영상의 영상처리 과정

위와 같은 방법으로 실내실험장치를 이용한 포도까지의 거리를 측정한 결과를 <표 2-4>와 같이 나타났다.

<표 2-4> 실내실험장치를 이용한 포도의 거리검출 결과

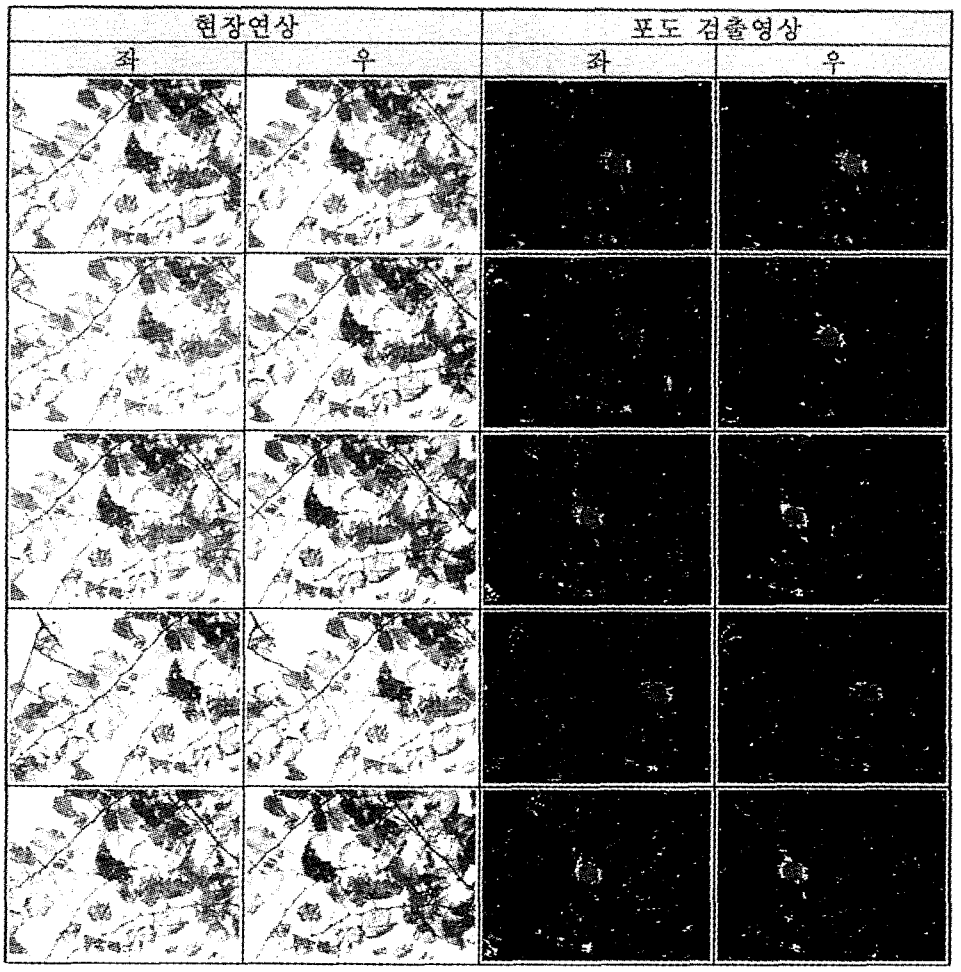
| No. | 실제거리(mm) | 계측거리(mm) | 오차(mm) |
|---|---|---|---|
| 1 | 500 | 495.681 | 4.319 |
| 2 | 600 | 604.141 | 4.141 |
| 3 | 700 | 702.068 | 3.309 |
| 4 | 800 | 802.539 | 5.319 |
| 5 | 900 | 898.550 | 3.474 |
| 6 | 1000 | 996.047 | 3.953 |
| 7 | 1100 | 1100.540 | 0.540 |
| 8 | 1200 | 1208.895 | 8.895 |
| 9 | 1300 | 1296.890 | 12.278 |
| 10 | 1400 | 1396.117 | 25.943 |
| **평균** | - | - | **7.217** |
| **표준편차** | - | - | **7.342** |

위에서 살펴보면 실제거리 1100mm까지는 오차가 5mm이하로 비교적 정확한 계측이 가능한 것을 알 수 있다. 그러나 실제거리 1200mm이상에서는

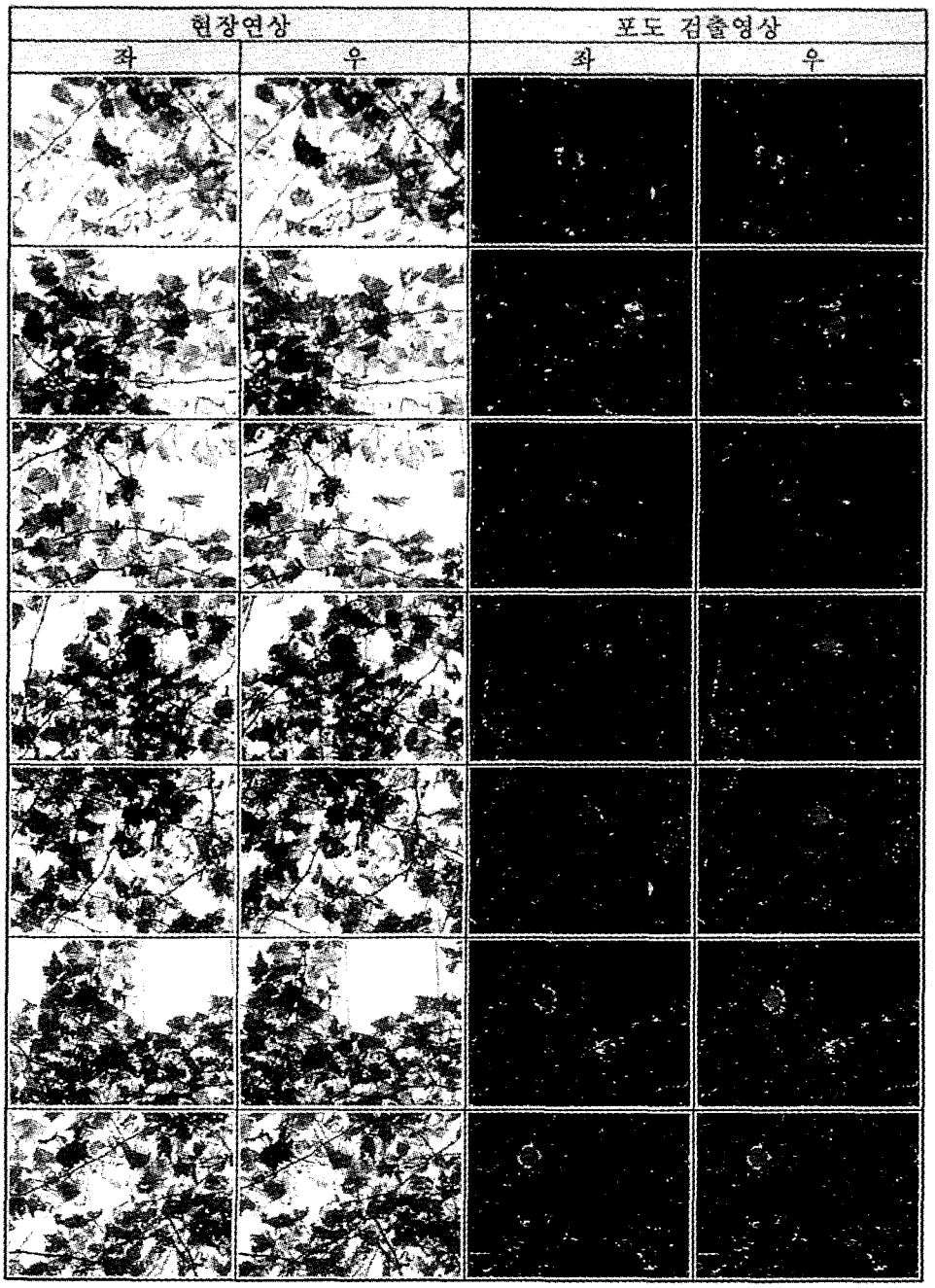
10mm 이상으로 비교적 오차가 크게 나타난 것을 알 수 있다. 이는 거리가 멀어지면서 영상의 화소값을 통한 거리 계측에서 단위화소에 해당하는 실제거리의 정확도가 감소하기 때문으로 판단되었다. 또한 오차의 전체 평균은 약 7.2mm이 었으며, 오차의 표준편차는 7.342mm로 나타났다. 따라서 이러한 결과는 생포도의 수확을 위해서는 그리퍼나 엔드이펙터 등에 의한 손상을 발생할 가능성이 높다. 따라서 계측 거리의 정확도를 높이기 위해서는 높은 해상도의 카메라 사용이 요구되며, 또한 그리퍼의 접근 방향 등을 고려한 시스템의 개발이 필요할 것으로 판단되었다.

나. 스트레오 비젼에 의한 포도 검출

실험 방법에서 의해서 현장에서 70가지의 상황에서 좌우카메라를 통하여 영상을 획득하였다. 획득한 영상을 전처리를 통하여 포도를 인식할 수 있도록 하였으며, 좌우 영상에서의 포도 인식 좌표를 이용하여 스트레오 메칭을 통한 포도까지의 거리를 계측하였다. 거리계측을 위해 획득한 현장의 포도 영상처리 결과를 <그림 2-31>과 <그림 2-32>에서 나타내었다.




<그림 2-31> 현장에서의 포도검출과정 예 (I)




<그림 2-32> 현장에서의 포도검출과정 예 (II)

위의 영상을 이용한 현장에서의 포도 위치 검출 후 두 대의 카메라 중심에서 포도까지의 거리를 계측한 결과는 <표 2-5>와 같다.

<표 2-5> 현장에서 임의의 포도영상 거리검출 결과

| No. | 실제거리(mm) | 계측거리(mm) | 오차(mm) |
|---|---|---|---|
| 1 | 1011 | 973.607 | 37.393 |
| 2 | 1032 | 1035.151 | 3.151 |
| 3 | 1077 | 1080.387 | 3.387 |
| 4 | 1100 | 1104.424 | 4.424 |
| 5 | 1125 | 1129.486 | 4.486 |
| 6 | 1155 | 1155.639 | 0.639 |
| 7 | 1180 | 1182.955 | 2.955 |
| 8 | 1210 | 1211.513 | 1.513 |
| 9 | 1239 | 1241.397 | 2.397 |
| 10 | 1270 | 1272.700 | 2.700 |
| 11 | 1306 | 1305.524 | 0.476 |
| 12 | 1340 | 1339.980 | 0.020 |
| **평균** | - | - | **5.295** |
| **표준편차** | - | - | **10.214** |

전체적으로 실제거리와 계측거리간의 오차가 5mm 이내로 비교적 정확한 계측이 이루어진 것을 알 수 있다. 단지 1번 포도 영상의 경우 37mm정도의 큰 오차를 보이고 있는 것을 알 수 있다. 이는 실제거리 계측과정에서 발생한 오차로 생각되며 전체적인 결과에 큰 영향을 미치지 않을 것으로 판단되었다. 오차의 평균은 약 5mm로 나타났으며, 이는 1번 영상을 제외하면 더욱더 감소할 것으로 생각된다. 따라서 스트레오 영상을 이용한 포도까지의 거리계측을 위한 영상처리 시스템은 현장에서도 적용가능 할 것으로 판단되었다. 다만 앞에서도 언급하였지만, 포도의 수확을 위한 그리퍼의 접근 방향을 고려한 적절한 알고리즘의 선택이 필요할 것으로 판단되었다. 본 연구에서 개발한 영상처리 알고리즘은 포도하부에서 거리를 계측하는 방법으로 측면 등에서도 고려할 수 있지만, 그리퍼의 접근이나, 영상의 획득 등에서 아래 방향이 유리할 것으로 판단하였다.

## 제 6 절 요약 및 결론

본 연구에서는 포도 수확기 개발을 위해 가장 중요한 기술인 영상처리시스템을 개발하고자 두 대의 카메라를 이용한 스트레오 영상 시스템을 구성하였다. 따라서 3차원 공간상의 포도를 카메라로부터 얻어진 2차원 영상으로부터 포도를 정확히 인식하고 인식된 포도를 정적인 물체의 3차원 시각을 고려하여 신속하고 정확하게 거리정보를 획득할 수 있는 3차원 시각장치를 구현하고자 수행한 결과 다음과 같은 결론을 얻었다.

가. 현장에서의 포도의 형상과 위치를 인식하기 위한 연상메모리 적용은 이미지 프로세싱 시간을 줄이기 위한 알고리즘 개발이 필요하였다. 또한 학습 패턴의 가로 세로의 크기가 다른 경우에도 연상메모리 적용이 가능한 알고리즘이 필요하였다.

나. 모형 포도를 이용한 실내에서의 스트레오 영상 거리 계측 결과 실제거리 1100mm까지는 오차가 5mm이하로 비교적 정확한 계측이 가능한 것을 알 수 있다. 그러나 실제거리 1200mm이상에서는 10mm 이상으로 비교적 오차가 크게 나타난 것을 알 수 있다. 이는 거리가 멀어지면서 영상의 화소값을 통한 거리 계측에서 단위화소에 해당하는 실제거리의 정확도가 감소하기 때문으로 판단되었다.

다. 현장에서 획득한 포도 영상을 이용한 스트레오 영상 거리계측 결과 실제거리와 계측거리간의 오차가 5mm 이내로 비교적 정확한 계측이 이루어진 것을 알 수 있다. 따라서 스트레오 영상을 이용한 포도까지의 거리계측을 위한 영상처리 시스템은 현장에서도 적용가능 할 것으로 판단되었다.

참고 문헌

[1] Gopalasamy Athithan., Chandan Dasgupta. 1997. On the Problem of Spurious Patterns in Neural Associative Memory Models. IEEE Trans. Neural Network. Vol.8(6);1483~1491.

[2] Hagan., Demuth., Beale. 1996. Neural Network Design. PWS Press.

[3] Kah Kay. Sung., Tomaso Poggio. 1998. Example-Based Learning for View-Based Human Face Detection. IEEE Trans. on Pattern Analysis and Machine Intelligence. Vol.20(1);39~50.

[4] Lee, Dae-Weon. 1990. A robotic and vision system for locating and transferring container grown tobacco seedlings. Ph. D. Thesis Department of biological and agricultural engineering, North Carolina state university, Rleigh, NC.

[5] Pla, F., F.Juste, F. 1993. Color Segmentation based on a Light Reflection Model to Locate Citrus Fruits for Robotic Harvesting. Computer Electron. Agric. Vol.9(3):5 3~70.

[6] Harrell, R. C., P. D. Adsit. 1990. The Florida Robotic Grove-Lab. Trans of the ASAE Vol.8(33);391~399.

[7] 김상엽, 문종섭. 1996. 양방향 연상 메모리의 개선된 학습 방법. 정보과학회 논문지. Vol.23(3);280~286.

[8] 양현승. 1992. 신경회로망과 컴퓨터 비젼. 정보과학회지. Vol.10(2);39~48.

[9] William Kleitz. Digital Electronics. 1996

[10] Devaaemink, R. M. 1985. vision systems and robotics in food processing. ASAE and SME, Proceedings of the Agri-Mation I Conference & Exposition.

[11] Lea, Dae-Weon, A Robotic and Vision System for Locating and Transferring Container Grown Tabacco Seedling, Ph. D THesis, Department

of Biological and Agricultural Engineering, North Carolina State University, Raleigh, NC. 1990

[12] Hwang, H.,f. e. sistler. 1985. The implementation fo a robotic machine. ASAE and SME, Proceedings of the Agri-Mation I Confernce & Exposition.

[13] Mikell P. Groover, Mitchell Weiss, Roger N. Nagel, Nkcholas G. Odrey, Industrial Robotics; Technology, Programming, and Applications, McGraw-Hill Book Company, 1986

[14] KIM Ki Kae, Saigehisa OZAKI Takayuki KOGIMA. Development of an automatic robot system for a vegetable factory. 1955. JSAM Vol.1

[15] Naoshi KONDO, Yoshiaki NISHITSUJI, Yasunori SHIBANO, Kentaro MOHRI, Mitsuji MONTA, Hisaya YAMADA. 1995. Visual feedback control of pretty-tomato harvesting robot. JSAM Vol.1

<부 록> 영상처리 프로그램

```
// ImageSource.h: interface for the CImageSource class.
//
#if !defined(AFX_IMAGESOURCE_H__D1063B98_0F04_40F6_AB86_E31DCB9DD6D7__INCLUDED_)
#define AFX_IMAGESOURCE_H__D1063B98_0F04_40F6_AB86_E31DCB9DD6D7__INCLUDED_

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

class CImageSource
{
public:
        float HLSValue(float m1, float m2, float H);
        void HLSToRGB();
        void RGBToHLS();
        void BuffColor();
        void TempColor();
        void ResultColor();
        void InitTempColor();
        void InitBuffColor();
        void HistogramGray();
        void HistogramColor();
        void SobelMaskColor();
        void CannyMask();
        void Thining();
        void LaplacianMask();
        void RobertMask();
        void PrewittMask();
        void SobelMask();
        void HistogramStColor();
        void HistogramSt();
        void HistogramEqColor();
        void HistogramEq();
        int GetOrgRValue(int i, int j);
        int GetOrgGValue(int i, int j);
        int GetOrgBValue(int i, int j);
        int GetOrgGrayValue(int i, int j);
        int GetRstRValue(int i, int j);
        int GetRstGValue(int i, int j);
```

```
        int GetRstBValue(int i, int j);
        int GetRstGrayValue(int i, int j);
        int GetTmpRValue(int i, int j);
        int GetTmpGValue(int i, int j);
        int GetTmpBValue(int i, int j);
        int GetTmpGrayValue(int i, int j);
        int GetBufRValue(int i, int j);
        int GetBufGValue(int i, int j);
        int GetBufBValue(int i, int j);
        int GetBufGrayValue(int i, int j);

        CImageSource();
        virtual ~CImageSource();

        COLORREF m_cOrgColor[320][240];
        COLORREF m_cRstColor[320][240];
        COLORREF m_cTmpColor[320][240];
        COLORREF m_cBufColor[320][240];
        COLORREF m_cHLSColor[320][240];

        int HistoGray[256];
        int HistoRed[256];
        int HistoGreen[256];
        int HistoBlue[256];
        int Sum_Of_HistoGray[256];
        int Sum_Of_HistoRed[256];
        int Sum_Of_HistoGreen[256];
        int Sum_Of_HistoBlue[256];

        int Gray_HistoLevel[240][256];

        float m_fHColor[320][240];
        float m_fSColor[320][240];
        float m_fLColor[320][240];

};

#endif // !defined(AFX_IMAGESOURCE_H__D1063B98_0F04_40F6_AB86_E31DCB9DD6D7__INCLUDED_)
```

```
// ImageSource.cpp: implementation of the CImageSource class.
//

#include "stdafx.h"
#include "ImageSource.h"
#include "math.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CImageSource::CImageSource()
{
}

CImageSource::~CImageSource()
{
}
int CImageSource::GetOrgRValue(int i, int j)
{
        return GetRValue(m_cOrgColor[i][j]);
}

int CImageSource::GetOrgGValue(int i, int j)
{
        return GetGValue(m_cOrgColor[i][j]);
}

int CImageSource::GetOrgBValue(int i, int j)
{
        return GetBValue(m_cOrgColor[i][j]);
}

int CImageSource::GetOrgGrayValue(int i, int j)
{
        return (int)((GetOrgRValue(i, j) + GetOrgBValue(i, j) + GetOrgGValue(i, j))/3)
}
```

```
int CImageSource::GetRstRValue(int i, int j)
{
        return GetRValue(m_cRstColor[i][j]);
}

int CImageSource::GetRstGValue(int i, int j)
{
        return GetGValue(m_cRstColor[i][j]);
}

int CImageSource::GetRstBValue(int i, int j)
{
        return GetBValue(m_cRstColor[i][j]);
}

int CImageSource::GetRstGrayValue(int i, int j)
{
        return (int)((GetRstRValue(i, j) + GetRstBValue(i, j) + GetRstGValue(i, j))/3)
}

int CImageSource::GetTmpRValue(int i, int j)
{
        return GetRValue(m_cTmpColor[i][j]);
}

int CImageSource::GetTmpGValue(int i, int j)
{
        return GetGValue(m_cTmpColor[i][j]);
}

int CImageSource::GetTmpBValue(int i, int j)
{
        return GetBValue(m_cTmpColor[i][j]);
}

int CImageSource::GetTmpGrayValue(int i, int j)
{
        return (int)((GetTmpRValue(i, j) + GetTmpBValue(i, j) + GetTmpGValue(i, j))/3)
}

int CImageSource::GetBufRValue(int i, int j)
{
        return GetRValue(m_cBufColor[i][j]);
}
```

```
int CImageSource::GetBufGValue(int i, int j)
{
	return GetGValue(m_cBufColor[i][j]);
}

int CImageSource::GetBufBValue(int i, int j)
{
	return GetBValue(m_cBufColor[i][j]);
}

int CImageSource::GetBufGrayValue(int i, int j)
{
	return (int)((GetBufRValue(i, j) + GetBufBValue(i, j) + GetBufGValue(i, j))/3)
}

void CImageSource::InitTempColor()
{
	int i, j;

	for (j=0; j<240; j++)
	{
		for (i=0; i<320; i++)
		{
			m_cTmpColor[i][j] = RGB(0, 0, 0);
		}
	}
}

void CImageSource::InitBuffColor()
{
	int i, j;

	for (j=0; j<240; j++)
	{
		for (i=0; i<320; i++)
		{
			m_cBufColor[i][j] = RGB(0, 0, 0);
		}
	}
}

void CImageSource::ResultColor()
{
	int i, j;
```

```
        for (j=0; j<240; j++)
        {
                for (i=0; i<320; i++)
                {
                        m_cRstColor[i][j] = m_cTmpColor[i][j]
                }
        }
}

void CImageSource::TempColor()
{
        int i, j;

        InitTempColor();

        for (j=0; j<240; j++)
        {
                for (i=0; i<320; i++)
                {
                        m_cTmpColor[i][j] = m_cRstColor[i][j]
                }
        }
}

void CImageSource::BuffColor()
{
        int i, j;

        InitBuffColor();

        for (j=0; j<240; j++)
        {
                for (i=0; i<320; i++)
                {
                        m_cBufColor[i][j] = m_cRstColor[i][j]
                }
        }
}

void CImageSource::HistogramEq()
{
        int i, j, z;
        int k = 0, sum = 0, TotalPixel = 0;

        for (i=0; i<256; i++)
```

```
        {
                HistoGray[i] = 0;
                Sum_Of_HistoGray[i] = 0;
        }

        for (j=0; j<240; j++)
        {
                for (i=0; i<320; i++)
                {
                        k = GetRstGrayValue(i, j);
                        HistoGray[k] = HistoGray[k] + 1;
                }
        }

        for (i=0; i<256; i++)
        {
                sum = sum + HistoGray[i];
                Sum_Of_HistoGray[i] = sum;
        }

        TotalPixel = 320 * 240;

        for (j=0; j<240; j++)
        {
                for (i=0; i<320; i++)
                {
                        k = GetRstGrayValue(i, j);
                        z = (int)(Sum_Of_HistoGray[k] * (255.0 / TotalPixel))

                        m_cRstColor[i][j] = RGB(z, z, z);
                }
        }
}

void CImageSource::HistogramEqColor()
{
        int i, j, r, g, b;
        int rk = 0, gk = 0, bk = 0;
        int rsum = 0, gsum = 0, bsum = 0;
        int TotalPixel = 0;

        for (i=0; i<256; i++)
        {
                HistoRed[i] = 0;
                HistoGreen[i] = 0;
                HistoBlue[i] = 0;
```

```
            Sum_Of_HistoRed[i] = 0;
            Sum_Of_HistoGreen[i] = 0;
            Sum_Of_HistoBlue[i] = 0;
        }

        for (j=0; j<240; j++)
        {
            for (i=0; i<320; i++)
            {
                rk = GetRstRValue(i, j);
                HistoRed[rk] = HistoRed[rk] + 1;
                gk = GetRstGValue(i, j);
                HistoGreen[gk] = HistoGreen[gk] + 1;
                bk = GetRstBValue(i, j);
                HistoBlue[bk] = HistoBlue[bk] + 1;
            }
        }

        for (i=0; i<256; i++)
        {
            rsum = rsum + HistoRed[i];
            Sum_Of_HistoRed[i] = rsum;
            gsum = gsum + HistoGreen[i];
            Sum_Of_HistoGreen[i] = gsum;
            bsum = bsum + HistoBlue[i];
            Sum_Of_HistoBlue[i] = gsum;
        }

        TotalPixel = 320 * 240;

        for (j=0; j<240; j++)
        {
            for (i=0; i<320; i++)
            {
                rk = GetRstRValue(i, j);
                gk = GetRstGValue(i, j);
                bk = GetRstBValue(i, j);
                r = (int)(Sum_Of_HistoRed[rk] * (255.0 / TotalPixel));
                g = (int)(Sum_Of_HistoGreen[gk] * (255.0 / TotalPixel))
                b = (int)(Sum_Of_HistoBlue[bk] * (255.0 / TotalPixel));

                m_cRstColor[i][j] = RGB(r, g, b);
            }
        }
}
```

```
void CImageSource::HistogramSt()
{
        int i, j, z;
        int lowthresh = 0, highthresh = 255;
        int LookUpTable[256];
        float scalefactor;

        for (i=0; i<256; i++)
        {
                HistoGray[i] = 0;
        }

        for (j=0; j<240; j++)
        {
                for (i=0; i<320; i++)
                {
                        HistoGray[GetRstGrayValue(i, j)]++
                }
        }

        for (i=0; i<256; i++)
        {
                if (HistoGray[i])
                {
                        lowthresh = i;
                        break;
                }
        }

        for (i=255; i>0; i--)
        {
                if (HistoGray[i])
                {
                        highthresh = i;
                        break;
                }
        }

        for (i=0; i<lowthresh; i++)
        {
                LookUpTable[i] = 0;
        }

        for (i=highthresh; i>0; i--)
        {
```

```
                LookUpTable[i] = 255;
        }

        scalefactor = (float)(255.0 / (highthresh - lowthresh));

        for (i=lowthresh; i<highthresh; i++)
        {
                LookUpTable[i] = (int)((i - lowthresh) * scalefactor)
        }

        for (j=0; j<240; j++)
        {
                for (i=0; i<320; i++)
                {
                        z = LookUpTable[GetRstGrayValue(i, j)];
                        m_cRstColor[i][j] = RGB(z, z, z);
                }
        }
}

void CImageSource::HistogramStColor()
{
        int i, j, r, g, b;
        int lowthreshr = 0, highthreshr = 255;
        int lowthreshg = 0, highthreshg = 255;
        int lowthreshb = 0, highthreshb = 255;
        int LookUpTableRed[256];
        int LookUpTableGreen[256];
        int LookUpTableBlue[256];
        float rscalefactor;
        float gscalefactor;
        float bscalefactor;

        for (i=0; i<256; i++)
        {
                HistoRed[i] = 0;
                HistoGreen[i] = 0;
                HistoBlue[i] = 0;
        }

        for (j=0; j<240; j++)
        {
                for (i=0; i<320; i++)
                {
                        HistoRed[GetRstRValue(i, j)]++;
                        HistoGreen[GetRstGValue(i, j)]++;
```

```
                HistoBlue[GetRstBValue(i, j)]++
        }
}

for (i=0; i<256; i++)
{
        if (HistoRed[i])
        {
                lowthreshr = i;
                break;
        }
}

for (i=0; i<256; i++)
{
        if (HistoGreen[i])
        {
                lowthreshg = i;
                break;
        }
}

for (i=0; i<256; i++)
{
        if (HistoBlue[i])
        {
                lowthreshb = i;
                break;
        }
}

for (i=255; i>0; i--)
{
        if (HistoRed[i])
        {
                highthreshr = i;
                break;
        }
}

for (i=255; i>0; i--)
{
        if (HistoGreen[i])
        {
                highthreshg = i;
                break;
```

```
        }
}

for (i=255; i>0; i--)
{
        if (HistoBlue[i])
        {
                highthreshb = i;
                break;
        }
}

for (i=0; i<lowthreshr; i++)
{
        LookUpTableRed[i] = 0;
}

for (i=0; i<lowthreshg; i++)
{
        LookUpTableGreen[i] = 0;
}

for (i=0; i<lowthreshb; i++)
{
        LookUpTableBlue[i] = 0;
}

for (i=highthreshr; i>0; i--)
{
        LookUpTableRed[i] = 255;
}

for (i=highthreshg; i>0; i--)
{
        LookUpTableGreen[i] = 255;
}

for (i=highthreshb; i>0; i--)
{
        LookUpTableBlue[i] = 255;
}

rscalefactor = (float)(255.0 / (highthreshr - lowthreshr))
gscalefactor = (float)(255.0 / (highthreshg - lowthreshg))
bscalefactor = (float)(255.0 / (highthreshb - lowthreshb))
```

```
        for (i=lowthreshr; i<highthreshr; i++)
        {
                LookUpTableRed[i] = (int)((i - lowthreshr) * rscalefactor);
        }

        for (i=lowthreshg; i<highthreshg; i++)
        {
                LookUpTableGreen[i] = (int)((i - lowthreshg) * gscalefactor)
        }

        for (i=lowthreshb; i<highthreshb; i++)
        {
                LookUpTableBlue[i] = (int)((i - lowthreshb) * bscalefactor);
        }

        for (j=0; j<240; j++)
        {
                for (i=0; i<320; i++)
                {
                        r = LookUpTableRed[GetRstRValue(i, j)];
                        g = LookUpTableGreen[GetRstGValue(i, j)];
                        b = LookUpTableBlue[GetRstBValue(i, j)];
                        m_cRstColor[i][j] = RGB(r, g, b);
                }
        }
}

void CImageSource::SobelMask()
{
        int i, j, x, y;
        int CenterValueX = 0, CenterValueY = 0;
        int sum = 0;
        int SobelMaskX[3][3] = {-1, 0, 1,
                                                        -2, 0, 2,
                                                        -1, 0, 1};
        int SobelMaskY[3][3] = {1, 2, 1,
                                                        0, 0, 0,
                                                        -1, -2, -1}

        InitTempColor();

        for (j=0; j<237; j++)
        {
                for (i=0; i<317; i++)
                {
                        for (y=0; y<3; y++)
```

```
                        {
                                for(x=0; x<3; x++)
                                {
                                        CenterValueX   +=   GetRstGrayValue(i+x
j+y) * SobelMaskX[x][y];
                                        CenterValueY   +=   GetRstGrayValue(i+x
j+y) * SobelMaskY[x][y];
                                }
                        }
                        sum = abs(CenterValueX) + abs(CenterValueY);
                        if (sum > 255)
                        {
                                sum = 255;
                        }

                        m_cTmpColor[i+1][j+1] = RGB(sum, sum, sum);
                        CenterValueX = 0;
                        CenterValueY = 0;
                        sum = 0;
                }
        }

        ResultColor();
        for(j=0; j<240; j++)
        {
                for(i=0; i<320; i++)
                {
                        if (GetRstGrayValue(i, j) < 100)
                        {
                                m_cRstColor[i][j] = RGB(0, 0, 0);
                        }
                        else
                        {
                                m_cRstColor[i][j] = RGB(255, 255, 255);
                        }
                }
        }
}

void CImageSource::PrewittMask()
{
        int i, j, x, y;
        int CenterValueX = 0, CenterValueY = 0;
        int sum = 0;
        int PrewittMaskX[3][3] = {-1, 0, 1,
                                                                -1, 0, 1,
```

```
                                                        -1, 0, 1};
        int PrewittMaskY[3][3] = {1, 1, 1,
                                                        0, 0, 0,
                                                        -1, -1, -1};


        InitTempColor();

        for (j=0; j<237; j++)
        {
                for (i=0; i<317; i++)
                {
                        for (y=0; y<3; y++)
                        {
                                for(x=0; x<3; x++)
                                {
                                        CenterValueX   +=   GetRstGrayValue(i+x
j+y) * PrewittMaskX[x][y];
                                        CenterValueY   +=   GetRstGrayValue(i+x
j+y) * PrewittMaskY[x][y];
                                }
                        }
                        sum = abs(CenterValueX) + abs(CenterValueY);
                        if (sum > 255)
                        {
                                sum = 255;
                        }

                        m_cTmpColor[i+1][j+1] = RGB(sum, sum, sum);
                        CenterValueX = 0;
                        CenterValueY = 0;
                        sum = 0;
                }
        }

        ResultColor();
        for(j=0; j<240; j++)
        {
                for(i=0; i<320; i++)
                {
                        if (GetRstGrayValue(i, j) < 25)
                        {
                                m_cRstColor[i][j] = RGB(0, 0, 0);
                        }
                        else
                        {
```

```
                                        m_cRstColor[i][j] = RGB(255, 255, 255);
                                }
                        }
                }
}

void CImageSource::RobertMask()
{
        int i, j, x, y;
        int CenterValueX = 0, CenterValueY = 0;
        int sum = 0;
        int RobertMaskX[3][3] = {0, 0, -1,
                                                                0, 1, 0,
                                                                0, 0, 0};
        int RobertMaskY[3][3] = {-1, 0, 0,
                                                                0, 1, 0,
                                                                0, 0, 0};

        InitTempColor();

        for (j=0; j<237; j++)
        {
                for (i=0; i<317; i++)
                {
                        for (y=0; y<3; y++)
                        {
                                for(x=0; x<3; x++)
                                {
                                        CenterValueX    +=   GetRstGrayValue(i+x
j+y) * RobertMaskX[x][y];
                                        CenterValueY    +=   GetRstGrayValue(i+x
j+y) * RobertMaskY[x][y];
                                }
                        }
                        sum = abs(CenterValueX) + abs(CenterValueY);
                        if (sum > 255)
                        {
                                sum = 255;
                        }

                        m_cTmpColor[i+1][j+1] = RGB(sum, sum, sum);
                        CenterValueX = 0;
                        CenterValueY = 0;
                        sum = 0;
                }
        }
```

```
	ResultColor();
	for(j=0; j<240; j++)
	{
		for(i=0; i<320; i++)
		{
			if (GetRstGrayValue(i, j) < 25)
			{
				m_cRstColor[i][j] = RGB(0, 0, 0);
			}
			else
			{
				m_cRstColor[i][j] = RGB(255, 255, 255);
			}
		}
	}
}

void CImageSource::LaplacianMask()
{
	int i, j, x, y;
	int CenterValue = 0;
	int sum = 0;
	int LaplacianMask[3][3] = {-1, -1, -1,
                                   -1, 8, -1,
                                   -1,    -1
-1};

	InitTempColor();

	for (j=0; j<237; j++)
	{
		for (i=0; i<317; i++)
		{
			for (y=0; y<3; y++)
			{
				for(x=0; x<3; x++)
				{
					CenterValue   +=   GetRstGrayValue(i+x
j+y) * LaplacianMask[x][y];
				}
			}
			sum = abs(CenterValue);
			if (sum > 255)
			{
				sum = 255;
```

```
                        }

                        m_cTmpColor[i+1][j+1] = RGB(sum, sum, sum);
                        CenterValue = 0;
                        sum = 0;
                }
        }

        ResultColor();
        for(j=0; j<240; j++)
        {
                for(i=0; i<320; i++)
                {
                        if (GetRstGrayValue(i, j) < 40)
                        {
                                m_cRstColor[i][j] = RGB(0, 0, 0);
                        }
                        else
                        {
                                m_cRstColor[i][j] = RGB(255, 255, 255)
                        }
                }
        }
}

void CImageSource::Thining()
{
        int i, j;
        int ca = 0, cb = 0, cc = 0, cd = 0, total = 0;
        int npl = 0, spl = 0, hv = 0;
        int count = 0, check = 0, flag = 0, cz = 0;

        do
        {
                for (j=1; j<239; j++)
                {
                        for (i=1; i<319; i++)
                        {
                                m_cTmpColor[i][j] = RGB(0, 0, 0);
                        }
                }

                flag = 0;
                check = count % 2;
                count++;
```

```
for (j=1; j<239; j++)
{
        for (i=1; i<319; i++)
        {
                if (GetRstGrayValue(i, j) == 255)
                {
                        ca = 0;
                        cb = 0;
                        cc = 0;
                        cd = 0;
                        spl = 0;

        npl = GetRstGrayValue(i-1, j-1) + GetRstGrayValue(i-1, j)
        + GetRstGrayValue(i-1, j+1) + GetRstGrayValue(i, j-1)
        + GetRstGrayValue(i, j+1) + GetRstGrayValue(i+1, j-1)
        + GetRstGrayValue(i+1, j) + GetRstGrayValue(i+1, j+1);
                        if (npl >= 2*255 && npl <= 6*255)
                        {
                                ca = 0;
                        }
                        else
                        {
                                ca = 1;
                        }

if (GetRstGrayValue(i-1, j) == 0 && GetRstGrayValue(i-1, j+1) == 255)
                        {
spl++;
                        }
if (GetRstGrayValue(i-1, j+1) == 0 && GetRstGrayValue(i, j+1) == 255)
                        {
                                spl++;
                        }
if (GetRstGrayValue(i, j+1) == 0 && GetRstGrayValue(i+1, j+1) == 255)
                        {
                                spl++;
                        }
if (GetRstGrayValue(i+1, j+1) == 0 && GetRstGrayValue(i+1, j) == 255)
                        {
                                spl++;
                        }
if (GetRstGrayValue(i+1, j) == 0 && GetRstGrayValue(i+1, j-1) == 255)
                        {
                                spl++;
                        }
if (GetRstGrayValue(i+1, j-1) == 0 && GetRstGrayValue(i, j-1) == 255)
```

```
                                        {
                                                spl++;
                                        }
                if (GetRstGrayValue(i, j-1) == 0 && GetRstGrayValue(i-1, j-1) == 255)
                                        {
                                                spl++;
                                        }
                if (GetRstGrayValue(i-1, j-1) == 0 && GetRstGrayValue(i-1, j) == 255)
                                        {
                                                spl++;
                                        }
                                        if (spl == 1)
                                        {
                                                cb = 0;
                                        }
                                        else
                                        {
                                                cb = 1;
                                        }

                                        if (check == 0)
                                        {
cc = GetRstGrayValue(i-1, j) * GetRstGrayValue(i, j+1) * GetRstGrayValue(i+1, j);
cd = GetRstGrayValue(i, j+1) * GetRstGrayValue(i+1, j) * GetRstGrayValue(i, j-1);
                                        }
                                        else
                                        {
cc = GetRstGrayValue(i-1, j) * GetRstGrayValue(i, j+1) * GetRstGrayValue(i, j-1);
cd = GetRstGrayValue(i-1, j) * GetRstGrayValue(i+1, j) * GetRstGrayValue(i, j-1);
                                        }

                                        total = ca || cb || cc || cd;
                                        if (total)
                                        {
                                        m_cTmpColor[i][j] = RGB(255, 255, 255);
                                        }
                                        else
                                        {
                                                flag = 1;
                                        }
                                }
                        }
                }
                for (j=1; j<239; j++)
                {
                        for (i=1; i<319; i++)
```

```
                        {
                                m_cRstColor[i][j] = m_cTmpColor[i][j];
                        }
                }
        } while (flag);

        for (j=1; j<239; j++)
        {
                for (i=1; i<319; i++)
                {
                        hv = 0;
                        if (GetRstGrayValue(i, j) == 255)
                        {
                if (GetRstGrayValue(i-1, j) == 255 && GetRstGrayValue(i, j+1) == 255)
                                {
                                        hv++;
                                }
                if (GetRstGrayValue(i, j+1) == 255 && GetRstGrayValue(i+1, j) == 255)
                                {
                                        hv++;
                                }
                if (GetRstGrayValue(i+1, j) == 255 && GetRstGrayValue(i, j-1) == 255)
                                {
                                        hv++;
                                }
                if (GetRstGrayValue(i, j-1) == 255 && GetRstGrayValue(i-1, j) == 255)
                                {
                                        hv++;
                                }

                                if (hv == 1)
                                {
                                        m_cRstColor[i][j] = RGB(0, 0, 0);
                                }
                        }
                }
        }
}

void CImageSource::CannyMask()
{
        int i, j, x, y;
        int c = 0, cc = 0;
        int CenterValue = 0;
        int CannyMask[5][5] = {2, 4, 5, 4, 2,
                                                        4, 9, 12, 9, 4,
```

```
                                                    5, 12, 15, 12, 5,
                                                    4, 9, 12, 9, 4,
                                                    2, 4, 5, 4, 2};


        InitTempColor();

        for (j=0; j<240; j++)
        {
                for (i=0; i<320; i++)
                {
                        for (y=0; y<5; y++)
                        {
                                for(x=0; x<5; x++)
                                {
                        CenterValue += GetRstGrayValue(i+x, j+y) * CannyMask[x][y];
                                }
                        }
                        if (CenterValue > 255)
                        {
                                CenterValue = 255;
                        }

m_cTmpColor[i+2][j+2] = RGB((int)CenterValue, (int)CenterValue, (int)CenterValue);
                        CenterValue = 0;
                }
        }

        for (j=1; j<240; j++)
        {
                for (i=1; i<320; i++)
                {
                        cc = -GetRstGrayValue(i-1, j-1) - 2 * GetRstGrayValue(i-1
j) - GetRstGrayValue(i-1, j+1);
                        cc += GetRstGrayValue(i+1, j-1) + 2 * GetRstGrayValue(i+1
j) + GetRstGrayValue(i+1, j+1);
                        c = abs(cc);

                        cc = -GetRstGrayValue(i-1, j-1) - 2 * GetRstGrayValue(i
j-1) - GetRstGrayValue(i+1, j-1);
                        cc += GetRstGrayValue(i-1, j+1) + 2 * GetRstGrayValue(i
j+1) + GetRstGrayValue(i+1, j+1);
                        c += abs(cc);

                        if (c > 255)
                        {
```

```
                                        c = 255;
                                }
                                else if (c < 100)
                                {
                                        c = 0;
                                }
                                m_cTmpColor[i][j] = RGB((int)c, (int)c, (int)c);
                        }
                }

                ResultColor();
                for(j=0; j<240; j++)
                {
                        for(i=0; i<320; i++)
                        {
                                if (GetRstGrayValue(i, j) < 100)
                                {
                                        m_cRstColor[i][j] = RGB(0, 0, 0);
                                }
                                else
                                {
                                        m_cRstColor[i][j] = RGB(255, 255, 255)
                                }
                        }
                }
}
void CImageSource::HistogramColor()
{
        int i, j;
        int rk = 0, gk = 0, bk = 0;

        for (i=0; i<256; i++)
        {
                HistoRed[i] = 0;
                HistoGreen[i] = 0;
                HistoBlue[i] = 0;
        }

        for (j=0; j<240; j++)
        {
                for (i=0; i<320; i++)
                {
                        rk = GetRstRValue(i, j);
                        HistoRed[rk] = HistoRed[rk] + 1;
                        gk = GetRstGValue(i, j);
                        HistoGreen[gk] = HistoGreen[gk] + 1;
```

```
                        bk = GetRstBValue(i, j);
                        HistoBlue[bk] = HistoBlue[bk] + 1;
                }
        }
}

void CImageSource::HistogramGray()
{
        int i, j;
        int k = 0;

        for (i=0; i<256; i++)
        {
                HistoGray[i] = 0;
        }

        for (j=0; j<240; j++)
        {
                for (i=0; i<320; i++)
                {
                        k = GetRstGrayValue(i, j);
                        HistoGray[k] = HistoGray[k] + 1;
                }
        }
}

void CImageSource::RGBToHLS()
{
        int i, j;
        float max, min, R, G, B, delta, H, L, S;

        for (j=0; j<240; j++)
        {
                for (i=0; i<320; i++)
                {
                        R = (float)(GetRstRValue(i, j) / 255.)
                        G = (float)(GetRstGValue(i, j) / 255.)
                        B = (float)(GetRstBValue(i, j) / 255.)

                        max = R;
                        if (max < G)
                        {
                                max = G;
                        }
                        if (max < B)
                        {
```

```
        max = B;
}

min = R;
if (min > G)
{
        min = G;
}
if (min > B)
{
        min = B;
}

L = (float)((max + min) / 2.);
if (max == min)
{
        S = 0.;
        H = (float)-0.1;
}
else
{
        delta = max - min;
        if (L < 0.5)
        {
                S = (float)(delta / (max + min));
        }
        else
        {
                S = (float)(delta / (2. - max - min))
        }
        if (R == max)
        {
                H = (float)((G - B) / delta);
        }
        else if (G == max)
        {
                H = (float)(2. + (B - R) / delta);
        }
        else if (B == max)
        {
                H = (float)(4 + (R - G) / delta);
        }
        H *= 60.;
        if (H < 0.)
        {
                H += 360.;
```

```
                        }
                    }
                    m_fHColor[i][j] = H;
                    m_fLColor[i][j] = L;
                    m_fSColor[i][j] = S;
                }
            }
}

void CImageSource::HLSToRGB()
{
        int i, j, r, g, b;
        float m1, m2, R, G, B, H, L, S;

        for (j=0; j<240; j++)
        {
                for (i=0; i<320; i++)
                {
                        H = m_fHColor[i][j];
                        L = m_fLColor[i][j];
                        S = m_fSColor[i][j];

                        if (L <= 0.5)
                        {
                                m2 = (float)(L * (1. + S));
                        }
                        else
                        {
                                m2 = (float)(L + S - L * S);
                        }
                        m1 = (float)(2. * L - m2);
                        if (S == 0.)
                        {
                                if (H == (float)-0.1)
                                {
                                        R = G = B = L;
                                }
                                else
                                {
                                        R = G = B = 0.;
                                }
                        }
                        else
                        {
                                R = HLSValue(m1, m2, (float)(H + 120.))
                                G = HLSValue(m1, m2, H);
```

```
                                B = HLSValue(m1, m2, (float)(H - 120.))
                        }
                        r = (int)(R * 255.);
                        g = (int)(G * 255.);
                        b = (int)(B * 255.);
                        if (r > 255)
                        {
                                r = 255;
                        }
                        if (g > 255)
                        {
                                g = 255;
                        }
                        if (b > 255)
                        {
                                b = 255;
                        }
                        if (r < 0)
                        {
                                r = 0;
                        }
                        if (g < 0)
                        {
                                g = 0;
                        }
                        if (b < 0)
                        {
                                b = 0;
                        }
                        m_cHLSColor[i][j] = RGB(r, g, b);
                }
        }
}

float CImageSource::HLSValue(float m1, float m2, float H)
{
        if (H > 360.)
        {
                H -= 360.;
        }
        if (H < 0.)
        {
                H += 360.;
        }
        if (H < 60.)
        {
```

```
                return (float)(m1 + (m2 - m1) * H / 60.);
        }
        if (H < 180.)
        {
                return m2;
        }
        if (H < 240.)
        {
                return (float)(m1 + (m2 - m1) * (240. - H) / 60.)
        }
        return m1;
}
```