최 종
연구보고서

CAN 기반의 농업기계 무인화 기술 개발

Development of Autonomous Operation Technology for Agricultural Machinery Based on CAN (Controller Area Network)

연구기관

강원대학교

농림부

제 출 문

농림부 장관 귀하

본 보고서를 “CAN 기반의 농업기계 무인화 기술 개발” 과제의 최종보고서로 제출합니다.

2005 년 7 월 14 일

주관연구기관 : 강원대학교
총괄연구책임자 : 신 범 수
세부연구책임자 : 김 상 헌
연 구 원 : 최 영 대
연 구 원 : 박 재 언
연 구 원 : 이 수 성
연 구 원 : 김 성 용
연 구 원 : 박 대 순
연 구 원 : 전 영 표
연 구 원 : 이 상 기
연 구 원 : 조 재 복
연 구 원 : 오 재 성
연 구 원 : 이 경 민
연 구 원 : 방 은 준
연 구 원 : 방 제 천

요 약 문

Ⅰ. 제 목

CAN 기반의 농업기계 무인화 기술 개발

Ⅱ. 연구개발의 목적 및 필요성

농업 노동력의 수적 감소와 질적 저하는 개발된 농업기계마저도 그 운용에 어려움을 겪게 되고 또한 위험한 작업을 기피하는 추세는 농업생산성의 저하로 이어져 국내외 많은 농업기계공학자들은 농업 생산 시스템을 공장에서 공산품을 생산하는 방식으로 전환하기 위하여 새로운 개념의 농업 생산 시스템의 연구, 개발 노력을 경주하기 시작하였다. 농업용 로봇 개발에 관한 시도는 이러한 노력의 대표적 사례 중 하나이나 여타 산업 분야와 달리 특히 노지에서의 농작업은 매우 제한적이고 균일하지 못한 작업환경에서 이루어지기 때문에 무인화 기술을 확립하는 데 어려움이 있다.

센서 및 컴퓨터 등 전자기술의 발전은 균일치 못한 주변 환경이지만 농용로봇의 개발 가능성을 한층 높여 주었다. 특히 다양한 센서와 조작기 등을 사용해야 하는 농업기계는 컴퓨터 한 대로 모든 시스템을 중앙 제어하기에는 한계가 있다. 그래서, 무인주행 및 무인작업을 효율적으로 수행하기 위하여 다수의 컴퓨터 등을 사용하는 분산제어가 요구된다. CAN은 복잡해져 가는 자동차의 전장제어 시스템에 사용하기 위하여 개발된 분산제어의 일종으로 특히 와이어링하네스를 단출하게 구성해 주는 등 그 응용성이 모든 분야로 확대되어 가고 있는 실정이다. 농업기계도 예외가 아니라 이미 미국 등에서는 트랙터에도 CAN을 장착하여 그 효율성이 인정 받고 있어 앞으로 고성능 농업기계에 표준으로 장착될 것을 쉽게 예상할 수 있다.

따라서, 본 연구는 다수의 센서 및 조작기 등을 마이크로프로세서를 이용하여 CAN 버스로 연결하여 분산 제어 시스템을 구성하고, 그것을 무인주행 및 무인작업이 가능한 과수방제기의 개발에 적용함으로써 CAN의 농업기계 적용 가능성을 검토하고 또한 원천 기술을 확보하기 위해 수행되었다.

Ⅲ. 연구개발 내용 및 범위

1. 궤도형 주행장치의 무인/자동 조작기 개발

○ 주클러치/브레이크 레버 및 좌우측 조향 클러치 레버 조작기로서 유압실린더의 선정, 유압제어 시스템 분석 및 설계

○ 드로틀레버 조작기, 약액살포부 자동제어 시스템, 주행속도 측정장치 개발

2. 주행경로 결정 센서 개발

○ 전동식 트롤리 개발

○ 경사각도 측정장치 개발

○ 측면옵셑 및 방향각 측정 알고리즘 개발

○ 오버헤드 가이던스 시스템 개발

3. CAN 버스 시스템의 구조 분석 및 설계

○ CAN 원리 규명, 분석

○ CAN 버스 시스템 구축

○ CAN 구동 H/W 및 S/W 개발

4. 센서/조작기와 마이크로프로세서의 인터페이스 개발

○ 센서/조작기 인터페이스 개발

○ CAN 버스 시스템 구성 및 과수방제기 시스템 제어 구현

○ 원격제어가 가능하도록 무선조종기 시스템을 개발

○ ARM보드를 이용한 가상터미널 개발

5. 무인주행 및 방제작업제어 알고리즘 개발

○ 무인주행 제어 알고리즘 개발 : 퍼지 모델 개발

○ 컴퓨터 시뮬레이션 프로그램 개발

6. 약액살포제어기 개발

○ 약액탱크 수위 센서, 과수열 끝/시작 감지 센서 개발

○ 약액 살포기의 노즐 제어기 개발.

7. 성능 평가

○ 무인주행 성능 평가

○ 무인작업 성능 평가

Ⅳ. 연구개발 결과 및 활용에 대한 건의

1. 연구 개발 결과

본 연구에서는 다양한 센서와 조작기 등을 마이크로프로세서에 인터페이스하여 CAN 버스를 구성한 후 분산제어 방식을 통해 과수방제기를 무인화할 수 있는 기술을 개발하였다. 과수원에서 현장 성능 평가 결과 과수방제기는 오버헤드 가이던스 레일을 추종하여 하나의 조향 제어 알고리즘으로 약액살포 구간인 직선 구간과 다음 과수열로 이동, 진입하기 위한 곡선 구간에서 원활하게 제어가 가능한 것으로 평가되었다.

구체적으로 본 연구를 통해 얻어진 결과는 다음과 같다.

가. 궤도형 주행장치를 장착한 과수방제기의 무인화를 위하여 적정한 유압실린더를 사용하여 무인/자동 조작기를 개발하였다. 유압 시스템의 정밀 분석을 통해 시스템 설계 기술을 확보하였다. 이 외에 전동실린더와 선형포텐시오메터를 사용한 드로틀레버 조작기, 약액살포부 on/off 자동제어 시스템, 주행속도 측정장치 등이 개발되었다.

나. 기계적 방식으로 차량의 자세(측면옵셑과 방향각)를 측정할 수 있는 주행경로 결정 센서와 측정 알고리즘이 개발되었다. 오버헤드 가이던스 레일을 설계하여 과수원에 설치하였으며 DC모터로 구동되는 전동식 트롤리가 개발되었다. 케이블의 길이를 자동 조절되는 케이블 권치장치가 개발되었으며 이때 케이블의 각도를 측정할 수 있는 경사각도 측정장치가 개발되었다.

다. CAN의 작동원리 등 이론적 규명을 통해 CAN 기능이 있는 마이크로프로세서를 구동하여 CAN 통신을 할 수 있는 프로그램을 개발하였다. T89C51CC01 및 PIC18F448 마이크로프로세서를 기반으로 하는 ECU를 설계하여 PCB로 제작하였다.

라. 아날로그전압, 카운터, 디지털 스위치 등의 센서/조작기와 마이크로프로세서의 인터페이스 회로가 되었으며 총 16개의 ECU로 CAN 버스 시스템을 구성하여 분산제어를 구현하였다.

마. 과수열 끝/시작 감지 센서를 개발하고 약액탱크 수위센서, 주행속도 측정센서를 이용하여 약액살포 제어기를 구성하였다.

바. 궤도형 차량의 이동변위에 따른 수학적 모델을 개발하고 시뮬레이션 프로그램을 개발하였다. 무인주행 제어 알고리즘으로 퍼지 제어기를 개발하였다.

2. 활용에 대한 건의

본 연구에서 개발한 CAN 버스 시스템은 센서와 조작기 등을 필요한 만큼 농업기계 제어 시스템, 농용 로봇 등에 쉽게 적용할 수 있다는 장점이 있다. 특히 제어 및 배선이 복잡한 경우에 그 효용성이 증대되는 바 상용화 단계로의 기술개발 연구가 추가적으로 필요하다. 또한 CAN 버스 기술이 적용된 과수방제기는 무인주행/무인작업에 대한 기술력이 확보되었으므로 상용화 단계로의 추가적인 개발 연구 후 현장에 적용하기 위한 시범 사업 등을 추진하여 농민들의 복지 향상에 기여해야 할 것으로 판단된다.

특히, 현장의 농업기계 정비공장 관계자들에 의하면 콤바인의 문제점 중 하나가 농한기 중에 쥐 등이 배선을 갉아먹어 고장이 났을 때 배선의 복잡함으로 인해 고장 수리에 어려움이 많다고 하는데, 관심있는 기업과 공동 연구를 추진하여 부분적으로라도 CAN 버스로 대체를 추진하여 콤바인 성능 개선에 기여하도록 해야 할 것이다. 본 연구에서 개발한 센서 또는 조작기의 마이크로프로세서와의 인터페이스 기술, CAN 버스 시스템 구성 및 구현 기술 등을 활용해 기존의 시스템에도 센서들과 조작기 등을 쉽고 간편하게 추가할 수 표준 시스템을 개발하여 농업기계의 부분 자동화 분야 및 일반 기계의 자동화 분야에 활용하도록 하여야 할 것이다.

과수방제기의 무인주행/무인작업을 위해 개발한 주행경로 결정 센서 및 제어 알고리즘에 대한 특허를 출원하며, 관심있는 기업체에 무인 과수방제기의 기술을 제공하고 공동 연구하여 지자체의 협조를 받아 과수원 무인 방제 시스템의 시범 사업을 추진할 필요가 있을 것으로 판단된다. 확보된 CAN 관련 기술을 필요로 하는 농업기계 관련 산업체에 적극 홍보·보급함으로써 세계 수준의 기술력을 확보할 수 있도록 해야 할 것이다.

SUMMARY

I. Title

Development of Autonomous Operation Technology for Agricultural Machinery Based on CAN (Controller Area Network)

II. Objectives and Necessity

Decreasing number and quality of farm labor make it getting hard to operate the agricultural machinery developed already. In addition, the trend of avoiding some dangerous work results in the decrease of agricultural productivity. Many agricultural engineers and scientists now start to pay attention to R&D about agricultural product system with new paradigm to convert the production system of agriculture into a kind of industrial production system. Initiating to develope an agricultural robot is one of typical examples of such an effort. Contrary to other industry, it is more difficult to succeed the implementation of autonomous technology because the agricultural work in a field is performed in very limited and in homogeneous working condition.

However, the possibilities of developing the agricultural robot is getting increased as the electronic technology of sensors and computers is rapidly developed even no matter how the environmental condition is irregular. The central control system with only single computer system for the control of agricultural machinery has some limitations because new agricultural machine uses many types of sensors and actuators to be automatically controlled. It is needed to adopt a distributed control system equipped with many computers for the agricultural machinery to perform efficiently some autonomous operations. CAN is one of distributed control system to be used in the sophisticated automobile electronics and its applicability is getting expanded to various areas including the agricultural machinery. John Deere has

equipped CAN bus on their large-scale tractors since 1992 and the efficiency fo CAN bus is now well recognized, therefore, it is easily expected that CAN bus may be equipped as a standard for high-power agricultural machinery.

Thus, this project was initiated to investigate the applicability of adopting CAN in the domestic agricultural machinery and to secure the fundamental and original technology, if applicable, by constructing a distributed control system with microprocessors interfaced with many types of sensors and actuators and applying that to a fully autonomous orchard sprayer.

III. Content and Scope of Research

Content and scope of research and development are as following:

1. Development of autonomous actuators in a crawler type travel equipment
   - Proper hydraulic cylinders as actuators for steering clutches and main clutch/brake are selected; analysis and design of hydraulic systems will be carried out.
   - Throttle lever actuator, automatic spraying controller and ground speed sensor are developed.
2. Travel path determining sensor
   - Electrically-operated trolley, Dual-axes tilting angle of a lever and Wire-wound device are developed with overhead guidance rail.
   - Measurement algorithm for lateral offset and heading angle is also developed.
3. Analysis and Design of CAN bus system
   - Principles of CAN are studied and analyzed.
   - CAN bus system will be constructed.
   - H/W and S/W to drive CAN are developed.
4. Development of microprocessor interface with sensors/actuators
   - Interface circuits for sensors/actuators with driver are developed.
   - CAN bus is constructed to implement the control of autonomous orchard sprayer.
   - Remote controller and Virtual terminal using ARM board are developed.

5. Development of autonomous driving/chemical application algorithm
   - Mathematical model is developed for the computer simulation.
   - Fuzzy controller is developed based on rule-based control.
   - Nozzle on/off controller for chemical applicator is developed
6. Development of chemical application control system
   - tank level sensor and tree row start/end sensor are developed.
7. Evaluation of system performance
   - autonomous guidance and chemical application are evaluated.

IV. Results and Suggestion

1. R & D results

A technology to automate the orchard sprayer was developed through the distributed control system consisted of CAN bus, where many ECUs were interfaced with various sensors and actuators. The evaluation of system performance in a orchard showed that the orchard sprayer could travel smoothly and securely by tracking the overhead guidance rail over the straight path, where the chemical was expected to be sprayed as well as over the turning path, where the orchard sprayer turned around to move to next tree row.

Following shows summary of results obtained from this research.

1) For automation of crawler type orchard sprayer, proper size of hydraulic cylinders were selected as the autonomous actuators. Design technology for the hydraulic system was obtained by thorough analysis of the system. In addition, the throttle lever actuator using a electrically operated cylinder and a linear potentiometer, the chemical application on/off controller and ground speed measuring device were developed.

2) Travel path determining sensor, a mechanical type sensor, was developed with the measurement algorithm to measure the attitude of sprayer in terms of lateral offset and heading angle. Overhead guidance rail system was devised and fabricated in the lab and the orchard farm. The electrically

operated trolley system was developed as well. The wire-winding device was developed in this research for the wire to be automatically adjusted by the distance between the trolley and the travel path determining sensor. Dual-axes tilting angle sensor was also developed to measure the angle the wire makes.

3) By thorough study about CAN principles the program for CAN communication over the microprocessors with CAN function was developed. Thus, ECUs based on T89C51CC01 and PIC 18F448 were designed and constructed with PCB.

4) Interface circuits for actuators and ensors such as analog voltage, pulse counter and digital switch were developed and the distributed control system was implemented with 16 ECUs.

5) Chemical application controller was developed using tree row start/end detecting sensor, tank liquid level sensor and ground speed sensor.

6) Mathematical model for the motion of orchard sprayer was developed, and the computer simulation program was written with MFC. Fuzzy logic controller was developed based on the autonomous travel control algorithm.

2. Suggestion

CAN bus system developed in this research could enable the agricultural machinery control system or agricultural robot to use as many sensors and actuators as needed. Specially, where control and wiring are too complicated, its efficiency is expected to be dramatically increased. Further research is, therefore, needed for the development of commercialization level. Since the know-how about the autonomous operation of orchard sprayer using CAN bus technology was obtained, the further research toward the commercialization is also needed. After that, a model project is necessary to be launched with the aid of local government, which will result in the improvement of farmer's welfare. As another example, CAN bus could replace some part of complicated wiring where the maintenance and repair service is hard to be done. Corporation research with a company will enhance the

performance of combine harvester. Also it is suggested that a simple sensor or actuator system with CAN function is developed to be used in the conventional system to improve its performance. Then this kind of technology could be applied in the partial automation of agricultural machinery or industrial machinery.

A patent for travel path determining sensor and control algorithm for autonomous orchard sprayer is needed to be applied and this technology is better be transferred to a company to make it realize in the real world. CAN related technology is specially needed to be provided for them to obtain the high-tech technology.

CONTENTS

Chapter 1. Introduction/Summary of Research · · · · · · · · · · 17

Section 1. Needs of research · 17

Section 2. Objectives and Content of research · · · · · · · · · · · · · 22

1. Objective of research · 22

2. Content of research · 22

Chapter 2. State of Arts · 26

Section 1. State of arts · 26

1. CAN bus · 26

2. Autonomous orchard sprayer · 26

3. Autonomous gantry system · 27

4. Autonomous guidance for tractor and combine harvester · · · · · · · 27

Section 2. Prospective of research · · · · · · · · · · · · · · · · · · 28

Chapter 3. Contents and Results · · · · · · · · · · · · · · · · · · 30

Section 1. Development of automated actuator · · · · · · · · · · · · · 30

1. Steering/Travelling actuator · 30

2. Throttle lever actuator · 68

3. Chemical applicator actuator · 69

4. Ground speed sensor · 71

Section 2. Development of travel path determining sensor · · · · · · · · 72

1. Travel path determining sensor · · · · · · · · · · · · · · · · · · · 72

2. Overhead guidance rail · 87

Section 3. Analysis and Design of CAN bus system · · · · · · · · · · · 90

1. CAN principles and operations · · · · · · · · · · · · · · · · · · · 90

2. Selection of microprocessor with CAN function · · · · · · · · · · · 98

3. Design and Construction of ECU based on CAN · · · · · · · · · · · 100

4. Development of CAN driver program · · · · · · · · · · · · · · · · 114

Section 4. Microprocessor Interface with sensors/actuators · · · · · · · 118
1. Hydraulic cylinder(steering/travelling) controller · · · · · · · · · 118
2. Throttle lever controller · 121
3. Chemical applicator controller · · · · · · · · · · · · · · · · · 123
4. Travel path determining sensor · · · · · · · · · · · · · · · · 125
5. Tilt sensor · 127
6. Ultrasonic sensor · 127
7. Ground speed/ravelled distance sensor · · · · · · · · · · · · · 136
8. Tree row end/start detecting sensor · · · · · · · · · · · · · · 139
9. Tank liquid level sensor · 140
10. Remote controller · 141
11. Virtual terminal · 143
Section 5. Integration of system on CAN bus · · · · · · · · · · · · · 146
1. ECU #15 : Ground speed sensor · · · · · · · · · · · · · · · · · 146
2. ECU #4 : Tilt sensor · 146
3. ECU #3 : Travel path determining sensor · · · · · · · · · · · · 146
4. ECU #1_1 : Fuzzy controller · · · · · · · · · · · · · · · · · · · 147
5. ECU #1 : Steering/Main clutch controller · · · · · · · · · · · · 147
6. ECU #2 : Trolley actuator · 148
7. ECU #8 : Travelled distance sensor · · · · · · · · · · · · · · · 149
8. ECU #5: Tree row end/start detecting sensor · · · · · · · · · · 149
9. ECU #6.1 : Left chemical applicator with liquid level sensor · · · 149
10. ECU #6.2 : Right chemical applicator · · · · · · · · · · · · · · 149
11. ECU #7 : Obstacle detecting sensor and emergency switch · · · 150
12. ECU #14 : Remocon actuator(Remocon Receiver) · · · · · · · · 150
13. ECU #14_1 : Remocon actuator(remocon Transmitter) · · · · · · 150
14. ECU #12 : Serial-CAN communication converter · · · · · · · · 150
15. ECU #14 : Throttle actuator · · · · · · · · · · · · · · · · · · · 151
16. ECU #11 : Geo sensor · 151
Section 6. Development of control algorithm · · · · · · · · · · · · · 155
1. Autonomous guidance control algorithm · · · · · · · · · · · · · 155

2. Chemical application control algorithm · · · · · · · · · · · · · · 170
Section 7. Evaluation of system performance · · · · · · · · · · · · · · 172
1. Performance of autonomous guidance control · · · · · · · · · · · 175
2. performance of chemical application control · · · · · · · · · · · · 175

Chapter 4 Achievements and Contributions · · · · · · · · · · · 179
Section 1. Achievements · 179
Section 2. Contribution to related area · · · · · · · · · · · · · · · · · 179

Chapter 5 Scheme to utilize Research Results · · · · · · · · · 181

Chapter 6 References · 182

Appendices · 184

목 차

제 1 장 연구개발과제의 개요 · 17
제 1 절 연구개발의 필요성 · 17
제 2 절 연구개발의 목표와 내용 · · · · · · · · · · · · · · · · · · 22
1. 연구개발 목표 · 22
2. 연구개발 내용 · 22

제 2 장 국내외 기술개발 현황 · · · · · · · · · · · · · · · · · · · 26
제 1 절 기술개발 현황 · 26
1. CAN 버스 · 26
2. 과수방제기의 무인주행 · 26
3. 갠트리시스템의 무인주행 기술 · · · · · · · · · · · · · · · · · 27
4. 트랙터, 콤바인 등의 무인주행 기술 · · · · · · · · · · · · · · · 27
제 2 절 앞으로의 전망 · 28

제 3 장 연구개발수행 내용 및 결과 · · · · · · · · · · · · · · · · 30
제 1 절 궤도형 주행장치의 무인/자동 조작기 개발 · · · · · · · · · · 30
1. 조향/주행장치 조작기 · 30
2. 드로틀레버 조작기 · 68
3. 약액살포 노즐 조작기 · 69
4. 주행속도 측정장치 · 71
제 2 절 주행경로 결정 센서 개발 · · · · · · · · · · · · · · · · · · 72
1. 주행경로 결정센서 · 72
2. 오버헤드 가이던스 레일 · 87
제 3 절 CAN 버스 시스템의 구조 분석 및 설계 · · · · · · · · · · · 90
1. CAN 특성 및 작동원리 · 90
2. 적합한 CAN의 선정 · 98
3. CAN기반의 ECU 설계 및 제작 · · · · · · · · · · · · · · · · 100
4. CAN 버스 구동 프로그램 개발 · · · · · · · · · · · · · · · · · 114

제 4 절 센서/조작기-마이크로프로세서의 인터페이스 · · · · · · · · · · 118
1. 유압실린더(조향/주행) 제어기 · · · · · · · · · · · · · · · · · · 118
2. 드로틀 레버 제어기 · 122
3. 약액살포 제어기 · 123
4. 주행경로 결정 센서 · 125
5. 경사센서 · 127
6. 초음파센서 · 127
7. 주행속도/이동거리 측정 센서 · · · · · · · · · · · · · · · · · · · 136
8. 과수열 끝/감지 센서 · 139
9. 약액탱크 수위 측정 센서 · 140
10. 리모트 콘트롤러 · 141
11. 가상터미널 · 143
제 5 절 CAN 버스 시스템 총합 · 146
1. ECU #15 : 주행속도 센서 · 146
2. ECU #4 : 경사 센서 · 146
3. ECU #3 : 주행경로 결정 센서 · · · · · · · · · · · · · · · · · · · 146
4. ECU #1_1 : 퍼지제어기 · 147
5. ECU #1 : 조향/주행 클러치 제어기 · · · · · · · · · · · · · · · · 147
6. ECU #2 : 트롤리 조작기 · 148
7. ECU #8 : 이동거리 측정 측정 센서 · · · · · · · · · · · · · · · · 149
8. ECU #5: 과수열 끝/시작점 감지 센서 · · · · · · · · · · · · · · · 149
9. ECU #6.1 : 좌측 약액살포 제어기와 약액탱크 수위센서 · · · · · · 149
10. ECU #6.2 : 우측 약액살포 제어기 · · · · · · · · · · · · · · · · 149
11. ECU #7 : 장매물감지 센서 및 비상정지 스위치 · · · · · · · · · · 150
12. ECU #14 : 리모트콘트롤 조작기(리모콘 수신부) · · · · · · · · · 150
13. ECU #14_1 : 리모트콘트롤 조작기(리모콘 송신부) · · · · · · · · 150
14. ECU #12 : 시리얼-CAN통신 변환기 · · · · · · · · · · · · · · · 150
15. ECU #14 : 드로틀 조작기 · 151
16. ECU #11 : 지자기 센서 · 151
제 6 절 무인주행 및 방제작업 제어 알고리즘 개발 · · · · · · · · · · · 155
1. 무인주행 제어 알고리즘 · 155

2. 방제작업 제어 알고리즘 · 170
제 7 절 성능 평가 · 172
1. 무인주행 제어 성능 · 172
2. 약액살포 제어 성능 · 175

제 4 장 목표달성도 및 관련분야에의 기여도 · · · · · · · · · · · 179
제 1 절 목표 달성도 · 179
제 2 절 관련 분야에의 기여도 · 179

제 5 장 연구개발결과의 활용계획 · · · · · · · · · · · · · · · · · 181

제 6 장 참고문헌 · 182

부록 · 184

제 1 장 연구개발과제의 개요

제 1 절 연구개발의 필요성

지난 세기까지 인류는 농업의 기계화를 통하여 획기적으로 농업 생산성을 향상시킴으로써 인류 복지 증진에 지대한 공헌을 하여왔다. 그러나 급속한 산업화와 도시화는 농촌 인구를 도시로 또한 타 산업분야로 이동하게 만듦으로써 농촌 노동력의 절대적 감소로 이어졌고 더구나 농촌에는 떠날 수 없는 노년층만 남게 되어 노동력의 질적 저하는 심각한 상황이라고 할 수 있다. 특히 출산율 저하는 젊은 노동력이 농업생산 현장에 공급될 수 있는 가능성을 희박하게 만들고 있으며 현재의 농촌 노동력의 대다수인 노·장년층 또한 머지않은 미래에 노동력을 상실하게 되고 결국 우리는 최소한의 먹거리 조차도 생산할 인력을 확보하지 못하는 위기 상황이 닥쳐올 것으로 기대된다. 특히 지난 세기 말부터 시작된 WTO 체제로 국제 무역 체계의 변환은 값싼 농산물의 수입 자유화, 쌀 시장 개방 등 우리 농업의 존재 자체를 위협하는 상황에 이르게 되었다. 일부 경제학자들의 단순 비교우위 논리로 쉽게 농산물을 수입해다 먹으면 되지 않겠는가고 접근하는 시각은 우리의 기본 먹거리를 남의 나라에 의존하게 됨으로써 해당 국가들에 종속적인 관계로 전락할 수도 있는 우를 범할 수 있다. 이러한 상황은 비단 우리에게만 국한된 문제는 아니고 전세계적으로 산업 및 경제 발전을 중시하는 나라에서 공통적인 문제이며 이것은 향후 인류의 생존 자체를 위협할 수 있는 상황이 될 수 있기 때문에 충분한 대비가 이루어져야 한다.

사실 농업은 여러 산업 분야중 대표적인 3D 분야로 분류할 수 있다. 열악한 작업 환경에서 작업을 수행할 때 대두되는 작업자의 건강 및 사고 위험 문제, 농업생산시 필연적으로 수반되는 환경 오염 문제, 위험하고 고된 작업임에도 충분치 못한 경제적 이익 실현 등의 구조적인 문제를 갖고 있는 분야이다. 특히 앞에서 언급한 바와 같이 농업 노동력의 수적 감소와 질적 저하는 개발된 농업기계마저도 그 운용에 어려움을 겪게 되고 또한 위험한 작업을 기피하는 추세는 농업생산성의 저하로 이어져 인류의 식량 확보가 어려워지는 상황이 도래할 수 있다. 따라서, 국내외 많은 농업기계공학자들은 농업 생산 시스템을 공장에서 공산품을 생산하는 방식으로 전환하기 위하여 새로운 개념의 농업 생산 시스템의 연

구, 개발 노력을 경주하기 시작하였다. 농업용 로봇 개발에 관한 시도는 이러한 노력의 대표적 사례중 하나로서, 과일 수확 로봇에 관한 많은 연구 개발이 수행되고 있으며 경운 정지용 트랙터, 벼, 옥수수 등 곡물 수확기, 방제/시비기 등 기존 농작업기들의 로봇화를 위한 무인주행 기술 및 작업 자동화 기술 개발에 관한 연구도 활발히 진행되고 이는 실정이나, 여타 산업 분야와 달리 특히 노지에서의 농작업은 매우 제한적이고 균일하지 못한 작업환경에서 이루어지기 때문에 무인화 기술을 확립하는 데 어려움이 있다.

강원대학교 연구팀은 1996년부터 과수방제기의 무인화를 위한 연구를 수행한 바 있다. 초기의 대상 과수원은 포도원으로서 일정 간격으로 지주대를 세우고 천정에 줄을 설치한 후 그 위에 포도를 재배하는 천정 재배 방식이었다. 주행경로를 결정하기 위한 센서로서 초음파센서 두 개를 전방 좌우측을 비스듬히 지향하도록 설치한 후 과수방제기에서 좌우측 포도나무 지주대까지의 거리를 측정하여 과수방제기의 예상 주행경로에서의 자세(측면 옵셑)를 구하여 측면 옵셑을 0으로 만드는 방향으로 조향을 하게 하였었다. 이 시스템은 저비용 유도 시스템의 실현 가능성을 보여 주었지만 잎이나 가지에서의 초음파의 불규칙적인 반사나 진동에 따른 잡음 등으로 오작동하는 경우가 있었으며 지주대를 가급적 균일한 간격으로 배치를 해야하고 가지정리와 같은 준비 과정이 필요하였다. 근래에 들어 우리나라 과수농가들은 거센 바람이나 과일의 무게 때문에 나뭇가지가 꺾여지는 것을 방지하기 위하여 그린하우스 구조물처럼 U-자 모양의 구조물을 과수열을 따라 두 나무 사이에 거꾸로 세우고 나뭇가지를 이 구조물에 묶어주는 재배 방법을 택하고 있다(그림 1.1.1). 따라서, 과수원은 터널과 같은 형상을 띄게되며 그러한 밀폐된 공간에서 1년에 10여 차례 과수방제기로 방제작업을 한다는 것은 운전자가 농약에 노출될 위험성이 매우 높아지는 것을 의미한다. 이러한 점을 인식한 본 연구팀은 초음파와 같은 비접촉식 방식 센서의 불확실성을 제거하기 위하여 새로운 개념의 기계적 방식 센서를 고안하게 되었다. 이것은 기존의 과수원 구조물에 오버헤드 방식의 가이드레일을 설치하고, 가이드레일 상에서 이동가능한 트롤리와 과수방제기를 케이블로 연결하여 과수방제기가 예상주행선인 가이드레일의 지표면 투사선을 벗어나면 케이블이 특정 각도를 이루게 되므로 이 각도를 0으로 만드는 방향으로 조향 클러치를 조작하면 무인 주행이 가능해진다는 원리이다(Shin *et al.*; 2002). 그러나, 실제 포장에 나가서 작업할 때는 지면의 불규칙으로 인한 각도 측정에서의 오차, 제어 시스템에 인가되는 노이지 등으로 인

하여 제어 시스템의 안정성이 크게 떨어지게 된다.

따라서, 기본적으로 과수방제기의 측면 옵셑에 의해 야기되는 각도 측정 센서뿐 아니라 실제 과수원에 적용 가능한 과수방제기가 되기 위해서는 수평센서를 이용하여 측면 옵셑에 의한 각도를 보정해 주어야 하며 무인주행 뿐 아니라 무인작업을 해야하므로 각종 조작기의 자동화, 약액살포부의 자동화, 비상 정지 기능 등 수 개의 센서와 조작기가 추가되어야 하는 바, 마이크로프로세서 하나로 모든 시스템을 중앙 제어하기에는 한계가 있고 특히 배선이 복잡해져서 실용성이 떨어진다. 그래서, 무인주행 및 무인작업을 효율적으로 수행하기 위하여 분산제어가 요구되며 다수의 마이크로프로세서를 CAN 버스로 연결하면 이와 같은 목적을 달성할 수 있게된다. CAN 버스 시스템은 다른 제어방식과 달리 두 줄의 전선만 사용하므로 배선이 간편하고 확장성이 용이할뿐더러 저가이고, 고속 통신이 가능하며 빠른 응답특성 등의 장점(Farsi et al., 1999)으로 인해 경제적이고 실용적인 제어 시스템을 구축할 수 있을 것으로 판단되었다.

본 제어기술을 과수방제기의 제어시스템에 적용하고 오버헤드가이던스 장치 및 설치에 관한 표준화 등이 제시되면 저가이고 안정성이 높은 노지 과수원 무인 방제 시스템의 개발이 가능하게 되어 향후 우리나라 과수산업의 생산성 향상에 이바지할 수 있을 것으로 판단된다. 또한 CAN 버스 기술은 선진국의 동향을 볼 때 오래지 않아 우리나라에서도 향후 트랙터 및 그 부속 작업기, 콤바인 등 여타 농업기계에 적용될 것으로 예상되는 바 본 연구의 필요성이 제기되었다.

한편, 연구의 필요성을 기술적 및 사회 경제적 측면에서 살펴보면 다음과 같이 요약될 수 있다.

1) 기술적 측면

근래들어 자동차 산업에서 사용되던 Controller Area Network(CAN) 기반 제어 방식이 기계, 자동화시장에 쓰이기 시작되었고 또한 미국에서는 최근 생산되는 농용트랙터에 CAN 버스 시스템이 장착되어 시판되고 있으며 부속작업기들의 제어를 위하여 CAN 버스에 연결할 수 있도록 표준화 작업(ISO11783)이 진행되고 있다. 지속가능하고 친환경적인 정밀농업 기술을 확보하기 위하여 무인주행에 적용될 다양한 센서 및 조작기(actuator)의 개발이 요구됨. 여러 장치를 사용하고 또 필요에 따라 장비를 추후에 부가적으로 설치해야할 경우 제어시스템이 복잡해 지고 또 유연성이 떨어지게되는데, CAN 버스 시스템은 분산제어를 가능하게 해 줌으로써 언제든지 부가장비를 쉽게 설치할 수 있고 제어 성능을 획기적으로

향상시킬 수 있다.

또한, 농업 노동력의 절대적 감소 추세에 따라 멀지 않은 미래에 농작업의 무인화가 요구될 것으로 예상되는 바, 무인 농작업기 개발을 위한 기초 연구가 요구고 있으며, 특히 제한적이고 또한 균일하지 못한 작업환경에 적용될 로외차량 및 장치(Off-Road Equipment) 들의 로봇화 기술 개발중 주행부 개발이 우선적으로 이루어져야 할 필요가 제기된다.

근래들어 과수 재배양식이 밀식재배, 포도원의 하우스화, 왜성 배원의 가지 지지프레임 설치 등의 방식으로 변화하는 추세이며, 이에 따라 작업공간이 협소해고 밀폐되므로 방제작업시 작업자가 농약에 노출될 위험이 대단히 높아짐에 따라 무인방제 작업기의 개발이 필요하다(그림 1.1.1 참조).

2) 경제·산업적 측면

안정적(reliable)이고 경제적(cost-effective)인 무인주행 기술의 개발을 통하여 농업 산업의 생산성을 더욱 향상시킬 필요가 있으며, 실용가능한 첨단 기술이 적용된 농업기계로 해외시장을 확보하여 국내의 농업기계산업의 활로를 모색할 필요가 있는 시점이다.

3) 사회·문화적 측면

급격한 고령화사회로 접어들며 기존의 농업생산 종사자가 줄어들고 새로이 유입되는 농업 노동력이 없는 현실에서, 첨단 기계 및 시설의 운용 등으로 농업을 3D 업종이 아닌 비즈니스급의 2차산업으로 전환되어야 할 필요가 있으며, 우리 민족의 주곡인 쌀을 비롯하여 고부가가치의 채소, 원예 산업 등은 절대적으로 포기할 수 없는 산업이며, 강대국의 식량무기화에 대처해야 하는 측면에서 상기와 같은 연구 개발이 필요하다.

그림 1. 대표적인 포도원 및 배원의 전경

제 2 절 연구개발 목표와 내용

1. 연구개발 목표

○ 신뢰성이 높고 저가인 CAN 버스 시스템을 제어장치에 적용함으로써 관련 시스템의 설계 및 응용 기술을 확보한다.
○ 위 제어장치를 과수 방제기에 적용하여 포장에 설치된 오버헤드가이던스에 의한 무인주행을 하며 방제작업 목표를 달성할 수 있는 무인 과수방제기를 개발한다.

2. 연구개발 내용

가. 궤도형 주행장치의 무인/자동 조작기 개발

공시 기종인 궤도형 주행장치가 장착된 과수 방제기에서 작업자가 조작하는 주클러치/브레이크 레버 및 좌우측 조향 클러치 레버를 무인으로 조작할 수 있도록 유압 조작기를 선정 장착한다. 각 조작기 레버의 특성을 분석하여 적합한 유압 조작기를 선정하고 조작 성능을 평가한다. 유압조작기의 작동 제어 알고리즘을 개발한다.

작업시 및 성능 실험시 항상 일정한 주행속도를 유지, 확보하기 위하여 드로틀레버를 조작할 수 있는 전동 실린더 조작기를 선정 장착한 후 제어알고리즘을 개발하며, 궤도의 구동륜 스프로켓의 회전속도를 측정하여 과수 방제기의 주행속도를 측정하는 주행속도 측정장치를 개발한다.

약액살포부 무인조작기를 개발한다.

나. 주행경로 결정 센서 개발

주행경로 결정 센서는 과수 방제기의 무인주행을 위한 필수 요소로서 주행 기준선에 대하여 차량의 자세, 즉 차량이 주행 기준선으로부터 얼마나 벗어나 있는 지를 나타내는 측면 옵셑과 차량이 현재 지향하고 있는 방향각을 정확히 측정할 수 있어야 한다. 연구 대상 작업 차량 및 동 연구가 적용될 과수원에서는 차량의 위치를 절대좌표계로 나타낼 수 있는 GPS의 사용이 제한되므로 하나의 주행 기준선을 설정하여 그에 대한 상대좌표로서 차량의 위치를 측정할 수 있는

방법이 고안되었다. 상대좌표를 측정하기 위하여 기계시각, 초음파 센서, 레이져 센서 등의 비접촉 방식이 적용될 수 있으나 안정성 및 실용화 측면에서 접촉식 방식의 센서 시스템이 개발된다.

이것은 지상부에 가이던스 레일을 설치하고 차량으로 하여금 가이던스 레일을 추종하며 조향하는 방식으로, 1) 가이던스 레일 상에서 차량과 연결된 케이블을 이동시킬 수 있는 트롤리가 개발되며, 2) 차량의 자세에 따라 달라지는 트롤리와 차량을 연결하는 케이블의 기울어진 각도를 측정할 수 있는 장치가 개발되고, 3) 노면의 높낮이 및 차량의 자세에 따라 케이블의 길이가 변하게 되므로 적정한 장력으로 케이블이 풀어지고 당겨질 수 있는 케이블 권선 장치가 개발되고, 4) 케이블의 현재 길이를 측정하는 장치가 개발되며, 또한 5) 지면의 경사에 따라 케이블의 기울어진 각도로부터 측정된 측면 옵셑이 변하게 되므로 이를 보정하기 위한 경사 센서가 개발되고, 6) 측정 시스템 전체를 총합하여 최종적으로 차량의 측면 옵셑과 방향각을 구할 수 있는 측정 알고리즘을 개발한다. 가이던스 레일상을 움직이는 트롤리는 차량이 이동을 시작해야만 그에 따라 이동하게 되므로 항상 차량의 현재 위치보다 전 단계에서의 위치에 대한 자세를 측정하게 되므로 발생되는 오차를 최소화하기 위하여, 7) 트롤리의 위치 및 이동속도를 차량의 위치 및 주행속도에 동기 시킬 수 있는 제어기가 개발된다.

가이던스 레일의 경제성 및 설치 용이성, 유지 보수의 용이성 등은 본 연구에서 개발할 주행경로 결정 센서의 기술적 타당성 및 실용성에 영향을 미치게 된다. 따라서, 선회 구간을 포함하여 가이던스 레일의 구조 및 설치 등에 관한 연구를 진행한다.

다. CAN 버스 시스템의 구조 분석 및 설계

CAN 버스는 전장 장치에서 배선의 간편성 및 유지보수의 용이성, 분산제어 방식의 적용 등으로 자동차 분야에서 활발히 적용되고 있으며 미국 및 유럽의 트랙터, 콤바인 등 대형 기종 중심으로 이미 적용이 되고 있는 기술이다. 향후 국내의 농업기계에도 적용 가능할 수 있는 기술력을 확보하고 그 적용 가능성을 규명하기 위하여 CAN 버스 시스템의 구조 분석에 대한 규명 연구가 수행되며 이를 바탕으로 CAN 버스 시스템을 설계한다.

1) CAN의 작동 원리를 규명하고, 2) 다양하게 개발되어 있는 CAN 시스템을 구성할 수 있는 마이크로프로세서 중에서 적용성이 우수한 것을 선정하고, 3)

CAN 버스 시스템을 구성할 수 있는 단위 ECU(Electronic control unit)를 설계하여 PCB로 제작하고, 4) 이를 구동할 수 있도록 라이브러리를 포함한 기본 펌웨어(Firmware)를 개발한다. 5) 여러 개의 ECU들을 CAN 버스로 구성하여 통신 성능을 평가한다.

라. 센서/조작기와 마이크로프로세서의 인터페이스 개발

CAN 버스 시스템은 제어 시스템의 분산제어를 가능하게 해 주므로 본 연구에서는 CAN 버스 시스템의 적용성을 검토하기 위하여 가급적 다양한 센서와 조작기 등을 설치 운용한다. 주행경로 결정 센서에서의 각도 측정 및 케이블 길이 측정 센서, 경사 센서, 장애물 감지를 위한 초음파 및 광센서, 차량의 주행속도 측정 센서, 엔진 속도 측정 센서 등과 조향 클러치 및 메인 클러치/브레이크 조작기, 드로틀레버 조작기 등을 마이크로프로세서와 연결하기 위한 하드웨어로서 인터페이스를 개발하고 각각의 구동 소프트웨어인 펌웨어를 개발한다.

원격제어가 가능하도록 무선조종기 시스템을 개발한다.

마. 가상터미널(Virtual Terminal)로서 ARM보드의 개발

전체 제어 시스템의 작동 상황을 쉽게 파악할 수 있도록 터치스크린/모니터가 장착된 ARM9.0 단일보드컴퓨터를 구동하기 위한 인터페이스 및 구동 소프트웨어를 개발한다.

바. 무인주행 및 방제작업제어 알고리즘 개발

공시 연구 대상 기종인 궤도형 주행장치가 장착된 과수방제기의 무인 주행을 위한 제어 알고리즘을 개발한다. 1) 좌우측 조향 클러치의 작동 특성을 규명하여 과수방제기의 조향각과 클러치 작동 시간과의 관계식을 도출한다. 2) 컴퓨터 시뮬레이션을 위하여 궤도형 차량의 조향 제어 모델식을 개발하며 그에 따른 컴퓨터 시뮬레이션 프로그램을 개발한다. 3) 조향 제어는 차량의 현재 자세, 즉 측면 옵셑과 방향각에 따라 필요한 만큼의 조향이 이루어질 수 있도록 해당 조향클러치의 작동 시간을 제어할 수 있도록 제어 프로그램을 개발한다. 4) 전문가의 지식 및 경험에 근거하여 현재의 차량 자세 정보로부터 어느 방향으로 몇 도의 조향을 해야하는 지를 파악하여 규칙기반을 만들어 측면 옵셑 및 방향각을 입력으로하고 클러치 작동시간을 출력으로하는 퍼지 제어 시스템을 개발한다. 5) 무인

주행 성능 평가를 통하여 퍼지 제어 모델을 수정 보완한다.

약액 살포기의 on/off 제어 여부를 판단하는 센서는 차량의 앞쪽에 위치하고 약액살포 노즐은 차량 후방에 위치하므로 과수열의 끝과 시작을 감지하더라도 실제 약액살포 노즐의 제어는 차량이 센서 위치와 노즐 위치의 차이만큼 더 진행한 후 제어가 이루어져야 한다. 또한 약액 탱크내 약의 남은 양을 수위로써 판단하여 한 과수열의 중간 지점에서 모든 약액이 소진되어 작업을 중지하는 경우가 발생하지 않도록 해야 한다. 이와 같은 목적을 달성하기 위하여 방제작업 제어 알고리즘을 개발한다.

사. 약액살포제어기 개발

약액살포 제어에 사용할 약액탱크 수위 센서, 과수열 끝 및 시작을 감지할 수 있는 센서와 약액살포용 노즐을 제어하기 위한 솔레노이드 밸브 조작기 등을 작동할 수 있도록 인터페이스 회로와 구동 소프트웨어를 개발한다.

아. 성능 평가

개발된 제어 시스템의 성능을 평가하기 위하여 실험실내에 줄 사이의 간격이 3 m를 유지하도록 8 m의 직선 구간에 해당하는 오버헤드 가이던스 레일 두 줄을 설치하고, 곡률 반경 1 m짜리 두 개와 직선 1 m짜리 하나로 선회구간을 구성하여, 1) 직선 구간에서 차량의 초기 자세 조건 및 주행속도를 달리하여 주행 기준선과 실제 주행궤적의 차이에 대한 편차의 RMS값으로 무인 주행 제어 성능을 평가한다. 2) 한 줄이 끝나고 인접한 줄로 이동하기 위한 선회구간에서 무인 주행 성능을 평가한다. 실내 실험시 정확한 궤적을 측정하기 위한 궤적 표시 장치를 개발한다.

제어 시스템의 실제 과수원 적용 가능성을 규명하기 위하여 강원대학교 부속 농장 과수원(과수열의 길이 40 m, 과수열 간의 폭 4 m)에 두 줄의 직선 구간과 선회 구간을 설치한 후, 노면의 경사도가 균일하지 않은 자연 토양 조건에서 무인 주행 성능을 평가한다.

방제작업 성능평가로서 과수열의 시작과 끝에서 약액살포 노즐 제어용 솔레노이드밸브의 작동 상황을 평가한다.

제 2 장 국내외 기술개발 현황

제 1 절 기술개발 현황

1. CAN 버스

CAN버스 기술은 1983년 독일에서 고안되어 자동차산업에 주로 사용되었으나 근래에 들어 기계, 자동화 장치 등 분산제어가 필요한 곳에 많이 이용되고 있으며 농업기계 분야에서는 미국 (Reid, 2002; Zhang, 2002) 및 유럽 등에서 대형 트랙터 및 콤바인의 전장 시스템 및 작업기 제어 시스템에 적용되고 있다. 특히 버스 및 트럭에 관한 CAN버스 규약인 SAE의 J1949와 함께 농업용 트랙터의 전장 제어 시스템에서의 CAN 통신 규약으로 ISO11783이 제정되었으며 이와 관련하여 계속적인 연구 및 수정 작업이 이루어지고 있다.

Wei et al. (2001)는 광학센서로 잡초를 감지하여 잡초에만 농약을 살포할 수 있도록 두 개의 광센서, 스프레이 노즐 액츄에이터와 GPS로 구성한 시스템을 CAN 기반에서 제어한 바 있음.

- 국내에서는 웹 기반의 온실 원격 제어 시스템의 개발 연구에서 온실내의 각종 환경 변수의 모니터링 및 모터 제어를 위하여 시스템의 확장성이 용이한 분산형 제어시스템으로서 CAN 버스 시스템을 적용한 바 있음.(Heo et al., 2002)

2.. 과수원 방제기의 무인주행

국내에서 과수원 방제기의 무인화에 관련된 연구가 활발히 진행되었으나 아직 실용화에 이르지는 못한 실정이다. 관련 연구들은 주로 무인 주행에 관한 것으로 다양한 센서들이 적용되었다. 대표적으로 지중 유도 케이블에 의한 방식(Jang et al., 1995), 기계시각을 이용하는 방식(Cho and Ki, 1999; Ki et al., 1996; Jang et al., 1998), 기계시각과 DGPS를 이용하는 방식(Lee et al., 1998) 등의 연구가 수행된 바 있다.

국외의 경우 미국의 Deere and Company 사는 완전 무인작업이 가능한 과수원 방제기를 개발하여 발표한 바 있음. 그러나 미국의 과수원은 조간 간격이 6m

정도로 충분히 넓어 DGPS 운용에 지장이 없고 또한 정밀급 기자재를 사용할 수 있어서 이 기술 개발 및 적용이 가능했다고 판단된다.(Resources, 2001)

한편, 강원대학교 연구팀(Kim et al., 1999)은 초음파센서로 과수 또는 과수 지지대 등을 감지하여 소형 과수방제기의 무인주행을 시도한 바 있으나, 과수의 불규칙한 가지 등이 있거나 균일한 굵기의 과수가 일정한 간격으로 배치되어 있어야 하는 등의 제한 조건이 있으며 아직 실용화하기에는 안정성이 부족한 상태이다. 따라서, 포도원, 배원 또는 감귤원 같이 밀폐된 공간에서는 비접촉식 센서의 사용보다는 기계적 방식의 센서가 보다 현실적으로 실용화 가능한 기술이라고 판단된다.

3. 갠트리시스템의 무인주행 기술

Tillet and Nybrant(1990)는 지중 유도케이블 방식을 이용하여 갠트리 시스템(폭 9m, 궤도형 주행장치)을 ±60mm 의 정확도로 무인주행시킨 바 있으며, Siemens and Coates(2000)는 케이블 유도방식의 농작업기가 부착된 갠트리 시스템(폭 175m, 5개의 차륜형 주행타워)의 무인주행을 위하여 지상 0.7m에 가이던스 케이블을 설치하고 steering queel이라는 기계식 센서를 개발하여 갠트리 시스템의 앞 과 뒷 부분에서의 가이던스 케이블까지의 거리를 측정하여 그 차의 크기에 따라 조향을 하는 방식으로 ±15cm 의 정확도로 제어할 수 있었다고 보고하였다. 또한 일정한 거리를 이동한 후 정지하여 케이블에 부착된 농작업기로 작업을 수행해야하므로 갠트리 시스템이 포장에 설치된 인덱싱 장치를 감지하여 목표 정지위치의 ±6.1cm 이내에서 제어 가능하였다고 보고하였다.

4. 트랙터/콤바인 등의 무인주행 기술

미래의 농업생산을 무인화하기 위한 무인주행기술 연구가 국내외적으로 활발하게 이루어지고 있는데 무인 주행을 할 때 주행 기준선을 설정하는 방식에 따라서 다음의 두 가지 경우가 대표적이다(Reid et al., 2000; Torii, 2000).

1) 포장내 농업기계의 절대좌표를 찾아서 무인주행하는 경우

대 부분의 경우 GPS를 사용하여 농업기계의 현재 위치를 구하여 목표 지점으로 이동할 수 있게 한다. 미국의 John Deere 사는 소형 운반차 Gator에 정밀

DGPS를 장착하여 통제용 컴퓨터의 지역 지도에 주행경로를 제시해 주면 또는 해당 작업구간을 지정해 놓으면 줄과 줄간의 중복(overlap)이나 건너뜀(skip)없이 작업할 수 있는 기술을 개발하였다(Reid, 2002). 미국의 University of Illinois 연구팀은 미리 작성해 놓은 지도에 따라 차고지부터 포장까지의 이동은 물론 농작업을 무인으로 수행하는 기술을 연구하였음(Zhang and Guo., 2002). 또한 이들은 저가의 GPS 와 INS 센서를 이용하여 고성능의 DGPS를 대체할 수 있는 시스템 개발에 관한 연구를 수행한 바 있음.

2) 포장내 농업기계에서 어떤 기준에 대한 상대위치를 찾아서 무인주행하는 경우

기계시각을 이용하여 작물열을 감지하여 농업기계를 무인주행하는 경우가 있었으며(Tillet et al., 2002; Marchant, 1996), Toda et al.(1999)은 초음파센서로 작물열을 감지하여 모바일 로봇의 무인주행 방법을 연구하여 가상적인 필드조건에서 위치오차 33.6mm, 방향오차 3.2°내에서 무인주행이 가능하다고 하였다.

무인주행기술의 확보를 위한 노력이 많이 경주되고 있으나, 대부분의 경우 정밀도가 높은 고가의 DGPS시스템을 요구하거나 기계시각 등 포장에서 사용할 때의 안정성 등으로 인하여 아직 실용화의 단계에는 이르지 못하였으므로, 농작업 환경과 같은 불균일하고 열악한 조건에서는 앞서 인용한 Siemens and Coates(2000)의 연구에서와 같이 첨단 고가의 비접촉 방식의 센서보다는 기계적 센서를 사용할 때 오히려 안정한(reliable and robust) 시스템이 될 수 있으며 경제성도 있으므로 실용화 가능성이 매우 높다고 판단된다.

제 2 절 앞으로 전망

본 연구는 아직 국내에서 많이 적용되고 있지 않은 CAN 버스 시스템의 응용에 관한 선도연구가 될 것으로 전망되며, 향후 한국 농기계산업은 고부가가치의 제품으로 미국, 유럽 등의 틈새 시장으로의 진출을 모색해야 할 것으로 예상되는 바 첨단 기술력의 확보가 요구될 것으로 보여진다. 또한 농업 생산에 종사할 수 있는 인구가 급격히 줄어들고 또한 위험성이 높은 방제작업 등의 기피 현상에 따라 미래의 농작업중 많은 부분이 농용로봇에 의하여 이루어질 것으로 예상된다. 특히 일본에서 진행되고 있는 수도작에서의 갠트리시스템 적용 연구에 비추어 볼 때 향후 전작 분야에도 갠트리시스템의 도입이 예상되며, 사회적으로 농업생산에 따른 환경 오염 방지 및 유기농업의 요구가 더욱 확대될 것으로 예상되

는 바, 지속가능하고 친환경 농업이 가능한 정밀농업 관련 기술 개발이 더욱 거세게 요구될 것으로 판단된다.

제 3 장　연구개발수행 내용 및 결과

제 1 절 궤도형 주행장치의 무인/자동 조작기 개발

1. 조향장치 조작기

가. 스피드 스프레이어의 주행을 위한 동력전달 체계

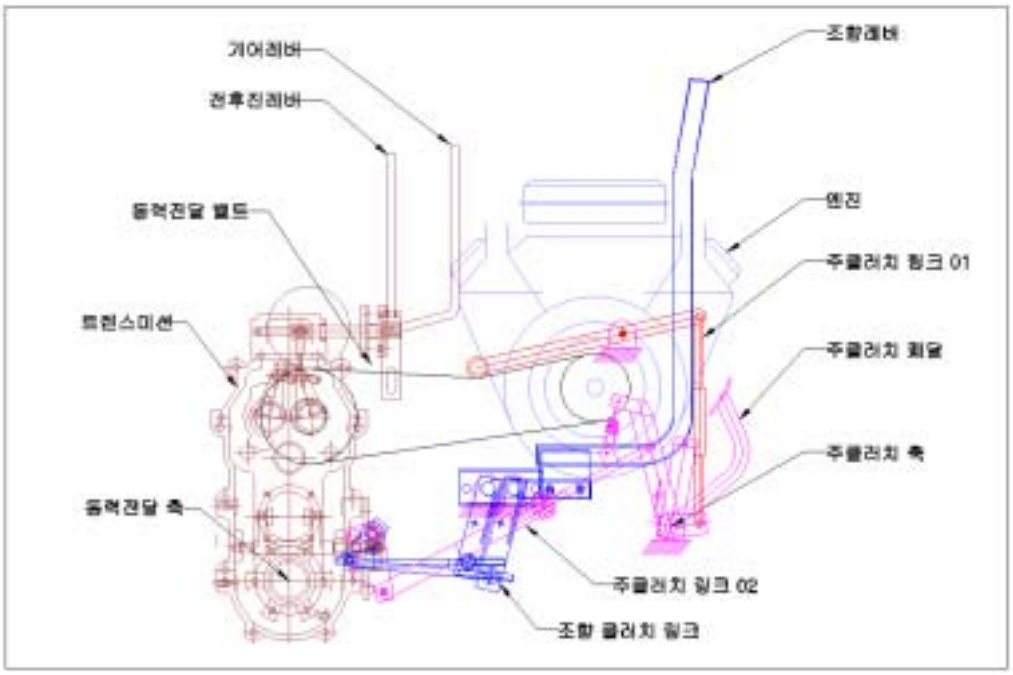


그림 3.1.1 동력전달체계 모식도

그림 3.1.1은 스피드 스프레이어(SS기)의 전체적 동력전달 체계를 한눈에 볼 수 있도록 엔진, 트렌스미션, 주클러치 폐달, 조향레버를 간단하게 나타낸 그림이다.

엔진으로부터 오는 동력은 밸트에 의해 트렌스미션으로 전해지고 전해진 동력은 전후진 레버와 기어레버의 조합에 의해 감속되어 동력전달 축에 이어지게

된다. 엔진에서 트렌스미션을 연결해주는 밸트는 주 클러치 링크 01에 의해 밸트를 누르거나 느슨하게 할 수 있으며 그림과 같이 눌러준 상태로 텐션이 발생하면 동력이 전달되는 상황이고 주클러치 링크 01이 밸트로부터 떨어져 밸트가 느슨하게 되면 엔진에서 트렌스미션으로 가는 동력이 끊어지게 된다. 주클러치 폐달은 이렇게 트랜스미션으로 가는 동력 전체를 끊어줄 뿐만 아니라 주클러치 링크 02를 작동시켜 트렌스미션 내부에서 동력전달축으로 이어지는 기어를 분리해 줌으로써 역시 동력을 끊어주게 된다. 이와 같이 동력을 두 번 끊어주는 것은 밸트를 끊어주는 순간 잔여시간에 의해 약하게 트렌스미션으로 전해지는 동력을 동력전달축으로 전해지지 못하게 할 뿐만 아니라 내리막길에서 좌우 차륜의 브레이크를 잡아 차량이 흘러내려가는 것을 방지해 주기 위함이다.

조향레버는 동력이 트렌스미션에 전해져서 동력전달 축까지 전해진 상황에서 좌우동력체로 각각 이어지는 기어를 떨어뜨리면서 동력을 끊어주고 이어줌을 반복하며 조향을 할 수 있도록 하는 장치이다. 다시말해 전해지는 동력을 강제로 끊어줌으로써 조향을 할 수 있는 Skid 조향방식을 채택하였다.

1) 주클러치의 작동 특성

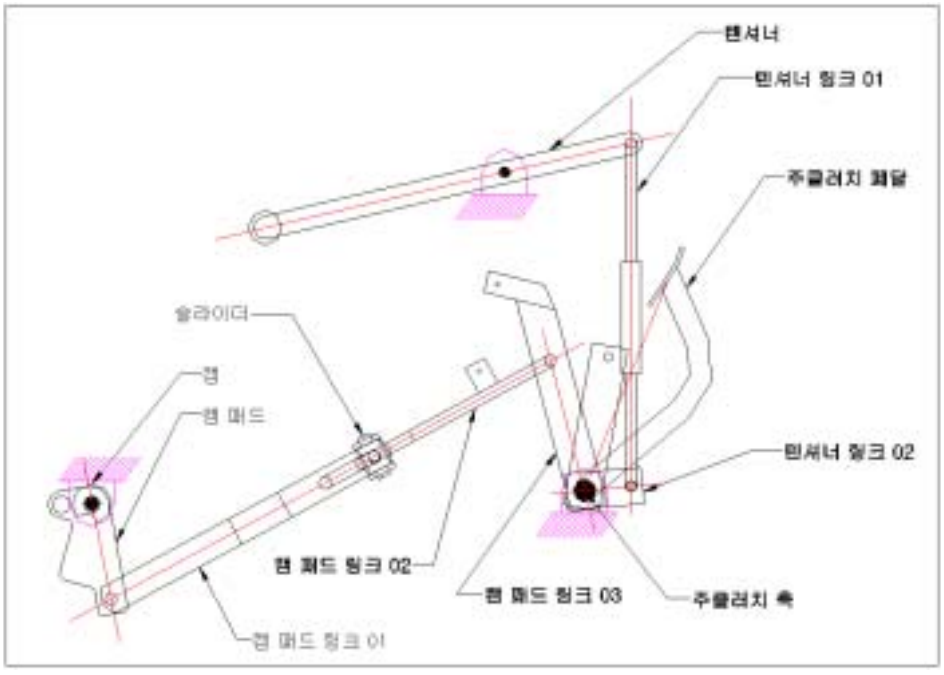


그림 3.1.2 주클러치 모식도

그림 3.1.2는 동력전달체계에서 주클러치에 해당되는 부분만을 한눈에 알아보기 쉽도록 따로 모식화 한 그림이다.

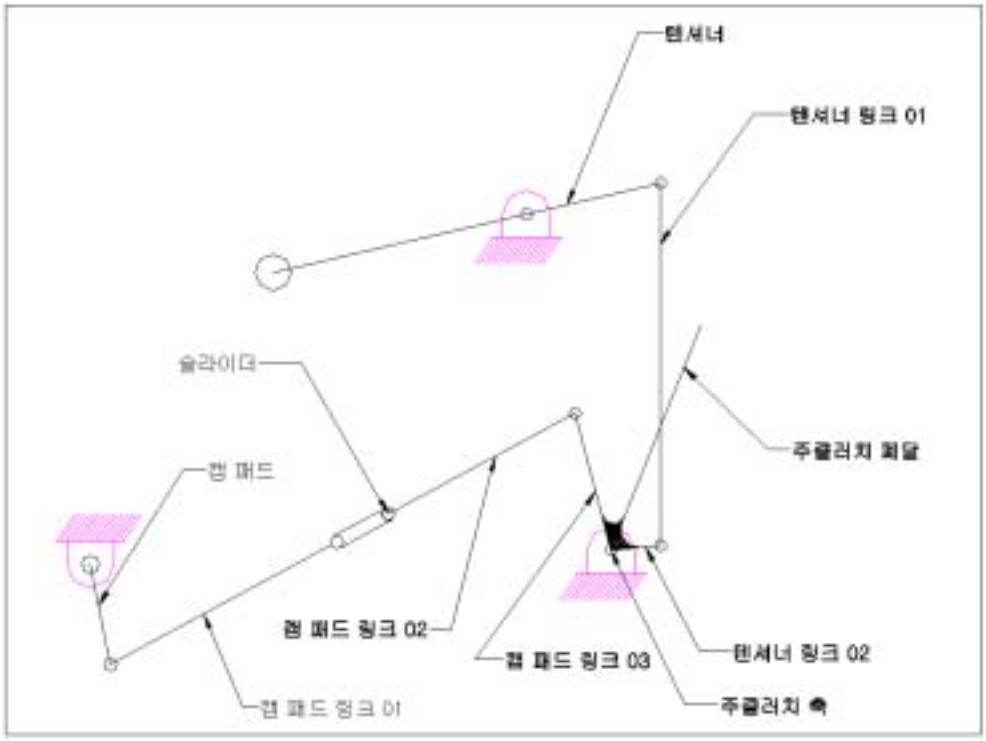


그림 3.1.3 주클러치 선형도

그림 3.1.3에서 주클러치 페달, 텐셔너 링크 02, 캠 패드 링크 03은 주클러치 축 한개에 모두 용접되어 있어 일체형으로 주클러치 페달을 밟으면 모두 같이 움직이도록 설계되어있다.

슬라이드라 하는것은 좌우 조향장치를 당길 때 캠 패드를 당기게 되고 그때 캠 패드 링크 02로 전달되는 당김을 끊어줌으로서 조향레버를 당겼을 때 클러치와 차별적으로 좌우의 동력만을 제어할 수 있도록 하는 장치이다. 다시 말해 주클러치 페달을 밟았을 때는 슬라이드는 그림 3.1.3에서와 같은 형태로 일체형이 되어 움직이고 조향레버를 당길 때에만 슬라이더가 작용하여 조향하려는 쪽만의

동력을 끊어주고 브레이크를 잡아주게 된다.
주클러치 페달, 텐셔너 링크 02, 캠패드 링크 03이 일체형이므로 주클러치 페달을 밟아주게 되면 텐셔너 링크 02와 연결된 텐셔너 링크01과 텐셔너가 같이 움직임으로서 동력을 전달하는 벨트를 누르고 있던 텐셔너가 떨어져서 엔진으로부터 나와 트랜스미션으로 가는 동력을 끊어주게 된다.

주클러치의가 회전할 수 있는 최대 변위와 동력이 끊어질 때까지의 변위를 파악하고 이것을 컴퓨터 프로그램을 사용하여 캠 패드가 1deg/sec씩 돌아갈 때 주클러치의 변위(xdeg/sec)를 체크해 보고 이것을 기준으로 캠 패드가 1deg씩 돌아갈 때 주클러치 축이 몇deg씩 돌아가는지 표 3.1.1과 같이 알아보았다.

표 3.1.1 주클러치의 변위

캠 패드의 변위(deg)	주클러치 축의 변위(deg)	캠 패드의 변위(deg)	주클러치 축의 변위(deg)
0	0	10	6.178917913
1	0.635576324	11	6.76838392
2	1.267971117	12	7.350873623
3	1.897070145	13	7.925831809
4	2.522536725	14	8.49266731
5	3.144034777	15	9.050751592
6	3.761214311	16	9.599417316
7	4.373703876	17	10.13795686
8	4.981106337	18	10.66562078
9	5.582996238	19	11.18161629

위의 이론값들과 실제 측정에 의해 동력이 끊어지는 지점을 비교하여 정리하면 다음과 같다.

캠 패드가 최초 10deg에서 동력이 끊어지는 것을 확인하였고 제어를 위해 주클러치의 축을 0deg～10deg까지 제어해야 할 것을 확인하였다.

2) 조향클러치의 작동특성

가) 좌측 조향클러치

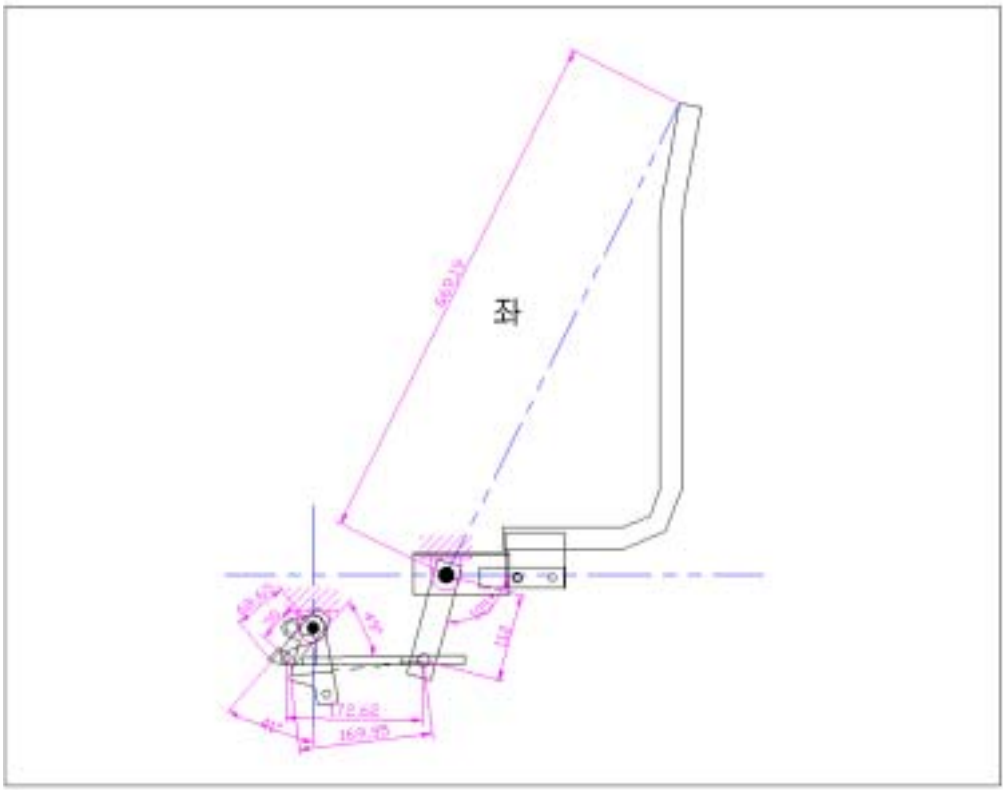


그림 3.1.4 좌측 조향클러치 전개도(치수)

그림 3.1.4는 한눈에 보기 쉽도록 좌측 조향클러치만을 따로 분리해서 간략하게 그림으로 나타낸 것이다.

검은 점으로 표기한 부분이 힌지점을 나타낸다. 우측상단의 힌지점은 레버의 힌지점이고, 좌측하단의 힌지점은 트렌스미션의 힌지점이다.

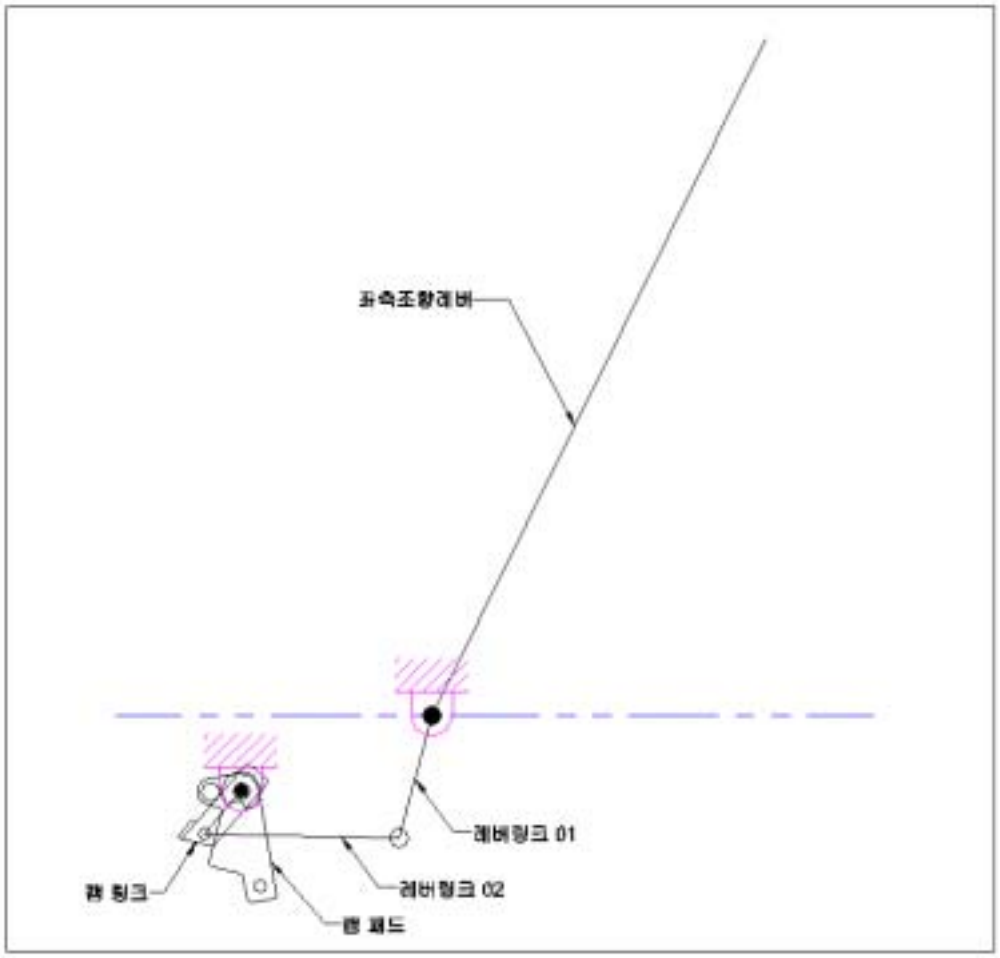


그림3.1.5 좌우측 조향클러치 선형도

캠 링크가 움직이기 시작하여 캠 패드에 닿기까지는 dead band 구간이므로 이 구간(26deg)에서는 동력을 끊어 주는데에는 아무런 영향을 끼치지 않는다. 캠 패드가 움직이기 시작하여 최초 10deg가 될 때 동력이 끊어지므로 dead band 26deg를 여기에 더하면 **캠 링크가 최초 36deg움직일 때 브레이크가 잡힘과 동시에 동력이 끊어지게 된다.**

따라서 위에서 한 방법과 마찬가지로 이것을 solidworks에서 링크를 구성하여 visualnastran을 통해 움직임을 파악하여 표 3.1.2로 나타내었다.

표 3.1.2 좌측 조향레버의 변위

캠 링크의 변위(deg)	좌측조향레버의 변위(deg)	캠 링크의 변위(deg)	좌측조향레버의 변위(deg)
1	0.420727435	19	6.479131902
2	0.833156456	20	6.726112967
3	1.237256745	21	6.963230147
4	1.632873314	22	7.19043408
5	2.019874595	23	7.40767731
6	2.398142951	24	7.614914058
7	2.767570053	25	7.812099978
8	3.128054574	26	7.999191927
9	3.479501004	27	8.17614773
10	3.821818983	28	8.342925951
11	4.154922897	29	8.499485664
12	4.478731574	30	8.645786229
13	4.793168037	31	8.781787065
14	5.098159281	32	8.907447441
15	5.393636047	33	9.022726253
16	5.679532604	34	9.127581819
17	5.955786524	35	9.221971673
18	6.222338458	36	9.305852364

표 3.1.2에서 보는 바와 같이 **캠 링크가 최초 36deg돌아갈 때 동력이 끊어지므로 그때까지의 조향레버는 9.3deg까지만 제어해 주면된다.**

나) 우측 조향레버

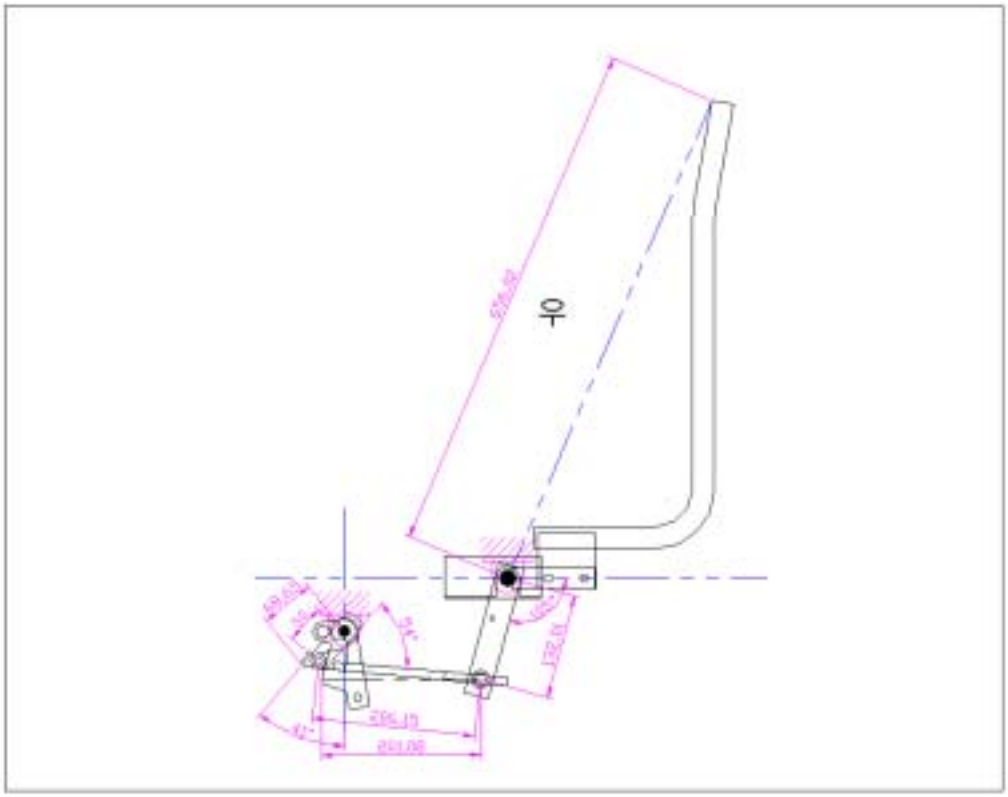


그림 3.1.6 우측 조향클러치의 전개도(치수)

그림 3.1.6은 그림 3.1.5와 같이 우측 조향클러치만을 따로 분리해서 간략하게 나타낸 그림이다. 검은점으로 표기한 부분이 역시 힌지점을 나타낸다.

치수로 알 수 있듯이 좌측과 우측의 힌지점의 위치가 약간의 차이가 남을 확인할 수 있다.

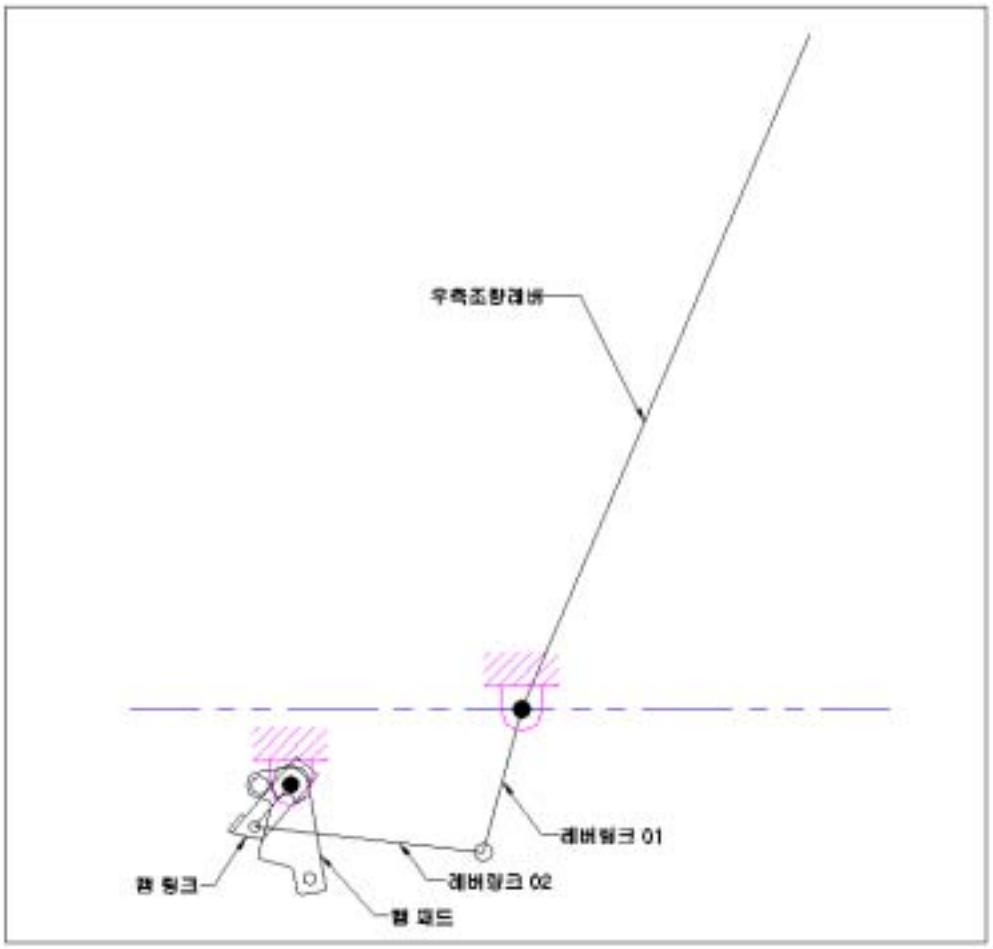


그림 3.1.7 우측 조향클러치의 선형도

우측 조향클러치도 좌측 조향클러치와 마찬가지로 캠 링크가 캠 패드에 닿기까지 dead band가 존재하므로 26deg까지는 아무런 영향을 끼치지 않고 26deg돌아간 후에 캠 패드와 닿기 시작하여 캠 패드를 10deg까지 돌려주게 되면 브레이크가 잡힘과 동시에 동력이 끊어지게 된다. 다시 말해 **좌측조향레버를 최초 36deg까지 당겼을 때 동력이 끊어지는 것과 마찬가지로 우측조향레버 역시 최초 36deg까지 당겼을 때 동력이 끊어지게 된다.**

따라서 같은 방법으로 solidworks에서 링크를 구성하여 visualnastran을 통해 움직임을 파악하여 표 3.1.3에 나타내었다.

표 3.1.3 우측 조향레버의 변위

캠 링크의 변위(deg)	우측조향레버의 변위(deg)	캠 링크의 변위(deg)	우측조향레버의 변위(deg)
1	0.352734636	19	5.298217282
2	0.697706756	20	5.490290652
3	1.034900314	21	5.673242887
4	1.364193405	22	5.847018626
5	1.68548241	23	6.011563224
6	1.998674023	24	6.166822595
7	2.303681359	25	6.312743069
8	2.600422028	26	6.449271234
9	2.888817146	27	6.576353787
10	3.168790798	28	6.693937391
11	3.440269706	29	6.801968524
12	3.703183011	30	6.900393337
13	3.957462088	31	6.989157512
14	4.203040387	32	7.068206125
15	4.439853281	33	7.137483505
16	4.66783792	34	7.196933109
17	4.886933079	35	7.246497384
18	5.097079007	36	7.286117646

표le 3.1.3에서 보는 바와 같이 캠 링크가 최초 36deg 돌아갈 때 동력이 끊어지므로 그때까지의 우측조향레버는 7.3deg까지만 제어해 주면된다.

좌측 조향레버는 9.3deg까지 제어를 하는 반면 우측 조향레버는 7.3deg만 제어해

도 되는 것은 위에서 말했듯이 좌우측 조향클러치 각각의 두개의 힌지점들이 그 위치가 서로 다르기 때문에 차이가 나는 것이다.

나. Clutch 및 조향레버 조작을 위한 유압실린더 선정

1) Clutch용 유압실린더

위에서 알아본 바와 같이 주클러치 축을 10deg 이상을 밀어줄 수 있는 행정과 주클러치 축에 걸리는 토크를 이겨낼 수 있는 힘을 가진 실린더를 선정하면 된다.

토크를 알아보기 위해 주클러치 축에 0.5m되는 임의의 bar를 연결하여 그 끝을 용수철저울을 사용하여 주클러치가 완전히 잡힐 때까지의 최대 힘을 측정한 결과 6.76kg_f임을 확인하였고 따라서 주클러치 축에 걸리는 최대 토크 T는 다음과 같다.

$$T = F * L = (6.76*9.8)*0.5 = 33.124\ Nm$$

실린더를 장착할 위치는 주클러치 축의 중심으로부터 0.2m되는 위치이므로 토크가 46.37Nm일 때 그 위치에서 밀어줘야 할 힘은

$$F * 0.2 = 33.124$$

이므로 **F = 165.62N** 이다.

따라서 165.62N 이상의 힘을 낼 수 있는 실린더를 선정하기로 하였다.

또한 실린더의 행정길이는 주클러치 축의 중심으로부터 0.2m되는 위치에서 10deg이상 밀어주어야 하므로 약 35mm이상만 되면 된다.

유압펌프에서 나오는 유체의 압력 P가 10kg/cm^2 일 때 내경이 20mm인 실린더를 사용한다면 그때 실린더가 낼 수 있는 힘은 다음과 같다.

$$F = P * A = 10kg/cm^2 * 10^2\pi = 31.4kg = 307.72N$$

따라서 **주클러치용 실린더는 내경이 20mm이고, 행정길이가 35mm이상이 되는 실린더를 선정하였다.**

2) 조향 클러치용 유압실린더

조향 클러치 축에 걸리는 토크는 주클러치 축에 걸리는 토크보다 작게 측정되었다.
행정의 길이는 조향 레버 축의 중심으로부터 실린더를 장착할 위치가 280mm되는 위치에서 36deg이상 밀어주어야 하므로 173mm이상만 되면 된다. 단, dead band가 존재하므로 사실상 173mm보다 작은 행정을 가진 실린더를 사용해도 무방하여 **주클러치용 실린더와 조향 클러치용 실린더를 모두 같은 크기와 같은 행정의 조건을 가진 것으로 선정 하였다.**

그림 3.1.8은 선정한 실린더의 치수를 나타낸 것이다.

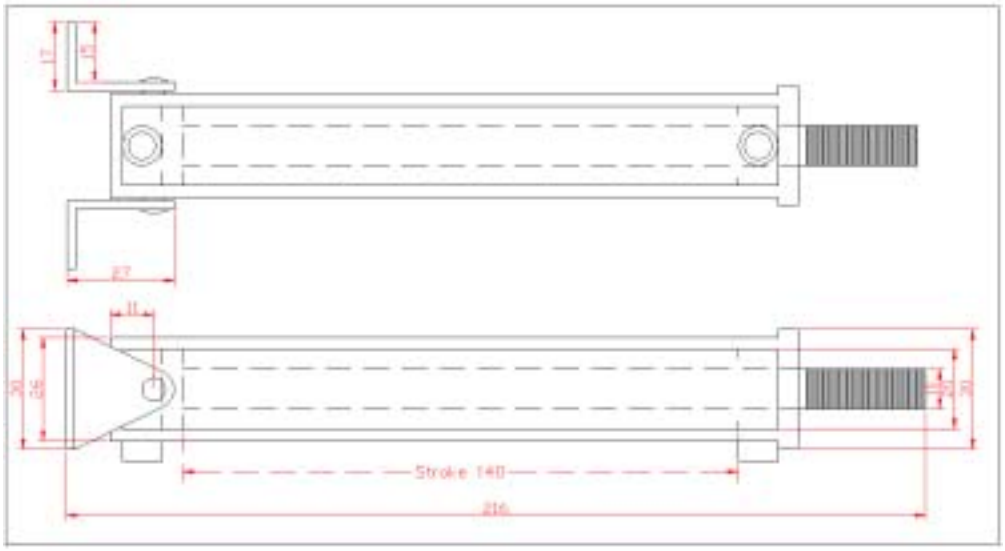


그림 3.1.8 선정한 실린더 치수

다. 밸브의 선정

밸브는 모두 전자로 제어를 할 수 있어야 하기 때문에 4port 3way Solenoid Valve를 사용하였으며 단, 주클러치 밸브는 안전을 위해 시동이 꺼진 생태에서도 항상 클러치가 잡혀서 움직이지 못하도록 neutral일 때에도 모든 포트가 막혀있는 구조로 선정 하고, 좌우 조향클러치 밸브는 시동이 꺼진 상태로 neutral일 때 조향레버를 당기면 당겨져서 브레이크를 잡을 수 있도록 A,B 포트가 드레인 라인과 통하도록 되어있는 구조로 선정하였다.

그림 3.1.9은 Solenoid Valve의 구조를 나타낸 것이다.

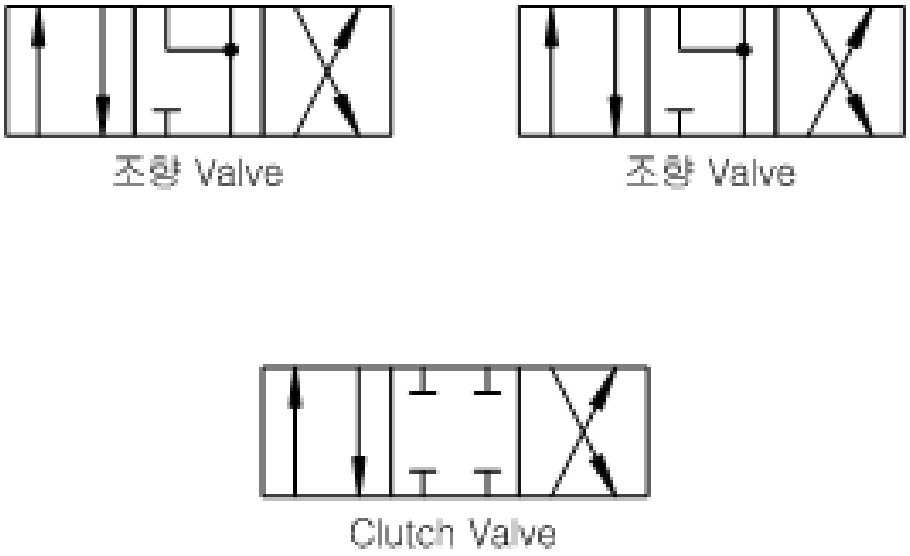


그림 3.1.9 Solenoid Valve의 구조

라. 유압시스템의 구성 및 이해

1) 유압시스템의 구성

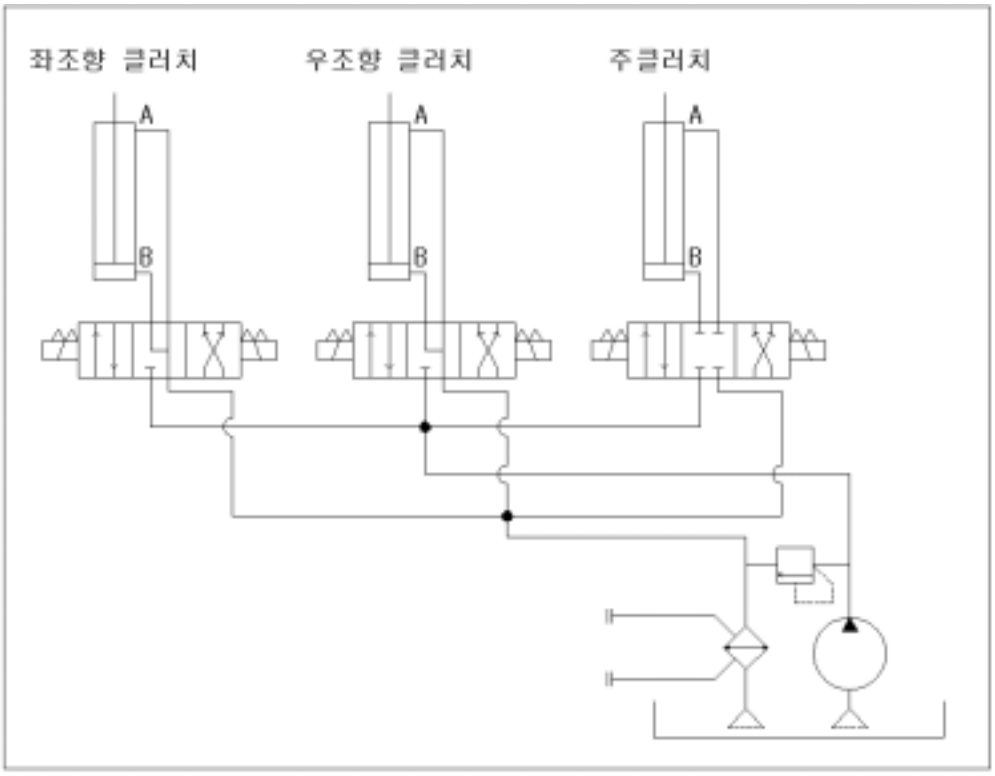


그림 3.1.10 유압시스템의 회로도

유압 밸브와 실린더는 반응이 같아야 하기 때문에 모두 같은 사양으로 구성하였고, P라인에 릴리프 밸브를 구성하였다. 또한 환원되는 유압유를 냉각시키기 위해 드레인 되는 T라인에 냉각기를 구성하였다.

또한 실린더로 들어가고 나오는 유체의 압력과 유량의 특성을 알기위해 각 실린더의 A, B 포트에 퀵 커플러로 구성하여 실린더로 들어가고 나오는 위치에서 원하는 실린더에 유량 및 압력센서를 쉽게 탈 부착 할 수 있도록 하였다.

그림 3.1.11은 실제 실린더의 장착 모습을 나타내고 있다.

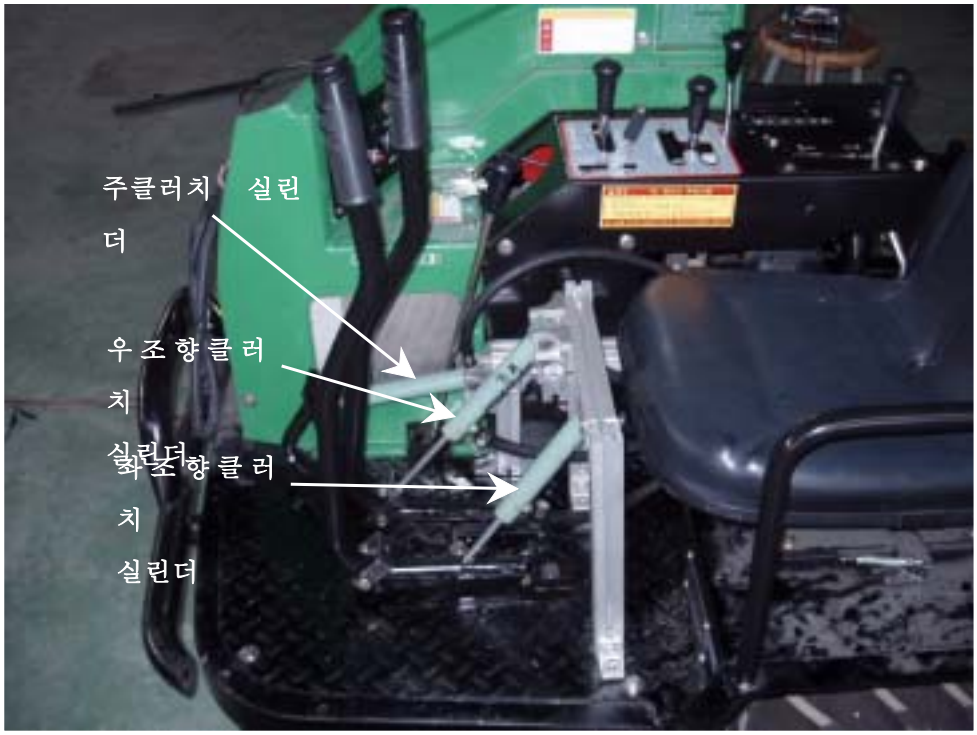


그림 3.1.11 실린더 장착 모습

2) 유압장치 구조의 이해

그림 3.1.10에서 보는바와 같이 모든 실린더는 A방향으로 유체가 흘러 들어가면 실린더의 arm을 끌어당기도록 되어있다. 조향 실린더에서 유체를 A방향으로 흘려보내게 되면 구동되던 바퀴의 동력을 끊어 동력을 끊은 쪽으로 회전을 하게 되어 있고, 주클러치에서는 B방향으로 유체를 흘려보내게 되면 밸트의 장력을 늘어뜨려 바퀴로 가는 모든 동력을 끊어 구동을 멈추도록 되어 있다.

세 개의 솔레노이드 밸브는 전기가 흐르지 않을때는 항상 중립을 유지하고 있다가 전기가 통하게 되면 다른 전기신호가 들어오기 전까지는 항상 정방향(왼쪽)으로 유체를 흘려보내게 되어있다. 이것은 운전자가 모르고 기어를 넣어놓은 상태에서 시동을 걸때 차량이 급발진하여 사고가 나는 것을 방지하기 위하여 항상 시동을 걸 때는 차량이 움직이지 않도록 클러치를 잡아주기 위해 구성을 한 것이다.

마. 유압시스템의 유량특성

1) 주클러치의 유량특성

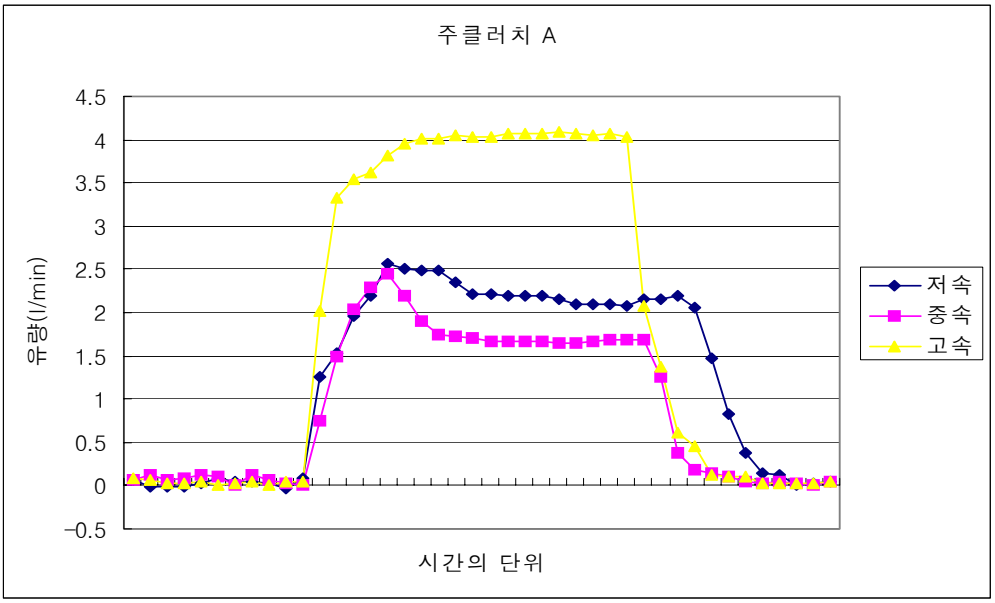


그림 3.1.12 주클러치 실린더 A로 유체를 흘려보냄

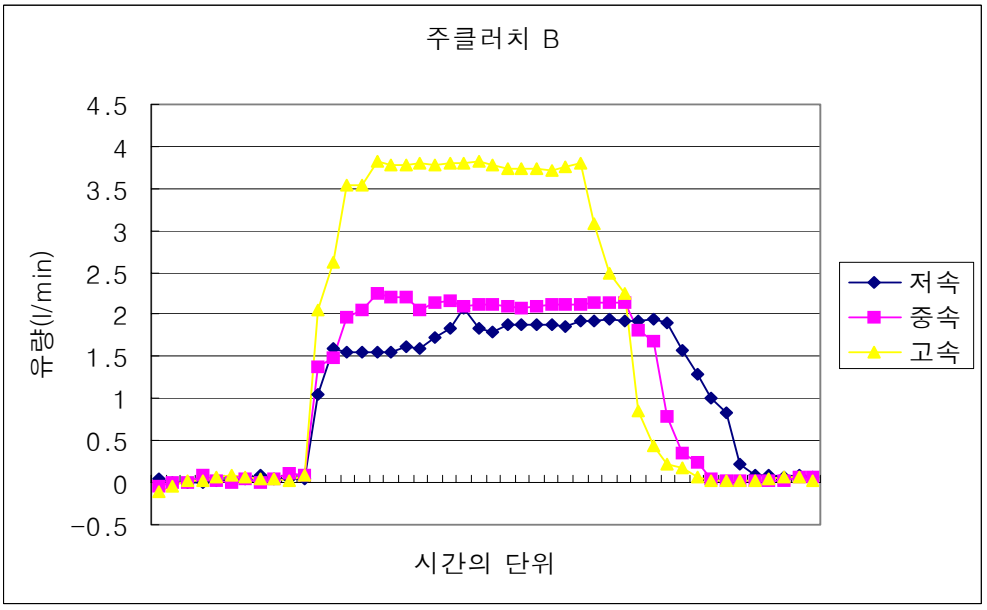


그림 3.1.13 주클러치 실린더 B로 유체를 흘려보냄

위의 data값들은 10msec 즉, 0.01초당 1포인트씩 출력하여 얻어진 값들이다.

최초 유량이 흐르지 않는 상태에 있다가 솔레노이드 밸브에 전기신호가 주어져 밸브가 작동하고 그에 따라 실린더가 움직이면서 실린더로 들어가는 유체의 양을 시간의 흐름에 따라 그래프로 표기한 것이다.

그래프의 해석은 data의 신뢰성이 가장 높고 안정적인 중속일 때의 data를 기준으로 해석한다.

최초 유량이 급상승하는 구간은 막혀 있던 유체가 신호를 받아 서서히 움직이기 시작하는 구간으로 0.05초 내에 정상적인 유체의 흐름을 나타내고 실린더를 밀거나 당기는 시간은 전 속도에서 0.20초~0.25초 내에 실린더가 움직임을 알 수 있다.

그 이후 포인트가 떨어지는 구간은 솔레노이드 밸브에 가해지던 전기 신호가 끊어지면 잔류 유체가 압축되어 들어온 용적만큼 실린더 내로 들어감을 나타낸다.

그림 3.1.12과 그림 3.1.13의 그래프에서 보는 바와 같이 유량이 상승하다가 조금 떨어지면서 일정유량을 흘려보냄을 알 수 있는데 이것은 일종의 damping 현상으로 안 움직이려는 물체를 억지로 밀 때 최초 힘이 많이 들어가는 것처럼 실린더를 움직이기 직전까지 밀려오던 유체가 조금 압축되었다가 실린더가 움직이기 시작하면서 한번에 몰아두었던 유체를 흘려보낼 때 나타날 수 있는 현상이다. 고속으로 갈수록 이런 현상이 적게 나타나는 것은 실험 data를 출력하는 시간보다 짧은 시간에 그런 현상이 이미 이루어지기 때문에 나타나지 않는 것일 뿐 저속일 때나 고속일 때나 모두 같은 현상이 일어난다.

아래 그림 3.1.14는 중속을 중심으로 위의 그래프를 한눈에 알아볼 수 있도록 상승구간, 실린더 작동구간, 유체 압축구간으로 나누어 간략하게 나타낸 그림이다.

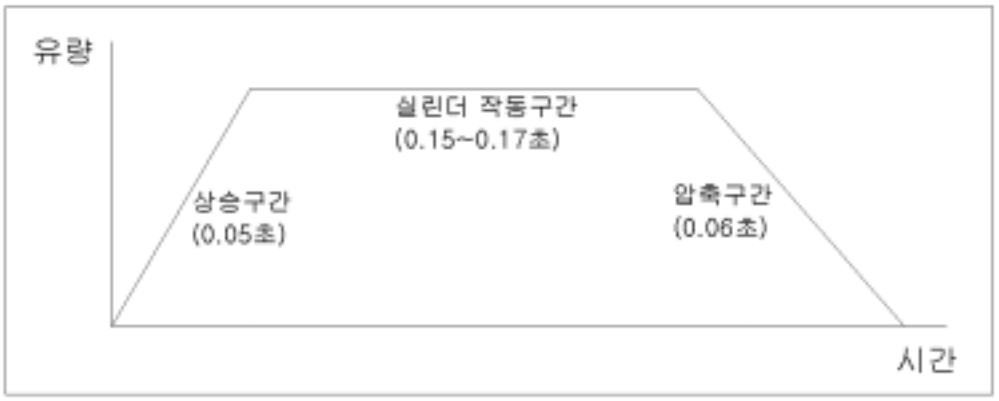


그림 3.1.14 주클러치의 구간별 유량특성

2) 좌조향 클러치의 유량특성

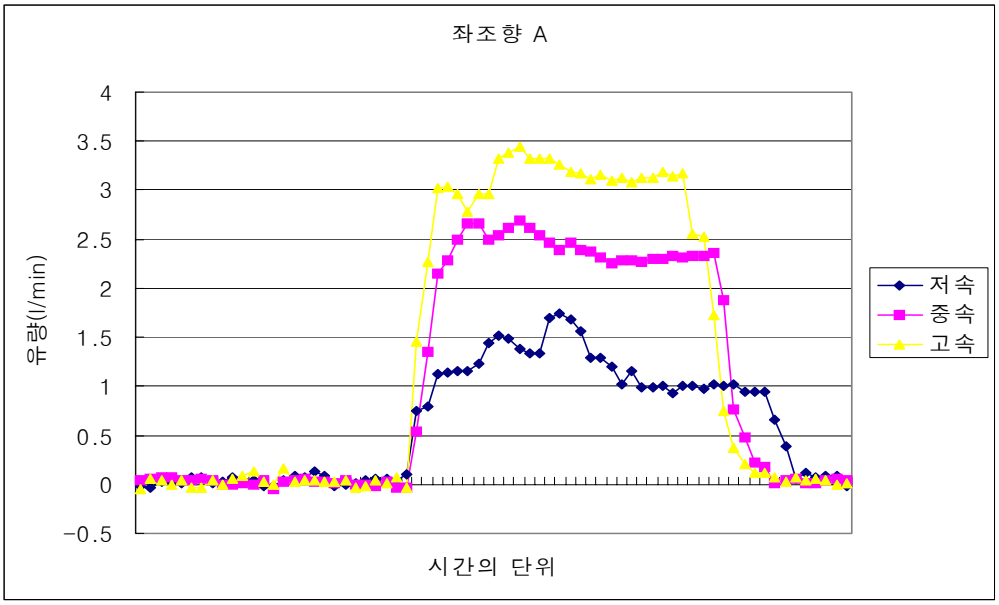


그림 3.1.15 좌조향 실린더 A로 유체를 흘려보냄

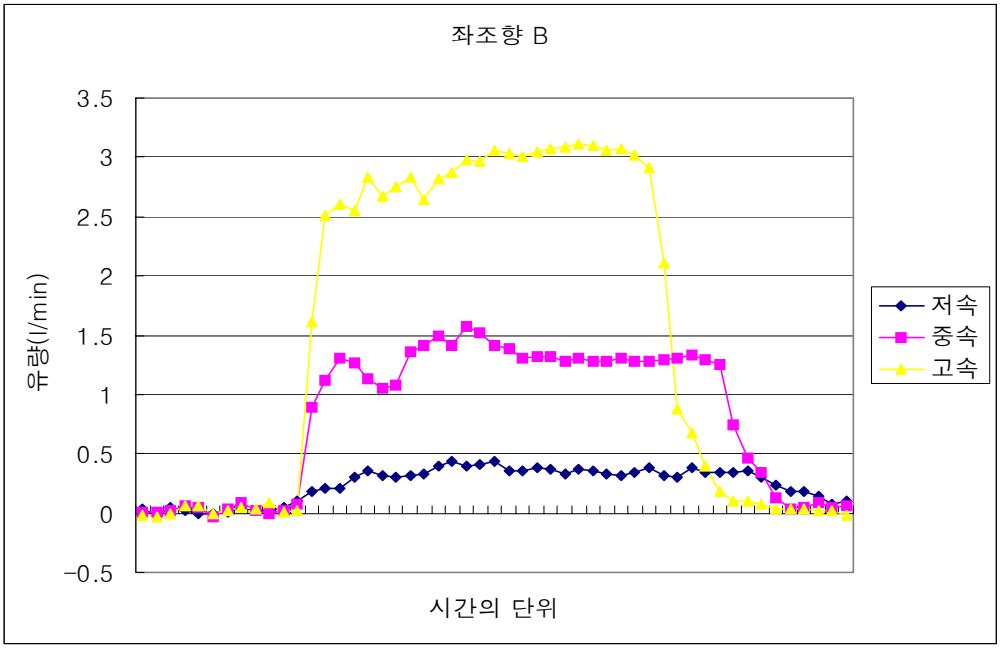


그림 3.1.16 좌조향 실린더 B로 유체를 흘려보냄

좌조향 실린더에서는 최초 유체가 흐르기 시작하여 정상적인 흐름이 있을 때까지 A로 유체를 흘려보냈을 때와 B로 유체를 흘려보냈을 때의 차이를 알 수 있다. 다시 말해 A로 유체를 흘려보냈을 때는 상승구간까지 0.07초 B로 유체를 흘려보냈을 때는 0.03초를 경과함을 알 수 있는데 이 차이는 A로 유체를 흘려보낼 때는 레버를 밀고 있는 스프링의 장력 때문에 레버가 당겨지지 않으려는 힘을 받고 있어 B로 흘려보낼 때 보다 다소 시간이 더 많이 걸린다는 것을 알 수 있고 반대로 B로 유체를 흘려보낼 때는 밀어내려는 스프링의 힘과 유체의 힘이 합쳐져서 그만큼 빠른 시간내에 실린더를 움직이는 것을 알 수 있다.

위에서 설명한바와 마찬가지로 두 경우 모두 damping 현상은 일어나고 있고, 저,중,고속일 때를 비교해 보면 고속으로 갈수록 유체의 양은 늘어나고 그만큼 작동시간은 짧아짐을 확연히 알 수 있다.

A로 유체를 흘려보낼 때나 B로 유체를 흘려보낼 때 실린더가 움직인 총 작동시간은 0.3초(중속)로 같음을 알 수 있다. 또한 잔류유체가 유입되어 압축되어지는 시간도 0.06초와 0.05초로 거의 같음을 알 수 있다.

아래 그림 3.1.17은 중속을 중심으로 위의 그래프를 한눈에 알아볼 수 있도록 상승구간, 실린더 작동구간, 유체 압축구간으로 나누어 간략하게 나타낸 그림이다.

그림 3.1.17 좌조향 클러치의 구간별 유량특성

3) 우조향 클러치의 유량특성

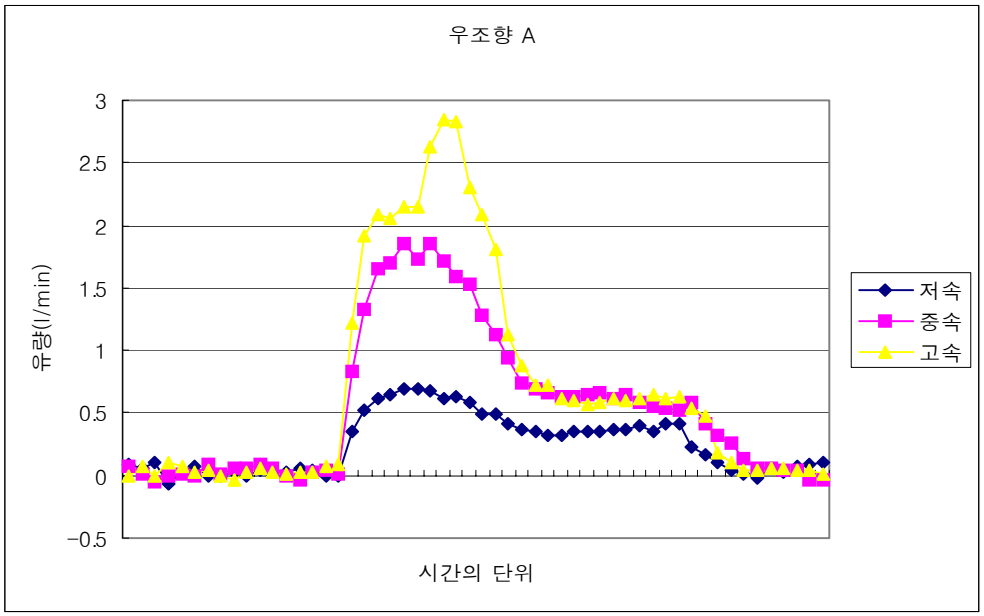


그림 3.1.18 우조향 실린더 A로 유체를 흘려보냄

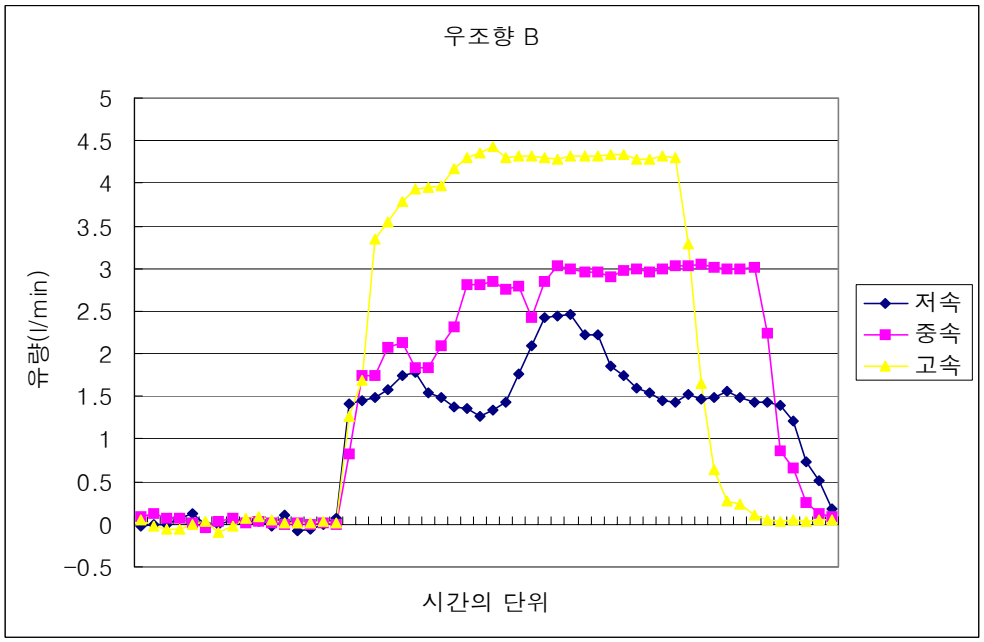


그림 3.1.19 우조향 실린더 B로 유체를 흘려보냄

우조향의 경우 A로 유체를 흘려보낼 때 damping 현상이 심하고 그 이후 유체의 양도 다른 실린더에 흐르는 양에 비해 현저히 적음을 알 수 있지만 실린더를 작동하는 시간은 다른 실린더와 같이 약 0.3초 내외로 컨트롤 하는데 있어서 큰 문제가 없음을 보여준다.

아래 그림 3.1.20은 중속을 중심으로 위의 그래프를 한눈에 알아볼 수 있도록 상승구간, 실린더 작동구간, 유체 압축구간으로 나누어 간략하게 나타낸 그림이다.

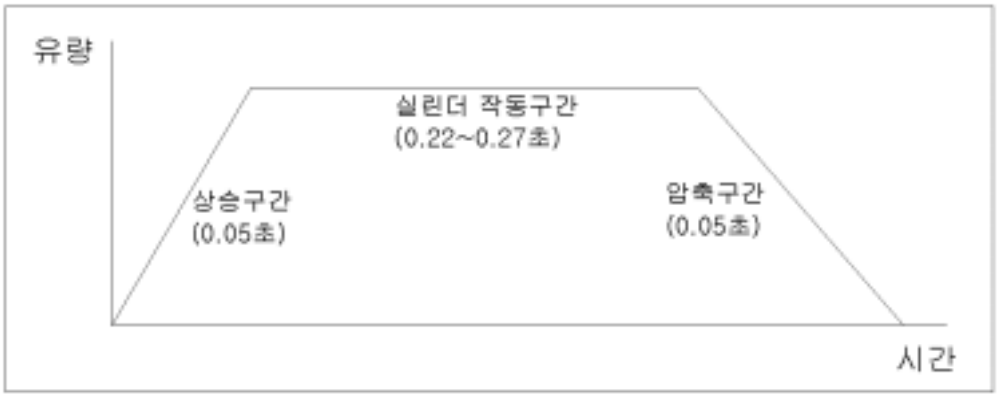


그림 3.1.20 우조향 클러치의 구간별 유량특성

4) 유량센서의 검증

유량의 모든 data에서 센서로부터 측정되어진 값들을 Calibration 식에 의해 l/min의 유량값으로 환산하면 유량이 흐르지 않는 구간에서도 약 1l/min의 유체가 흐르는 것으로 확인되었다. 따라서 센서와 Calibration 식의 정확성을 파악하기 위해 SS기에 장착된 동일한 펌프를 사용하여 센서로부터 얻어지는 data값과 실제로 흘러가는 유체를 비이커에 담아 임의로 부하를 주어가면서 측정값을 비교해 보았다.

Table3.1.4

	모듈값	실측정값	차이
무부하	6.497966756	5.60554	0.892427
	6.692589	5.95041	0.742179
	6.683804	5.714285	0.969519
	6.903759	5.480769	1.42299
	6.664413	5.311475	1.352938
평균	6.688506	5.6124958	1.076011
5kg load	6.289143973	5.392156	0.896988
	6.655389	5.331325	1.324064
	6.721902	5.7096774	1.012225
	6.705845673	5.64263	1.063216
	6.676437	5.49689	1.179547
평균	6.609744	5.51453568	1.095208
10kg load	6.720672	6.0251	0.695572
	6.536763	5.60439	0.932373
	6.578281	5.742188	0.836093
	6.665561	6.244344	0.421217
	6.664039	5.793	0.871039
평균	6.633063	5.881804	0.751259
15kg load	6.592715	5.576208	1.016507
	6.694025	5.976095	0.71793
	6.652691	5.55066	1.102031
	6.716218	5.707317	1.008901
	6.635016	5.675676	0.95934
평균	6.658133	5.697191	0.960942

실험으로부터 모듈값과 실측정값은 0.7~1.3(l/min), 평균 0.970855 l/min의 차이를 보이고 있다.

이 사실로 보아 센서로부터 얻어지는 data값을 Calibration 식에 의해 환산한 값은 실제 흘려가는 유량과 비교해 볼 때 약 0.97l/min 이 더 많게 측정됨을 확인 할 수 있었다.

따라서 실제 흐르는 유량은 측정되어진 값에서 0.97l/min을 수정하여 표시하였다.

바. 유압시스템의 압력특성

1) 주클러치의 압력특성

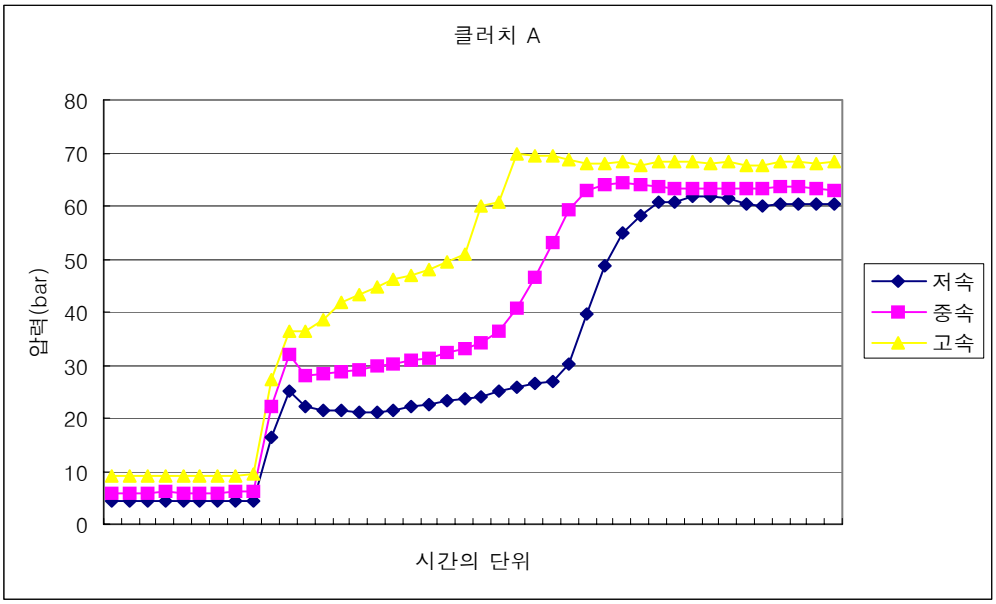


그림 3.1.21 주클러치 실린더 A로 유체를 흘려보냄

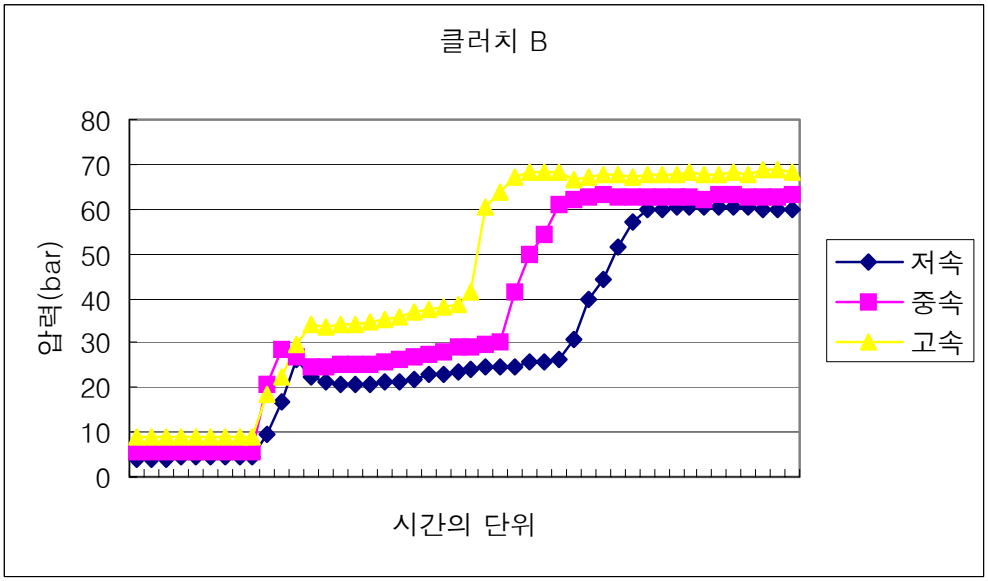


그림 3.1.22 주클러치 실린더 B로 유체를 흘려보냄

유량에서 살펴본 바와 달리 압력에서는 data의 분포가 상대적으로 고름을 알 수 있다. 그래프의 곡선이 S자곡선의 유형을 따라가며 유량특성과 같은 damping 현상이 일어남을 확인 할 수 있다.

최초 압력이 급상승하는 구간은 실린더를 밀어내기 직전까지 압력이 커지며 이후 압력이 완만한 상승을 보여주는 구간은 실린더를 움직이는 구간임을 알 수 있다. 이후 0.03~0.06초의 급상승 하는 구간은 유량특성에서 살펴본바와 같이 잔류유체가 펌프의 힘에 의해 추가로 흘러들어가면서 유체가 압축됨을 의미한다.
실린더의 작동시간은 두 경우 약 0.15초 내외로 작동이 완료됨을 알 수 있다.

아래 그림 3.1.23은 중속을 중심으로 위의 그래프를 한눈에 알아볼 수 있도록 상승구간, 실린더 작동구간, 유체 압축구간으로 나누어 간략하게 나타낸 그림이다.

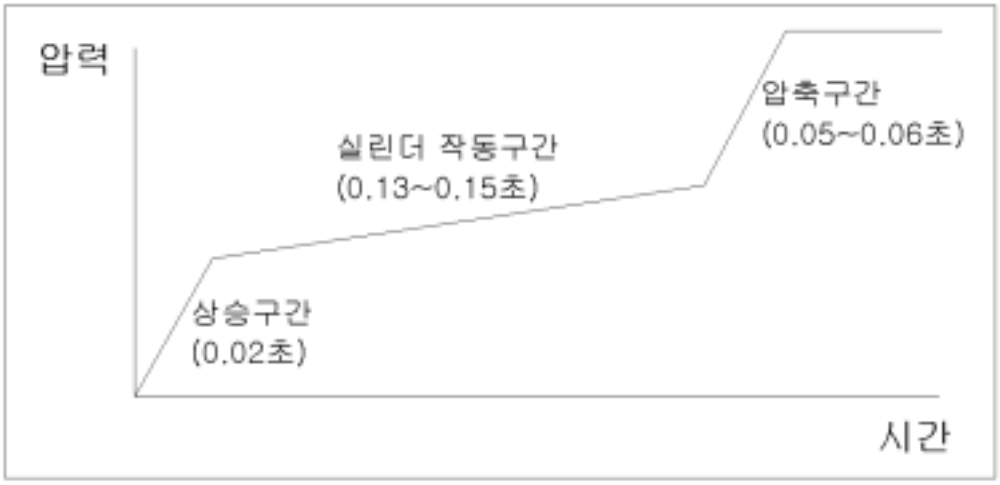


그림 3.1.23 주클러치의 구간별 압력특성

2) 좌조향 클러치의 압력특성

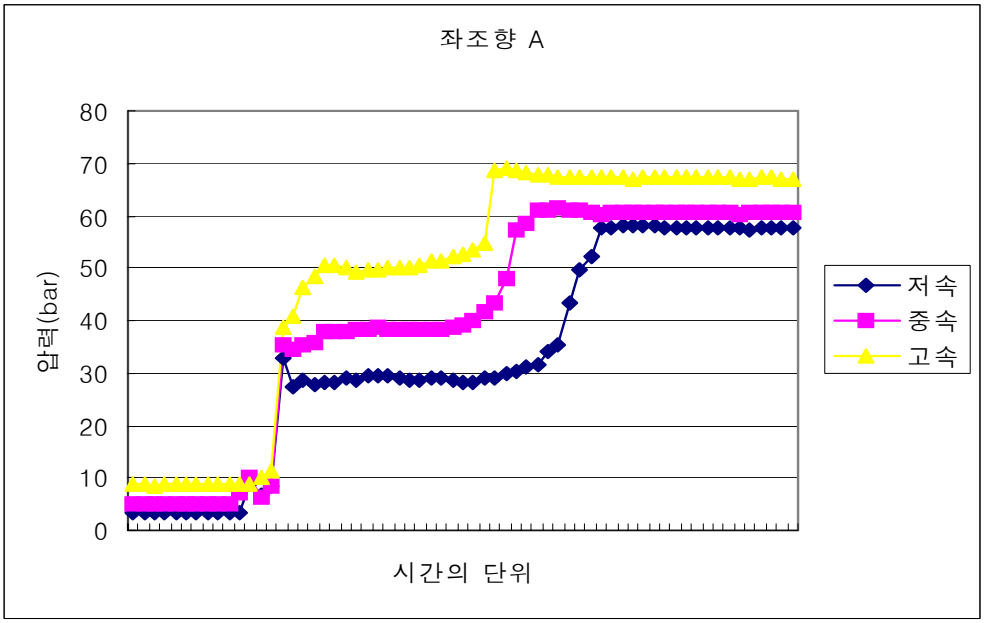


그림 3.1.24 좌조향 실린더 A로 유체를 흘려보냄

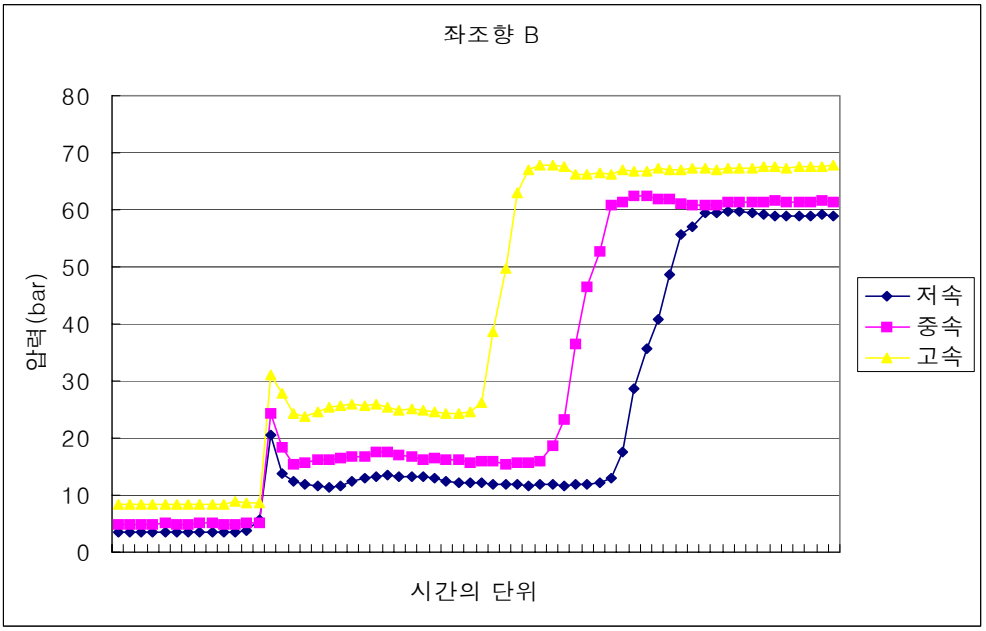


그림 3.1.25 좌조향 실린더 B로 유체를 흘려보냄

주클러치의 곡선과 거의 같은 유형의 형태를 보여주고 있으며 주클러치와 다르게 실린더의 작동구간에서 B로 흘려보낼 때가 A로 흘려보낼 때에 비해 압력이 현저히 떨어져 있음을 알 수 있다. 이것은 조향 레버를 당기고 있는 트렌스미션 내부의스프링 장력에 의해 B로 유체를 흘려보낼 때는 유체의 힘과 스프링의 힘이 같이 작동하여 A로 흘려보낼 때 보다 압력이 현저히 떨어지는 것이다.

실린더의 작동시간은 A로 흘려보낼 때와 B로 흘려보낼 때 약 0.25초 이내에 작동이 완료됨을 알 수 있다.

아래 그림 3.1.26은 중속을 중심으로 위의 그래프를 한눈에 알아볼 수 있도록 상승구간, 실린더 작동구간, 유체 압축구간으로 나누어 간략하게 나타낸 그림이다.

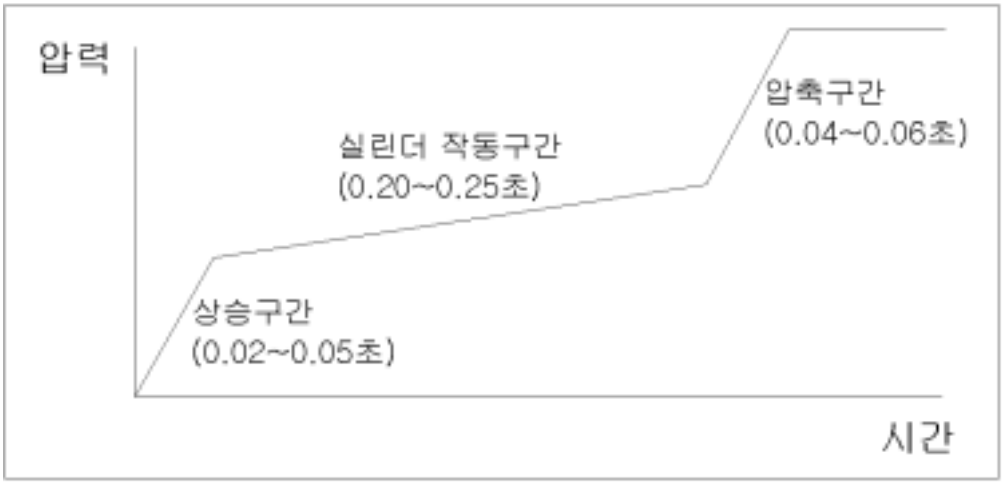


그림 3.1.26 좌조향 클러치의 구간별 압력특성

3) 우조향 클러치의 압력특성

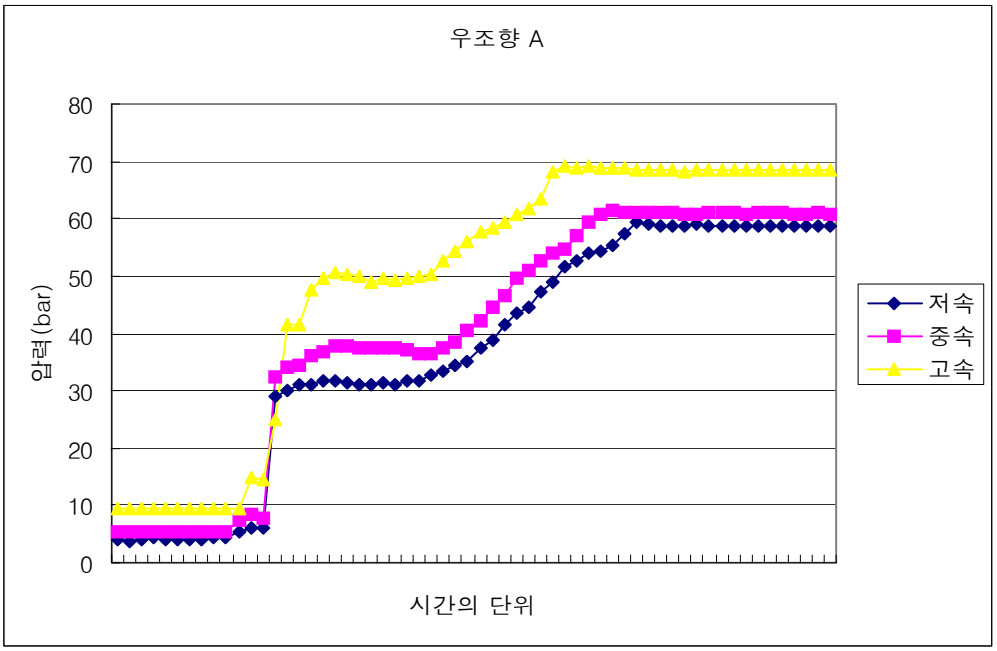


그림 3.1.27 우조향 실린더 A로 유체를 흘려보냄

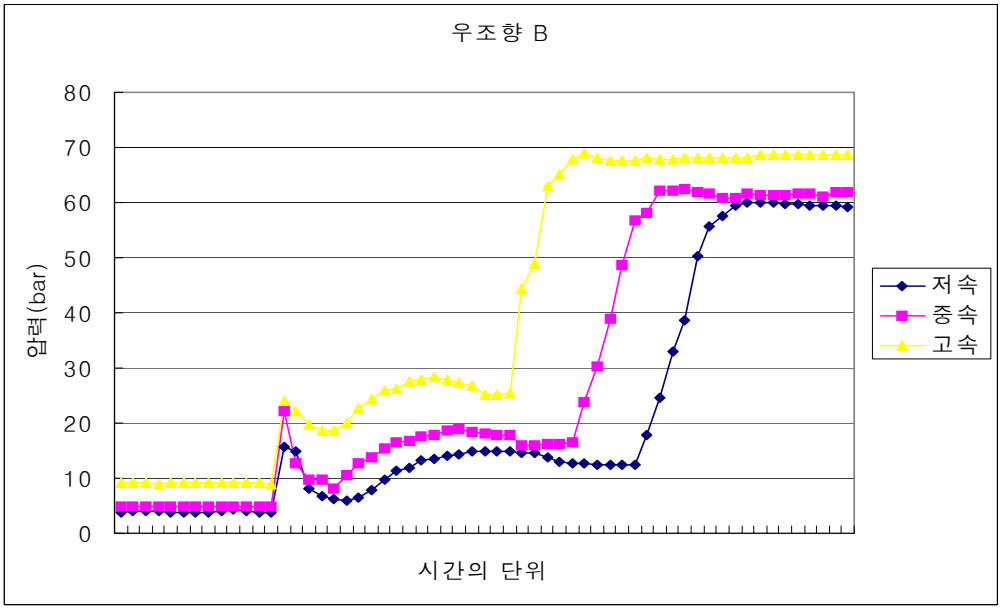


그림 3.1.28 우조향 실린더 B로 유체를 흘려보냄

좌조향일 때와 거의 같은 유형의 곡선을 보여주는데 A로 유체를 흘려보낼 때 다른 곡선들에 비해 실린더 작동후 압력이 상승하는 구간이 비교적 완만함을 볼 수 있다. 유량의 흐름에서도 마찬가지고 우조향 A로 유체를 흘려보낼 때 다른 곡선들과 차이를 볼 수 있었는데 이것은 조향장치의 특성상 우조향 A로 유체를 흘려보낼 때 즉, 레버를 당길 때는 실린더의 작동이 단기간 내에 이루어지고 유체의 압축이 서서히 이루어짐을 알 수 있다. 이것은 좌조향과 우조향의 기계적 특성상 동일 할 수 없기 때문에 이러한 현상이 나타났다.

또한 모든 압력의 그래프에서 엔진 RPM이 저속 중속 고속일 때 실린더가 작동한 이후에 고속으로 갈수록 압력도 같이 높아져 있음을 알 수 있다. 이것은 전기로 밸브를 작동시켜주는 시간은 항상 같은데 밸브로부터 센서를 거쳐 실린더로 흘러들어가는 유량이 엔진 RPM이 빨라질수록 그만큼 더 빨리 실린더로 유입되기 때문에 나타나는 현상이라 볼 수 있다.

아래 그림 3.1.29은 중속을 중심으로 위의 그래프를 한눈에 알아볼 수 있도록 상승구간, 실린더 작동구간, 유체 압축구간으로 나누어 간략하게 나타낸 그림이다.

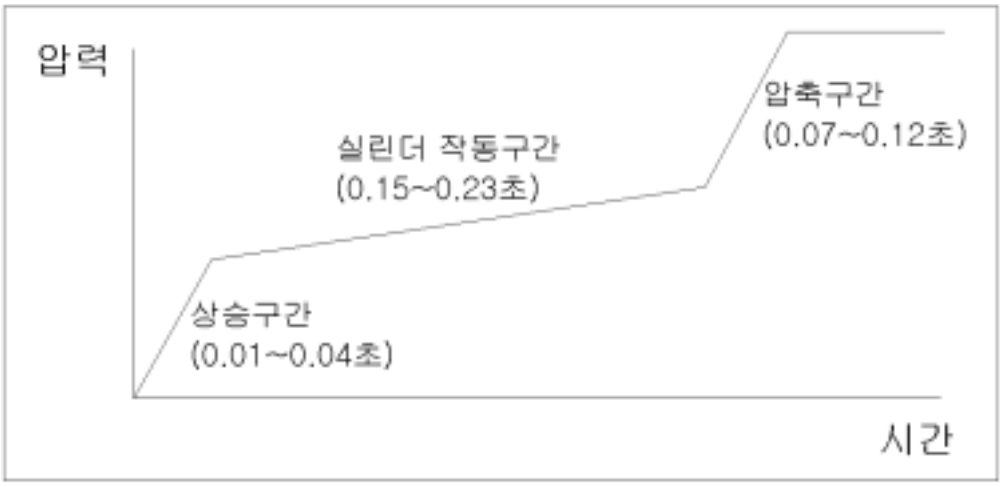


그림 3.1.29 좌조향 클러치의 구간별 압력특성

사. 작동특성의 비교

주클러치와 각각의 조향클러치를 중속일 때를 기준으로 하여 비교한다.
주클러치와 조향클러치 모두 유체를 A방향으로 흘려보낼 때는 실린더의 arm을 당길 때이고, B방향으로 유체를 흘려보낼 때는 실린더의 arm을 밀 때이다.

표 3.1.5

		유량에 따른 작동시간	압력에 따른 작동시간	유량에 따른 유체 압축시간	압력에 따른 유체 압축시간
주클러치	당길때	0.15초	0.13초	0.06초	0.05초
	밀때	0.17초	0.15초	0.06초	0.06초
좌조향	당길때	0.23초	0.20초	0.06초	0.04초
	밀때	0.27초	0.25초	0.05초	0.06초
우조향	당길때	0.22초	0.15초	0.05초	0.12초
	밀때	0.27초	0.23초	0.05초	0.07초

표 3.1.5에서 보는바와 같이 거의 비슷한 시간 내에서 실린더의 작동이 이루어지며 미소한 차이지만 당길 때 보다 실린더 arm을 밀어줄 때 소요시간이 더 걸림을 확인할 수 있다. 하지만 근소한 차이라 할 수 있으며 모든 작업이 0.3초 이내에는 완료가 됨을 알 수 있다. 따라서 유체가 압축되는 시간을 제외하고 실린더의 작동시간이 최대 0.3초 이므로 작업기를 control 하는데 있어서 control 시간을 0.3초 이상으로 잡고 작업기를 control 한다면 조향하는데 있어서 문제는 없다.

아. Data의 역학적 해석

원활한 조향을 위해 유압장치를 설치하기전 조향장치에 연결되어 있는 장력 스프링들은 모두 제거하였고 클러치는 엔진으로부터 오는 동력을 끊어주려면 어느정도의 장력을 필요로 하기 때문에 클러치의 장력스프링은 제거 하지 않았다. 이로서 주클러치를 제어하는 실린더가 가장 힘을 많이 쓰기 때문에 이하 역학적 해석에서는 주클러치 실린더를 기준으로 해석한다.

먼저 주클러치 축의 토크(torque)를 측정하기 위해 주클러치 축에 1050mm의 봉을 연결하고 이 봉을 tangent방향으로 당길 때 요구되는 힘을 측정하여 주클러치 축으로부터 주클러치 실린더가 밀어주는 위치점에서 요구되어지는 힘을 다음과 같이 산출하였다.

1050mm의 봉에서 tangent방향으로 당기는 힘을 F_0라 하고, 주클러치 축에서 실린더가 밀어주는 위치점까지의 거리가 202mm이고 이 위치에서 주클러치를 tangent방향으로 밀어주는 힘을 F_1이라고 하면,

$F_1 = \frac{F_0 * 1050}{202}$ 로부터 F_1값을 찾을 수 있다.

주클러치가 축을 중심으로 16deg 까지 돌아가고 한계점까지 돌아가는데 요구되어지는 힘이 다음과 같이 측정되었다.

표 3.1.6

Deg / F(kg_f)	0	2	4	6	8	10	12	14	16
F_0	0	1.7	2.2	2.5	2.6	2.8	3.2	3.4	3.4
F_1	0	8.84	11.44	13	13.51	14.55	16.63	17.67	17.67

주클러치 실린더는 주클러치를 밀어줄 때 축으로부터 tangent방향으로 주클러치를 밀어주지 못하므로 F_1의 힘에서 실린더가 밀어줄 때 요구되어지는 힘(F_2)를 다음과 같이 산출하였다.

F_1과 F_2사이의 각을 θ라 하면,

$F_1 = F_2\cos\theta$ or $F_2 = \frac{F_1}{\cos\theta}$ 임을 알 수 있다.

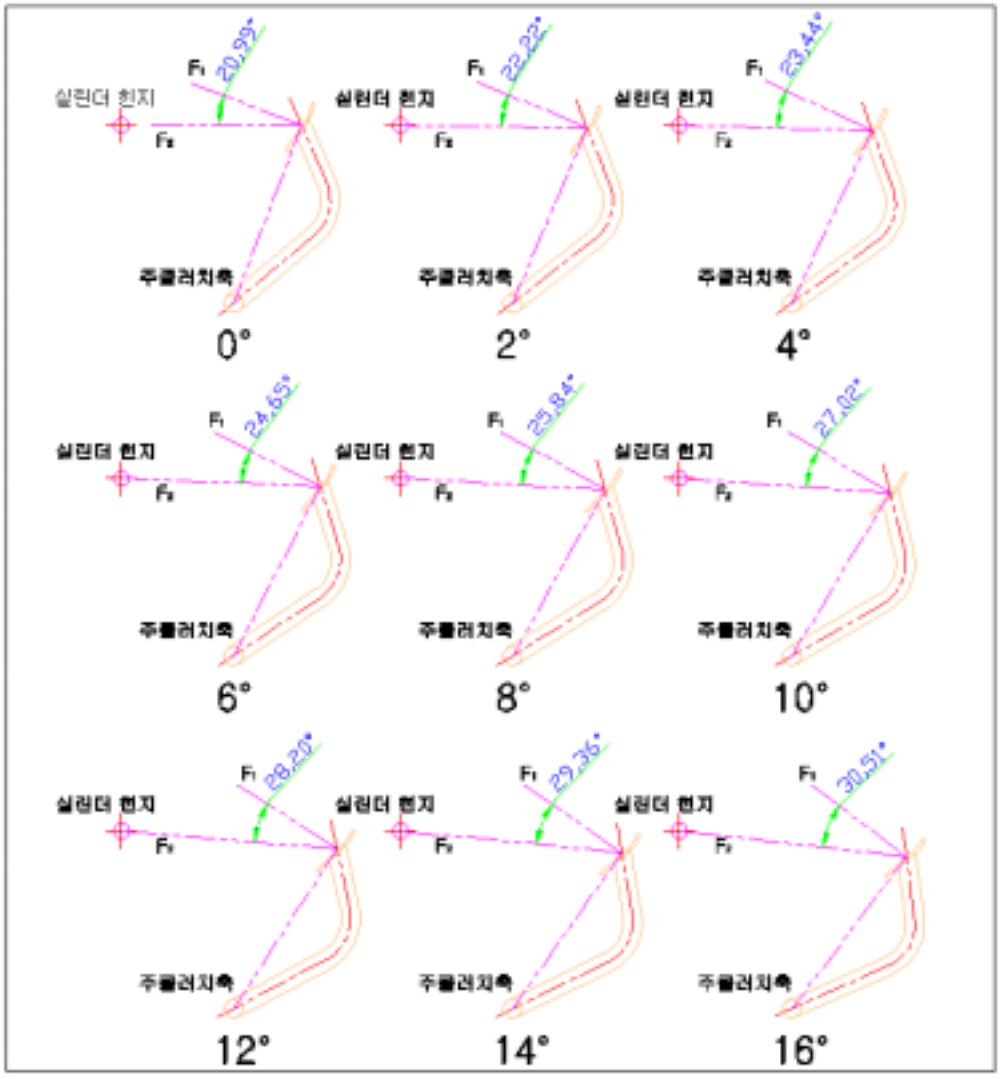


그림 3.1.30 주클러치 회전에 따른 θ값

F_1과 F_2사이의 각θ를 알기 위해 그림 3.1.30과 같이 AutoCAD를 이용하여 클러치를 2deg씩 회전해 가면서 θ를 측정하였고 이를 토대로 F_2를 산출하였다.

표 3.1.7

Deg.	0	2	4	6	8	10	12	14	16
$F_2(kg_f)$	0	9.55	12.48	14.3	15.01	16.33	18.88	20.26	20.5

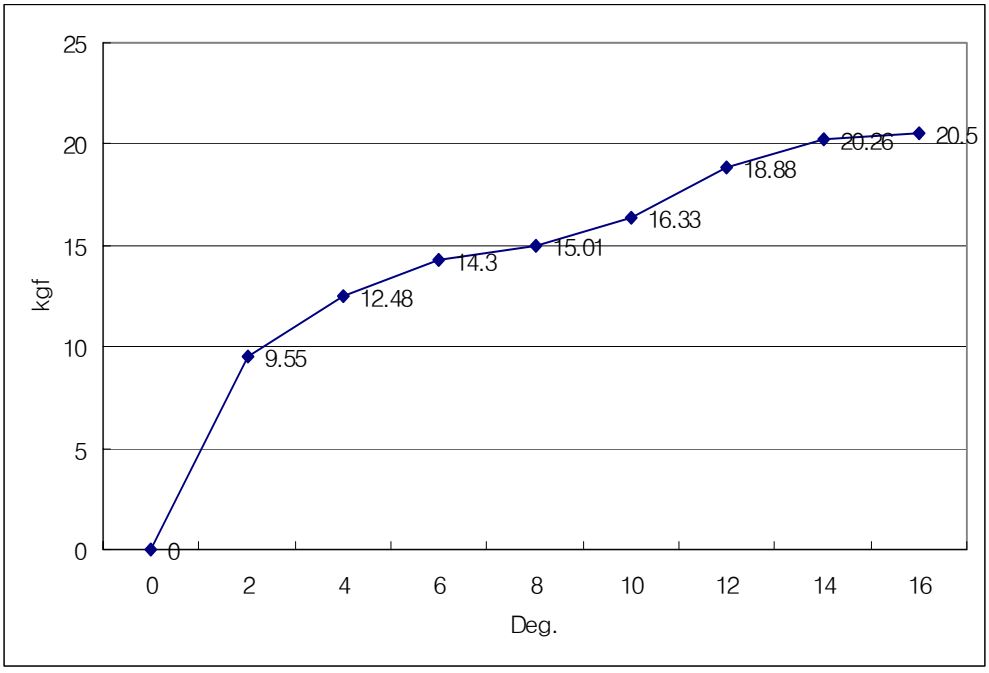


그림 3.1.31 주클러치 실린더에 요구되어지는 힘(F_2)

그림 3.1.31은 산출되어진 F_2값을 그래프로 나타낸 것이다.

그림 3.1.21의 중속일 때 그래프에서 힘과 압력의 관계식에 의해 실린더가 주클러치를 밀고 있는 힘을 알아낼 수 있다.

$F = P * A$

내경이 20mm인 실린더의 단면적

$$A = \frac{D^2\pi}{4} = \frac{20^2\pi}{4} = 314.16mm^2$$

따라서 중속일 때의 압력으로부터 그림 3.1.32과 같이 주클러치를 미는 힘을 찾아낼 수 있다.

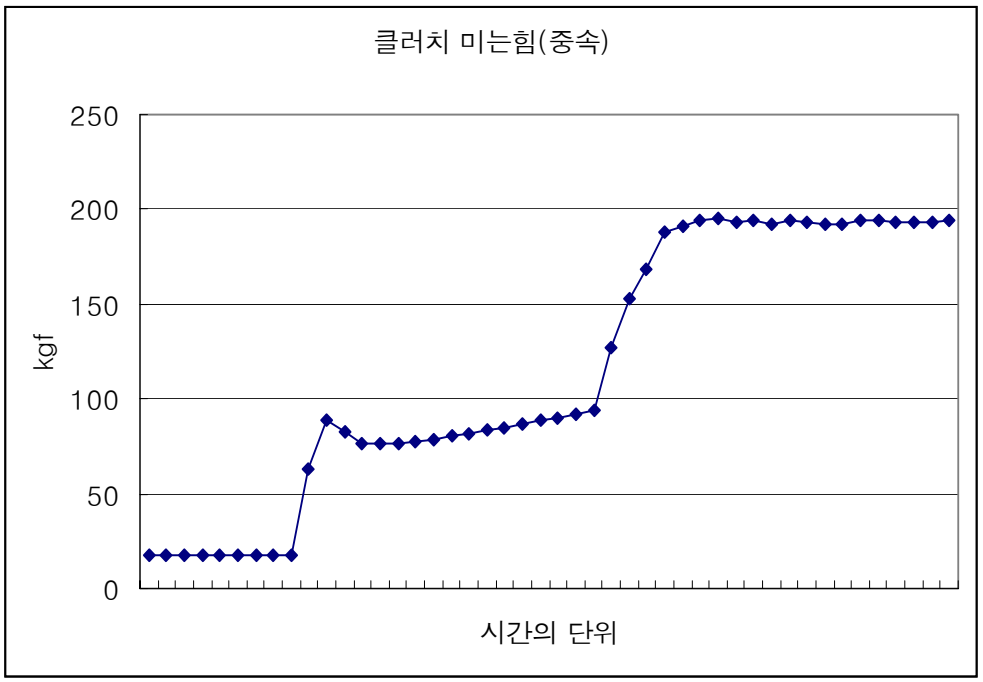


그림 3.1.32 압력으로부터 산출한 실린더가 주클러치를 미는 힘(중속)

그림 3.1.31와 그림 3.1.32에서 알 수 있듯이 실린더가 주클러치를 미는 힘은 최대 거의 200kg_f에 가까워지는데 실제 주클러치가 밀리는 힘은 최대 20.5kg_f에 불과하다. 따라서 유체가 실린더에 밀려들어오는 순간 이미 주클러치를 밀어줄 수 있는 충분한 힘을 갖고 일을 시작함을 알 수 있다. 이와 달리 초기에 힘을 충분히 갖지 못하고 작업을 시작한다면 실린더를 작동하는데 있어서 반응속도가 느려질 것이다.

자. 요약

1) 동력전달 체계

엔진의 동력은 밸트를 통해 트렌스 미션으로 전해지고 밸트의 장력을 늘이거나 줄임으로 전해지는 동력을 끊어주거나 전하게 된다. 또한 전해지는 동력은 조향 클러치에 의해 원하고자 하는 차륜의 바퀴를 정지시켜줌으로서 미끄러지며 회전을 하게 된다.

2) 주클러치

엔진과 트렌스미션을 연결해 주는 밸트의 장력을 조절하는 것이 주클러치 이다.

제어를 위해 동력이 끊어지는 주클러치의 변위를 알아본 결과 캠패드가 최초 10deg에서 동력이 끊어지며 이때 주클러치축을 0deg~10deg까지 제어해야할 것을 확인하였다.

3) 좌조향 클러치

좌조향 클러치는 캠 링크가 최초 36deg 돌아갈 때 동력이 끊어지므로 이때 조향레버는 9.3deg까지 제어해 주면된다.

4) 우조향 클러치

우조향 클러치는 캠 링크가 최초 36deg 돌아갈 때 동력이 끊어지므로 이때 조향레버는 7.3deg까지 제어해 주면된다.

5) 유압실린더의 선정

유압실린더를 선정하기 위해 실린더가 장착될 위치를 파악하고 그 위치에서 실린더가 필요로 하는 힘을 구한결과 주클러치와 좌우 조향클러치에서 165.62N 이상의 힘을 발산할 수 있으면 되었다.

따라서 실린더의 내경이 20mm이고 행정의 길이가 140mm인 실린더를 선정하여 제작하였다.

6) 밸브의 선정

밸브는 모두 전자로 제어를 할 수 있어야 하기 때문에 4port 3way Solenoid Valve를 사용하였으며 단, 주클러치 밸브는 안전을 위해 시동이 꺼진 생태에서도 항상 클러치가 잡혀서 움직이지 못하도록 neutral일 때에도 모든 포트가 막혀있는 구조로 선정 하고, 좌우 조향클러치 밸브는 시동이 꺼진 상태로 neutral일 때 조향레버를 당기면 당겨져서 브레이크를 잡을 수 있도록 A,B 포트가 드레인 라인과 통하도록 되어있는 구조로 선정하였다.

7) 유압시스템의 구성

유압시스템은 SS기에 장착되어진 펌프를 사용하여 조향을 위한 3개의 실린더, 전자제어가 가능한 솔레노이드 밸브 3개, 릴리프 밸브 1개, 오일쿨러 1개로 구성하였다. 구성된 유압시스템은 시동이 걸림과 동시에 어떤 상황에서도 최초 전기가 솔레노이드 밸브에 전해지면 클러치를 밀어 동력을 끊어주게 되어 있어 차량의 급발진 또는 내리막길에서의 미끌림을 방지하였다.

8) 유량 및 압력특성

Table 3.1.5에서 확인한 바와 같이 모든 실린더의 작동시간이 0.3초 이내에 이루어짐을 알 수 있고 이것을 토대로 최소 control time을 0.3초로 하였다.

9) Data의 역학적 해석

실제로 측정해 본 결과 주 클러치 실린더가 장착되어진 위치에서 주클러치축을 회전시키는데 요구되는 힘은 최대 약 20kg_f이였으며 실린더가 밀어주는 힘은 최대 200kgf에 가깝다. 따라서 유체가 실린더에 밀려들어오는 순간 이미 주클러치를 밀어줄 수 있는 충분한 힘을 갖고 일을 시작함을 알 수 있다. 이와 달리 초기에 힘을 충분히 갖지 못하고 작업을 시작한다면 실린더를 작동하는데 있어서 반응속도가 느려질 것이다.

차. 결론

- 동력전달체계를 파악하고 자율주행을 위한 시스템을 선정하였다.
- SS기의 구조를 파악하고 유압시스템의 장착 위치를 선정하였다.
- 실린더의 크기 및 행정을 선정하기 위해 주클러치와 조향클러치를 움직여 가며 각부위의 변위를 측정후 제어할 위치를 선정하여 구조물을 구축하였다.
- 실린더를 장착하여 제어할 수 있도록 그 위치에서 필요로 하는 힘을 산출하였다.
- 필요로 하는 힘 및 행정을 고려하여 실린더를 제작하였다.
- 안전성을 고려하여 실린더를 제어할 수 있는 솔레노이드 밸브를 선정하였다.
- 실린더와 솔레노이드 밸브를 바탕으로 유압시스템을 구성하였다.
- 구성되어진 유압시스템을 구동하면서 이상유무를 파악하고 성능을 파악하였다.
- 파악한 성능을 토대로 해석 및 분석하였다.

2. 드로틀레버 조작기

드로틀 레버는 12V DC로 구동되는 전동실린더에 부착되어 밀고 당기는 방식으로 구성하였다. 사용된 전동실린더의 주요 제원은 행정 50 mm, 추력 10kg_f, 속도 50 mm/s이며 드로틀 레버가 한 점을 중심으로 원호를 그리면서 이동하기 때문에 전동실린더의 끝을 차체 본체에 핀으로 연결하였다. 그림 3.1.33은 과수방제기에 장착된 전동실린더로 작동되는 드로틀 레버를 나타낸 것이다.

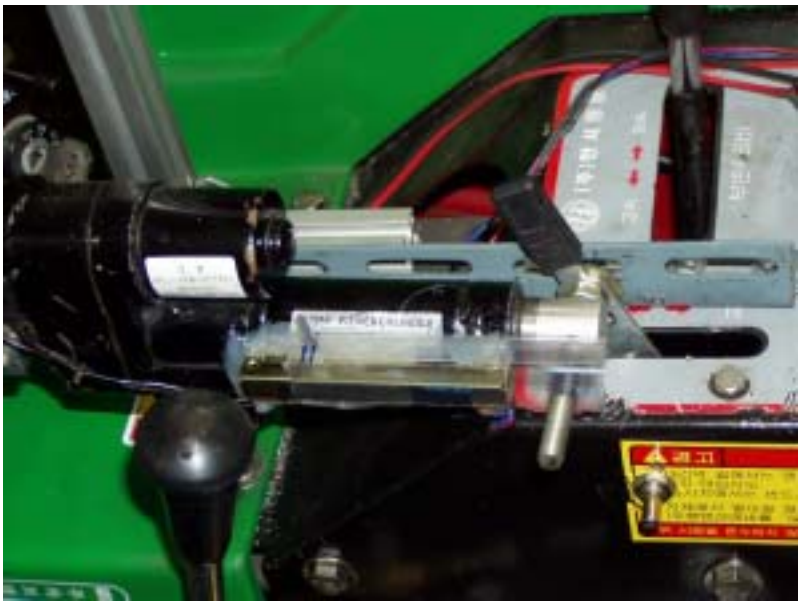


그림 3.1.33 드로틀 레버 자동 조작기.

3. 약액살포 노즐 제어기

본 연구에서 사용한 과수방제기의 약액 살포부(그림 3.1.34(a))는 독립된 엔진으로 작동되며 펌프, 송풍팬과 좌, 우측을 향하여 각각 3개씩의 노즐과 상부를 향한 4개 등 총 10개의 노즐이 장착되어 있다. 운전석 옆에 부착된 수동식 밸브 3개를 이용하여 필요한 부분의 노즐을 on/off할 수 있도록 설계되어 있다.

무인 조작을 가능하게 하기 위하여 지지대에서 노즐을 분리하여 사이에 DC 12V 코일(ZB12 IMQ, Parker, Italy)로 작동하는 솔레노이드 밸브(PM1331N, Parker, Italy)를 장착하였다(그림 3.1.34(b)).

(a) 전체 노즐의 배열

(b) 솔레노이드 밸브 장착

그림 3.1.34 무인조작이 가능한 약액살포부

사용된 솔레노이드 밸브는 N.C. 방식이고 pilot에 의해 작동되는 것으로서 오리피스 직경이 13 mm, 연결부위는 3/8“이며 작동 압력은 10 kg_f/cm^2 이다.

4. 주행속도 측정장치

공시 과수방제기의 주행속도를 측정하기 위하여 궤도의 좌우측 구동 스프로켓 각각에 로타리엔코더(500 pulse/rev)와 근접스위치 센서를 부착하였다. 그림 3.1.35는 주행속도 측정장치가 장착된 모습이다.

그림 3.1.35 주행장치 측정장치의 장착.

제 2 절 주행경로 결정 센서 개발

본 연구에서 과수 방제기의 주행경로의 기준은 오버헤드 가이던스 레일의 지상 투사선이다. 차량이 주행기준선으로부터 벗어나 있으면 그림 3.2.1(a)에서 보는 바와 같이 인장 케이블에 의해 각도가 형성된다. 따라서, 차량의 측면옵셋, x_offset은 식 (3.2.1)과 같이 구할 수 있다.

$$x\_offset = L \cdot sin(\alpha) \tag{3.2.1}$$

여기서, L ; 케이블의 길이.

또한 차량의 방향각, θ는 그림 3.2.1(b)에서와 같이 정의할 수 있으며, 지자기 센서 또는 자이로센서와 같이 별도의 방위각을 측정하는 장치를 사용할 수 있으나, 본 연구에서는 현재의 측면옵셋, x_offset_c과 일정시간 차량이 주행한 후의 측면옵셋, x_offset_n을 측정하여 식 (3.2.2)에서와 같이 삼각함수로 구할 수 있다.

$$\theta = arcsin(\frac{x\_offset_n - x\_offset_c}{D}) \tag{3.2.2}$$

여기서, D ; 일정시간 과수방제기가 이동한 거리.

주행경로 결정 센서는 그림 3.2.1에서 보는 바와 같이 트롤리, 케이블이 자유스럽게 당겨지고 감길 수 있는 케이블 권선장치, 케이블의 경사각도를 측정할 수 있는 경사각도계(Inclinometer) 등으로 구성된다.

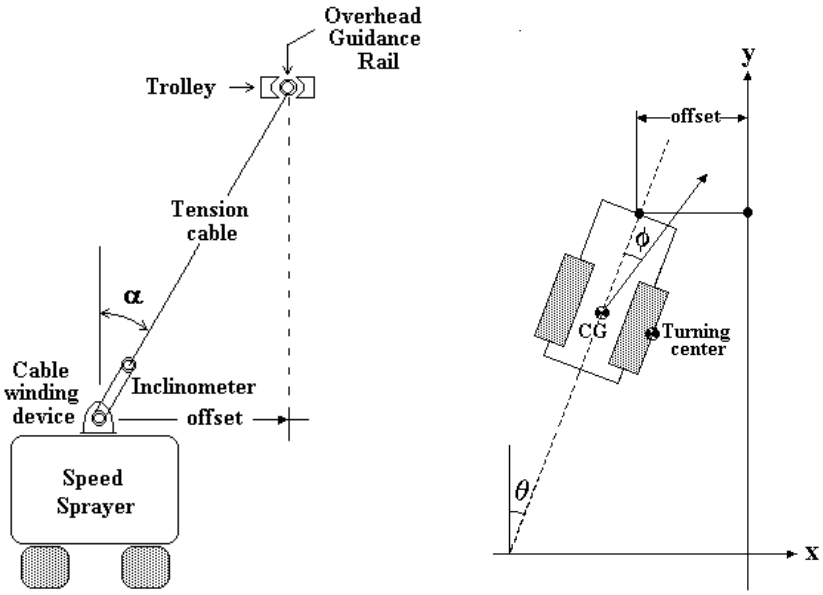
1. 주행경로 결정 센서

가. 센서의 개발

1) 1차 프로토타입 센서

1차 프로토타입 주행경로 결정 센서가 그림 3.2.2에서와 같이 개발되었다. 그림 3.2.2(a)에서와 같이 직경 20 mm의 축에 구멍을 뚫고 0.5m 길이의 전산볼트를 끼워 전산볼트의 경사짐에 따라 2개의 지지 베어링사이에서 축이 자유스럽게 회전되므로 축의 한 끝에 포텐시오메터를 장착하여 측면옵셋에 의한 케이블(막

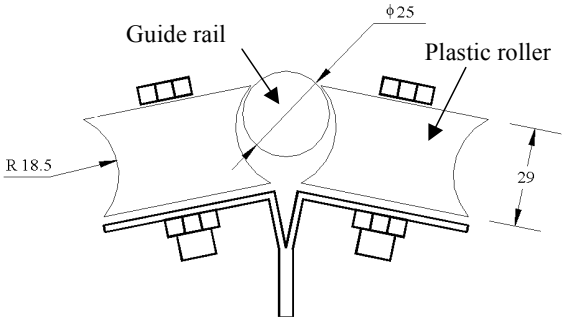
대)의 기울어진 각도를 측정하도록 하였다. 장력 케이블은 길이 1 m정도의 코일 스프링을 사용하여 한 끝은 오버헤드 가이던스 레일 상의 트롤리에 연결하고 다른 한 끝은 경사각 측정장치의 전산볼트에 끼워넣어 케이블의 길이 변화에 대처하고 또한 케이블의 기울어짐에 따라 자연스럽게 축을 회전할 수 있도록 하였다. 트롤리는 직선구간 뿐만 아니라 곡선구간에서도 매끄럽고 슬립이 없이 이동 하기위한 방식으로 고안되었다. 트롤리는 그림 3.2.2(b)에서와 같이 4개의 플라스틱 재질의 롤러로 제작하였으며 직경 25 mm의 오버헤드 가이던스 레일 상에서 어느 정도 좌우로 요동하며 부드럽게 이동할 수 있도록 설계, 제작하였다.



(a) 후면에서 보았을 때 측면옵셑 (b) 위에서 보았을 때 방향각

그림 3.2.1 과수방제기의 자세.

(a) Inclinometer and cable assembly



(b) trolley

그림 3.2.2 1차 프로토타입 주행경로 결정 센서.

주행경로 결정 센서에 대한 성능평가 결과 전산볼트의 무게에 의해 케이블이 처짐으로써 측면 옵셋 측정시 오차가 발생하였고 또한 트롤리는 항상 차량이 진행함에 따라 차량이 트롤리를 끄는 방식으로 이동하게 되므로 차량 진행방향에 대하여 처짐 현상이 나타났다. 그러나, 추후 노지에서의 무인주행 성능 평가 후 적당한 오차 범위내에서 작동이 가능하다고 판단되면 가격 대비 성능을 고려하여 사용 가능할 수도 있다고 판단되었다.

2) 2차 Prototype 센서

1차 프로토타입 센서에서의 문제점을 극복하고자 동력 트롤리를 개발하였다. 트롤리는 양쪽으로 벌린 상태로 유도레일의 중간부분에서도 쉽게 장착할 수 있도록 2개의 piece로 나누어 가위스프링을 넣어 제작하였으며 각 piece에 우레탄 소재의 가이드롤러 2개씩을 설치하고 또한 유도레일을 타고 트롤리를 이동시킬 수 있도록 각 piece에 탄성의 고무 롤러를 장착하여 소형 DC 모터로 구동할 수 있도록 하였다(그림 3.2.3).

길이가 자동 조절되는 케이블 권취 블록은 용수철 태엽을 이용하여 케이블이 적정한 탄성을 갖으며 자유롭게 풀림과 당김 기능을 할 수 있도록 고안되었다(그림 3.2.4). 태엽 스프링은 두께 0.5mm, 폭 10mm, 길이 2.5m의 스프링판을 10회 이상 감아서 엔지니어링 플라스틱으로 가공한 케이스에 집어넣고 이것이 케이블을 감아주는 원통을 돌릴 수 있도록 하였다. 여기서, 회전 중심 축에 포텐시오메터를 장착하여 케이블의 길이를 측정할 수 있도록 하였다.

케이블이 이루는 각도를 측정하기 위한 경사각도계(Inclinometer)는 그림 3.2.5에서 보는 바와 같이 개발하였다. 중심에 뚫린 구멍으로 케이블을 통과시켜 케이블에 따라 자유로이 움직일 수 있는 막대를 2개의 직각으로 교차되어 반원 모양을 따라 적정한 각도 범위내의 공간상에서 어느 방향으로도 이동가능하도록 장치를 구성하였다. 각 반원의 중심축에 고정밀도의 포텐시오메터(진흥물산)를 장착함으로써 각 방향에서 이동한 각도를 분리 측정할 수 있도록 하였다. 즉, 한 축은 측면옵셋에 의해 야기되는 각도를 측정하며 다른 한 축은 트롤리의 차량에 대한 상대 위치(앞 서거나 처지는 경우)에 의한 각도를 측정할 수 있도록 하였다. 이 경사각도계는 그림 3.2.4의 케이블 권취블록 위에 장착된다.

그림 3.2.6은 오버헤드 가이던스 레일 상에 설치된 트롤리와 과수방제기 상에 설치된 케이블 권취블록, 각도 측정 장치가 조립된 모습이다.

한편, 새로 고안된 동력 트롤리를 장시간 사용하다보니 DC모터에 의해 구동되는 탄성 고무 롤러가 닳아서 정상적으로 트롤리를 구동하지 못하는 문제가 발생하였으며 또한 두 piece의 트롤리를 적정한 장력으로 조여서 오버헤드 가이던스 레일 상에서 트롤리의 이동을 원활하게 해 주는 역할은 하는 가위스프링의 탄성이 떨어져 트롤리가 레일 상에서 이탈하는 문제점이 노출되었다. 또한 1차 프로토타입 센서에서의 트롤리에 비해 2차에서 고안한 트롤리의 길이가 길어지면서 작은 곡률반경의 곡선 구간을 부드럽게 돌아가지 못하는 문제점이 노출되어 3차 프로토타입 센서로 수정 개발하였다.

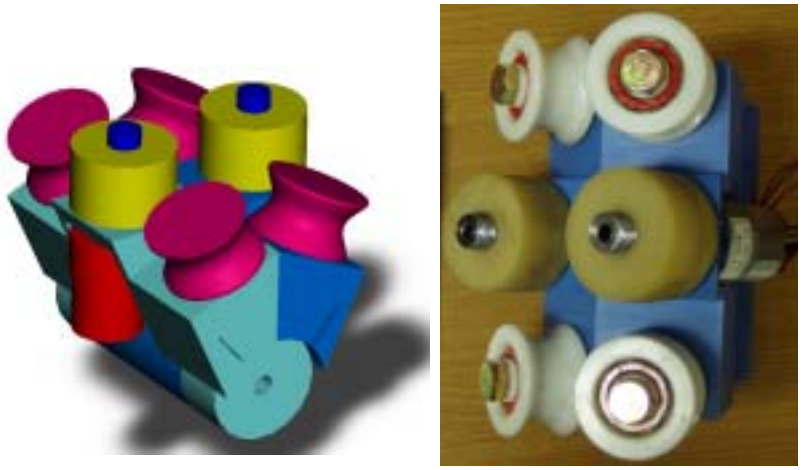


그림 3.2.3 동력 트롤리의 모식도 및 실제 모습

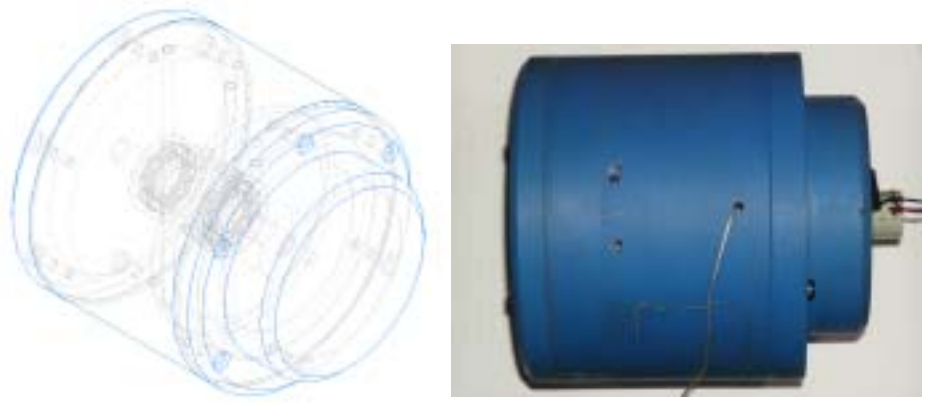


그림 3.2.4 케이블 권취장치의 모식도와 실제 모습.

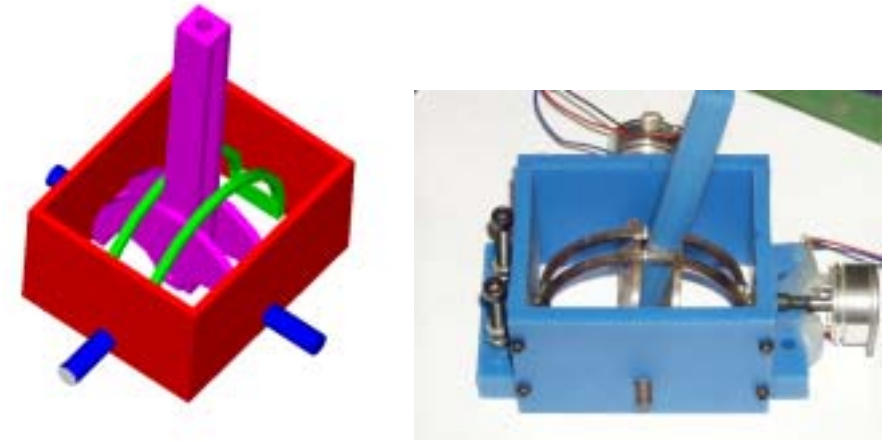


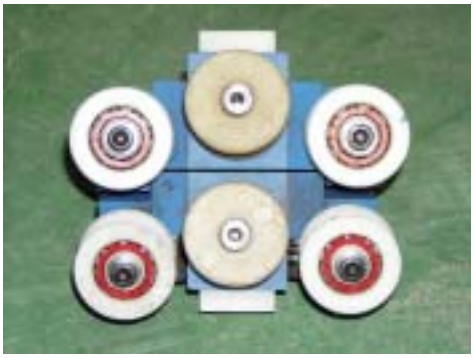
그림 3.2.5 각도 측정장치의 모식도와 실제 모습.

그림 3.2.6 조립된 주행경로 결정센서의 모습(2차 프로토타입).

3) 3차 프로토타입 센서

먼저 트롤리로부터 가위스프링을 제거하고 트롤리를 길이방향으로 3조각으로 분리하고 전체 길이를 최대한 줄여서 곡률반경이 적은 경우에도 트롤 리가 곡선 구간을 부드럽게 이동할 수 있도록 하였다(그림 3.2.7 (a)). 분리된 각 쌍에 6각 볼트를 설치하여 볼트의 조임 여부에 따라 3 조각이 유도레일에서 독립적으로 가이드휠 쌍의 간격이 조절될 수 있도록 하였다(그림 3.2.7 (b)). DC모터에 의해 구동되는 고무 휠 부분에는 볼트에 압축스프링을 추가하여 고무휠의 마모에 관계없이 항상 고무휠이 유도레일에 밀착되게 함으로써 안전한 트롤리를 개발할 수 있었다. 그림 3.2.7은 3차로 수정된 트롤리의 외관도이다.

그림 3.2.8은 3차 프로토타입 트롤리가 오버헤드가이던스 레일에 장착된 모습을 나타낸 것이다. 그림 3.2.9는 주행경로 결정 센서가 과수방제기에 장착된 모습이다.



(a) (b)

그림 3.2.7 3차 프로토타입 센서에서의 수정된 트롤리.

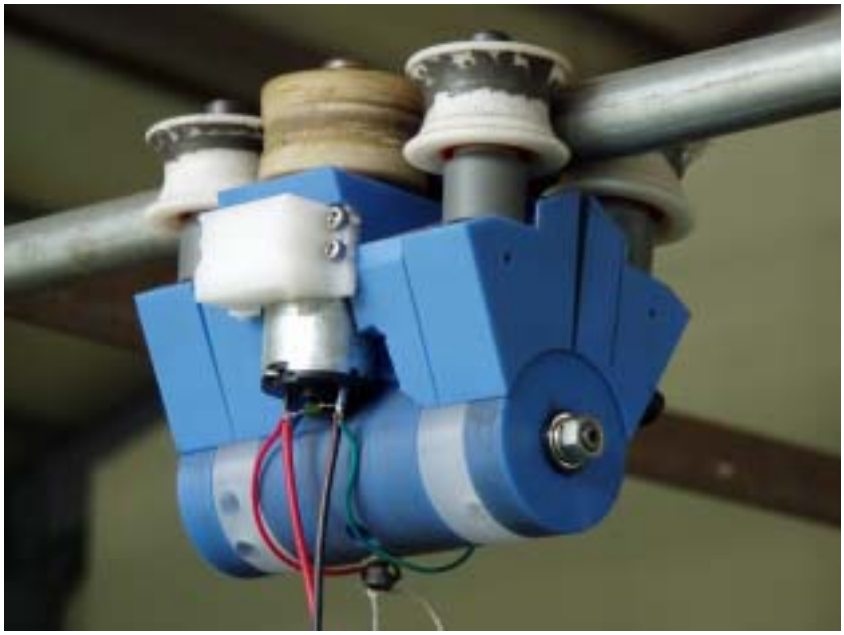


그림 3.2.8 오버헤드 가이던스 레일에 장착된 트롤리.

그림 3.2.9 과수방제기의 무인주행을 위한 주행경로 결정센서.

나. 각도 및 케이블 길이 측정 장치에 대한 측도 설정

케이블의 각도 및 길이는 모두 로타리 타입 포텐시오메터에 의해 측정된다. 경사각도계의 각도 측정에 사용된 포텐시오메터는 1회전용이며, 케이블 권취 장치의 포텐시오메터는 10회전용을 사용하였다.

입력회로로 전압 분배 회로를 구성하여 마이크로프로세서의 A/D 핀에 입력한 전압 변화를 디지털 값으로 나타내었다. 각 센서부의 측도 설정을 위하여 오버헤드 가이던스 레일에 트롤리를 장착하고 케이블 권취블록을 이동식 카트에 설치한 후 한 축을 90도로 고정시킨 상태에서 고정축과 직교하는 방향으로 카트를 움직여 케이블의 각도를 제어하면서 A/D 변환된 데이터를 수집하였다.

측도 설정식을 구하기 위하여 측면옵셑을 측정하는 부분의 각도, x_angle에 대한 AD 변환값과의 선형회귀식을 구한 결과,

$$x\_angle = -0.0183 \cdot AD + 36.487 \ , \ [\mathrm{deg}] \qquad (3.2.3)$$

여기서, AD ; AD변환된 디지털 값이며 12bit A/D 변환기로 변환한 것이다.

이때 선형회귀식의 r^2는 0.9991이었다. 측면옵셑각이 -인 경우는 차량이 주행기준선의 왼쪽에 있는 경우를 나타낸다.

같은 방법으로 트롤리 진행방향에 대한 각도, y_angle에 대한 측도 설정식은,

$$y\_angle = 0.0188 \cdot AD - 40.224 \ , \ [\mathrm{deg}] \tag{3.2.4}$$

이 되며, 이때 선형회귀식의 r^2는 0.9991이었다. y_angle이 -인 경우는 트롤리가 차체에 비하여 쳐져있는 상태를 나타낸 것이다.

한편, 케이블의 길이, z_length에 대한 측도 설정치는 다음과 같다.

$$z\_length = 0.32 \cdot AD + 0.802 \ , \ [\mathrm{cm}] \tag{3.2.5}$$

이때 선형회귀식의 선형회귀식의 r^2는 0.9991이었다.

다. 지면의 경사에 의한 경사각도계(Inclinometer) 측정값 보정

지면의 경사도를 측정하기 위하여 2축 경사계(DM-1, DAS Technology, Korea)를 사용하였다. 이것은 경사도에 따라 롤링 및 피칭에 대한 아날로그전압을 출력하므로 마이크로프로세서의 A/D 변환 기능을 사용할 수 있다. 측도 설정결과 롤링과 피칭에 대한 변환식은 각각 식 (3.2.6)과 식 (3.2.7)과 같다.

$$roll = 0.0348 \cdot AD - 70.922 \ , \ [\mathrm{deg}] \tag{3.2.6}$$

$$pitch = 0.036 \cdot AD - 74.273 \ , \ [\mathrm{deg}] \tag{3.2.7}$$

이때, 각각의 r^2 는 0.997과 0.9991 이었다. 한편, -롤링 각은 차체가 진행방향에 대하여 좌측으로 기울어진 상태를 의미하며, -피칭 각은 차체가 앞으로 기운 상태를 나타낸다.

Inclinometer는 지면의 경사와 관계없이 차체 수평면에 대하여 연직의 방향을 0°로 지시한다. 그러나, 측면옵셑을 결정하는 각, α는 트롤리에서의 연직하방선이 기준이므로 지면의 경사도 만큼 잘못된 값을 측정하게 된다. 그림 3.2.9는 차량의 측면옵셑과 지면경사도를 달리했을 경우 Inclinometer에서 측정되는 각도들을 시

뮬레이션한 것이다. 그림에서 ε은 당해 조건에서 Inclinometer에 의해 측정된 값으로 Inclinometer의 막대가 우측으로 기우는 것을 - 각으로 보았을 때, 즉 차량은 주행기준선의 좌측에 위치할 때 - 측면옵셋을 나타내는 것으로, 차량의 위치 여부에 관계없이 보정된 각도, α는 식 (3.2.8)에서와 같이 구할 수 있다.

$$\alpha = \epsilon - \delta \tag{3.2.8}$$

여기서, δ ; 지면경사도로서 -는 차체가 왼쪽으로 기운 것을 의미한다.
사용된 2축 경사계는 ±30°의 범위내에서 2축 방향으로 경사도 측정이 가능하며 피치축의 경우 교정각도 0°, -20°, +20°에 대해 각각 2500 mV, 1800 mV 및 3200 mV를 출력하며 롤의 경우에도 각각 2501 mV, 1801 mV 및 3200 mV를 출력한다. 두 각의 감도는 35.0 mV/°이었다.

라. 측면옵셋과 방향각 측정 알고리즘

측면옵셋은 앞의 나. 항에서 구한 케이블의 현재 길이, z_length와 다. 항의 지면 경사도에 따른 측면옵셋각도, x_angle의 보정치를 이용하여 식 (3.2.1)로부터 구할 수 있다.

방향각은 식 (3.2.2)에서 보는 바와 같이 일단 차량이 제어주기 동안 진행하고 난 후에 계산이 가능하다. 주어진 시간 동안의 이동거리는 좌우측 구동륜의 평균 회전속도로부터 궤도의 선속도를 구하여 계산되어진다. 즉, 이동거리, D는

$$\begin{aligned} D &= v_s \cdot \Delta t \\ &= Rw \cdot \Delta t \\ &= R\frac{2\pi}{60} N \frac{\Delta t}{1000} \end{aligned} \tag{3.2.8}$$

여기서, R ; 궤도 구동륜 스프로켓의 반경, 85 [mm], N ; 궤도의 회전속도 [rpm], Δt ; 제어 주기 [ms]이다.

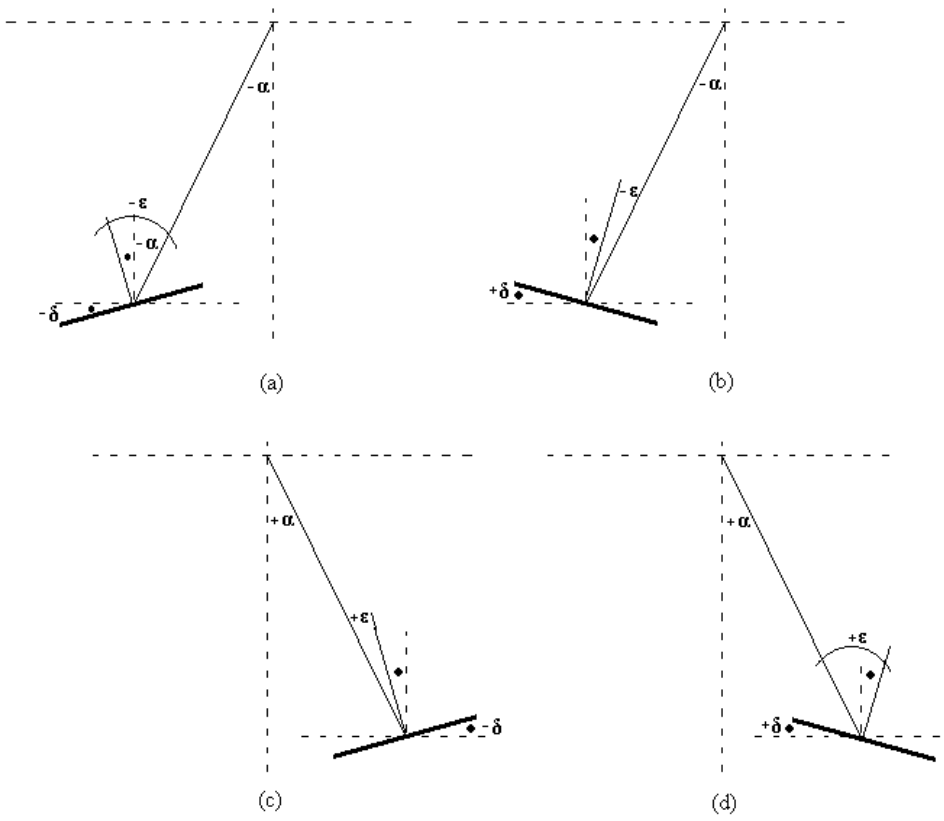


그림 3.2.10 지면의 경사도에 따른 차량의 측면옵셀 측정각.

한편, 제어 프로그램의 작성을 용이하게 수행하기 위하여 방향각을 나타낼 때 일정한 규칙이 필요하다. 즉, 차량이 주행기준선인 오버헤드 가이던스 레일을 기준으로 오른쪽을 지향할 때를 + 방향각, 왼쪽을 지향할 때를 - 방향각으로 삼는다면, 차량의 측면옵셀의 부호와 관계없이 방향각, θ는 식 (3.2.2)에서 구할 수 있다.

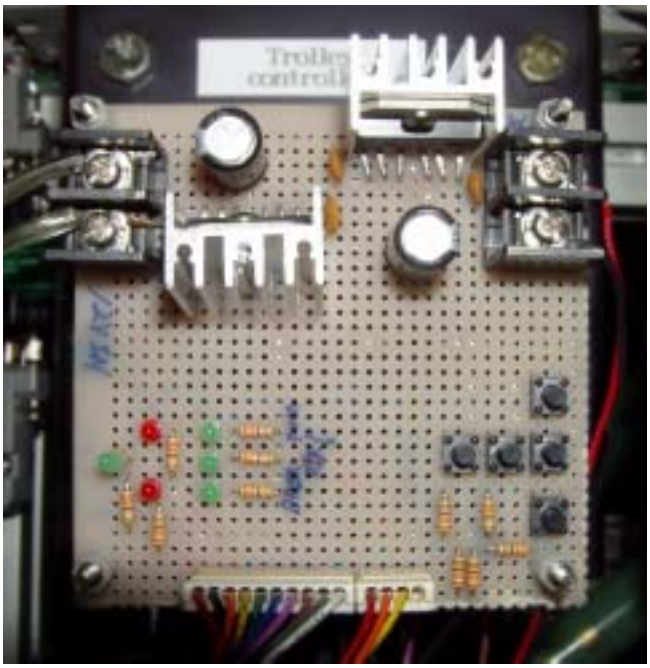
바. 동력 트롤리의 위치 및 속도 제어기

1차 프로토타입으로 개발하였던 무동력 트롤리가 과수방제기가 진행함에 따라 항상 뒤쳐져서 따라오게 되므로 현재의 차량 위치에서의 측면옵셀을 정확히

측정하기가 어려웠기 때문에 12V DC 전원용 소형 모터로 트롤리를 구동하는 전동 트롤리를 개발하였다.

트롤리는 차량의 주행속도와 동기하여 이동해야 하므로 차량이 정지해 있는 동안은 작동하지 말아야 하며 차량이 이동을 시작하면 전 항에서 기술한 Inclinometer의 각도 측정 레버가 차량 진행 방향에 대하여 뒤쪽으로 제껴지기 시작하여 미리 설정한 데드밴드 영역을 벗어나면 트롤리는 전진한다. 트롤리의 전진속도가 빠르면 Inclinometer의 각도 측정 레버가 앞쪽으로 제껴지고 데드밴드 영역을 지나치면 트롤리가 후진하는 방식으로 차량의 주행속도와 동기시키는 제어 알고리즘을 개발하였다. 데드밴드는 뒷쪽으로 제껴질 때 1°, 앞쪽으로 제껴질 때 -5°로 셑팅 되었다.

그림 3.2.11은 이 제어기의 회로도이며, 모터의 정역전을 수시로 수행하므로 LMD 18200 DC 모터 전용 구동 칩을 사용하였다. LMD 18200은 3A H-브릿지로 설계되었으며 3개의 제어 신호(정지, 회전방향, 회전속도)로 DC모터의 위치 및 속도제어를 할 수 있다. 본 연구에서는 LMD 18200 칩 두 개를 병렬로 연결하여 연속적으로 6A까지 DC모터에 전류 공급이 가능하도록 하였다. 그림 3.2.12는 트롤리 구동 모터의 제어 알고리즘의 흐름도를 나타낸 것이다.



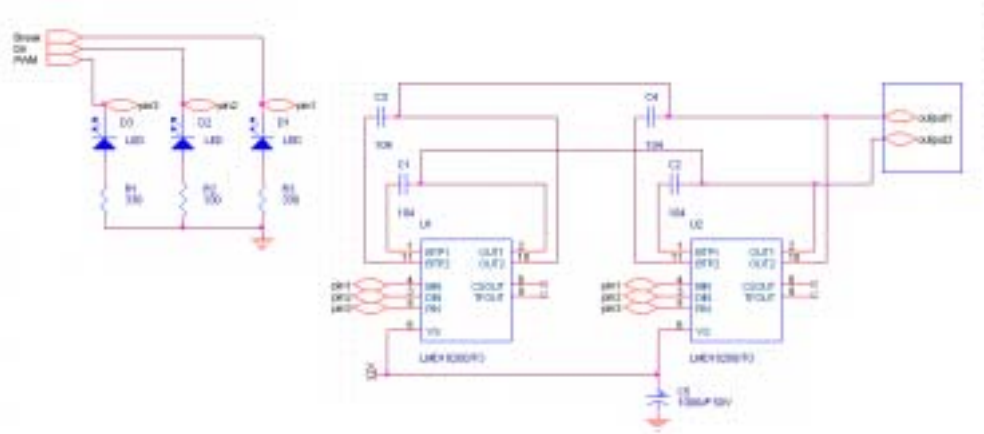


그림 3.2.11 트롤리 제어기의 인터페이스 회로.

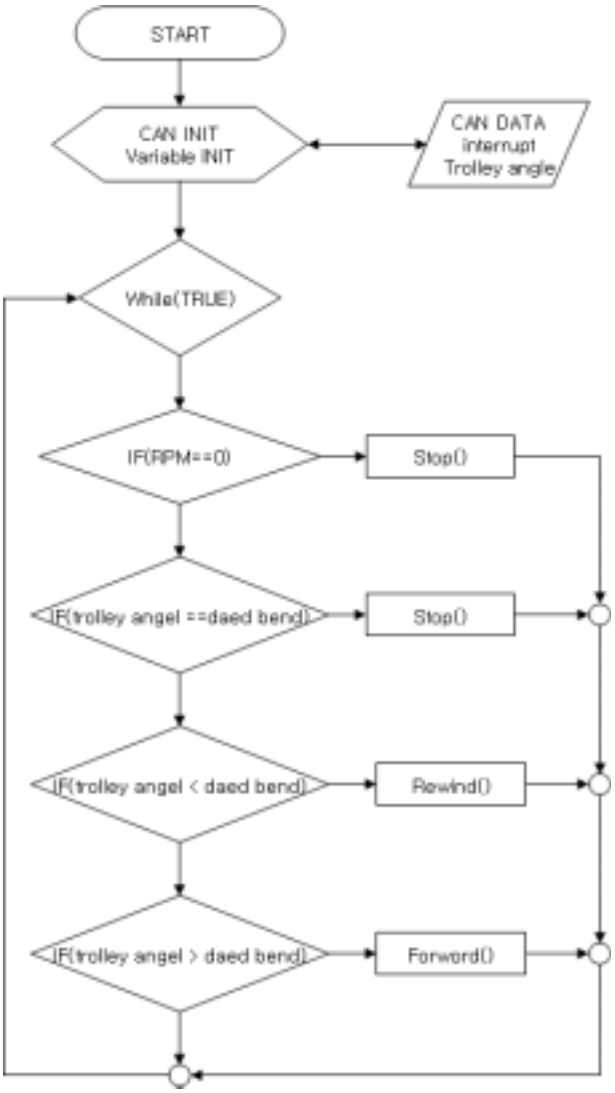


그림 3.2.12 트롤리 구동 제어 흐름도.

2. 오버헤드 가이던스 레일

오버헤드 가이던스 레일은 무게 2 kg의 트롤리만 지지하면 충분하므로 시중에서 쉽게 구할 수 있는 직경 25 mm짜리 비닐하우스용 파이프를 사용하였다. 오버헤드 가이던스 레일을 과수원의 기존 구조물에 설치하기 위하여 그림 3.2.13에서 보는 바와 같이 폭 70 mm, 길이 200 mm, 두께 1 mm의 철편을 가이던스 레일에 용접하여 측면옵셋에 따라 트롤리가 좌우로 약간씩이나마 요동할 수 있도록 제작하였다. 레일의 연결핀은 비닐하우스 건축시 사용되는 기성 제품을 사용하였으며 경우에 따라서 연결 부위에 최소한의 그라인딩 작업을 수행하였다.

하나의 과수열이 끝나고 인접 과수열로 이동하기 위한 곡선 구간의 가이던스 레일은 파이프를 곡률 반경 1 m 짜리 1/4 원으로 구부리고 2개의 조각 사이에 적정한 길이의 직선구간을 삽입하여 과수열의 간격에 맞추었다.

실험실내에 설치한 오버헤드 가이던스 레일은 별도의지지 구조물을 설치한 후 그것을 지지대로 하여 레일을 설치하였다. 직선구간의 총 길이는 8 m이며 두 직선구간 사이의 간격은 3 m 이다. 그림 3.2.14는 실험실에 설치한 오버헤드 가이던스 레일의 설치 모습이다.

그림 3.2.15와 3.2.16은 과수열의 길이 40 m, 과수열간 간격 6m 인 강원대학교 부속농장 배과수원에 오버헤드 가이던스 레일을 설치한 모습이다. 직선 구간의 길이는 약 40 m 이며, 과수열 끝의 선회 구간은 곡률반경 1m 짜리 1/4원 곡선 파이프 2개와 직선 파이프 4 m짜리 하나를 연결하여 구성하였다. 가이던스 레일을 과수원 구조물에 지지하기 위하여 비닐하우스에서 사용하는 유도파이프 거치대를 레일에 용접한 후 구조물에 고정하였다. 실제 과수원에서의 성능 평가를 위하여 과수열 2줄에만 가이던스 레일을 설치하였다. 한편, 곡선 구간을 설치하는 부분에는 기존의 구조물이 없기 때문에 여러 개의 지주대를 설치한 후 곡선 구간 가이던스 레일을 고정하였다.

그림 3.2.13 오버헤드 가이던스 레일을 기존 구조물에 연결한 모습.

그림 3.2.14 실험실에 설치한 오버헤드 가이던스 레일의 전경.

그림 3.2.15 강원대학교 농장 과수원에 설치한 오버헤드 가이던스 레일(직선구간).

그림 3.2.16 강원대학교 농장 괴수원에 설치한 곡선 구간 오버헤드 가이던스 레일과 지지대.

제 3 절 CAN 버스 시스템의 구조분석 및 설계

1. CAN 특성 및 작동 원리

가. 특성

CAN은 실시간 어플리케이션을 위한 2선식 시리얼 통신규격이며 최대 1 M bit/sec의 속도로 작동하며 높은 신뢰성을 갖고 있다. CAN은 자동차의 전자제어 시스템 분야에서 수많은 ECU와 센서류를 효율적으로 접속하여 대량의 배선을 감소시키기 위하여 독일의 로버트 보쉬 GmbH 사에 의해 개발된 것으로 현재에는 ISO 11898(고속통신용) 및 ISO 11519(저속통신용)으로 국제규격화되어 있다. 그것은 분산배치된 스위치와 센서로부터 제어기기에 이르는 다양한 디바이스를 접속하는 것으로 단말(node)의 접속이 용이하여 시스템 구성의 변경도 유연하게 대응되는 등 우수한 확장성을 실현할 수 있다. 즉, CAN은 분산제어시스템으로서 여러 센서나 조작기 등을 각각 또는 수 개씩을 하나의 마이크로콘트롤러에 연결하여 필요한 데이터 통신을 하게 한다. 이때 마이크로콘트롤러를 소위 하나의 노드라고 할 수 있는데, 한 노드에서 모든 노드로 데이터를 전송하거나(broadcasting mode), 한 노드에서 특정의 노드로의 데이터 전송은 물론 한 노드에서 다른 노드의 데이터를 요청하면 책임있는 노드에서 데이터를 전송해 주는 방식으로 데이터 통신이 이루어진다. 즉, 하나의 2선식 버스라인에 물려있는 모든 노드들은 CAN protocol에 의해 버스에의 악세스가 결정되는데, 여러 개의 노드에서 동시에 데이터를 전송하고자 한다면 가장 우선도가 높은 노드가 버스 악세스에 대한 우선적 권리를 갖으며 나머지 노드들은 그 노드가 전송을 완료할 때까지 기다렸다가 데이터를 재전송하게 된다. 즉, CAN protocol은 message-oriented protocol로서 메시지를 받을 노드의 주소로 데이터를 전송하는 것이 아니라 전송되는 메시지의 Message Identifier에 의해 순전히 수신기 입장에서 받아들일지 말지를 결정하는 방식이다.

나. 작동원리

1) 메시지 송신

CAN 버스는 기본적으로 “wired-AND" 방식으로 작동하므로 버스상에 0V(우세 비트)와 5V(열세 비트)가 동시에 나타나면 0V가 된다. 즉, 그림 3.3.1에서와

같이 3개의 노드에서 동시에 버스에 악세스를 시도한다면 bit-wise arbitration에 의해 앞부분에 우세 비트를 많이 보유하고 있는 메시지가 최종적으로 버스를 차지하게 된다. 다른 메시지들은 버스가 free하게 되는 지를 계속 감시하다가 노드 3의 데이터 전송이 완료되고 난 후 자기들끼리의 경쟁에 의해 마찬가지 방식으로 버스에 악세스하게 된다. 버스는 열세 비트가 11개 이상 계속 나타나면 free한 상태를 나타내는 것이다. 따라서, CAN protocol에서는 메시지가 파괴되지 않은 상태로 우선순위에 따라 순차적으로 데이터의 송신이 가능하다.

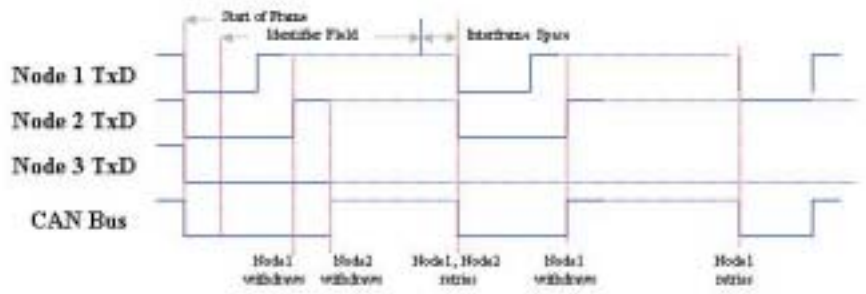


그림 3.3.1 CAN에서의 메시지 중재 원리.

2) 프레임 포맷

CAN protocol에서는 Frame이라 하는 데이터열 구조에 Message ID와 최대 8 byte 크기의 데이터 등을 실어 버스에 악세스하게 된다. 그림 3.3.2에서와 같이, 프레임은 한 bit의 우세비트(Start Of Frame)로 시작되어 12 개 또는 32개의 중재용 bit, 후속되는 데이터의 크기를 알려주는 6 개의 control bit의 순으로 구성되며, 그 후에 보내고자 하는 데이터가 따라온다. 데이터 영역 다음에는 이 프레임이 데이터인지 아니면 다른 노드에 데이터를 요청하는 것인지를 표시하는 RTR bit와 수신된 메시지가 정상적인지를 체크할 수 있는 15 bit의 CRC 영역, 수신노드 쪽에서 수신여부를 확인해주는 ACK 영역, 프레임의 끝을 알리는 7개의 열세비트로 구성된 EOF 영역과 3 개 이하의 열세비트로 구성된 Intermission 영역으로 구성된다.

프레임은 11개의 bit로 ID를 구성하는 Base 포맷(CAN 2.0A)과 29개의 bit로 id를 구성하는 Extended 포맷(CAN 2.0B)이 있으며 CAN 2.0A에서는 모두 2048개(2^{11})의 ID를 사용하나 CAN 2.0B에서는 2^{29}개의 id(536,870,912 개)를 부여할 수 있다. V2.0A로 컴파일하는 경우에는 단지 CAN 2.0A 포맷만이 사용가능하며 V2.0B로 컴파일할 때는 두 가지 포맷을 한 시스템에서 동시에 사용가능하나 CAN 2.0A 포맷이 우선순위가 높다.

이러한 프레임은 모두 4가지 형태가 있는데, Data Frame은 하나의 송신기(transmitter)로부터 하나 또는 수 개의 수신기(receiver)로 데이터를 보내는데 사용된다. Remote Frame은 한 수신기가 다른 노드의 Data Frame의 전송을 요청할 때 사용된다. Error Frame은 버스 노드에 의해 감지된 에러신호를 보내거나 프레임을 파괴하는데 사용되며, Overload Frame은 선행 및 후속 데이터간 또는 Remote Request Frame 간에 추가적인 지연을 제공하거나 구체적인 에러 신호를 내 보내는데 사용된다. Error Frame과 Overload Frame은 그림 3.3.2와는 다른 구조로 되어 있으며 통신 상 에러가 발생하거나 통신 중 제거할 필요가 이는 Data Frame을 파괴한다.

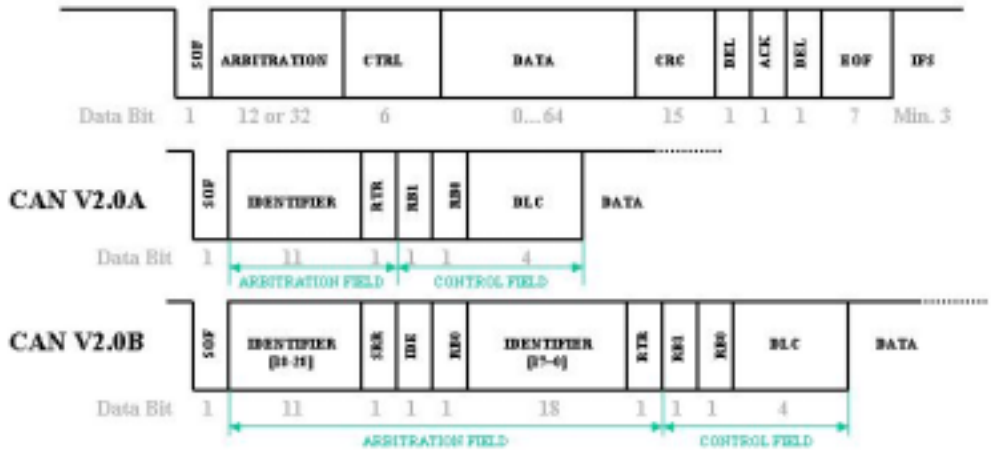


그림 3.3.2 데이터 및 원격요청 메시지의 프레임 포맷.

3) BasicCAN 과 FullCAN

CAN에서 송, 수신을 위해 들어오고 나가는 데이터를 처리하는 방식에 따라 BasicCAN모드와 FullCAN모드로 나뉘어 진다. BasicCAN은 CAN 메시지 전송과 수신을 처리하기 위하여 호스트 cpu를 필요로 하며 10 byte 크기의 송신버퍼 하나와 같은 크기의 수신 버퍼 하나만을 갖는다. 수신버퍼는 FIFO 큐에 대하여 윈도우로서 보여지는데, 따라서 단지 처음 들어온 메시지만 host processor에 보여지게 된다. 특히 수신버퍼는 별도의 가수신버퍼(“shadow buffer")를 두어 프로세서가 수신버퍼로부터 한 메시지를 읽고 있는 도중에 들어오는 메시지를 임시로 저장하게 한다. 즉 프로세서는 동시에 두 개의 메시지 데이터 수신 가능하며, 세 번째 메시지를 수신하기 전에 최소한 하나의 수신된 메시지를 읽어야만 세 번째 메시지의 수신이 가능하며 그렇지 못하면 data가 overrun되어 메시지를 수신하지 못하게 된다. FullCAN 모드는 송, 수신에 보다 유연성, 확장성을 부여해 주기 위하여 보다 많은 수의 송, 수신 버퍼를 다룰 수 있는 방식으로 고성능의 CAN 콘트롤러에서 지원된다. 예를들면, T89C51CC01과 같은 CAN 콘트롤러는 20 byte로 구성된 15개의 channel을 Receiver, Transmitter 및 Receiver buffer로 설정하여 사용 가능하며, 채널 1개당 8 byte의 cyclic data shift register와 4 byte의 ID Tag register, 4 byte의 ID Mask register, 2 byte의 Status/ Control register, 2 byte의 Time Stamp register 등의 총 20 byte로 구성되어 보다 용이하고 많은 수의 데이터의 송 수신이 가능하게 된다. 특히, 메시지 전송, 메시지 수신 그리고 최대 16개 메시지들의 저장을 CAN controller에 맡김으로서 cpu는 자유롭게 다른 작업을 처리할 수 있다는 장점을 갖는다. cpu는 정보가 필요할 때 CAN controller를 참조하면 된다.

4) 메시지 수신

네트웍상의 각 노드들은 bus상에 있는 프레임에 대하여 자신에게 관련되는지의 여부를 감시하여 메시지를 수신하거나 또는 거부한다. 모든 메시지는 수신버퍼에 저장되기 전에 수신필터(acceptance filtering)를 통과해야 한다. 수신측 노드에서 적절한 필터링을 통과한 메시지만을 수신하게 되는 원리이다. 따라서, 프로세서가 시작하기 전에 2개의 레지스터(Acceptance Code 또는 ID Tag, Acceptance Mask 또는 ID Mask)가 반드시 세팅되어야 한다. CAN controller마다 메시지 수신에 관한 규정이 다르긴 하지만 Atmel 계열로 CAN 2.0A Base 포

맷을 사용하는 경우에는, 어떤 메시지가 수신 가능하려면 반드시 11 bit의 메시지 id 중 8개의 MSB 비트들(ID10 – ID3)이 Acceptance Code register의 데이터와 일치하여야 한다. Acceptance Mask register는 어떤 bit가 실제로 메시지 필터링에 관련되는 지를 결정하는데, 즉 그 bit와 수신된 메시지 id에서의 bit 값은 반드시 일치하여야 한다. 여기서 관련된 bit들은 '0'으로 세팅되며 무관련의 것들은 '1'로 세팅된다. 필터링은 메시지 id의 상위 8 개의 bit에 의해 이루어지며 하위 3개의 bit (ID2-ID0)는 관계가 없으므로 콘트롤러는 하나의 메시지 뿐 아니라 연속한 id를 갖는 8개의 메시지를 수신하게 된다. 그래서, 만일 두 acceptance register가 모두 0을 갖는다면, id 0부터 id 7까지의 메시지가 콘트롤러의 수신 버퍼 에 전달된다. 한편, Acceptance Mask register의 ID3 bit를 1로 세팅하면, 즉 ID3가 무관하다고 선언되면 ID0부터 ID15까지의 메시지가 수신된다. 즉, 이와 같이 레지스터가 세팅되면 0b00000000000 – 0b00000001111까지 16개와 그 ID가 일치하는 메시지는 수신이 가능하게 된다 (그림 3.3.3).

그림 3.3.3 BasicCAN 수신 필터링 원리(예).

5) 통신 에러

CAN 버스에서는 다음과 같이 5가지 에러검출 메커니즘이 있다.

ⓐ Message 수준에서

i) CRC (Cyclic Redundancy Checks): 계산식에 따라 송신측에 부과한 data를 수신측에서 해석하여 수신 data에 오류가 있는지 확인한다.

ii) Frame Checks :

iii) Acknowledgment Error Checks : 송신측이 송신한 메시지가 타 노드에 인식되지 않았을 경우, 에러로 감지한다.

ⓑ 비트 수준에서

통신회선상의 신호레벨과 송신레벨에 이상이 발생한 경우, ERROR를 검지한다.

iv) Bit Monitoring

v) Bit Stuffing : 연속적으로 5개 bit 이상이 동일한 값으로 송신될 때 6번째 bit에 값이 반대인 stuffing bit을 추가 송신할 때 송수신 시 발생할 에러를 감지한다.

통신 중에 에러가 발생하면, 그 횟수가 Error Counts에 저장되는데, 받기에러는 가중치 1, 보내기 에러는 가중치 8로 레지스터에 저장한다. 한편 에러 모드는 다음과 같이 구분되는데.

i) Normal Mode : 에러의 계수가 0인 경우로서 정상 통신 상태이다.

ii) Error Active Mode : 만약에 에러의 개수가 0보다 크면 node는 Error Active Mode로 들어간다. Error Active Mode는 여전히 Full Function으로 작동되지만, 경계조건에 있다. 다음 들어올 메시지에 에러가 없는 경우 에러 카운트 레지스터의 값이 '1' 줄어든다. 만약에 계속 에러가 발생하지 않으면 에러 카운트 레지스터의 값이 0이 되고 Error Active node 는 Normal Mode로 바뀐다.

iii) Bus-Off Mode : 에러 카운트 레지스터의 값이 255를 초과할 경우 "Bus-Off mode"로 들어가면서 해당 노드는 저절로 오프라인이 되어 나머지 CAN 통신을 안전하게 해 준다. Bus-Off Mode의 노드는 재 초기화 과정을 거쳐 Error Active mode로 돌아 올 수 있다. 그림 3.3.4는 T89C51CC01에서의 에러모드의 작동 원리를 나타낸 것이다. 에러 count가 128회 이하인 경우는 Error Active Mode로서 active error frame을 포함한 모든 프레임을 보낼 수 있다. count가 128을 초과하여 256이 될 때는 Error Passive Mode가 되는데 이때는 active error frame을 제외한 모든 프레임을 보낼 수 있다. count가 256을 초과하면 Bus-Off Mode가 되

며 해당 노드는 버스로부터 분리된다(Off-Line).

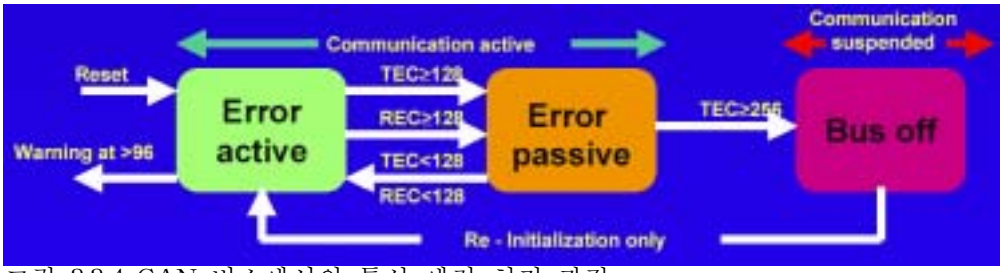


그림 3.3.4 CAN 버스에서의 통신 에러 처리 과정.

다. Higher Layer Protocol

CAN은 데이터 통신의 기본모델인 7-Layer ISO/OSI Reference Model에서 기본적으로 최하위계층인 Physical Layer와 Data Link Layer를 규정하고 있다. 즉, 추구하는 데이터 통신 시스템이 독립적이고 지역적으로 사용한다면 이 두 계층에서만으로도 충분히 네트워크 구성 및 통신이 가능하나 다른 제어기와의 호환성을 증대시키는 국제규격을 따르고자 한다면 미국자동차학회(SAE)에서 정의한 J1939(Serial Control and Communications Vehicle network)나 ISO 11783(Serial Control and Communications Tractors and machinery for agriculture equipment Network)과 같은 고위 계층에서 정의되는 protocol을 사용하는 것이 바람직하다. 일반적으로 작동하는 원리는 CAN의 특성 및 작동 원리에서 규명한 바와 동일하나 데이터의 구조 측면에서 차이가 있다. 따라서, 향후 ISO 11783을 추종하여 보다 일반적으로 표준적인 통신규약을 사용하기 위하여 계속적인 연구가 필요하다.

1) 메시지 프레임의 구조

ISO 11783 또는 J 1939에서의 메시지 프레임은 CAN 2.0B의 확장 포맷을 기본으로 하여 고유의 방식으로 ID 영역을 구분하여 사용한다(그림 3.3.5). 왜냐하면 트랙터나 일반차량의 경우 수 많은 센서 또는 조작기를 사용하는데, 이들을 그룹으로 나누어 관리하는 것이 편리하기 때문이다.

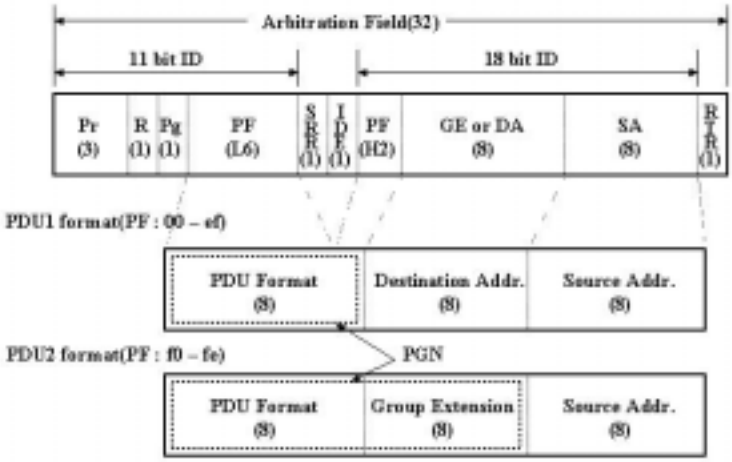


그림 3.3.5 ISO11783 또는 SAE J1939.에서 CAN 식별자의 구조.

이 메시지의 구조는 크게 메시지의 Priority(처음 3 bit), Parameter Group Number(18 bit), 및 Source Address(8 bit)로 되어 있다. 그림 3.3.5에서 보는 바와 같이, Process Data Unit 포맷 필드(PF) 값에 따라 PDU1 포맷과 PDU2 포맷으로 구분하여 사용할 수 있는데 전자는 특정 노드로 데이터를 전송하는 방식(node-oriented transmission)이며 후자는 방송하는 것과 같이 하나의 노드에서 모든 노드로 메시지를 전송하고 관련된 노드에서만 같은 수신하는 방식(message-oriented transmission)이 된다.

라. ID 부여 방법

본 연구에서는 CAN 2.0A의 Base 포맷과 CAN 2.0B의 Extended 포맷을 이용하는 ID 부여 방법을 모두 검토하였다. 특히 본 연구에서 CAN 기반 기술을 적용하려고 하는 과수방제기의 경우는 아직 표준 규격이 제정되어 있지 않을 뿐 아니라 사용될 ID의 종류가 많지 않기 때문에 CAN 2.0A에 근거하여 ID가 만들어졌으며, 향후 농용트랙터나 콤바인 등 CAN 규격이 이루어진 경우에의 적용을 대비하여 가상적으로 ID를 부여하는 방안을 검토하였다. 이때 ID는 ISO11783/J1939 규정에 충실하고자 하였으나 정의되지 않은 부분은 임의로 부여하였다.

2. 적합한 CAN의 선정

CAN controller는 중간단계의 메세지 저장 능력만 제공하는 것으로 host controller와 함께 사용되어야 하는데, 현재 시판되고 있는 CAN controller은 크게 외장형(Stand-alone CAN Controllers)과 내장형(Microcontrollers with Integrated CAN Controllers)으로 구분된다. 외장형 CAN controller의 경우는 host controller 와 인터페이스하여 사용해야 하므로 내장형에 비하여 회로의 크기도 커지고 불편한 점이 많이 있다. 표 3.3.1는 조사된 CAN controller들의 특성을 나타낸 것이다.

본 연구에서는 Atmel 사의 T89C51CC01와 Microchip사의 PIC18FXX8 마이크로프로세서 등 두 종류를 선정하였다.

표 3.3.1 시판되고 있는 주요 CAN controller

Vendor	Model	Type	Remarks
Philips	SJA 1000	2.0B, Stand-alone, BasicCAN	
	XA-C3	2.0B, Integrated	
Intel	82527	2.0B, Stand-alone, FullCAN	
	87C196CA/CB	Integrated	16 bit version
Microchip	PIC18FXX8	2.0B, Integrated, FullCAN	
	MCP2510	2.0B, Stand-alone, FullCAN	SPI interface
Infineon	82C900	2.0B, Stand-alone, FullCAN	twin CAN
Atmel	T89C51CC01	2.0B, Integrated, FullCAN	8 bit
Motorola	MC68HC05XX	2.0A, Integrated	8 bit
	MC68376	2.0B, Integrated	32 bit

표 3.3.1에서 나타난 바와 같이 T89C51CC01은 내장형 CAN controller로서, cpu로 Intel의 8051 core를 사용하기 때문에 8051 및 80196 계열의 마이크로콘트

롤러를 많이 사용해 본 연구팀에서 cpu의 기능을 충분히 활용할 수 있었기 때문이다. 또한 PIC 18시리즈는 CAN 기능을 탑재하여 새로 출시한 마이크로프로세서로서 풍부한 라이브러리 지원으로 사용하기가 용이하다는 장점이 있다. 2년여의 연구 기간에 걸쳐 1차년도에는 T89C51CC01 중심으로 2차년도에는 PIC 18F448 중심으로 연구가 수행되었다.

가. Atmel 89C51CC01

89C51CC01은 34개의 Special Function Register의 형태로 8051의 내부 메모리 범위내에서 15개의 독립적인 메시지 버퍼(채널) 사용이 가능하며, 각각은 CAN 2.0A 또는 CAN 2.0B 모드로 수신채널, 송신채널 및 수신버퍼채널 등으로 사용될 수 있다. 1 KB의 on-chip XRAM, 32 KB의 on-chip FLASH, 2 KB의 on-chip FLASH for boot loader, 2 KB의 on-chip EEPROM, 14-source 4-level interrupts, 21-bit watchdog timer, 8-channel A/D converter with 10 bit resolution 외에 3 개의 Timer/Counter, 다수의 일반 I/O 포트를 갖고 있다. 외부 클록으로는 12 MHz를 사용한다.

구동 소프트웨어는 Keil사의 PK51 C 컴파일러로 작성되며 ANSI C를 표준으로 사용하므로 프로그램이 용이하다.

나. Microchip PIC18F448

PIC 18FXX8 시리즈는 16 KByte의 내부 메모리, 768 Byte의 데이터 메모리를 갖고 있으며 21개의 인터럽트 소스와 4개의 timer를 내장하고 있고 Capture/Compare/PWM 모듈을 갖고 있다. 또 8개의 AD 채널이 있으며 2개의 아날로그 비교기가 있다. 외부 클록으로는 20 MHz를 사용한다.

CAN controller로서 6개의 full (standard/extended identifier) acceptance filter를 갖으며 이 중 2개는 우선순위가 높고 나머지 4개는 우선순위가 낮은 수신버퍼이다. 2개의 full acceptance mask 가 있으며 각각의 우선순위가 달라진다. 우선순위 지정과 취소 기능이 있는 3개의 송신버퍼가 있다.

구동 소프트웨어는 CCS-C 컴파일러를 이용하여 작성하며 풍부한 라이브러리는 프로그램을 매우 용이하게 한다는 장점이 있다.

3. CAN 기반의 ECU(Electronic Control Unit) 설계 및 제작

가. T89C51CC01

1) 시험용 ECU의 설계(1차년도)

가) CPU와 CAN driver, RS232C 통신을 위한 인터페이스 회로 설계

CAN controller를 CAN bus에 연결하기 위해서 CAN 트랜시버 칩이 필요하다. 본 연구에서는 고속통신이 가능한 Phillips 사의 82C250 칩을 사용하였다. 프로그램의 다운로드 및 RS232 시리얼 통신을 위하여 MAX232를 사용하여 시리얼 포트를 설치하였다. 향후 사용할 목적으로 4개의 A/D 채널에 대한 입력 포트를 커넥터 연결단자로 미리 빼 놓았으며 일반 I/O 포트에 4개의 LED를 부착하여 ECU의 작동상태를 가시적으로 나타낼 수 있도록 하였다(그림 3.3.6).

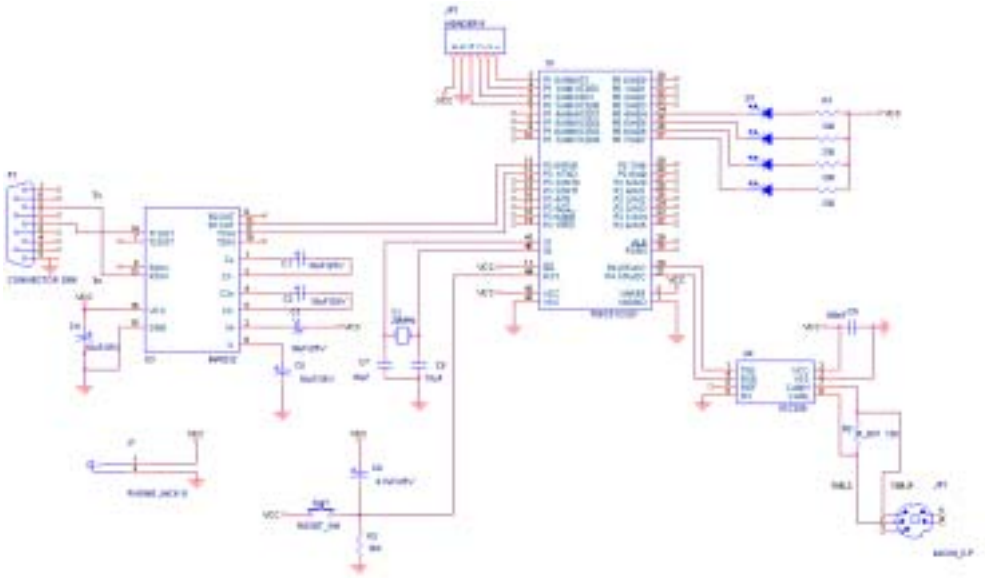


그림 3.3.6 T89C51CC01 기반 ECU의 회로도(테스트용).

나) CAN bus

CAN_H 및 CAN_L를 위해 2 가닥의 twisted pair선을 CAN bus로 사용하였으며 bus의 양 끝단에는 120Ω의 터미네이터를 장착하였다.

다) 프로그래밍

ECU 작동 프로그램의 흐름도를 그림 3.3.7에 나타내었으며 C언어로 작성한 프로그램을 KEIL 사의 PK-51 컴파일러로 컴파일한 후 롬라이터(LT32, Advantech, Taiwan)를 사용하여 마이크로콘트롤러의 EEPROM에 다운로드하여 프로그램을 실행시켰다.

그림 3.3.7 CAN 구동 흐름도(T89C51CC01).

라) 통신 성능 평가

4개의 ECU를 제작하여 CAN 버스에 연결한 후 각각의 ECU에 임의의 ID를 부여한 후, 임의의 data packet을 방송 모드, 하나의 노드에서 특정 노드로 전송하는 모드, 데이터 요청하여 받는 모드 등 CAN에서의 데이터 통신 방법을 모두 적용하여 ECU가 원활하게 작동하는 것을 확인하였다.

정량적인 통신 성능 평가를 위하여 두 개의 ECU에 임의의 ID를 부여하고 polling mode, interrupt mode, request remote mode 등의 3가지 방법으로 10ms 주기로 10,000개의 데이터를 전송하였을 때 CAN 프로토콜에서의 error인 Stuff error, CRC error Frame error, Bit error, Acknowledgement error를 조사한 결과를 표 3.3.2에 나타내었다. interrupt mode의 경우는 error가 전혀 없었으며 polling mode와 Request remote mode에서는 약간의 error가 관측되었으나 CAN의 작동 특성상 error가 발견되면 재전송을 하게 되므로 송신한 데이터 10,000개를 100% 수신하는 것으로 나타났다. 실제로 CAN 버스 상에서 데이터 송수신간에 error가 나타날 확률은 거의 없다(Etschberger, 2001)고 할 수 있다.

표 3.3.2 통신 중에 발생하는 에러의 평균 수

Mode	No. of data		S error	C error	F error	B error	A error
	Transmit	Receive					
Polling	10,000	10,000	18.6	24.6	3	0	0
Interrupt	10,000	10,000	0	0	0	0	0
Request remote	10,000	9,999	20	30	1	7	15

마) CAN 네트웍의 구성

우선 4 개의 ECU를 제작하여 센서, 조향장치 조작기, 가상터미날 등으로 그림 3.3.8에서와 같이 CAN 네트웍을 구성하였다. ECU 1은 조향장치 조작기로 조향클러치를 제어하는 것이 주 목적으로 CAN 버스를 통해 조향에 필요한 정보를 수신 받은 후 유압실린더를 제어하는 제어기(그림 3.3.7)로 필요한 데이터를 출력한다. 즉, ECU 1이 수신하는 메시지는 표 3에서와 같으며 또한 가상 터미널 쪽으로 좌우 조향 클러치의 현재 상태를 송신한다. ECU 2 와 ECU 3는 과수방제

기의 지역위치를 측정하기 위한 센서들을 관장한다. 측정된 정보를 일정한 주기로 CAN 버스상에 송신하게 된다. ECU 4는 가상 터미널를 구동하는데 사용된다. 가상터미널은 제어 시스템 상의 필요한 정보 등을 네트웍 상에서 수신받아 보여주거나 또는 전체 시스템의 시작과 종료 등 스위치 신호를 CAN 네트웍상에 송신하게 된다. 가상터미널인 원보드컴퓨터(ARM)는 ECU4와 RS232 통신으로 연결된다.

전체 제어시스템은 원보드 컴퓨터에 전력을 공급하는 것으로부터 작동되며, 이것은 전력제어기를 작동시켜 각 ECU에 전력이 공급되면 각 ECU 들은 각자의 기능을 수행할 수 있도록 초기화된다. 가상터미널에서 제어 시스템의 시작 버튼을 누름으로써 CAN 버스가 작동되며 시스템은 제어되기 시작한다.

바) 메시지 ID

본 CAN 네트웍에서 사용할 메시지 및 각 ECU에서 메시지의 송, 수신 상태를 표 3.3.3에 나타내었다. ECU1은 ECU2로부터 x-, y- 각도 정보(X_Offset, Y_Offset), ECU3로부터 초음파 센서에 의한 좌우측 거리 정보(L_Distance, R-Distance), ECU4로부터 Start, Stop 정보를 수신한다. 또한 좌우 조향클러치의 작동 상태를 필요한 ECU에서 수신 가능하도록 방송 모드로 송신한다.

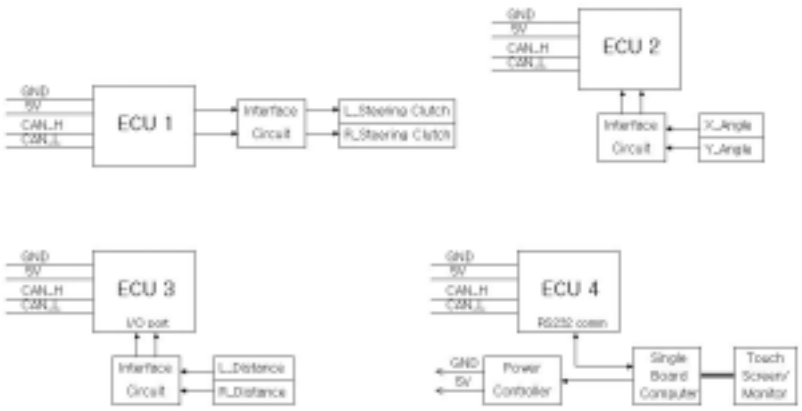


그림 3.3.8 CAN을 구성하는 4개의 ECU.

표 3.3.3 List of messages used in the proposed CAN system

ECU	Data (or Message)	TX/RX	Channel No.	Remarks (Trans. rep. rate)
ECU1	X_Offset	receiver	0	
	Y_Offset	receiver	1	
	L_Distance	receiver	2	
	R_Distance	receiver	3	
	Start	receiver	4	
	Stop	receiver	5	
	L_Clutch status	transmitter	6	500ms
	R_Clutch status	transmitter	7	500ms
ECU2	X_Offset	transmitter	0	100ms
	Y_Offset	transmitter	1	100ms
ECU3	L_Distance	transmitter	0	150ms
	R_Distance	transmitter	1	150ms
ECU4	Start	transmitter	2	on change of Start sw
	Stop	transmitter	3	on change of Stop sw
	L_Clutch status	receiver	0	
	R_Clutch status	receiver	1	

이러한 송, 수신을 위하여 15개의 메시지 버퍼(Channel)를 송신 및 수신에 할당한다. 그러나, 앞서 언급한 바와 같이, Higher Layer Protocol인 ISO11783 또는 SAE J1939의 규약에 따른 방식으로 메시지 ID를 표현하면 사용할 메시지의 숫자를 크게 줄일 수 있다. 일반적으로 CAN에서는 하나의 메시지에 8 byte의 데이터를 실어 보낼 수 있기 때문에 예를들어 ECU2의 각도 측정 메시지인 X_Offset 과 Y_Offset을 하나의 메시지 offset으로 하여 두 메시지에 있던 데이터를 하나로 묶어서 송신할 수 있다. 따라서, 최종적으로 각 메시지에 부여하는 ID를 표 3.3.4에서와 같이 설정할 수 있었다.

표 3.3.4 메시지의 구조와 ID의 할당

		Message			
		Offset	Distance	Start/Stop	Clutch_status
Transmission repetition rate		100ms	150ms	on change of s/w state	500ms
Default priority		3	6	0	6
Data length		8 byte	8 byte	8 byte	8 byte
PGN	Data page	0	0	0	0
	PDU format	241	241	203	241
	PDU specific	100	102	128	103
PGN		$F164_{16}$	$F166_{16}$	$C680_{16}$	F167
Source address		208	209	38	210
ID		$CF164D0_{16}$	$18F166D1_{16}$	$C68026_{16}$	$18F167D2_{16}$
Data	Byte1,2	x_offset	l_distance	not defined	l_/r_clutch
	Byte3,4	y_offset	r_distance	not defined	not defined
	Byte5,6	not defined	not defined	start/stop	not defined
	Byte7,8	not defined	not defined	not defined	not defined

그림 3.3.9는 제작된 ECU들이 CAN 버스에 연결된 상태의 모습으로 가상터미널은 ECU4에 시리얼 통신으로 연결되어 있다. ECU2 와 ECU3에서 가상의 센서 데이터를 주어진 주기로 송신하고 ECU1에서는 가상의 조향클러치 상태 데이터를 송신하게 한 후, ECU 1에서와 ECU4에서 수신된 데이터를 비교한 결과 CAN 버스상에서의 통신은 전혀 에러가 발생하지 않았다.

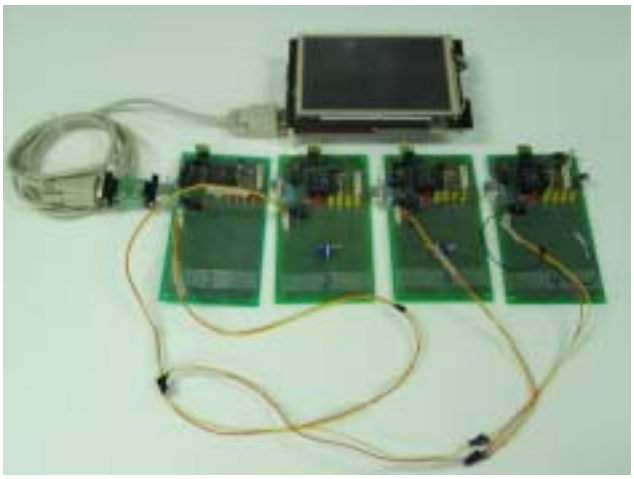


그림 3.3.9 단순한 CAN 버스 시스템 구성.

2) T89C51CC01 기반 ECU의 설계 및 제작(2차년도)

1차년도의 연구 결과를 바탕으로 과수방제기에 적용할 T89C51CC01 기반의 ECU를 설계, pcb로 제작하였다. 그림 3.3.10에서 보는 바와 같이 사용 빈도가 높을 것으로 예상되는 입출력 단자들을 복층 터미널블록 2개를 이용하여 외부와 인터페이스될 수 있도록 설계하였다. 자주식 농업기계에서 기본적으로 12V DC 배터리를 사용하므로 ECU로 입력되는 전원을 7805를 통해 5V로 만들었으며 또한 T89C51CC01의 AD 변환기에서 사용 가능한 최대 전압이 3V 이므로 1117를 사용하여 2.5V로 다운하여 AD 변환기로 입력될 센서 등의 기준전원으로 사용할 수 있도록 하였다. 4개의 디지털 입력은 PC847 포토커플러를 통해 신호가 마이크로프로세서로 입력되도록 회로를 설계하였다. 8개의 출력단은 5V 코일의 SPDT 릴레이(NAIS, Japan)를 통해 외부의 부하와 연결될 수 있도록 하였으며 동시에 신호 출력 여부를 확인할 수 있도록 각각에 LED를 부착하였다. MAX232를 이용하여 시리얼포트(D-SUB9, F)를 1개 설치하고 82C250 CAN 드라이버를 통해 CAN 버스에 연결될 수 있도록 CAN 포트(D-SUB9, M)를 1개 설치하였다. 구형파 입력 또는 외부 신호의 주파수 측정 등의 목적으로 timer 0와 timer1 핀 입력단자도 설계에 포함하였다.

표 3.3.5는 ECU 제작에 소용된 부품명세서이며 그림 3.3.11은 조립이 완료된 모습이다. PCB는 양면으로 제작되었으며 릴레이들은 기판의 후면부에 부착되도

록 하였다.

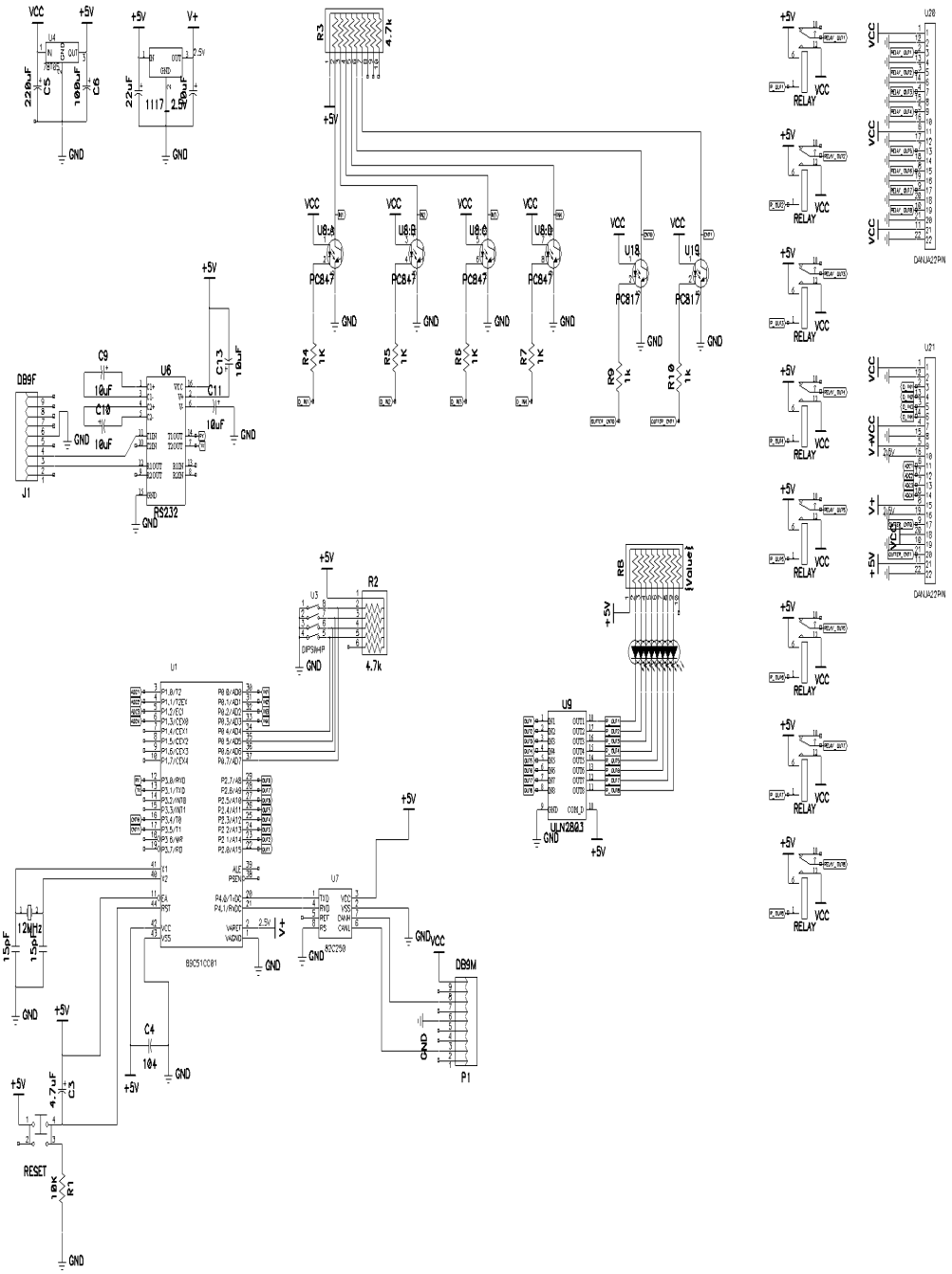


그림 3.3.10 T89C51CC01 기반 ECU의 설계도(프로토타입)

표 3.3.5 ECU 제작에 소요된 부품 명세서(T89C51CC01)

부 품 명	규 격	개 수	비 고
ECU	89C51CCO1CA-IM	1	•
전해콘덴서	4.7㎌	1	C3
	10㎌	5	C9,C10,C11,C13
	22㎌	1	C7
	100㎌	1	C5
	220㎌	1	C6
세라믹콘덴서	15	2	•
	104	1	•
저항	1kΩ	6	1K
	10kΩ	1	10K
어레이저항	4.7kΩ - 9핀	1	4.7K
	4.7kΩ - 5핀	1	4.7K
	3.3kΩ - 9핀	1	330
크리스탈	12㎒	1	XTAL
드라이버	ULN2803A	1	2803A
	PCA82C250	1	82C250
	PC817 - 16핀	1	PC847
	PC817 - 4핀	2	U18,U19
	MAX232	1	DS232A
시리얼포트	9핀DUB(RS-232)	2	•
단자	22핀	2	•
릴레이	DSIE-M-DC5V	8	RELAY
정전압	L7805CV	1	78T05
LED	•	8	D1,2,3,4,5,6,7,8
TR	•	1	U5
DIP스위치	4극	1	DIPSW4P

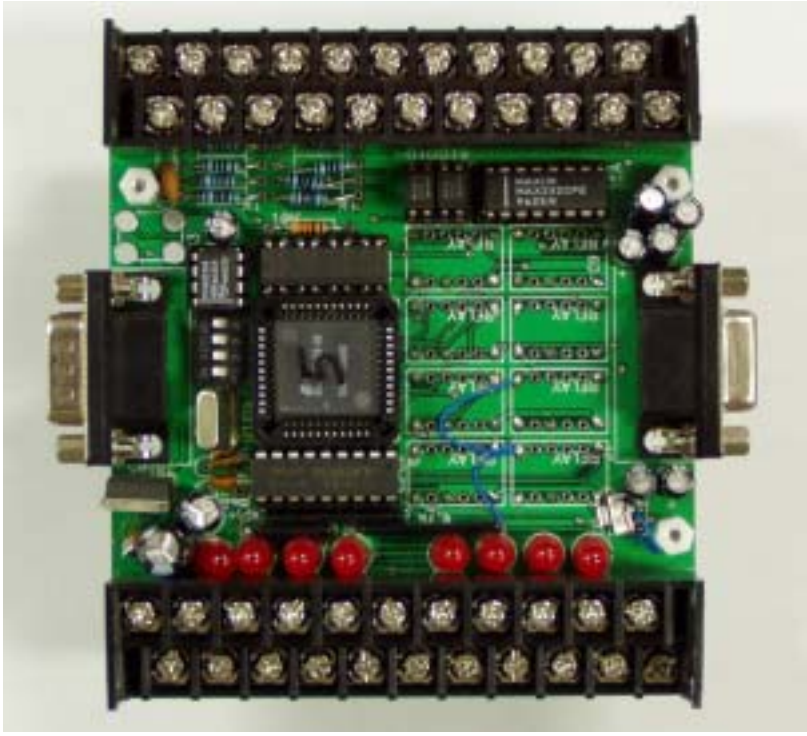
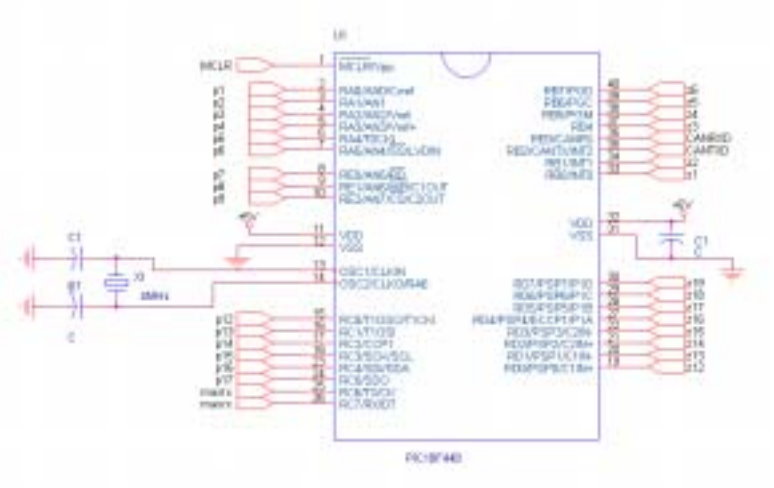


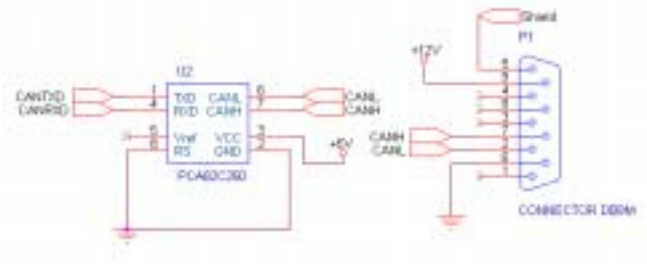
그림 3.3.11 T89C51CC01 기반 ECU의 완성된 모습(프로토타입).

나. PIC 18F448 기반 ECU의 설계 및 제작

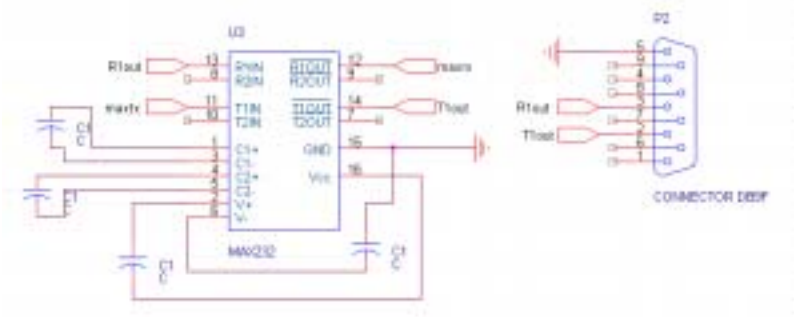
PIC 기반 ECU는 그림 3.3.12에서 보는 바와 같이 설계되어 PCB로 제작되었다. 가급적 마이크로프로세서의 모든 입출력 단자를 외부와 인터페이스가 용이하도록 복층 터미널블록 2개를 사용하여 단자대를 만들었으며 케이스에 장착할 때 용이하도록 CAN 포트와 시리얼포트를 한쪽 방향에 설치하고 입출력단자대를 그 반대 방향에 위치되도록 설계하였다. 그림 3.3.13은 조립이 완료된 ECU의 모습이며 표 3.3.6은 사용된 부품 명세서이다.



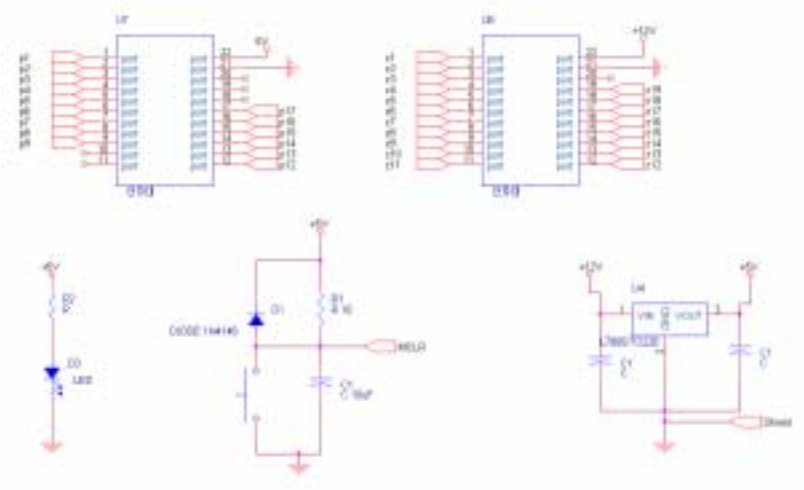
(a) PIC 18F448



(b) CAN 포트



(c) 시리얼 포트



(d) 입출력단자대 및 DC-DC변환기

그림 3.3.12 PIC 18F448 기반 ECU의 설계도(프로토타입)

그림 3.3.13 PIC 18F448 기반 ECU의 완성된 모습(프로토타입).

표 3.3.6 ECU 제작에 소요된 부품 명세서(PIC 18F448)

부 품 명	규 격	개 수	비 고
CPU	PIC18F448	1	•
전해콘덴서	10㎌	5	C4,C5,C7,C8,C9,C10
세라믹콘덴서	104	2	C3,C6
	30	2	C1,C2
저항	100kΩ	1	R1
	330Ω	1	R2
다이오드	IN4001	1	D1
LED	•	1	D3
크리스탈	20㎒	1	X1
RESET스위치	•	1	S1
드라이버	MAX232	1	U3
	82C250	1	U2
단자	22핀	2	U7,U8
시리얼포트	9핀DUB(RS-232)	2	P1,P2
정전압	78T05(방열판포함)	1	U4

다. CAN 버스 선

CAN 버스는 CAN_H 및 CAN_L의 두 가닥의 TP(twisted pair) 선으로 구성된다. 여기서 사용된 통신선은 UTP선으로 5 조의 TP 선이 있어 한 조를 CAN 통신용으로 배정하고 다른 한 조는 전원 공급선으로 사용하였다. 버스로부터 각

노드로 연결되는 T형 커넥터를 만들기 위하여 별도의 PCB를 제작하여 D-SUB9 커넥터 3개를 설치하였다(그림 3.3.14). CAN 버스와 ECU 연결에 필요한 길이만큼 연결 케이블을 제작하여 CAN 버스를 구성하였다. 그림 3.3.15는 3개의 ECU를 CAN 버스로 구성할 때 케이블 배열의 모습을 보여준다. 그림에서 좌우측 끝단에 120Ω짜리 종말저항이 장착된 커넥터이며 오른쪽 끝단에는 전원 공급 단자를 함께 설치하였다.

그림 3.3.14 T형 커넥터.

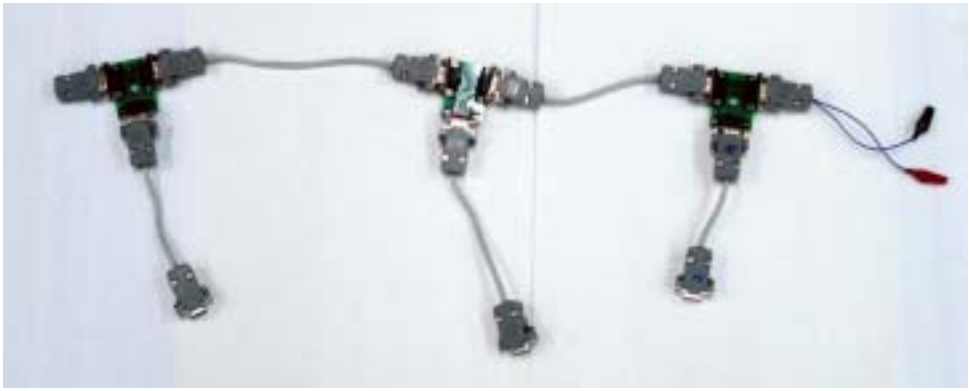


그림 3.3.15 CAN 버스 T형 연결단 및 종단 저항이 설치된 연결단.

4. CAN 버스 구동 프로그램 개발

CAN을 구현하기 위해서는 마이크로프로세서에서 CAN controller와 CAN driver를 통해 송・수신의 시리얼통신을 수행하여야 한다. 즉, 마이크로프로세서

에서 CAN 버스에 악세스할 수 있는 구동 프로그램이 먼저 개발되어야 한다.

단위 마이크로프로세서 수준에서의 CAN 구동 프로그램은 아직 안정화되었다거나 일반적으로 보급되어 있는 것은 아니기 때문에 기존의 컴파일러 제작사에서 제공하는 기본 라이브러리나 샘플 프로그램을 연구하고 마이크로프로세서와 CAN 통신에 관련된 모든 레지스터 등을 규명하여 CAN 통신을 구현할 수 있는 프로그램을 자체적으로 개발하였다.

CAN은 크게 3가지 모드로 나뉘어 통신을 한다. 첫째는 polling 모드로 작동하는 것으로 이것은 일정시간마다 CAN 버스를 통해 수신된 데이터가 들어 있는 버퍼를 polling 방식으로 체그한다든지 송신을 할 때는 특정 시간마다 메시지를 구성하여 CAN driver를 통해 송신하는 방식이다. 이 방식은 최악의 경우 데이터의 overrun으로 인한 제대로 메시지를 받지 못할 수도 있다. 이런 방식에 비해 interrupt 모드는 CAN으로 데이터를 송신하거나 수신할 때 인터럽트가 걸리게 하여 인터럽트 플래그를 확인한 후 데이터를 송수신하게 되므로 예비 성능 시험 결과에서와 같이 데이터가 유실되는 경우는 발생하지 않는다. 원격요청 모드는 하나의 ECU에서 다른 ECU의 특정 메시지를 요청하여 받는 방식으로 요청을 받은 ECU는 상대방에서 원하는 메시지를 송신하는 방식으로 안정적으로 통신이 가능한 방법이라고 할 수 있다.

가. Atmel 사의 T89C05CC01

라이브러리 파일은 먼저 CAN 통신 관련 레지스트리를 정의한다. 이것은 총 15개의 채널을 우리가 원하는 대로 polling 모드, interrupt 모드, remote request 모드 등으로 세팅할 수 있다. 특히 채널을 세팅할 때는 ID 포맷을 표준형으로 할지 확장형으로 할지, 송신용인 지 수신용인지, 어떤 모드로 작동하는 지, 데이터의 길이 등을 정의한다. 채널을 정의할 때 동시에 마스크필터도 정의해야 하는데 일반적으로 송신용인 경우 필터를 0x3fffhf 설정하여 어떤 ID의 메시지도 송신 가능하도록 하며 수신용인 경우는 원하는 ID의 메시지만 들어 올 수 있도록 필터를 설정하게 된다. 다음은 CAN 통신에서의 통신 속도를 결정해야 한다. 이후에 다음과 같은 함수들이 정의된다.

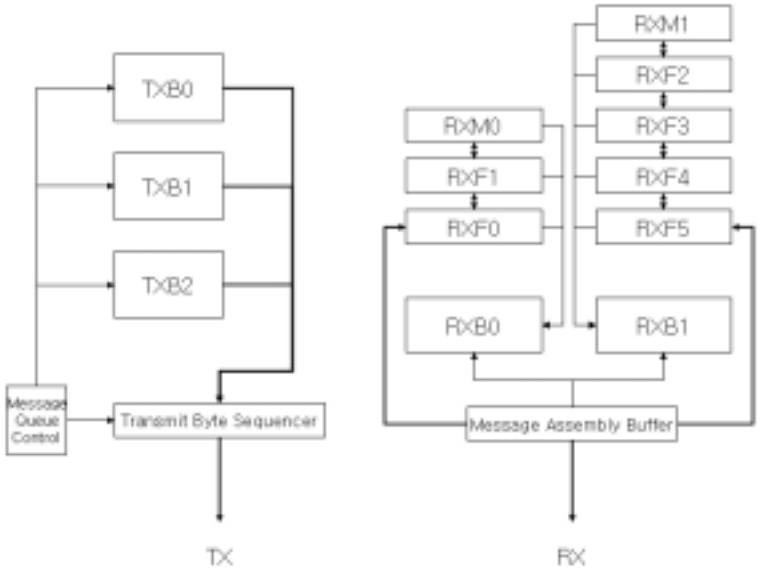
공통으로 CanInit()이 정의되고, polling 모드로 작동하는 CanSend()와 CanRead()가 정의되었으며, interrupt 모드로 작동하기 위하여 interrupt를 발생시켜주는 CanInterrupt() 및 interrupt service routine인 CanSendIsr()와

CanReadIsr() 등의 함수를 정의하였다, 또한 remote request 모드를 위해 CanReqRemote()와 CanGetRemote()을 작성하였다.

나. Microchip 사의 PIC18F448

CAN 모듈은 주변 장치나 마이크로 컨트롤러의 유용한 통신수단으로 잡음이 많은 환경 안에서 사용될수 있도록 설계 되었으며 PIC18F448은 PCA82C250 드라이버를 통해 CAN에 적용될 수 있다. CCS-C에서는 CAN-18xxxx.h와 CAN-18xxxx.c 로 구성된 라이브러리를 제공한다. 이것은 PIC18F448에서 사용할 수 있는 레지스터를 정의 한다.

PIC18F448은 CAN으로 2개의 Receive Buffer 와 3개의 Transmit Buffer 제공하며 두 Receive Buffer에는 2개의 ID Mask와 6개의 ID filter를 가지고 있다. ID는 Standard/Extended로 구성되며 Mask와 filter로 필요한 ID를 수신 할 수 있다. 프로그램에서 두 개의 Mask와 filter을 정의 하여 사용하였다.
Buffer의 구성은 다음 다이어그램과 같다.



한편, 라이브러리는 공통적으로 다음과 같이 함수를 정의 한다.
can_init() - Configures the PIC18xxx8 CAN peripheral
can_set_baud() - Sets the baud rate control registers

can_set_mode() - Sets the CAN module into a specific mode
can_set_id() - Sets the standard and extended ID
can_get_id() - Gets the standard and extended ID
can_putd() - Sends a message/request with specified ID
can_getd() - Returns specifid message/request and ID
can_kbhit() - Returns true if there is data in one of the receive buffers
can_tbe() - Returns true if the transmit buffer is ready to send more data

기본적인 통신은 poling mode와 Interrupt mode가 있으며 Interrupt mode는 메시지가 Receive Buffer에 수신 되었을 때 발생되는 인터럽트 레지스터에 의해 인터럽트 함수가 활성화 되며 이 때 can_kbhit()가 활성화 되고 can_getd() 함수로 미리 정의 해둔 변수에 메시지를 저장한다. poling mode는 일정한 시간이나 Data의 필요에 의해 can_kbhit()가 활성화를 확인한 후 can_getd()를 통해 메시지를 받는다. Transmit은 get_putd() 함수로 CAN상에 Load 되며 Interrupt mode는 사용하지 않았다.

제 4 절 센서/조작기-마이크로프로세서의 인터페이스

1. 유압실린더(조향/주행) 제어기

좌우 조향클러치 및 주클러치/브레이크를 작동하는 유압실린더의 전자방향제어 밸브를 작동시키기 위하여 릴레이(JW2SN-DC12V, NAIS)를 이용한 인터페이스 회로를 개발하였다. 마이크로프로세서의 디지털 출력단을 통해 해당 릴레이에 High 신호를 출력하면 유압실린더의 방향제어 솔레노이드 밸브가 작동하여 유압실린더를 작동시키게 되며 Low 신호에 의해 유압실린더는 원래의 위치로 복귀하게 된다. 마이크로프로세서 출력 신호로 직접 릴레이를 구동할 수 없기 때문에 TR을 통해 출력 신호를 증폭할 수 있도록 인터페이스를 구성하였으며, 그림 3.4.1은 릴레이 인터페이스 회로도이고 그림 3.4.3의 우측에 보이는 것이 실제 제작되어 사용되는 인터페이스 회로이다. 클러치 제어용 ECU로부터의 신호가 이곳으로 출력되어 유압 실린더들의 제어가 수행된다.

유압실린더의 작동 특성 실험을 용이하게 수행하기 위하여 그림 3.4.3의 좌측 사진에서와 같이 별도의 마이크로프로세서로 인터페이스 회로(그림 3.4.2)를 구성하여 스위치를 이용하여 수동으로 해당 유압실린더를 작동 시키거나 타이머를 이용하여 해당 유압 실린더를 특정 시간만 작동할 수 있도록 하였다.

편의성 증진을 위하여 유압실린더 작동 상황 표시기와 자동운전 모드 및 리모콘운전 모드를 선택할 수 있는 인터페이스를 제작하였다(그림 3.4.4).

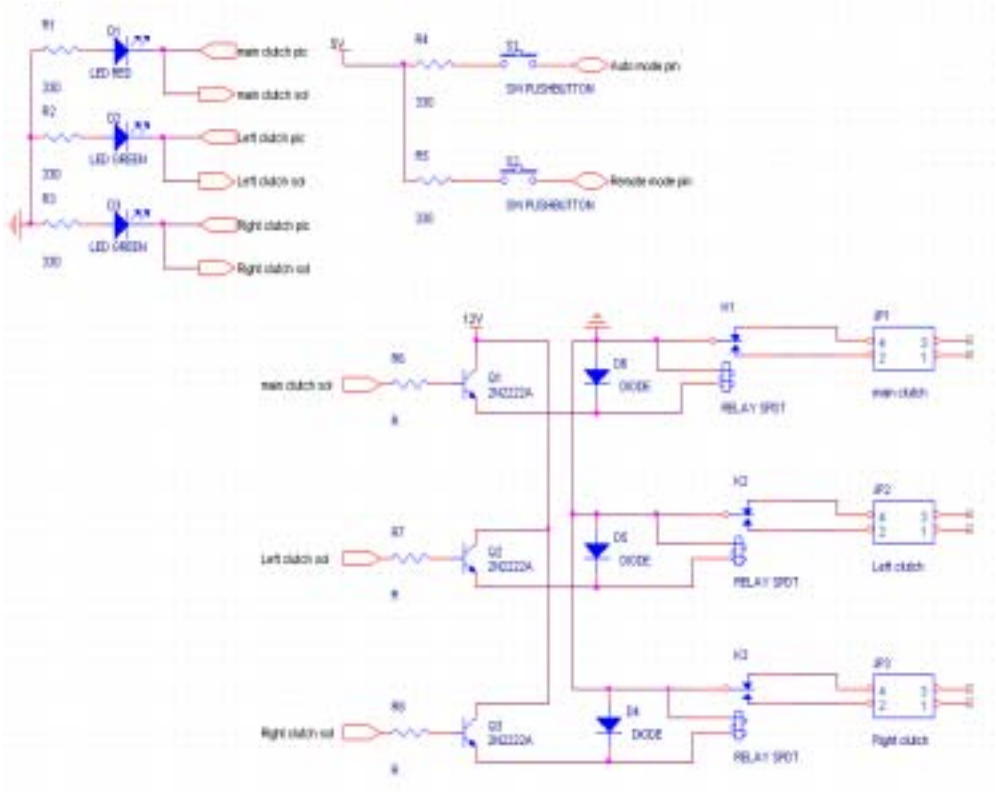


그림 3.4.1 유압 실린더 제어기 회로도.

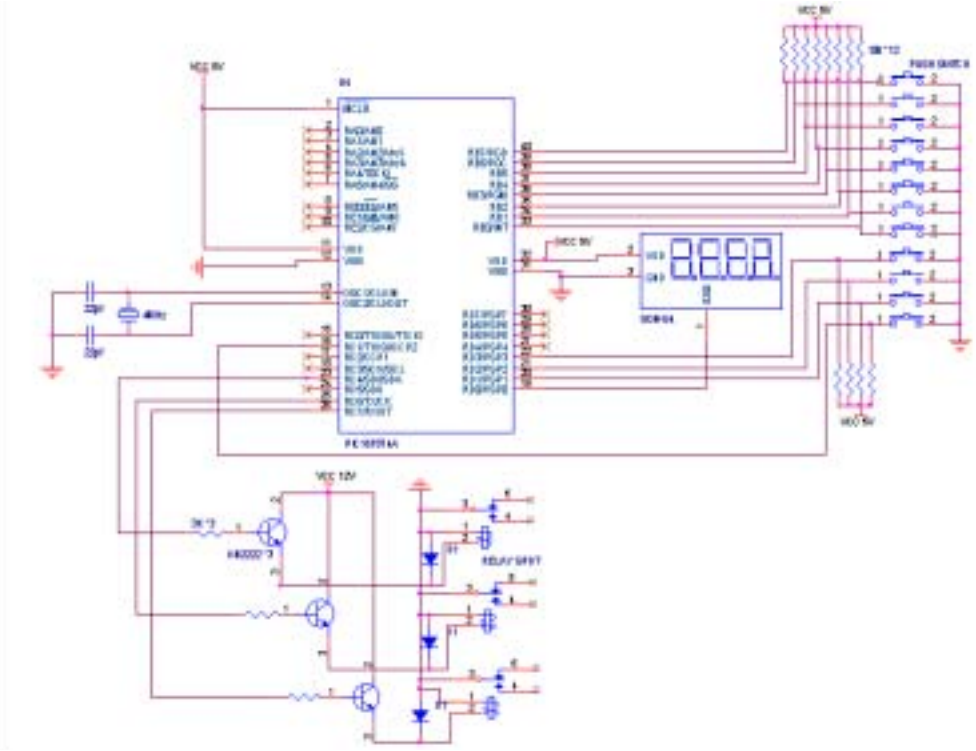


그림 3.4.2 유압실린더 독립 구동 인터페이스 회로.

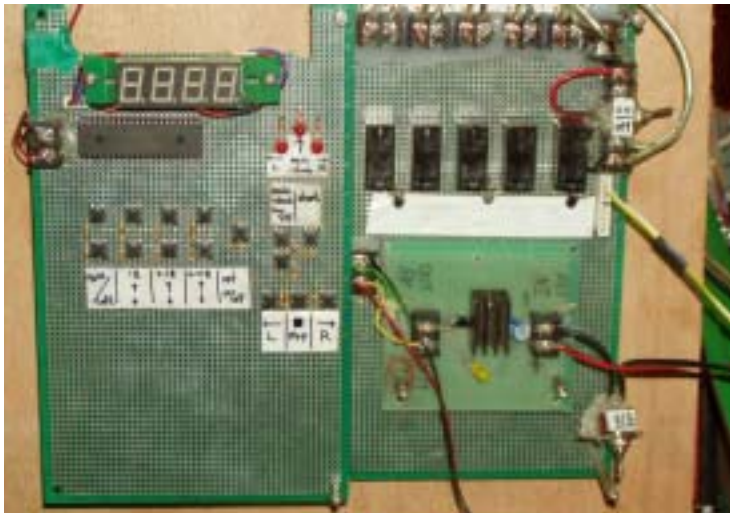


그림 3.4.3 유압실린더 제어기 인터페이스 회로.

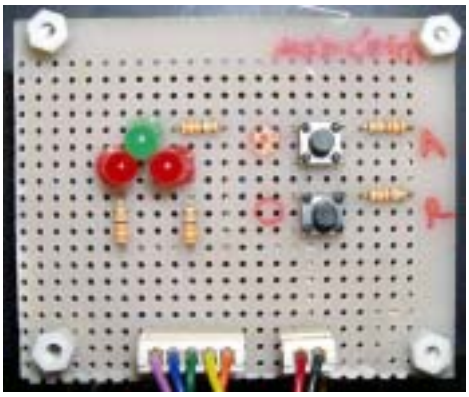
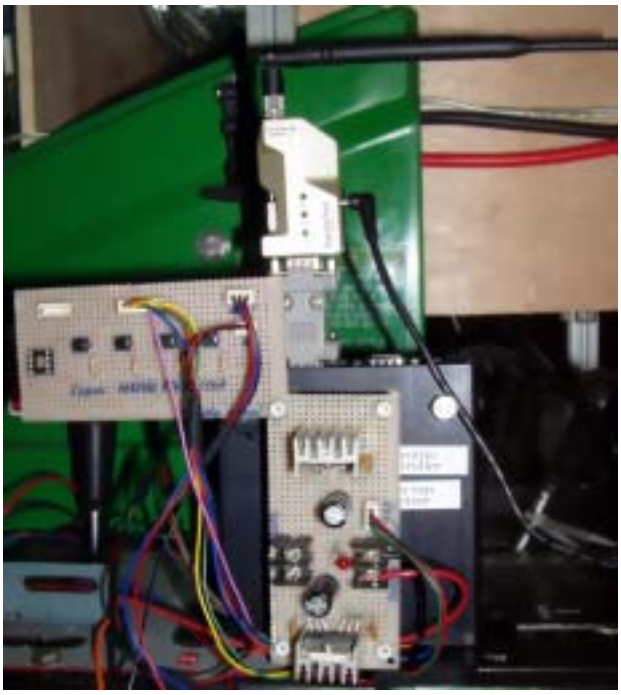


그림 3.4.4 유압실린더 작동상황 표시기 및 운전모드 선택 스위치.

2. 드로틀레버 제어기

드로틀 레버 제어기는 12V DC로 구동되는 전동실린더의 모터를 제어하여 실린더의 위치 제어를 수행함으로써 드로틀 레버가 요구되는 곳에 위치하게 하여 엔진의 회전속도를 제어할 수 있게 한다. 그림 3.4.5의 인터페이스 회로도에서 보는 바와 같이 DC모터 전용 드라이버인 LMD18200 2개를 병렬로 사용하여 전동실린더를 제어하였다. 한편 드로틀 레버의 위치는 선형 포텐시오메터에 의해 측정되어 피드백되어 전동실린더의 위치 제어를 가능하도록 하였다. 포텐시오메터의 출력은 전압분배회로로 구성되어 마이크로프로세서의 AD 핀으로 입력되며, 전동실린더가 미리 설정해 놓은 AD 값을 출력하는 위치에 오면 정지하도록 단순 P-제어를 적용하였다.

편의성 증진을 위하여 별도로 4개의 PB 스위치를 인터페이스 기판에 설치하여 수동 운전시 필요한 엔진 회전속도를 획득할 수 있도록 하였다.



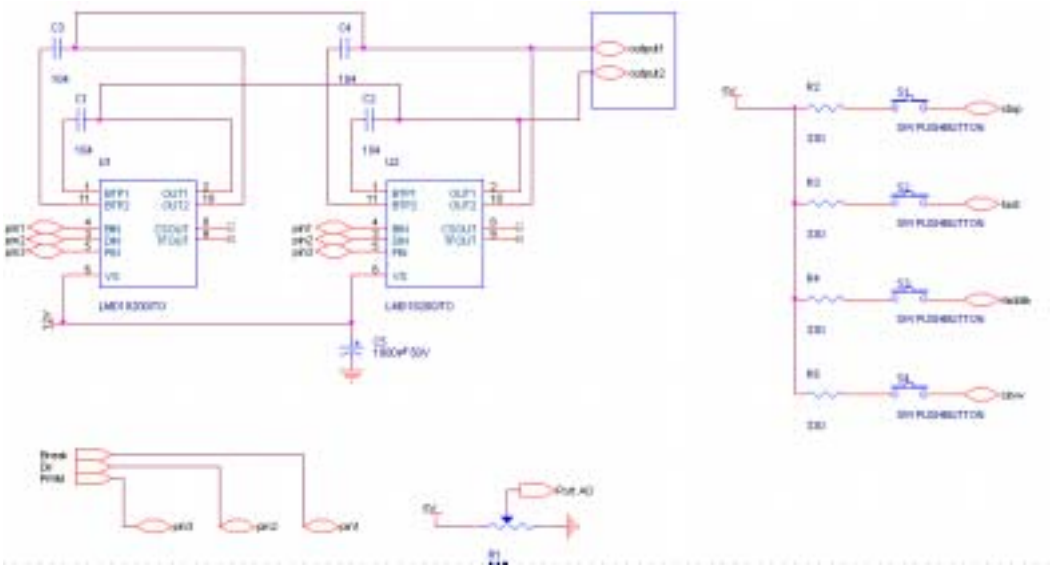
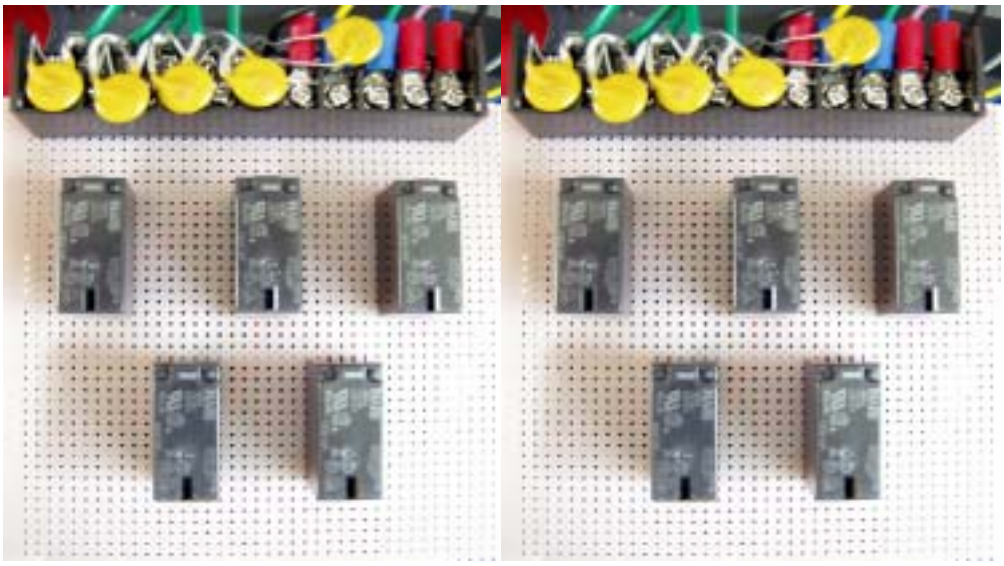


그림 3.4.5 드로틀레버 제어기 인터페이스.

3. 약액살포 제어기

솔레노이드 밸브를 조작하기 위하여 SPDT 릴레이를 이용한 인터페이스 회로를 개발하였다.

T89C51CC01의 경우 디지털 출력 단자를 8개만 외부로 인출해 놓은 상태이므로 한번에 5개씩 두 개의 마이크로프로세서를 이용하여 제어하는 방식을 채택하였으며, 그림 3.4.6은 한쪽 방향, 즉 5개의 노즐에 대한 인터페이스 회로도로서 이미 T89C51CC01의 출력이 릴레이를 통하였으므로 디지털 출력 단자로 High 신호를 출력하면 해당 릴레이와 LED가 작동하고 그럼으로써 인터페이스상의 릴레이를 구동하여 최종적으로 솔레노이드밸브를 작동하도록 하였다.



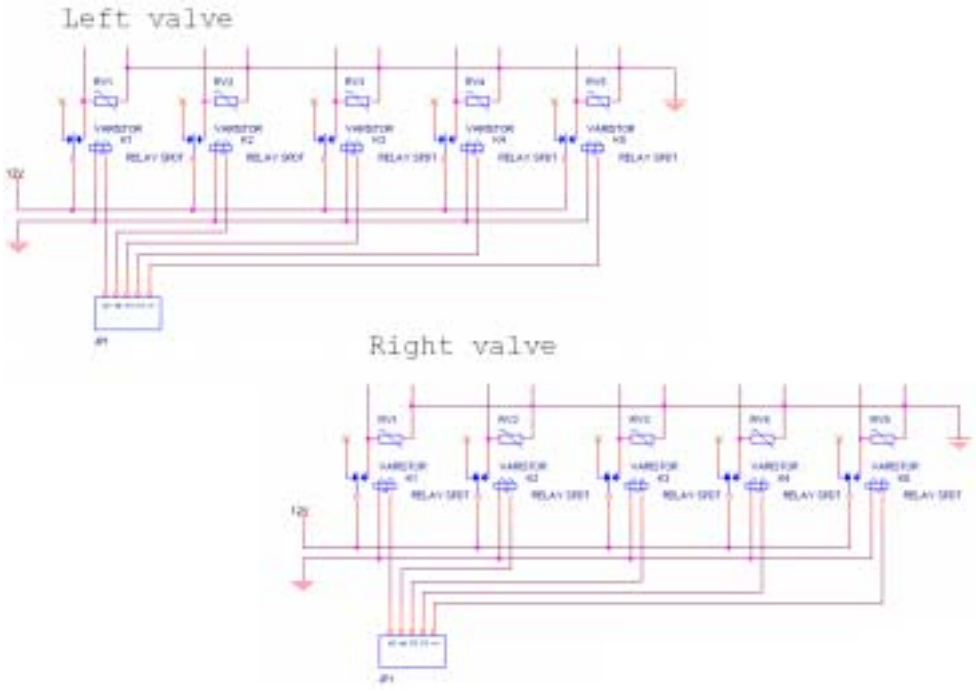
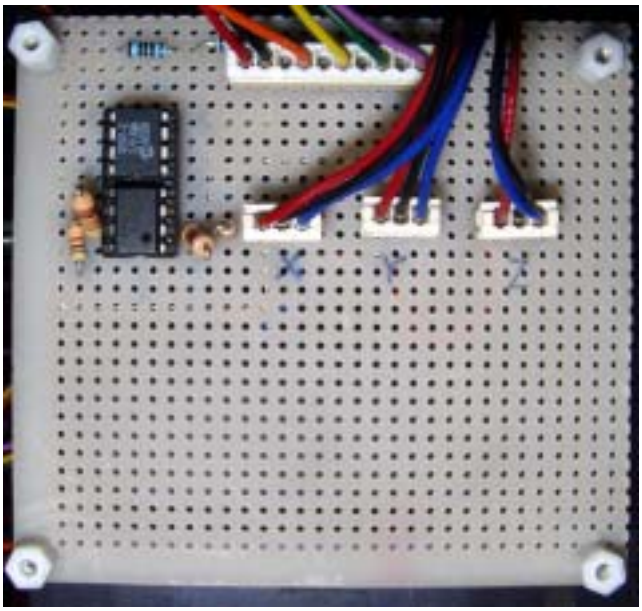


그림 3.4.6 솔레노이드밸브 제어용 릴레이 인터페이스 회로.

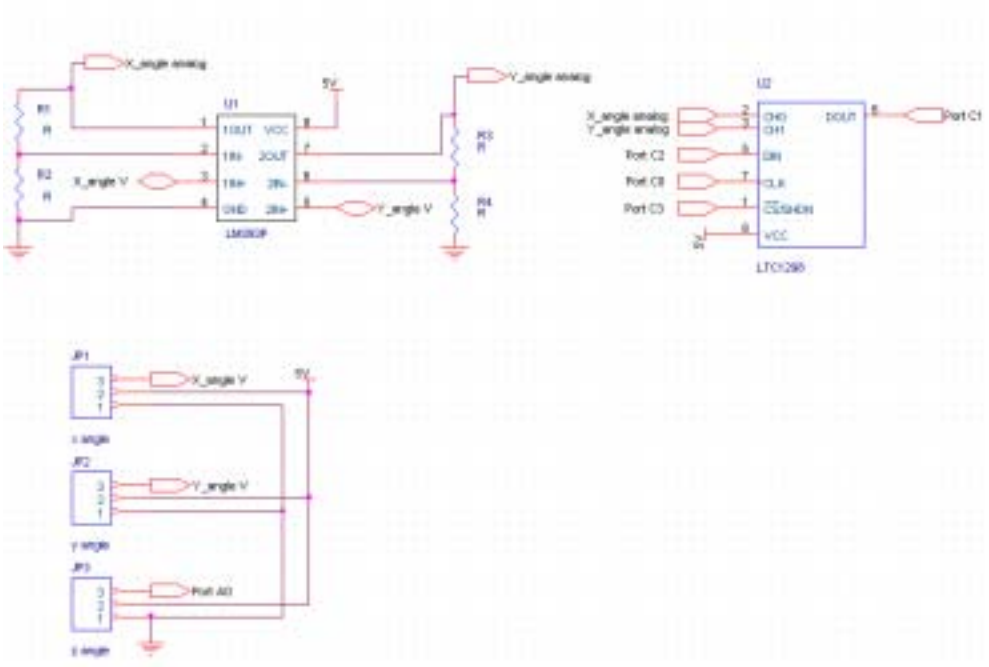
4. 주행경로결정 센서

주행경로결정 센서는 앞의 3장 2절에서 설명한 바와 같이 각도 측정 레버에 의해 움직이는 두 축에 장착된 로타리 포텐시오메터와 케이블 권취블록 중심축에 장착된 로타리 포텐시오메터 등 총 3개의 포텐시오메터에 대하여 전압부내회로로 입력회로를 구성하여 마이크로프로세서로 입력되도록 하였다. 그림 3.4.7은 실제 사용중인 인터페이스 회로로서 측면옵셀 및 트롤리 위치 측정용 포텐시오메터는 별도의 AD변환기 IC를 통해 입력되며 케이블 길이를 측정하는 포텐시오메터의 출력은 직접 PIC 18F488의 A/D 핀에 입력되도록 하였다. PIC 18 시리즈의 경우 다 채널의 AD변환을 수행할 때 안정적이지 못한 것으로 나타나 마이크로프로세서 전용 AD칩으로 개발된 LTC1298(Linear Technology)을 사용하였다. 이것은 12bit의 분해능을 갖으며, 2 채널의 AD 변환이 가능하며, 5V로 작동 가능하고 60μs의 변환 시간, 11.1 ksps 샘플링율을 갖고 있다. 특히 PIC 마이크로프로세서 제조사인 Microchip사에서 드라이버를 제공해 주기 때문에 프로그래밍 등이 간편하다는 장점이 있다. 그림 3.4.7의 인터페이스 회로도에 나와 있는 것 같이 LTC1298의 출력선(Dout)은 serial data link를 통해 마이크로프로세서로 입력되며, 마이크로프로세서로부터의 AD 변환 시작 신호와 클록 신호를 공급받도록 되어 있다.

또한, 본 시스템에서의 각도 측정 레버의 작동 범위는 약 ±30°로 제한되기 때문에 포텐시오메터의 출력단의 전압 변화는 크지 않았다. 따라서, 각도 측정의 분해능을 증가시키기 위하여 Op. Amp(LM393P)를 이용하여 주어진 작동 범위에서 포텐시오메터의 전압출력이 0 ~ 5V가 되도록 신호가공(signal conditioning)을 한 후 LTC1298에 입력되도록 하였다.



(a) 인터페이스 회로



(b) LM393P 및 LTC1298 인터페이스 회로도

그림 3.4.7 주행경로센서 인터페이스 회로.

5. 경사 센서

지면의 굴곡에 따른 차량의 경사도를 측정하기 위한 2축 경사 측정 센서도 출력은 아날로그전압이다. 따라서, 주행경로 측정 센서에서와 같이 LTC1298을 이용한 인터페이스 회로를 구성하여 차체의 롤과 피치를 측정하였다. 그림 3.4.8은 사용된 경사센서의 인터페이스 회로이다.

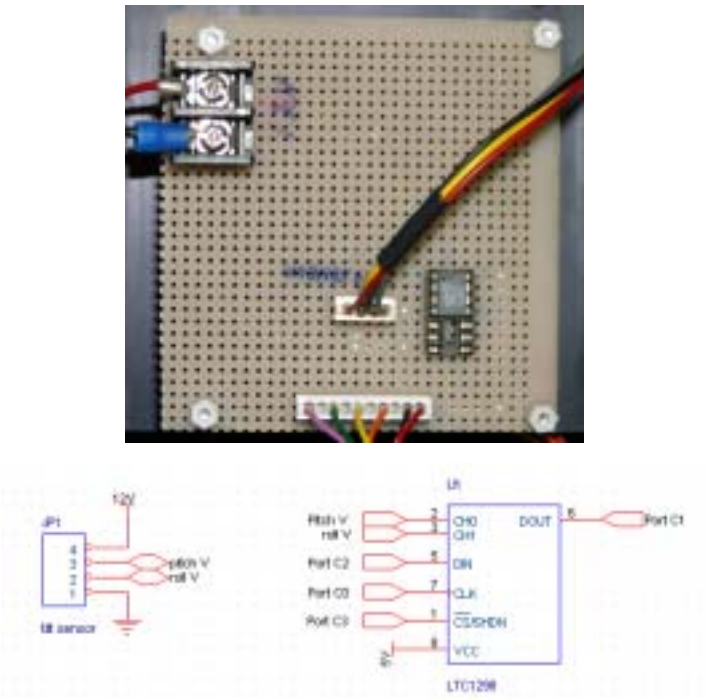


그림 3.4.8 경사센서 인터페이스 회로.

6. 초음파 센서

무인 주행 차량의 전, 측방에 위치한 장애물을 감지하여 차량이 장애물과 충돌하는 것을 방지하기 위한 하나의 안전 센서로서 초음파센서를 사용하였다. 초음파센서는 어떤 목표물까지의 거리를 측정하는 것으로서 발사된 초음파 빔의 범위(±9°)안에서 센서와 가장 가까이 있는 장애물까지의 거리를 측정하게 된다. 본 연구에서는 초음파 센서로부터 약 50 cm 이내에 장애물이 있으면 경보를 발

령하고 충돌 상황이 지속된다면, 즉 장애물까지의 거리가 가까워지면 차량을 정지시킬 수 있도록 하였다.

총 4개의 초음파센서가 차량의 선단 좌우측에 하나는 전방을 지향하고 다른 하나는 측방을 지향하도록 설치하였다.

가 센서의 특성

본 연구에서 사용된 초음파 모듈은 SensComp사의 600 series smart sensor인 SE-600-1이다(그림 3.4.9). 스마트 센서는 Polaroid사의 6500 series sensor를 계량한 제품으로 다음과 같은 특징을 가지고 있다.

- 송·수신 일체형으로 측정가능 거리는 0.15 m ~ 10.7 m이다.
- 거리 측정 정확도는 전 구간에 걸쳐 약 ±0.1%정도이며 연속 발신율은 약 5 Hz이며 외부에서 트리거될 수도 있다.
- 6 ~ 24V DC 전압이 필요하며, 초음파가 발사되는 동안 전류는 2 A, 발사 후에는 55 mA가 요구된다.

그림 3.4.9 SE-600-1초음파의 실제 모습

나. 거리 측정 원리

음파는 공기 중에서 일정한 속도로 진행하며 어떤 물체에 부딪히면 반사되어 돌아온다. 즉 특정 파장대의 초음파를 발사한 후 목표물에 반사되어 돌아오는 시간을 측정하면 초음파가 해당 목표물까지 왕복하는 데 소요되는 시간 (Δt)

이므로 정확한 음속($V_{ultrasonic\ wave}$)을 안다면 왕복한 거리를 구할 수 있다. 따라서 장애물까지의 거리, D_u 는

$$D_u = \frac{V_{ultrasonic\ wave}\Delta t}{2} \tag{3.4.1}$$

이 된다.

음파의 공기중 전파속도는 주변 온도에 따라 영향을 받는다. 즉 주어진 온도에 대하여 전파속도, $V_{ultrasonic\ wave}$는 다음 식과 같다.

$$V_{ultrasonic\ wave} = 331.4 + 0.6T \tag{3.4.2}$$

여기서, T ; 주변온도(℃).

즉, 보다 정확한 거리 측정을 위해서는 초음파센서에 온도 보정을 하여야 한다.

다. 온도 보정

온도센서로써 반도체 온도센서인 LM35를 사용하였다. 그림 3.4.10의 특성곡선에서 보는 바와 같이 LM35는 매우 선형적이며, 1℃당 10mV의 출력전압이 변화한다.

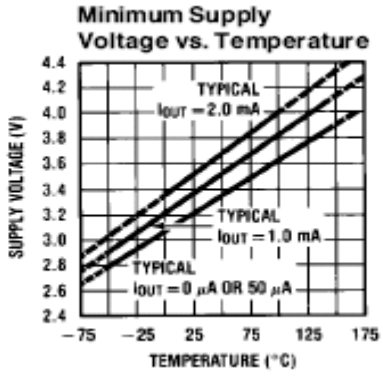


그림 3.4.10 온도별 전압변화

온도를 10℃씩 변화시키며 10bit A/D 변환기로 LM35의 출력을 측정한 결과,

AD 변환된 디지털 값과 실제 온도와의 관계식은

$$T = 0.4903\,AD + 0.8928 \tag{3.4.3}$$

이때 r^2는 0.9988이었으며, 여기서 AD는 LM35의 AD변환된 디지털 값이다.

따라서, 온도 보정을 고려하여 음파의 속도를 구하면,

$$V_{ultrasonic\ wave} = 331.4 + 0.6 \cdot (0.4903\,AD + 0.8928) \tag{3.4.4}$$

이 된다.

다. 초음파센서와 마이크로프로세서의 인터페이스

PIC 마이크로 프로세서를 이용하여 초음파센서를 구동하였다. 본 연구에서 사용한 초음파센서는 외부로부터 INIT신호를 받으면 초음파를 발사하고 목표물에 반사된 초음파를 수신하게 되면 ECHO신호를 외부로 출력한다. 이 INIT신호와 ECHO 신호 사이의 시간 차가 바로 초음파센서가 목표물까지 왕복하는데 걸리는 시간이다. 식(3.4.4)와 식 (3.4.1)을 이용하여 목표물까지의 거리를 측정할 수 있다.

사용된 초음파센서가 모두 4개이므로 각각을 작동할 수 있는 INIT 신호선 4개를 디지털 출력 단자에 연결하고 각 초음파센서로부터의 ECHO 신호는 PIC의 외부 인터럽트 핀으로 연결하였다. 즉, INIT 신호가 High가 될 때 초음파가 발사되며 동시에 타이머2를 가동하여 외부인터럽트가 걸릴 때 타이머2를 중지한 후 타이머2로부터 경과 시간을 읽어 들이면 된다.

한편, 초음파센서는 초음파 발사시 많은 전력을 요구하므로 4개의 초음파센서를 동시에 작동하지 않고 순차적으로 작동시켰으며 따라서 4개의 초음파센서로부터의 ECHO 신호를 OR gate로 합하여 마이크로프로세서에 입력하여도 현재 작동중인 초음파센서로부터의 ECHO 신호가 인터럽트로 걸리게 되는 구조로 인터페이스를 구성하였다. 초음파 센서는 INIT 신호가 들어오기 전에 최소 8 ms의 안정시간이 요구되며 INIT 신호 High 유지 시간을 30 ms로 셑팅함으로써 약 5 m까지의 거리를 측정할 수 있도록 회로를 설계하고 구동 프로그램을 개발하였다. 따라서, 4개의 초음파 센서가 1회 장애물을 스캔하는 데 소요되는 시간은 152 ms 가 된다.

4개의 초음파센서를 순차적으로 구동시키기 위하여 그림 3.4.11에서 보는 바와 같이 INIT 신호를 발생시켰다. 그림 3.4.12는 회로도로서 INIT 신호(RB1 - RB4) TR을

거쳐 각 초음파센서의 INIT선으로 연결되며 각 센서로부터의 ECHO 신호는 7432 OR gate를 통해 외부인터럽트(RB0/INT)롤 입력된다. LM35 온도센서는 AD 입력 0번 채널로 마이크로프로세서에 인터페이스된다. LCD를 추가하여 실시간으로 측정 거리의 확인이 가능하도록 하였으며 MAX232를 이용하여 시리얼포트 한 개를 설치하고 시리얼통신으로 외부 출력이 가능하도록 하였다.

그림 3.4.13 - 3.4.15는 본 초음파센서 거리 측정 시스템을 구동하기 위한 측정 알고리즘의 순서도를 나타낸 것이다. 필요한 경우 주행속도 측정하기 위한 궤도 구동 스프로켓에 설치된 로타리엔코더 신호를 입력(타이머 0)받을 수 있도록 하여 차량이 일정한 거리를 진행할 때마다 거리를 측정할 수 있도록 하였다.

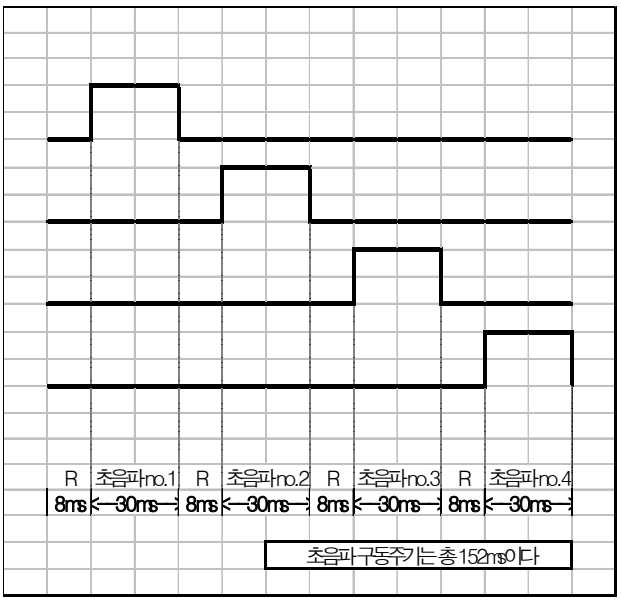


그림 3.4..11 4개의 초음파센서에 대한 구동주기

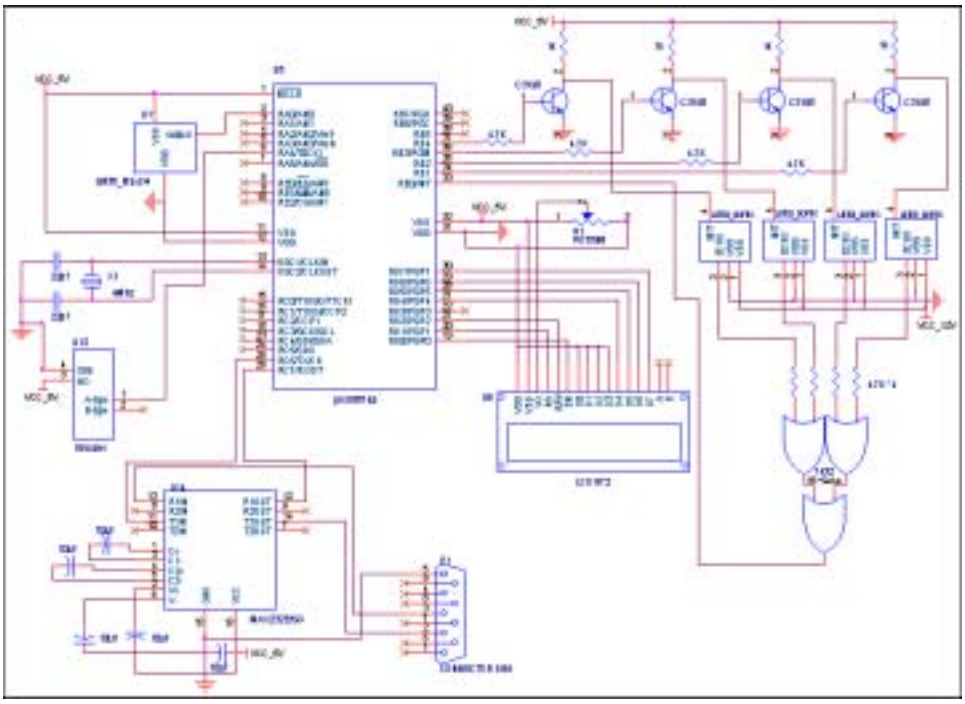


그림 3.4.12 초음파센서 구동 시스템의 회로도 .

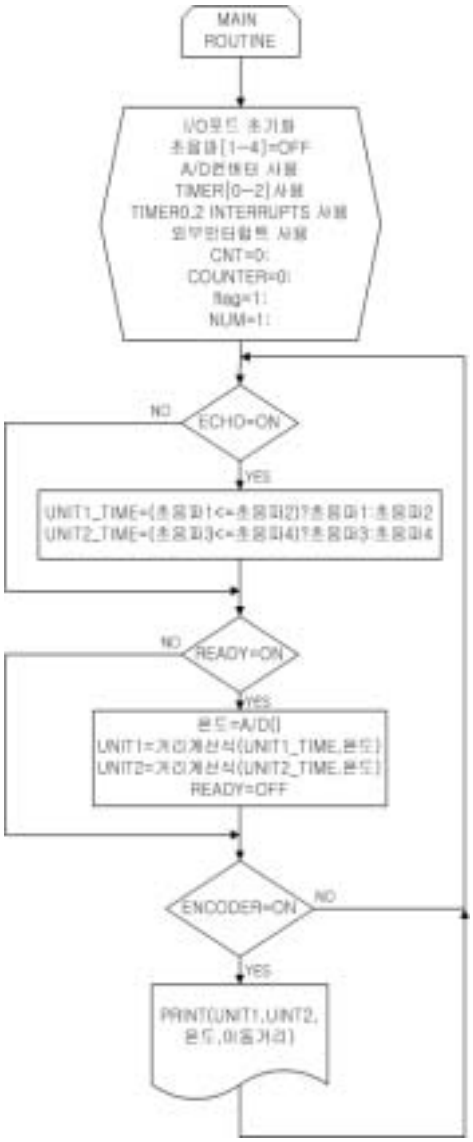


그림 3.4.13 Main 루틴.

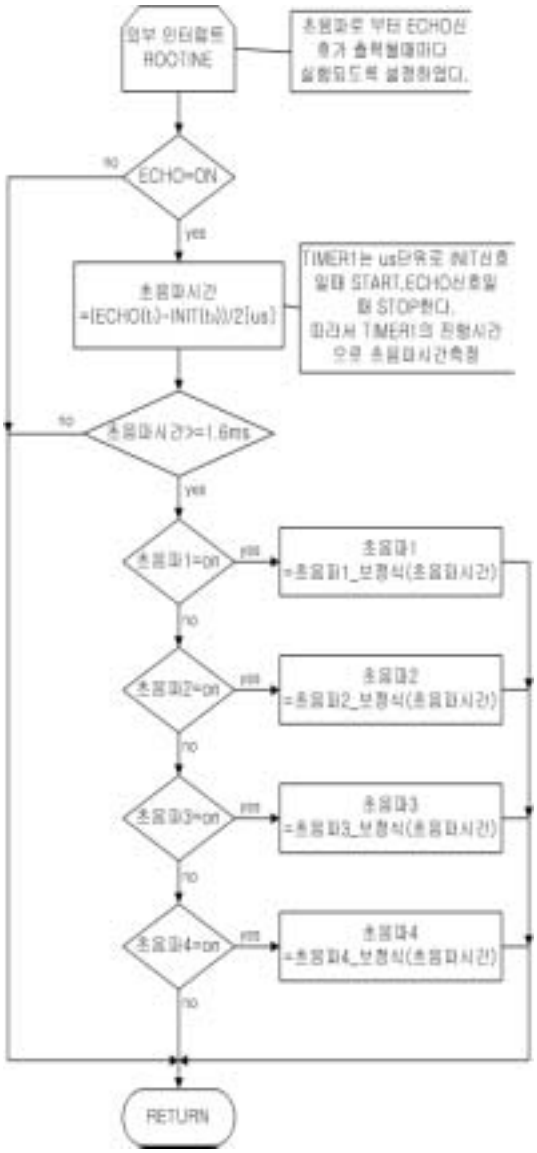


그림 3.4.14 외부인터럽트 루틴.

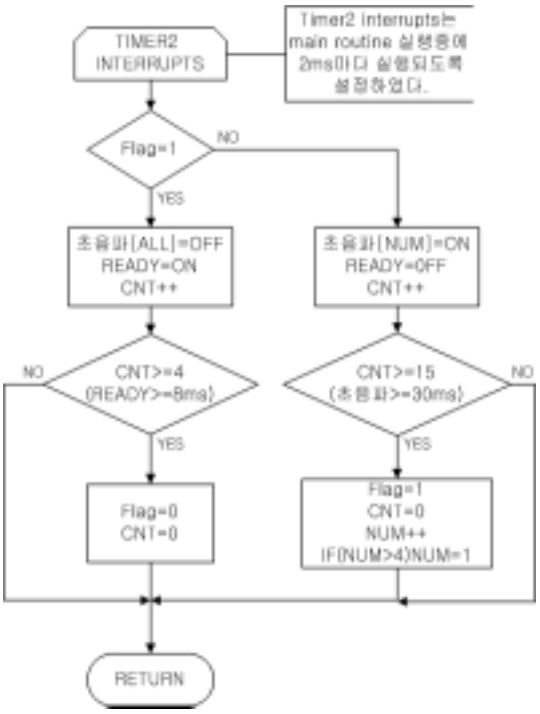
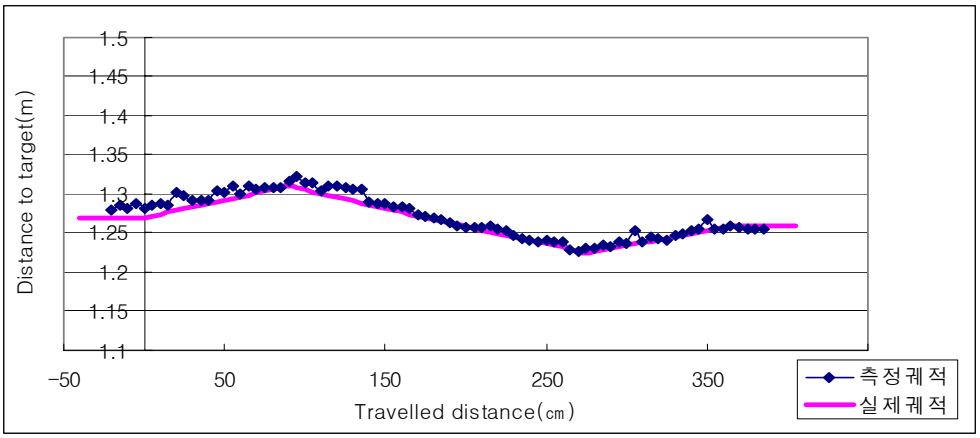


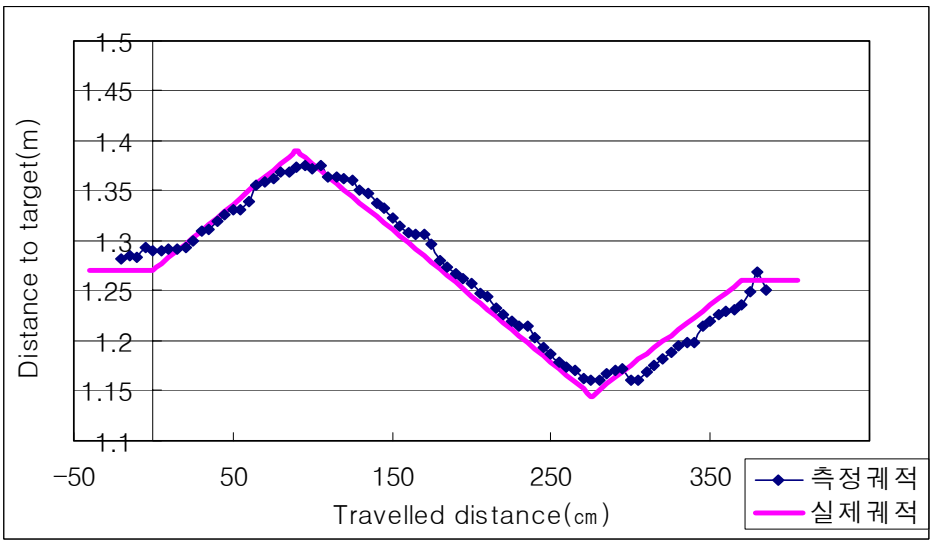
그림 3.4.15 타이머2 인터럽트 루틴.

마. 성능 평가

초음파센서가 스캔하며 이동하는 측면에 평판으로 목표물을 설치하고 초음파 센서를 목표물로부터 멀어지다가 다시 가까워지게 하면서 거리를 측정하였다. 이때 목표물까지의 최대거리 및 최소거리 차이는 8 cm 이었다. 그림 3.4.16은 실제 거리와 측정거리를 나타낸 것으로, 목표물까지의 실제 거리와 초음파센서에 의해 측정된 거리의 편차를 RMS 값으로 구해 본 결과 그림 3.4.16(a)의 경우는 약 8 mm, (b)의 경우는 13 mm 로 나타나 매우 정확하게 목표물까지의 거리를 측정한다고 판단하였다.



(a) 최소 및 최대 거리 차 : 8 cm



(b) 최소 및 최대 거리 차 : 24 cm

그림 3.4.16 초음파측정시스템의 거리 측정 성능

7. 주행속도/이동거리 측정 센서

가. 로타리엔코더

1) 작동 원리

궤도 차량의 주행속도를 측정하기 위하여 그림 3.1.25에서와 같이 로타리엔코더를 구동 스프로켓의 축에 커플러를 이용하여 직결한 후, 궤도 지지부에 고정하

였다. 본 연구에서 사용한 로타리엔코더는 12V DC 전원으로 구동되며 1회전 당 500 펄스를 출력하기 때문에 단위 시간 동안 몇 개의 펄스가 입력되었는 지를 측정하면 궤도부의 회전속도 및 이동거리를 구할 수 있다.

마이크로프로세서에서는 타이머/카운터를 이용하여 외부로부터 입력되는 펄스의 수를 셀 수 있는 기능이 있다. 다만 타이머/카운터는 TTL 신호만의 입력이 가능하기 때문에 로타리엔코더에서 12V로 출력되는 펄스열을 TTL 신호로 다운해야 하며 동시에 정확한 구형파로 만들기 위한 신호처리 기능이 요구된다. 그림 3.4.17의 인터페이스 회로도에서 보는 바와 같이 로타리엔코더의 A상 출력을 TR을 거치게 함으로써 TTL로 정형화된 펄스를 마이크로프로세서에 입력할 수 있게 된다. 좌우측 구동 스프로켓에 설치된 로타리엔코더로부터의 출력은 타이머/카운터0와 타이머/카운터1에 각각 입력되며 타이머2를 이용하여 마이크로프로세서의 내부클록을 카운트하여 매 1 ms마다 오버플로 인터럽트를 발생시켜 인터럽트의 개수를 이용하여 측정 기준 시간을 만들었다. 즉, 50 ms 마다 타이머/카운터0와 타이머/카운터1의 값을 읽고 즉시 타이머들의 시정수를 초기화함으로써 연속적으로 회전수의 측정이 가능해 진다.

따라서, 각 로타리엔코더를 통한 구동 스프로켓의 회전수를 구하면,

$$rev\_speed = (\frac{timer\_value}{500})(\frac{60}{interval\_time \times 10^{-3}}) \text{ , [rpm]} \quad (3.4.5)$$

여기서, timer_value : 해당 타이머/카운터의 출력 디지털 값, interval_time : 측정 기준 시간으로 ms 로 나타낸다.

한편, 구동 스프로켓의 유효 반경, R_s은 약 8.5 cm 이므로, 차량의 주행속도는

$$ground\_speed = R_s \cdot w = R_s(\frac{2\pi}{60})rev\_speed \text{ , } \quad \text{[cm/s]} \quad (3.4.6)$$

이 된다.

한편, 주어진 시간동안의 이동거리는 차량의 주행속도에 기준시간을 곱하면 구할 수 있다.

$$travel\_distance = R_s(\frac{2\pi}{60})(\frac{timer\_value}{500}) \times 60 = R_s(2\pi)(\frac{timer\_value}{500}) \quad , [cm] \qquad (3.4.7)$$

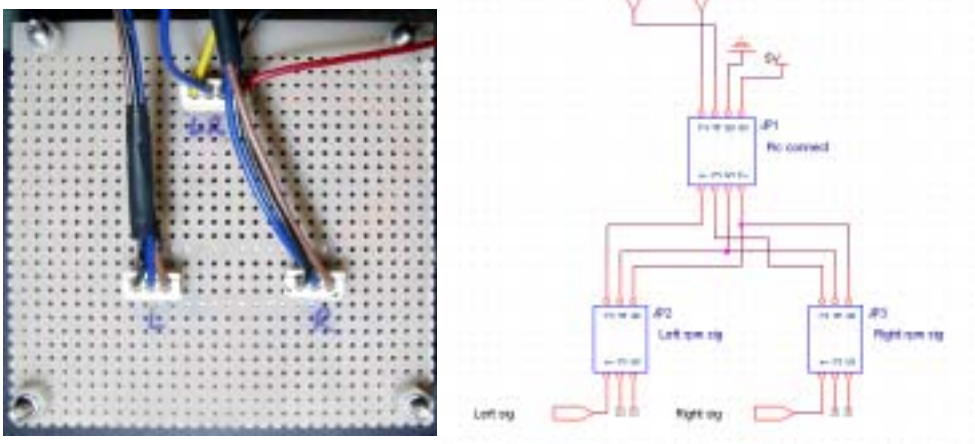


그림 3.4.17 로타리엔코더 인터페이스 회로.

2) 작동 성능

부변속 저, 고단 및 드로틀 레버 저, 중, 고속 위치에서의 좌우측 구동 스프로켓에서 측정된 회전속도는 표 3.4.1에 나타내었다.

표 3.4.1 주행속도 측정센서 데이터

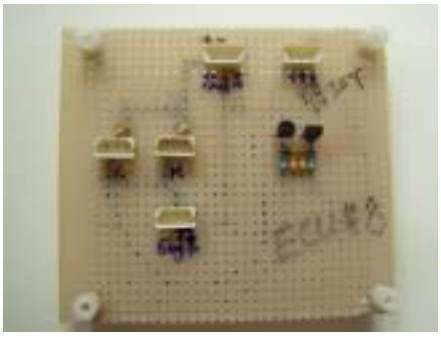
throttle / shift	slow		middle		fast	
	Left	Right	Left	Right	Left	Right
Low - 1	28	28	34	34	48	48
Low - 2	59	59	67	67	99	99
High - 1	47	47	54	54	80	80
High - 2	94	94	114	114	164	164

나. 근접스위치 센서 이용

궤도의 구동 스프로켓의 산과 골의 차이를 이용하여 궤도 지지부에 근접스위치를 장착하였다. 구동 스프로켓의 산 부분이 근접스위치에 접근하면 센서는 12V를 출력하며 골 부분에서는 0V를 출력하게 되므로 구동 스프로켓의 회전변위 및 회전속도를 쉽게 측정할 수 있다. 여기서도 근접스위치의 12V 신호를 TTL 신호로 전환하고 펄스의 정형화를 위하여 TR을 거쳐 마이크로프로세서의 외부 인터럽트에 입력되도록 하였다. PIC 마이크로프로세서는 외부 인터럽트 신호가 입력될 때 High-to-Low 또는 Low-to-High의 에지트리거 선택이 가능하나 T89C51CC01의 경우 High-to-Low 에지트리거만 가능하므로 초기 마이크로프로세서의 power-on-reset 시 펄스가 입력된 것으로 간주되므로 주의가 요망된다.

구동 스프로켓에는 총 8개의 이가 있으므로 차량의 이동거리를 계산할 수 있으며 주행속도는 스프로켓 이 사이의 간격을 두 펄스 사이의 시간을 측정하여 나누면 구할 수 있다.

좌우측 구동 스프로켓의 근접스위치는 각각 외부 인터럽트0와 외부 인터럽트1으로 입력되며, 그림 3.4.18은 근접스위치 센서의 인터페이스 회로도이다.



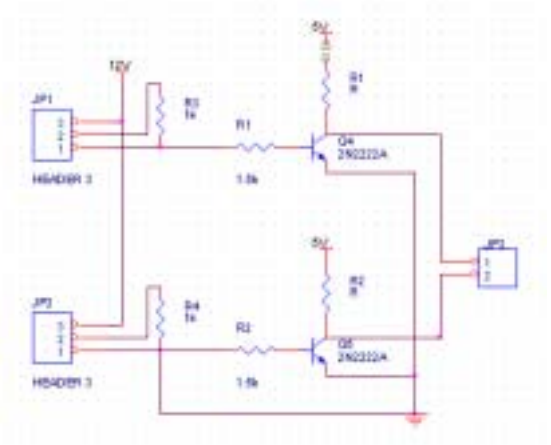


그림 3.4.18 근접스위치 센서의 인터페이스.

8. 과수열 시작/끝 감지 센서

과수열의 시작과 끝의 감지는 약액살포 노즐 제어를 위하여 꼭 필요한 정보이다. 본 연구에서는 오버헤드 가이던스 레일에 과수열의 시작과 끝을 알리는 표

식을 설치하고 트롤리의 좌우측에 광을 이용한 근접스위치 센서를 부착하여 그것을 인식하는 방법을 고안하였다.

마이크로프로세서와의 인터페이스는 그림 3.4.18에서와 같은 방식으로 과수열의 끝단에 도달하면 좌측 근접스위치 센서가 작동하도록 하고 곡선 구간을 돌아 새로운 과수열의 시작단에 오면 우측 근접스위치 센서가 작동하도록 하였다.

표식은 그림 3.4.19에 보는 바와 같이 함석판을 잘라서 오버헤드 가이던스를 고정하고 있는 과수원의 구조물에 걸어 두는 방식으로 만들었으며, 트롤리 좌우측에 근접스위치를 부착하였다.

그림 3.4.19 과수열 시작/끝 감지센서와 표식.

9. 약액탱크 수위 측정 센서

사용된 수위센서는 줄에 매달려서 약액탱크의 수위가 충분하면 눕혀져 있는 상태로 있다가 수위가 설정된 위치보다 떨어지게 되면 수위센서가 똑바로 줄에 매달리는 상태기 되어 스위치가 on되는 상황이 된다. 이 스위치의 신호를 마이크로프로세서의 디지털 입력 단자에 연결하여 매 50 ms마다 스위치의 상태를 읽어

on/off 상태를 CAN 버스로 송신할 수 있도록 하였다.

그림 3.4.20 약액탱크내 설치된 수위 센서.

10. 리모트 콘트롤

필요한 경우 운전자가 리모콘을 사용하여 과수방제기를 조작할 수 있도록 하기 위하여 리모콘과 제어시스템의 인터페이스를 제작하였다. 사용된 리모콘은 2개의 스틱을 상하 좌우로 움직여 서보모터를 제어하는 4 채널용으로 왼쪽 스틱은 조작후 손을 떼었을 때 중심위치로 복귀하는 구조이고 오른쪽 스틱은 상하 방향에 대하여 스틱을 둔 위치에 정지할 수 있는 구조이다. 따라서 왼쪽 스틱을 주행 및 조향제어용으로 설정하고 오른쪽 스틱은 과수방제기의 주행속도 제어용으로 설정하였다.

리모콘의 각 스틱 조작에 해당하는 포텐시오메터에서 선을 인출한 후, 89C51 마이크로프로세서로부터 2.5V 전원을 인가하여 스틱 위치에 따른 전압 출력이 되도록 입력회로를 구성하여 마이크로프로세서의 A/D 단자에 연결하였다. AD 변환된 데이터로부터 키의 위치를 확실하게 구분할 수 있도록 구간을 설정한 후 표 3.4.2에서 보는 바와 같이 키 조작에 따른 데이터를 할당하였다. 여기서, 3 채널의 데이터를 1 byte 크기의 데이터로 만들기 위하여 MainClutch, SpeedInfo 및 SteeringInfo의 데이터 3개를 bit OR하여 1 byte 크기의 데이터로 만들어 무선 시리얼버스를 통해 CAN 버스에 연결될 ECU로 송신하도록 하였다.

표 3.4.2 리모콘 키 조작에 따른 데이터 할당

			P2 : On/Off	
MainClutch	Off	0010 0000	R5/R4	L 스틱 하
	On	0011 0000	R4/R5	L 스틱 상
SpeedInfo	Stop	0000 0000		R 스틱 하
	Slow	0000 0100	R2/R3	〃
	Middle	0000 1000	R3/R2	〃
	Fast	0000 1100	R2R3/	R스틱 상
SteeringInfo	Left	0000 0010	R6/R7	L스틱 좌
	No action	0000 0011	/R6R7	L스틱 중
	Right	0000 0001	R7/R6	L스틱 우

그림 3.4.21 리모트 콘트롤러와 무선 송신용 마이크로프로세서 인터페이스.

11. 가상터미널

본 연구에서 가상터미널로 사용한 원보드컴퓨터는 cpu로 266MHz속도의 ARM920T을 사용하며 8, 16, 32 bit 버스 인터페이스를 위한 각각 별도의 메모리가 있어 주변장치와의 인터페이스가 용이하고 특히 110 x 78 mm 크기의 소형으로 고속연산을 요하는 경우, 그래픽 처리가 필요한 경우 등 일반 PC 급에서 가능한 기능을 배터리만을 사용하여 작동시킬 수 있다는 큰 장점을 갖고 있다. 10Mbps급의 LAN, USB, 채널, 3 개의 시리얼 채널을 갖고 있으며 특히 LCDTFT, 터치스크린 등을 편리하게 사용할 수 있는 시스템이다.
풍부한 라이브러리를 이용하여 C언어로 프로그램하며, ARM 컴파일러로는 여러 가지가 있으나, 가장 많이 사용되고 있는 것은 ARM 사의 ARM Developer Suite(ADS) 1.2와 Metrowerks 사의 CodeWarrior For ARM(CW ARM) 1.2이다. 사실, ADS와 CW ARM은 같은 컴파일러로 컴파일러는 ARM사에서 제작하였느나, 그 통합 환경은 Metrowerks 사에서 제작하였다.하지만 각 회사에서 소프트웨어 디버거(Armulator)을 따로 제작하여 내장하였다. 디버거 사용법은 ADS가 좀 편리하나, CW ARM에 비해 가격이 많이 높다. 본 ARM보드는 TFT LCD를 인터페이스하고 있으며 라이브러리로 지원을 하기 때문에 여타의 컨트롤러와 비슷하게 특수기능 레지스터를 초기화 시켜주고 라이브러리에서 지원되는 부함수를 인용하여 프로그램을 작성하면 된다. 작성된 프로그램은 ADS 컴파일러(평가판)을 이용하여 컴파일링을 하면 이미지 파일(*.rom 혹은 *.bin)이 생성되고 생성된 이지미 파일을 CyberLab사에서 지원되는 ArmDown프로그램을 이용하여 ARM보드의 Flash메모리에 다운로드 할 수 있다. 다운로드가 완료되면 자동 리셋이되면서 프로그램이 실행이 되게 된다.

본 연구에서는 그림 3.4.22에서 보는 바와 같이 터치스크린에서 전체 제어 시스템에 대하여 Auto mode와 Remote_Control mode를 선택하고 또한 주행속도 설정을 위한 드로틀레버의 위치를 설정하는 2개의 입력부위를 두었고, 차량의 현재 측면옵셑과 방향각, 평균 주행속도, 좌우측 약액살포 노즐의 작동상태 및 약액탱크 충전 여부를 모니터 상에 출력할 수 있도록 프로그램하였다.

따라서, ARM보드에서 ECU로 출력하는 시리얼통신 데이터의 포맷은

TX : $M<mode>S<speed>*checksum<cr><lf>
여기서, mode = 0 ; Auto, 1 ; Remote
Speed = 1 ; slow, 2 ; medium, 3 ; fast
checksum ; XOR ?

으로 나타낼 수 있고, ECU에서 display를 위해 수신하는 시리얼통신 데이터의 포맷은

RX : $O<offset>H<heading angle>G<ground speed>L<left nozzles>R<right nozzles>T<tank empty>*checksum<cr><lf>
여기서, offset = -50 to 50 (정수)
heading angle = -20.0 to 20.0 (소수 1자리)
ground speed = 10.0 - 70.0 (소수 1자리)
left nozzles = xxx11111 (2진수 8자리 : 1 - nozzle on
0 - nozzle off
right nozzles = xxx11111 (2진수 8자리 : 1 - nozzle on
0 - nozzle off
tank empty? = 1 (2진수 1자리 또는 Y or N)
1 - yes
0 - no
checksum : XOR

으로 나타낼 수 있다.

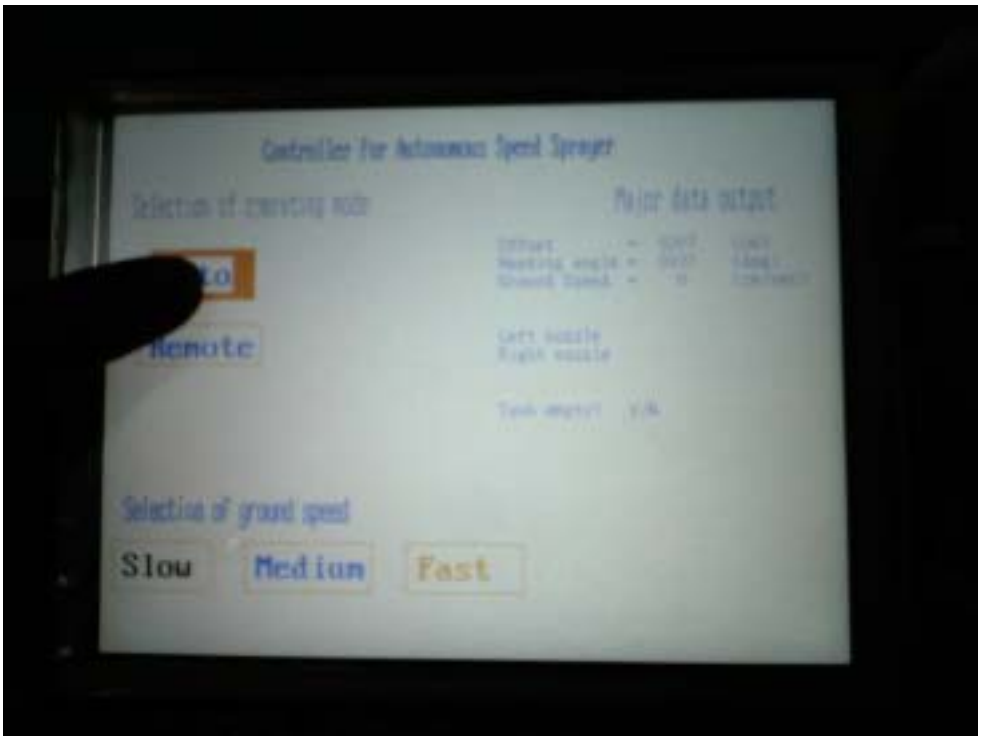


그림 3.4.22 가상터미널 설정 화면.

제 5 절 CAN 버스 시스템의 총합

앞에서 개발한 센서/조작기가 각각 해당 ECU에 인터페이스되고, 모든 ECU들을 CAN 버스에 연결함으로써 본 연구의 대상 농업기계인 과수방제기의 무인화를 시도하였다.

1. ECU #15 : 주행속도 센서

좌우측 구동 스프로켓에 장착된 각각의 로타리엔코더로부터의 신호를 입력받아 식 (3.4.5)에 의해 각각의 주행속도, N_L 및 N_R을 16 bit 정수형 데이터로 표현하여 4 byte길이의 하나의 메시지(Ground_speed)에 담아 폴링모드로 50 ms마다 CAN 버스를 통해 송신한다.

2. ECU #4 : 경사 센서

차체의 기울어짐은 2축 경사센서(Tilt sensor)로부터의 아날로그 전압출력을 LTC1298의 AD 변환을 통해 읽어 들인 후 식 (3.2.6)과 (3.2.7)의 측도설정식을 이용하여 각도로 변환하여 롤 및 피치각을 부호있는 16bit 데이터로 만들어 4 byte길이의 하나의 메시지(TiltSense)에 실어 폴링모드로 50 ms마다 CAN 버스에 송신한다.

이때, 소수점 이하 한자리까지를 유효한 각도로 간주하여 측정된 각도에 10을 곱한 후 정수형으로 casting 한다. 이 메시지를 수신한 ECU에서 경사센서의 데이터를 활용할 때는 다시 10으로 나누어 실수형 데이터로 casting하여 사용한다.

3. ECU #3 : 주행경로 결정 센서

ECU#3은 경사 센서(ECU #4)와 주행속도 측정 센서(ECU #1)로부터 송신되는 메시지를 인터럽트 모드로 수신하여 cpu가 다른 임무를 수행하는 동안에도 수신 데이터가 overrun하지 않도록 하였다. 또한 주행경로 결정센서의 Inclinometer의 x축, y축 각도를 측정하는 포텐시오메터로부터의 아날로그 전압 출력을 LTC1298을 통해 AD 변환하고, 케이블의 길이는 ECU 자체의 AD 변환기로 변환하여 식 (3.2.3), (3.2.4) 및 (3.2.5)의 측도 설정식을 통해 각도 및 길이로 환산된다. 한편, 측면옵셀을 결정하는 각도는 식 (3.2.8)에서와 같이 경사센서로부터의

롤 각으로 보정한 후 최종적으로 식 (3.2.1)과 같이 측면옵셀을 결정한다. 차체의 방향각은 일단 차량이 주행을 시작하고 난 후 주행속도 측정센서의 회전속도로부터 주어진 시간동안의 이동거리를 계산하여 차량의 직전 위치에서 구한 측면옵셀의 차이를 이용하여 식 (3.2.2)와 같이 결정된다.

현재의 측면옵셀과 방향각은 부호있는 16bit 데이터로 변환하여 4 byte길이의 메시지(TravelPath)에, 경사센서의 피치각을 이용하여 보정된 y축 각도 및 케이블길이도 각각 16 bit 데이터로 표현되어 4 byte길이의 메시지(TrollyPath)에 실려 50 ms마다 폴링모드로 CAN 버스에 송신된다.

4. ECU #1_1 : 퍼지 제어기

퍼지 제어기 ECU는 주행경로 결정센서(ECU #3)로부터 TravelPath라는 메시지를 수신 받아 차량의 현재 자세인 방향각과 측면 옵셀에 따라 작동할 클러치의 선택과 클러치 작동시간을 결정하는 기능을 수행한다.

본 연구에서 무인주행 제어는 기본적으로 측면옵셀과 방향각을 입력변수로 하고 클러치 작동시간을 출력변수로 하는 퍼지 제어 알고리즘으로 수행된다. 제어 알고리즘은 다음의 7절에서 상세히 설명될 예정이고, 본 연구에서는 Matlab을 이용하여 퍼지 알고리즘을 구현한 후 방향각과 측면 옵셀값을 변화시키며 시뮬레이션을 통하여 각각의 경우에 대한 클러치 작동 시간을 구한 후, 이것을 look-up table로 만들어 방향각과 측면옵셀이 주어지면 바로 look-up table을 참조하는 방식으로 프로그램하였다. 특히 마이크로프로세서의 메모리 용량은 제한되어 있으므로 look-up table 자체도 부호있는 8bit로 만들기 위해 시뮬레이션을 통해 얻어진 퍼지 제어기의 기존 클러치 작동시간을 4로 나누어 사용하였다.

따라서, 퍼지 제어기 ECU는 측면옵셀과 방향각 데이터를 수신하여 look-up table을 참조하고 tableml 출력에 다시 4를 곱하여 부호있는 16 bit 데이터로 변환한 후 메시지(Outtime)에 실려 50 ms마다 폴링모드로 CAN 버스를 통해 송신된다.

5. ECU #1 : 조향/주행 클러치 제어기

조향/주행 클러치 제어기는 과수방제기의 주행제어를 위한 주클러치/브레이크의 작동 및 조향제어를 위한 좌우측 조향 클러치의 작동을 제어한다.

ECU #1은 CAN 버스를 통해 한 메시지(joysticksig)에서 주행제어 정보로서 주클러치/브레이크 작동 신호를 전달 받아 해당 유압실린더를 조작하여 차량을 출발하게 하거나 정지시킨다. 또한 메시지(Outtime)을 수신하여 조향제어 정보로서 클러치 작동시간 데이터를 전달받는다. 이 메시지를 통해 전달되는 클러치 작동시간(out-time)은 부호 있는 16 bit 정수형 데이터로 - 값은 좌회전, 즉 왼쪽 클러치를 클러치 작동 시간만큼 disengage시키고, +값을 갖을 때는 오른쪽 클러치를 주어진 시간 동안 disengage 시켜 필요한 각도만큼 우회전이 되도록 한다. 만일 클러치 작동시간이 0이라면 조향을 하지 말라는 명령이므로 과수방제기는 제어주기 시간동안 그대로 직진을 한다.

ECU #1에서는 필요한 데이터들이 수신과정에서 overrun되어 유실되는 것을 방지하기 위하여 인터럽트 모드로 수신한다. 본 ECU에서 수신되어 사용되는 메시지들이 대개 50 ms의 주기로 CAN 버스에 올라오게되므로 클러치의 제어도 대략적으로 50 ms마다 이루어진다고 할 수 있다. 그러나, 본 연구에서의 궤도형 주행장치는 조향클러치가 jaw 형식이므로 조향하기 위하여 조향클러치를 disengage 시킬 때 최소한 150 ms 이상의 시간이 필요하며 클러치가 완전히 disengage되기 전까지 즉 클러치의 이빨들이 아직 물려 있는 시간 동안은 직진을 계속하고 퍼지 제어기에서 송신한 out_time 동안 클러치의 disengage 상태를 유지하게 되며 이때 조향이 이루어지는 방향의 클러치가 정지하여 해당 궤도를 중심으로 disengage 유지시간동안 선회가 이루어지고 또 다시 클러치를 re-engage 시켜주기 위하여 최소 150 ms 이상의 시간이 필요하게 된다.

따라서, 조향/주행 클러치 제어기는 매 50 ms마다 차량의 자세를 확인하기는 하나 일단 조향이 필요하여 조향작용이 시작되면 CAN 버스를 통해 전달되는 메시지들을 참조하지 않게 되며 그것들은 매 50 ms 마다 최신의 것으로 갱신되다가 클러치가 조향 작용을 완료하고 첫 번 째 주기에서 비로소 조향 작용을 해야할지 말아야 할지를 결정하는데 사용된다.

이 외에도 ECU #1은 무인주행을 함에 있어서 꼭 필요한 위급 상황시 차량 정지시킬 수 있도록 장애물감지 여부 메시지(ObstaclSens), 비상 스위치 작동 여부 메시지(Emergency) 등을 수신한다.

6. ECU #2 : 트롤리 조작기

트롤리 조작기 ECU는 주행경로 결정센서(ECU #3)로부터의 메시지

(TrollyPath)를 수신받아 y축 각도를 분리한 후 제어 알고리즘에 의해 트롤리를 조작하는 인터페이스 회로를 제어한다.

7. ECU #8 : 이동거리 측정 센서

근접스위치를 사용한 차량의 이동거리를 측정하는 센서는 좌우측 구동 스프로켓의 이빨 하나가 넘어갈 때마다 해당 외부 인터럽트가 걸리게 하여 인터럽트 서비스루틴으로 실행이 넘어간 후 플래그를 메시지에 실어 CAN 버스에 송신하게 된다. 즉, 왼쪽 플래그 메시지(L_ground_dist)와 오른쪽 플래그 메시지(R_ground_dist)는 event가 발생하는 순간에만 CAN 버스에 메시지를 송신하는 방식이다.

8. ECU #5 : 과수열 끝/시작점 감지 센서

트롤리의 좌우측에 장착된 근접센서 스위치로부터의 출력이 ECU #8에서와 같은 방식으로 좌측 근접센서로부터의 메시지(Row_end)와 우측 근접센서로부터의 메시지(Row_start)가 event가 발생하는 순간에만 CAN 버스로 송신된다.

9. ECU #6.1 : 좌측 약액살포 제어기와 약액탱크 수위센서

좌측 액액살포 제어기 ECU는 ECU #5로부터 과수열이 시작되는 메시지(Row_start)와 과수열이 끝나는 메시지(Row_end), ECU #8으로부터 이동거리 측정용 좌측 플래그 메시지(L_ground_dist)를 수신하여 이 ECU에 인터페이스되어 있는 노즐 솔레노이드 밸브 구동용 릴레이를 제어하게 된다.

한편, 약액탱크 수위센서는 탱크내의 약액이 일정 수준 이하로 떨어지면 작동하는 스위치 방식 센서로서 매 100 ms 마다 스위치 작동 여부를 확인하여 스위치 on/off 여부를 나타내는 플래그를 메시지(TankLevel)에 실어 CAN 버스에 송신한다.

10. ECU #6.2 : 우측 약액살포 제어기

우측 액액살포 제어기 ECU는 ECU #5로부터 과수열이 시작되는 메시지(Row_start)와 과수열이 끝나는 메시지(Row_end), ECU #8으로부터 이동거리 측정용 우측 플래그 메시지(R_ground_dist)를 수신하여 이 ECU에 인터페이스되어

있는 노즐 솔레노이드 밸브 구동용 릴레이를 제어하게 된다.

11. ECU #7 : 장애물감지 센서 및 비상정지 스위치

장애물 감지 센서 ECU는 초음파 센서 4개와 인터페이스되어 있으며 4개중 하나의 초음파센서에라도 장애물이 감지되면 장애물감지 플래그를 메시지(ObstaclSens)에 실어 CAN 버스에 송신한다. 또한 비상정지 스위치의 눌림 여부를 확인하여 비상정지 플래그를 메시지(Emergency)에 실어 CAN 버스에 송신하게 되는데 이는 매 50 ms 마다 폴링모드로 송신된다.

12. ECU #14 : 리모트콘트롤 조작기(리모콘 수신부)

리모트콘트롤 조작기 ECU는 리모콘과 연결된 마이크로프로세서로부터의 리모콘 키 값에 대응하는 데이터를 Bluetooth(무선 시리얼 통신 장비)를 통해 시리얼로 전달받아 주클러치 조작, 드로틀제어기 조작, 조향클러치 조작에 관한 정보를 8 bit의 정수형 데이터로 만들어 메시지(joysticksig)를 폴링모드로 50 ms마다 CAN 버스에 송신한다.

13. ECU #14_1 : 리모트콘트롤 조작기(리모콘 송신부)

본 연구에서 사용한 리모트콘트롤러는 4 채널의 키 값을 전달할 수 있는 것으로서 각 키 조작에 대응하는 주클러치 조작, 드로틀 제어기 조작 및 조향클러치 조작에 대한 정보를 한 byte 크기의 데이터로 만들어 무선 시리얼통신을 통해 ECU #14로 전달한다.

14. ECU #12 : 시리얼-CAN통신 변환기

시리얼-CAN 통신 변환기 ECU는 CAN 버스를 가상터미널(Virtual terminal)에 연결하기 위하여 사용되었다. 가상터미널에서 무인주행을 위한 자동운전모드와 리모콘조작 모드 및 주행속도 설정 정보를 시리얼을 통해 입력 받아 1 byte의 데이터로 만들어 메시지(SessionControl)를 만들어 폴링 모드로 50 ms마다 CAN 버스로 송신한다. 또한 무인주행 제어기의 작동과 관련한 주요 정보들을

CAN 버스로부터 수신하여 시리얼통신을 통해 가상터미널에 보내 모니터에 출력되도록 하였다.

15. ECU #14 : 드로틀 조작기

드로틀 조작기 ECU는 ECU #12의 메시지(SessionControl), ECU #14의 메시지(joysticksig)를 CAN 버스로부터 수신하여 드로틀 조작기와 관련된 주행속도 설정 정보만을 추출한 후 본 ECU와 인터페이스되어 있는 전동실린더를 제어하게 된다. 또한, 동시에 본 ECU의 인터페이스에 설치한 PB 스위치 작동에 의해서도 드로틀 조작기가 작동되도록 하였다.

16. ECU #11 : 지자기 센서

지자기 센서 ECU는 향후 지자기센서로부터의 데이터를 이용하여 보다 안정적인 제어를 구현하기 위하여 단순히 방향각 데이터만을 참조하기 위하여 CAN 버스에 추가하였다. 지자기센서(TCM2 Electronic sensor module, PNI Co., USA)는 시리얼통신을 통해 다음과 같은 포맷으로 마이크로프로세서에 측정 데이터를 송신한다.

$C<compass>P<pitch>R<roll>X<Bx>Y<By>Z<Bz>T<temp>E<error>*checksum<cr><lf>

지자기 센서 ECU는 위의 데이터를 입력받아 compass 정보만을 분리하여 2byte 크기의 데이터를 갖는 메시지(GeoSens)를 만들어 50 ms마다 폴링모드로 CAN 버스에 송신한다.

표 3.6.1에 본 연구에서 사용하는 메시지 맵과 특징 등을 정리하여 나타내었다.

모든 ECU들은 각자의 Firmware를 갖으며, 마이크로프로세서로 다운로드되어 실행된다. 앞에서 언급한 것 중 주요 ECU에 대한 Firmware를 부록에 수록하였다.

표 3.6.1 Message Map

ECU		Message	Signal	Start Bit	Bit Length	Unit	Range
1 PIC	RX	TravelPath (0x030)	X_Offset Heading_angle	50ms	16 16	mm deg	
		TiltSens (0x033)	Roll Pitch		16 16	deg deg	
		ObstaclSens (0x022)	Obstacl_flag				
		Emergency (0x020)	Emerge_flag				
		RemoteControl (0x050)	Remote_MainClutch Remote_SteeringInfo		16 16		On/Off Left/Right
2 PIC	RX	TravelPath (0x030)	Y_angle Z_length	50ms	16 16	deg mm	**Trolley motor control**
3 PIC	TX	**TravelPath** (0x030)	Heading_angle X_Offset Y_angle Z_length	50ms	16 16 16 16	deg mm deg mm	**Inclinometer**
4 PIC	TX	**TiltSens** (0x033)	Roll Pitch	50ms	16 16	deg deg	**Tilt sensor**
5 ?	TX	Row_end (0x025) Row_start (0x027)	Row_End Row_start	50ms	8 8		Prox. s/w (tree row)
6.1 Atmel	RX	Row_end (0x025) Row_start (0x27)	Row_End Row_start				**Left Nozzle**
		GroundDistance (0x65)	L_ground_dist				
	TX	**TankLevel** (0x036)	TankEmpty	50ms	1		**Tank level sensor**
6.2 Atmel	RX	Row_end (0x025 : First H) Row_start	Row_End Row_start				**Right Nozzle control**
		GroundDistance (0x65)	R_ground_dist				
7	TX	ObstaclSens (0x022)	Obstacl_flag	50ms	1		Obstacle sensor(U_sonic)
		Emergency (0x020)	Emerge_PbSW	on event	1		PB s/w

8 Atmel	TX	L_ground_dist (0x065) R_ground_dist (0x067)		on event	16 16		**Prox. sw on sprocket**
		(EngineSpeed) (0x062)	(Engine_spd)	100ms	8		E/G keybox
9 PIC	RX	SessionControl (0x010) RemoteControl (serial)	SpeedSetup				**throttle control**
	TX	RemoteControl (0x050)	Remote_SpeedInfo	on event	16		
11 PIC	TX	GeoSens (0x85)	compass	50ms	16		
12 PIC	TX	SessionControl (0x010)	SessionStart SessionEnd SpeedSetup	on event	1 1 2		control signal from Armboard (touch screen)
	RX	Ground_Speed (0x060)	Spd_Encoder_L Spd_Encoder_R				display to Armboard (Monitor)
14 Atmel	TX	RemoteControl (serial)	Remote_MainClutch Remote_SpeedInfo Remote_SteeringInfo	on event	2 2 2		**Remocon**
15 Atmel	TX	Ground_Speed (0x060)	Spd_Encoder_L Spd_Encoder_R	50 ms	8 8		**Rotary encoder**

그림 3.5.1은 각 ECU들이 과수방제기에 장착되어 네트웍을 구성한 모습을 보여준다.

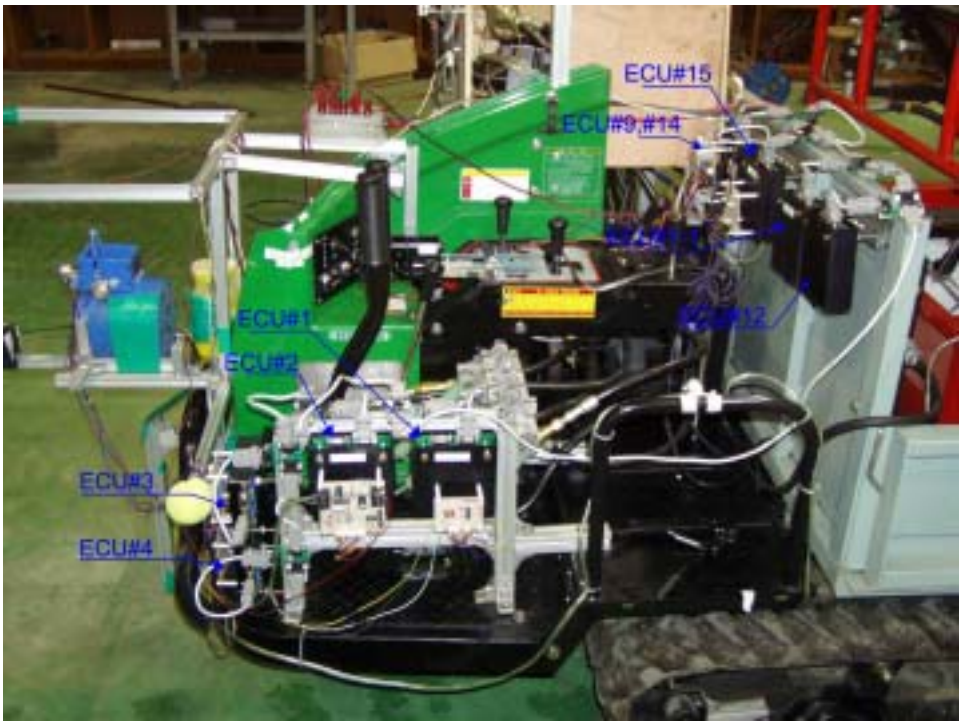


그림 3.5.1 각 ECU들이 과수방제기에 장착되어 네트웍을 구성한 모습.

제 6 절 무인주행/방제작업제어 알고리즘

1. 무인주행 제어 알고리즘

가. 제어 알고리즘

1) 작동 원리

무인 과수방제기의 조향제어는 주행경로 결정 센서로부터 구한 차량의 현재 자세, 즉 측면옵셑과 방향각에 의해 조향각이 결정되어 제어가 이루어지는 방식이다. 이때 조향각을 결정하기 위한 퍼지 제어기를 개발하였다.

궤도형 주행장치는 한 쪽의 조향클러치를 조작하여 동력을 차단(disengage)하게 되면 반대편 궤도로만 동력이 전달되어 동력이 차단된 쪽의 궤도를 중심으로 클러치를 조작하고 있는 시간만큼 조향을 하게 된다 특히 궤도형 주행장치의 조향클러치는 jaw 타입으로 클러치의 이가 완전히 빠질 때까지는 아직 조향이 되지 않는 상태로 직진하다가 이가 빠지는 순간부터 조향을 시작하는 구조이다. 이런 종류의 차량들은 클러치가 끊어져 있는 동안에만 방향이 전환되는 메카니즘의 특성 때문에 실제의 방향전환 각은 주어진 차량속도에 대해 클러치가 끊어져 있는 시간에 의해서 결정이 된다. 그러나, 이미 언급했던 것과 같이, 클러치를 한 번 작동하기 위해서 시스템은 유압실린더를 밀고 다시 당겨야 하므로 350ms의 최소 시간이 필요하게 된다. 만약에 350ms보다 작은 시간을 클러치 동작에 할애한다면 시스템의 성능에 역효과를 야기 시킬 수 있는데 클러치가 아예 동작하지 않거나 그 시간동안 클러치의 동작을 중지 시킬 수도 없게 된다. 특히 대 부분의 농작업기와 마찬가지로 과수방제기의 조향 클러치나 작동 메카니즘 자체가 정교하게 제작된 것이 아니라 상황에 따라, 즉 부하 정도 및 노면 상태, 운전 조건 등에 따라 클러치 작동 시간 등이 변할 수 있다는 것이 정밀한 제어를 어렵게 한다.

그림 3.6.1은 본 연구에서 개발한 조향 제어 알고리즘의 순서도를 나타낸 것이다. 시스템의 제어주기는 CAN 버스를 통해 센서 데이터들이 전달되는 50 ms로 하고 조향클러치의 제어는 1 ms 단위까지도 가능하도록 되어 있다. 먼저 매 50 ms마다 인터럽트 신호에 맞추어 조향 제어기는 여타 센서로부터 조향 제어에 필요한 차량의 자세 정보를 받는다. 이 정보는 바로 퍼지 제어기로 입력되어 몇 도의 각도로 어느 방향으로 조향해야 하는 지의 정보를 출력 받는다. 만일 조향

할 필요가 없다면 차량은 제어주기 동안 직진하고 다시 인터럽트 서버스 루틴으로 들어와 자세 정보로서 조향정보를 구하게 된다. 만일 차량이 조향되어야 할 필요가 있으면 작동 시켜야 할 클러치를 선택하고 조향 작용을 시작한다. 앞에서 설명한 바와 같이 jaw 타입 클러치는 완전히 클러치가 disengage될 때까지 다소간의 시간이 필요한데(disengage_time), 본 과수방제기의 경우 좌측 클러치는 약 150 ms, 우측 클러치는 약 200 ms 정도이다. 엄밀히 말하면 이 시간 동안에 차량은 직진을 계속하게 되며, 클러치가 완전히 disengage 된 후 클러치를 잡고 있는 시간만큼 주행속도에 따라 차량의 자세는 회전된다. 이후 클러치를 re-engage 시키기 위하여 disengage시 만큼의 시간이 필요하며 당연히 이 시간 동안 차량은 다시 직진을 하게 되며 한 사이클의 조향 작용을 완료하게 된다. 즉, 조향 작용이 한 번 시작되면 제어 주기인 50 ms는 무의미해지며 한 사이클의 조향 작용이 완료되고 나서야 비로소 시스템은 다시 제어 주기에 동기하게 된다. 그러나, 클러치 조향작용이 이루어지는 동안에도 차량의 자세 제어 값은 계속 입력되며 항상 최신의 것으로 업데이트되어 시스템이 언제든지 참조할 수 있다는 것이 본 연구에서 CAN을 이용하는 큰 장점이라고 할 수 있다.

2) 조향 클러치의 작동 특성

과수 방제기의 조향클러치 작동 특성을 규명하기 위하여 실험실에서 과수방제기를 주행시키며 일정시간 클러치를 disengage 시켰을 때 초기의 방향각과 조향이 완료된 후의 방향각의 차를 조향각으로 나타내었다.

그림 3.6.2와 그림 3.6.3은 좌우측 클러치의 작동 특성을 나타낸 것이다. 클러치 작동시간이 적은 경우에는 작동시간과 조향각 사이에 비선형적인 관계가 있는 것으로 보이나 작동시간이 길어지면서는 작동시간과 조향각 사이에 선형적 비례관계가 있는 것으로 나타났다. 이것은 너무 짧은 시간의 조작은 해당 궤도에 동력이 충분히 차단되고 또 전달되고 하는 것이 일정치 못하며 과수방제기에 적용된 궤도형 주행장치의 한계이고 또한 조향각도를 측정할 때 너무 적은 각으로 선회를 했기 때문에 나타날 수 있는 오차로 판단된다. 그래서, 본 연구에서는 주어진 주행속도 조건에서 조향각과 클러치 작동시간에 대한 선형회귀식을 구하여 제어에 이용하였다. 또한 동일한 클러치 작동시간에서 주행속도에 따른 조향각도 매우 선형적인 것으로 나타났다. 즉, 주행속도가 고속인 경우를 기준으로 차량의 주행속도를 달리했을 때 동일한 조향각을 얻기 위해서는 고속의 경우의 데이터

에 일정한 요소를 곱하는 것만으로도 주행속도에 연관된 제어 시스템을 만들 수 있을 것으로 판단하였다. 즉, 좌측 클러치의 경우 조향각 10°를 얻기 위하여 주행속도가 고속인 경우 283 ms의 작동시간이 필요하나 중속의 경우 430 ms, 저속의 경우 638 ms가 필요하므로 이는 고속의 1.8배와 2.3배를 해 주면 된다. 우측 클러치의 경우는 고속인 경우 273 ms의 작동시간이 필요하나 중속의 경우 454 ms, 저속의 경우 598 ms가 필요하므로 이는 고속의 1.7배와 2.2배를 해 주면 된다.

3) 퍼지 제어기

직접 과수 방제기를 운전하면서 여러 가지 차량의 자세로부터 어떻게 조향을 해야 하는 지에 대한 검토를 수행하여 그것에 기초한 퍼지 제어기를 개발하였다. 운전자는 결국 차량의 측면옵셑과 방향각을 보면서 조향을 하게 되므로 두 변수를 퍼지 제어기의 입력변수로 사용하였다. 먼저 측면옵셑은 주행 기준선에 대하여 좌측으로 -50 cm, 우측으로 + 50 cm 이내에서 운전하는 것을 목표로 하고 차량의 방향각 또한 주행기준선 방향을 중심으로 우측을 향할 때 + 20°, 좌측을 향할 때 -20°를 운전 중 차량이 취해질 수 있는 방향각으로 보고 퍼지 입력 변수들의 멤버쉽 함수를 결정하였다. 그림 3.6.4(a)에서 보는 바와 같이, 측면 옵셑에 대한 입력변수에서 언어변수로 offset_left_large(OLL), offset_left_small (OLS), offset_zero(OZ), offset_right_small(ORS), 그리고 offset_right_large(ORL) 등의 5가지 였으며, 각각은 삼각형 혹은 사다리꼴의 멤버쉽 함수를 갖도록 정의하였다. 마찬가지로 그림 3.6.4(b)에서와 같이 방향각도 heading_left_large(HLL), heading_left_small (HLS), heading_zero(HZ), heading_right_small(HRS), heading_right_large(HRL) 등의 5가지 였으며, 각각은 삼각형 혹은 사다리꼴의 멤버쉽 함수를 갖도록 정의하였다. 퍼지 제어기의 출력은 유압실린더를 움직여 클러치가 끊긴 후 유지되는 시간을 나타내도록 하였다. 퍼지 출력 변수는 0초에서 약 0.5초의 클러치 작동 시간에 대하여 steering_left_large(SLL), steering_left_medium(SLM), steering_left_small(SLS), steering_zer(SZE), steering_right_small(SRS), steering_right_medium(SRM), steering_right_large (SRL) 등의 7개의 언어 변수로 정의 되었으며 각각은 싱글톤 멤버쉽 함수로 이루어져 있다. 차량 조향 경험에 근거하여 If - Then 형식의 퍼지제어 규칙을 개발 하였다. 표 3.6.1은 퍼지 제어 규칙을 나타낸 것이다. Sugeno의 퍼지 추론방식

으로부터 얻어졌고 식(3)과 같은 가중치 평균법으로 디퍼지화 되었다.

$$U_{Wa} = \frac{\sum_{i=1}^{n} u_i \mu(u_i)}{\sum_{i=1}^{n} \mu(u_i)} \tag{3.6.1}$$

여기서 $\mu(u_i)$ 와 u_i 는 각각 i 번째 출력 멤버쉽함수의 정도와 싱글톤 값이다.

Matlab을 이용해 위 퍼지 제어기를 시뮬레이션한 결과를 그림 3.6.5에 나타내었다. 비퍼지기화를 통해 나온 출력 값을 표 3.6.2에서와 같이 look-up table 형식으로 만들어 제어프로그램에서 배열변수에 입력하여 활용하였다.

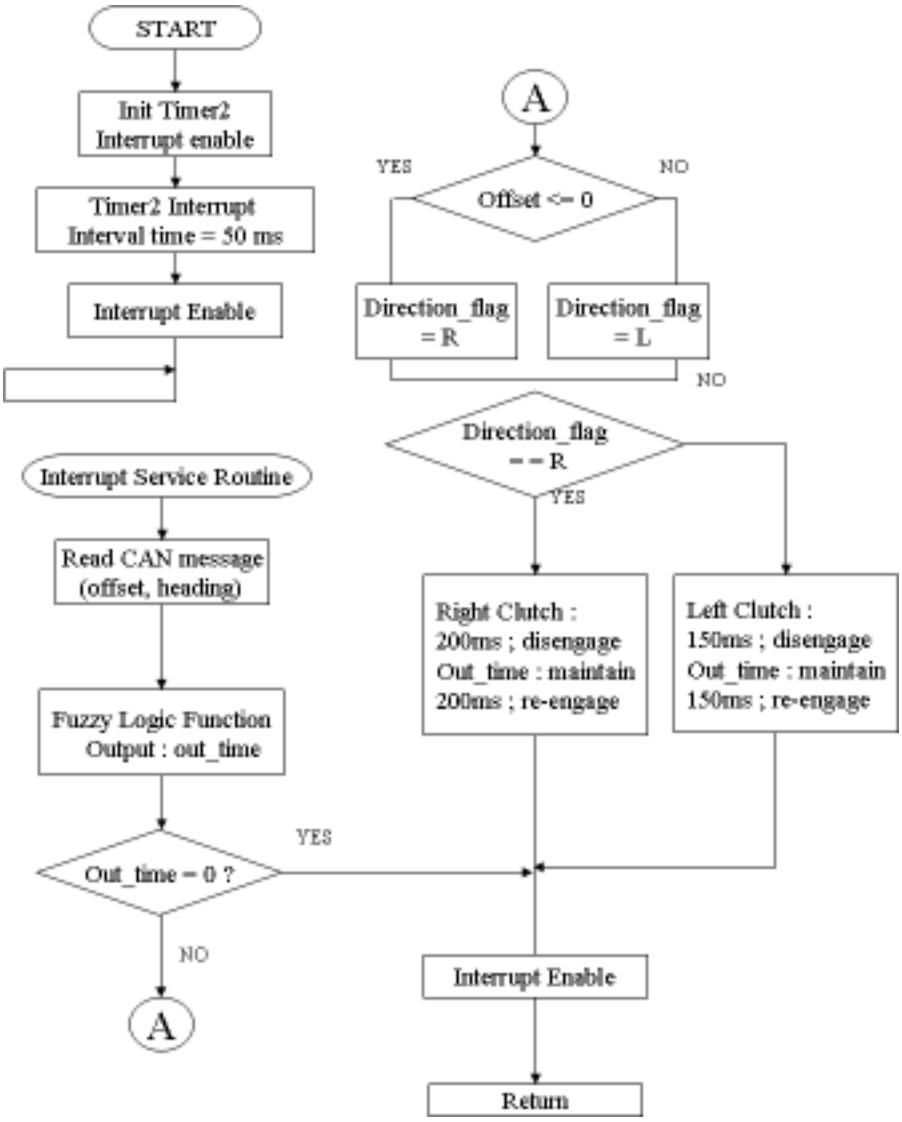


그림 3.6.1 조향제어 알고리즘의 순서도.

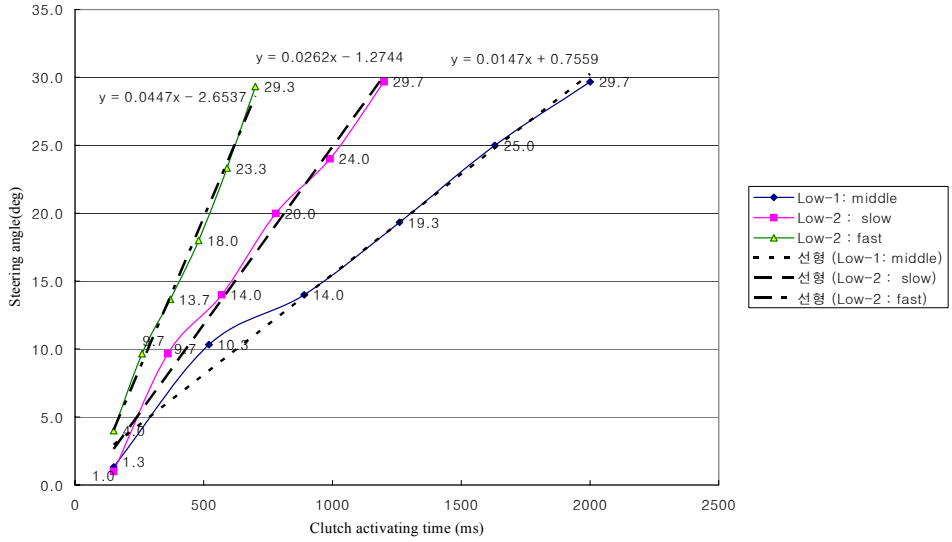


그림 3.6.2 좌측 조향 클러치 작동 특성.

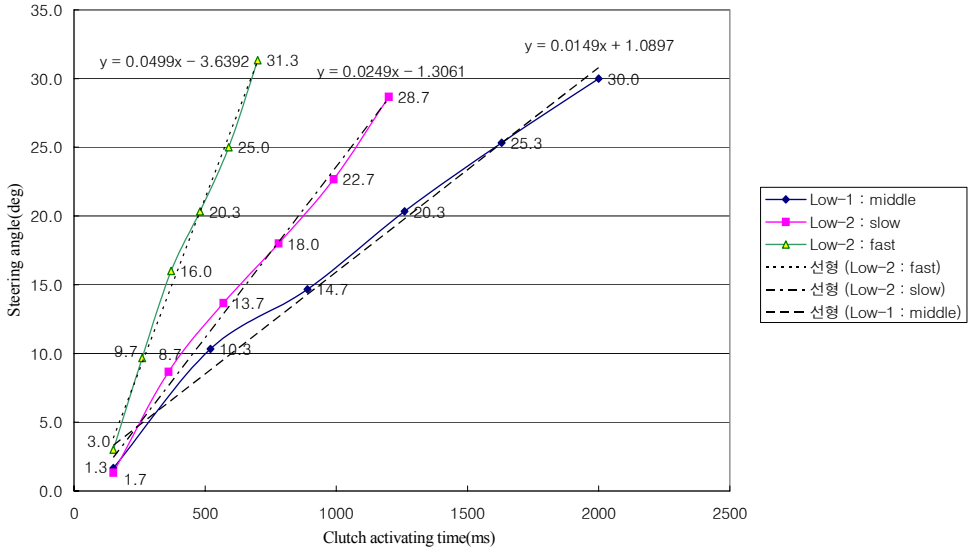


그림 3.6.3 우측 조향 클러치 작동 특성.

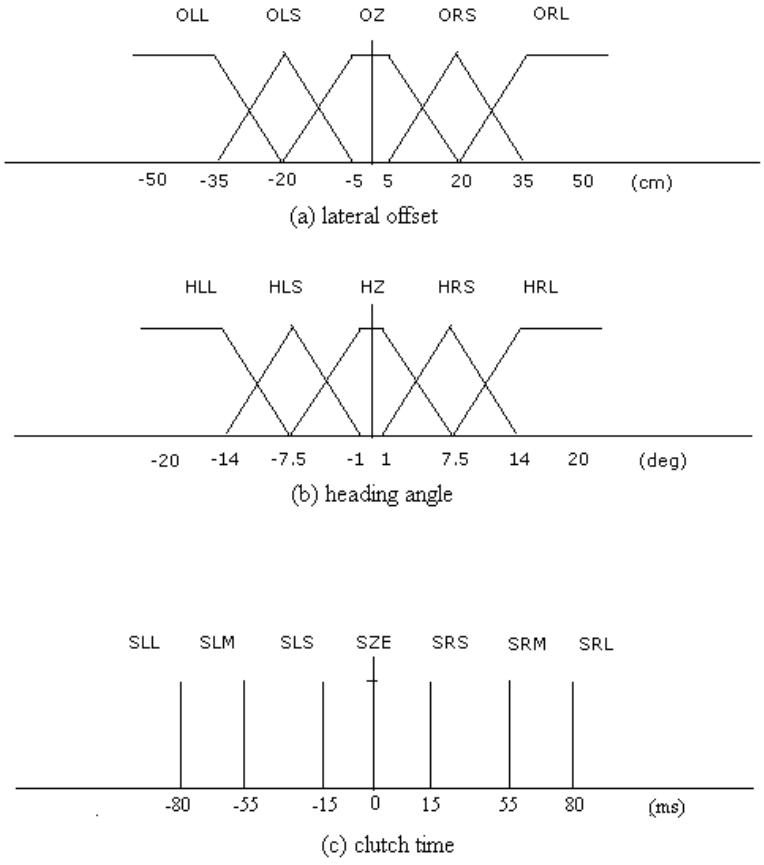


그림 3.6.4 퍼지 멤버쉽 함수 ; (a),(b) : 입력, (c) : 출력.

표 3.6.1 퍼지 제어 규칙

Rule #1	If[X_offset(Input1)=OLL] and [Degree(Input2)=HLL] Then [Output=SRL]
Rule #2	If[X_offset(Input1)=OLL] and [Degree(Input2)=HLS] Then [Output=SRM]
Rule #3	If[X_offset(Input1)=OLL] and [Degree(Input2)=HZ] Then [Output=SRM]
Rule #4	If[X_offset(Input1)=OLL] and [Degree(Input2)=HRS] Then [Output=SZE]
Rule #5	If[X_offset(Input1)=OLL] and [Degree(Input2)=HRL] Then [Output=SZE]
Rule #6	If[X_offset(Input1)=OLS] and [Degree(Input2)=HLL] Then [Output=SRS]
Rule #7	If[X_offset(Input1)=OLS] and [Degree(Input2)=HLS] Then [Output=SRS]
Rule #8	If[X_offset(Input1)=OLS] and [Degree(Input2)=HZ] Then [Output=SRS]
Rule #9	If[X_offset(Input1)=OLS] and [Degree(Input2)=HRS] Then [Output=SZE]
Rule #10	If[X_offset(Input1)=OLS] and [Degree(Input2)=HRL] Then [Output=SLS]
Rule #11	If[X_offset(Input1)=OZ] and [Degree(Input2)=HLL] Then [Output=SZE]
Rule #12	If[X_offset(Input1)=OZ] and [Degree(Input2)=HLS] Then [Output=SZE]
Rule #13	If[X_offset(Input1)=OZ] and [Degree(Input2)=HZ] Then [Output=SZE]
Rule #14	If[X_offset(Input1)=OZ] and [Degree(Input2)=HRS] Then [Output=SZE]
Rule #15	If[X_offset(Input1)=OZ] and [Degree(Input2)=HRL] Then [Output=SZE]
Rule #16	If[X_offset(Input1)=ORL] and [Degree(Input2)=HLL] Then [Output=SRS]
Rule #17	If[X_offset(Input1)=ORL] and [Degree(Input2)=HLS] Then [Output=SZE]
Rule #18	If[X_offset(Input1)=ORL] and [Degree(Input2)=HZ] Then [Output=SLS]
Rule #19	If[X_offset(Input1)=ORL] and [Degree(Input2)=HRS] Then [Output=SLS]
Rule #20	If[X_offset(Input1)=ORL] and [Degree(Input2)=HRL] Then [Output=SLS]
Rule #21	If[X_offset(Input1)=ORS] and [Degree(Input2)=HLL] Then [Output=SZE]
Rule #22	If[X_offset(Input1)=ORS] and [Degree(Input2)=HLS] Then [Output=SZE]
Rule #23	If[X_offset(Input1)=ORS] and [Degree(Input2)=HZ] Then [Output=SLM]
Rule #24	If[X_offset(Input1)=ORS] and [Degree(Input2)=HRS] Then [Output=SLM]
Rule #25	If[X_offset(Input1)=ORS] and [Degree(Input2)=HRL] Then [Output=SLL]

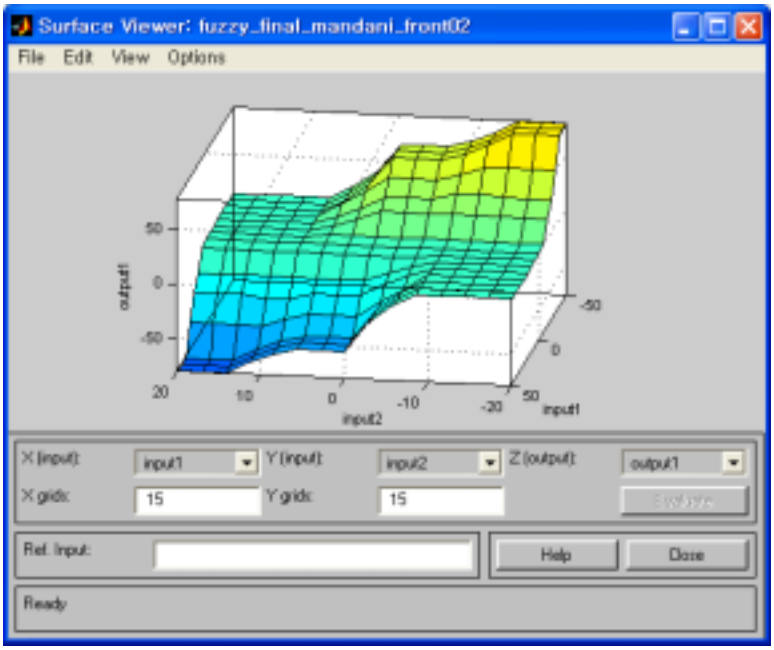


그림 3.6.5 퍼지 제어기 시뮬레이션 결과.

표 3.6.2 퍼지 제어기 look-up table

구분	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	20	20	20	20	17	16	14	13	13	13	13	8	5	2	0	0	0	0	0	0	0
1	20	20	20	20	17	16	14	13	13	13	13	8	5	2	0	0	0	0	0	0	0
2	20	20	20	20	17	16	14	13	13	13	13	8	5	2	0	0	0	0	0	0	0
3	20	20	20	20	17	16	14	13	13	13	13	8	5	2	0	0	0	0	0	0	0
4	14	14	14	14	14	13	11	10	10	10	10	8	5	3	0	0	0	0	0	0	0
5	11	11	11	11	11	11	9	8	9	8	8	6	4	3	0	0	0	0	0	0	0
6	6	6	6	6	6	6	6	6	6	6	6	4	2	1	0	0	0	0	0	0	0
7	6	6	6	6	6	6	6	6	6	6	6	4	2	1	0	0	0	0	0	0	0
8	2	2	2	2	2	2	2	2	2	2	2	2	2	1	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	-1	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
13	0	0	0	0	0	0	0	-1	-2	-4	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6
14	0	0	0	0	0	0	0	-1	-2	-4	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6
15	0	0	0	0	0	0	0	-3	-5	-7	-9	-9	-9	-9	-9	-12	-11	-11	-11	-11	-11
16	0	0	0	0	0	0	0	-3	-5	-8	-11	-11	-10	-11	-11	-13	-14	-14	-14	-14	-14
17	0	0	0	0	0	0	0	-2	-5	-9	-14	-14	-14	-14	-14	-16	-18	-20	-20	-20	-20
18	0	0	0	0	0	0	0	-2	-5	-9	-14	-14	-14	-14	-14	-16	-18	-20	-20	-20	-20
19	0	0	0	0	0	0	0	-2	-5	-9	-14	-14	-14	-14	-14	-16	-18	-20	-20	-20	-20
20	0	0	0	0	0	0	0	-2	-5	-9	-14	-14	-14	-14	-14	-16	-18	-20	-20	-20	-20

* all data were evenly divided by 4 to be stored in a microprocessor

나. 컴퓨터 시뮬레이션

퍼지 제어기의 튜닝을 수행하고 또한 제안된 제어시스템의 안정성 등을 검토하기 위하여 컴퓨터 시뮬레이션을 위한 수학적 모델과 프로그램을 개발하였다. 시험차량의 이동 변위에 대한 수학적 모델은 지표좌표를 사용하여 규정하였다. 그림 3.2.1(b)에서와 같이 차량의 초기 위치는 주행경로 결정센서의 측면옵셋(offset)과 방향각(θ)으로 나타낼 수 있다. 한편, 지표 좌표의 y축이 차량의 중심점을 지나가는 것으로 가정하면, 초기 위치에서의 무게중심점의 좌표(CG)는 다음과 같이 계산된다.

$$x(t) = -sin(\theta(t)) \cdot CG2Inc + offset$$
$$y(t) = 0 \qquad (3.6.2)$$

여기서 offset은 차량이 주행기준선 기준으로 좌측에 있을 때 (-) 부호로 표시된다. CG2Inc 는 CG에서 Inclinometer까지의 거리로 161 cm이다. $\theta(t)$ 는 가상의 수직선으로부터 측정된 차량의 현재 방향각이고 차량이 수직선으로부터 왼쪽으로 놓이게 될 때 (-)부호를 가지게 된다.

실제 무인주행 제어 시스템은 Inclinometer로부터 측면옵셋값을 측정하는 반면에, 컴퓨터 시뮬레이션에서의 측면옵셀은 식 (3.6.2)를 역으로 하여 다음과 같이 계산되어진다.

$$offset = x(t) + sin(\theta(t)) \cdot CG2Inc \qquad (3.6.3)$$

이 값은 조향 클러치가 끊어짐을 유지하는 시간(혹은 방향전환을 하는 시간)을 결정하기 위하여 퍼지제어기의 입력으로 사용된다.

퍼지제어기의 출력이 0, 즉 조향할 필요가 없다고 하면 차량은 현재의 방향각을 유지한 채 제어 주기 동안 직진을 하게 된다. 따라서, 직진이 완료된 시점에서 무게 중심점의 좌표는 식 (3.6.4)와 같다.

한편, 조향이 요구되면 조향클러치의 조작은 3가지 단계로 나타낼 수 있다. 본 연구에서 사용된 조향클러치가 jaw 타입이기 때문에 구동 동력은 클러치의 이가 완전히 떨어지기 전까지는 구동 스프로켓에 전달이 된다. 이런 이유로 유압실린더가 조향클러치가 끊어지는 위치로 이동하고 있는 중에도 차량은 동일한

방향각을 유지한 채 전진하게 되므로 차량의 무게중심은 식(3.6.4)에서와 같이 구할 수 있다. 단 여기서 클러치 작동시간은 클러치를 disengage 시키는 데 필요한 시간이 된다.

$$\begin{pmatrix} x(t+1) \\ y(t+1) \end{pmatrix} = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} + \begin{bmatrix} \sin(\theta(t)) \\ \cos(\theta(t)) \end{bmatrix} * Vel * IntervalTime \qquad (3.6.4)$$

$$\theta(t+1) = \theta(t)$$

여기서 Vel 은 지면에서의 차량속도이고 $IntervalTime$ 은 제어주기 혹은 인터럽트 주기(=50ms)이다.

클러치가 동작하는 두 번째 단계에서는 차량이 조향을 시작한다. 구동동력이 단지 구동 트랙에만 전달이 되기 때문에 트랙의 바깥 중심선과 CG로부터의 연장된 교차점인 정지 트랙의 한점에서 회전이 일어난다. 이는 브레이크가 걸려있는 트랙의 회전중심에서 회전이 일어나게 되고 CG는 또한 차량의 회전각에 비례하여 회전을 하게 된다. 회전중심에 놓인 트랙은 차량의 현재위치로부터 결정이 된다. 만약에 차량이 천장유도레일의 왼쪽 편에 위치했다면 오른쪽으로 조향을 해야 하는데 이는 오른쪽 트랙이 회전중심이 됨을 의미한다. 오른쪽으로 회전을 하는 경우 오른쪽 트랙의 순간 회전중심의 좌표는 다음과 같이 구할 수 있다.

$$\begin{pmatrix} Cx(t) \\ Cy(t) \end{pmatrix} = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} + \begin{bmatrix} \cos(\theta(t)) \\ -\sin(\theta(t)) \end{bmatrix} * \frac{SSWidth}{2} \qquad (3.6.5)$$

여기서 $SSWidth$ 는 양쪽 트랙의 중심간의 거리로 124cm이다.

왼쪽으로 조향할 경우엔 다음과 같이 된다.

$$\begin{pmatrix} Cx(t) \\ Cy(t) \end{pmatrix} = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} + \begin{bmatrix} -\cos(\theta(t)) \\ \sin(\theta(t)) \end{bmatrix} * \frac{SSWidth}{2} \qquad (3.6.6)$$

차량이 $(Cx(t), Cy(t))$ 를 중심으로 조향을 할 경우 한번 제어되는 동안의 회전각 ϕ는 다음과 같이 계산되어진다.

$$\phi = \frac{Vel * IntervalTime}{SSWidth} \tag{3.6.7}$$

차량이 오른쪽으로 회전할 때 CG의 다음위치와 방향각은 식(3.6.6)과 같이 회전중심의 좌표인 $(Cx(t), Cy(t))$ 에 기초하여 계산이 되어진다.

$$\begin{pmatrix} x(t+1) \\ y(t+1) \end{pmatrix} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} - \begin{bmatrix} \cos(\phi)-1 & -\sin(\phi) \\ \sin(\phi) & \cos(\phi)-1 \end{bmatrix} \begin{pmatrix} Cx(t) \\ Cy(t) \end{pmatrix}$$
$$\theta(t+1) = \theta(t) + \phi \tag{3.6.8}$$

같은 방법으로 식(3.6.6)을 참고하여 왼쪽으로 조향 할때의 CG의 다음 위치는 다음과 같이 구할 수 있다.

$$\begin{pmatrix} x(t+1) \\ y(t+1) \end{pmatrix} = \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix} \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} - \begin{bmatrix} \cos(\phi)-1 & \sin(\phi) \\ -\sin(\phi) & \cos(\phi)-1 \end{bmatrix} \begin{pmatrix} Cx(t) \\ Cy(t) \end{pmatrix}$$
$$\theta(t+1) = \theta(t) - \phi \tag{3.6.9}$$

조향시간은 FLC의 출력값 동안 지속된다.

세번째 단계로, 조향이 완료된 후 클러치는 재결합을 시작한다. 처음단계와 비슷하게 CG의 다음 위치는 식(3.6.4)와 같고 차량은 클러치 re-engage 시간 동안 식(3.6.8)의 방향각으로 직진하게 된다.

이와 같은 알고리즘에 근거하여 MFC로 컴퓨터 시뮬레이션 프로그램을 개발하였다. 그림 3.6.6은 시뮬레이션 프로그램을 보여 준다.

컴퓨터 시뮬레이션은 지면이 완전히 평평하고, 트랙과 지면과는 슬립이 없다는 가정하에 수행되었으며, 앞 서 과수방제기 시운전 경험에 맞추어 작성한 퍼지 제어 변수와 제어 규칙을 사용하였다. 여기서 지면에서의 차량 속도는 20cm/s로 하였다.

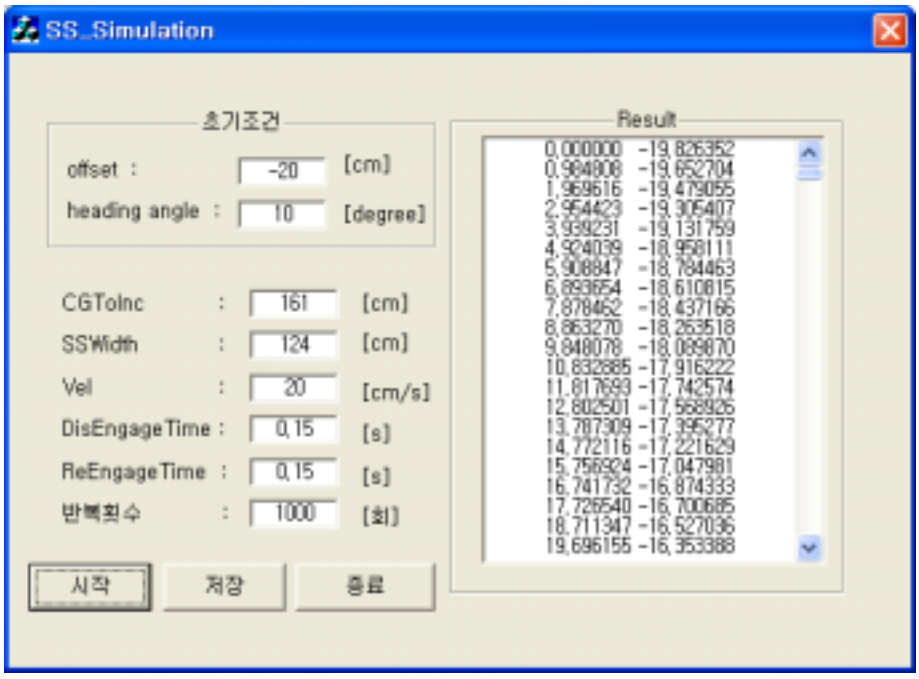


그림 3.6.6 컴퓨터 시뮬레이션 프로그램.

다. 시뮬레이션 결과

여러 가지 초기 차량의 자세 조건에 대한 시뮬레이션 결과를 그림 3.6.7 - 그림 3.6.9에 나타내었다. 모든 경우 초기 조건에서 차량의 궤적은 목표 주행선으로 수렴하는 것으로 나타났다. 그러나, 초기 측면옵셋이 존재하는 상태에서 차량이 주행기준선 쪽을 향하고 있는 경우, 즉 조향이 필요치 않을 것으로 예상되는 조건에서 차량의 주행 궤적은 목표 주행선을 넘어 오버슈트 현상이 나타났다. 측면옵셋이 -20 cm인 경우나 -40 cm인 경우에 오히려 차량이 방향각이 주행기준선과 반대 방향으로 향하고 있을 때 주행 제어 성능이 우수한 것으로 보여진다. 같은 조건에서 10 cm/s부터 100 cm/s까지 주행속도를 달리하며 시뮬레이션을 실시해 본 결과 궤적에는 큰 차이가 없는 것으로 나타났다.

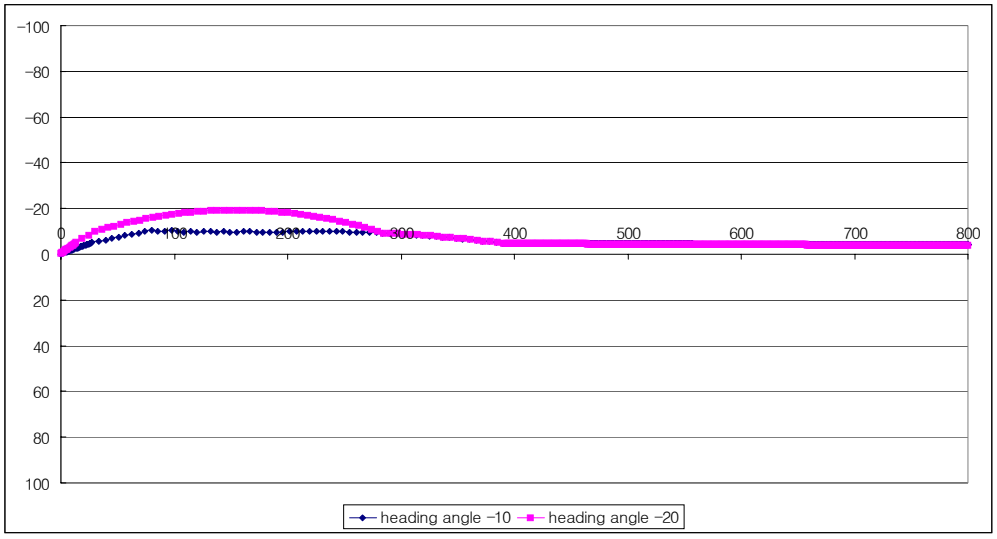


그림 3.6.7 측면옵셀 0 cm에서 방향각 -10°, -20°인 경우의 주행 궤적.

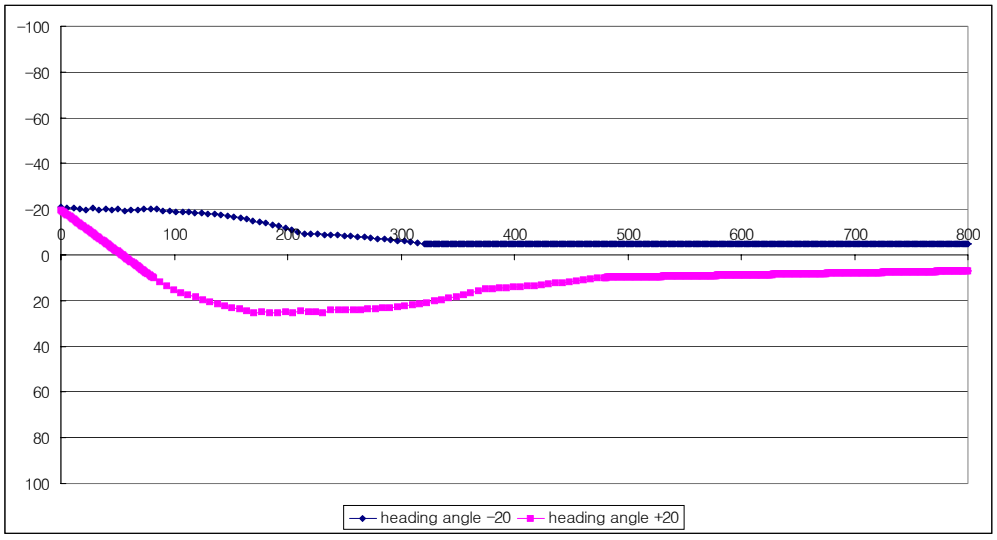


그림 3.6.8 측면옵셀 -20 cm에서 방향각 ±20°인 경우의 주행 궤적.

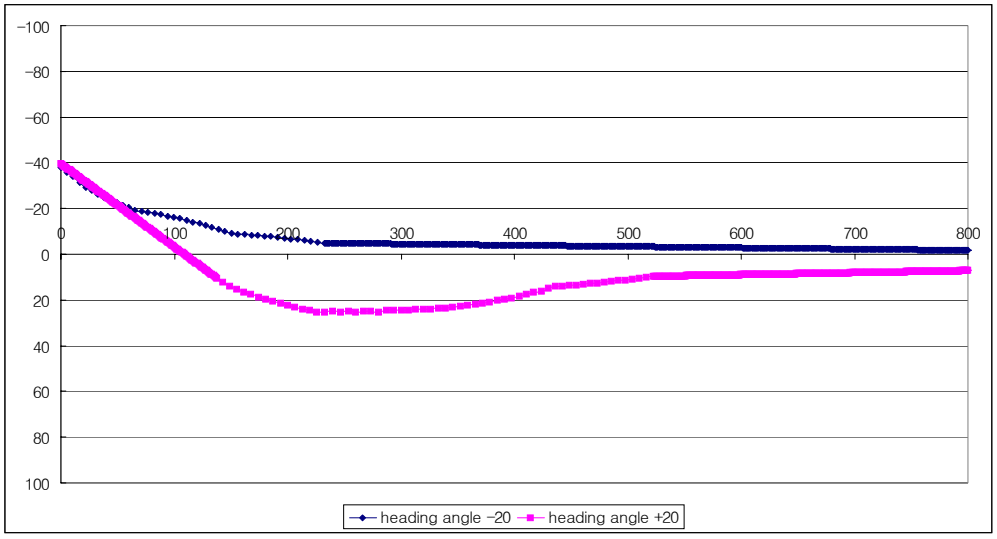


그림 3.6.9 측면옵셀 -40 cm에서 방향각 ±20°인 경우의 주행 궤적.

2. 방제작업 제어 알고리즘

약액살포 노즐은 과수방제기의 후미에 위치해 있으나 과수열의 시작과 끝을 감지하는 센서는 과수방제기의 앞 부분인 트롤리에 설치되어 있다. 즉, 과수열 시작/끝 감지센서가 과수열의 시작점을 감지하고 나서 과수방제기의 후미가 과수열의 시작부에 도달했을 때 약액살포가 시작되어야 하며 또한 과수열 시작/끝 감지 센서가 과수열의 끝을 감지했다하더라도 노즐부위가 과수열의 끝단에 도달했을 때 약액살포를 멈추어야 한다.

본 연구의 과수방제기에서는 노즐 위치와 트롤리가 위치할 차량의 선단부까지는 약 280 cm의 차이가 있다. 한편, 구동 스프로켓에 설치된 이동거리 측정센서는 구동 스프로켓의 이빨 하나 사이의 간격은 지면 상의 거리로 약 6.7 cm 정도이다. 따라서, 과수열의 시작과 끝을 감지하고 나서 구동 스프로켓의 이빨이 약 42개를 넘어가고 난 이후에 노즐의 on/off가 작동되어야 한다.

적정한 거리만큼의 지연후에 노즐이 작동하게 하기 위하여 제어 프로그램에 크기가 42개인 배열 변수를 선언하고 과수열 시작/끝 감지 센서가 감지를 시작하는 순간의 노즐 on/off 데이터 정보를 첫 번째 스택에 집어 넣고 스프로켓의 이

빨이 하나 넘어가면 두 번째의 on/off 정보를 두 번 째 스택에 집어 넣고를 반복하면서 첫 번째 스택변수가 42번째로 이빨이 넘어가게 되면 출력되는 방식으로 약액 살포 노즐을 제어 하였다. 그림 3.6.10은 위 제어 알고리즘의 순서도를 나타낸 것이다.

제어 프로그램을 부록에 수록하였다.

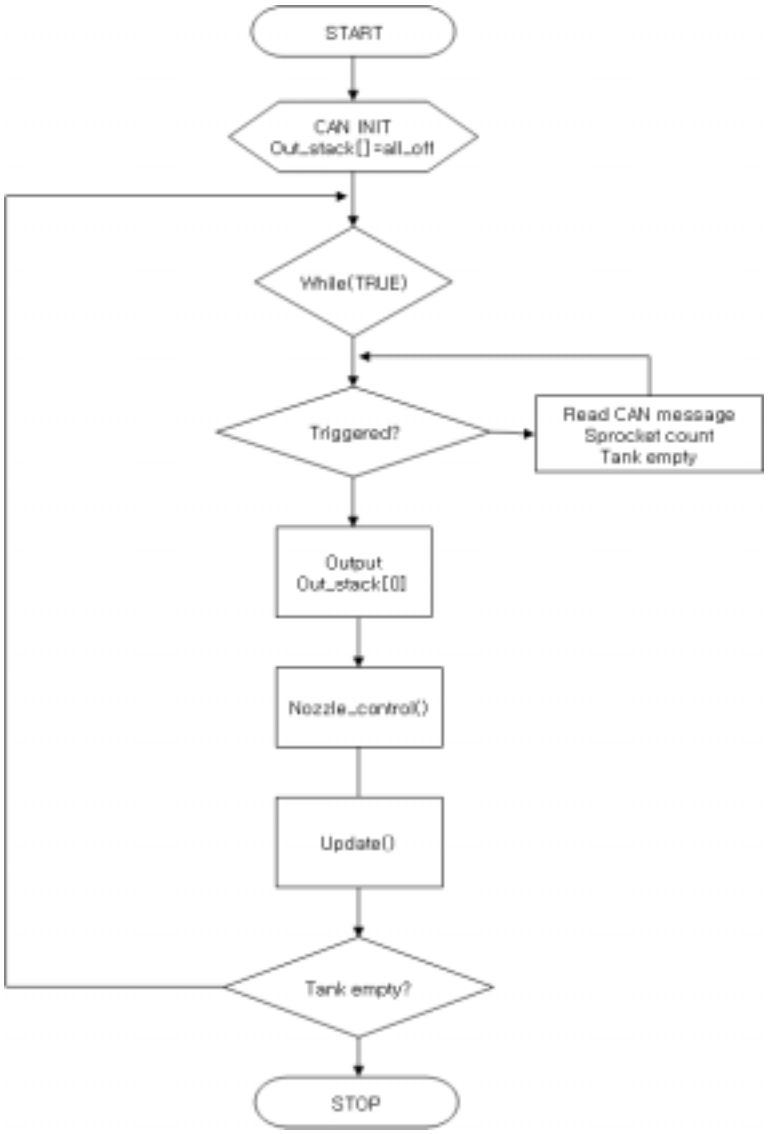


그림 3.6.10 약액살포 제어 알고리즘의 순서도.

제 7 절 성능 평가

1. 무인주행 제어성능

가. 성능 평가 방법

과수방제기의 무인주행 제어 성능은 차량이 주행 기준선인 오버헤드 가이던스 레일의 지면 투사선으로부터 얼마나 벗어나 있는 지 차량의 주행 궤적을 측정하여 나타낼 수 있다. 조향제어 성능을 평가하기 위하여 실험실에 오버헤드 가이던스 레일을 설치하였다(그림 3.2.14). 차량의 실 주행경로는 차량의 선단 중앙에 일정한 시간 간격으로 물감을 떨어뜨리는 장치를 고안하여 설치(그림 3.7.1)한 후 궤적으로부터 매 20cm 마다 주행기준선과의 편차를 자로 측정하여 데이터화 하였다. 평가할 항목은 i) 직선 구간과 곡선 구간에 대하여 차량이 얼마나 주행기준선을 잘 추종 하느냐와 ii) 출발시 다양한 차량의 자세로부터 차량이 얼마나 빨리 목표 경로를 찾는 가 이다.

본 연구의 예비 실험 결과 실험실처럼 노면이 일정한 경우 직선 구간에서 초기에 0 cm의 측면옵셋과 0°의 방향각에서는 트롤리가 부드럽게 작동하며 차량을 유도하여 전혀 조향 작용이 일어나지 않았다. 따라서, 성능 실험은 표 3.7.1에 나와 있는 바와 같은 초기 조건으로 측면옵셋과 방향각을 달리하며 직선 구간에서 차량이 어떻게 주행하는 지 검토하였다. 주행속도는 고속의 경우 실험실내 공간이 협소하기 때문에 안전상의 이유로 수행되지 못하고 중속(약 26 cm/s)의 경우에 한하여 성능 실험을 수행하였다. 예비 실험을 통해 주행속도가 작을 경우 직선 구간이나 곡선 구간에서 모든 경우 원활하게 제어되는 것을 확인하였다.

표 3.7.1 성능 평가를 위한 차량 자세의 초기 조건

	lateral offset		
	0 cm	-20 cm	-40 cm
heading angle	10°	-20°	-20°
	20°	20°	20°

그림 3.7.2 궤적측정장치와 지표면의 궤적 표시.

나. 결과 및 고찰

그림 3.7.3은 초기 측면옵셑이 0인 상태에서 방향각만 각각 -10°, -20°를 준 경우이다. 일단 현재의 방향각의 방향으로 차량이 진행하다가 측면 옵셑이 커지면서 조향 작용을 시작하게 된다. 방향각이 큰 경우 조향 작용이 빨리 시작되었으며 180 cm 지점에 도달했을 때 급격한 조향을 하다보니 차량이 오히려 주행기준선을 넘는 경우가 발생한 것으로 보인다. 초기 방향각이 적은 경우가 보다 안정적으로 주행을 하는 것을 확인할 수 있다. 궤적이 차량의 앞 부분에서 측정되므로 차량이 약 200 cm 진행 후 직선 구간을 따라 안정적으로 무인주행을 한다고 판단할 수 있다. 그림 3.7.4는 초기 측면옵셑이 -20 cm인 상태에서 차량이 주행기준선 쪽을 바라보는 경우(+20°)와 오히려 반대쪽으로 향해 있는 경우(-20°)의 실 주행 경로를 나타낸 것이다. 방향각이 +20°인 경우 측면옵셑에 의한 약간의 클러치 조작에 의해 주행기준선을 넘어버리는 현상이 나타났다. 200 cm 지점에서 다시 반대 방향으로 많은 각도를 선회하고 다시 주행 기준선을 넘어가는 지그재그 형태로 주행된 경우이다. 방향각이 -20°인 경우 오히려 차량은 초기 측

면 옵셑에 의한 클러치 조작에 의해 어느 정도 안정적인 방향각을 띌 수 있어 그 이후는 비교적 부드럽게 직선 구간을 주행한 것으로 나타났다. 현재의 센서로는 초기의 방향각, 즉 출발하기 직전의 방향각을 측정할 방법이 없기 때문에 나타날 수 있는 현상으로 보여진다. 그림 3.7.5에서와 보는 바와 같이 초기 측면 옵셑이 크게 존재하는 경우 처음에 조작해야할 클러치의 조향각이 매우 크게된다. 따라서, 그림 3.7.4의 경우보다 더 심하게 주행기준선을 넘는 오버슈트 현상이 일어나고 다시 반대 방향으로 조향하는 운전을 통해 목표 주행선을 찾아갈 수밖에 없는 현상이 나타난다고 판단된다. 수 차례의 반복 실험을 통해 차량의 초기 위치가 양호하지 못한 경우에도 1-2회 정도의 지그재그 모션으로 2-3m 진행 후 목표 주행선으로 진입하는 것을 확인할 수 있었다. 과수방제기의 실주행 경로를 앞 절의 컴퓨터 시뮬레이션과 비교해 보면 주행 패턴이 유사하나 더 과다하게 조향이 이루어지는 현상은 실제의 경우 차량의 조향 제어 동작이 일정치 못하거나 늦기 때문으로 판단된다. 또한 주행경로 결정 센서의 경사각도측정 레버가 차량이 움직이면서 발생하는 차체 진동, 트롤리가 이동하면서 발생하는 진동 등으로 끊임없이 진동하면서 측면옵셑 또는 방향각을 결정하는데 있어서 오차를 야기할 소지를 갖고 있기 때문으로 판단된다. 따라서, 경사각도 측정 장치에 적정한 댐핑 기능을 추가하거나 소프트웨어적으로 데이터를 가공할 필요성이 있는 것으로 판단된다.

그림 3.7.6은 곡선 구간에 대한 실 주행 경로의 궤적을 보여 준다. 이것은 좌측 과수열에서 진입하여 오른쪽으로 돌아 우측 과수열로 진입하는 상황으로 곡선 구간 진입 초기 차량은 직진하다가 측면옵셑이 증가하면서 왼쪽으로 조향을 했다가 다시 트롤리를 추종하는 양상을 보였다. 곡선 구간을 돌아 우측 과수열로 진입하는 순간은 바깥 쪽으로 치우쳐 도는 현상이 나타났다. 이것은 트롤리가 위치상 선회하는 구간에서 경사각도계 레버의 y_angle이 트롤리 모터의 속도를 제어하는 데드밴드 영역내에 있는 것처럼 측정되어 트롤리가 너무 앞 서 나가 있는 상태가 되어 현재의 측면옵셑을 정확히 감지하지 못하는 상황이 나타나기 때문으로 판단된다. 또한 곡선구간내에서 직선구간이 짧은 경우 차량이 충분히 목표주행선에 걸쳐 있지 못하게 되며 이것은 초기 측면옵셑이 있는 상황에서의 직선 구간 주행에서와 같이 과도한 측면옵셑을 줄이기 위해 과도한 조향이 일어나는 상황이 발생한 것으로 판단된다. 이러한 현상은 과수열간의 간격이 넓은 실제 포장에서는 곡선 구간내 직선 부분의 길이가 충분히 길어져서 극복할 수 있는

문제라고 판단되며 또는 곡선구간으로 진입시 주행속도를 저속으로 설정하여 진행한다면 바깥 쪽으로 과다하게 치추쳐 도는 것을 방지할 수 있을 것으로 판단된다.

다. 실제 과수원에서의 성능 평가

강원대학교 부속농장 배 과수원(그림 3.2.15)에서 무인 주행 성능 실험을 수행하였다(그림 3.7.7). 궤적을 측정할 수 없어서 저속, 중속에 대한 주행 상황만을 검토하고 비디오로 촬영하였다. 그림 3.2.15에서 보는 바와 같이 과수원의 주행로는 한 두 군데를 제외하고 비교적 평탄했으며 부드러운 토양에 풀이 자라고 있어서 과수방제기의 차체 진동을 줄여주는 효과가 있어서 무인 주행 성능은 실험실의 깨끗한 콘크리트 바닥에서 보다 오히려 부드럽게 주행함을 알 수 있었다. 특히 곡선 구간의 경우 과수열의 폭이 6 m 이므로 곡선구간을 진입하여 우회전한 후 목표 주행선을 추종할 수 있을 만큼 충분한 직선 구간이 있어서 다음 과수열로의 진입시 실험실에서와 달리 바깥 쪽으로 많이 치우치지 않고 완만한 곡선으로 선회를 완료할 수 있었다. 그림 3.7.8은 곡선 선회 구간에서 과수방제기의 궤적을 보여 준다.

2. 약액살포 제어 성능

강원대학교 부속농장에서 과수방제기의 무인주행 성능 실험을 수행할 시 약액살포부를 작동시킬 수 없었다. 현재 개발이 완료된 각종 ECU 들이 과수방제기에 개방된 상태로 설치되어 있기 때문이었다. 약액살포부 제어 성능 실험은 실험실에서 오버헤드 가이던스레일에 설치한 과수열 끝/시작점 센서로부터 데이터를 받아 약액살포제어기 ECU에 의해 노즐 on/off 솔레노이드 밸브가 일정한 지연시간을 갖고 on 또는 off 되는 지의 여부를 판단하였다. 솔레노이드 밸브 작동시 ECU의 출력을 표시하는 LED를 통해 밸브의 작동 상태를 확인할 수 있었다. 본 연구가 적용되는 과수원의 경우는 과수들 사이에 빈 공간이 아예 없기에 과수의 크기 또는 형상 등에 따라 노즐들을 개별 조작해야할 이유가 없었다. 다만, 노즐을 좌우측으로 구분하여 과수열이 시작될 때 약액살포를 시작하고 과수열이 끝나면 약액살포를 중지하되 좌회전하는 경우는 선회하면서도 좌측 노즐만 켜서 방제를 하고 우측열은 노즐을 잠그는 방식의 제어로 충분하다고 판단된다.

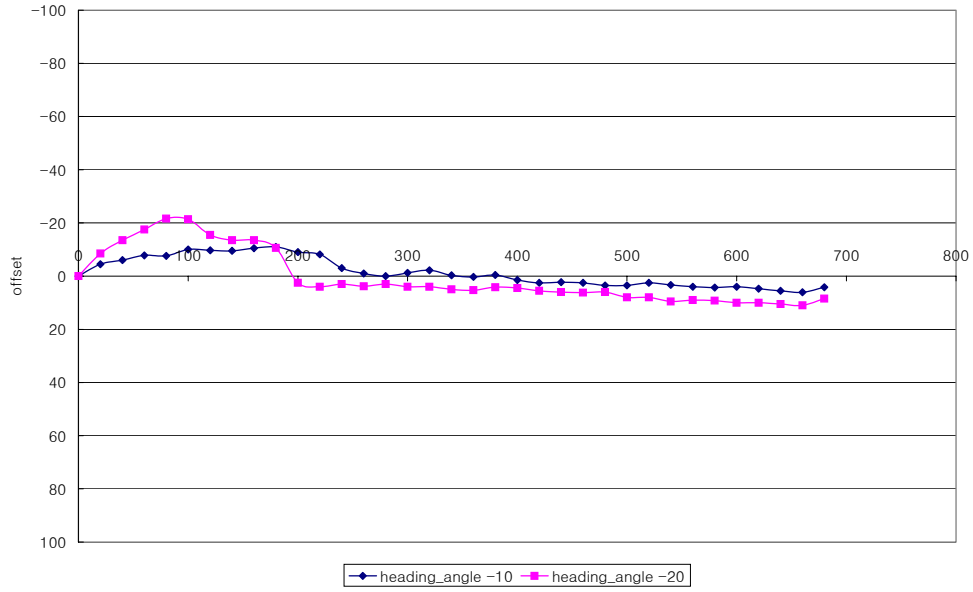


그림 3.7.3 측면옵셀 0 cm에서 방향각 -10°, -20°인 경우의 실제 주행 궤적.

그림 3.7.4 측면옵셀 -20 cm에서 방향각 ±20°인 경우의 실제 주행 궤적.

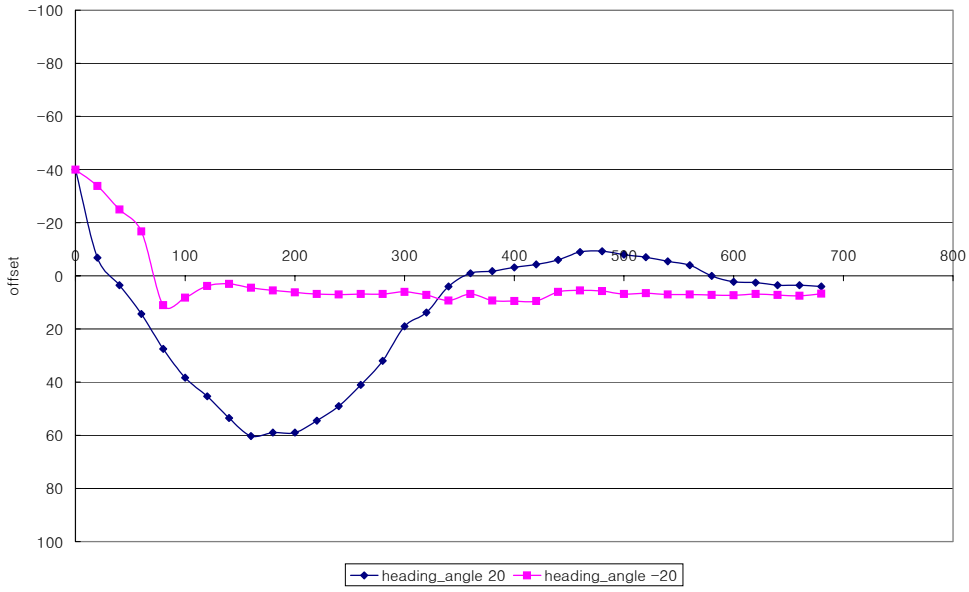


그림 3.7.5 측면옵셋 -40 cm에서 방향각 ±20°인 경우의 실제 주행 궤적.

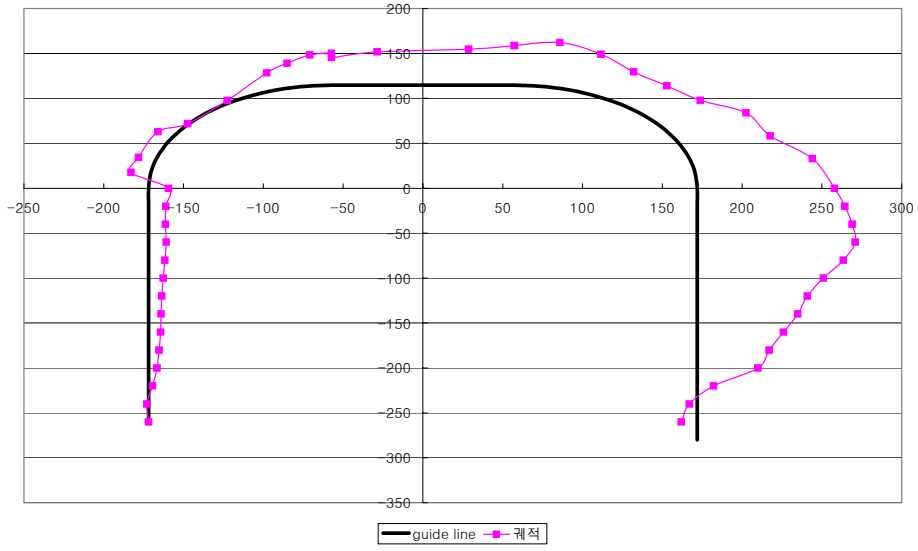


그림 3.7.6 곡선구간에서 실제 주행 궤적.

그림 3.7.7 과수원에서의 성능 실험 모습.

그림 3.7.8 과수원에서의 선회구간 궤적.

제 4 장 목표달성도 및 관련분야에의 기여도

제 1 절 목표 달성도

본 연구는 농업기계에의 적용이 확대될 것으로 예상되는 CAN 버스 시스템의 작동 원리를 규명하여 CAN 버스 시스템의 기술을 구현하여 과수방제기의 무인화에 적용하는 것을 목적으로 수행되었다. 구체적으로 CAN 버스를 구성할 CAN 통신 기능이 있는 마이크로프로세서의 ECU를 설계, 제작하고 구동 프로그램을 개발하며, 과수방제기의 무인화에 필요한 주행경로 결정 센서의 개발 및 여타 필요한 다양한 센서와 무인 조작기를 마이크로프로세서와 인터페이스, 시스템 제어 알고리즘 개발 등 궁극적으로 무인 과수 방제기를 개발하는 연구이다.

모든 연구 내용은 계획에 의거하여 목표치를 달성하였으며 상용화 시스템의 개발을 목적으로 관심있는 기업체와 공동 연구를 수행할 예정이며 현장의 농민들과도 접촉하여 사용자 편의 사항, 안전장치들에 대한 요구 사항 등을 청취하여 지속적인 연구 개발을 통해 과수 농가의 생산성 향상에 기여하도록 할 것이다.

1차년도에 주요 연구 내용이었던 기존의 과수방제기의 무인 운전 조작기의 개발 및 적용, 주행경로 결정 센서와 트롤리가 개발되었고, CAN 버스 시스템에 대한 기초 연구 및 간단한 형태의 CAN 버스 구성 등을 통해 CAN 기능의 ECU와 CAN 구동 프로그램 등을 개발하였다. 2차년도에는 무인 조작기로 사용되는 유압시스템에 대한 분석 및 설계, 오버헤드 가이던스 레일의 개발, 다양한 센서 및 조작기의 ECU에의 인터페이스 회로 개발, 무인 주행 및 무인 작업 알고리즘 개발을 통해 CAN 기반의 무인 과수방제기가 구현되었다. 실험실 및 과수원 현장에서의 성능 평가를 통해 실용화 가능성이 매우 높은 것으로 판단되었다.

제 2 절 관련분야에의 기여도

본 연구의 결과는 농업기계의 전장제어 분야나 무인 자동화 분야에 CAN을

적용할 수 있는 기반을 제공할 수 있을 것으로 판단된다. 또한 무인 과수방제기는 농민 및 온실 시공 업체 등으로부터 그 활용 가능성이 매우 높다는 긍정적인 평가를 받았다. 따라서, 예상되는 관련 분야에 기여할 부분은 다음과 같다.

- 농업기계 분야에의 CAN 버스의 적용 가능
- 마이크로프로세서로 구동되는 센서 및 조작기 등의 개발로 농기계 산업체의 고부가가치 시스템 개발 가능
- 온실 등의 밀폐 공간에서 고효율 및 저가의 약액 살포 장치 개발 가능
- 전작 갠트리 시스템의 개발 가능

제 5 장 연구개발결과의 활용계획

본 연구에서 개발한 CAN 버스 시스템은 센서와 조작기 등을 필요한 만큼 농업기계 제어 시스템, 농용 로봇 등에 쉽게 적용할 수 있다. 특히 제어 및 배선이 복잡한 경우에 그 효용성이 증대되는 바 상용화 단계로의 기술 개발 연구가 추가적으로 필요하다, CAN 버스 기술이 적용된 과수방제기는 무인주행/무인작업에 대한 기술력이 확보되었으므로 상용화 단계로의 추가적인 개발 연구 후 현장에 적용하기 위한 시범 사업 등을 추진하고자 하며, 구체적인 활용 계획은 다음과 같다.

- 현재 콤바인의 문제점 중 하나가 농한기 중에 쥐 등이 배선을 갉아먹어 고장이 났을 때 배선의 복잡함으로 인해 고장 수리에 어려움이 많다고 알려져 있다. 따라서, 관심있는 기업과 공동 연구를 통해 부분적으로라도 CAN 버스로 대체를 추진하여 콤바인 성능 개선에 기여한다.

- 센서 또는 조작기의 마이크로프로세서와의 인터페이스 기술, CAN 버스 시스템 구성 및 구현 기술 등을 활용해 기존의 시스템에도 센서들과 조작기 등을 쉽고 간편하게 추가할 수 표준 시스템을 개발하여 농업기계의 부분 자동화 분야 및 일반 기계의 자동화 분야에 활용한다.

- 과수방제기의 무인주행/무인작업을 위해 개발한 주행경로 결정 센서 및 제어 알고리즘에 대한 특허를 출원한다.

- 관심있는 기업체에 무인 과수방제기의 기술을 제공하고 공동 연구하여 지자체의 협조를 받아 과수원 무인 방제 시스템의 시범 사업을 추진한다.

- 확보된 CAN 관련 기술을 필요로 하는 농업기계 관련 산업체에 적극 홍보·보급함으로써 세계 수준의 기술력을 확보할 수 있도록 기여한다.

제 6 장 참고문헌

1. Cho, S.I. and N.H. Ki. 1999. Autonomous speed sprayer guidance using machine vision and fuzzy logic. Transactions of the ASAE. 42(4):1137-1143.

2. Farsi, M., K. Ratcliff and M. Barbosa. 1999. An overview of Controller Area Network. Computers & Control Engineering Journal June 1999.:113-120.

3. Hague, T. and N.D. Tillet. 1996. Navigation and control of an autonomous horticultural robot. Mechatronics. 6(2):165-180.

4. Heo, W.S., J.H. Shin, S.G. Lee, K.W. Kim, M.W. Cho and H.T. Kim. 2002. Development of Web-Based Control System for Greenhouse Teleoperation. Journal of the Korean Society of Agricultural Machinery. 27(4):349-354.

5. Jang, I.J., T.H. Kim and M.D. Cho. 1995. Development of unmanned speed sprayer- (Part I): Remote control and induction cable system. Journal of the Korean Society of Agricultural Machinery. 20(3):209-216.

6. Jang, I.J., T.H. Kim and S.H. Eum. 1998. Development of unmanned speed sprayer- (Part II): Guidance control using image processing. Journal of the Korean Society of Agricultural Machinery. 23(3):291-304.

7. Ki, N.H., S.I. Cho and C.H. Choi. 1996. Autonomous speed sprayer using machine vision and fuzzy logic (Part II): Real operation. Journal of the Korean Society of Agricultural Machinery. 21(2):175-184.

8. Kim, S.H., B.S. Shin, K.I. Kim and Y.M. Koo. 1999. Development of speed sprayer with control of chemical application. Final Report. ARPC.

9. Lee, J.H., S.I. Cho and J.Y. Lee. 1998. Autonomous speed sprayer using DGPS and fuzzy control - (Part II): Real operation. Journal of the Korean Society of Agricultural Machinery. 23(4):487-496.

10. Marchant, J.A. 1996. Tracking of row structure in three crops using image analysis. Computers and Electronics in Agriculture. 15:161-179

11. Reid, J.F. 2002. Personal Communication. John Deere Technical Center,

Moline, IL, USA

12. Reid, J.F., Q. Zhang, N. Noguchi and Monte Dickson. 2000. Agricultural automatic guidance research in North America. Computers and Electronics in Agriculture. 25:155-167

13. Siemens M.C. and W.E. Coates. 2000. Control system for the mobile truss of a cable-drawn farming system. Applied Engineering in Agriculture. 16(3):211-216

14. Shin, B.S., S.H. Kim and J.E. Park. 2002. Autonomous operation of orchard sprayer using overhead guidance. Proceedings of ATOE Conference : 123-131. ASAE, St. Joshep, MI.

15. Tillet, N.D., T. Hague and S.J. Miles. 2002. Inter-row vision guidance for mechanical weed control in sugar beet. Computers and Electronics in Agriculture. 33:163-177

16. Tillet, N.D. and T.G. Nybrant. 1990. Leader cable guidance of an experimental field gantry. J. agric. Engng Res. 45:253-267

17. Toda, M., O. Kitani, T. Okamoto and T. Torii. 1999. Navigation method for a mobile robot via sonar-based crop row mapping and fuzzy logic control. J. Agric. Engng Res. 72:299-309

18. Torii, T. 2000. Research in autonomous agricultural vehicles in Japan. Computers and Electronics in Agriculture. 25:133-153

19. Wei, J., N. Zhang, N. Wang, D. Qard, Q. Stoll, D. Lenhert, M. Neilsen, M. Mizuno and G. Sing. 2001. Design of an embedded weed-control system using controller area network. ASAE paper No. 013033. ASAE, St. Joshep, MI

20. Zhang, Q. and L. Guo. 2002. Personal Communication. University of Illinois at Urbana-Champaign.

부 록

1. T89C51CC01 driver 파일

```
//*****************************************************************************
// Module          On-Chip CAN Interface
// Filename        Can_drv.C
// Controller      AtmelWM 89C51CC01
//*****************************************************************************
#include <t89c51cc01.h>
#include <string.h>
#include "can_drv.h"    // define user CAN id


// Macros for CAN Message Object Control and DLC register (CANCONCH)
#define  CanTX          0x40        // Transmit
#define  CanRX          0x80        // Receive
#define  CanRXB         0xC0        // Receive in buffered mode
#define  CanCONCH       0xC0        // Mask for the CONCH field
#define  CanRPLV        0x20        // Remote Transmission Reply ready & valid
#define  CanIDE         0x10        // ID extended (29-bit)
#define  CanNOIDE       0x00        // ID extended (11-bit)

// Macros for CAN General Control register (CANGCON)
#define  CanABRQ        0x80        // Abort request
#define  CanOVRQ        0x40        // Overload frame request
#define  CanTTC         0x20        // Network in Timer Trigger communication
#define  CanSYNCTTC     0x10        // Synchronisation of TTC
#define  CanAUTOBAUD    0x08        // autobaud mode
#define  CanENA         0x02        // enable CAN controller
#define  CanGRES        0x01        // general reset

// Macros for CAN General Interrupt Enable register (CANGIE)
#define  CanENRX        0x20        // Enable receive interrupt
#define  CanENTX        0x10        // Enable transmit interrupt
#define  CanENERCH      0x08        // Enable message object error interrupt
#define  CanENBUF       0x04        // Enable BUF interrupt
#define  CanENERG       0x02        // Enable general error interrupt

// Macros for CAN General Interrupt register (CANGIT)
#define  CanCANIT       0x80        // General Interrupt Flag
#define  CanOVRTIM      0x20        // Overrun CAN Timer
#define  CanOVRBUF      0x10        // Overrun Buffer
#define  CanSERG        0x08        // Stuff error General
#define  CanCERG        0x04        // CRC    error General
#define  CanFERG        0x02        // Form   error General
#define  CanAERG        0x01        // Acknowledgement error General
```

```
// Macros for CAN Message Object Status register (CANSTCH)
#define  CanDLCW      0x80      // Data length code warning
#define  CanTXOK      0x40      // Transmit OK
#define  CanRXOK      0x20      // Receive OK
#define  CanBERR      0x10      // Bit error
#define  CanSERR      0x08      // Stuff error
#define  CanCERR      0x04      // CRC   error
#define  CanFERR      0x02      // Form  error
#define  CanAERR      0x01      // Acknowledgement error

// Macros for ID initilisiation in CANIDTx, CANIDMx registers
#define  CanID29(v)  (((unsigned long)(v))<<3)
#define  CanID11(v)  (((unsigned long)(v))<<21)//22
#define  CanIDEMSK   0x01      // enable comparison in CANIDM register
#define  CanRTRMSK   0x04      // enable Remote Transmission request in
CANIDM register

// Set value in CANIDTx registers
#define  CanSetIDT(v) CANIDT1=((unsigned char)((v)>>24));  \
                      CANIDT2=((unsigned char)((v)>>16));  \
                      CANIDT3=((unsigned char)((v)>> 8));  \
                      CANIDT4=((unsigned char)(v)) ^ conf;

// Set value in CANIDMx registers
#define  CanSetIDM(v) CANIDM1=((unsigned char)((v)>>24));  \
                      CANIDM2=((unsigned char)((v)>>16));  \
                      CANIDM3=((unsigned char)((v)>> 8));  \
                      CANIDM4=((unsigned char)(v));

// Convert Channel No. for CANPAGE register
#define  CanChannel(v)  ((v)<<4)


bit InterruptReadFlag=0;


#define  ID0     (CanID11(0x025))   // transmitter of left distance in polling mode
#define  ID1     (CanID11(0x097))   // receiver of right distance in polling mode
#define  ID2     (CanID11(0x099))

#define  ID3     (CanID11(0x072))
#define  ID4     (CanID11(0x072))

#define  ID5     (CanID11(0x0c1))
#define  ID6     (CanID11(0x0d1))

#define  ID7     (CanID11(0x061))   // transmitter of right distance in polling mode
```

```
// Define Channel 0 Object Identifiers (handled in polling mode)

#define  ID0MSK  (CanID11(0x3ff) | CanIDEMSK)
#define  ID0TYP  (CanTX | CanNOIDE | 2)          // Transmit, ID extended (29-bit),
                                                 // 1 Byte message length

// Define Channel 1 Object Identifiers (handled in polling mode)

#define  ID1MSK  (CanID11(0x0f0) | CanIDEMSK)
#define  ID1TYP  (CanRX | CanNOIDE | 2)          // Receive, ID extended (29-bit),
                                                 // 1 Byte message length

// Define Channel 2 Object Identifiers (handled in interrupt mode)

#define  ID2MSK  (CanID11(0x3ff) | CanIDEMSK)
#define  ID2TYP  (CanTX | CanNOIDE | 2)          // Transmit, ID extended (29-bit),
                                                 // 8 Byte message length
#define  ID2IntChk (CANSIT2 & 0x04)         // Interrupt Status for Message Object 2

// Define Channel 3 Object Identifiers (handled in interrupt mode)

#define  ID3MSK  (CanID11(0x0f0) | CanIDEMSK)
#define  ID3TYP  (CanRXB | CanNOIDE | 2)          // Receive, ID extended (29-bit),
                                                 // 8 Byte message length
#define  ID3IntChk (CANSIT2 & 0x08)         // Interrupt Status for Message Object 3

// Define Channel 4 Object Identifiers (handled in interrupt mode)

#define  ID4MSK  (CanID11(0x0f0) | CanIDEMSK)
#define  ID4TYP  (CanRXB | CanNOIDE | 2)          // Receive, ID extended (29-bit),
                                                  // 8 Byte message length
#define  ID4IntChk (CANSIT2 & 0x10)       // Interrupt Status for Message Object 4

// Define Channel 4 Object Identifiers ('client' for remote frame mode)

#define  ID5MSK  (CanID11(0x3ff) | CanIDEMSK)
#define  ID5TYP  (CanTX | CanNOIDE | 2)          // Transmit, ID extended (29-bit),
                                                     // 8 Byte message length

// Define Channel 5 Object Identifiers ('server' for remote frame mode)
#define  ID6MSK  (CanID11(0x3ff) | CanIDEMSK)
#define  ID6TYP  (CanRX | CanNOIDE | 2)          // Receive, ID extended (29-bit),
                                                 // 8 Byte message length


#define  ID6MSK  (CanID11(0x3ff) | CanIDEMSK)
```

```
#define  ID6TYP  (CanRX | CanNOIDE | 2)          // Receive, ID extended (29-bit),
                                                 // 8 Byte message length
//////////////////////////////////////////////////////////////////////////////


// IDxTYP table for all used message objects
unsigned char code id_typ[] =  {
  ID0TYP,
  ID1TYP,
  ID2TYP,
  ID3TYP,
  ID4TYP,
  ID5TYP,
  ID6TYP
};


// ===== CAN Baudrate Calculation ======================
// Tscl =      (1+BRP)/(XTAL)   // for X2 Mode = ON
// Tscl =   2*(1+BRP)/(XTAL)   // for X2 Mode = OFF
// Tbit = (1 + (PRS+1) + (PHS1+1) + (PHS2+1)) * Tscl

// Examples:
// 500KHz CAN Baudrate,  X2 Mode = OFF,  XTAL=12MHz
// BRP = 0, PRS = 2, PHS1 = 3, PHS2 = 3
// Tscl = 2/12.0MHz = 0.167 ㎲ec
// Tbit = (1 + (2+1)+(3+1)+(3+1)) * Tscl = 12 * 0.167uSec = 2 ㎲ec -> 500KHz
Baudrate

// Examples:
// 1MHz CAN Baudrate,  X2 Mode = ON,  XTAL=20MHz
// BRP = 1, PRS = 1, PHS1 = 3, PHS2 = 2
// Tscl = 2/20.0MHz = 0.100 ㎲ec
// Tbit = (1 + (1+1)+(3+1)+(2+1)) * Tscl = 10 * 0.100uSec = 1 ㎲ec -> 1MHz
Baudrate

#define BRP     0
#define SJW     0
#define PRS     2
#define PHS1    3
#define PHS2    3


/*
 * CAN Initialization:
 *     - reset CAN controller
 *     - initialize CAN message object (ID, MSK, control)
```

```
 *     - clear message object status register
 *     - clear unused message object (ID, MSK, control, status)
 *     - enable CAN controller
 */

void CanInit (unsigned char conf) {
  unsigned char i;

  CANGCON |= CanGRES;                    // Reset CAN controller
  ECAN = 0;                             // disable CAN interupt
  ETIM = 0;                             // disable CAN timer overrun interrupt

// Init CAN Message Object 0  (Transmit)
  CANPAGE = CanChannel (0);
  CanSetIDT(ID0);
  CanSetIDM(ID0MSK);
  CANCONCH = 0;                          // mark no information for transmit objects
  CANSTCH  = 0;

// Init CAN Message Object 1  (Receive)
  CANPAGE = CanChannel (1);
  CanSetIDT(ID1);
  CanSetIDM(ID1MSK);
  CANCONCH = ID1TYP;                     // Object waits for data
  CANSTCH  = 0;

// Init CAN Message Object 2  (Transmit)
  CANPAGE = CanChannel (2);
  CanSetIDT(ID2);
  CanSetIDM(ID2MSK);
  CANCONCH = 0;                          // mark no information for transmit objects
  CANSTCH  = 0;

// Init CAN Message Object 3  (Receive)
  CANPAGE = CanChannel (3);
  CanSetIDT(ID3);
  CanSetIDM(ID3MSK);
  CANCONCH = ID3TYP;                     // Object waits for data
  CANSTCH  = 0;

// Init CAN Message Object 4  (Receive)
  CANPAGE = CanChannel (4);
  CanSetIDT(ID4);
  CanSetIDM(ID4MSK);
  CANCONCH = ID4TYP;                     // Object waits for data
  CANSTCH  = 0;
```

```
// Init CAN Message Object 5  (for Remote Requests)
  CANPAGE = CanChannel (5);
  CanSetIDT(ID5);
  CanSetIDM(ID5MSK);
  CANCONCH = 0;                              // mark no information for remote request
objects
  CANSTCH  = 0;

// Init CAN Message Object 6  (for Remote Transmits)
  CANPAGE = CanChannel (6);
  CanSetIDT(ID6);
  CanSetIDM(ID6MSK);
  CANCONCH = ID6TYP;                         // mark no information for remote transmit
objects
  CANSTCH  = 0;

// Clear unused Message Object Buffers
  for (i = CanChannel (7); i <= CanChannel(14); i += CanChannel(1))  {
    CANPAGE = i;
    CanSetIDT(0);                            // Clear Object Buffer
    CanSetIDM(0);
    CANCONCH = 0;
    CANSTCH  = 0;
  }

// Set CAN Baudrate
  CANBT1 = ((BRP)<<1);                       // set CANBT1 register (BRP timing)
  CANBT2 = ((PRS)<<1) | ((SJW)<<5);   // set CANBT2 register (PRS & SJW
timing)
  CANBT3 = ((PHS1)<<1)| ((PHS2)<<4); // set CANBT3 register (PHS1 & PHS1
timing)

  CANGCON |= CanENA;                         // enable CAN controller

// If you are using interrupt driven CAN I/O enable interrupts

  CANGIE |= CanENRX | CanENTX;          // enable TX and RX interrupt
  CANIE2 |= 0x0C;                          // enable TX and RX interrupt
  ECAN = 1;                                // enable general CAN interupt
}


/** CAN I/O Routines for Polling Mode ***********************************/

/*
 * CanSend:
 *    Input Parameter:  ch :=  message object channel (0 .. 14)
```

```
 *                          p :=  Pointer to data
 *
 *    Return Value:        0     message transferred to buffer, transmission started
 *                        -1     message object not defined for transmission
 *                        -2     previous message not yet transferred
 *
 *     - check if CAN message object is defined for transmit
 *     - check if a previous message is transmitted OK
 *     - copy new message to message object buffer
 *     - set message object for transmission
 */
char CanSend (unsigned char ch,  unsigned char *p)  {
  unsigned char i;
  static unsigned char typ;

// check if CAN message object is defined for transmit
  if (ch >= sizeof (id_typ))        return (-1);
  typ = id_typ[ch];
  if ((typ & CanCONCH) != CanTX)  return (-1);

  i = typ & 0xF;                      // message length
  CANPAGE = CanChannel(ch);         // select CAN message object

  if ((CANCONCH & CanCONCH))  {   // CAN channel used before?
    if (!(CANSTCH & CanTXOK)) {
      return (-2);                  // previous message not yet send!
    }
  }

  CANCONCH = 0;                       // reset previous status
  CANSTCH  = 0;
  while (i)  {                        // copy information to message buffer
    CANMSG = *p++;
    i--;
  }
  CANCONCH = typ;                     // send information
  return (0);                         // message object copied to buffer
}



/*
 * CanRead:
 *    Input Parameter:  ch :=  message object channel (0 .. 14)
 *                          p :=  pointer to data buffer
 *
 *    Return Value:        0     n bytes of message transferred to data buffer
```

```
 *                      -1      message object not defined for receiving
 *                      -2      no message available
 *                      -3      messsage data length differs from definition
 *
 *     - check if CAN message object is defined for receiving
 *     - check if a message is received
 *     - copy received message to data buffer
 *     - set message object for receiving
 */
char CanRead (unsigned char ch,  unsigned char *p, unsigned char *address)  {
  unsigned char i, typ;

// check if CAN message object is defined for reception
  if (ch >= sizeof (id_typ))        return (-1);
  typ = id_typ[ch];
  if ((typ & CanCONCH) != CanRX)  return (-1);

  i = typ & 0xF;                      // message length
  CANPAGE = CanChannel(ch);         // select CAN message object

  if (!(CANSTCH & CanRXOK)) {
    return (-2);                      // no message available
  }

  if (CANSTCH & CanDLCW)  {
    return (-2);                      // incoming message does not have expected length
  }


////////////////////////////////////////////////////////////////////////////

  if(CANCONCH & CanIDE)
  {
    //pt_st_can_rx->id.tab[0] = CANIDT1 >> 3;
    //pt_st_can_rx->id.tab[1] = (CANIDT1 << 5) | (CANIDT2 >> 3);
    //pt_st_can_rx->id.tab[2] = (CANIDT2 << 5) | (CANIDT3 >> 3);
    //pt_st_can_rx->id.tab[3] = (CANIDT3 << 5) | (CANIDT4 >> 3);
        *address=(CANIDT3 << 5) | (CANIDT4 >> 3);
  }
  else
  {
    //pt_st_can_rx->id.std = (CANIDT1 << 3) | (CANIDT2 >> 5);
        *address=(CANIDT1 << 3) | (CANIDT2 >> 5);
  }

////////////////////////////////////////////////////////////////////////////
```

```
  CANCONCH = 0;                    // reset previous status
  CANSTCH  = 0;
  while (i)  {                     // copy information to message buffer
    *p++ = CANMSG;
    i--;
  }
  CANCONCH = typ;                  // set channel for new receiving
  return (0);                      // message object copied to buffer
}


/********CAN I/O Routines for Remote Frame Handling ********************/
/*
 * CanReqRemote:    Request Remote Frame
 *    Input Parameter:  ch :=  message object channel (0 .. 14)
 *
 *    Return Value:      0      n bytes of message transferred to data buffer
 *                      -1      message object not defined for transmission
 *
 *     - check if CAN message object is defined for transmission
 *     - set message object for remote frame request
 */
char CanReqRemote (unsigned char ch)  {
  unsigned char typ;


// check if CAN message object is defined for transmission
  if (ch >= sizeof (id_typ))       return (-1);
  typ = id_typ[ch];
  if ((typ & CanCONCH) != CanTX)  return (-1);

  CANPAGE = CanChannel(ch);        // select CAN message object
  CANCONCH = 0;                    // reset previous status
  CANSTCH  = 0;
  CANIDT4 |= CanRTRMSK;              // Set Remote transmission request value
  CANCONCH = typ;                  // set channel for transmission
  return (0);
}

/*
 * CanGetRemote:    Get Data from Remote Frame Request
 *    Input Parameter:  ch :=  message object channel (0 .. 14)
 *                       p :=  pointer to data buffer
 *
 *    Return Value:      0      n bytes of message transferred to data buffer
```

```
 *                        -1      message object not defined for transmission
 *                        -2      incoming message does not have expected length
 *                        -3      no remote message received
 *
 *     - check if CAN message object is defined for transmission
 *     - check if a message is received
 *     - copy received message to data buffer
 */
char CanGetRemote (unsigned char ch,  unsigned char *p)  {
  unsigned char i, typ;

// check if CAN message object is defined for transmission
  if (ch >= sizeof (id_typ))        return (-1);
  typ = id_typ[ch];
  if ((typ & CanCONCH) != CanTX)  return (-1);
  i = typ & 0xF;                      // message length

  CANPAGE = CanChannel(ch);         // select CAN message object
  if (!(CANSTCH & CanRXOK)) {
    return (-3);                    // no CAN message received
  }

  if (CANSTCH & CanDLCW)  {
    return (-2);                    // incoming message does not have expected length
  }

  CANCONCH = 0;                         // reset previous status
  CANSTCH  = 0;
  while (i)  {                      // copy information to message buffer
    *p++ = CANMSG;
    i--;
  }
  return (0);                       // message object copied to buffer
}


#pragma NOAREGS                             // function called from interrupt, no ARx
symbols!
/*
 * CanSetRemote:
 *    Input Parameter:  ch :=  message object channel (0 .. 14)
 *                        p :=  pointer to data buffer
 *
 *     - check if CAN message object is defined for remote reply
 *     - Fil data buffer with data
 */
char CanSetRemote (unsigned char ch,  unsigned char *p)  {
```

```
  unsigned char i, typ;
  unsigned char save_canpage;

// check if CAN message object is defined for reception
  if (ch >= sizeof (id_typ))        return (-1);
  typ = id_typ[ch];
  if ((typ & CanCONCH) != CanRX)  return (-1);

  i = typ & 0xF;                      // message length

  save_canpage = CANPAGE;             // Save the current CANPAGE (to work in
parallel with polling mode)
  CANPAGE = CanChannel(ch);         // select CAN message object

  CANCONCH &= ~CanRPLV;               // reset information valid

  while (i)  {                        // copy information to message buffer
    CANMSG = *p++;
    i--;
  }
  CANCONCH |= CanRPLV;                // set information valid
  CANPAGE = save_canpage;           // restore the old CAN page (required for polling
mode)

  return (0);                         // message object copied to buffer
}

#pragma AREGS

/***** Interrupt Driven CAN I/O Routines ************************************/

// definitions for transmit (output) buffer (CAN Message Object 2)
#define  OLEN  16                     // number of buffers in outbuf
unsigned char  ostart = 0;            // transmission buffer start index
unsigned char  oend = 0;              // transmission buffer end index
unsigned char  xdata outbuf[OLEN][8]; // transmission buffer
bit            tx_error;              // set when transmit error occur

// definitions for receive (input) buffer (CAN Message Object 3)
#define  ILEN  16                     // number of buffers in inbuf
unsigned char  istart = 0;            // receive buffer start index
unsigned char  iend = 0;              // receive buffer end index
unsigned char xdata inbuf[ILEN][8];   // receive buffer
unsigned char xdata idArr[ILEN];              //changed 4.30.2005


bit            rx_error;              // set when receive error occur
```

```
#pragma REGISTERBANK (1)                    // function called from Register Bank 1


/*
 * Read Data into inbuf (for CAN channels handled by Interrupt
 */


static void ReadToInBuf (void)  {
  unsigned char i;
  unsigned char xdata *p;

  unsigned char xdata *address;
//unsigned int      AddressInt;

  if (!(CANSTCH & CanRXOK))  {
    rx_error = 1;                          // not a receive OK interrupt
  }

////////////////////////////////////////////////////////////////////////////

  if(CANCONCH & CanIDE)
  {
        *address=(CANIDT3 << 5) | (CANIDT4 >> 3);
  }
  else
  {
        *address=(CANIDT1 << 3) | (CANIDT2 >> 5);
  }

////////////////////////////////////////////////////////////////////////////

  CANCONCH = 0;                            // reset previous status
  CANSTCH  = 0;

  p = inbuf[istart & (ILEN-1)];

        address = idArr[istart];                              //changed  4.30.2005

  istart++;
  for (i = sizeof (inbuf[0]); i != 0; i--)  {  // copy information to message buffer
    *p = CANMSG;
    p++;
  }

InterruptReadFlag=1;          // For LED
```

```
  CANCONCH = ID3TYP;                    // set channel for new receiving
}


/*
 * CanInterrupt:
 *    - called by hardware when a CAN message is send or received
 *    - checks if Channel 2 interrupt (transmission) occured
 *        - if new transmit data, copy new data to Channel 2 object buffers
 *    - checks if Channel 3 interrupt (receiving) occured
 *    - copy received data, copy data from Channel 3 object buffers
 */

void CanInt(void)
{

  unsigned char i;
  unsigned char typ;
  unsigned char save_canpage;
  unsigned char xdata *p;

  save_canpage = CANPAGE;               // Save the current CANPAGE (to work in
parallel with polling mode)

// Handle Interrupt for Transmit Channel 2
  if (ID2IntChk)   {                    // Check Channel 2 Interrupt (transmit buffer)
    CANPAGE = CanChannel(2);
    if (!(CANSTCH & CanTXOK)) {
      tx_error = 1;                     // not a transmit OK interrupt
    }

    CANCONCH = 0;                       // reset previous status
    CANSTCH  = 0;
    if (ostart != oend)  {
      p = outbuf[oend & (OLEN-1)];
      oend++;
      typ = id_typ[2];
      i = typ & 0xF;                    // message length

      while (i)  {                      // copy information to message buffer
        CANMSG = *p;
        p++;
        i--;
      }
      CANCONCH = typ;                   // send information
    }
  }
```

```
// Handle Interrupt for Receive Channel 3/4
  if (ID4IntChk)   {                    // Check if data are available in Channel 4
    CANPAGE = CanChannel (4);
    ReadToInBuf ();                      // Read Data into inbuf
  }

  if (ID3IntChk)   {                    // Check if data are available in Channel 4
    CANPAGE = CanChannel (3);
    ReadToInBuf ();                      // Read Data into inbuf
  }



  CANPAGE = save_canpage;                 // restore the old CAN page (required for
polling mode)
}

#pragma REGISTERBANK (0)              // use Register Bank 0 for following code


/*
 * CanSendIsr:
 *    Input Parameter:   ch :=   message object channel (0 .. 14)
 *                          p :=   pointer to data buffer
 *
 *    Return Value:       0       n bytes of message transferred to data buffer
 *                       -1       message object not defined for transmission
 *                       -2       no message available
 *                       -3       messsage data length differs from definition
 *
 *     - check if CAN message object is defined for receiving
 *     - check if a message is received
 *     - copy received message to data buffer
 *     - set message object for receiving
 */
char CanSendIsr (unsigned char ch,  unsigned char *p)  {
  unsigned char i, typ;

// check if CAN message object is defined for transmit
  if (ch >= sizeof (id_typ))         return (-1);
  typ = id_typ[ch];
  if ((typ & CanCONCH) != CanTX)   return (-1);

  i = typ & 0xF;                       // message length
```

```
  ECAN = 0;                          // disable CAN interupt

  CANPAGE = CanChannel(ch);          // select CAN message object
  if ((CANCONCH & CanCONCH))  {   // CAN channel busy?
                                     // yes copy to interrupt buffer
    memcpy (outbuf[ostart & (OLEN-1)], p, sizeof(outbuf[0]));
    ostart++;
  }
  else  {                            // not, transfer to message object
    CANCONCH = 0;                       // reset previous status
    CANSTCH  = 0;
    while (i)  {                     // copy information to message buffer
      CANMSG = *p++;
      i--;
    }
    CANCONCH = typ;                     // send information
  }
  ECAN = 1;                          // enable CAN interupt

  return (0);
}

/*
 * CanReadIsr:
 *    Input Parameter:  ch :=  message object channel (0 .. 14)
 *                       p :=  pointer to data buffer
 *
 *    Return Value:      0     n bytes of message transferred to data buffer
 *                      -1     message object not defined for receiving
 *                      -2     no data available
 *
 *     - check if CAN message object is defined for receiving
 *     - check if a message is received
 *     - copy received message to data buffer
 *     - set message object for receiving
 */
char CanReadIsr (unsigned char ch, unsigned char *p , unsigned char *address)  {
  unsigned char i, typ;

// check if CAN message object is defined for transmit
  if (ch >= sizeof (id_typ))        return (-1);
  typ = id_typ[ch];
  if ((typ & CanCONCH) != CanTX)  return (-1);

  i = typ & 0xF;                     // message length

  if (istart == iend)               return (-2);
```

```
    memcpy (p, inbuf[iend & (ILEN-1)], i);

  *address=idArr[iend];

    iend++;
    return (0);
}
```

2. PIC 18F448 driver 파일

```
/* CAN Library routines for Microchip's PIC18Cxx8 and 18Fxx8 line    */

#include <can-18xxx8.h>

#if CAN_DO_DEBUG
 #define can_debug printf
#else
 #define can_debug
#endif


//macros
#define can_kbhit()                    (RXB0CON.rxful || RXB1CON.rxful)
#define  can_tbe()                             (!TXB0CON.txreq  ||  !TXB1CON.txreq  ||
!TXB2CON.txreq)
#define can_abort()                    (CANCON.abat=1)


// can_init()
void can_init(void) {
    can_set_mode(CAN_OP_CONFIG);     //must be in config mode before params can
be set
    can_set_baud();

    RXB0CON=0;
    RXB0CON.rxm=CAN_RX_VALID;
    RXB0CON.rxb0dben=CAN_USE_RX_DOUBLE_BUFFER;
    RXB1CON=RXB0CON;

    CIOCON.endrhi=CAN_ENABLE_DRIVE_HIGH;
    CIOCON.cancap=CAN_ENABLE_CAN_CAPTURE;

    can_set_id(RX0MASK, RX0_MASK, CAN_USE_EXTENDED_ID);  //set mask 0
    can_set_id(RX0FILTER0, RX0_FILTER0, CAN_USE_EXTENDED_ID);   //set filter 0
of mask 0
    can_set_id(RX0FILTER1, RX0_FILTER1, CAN_USE_EXTENDED_ID);   //set filter 1
of mask 0

    can_set_id(RX1MASK, RX1_MASK, CAN_USE_EXTENDED_ID);  //set mask 1
    can_set_id(RX1FILTER2, RX1_FILTER2, CAN_USE_EXTENDED_ID);   //set filter 0
of mask 1
    can_set_id(RX1FILTER3, RX1_FILTER3, CAN_USE_EXTENDED_ID);   //set filter 1
of mask 1
    can_set_id(RX1FILTER4, RX1_FILTER4, CAN_USE_EXTENDED_ID);   //set filter 2
of mask 1
    can_set_id(RX1FILTER5, RX1_FILTER5, CAN_USE_EXTENDED_ID);   //set filter 3
```

```
of mask 1

    set_tris_b((*0xF93 & 0xFB ) | 0x08);    //b3 is out, b2 is in

    can_set_mode(CAN_OP_NORMAL);
}


// can_set_baud()
void can_set_baud(void) {
    BRGCON1.brp=CAN_BRG_PRESCALAR;
    BRGCON1.sjw=CAN_BRG_SYNCH_JUMP_WIDTH;

    BRGCON2.prseg=CAN_BRG_PROPAGATION_TIME;
    BRGCON2.seg1ph=CAN_BRG_PHASE_SEGMENT_1;
    BRGCON2.sam=CAN_BRG_SAM;
    BRGCON2.seg2phts=CAN_BRG_SEG_2_PHASE_TS;

    BRGCON3.seg2ph=CAN_BRG_PHASE_SEGMENT_2;
    BRGCON3.wakfil=CAN_BRG_WAKE_FILTER;
}

void can_set_mode(CAN_OP_MODE mode) {
    CANCON.reqop=mode;
    while( (CANSTAT.opmode) != mode );
}


// can_set_id()
//
// Configures the xxxxEIDL, xxxxEIDH, xxxxSIDL and xxxxSIDH registers to
// configure the defined buffer to use the specified ID
//
//    Paramaters:
//       addr - pointer to first byte of ID register, starting with xxxxEIDL.
//                For example, a pointer to RXM1EIDL
//       id - ID to set buffer to
//       ext - Set to TRUE if this buffer uses an extended ID, FALSE if not
//
void can_set_id(int* addr, int32 id, int1 ext) {
    int *ptr;

    ptr=addr;

    if (ext) {   //extended
        //eidl
        *ptr=make8(id,0); //0:7
```

```
        //eidh
        ptr--;
        *ptr=make8(id,1); //8:15

        //sidl
        ptr--;
        *ptr=make8(id,2) & 0x03;     //16:17
        *ptr|=(make8(id,2) << 3) & 0xE0; //18:20
        *ptr|=0x08;


        //sidh
        ptr--;
        *ptr=((make8(id,2) >> 5) & 0x07 ); //21:23
        *ptr|=((make8(id,3) << 3) & 0xF8);//24:28
    }
    else {    //standard
        //eidl
        *ptr=0;

        //eidh
        ptr--;
        *ptr=0;

        //sidl
        ptr--;
        *ptr=(make8(id,0) << 5) & 0xE0;

        //sidh
        ptr--;
        *ptr=(make8(id,0) >> 3) & 0x1F;
        *ptr|=(make8(id,1) << 5) & 0xE0;
    }
}

// can_get_id()
//
// Returns the ID of the specified buffer.  (The opposite of can_set_id())
// This is used after receiving a message, to see which ID sent the message.
//
//    Paramaters:
//       addr - pointer to first byte of ID register, starting with xxxxEIDL.
//              For example, a pointer to RXM1EIDL
//       ext - Set to TRUE if this buffer uses an extended ID, FALSE if not
//
//    Returns:
```

```
//      The ID of the buffer
//
int32 can_get_id(int * addr, int1 ext) {
    int32 ret;
    int * ptr;

    ret=0;
    ptr=addr;

    if (ext) {
        ret=*ptr;   //eidl

        ptr--;      //eidh
        ret|=((int32)*ptr << 8);

        ptr--;      //sidl
        ret|=((int32)*ptr & 0x03) << 16;
        ret|=((int32)*ptr & 0xE0) << 13;

        ptr--;      //sidh
        ret|=((int32)*ptr << 21);

    }
    else {
        ptr-=2;     //sidl
        ret=((int32)*ptr & 0xE0) >> 5;

        ptr--;      //sidh
        ret|=((int32)*ptr << 3);
    }

    return(ret);
}



// can_putd()
//
// Puts data on a transmit buffer, at which time the CAN peripheral will
// send when the CAN bus becomes available.
//
//     Paramaters:
//         id - ID to transmit data as
//         data - pointer to data to send
//         len - length of data to send
//         priority - priority of message.  The higher the number, the
//                    sooner the CAN peripheral will send the message.
```

```
//                      Numbers 0 through 3 are valid.
//          ext - TRUE to use an extended ID, FALSE if not
//          rtr - TRUE to set the RTR (request) bit in the ID, false if NOT
//
//      Returns:
//          If successful, it will return TRUE
//          If un-successful, will return FALSE
//
int1 can_putd(int32 id, int * data, int len, int priority, int1 ext, int1 rtr) {
    int i;
    int * txd0;
    int port;
//2005.7.28 수정  by Park Daesoon
//    txd0=&TXRXBaD0;
    txd0=&TXB0D0;
     // find emtpy transmitter
     //map access bank addresses to empty transmitter
    if (!TXB0CON.txreq) {
        CANCON.win=CAN_WIN_TX0;
        port=0;
    }
    else if (!TXB1CON.txreq) {
        CANCON.win=CAN_WIN_TX1;
        port=1;
    }
    else if (!TXB2CON.txreq) {
        CANCON.win=CAN_WIN_TX2;
        port=2;
    }
    else {
        #if CAN_DO_DEBUG
            can_debug("\r\nCAN_PUTD() FAIL: NO OPEN TX BUFFERS\r\n");
        #endif
        return(0);
    }

    //set priority.
    TXBaCON.txpri=priority;

    //set tx mask
    can_set_id(TXRXBaID, id, ext);

    //set tx data count
    TXBaDLC=len;
    TXBaDLC.rtr=rtr;

     for (i=0; i<len; i++) {
```

```
        *txd0=*data;
        txd0++;
        data++;
      }

    //enable transmission
    TXBaCON.txreq=1;

    CANCON.win=CAN_WIN_RX0;

    #if CAN_DO_DEBUG
                can_debug("\r\nCAN_PUTD():  BUFF=%U  ID=%LX  LEN=%U  PRI=%U
EXT=%U RTR=%U\r\n", port, id, len, priority, ext, rtr);
                if ((len)&&(!rtr)) {
                    data-=len;
                    can_debug("  DATA = ");
                    for (i=0;i<len;i++) {
                        can_debug("%X ",*data);
                        data++;
                    }
                    can_debug("\r\n");
                }
    #endif

    return(1);
}

// can_getd()
//
// Gets data from a receive buffer, if the data exists
//
//      Returns:
//        id - ID who sent message
//        data - pointer to array of data
//        len - length of received data
//        stat - structure holding some information (such as which buffer
//                 recieved it, ext or standard, etc)
//
//      Returns:
//        Function call returns a TRUE if there was data in a RX buffer, FALSE
//        if there was none.
//
int1 can_getd(int32 & id, int * data, int & len, struct rx_stat & stat)
{
     int i;
     int * ptr;
```

```
    if (RXB0CON.rxful) {
        CANCON.win=CAN_WIN_RX0;
        stat.buffer=0;

        CAN_INT_RXB0IF=0;

        stat.err_ovfl=COMSTAT.rx0ovfl;
        COMSTAT.rx0ovfl=0;

        if (RXB0CON.rxb0dben) {
          stat.filthit=RXB0CON.filthit0;
        }
    }
    else if ( RXB1CON.rxful )
    {
        CANCON.win=CAN_WIN_RX1;
        stat.buffer=1;

        CAN_INT_RXB1IF=0;

        stat.err_ovfl=COMSTAT.rx1ovfl;
        COMSTAT.rx1ovfl=0;

        stat.filthit=RXB1CON.filthit;
    }
    else {
      #if CAN_DO_DEBUG
          can_debug("\r\nFAIL ON CAN_GETD(): NO MESSAGE IN BUFFER\r\n");
      #endif
      return (0);
    }

    len = RXBaDLC.dlc;
    stat.rtr=RXBaDLC.rtr;

    stat.ext=TXRXBaSIDL.ext;
    id=can_get_id(TXRXBaID,stat.ext);
//2005.7.28 수정  by Park Daesoon
//    ptr = &TXRXBaD0;
    ptr = &RXB0D0;
    for ( i = 0; i < len; i++ ) {
        *data = *ptr;
        data++;
        ptr++;
    }

    // return to default addressing
```

```
    CANCON.win=CAN_WIN_RX0;

    stat.inv=CAN_INT_IRXIF;
    CAN_INT_IRXIF = 0;

    if (stat.buffer) {
      RXB1CON.rxful=0;
    }
    else {
      RXB0CON.rxful=0;
    }

    #if CAN_DO_DEBUG
        can_debug("\r\nCAN_GETD(): BUFF=%U ID=%LX LEN=%U OVF=%U ",
stat.buffer, id, len, stat.err_ovfl);
        can_debug("FILT=%U RTR=%U EXT=%U INV=%U", stat.filthit, stat.rtr,
stat.ext, stat.inv);
        if ((len)&&(!stat.rtr)) {
            data-=len;
            can_debug("\r\n    DATA = ");
            for (i=0;i<len;i++) {
              can_debug("%X ",*data);
              data++;
            }
        }
        can_debug("\r\n");
    #endif

    return(1);
}
```

3. ECU #1 : 조향/주행 클러치 제어기

```
/*****************************************************************
  ECU 1 : main controller
          auto_mode ; receiving "out_time" through CAN
                                              "ground speed"
                                               "remote_signal"
                      remote_mode ; receiving key values from remocon
*****************************************************************/
#include <18F448.h>
#use delay(clock=20000000)
#fuses NOWDT,WDT128,HS, NOPROTECT, NOOSCSEN, NOBROWNOUT, BORV20,
NOPUT, NOCPD, NOSTVREN, NODEBUG, NOLVP, NOWRT, NOWRTD, NOWRTB,
NOCPB, NOWRTC, NOEBTR, NOEBTRB
#use rs232(baud=57600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)

//port setting
#byte porte = 0xF84
#byte portd = 0xF83
#byte portc = 0xF82

#bit main_clutch = portd.5
#bit r_clutch = portd.6
#bit l_clutch = portd.7

#bit main_led = porte.0
#bit r_led= porte.1
#bit l_led = porte.2


//baud 500k seting
#define CAN_DO_DEBUG FALSE
#define CAN_USE_EXTENDED_ID 0
#define CAN_BRG_SYNCH_JUMP_WIDTH  0
#define CAN_BRG_PRESCALAR  1
#define CAN_BRG_SEG_2_PHASE_TS    TRUE
#define CAN_BRG_SAM 0
#define CAN_BRG_PHASE_SEGMENT_1  2
#define CAN_BRG_PROPAGATION_TIME 2
#define CAN_BRG_WAKE_FILTER FALSE
#define CAN_BRG_PHASE_SEGMENT_2 2
#define CAN_USE_RX_DOUBLE_BUFFER 0

//mask & filter define
#define RX0_MASK  0x0ff//set mask 0
#define RX0_FILTER0    0x050//set filter 0 of mask 0
```

```
#define RX0_FILTER1    0x035//set filter 1 of mask 0
#define RX1_MASK   0x0ff //set mask 1
#define RX1_FILTER2    0x035//set filter 0 of mask 1
#define RX1_FILTER3    0x050//set filter 1 of mask 1
#define RX1_FILTER4    0x000//set filter 2 of mask 1
#define RX1_FILTER5    0x000//set filter 3 of mask 1
//ID Define
#define Outtime         0x035
#define joysticksig   0x050
#define Ground_speed 0x060

#include <can-18xxx8.c>

#define disenage_time_left    200         // 180 ms
#define disenage_time_right    150         // 150 ms

//#define reenage_time    150         // 150 ms
int16 reenage_time;

//timer2 variable
int16 ms=0;
int16 cnt=0;

//CAN_Variable
struct rx_stat rxstat;

int32 rx_id;
int16 in_data[2];
int rx_len;


int16 out_data[2];
int tx_len=4;

//main variable
int mode_flag=0;
int16 main_clutch_state=0x0000;
int16 steering_state=0x0000;
int16 temp=0;
signed int16 out_time=0;   // 16 bit (-32,768 ... +32767)
int16 acting_time;
int out_time_flag=0;
int8 direction_flag; // if(direction_flag ==0) left else right

void key_check()//mode check
{
        if(bit_test(portd,0)==0) //19 auto mode button
```

```
        {
            mode_flag=1;
        }
        if(bit_test(portd,1)==0) //20 remote mode button
        {
            mode_flag=0;
        }
}
void remote_activation()
{

////////////////////////////////////--메인클러치 on/off
        if(main_clutch_state==0x0030) //19 main cluch Start button
        {
            main_clutch = 1;
            main_led=0;
        }
        if(main_clutch_state==0x0020) //20 main cluch stop button
        {
            main_clutch = 0;
            main_led=1;
        }

        if(steering_state==0x0001)  //21 오른쪽 방향 클러치 on
        {
            r_clutch = 1;
            l_clutch = 0;
            r_led = 0;
            l_led = 0;
        }
        if(steering_state==0x0002)  //22 왼쪽 방향 클러치 on
        {
            r_clutch = 0;
            l_clutch = 1;
            r_led = 1;
            l_led = 0;
        }
        if(steering_state==0x0003)  //27 오른쪽,왼쪽  방향 클러치 off
        {
            r_clutch = 0;
            l_clutch = 0;
            r_led=1;
            l_led=1;
        }
}

void activate_clutch(int16 acting_time)
```

```
{
    cnt = 0;
            while(cnt < acting_time)
            {

        if(direction_flag == 1) // turn right
                        {
            l_clutch=0;
            r_clutch=1;
            l_led=1;
            r_led=0;
        }
        else            // turn left
                        {
            l_clutch=1;
            r_clutch=0;
            l_led=0;
            r_led=1;
        }

    } // end of while(cnt < out_time)
    // re-engage clutches
    cnt = 0;
    while(cnt < reenage_time)
    {
        l_clutch=0;
        r_clutch=0;
        l_led=0;
        r_led=0;
    }
}


void display_data()
{
    printf("\r\n    outtime:%ld    acting_time:%lu    direct:%d    ",out_time,    acting_time,
direction_flag);
}

void CAN_VARIABLE_CLEAR()
{
    //CAN_VARIABLE_CLEAR
    int i=0;
    for(i=0;i<2;i++)
    {
        out_data[i]=0;
    }
```

```
    for (i=0;i<2;i++)
    {
        in_data[i]=0;
    }

}
void CAN_SELECT_DATA()
{
    Switch(rx_id)
    {
        case Outtime:
                    {
                        out_time=in_data[0];
                        temp=in_data[1];
                        out_time_flag = 1;
                        break;
                    }
                    case joysticksig:
                    {
                        main_clutch_state=in_data[0];
                        steering_state=in_data[1];
                        break;
                    }
                    /*
                    case Ground_speed:
                    {
                        left_rpm=in_data[0];
                        right_rpm=in_data[1];
                    }
                    */

    }
    CAN_VARIABLE_CLEAR();
}
#int_TIMER2
TIMER2_isr()
{
    ms++;
    cnt++;
}
//intruppt mode receiver
#int_CANRX1
CANRX1_isr()
{
        if(can_kbhit())
        {
```

```
            if(can_getd(rx_id, &in_data[0], rx_len, rxstat))
            CAN_SELECT_DATA();
        }
}

#int_CANRX0
CANRX0_isr()
{
        if(can_kbhit())
        {
            if(can_getd(rx_id, &in_data[0], rx_len, rxstat))
            CAN_SELECT_DATA();
        }
}


void main()
{
    signed int16 out_time_temp;  // 16 bit (-32,768 ... +32767)
    set_tris_d(0x1f);
    set_tris_e(0x00);

    setup_timer_2(T2_DIV_BY_16,33,10);//1ms
//initialize
    main_led=1;
    r_led=1;
    l_led=1;
// initialize clutches
    main_clutch = 0;  // main_cluch is disengaged
    r_clutch = 0;       // r_cluch is engaged
    l_clutch = 0;       // l_cluch is engaged

    can_init();

    enable_interrupts(INT_TIMER2);
    enable_interrupts(INT_CANRX1);
    enable_interrupts(INT_CANRX0);
    enable_interrupts(GLOBAL);

    CAN_VARIABLE_CLEAR();
    ms=0;
    cnt=0;

    while(TRUE)
    {
        key_check();
        if(mode_flag==0)
```

```
{
    cnt=0;
    remote_activation();
}
else if(out_time_flag==1)
{
    out_time_flag=0;
    if(main_clutch_state==0x0030) //19 main cluch Start button
    {
        main_clutch = 1;
        main_led=0;
    }
    if(main_clutch_state==0x0020) //20 main cluch stop button
    {
        main_clutch = 0;
        main_led=1;
    }
    if(out_time !=0)
    {
        if(out_time < 0 )
                                {
            direction_flag = 0; // turn left - disengage left
            out_time = - out_time;
            reenage_time=200;
        }
            else
        {
                direction_flag = 1; // turn right
          reenage_time=150;
        }
        out_time_temp=(int16)out_time*2;
        if(direction_flag==0)
        {
        acting_time = disenage_time_left + out_time_temp;
        }else
        {
        acting_time = disenage_time_right + out_time_temp;
        }
        activate_clutch(acting_time);
    }
    else
    {
        //nothing
    }
}

if(ms>50)
```

```
        {
            ms=0;
            display_data();
        }

    }
}
```

4. ECU #3 : 주행경로결정센서

```
#include <18F448.h>
#device ADC=10
#use delay(clock=20000000)
#fuses NOWDT,WDT128,HS, NOPROTECT, NOOSCSEN, NOBROWNOUT, BORV20, NOPUT, NOCPD, NOSTVREN, NODEBUG, NOLVP, NOWRT, NOWRTD, NOWRTB, NOCPB, NOWRTC, NOEBTR, NOEBTRB
#use rs232(baud=57600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)

//baud 500k seting
#define CAN_DO_DEBUG FALSE
#define CAN_USE_EXTENDED_ID 0
#define CAN_BRG_SYNCH_JUMP_WIDTH  0
#define CAN_BRG_PRESCALAR  1
#define CAN_BRG_SEG_2_PHASE_TS    TRUE
#define CAN_BRG_SAM 0
#define CAN_BRG_PHASE_SEGMENT_1  2
#define CAN_BRG_PROPAGATION_TIME 2
#define CAN_BRG_WAKE_FILTER FALSE
#define CAN_BRG_PHASE_SEGMENT_2 2
#define CAN_USE_RX_DOUBLE_BUFFER 0

//mask & filter define
#define RX0_MASK  0x0ff//set mask 0
#define RX0_FILTER0    0x033//set filter 0 of mask 0
#define RX0_FILTER1    0x060//set filter 1 of mask 0
#define RX1_MASK  0x0ff //set mask 1
#define RX1_FILTER2    0x033//set filter 0 of mask 1
#define RX1_FILTER3    0x060//set filter 1 of mask 1
#define RX1_FILTER4    0x000//set filter 2 of mask 1
#define RX1_FILTER5    0x000//set filter 3 of mask 1
//ID Define
#define TravelPath 0x030 //headingangle & x-offset
#define TrollyPath 0x031
#define TiltSense  0x033
#define Ground_speed 0x060

#include <can-18xxx8.c>
#include <ltc1298.c>
#include <math.h>

#define sprocket_r 85    // radius of driving sprocket in a track [mm]
#define t_interval 50    // time interval for heading angle calculation [ms]

//timer2 variable
int16 ms=0;
int16 heading_cnt=0;
```

```
//CAN_Variable
struct rx_stat rxstat;

int32 rx_id;
int16 in_data[2];
int rx_len;


int16 out_data[2];
int tx_len=4;

//main variable
signed int16 pitch=0;
signed int16 roll=0;
int16 left_rpm=0;
int16 right_rpm=0;
float x_angle=0;
float y_angle=0;
float z_length=0;
float heading_angle=0;
float x_offset=0;
float x_offset_temp=0;
float angle_temp;

int16 x_value,y_value,x_data,y_data;
//char ch0_volt_string[6];
//char ch1_volt_string[6];
int16 z_value;
int tx_flag=0;

void heading_math()
{
    if(left_rpm==0 && right_rpm==0){heading_angle=0;}
    else
    {
/*        if(x_offset<0)
        {
heading_angle=asin((x_offset_temp-x_offset)/((float)(left_rpm+right_rpm)*0.5)*sprocket_r
*t_interval*PI/3000.);
                heading_angle=heading_angle*180/PI;

                x_offset_temp=x_offset;
            }
                out_data[0]=(signed int16)(heading_angle*10);
                out_data[1]=(signed int16)(x_offset/10);

                can_putd(TravelPath, out_data, tx_len, 0, 0, 0);
```

```
}

void read_ad()
{
    x_data = read_analog(0);
    delay_us(50);
    y_data = read_analog(1);
    delay_us(50);
    //convert_to_volts(x_data,ch0_volt_string );
    //delay_us(100);
    //convert_to_volts(y_data,ch1_volt_string );
    //delay_us(100);
    x_value=((x_data>>4)&(0x0fff));
    y_value=((y_data>>4)&(0x0fff));
}

void display_data()
{
printf("\r\n hd_angle:%f x_offset:%f z_length:%f",heading_angle,x_offset,z_length);
printf(" x_angle:%f y_angle:%f",x_angle,y_angle);
}

void CAN_VARIABLE_CLEAR()
{
    //CAN_VARIABLE_CLEAR
    int i=0;
    for(i=0;i<2;i++)
    {
        out_data[i]=0;
    }

    for (i=0;i<2;i++)
    {
        in_data[i]=0;
    }

}

void CAN_SELECT_DATA()
{
    Switch(rx_id)
    {
        case TiltSense:
        {
            roll=in_data[0];
            pitch=in_data[1];
            break;
```

```
        }

        case Ground_speed:
        {
            left_rpm=in_data[0];
            right_rpm=in_data[1];
            break;
        }
    }

        CAN_VARIABLE_CLEAR();
}
#int_TIMER2
TIMER2_isr()
{
    if(heading_cnt>50)
    {
        heading_cnt=0;
        heading_math();
    }

    heading_cnt++;
    ms++;
}
//intruppt mode receiver
#int_CANRX1
CANRX1_isr()
{
        if(can_kbhit())
        {
            if(can_getd(rx_id, &in_data[0], rx_len, rxstat))
            CAN_SELECT_DATA();
        }
}

#int_CANRX0
CANRX0_isr()
{
        if(can_kbhit())
        {
            if(can_getd(rx_id, &in_data[0], rx_len, rxstat))
            CAN_SELECT_DATA();
        }
}
```

```
void main()
{
    setup_timer_2(T2_DIV_BY_16,33,10);//1ms
    setup_adc_ports(RA0_ANALOG);
    setup_adc(ADC_CLOCK_DIV_32);
    set_adc_channel(0);

    can_init();
    adc_init();//ltc1298 init

    enable_interrupts(INT_TIMER2);
    enable_interrupts(INT_CANRX1);
    enable_interrupts(INT_CANRX0);
    enable_interrupts(GLOBAL);

    CAN_VARIABLE_CLEAR();
    heading_cnt=0;
    ms=0;

    while(TRUE)
    {
/*      if(can_tbe() && can_cnt>25)
      {
          can_cnt=0;
          if(tx_flag==0)
          {
              tx_flag=1;
              out_data[0]=heading_angle;
              out_data[1]=x_offset;

              can_putd(TravelPath, out_data, tx_len, 0, 0, 0);
            //          id              data   len pri ext rtr
          }else if(tx_flag==1)
          {
              tx_flag=0;
              out_data[0]=y_angle;
              out_data[1]=z_length;

              can_putd(TrollyPath, out_data, tx_len, 0, 0, 0);
            //          id              data   len pri ext rtr
          }
      }
*/
      if(ms>50)
      {
          ms=0;
          read_ad();
```

```
            z_value=read_adc();

            x_angle=((-0.0183*(float)x_value)+36.487)+(float)roll/10;
            y_angle=((0.0188*(float)y_value)-40.224)+(float)pitch/10;
            z_length=((4.138*z_value)+34.5);//mm result out

                out_data[0]=(signed int16)y_angle*10;
                out_data[1]=(signed int16)z_length;

                can_putd(TrollyPath, out_data, tx_len, 0, 0, 0);

            angle_temp=((x_angle)*PI/180.);
            x_offset=z_length*sin(angle_temp);
            //x_offset=sin(angle_temp)/cos(angle_temp)*(float)129;

            display_data();
        }

    }
}
```

5. ECU #6.1 : 좌측 약액살포제어기와 약액수위센서

```
/*********************************************************
    ECU 6_1 : for Chemical Applicator Control
              - receiving 1st half of Message 0x27(1 byte) from ECU 8 : triggered
              - 0x25(1 byte) from ECU 5 : Row_end sensor
              - DO's for solenoid valves on left hand side
              - tank water level(0x36) is sensed
                    and transmitted when empty only
                  Input 0(pin 3) <-- sensor --> 5V(pin 21)
*********************************************************/
//#include <absacc.h>
#include <string.h>
#include <stdio.h>
#include <compiler.h>
#include <t89c51cc01.h>
//#include <REG51CC01.H>
#include "can_drv.h"                    // Interface to the CAN Functions

unsigned char buffer[8];                        // buffer for receiving objects
unsigned char addr[1];
unsigned short analog [4];          // voltage on analog Pins AN0 .. AN3

//  type definition ////////////////////////////////

// char                                 %bd
// int                                  %d
// long                                 %ld
// unsigned char   %bu
// unsigned int             %u
// unsigned long   %lu
// float                    %
////////////////////////////////////

#define   RLY0    P2_0
#define   RLY1    P2_1
#define   RLY2    P2_2
#define   RLY3    P2_3
#define   RLY4    P2_4
#define   RLY5    P2_5
#define   RLY6    P2_6
#define   RLY7    P2_7

#define   Input0    P0_0
#define   Input1    P0_1
#define   Input2    P0_2
#define   Input3    P0_3
```

```
#define   ID0                       P0_4
#define   ID1                       P0_5
#define   ID2                       P0_6
#define   ID3                       P0_7

#define   IDVal0          0x10
#define   IDVal1          0x20
#define   IDVal2          0x40
#define   IDVal3          0x80



#define   AD0             P1_0
#define   AD1             P1_1
#define   AD2             P1_2
#define   AD3             P1_3


static unsigned char level[2];

unsigned char Flag100, Flag500;
unsigned int FlagCount=0;
unsigned char flag;

bit PollingReadFlag=0,RemoteReadFlag=0;
extern bit InterruptReadFlag;


/*****************************************************************/
/*          Timer 0 interrupt service function                   */
/*          executes each 1ms @ 12 MHz Crystal Clock             */
/*****************************************************************/
#define INT_CLOCK          0.01       // timer interval 0.001Sec
#define XTAL       12000000
#define X2                  0    // set to 1 if in X2 Mode

#define _CLOCKVAL   ((unsigned int) ((XTAL/(12/(X2+1))) * INT_CLOCK))

#define T0_CLOCK ((-_CLOCKVAL)+13)


/*****************************************************************/
/*          Timer 2 interrupt service function                   */
/*          executes each 1ms @ 12 MHz Crystal Clock             */
/*****************************************************************/

#define TIMER2_2ms    (0xFFFF - 2000)
```

```
void uart_init (void)
{
  SCON = 0x50;
  TMOD = TMOD | 0x20 ;        /*  Timer1 in mode 2 & not gated */

  TH1 = 0xF3;                 /*  4800 bauds at 12 MHZ */
  TL1 = 0xF3;

  PCON |= 0X80; //double baud rate 2400*2    SMOD = 1
  TCON |= 0x40;
  TI=1;
}


void init_timer0(void)
{
                              /* setup the timer 0 interrupt */
  TL0  = (unsigned char) (T0_CLOCK);
  TH0  = (unsigned char) (T0_CLOCK >> 8);
  TMOD = TMOD | 0x01;              /* select mode 1                */
  TR0  = 1;                        /* start timer 0                */
  ET0  = 1;                        /* enable timer 0 interrupt     */
}


Interrupt(timer0(void), 1)     //1ms주기 인터럽트
{

  unsigned char i;
  unsigned int  v;

  TR0 = 0;  // stop timer
  v = TL0 | (TH0>>8);
  v    += (unsigned int) (T0_CLOCK);
  TL0  = (unsigned char) v;
  TH0  = (unsigned char) (v >> 8);
  TR0 = 1;  // start timer

  for (i = 0; i != 4; i++)  {                 // loop for 4 A/D inputs
    ADCON = (0x28 | i);                       // select A/D input
    while (!(ADCON & 0x10));                  // wait for A/D result
    analog[i] =  (ADDH<<2) | (ADDL);          // result of A/D process

  }
```

```
CanSetRemote(6, (void *) analog);       // update remote message with new A/D values
}


void init_timer2(void)
{
/* 16 bit auto reload mode */
/*-------------------------*/
  T2CON = 0;
  TH2 = 0;
  TL2 = 0;
  RCAP2L = LOW(TIMER2_2ms);
  RCAP2H = HIGH(TIMER2_2ms);

/* enable timer 2 interrupt. */
/*---------------------------*/
  ET2 = 1;

/* run timer 2. */
/*--------------*/
  TR2 = 1;
}



Interrupt(fct_timer2_it(void), 5)          //2ms주기 인터럽트
{
  TF2 = 0; /* raz flag interrupt. */

FlagCount++;

          if(FlagCount%50==0) {Flag100=1;}
          if(FlagCount%250==0) {Flag500=1;FlagCount=0;}
}



/*
 * Send a string via the Can Interface
 */
static void SendString (char *s)  {
  unsigned char len, i;

  len = strlen (s) + 1;                          // length of the string to send
  while (len)  {
    if (len > 8)   i = sizeof (buffer);              // split the message into CAN
size-compatible strings (8 Bytes) if bigger
    else           i = len;
```

```
      memcpy (buffer, s, i);                    // copy to send buffer
      len -= i;
      s   += i;
      if (CanSendIsr (2, buffer) < 0)  return; // send message
    }
}


/*
 * Read a string from the Can Interface
 */
static void ReadString (void)  {
  unsigned char i;

  if (CanReadIsr (2, buffer) < 0)  return;  // no new message received by interrupt
  for (i = 0; i < 8; i++)  {
    if (!buffer[i])  return;                // no more info in receive buffer
    putchar (buffer[i]);                    // buffer out on serial port
  }
}

/****************************************
 for nozzle control
****************************************/
#define all_on  0x1f
#define all_off 0x00

#define app_delay 42  // distance between sensor and nozzle , number of wheel counts
                      // 5cm x 20 = 1m delay

unsigned int trig, cycle, count, stopcount, stop;
unsigned char outword_L;
unsigned char out_stack_L[app_delay];
unsigned char left_distance;

void SET_UP(void)
{
    unsigned char i;
    for(i=0; i<app_delay; i++)
    {
     out_stack_L[i] = all_off;
    }
    outword_L = all_off;
    P2 = outword_L;
    printf("P2 in setup = %bu\n", P2);
}
```

```
unsigned char TRIGGER(void)
{

  if (CanRead (1, buffer,addr) == 0) // check whether triggered
       {    PollingReadFlag=1;            //polling mode
//             printf("CAN Read= %bu ,address= %bu\n", *buffer,*addr);
       } // end of if(CanRead(    ))
  if( *addr == 0x65)
  {
    printf("CAN Read from 0x65= %bu   , address = %bu\n",*buffer,*addr);
    RLY5 = 1;
    return 0;
  }
  else
  {
    RLY5 = 0;
          return 1;
  }

}

void nozzle_control_left(void)
{
    if(left_distance < 100 & left_distance >10) outword_L = all_on;
    else outword_L = all_off;
}

void UPDATE(void)
{
   unsigned char i;
   for(i = 0; i<app_delay-1; i++)
   {
     out_stack_L[i]=out_stack_L[i+1];
   }
   out_stack_L[app_delay - 1] = outword_L;

   outword_L=all_off;
}

// sensing the lowest level in a chemical tank
unsigned char read_tank_level(void)   {
   if(Input0 != 0)  {
    RLY6 = 1;
    return(1);
  }
   else
    return(0);
```

```
}

//////////////////////////////////////////////////////////////
///                          main function
//////////////////////////////////////////////////////////////

void main (void)
{
  bit CanMessageReq = 0;         // Remote Frame Requested

  bit LedFlag=0;

// init serial interface
uart_init();


// init Timer0,Timer2 interrupt
init_timer0();
init_timer2();

/* setup ADC */
  ADCF    = 0x0F;    /* open ports for the ADC (P1.0 - P1.3) */
  ADCLK   = 20/4;                                    // set ADCLK


  CanInit (1);//(DipSwitch & 0x01) << 3);


  EA = 1;

   RLY5=RLY6=RLY7 = 0; // level sensor arlam initialize

                                          // enable interrupts
  SET_UP();// initialize solenoid valves


  while(1)
  {
    while(TRIGGER() == 0)
    {
      if (CanRead (1, buffer,addr) == 0)
      {    PollingReadFlag=1;            //polling mode
//              printf("CAN Read= %bu ,address= %bu\n", *buffer,*addr);
            if(*addr == 0x25)
            {
             left_distance = *buffer;
              printf("CAN Read from 0x25= %bu\n", left_distance);
```

```
//       count++;
            }
   }

    P2 = out_stack_L[0];
    printf("P2 in first main()= %bu\n", P2);//      P2=*buffer;
     nozzle_control_left();
     UPDATE();
   } // end of while(TRIGGER() == 0);

/*
     if (CanMessageReq)  {                // CAN remote frame requested before?
       if  (CanGetRemote  (5,  buffer)  ==  0)  {     //check  &  read  requested  remote
                                          //frame
                    RemoteReadFlag=1;

         CanMessageReq = 0;                              // remote frame is there
////          PrintAnalogValues ((unsigned short *) buffer);
       }
     }//end of if (CanMessageReq)

*/
     ReadString();                       // read CAN message received in interrupt



     if (Flag500==1)
     {    Flag500=0;

//     if (CanReqRemote (5) == 0) {                // request CAN remote frame
//                                  CanMessageReq = 1;// set marker
//                                          }


//               SendString("This is a string send via CAN interrupt\n");

         if( read_tank_level())
          {
                        if (CanSend (0, &level) == 0)
                  {//polling mode
                   //printf("CAN Send= %bd\n",CntInfo);
                       level = Input0;
                  }
           }


          }//end of if (Flag500==1)
```

```
////////////////////////////////////////////////////////////////////////////
////////  CAN Communication Status ////////////////////////////////////////////

                        if(Flag100)
                        {               Flag100=0;

                        if(LedFlag){LedFlag=0;
                        RLY7=0;
                        }

                        else
                      {LedFlag=1;
                       RLY7=1;
                        }
              } // end of if(Flag100)



   } //end of while(1)
    P2 = all_off;
}

Interrupt(CanInterrupt(void), 7)
{
              CanInt();
}
```

[별지 제6호 서식]

농림기술개발사업 연구결과 활용계획서

1. 연구과제 개요

사 업 구 분	(첨단기술개발사업)		분 야	기계화
연 구 과 제 명	CAN 기반의 농업기계 무인화 기술개발			
주관연구기관	강원대학교		총괄연구책임자	신범수
연 구 개 발 비	정부출연금	기업부담금	연구기관부담금	총연구개발비
	140,000천원			140,000천원
연구개발기간	2003. 7.15 - 2005. 7. 14			
주요활용유형	■ 산업체이전 □ 교육 및 지도 □ 정책자료 □ 기타( ) □ 미활용 (사유:)			

2. 연구목표 대비 결과

당초목표	당초연구목표 대비 연구결과
① CAN 기반 시스템 제어 기술 개발	CAN통신 기능의 마이크로프로세서(T89C51CC01, PIC18F448)로 ECU(Electronic Control Unit)를 구성하여 각종 센서 또는 조작기 등을 인터페이스하여 CAN 버스로 구성 후, 전체 시스템을 제어할 수 있도록, ECU의 H/W 설계, 구동 S/W 설계 기술을 개발함.
② 과수방제기의 무인화 기술 개발	과수방제기의 무인주행 제어를 위한 센서 시스템과 무인주행 제어 알고리즘을 개발함. 오버헤드가이던스 레일을 개발하고 설치 방법을 제시함. 전동트롤리, 각도측정 및 케이블권취장치 등 차량자세 측정 센서시스템을 개발함,

3. 핵심기술

구분	핵 심 기 술 명
①	CAN 버스를 구성할 수 있는 ECU의 H/W 및 S/W 개발 기술
②	오버헤드가이던스 레일을 이용한 과수방제기의 무인주행 기술

4. 연구결과별 기술적 수준

구분	핵심기술 수준					기술의 활용유형(복수표기 가능)				
	세계최초	국내최초	외국기술 복 제	외국기술 소화·흡수	외국기술 개선·개량	특허출원	산업체이전 (상품화)	현장애로 해 결	정책자료	기타
①의 기술				v			v			
②의 기술		v				v	v			

* 각 해당란에 v 표시

5. 각 연구결과별 구체적 활용계획

핵심기술명	핵심기술별 연구결과활용계획 및 기대효과
①의 기술	관심있는 업체에 CAN 버스를 이용한 시스템 제어기술 이전 및 보급. 센서 또는 조작기 등의 단위 ECU를 개발하여 기존의 콘트롤러에 용이하게 인터페이스될 수 있도록 개발하여 상용화를 추진함
②의 기술	관심있는 업체에 과수방제기 무인화를 위한 제어 기술을 이전하고, 업체와의 공동 개발을 통해 상용화 단계로 시스템을 업그레이드한 후, 지자체 등과 협조하여 시범 사업으로 추진할 수 있음

6. 연구결과의 기술이전조건(산업체이전 및 상품화연구결과에 한함)

핵심기술명			
이전형태	□무상 □유상	기술료 예정액	천원
이전방식	□소유권이전 □전용실시권 □통상실시권 □협의결정 □기타( )		
이전소요기간		실용화예상시기	
기술이전시 선행조건			

* 핵심기술이 2개 이상일 경우에는 각 핵심기술별로 위의 표를 별도로 작성
* 기술이전시 선행요건 : 기술실시계약을 체결하기 위한 제반 사전협의사항(기술지도, 설비 및 장비 등 기술이전 전에 실시기업에서 갖추어야 할 조건을 기재)
* 실용화예상시기 : 상품화인 경우 상품의 최초 출시 시기, 공정개선인 경우 공정개선 완료시기 등