IMPLEMENTATION OF A FUZZY LOGIC BASED SEEDING DEPTH CONTROL SYSTEM

By

THOMAS R. TESSIER

A Thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada


© T. Tessier; January 20, 1997
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-23523-8
IMPLEMENTATION OF A FUZZY LOGIC BASED
SEEDING DEPTH CONTROL SYSTEM

BY

THOMAS R. TESSIER

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

Thomas R. Tessier 1997 (c)

Permission has been granted to the Library of The University of Manitoba to lend or sell copies of this thesis/practicum, to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film, and to Dissertations Abstracts International to publish an abstract of this thesis/practicum.

The author reserves other publication rights, and neither this thesis/practicum nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.
ABSTRACT

Imprecise seeding depth can lead to lower crop yields and increased power requirements. For example, a variation of 2.56 cm (1 inch) from optimum depth results in 45%, 25%, and 15% reduction in the emergence of soybeans, sorghum, and wheat, respectively. Problems encountered with traditional linear seeding depth control systems include oscillation, instability, and inaccurate depth control in undulating terrain. Variations of 1.27 cm to 4.45 cm (0.5 to 1.75 inches) are not uncommon.

The control system in this research utilizes a microprocessor-based fuzzy logic control process in combination with a novel seeding depth gauge system. This system combines the human control reasoning of fuzzy logic with an electro-hydraulic seeding depth controller. The fuzzy algorithm is modelled upon expert human control by means of linguistic variables that describe the error between the measured seeding depth and the desired depth set point. The inputs of the fuzzy control rules are the error between the measured seeding depth and the desired seeding depth, and the rate of change in error (the difference between the present and a past sampling instance). These variables describe the direction of the error (positive, P or negative, N) as well as the magnitude (small, S, medium, M or big B). The change in error is monitored in order to provide differential control. Stability of the control system is ensured by monitoring these proportional and differential input signals. The microprocessor unit used in the controller is the Motorola MC68HC16Z1.

Three scenarios were examined: a 3-membership function, a 5-membership function, and a 7-membership function fuzzy control algorithm. In each algorithm simulation, the variation in output gain, complexity, and number of MPU cycles to execute was observed. The 5-membership function algorithm using triangular membership functions was found to provide the best combination of control and complexity for this application.

Two controller units were constructed, for testing in laboratory simulations and under field conditions. The 5-membership function algorithm was tested under field conditions. Data was logged from the field trials, with error mean from the set point being 2 mm, with variation and standard deviation being 17.1 cm and 2.8 cm, respectively, based on converted A/D readings.
ACKNOWLEDGEMENTS

I wish to acknowledge the guiding efforts of my advisor, Dr. Witold Kinsner, for providing the topic of this research and for taking the photos of the controller modules. His encouragement and faith in what his students can achieve serves as testament to his dedication to the advancement of knowledge.

I also wish to express my heartfelt thanks to George Gamby, President of Profile Resource Management. George and I spent many days and evenings at the workshop and in the fields of Gamby Farms Ltd. during both static tests throughout 1994, 1995 and 1996, and field testing in fall '96. I also wish to acknowledge the support of Gamby Farms Ltd. for the use of the workshop, tractor and discer throughout this research, as well as Joseph Gamby for his assistance with the sensors.

In pursuing a cross-disciplinary approach to research, I had the help and support of many graduate students in the Department that were always willing to take the time to explain a new concept or approach to a problem, and always to share their enthusiasm for learning. Special thanks goes to Dr. Ken Ferens, Dr. Armein Langi, Geoff Stacey, Warren Greider, Eric Jang, and Alex McIlraith.

I wish to express thanks to my colleagues at Akjuit Aerospace Incorporated. Their encouragement and support are very much appreciated. Thanks also goes to Vlad Kowalyk of New Holland Canada, who supplied the sensors and the proportional valve, as well as advice on suppression of noise in electrical devices on agricultural tractors.

This work was financially supported in part by Profile Resource Management, Inc. of Winnipeg and the Natural Sciences and Engineering Research Council of Canada. Technical and software support was also received from the Department of Electrical and Computer Engineering of the University of Manitoba.

This thesis is dedicated to my supportive parents, Gail and Clifford Tessier, and my loving wife Angela, who worked with me every step of the way to the conclusion of this work. It is also dedicated to our daughter Adrienne, who always keeps life in perspective for her parents!
# Table of Contents

Abstract ........................................................................................................ iii
Acknowledgements ......................................................................................... iv
Table of Contents ............................................................................................ v
List of Figures .................................................................................................. ix
List of Equations ............................................................................................... xi
List of Abbreviations ......................................................................................... xii

Chapter 1:
Introduction .................................................................................................... 1
1.1 Background and Motivation ....................................................................... 1
1.2 Thesis Objectives ....................................................................................... 2
1.3 Overview of Thesis Chapter Contributions ............................................. 3

Chapter 2:
Control of Seeding Depth ............................................................................... 7
2.1 Effects of Seeding Depth on Crop Emergence .......................................... 7
2.2 Types and Accuracy of Seeding Implements ........................................... 9
2.3 Advantages to using discer seeders in the Red River Valley ................. 13
2.4 Methods of Depth Control ..................................................................... 14
2.5 Chapter Overview ...................................................................................... 16

Chapter 3:
Fuzzy Logic Control ...................................................................................... 19
3.1 Fuzzy vs. Conventional Control ............................................................. 19
3.2 Formulating a Fuzzy Logic Control Algorithm ....................................... 22
   Fuzzy Control Rules ................................................................................. 22
   Membership Functions .............................................................................. 23
   Fuzzification ............................................................................................ 24
   Fuzzy Inference ....................................................................................... 26
   Defuzzification ......................................................................................... 31
   Summary of the Decision Process .......................................................... 34
3.3 Representation and Analysis of a Fuzzy Logic Controller ....................... 34
3.4 Design of the Fuzzy Logic Controller .................................................... 36
3.5 Chapter Overview ..................................................................................... 39
Chapter 4:

System Definition of the Fuzzy Seeding Depth Controller ........................................ 41
  4.1 Results of Seeding Depth Studies ................................................................. 41
  4.2 System Definition Criteria ............................................................................ 44
  4.3 Seeding Depth Control System Hardware ...................................................... 46
  4.4 Seeding Depth Control System Software ....................................................... 50
  4.5 Chapter Overview ......................................................................................... 51

Chapter 5:

Detailed Hardware Design of the Fuzzy Logic-Based Depth Controller ............... 53
  5.1 The Controller Module .................................................................................... 55
  5.2 The Depth Sensor Module .............................................................................. 61
  5.3 The Hydraulic Control Module ..................................................................... 63
  5.4 Electrical Noise and EMI Suppression ......................................................... 65
  5.5 Power Supply Module .................................................................................. 65
  5.6 Chapter Overview ......................................................................................... 67

Chapter 6:

Software Design of the Fuzzy Logic-Based Seeding Depth Controller ............... 69
  6.1 Initialization and Data Preparation Routines ................................................. 70
    Initialization ................................................................. 70
    CHECK_KEYS Routine .................................................. 72
    SAMPLE Routine .......................................................... 77
    ERROR_CONVERT Routine ........................................... 78
    ERROR_BUFFER Routine ............................................... 78
  6.2 The Fuzzy Control Subroutine: ................................................................. 78
    FUZZYTEST Routine ...................................................... 80
  6.3 Hydraulic, LCD Display and Data Logging Routines ............................... 84
    HYD_CONT Routine ...................................................... 84
    Display_Depth Routine .................................................. 85
    Send_Data Routine ........................................................ 86
  6.4 Chapter Overview ......................................................................................... 88

Chapter 7:

System Implementation ....................................................................................... 89
  7.1 Field Test and Data Logging Mode ............................................................... 89
    Operational Issues .......................................................... 90
    Field Operation ............................................................... 91
  7.2 Calibration Mode .......................................................................................... 96
    Keypad Functions ............................................................. 97


<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3</td>
<td>Chapter Overview</td>
<td>101</td>
</tr>
<tr>
<td>8.0</td>
<td>Discussion and Analysis of Fuzzy Logic Based Seeding Depth Controller</td>
<td>103</td>
</tr>
<tr>
<td>8.1</td>
<td>Fuzzy Control</td>
<td>103</td>
</tr>
<tr>
<td>8.2</td>
<td>Simulation</td>
<td>107</td>
</tr>
<tr>
<td>8.3</td>
<td>Lab Testing</td>
<td>114</td>
</tr>
<tr>
<td>8.4</td>
<td>Field Testing</td>
<td>115</td>
</tr>
<tr>
<td>8.5</td>
<td>Chapter Summary</td>
<td>123</td>
</tr>
<tr>
<td>9.0</td>
<td>Conclusions And Recommendations</td>
<td>125</td>
</tr>
<tr>
<td>9.1</td>
<td>Performance of the Sensors</td>
<td>125</td>
</tr>
<tr>
<td>9.2</td>
<td>Hardware Observations and Recommendations</td>
<td>126</td>
</tr>
<tr>
<td>9.3</td>
<td>Algorithm Development Recommended Improvements</td>
<td>127</td>
</tr>
<tr>
<td>9.4</td>
<td>Production of a Fuzzy CASE Tool</td>
<td>129</td>
</tr>
<tr>
<td>9.5</td>
<td>Future Work</td>
<td>129</td>
</tr>
<tr>
<td>9.6</td>
<td>Chapter Overview</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td>133</td>
</tr>
<tr>
<td>10.0</td>
<td>Appendix A: Fuzzy Depth Controller Schematic Diagrams</td>
<td>A-1</td>
</tr>
<tr>
<td>11.0</td>
<td>Appendix B: Software Flowchart - Fuzzy Controller</td>
<td>B-1</td>
</tr>
<tr>
<td>B.1</td>
<td>Controller Initialization</td>
<td>B-1</td>
</tr>
<tr>
<td>B.2</td>
<td>Main Program Execution 2</td>
<td>B-8</td>
</tr>
<tr>
<td></td>
<td>CHECK_KEYS Flowchart</td>
<td>B-8</td>
</tr>
<tr>
<td></td>
<td>RAISED_POSITION_SET</td>
<td>B-10</td>
</tr>
<tr>
<td></td>
<td>Setting Disk Raised Position</td>
<td>B-11</td>
</tr>
<tr>
<td></td>
<td>Setting Disk Raised Position</td>
<td>B-12</td>
</tr>
<tr>
<td></td>
<td>Choosing Imperial or Metric Units</td>
<td>B-13</td>
</tr>
<tr>
<td></td>
<td>MOVETO_UP</td>
<td>B-14</td>
</tr>
<tr>
<td></td>
<td>SET_OPERATINGDEPTH</td>
<td>B-15</td>
</tr>
<tr>
<td></td>
<td>FIELD_0_POSITION_SET</td>
<td>B-16</td>
</tr>
<tr>
<td></td>
<td>CALIBRATE</td>
<td>B-17</td>
</tr>
<tr>
<td></td>
<td>90 Deg. Sensor Arm Position Set</td>
<td>B-18</td>
</tr>
<tr>
<td></td>
<td>Disc Sensor Calibration</td>
<td>B-19</td>
</tr>
<tr>
<td></td>
<td>Calibrate Front Sensor</td>
<td>B-20</td>
</tr>
<tr>
<td></td>
<td>MOVETO_DOWN</td>
<td>B-21</td>
</tr>
<tr>
<td></td>
<td>Sample Depth Sensors</td>
<td>B-22</td>
</tr>
<tr>
<td></td>
<td>ERROR_CONVERT</td>
<td>B-23</td>
</tr>
<tr>
<td></td>
<td>ERROR_BUFFER</td>
<td>B-24</td>
</tr>
<tr>
<td></td>
<td>FUZZYTEST</td>
<td>B-25</td>
</tr>
</tbody>
</table>
Appendix C: Software Source Code - Fuzzy Controller .............................................C-1
Appendix D: PC Logging Program ..............................................................................D-1
Appendix E: Mathcad Simulation ..................................................................................E-1
LIST OF FIGURES

Fig. 1.1. Typical discer seeder .................................................................2
Fig. 2.1. Crop emergence as a function of seeding depth .....8
Fig. 2.2. Double disc seed drill opener detail ..........10
Fig. 2.3. Airseeder system with high capacity seed hopper, air blower and cultivator 11
Fig. 2.4. One way discer seeder detail .................................12
Fig. 2.5. Discer seeder used in this research ............................14
Fig. 2.6. Illustration of the distance from the cab to the disk gangs .....15
Fig. 3.1. Membership functions, each a reference fuzzy set of the term set for error 24
Fig. 3.2. Fuzzification of the crisp input X0 into the membership functions of error .......25
Fig. 3.3. Fuzzification of the crisp input X1 into the membership fcns of chg in error 25
Fig. 3.4. Fuzzy control rule summary ...............................28
Fig. 3.5. Alpha cuts for the output Dk(k) when N = 0.0, Z = 0.3, P = 0.25. 32
Fig. 3.6. The resulting mean of maxima (MOM) output for change in output. 33
Fig. 3.7. Geometric representation of fuzzy sets A and B 35
Fig. 3.8. A fuzzy control output surface ..........36
Fig. 3.9. The Fuzzy Logic Controller (FLC) in the framework of a control system 38
Fig. 4.1. View of tandem discsers from the tractor cab ..........43
Fig. 4.2. System block diagram-seeding depth controller 48
Fig. 5.1. Lab test unit .........................................................53
Fig. 5.2. Field test and data logging unit ...........................54
Fig. 5.3. Controller main modules ................................54
Fig. 5.4. Main control box connectors .................................55
Fig. 5.5. Block Diagram of the MC68HC16Z1 showing main modules and ports 56
Fig. 5.6. Controller module main components 57
Fig. 5.7. Keypad connections to Port F of MC68HC16Z1 60
Fig. 5.8. Front sensor gauge wheel assembly ..........................61
Fig. 5.9. Hydraulic control subsystem components 63
Fig. 5.10. Hydraulic proportional valve 64
Fig. 5.11. Power Supply Module .................................66
Fig. 6.1. Keypad scanning directions .................................73
Fig. 6.2. Actual and ideal signals from a switch, for key debounce routine 73
Fig. 6.3. Fuzzy rules output table .................................79
Fig. 6.4. Hydraulic valve detail .................................85
Fig. 7.1. LCD interface display in operation .............................92
Fig. 7.2. Keypad functions .............................................93
Fig. 7.3. Calibration functions - 90 degree setting and theta correction ..........96
Fig. 8.1. Fuzzy control rules for 7 membership function algorithm.................................104
Fig. 8.2. Manifold Editor input membership function example........................................109
Fig. 8.3. Manifold Editor - output membership function for 7 memb fcn algorithm...........109
Fig. 8.4. FIDE simulation screen example: input and output memb. fcn.s........................110
Fig. 8.5. Input membership functions for change in error from prev. depth reading........111
Fig. 8.6. Output membership functions for depth adjustment........................................111
Fig. 8.7. Fuzzy control surface (from FIDE simulation)...............................................112
Fig. 8.8. Field data logger status display screen.............................................................116
Fig. 8.9. Portion of data output file showing data format for PC logger..........................118
Fig. 8.10. Time domain plot of the sensor positions.........................................................119
Fig. 8.11. Error plot from PC logger, 5 membership function algorithm..........................120
Fig. 8.12. FFT plot from PC logger, front sensor, for field test 5......................................121
Fig. 8.13. FFT plot from PC logger, disc sensor, for field test 5.......................................121
Fig. 8.14. Error plot from the 5 membership function algorithm......................................122
Fig. 8.15. Field trial 5 error from set point, over 1400 samples......................................122
LIST OF EQUATIONS

3.1 General form of the control law ................................................................. 20
3.2 Fuzzy logic controller (FLC) representation ............................................. 20
3.3 Error change .............................................................................................. 21
3.4 Error sum .................................................................................................. 21
3.5 Control change ......................................................................................... 21
3.6 Min operator ............................................................................................ 29
3.7 Max operator ............................................................................................ 29
3.8 Alpha cut output function ......................................................................... 31
3.9 MOM method defuzzification .................................................................... 32

8.1 Min of inputs ......................................................................................... 106
8.2 Membership of the min of the input variables (x,y) ............................... 107
8.3 Max of all fuzzy phrases ......................................................................... 107
**LIST OF ABBREVIATIONS**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>amperes</td>
</tr>
<tr>
<td>AC</td>
<td>alternating current</td>
</tr>
<tr>
<td>A/D</td>
<td>analog to digital converter</td>
</tr>
<tr>
<td>CAL</td>
<td>calibrate</td>
</tr>
<tr>
<td>CNT</td>
<td>control</td>
</tr>
<tr>
<td>COA</td>
<td>center of area defuzzification method</td>
</tr>
<tr>
<td>CONT</td>
<td>control</td>
</tr>
<tr>
<td>COP</td>
<td>microprocessor unit watchdog timer</td>
</tr>
<tr>
<td>cm</td>
<td>centimetre</td>
</tr>
<tr>
<td>CPU</td>
<td>central processing unit</td>
</tr>
<tr>
<td>DC</td>
<td>direct current</td>
</tr>
<tr>
<td>DEG</td>
<td>degrees</td>
</tr>
<tr>
<td>DOS</td>
<td>disk operating system</td>
</tr>
<tr>
<td>DSP</td>
<td>digital signal processing, digital signal processor</td>
</tr>
<tr>
<td>EPROM</td>
<td>erasable, programmable read-only memory</td>
</tr>
<tr>
<td>ERR</td>
<td>error</td>
</tr>
<tr>
<td>FCN</td>
<td>function</td>
</tr>
<tr>
<td>FFT</td>
<td>fast Fourier transform</td>
</tr>
<tr>
<td>FLC</td>
<td>fuzzy logic controller</td>
</tr>
<tr>
<td>GPT</td>
<td>general purpose timer of the MC68HC16</td>
</tr>
<tr>
<td>HYD</td>
<td>hydraulic</td>
</tr>
<tr>
<td>Hz</td>
<td>hertz, cycles per second</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines</td>
</tr>
<tr>
<td>in</td>
<td>inch</td>
</tr>
<tr>
<td>I/O</td>
<td>in/out lines and/or referring to a microcontroller I/O subsystem</td>
</tr>
<tr>
<td>kHz</td>
<td>kilohertz, 1,000 cycles per second</td>
</tr>
<tr>
<td>KHz</td>
<td>kilohertz, binary designation, 1,024 cycles per second</td>
</tr>
<tr>
<td>km</td>
<td>kilometre</td>
</tr>
<tr>
<td>LCD</td>
<td>liquid crystal display</td>
</tr>
</tbody>
</table>
LED  light emitting diode
ma  milliamperes, 1/1,000 of an amp
MAX  maximum function
MC68HC11  Motorola 8-bit microprocessor unit
MC68HC16  Motorola 16-bit microprocessor unit
MEMB  membership
MHz  megahertz, million cycles per second
MIN  minimum function
mm  millimetre
MOM  mean of maxima defuzzification method
MOV  metal oxide varistor
MPU  microprocessor unit
MS  Microsoft
OM  organic matter
PC  IBM-compatible personal computer
PD  proportional-differential controller
PI  proportional-integral controller
PID  proportional-integral-differential controller
PWM  pulse-width modulation
QSM  queued serial module of the MC68HC16
RAM  random access memory
ROM  read-only memory
RS232  serial interface port, adhering to RS232 standard
SIM  system integration module of the MC68HC16 MPU
SRAM  standby, battery-backed RAM in the MC68HC16 MPU
STD DEV  standard deviation
SYS  system
V  volts
VAR  variance
VLSI  very large scale integration
CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

The depth of seeding of field crops is a critical crop production factor. Of all crop management parameters that must be controlled, this is one of the few that requires no additional cash outlay, yet can have a profound effect on the final yield and grade of a crop if not properly controlled.

Each type of seeding implement has unique characteristics that affect how well it maintains accurate depth while seeding. Among the types which pose the greatest control accuracy challenge is the one-way discer seeder. This type of implement has been replaced by other types on many Western Canadian farms for various reasons, not the least of which is the difficulty in maintaining consistent seeding rate and depth under conditions of varying terrain and soil conditions. However, one-way discer seeders are still predominant in many parts of the Red River Valley of Manitoba, as it has the advantages of both low capital cost combined with high efficiency in penetrating the characteristic heavy clay-based soil textures predominant in the region. Seeding with discer seeders can often commence without the need of spring pretillage of the field. By eliminating this spring pretillage, fuel costs are lower, while the threat of soil erosion is also reduced. Additional advantages are a lower labour requirement due to fewer tillage passes over the field, thus resulting in food producers completing seeding earlier in our short growing season. This, in turn, reduces the risk of late season frost damage to the crop.
Fig. 1.1. Typical discer seeder.

With the problems discer seeders have in maintaining consistent seeding depth, agricultural crops may suffer in both yield and quality when discer seeders are used and due care is not taken to minimize excessive errors in seeding depth. These negative effects of improper seeding depth have been observed by many researchers. For instance, in manually controlled discer seeders, Dyck, at the University of Saskatchewan, found that depth variations of plus or minus 1.27 to 4.45 cm (0.5 to 1.75 inches) were not uncommon [Dyck75]. This faulty depth regulation may also lead to significant increases in power requirement for heavy soil, particularly for large equipment [GKWD81b].

1.2 Thesis Objectives

The objective of this thesis is to present a control system that combines the speed and precision of a digital system with the control reasoning of a human operator by the use of fuzzy logic control. The seeding depth controller described in this thesis combines the
human control reasoning of fuzzy logic with a traditional electrohydraulic seeding depth controller to provide optimal depth control with minimal input from the operator.

1.3 Overview of Thesis Chapter Contributions

The order in which thesis topics are presented reflect the research approach. The first three chapters provide background information. These include a definition of the problem of maintaining the correct seed depth using conventional technology, a presentation of the basic concepts of fuzzy logic control and why this approach was chosen in this study. Thus, the problems to be addressed by this research are defined. These chapters are then followed by three chapters which present the system organization, hardware details, and software design. Finally, the last two chapters present the results and recommendations from this research. The appendices contain information related to the execution of the hardware and software development, including schematics, specifications, and source code.

Chapter 2

The problem of maintaining correct seeding depth and current methods used to control seed depth are outlined in this chapter. The approach taken is from the viewpoint of agricultural researchers and producers, using terms commonly used in agricultural research and on prairie cereal farms. This allows for an intuitive definition of the problem that is consistent with the reason fuzzy control is being investigated for this application: no accurate mathematical model of a conventional seed depth controller is found in the literature that can define how such a system should respond to a wide variety of field conditions.
Chapter 3

The theory of fuzzy logic controllers is presented in Chapter 3 of this thesis. This will be compared to the basic theories of conventional control law, and how this relates to some of the problems encountered in conventional seeding depth control methods. The reader is stepped through the process of defining a control problem, generating rules, and how the fuzzy inference process works. A simple example is used to illustrate the major points. The final portion of this chapter also provides background information on the topic of some methods used in the design of fuzzy logic controllers.

Chapter 4

Once the problems with using conventional control law in designing seeding depth controllers has been presented, the system definition section will explain how the theories of fuzzy control can be applied to solve the problem. By defining the problem and the approach to a solution, the required hardware and software specifications are developed. Various choices in hardware and types of software algorithms will be explored. Finally, the chosen design will be outlined, with an explanation of why this route was chosen.

Chapter 5

A more detailed block diagram will be presented in the Detailed Hardware Design of the Fuzzy Seeding Depth Controller, with an explanation of the system architecture and organization. The implementation will also be described, with special focus on how the controller is operated by the user.
Chapter 1: Introduction

Chapter 6
In this chapter, the objectives and functions of the software design are presented. This section will also outline how this relates to the rest of the system architecture. The software organization and implementation will also be explained here.

Chapter 7
The integration of the hardware and software to produce a working fuzzy seeding depth control system is explained in the System Implementation section. This includes the process of putting the different parts of the system together and verifying that the system performs to specifications.

Chapter 8
In the Discussion and Analysis of the Fuzzy Seeding Depth Controller section, theoretical and field experimental results are presented. The design of the experiments is explained, with emphasis on the test methods and what parameters were being tested. The final results will also be discussed, with an explanation of why the best results meet the test criteria.

Chapter 9
Finally, the Conclusions and Recommendations summarizes the proven points and discoveries, as well as contributions to science and technology. The direction of future work in this area will also be presented based on what knowledge was gained from this research.

The approach of this research is to combine the theoretical aspects of fuzzy logic control with the practical aspects of controller implementation and testing. The first step
of this process, the definition of the problem of accurate seeding depth control, will be defined in Chapter 2.
Chapter 2: Control of Seeding Depth

CHAPTER 2
CONTROL OF SEEDING DEPTH

In this Chapter, the problem of maintaining consistent and accurate seeding depth control is presented. The crop emergence and crop health consequences of poor seeding depth control are presented in Section 2.1. In Section 2.2, the most common methods of seeding depth control with discer seeding implements are described and compared.

2.1 Effects of Seeding Depth on Crop Emergence

Lack of adequate depth control may lead to variations in the depth of seeding. The negative effects of improper seeding depth have been observed by many researchers. For instance, in manually controlled discer seeders, Dyck, at the University of Saskatchewan, found that depth variations of plus or minus 1.27 to 4.45 cm (0.5 to 1.75 inches) were not uncommon [Dyck75]. This faulty depth regulation may lead to significant increases in power requirement in heavy soils, particularly with large equipment [GKWD81b], [BBKR83].

Crop emergence can be significantly reduced by small variations in depth. This fundamental problem has been studied for many years, and the effects are quite well established in the literature [MuAr39], [Sund64]. In some crops, mathematical models have even been developed that predict emergence based on seeding depth [HoCl74]. The effect can be quite severe, given the nonlinearity of crop emergence functions.
Fig. 2.1. Crop emergence as a function of seeding depth. (After [MoGe85a], [KGFT93]).

Using soybeans, with a peak mean of 3 cm (1.17 inches), a variation of only 1 cm (0.39 inches) from this peak results in a reduction in emergence of 5%. In comparison, should seeding depth be erroneously set to 4 cm (1.56 inches), a variation of only 1 cm (0.39 inches) in depth above and below this set point will result in reduced emergence by 5% and 30%, respectively, as shown in Fig. 1 [MoGe85a]. Therefore, the benefits of precise seed depth control are significant.

Results such as those presented in Fig. 1 are generally true for crops in which the main independent variable, that is to say, the only major limiting factor, is the depth of seeding. Other, less optimal situations typically arise on prairie fields that can further magnify the effects of improper seeding depth. Such extra limitations are characterized by
adding additional stress on the emerging crop, in addition to the stress induced by improper seeding depth.

Stress factors due to the effects of nature include conditions that may slow crop emergence, such as extreme environmental conditions. Soils which are extremely hot, cold, wet or dry all produce suboptimal conditions for growth. Other natural factors that can have a magnifying effect on poor depth control include intense weed competition, crusted soil surface, and the presence of soilborne diseases and insect pests [HaDP77].

Man-made stress factors have also been documented to magnify the negative effects of poor seed depth control. These include the presence of excessive herbicide residues such as triallate and trifluralin, especially in soils low in organic matter (under 2.5%) [WSSA83]. This problem can be very severe in areas such as central Saskatchewan, where rolling soil with low organic matter eroded knolls are often seeded deeper than surrounding areas. This deeper seeding is due to depth control system of the seeder being unable to adequately respond to the sudden change in terrain.

2.2 Types and Accuracy of Seeding Implements

There are many different types of implements used in western Canada for seeding cereal and oilseed crops. In terms of the types of tillage or soil disturbance methods most commonly by these implement types in both cereals and oilseed crops, three are in most common use today: the double disc press drill, the airflow or airseeder cultivator system, and one-way discer seeders.
Chapter 2: Control of Seeding Depth

The double disc press drill is the most accurate seeding implement used on the prairies, with two flat discs penetrating the soil at an angle of only a few degrees from the direction of travel.

![Diagram of double disc seed drill opener detail]

**Fig. 2.2.** Double disc seed drill opener detail.

The discs open the soil in such a way that a furrow is produced at the desired depth. Seed is then dropped into the furrow, and soil is placed over the seed. Soil covers the seed by two processes: soil falling back into the furrow once the discs have passed by, and by the compression action of a flat wheel or tire, about twice the width of the furrow, pushing soil down over the seed and packing it. Depth control with these implements is generally by means of mechanical control, by a stop bolted to the ram shaft of the hydraulic disc lift cylinder. Most double disc press drills also have springs which act upon the double disc soil penetrating assemblies, thus keeping fairly consistent depths even in undulating terrain. The packing discs are mounted rigidly to the seeder frame, and provide a large surface area for the seeder to ride on (as wide as the entire implement). Although they are very accurate, these systems are well suited only for loose soils that have been
preworked by cultivating and harrowing or on very light textured soils. Some zero tillage seeders use this system, but often need considerable weight and tractor horsepower to penetrate the soil to an adequate depth without field preworking. Zero tillage double disc press drills are excellent for soil and moisture conservation, and in fact have been designed to preclude the need for any pretillage prior to seeding, thus eliminating virtually all soil erosion. Unfortunately, they also tend to require a relatively high labour, draft, and maintenance requirement per metre of width than do other systems.

Fig. 2.3. Airseeder system with high capacity seed hopper, air blower and cultivator.

The airseeder or airflow system is increasing in popularity in western Canada, due to its lower labour requirement, maintenance, and speed of seeding. These implements use a very low lift cultivator shovel as the means of soil penetration. Depth control with these systems is a challenge, with many companies using different configurations of depth guide wheels, dual tires for floatation over soft soil, electric over hydraulic conventional control systems, and various mechanical means [GKWD81a]. Each rigid cultivator shank may
have one or two springs holding it in place. The design is a compromise between accuracy of seed placement and seedbed quality to time savings and low labour requirement.

By virtue of it’s design, the discer seeder does not use the kind of precise seed placement techniques used by other systems. One of the major drawbacks of using one-way discer seeders is the lack of precise seeding depth control. In fact, depending on how well the seed outlet spouts are placed in relation to the discs, the seed could be placed anywhere from a narrow vertical band less than 1 cm wide, to as much as 4 or 5 cm in a nonuniform distribution. Seeding rate also varies as the discs gang swings from side to side over different soil conditions, thus widening or narrowing the area seeded per disc unit.

![Fig. 2.4. One way discer seeder detail.](image)

The discs can be described as the concave circular steel tools with a sharpened edge making up the working part of a discer seeder [Meri93]. Discer seeders use gangs of single concave discs that penetrate the soil at an angle a few degrees from the direction of travel. This angle opens the concave side of the disc to the soil, where the edge of the disc
cuts into the soil. The larger the angle of the disc to the soil, the larger the cross-section of soil the disc is exposed to in the direction of travel. As a result, as the angle of the disc to the soil increases, so too does the draft, increasing the power requirement from the tractor to maintain the same speed. Likewise, with the seed filling the hoppers above the discs, this added weight can compress the tires holding up the discer unit. This compression acts to push the discer deeper into the soil, adding to the seeding depth. Likewise, with more depth, draft increases, which adds a traverse force on the discer against the wheels, causing the discer to swing perpendicular to the direction of travel, thus widening the width of cut across the whole discer. Now, with a seed depth that is deeper than that set by the operator, there is also an decrease in seeding rate proportional to the decreased width covered by the discer in each pass at this depth. An expert operator is able to weigh the different factors that affect these problems when choosing the best compromises for depth setting, including such parameters as wheel positioning, speed, and tire pressure.

2.3 Advantages of using discer seeders in the Red River Valley

Agricultural producers choose a seeding method based on a number of criteria. These include local soil texture(s), local experience, types of crops grown, costs, pre-seeding preparation methods and labour requirements. In the Red River Valley, producers on heavy soils such as Osborne Clay have two major problems: incorporating straw left on the soil surface from the previous year's harvest, and penetrating the heavy soil deep enough while seeding without producing too many soil lumps. One-way discer seeders are well suited for these conditions. As an additional benefit of using discer seeders, many producers find that they can seed straight into the soil in the spring without pretillage
preparation. This eliminates the time, expense, soil moisture loss and increased risk of soil erosion associated with pre-seeding tillage. Seeding with these implements has been popular for many years in this region, and is still common, although other seeding implements are being improved and are slowly being adopted over discer seeding.

2.4 Methods of Depth Control

The oldest depth control method that many farmers still rely on is their own visual estimate of the amount of a seeding implement buried below the soil surface and their "feel" for the sound and vibration of increased engine load when the seeder digs into the ground. Many other schemes have been introduced to improve on this method [Froe92].

With the use of a computer controlled electromechanical depth monitor for the hydraulic depth control mechanism of a seeder, average seeding depths varied within a 2 cm range [Dyck75].

Fig. 2.5. Discer seeder, (typical of the types used in the Red River Valley of Manitoba), used in this research.
Fig. 2.6. Illustration of the distance from the cab to the disk gangs.

Depth control and monitoring devices are based on mechanical or electrohydraulic control. These methods do not always provide consistent accuracy of seeding depth over varying terrain conditions, especially on one-way discer seeders. Common methods for controlling the depth of discer seed implements in uneven terrain are tiring for the operator, and are inaccurate, leading to increased power requirements and reduced crop yields. The seeder, which is used for seeding and on occasion tillage, is raised and lowered by hydraulic cylinders which are actuated by manual or electrical control of hydraulic flow valves. As fuel and machinery costs rise, efforts to maximize efficiency lead to the use of larger implements such as 58-foot wide cultivators/air seeders and the practice of hitching several discers [BBKR83]. These arrangements severely hamper the
effectiveness of even an expert human operator who must estimate the depth of the rear furrow as far as 20 m away. Coupled with this, the operators’ vision is often obscured by dust, and they must mentally average out the depth for varying soil conditions. Errors in the estimates increase with large variances in soil moisture, topography, soil texture, and draft load as the seed empties out of the hoppers. In discer seed implements, this is further compounded by the fact that the width of the area seeded also changes with large variations in draft loads.

Fuzzy control offers a paradigm in which control rules based on a human operator’s experience are used to produce control algorithms that have been effective in controlling nonlinear, dynamic plants that proved to be extremely difficult or too complex to be modelled mathematically for creating effective conventional controllers [Mamd93]. The problems encountered with one-way discer depth control, with its combinations of varying draft loads, soil types, soil conditions and topography certainly make this seeding problem a good candidate for determining the results of applying a fuzzy logic control algorithm [YOTN87].

2.5 Chapter Overview

The importance of accurate seeding depth and its effect upon crop vigour and yield provide the main motivation for the investigations in this research. Characteristics and limitations inherent in different seeding methods and depth controller types were given. The prevalence of one-way discer seeders and their advantages and disadvantages for use in the Red River Valley of Manitoba was also explained.
To date, the prevalent types of seeding depth control methods have included visual judging of depth with mechanical control actuation, mechanical “stops” on hydraulic cylinders, and conventional electrohydraulic controllers. Due to the nonlinear nature of depth control under varying soil and topographic conditions, problems have been experienced with these control methods. A more suitable control regime was investigated, with an eye to providing more consistent control by accurate sensing of the seeder depth, in combination with the ability to vary hydraulic control signal gain based on the depth position at the current instant in time and the position in the past. Fuzzy logic control is well suited for this type of nonlinear application, and is evaluated in this research.
Chapter 3: Fuzzy Logic Control

CHAPTER 3

FUZZY LOGIC CONTROL

Fuzzy logic control is well suited to control problems where time-varying, nonlinear inputs characterize the system [KiMa77]. Examples include a wide variety of industrial and domestic uses, from controlling the braking of high-speed commuter trains in Japan, to the control of operating cycles of washing machines [Will94].

The concept of fuzzy logic, and specifically, fuzzy set theory, can be directly attributed to Lotfi A. Zadeh, of the University of California at Berkeley. In the first paper published on this topic, Zadeh described fuzzy sets and the underlying theory behind them [Zade65].

In this Chapter, a description of fuzzy logic by comparing it to what can be called "conventional control law" in Section 3.1. In 3.2, there is a step by step explanation and example of how one would formulate a fuzzy control algorithm. Once the basic principles of fuzzy logic control have been presented, Section 3.3 deals with the issue of how to represent and model a fuzzy logic controller.

3.1 Fuzzy vs. Conventional Control

One common feature of conventional control law is that it is based on algorithms that can be described precisely by equations. They generally require a formalized analytical description of the controlled system in the form of a mathematical model in order to synthesize the algorithm [Mamd93]. Fuzzy control algorithms, on the other hand,
are constructed using logical rules. Each rule defines what the output of the controller must be, based on the specific value(s) of input state(s). Rather than being derived purely as the result of creating a mathematical model, the rules are often formulated under the direction of an expert human operator. Linguistic control algorithms are used, which are constructed from a rule base that in turn is formulated from human experience and intuition. The theoretical and practical aspects of the dynamics of the controlled system may also be used [Zade73].

Under conventional control law, algorithms (such as proportional, derivative, integral, PD, PID) can all be derived from the general form of the control law for different functions of various orders [Dorf89]

\[ \mu (k) = f(e(k), e(k-1), \ldots, e(k-v), u(k-1), u(k-2), \ldots, u(k-v)) \] (3.1)

where

- \( f \) is, in general, a nonlinear function,
- \( k \) is the instance or sample number,
- \( e \) is the error between the setpoint and the output of the controller,
- \( v \) defines the order of the controller, and
- \( u \) is the change in output.

The fuzzy logic controller (FLC), on the other hand, can often be represented in a similar form

\[ \mu (k) = F(e(k), e(k-1), \ldots, e(k-v), u(k-1), u(k-2), \ldots, u(k-v)) \] (3.2)
Chapter 3: Fuzzy Logic Control

where \( F \) is the control law described by a rule base. It should be noted, however, that an FLC is not a type of difference equation or transfer function as in conventional control logic. In fact, the representation above is only limited to situations where it is not difficult to describe it verbally. This is especially true when it is considered that a FLC uses a knowledge base, so past higher-order values of both the control action \( u \) and the error \( e \) are not always needed in order to complete the fuzzy algorithm. Since the FLC is based on rules, it is possible to incorporate combinations of the error \( e \) and control \( u \) that are relevant to the real-world operation of the plant in ways not possible with conventional control law [YaFi94]. For example, the various operations can be easily incorporated into an FLC's control rules, including

\[
\Delta e (k) = e(k) - e(k-1) \tag{3.3}
\]

error change

\[
\sum_{l=0}^{k} e(l) = e(k-1) \tag{3.4}
\]

error sum

\[
\Delta u (k) = u(k) - u(k-1) \tag{3.5}
\]

control change

In conventional control law, one of the main paradigms is the concept of analyticity. In the FLC, the main paradigm is the concept of a knowledge-based algorithm operating as a kind of expert knowledge-based system, containing the control algorithm in a simple rule base.
3.2 Formulating a Fuzzy Logic Control Algorithm

In constructing a fuzzy logic controller, it is necessary to fully understand the plant to be controlled. This knowledge can originate as the control actions of an expert human operator for each set of input variables. It can also originate as a set of data containing observations of a properly operating system. In a few circumstances, the rule base can be derived from a conventional control transfer function [YaFi94].

Through the next several sections of this Chapter, an example fuzzy controller with nine rules, two input variables and one output variable will be used to illustrate the concept being explained.

3.2.1 Fuzzy Control Rules

Once the system to be controlled is understood, the next step is to construct the fuzzy control rules. The control rules constitute the fuzzy control algorithm. These rules are constructed using logical (Boolean) "if-then" rules. The rules are based on the implementation of human understanding and thinking of the plant to be controlled.

Rules may appear to be simple intuitive statements based on an observation or a base of experience, but in reality they have a very specific syntax. The following is an example of the form they would follow.

\[
\text{IF antecedent 1 AND antecedent 2 THEN consequent 1 AND consequent 2}
\]

\[
\text{ALSO}
\]

where

- The "IF" portion of the rule contains the antecedents. This part of the rule
Chapter 3: Fuzzy Logic Control

contains the conditions that affect the plant.

- AND is one of the allowable fuzzy logic Boolean operators, (such as AND, OR, NOT),

- the antecedents are in the form of an input variable equal to some value. For example, “temperature is HOT”, where temperature is the input variable, and hot is one of the membership function labels associated with temperature.

- the “THEN” portion of the rule contains the consequent. This gives the output or conclusions, should the conditions in the antecedent part of the rule be satisfied. If the output of a consequent applies, it is said that the rule “fires”.

- the consequents are in the form of an output variable equal to some value. For example, “fan speed is HIGH”, where fan speed is the output variable, and high is one of the membership function labels associated with temperature.

- ALSO is the equivalent of the Boolean operator OR.

The terms, or linguistic labels qualitatively describe the input and output in each rule. Descriptors such as Positive, Negative and approximately zero are examples of linguistic labels.

In the application described in this thesis, the FLC describes the relationship between error $e(k)$, change in error $\Delta e(k)$ [TeKi93].

3.2.2 Membership Functions

Each variable in a rule constitutes the measured observation or output control action, such as the error, change in error, and output. The linguistic labels within the rules each constitute a fuzzy set, or a reference fuzzy set. When a number of reference fuzzy sets
are grouped for a specific variable, this is a term set. The variable being measured or calculated exists within a finite range of values, or the universe of discourse. In summary, this means that for one variable, for example, error, $e(k)$, the term set contains a number of reference fuzzy sets (negative, approximately zero, and positive), all defined over the same universe of discourse. Each reference fuzzy set represents the associated linguistic variable.

![Membership Functions](image)

Fig. 3.1. Membership functions, each a reference fuzzy set of the term set for error, $e(k)$.

Generally, the scaling of variables and reference fuzzy sets within a universe of discourse ranges over the input variable values. The shape of membership functions can vary; triangular membership functions are quite easy to code in software, but other shapes can include Gaussian, rectangular (non-fuzzy or crisp control), and variations of different nonlinear functions and curves.

3.2.3 Fuzzification

The process of converting crisp input values of the FLC variable(s) into a level of membership in each membership function is called fuzzification. As each input variable is
evaluated, membership values in each reference fuzzy set (for each membership function) are determined.

![Membership Functions of Error and Change in Error](image)

**Fig. 3.2.** Fuzzification of the crisp input $X_0$ into the membership functions of error, $e(k)$.

**Fig. 3.3.** Fuzzification of the crisp input $X_1$ into the membership functions of change in error, $\Delta e(k)$.
The degree of membership of an input in a membership function is determined by comparing the value of the input to the values defined for each membership function. The result, therefore, is the level of membership that the input value has in each membership function, described as a number between 0 and 1. For example, the crisp input for error may be a value of -0.1 units. In Fig. 3.2, this value of -0.1 units gives a degree of membership in negative of 0.25, and a degree of membership of 0.75 in approximately zero.

3.2.4 Fuzzy Inference

The degree of membership in each fuzzy set is then determined for all other input variables. Once complete, the membership values for each membership function are substituted into the rules governing the process of the fuzzy controller. This process, called rule evaluation, or fuzzy inference, applies the fuzzy input values from the fuzzification process to the rules that govern the controller. Each rule is then evaluated to calculate the “level of firing” for each rule.

Here is an example of a simple fuzzy rule base with 9 rules:

1. **IF** the error is positive **AND** the change in error is positive **THEN** the change of the control is negative;

   **ALSO**

2. **IF** the error is positive **AND** the change in error is approximately zero **THEN** the change of the control is negative;

   **ALSO**

3. **IF** the error is positive **AND** the change in error is negative **THEN** the change
of the control is approximately zero;

ALSO

4. **IF** the error is approximately zero **AND** the change in error is positive **THEN**
   the change of the control is negative;

ALSO

5. **IF** the error is approximately zero **AND** the change in error is approximately zero **THEN** the change of the control is zero;

ALSO

6. **IF** the error is approximately zero **AND** the change in error is negative **THEN**
   the change of the control is positive;

ALSO

7. **IF** the error is negative **AND** the change in error is positive **THEN** the change of the control is approximately zero;

ALSO

8. **IF** the error is negative **AND** the change in error is approximately zero **THEN**
   the change of the control is positive;

ALSO

9. **IF** the error is negative **AND** the change in error is negative **THEN** the change of the control is positive.

The complete rule base that describes the control actions of the FLC in this example under a range of states can be summarized in the following chart:
Chapter 3: Fuzzy Logic Control

Fig. 3.4. Fuzzy control rule summary.

This intuitive description of control action under a range of conditions can serve as a very accurate summary of the actions of the system without a detailed mathematical model.

The value of each antecedent of each rule is determined, and substituted into each rule. The rule strength or degree of truth of each rule is then determined by evaluating the antecedents. In this case, the Boolean operator linking the antecedents is AND. This means that the rule takes on the lowest value of each antecedent for the rule. Using the antecedent notations given in Fig. 3.4, we substitute the degrees of membership in error \( e(k) \) and change in error \( \Delta e(k) \) into the rules of the sample rule base to obtain the degree of membership in the respective output membership functions, \( \Delta u(k) \).

1. \textbf{IF} \( e(k) \) is \( P \) (0.0) \textbf{AND} \( \Delta e(k) \) is \( P \) (0.6) \textbf{THEN} \( \Delta u(k) \) is \( N \) (0.0) \textbf{ALSO}
2. \textbf{IF} \( e(k) \) is \( P \) (0.0) \textbf{AND} \( \Delta e(k) \) is \( Z \) (0.3) \textbf{THEN} \( \Delta u(k) \) is \( N \) (0.0) \textbf{ALSO}
3. \textbf{IF} \( e(k) \) is \( P \) (0.0) \textbf{AND} \( \Delta e(k) \) is \( N \) (0.0) \textbf{THEN} \( \Delta u(k) \) is \( Z \) (0.0) \textbf{ALSO}
4. IF $e(k)$ is $Z(0.75)$ AND $\Delta e(k)$ is $P(0.6)$ THEN $\Delta u(k)$ is $N(0.6)$ ALSO 

5. IF $e(k)$ is $Z(0.75)$ AND $\Delta e(k)$ is $Z(0.3)$ THEN $\Delta u(k)$ is $Z(0.3)$ ALSO 

6. IF $e(k)$ is $Z(0.75)$ AND $\Delta e(k)$ is $N(0.0)$ THEN $\Delta u(k)$ is $P(0.0)$ ALSO 

7. IF $e(k)$ is $N(0.25)$ AND $D_e(k)$ is $P(0.6)$ THEN $\Delta u(k)$ is $Z(0.25)$ALSO 

8. IF $e(k)$ is $N(0.25)$ AND $\Delta e(k)$ is $Z(0.3)$ THEN $\Delta u(k)$ is $P(0.25)$ALSO 

9. IF $e(k)$ is $N(0.25)$ AND $\Delta e(k)$ is $N(0.0)$ THEN $\Delta u(k)$ is $P(0.0)$ 

We now have assigned degrees of truth or *degree of firing* (DOF) to each rule, based on the values of the antecedent input values. Each of these values were determined using an *AND*-type (or intersection) conjunctive connection, the $\text{Min} \land$ operator, between the two inputs. This can be expressed in the following form:
\[
\tau_i = B_{i1}(x_1) \land B_{i2}(x_2) \quad (3.6)
\]

Where $\tau_i$ is the DOF for the $i^{th}$ rule with respect to the input values within the universes of discourse ($U_n$), $U_1 = x_1$ and $U_2 = x_2$. $B_{i1}$ and $B_{i2}$ denote the membership functions of antecedent 1 and 2 for rule $i$.

Now that we have the rule strengths for each rule, we can proceed with the aggregation of these values using the *OR*-type (or union) disjunctive connective *ALSO*. This operation is conducted using the $\text{Max} \lor$ operator

\[
F(y) = \lor F_i(y) = \lor (\tau_i \land D_i(y)) \quad (3.7)
\]
In our example FLC rules, this equation would be evaluated as:

1. ...change of the control is negative (value of rule = $F(1) = 0.0$) ALSO

2. ...change of the control is negative (value of rule = $F(2) = 0.0$) ALSO

3. ...change of the control is zero (value of rule = $F(3) = 0.0$) ALSO

4. ...change of the control is negative (value of rule = $F(4) = 0.6$) ALSO

5. ...the control is zero (value of rule = $F(5) = 0.3$) ALSO

6. ...the control is positive (value of rule = $F(6) = 0.0$) ALSO

7. ...change of the control is zero (value of rule = $F(4) = 0.25$) ALSO

8. ...the control is positive (value of rule = $F(5) = 0.25$) ALSO

9. ...the control is positive (value of rule = $F(6) = 0.0$)

The result of the fuzzy ORing operation yields degrees of memberships of the consequents in each membership function of the change in output $\Delta u(k)$:

- negative change in control = 0.0
- zero change in control = 0.3
- positive change in control = 0.25

These values now can be applied to the output fuzzy sets, in the process of defuzzification.
3.2.5 Defuzzification

Now that we have the values of the consequents, we move on to the final step, \textit{defuzzification}. In this process, a single crisp output value is calculated by any one of a number of defuzzification techniques. Three common defuzzification methods are the Center of Area (COA, or centroid method), the Singleton method, and the Mean of Maxima (MOM).

In the centroid method, each output membership function is truncated above the indicated level of its degree of membership. This is called an \textit{alpha cut}. All the alpha cuts for each membership functions of the term set \textit{output} are determined, with the center of area calculated as a fuzzy centroid. Here, we consider the finite universe of discourse $Y$ and the discrete membership functions $F(y)$:

$$
 y_{out} = \frac{\sum_{j=1}^{n} F(y_j) y_j}{\sum_{j=1}^{n} F(y_j)}
$$

This results in the output being the result of a weighted average taken as the center of mass of an area comprising the combined areas under the alpha cuts. In the example given above for the nine membership functions, the output is represented graphically Fig. 3.5.
for output $\Delta u(k)$: membership in Negative = 0.0

membership in Approximately Zero = 0.3

membership in Positive = 0.25

Fig. 3.5. Alpha cuts for the output $\Delta u(k)$ when $N = 0.0$, $Z = 0.3$, $P = 0.25$.

In the MOM method, the defuzzified value is determined as the mean of all values of the universe of discourse having the maximum membership values:

$$ y_{out} = \frac{1}{q} \sum_{j \in J} y_j $$

(3.9)
Where $J$ is the set of all elements of the universe of discourse of $Y$ that have the maximum value of $F(y)$. The element $q$ is the cardinality of $J$.

The singleton defuzzification method is commonly used on microcontrollers, where speed of calculation is often a limiting factor. Generally, singleton defuzzification results are based on a lookup table for values within a given range, rather than using extra clock cycles to compute a summation. This does not result in values that are as precise or finely-tuned as the previous two methods. It is important, therefore, to ensure the effectiveness of the plant to be controlled is not compromised by the reduced granularity of this method.

\[
\text{for output } \Delta u(k): \quad \text{membership in } \text{Negative} = 0.0 \\
\quad \text{membership in } \text{Approximately Zero} = 0.3 \\
\quad \text{membership in } \text{Positive} = 0.25
\]

\[
\text{Negative} \quad \text{Approximately Zero} \quad \text{Positive} \\
\begin{array}{c}
\text{Membership} \\
0.25 \\
0.5 \\
1.0
\end{array}
\]

These areas not used in calculation - they do not constitute local maxima

Fig. 3.6. The resulting mean of maxima (MOM) output for change in output, $\Delta e(k)$.

In the singleton method, the rule consequent values are plotted on the Universe of Discourse of the output range. The average value between the points is then calculated. This average is the resulting output value.
3.2.6 Summary of the Decision Process

The process of calculating the crisp output of a FLC can be summarized as follows:

1. Determine the degree of firing (DOF) of the rules. This is determined by taking the minimum (AND) of the antecedents. The value of antecedents were determined by the level of matching the input values with the respective fuzzy set for the rule.

2. In the next step, fuzzification (or fuzzy implication), the degree of membership in each consequent fuzzy set is determined, according to the respective rules' DOF.

3. Rule aggregation takes place when the maximum (OR) function is performed on the consequent membership functions across all of the rules. This produces the final output from the rule base.

4. Finally, in defuzzification, a discrete output value is determined by calculating the fuzzified centroid of the output fuzzy set.

3.3 Representation and Analysis of a Fuzzy Logic Controller

As the fuzzy logic process often involves two input variables and one or more output variables, FLC's can be represented geometrically. The antecedents of each rule can be mapped over a 2 dimensional area, with one antecedent on each side. In Fig. 3.7, the universe of discourse for each antecedent is represented by two dimensions \(X_1\) and \(X_2\), (the fuzzy input space) while the membership functions are represented by \(B_{i1}\) and \(B_{i2}\). 
Chapter 3: Fuzzy Logic Control

The inputs from the universes of discourse \((U_1, U_2)\) are mapped to what is essentially a state space for the input variables (antecedents) in the dimensions \(X_1\) and \(X_2\). Each overlapping area \(A\) and \(B\) indicate rules of the form

\[
\text{IF } U_1 \text{ (for example, error) is } B_{11} \text{ AND } U_2 \text{ (for example, change in error) is } B_{12} \\
\text{THEN } V \text{ (for example, change of the control) is } D_i
\]

Thus, the output membership function is determined by the consequent. The output, (for example, the control action) \(D_i\), is within the universe of discourse \(V\). It is shown in Fig. 3.7 as the \(Y\) axis. The DOF of this output is a result of the fuzzy inference process. Therefore, this representation is similar to a control surface with the overlapping regions of the control rules analogous to areas of gradual slope on the surface (Fig. 3.8).

![Geometric representation of fuzzy sets A and B (from YaFi94).](image)

Fig. 3.7. Geometric representation of fuzzy sets A and B (from YaFi94).

It is important at this point to understand the basic operation of the fuzzy controller. The relationship to conventional control can be made by describing the actions
of the fuzzy algorithm. For a Proportional-Integral fuzzy controller, the change in control is related to the error and its change.

Fig. 3.8. A fuzzy control output surface.

It should be noted that the degree of overlap of membership functions within the state space, for example, regions A and B in Fig. 3.7, are important in the measurement of the fuzziness of the FLC. Should these regions not overlap at all, the result would be a disjoint collection of rules which operate as a variable gain relay. Only one rule could fire at a time.

3.4 Design of the Fuzzy Logic Controller

Once it has been determined that a control system utilizing fuzzy logic will be constructed, there are several approaches that can be taken to do this. The traditional method is to put together a rule base in a similar way that an expert system would be built. An expert operator is asked to describe their actions when the system is in various states. As this may not produce optimal control actions, the operation and characteristics of the plant to be controlled are further studied, much in the same way as for conventional
control. The rules are then formulated and the first membership functions are put in place. The controller results are then found by simulation or testing. Suitable changes are made to the membership functions and/or rules until the results are satisfactory.

Another approach to the formulation of a fuzzy controller is the template method [MacV77]. This method uses a standard rule base that is modified to fit the particular controller. It is a more formal approach to producing fuzzy control rules than the traditional method described above.

The basis of the template method is that rules are produced based on three metarules:

1. If ERROR $e(k)$ and CHANGE IN ERROR $\Delta e(k)$ are 0, the CHANGE IN CONTROL $\Delta u(k)$ is 0.

2. If ERROR $e(k)$ is approaching 0 at a satisfactory rate, the CHANGE IN CONTROL $\Delta u(k)$ is 0.

3. If ERROR $e(k)$ is not approaching 0 at a satisfactory rate, the CHANGE IN CONTROL $\Delta u(k)$ is in nonzero, with the magnitude and direction depending on the direction and the amount of ERROR $e(k)$ and CHANGE IN CONTROL $\Delta u(k)$.

These metarules are then expanded to cover the input states and the corresponding expected outputs of the system to be controlled.

With having defined the rules and membership functions of a fuzzy logic control algorithm, the next step is to implement this into a workable controller. The example we have been describing as the example in this Section is similar in function to a conventional
PI type controller. As such, the block diagram describing the hardware of a Mamdani-type FLC is similar to that of the block diagram of a conventional PI controller.

Fig. 3.8 shows a basic block diagram of a Mamdani-type FLC. The input error signal, \( e(k) \), a direct input to the FLC, is also stored in memory for the next sampling instance, and compared to the previous sampling instance to determine the change in error. The defuzzified output is also compared to the previous output value, which provides a feedback mechanism to prevent control oscillation when output varies around the set point.

![Block Diagram of Fuzzy Logic Controller](image)

- \( e(k) \) - error in instance (k)
- \( \Delta e(k) \) - change in error between instance (k) and (k - 1)
- \( u(k) \) - output in instance (k)
- \( \Delta u(k) \) - change in output between instance (k) and (k-1)

Fig. 3.9. The Fuzzy Logic Controller (FLC) in the framework of a control system (from YaFi94).

Appropriate sensory and control devices are chosen to meet the original criteria for the fuzzy controller inputs and outputs. Appropriate controller computer design
specifications are also worked out, given the fuzzy code to be implemented, the speed of actuation needed, and other considerations in the design such as cost, power availability, size requirements, and operating environment. Once the controller has been built, the control actions are tested against the expected results from the theoretical and simulated predictions. Appropriate changes in fuzzy control algorithm and hardware are then made if necessary.

3.5 Chapter Overview

A description of the theory of fuzzy logic control relevant to the research in this thesis was presented here. Of course, there is a growing body of research found in the literature about fuzzy set theory and fuzzy arithmetic methods as they pertain to fuzzy controllers, so there are other interesting approaches that can be taken in any further research, as explained in Chapter 8.

In formulating a fuzzy controller, expert operator advice is used to produce the rules and definition of the limits of fuzzy sets that govern its operation. The number of membership functions depends on how smooth the controller output must be for proper operation. In implementing the resulting controller, extra hardware-dependent considerations will also affect the number and shape of rules. An example of this is the degree of accuracy of A/D converters (for example, 8-bit vs. 16-bit), linearity of sensors, and the amount of precision needed for the output actions.

In considering the criteria for the system definition in the next Chapter, the advantages and limitations of a fuzzy controller were considered. Defuzzification methods covered in this Chapter included COA, MOM and the singleton method. The differences
between these methods have a significant impact when considering an efficient MPU implementation.
CHAPTER 4

SYSTEM DEFINITION OF THE FUZZY SEEDING DEPTH CONTROLLER

In this Chapter, the basis for the hypothesis that a fuzzy logic controller is suitable for use in controlling the depth of seeding of a one-way discer seeder is developed. As discussed in Chapter 2, the main methods of controlling seed depth are manual, mechanical, electrical over hydraulic limit switching, and computer-based controllers using conventional control laws. In Section 4.1, the results of researchers that have investigated these methods are presented. This is followed by a discussion of the main attributes of each method that has been investigated, as well as the attributes of fuzzy logic control. The criteria for a system to investigate the suitability of fuzzy logic control for the control of the seeding depth of a one-way discer seeder is developed in Section 4.2. A discussion of the specifications of the hardware and software to conduct the research, is given in Section 4.3. A discussion of the choices for hardware and software implementation will also be given here.

4.1 Results of Seeding Depth Studies

The problem of correctly positioning the seeds of cereal and oilseed crops beneath the surface (and on the surface, in the case of seeding with granular applicators and aircraft) of prairie fields has been a topic of study for decades [MuAr39].
Chapter 4: System Definition

Present methods of controlling seeding depth include the use of manual, mechanical, and electromechanical systems and techniques. These methods do not always provide consistent depth accuracy, which may lead to seed placement at less than optimal depths for vigorous crop emergence. This is especially true for variable terrain conditions and when using one-way discer seeders.

Virtually all one-way discer type seeding implements come equipped from the factory with a mechanical stop that is bolted at the desired position on the shaft of the hydraulic ram that controls the depth of the implement. As this only provides a very rough control of seed depth, most operators find that they still must manually adjusting depth by actuating a hydraulic control lever. This can lead to great variation in depth, as the operator must estimate the depth of seeding visually, sometimes obscured through dust and at distances of up to 20 m. This can be very tiring for an operator when conditions are such that constant adjustment is necessary. Errors increase under conditions of varying soil moisture, texture, and varying draft loads due to changes in the weight of the implement as the seed empties out of the hoppers.

A number of commercial seeding depth controllers have been made available to overcome the deficiencies of manual control. These controllers utilize mechanical or electromechanical PI, PD, and PID type control. Even with these systems, however, problems have been encountered with lack of accuracy, as well as oscillations about the depth set point. Seeding depth variations of 1.27 cm to 4.45 cm (0.5 to 1.75 inches) have been observed [MoGe85a]. This is due to the nonlinear nature of controlling seed depth-in a PID-type controller, for example, the transfer function must take into account each
necessary change in the error sum scaling and control change as field conditions vary. This leads to a complicated equation that can be quite unwieldy to incorporate into an electromechanical system, and virtually impossible to apply to a purely mechanical controller at a reasonable cost.

Fig. 4.1. View of tandem discers from the tractor cab.

It has also been shown that it is possible to maintain average seeding depths to within a 2 cm range by using microcomputer control utilizing conventional control laws [Dyck75]. However, as these methods of depth monitoring and control have increased seeding accuracy, so too have problems with complexity of operation for the operator and persistent problems with overshooting the control set point. They also do not incorporate the human operator’s ability to adjust to changing field conditions [Dyck75]. New digitally-controlled units have eliminated some of the overshoot problems [Pitt91] but still require the operator to adjust gain and damping factors (for example, 15 different settings on the Senstek DC-3) [Sens89].
Modelling a conventional controller to meet varied input states has been a persistent challenge. No mathematical model of a conventional seeding depth controller was found in the literature. As fuzzy logic control has been effective in control problems where nonlinear inputs characterize the system, this approach is being investigated to determine how well it solves the problems encountered with current conventional manual and electronic systems, since seeding depth control is a nonlinear control problem.

4.2 System Definition Criteria

The type of control needed for seeding depth is based on two variables: the depth at the present time as compared to the set point, and the rate of this error change. For a conventional controller, the error alone would be enough to formulate a control algorithm, while more complicated computer-controlled depth control units would use change in error with variable gain adjustments.

The fuzzy controller developed here would most resemble a PI-type of conventional controller. This means that the fuzzy if-then rules describe the relationship between the change in control (output) and the error and change in error (inputs). For the purposes of this research, the procedures to be used here include an overview of the mechanics of the process to be controlled, generation of various fuzzy control algorithms for simulation, simulation in FIDE, Manifold Editor and Mathcad, testing on a lab bench, and testing in the field. A laboratory test fixture with a microcontroller evaluation board was used for software and hardware development. For field testing, a proof of concept control system was built for testing the control algorithms in the field.
When devising the system to be tested, it became clear that a number of requirements had to be met. In order that the system provide reliable data, a number of criteria had to be met in the design and construction of all hardware.

1. The controller be of sufficient speed and memory capacity to operate as a Mamdani - type fuzzy logic controller.

2. The entire system be designed to provide sufficient electrical noise immunity in order that all measurements are accurate and reproducible.

3. That all mechanical components, including sensors, be free from excessive variation in accuracy.

4. The controller be easily reprogrammed in the laboratory and field.

5. All electronic components be sufficiently protected from surges and fluctuations in voltage levels of the tractor electrical system.

6. The ground detection sensor be sensitive enough to allow operation within the minimum error defined for accurate seeding.

7. The controller MPU operate fast enough to allow for all necessary calculations and operations to occur and still have a sensor sampling rate fast enough to prevent aliasing.

8. The hydraulic valve have sufficient bandwidth to raise and lower a disc type seeding implement as steps as small or large as necessary, with sufficient speed to keep seeded depth at optimum.

9. The rate of oil flow of the hydraulic valve must be variable to allow full use of the fuzzy logic control capabilities of both direction and speed.
10. The system must have a means of logging data in order that algorithm operation and efficiency can be precisely measured.

11. The tractor mounted controller must not impede the safe operation of the tractor while operating performance and other measurements are made.

Therefore, it must easily fit in the cab, with an interface that is easily viewed and operated.

4.3 Seeding Depth Control System Hardware

A complete seeding depth control system has four basic parts: the signal processor/control (or simply controller) module, depth sensing module, solenoid valve module, and power supply module. For the field data collection unit, all the electronics of these modules are protected in a single shielded aluminum box that is operated in the cab of the tractor. Information about the depth in the form of variable DC signals are sampled from depth sensors, through the sensor module. An analog control signal is generated, which controls an electrohydraulic solenoid valve through the hydraulic control module.

The controller module includes all controller memory, interfaces to other modules, operator user interfaces, and the CPU which conducts the computation functions. The controller has been designed so that it can display information for the operator while allowing the operator to control normal operating and calibration functions. The controller also comes equipped with suitable interfaces for gathering sensor input information, transmitting control signal information to the hydraulic control module and to an external PC logger through a serial interface.
Chapter 4: System Definition

The computational role of the controller is to compare the input signals from the depth sensors to a set point, and calculate an appropriate control signal to correct any difference in an optimal way. The decision about the type of microcontroller used was based on a number of criteria, which include

- sufficient addressed memory capacity to be able to store the necessary computer code;

- sufficient speed to allow for at least one sampling every 6 inches in the field, at a speed of 6 miles per hour;

- as many of the necessary peripherals as possible, in order to reduce chip count, which would result in higher reliability and lowered complexity (important when debugging);

- adequate monitors and compilers to be able to write, compile and debug assembly code effectively; and

- the availability of an evaluation board for laboratory testing.

MPUs (microprocessor units) are VLSI chips that include not only a CPU core for computations, but also have on-chip modules for such functions as I/O, RAM, ROM, EPROM or static RAM memory, pulse wide modulation module(s), A/D converters. There are a very large number of chips available from a number of vendors with variations of these features, with the major ones considered for this work. Some of these include

- Motorola MC68HC11, with 8-bit CPU, I/O, A/D's. Runs at 2 MHz, and has less addressable memory space.
Chapter 4: System Definition

- Motorola MC68HC05, with 8-bit CPU, I/O, A/D’s. Very small addressable memory space, and runs at only 2 MHz.

- Texas Instruments TMS320 DSP series, with very fast operation for DSP functions, with I/O and external A/D’s. Evaluation board version not readily available, and not as well suited to controller functions with less I/O ports.

In order that it meet all these requirements, the controller is based upon the Motorola MC68HC16 microcomputer unit (MPU), with on-board analog-to-digital converters, memory, and a 6805-based microprocessor. This MPU runs at a speed of 16.78 MHz, which has proven to be adequate to conduct all computations and functions fast enough to provide a sufficient sampling rate of 23 times per second.

![System block diagram-seeding depth controller.](image_url)

Fig. 4.2. System block diagram-seeding depth controller.
Chapter 4: System Definition

The depth sensor unit used is a simple novel design, producing a DC voltage signal proportional to the height of the seeder frame above the ground. A second sensor monitors the position of the discs relative to the frame.

Highly linear potentiometers convert the angle variations of the two sensor assemblies into voltage signals. As it is difficult to maintain high accuracy with one-way discer seeders, the sensors were selected to meet this worst-case scenario.

A gauge wheel used in the sensor uses a wide, low-pressure tire designed to ride on the soil regardless of the soil consistency. This method should also be unaffected by the varying actual height profile of a zero-tillage stubble field surface and to moderate field trash (straw), both of which are read by ultrasonic sensors [GKWD81a].

The hydraulic control module is the interface and buffer between the 5 V control module and the dual 12 V hydraulic solenoid valves. This module also serves to convert the output PWM signal from the controller to two opposite signals, one for each solenoid valve. A proportional hydraulic valve is the best choice for this application, as it is capable of both variable rates of oil flow through the main spool and fast enough actuation in either direction for this application.

The electrohydraulic solenoid valves control the height of the seeding implement. They are driven by opposing signals from the hydraulic control module: one which forces the main hydraulic spool to produce a positive hydraulic flow in order to move the hydraulic ram piston in one direction, and the other to allow oil to drain from the opposite side of the piston of the hydraulic ram, thus allowing the piston to move. The signals can be reversed to move the hydraulic ram piston in the opposite direction, or equalized to
keep the piston in place. A small, constant dither signal was imparted on the main spool at 128 Hz. This is required in these types of proportional valves, as it maintains an oil film on all moving parts, reducing spool wear and improving reaction time.

4.4 Seeding Depth Control System Software

The software design requirements are based upon the requirements for the fuzzy logic controller and the practical requirements of this research.

1. The operator must be able to calibrate and control the fuzzy logic controller,
2. The operator must be able to log data to verify results.
3. The interface must also be as simple as possible in order to prevent mistakes or pose a safety hazard by distracting the operator.
4. The fuzzy logic algorithm must provide for a target accuracy of 1 cm.

The software flow is given in Fig. 4.4. Once the controller has been initialized, it then checks for any operator input. This is an interrupt-driven routine, in that if a keypress has been detected, the seed depth control operations in the main flow of the code is temporarily suspended in order to execute the operator’s command. The operations that are controlled by the operator include the calibration routines, raising and lowering of the discs, measuring units and depth control settings.

Once past the keypress detection stage, the controller proceeds to the sampling routine. On the discer seeder, there are two sensors, one on the disc arm and one on the depth sensor in the front. Both sensors are sampled, and the error calculated based on the distance from the desired seeding depth set point.
In the error conversion stage, the result is converted to a form suitable for use in the fuzzy logic control algorithm and the depth display algorithm. The units used in these calculations are in thousandths of an inch, as this provides the best combination of roundoff error reduction and computational efficiency for later operations.

Once the error has been calculated, converted and stored, the fuzzy control algorithm reads the error from the current sensor readings and the reading from the 7th previous sampling instance to put through the fuzzy inference engine. A decision value is determined, and passed on the hydraulic control routine.

The hydraulic control routine takes the value from the fuzzy decision, and scales it appropriately based on the type of hydraulic valve being controlled. It is then converted to a pulse width modulated (PWM) signal, from the PWMA port on the MPU.

The value of the current depth is read from the registers storing the converted depth sensor values and displayed in the LCD screen for the operator. The units can be either imperial or metric, depending on which choice is made by the operator.

Finally, data from the sensors, error, change in error and fuzzy decision is converted to an RS232 signal and transmitted to a PC laptop serving as a logger. The data can then be analyzed to determine how well the process is working. Once this operation is completed, the program loops back to the keypress detection routine to begin the entire process over again.

4.5 Chapter Overview

The operating and construction requirements for the fuzzy seeding depth controller were given in this Chapter. These requirements are the result of the necessity for a certain
degree of accuracy over a variety of soil and topographical conditions. Results of
experiments explained in the literature showed the strengths and limitations of different
seeding systems and control methods. Problems with current systems were considered,
with an eye to setting system specifications that meet or exceed the accuracy which can be
gained using commercially available equipment and control philosophies. The primary
objectives were the elimination of excessive oscillation while maintaining as much
simplicity in the controller program code and hydraulic system as possible, and maintaining
depth to at least 2 cm.

Once the main functional criteria were established, further definition of the
hardware and software was the next step. The choice of the MC68HC16Z1 MPU was
justified by its capabilities to meet all the necessary I/O, memory, availability and processor
speed requirements. The choice of a proportional hydraulic valve offers the system a robust
control actuator that is compatible with the variable output gain needed in this
implementation of fuzzy control. The depth sensors met the criteria for accuracy, with noise
suppression maintained by shielded cable and differential inputs. Finally, the flow of the
software code actions to tie all these hardware components and the fuzzy control algorithm
together was defined. Most of the code is used in measuring the sensor positions,
calibration, processing the input signals and displaying the depth on the LCD.

With all of the major hardware criteria defined, and hardware types chosen in order
to meet or exceed these criteria, the fuzzy depth control system can now be described in
detail. In Chapter 5, the controller modular hardware design is described, based on the
general requirements justified in this Chapter.
CHAPTER 5

DETAILED HARDWARE DESIGN OF

THE FUZZY LOGIC - BASED DEPTH CONTROLLER

The system described here has been constructed and is being tested on a one-way discer type seeding implement. There were two versions of this system built: a preliminary version built with the MC68HC16Z1EVB evaluation board from Motorola, and a field test unit for operating from a tractor to log pertinent controller state data in the field for later evaluation.

The seeding depth control system description will be by modules, as described in the previous Chapter. Section 5.1 will cover details relevant to the controller module, 5.2 will describe the depth sensor module, with Section 5.3 covering the hydraulic control module. The power supply and conditioning module will be explained in Section 5.4.

Fig. 5.1. Lab test unit.
The hardware descriptions in sections 5.1 to 5.3 will be based upon the basic operations and components of the controller, with Section 5.4 also covering noise and EMI considerations that apply to the power supply and signal cables.

![Field test and data logging unit](image1)

**Fig. 5.2.** Field test and data logging unit.

![Controller main modules](image2)

**Fig. 5.3.** Controller main modules.
5.1 The Controller Module

The controller includes the hardware involved in the processing of the input signal to produce the desired output control signal. This includes analog to digital (A/D) conversion of the input signal, comparison of this signal to preset parameters in software, and the fuzzy logic decision process that controls a pulse width modulated output signal. This module also serves as the operator interface, so is located in the tractor cab next to the operator. It has a numeric keypad for input of calibration and control point information from the operator, and an LCD digital display that serves as a visual interface (see Fig. 5.3).
Chapter 5: Detailed Hardware Design

The microprocessor unit employed in this application is the Motorola MC68HC16Z1 (referred hereafter as the HC16). The HC16 is a microcontroller that is suited to this task because of the suite of on-chip features and a clock speed of 16.78 MHz.

Features utilized in the fuzzy seeding depth controller include the A/D converters, on-chip general purpose timer (GPT), queued serial module (QSM), and 1 KB of battery-backed static RAM.

![Block Diagram of the MC68HC16Z1 showing main modules and ports.](image)

Fig. 5.5. Block Diagram of the MC68HC16Z1 showing main modules and ports.

The A/D converters are selectable between 8 or 10 bit, with 8-bit conversions being used in this application. This is because it provides adequate accuracy in monitoring the depth sensors, while enabling the 8-bit A and B registers to be used in the CPU16. This results in a reduction in complexity in programming and allows two bytes to be used for
data, not 3. They are successive approximation (SA) type A/D converters, with a sample and hold (S&H) unit to increase the effective aperture time [Kins91]. As the A/D module can run at up to 2.1 MHz, an 8-bit conversion only requires 9 microseconds (ms) to complete a conversion.

Fig. 5.6. Controller module main components.

The QSM is used to communicate between the HC16 and a PC in order to log test results. The QSM is configured for 9600 baud full duplex asynchronous communication in this mode. Serial communications signals from the QSM are converted to RS-232 levels by a Motorola 145450 serial interface. This chip was chosen over the more common 1488 and 1489 chip combination in order to reduce the part count, and eliminate the necessity for plus and minus 13 V supplies. The 145450 uses four 10 uF electrolytic capacitors in a
charge pump circuit to produce its own plus and minus 10 V supplies. This is useful for the field test unit, as this reduces the size of the boards and box needed to do the control and data logging. Likewise, whenever the necessary functions can be implemented with fewer parts, there are less chances for failures in initial debugging and in operation.

On-chip static RAM is used to store program variables, such as the calibration values. These variables are maintained by a lithium 3 V battery while the unit is in a battery backup standby mode when the main power is switched off. As all the registers in the HC16 are memory-mapped, there are no other special addressing procedures needed to access this memory other than to ensure the correct base address of the 64 KB page is in the K register before reading or writing these locations.

In order that there would be enough room around the HC16 chip to use wire wrapping in the construction of the field unit, a single sided PC board was made. This enabled the HC16 to be securely soldered in place using surface mount technology, while allowing the critical clock circuitry to be assembled on the same board. There are several advantages to having the clock crystal and capacitors mounted directly to the PC board: the distance to the HC16 pins is minimized; problems due to stray capacitance of wire wrap pins affecting the clock waveform are eliminated; and putting the crystal on the PC board allows for a foil ground plane to be extended under the crystal, thus further ensuring stability of the output waveform.

The HC16 is also capable of addressing up to 1 MB of program and data space, of which 32 KB of program space are used in this application. Two fast (70 ns) 256K EPROMS or 70 ns 256K battery-backed RAM acting as pseudo-ROM can be used in the
controller. There is a jumper block that sets the correct addressing, chip select and chip enable pins for the appropriate type of memory configuration in use. The ability to use battery-backed RAM is very advantageous for field use. The two RAM chips are each powered by a Dallas Semiconductor SmartSocket. This way, the chips can be programmed in the lab bench controller setup, which is connected to a PC running the HC16 assembler (MASM16) and control program (EVB16). Once the chips are programmed, the whole system is powered down, and the chips can then be removed and placed into the memory sockets of the tractor unit for in-field testing. This way, it is not necessary to have an EPROM burner and a large collection of EPROMs and an EPROM eraser on hand, so quick changes to the code can be made easily on the spot.

Within the GPT are two pulse width modulation (PWM) units. One of these (PWMA) is used to produce the output signal for controlling the hydraulic valve. The output from PWMA is a periodic square waveform set to 128 Hz, which is optimal for the Rexroth proportional valve being used in this research. The duty cycle of the waveform is then modified in software to cause the appropriate valve action.

The keyboard and LCD display are both mounted on the top of the controller box for easy access by the operator. Both are socketed in order that they can be easily removed if it is necessary to repair connections or add connections or parts underneath them later. A plexiglass cover protects the LCD display.

The keypad is a hexadecimal keypad, arranged in a $4 \times 4$ matrix for operator input. Key press detection is by scanning rows and columns via the 8 I/O lines of control port F, as shown in Fig. 5.4. When a keypress is detected, an interrupt routine is triggered in
software that is controlled by the GPT. This software subroutine will then intermittently resample the line for 10 milliseconds. This enables the microcontroller to determine whether a detected key press was due to electrical interference (noise) or whether the event was a real keypress by the operator. This serves as a software-based key debounce function. Pullup resistors on each line maintain a steady 5 V level on the port pins until pulled low by a keypress.

![Keypad connections to Port F of MC68HC16Z1.](image)

A 16-segment LCD display serves as the visual display interface for the operator. The LCD has an adjustment for contrast, and is set for maximum visibility by a potentiometer inside the controller unit. Instructions are supplied to the operator via the LCD in natural language to facilitate calibration and operation. Present depth is displayed while in normal seed depth control operation. The unit used is the single line, 16 character Optrex DMC40401 which has its own built-in control logic and oscillator, thus requiring no external parts. Custom characters could be programmed and stored into display memory, such as arrows indicating depth correction signals.
5.2 The Depth Sensor Module

The depth sensor module includes the sensor mechanism mounted on the discer (see Fig. 5.5), the shielded cables running from the discer to the tractor cab, and the two differential sensor interfaces that produce the controller A/D input signals. The front depth sensor used here is a simple, novel design that produces a varying DC signal based on the angle of a gauge arm from the soil surface to the discer frame. A second sensor signal indicates the angle of the disc gang to the implement frame. Highly linear, calibrated potentiometers convert these angles to voltage signals. Noise suppression is achieved by the use of a differential input buffer, and shielded cabling grounded to the controller box.

![Fig. 5.8. Front sensor gauge wheel assembly.](image)

The novel ground-sensing unit used in the design is key to obtaining reliable depth readings. A gauge wheel used in the sensor uses a low pressure tire designed to ride on the soil surface, regardless of the implement weight or soil consistency. The adjustable weight of the gauge wheel allows the operator to prevent excessive sinking in soft soil. Similarly, the use of a low-pressure tire to sense ground level serves to dampen out insignificant topographical features, such as small furrows, while still giving reliable readings in fields with standing stubble. In modelling this system, the sensor assembly itself serves to act as an integrator and a low pass filter. This is particularly important for minimum and zero
Chapter 5: Detailed Hardware Design

tillage fields, where straw and stubble are not left intact in order to minimize soil erosion. This dampening feature, combined with analysis of the rate of change in depth by the fuzzy logic algorithm, filters out the effects of sudden changes in depth reading caused by large soil clods or furrows. These features provide this sensor design with a distinct accuracy advantage over ultrasonic sensors and other types of mechanical sensors presently in common use, although it could be argued that low-pass filtering of the signals from these units can be done in software.

The sensors themselves are highly linear (< 0.5% variation across 85 degrees of travel) units that are sealed from moisture and dirt. They employ robust mounting flanges, and employ spring pressure to ensure that they always are in contact with the machined sensor linkage. Cabling is with 6-conductor, 100% shielded cable. The shielding is grounded at the tractor, and is not connected to the ground wire on the sensors, in order to prevent a ground loop. Cables are mounted on the seeding implement in an elevated ring, to prevent damage from dragging on the ground. An Amphenol-type screw connector is used to securely fasten the cable to the controller box and provide grounded shielding at the cable to connector interface.

Each of the sensors are buffered by TMS4066 operational amplifiers. This serves to protect the HC16 A/D converter inputs from any high voltage spikes that may be induced in the sensor cabling. The operational amplifiers also serve a noise elimination function, as they employ differential inputs. This technique of noise suppression has been shown to eliminate noise to about 60 dB. These op amps are also unique in that they operate as single-supply, rail-to-rail differential amplifiers. The advantage of this is that no
Chapter 5: Detailed Hardware Design

An extra negative power supply is needed to run the op amps. The rail-to-rail feature also means that the op amp inputs are sensitive right across the power voltage level, so can detect signals from 0 to 5 V. This eliminates the need for additional analog scaling components, thus reducing part count and eliminating the extra variables of temperature and age effects on the input scaling to the op amp differential inputs.

5.3 The Hydraulic Control Module

An electrohydraulic proportional valve adjusts the height of the seeding implement (see Fig. 5.7). It is driven by a 12 V pulse width modulated signal in order to control both the speed and direction of disc depth adjustment (Fig. 5.6).

![Hydraulic control subsystem components.](image)

This module consists of power field effect transistor (FET) switches, protection diodes, logic circuitry to divide the pulse modulated signal from the PWM units on the microcontroller, and a proportional hydraulic valve controlled by two solenoids. The HC7414 CMOS inverting buffer protect the control module circuitry from any electrical shorts or power spikes in the 12 volt section of the hydraulic control module. Protection
diodes allow back EMF in the inductors of the hydraulic valve solenoids to bypass the FETs when the current is cut by the switch logic.

The hydraulic solenoid valves operate at a frequency of 128 Hz. This produces a dither on the main valve control spool that significantly increases its response speed over a valve that must operate from a state of rest. At a duty cycle of 50%, an equal force is imparted on the main spool in both directions. This results in no net oil flow to the discer’s lift cylinder through either of the main spool’s ports. However, as that duty cycle changes, there is a proportional change in the flow of oil from the valve to the hydraulic cylinder controlling the implement depth. Therefore, by adjusting the control signal duty cycle above or below 50%, the speed and direction of oil flow to the cylinder is controlled.

The cable to the solenoid valves is a 4-stranded, 100% shielded cable with the shielding grounded to the controller box. In order to prevent ground loops, the grounded
Chapter 5: Detailed Hardware Design

shielding is not connected to the ground of the solonoid control wires. The cables are connected to the controller with and Amphenol-type steel screw connector that is grounded to shield the solonoid control wiring where it meets the connector. The solonoid end of the cable is connected to a plastic plug which is provided by Rexroth and custom designed to fit over three solonoid lugs and bolt into place.

5.4 Electrical Noise and EMI Suppression

Constructing electronic equipment using microprocessors for industrial and agricultural environments poses unique challenges. Agricultural tractors pose a unique hazard to electronic devices, as EMI sources as varied as the tractor electrical system, wireless communication devices and high power overhead lines may all be present at the same time in worst-case situations. This is especially true for systems such as that used in this research, where cables to sensors also act as very good antennas!

Precautions were also implemented to protect the controller unit from power fluctuations within the tractor electrical system. For example, in a load dump, in which the connection between the batteries and the charging system is suddenly severed while the motor is running and the electrical system is otherwise functioning normally, voltage spikes have been recorded to over 400 volts [Harr94].

The controller box itself serves as an RFI shield. It is constructed of aluminum, with only minimal openings where needed to provide user and cable interfaces. To minimize the amount of cable within the controller box that may carry noise from the tractor electrical system, shielded lines to the power supply module only pass beside the
hydraulic control module, and do not pass through the control module. The power supply module is also separated from the adjacent modules by aluminum shielding.

5.5 Power Supply Module

The 12 V power cable from the tractor electrical system connects directly to a 5 A fuse, then to a 24 V metal oxide varistor (MOV) that is designed to take any voltage spikes above 24 V to ground. Likewise, a 500 mH ferrite core choke and a 300 mF electrolytic capacitor serves as a low-pass LC circuit to resist large, low-frequency signals or spikes. This is followed by a 100 mF electrolytic capacitor, to further take higher frequencies to ground.

![Diagram of Power Supply Module](image)

**Fig. 5.11.** Power Supply Module.

In the voltage regulator circuit there are two 10 mF capacitors, which also act as filters. A 0.01 mF capacitor also acts to take even higher frequency signals to ground. Finally, regulated 5V power to the controller module is supplied via shielded cables which pass over the aluminum shielding to supply the module. 12 V power to the hydraulic control module is via a shielded cable which passes over the shielding between the power supply module and the hydraulic control module.
Liberation use of decoupling capacitors and several inductors throughout the controller module keep voltage variations to a minimum. Special attention is paid to the memory supply pins and the A/D controller power and reference pins. Decoupling capacitors are especially important considering that the majority of parts in the controller module were built using wire-wrap construction. With the HC16 clock speed at 16.78 MHz, this is just below the generally accepted limit of 20 MHz for wire-wrapped circuits. There is also an aluminum shield between the hydraulic control module and the controller module to reduce noise effects that may occur due to the back EMF in the solenoid cables generated by the collapsing fields around the coils in the valve solenoids. All of the modules were built as small as possible to minimize the length of connecting wires.

5.6 Chapter Overview

When reading the detailed hardware design description, it is not immediately apparent that the hardware choices used in this study are the result of extensive searches for the appropriate components. For example, there are hundreds of types of potentiometers in a wide array of packages available, but very few have the combined capabilities of being dustproof, able to be securely mounted, and a high degree of linearity.

Once all the suitable testing hardware was found, the software was designed and written. Hardware characteristics were taken into account in this process, including the speed and zero position of the proportional valve main spool, and the linearity of the sensors. In the next Chapter, this software design is described in detail.
Chapter 6: Software Design

CHAPTER 6
SOFTWARE DESIGN OF THE FUZZY LOGIC-BASED SEEDING DEPTH CONTROLLER

Software design in this Chapter describes the fuzzy logic controller and how the most important software tools associated with this research were used. The fuzzy logic controller itself is written in the 16-bit MC68HC16 assembler code, with portions of the fuzzy logic routine written in 8-bit MC68HC11 assembler code. This format is fully compatible with the MC68HC16Z1. Other programs written for the purpose of this research, but not directly used in the formation of the fuzzy logic controller code, are discussed in Chapter 7, System Implementation. Source code of all software presented in this thesis can be found in the appropriate appendices.

This Chapter is presented in three sections, each covering functions occurring before, during, and after fuzzy decision routines. These sections are detailed descriptions of the block diagram components and primary system functions given in Chapter 4. Each subsection describes a major subroutine, in the order they are executed in the microcontroller. The subsections are:

1. Initialization - Initialize MPU modules and other registers.
2. CHECK_KEYS - Detect a keypress and identify key, execute key function.
3. SAMPLE - Sample the sensors.
4. ERROR_CONVERT - Convert the sensor signals to a usable form.
Chapter 6: Software Design

5. ERROR_BUFFER - Store current error, read error 7 samples previously.

6. FUZZYTEST - Fuzzy control decision routine.

7. HYD_CONT - Send the control signal to the hydraulic valve.

8. DISPLAY_DEPTH - Display depth on the LCD.

9. SEND_DATA - Send data to the PC data logger computer.

Section 6.1 describes the program and microcontroller initialization, calibration and data preparation for the fuzzy logic decision routines. Section 6.2 presents the fuzzy routines, while 6.3 covers the hydraulic control, data logging and LCD display functions. A detailed flow chart of program execution is given in Appendix B.

6.1 Initialization and Data Preparation Routines

6.1.1 Initialization

Each of the modules of the HC16 are controlled by memory-mapped registers that must have initial values written to them in order to perform in the appropriate manner [Moto92b].

The first module initialized is the SIM (system integration module), controlling bus and software watchdogs, configuration parameters for the HC16, system clock, system address and data buses, chip selects and a system test block.

The system clock provides timing signals to all operations in the MPU. In the MPU configuration used for this research, the clock is driven by an external 32.768 kHz watch crystal. This in turn controls the speed of an internal oscillator (PLL, or phase lock loop). The high frequency output from the PLL is divided down by a divider/prescaler and the output compared to the phase of the input crystal frequency. Once the PLL locks on
this signal, the HC16 boots up. The speed of the HC16 is determined by the value passed to a divide by two prescaler bit, a 3-bit prescaler in the PLL feedback divider, and a 6-bit modulo 64 down counter. System speed with a 32.768 kHz crystal can be set from as low as 4 Hz, to values in excess of the maximum rated operating frequency of 16.78 MHz. The HC16 is set at this maximum allowable speed by the initialization routine.

The modules are mapped to a 4 Kilobyte block in the MPU. The location of this block is set by the module mapping bit in the SIM module configuration register. This bit is set at initialization so that the addresses are spread within the range from $FF000 to $FFFF.

The the SWE (software watchdog enable) is also turned off to prevent immediate software interruption by the watchdog. This is a timing circuit that, if not disabled, would continually interrupt the operation of the CPU if the software watchdog timer location was not written to within a specified time.

Initial values for the CPU (central processing unit) also are set. The memory page registers are set so that the code will start at the correct memory page base location. Each memory page on the HC16 is 64 Kilobytes.

The SRAM (standby RAM module) consists of 1 Kilobyte of fast static RAM and a control register block. This memory is where calibration and scaling values are stored, as they can be maintained by a backup battery when the MPU powers down. This memory can also be mapped anywhere within the entire HC16 memory space (as long as it does not overlap any module registers or the bootup and interrupt locations from $00 - $200). It is initialized by setting it to the base location $10000.
Chapter 6: Software Design

The QSM (queued serial module) is initialized to be able to send and receive data at 9600 baud. This module sends and receives signals at TTL levels. It is connected to the MC145407 RS-232 interface chip.

The ADC (analog to digital converter) module is enabled by writing to the associated control registers for ADC channels 0 and 1. Channel 0 is the input for the front, or depth gauge wheel sensor, and ADC1 is the input for the rear, or disc sensor.

GPT (general purpose timer) initialization values are written to the appropriate control registers, setting the frequency of PWMA to a rate of 128 Hz. The duty cycle of this signal carries the information to the hydraulic solenoid valves, while the frequency remains constant throughout operation.

Initial values for controller operation variables are set once all the modules are enabled. These include calculation constants for the theta correction in the sensors, rough sensor zero positions, sensor mode settings, and rough calibration factors such as seeding depth setpoint and raised height for lifting out of the ground. Initialization subroutines that enable the appropriate I/O ports for the LCD display and keypad are also run.

Once the HC16 modules and calculation values have completed initialization, the fuzzy logic controller code enters the main software operation loop.

6.1.2 CHECK_KEYS Routine

This code routine checks to see if a key has been pressed. If this is true, the key is identified and the appropriate action is taken.
The keypad is scanned in this routine, on every loop of the main program. If a low signal is detected on one of the lines of the keypad port (Port F), the code immediately branches to a key debounce subroutine. Port F is scanned again in 10 ms, to see if a keypress (low) signal is still present. If so, it is assumed that a key was pressed, (that the first low signal detected was not caused by noise). The program then branches to a routine to identify the key pressed. If the identification produces a jump to a subroutine, the HC16 then executes the appropriate function. These functions include calibration and control routines for the fuzzy logic controller.

Fig. 6.1. Keypad scanning directions.

Fig. 6.2. Actual and ideal signals from a switch, illustrating the need for a key debounce routine.
Once the routine has finished, the program exits the CHECK_KEYS routine and continues on to the SAMPLE routine.

Control and calibration subroutines are entered, depending on which key is pressed. The function of each keypress is explained in Chapter 7, System Implementation. Calibration subroutine structure will be explained here.

There are three major types of calibration subroutines. The first, and most simple to implement in code, are operator set point value settings. These are the values for desired display units (metric or imperial), sensor modes (both sensors or front only are used), seeding depth and raised height. Setting these are a simple matter of entering the numeric values when prompted by the appropriate message on the LCD.

The second type of calibration setting includes values that are read from sensors, then directly stored to their respective locations in memory. This includes the setting for the field “zero depth” position. No calculations occur in this setting; it is a straight “read - and - store” operation.

The third type of calibration setting includes values that are determined by reading sensor positions, and using these in calculations to determine the appropriate scaling and conversion factors. These calibration subroutines include the sensor scaling factor and the theta correction factor.

This scaling value will be unique to each different combination of depth sensor and seeding implement combination. Once it is set, it does not have to be updated unless there is a change in the position (such as the location or rotational pitch) of sensors relative to the implement they are mounted on. Therefore, this calibration should be repeated
anytime sensor mechanical mountings are remounted or any changes are made to the sensor system, such as a change in gauge depth wheel tire pressure.

The scaling factor is obtained by calculating the difference between the 6" height position of the disks and the 0" (ground level) position of the disks, and then dividing this difference into 6 to obtain sensor units per inch of travel. These measurements are made on a solid, level surface, such as on a concrete pad. It is critical that the surface be level and all measurements be as accurate as possible, as any error in this calibration will result in the same relative error throughout all depth sensing operations.

The theta correction factor is also determined in this operation. It is necessary to correct for this error due to the rate of change of sensor angle per change in depth becoming smaller as the angle increases. The disc gang is raised so that the sensor linkage for the rear sensor is positioned in such a way that the connecting arm from the sensor and the connecting arm to the discs are precisely 90 from one another. The front sensor wheel is also positioned so that the sensor connecting arm is precisely positioned 90 from the sensor arm connecting the ground sensor. The positions of both sensors are now stored as the zero degree angle position.

By using 8-bit ADC conversion rather than 10 bit, each degree of movement of the sensors used here translate to a change in ADC reading of precisely $3. This is due to the fact that in an 8-bit ADC, the range is from decimal 0 to 255. The total degrees of rotation of the sensor type used in this research is 85. Thus, decimal $255/85 = 3$. It has also been found that a variation of 1 unit of ADC measurement with the discer and sensor combination used in this research was the equivalent of 14 units per inch around the zero
point, to about 12 units at the 3 inch depth point. This is sufficient to give the 0.1 inch sensor accuracy desired for this controller. Using the 10-bit setting for the ADC would provide finer readings, as there would be decimal 12 units rather than 3 per degree. However, having to use 10 bits of accuracy would complicate programming, as the 8-bit accumulators A or B could not be used independently with the ADC readings in the controller routines (such as conversion, data logging and memory management operations).

Once the 90 sensor linkage positions are determined, the number of degrees of rotation of the individual sensors from this position is simply this difference divided by 3. Therefore, the degrees of movement of the 6 inch positions from the 90 points of the sensors is known. This gives the value of the opposite side of a right angle triangle (is equal to 6 inches). Basic trigonometry gives the length of a hypotenuse equal to the length of the opposite side divided by the sine of the opposite angle. Using this “opposite” value (6 inches) and the angle from 90, the length of the hypotenuse is determined. This hypotenuse length now becomes a constant for use in all other calculations.

Other depth values can then be determined by converting the sensor readings into degrees, then converting the degrees rotation into sensor position above or below the ground by applying the scaling factor obtained in the 6 inch calibration routine, and using this to determine the length of the opposite side of a right angle triangle to get the depth value. In order to prevent excessive roundoff error throughout the correction calculations, the sensor values are converted to thousands of an inch in a 16-bit format. This process includes a hexadecimal to decimal base conversion subroutine.
6.1.3 SAMPLE Routine

Each of the two sensors are sampled in this subroutine. If the controller is in normal operation mode, the signals are passed onto the next subroutine. If it is in a single sensor mode, only the respective single sample is passed on.

The sensors are highly linear resistive elements operating by indicating the position of a contact on a circular track. In order to preserve accuracy in reading the sensor positions, the theta correction factor is employed for all readings. This works by taking the sine of the angle from the 90 position of the sensor linkage, where it is precisely 90 from the sensor. In order to ensure maximum calculation speed, an 8-bit sine lookup table is used, rather than calculating each value by division. In the HC16, the sine table technique takes 6 clock cycles to load the value. If sine had had to be calculated, it would take 34 cycles to clear, load and scale the ADC reading (in degrees) and the value 1 into another accumulator, then 24 more cycles to divide these two, and another 10 cycles to scale them again, for a total of 56 clock cycles. This would have to be conducted twice every time through the program, resulting in an unnecessary loss of 124 clock cycles per main program execution. The only downside of using the sine table is that it takes up extra EPROM memory, but with only about 10 Kilobytes used out of a total of 32 Kilobytes available, this is not an issue. In this design, optimizations have been made for speed whenever possible, as long as accuracy is also within specifications.

Initial conversion values for the theta correction subroutine are written to the respective registers during controller initialization. This allows the operator to raise the implement for transport to the area of use. Once at the field, the operator then goes through
a calibration routine that sets the registers with the correct conversion and setting values for the current soil conditions and desired depth.

6.1.4 ERROR_CONVERT Routine

Once the positions of the sensors are determined and corrected for theta error, the error is converted from thousands of an inch to tenths. This facilitates the calculations that will include only addition, subtraction, and maximum and minimum comparisons.

6.1.5 ERROR_BUFFER Routine

In this routine, the change in error is determined by taking the present error reading and subtracting it from the error reading 7 samples previously. This equates to about 1/2 a second, or the equivalent of one metre of distance covered at 6 mph. This value is written to the CHANGE_IN_ERROR variable for use in the next routine.

6.2 The Fuzzy Control Subroutine:

The fuzzy control algorithm used here is a Mamdani-type controller, based upon the max-min composition. This type of fuzzy controller has been applied to a wide variety of industrial processes since it was introduces 20 years ago [KiPe90, Mura84]. This control strategy is based upon the experience of a human operator, who linguistically describes the control actions as a set of heuristic decision rules in the form of "if this occurs, then do that" [KiMa77]. The result is that the control algorithm incorporates the "grayness" or "fuzzyness" of natural human decision processes, where gradual changes are perceived on a gradient rather than in terms of crisp absolutes.

The inputs to the fuzzy control rules include the error from the desired depth set point, and the change in the error between the present and last sampling instances. This
error change leads to differential control [KGFT93]. These proportional and differential input signals should guarantee stability of the control system.

Linguistic variables describe the direction (either positive, P or negative, N), while the size of the change is described as small, S, medium, M and big, B (see Table 1). Rules used in this fuzzy logic controller are of the form:

IF error is PM and change in error is NS then the output is PM.

The fuzzy output variables are the desired speed and direction of change in seeding depth. As for the fuzzy inputs, these are described by means of a linguistic variable, for example, Positive Medium. This value is changed to an 8-bit number by defuzzification. Both Mamdani-type (utilizing membership functions) and singleton (center of gravity) defuzzification methods are being considered for accuracy and speed of execution.

<table>
<thead>
<tr>
<th>OUTPUT</th>
<th>ERROR, ( \epsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NB</td>
</tr>
<tr>
<td>PB</td>
<td>NB</td>
</tr>
<tr>
<td>PM</td>
<td>NB</td>
</tr>
<tr>
<td>PS</td>
<td>NB</td>
</tr>
<tr>
<td>0</td>
<td>NM</td>
</tr>
<tr>
<td>NS</td>
<td>NM</td>
</tr>
<tr>
<td>NM</td>
<td>NS</td>
</tr>
<tr>
<td>NB</td>
<td>NS</td>
</tr>
</tbody>
</table>

Fig. 6.3. Fuzzy rules output table.
6.2.1 FUZZYTEST Routine

The fuzzy algorithm is based on the Mamdani fuzzy control algorithm. Membership functions were originally determined by using Mathcad and a Microsoft Windows graphics-based fuzzy logic simulation and analysis tool, FIDE (Fuzzy Inference Development Engine) from Aptronix. The different membership functions were set up and run through simulations.

Although fuzzy algorithms can easily be coded in a high-level language such as C, FIDE speeds up development with increased accuracy by the use of several valuable development tools. Each membership function can be viewed graphically, and quickly changed using the mouse. Likewise, FIDE instantly generates the fuzzy control surface for any set of membership functions and inputs. Once a given combination of membership functions produce an acceptable gain level of monotonic control, a visual simulation can be run with inputs from either artificially generated signals and actual recorded field data. If, at this stage, the control looks like it will produce the appropriate results, FIDE can immediately generate 8-bit (MC68HC11 compatible) assembler code. This code can then be pasted into the controller code and compiled. The code automatically takes into account scaling factors for sensor inputs and the hydraulic control output signal. These scaling factors are used to match the fuzzy output signal to the actuation speed characteristics of the type of hydraulic proportional valve used. Therefore, code is produced quickly and accurately that can be tested in the field. Some setting up of the code is necessary, such as naming variables and memory location redefinition, but this is trivial compared to having to rewrite all fuzzy membership function tables manually. Of course,
membership functions can be modified by editing the tables, but it is much more productive to modify only small part of the code in the field rather than having to rerun all the simulations and repaste code for only minor variations.

In this routine, the fuzzy logic algorithm operates the four steps of determining a control decision:

1. The input values ERROR and CHANGE_IN_ERROR are fuzzified according to the lookup tables generated from the membership functions in the FIDE simulation. The degree of membership for each membership function is then passed on the fuzzy inference engine.

2. The fuzzy values are evaluated according to the rules. The DOF for each rule is determined.

3. The rules are aggregated, and the rules that apply are determined.

4. The fuzzy decision from the rule analysis is defuzzified. The algorithm uses the singleton method of defuzzification. Although this method is not as elegant as the COA or MOM methods, it is much more efficient to code and execute in assembler, and does provide sufficient 8-bit accuracy for this application.

Using the Mamdani max-min preposition is very conducive to the assembly code mnemonics in microcontrollers, as these operations can be conducted with only three main steps: two register load commands, and a comparison for size.

For the MAX operation, the equivalent is a union operation. This operation is given here in MC68HC16 assembler language.
Chapter 6: Software Design

<table>
<thead>
<tr>
<th>Assembler Command</th>
<th>Clock Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldaa first_number</td>
<td>6</td>
</tr>
<tr>
<td>dab second_number</td>
<td>6</td>
</tr>
<tr>
<td>cba</td>
<td>6</td>
</tr>
<tr>
<td>bmi b_is_larger</td>
<td>6</td>
</tr>
<tr>
<td>mva result_of_max</td>
<td>8</td>
</tr>
<tr>
<td>rts</td>
<td>12</td>
</tr>
</tbody>
</table>

b_is_larger (title of the following subroutine)

movb result_of_max                  | 8            |

rts                                 | 12           |

This is a total of 44 clock cycles per MAX operation, or $7 \times 44 = 308$ cycles for the first group of MAX operations.

The MIN operation is equivalent to the intersection operation. It is given here in MC68HC16 assembler language.

<table>
<thead>
<tr>
<th>Command</th>
<th>Clock Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldaa first_number</td>
<td>6</td>
</tr>
<tr>
<td>dab second_number</td>
<td>6</td>
</tr>
<tr>
<td>cba</td>
<td>6</td>
</tr>
<tr>
<td>bpl a_is_larger</td>
<td>6</td>
</tr>
<tr>
<td>mva result_of_min</td>
<td>8</td>
</tr>
<tr>
<td>rts</td>
<td>12</td>
</tr>
</tbody>
</table>

a_is_larger (title of the following subroutine)
movb result _of_min 8
rts 12

This is a total of 44 clock cycles per MIN operation, or $7 \times 44 = 308$ cycles for the first group of MAX operations.

At the end of this MIN operation, the resulting fuzzified value is then defuzzified. It is compared to the values for output, in this case, 4, and the two closest values are averaged.

Fuzzification and defuzzification of the values for each membership function is very computation intensive unless done with the aid of lookup tables. Without lookup tables, individual slopes would have to be calculated for each membership function—triangular and trapezoidal membership functions can be handled by simple division, but other functions can be quite complex. Of course, with look up tables, you trade accuracy for speed. For the purpose of this research, the accuracy using 8-bit lookup tables for the fuzzy inference engine is sufficient. This matches the format of the data and the A/D input signal accuracy, which is 1/2 of the LSB.

The output range for the controller is between decimal 120 to 135 (CHECK). This is due to the characteristics of the proportional valve used and the hydraulic system of the tractor used. The tractor used in testing was a Case International 7210 Magnum with a closed center hydraulic system. This type of system maintains constant pressure at all times, so will provide a consistent hydraulic control response regardless of the demands on the system.
6.3 Hydraulic, LCD Display and Data Logging Routines

6.3.1 HYD_CONT Routine

Once a fuzzy decision has been produced, the control signal is sent to the pulse width modulation unit A (PWMA) of the MPU. This modulates the duty cycle of a 128 Hz signal that controls two solenoid valves that control the position of the main spool on a proportional hydraulic valve. By having a constant 128 Hz pulse on the main spool, a flow of oil keeps the main spool moving. The motion is not enough to significantly change the position of the hydraulic ram on the seeder, but is essential to keep valve response time at an optimum level. The duty cycle of the two small solenoids that control the main spool are equal when the main spool is in the proper position. When the seed depth (disk position) has to be changes, the spool must be moved off center by changing the duty cycle of the tow small solenoid. This is done by changing the duty cycle of the signal from PWMA. The signal is split, with one signal controlling one solenoid, and the other, inverted signal controlling the other. Therefore, when the duty cycle is changed, one solenoid will open wider, with the other opening with an equal, opposite narrower opening. The difference in pressure moves the main spool one way or the other, thus acting as a hydraulic amplifier by allowing a large volume of oil to move from one side of the seeder hydraulic cylinder to the other.

6.3.2 Display_Depth Routine

Now that the control signal has been sent to the discer, the controller moves into the LCD depth display routine. The depth error value is compared to the soil surface value, and the difference is displayed on the LCD. The routine includes a subroutine that
converts the depth number to inches or centimetres and tenths of an inch or millimetres. Once all the numbers have been calculated, the LCD display routines are called to put each character on the screen. The unit preference can be selected by the operator.

![Fig. 6.4. Hydraulic valve detail.](image)

The Optrex display used in the controller has its own internal memory and display oscillator circuitry. As a result, any characters written to the display will stay on the screen until the “clear display” command is sent by the controller. In order to reduce display flicker due to unnecessary display writes, the code in this routine will check the contents of the display registers before writing a character to the display. This way, if the desired character is already displayed on the screen, the write routine for this character is skipped. By doing this, LCD flicker due to constant redraws is virtually eliminated.
6.3.3 Send_Data Routine

In order to subjectively test the performance of the controller, a serial communication data logging routine is utilized. Data read from registers in the controller is sent to a laptop computer for saving in a log file and later analysis.

Data is read from the ERROR, CHANGE_IN_ERROR, and OUTPUT registers in order that these calculations can be checked during program debugging to ensure that they are correct. These data allow cross-checking the controller fuzzy decision with the expected output from previous simulations. Also, if any changes are made in the field, the results of these changes can be recorded and monitored.

Sensor position information is important to ensuring that the ERROR and CHANGE_IN_ERROR calculations are correct. By having the unprocessed 8-bit ADC 0 and 1 readings over the time of the trial, the entire process of calculating ERROR, CHANGE_IN_ERROR, the fuzzy decision and final hydraulic valve output value can be evaluated. Parts of these data can be used as input values to the simulator in FIDE and the simulator written in Mathcad to test a new fuzzy algorithm. Likewise, the fuzzy output values from the field unit can be tested against the results of FIDE and Mathcad simulator calculations to ensure that this part of the program is giving the expected results.

The ADC 0 and 1 readings can be used in other Mathcad routines to test how well the controller calibration and operation subroutines work. These readings also can be used as a record of the ground contours at the test site.

The sister PC routine operates by checking for valid data being in the serial buffer, reading the data, and sending an acknowledge signal. Once received, the controller
Chapter 6: Software Design

increments an index, and sends the next data byte. Both the controller and the PC are synchronized to prevent data skew. In total, 10 bytes are sent per sampling instance. The PC logging program automatically formats the data for saving into an ASCII file for easy importing into a spreadsheet, mathematics or signal analysis program. As data are being collected, a bar graph display on the PC allows the operator to monitor the progress of the logging and control operations, thus detecting obvious problems early.

When the controller has finished the SEND_DATA routine, main program execution loops back to the CHECK_KEYS routine, to begin the next loop through the MAIN program subroutines.

The present system runs fast enough to log data 23 times a second. This is equivalent to one sample being taken every 7 cm (2 1/2 inches) at a speed of 10 km/hr (6 mph).

Several versions of the control software are used in different parts of the research. The program described here referred to the first version, where the operational purpose is to log data pertinent to controller operation, as well as control seeding depth. The second program version is intended for use when logging is not needed, such as when debugging controller code with the tractor and discer in a stationary position. In this version, the SEND_DATA program routine is commented out before compiling. Finally, an FFT routine for the MC68HC11 was adapted to the 16-bit MC68HC16Z1, for determining the operating frequencies of the front depth sensor.
6.4 Chapter Overview

A controller of this type takes as much effort producing the code for support routines as it does to write the core control algorithm. Calibration was especially challenging, as it necessitated a considerable amount of stationary testing with the discer to ensure that the readings were consistent and accurate. It was also discovered at this point in code development that a single sensor on the front of the implement would not be accurate, as it would not be able to give the position of the discs as they changed. A single sensor also would not be able to discern sinking of the discer into the soil from discer rocking motion over the axis of the discs and implement tires.

The hardware has been designated, as well as the software design defined. This now takes us to Chapter 7, Fuzzy Depth Control System Implementation, where these two design activities are combined to form the total system.
The emphasis in this Chapter is to explain how hardware, software and the process of simulation and verification combine in the formation of a system suitable for evaluating the suitability of fuzzy logic for the application of seed depth monitoring and control. In Section 7.1, the field test and data logging functions are discussed. Section 7.2 presents how the calibration routines work. Finally, Section 7.3 deals with how data is verified and new fuzzy routines simulated and tested. All HC16 software was written directly in assembly language to maximize execution speed. Software written for the PC was in Turbo C [Borl88], with simulation in FIDE script and within Mathcad.

7.1 Field Test and Data Logging Mode

As the purpose of the depth controller being described here is to test the suitability of fuzzy logic for the application of seeding depth control, it is important to have a means of collecting field data to use in software simulation, evaluation and design. The field test and data logging mode is used to control seeding depth and record observations and control parameters, thus, measuring actual field conditions and recording them for analysis.

This mode is used when operating as a depth controller and data logger. The sampling rate can be set to up to 24 samples per second, which translates to one sample every 12.35 cm at a speed of 12 km/hr.
7.1.1 *Operational Issues*

The controller box is constructed so that it can easily fit on the side of the cab of the tractor, within easy reach and sight of the operator. Power is provided by the cigarette lighter through a power cord constructed for this purpose. A power switch on the case operates in series with the power connection to the power supply module. A 5 ampere fast-blown fuse is also installed, to protect the board and tractor circuits from short circuits. Cables to the sensors and the hydraulic valve solenoids are connected to the controller on the side of the box facing the rear of the tractor. Cables were run through the rear of the tractor cab.

Some ergonomic issues were taken into consideration in the design of the control box even though it was constructed for research purposes, and would not have many hours of use in the field. The interface for the controller was made to be simple, in order that the design met the criteria for making operation as safe and non-distracting to the operator as possible. One reason the controller box was constructed as small as possible was to enable it to be easily placed beside the operator without impeding operator movements. The operator is able to see the display while turned partially backwards when watching the discs. Likewise, the box could be fitted over the dashboard above the steering wheel, allowing the operator to see the display without turning or looking to the side, away from the direction of travel. If the controller box was cumbersome to use and obstructed the operator's movements in the cab, this would pose a serious distraction for the tractor operator and pose a safety hazard to anyone working behind the tractor while it was running.
7.1.2 Field Operation

On powering up, the HC16 finds the initialization vectors starting at location $00$, goes through the module and memory initialization steps, and enters the main program loop. Upon entering the SEND_DATA routine, the HC16 copies the parameters to be logged to temporary locations. These include the values for the 8-bit format sensor position readings from ADC 0 and 1, the 16-bit quantities for error and change in error, and finally, the 8-bit fuzzy decision output.

As there are no hardware handshaking lines, serial communication relies on passing handshaking characters between the PC logger and the HC16. The HC16 enters a loop where the Receive Data Register is checked for new data. If the Start signal ($SF$) is received from the PC logger, the routine continues. If there is noise on the line or the PC logger is out of synchronization, $SF$ will not be received and the HC16 goes back to the wait loop for the correct character. Therefore, in order for the HC16 to work, it must be connected by a null modem cable to the PC logger computer. The controller must be started first, then the PC logger program. Using a specific start character eliminates the possibility of data being written to the log file that is out of the correct sequence. Of course, as the log file is formatted in a certain way, if information was out of sequence it would be obvious once this text format file is inspected.

Once the Start command is received by the HC16, it then sends the first data in byte format. After each byte, the HC16 checks the Receive Data Register for the character "E", which is the Continue signal from the PC logger. In this way, the data is sent from the HC16, saved by the logger program, and an acknowledge signal is sent to the HC16 by the
logger when it is ready to receive the next character. Once this process has been repeated 6 times, the HC16 returns to the top of the main program loop of the controller code, and continues until execution again reaches the SEND_DATA routine, where the Receive Data Register is again checked for the Start character.

While the HC16 is executing code for the other routines in the main program loop, the PC logger program reformats the data received for storage, stores it to the log file, and writes a bar graph line for each data item to the PC screen. The bar graphs are a graphical representation of the processes occurring in the fuzzy logic controller. Extreme conditions can be spotted immediately, and the controller stopped for code modification and checking. The data logger can be stopped by pressing the Escape key on the PC. The HC16 will continue execution until it reaches the Receive Data Register test loop, where it will continue looping until the logger is restarted. This way, the controller can be stopped and restarted without having to reboot.

There is also a version of the HC16 controller code that has the data logging routine commented out. This is used for initial installation and check-out of the system when it is being installed in the tractor and hooked up to the sensors and the hydraulic valve.

![DEPTH 2.1 INCHES](image)

Fig. 7.1. LCD interface display in operation.
Chapter 7: System Implementation

The LCD interface is very simple. While the controller is operating, the LCD displays the depth in the units that are chosen by the operator. It is bolted to the top lid of the control box above the keypad. A 20 cm ribbon cable connects the pc board bolted to the box lid to the controller module main board. The controller box can be tilted toward the operator if the angle of the LCD is too high to see it properly.

![Keypad functions diagram](image)

Fig. 7.2. Keypad functions.

The alpha keys on the keypad are used for initiating calibration and control of the depth control operations. The "A" key starts the set raised height and modes subroutines. The "B" key is for raising the discs, while the "C" key is used to start the subroutine for setting the seed depth. The "D" key is used for starting the process of setting the ground position. The "E" key is used for initiating calibration, while the "F" key is used to lower the discs.

By hitting any key except the "F" key, the controller stops operating as a seed depth controller and enters the chosen subroutine. The key functions were chosen based
on how an operator would intuitively remember the functions. For example, the "Raise Discs" command key is located above the "Lower Discs" command key.

The keypad type chosen is a membrane keypad with raised keys. This keypad requires some effort on the part of the operator to activate individual keys, thus reducing accidental function activation.

The most common functions were located on the edges of the keypad. For example, the raise discs command was located on the right side of the keypad, so the operator can operate it with their thumb with the rest of the hand steadying the thumb by resting on the right side of the box. Steadying the thumb is important, since this button would be pressed while the tractor is in motion over the field. The position also reduces the likelihood that the operation of the controller would be interrupted by hitting the wrong key, as the number "7" key above the "B" will have no effect, and the "F" key below it will not interrupt controller operation, either. If the "A" key is hit accidentally, the controller will stop operation, but quickly hitting the "B" key will cause the controller to recover and raise the disks. This error would not be likely to occur often, as the operator must stretch the thumb farther away from the right edge of the box. The "F" key is located on the right hand lower side, so that it is easily found and operated when the operator wishes to drop the discs and resume operation.

While seeding, the operator need only use two keys: the UP position key for raising the seeder, and the DOWN position key for lowering the discs to operating position. Setting the disc raised position provides the UP key subroutine with the desired height for the discs to be raised above the ground when actuated. The DOWN position key
releases the control software from a WAIT loop (initiated with an UP key press) that returns the program to normal seed depth operation. When the discs are in the lifted position, it displays "DISCS RAISED" and will lower them by hitting the "F" key on the keypad. This command directs the fuzzy controller to resume normal operation, so the discs descend to the depth set point.

Two main parameters to measure are the shape of the topography of the field (S), and the characteristic rates of change as one moves across the field (dS/dt). A data sampling function in this mode samples both the discer sensors at a relatively high speed (24 Hz) in order to provide accurate profiles of the soil surface for use in simulation of the control system in the lab. The sample data are collected by the control module, and transmitted asynchronously to a portable computer serving as a data storage unit. This provides a time-domain waveform that is used as the perturbation input in system simulation and for analysis of how the controller responds to different inputs.

The rate of change in field topography is important to know to ensure that the sampling rate used in the depth control operating mode (normal operation mode) satisfied the Nyquist sampling theorem. This states that in order to prevent errors due to aliasing, the sample rate must be at least double the rate of change of the signal. In actual use, this sample rate is set to about 1.6 times the rate of change of the signal. An HC16-native, 16-bit fast Fourier transform (FFT) was written for this purpose, based on the FFT produced by Ron William of Ohio University for the MC68HC11, which in turn is based on the FFT written by Richard Lord [Lord79]. Output is sent to a portable computer, which logs the data and provides a real-time graphical display of the results. The FFT used takes
advantage of the 16-bit operating capabilities and digital signal processing commands of the HC16.

7.2 Calibration Mode

The operating mode also includes the keypad functions for calibration and control. All of these functions are conducted while the tractor is stopped. As the LCD is only 16 characters wide, where messages encompass more than 16 characters, each line of the message is displayed with a pause of about one second between lines.

Fig. 7.3. Calibration functions - 90 degree setting and theta correction.

In the operation of the controller, the first operation to be conducted is the calibration operation. This ensures that all subsequent sensor readings are interpreted correctly. The three control parameters described here include setting the raised position of the disks, setting the operating depth (the depth set point), and the field surface level. Raised level and operating depth are programmed by pressing numeric keys when
prompted to do so. These parameters can be set in either metric or imperial units. The units are chosen as a sub-menu of the raised discs level setting. Field surface level is found by lowering the disks to the ground, then pressing a key to store the A/D value directly to static RAM as a variable. Once these three values are set after calibration, the depth controller is ready to operate.

7.2.1 Keypad Functions

The keypad functions include one key used to start calibration routines, three keys for setting parameters, and the two previously mentioned used in controlling the height of the discer.

The "A" key starts the subroutine for setting the height of the discs when lifted, and choosing the operating modes.

When the "A" is pressed, the controller displays the message "Choose Function: Press 0 to pick sensor modes, 1 to set seeder raised position, 2 to set units". Pressing the "0" key causes the controller to display "Choose Scan Mode Press 0 for dual sensors, 1 for single sensor". Pressing "0" at this point causes the controller to determine disc position by sampling both the front and rear sensors (this is also the default mode, and is part of the controller initiation settings on initial start-up). Pressing "1" sets the controller to use sensor data from the front depth wheel sensor only. This mode is used in studies where the frame height above the ground is the only parameter desired, as in determining the amount of wheel sinkage on a given field. Once either the "0" or "1" keys are pressed,
and the controller finishes writing to the respective register, the controller resumes depth control operation.

Pressing “1” to set seeder raised position, the message “ENTER INCHES: “ is displayed. The operator now presses a numeric key to represent the inches raised. The controller then displays the message “ENTER TENTH INCH”. Once the operator has pressed a numeric key for tenths, the controller resumes depth control operation.

Pressing “2” for setting units results in the display “0 for imperial, hit 1 for METRIC”. Choosing “0” sets the units to imperial, and the displayed depth is shown in inches and tenths with the word “INCHES” on the LCD display. Choosing “1” sets units to metric, and the units are displayed in centimetres and millimetres, with “CM” on the LCD.

The “C” key triggers the subroutine for setting the seeding depth. The LCD displays the message “SET SEED DEPTH”, then “HIT ANY KEY”. Once the operator has pressed a numeric key, the LCD shouts, “ENTER INCHES:”. When the operator presses a numeric key, the value is stored as the inch setting. The LCD then displays, “ENTER TENTH INCH”. Again, when the operator presses on a numeric key, this value is saved as the tenth of an inch setting. Once this has been done, the controller resumes depth control operation.

The “D” key initiates the “set ground level” subroutine. The LCD displays “SET SOIL SURFACE” message, followed by “HIT ANY KEY” several seconds later. Once the operator has pressed a numeric key, the message “LOWER TO GROUND” appears, followed by “HIT KEY TO STORE”. The operator manually lowers the discs to the
ground on the field, using the hydraulic levers. Once on the ground, the operator presses a numeric key to store the value as the “Zero” depth position, or ground surface. Once this is complete, the controller resumes depth control operation.

The “E” key starts the calibration subroutines. For calibrating both the front and rear sensors, the implement is moved to a firm, level surface where the disks do not sink into the ground when lowered. The first variable to be set is the point in which sensor linkage arms are 90 degrees to the depth linkage arms. Finally, the ground positions of the sensors, and their positions with the disks at 6 inches above the ground are recorded. This results in a characteristic calibration factor being calculated for the specific sensor setup, that is used to provide the correct depth in normal operation.

When the “E” key is pressed, the controller prints “Choose Function: Press 0 to set sensors 90 deg to linkages, 1 to cal disk sensor, 2-cal front sens” to the LCD display. If the operator presses “0”, the message “PROCEDURE TO STORE THE SENSOR POSITION 90 DEG TO DISK LINKAGE” is displayed, followed by “HIT ANY KEY”. Once the operator has pressed a numeric key, the message “Position disk so sensor arm is 90 deg to disk arm” is displayed. This is followed by the “HIT KEY TO STORE” message. The positions of the rear and front sensor arms are then set 90° from the respective connecting arms to the discs and front sensor. When the operator presses a numeric key, the values in the sensors are stored as the position where the sensor arms are perpendicular to the connection arms to the disc gangs and front depth sensor assembly. Once the values are stored, the controller resumes seed depth control operation.
Chapter 7: System Implementation

Pressing the "1" key will start the subroutine for calibrating the disc sensor. The controller displays the "DISC SENSOR CAL" message on the LCD. This is followed by "ON LEVEL SURFACE", then "HIT ANY KEY". The operator presses a numeric key, and the "LOWER TO SURFACE" message is displayed, followed by "HIT ANY KEY". The operator then lowers the discs to the ground manually using the hydraulic levers, and then presses a numeric key to save the value. The message "RAISE DISKS UP SIX INCHES" is displayed, followed by "HIT ANY KEY". The discs are lifted manually, and the distance from the ground to the bottom edge of the disc is measured. The discs are then raised or lowered until they are 6 inches above the ground. To ensure that the height reading is not in error due to sagging of the disc gang when above the ground, make the measurement at the centre of the disk gang.

Once the discs have been set to 6 inches, the operator presses a numeric key, and then the "HIT KEY TO STORE" display is written to the LCD. Once the value is written to memory and the respective calculations have been made, the controller returns to seed depth control operation.

Pressing the "2" key will start the subroutine for calibrating the front sensor. The controller displays the "FRONT SENSOR CAL" message on the LCD. This is followed by the "ON LEVEL SURFACE" and "HIT ANY KEY" LCD messages. The operator presses a numeric key, and the "LOWER TO SURFACE" message is displayed, followed by "HIT ANY KEY". If the front sensor is on the ground, a numeric key is pressed to save the value. The message "RAISE DISKS UP SIX INCHES" is displayed, followed by "HIT ANY KEY". The sensor is lifted six inches on measured blocks.
Once the sensor wheel has been blocked up 6 inches, the operator presses a numeric key, and then "HIT KEY TO STORE" is displayed on the LCD. Once the value is written to memory and the respective calculations have been made, the controller returns to seed depth control operation. Once sensor calibration is complete, the parameters for operation can be set.

7.3 Chapter Overview

The operation of the fuzzy depth controller has been designed to take advantage of the combination of a simple 16-key keypad and the easy programmability of the LCD display. Each of the six alphabetical keys on the keypad control a separate depth controller set of functions.
CHAPTER 8

DISCUSSION AND ANALYSIS OF FUZZY LOGIC BASED

SEEDING DEPTH CONTROLLER

Increasing depth control accuracy is an important means of maximizing crop returns without additional input costs or time. The limitations of common seeding depth control methods and the suitability of fuzzy control for this application have been presented in previous Chapters. A proof of concept controller design was built using the MC68HC16Z1 microcontroller. Testing of operating functions has been completed.

8.1 Fuzzy Control

The fuzzy controller is a Mamdani-type controller, based upon the max-min composition. The Mamdani type of fuzzy controller, first introduced 20 years ago [KiMa77], has been applied to a variety of industrial processes. For example, it has been used in heat exchangers, cars, and washing machines [KiPe90, Mura84]. Thus, fuzzy control is gaining in popularity in control applications that use nonlinear, time varying inputs. Seeding depth control also presents this type of problem.

In utilizing fuzzy logic in the decision process of the seeding depth controller, the state of the seeder position is described using linguistic labels. The fuzzy membership function mA(x) describes the extent to which the adjusted value x of the two sensor inputs belong to a specific linguistic input classification describing the error from the set point. Similarly, mB(y) describes the degree to which the difference between the depth reading
seven samples previously and the present reading belongs to a specific linguistic classification for the change in error (seven samples represents about 1/3 second for the data logging controller, and about 1/7 second for the non-data logging controller-using adjacent samples provided very little difference in error). For instance, the measured error from the depth controller set point input may be classified by its degree of membership in the linguistic classification, Negative Medium, and the error change from the previous depth reading classified by its degree of membership in Positive Small (see Fig. 8.1).

![Table](image)

**Fig. 8.1.** Fuzzy control rules for 7 membership function algorithm, from FIDE simulation.

A list of linguistic control rules were formed based on the fuzzy relation. Each rule is based on the advice from an expert human operator. Linguistic variables are set up in the manner illustrated in the following example.
Chapter 8: Discussion and Analysis

IF <INPUT VARIABLE #1> IS <ANTECEDENT> AND <INPUT #2> IS <ANTECEDENT> THEN <OUTPUT VARIABLE> IS <CONSEQUENT>

where

<INPUT VARIABLE #X> are the input variables being monitored;

<ANTECEDENT> are the input fuzzy membership functions;

<OUTPUT VARIABLE> is the controlling signal; and

<CONSEQUENT> is the fuzzy output membership function

In the first example for this research, seven linguistic variables for Error, and seven linguistic variables for Change in Error were constructed in the following form.

IF ERROR IS NB AND ERROR CHANGE IS PB THEN CHANGE IN DEPTH IS NM

ELSE

IF ERROR IS NM AND ERROR CHANGE IS PB THEN CHANGE IN DEPTH IS NS

ELSE

.

.

.

.

49th rule.

A total of 49 rules constitute the summary of the expert operator’s experience in a form usable by the fuzzy logic controller. The fuzzy depth control algorithm is therefore defined by these rules. The control action is determined by the max-min product of the
depth reading input \( x \) with the fuzzy depth control algorithm, \( D \), denoted by \( x^* D \). The fuzzy depth control algorithm \( D \) consists of fuzzy relationships between the inputs \( x \) and \( y \), and the control signal output, \( z \). Additional tests with 25 rules and 9 rules were run with 5 membership functions for error and change in error, and 3 membership functions for error and change in error, respectively.

The discrete numbers representing the error and change in error are fuzzified. This means that the discrete input values are classified as being members of given fuzzy membership functions if their values fall within a given membership functions' range of values. The degree of membership in each given membership function is given by a number between \([0,1]\). This is also said to represent the degree of membership in the respective fuzzy set. The algorithm is based on the max–min composition of these values, represented by the fuzzy control rules. For example, when \( X \) is the degree of membership of a given depth input in negative medium, and \( Y \) is the degree of membership of the corresponding change in error in positive small, the fuzzy phrase \( P \) describes the min of the inputs, or \( X \times Y \)

\[
mP(x,y) = \min\{m-\text{ve medium}(x), m+\text{ve big}(y)\} \tag{8.1}
\]

According to the fuzzy rules presented in Fig. 1, \( mP(x,y) \) describes the membership of the min of the input variables (\( x,y \)) in the output variable negative small. Mathematically, this can be represented by \( m-\text{ve small}(x,y,z) = X \times Y \times Z \), denoted by fuzzy phrase \( Q \)
\[ mQ(x,y) = \min \{ mP(x,y), m-e \text{ small}(z) \} \] (8.2)

The max of all the fuzzy phrases QN gives the fuzzy value S of the output signal, where N is 1 to 49, 1 to 25, or 1 to 9, corresponding to the number of fuzzy rules in the controller

\[ mS(x,y,z) = \max \{ mQ1(x,y,z), mQ2(x,y,z), \ldots, mQ49(x,y,z) \} \] (8.3)

The value, \( mS(x,y,z) \), is then defuzzified using the center of gravity (COG) approach. This approach determines the output using the weighted average of singleton output functions. The resultant output signal voltage is then applied to an electrohydraulic proportional valve.

8.2 Simulation

In preparing for simulations, the objective was to approximate the decisions of an expert operator as closely as possible. The first step was to form the fuzzy control rules. These provided the reasoning for the actions, and serve as the implementation of an operator's actions. A combination of the "conventional" and the "template" method was used. The next step was to determine the control parameters based on the mechanical characteristics of the plant to be controlled. The maximum height and depth the discs could obtain are the limits of movement and the limits the membership function Error (about half this total distance negative, and half the total distance positive) could obtain. In a similar fashion, the working depths of different seed types were considered and compared to the maximum movement of the discs in order that the best membership function shapes could be determined. Finally, the maximum speed of movement of the front sensor and disks was considered, in order that the sampling rate was fast enough to
satisfy the Nyquist sampling theorem. By observing the motion of the discs and front sensor, this was deemed to be about 75 cm/second. This provided the information needed to put together the membership functions for Rate of Change. Therefore, it was possible to assess the performance of fuzzy logic control algorithms by comparing results to the desired output.

The values for the fuzzy controller were then simulated, using Manifold Editor (Version 1.8.01-100049, Fuzzy Systems Engineering, San Diego, California, USA), Mathcad (Version 6.0 - MathSoft, 101 Main Street, Cambridge, Massachusetts, USA 02142) and FIDE (Fuzzy Inference Development Engine, from Aptronix, CA, USA). Manifold Editor enables the user to graphically edit membership functions, set input values, and view output values given different input values. Mathcad is designed to operate like a clean sheet of paper, where the user enters equations and variables just as they would with pen and paper. Data files can also be imported, so once field readings were available, these were imported and processed. FIDE is meant to be a complete fuzzy logic development tool, with graphic membership function editing, simulation functions, viewing of control surfaces and generation of 8-bit code for several microcontrollers.

In using Manifold Editor, initial input parameters for membership functions were entered, then analyzed using the Manifold Walker. As each input was changed, the resulting output value was noted and recorded. Output values were compared to the desired values.

Output values for this application must be monotonic, in order to preserve the stability of the controller. Output values of the Manifold Walker were graphed to ensure
monotonicity. Minor adjustments were made to the input and output membership functions to ensure this. The fuzzy membership functions and their final values are indicated in Figs. 7.2, 7.3 and 7.4.

Fig. 8.2. Manifold Editor input membership function example - 7 membership function algorithm. Error on left, Change in Error on right.

Fig. 8.3. Manifold Editor - output membership function for 7 membership function algorithm.
Fig. 8.4. FIDE simulation screen example: input and output membership functions for the 7-membership function algorithm.

The membership functions in “Error” (Fig. 8.2) refer to the degree of error between the actual depth of the implement to the desired target depth (the set point = 0). The full scale readings vary from −15 cm (15 cm below the set point) to +15 (15 cm above the set point). In the “Change in Error” membership functions (Fig. 8.2) are the rate of change in between the depth sensor readings is measured, in terms of cm/second.
Chapter 8: Discussion and Analysis

Fig. 8.5. Input membership functions for change in error from previous depth reading.

The output membership functions, in “Depth Adjust” (Fig. 5), refer to the distance the implement must move in order to approach the set point. This is expressed in terms of \(-\text{cm}\) (distance to move down) and \(+\text{cm}\) (distance to move up).

Fig. 8.6. Output membership functions for depth adjustment.
Once the values of the output were recorded from the Manifold Walker, a surface plot was produced to give a graphical representation of the fuzzy controller output (Fig. 8.8).

![Fuzzy control surface](image)

Fig. 8.7. Fuzzy control surface (from a FIDE simulation). The X axis represents positive 64 mm to negative 64 mm change in error, the Y axis representing positive 125 mm to negative 125 mm error from the set point, and the Z axis the change in output of -15 cm/second to 15 cm/second.

The control surface shows the monotonic control characteristics of the fuzzy controller. The surface near the zero error and zero change in depth point shows a gradual positive slope, where control signals need to correct the depth a small amount. Areas closer to the extremes of the control surface have a higher rate of change in the slope. By inspection, no part of the curve is seen to take on a reciprocal slope, therefore monotonicity is ensured.

Mathcad required complete programming of the fuzzy control algorithms. It also allows the user to use programming functions, including loops, iterations and if-then statements. The variety of graphing functions also made it possible to instantly see the effect small algorithm changes had on fuzzy control, as a two dimensional output result and as a three dimensional surface plot. As it was possible to produce a variety of output
results quickly while knowing exactly what calculations were taking place to produce them, this was the tool in which most simulation was done. It was more time consuming than other tools but had the important advantage of having the flexibility to modify the fuzzy control algorithm and analyze results statistically.

A 7-membership, 5-membership and 3-membership function fuzzy controller was set up in Mathcad to compare the results. Comparison of results included several parameters. One comparison made was the granularity of data. The fineness of control was compared in the three regimes. Obviously, the minimum control achievable on output is one unit in 256, or 8-bit control. The minimum amount of each membership function output was compared to this.

Another criteria for comparison was how well each regime fit into the requirements for the fuzzy depth controller. For example, in the 3-membership function, was necessary to reduce the slopes of the Zero function to the point it was no longer possible to achieve fast response time on extreme Error readings, while maintaining fine (slow moving) control when there is very little error. A well-designed fuzzy controller should be able to quickly move the discs to the set depth, without overshooting the target. In conventional seeding depth controllers, lack of overshoot is often attainable only at the expense of speed and/or accuracy by the use of a “dead band”. It is worthy to note here that as the slopes of the membership functions of a fuzzy controller becomes steeper, the controller comes closer and closer to approximating the actions of a conventional, non-fuzzy controller. The 5- and 7-membership function algorithms both provided sufficient speed of action with accuracy (granularity) of control for this application.
Chapter 8: Discussion and Analysis

Work with FIDE dealt mainly with the assembly code producing capabilities of this program. Membership functions were set up and verified, then the assembly code produced. The version of FIDE being used here produced assembly code for the MC68HC11. Although this is 8-bit code, the MC68HC16 compiler MASM16.EXE will properly compile this code for use in the HC16. The code to be tested in the controller hardware is placed in the fuzzy controller assembly code, and the respective variable names adjusted and memory space allocated to allow it to work with the rest of the code. FIDE version 1.0 uses only singleton output functions. This is appropriate, given the characteristics of the hydraulic valve. Although the valve is proportional, the speed of operation does not vary considerably once the PWMA output exceeds 25 hex units from center, or about one fifth of the total PWMA output possibility is usable in this way.

8.3 Lab Testing

Once a new algorithm would be produced in FIDE, it was compiled with the HC16 controller code and tested in the MC68HC16EVB evaluation board laboratory test unit. The sensors were taken through their entire range to observe the depth reading results on the LCD display. An oscilloscope was also used to examine the PWMA output waveform as the inputs were varied to the controller. Once it was determined that code produced a monotonic control action and approximated what was needed to work in the field, the lab unit would then be connected to the actual disk sensors and hydraulic proportional valve. A tractor connected to the discer supplied hydraulic pressure for the testing. In order to simulate a "field-zero" condition, the discer was operated in one spot, but the "zero"
position would be set several inches above the ground in order that control on both sides of the set point could be monitored.

Without the damping effect of the soil, it was found that the discs would oscillate when using the 7-membership and 5-membership function algorithms. Oscillation was stopped by scaling the membership algorithm outputs down 50% in speed. It was found that once oscillation began, the resonant frequency of the discer would be obtained quite easily, as it rocked back and forth on two balloon tires. The tires are located about 10 feet from the other, so the result of this distance was a lever action using the implement frame that tended to magnify the rocking motion of the discer. Thus, with a combination of large rubber tires suspending several tons of steel discs and associated hardware, oscillation was very easily achieved by several quick movements of the front sensor. If this work was to use a conventional, "bang-bang" hydraulic valve that opened and closed at full throttle, the oscillation problem would only be defeated by using a valve that would be too small to move the discs quickly when the error or rate of error change was high. Cascading several valves would also work, but would add complexity in the hydraulic valve plumbing and lead to increased possibility of leaks, complexity for servicing, cost and increased risk of failure due to a higher number of electric solenoid valves and associated wiring.

8.4 Field Testing

Once several fuzzy control algorithms controlled the disc position well, they were tried under field conditions. The lab MC68HC16EVB control unit was suitable for use on a table in the laboratory and stationary testing, but would serve to be impractical for use in the field when the tractor was in motion. An umbilical setup could have been set up for use
with a vehicle following the moving disc, but a much simpler tractor cab mounted unit was constructed in a small aluminum box for testing and logging of results. Shielded control wires connected to the sensors and the hydraulic proportional valve were run into the cab, and securely connected by amphenol-type shielded screw connectors. A shielded aluminum box enclosed the controller electronics, with shielded compartments for the power supply and hydraulic signal switching. Power was supplied to the unit via the tractor's 12V cigarette lighter socket.

![Fig. 8.8. Field data logger status display screen.](image)

In order to facilitate logger operation and ensure it is functioning correctly, a screen display was produced that graphically indicated the current readings being received from the controller (Fig 7.9). In order that it would function on any PC, and not just a
Windows-3.1 or above capable machine, the software was written in Turbo C, with graphics set up for use in MS-DOS.

The screen was set up with a simple interface for easy interpretation of the state of the controller. The text across the bottom is as follows: Error, Error Chg (change in error), Fuzzy Decision and the sensor position readings Fnt Sensor (front sensor, on the gauge wheel) and Dsk Sensor (the rear sensor, on the disk gang). Short lines on the vertical access of each screen parameter represent the median positions of error, change in error, and fuzzy decision when error and change in error are zero. For the disk sensor positions, the median line represents a hexadecimal reading of $7F$ or 127 decimal, the center of the sensor range. The line at the top represents $FF$, or 255, with the bottom axis of the graph representing 0.

The PC logger graphic display proved to be invaluable in field testing. In some of the tests, an incorrectly wired sensor was causing a pronounced damping effect on the level of error, producing values that were much below the expectation. The deficiency was spotted immediately with the display values, and the changes made. In another example, a sensor was not clamped in the center of its total operating throw. This meant that the sensor would read zero from time to time, resulting in incomplete data and erroneous control action. Sensor connections were loose in one experiment, resulting in wildly changing output values. Steps were then taken to eliminate this effect. If the logger had not had the display, it would have led to further delays in development of an operating controller, since problems would not have been apparent until the logged data was graphed.
Chapter 8: Discussion and Analysis

A number of different trials were run with the 7 membership function algorithm, on unworked stubble, soil disced once and soil disced twice. Tractor speed was a constant 6.8 miles per hour, typical for seeding. It was found that the low-pressure tire of the front sensor followed the soil contours very well, but would be easily lifted up by piles of straw larger than 1” thick. Also, if there were intermittent piles of straw within several feet of each other, the resonant vertical motion frequency of the sensor would sometimes be obtained, leading to steady bouncing of the sensor up to 4” off the ground. In this case, the algorithm with the speed of the discs dampened by 50% was unable to move the discs fast enough to follow the bouncing, so the discs actually remained quite stable as the front sensor bounced across the set point.

Data was collected giving the error, change in error, fuzzy decision and the sensor position readings from the A/D converters. This way, the sensor readings could be converted to inches, and the action of the fuzzy control algorithm checked against what was expected.

![Diagram](image)

Fig. 8.9. Portion of data output file showing data format for PC logger.
A sample of the data collected is given in Fig. 8.9. A time domain plot of the sensor position over time is also given in Fig. 8.10. Error, change in error, and fuzzy control output is given in Figures 8.11, 12, and 13, respectively.

![Error, Chg in Error and Output](image)

**Fig. 8.10.** Time domain plot of the sensor positions.

The FFT of the data is given in Fig. 8.13 and Fig. 8.14. This shows that there was a considerably higher amount of movement in the front sensor, with most motion to about 5 Hz, with a smaller amount to the full measurable bandwidth of 9 Hz. The disc sensor reading shows a much different scenario, with the majority of action to about 2 Hz. The readings in the graph were taken at a sample rate of 24 samples/second.

In an excerpt from Trial 1 Fig. 8.14, the error over time show that the discs had peaks up to 5 inches from either side of the set point, with the greatest variation due mainly to sensor bouncing on lumps of stubble in the unworked stubble field portion of the
test. However, error peaks of several inches are expected in the field trial, since the soil topography will change rapidly as the discer moves across the field.

![Error and Sensor Position Relationship](image.png)

**Fig. 8.11.** Example plot of error and sensor position for 100 samples from the PC logger data file for the 5 membership function algorithm.

![FFT of Front Sensor](image.png)

**Fig. 8.12.** FFT plot from PC logger, front sensor, for field test 5.
Chapter 8: Discussion and Analysis

Fig. 8.13. FFT plot from PC logger, disc sensor, for field test 5.

Fig. 8.14. Error plot from the 5 membership function algorithm, field test 5, with mean, standard deviation and variance measurements.
Fig. 8.15. Field trial 5 error from set point, over 1400 samples (78 seconds), with statistical measurements for these data.

Figure 8.15 shows the results of the field trial. The units for error, FLD_ERR_n, are in decimal, and are the difference from the set point to the current depth position as read by the two A/D converters. Each decimal unit corresponds to one hexadecimal unit for the A/D readings, thus the range of the graph is from 60 to -60, representing the hexadecimal values 7C to -7C actually recorded by the PC logging program. The approximate number of samples used in this calculation are 1,450, representing approximately 78 seconds of sampling time over both unworked stubble, stubble disced once and stubble disced two
times. These areas are representing different soil topographies and discer sink rates into varying degrees of support from the soil. The unworked stubble represents very solid soil, in which the tractor and discer sink only about 1 inch, with level soil and 1-2 inches of intermittent straw cover. The area disced once represents an area with soil ridges 4-6 inches deep, with tractor and discer sinking about 3 inches. The area disced twice has a more even topography, with soil ridges about 2-3 inches deep, but with a higher discer and tractor sink rate, about 4-5 inches.

The mean of the error over these different regions was 0.954 of a hexadecimal A/D unit, or about 1.0 units considering two significant digits due to sampling accuracy. This represents an error overall of approximately 0.10 inches (2 mm), based on the calibration value of (decimal) 14 hexadecimal units per inch in the calibration sequence. The standard deviation for these readings was 15.451 units, or 1.1 inches (28 mm). Variance was 238.784, which is the square of the standard deviation.

The significance of these results is that although the sensor was moving across the set point a significant amount, with peaks as many as 4 inches (10.2 cm) from either side of the set point, the seeder was able to maintain a mean error within the 1 cm goal set in the criteria for the controller to operate with adequate accuracy for seeding depth control.

8.5 Chapter Summary

In this Chapter, results from laboratory simulations and field results were presented. While stationary testing indoors in the Gamby farmstead workshop and in the field, minor code revisions were made with the aid of the battery-backed RAM module. However, after each day in the field testing a major variation in an algorithm or controller
code procedure, major rewrites had to be conducted off-site. Extra testing of different fuzzy control algorithms, and possibly an improved PC logger routine for faster operation could have been conducted if it was not for the onset of winter 1996. Field results obtained are only for the 5-membership function algorithm, and although it appeared to work quite well for this operation, additional testing of variations to fine-tune this code and compare it to other membership functions would have added further interesting results for comparison in this research. In all, however, the objective of this work was to test the suitability of fuzzy control for the application of seeding depth control with one-way discer seeders, and this objective was achieved.

In Chapter 9, Conclusions and Recommendations outline the contributions of this work, and present thoughts on future work based on the research in this thesis.
CHAPTER 9

CONCLUSIONS AND RECOMMENDATIONS

Improving seed depth control can improve crop performance. The most common seed depth control methods and the limitations of their accuracy were described. Fuzzy logic controllers are well suited for control applications like seeder depth control, where the inputs are nonlinear and time-invariant. A model of a seeding depth controller utilizing fuzzy logic has also been described. Simulation results of the model were presented in the form of the final structures of the membership functions and the resulting control surface.

9.1 Performance of the Sensors

Field conditions can affect the performance of the depth sensor wheel. It was found that compacted straw on the field from the previous harvest was further packed by the action on the wheel, but that the reading would be for the disc depth was shown to be deeper than it actually was. As a result, the disc would be lifted by the erroneous amount. The fuzzy algorithm did tend to low-pass filter small imperfections in depth, such as that caused by wheel tracks in the field. However, the “very large” membership functions did cause the discs to move quickly when there were large changes in depth, such as in drainage ditches and wide ruts. The integration actions of the depth gauge tire, mass of the sensor and time for the hydraulics to move the disc depth control cylinder also add enough time lag that the discs can adjust to changes in depth 2 m ahead, at the wheel sensor.
The front wheel sensor essentially gave a reading of the amount of main wheel sinkage and discer swivel as it crossed various soil conditions - soil that is previously cultivated and loose, soil compacted by large agricultural vehicles, and soil with varying moisture contents. In soil with straw cover less than 70% [reference on measuring straw cover] the effect of straw on the depth reading was not significant. Different weights of the sensor, as well as different tire pressures and even the use of a fluid-filled or solid rubber tire could also be tested for immunity from straw and soil clump "noise".

9.2 Hardware Observations and Recommendations

In actually building and testing a field data collection version of the fuzzy seeding depth controller, it was found that the construction of the device was critical. Any wires within the unit that were too short or under excessive pressure soon came loose or became disconnected in the vibration of the tractor cab. It is necessary to ensure all connectors lock into place, and are securely soldered to prevent separation in the connectors. Likewise, the original sensors had contacts that easily corroded if not completely covered in storage. It was necessary to clean the corrosion from both the contact ends to prevent periodic signal interruptions due to vibration and shock as the discer moved across the field. It was also necessary to ensure all wires to the discer were properly secured to prevent dragging and catching on the ground.

The proportional valve has a variable speed setting that varied the rate of oil flow through the main spool. It was found that at the high rate, the discs would respond much more crisply to controller commands. However, if the discs were raised out of the ground, thus eliminating soil resistance to movement, the discs would oscillate across the set point
until they contacted the ground. This was noticed using the 5-membership function algorithm and the 7-membership function algorithm, without 50% gain reduction. At that point, the ground would slow the disc gang movement, and stop the oscillation.

9.3 **Algorithm Development Recommended Improvements**

There are three additional techniques which would have sped up algorithm development: the use of an analog simulator, use of a hydraulic valve and ram driven by a motorized hydraulic pump, and the use of a soil box.

An analog simulator can be built out of a combination of RC circuits and operational amplifiers, each element of which would model a different part of the controller and discer system. Sensor inputs would be the combined voltages from simulator feedback and an added perturbation, to simulate the changing soil contours. The action of the wheel sensor assembly can be simulated as a low-pass filter, with only changes above the minimum characteristic frequency of the sensor passing to the controller. Likewise, the integration action of the hydraulic hoses, hydraulic proportional valve, hydraulic ram and the soil itself can be modelled with op amps. Thus, an output voltage would feed through the simulator, with the output (representing the changing disc position) feeding back to the sensors as a slowly changing signal, at the characteristic rate of change of the plant being controlled.

In this research, hydraulic control routines were tested by first travelling to the discer’s location in a heated workshop near Stony Mountain, setting up the laboratory test unit with a new controller code revision to further test the characteristics of the hydraulic proportional valve/tractor/ram and hose combination, returning to the lab the next day to
recode a set of control actions, returning again to the farm to test the code several days or weeks later. Having access to a hydraulic valve and ram powered by an electric hydraulic pump at the University would have enabled development of the output parameters of the fuzzy control algorithm to take place much more rapidly. Controller code changes could be made quickly on a PC located at such a hydraulic testbed, enabling the researcher to take whatever time is necessary to study the characteristics of the system and produce the desired output. A hydraulic bench apparatus at the University would also have enabled more detailed measurements to be made of the control action, such as litres of oil moving per second, oil pressure, optimization of hose configurations and connections for minimum oil flow resistance and heating. A more detailed evaluation of different valve spool combinations, oil flow rates and throttle positions in investigating the optimal actuation rate could also have been undertaken.

A soil box, such as the unit in the Agricultural Engineering Annex, would have allowed algorithm and controller development to continue throughout the winter months and without the necessity of travelling to a remote location. More extensive testing of both conventional and fuzzy control algorithms would then be possible, as different soil contours could be set up for testing under varying terrain conditions.

The simulations done in Mathcad were facilitated by the Version 6.0 capabilities to program in if-then loops and other iterations [Math96]. Otherwise, the only other option in pursuing an effective simulation would have been to write it in a programming language, such as C or C++ - this would have required more time spent on the implementation of the test code and interface, and less time on evaluation of results. Of course, if the time was
available to pursue this option, a useful CASE tool could have been produced, as outlined in the next Section.

9.4 **Production of a Fuzzy CASE Tool**

Having produced the control surfaces in three dimensions and controller outputs in two dimensions, it becomes obvious how to produce a CASE tool that would provide a fast method of simulating the results of varying membership function and rule variables. Such a tool could also be set up so that assembly code could be produced based on the shape of the membership functions and the rules. The assembler code generation capability could also be set up for a number of microcontrollers, with a menu for the user to choose the appropriate program path. The code generation portion of the program could use several branches for different defuzzification function types that can be chosen by the operator. A general fuzzy inference engine can be used for determining rule strengths. The number and shape of membership functions could be varied by the operator, using not only point-and-click modifications with the mouse, but also have the option of choosing and modifying functions for different membership function shapes - such as Gaussian distributions, normal distributions, and quadratic functions describing the result of a membership function shape determined by other means, such as by a unsupervised training of a neural network.

9.5 **Future Work**

In this research, the 5-membership fuzzy algorithm was tested in the field. It would be advantageous to extend this into testing of more variations of the fuzzy control algorithms. For example, a Gaussian membership function shape for the Zero membership
function could be evaluated, with a 3-membership function algorithm to determine if more gradual control can be achieved than was found in the laboratory simulations. The 3- and 7-membership function algorithms could also be tried in the field. Variations on defuzzification, for example, the mean of maximum, could also be tested to see how well it works with the hydraulic valve.

In this system, it should be noted that although the control algorithm is based on fuzzy control, it operates entirely using discrete data. The sensors are imprecise to a small degree (although within tolerances for this application). Therefore, the controller algorithm interprets the analog signals and classifies them by degree of membership of different membership functions. Rather than undergoing this process in such a discrete way, fuzzy sensors could be utilized that would send the controller signals giving the degree of membership in membership functions directly.

Fuzzy control is a relatively new control method for agricultural applications. Control of agricultural implements usually means that heavy machinery is used, which can be difficult to predict or generalize. There are a number of nonlinear control applications that can be investigated in this field.

This research, concerned with the depth of seeding, is only a small part of what can be investigated for the use of artificial intelligence techniques such as fuzzy control. Pesticide and fertilizer application takes on many forms, with granular, low and high volume liquid, mist, gas, and emulsion methods used commonly on the prairies, each with their own unique challenges. The combination of accurate positioning and a better understanding of crop responses further adds a number of new dimensions that can be
Chapter 9: Conclusions and Recommendations

explored in these areas. Increased efficiency in the harvesting and handling of crops can take the form of reduced combining losses through more accurate machine controls, better sensors in stored grain and reduced shrinkage when handling large volumes. Neural networks and genetic algorithms are being applied to market data in order to reduce the risk individual growers and agribusiness enterprises face when planning future crop prices and sales levels.

In the area of optimization of rules, it was found that the controller performance could be changed considerably with minor changes in rules. The use of a neural network method in determining rules, such as that proposed by Hung [Hung95] that uses a nonsupervised learning self-organizing map to quantize the input vectors into groups, and then using a learned vector quantization backpropagation network to determine the optimal number of antecedents, consequents and the number of rules [HeKP91].

9.6 Chapter Overview

Recommendations are based on the results of the previous Chapter. Appendices A through E contain the controller schematic diagrams, software flow charts, source code for the controller, source code for the PC logging program, and an example of Mathcad simulation for a 5-membership "template" algorithm.
REFERENCES


References


References


References


References


References


References


References


APPENDIX A

FUZZY DEPTH CONTROLLER SCHEMATIC DIAGRAMS

The schematic diagrams were first produced in PADS, then edited for size as a postscript file in Adobe Illustrator. Several parts were not found in the libraries, and had to be made. This included the MC68HC16Z1 MPU, the MC145450 serial interface, and the inductors.
**APPENDIX B**

**SOFTWARE FLOWCHART - FUZZY CONTROLLER**

### B.1 Controller Initialization

Once initialization is complete, main program execution begins (box A1).
B.2 Main Program Execution

The main program consists of eight main parts. Each part is a subroutine that is executed in sequence:

1. CHECK_KEYS - This module checks to see if a key has been pressed. It scans the keypad on every loop. If a low signal is detected on the keypad Port F, the code immediately branches to a key debounce subroutine. After 10 ms, Port F is then rescanned to see if a keypress (low) signal is still present. If so, it then identifies the key pressed and jumps to the appropriate function.

2. SAMPLE - Each of the two sensors are sampled in this subroutine. If the controller is in normal operation mode, the signals are passed onto the next subroutine. If it is in a single sensor mode, only the respective sample is passed on.

The sensors are highly linear resistive elements operating by indicating the position of a contact on a circular track. In order to preserve accuracy in reading the sensor positions, a theta correction factor is employed. This works by taking the sine of the angle from the “zero” position of the sensor linkage, where it is precisely 90 from the sensor. In order to ensure calculation speed and accuracy, at 16-bit sine look-up table is used, rather than having to calculate each value by division.

Initial conversion values for the theta correction subroutine are written to the respective registers during controller initialization. This allows the operator to raise the implement for transport to the area of use. Once at the field, the operator then goes through a calibration routine that sets the registers with the correct conversion and setting values for the current soil conditions and desired depth.
In order to prevent excessive roundoff error throughout the correction calculations, the sensor values are converted to thousands of an inch in a 16-bit format. This process includes a hexadecimal to decimal base conversion subroutine.

3. ERROR_CONVERT - Once the positions of the sensors are determined and corrected for theta error, the error is converted from thousands of an inch to tenths. This facilitates the calculations that will include only addition, subtraction, and maximum and minimum comparisons.

4. ERROR_BUFFER - In this routine, the change in error is determined by taking the present error reading and subtracting it from the error reading 10 (CHECK) samples previously. This equates to about 1/2 second previously, the equivalent of one metre of distance covered at 8 km/hr. This value is written to the CHANGE_IN_ERROR variable for use in the next routine.

5. FUZZYTEST - The fuzzy algorithm is based on the Mamdani fuzzy control algorithm. Membership functions were originally determined by using a Microsoft Windows graphics-based fuzzy logic simulation and analysis tool, FIDE (Fuzzy Inference Development Engine) from Aptronix. The different of the membership functions were set up and run through a simulation.

Although fuzzy algorithms can easily be coded in a high-level language such as C, FIDE speeds up development with increased accuracy by the use of several valuable development tools. Each membership function can be viewed graphically, and quickly changed using the mouse. Likewise, FIDE instantly generates the fuzzy control surface for any set of membership functions and inputs. Once a given combination of membership
functions produce an acceptable gain level of monotonic control, a visual simulation can be run with inputs from both artificially generated signals and actual recorded field data. If, at this stage, the control looks like it will produce the appropriate results, FIDE can immediately generate 8-bit (MC68HC11 compatible) assembler code. This code can then be pasted into the controller code and compiled. The code automatically takes into account scaling factors for sensor inputs and the hydraulic control output signal. Therefore, code is produced quickly and accurately that can be tested in the field.

In this routine, the fuzzy logic algorithm operates the four steps of determining a control decision:

- The input values ERROR and CHANGE_IN_ERROR are fuzzified according to the lookup tables generated from the membership functions in the FIDE simulation. The degree of membership for each membership function is then passed on the fuzzy inference engine.

- The fuzzy values are evaluated according to the rules. The DOF for each rule is determined.

- The rules are aggregated, and the rules that apply are determined.

- The fuzzy decision from the rule analysis is defuzzified. The algorithm uses the singleton method of defuzzification. Although this method is not as elegant as the COA or MOM methods, it is much more efficient to code and execute in assembler, and does provide sufficient 8-bit accuracy for this application.

The output range for the controller is between decimal 120 to 135 (CHECK). This is due to the characteristics of the proportional valve used and the hydraulic system of the
Appendix B: Software Flowchart

tractor used. The tractor used in testing was a Case International 5240 (CHECK) with a closed center hydraulic system. This type of system maintains constant pressure at all times, so will provide a consistent hydraulic control response regardless of the demands on the system.

6. HYD_CONT - Now that a fuzzy decision has been produced, the control signal is sent to the pulse width modulation unit A (PWMA) of the MPU. This modulates the duty cycle of a 128 Hz signal that controls two solenoid valves that control the position of the main spool on a proportional hydraulic valve. By having a constant, opposing 128 Hz pulse on the main spool, a constant flow of oil keeps the main spool in constant motion. The motion is not enough to significantly change the position of the hydraulic ram on the seeder, but is essential to keep valve response time at an optimum level. The duty cycle of the two small solenoids that control the main spool are equal when the main spool is in the proper position. When the seed depth (disk position) has to be changes, the spool must be moved off center by changing the duty cycle of the tow small solenoid. This is done by changing the duty cycle of the signal from PWMA. The signal is split, with one signal controlling one solenoid, and the other, inverted signal controlling the other. Therefore, when the duty cycle is changed, one solenoid will open wider, with the other opening with an equal, opposite narrower opening. The difference in pressure moves the main spool one way or the other, thus acting as a hydraulic amplifier by allowing a large volume of oil to move from one side of the seeder hydraulic cylinder to the other.

7. DISPLAY_DEPTH - Now that the control signal has been sent to the discer, the controller moves into the LCD depth display routine. The depth error value is compared to
the soil surface value, and the difference is displayed on the LCD. The routine includes a subroutine that converts the depth number to inches or centimetres and tenths of an inch or millimetres. Once all the numbers have been calculated, the LCD display routines are called to put each character on the screen. The unit preference can be selected by the operator.

8. SEND_DATA - In order to subjectively test the performance of the controller, a serial communication data logging routine is utilized. Data read from registers in the controller is sent to a laptop computer for saving in a log file and later analysis.

Data is read from the ERROR, CHANGE_IN_ERROR, and OUTPUT registers in order that these calculations can be checked. This data allows cross-checking the controller fuzzy decision with the expected output from previous simulations. Also, if any changes are made in the field, the results of these changes can be recorded and monitored.

Sensor position information is important to ensuring that the ERROR and CHANGE_IN_ERROR calculations are correct. By having the raw 8-bit A/D converter readings over time, the entire process of calculating ERROR, CHANGE_IN_ERROR, the fuzzy inference process and final output can be evaluated.

The sister PC routine operated by checking for valid (new) data being in the serial buffer, reading the data, and sending an acknowledge signal. Once received, the controller increments an index, and sends the next data byte. Both the controller and the PC are synchronized to prevent data skew. In total, 10 bytes are sent per sampling instance. The PC logging program automatically formats the data for saving into an ASCII file for easy importing into a spreadsheet, mathematics or signal analysis program. As data is being
collected, a bar graph display on the PC allows the operator to monitor the progress of the logging and control operations, thus detecting obvious problems early.

The present system runs fast enough to log data 23 times a second. This is equivalent to one sample being taken every 7 cm (2 1/2 inches) at a speed of 10 km/hr (6 mph).

The SEND_DATA routine is commented out before compiling for normal, non-logging operation. When the controller has finished this routine, it loops back to CHECK KEYS to begin the main control loop over again.
B.2.1 CHECK_KEYS Flowchart

- A1
- HAS A KEY BEEN PRESSED?
  - Y
    - ENABLE INTERRUPT FOR DEBOUNCE ROUTINE
    - RUN DEBOUNCE SUBROUTINE
    - HAS A KEY BEEN PRESSED?
      - Y
        - HEX CONVERSION ROUTINE
        - HAS A KEY BEEN IDENTIFIED?
          - Y
            - IS THE KEY HIGHER THAN 9?
              - Y
                - B
              - N
                - A2
            - N
          - N
        - N
      - N
    - N
  - N
B.2.2 RAISED_POSITION_SET

This function is initialized by hitting the "A" key on the keypad. It enters a menu of three choices, in which the operator can select "0" for setting sensor modes (front sensor only, rear only, or both), "1" for setting the seeder raised position for turns and transporting, and "2" for setting the units to either imperial or metric values.
B.2.2.a Setting Disk Raised Position

1. Clear LCD display
2. Display instructions - choose modes
3. Was a key pressed?
   - Yes (Y)
     - Was the key less than 3?
       - Yes (Y)
         - Display error message
       - No (N)
         - Was the key 0?
           - Yes (Y)
             - Store as front sensor mode
           - No (N)
             - Was the key 1?
               - Yes (Y)
                 - Store as rear sensor mode
               - No (N)
                 - Was the key 2?
                   - Yes (Y)
                     - Store as both sensor mode
                   - No (N)
                     - Display error message

4. Back (B)
B.2.2. Setting Disk Raised Position

1. **A3.2**
   - CLEAR LCD DISPLAY
   - DISPLAY INSTRUCTIONS - ENTER DESIRED RAISED HEIGHT
   - DISPLAY INSTRUCTION - ENTER INCHES
   - WAS A KEY PRESSED?
     - Y
     - N

2. **WAS THE KEY BETWEEN 0 AND 9?**
   - Y
   - N
   - DISPLAY ERROR MESSAGE
   - STORE
   - DISPLAY INSTRUCTION - ENTER TENTHS
   - WAS A KEY PRESSED?
     - Y
     - N
   - WAS THE KEY BETWEEN 0 AND 9?
     - Y
     - N
     - DISPLAY ERROR MESSAGE
     - STORE
     - CONVERT TO HEX UNITS USING CALIBRATION FACTORS

-B-12-
B.2.2. c Choosing Imperial or Metric Units
B.2.3 MOVETO_UP

This key command is for raising the disks out of the ground for turns and transporting. The discs are lowered when the "F" key is hit, which resumes normal controller operation.
B.2.4 SET_OPERATING_DEPTH

Appendix B: Software Flowchart
B.2.5 FIELD_0_POSITION_SET

Appendix B: Software Flowchart
B.2.6 CALIBRATE

In this routine, the calculation, scaling and conversion factors are set for the particular sensor/discer implement configuration in use. Once started by pressing the "E" key, the operator is given three calibration choices: "0" for setting the position where the sensor linkage is exactly 90 between each arm, "1" for calibrating the disk sensor height, and "2" for calibrating the front sensor.
B.2.6.a 90 Sensor Arm Position Set

Appendix B: Software Flowchart

[Flowchart diagram]

A7.1

CLEAR LCD DISPLAY

DISPLAY INSTRUCTIONS - POSITION BOTH SENSORS 90 BETWEEN SENSOR ARMS

DISPLAY INSTRUCTION - HIT KEY TO STORE

WAS A KEY PRESSED?

Y

N

SAMPLE DEPTH SENSORS

STORE VALUE FOR USE IN CALIBRATION ROUTINES

B
B.2.6. Disc Sensor Calibration

1. CLEAR LCD DISPLAY
2. DISPLAY MESSAGE - CALIBRATE DISK SENSOR
3. DISPLAY MESSAGE - LOWER DISCS TO GROUND
4. DISPLAY MESSAGE - PRESS ANY KEY TO STORE
5. CHECK KEYS
6. WAS A KEY PRESSSED?
7. SAMPLE DEPTH SENSORS
8. STORE VALUE FROM DISK SENSOR
9. DETERMINE SINE OF SENSOR ANGLE
10. CALCULATE THE SCALE FACTOR FOR DEPTH BY FINDING THE LENGTH OF THE HYPOTENUSE FOR A RIGHT-ANGLE TRIANGLE
11. STORE CALIBRATION SCALE FACTOR
12. B

Appendix B: Software Flowchart
B.2.6. c Calibrate Front Sensor

Appendix B: Software Flowchart

8.2.6. c Calibrate Front Sensor

- CLEAR LCD DISPLAY
- DISPLAY MESSAGE - CALIBRATE FRONT SENSOR
- DISPLAY MESSAGE - LOWER DISCS TO GROUND
- DISPLAY MESSAGE - PRESS ANY KEY TO STORE

- CHECK KEYS
- WAS A KEY Pressed?
  - Y
  - N

- SAMPLE DEPTH SENSORS
- STORE VALUE FROM FRONT SENSOR
- DETERMINE SINE OF SENSOR ANGLE

- CALCULATE THE SCALE FACTOR FOR DEPTH
  BY FINDING THE LENGTH OF THE HYPOTENUSE
  FOR A RIGHT-ANGLE TRIANGLE

- STORE CALIBRATION SCALE FACTOR FOR FRONT SENSOR
- B
B.2.7 MOVETO_DOWN

This routine is used to lower the disc to the field for use in normal controller operation. It is used in conjunction with the MOVETO_UP routine. When the "F" key is hit on the keyboard, the discs return to their set seeding depth.
B.2.8 Sample Depth Sensors
B.2.9 **ERROR_CONVERT**

Routine to convert the error from 1000's of an inch to tenths in hexadecimal. This converts it to a form compatible with the fuzzy inference engine and for display on the LCD.
B.2.10 ERROR_BUFFER

Routine to calculate the change in error from the 10th previous sample.

D
LOAD BUFFER INDEX
SAVE LATEST ERROR TO CIRCULAR BUFFER
CALCULATE CHANGE IN ERROR - CURRENT ERROR COMPARED TO 10 SAMPLES PREVIOUS
SAVE ERROR_CHANGE
E
B.2.11 FUZZYTEST

FUZZYTEST produces the fuzzy decision, based on the inputs, ERROR and ERROR_CHANGE.
B.2.12 HYD_CONT

Routine to scale the fuzzy output decision and send it to the hydraulic valve solenoids.
B.2.13 DISPLAY_DEPTH

Routine to convert from 1000's of an inch to tens and inches, or centimetres and milimetres, and display on the LCD screen.
**B.2.14 SEND_DATA**

Raw ADC sensor readings, error, change in error and the output fuzzy signal are output to a PC acting as a data logger via the built-in MC68HC16Z1 serial port.
APPENDIX C

SOFTWARE SOURCE CODE - FUZZY CONTROLLER

This appendix contains the code used in the Fuzzy Logic Seeding Depth Controller to sample both sensors, determine the fuzzy output and control the hydraulic valve controlling the seeder. The order in which they appear is (1) the make file for compiling, (2) the fuzzy controller code, (3) the table of equates for the register locations in the MC68HC16, (4) the initialization routine for the reset vectors, and finally (4) the initialization routine for the interrupt vectors. Only one interrupt is utilized in the controller code, the KEYPRESS interrupt used in the debounce routine.

Compile Make file: DEPTHROM.ASM
******************************************************************************
* DEPTHROM.ASM Call the functions for assembly for the
* seed depth controller
* Tom Tessier
******************************************************************************

INCLUDE ‘EQUATES.ASM’ *table of EQUates for common register addresses
INCLUDE ‘ORG00000.ASM’ *initialize reset vector
INCLUDE ‘ORG00008.ASM’ *initialize interrupt vectors

******************************************************************************
* Load the program modules
******************************************************************************

INCLUDE ‘field2.ASM’ *Depth controller equates table and main routine

BDM: BGND *EXCEPTION VECTORS POINT HERE
* AND PUT THE MPU IN BACKGROUND MODE

******************************************************************************
* FIELD2.ASM - First field operational version.
* Version 2 logs data, and does not initialize the
* calibration values-the SRAM is battery-backed. The
* values already exist in RAM before this version is
* run.

- C-1 -
This version series has a 2nd hypotenuse and calibration routine for the front sensor. Error change is now in range from 0 to FE rather than in the range from 0 to 7F, as in FUZZY2.ASM. Can provide info to fiddata.c on the serial port, including error, change in error, and fuzzy control signal, once this function is uncommented in MAIN.

Since the fuzzy engine goes in the opposite direction the 25 GPM hydraulics move, the HYD_SIGNAL output of the fuzzy engine is inverted—i.e. 80 is made 7E. This same place in the code is good for scaling the signal to fit the valve—it is divided by 2 here to reduce the gain. See code around line 2175.

* Changed “BLE” to “BPL” in seed depth setting to prevent code from jumping to metric. Also, commented out the inch conversion steps a few lines below this code.

This way, the error will be calculated in 1000's of an inch, not something divided by 60.

Set the variable “INPUTS” for the error, output to the SCI routine to the PC logger. INPUTS is also set to be cleared at the beginning of program execution so an erroneous value is not sent for the lower byte.

The value of INPUTS is also byte-flipped so it is properly recorded by the logger.

-Initialize the ports on the MC68HC16EVB board and run the main routine and keypad routines

***Ports assigned:***
Keypad- Port F
A/D converter- Port AD
LCD- Port GP (data) and SIM (instructions)
Output- Port

ORG $200 *start at $200
include ‘initsys.asm’ *set K registers, COP off, speed 16.78 MHz
include ‘initram.asm’ *turn on Standby RAM (SRAM) at $10000
include ‘initsci.asm’ *set sci baud rate to 9600, enable rec/trans on sci

*Set up Port F for the keypad
*Write to PORTF to set up as I/O:

LDAB #$00
STAB PFFPAR *Set port F up as I/O
LDAA #$0F
STAA DDRF *configure 0-3 output, 3-7 input: port F
LDAB #$FF
STAB PORTF0 *Set port F data registers
**Appendix C: Controller Source Code**

```
* Set up the A/D converter in Port AD
* Write to A/D module configuration register:
* 
  LDAB #$00  
  STAB ADCMCR *for stop/freeze/supervisor modes-set to normal
  *LDE #$0083 *10-bit resolution
  LDE #$0003 *8-bit resolution and 8 A/D clock periods (fastest)
  STE ADCTL0 *A/D control register 0-set up for 8-bit conversion
  *STE ADCTL1 *EXPERIMENTAL----A/D CONTROL REGISTER 1

* Set up Port GP and SIM for LCD
* 
  LDD #$00FF  
  STD CSPAR0 *configure PINS CS3-CS5 port C of SIM FOR I/O
  LDAA #00  
  STAA CSPDR *CLEAR BITS OF PORT C DATA REGISTER
  LDD #$FF00  
  STD PDDR *Ensure GPT data direction register in output only
  LDD #$0000  
  STD OC1M *Ensure OC1 Action Mask Register can’t affect pin
  *Write to GPTPDR to output data

* Set up GPT PWMA for 128 Hz
* 
  LDE #$0008 *256 hz
  STE PWMC *#0018=128 hz, #0028=64 hz, #0038=32 hz

* Save initial values for seeder positioning
* 
  LDAB #$01  
  TBEK * point EK to bank 0 for RAM access
  LDAA #$09  
  STAA CIRC_ERROR_BUFFER ;load $A into circular chg in error buffer
  LDE #$FFFF  
  STE UNITS
  LDE #$00FF *hex for 2.56 cm/inch, times 100
  STE CM_CONV_FACTOR
  CLR E

  STE INPUTS
  STE DEPTH_INCHES
  STE OP_DISPLAY_ON
  STE SEED_DEPTH_ERROR
  STE PREVIOUS_ERROR
  STE ERROR_CHANGE
  STE SIGN_FLAG
  STE KEYCHECK
  STE ADC_CHANNEL
  LDE #ERROR_TABLE
  XGEZ
```
Appendix C: Controller Source Code

LDE #$9999
STE HYDRAUL TIMER1
LDE #$0004
STE HYDRAUL TIMER2
LDE #$FFF
STE HALFSEC_INIT
LDAB #$0F
TBEK * point EK to bank F for register access

********************************************************************

* Initialize Calibration Factors

********************************************************************

* LDAB #$01 * point EK to bank 0 for RAM access
* LDE #$00B0
* STE RAISED
* LDE #$008C
* STE DISC_NINETY_DEG_POS
* STE FRONT_90_READING
* LDE #$0032
* STE CORRECTED_FRONT_ZERO
* LDE #$0032
* STE DISC_LEVEL_ZERO
* LDE 0032
* STE CORRECTED_FIELD_ZERO
* LDE #$004B
* STE OPERATINGDEPTH
* CLRE
* STE CALIB_DEPTH
* LDE #$1E00
* STE HYPOTENEUSE_DISC
* STE HYPOTENEUSE_FRONT
* LDE #$00DE
* STE FRONT_GROUND_ZERO
* LDAB #$0F
* TBEK * point EK to bank 0 for RAM access

********************************************************************

* LCD-INIT

LCD_INIT LDAA #$38
STAA GPTPDR *CONFIGURE LCD AS 8-BIT DATA LENGTH, 1 DISPLAY
LINE, FONT 5 X 10 DOTS
JSR LCD_INST
LDAA #$01 *clear the display, puts cursor at position 0
STAA GPTPDR
JSR LCD_INST
LDAA #$0C *turns on display, cursor on and blinking
STAA GPTPDR
JSR LCD_INST
LDAA #$80 *SET DDRAM ADDRESS
STAA GPTPDR
JSR LCD_INST

*Display initial LCD message
LDX #GREETINGS *point to the appropriate ASCII string
JSR SEND_STRING *print string to led
Appendix C: Controller Source Code

JSR ONESEC_DELAY
LDX GREETINGS2 *point to the appropriate ASCII string
JSR SEND_STRING *print string to lcd
JSR ONESEC_DELAY
LBS MAIN

*****************************************************************************

* LCD OPERATING ROUTINES: LCD_DATA set LCD in data display mode
  * LCD_INST set LCD in instruction mode
  * LCD_DELAY delay routine for LCD enable function
  * CLEAR_DISPLAY display clearing routine

LCD_DATA:
LDAA #$01
STAA CSPDR
LDAA #$05
STAA CSPDR
JSR LCD_DELAY
LDAA #$01
STAA CSPDR
RTS

LCD_INST:
LDAA #$00 *WRITE INSTRUCTIONS TO LCD-HAVE INST. IN GPTPDR
STAA CSPDR *SET IN INSTRUCTION MODE
JSR LCD_DELAY ******EXPERIMENTAL-DUE TO OC NOT WORKING ON INIT
LDAB #$04
STAB CSPDR *SET ENABLE HIGH
JSR LCD_DELAY
STAA CSPDR *SET ENABLE LOW
RTS  *GET OUTTA HERE

LCD_DELAY:
LDAB #$01
TBK
NOP
NOP
NOP
NOP
STE TEMP_DELAY2 *put previous data in here so you don’t lose it
LDE #$00FF
STE TEMP_DELAY

LCD_DELAY_DEC:
DECW TEMP_DELAY
BGT LCD_DELAY_DEC
LDE TEMP_DELAY2
LDAB #$0F
TBK
RTS

CLEAR_DISPLAY: *resets display to prevent problems
LDAB #$0F
TBK
LDAA #$01 *called just before info is displayed
STAA GPTPDR
JSR LCD_INST
LDAB #$01
TBK
RTS
Appendix C: Controller Source Code

* MAIN ROUTINE-calls keypad scanner, sensor sample
* subroutine, fuzzy logic decision routine, hydraulic
* control output, depth and LCD display subroutine,
* and sending data to the PC for logging results.
* Uncomment JSR SEND_DATA to start data logging.
*********************************************************************
MAIN:
JSR CHECK_KEYS ;see if a key has been pressed-if so, debounce
JSR SAMPLE ;sample the 2 sensors via A/D0 and A/D1
JSR ERROR_CONVERT ;convert error from 1000's inch to 10's in hex
JSR ERROR_BUFFER ;calculate change in error for fuzzy routine
JSR FUZZYTEST ;calculate the fuzzy decision
JSR HYD_CONT ;output the signal to the hydraulics
JSR DISPLAY_DEPTH ;display depth and messages on LCD
JSR SEND_DATA ;log error, error change, and fuzzy decision on PC
BRA MAIN ;loop back for next round
*********************************************************************

* KEY CHECKING ROUTINE: CHECK_KEYS read port F to see if any keys hit *
* RETURN_DEBOUNCE program returns here after debounce interrupts then *
* checks key function
* DEBOUNCE interrupt 22 times then recheck key for signal
* KEY_IRQ actions taken on each interrupt-int disabled after 22 times
* READ_KEYS check key id, with READ_COLUMN, KEY_ID1, KEY_ID2, KEY_IDED, *
* HEXCONV (which returns the key’s id in KEY)
* *
*********************************************************************
CHECK_KEYS:
LDAB #$0F
TBEK
LDAA #$F0 *Set all column lines low
STAA PORTF0
LDAB PORTF0 *Load PORTF
CBA *See if any keys are pressed-any high nibble bit is set
BNE DEBOUNCE *Sets up the GPT to interrupt the CPU 6 times
*to allow for at least 20 ms for the key pressed
*to stop bouncing. On the 6th interrupt, the keypad
*is read to determine which, if any, key was actually
*pressed.
RTS

RETURN_DEBOUNCE:
NOP *HAVING TROUBLE WITH #$01 NOT GOING INTO acc B
NOP *WHEN RETURNING FROM INTERRUPT--SO ADDED NOPS
NOP
NOP
LDAB #$01
TBEK
TSTW KEYCHECK
BNE FINISHED_KEYS *if you don’t want to id the key yet, RTS
JSR HEXCONV
LDAB #$01
TBEK
TSTW HEXKEY_HIT
BEQ FINISHED_KEYS
JSR CHECK_FUNCTION
FINISHED_KEYS:
APPENDIX C: CONTROLLER SOURCE CODE

RTS

DEBOUNCE:
LDAB #$01    ; point EK to bank 0 for RAM access
TBEK #$16
LDAA IRQ_COUNT ; put the countdown amount for 22 IRQ's in standby ram
CLRD
STD IRQ_DONE ; initialize to prevent repeating IRQ in endless loop
CLRW OP_DISPLAY_ON
LDAB #$0F
TBEK
LDAA #$73 ; enable the pulse accumulator counter to work, gated
STAA PACTL ; mode, so that it will count & interrupt on demand.
CLRA
STAA PACNT ; set pulse accumulator counter to 0
TPD ; transfer CCR to Acc. D
LDD #$00A0 ; set up D for clearing CCR and setting IRQ mask to 5
TDP ; transfer D to CCR. DDR is now ready to accept int 5
* from the GPT timer overflow interrupt.
NOP ; *NOPS to add time for the 286 debugger to catch up
NOP
NOP
LDD #$0086 ; MCR doesn't exist in the equates table!! It's called
* GPTMCR.
STD GPTMCR ; store IRQ 6 for IMB to module configuration register
LDD #$4004 ; enable OC4 interrupt in the QSM, and set timer
* timer prescaler to 64-interrupt every 64*5FF clocks
* equal to 0.98 ms-20 ms delay, read on 22nd int.
STD TMSK1
LDD #$7660 ; set GPT interrupt priority to OC4 (number 7).
STD ICR ; interrupt level to IRQ 6, vector base address to 6
* 6X, so 6*16=96 in decimal.
* so, IRQ vector will be 67.
* the CPU decodes as 67*2 by
* left shifting the IRQ vector, using IRQ service routine
* label at location $134. Changed label BDM at "96"
* to KEY_IRQ
BRA RETURN_DEBOUNCE ; now that the interrupt is set, return
* and let the keyboard be read by the interrupt routine

KEY_IRQ:
NOP
NOP
LDAB #$01 ; point EK to bank 0 for RAM access
TBEK IRQ_DONE ; set in HEXCONV
TSTW
LBGT RETURN_DEBOUNCE ; get out if keys already read after IRQ's
DEC IRQ_COUNT ; decrement the counter for each interrupt request
LDAB #$0F
TBEK
LDD #$0000 ; point EK to bank F for register access
STD TFLG1 ; clear the GPT interrupt flag so the CPU isn't
* interrupted again immediately following RTI
LDAB #$01 ; point EK to bank 0 for RAM access
TST IRQ_COUNT ; resetting
LBEX READ_KEYS ; branch after 22 ints, to keyboard reading routine.
LDAB #$0F ; restoring of k register occurs after br to READ_KEYS
TBEK
RTI
Appendix C: Controller Source Code

READ_KEYS:
LDAB #$0F
TBEK
LDD #$0000 *Disable OC4 interrupt in the QSM. and leave timer
*prescaler at 64.
STD TMSK1 *Now, the interrupt from the GPT is completely
disabled.
LDD #$0000 *Set GPT interrupt priority to OC4 (number 7),
STD ICR *interrupt level to IRQ 6, vector base address to 6
LDAA #$F0 *Set all column lines low
STAA PORTFO
LDAB PORTFO *Load PORTF
CBA *See if any keys are pressed- any high nibble bit is set
LBFEQ NO_KEY_PRESSED *leave if no indication of keypress after delay
LDAB #$01
TBEK
LDE #$FFFF *Set flag so program returns to routine properly
STE IRQ_DONE
STE KEYPRESSED *set KEYPRESSED flag so other functions can use it
LDAB #$0F
TBEK
TDP *Transfer CCR to Acc. D
LDD #$00E0 *Set up D for clearing CCR and setting IRQ mask to 7
TDP *Transfer D to CCR. DDR now will refuse int 6 & less.
LDAA #$F7
READ_COLUMN:
STAA PORTFO *Write to column 1
LDAB PORTFO
CBA *See if this column was pressed
BNE KEY_ID1 *If this column was pressed, see which row it is
TDE
TDP
TDP
ORD #$100
TDP
TDE
RORA *Shift right to check next column (enters 1 on MS bit)
LBCC NO_KEY_PRESSED *When carry is clear, leave routine
LBRA READ_COLUMN *If it wasn’t this column, go and check the next one

KEY_ID1:
LDAA #$01
XGAB * TBEK only works with Acc B-so exchange B,A
TBEK * point EK to bank 0 for RAM access
CLRW KEY * start key hex identity register at 0
LDE #$0011
STE KEY_ERR_COUNTER * load $10 to count down each time a different
* key identity is tested-just in case the key
* byte in acc B doesn’t match any in the
* table KEY_TABLE-this prevents an endless loop

KEY_ID2:
DECW KEY_ERR_COUNTER
LBEC NO_KEY_PRESSED
LDAB KEY+1
CLRW INDEX
STAB INDEX+1
LDE #KEY_TABLE
ADDE INDEX
STE INDEX
LDX INDEX
LDAB X
CBA
LBEQ KEYIS_IDED *if equal, branch to HEXCONV, and compare
INCW KEY *check to see if it's the next number higher
BNE KEY_ID2

KEYIS_IDED:
LDAB #$0F
TBEK
RTI

HEXCONV:
LDAB #$01 *acc B is the number pressed-KEY is hex
TBEK
LDE #$00
STE HEXKEY_HIT
LDAB KEY+1 *at this point<key, so reload number and
SUBB #$0A *subtract A; if acc B is 9 or less, will be neg
LBGE HEXNUM_CONV *branch to HEXNUM_CONV below if A to F
LDAB KEY+1 *store it to KEY for later use
ADDB #$30 *then add #$30 to make an ASCII
STAB KEYASCII *code. Save to KEY for later use.
LDAB #$0F *reset to bank F so MPU will call peripherals
TBEK *point EK to bank F for register access
RTS

HEXNUM_CONV: *convert numbers A to F to ASCII and implement
LDAB KEY+1 *load the number pressed to B
ADDB #$40 *make it an ASCII letter value
STAB KEYASCII *store it to KEY for later use
LDAA #$0A *load acc A to see if B is $0A
CBA *compare acc A to acc A (B-A=x)
LDE #$0001
STE HEXKEY_HIT
RTS

******************************************************************************

*CHECK_FUNCTION is a subroutine called in RETURN_DEBOUNCE
* CHECK_FUNCTION checks the keys pressed to see if they are for controller*
* settings. This calls the following functions:
* -RAISED_POS_SET set the position of the discs when raised and determine*
* whether 1 or 2 sensor mode is used. Also for determining whether metric*
* or imperial units are used
* -MOVETO_UP raise the disks
* -SET_OPER_DEPTH set seeding depth
* -FIELD_0_POS_SET set field surface position
* -CALIBRATE is key to accurate operation. Sets the sensor positions
* 90 degrees to the connecting arm so a correction factor can be used
* as the arm moves from the horizontal.
* -MOVETO_DOWN resume normal operation of the controller, from the*
* raised position.

******************************************************************************

CHECK_FUNCTION
LDAB #$01
TBEK
LDE #$0000
STE KEYPRESSED *clear keypressed flag for the next key read
LDAB KEY+1 *load the number pressed to B
CBA
LBEQ RAISED_POS_SET *if they are the same (x=0), branch to A fcn.

-C-9-
Appendix C: Controller Source Code

INCA    *acc A is now $0B
CBA
LBEQ MOVETO_UP  *raise the seeder to the up position-turns, etc
INCA    *acc A is now $0C
CBA
LBEQ SET_OPER_DEPTH *set the depth of seeding
INCA    *acc A is now $0D
CBA
LBEQ FIELD_POS_SET *set the zero" depth position in the field
INCA    *acc A is now $0E
CBA
LBEQ CALIBRATE  *set the 6" depth position and zero position
* on a level surface
INCA    *acc A is now $0F
CBA
LBEQ MOVETO_DOWN  *put the seeder to the operating position

RAISED_POS_SET: *Choose, and set, sensor modes and position of raised seeder
JSR CLEAR_DISPLAY
LDX #CHOOSE_FUNCTION_MSG *choose between modes or raised pos
JSR SEND_STRING  *print it to the lcd screen
JSR TWSEC_DELAY
LDX #PICK_MODE_OR.Raise1
JSR SEND_STRING  *print it to the lcd screen
JSR TWSEC_DELAY
LDX #PICK_MODE_OR.Raise2
JSR SEND_STRING  *print it to the lcd screen
JSR TWSEC_DELAY
LDX #PICK_MODE_OR.Raise3
JSR SEND_STRING  *print it to the lcd screen
JSR TWSEC_DELAY
LDX #PICK_MODE_OR.Raise4
JSR SEND_STRING  *print it to the lcd screen
JSR TWSEC_DELAY
LDX #PICK_MODE_OR.Raise5
JSR SEND_STRING  *print it to the lcd screen
JSR TWSEC_DELAY
JSR WAITPER_KEYPRESS
JSR HEXCONV
LDAB #$01
TBEK
LDAB KEY+1  *load key number
SUBB #$03  *subtract 3; if acc B is 2 or less, will be neg
LBGE ERROR_PRESS  *and be an error
TSTW KEY
LBNE CHECKER_1 OR_2

LDX #CHOOSE_MODE_MSG *point to the mode message
JSR SEND_STRING  *print it to the lcd screen
JSR TWSEC_DELAY
JSR CLEAR_DISPLAY
LDX #HIT_KEY_MSG
JSR SEND_STRING
JSR WAITPER_KEYPRESS
JSR CLEAR_DISPLAY
LDX #MODE_A
JSR SEND_STRING
JSR TWSEC_DELAY
LDX #MODE_B
JSR SEND_STRING
JSR TWSEC_DELAY
Appendix C: Controller Source Code

LDX #MODE_C
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR WAITER_KEYPRESS
JSR HEXCONV
LDAB #$01
TBEK
LDAB KEY+1  *load key number
SUBB #$02  *subtract A; if acc B is 1 or less, will be neg
LBGE ERRORPRESS  *error if 2-F
CLRW ADC_CHANNEL  *clear location of raised value
LDAB KEY+1  *load key number
STAB ADC_CHANNEL
LDAB #$0F
TBEK
RTS

CHECKFER I OR 2:
LDAB KEY+1
DECB
LBNE SET_UNITS

SET_RAISED_POS:
LDX #SET_RAISE_MSG  *point to lcd message for setting raised pos
JSR SEND_STRING  *print it to the lcd screen
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #HIT_KEY_MSG
JSR SEND_STRING
JSR WAITER_KEYPRESS
JSR CLEAR_DISPLAY
LDX #ENTER_HEIGHT
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR WAITER_KEYPRESS
JSR HEXCONV
LDAB #$01
TBEK
LDAB KEY+1  *load key number
SUBB #$0A  *subtract A; if acc B is 9 or less, will be neg
LBGE ERRORPRESS  *error if A-F
CLRW RAISED  *clear location of raised value
LDAB KEY+1  *load key number
ASLB
ASLB
ASLB
ASLB  *put inch or cm in MSB position of 8 bit number
STAB RAISED+1
JSR CLEAR_DISPLAY
LDX #ENTER_IOTH_HEIGHT
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR WAITER_KEYPRESS
JSR HEXCONV
LDAB #$01
TBEK
LDAB KEY+1  *load key number
SUBB #$0A  *subtract A; if acc B is 9 or less, will be neg
LBGE ERRORPRESS  *error if A-F
LDAB KEY+1  *load key number
ADDB RAISED+1  *add in or cm to tenths or mm
STAB RAISED+1  *store as raised number
Appendix C: Controller Source Code

*LDAA UNITS    *see if inches or cm
*TSTA
*LBEQ RAISED_METRIC
JSR CLEAR_DISPLAY
LDX IN_CONV_FACTOR2 *divide inches & tenths by total possible (60)
LDAB RAISED+l    *load actual in and tenths
FDIV    *answer in IX-MSB is sign bit
XGDX    *transfer IX to D-acc A is 2nd $ 3rd MSB's
LDAB #$7F  *correct for 6" depth by subtracting from total
SUBB CALIB_DEPTH  *subtract the amt. for 6" depth
MUL
ADD A #$7F    *correct for being above ground
STAA RAISED+1    *store remainder (in D-acc A is MSB of remainder)
*LDE RAISED    *CAN IT-JUST USE DIRECT VALUES FROM SAMPLE
*JSR ANGLE_CORRECTION  *convert to 1000's of an inch and adjust for angle
LDAB #$0F
TBEK
RTS

SET_UNITS:
JSR CLEAR_DISPLAY
LDX #PICK_UNIT_MESSAGE1 *point to the appropriate ASCII string
JSR SEND_STRING *print string to lcd
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #PICK_UNIT_MESSAGE2 *point to the appropriate ASCII string
JSR SEND_STRING *print string to lcd
JSR TWOSEC_DELAY
JSR WAITER_KEYPRESS
JSR HEXCONV
LDAB #$01
TBEK
LDAB KEY+l     *load key number
SUBB #$02
LBGE ERROR_PRESS *error if above 1
TSTW KEY
BNE SET_METRIC
LDE #$FFFF
STE UNITS
RTS

SET_METRIC:
CLRE
STE UNITS
RTS

MOVETO_UP:
JSR CLEAR_DISPLAY
LDX #RAISE_MESSAGE1 *point to the appropriate ASCII string
JSR SEND_STRING *print string to lcd
JSR TWOSEC_DELAY
LDX #RAISE_MESSAGE2
JSR SEND_STRING *print string to lcd
LDAB #$01
TBEK
LDAA RAISED+1 *load variable for rate of speed of lift
LDAB #$0F
TBEK
STAA PWMA    *send the value to the hydraulics
LDAB #$01
TBEK
RAISED_WAIT:
JSR SAMPLE  *see what the height of the discs are
****SUBTRACT ACTUAL HEIGHT FROM THE SET HEIGHT-AS LONG AS IT IS BELOW
****THE SET HEIGHT IT WILL KEEP RAISING THE IMPLEMENT
LDAB #501
TBEK
LDAB RAISED+1  *LOAD HEIGHT YOU WANT TO RAISE TO
SUBB DISC_READING+1  *SUBTRACT THE ACTUAL HEIGHT
BPL RAISED_WAIT  *BRANCH IF OVERFLOW CLEAR-IE. STILL BIG NUMBER
   *MINUS SMALLER NUMBER-V BIT SET WHEN HIGHER
   *THAN THE RAISED SET POINT
LDAA #$7F  *discs are up to the required height-stop disc movement
LDAB #$50F
TBEK
STAA PWMA  *by loading neutral movement (7F) and store to PWMA
JSR WAITFER_KEYPRESS
JSR HEXCONV
LDAB #501
TBEK
LDAB KEY+1
SUBB #$50F
BNE RAISED_WAIT
LDAB #$50F
TBEK
RTS

SET_OPER_DEPTH:
JSR CLEAR_DISPLAY
LDX #SET_OPER_MSG  *point to lcd message for setting raised pos
JSR SEND_STRING  *print it to the lcd screen
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #HIT_KEY_MSG
JSR SEND_STRING
JSR WAITFER_KEYPRESS
JSR HEXCONV
LDAB #501
TBEK
LDAB KEY+1  *load key number
SUBB #$50A  *subtract A; if ace B is 9 or less, will be neg
LBGE ERROR_PRESS  *error if A-F
LDAB #$501
TBEK
CLRW RAISED2  *clear location of raised value
LDAB KEY+1  *load key number
LDAA #$50A
MUL  *inches times decimal 10
CLRW OPERATING_DEPTH
STAB OPERATING_DEPTH+1
LDX #ENTER_10TH_DEPTH
JSR SEND_STRING  *print it to the lcd screen
JSR TWOSEC_DELAY
JSR WAITFER_KEYPRESS  *second- tenth inch or mm entry
JSR HEXCONV
LDAB #501
Appendix C: Controller Source Code

```assembly
TBEK
LDAB KEY+1    *load key number
SUBB #$0A      *subtract A; if acc B is 9 or less, will be neg
LBGE ERROR_PRESS *error if A-F
LDAB KEY+1    *load key number
ADDB OPERATING_DEPTH+1 *add in or cm to tenths or mm
STAB OPERATING_DEPTH+1 *store as raised number
LDE OPERATING_DEPTH
LDD #$0064
EMUL
STD OPERATING_DEPTH *store as 1000's of an inch
LDAA UNITS    *see if inches or cm
TSTA
BPL OPERATING_METRIC
JSR CLEAR_DISPLAY
  * LDX IN_CONV_FACTOR2 *divide inches & tenths by total possible (60)
  * LDAB RAISED2 *load actual in and tenths
  * IDIV *remainder (in D-acc A is MSB of remainder)
  * STAA OPERATING_DEPTH *store remainder (in D-acc A is MSB of remainder)
  LDAB #$0F
  TBEK
  RTS

OPERATING_METRIC:
  NOP
  NOP
  NOP
  NO
  RTS

FIELD_0_POS_SET: *set the zero depth position in the field
JSR CLEAR_DISPLAY
LDX #$FIELD0_SET_MSG *point to the appropriate ASCII string
JSR SEND_STRING *print string to lcd
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #$HIT_KEY_MSG
JSR SEND_STRING
JSR WAITFOR_KEYPRESS
JSR CLEAR_DISPLAY
LDX #$LOWER_GRND_LEVEL
JSR SEND_STRING *print it to the lcd screen
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #$HIT_TO_STORE
JSR SEND_STRING
JSR WAITFOR_KEYPRESS
JSR SAMPLE *sample the sensors again
LDAB #$01    *set page-SAMPLE leaves it as $0F
TBEK
LDE DISC_READING *correct for angle of sensor to linkage
JSR ANGLE_CORRECTION *correct for angle of sensor to linkage
  *input to algorithm is in acc E, with answer put in E
STE CORRECTED_FIELD_ZERO
LDE FRONT_READING
JSR ANGLE_CORRECTION *correct for angle of sensor to linkage
  *input to algorithm is in acc E, with answer put in E
STE FRONTGROUND_ZERO
LDAB #$01
TBEK
RTS
```

---

-C-14-
CALIBRATE:  *first, choose mode-either (0) calibrate the sensor by setting
   *to 90 degree angle from the connecting rod, or (1) set the
   *position for the level position on a hard, level surface
   *and then the position 6” above that (disc sensor). (2) is
   *for calibrating the front sensor, 6” above a level surface

*first, choose the mode:
J SR  CLEAR_DISPLAY
LDX  #CHOOSE_FUNCTION_MSG  *choose between 90 deg calib or 6” deep calib
J SR  SEND_STRING  *print it to the lcd screen
J SR  TWOSEC_DELAY
LDX  #PICK_CALIB_90_OR_6IN1
J SR  SEND_STRING  *print it to the lcd screen
J SR  TWOSEC_DELAY
LDX  #PICK_CALIB_90_OR_6IN2
J SR  SEND_STRING  *print it to the lcd screen
J SR  TWOSEC_DELAY
LDX  #PICK_CALIB_90_OR_6IN3
J SR  SEND_STRING  *print it to the lcd screen
J SR  TWOSEC_DELAY
LDX  #PICK_CALIB_90_OR_6IN4
J SR  SEND_STRING  *print it to the lcd screen
J SR  TWOSEC_DELAY
LDX  #PICK_CALIB_90_OR_6IN5
J SR  SEND_STRING  *print it to the lcd screen
J SR  TWOSEC_DELAY
J SR  WAITFER_KEYPRESS
J SR  HEXCONV
LDAB  #S01
TBEK
LDAB  KEY+1  *load key number
SUBB  #S03  *subtract 3; if acc B is 2 or less, will be neg
LBGE  ERROR_PRESS  *and be an error
TSTW  KEY
LBNE  CHECKFER_1 OR 2 FOR CALIB

*set the 90 deg position:
J SR  CLEAR_DISPLAY
LDX  #CALIB_90_DEG1  *point to the appropriate ASCII string
J SR  SEND_STRING  *print string to lcd
J SR  TWOSEC_DELAY
J SR  CLEAR_DISPLAY
LDX  #CALIB_90_DEG2  *point to the appropriate ASCII string
J SR  SEND_STRING  *print string to lcd
J SR  TWOSEC_DELAY
J SR  CLEAR_DISPLAY
LDX  #CALIB_90_DEG3  *point to the appropriate ASCII string
J SR  SEND_STRING  *print string to lcd
J SR  TWOSEC_DELAY
J SR  CLEAR_DISPLAY
LDX  #CALIB_90_DEG4  *point to the appropriate ASCII string
J SR  SEND_STRING  *print string to lcd
J SR  TWOSEC_DELAY
J SR  CLEAR_DISPLAY
LDX  #HIT_KEY_MSG
J SR  SEND_STRING
J SR  WAITFER_KEYPRESS
J SR  CLEAR_DISPLAY
LDX  #SET_TO_90_MSG1
J SR  SEND_STRING  *print it to the lcd screen
Appendix C: Controller Source Code

JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #SET_TO_90_MSG2
JSR SEND_STRING *print it to the lcd screen
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #SET_TO_90_MSG3
JSR SEND_STRING *print it to the lcd screen
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #HIT_TO_STORE
JSR SEND_STRING
JSR WAITFOR_KEYPRESS
JSR SAMPLE
LDAB #$01
TBEK
LDE DISC_READING
STE DISC_NINETY_DEG_POS
LDE FRONT_READING
STE FRONT_90_READING
RTS

CHECKFER_1 OR 2 FOR CALIB:
LDAB KEY+1
DECB
LBNE CALIB_FRONT_SENSOR

CALIB_BACK_SENSOR:
JSR CLEAR_DISPLAY
LDX #LEVEL0_SET_MSG *point to the appropriate ASCII string
JSR SEND_STRING *print string to lcd
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #LEVEL_SURF_CALIB
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #HIT_KEY_MSG
JSR SEND_STRING
JSR WAITFOR_KEYPRESS
JSR CLEAR_DISPLAY
LDX #DISC_LEVEL_ZERO_MSG
JSR SEND_STRING *print it to the lcd screen
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #HIT_TO_STORE
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR WAITFOR_KEYPRESS
JSR SAMPLE
LDAB #$01
TBEK
LDE DISC_READING *use disk reading only for this
STE DISC_LEVEL_ZERO
****now, calibrate disc sensor for 6" deep****
JSR CLEAR_DISPLAY
LDX #SIX_IN_CALIB0
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #SIX_IN_CALIB1
Appendix C: Controller Source Code

```assembly
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #HIT_KEY_MSG
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR WAITFOR_KEYPRESS
JSR CLEARDISPLAY
LDX #HIT_TO_STORE
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR WAITFOR_KEYPRESS
JSR SAMPLE
CLR D
LDAB #$01
TBEK
LDE DISC_READING
SUBE DISC_NINETY_DEG_POS
BPL LOAD_DISC_SINE
NEGE
LOAD_DISC_SINE:
  XGEX
  LDY #SINE_TABLE *now we have the sine of the sensor angle at 6"
  XGEY
  CLRA
  CLR W CALIB DEPTH
  LDAB E.X * sine of sensor position at 6"
  STA B CALIB_DEPTH+1 * EXPERIMENTAL-CHANGE THIS VARIABLE NAME
  LDE DISC_LEVEL_ZERO
  SUBE DISC_NINETY_DEG_POS * now, get the sine of the angle between ground and 90 deg pos
  TSTE
  BMI NEGATIVE_CALIB
  XGEX
  LDY #SINE_TABLE * now we have the sine of the sensor angle at 6"
  XGEY
  CLRA
  LDAB E.X * sine of sensor position at 6"
  CLR W TRUE_0_DEPTH
  STA B TRUE_0_DEPTH
  LDD CALIB DEPTH
  SUBD TRUE_0_DEPTH * subtract the sine of the difference between
  * DISC_LEVEL_ZERO and DISC_NINETY_DEG_POS from the difference
  * between the 6" point and DISC_NINETY_DEG_POS
  FINISH_CALIB:
  XGDX
  CLR E
  LDD #$1770 * this is 6000 thousandths of an inch in hex
  EDIV * divide the sine into the length of the opposite
  * side - this gives the hypotenuse in thousandths of an inch
  XGDX * multiply by 256 - you had DIVIDED by a whole number
  ASLD * (multiplied by sine * 256)
  ASLD * (alt - 179 for vertical bar!)
  ASLD * 
  ASLD * 6" depth is \ hypotenuse
  ASLD * equal to the \ 
  ASLD * opposite side-> \ 
  ASLD * \ ________\ < sine of this angle
  XGDE
  STE HYPOTENEUSE_DISC * sine of angle * opposite = HYPOTENEUSE_DISC
  LDAB #$0F * IN THOUSANDSTH OF AN INCH
  TBEK
```

---

-C-17-
Appendix C: Controller Source Code

RTS

NEGATIVE_CALIB:
NEGE
XGE
LDY #SINE_TABLE *now we have the sine of the sensor angle at 6"
XGEY
CLRA
LDAB EX *sine of sensor position at 6"
ADDD CALIBDEPTH *subtract the sine of the difference between
*DISC_LEVEL_ZERO and DISC_NINETY_DEC_POS from the difference
*between the 6" point and DISC_NINETY_DEC_POS
BRA FINISH_CALIB

CALIB_FRONT_SENSOR:
JSR CLEAR_DISPLAY
LDX #LEVELF_SET_MSG *point to the appropriate ASCII string
JSR SEND_STRING *print string to lcd
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #LEVEL_SURF_CALIB
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #HIT_KEY_MSG
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR HIT_KEYPRESS
JSR CLEAR_DISPLAY
LDX #DISC_LEVEL_ZERO_MSG
JSR SEND_STRING *print it to the lcd screen
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #HIT_TO_STORE
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR HIT_KEYPRESS
JSR TWOSAMPLE
LDAB #$01
TBEK
LDE FRONT_READING *use disk reading only for this
STE FRONT_LEVEL_ZERO
****now, calibrate for 6" deep****
JSR CLEAR_DISPLAY
LDX #SIX_IN_CALIB0
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #SIX_IN_CALIB1
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR CLEAR_DISPLAY
LDX #HIT_KEY_MSG
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR WAITER_KEYPRESS
JSR CLEAR_DISPLAY
LDX #HIT_TO_STORE
JSR SEND_STRING
JSR TWOSEC_DELAY
JSR WAITER_KEYPRESS
JSR TWOSAMPLE
Appendix C: Controller Source Code

CLRD
LDAB #$01
TBEX
LDE FRONT_READING
SUBE FRONT_90_READING
BPL LOAD_FRONT_SINE
NEGE
LOAD_FRONT_SINE:
  XGEX
  LDY #SINE_TABLE *now we have the sine of the sensor angle at 6"
  XGEY
  CLRA
  CLRW CALIB_DEPTH
  LDAB E.X *sine of sensor position at 6"
  STAB CALIB_DEPTH+1
  LDE FRONT_LEVELZERO
  SUBE FRONT_90_READING *now, get the sine of the angle between ground and 90 deg pos
  TSTE
  LBMI NEGATIVE_CALIB
  XGEX
  LDY #SINE_TABLE *now we have the sine of the sensor angle at 6"
  XGEY
  CLRA
  LDAB E.X *sine of sensor position at 6"
  CLRW TRUE_0_DEPTH
  STAB TRUE_0_DEPTH
  LDD CALIB_DEPTH
  SUBD TRUE_0_DEPTH *subtract the sine of the difference between
  *FRONT_LEVELZERO and FRONT_NINETY_DEG_POS from the difference
  *between the 6" point and FRONT_NINETY_DEG_POS
FINISH_CALIBB:
  XGDX
  CLR E
  LDD #$1770 *this is 6000 thousandths of an inch in hex
  EDIV *divide the sine into the length of the opposite
  *side-this gives the HYPOTENUSE in thousandths of an inch
  XGDX *multiply by 256-you had DIVIDED by a whole number
  ASLD *(multiplied by sine * 256)
  ASLD *(alt-179 for vertical bar!)
  ASLD *
  ASLD * 6" depth is \ hypotenuse
  ASLD * equal to the \ 
  ASLD * opposite side-> \ < sine of this angle
  XGDE
  STE HYPOTENUSE_FRONT *sine of angle * opposite = hypotenuse
  LDAB #$0F *IN THOUSANDSTH OF AN INCH
  TBEX
  RTS
NEGATIVE_CALIBB:
  NEGE
  XGEX
  LDY #SINE_TABLE *now we have the sine of the sensor angle at 6"
  XGEY
  CLRA
  LDAB E.X *sine of sensor position at 6"
  ADDD CALIB_DEPTH *subtract the sine of the difference between
  *DISC_LEVELZERO and DISC_NINETY_DEG_POS from the difference
  *between the 6" point and DISC_NINETY_DEG_POS
  BRA FINISH_CALIBB
Appendix C: Controller Source Code

MOVETO_DOWN:
    JSR CLEAR_DISPLAY
    LDX #LOWER_MESSAGE *point to the appropriate ASCII string
    JSR SEND_STRING *print string to lcd
    JSR TWOSEC_DELAY
    LDAB #$01
    TBEK
    LDE #OPERATING_DEPTH
    STE OUTPUT *send the value to the hydraulic control routine
    LDE #HYDRAUL_TIMER1 *load the number to count down
    STE COUNTDOWN1
    LDE HYDRAUL_TIMER2 *load the number of times the number will be
    STE COUNTDOWN2 * counted down
    JSR HYD_CONT *move the hydraulics
    LDAB #$0F
    TBEK
    RTS
********************************************************************

KEYPRESS AND DELAY UTILITY ROUTINES: WAITFER_KEYPRESS loop for waiting for
*a key to be pressed for calibration and setting routines
*ERROR_PRESS kicks in when the wrong key is pressed
*HALFSEC_COUNT a timing loop for putting messages on the LCD screen
*ONESEC_DELAY and TWOSEC_DELAY calls HALFSEC_COUNT several times.
*NOKEYPRESSED clears key pressed flag if none pressed.
********************************************************************

WAITFER_KEYPRESS:
    LDAB #$01
    TBEK
    LDE #$0001
    STE KEYCHECK *set flag-no key id, only check for press
    LDAB #$0F
    TBEK
    JSR CHECK_KEYS
    LDAB #$01
    TBEK
    TSTW KEYPRESSED *see if a key had been pressed
    LBEQ WAITFER_KEYPRESS
    LDE #$0000
    STE KEYPRESSED
    LDE #$0000
    STE KEYCHECK
    RTS

ERROR_PRESS:
    JSR CLEAR_DISPLAY
    LDX #ERR_PRESS_MSG0
    JSR SEND_STRING *print string to lcd
    JSR TWOSEC_DELAY
    JSR CLEAR_DISPLAY
    LDX #ERR_PRESS_MSG1 *error if A-F
    JSR SEND_STRING *print string to lcd
    JSR TWOSEC_DELAY
    LDAB #$01
    TBEK
    RTS

HALFSEC_COUNT:
Appendix C: Controller Source Code

LDAB #$01
TBEK
MOVW HALFSEC_INIT,COUNTDOWN2

HALFSEC_LOOP
LDAB #$01
TBEK
DECW COUNTDOWN2 *decrement the main countdown
LDAB #$0F
TBEK
LDAB #$01
TBEK
TSTW COUNTDOWN2 *when done small count, test "large" main count
BNE HALFSEC_LOOP *loop back until done
LDAB #$0F
TBEK
*say, "Bye Bye" *(small wave)
RTS

ONESEC_DELAY:
JSR HALFSEC_COUNT
JSR HALFSEC_COUNT
RTS

TWOSEC_DELAY:
JSR HALFSEC_COUNT
JSR HALFSEC_COUNT
*JSR HALFSEC_COUNT *cout out these since it TAKES SO LONG!!!
*JSR HALFSEC_COUNT
RTS

NO_KEY_PRESSED:
LDD #$0000
TBEK
STD KEYPRESSED *clear keypressed flag, since none hit
LDAB #$0F
TBEK
RTS *return-this should be only RTS besides one in CHECK_KEYS

************************************************************************************
* *
* SAMPLE DEPTH SENSORS -Sampling subroutines for the A/D *
* Raw data goes into DISC_READING (for the discs) and the front *
* sensor into FRONT_READING. These are then translated into *
* 1000's of an inch, and corrected for the angle from 90 degrees.*
* This is stored as SEED_DEPTH_ERROR, which is the error in (true) *
* 1000's of an inch from ground level. SIGN_FLAG is set when the *
* error is above ground level, cleared if below. *
************************************************************************************

SAMPLE:
LDAB #$00
TBEK
LDE #$00
STE ADCMCR *for stop/freeze/supervisor modes-set to normal
LDE #$0003
STE ADCTL0 *8-bit, 8 ADC clock cycles
LDE #$0020 *8-bit resolution (mode 2) (mode2:4 inputs-0020)
*this does conversions on 4 inputs, starting with
*the one specified in bits CA-CD. So, AD0 in RJURR0,
*and AD1 supposed to be in RJURR1.
STE ADCTL1 *Set port F data registers. This starts conversion.
NOP *slight delay to allow conversion to complete
Appendix C: Controller Source Code

*speed of the ADC is 1/8 of chip clock speed (set in
*the STS bits of ADCTRL). It takes 16 ADC cycles to
*complete an 8-bit conversion, or 16*8=128 16.7 MHZ clocks

LDE #$000A

ADC_DELAY_LOOP: *for about 150 clock cycles, to allow for the ADC to finish
SUBE #$0001
BNE ADC_DELAY_LOOP
LDE RJurRO *right justified result register 0-result in 8 LSB's
LDAB #$01
TBEK
STE DISC_READING
TSTW ADC_CHANNEL
BNE ONE_CHANNEL
LDAB #$0F
TBEK
LDE RJurRI *right justified result register 1-result in 8 LSB's
LDAB #$01 *REGISTER FOR ADC INPUT #1
TBEK
STE FRONT_READING *store front sensor reading, no angle correction
JSR ANGLE_CORRECTION_FRONT *correct for angle of sensor to 90 degrees
*input to algorithm is in acc E, with answer put in E
STE CORRECTED_FRONT_READING
*LDE CORRECTED_FRONT_ZERO
*SUBE CORRECTED_FRONT_READING
***ASRE ***SHIFT RIGHT TO INC. ACCURACY
*STE CORRECTED_FRONT_READING *angle of sensor now compensated for
LDE DISC_READING
JSR ANGLE_CORRECTION *correct for angle of sensor to 90 degrees
 *input to algorithm is in acc E, with answer put in E
STE CORRECTED_DISC_READING *angle now compensated for
SUBE CORRECTED_FRONT_READING

ONE_CHANNEL:
STE CORRECTED_DEPTH_READING *store the raw depth data
LDE CORRECTED_FIELD_ZERO *load the zero point data (seed depth setting)
SUBE CORRECTED_DEPTH_READING *field_zero - depth_reading = depth in hex
*SUBE CORRECTED_FIELD_ZERO
STE SEED_DEPTH_ERROR *store difference (depth)
in thousands/inch, (in hex) so can be converted directly
TSTE LBMI ABOVE_GROUND *if above zero point, jump to above_ground
CLRD STD SIGN_FLAG *if seed depth is above the ground, clear sign_flag
LDAB #$0F
TBEK
RTS *go back to MAIN program loop

ANGLE_CORRECTION: *the original ADC readings remain in DISC_READING and
*FRONT_READING. The corrected readings are for displaying
*on the LCD and for the fuzzy routine, to ensure corrections
*are made based on the depth, not just the sensor position
*(as are the routines for calibration).
CLRW SIGN_FLAG
SUBE DISC_NINETY_DEG_POS *subtract zero position from reading
TSTE *see if answer was negative
BPL POS_ANGLE *branch here if answer is positive
NEGE *2's complement E
RESUME_ANGLE_CORRECTION:
XGEX
LDY #$SINE_TABLE *now we have the sine of the sensor angle at 6°
Appendix C: Controller Source Code

XGEY
CLRA
LDAB E X
LDE HYPOTENEUSE_DISC  *multiply by the hypotenuse
EMUL
TDEM
ASRM
ASRM
ASRM
ASRM
ASRM
ASRM
ASRM
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
TMET
TSTW SIGN_FLAG
BEQ BELOW_GRND_ADJUST
CLRW SIGN_FLAG
RTS

BELOW_GRND_ADJUST:
NEGE *2's comp E (0000 - E, not FFFF - E as in COME)
CLRW SIGN_FLAG
RTS *later.

POS_ANGLE:
LDD #FFFF
STD SIGN_FLAG
BRA RESUME_ANGLE_CORRECTION

ANGLE_CORRECTION_FRONT: ****SET UP FOR THE FRONT_READING FOR MORE ACCURACY****
*the original ADC readings remain in DISC_READING and
*FRONT_READING. The corrected readings are for displaying
*on the LCD and for the fuzzy routine, to ensure corrections
*are made based on the depth, not just the sensor position
*(as are the routines for calibration).
CLRW SIGN_FLAG
SUBE FRONT_90_READING *subtract zero position from reading
TSTE *see if answer was negative
BPL POS_ANGLE1 *branch here if answer is positive
NEGE *2's complement E
RESUME_ANGLE_CORRECTION_FRONT:
XGEY
LDY #$SINE_TABLE  *now we have the sine of the sensor angle at 6°
Appendix C: Controller Source Code

LDAB E,X
LDE HYPOTENUSE_FRONT *multiply by the hypoteneuse
EMUL
TEDM
ASRM
*divide by 256, since sine is a fraction
ASRM
ASRM
ASRM
ASRM
ASRM
ASRM
ASRM
ASRM
ASLM
*now, shift AM left 16 places for transferring to E
ASLM
*The TMET and TMER commands only transfer the
ASLM
*upper 16 bits of AM to E, so the answer has to be
ASLM
*there.
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
ASLM
TMET
TSTW SIGN_FLAG
BEQ BELOW_GRND_ADJUST1
CLRW SIGN_FLAG
RTS
BELOW_GRND_ADJUST1:
NEGE *2's comp E (0000 - E, not FFFF - E as in COME)
CLRW SIGN_FLAG
RTS *later.

POS_ANGLE1:
LDD #$FFFF
STD SIGN_FLAG
BRA RESUME_ANGLE_CORRECTION_FRONT

ABOVE_GROUND:
NEGE
STE SEED_DEPTH_ERROR *save this negative number
LDD #$FFFF
STD SIGN_FLAG *set the sign flag to indicate a negative number
RTS *(ABOVE_GROUND)

*******************************************************************************
*                          DISPLAY DEPTH AND MESSAGES ON LCD                *
*                          -converts SEED_DEPTH_ERROR to units                *
*                          and writes them to the LCD                        *
*******************************************************************************

DISPLAY_DEPTH:
***convert to decimal output***
LDAB #$01
Appendix C: Controller Source Code

TBEK
LDD  SEED_DEPTH_ERROR  *load D with divisor
TSTW  UNITS  *determine whether inches or centimetres
LBQ  CENTIMETRES

DECIMAL_CONV0:  *convert the MSB to decimal
    LDX  #$000A
    IDIV
    STD  DEPTH_1000TH  *remainder from modulo division is 1/1000
    LDX  #$000A
    IDIV
    STD  DEPTH_100TH  *remainder from modulo division is 1/100
    LDX  #$000A
    IDIV
    STD  DEPTH_10TH  *remainder from modulo division is 1/10
    LDX  #$000A
    IDIV
    STD  DEPTH_INCHES  *says inches, but should read “integers”
    JSR  DECIMAL_CONV1
    LBA  WRITE_DEPTH

DECIMAL_CONV1:  *convert the second digit of the hex frac to decimal
    LDE  DEPTH_1000TH
    SUBE  #$0005
    BPL  ADD_TO_DEPTH100TH
    LDE  DEPTH_100TH
    SUBE  #$05
    BPL  ADD_TO_DEPTH10TH
    RTS

ADD_TO_DEPTH100TH:
    INCW  DEPTH_100TH
    LDE  DEPTH_100TH
    SUBE  #$0005
    BPL  ADD_TO_DEPTH10TH
    RTS

ADD_TO_DEPTH10TH:
    INCW  DEPTH_10TH
    LDE  DEPTH_10TH
    SUBE  #$000A
    BPL  ADD_TO_INCHES
    RTS

ADD_TO_INCHES:
    CLR
    STE  DEPTH_10TH
    INCW  DEPTH_INCHES
    LDE  DEPTH_INCHES
    SUBE  #$000A
    BPL  ADD_TO_10UNITS
    RTS

ADD_TO_10UNITS:
Appendix C: Controller Source Code

CLRE
STC DEPTH_INCHES
INCW DEPTH_10INCHES
RTS

WRITE_DEPTH:  *send depth numbers to the LCD
LDAB #$01
TBEK
TSTW OP_DISPLAY_ON
BNE DISPLAY_IS_ON
TSTW UNITS
LBEQ METRIC
LDX #OPERATING_SCREEN_INCHES  *point to lcd message for setting raised pos

WRITE_OPERATING_SCREEN:
JSR SEND_STRING  *print it to the lcd screen
LDAB #$01
TBEK
LDE #$FFFF
STC OP_DISPLAY_ON

DISPLAY_IS_ON:
TSTW DEPTH_10INCHES  *see if there is a 10 inches number
LBEQ WRITE_CLEAR  *if zero, branch to write inches
LDAB #$0F
TBEK
LDAA #$87  *Location for the 7th lcd display position
STA GPTPDR
JSR LCD_INST
LDAB #$01
TBEK
LDAA DEPTH_10INCHES+1  *write 10 inches to LCD
ADDA #$30  *convert to ascii
LDAB #$0F
TBEK
LDAB GPTPDR
CBA
BEQ WRITE_INCH  *don’t write to screen unless it’s changed
STA GPTPDR
JSR LCD_DATA  *put it on the screen

WRITE_INCH:
LDAB #$0F
TBEK
LDAA #$C0  *Location for the 8th lcd display position
STA GPTPDR
JSR LCD_INST
LDAB #$01
TBEK
LDAA UNITS  *determine whether inches or centimetres
TSTA
LBGT METRIC  *if bit is not set, use the metric units
LDAA DEPTH_INCHES+1  *write inches to LCD
ADDA #$30  *convert to ascii
LDAB #$0F
TBEK
LDAB GPTPDR
CBA
BEQ CHECK_PERIOD  *don’t write to screen unless it’s changed
STA GPTPDR
JSR LCD_DATA  *put it on the screen

CHECK_PERIOD:
LDAA #$C1  *Location for the 9th lcd display position
STA GPTPDR
Appendix C: Controller Source Code

```
JSR LCD_INST
LDAA #$2E  *write a period to the screen
LDAB GPTPDR
CBA
BEQ CHECK_TENTH
STAA GPTPDR
JSR LCD_DATA

CHECK_TENTH:
LDAA #$C2  *Location for the 10th lcd display position
STAA GPTPDR
JSR LCD_INST
LDAB #$0F
TBEK
LDAA DEPTH_10TH+1  *write 1/10 inches to LCD
ADDA #$30  *convert to ascii
LDAB #$0F
TBEK
LDAB GPTPDR
CBA
BEQ CHECK_SIGN
STAA GPTPDR
JSR LCD_DATA

CHECK_SIGN:
LDAB #$01
TBEK
*determine whether the numbers are negative or positive
LDAB SIGN_FLAG+1  *check sign flag for negative
TSTB
BEQ BELOW_GRND  *if positive (not above ground) branch here
TSTW DEPTH_1INCHES
LBNE SEVENTH_PLUS_POS_WRITE
LDAA #$87  *Location for the 7th lcd display position
LDAB #$0F
TBEK
STAA GPTPDR
JSR LCD_INST
LDAA #$2B  *load a plus sign
LDAB GPTPDR
CBA
BEQ PLUS_IS_THERE
STAA GPTPDR
JSR LCD_DATA  *put it on the screen

PLUS_IS_THERE:
LDAB #$01
TBEK
TSTW DEPTH_1INCHES
LBEQ SEVENTH_PLUS_POS_CLR
CLRW SIGN_FLAG
RTS  *go back to the MAIN subroutine

SEVENTH_PLUS_POS_WRITE:
LDAA #$86  *Location for the 7th lcd display position
LDAB #$0F
TBEK
STAA GPTPDR
JSR LCD_INST
LDAA #$2B  *load a plus sign
LDAB GPTPDR
CBA
BEQ PLUS_IS_THERE2
STAA GPTPDR
```
Appendix C: Controller Source Code

JSR  LCD_DATA  *put it on the screen
PLUS_IS_THERE2:
  LDAB  #$01
  TBEK
  CLRW  SIGN_FLAG
  RTS       *go back to the MAIN subroutine

BELOW_GRND:
  CLRW  SIGN_FLAG
  TSTW  DEPTH_10INCHES
  LBEQ  SEVENTH_PLUS_POS_CLR
  LDAA  #$87  *Location for the 7th lcd display position
  LDAB  #$0F
  TBEK
  STAA  GPTPDR
  JSR  LCD_INST
  LDAA  #$20  *load a space
  STAA  GPTPDR
  JSR  LCD_DATA  *put it on the screen
  RTS

SEVENTH_PLUS_POS_CLR
  LDAA  #$86  *Location for the 7th lcd display position
  LDAB  #$0F
  TBEK
  STAA  GPTPDR
  JSR  LCD_INST
  LDAA  #$20  *load a space
  STAA  GPTPDR
  JSR  LCD_DATA  *put it on the screen
  LDAB  #$01
  TBEK
  CLRW  SIGN_FLAG
  LDAB  #$0F
  TBEK
  RTS

WRITE_CLEAR:
  LDAB  #$0F
  TBEK
  LDAA  #$87  *Location for the 9th lcd display position
  STAA  GPTPDR
  JSR  LCD_INST
  LDAA  #$20  *write a space to the screen
  LDAB  GPTPDR
  CBA
  LBEQ  WRITE_INCH  *if already a space, no need to write one
  LDAB  #$01
  TBEK
  TSTW  SIGN_FLAG
  LBNE  WRITE_INCH
  LDAB  #$0F
  TBEK
  STAA  GPTPDR
  JSR  LCD_DATA
  LBRA  WRITE_INCH  *write the inch position

CENTIMETRES:  *calculate the depth in centimetres-depth in acc D in inches (in hex)
  LDE  #$100
  EMUL
  LDX  #$0064
Appendix C: Controller Source Code

EDIV    *depth is now in hundredths/inch  
XGDX    *depth in 1,000ths/cm (nanometers!!) in acc E  
LBRA    DECIMAL_CONV0 *divide by 1,000, convert to decimal, write to screen  

METRIC:  
LDX    #OPERATING_SCREEN_CM *point to lcd message for setting raised pos  
LBRA    WRITE_OPERATING_SCREEN  

SEND_STRING:  *subroutine to send out the entire ASCII string  
LDAB    #$0F  
TBEK  
LDAA    #$80 *SET DDRAM ADDRESS  
STAA    GPTPD  
JSR    LCD_INST  
STRING_PREP:  
LDE    #$0008  
SEND_NEXT:  *get next byte in string as pointed to by IX  
LDAB 0,X *load 7 into E to implement decrement counter for  
*display position on lcd  
LBEQ    STRING_DONE *if B=00, then goto delay between messages  
STAB    GPTPD *transmit one ASCII character to the screen  
JSR    LCD_DATA  
AIX    #$01 *increment IX to point to the next byte  
SUBE    #$0001  
LBEQ    NEXT_8_LCD  
LBRA    SEND_NEXT  

NEXT_8_LCD:  
LDAA    #$C0 *Location for the 9th lcd display position  
STAA    GPTPD  
JSR    LCD_INST  
LBRA    STRING_PREP  

STRING_DONE:  
LDAA    #$02 *turns on display, cursor on and blinking  
STAA    GPTPD  
JSR    LCD_INST  
RTS  

******************************************************************************  
ERROR_CONVERT:  *change error in 1000’s inch to 10’s in hex--  
LDAB    #$01 *add or subtract from 3F (0 error position) to  

* ERROR CONVERSION ROUTINE convert SEED_DEPTH_ERROR (in 1000’s of an inch for LCD  
* display) to a form compatible with the fuzzy routine (8 bit format).  
* OPERATING_DEPTH (in 1000’s inch) is also converted by the same factor.*  
* These are then added or subtracted to reflect the error from the operating*  
* depth.  
* Convert so $0A = 1 inch in difference. This can be changed (ie, to *  
* $14 = 1 in) in order to provide scaling down for the 25 gallon per *  
* minute hydraulic valve spool. This requires dividing the error in *  
* 1000’s of an inch into error in 1/10th inch or so. This is then added*  
* or subtracted from the 0 error value to provide the appropriate FRONT.*  
* The answer is put into INPUTS and the RAISE_DEPTH_FLAG is set or cleared*  
* to reflect whether the discs have to be raised or lowered (this is for *  
* the correction factor for the PWMA signal to reflect the transfer *  
* function of the hydraulic valve.  
*  
******************************************************************************
Appendix C: Controller Source Code

TBEK *put in form fuzzy engine understands (range 0-7E)
LDD SEED_DEPTH_ERROR
LDX #$64 *LOAD IX WITH decimal 100 (100 SINCE RANGE IS 0-7F NOT FF)
IDIV *integer divide the error in 1000's of an inch
XGDX *put answer in D
STD ERROR_CONV_FORMAT
LDD OPERATING_DEPTH *load operating depth in 1000's of an inch
LDX #$64 *LOAD IX WITH decimal 100 (100 SINCE RANGE IS 0-7F NOT FF)
IDIV *integer divide the error in 1000's of an inch
XGDX *so divided by the same factor as SEED_DEPTH_ERROR
STD OPERATING_DEPTH_CONV_FORMAT *operating depth now converted
*****BMI FINISH *branch if it's negative (ie-depth below set point)
*****LDX #$C8 *LOAD IX WITH decimal 200 (200 SINCE RANGE IS 0-7F NOT FF)
*****LDX #$190 *LOAD IX WITH decimal 400 (400 SINCE RANGE IS 0-7F NOT FF)
*****LDX #$1F4 *LOAD IX WITH decimal 500 (500 SINCE RANGE IS 0-7F NOT FF)
TSTW SIGN_FLAG *see if disks are above ground or not
BMI SUBTRACT *if negative, branch to subtract (disks are above grnd)
*****(disks below ground:)*****
LDD ERROR_CONV_FORMAT *sub. error (fr ground level) fr operating depth
NEGD *neg since below ground
ADD OPERATING_DEPTH_CONV_FORMAT
BMI FINISH *go here if depth below ground, and below set point
ADD #$007F *add to 0 error (0 error=3F) to make it bigger than 3F
STAB INPUTS *store for ERROR in fuzzy decision
LDE #$FFFF *clear flag since the discs have to be lowered
STE RAISE_DEPTH_FLAG *clear flag since discs are shallower than
RTS *they should be

FINISH:
ADD #$007F *adding a negative number makes error less than 3F
STAB INPUTS *store for ERROR in fuzzy decision
CLRE *set flag since the discs have to be raised
STE RAISE_DEPTH_FLAG *set flag since discs are deeper than they should be
RTS

SUBTRACT: *****(disks above ground:)*****
LDD ERROR_CONV_FORMAT *sub. error (fr ground level) fr operating depth
ADD OPERATING_DEPTH_CONV_FORMAT
ADD #$007F *to make it 10's inch in hex-ans in IX
STAB INPUTS *put error in 10's inch in ERROR for fuzzy decision
LDE #$FFFF
STE RAISE_DEPTH_FLAG *clear flag-discs are above ground and need to
RTS *be lowered

******************************************************************************
* CALCULATE ERROR CHANGE *
******************************************************************************
ERROR_BUFFER: ;each input/output will be 10 counts from each other
LDAB #$01
TBEK
TBZK
LDE #ERROR_TABLE
XGEZ
CLRE
CLRD
LDAB CIRC_ERROR_BUFFER ;load present error buffer index
TDE
CLRD

-- C-30 --
Appendix C: Controller Source Code

LDAB INPUTS ;load present error (10's inch) from set point
STAB EZ ;put in circular error buffer-10 bytes capacity
*AL #50001 ;increment Z to next position
DEC CIRC_ERROR_BUFFER ;decrement index
BEQ ZERO_Z_AND_BUFF ;if 10 locations covered, start at 0 again
JSR ERROR_CALC
LDAB #00
TBZK
RTS ;done-location to load as chg in error is Z + A

ZERO_Z_AND_BUFF:
LDAA #509
STAA CIRC_ERROR_BUFFER ;reset index counter
LDE #$ERROR_TABLE
XGEZ
JSR ERROR_CALC
LDAB #500
TBZK
RTS

ERROR_CALC: ;calculate the change in error
LDAB #$01
TBEK
CLRD
LDAB CIRC_ERROR_BUFFER
JSR ZERO_CHECK ;see if buffer is 1
CLRE
LDE
LDAA EZ ;load current error
SBA ;subtract error from 10 samples ago
ADDA #$3F ;SET ERROR IN RELATION TO 0
STAA ERROR_CHANGE ;store answer as the change in error
RTS

ZERO_CHECK:
DECB
BEQ ZERO_FIX
RTS

ZERO_FIX:
ADDB #$09
RTS

***************************************************************************
* *
* CALCULATE FUZZY LOGIC DECISION *
* *
***************************************************************************
* Change_in_Depth is stored in the HYD_CONTROL+1

ORG $1600
IMFTR
FDB MF1
FDB MF2
FDB MF3
FDB MF4
FDB MF5
FDB MF6
FDB MF7
FDB MF8
FDB MF9
Appendix C: Controller Source Code

FDB MF10
FDB MF11
FDB MF12
FDB MF13
FDB MF14
FDB MF15

MF1
FCB $02
FCB $0F
FCB $00
FCB $00
FCB $FF
FCB $2F
FCB $7C
FCB $FF
FCB $50
FCB $00
FCB $00
FCB $FA
FCB $00
FCB $00
FCB $FF

MF2
FCB $04
FCB $1F
FCB $00
FCB $00
FCB $00
FCB $2F
FCB $45
FCB $00
FCB $6A
FCB $B1
FCB $FF
FCB $81
FCB $00
FCB $00
FCB $FA
FCB $00
FCB $00
FCB $FF

MF3
FCB $04
FCB $1F
FCB $00
FCB $00
FCB $00
FCB $41
FCB $42
FCB $00
FCB $7F
FCB $44
FCB $FF
FCB $BB
FCB $00
FCB $00
FCB $FA
FCB $00
FCB $00
FCB $FF

MF4
Appendix C: Controller Source Code

FCB $04
FCB $1F
FCB $00
FCB $00
FCB $00
FCB $7E
FCB $75
FCB $00
FCB $A1
FCB $55
FCB $FF
FCB $D1
FCB $00
FCB $00
FCB $FA
FCB $00
FCB $00
FCB $FF

MF5
FCB $08
FCB $17
FCB $00
FCB $00
FCB $00
FCB $AF
FCB $5D
FCB $00
FCB $DB
FCB $00
FCB $FF
FCB $9
FCB $FF
FCB $FF
FCB $FF
FCB $FA
FCB $00
FCB $00
FCB $FF

MF6
FCB $02
FCB $0F
FCB $00
FCB $00
FCB $FF
FCB $1A
FCB $B1
FCB $FF
FCB $31
FCB $00
FCB $00
FCB $30
FCB $00
FCB $00
FCB $00
FCB $FF

MF7
FCB $04
FCB $1F
FCB $00
FCB $00
FCB $00
FCB $1A
FCB $7C
Appendix C: Controller Source Code

FCB $00
FCB $2E
FCB $F0
FCB $FF
FCB $3F
FCB $00
FCB $00
FCB $80
FCB $00
FCB $00
FCB $FF

MF8
FCB $04
FCB $1D
FCB $00
FCB $00
FCB $00
FCB $2E
FCB $11
FCB $00
FCB $3D
FCB $B9
FCB $FF
FCB $53
FCB $00
FCB $00
FCB $80
FCB $00
FCB $00
FCB $FF

MF9
FCB $04
FCB $1F
FCB $00
FCB $00
FCB $00
FCB $3F
FCB $CC
FCB $00
FCB $53
FCB $D7
FCB $FF
FCB $66
FCB $00
FCB $00
FCB $80
FCB $00
FCB $00
FCB $FF

MF10
FCB $00
FCB $0F
FCB $00
FCB $00
FCB $00
FCB $51
FCB $C2
FCB $00
FCB $66
FCB $00
FCB $FF
Appendix C: Controller Source Code

```
FCB $80
FCB $00
FCB $FF
FCB $FF

MF11
FCB $00
FCB $01
FCB $3B
FCB $00
FCB $FF
FCB $FF

MF12
FCB $00
FCB $01
FCB $5D
FCB $00
FCB $FF
FCB $FF

MF13
FCB $00
FCB $01
FCB $7F
FCB $00
FCB $FF
FCB $FF

MF14
FCB $00
FCB $01
FCB $A1
FCB $00
FCB $FF
FCB $FF

MF15
FCB $00
FCB $01
FCB $C3
FCB $00
FCB $FF
FCB $FF
EVEN
FUZZYTEST:
JSR INIT
LDA A INPUTS+$00
STAA IN_BUF
LDA B #$01
JSR MF_SUB
STAA TVLST+$00
LDA B #$02
JSR MF_SUB
STAA TVLST+$01
LDA B #$03
JSR MF_SUB
STAA TVLST+$02
LDA B #$04
JSR MF_SUB
STAA TVLST+$03
LDA B #$05
JSR MF_SUB
STAA TVLST+$04
LDA A ERROR_CHANGE *load error change
STAA IN_BUF
```
Appendix C: Controller Source Code

LDAB #06
JSR MF_SUB
STAA TVLST+$05
LDAB #07
JSR MF_SUB
STAA TVLST+$06
LDAB #08
JSR MF_SUB
STAA TVLST+$07
LDAB #09
JSR MF_SUB
STAA TVLST+$08
LDAB #0A
JSR MF_SUB
STAA TVLST+$09
LDAA TVLST+$04
STAA AND_REG
LDAA TVLST+$09
JSR MIN1_OP
LDAA AND_REG
STAA OR_REG
LDAA TVLST+$04
STAA AND_REG
LDAA TVLST+$08
JSR MIN1_OP
JSR MAX2_OP
LDAA TVLST+$04
STAA AND_REG
LDAA TVLST+$07
JSR MIN1_OP
JSR MAX2_OP
LDAA TVLST+$03
STAA AND_REG
LDAA TVLST+$09
JSR MIN1_OP
JSR MAX2_OP
LDAA TVLST+$02
STAA AND_REG
LDAA TVLST+$09
JSR MIN1_OP
JSR MAX2_OP
LDAB #3B
JSR DFZ_C
LDAA TVLST+$04
STAA AND_REG
LDAA TVLST+$06
JSR MIN1_OP
LDAA AND_REG
STAA OR_REG
LDAA TVLST+$03
STAA AND_REG
LDAA TVLST+$08
JSR MIN1_OP
JSR MAX2_OP
LDAA TVLST+$03
STAA AND_REG
LDAA TVLST+$07
JSR MIN1_OP
JSR MAX2_OP
LDAA TVLST+$02
STAA AND_REG
Appendix C: Controller Source Code

```
LDAA TVLST+508
JSR MIN1_OP
JSR MAX2_OP
LDAB #$5D
JSR DFZ_C
LDAA TVLST+504
STAA AND_REG
LDAA TVLST+505
JSR MIN1_OP
LDAA AND_REG
STAA OR_REG
LDAA TVLST+503
STAA AND_REG
LDAA TVLST+506
JSR MIN1_OP
JSR MAX2_OP
LDAA TVLST+503
STAA AND_REG
LDAA TVLST+505
JSR MIN1_OP
JSR MAX2_OP
LDAA TVLST+502
STAA AND_REG
LDAA TVLST+507
JSR MIN1_OP
JSR MAX2_OP
LDAA TVLST+501
STAA AND_REG
LDAA TVLST+509
JSR MIN1_OP
JSR MAX2_OP
LDAA TVLST+500
STAA AND_REG
LDAA TVLST+509
JSR MIN1_OP
JSR MAX2_OP
LDAB #$7F
JSR DFZ_C
LDAA TVLST+502
STAA AND_REG
LDAA TVLST+506
JSR MIN1_OP
LDAA AND_REG
STAA OR_REG
LDAA TVLST+501
STAA AND_REG
LDAA TVLST+508
JSR MIN1_OP
JSR MAX2_OP
LDAA TVLST+501
STAA AND_REG
LDAA TVLST+507
JSR MIN1_OP
JSR MAX2_OP
LDAA TVLST+501
STAA AND_REG
LDAA TVLST+506
JSR MIN1_OP
JSR MAX2_OP
LDAA TVLST+500
STAA AND_REG
```
Appendix C: Controller Source Code

LDAA TVLST+\$08
JSR MIN1_OP
JSR MAX2_OP
LDAB #$A1
JSR DFZ_C
LDAA TVLST+\$02
STAA AND_REG
LDAA TVLST+\$05
JSR MIN1_OP
LDAA AND_REG
STAA OR_REG
LDAA TVLST+\$01
STAA AND_REG
LDAA TVLST+\$05
JSR MIN1_OP
JSR MAX2_OP
LDAA TVLST+\$00
STAA AND_REG
LDAA TVLST+\$07
JSR MIN1_OP
JSR MAX2_OP
LDAA TVLST+\$00
STAA AND_REG
LDAA TVLST+\$06
JSR MAX1_OP
JSR MAX2_OP
LDAA TVLST+\$00
STAA AND_REG
LDAA TVLST+\$05
JSR MIN1_OP
JSR MAX2_OP
LDAB #$C3
JSR DFZ_C
JSR DIV_SUB
LDX #$00
LDAA OUT_REG *load A
CLRW HYD_SIGNAL *clear E to ensure MSB is clear
STAA HYD_SIGNAL+1 *store hydraulic signal value to LSB (7F=no change)
RTS
MIN1_OP CMPA AND_REG
BCC MIN1_E
STAA AND_REG
MIN1_E RTS
MAX2_OP LDAA AND_REG
CMPA OR_REG
BCS MAX2_E
STAA OR_REG
MAX2_E RTS
DFZ_C LDAA OR_REG
MUL
TAB
CLR A
ADDD TUH_REG
STD TUH_REG
CLR A
LDAB OR_REG
ADDD SUH_REG
STD SUH_REG
RTS
DIV_SUB LDD TUH_REG
LDX SUH_REG

-C-38-
Appendix C: Controller Source Code

FDIV
BCS DIV_1
XGDX
STAA OUT_REG
RTS
DIV_1  CLR OUT_REG
RTS
INIT  CLR TUL_REG
CLR TUH_REG
CLR SUL_REG
CLR SUH_REG
RTS
MF_SUB  ASLB
SUBB #$2
CLRA
ADDD #MF_OFF
XGDX
LDX $0,X
LDAA $0,X
STAA MF1_REG
INX
LDAA $0,X
STAA MF2_REG
LSL MF1_REG
LSL MF2_REG
INX
MF_1  XGDX
ADDD #$3
XGDX
LDAA $0,X
LSR MF1_REG
LSR MF2_REG
CMPA IN_BUF
BLO MF_1
LSR MF1_REG
BCS MF_2
JSR MF_S2
ADDB MF1_REG
BCC MF_4
LDAB #$FF
MF_4  TBA
RTS
MF_S2  DEX
LDAA $0,X
STAA MF1_REG
LSR MF2_REG
BCC MF_3
JSR MF_S1
LSRD
LSRD
LSRD
LSRD
TSTA
BEQ MF_7
LDAB #$FF
MF_7  RTS
MF_3  JSR MF_S1
TSTA
BEQ MF_5
LDAB #$FF
MF_5  RTS
Appendix C: Controller Source Code

MF_S1 DEX
LDA $0,X
STAA MF2_REG
LDAA IN_BUF
DEX
SUBA $0,X
LDAB MF2_REG
MUL
RTS

MF_2 JSR MF_S2
STAB MF2_REG
LDAA MF1_REG
SUBA MF2_REG
BCC MF_6
LDAA #$00
MF_6 RTS

*****************************************************************************
* SEND OUT SIGNAL TO HYDRAULIC CONTROL signals are adjusted to compensate *
* for the transfer function of the hydraulic valve *
*****************************************************************************

HYD_CONT:
LDAB #$01
TBEK ****FIRST--invert the hyd signal to reflect valve action
   *starts lifting at $77-max at $6B
   *starts lowering at $8F-max at $9B
LDAA HYD_SIGNAL+1 *load output from fuzzy engine
LDAB #$7F *load midpoint between 0 and FE
SBA *get the difference from the "middle"
BMI NEG_RESULT *beginning of control gain scaling-if negative, skip
   ****scaling for positive result of HYD_SIGNAL less than 7F
LSRA *scaling factor--divide by 2
NEGA *invert this difference
ABA *add it, to give equal magnitude in opposite direction
BRA SCALING_DONE *branch here when done the scaling
NEG_RESULT: *scaling for negative result of HYD_SIGNAL less than 7F
NEGA *invert this difference
LSRA *scaling factor--divide by 2
ABA *add it, to give equal magnitude in opposite direction

SCALING_DONE:
STAA HYD_SIGNAL+1
TSTW RAISE_DEPTH_FLAG *if set, discs must be lowered
BMI SUBTRACT_2 *IF FLAG NEGATIVE (discs must be lowered) BRANCH
   ****(disks need to be raised-below set point)****
LDE HYD_SIGNAL
SUBE #$000C *subtract 9 to bring 0 signal from 7F to 77
LDAB #$0F *since hyd valve won’t move at less than 89
TBEK
TED
STAB PWMA *
RTS
   ****(disks need to be lowered-above set point)****

SUBTRACT_2:
LDE HYD_SIGNAL
ADDE #$000D *add C to bring 0 signal from 7F to 8D
LDAB #$0F *since hyd valve won’t move at more that 71
**Appendix C: Controller Source Code**

```
TBEK TED STAB PWMA RTS

*******************************************************************************
*SEND DATA TO THE PC FOR LOGGING RESULTS *
*******************************************************************************

SEND_DATA:
LDAB #$01
TBEK
LDX #DATA_TO_SEND ;point to the beginning of data to send
LDE #$000A ;load data transmit index for 10 data points
STE DATA_INDEX ;save location index
CLRE
STE DATA_TO_SEND ;put in temp array
STE DATA_TO_SEND+2 ;clear word
LDD INPUTS ;load error from seeding set point
STAA DATA_TO_SEND+1 ;put 8-bit data in proper order
LDE ERROR_CHANGE ;get change in error from 10 readings previous
STAA DATA_TO_SEND+3 ;put 8-bit data in proper order
LDE HYD_SIGNAL ;get fuzzy decision output
STE DATA_TO_SEND+4 ;put in temp array
LDAB #$0F
TBEK
LDE RJRR1 ;get front sensor 8-bit a/d value
LDAB #$01
TBEK
STE DATA_TO_SEND+6 ;put in temp array
LDAB #$0F
TBEK
LDE RJRR0 ;get disc sensor 8-bit a/d value
LDAB #$01
TBEK
STE DATA_TO_SEND+8 ;put in temp array
LDAB #$0F
TBEK
PC_READY1:
LDAB SCSR+1 ;check "Receive Data Register Full" flag-should have
BITB #$40 ;START signal from PC in buffer
BEQ PC_READY1 ;if not, go back and check again
LDBA SCCR+1 ;load receive register into B
SUBB #$0F ;see if if has received an $F (this is the START sig)
BNE PC_READY1

SEND_STRING2: ;subroutine to send out the entire ASCII string
LDAB #$0F
TBEK
LDAB #$01
TBXK
LDAB 0.X ;get next byte in string as pointed to by IX
TBA ;data byte now is transferred to acc A
LDAB #$00
TBXK
JSR SEND_CH ;go send out the byte
AIX #$01 ;increment IX to point to the next byte
LDAB #$01
```
AppendX C: Controller Source Code

TBEK
DECW DATA_INDEX ; decrement transmit counter
TSTW DATA_INDEX ; see if 6 samples had been taken
BNE SEND_STRING2 ; loop back and do next byte in string
LDAB #$0F
TBEK
RTS

SEND_CH: ; subroutine to send out one byte to SCI
LDAB SCSR ; check "Transmit Data Register Enable" bit, if 0, means
ANDB #$01 ; register TDR still has data to be sent to tx serial shifter
BEQ SEND_CH ; if TDR is not empty, go back to check it again
TAB ; data byte transferred to B-had to do this since only acc B transfers to K
LDAA #$00 ; clear A to send a full word to SCDR ($FFC0E)
STD SCDR ; transmit one ASCII character to the PC

TC_LOOP: ; test the TC bit (transfer complete)
LDAB SCSR+1
ANDB #$80
BEQ TC_LOOP ; continue to wait until TC is set
LDAB SCSR
ANDB #$01
BEQ TC_LOOP
LDAB #$0F
TBEK
RTS ; finish sending out byte

********************************************************************
* LCD MESSAGES  *
********************************************************************

EVEN
OPERATING_SCREEN_INCHES DC 'DEPTH  INCH',$00
EVEN
OPERATING_SCREEN_CM DC 'DEPTH  CM',$00
EVEN
RAISE_MESSAGE1 DC 'LIFT IMPLIMENT ',$00
EVEN
RAISE_MESSAGE2 DC 'IMPLIMENT RAISED',$00
EVEN
LOWER_MESSAGE DC 'TO SEED DEPTH ',$00
EVEN
SET_RAISE_MSG DC 'SET RAISED POSN ',$00
EVEN
HIT_KEY_MSG DC 'HIT ANY KEY ','$00
EVEN
ENTER_HEIGHT DC 'ENTER INCHES: ','$00
EVEN
ENTER_10TH_HEIGHT DC 'ENTER TENTH INCH ',$00
EVEN
ERR_PRESS_MSG0 DC 'OH SHIT, ERROR; ','$00
EVEN

-C-42-
Appendix C: Controller Source Code

ERR_PRESS_MSG1 DC 'TIME FOR A BEER',$.00
   EVEN
SET_OPER_MSG DC 'SET SEED DEPTH',$.00
   EVEN
ENTER_DEPTH DC 'ENTER INCHES:',$.00
   EVEN
ENTER_10TH_DEPTH DC 'ENTER TENTH INCH',$.00
   EVEN
FIELD0_SET_MSG DC 'SET SOIL SURFACE',$.00
   EVEN
LOWER_GRND_LEVEL DC 'LOWER TO GROUND',$.00
   EVEN
HIT_TO_STORE DC 'HIT KEY TO STORE',$.00
   EVEN
LEVEL0_SET_MSG DC 'DISC SENSOR CAL',$.00
   EVEN
LEVEL1_SET_MSG DC 'FRONT SENSOR CAL',$.00
   EVEN
LEVEL_SURF_CALIB DC 'ON LEVEL SURFACE',$.00
   EVEN
DISC_LEVEL_ZERO_MSG DC 'LOWER TO SURFACE',$.00
   EVEN
SIX_IN_CALIB0 DC 'RAISE DISKS UP',$.00
   EVEN
SIX_IN_CALIB1 DC 'SIX INCHES',$.00
   EVEN
GREETINGS DC 'Fuzzy Seeding',$.00
   EVEN
GREETINGS2 DC 'Depth Controller',$.00
   EVEN
CHOOSE_MODE_MSG DC 'Choose Scan Mode',$.00
   EVEN
MODE_A DC 'Press 0 for dual',$.00
   EVEN
MODE_B DC 'sensors, 1 for ',$.00
   EVEN
MODE_C DC 'single sensor',$.00
   EVEN
CHOOSE_FUNCTION_MSG DC 'Choose Function:',$.00
   EVEN
PICK_MODE_OR_RAISE1 DC 'Press 0 to pick ',$.00
   EVEN
PICK_MODE_OR_RAISE2 DC 'sensor modes, 1 ',$.00
   EVEN
PICK_MODE_OR_RAISE3 DC 'to set seeder ',$.00
   EVEN
PICK_MODE_OR_RAISE4 DC 'raised position,',$.00
   EVEN
PICK_MODE_OR_RAISE5 DC '2 to set units ',$.00
   EVEN
PICK_UNIT_MESSAGE1 DC '0 for imperial ',$.00
   EVEN
PICK_UNIT_MESSAGE2 DC 'hit 1 for METRIC',$.00
   EVEN
PICK_CALIB_90_OR_6IN1 DC 'Press 0 to set ',$.00
   EVEN
PICK_CALIB_90_OR_6IN2 DC 'sensrs 90 deg to',$.00
   EVEN
PICK_CALIB_90_OR_6IN3 DC 'linkages, 1 to ',$.00
   EVEN
PICK_CALIB_90_OR_6IN4 DC 'cal disk sensor',$.00
**Appendix C: Controller Source Code**

```
EVEN
PICK_CALIB_90_OR_6IN5 DC '2-cal front sens',00
EVEN
CALIB_90_DEG1 DC 'PROCEDURE TO ',00
EVEN
CALIB_90_DEG2 DC 'STORE THE SENSOR',00
EVEN
CALIB_90_DEG3 DC 'POSITION 90 DEG ',00
EVEN
CALIB_90_DEG4 DC 'TO DISK LINKAGE ',00
EVEN
SET_TO_90_MSG1 DC 'Position disk so',00
EVEN
SET_TO_90_MSG2 DC 'sensor arm is 90',00
EVEN
SET_TO_90_MSG3 DC 'deg to disk arm ',00
EVEN

*******************************************************************************
* *
* TABLE FOR KEY IDENTIFICATION *
* *
*******************************************************************************

KEY_TABLE:
  DC.B $EE * 0
  DC.B $DE * 1
  DC.B $BE * 2
  DC.B $7E * 3
  DC.B $ED * 4
  DC.B $DD * 5
  DC.B $BD * 6
  DC.B $7D * 7
  DC.B $EB * 8
  DC.B $DB * 9
  DC.B $BB * A
  DC.B $7B * B
  DC.B $E7 * C
  DC.B $D7 * D
  DC.B $B7 * E
  DC.B $77 * F

*******************************************************************************
* *
* TABLE OF SINES *
* *
*******************************************************************************

SINE_TABLE:
  FCB $00,$01,$03,$05,$06,$08,$09,$0B
  FCB $0D,$0E,$0E,$11,$12,$14,$16,$17
  FCB $19,$1A,$1C,$1E,$1F,$20,$22,$23
  FCB $25,$26,$28,$29,$2B,$2C,$2E,$2F
  FCB $30,$32,$33,$35,$36,$38,$39,$3A
  FCB $3C,$3D,$3F,$40,$41,$43,$44,$45
  FCB $47,$48,$49,$4A,$4C,$4D,$4E,$4F
  FCB $51,$52,$53,$54,$55,$57,$58,$59
  FCB $5A,$5B,$5C,$5D,$5E,$5F,$60,$61
  FCB $62,$63,$64,$65,$66,$67,$68,$69
  FCB $6A,$6B,$6C,$6D,$6E,$6F,$6F
  FCB $70,$71,$72,$72,$73,$74,$74,$75
```
Appendix C: Controller Source Code

```
FCB \$75,\$76,\$76,\$77,\$78,\$78,\$79
FCB \$79,\$7A,\$7B,\$7B,\$7C,\$7C
FCB \$7C,\$7D,\$7D,\$7D,\$7E,\$7E,\$7E
FCB \$7E,\$7E,\$7E,\$7E,\$7E,\$7E,\$7E
FCB \$7E,\$7F,\$7F,\$7F,\$7F,\$7F,\$7F,\$7F extra row

ORG \$10000 *Equates start at \$1000

MF_OFF EQU \$1600
INPUTS RMB \$02
TVLSR RMB \$0F
IN_BUF RMB 1

MFI_REG RMB 1
MFI_REG RMB 1
AND_REG RMB 1
OR_REG RMB 1

TUH_REG RMB 1
TUL_REG RMB 1
SUH_REG RMB 1
SUL_REG RMB 1
OUT_REG RMB 1
BUFF_REG RMB 1

IRQ_COUNT RMB \$02 *10002 *Location of number of IRQ’s for debounce
KEY RMB \$02 *10008 *Hex value of the key being pressed
KEY_ERR_COUNTER RMB \$02 *1000A *Counter to prevent non-table key id from
*causing an endless loop
INDEX RMB \$02 *1000C *mem location for keypad values for comparison
*to acc A on each loop
RAISED RMB \$02 *1000E *raised position setting
RAISED2 RMB \$02 *1000E *raised position setting
DISC_LEVEL_ZERO RMB \$02 *10010 *zero position on a flat surface-used in calibration
CALIB_DEPTH RMB \$02 *10014 *6" depth position-obtained in calibration
OPERATING_DEPTH RMB \$02 *10016 *operating depth-seed depth set by operator
SEED_DEPTH_ERROR RMB \$02 *10018 *current error from operating depth set point
BINARY_FRACTION RMB \$02 *1001A *previous error value
ERROR_CHANGE RMB \$02 *1001C *difference between present and previous error
CM_CONV_FACTOR RMB \$02 *10026 *" " " cm-remainder derived from
*A/D-zero point / calib distance (ie, 6 inches)
CONV_TEMP RMB \$02 *1002C *temp for first step in conv hex fractional to decimal
SIGN_FLAG RMB \$02 *1002E *location for the negative flag (set when above grnd)

UNITS RMB \$02 *10030 *when set, uses inches (default). Zero, uses centimetres
HYDRAULIC_TIMER1 RMB \$02 *10032 *number to count down during HYDRAULIC delay
HYDRAULIC_TIMER2 RMB \$02 *10034 *2nd timer value-no. times to count down
COUNTDOWN1 RMB \$02 *10036 *mem location for counting utilities-1/4 second
COUNTDOWN2 RMB \$02 *10038 *mem location for 2nd counting location-1/2 second
KEYPRESSED RMB \$02 *1003A *flag to show that a key has been pressed 1=pressed
IN_CONV_FACTOR2 RMB \$02 *1003C *inch conversion factor for raised height setting

OUTPUT RMB \$02 *1003E *output value to the D/A conversion routine

TEMP_CONV RMB \$02 *10044 " " " TEMP in lcdmod.asm
TEMP_DELAY RMB \$02 *10046 *temp variable for delays
TEMP_DELAY2 RMB \$02 *10048 *temp variable for delay-acc E contents
HALFSEC_INIT RMB \$02 *1004C * " " half second
KEYASCII RMB \$02 *1004E *location for ascii identity for LCD printing

IRQ_DONE RMB \$02 *10050 *flag indicating keypad IRQ done (from HEXCONV)
KEYCHECK RMB \$02 *10052 *flag indicating checking for any key pressed only (not id of key)
HEXKEY_HIT RMB \$02 *10054 *flag indicating a key from A-F was hit

TEMP_CONV RMB \$02 *10056 *temporary variable location for decimal conversion

DEPTH.INCHES RMB \$02 *10056 *location for the decimal inches for lcd
DEPTH_10TH RMB \$02 *1005C *location for 1/10th inch decimal for lcd
DEPTH_100TH RMB \$02 *1005E *location for 1/10th inch decimal for lcd
```
Appendix C: Controller Source Code

DEPTH_1000TH  RMB $02  *10060 *location for 1/100th inch decimal for lcd roundoff
OP_DISPLAY_ON RMB $02  *10062 *flag showing that the operating display is on
ADC_CHANNEL  RMB $02  *location for telling which A/D to read from
DISC_READING RMB $02  *location for depth reading from discs
DEPTH_10INCHES RMB $02  *location for 10 inches numeral
HYD_SIGNAL  RMB $02  *location for output
DISC_NINETY_DEG_POS  RMB $02  *location for output

CORRECTED_FIELD_ZERO  RMB $02  *zero position set by operator
CORRECTED_DISC_READING RMB $02  *location for depth reading from discs
CORRECTEDDEPTH_READING RMB $02  *location for depth reading from discs

FRONT_READING  RMB $02  *location for A/D 1 for second sensor
CORRECTED_FRONT_ZERO  RMB $02  *location for FRONT zero reading
CORRECTED_FRONT_READING RMB $02  *location for front sensor corrected reading
FRONT_LEVEL_ZERO RMB $02  *GROUND POSITION FOR CALIBRATION OF FRONT SENSOR
FRONT_90_READING RMB $02  *90 DEG POSITION OF FRONT SENSOR
FRONT_GROUND_ZERO RMB $02  *GROUND READING FOR THE FIELD
HYPOTENEUSE_FRONT RMB $02  *hypotenuse for front, is hypot * a factor

TRUE_0_DEPTH  RMB $02  *temp location for difference between zero and 90_deg
HYPOTENEUSE_DISC  RMB $02  *constant value for angle correction factor
ERROR_TABLE  RMB $14  *change in error circular buffer
DATA_TO_SEND  RMB $0A
DATA_INDEX  RMB $02
RAISE_DEPTH_FLAG  RMB $02  *when set, means the discs are too deep-raise discs
ERROR_CONV_FORMAT RMB $02  *error from surface in 1000’s inch divided by 200
OPERATING_DEPTH_CONV_FORMAT  RMB $02  *operating depth (in 1000’s in) /factor
CIRC_ERROR_BUFFER  RMB $02  *circular error buffer index

Table of EQUATES for register locations:

*DESCRIPTION :THIS IS A TABLE OF EQUATES FOR ALL OF THE
*REGISTERS IN THE MC68HC16Z1.
******************************************************************************

***** SIM MODULE REGISTERS *****

SIMMCR  EQU $FA00  ;SIM MODULE CONFIGURATION REGISTER
SIMTR  EQU $FA02  ;SYSTEM INTEGRATION TEST REGISTER
SYNCR  EQU $FA04  ;CLOCK SYNTHESIZER CONTROL REGISTER
RSR  EQU $FA07  ;RESET STATUS REGISTER
SIMTRE  EQU $FA08  ;SYSTEM INTEGRATION TEST REGISTER (E CLOCK)
PORTE0  EQU $FA11  ;PORTE DATA REGISTER (SAME DATA AS PORTE1)
PORTE1  EQU $FA13  ;PORTE DATA REGISTER (SAME DATA AS PORTE0)
DDRE  EQU $FA15  ;PORTE DATA DIRECTION REGISTER
PEPAR  EQU $FA17  ;PORTE PIN ASSIGNMENT REGISTER
PORTFO  EQU $FA19  ;PORT F DATA REGISTER (SAME DATA AS PORTF1)
PORTF1  EQU $FA1B  ;PORT F DATA REGISTER (SAME DATA AS PORTF0)
DDBF  EQU $FA1D  ;PORT F DATA DIRECTION REGISTER
PPFAR  EQU $FA1F  ;PORT F PIN ASSIGNMENT REGISTER
SYPCR  EQU $FA21  ;SYSTEM PROTECTION CONTROL REGISTER
PICR  EQU $FA22  ;PERIODIC INTERRUPT CONTROL REGISTER
PITR  EQU $FA24  ;PERIODIC INTERRUPT TIMING REGISTER
SSWR  EQU $FA27  ;SOFTWARE SERVICE REGISTER
TSTSMRA  EQU $FA30  ;MASTER SHIFT REGISTER A
TSTSMRB  EQU $FA32  ;MASTER SHIFT REGISTER B
TSTSC  EQU $FA34  ;TEST MODULE SHIFT COUNT
TSTRC  EQU $FA36  ;TEST MODULE REPETITION COUNT
CREG  EQU $FA38  ;TEST SUBMODULE CONTROL REGISTER
DREG  EQU $FA3A  ;DISTRIBUTED REGISTER

- C-46 -
Appendix C: Controller Source Code

CSPDR EQU SFA41 : PORT C DATA REGISTER
CSPAR0 EQU SFA44 : CHIP-SELECT PIN ASSIGNMENT REGISTER 0
CSPAR1 EQU SFA46 : CHIP-SELECT PIN ASSIGNMENT REGISTER 1
CSPARBT EQU SFA48 : CHIP-SELECT BOOT BASE ADDRESS REGISTER
CSORB T EQU SFA4A : CHIP-SELECT BOOT OPTION REGISTER
CSBAR0 EQU SFA4C : CHIP-SELECT 0 BASE ADDRESS REGISTER
CSOR0 EQU SFA4E : CHIP SELECT 0 OPTION REGISTER
CSBAR1 EQU SFA50 : CHIP-SELECT 1 BASE ADDRESS REGISTER
CSOR1 EQU SFA52 : CHIP-SELECT 1 OPTION REGISTER
CSBAR2 EQU SFA54 : CHIP-SELECT 2 BASE ADDRESS REGISTER
CSOR2 EQU SFA56 : CHIP-SELECT 2 OPTION REGISTER
CSBAR3 EQU SFA58 : CHIP-SELECT 3 BASE ADDRESS REGISTER
CSOR3 EQU SFA5A : CHIP-SELECT 3 OPTION REGISTER
CSBAR4 EQU SFA5C : CHIP-SELECT 4 BASE ADDRESS REGISTER
CSOR4 EQU SFA5E : CHIP-SELECT 4 OPTION REGISTER
CSBAR5 EQU SFA60 : CHIP-SELECT 5 BASE ADDRESS REGISTER
CSOR5 EQU SFA62 : CHIP-SELECT 5 OPTION REGISTER
CSBAR6 EQU SFA64 : CHIP-SELECT 6 BASE ADDRESS REGISTER
CSOR6 EQU SFA66 : CHIP-SELECT 6 OPTION REGISTER
CSBAR7 EQU SFA68 : CHIP-SELECT 7 BASE ADDRESS REGISTER
CSOR7 EQU SFA6A : CHIP-SELECT 7 OPTION REGISTER
CSBAR8 EQU SFA6C : CHIP-SELECT 8 BASE ADDRESS REGISTER
CSOR8 EQU SFA6E : CHIP-SELECT 8 OPTION REGISTER
CSBAR9 EQU SFA70 : CHIP-SELECT 9 BASE ADDRESS REGISTER
CSOR9 EQU SFA72 : CHIP-SELECT 9 OPTION REGISTER
CSBAR10 EQU SFA74 : CHIP-SELECT 10 BASE ADDRESS REGISTER
CSOR10 EQU SFA76 : CHIP-SELECT 10 OPTION REGISTER

***** SPM MODULE REGISTERS *****
RAMCR EQU SFB00 : RAM MODULE CONFIGURATION REGISTER
RAMTST EQU SFB02 : RAM TEST REGISTER
RAMBAH EQU SFB04 : RAM BASE ADDRESS HIGH REGISTER
RAMBAL EQU SFB06 : RAM BASE ADDRESS LOW REGISTER

***** QSM ADDRESS MAP *****
QMCR EQU SFC00 : QSM MODULE CONFIGURATION REGISTER
QTST EQU SFC02 : QSM TEST REGISTER
QILR EQU SFC04 : QSM INTERRUPT LEVELS REGISTER
QIVR EQU SFC05 : QSM INTERRUPT VECTOR REGISTER
SCCR0 EQU SFC08 : SCI CONTROL REGISTER 0
SCCR1 EQU SFC0A : SCI CONTROL REGISTER 1
SCSR EQU SFC0C : SCI STATUS REGISTER
SCDR EQU SFC0E : SCI DATA REGISTER (FULL WORD, NOT LAST 8 BITS)
QPDR EQU SFC15 : QSM PORT DATA REGISTER
QPAR EQU SFC16 : QSM PIN ASSIGNMENT REGISTER
QDDR EQU SFC17 : QSM DATA DIRECTION REGISTER
SPCR0 EQU SFC18 : SPI CONTROL REGISTER 0
SPCR1 EQU SFC1A : SPI CONTROL REGISTER 1
SPCR2 EQU SFC1C : SPI CONTROL REGISTER 2
SPCR3 EQU SFC1E : SPI CONTROL REGISTER 3
SPSR EQU SFC1F : SPI STATUS REGISTER
RR0 EQU SFD00 : SPI REC.RAM 0
RR1 EQU SFD02 : SPI REC.RAM 1
RR2 EQU SFD04 : SPI REC.RAM 2
RR3 EQU SFD06 : SPI REC.RAM 3
RR4 EQU SFD08 : SPI REC.RAM 4
RR5 EQU SFD0A : SPI REC.RAM 5
RR6 EQU SFD0C : SPI REC.RAM 6
RR7 EQU SFD0E : SPI REC.RAM 7
RR8 EQU SFD10 : SPI REC.RAM 8
RR9 EQU SFD12 : SPI REC.RAM 9
RR10 EQU SFD14 : SPI REC.RAM A
RR11 EQU SFD16 : SPI REC.RAM B
Appendix C: Controller Source Code

```
RRC EQU $FD08 ; SPI REC.RAM C
RRD EQU $FD0A ; SPI REC.RAM D
RR5 EQU $FD0C ; SPI REC.RAM E
RRF EQU $FD0E ; SPI REC.RAM F
TR0 EQU $FD20 ; SPI TXD.RAM 0
TR1 EQU $FD22 ; SPI TXD.RAM 1
TR2 EQU $FD24 ; SPI TXD.RAM 2
TR3 EQU $FD26 ; SPI TXD.RAM 3
TR4 EQU $FD28 ; SPI TXD.RAM 4
TR5 EQU $FD2A ; SPI TXD.RAM 5
TR6 EQU $FD2C ; SPI TXD.RAM 6
TR7 EQU $FD2E ; SPI TXD.RAM 7
TR8 EQU $FD30 ; SPI TXD.RAM 8
TR9 EQU $FD32 ; SPI TXD.RAM 9
TRA EQU $FD34 ; SPI TXD.RAM A
TRB EQU $FD36 ; SPI TXD.RAM B
TRC EQU $FD38 ; SPI TXD.RAM C
TRD EQU $FD3A ; SPI TXD.RAM D
TRE EQU $FD3C ; SPI TXD.RAM E
TRF EQU $FD3E ; SPI TXD.RAM F
CR0 EQU $FD40 ; SPI CMD.RAM 0
CR1 EQU $FD41 ; SPI CMD.RAM 1
CR2 EQU $FD42 ; SPI CMD.RAM 2
CR3 EQU $FD43 ; SPI CMD.RAM 3
CR4 EQU $FD44 ; SPI CMD.RAM 4
CR5 EQU $FD45 ; SPI CMD.RAM 5
CR6 EQU $FD46 ; SPI CMD.RAM 6
CR7 EQU $FD47 ; SPI CMD.RAM 7
CR8 EQU $FD48 ; SPI CMD.RAM 8
CR9 EQU $FD49 ; SPI CMD.RAM 9
CRA EQU $FD4A ; SPI CMD.RAM A
CRB EQU $FD4B ; SPI CMD.RAM B
CRC EQU $FD4C ; SPI CMD.RAM C
CRD EQU $FD4D ; SPI CMD.RAM D
CRE EQU $FD4E ; SPI CMD.RAM E
CRF EQU $FD4F ; SPI CMD.RAM F
***** GPT MODULE REGISTERS *****
GPTMCR EQU $F900 ; GPT MODULE CONFIGURATION REGISTER
GPTMCR EQU $F902 ; GPT MODULE TEST REGISTER (RESERVED)
ICR EQU $F904 ; GPT INTERRUPT CONFIGURATION REGISTER
PDDR EQU $F906 ; PARALLEL DATA DIRECTION REGISTER
GPTPDR EQU $F907 ; PARALLEL DATA REGISTER
OC1M EQU $F908 ; OC1 ACTION MASK REGISTER
OC1D EQU $F909 ; OC1 ACTION DATA REGISTER
TCNT EQU $F90A ; TIMER COUNTER REGISTER
PACTL EQU $F90C ; PULSE ACCUMULATOR CONTROL REGISTER
PACNT EQU $F90D ; PULSE ACCUMULATOR COUNTER
TIC1 EQU $F90E ; INPUT CAPTURE REGISTER 1
TIC2 EQU $F910 ; INPUT CAPTURE REGISTER 2
TIC3 EQU $F912 ; INPUT CAPTURE REGISTER 3
TOC1 EQU $F914 ; OUTPUT COMPARE REGISTER 1
TOC2 EQU $F916 ; OUTPUT COMPARE REGISTER 2
TOC3 EQU $F918 ; OUTPUT COMPARE REGISTER 3
TOC4 EQU $F91A ; OUTPUT COMPARE REGISTER 4
TT45 EQU $F91C ; INPUT CAPTURE 4 OR OUTPUT COMPARE 5
TCTL1 EQU $F91E ; TIMER CONTROL REGISTER 1
TCTL2 EQU $F91F ; TIMER CONTROL REGISTER 2
TMSK1 EQU $F920 ; TIMER INTERRUPT MASK REGISTER 1
TMSK2 EQU $F921 ; TIMER INTERRUPT MASK REGISTER 2
TFLG1 EQU $F922 ; TIMER INTERRUPT FLAG REGISTER 1
TFLG2 EQU $F923 ; TIMER INTERRUPT FLAG REGISTER 2
```
Appendix C: Controller Source Code

CFORC EQU $F924 ; COMPARE FORCE REGISTER
PWM EQU $F924 ; PWM CONTROL REGISTER
PWMA EQU $F926 ; PWM REGISTER A
PWM B EQU $F927 ; PWM REGISTER B
PWM CNT EQU $F928 ; PWM COUNTER REGISTER
PWMUFA EQU $F92A ; PWM BUFFER REGISTER A
PWMUFB EQU $F92B ; PWM BUFFER REGISTER B
PRESCL EQU $F92C ; GPT PRESCALER

***** ADC MODULE REGISTERS *****
ADCMCR EQU $F700 ; ADC MODULE CONFIGURATION REGISTER
ADTEST EQU $F702 ; ADC TEST REGISTER
ADCPDR EQU $F706 ; ADC PORT DATA REGISTER
ADCTL0 EQU $F70A ; A/D CONTROL REGISTER 0
ADCTL1 EQU $F70C ; A/D CONTROL REGISTER 1
ADSTAT EQU $F70E ; ADC STATUS REGISTER

RJURR0 EQU $F710 ; RIGHT JUSTIFIED UNSIGNED RESULT REGISTER 0
RJURR1 EQU $F712 ; RIGHT JUSTIFIED UNSIGNED RESULT REGISTER 1
RJURR2 EQU $F714 ; RIGHT JUSTIFIED UNSIGNED RESULT REGISTER 2
RJURR3 EQU $F716 ; RIGHT JUSTIFIED UNSIGNED RESULT REGISTER 3
RJURR4 EQU $F718 ; RIGHT JUSTIFIED UNSIGNED RESULT REGISTER 4
RJURR5 EQU $F71A ; RIGHT JUSTIFIED UNSIGNED RESULT REGISTER 5
RJURR6 EQU $F71C ; RIGHT JUSTIFIED UNSIGNED RESULT REGISTER 6
RJURR7 EQU $F71E ; RIGHT JUSTIFIED UNSIGNED RESULT REGISTER 7

LJSRR0 EQU $F720 ; LEFT JUSTIFIED SIGNED RESULT REGISTER 0
LJSRR1 EQU $F722 ; LEFT JUSTIFIED SIGNED RESULT REGISTER 1
LJSRR2 EQU $F724 ; LEFT JUSTIFIED SIGNED RESULT REGISTER 2
LJSRR3 EQU $F726 ; LEFT JUSTIFIED SIGNED RESULT REGISTER 3
LJSRR4 EQU $F728 ; LEFT JUSTIFIED SIGNED RESULT REGISTER 4
LJSRR5 EQU $F72A ; LEFT JUSTIFIED SIGNED RESULT REGISTER 5
LJSRR6 EQU $F72C ; LEFT JUSTIFIED SIGNED RESULT REGISTER 6
LJSRR7 EQU $F72E ; LEFT JUSTIFIED SIGNED RESULT REGISTER 7

Initialization of the Reset Vectors: ORG00000.ASM

* Title: ORG00000
* Description: This file is included to set up the reset vector ($00000 - $00006).
* *
* *******************************************************************************
* *
ORG $0000 ; put the following reset vector information
; at address $00000 of the memory map
DC.W $0010 ; $k=0, $k=1, $k=0
DC.W $0200 ; pc=200 -- initial program counter
DC.W $03FE ; $sp=03fe -- initial stack pointer
DC.W $0000 ; $iz=0 -- direct page pointer

Initialization of Reset Vectors: ORG00008.ASM
Title: ORG00008
Description: This file initializes the interrupt/exception vectors ($0008 - $01fe).
If an interrupt occurs requiring the use of any of these vectors, program flow will continue at the label "bdm" which must be added by the programmer to his/her code to put the program into background debug mode or some other appropriate routine.

******************************************************************************

ORG $0008
;put the following code in memory
;starting at address $0008 of the map
;(after the reset vector).
;there is a total of 252 of these "DC.W BDM" lines

;Vector Number (in base 10)
;and Vector Description
DC.W BDM :4 Breakpoint (BKPT)
DC.W BDM :5 Bus Error (BERR)
DC.W BDM :6 Software Interrupt (SWI)
DC.W BDM :7 Illegal Instruction
DC.W BDM :8 Divide by Zero
DC.W BDM :9 (Unassigned Reserved)
DC.W BDM :10 (Unassigned Reserved)
DC.W BDM :11 (Unassigned Reserved)
DC.W BDM :12 (Unassigned Reserved)
DC.W BDM :13 (Unassigned Reserved)
DC.W BDM :14 (Unassigned Reserved)
DC.W BDM :15 Uninitialized Interrupt
DC.W BDM :16 (Unassigned Reserved)
DC.W BDM :17 Level 1 Interrupt Autovector
DC.W BDM :18 Level 2 Interrupt Autovector
DC.W BDM :19 Level 3 Interrupt Autovector
DC.W BDM :20 Level 4 Interrupt Autovector
DC.W BDM :21 Level 5 Interrupt Autovector
DC.W BDM :22 Level 6 Interrupt Autovector
DC.W BDM :23 Level 7 Interrupt Autovector
DC.W BDM :24 Spurious Interrupt
DC.W BDM :25 (Unassigned Reserved)
DC.W BDM :26 (Unassigned Reserved)
DC.W BDM :27 (Unassigned Reserved)
DC.W BDM :28 (Unassigned Reserved)
DC.W BDM :29 (Unassigned Reserved)
DC.W BDM :30 (Unassigned Reserved)
DC.W BDM :31 (Unassigned Reserved)
DC.W BDM :32 (Unassigned Reserved)
DC.W BDM :33 (Unassigned Reserved)
DC.W BDM :34 (Unassigned Reserved)
DC.W BDM :35 (Unassigned Reserved)
DC.W BDM :36 (Unassigned Reserved)
DC.W BDM :37 (Unassigned Reserved)
DC.W BDM :38 (Unassigned Reserved)
DC.W BDM :39 (Unassigned Reserved)
DC.W BDM :40 (Unassigned Reserved)
DC.W BDM :41 (Unassigned Reserved)
DC.W BDM :42 (Unassigned Reserved)
DC.W BDM :43 (Unassigned Reserved)
Appendix C: Controller Source Code

DC.W BDM ;44 (Unassigned Reserved)  
DC.W BDM ;45 (Unassigned Reserved)  
DC.W BDM ;46 (Unassigned Reserved)  
DC.W BDM ;47 (Unassigned Reserved)  
DC.W BDM ;48 (Unassigned Reserved)  
DC.W BDM ;49 (Unassigned Reserved)  
DC.W BDM ;50 (Unassigned Reserved)  
DC.W BDM ;51 (Unassigned Reserved)  
DC.W BDM ;52 (Unassigned Reserved)  
DC.W BDM ;53 (Unassigned Reserved)  
DC.W BDM ;54 (Unassigned Reserved)  
DC.W BDM ;55 (Unassigned Reserved)  
DC.W BDM ;56 User Defined Interrupt Vector 1  
DC.W BDM ;57 User Defined Interrupt Vector 2  
DC.W BDM ;58 User Defined Interrupt Vector 3  
DC.W BDM ;59 User Defined Interrupt Vector 4  
DC.W BDM ;60 User Defined Interrupt Vector 5  
DC.W BDM ;61 User Defined Interrupt Vector 6  
DC.W BDM ;62 User Defined Interrupt Vector 7  
DC.W BDM ;63 User Defined Interrupt Vector 8  
DC.W BDM ;64 User Defined Interrupt Vector 9  
DC.W BDM ;65 User Defined Interrupt Vector 10  
DC.W BDM ;66 User Defined Interrupt Vector 11  
DC.W BDM ;67 User Defined Interrupt Vector 12  
DC.W BDM ;68 User Defined Interrupt Vector 13  
DC.W BDM ;69 User Defined Interrupt Vector 14  
DC.W BDM ;70 User Defined Interrupt Vector 15  
DC.W BDM ;71 User Defined Interrupt Vector 16  
DC.W BDM ;72 User Defined Interrupt Vector 17  
DC.W BDM ;73 User Defined Interrupt Vector 18  
DC.W BDM ;74 User Defined Interrupt Vector 19  
DC.W BDM ;75 User Defined Interrupt Vector 20  
DC.W BDM ;76 User Defined Interrupt Vector 21  
DC.W BDM ;77 User Defined Interrupt Vector 22  
DC.W BDM ;78 User Defined Interrupt Vector 23  
DC.W BDM ;79 User Defined Interrupt Vector 24  
DC.W BDM ;80 User Defined Interrupt Vector 25  
DC.W BDM ;81 User Defined Interrupt Vector 26  
DC.W BDM ;82 User Defined Interrupt Vector 27  
DC.W BDM ;83 User Defined Interrupt Vector 28  
DC.W BDM ;84 User Defined Interrupt Vector 29  
DC.W BDM ;85 User Defined Interrupt Vector 30  
DC.W BDM ;86 User Defined Interrupt Vector 31  
DC.W BDM ;87 User Defined Interrupt Vector 32  
DC.W BDM ;88 User Defined Interrupt Vector 33  
DC.W BDM ;89 User Defined Interrupt Vector 34  
DC.W BDM ;90 User Defined Interrupt Vector 35  
DC.W BDM ;91 User Defined Interrupt Vector 36  
DC.W BDM ;92 User Defined Interrupt Vector 37  
DC.W BDM ;93 User Defined Interrupt Vector 38  
DC.W BDM ;94 User Defined Interrupt Vector 39  
DC.W BDM ;95 User Defined Interrupt Vector 40  
DC.W KEY_IRQ ;96 User Defined Interrupt Vector 41  
DC.W BDM ;97 User Defined Interrupt Vector 42  
DC.W BDM ;98 User Defined Interrupt Vector 43  
DC.W BDM ;99 User Defined Interrupt Vector 44  
DC.W BDM ;100 User Defined Interrupt Vector 45  
DC.W BDM ;101 User Defined Interrupt Vector 46  
DC.W BDM ;102 User Defined Interrupt Vector 47  
DC.W BDM ;103 User Defined Interrupt Vector 48  
DC.W BDM ;104 User Defined Interrupt Vector 49
Appendix C: Controller Source Code

DC.W BDM  :105 User Defined Interrupt Vector 50
DC.W BDM  :106 User Defined Interrupt Vector 51
DC.W BDM  :107 User Defined Interrupt Vector 52
DC.W BDM  :108 User Defined Interrupt Vector 53
DC.W BDM  :109 User Defined Interrupt Vector 54
DC.W BDM  :110 User Defined Interrupt Vector 55
DC.W BDM  :111 User Defined Interrupt Vector 56
DC.W BDM  :112 User Defined Interrupt Vector 57
DC.W BDM  :113 User Defined Interrupt Vector 58
DC.W BDM  :114 User Defined Interrupt Vector 59
DC.W BDM  :115 User Defined Interrupt Vector 60
DC.W BDM  :116 User Defined Interrupt Vector 61
DC.W BDM  :117 User Defined Interrupt Vector 62
DC.W BDM  :118 User Defined Interrupt Vector 63
DC.W BDM  :119 User Defined Interrupt Vector 64
DC.W BDM  :120 User Defined Interrupt Vector 65
DC.W BDM  :121 User Defined Interrupt Vector 66
DC.W BDM  :122 User Defined Interrupt Vector 67
DC.W BDM  :123 User Defined Interrupt Vector 68
DC.W BDM  :124 User Defined Interrupt Vector 69
DC.W BDM  :125 User Defined Interrupt Vector 70
DC.W BDM  :126 User Defined Interrupt Vector 71
DC.W BDM  :127 User Defined Interrupt Vector 72
DC.W BDM  :128 User Defined Interrupt Vector 73
DC.W BDM  :129 User Defined Interrupt Vector 74
DC.W BDM  :130 User Defined Interrupt Vector 75
DC.W BDM  :131 User Defined Interrupt Vector 76
DC.W BDM  :132 User Defined Interrupt Vector 77
DC.W BDM  :133 User Defined Interrupt Vector 78
DC.W BDM  :134 User Defined Interrupt Vector 79
DC.W BDM  :135 User Defined Interrupt Vector 80
DC.W BDM  :136 User Defined Interrupt Vector 81
DC.W BDM  :137 User Defined Interrupt Vector 82
DC.W BDM  :138 User Defined Interrupt Vector 83
DC.W BDM  :139 User Defined Interrupt Vector 84
DC.W BDM  :140 User Defined Interrupt Vector 85
DC.W BDM  :141 User Defined Interrupt Vector 86
DC.W BDM  :142 User Defined Interrupt Vector 87
DC.W BDM  :143 User Defined Interrupt Vector 88
DC.W BDM  :144 User Defined Interrupt Vector 89
DC.W BDM  :145 User Defined Interrupt Vector 90
DC.W BDM  :146 User Defined Interrupt Vector 91
DC.W BDM  :147 User Defined Interrupt Vector 92
DC.W BDM  :148 User Defined Interrupt Vector 93
DC.W BDM  :149 User Defined Interrupt Vector 94
DC.W BDM  :150 User Defined Interrupt Vector 95
DC.W BDM  :151 User Defined Interrupt Vector 96
DC.W BDM  :152 User Defined Interrupt Vector 97
DC.W BDM  :153 User Defined Interrupt Vector 98
DC.W BDM  :154 User Defined Interrupt Vector 99
DC.W BDM  :155 User Defined Interrupt Vector 100
DC.W BDM  :156 User Defined Interrupt Vector 101
DC.W BDM  :157 User Defined Interrupt Vector 102
DC.W BDM  :158 User Defined Interrupt Vector 103
DC.W BDM  :159 User Defined Interrupt Vector 104
DC.W BDM  :160 User Defined Interrupt Vector 105
DC.W BDM  :161 User Defined Interrupt Vector 106
DC.W BDM  :162 User Defined Interrupt Vector 107
DC.W BDM  :163 User Defined Interrupt Vector 108
DC.W BDM  :164 User Defined Interrupt Vector 109
DC.W BDM  :165 User Defined Interrupt Vector 110
Appendix C: Controller Source Code

DC.W BDM ;166 User Defined Interrupt Vector 111
DC.W BDM ;167 User Defined Interrupt Vector 112
DC.W BDM ;168 User Defined Interrupt Vector 113
DC.W BDM ;169 User Defined Interrupt Vector 114
DC.W BDM ;170 User Defined Interrupt Vector 115
DC.W BDM ;171 User Defined Interrupt Vector 116
DC.W BDM ;172 User Defined Interrupt Vector 117
DC.W BDM ;173 User Defined Interrupt Vector 118
DC.W BDM ;174 User Defined Interrupt Vector 119
DC.W BDM ;175 User Defined Interrupt Vector 120
DC.W BDM ;176 User Defined Interrupt Vector 121
DC.W BDM ;177 User Defined Interrupt Vector 122
DC.W BDM ;178 User Defined Interrupt Vector 123
DC.W BDM ;179 User Defined Interrupt Vector 124
DC.W BDM ;180 User Defined Interrupt Vector 125
DC.W BDM ;181 User Defined Interrupt Vector 126
DC.W BDM ;182 User Defined Interrupt Vector 127
DC.W BDM ;183 User Defined Interrupt Vector 128
DC.W BDM ;184 User Defined Interrupt Vector 129
DC.W BDM ;185 User Defined Interrupt Vector 130
DC.W BDM ;186 User Defined Interrupt Vector 131
DC.W BDM ;187 User Defined Interrupt Vector 132
DC.W BDM ;188 User Defined Interrupt Vector 133
DC.W BDM ;189 User Defined Interrupt Vector 134
DC.W BDM ;190 User Defined Interrupt Vector 135
DC.W BDM ;191 User Defined Interrupt Vector 136
DC.W BDM ;192 User Defined Interrupt Vector 137
DC.W BDM ;193 User Defined Interrupt Vector 138
DC.W BDM ;194 User Defined Interrupt Vector 139
DC.W BDM ;195 User Defined Interrupt Vector 140
DC.W BDM ;196 User Defined Interrupt Vector 141
DC.W BDM ;197 User Defined Interrupt Vector 142
DC.W BDM ;198 User Defined Interrupt Vector 143
DC.W BDM ;199 User Defined Interrupt Vector 144
DC.W BDM ;200 User Defined Interrupt Vector 145
DC.W BDM ;201 User Defined Interrupt Vector 146
DC.W BDM ;202 User Defined Interrupt Vector 147
DC.W BDM ;203 User Defined Interrupt Vector 148
DC.W BDM ;204 User Defined Interrupt Vector 149
DC.W BDM ;205 User Defined Interrupt Vector 150
DC.W BDM ;206 User Defined Interrupt Vector 151
DC.W BDM ;207 User Defined Interrupt Vector 152
DC.W BDM ;208 User Defined Interrupt Vector 153
DC.W BDM ;209 User Defined Interrupt Vector 154
DC.W BDM ;210 User Defined Interrupt Vector 155
DC.W BDM ;211 User Defined Interrupt Vector 156
DC.W BDM ;212 User Defined Interrupt Vector 157
DC.W BDM ;213 User Defined Interrupt Vector 158
DC.W BDM ;214 User Defined Interrupt Vector 159
DC.W BDM ;215 User Defined Interrupt Vector 160
DC.W BDM ;216 User Defined Interrupt Vector 161
DC.W BDM ;217 User Defined Interrupt Vector 162
DC.W BDM ;218 User Defined Interrupt Vector 163
DC.W BDM ;219 User Defined Interrupt Vector 164
DC.W BDM ;220 User Defined Interrupt Vector 165
DC.W BDM ;221 User Defined Interrupt Vector 166
DC.W BDM ;222 User Defined Interrupt Vector 167
DC.W BDM ;223 User Defined Interrupt Vector 168
DC.W BDM ;224 User Defined Interrupt Vector 169
DC.W BDM ;225 User Defined Interrupt Vector 170
DC.W BDM ;226 User Defined Interrupt Vector 171
Appendix C: Controller Source Code

DC.W BDM 227 User Defined Interrupt Vector 172
DC.W BDM 228 User Defined Interrupt Vector 173
DC.W BDM 229 User Defined Interrupt Vector 174
DC.W BDM 230 User Defined Interrupt Vector 175
DC.W BDM 231 User Defined Interrupt Vector 176
DC.W BDM 232 User Defined Interrupt Vector 177
DC.W BDM 233 User Defined Interrupt Vector 178
DC.W BDM 234 User Defined Interrupt Vector 179
DC.W BDM 235 User Defined Interrupt Vector 180
DC.W BDM 236 User Defined Interrupt Vector 181
DC.W BDM 237 User Defined Interrupt Vector 182
DC.W BDM 238 User Defined Interrupt Vector 183
DC.W BDM 239 User Defined Interrupt Vector 184
DC.W BDM 240 User Defined Interrupt Vector 185
DC.W BDM 241 User Defined Interrupt Vector 186
DC.W BDM 242 User Defined Interrupt Vector 187
DC.W BDM 243 User Defined Interrupt Vector 188
DC.W BDM 244 User Defined Interrupt Vector 189
DC.W BDM 245 User Defined Interrupt Vector 190
DC.W BDM 246 User Defined Interrupt Vector 191
DC.W BDM 247 User Defined Interrupt Vector 192
DC.W BDM 248 User Defined Interrupt Vector 193
DC.W BDM 249 User Defined Interrupt Vector 194
DC.W BDM 250 User Defined Interrupt Vector 195
DC.W BDM 251 User Defined Interrupt Vector 196
DC.W BDM 252 User Defined Interrupt Vector 197
DC.W BDM 253 User Defined Interrupt Vector 198
DC.W BDM 254 User Defined Interrupt Vector 199
DC.W BDM 255 User Defined Interrupt Vector 200

* This is the end of the org00008.asm file.
This appendix contains the source code used for the PC logging program. This program is designed to run on a laptop computer, and communicate with the MC68HC16EVB or the tractor cab controller that was built for this research. Data from the controller was displayed on the screen as lines proportional in length to the signal received. This proved to be very helpful while field testing, where no representation of the control actions was available using the LCD-only equipped controller. It was written in Borland Turbo C, and runs under DOS using the DOS graphic commands available under Turbo C. Although the logger program interface could have been improved and extra features easily added if it was written in Visual C for Windows, the program was written for DOS in order to maximize portability and minimize incompatibility problems. It was designed to run on a laptop PC with or without a hard disk, many of which have monochrome screens and do not run MS Windows.

The controller data logged included the error from the set point, the change in error, and the fuzzy decision output. A/D hexadecimal values from the sensors was also logged. This data, from the front and rear sensors, came directly from the registers for the A/D converters.
Appendix D: PC Logger Source Code

Source Code: LOGGER4.C

/* DATATEST.C. Display and store the sensor error, change in error, and */
/* fuzzy decision output. Test the text formatting. */
/* Added a second "while" loop so "START" doesn't keep getting sent */
#include <stdio.h>
#include <chios.h>
#include <conio.h>
#include <stdlib.h>
#include <float.h>
#include <graphics.h>
#include <math.h>
#include <conio.h>

#define START 0x0F
#define CONTINUE 0x0E
#define SAMPLES 64
#define COM1 0*/
/* 9600 baud, 8 bits, 1 stop, no parity */
#define DATA_READY 0x100
/* void graph(); */

unsigned char in, *filein, tmp;
unsigned int *fileout;
int i, outfile, waiting;

main()
{
  int register in, status;
  FILE *outfile, *filestor;
  int xtitle, ytitle;
  int g_driver, g_mode, COM1;
  int left, top, wide, bottom;

  xtitle = 130;
  ytitle = 50;

  printf("Hit 0 for COM1, 1 for COM2: ");
  scanf("%d", &COM1);
  nosound();

  if (((filein = (unsigned char *)malloc(128*sizeof(unsigned char))) == NULL)
  {
    printf("insufficient memory: malloc failed.in");
    exit(1);
  }

  //**********OPEN OUTPUT FILE, FOR SAVING DATA AT END OF GRAPHICAL ROUTINE****/
  if ((outfile = fopen("fuzdataout", "wt")) == NULL)
  {
    fprintf(stderr, "Cannot open output file.out");
  }

  bioscom(0, SETTINGS, COM1); /* 0 = initialize the com port */
  printf("Error opening %s. filestor\n", "filestor");
  perror("filestor");
  exit(0);

  //**********GRAPHICS TEXT AND OUTLINE ROUTINE*******

  if (registerbgidriver(CGA_driver) <0) exit(1);
  if (registerbgidriver(ATI_driver) <0) exit(1);
  if (registerbgidriver(EGA_VGA_driver) <0) exit(1);
  if (registerbgidriver(Herc_driver) <0) exit(1);
  if (registerbgidriver(IBM8514_driver) <0) exit(1);
  if (registerbgidriver(PC3270_driver) <0) exit(1);
  detectgraph(&g_driver, &g_mode);
  initgraph(&g_driver, &g_mode, ":\bgi");
Appendix D: PC Logger Source Code

outtextxy(xtitle + 50, ytitle, "Error, Change In Error, and Fuzzy Decision");
outtextxy(xtitle + 40, 430, " Logging Parameters");
outtextxy(xtitle - 14, 415, "Error Error Chg Fuzzy Decision Fat Sensor Dsk Sensor");
outtextxy(xtitle - 125, 480, " Magnitude");

line(114, 405, 598, 405); /* draw a white line on the bottom of the graph */
line(114, 405, 114, 150); /* draw a white line to the left of the graph */
line(598, 405, 598, 150); /* draw a white line to the right of the graph */
setbkcolor(8);

setcolor(12); /* set colour to yellow */
line(120, 272, 160, 272); /* draw in the new line 274 = 401-127 */
line(205, 272, 245, 272); /* center for no change in error */
line(434, 143, 474, 143); /* max for sensor position */
line(518, 272, 558, 272); /* half for sensor position */

SCI ROUTINE
i=0; /* initialize memory location */
while(1) /* start the main routine */
{
    /* clear graphics trigger flag */
bioscom(1, START, COM1); /* send start transmission ($F) to HC16 */
delay(1);
    while(<10)
    {
        status = bioscom(3, 0, COM1); /* get status of com port */
delay(1);
        /* keep checking for it-sending START again causes overflow in HC16*/
        if(status & DATA_READY) /* AND DATA READY ($8) with */
        {
            (in = bioscom(2, 0, COM1)); /* get data from receive port */
            filein[i] = in; /* assign value of "in" to loc i of filein */
delay(1);
            bioscom(1, CONTINUE, COM1); /* send rcv ack ($E) to HC16 */
delay(1);
            for(i=1; i<10; i=i)
            {
                status = bioscom(3, 0, COM1);
                delay(1); /* delay 1 millisecond */
                if(status & DATA_READY) /* AND DATA READY ($8) with */
                {
                    /* status-see if its "Ready" */
delay(1);
                    (in = bioscom(2, 0, COM1)); /* get data from receive port */
                    filein[j] = in; /* assign value of "in" to loc i of filein */
bioscom(1, CONTINUE, COM1); /* send rcv ack ($E) to HC16 */
j++;}
        }
    }
    /* increment memory location index */
    j=1;
    /* reset index to 0 */
    /* and set graphics trigger flag */
    if(kbhit()) /* this is used, not i, since i=0 at start */
    {
        /* if a key is hit on the keyboard */
        if(kbhk() == 'x1B') /* set in to key hit, if it's ESC, it's */
        {
            /* time to stop program execution and exit */
closegraph(); /* disable the graphics routine */
fclose(outtextxy); /* close the file */
return(0); /* exit program execution */
        }

        bioscom(1, in, COM1); /* if it wasn't ESC, send the char */
        /* out on the serial line (is only */
        if(j == 1)
        {
            for(i=0; i<10; i+=2) /* exchange the data in adjacent */

-D-3
Appendix D: PC Logger Source Code

```c
{  /* locations */
    tmp = filein[i];
    filein[i] = filein[i+1];
    filein[i+1] = tmp;
}
/* combine 8-bit data to form 16 bit */
/* put bytes in filein into words */
fileout = (unsigned int *)filein; /* in fileout */

/*******GRAPHICS ROUTINE**********/
/* clearing the screen was eliminated in favour of redrawing */
/* each line--this eliminates flicker, and leaves text in place */
bottom = getmaxy() - 80;
/* initial position of line */
left = 120;
setlinestyle(0,0,1);
for (i = 0; i < 5; i++)
{
    if(i<3)
    {
        left = 140 + (85*i); /* shifted position of next lines */
    }
    else
    {
        left = 200 + (85*i);
    }
    setcolor(0); /* set colour to black */
    if(i==1)
    {
        line(left, 0, left, bottom); /* draw black line over old one */
    }
    else
    {
        line(left, 100, left, bottom);
    }
    top = (bottom) - ((int)fileout[i]); /* draw the fff bars */
    if(i==1)
    {
        top = (bottom) - ((int)fileout[i]*2));
    }
    setcolor(14); /* set colour to yellow */
    line(left, top, left, bottom); /* draw in the new line */
}
for(i=0; i<1; i++)
{
    fprintf(outfile, "%u %u %u %u %u\n",
            fileout[i],fileout[i+1],fileout[i+2],fileout[i+3],fileout[i+4],fileout[i+5]); /* print to the file */
}
i=0;
}
}
```
APPENDIX E

MATHCAD SIMULATION

The simulations done in mathcad were produced in Version 4.0 at the outset of this research. With the release of Version 6.0, this work was expanded by using the capabilities to program in if-then loops and other iterations. Otherwise, it would have been necessary to produce all code in C or C++. 
Fuzzy Logic Simulation

This example is of a 5-membership function *template*, from which modifications are made to the number and shape of membership functions in order to optimize the output.

In the Section A, the membership functions for ERROR are defined. In Section B, the membership functions for CHANGE_IN_ERROR are defined. Section C has the fuzzy rule base, with Section D the rule aggregation functions. Section E has the singleton defuzzification scheme. In the definitions of the points on the triangular membership functions:

NBM = negative big middle point
NBR = negative big right point

NSL = negative small left point
NSM = negative small middle point
NSR = negative small right point

PSL = positive small left
PSM = positive small middle
PSR = positive small right

PBL = positive big left
PBM = positive big middle

ZL = zero left point
ZM = zero middle point
ZR = zero right point

As Mathcad 6.0 is capable of doing matrix operations quite effectively, a 21 x 21 input matrix is used in this example. The result is a 21 x 21 point output control surface for this set of inputs.

Since Mathcad must have each element explicitly defined, each of these definitions are given in vector and definition form.
y vector with dimension 1 row x 21 columns is the range of the crisp inputs:
\[ y = (100 \ 90 \ 80 \ 70 \ 60 \ 50 \ 40 \ 30 \ 20 \ 10 \ 0 \ 10 \ 20 \ 30 \ 40 \ 50 \ 60 \ 70 \ 80 \ 90 \ 100) \]

z vector with dimension 21 rows x 1 column is the number of times vector y is repeated:

\[ z = \frac{y}{5} \]

\[
\begin{bmatrix}
\end{bmatrix}
\]

x is the 21 row x 21 column matrix, made up of 21 rows of vector y divided by 5 for this example:

\[ x := \frac{y}{5} \]

The input membership function ERROR is made up of vector x:

Input: \( ERROR := x \)

The 21 x 21 matrix x is the basis for the example functions in this example.
ERROR is defined as x, while
CHANGE_IN_ERROR is defined as x transpose.
Section A: ERROR Membership Function Parameter Definitions:

```
NBM: 10  NSL: 10  ZL: 5  PSL: 0  PBL: 5
NBR:  5  NSM:  5  ZM: 0  PSM: 5  PBM: 10
NSR:  0  ZR:  5  PSR: 10
```

Indices for ERROR input vectors:

```
n  0..20
m  0..20
```

Definitions for the slopes for the symmetrical triangular membership functions:

```
NEG_BIG_SLOPE := \frac{1}{NBM - NBR}
ZERO_MEMB_SLOPE := \frac{1}{ZR - ZM}
POS_BIG_SLOPE := \frac{1}{PBM - PBL}
POS_SMALL_SLOPE := \frac{1}{PSR - PSM}
NEG_SMALL_SLOPE := \frac{1}{NSL - NSM}
```

Calculation Loops for fuzzifying crisp inputs in each membership function:

```
E_NEG_BIG_{m,n} := \begin{cases}
1 & \text{if } ERROR_{m,n} \leq \text{NBR} \\
\frac{1}{NBM - NBR} & \text{if } ERROR_{m,n} > \text{NBR} \\
[(ERROR - NBR)_{m,n} \cdot \text{NEG_BIG_SLOPE}] & \text{otherwise} \\
0 & \text{otherwise}
\end{cases}
```

```
E_NEG_SMALL_{m,n} := \begin{cases}
1 & \text{if } ERROR_{m,n} \geq \text{NSL} \\
\frac{1}{NSL - NSM} & \text{if } ERROR_{m,n} < \text{NSL} \\
[(ERROR - NSL)_{m,n} \cdot \text{NEG_SMALL_SLOPE}] & \text{if } ERROR_{m,n} \leq \text{NSM} \\
[(NSL - ERROR)_{m,n} \cdot \text{NEG_SMALL_SLOPE}] & \text{otherwise} \\
0 & \text{otherwise}
\end{cases}
```

```
E_ZERO_{m,n} := \begin{cases}
1 & \text{if } ERROR_{m,n} \geq ZL \\
\frac{1}{ZR - ZM} & \text{if } ERROR_{m,n} \leq ZR \\
[(ZL - ERROR)_{m,n} \cdot \text{ZERO_MEMB_SLOPE}] & \text{if } ERROR_{m,n} \leq ZM \\
[(ZM - ERROR)_{m,n} \cdot \text{ZERO_MEMB_SLOPE}] & \text{otherwise} \\
0 & \text{otherwise}
\end{cases}
```

E - 4
\[ E_{\text{POS\_SMALL}}_{m,n} := \begin{cases} \text{if } \text{ERROR}_{m,n} > \text{PSL} \\
\text{if } \text{ERROR}_{m,n} < \text{PSR} \\
|(|\text{ERROR}_{m,n}|\cdot \text{POS\_SMALL\_SLOPE}| \text{ if } \text{ERROR}_{m,n} < \text{PSM} \\
|(|\text{PSR} - \text{ERROR}_{m,n}|\cdot \text{POS\_SMALL\_SLOPE}| \text{ otherwise} \\
0 \text{ otherwise} \end{cases} \]

\[ E_{\text{POS\_BIG}}_{m,n} := \begin{cases} \text{if } \text{ERROR}_{m,n} \geq \text{PBL} \\
1 \text{ if } \text{ERROR}_{m,n} \geq \text{PBM} \\
|(|\text{ERROR} - \text{PBL}_{m,n}|\cdot \text{POS\_SMALL\_SLOPE}| \text{ otherwise} \\
0 \text{ otherwise} \end{cases} \]

Example of the shapes of three membership functions POSITIVE SMALL ERROR, ZERO ERROR and POSITIVE BIG ERROR for the given input array:
Section B: CHANGE_IN_ERROR Membership Function Parameter Definitions, similar to ERROR in shape:

\[ \begin{align*}
\text{CNBM} & : 10 \\
\text{CNSL} & : 10 \\
\text{CZL} & : 5 \\
\text{CPSL} & : 0 \\
\text{CPBL} & : 5 \\
\text{CNBR} & := -5 \\
\text{CNSM} & := -5 \\
\text{CZM} & := 0 \\
\text{CPSM} & := 5 \\
\text{CPBM} & := 10 \\
\text{CNSR} & := 0 \\
\text{CZR} & := 5 \\
\text{CPSR} & := 10
\end{align*} \]

The inputs for CHANGE_IN_ERROR are matrix \( x \), but are transposed: \( \text{CHG} \_\text{ERROR} := x^T \)

Definitions for the symmetrical triangular membership functions for CHANGE_IN_ERROR:

\[ \begin{align*}
\text{C\_NEG\_BIG\_SLOPE} & := \frac{1}{\text{CNBM} - \text{CNBR}} \\
\text{C\_ZERO\_MEMB\_SLOPE} & := \frac{1}{\text{CZR} - \text{CZM}} \\
\text{C\_POS\_BIG\_SLOPE} & := \frac{1}{\text{CPBM} - \text{CPBL}} \\
\text{C\_NEG\_SMALL\_SLOPE} & := \frac{1}{\text{CNSL} - \text{CNSM}} \\
\text{C\_POS\_SMALL\_SLOPE} & := \frac{1}{\text{CPSM} - \text{CPBL}}
\end{align*} \]

Calculation Loops for fuzzifying crisp inputs in each membership function:

\[ \begin{align*}
\text{CE\_NEG\_BIG}_{m,n} & := \begin{cases} 
1 & \text{if CHG\_ERROR}_{m,n} \leq \text{CNBR} \\
1 & \text{if CHG\_ERROR}_{m,n} \leq \text{CNBM} \\
1 & \text{if CHG\_ERROR}_{m,n} \leq \text{CNBM} \\
\left( \text{CHG\_ERROR}_{m,n} - \text{CNBR} \right) \cdot \text{C\_NEG\_BIG\_SLOPE} & \text{otherwise} \\
0 & \text{otherwise}
\end{cases} \\
\text{CE\_NEG\_SMALL}_{m,n} & := \begin{cases} 
1 & \text{if CHG\_ERROR}_{m,n} \geq \text{CNSL} \\
\left( \text{CNSL} - \text{CHG\_ERROR}_{m,n} \right) \cdot \text{C\_NEG\_SMALL\_SLOPE} & \text{if CHG\_ERROR}_{m,n} \leq \text{CNSM} \\
\left( \text{CHG\_ERROR}_{m,n} - \text{NSR} \right) \cdot \text{C\_NEG\_SMALL\_SLOPE} & \text{otherwise} \\
0 & \text{otherwise}
\end{cases} \\
\text{CE\_ZERO}_{m,n} & := \begin{cases} 
1 & \text{if CHG\_ERROR}_{m,n} \geq \text{CZL} \\
\left( \text{CZL} - \text{CHG\_ERROR}_{m,n} \right) \cdot \text{C\_ZERO\_MEMB\_SLOPE} & \text{if CHG\_ERROR}_{m,n} \leq \text{CZM} \\
\left( \text{CZL} - \text{CHG\_ERROR}_{m,n} \right) \cdot \text{C\_ZERO\_MEMB\_SLOPE} & \text{if CHG\_ERROR}_{m,n} \leq \text{CZM} \\
0 & \text{otherwise}
\end{cases}
\end{align*} \]
CE_POS_SMALL\_m,n := \[
\text{if } \text{CHG\_ERROR\_m,n} > \text{CPSL} \\
\quad \text{if } \text{CHG\_ERROR\_m,n} < \text{CPSR} \\
\qquad \text{| (CHG\_ERROR\_m,n - C\_POS\_SMALL\_SLOPE) | if } \text{CHG\_ERROR\_m,n} < \text{CPSM} \\
\qquad \text{| CPSR - CHG\_ERROR\_m,n | C\_POS\_SMALL\_SLOPE | otherwise} \\
\quad 0 \text{ otherwise} \\
\quad 0 \text{ otherwise}
\]

CE_POS_BIG\_m,n := \[
\text{if } \text{CHG\_ERROR\_m,n} \geq \text{CPBL} \\
\quad 1 \text{ if } \text{CHG\_ERROR\_m,n} \geq \text{CPBM} \\
\qquad \text{| (CHG\_ERROR\_m,n - CPBL) \cdot C\_POS\_SMALL\_SLOPE | otherwise} \\
\quad 0 \text{ otherwise}
\]

Membership function shapes are similar to those given above for ERROR.

Section C: Fuzzy Control Rules:

Definition for MIN function: \( \text{AND}(X, Y) := \text{if}(X \leq Y, X, Y) \)

Definition for MAX function: \( \text{OR}(X, Y) := \text{if}(X > Y, X, Y) \)

Rule Definition:

If ERROR is NB and CHANGE\_IN\_ERROR is PB then OUTPUT is Z
If ERROR is NB and CHANGE\_IN\_ERROR is PS then OUTPUT is PS
If ERROR is NB and CHANGE\_IN\_ERROR is Z then OUTPUT is PB
If ERROR is NB and CHANGE\_IN\_ERROR is NS then OUTPUT is PB
If ERROR is NB and CHANGE\_IN\_ERROR is NB then OUTPUT is PB
If ERROR is NS and CHANGE\_IN\_ERROR is PB then OUTPUT is NS
If ERROR is NS and CHANGE\_IN\_ERROR is PS then OUTPUT is Z
If ERROR is NS and CHANGE\_IN\_ERROR is Z then OUTPUT is PS
If ERROR is NS and CHANGE\_IN\_ERROR is NS then OUTPUT is PS
If ERROR is NS and CHANGE\_IN\_ERROR is NB then OUTPUT is PB

\[
\begin{align*}
Z_1(m,n) &:= \text{AND}(E\_NEG\_BIG\_m,n,CE\_POS\_BIG\_m,n) \\
PS_1(m,n) &:= \text{AND}(E\_NEG\_BIG\_m,n,CE\_POS\_SMALL\_m,n) \\
PB_1(m,n) &:= \text{AND}(E\_NEG\_BIG\_m,n,CE\_ZERO\_m,n) \\
PB_2(m,n) &:= \text{AND}(E\_NEG\_BIG\_m,n,CE\_NEG\_SMALL\_m,n) \\
PB_3(m,n) &:= \text{AND}(E\_NEG\_BIG\_m,n,CE\_NEG\_BIG\_m,n) \\
NS_1(m,n) &:= \text{AND}(E\_NEG\_SMALL\_m,n,CE\_POS\_BIG\_m,n) \\
Z_2(m,n) &:= \text{AND}(E\_NEG\_SMALL\_m,n,CE\_POS\_SMALL\_m,n) \\
PS_2(m,n) &:= \text{AND}(E\_NEG\_SMALL\_m,n,CE\_ZERO\_m,n) \\
PS_3(m,n) &:= \text{AND}(E\_NEG\_SMALL\_m,n,CE\_NEG\_SMALL\_m,n) \\
PB_4(m,n) &:= \text{AND}(E\_NEG\_SMALL\_m,n,CE\_NEG\_BIG\_m,n)
\end{align*}
\]
If ERROR is Z and CHANGE_IN_ERROR is PB then OUTPUT is NB
If ERROR is Z and CHANGE_IN_ERROR is PS then OUTPUT is NS
If ERROR is Z and CHANGE_IN_ERROR is Z then OUTPUT is Z
If ERROR is Z and CHANGE_IN_ERROR is NS then OUTPUT is PS
If ERROR is Z and CHANGE_IN_ERROR is NB then OUTPUT is PB
If ERROR is PS and CHANGE_IN_ERROR is PB then OUTPUT is NB
If ERROR is PS and CHANGE_IN_ERROR is PS then OUTPUT is NS
If ERROR is PS and CHANGE_IN_ERROR is Z then OUTPUT is NS
If ERROR is PS and CHANGE_IN_ERROR is NS then OUTPUT is Z
If ERROR is PS and CHANGE_IN_ERROR is NB then OUTPUT is PS
If ERROR is PB and CHANGE_IN_ERROR is PB then OUTPUT is NB
If ERROR is PB and CHANGE_IN_ERROR is PS then OUTPUT is NB
If ERROR is PB and CHANGE_IN_ERROR is Z then OUTPUT is NB
If ERROR is PB and CHANGE_IN_ERROR is NS then OUTPUT is NS
If ERROR is PB and CHANGE_IN_ERROR is NB then OUTPUT is Z

Section D: Aggregation of rules:

\[ \text{NB}(m,n) := (\text{NB}_1(m,n) \text{ NB}_2(m,n) \text{ NB}_3(m,n) \text{ NB}_4(m,n) \text{ NB}_5(m,n)) \]
\[ \text{NB}_\text{OUT}_{m,n} := \max(\text{NB}(m,n)) \]

\[ \text{NS}(m,n) := (\text{NS}_1(m,n) \text{ NS}_2(m,n) \text{ NS}_3(m,n) \text{ NS}_4(m,n) \text{ NS}_5(m,n)) \]
\[ \text{NS}_\text{OUT}_{m,n} := \max(\text{NS}(m,n)) \]

\[ \text{Z}(m,n) := (\text{Z}_1(m,n) \text{ Z}_2(m,n) \text{ Z}_3(m,n) \text{ Z}_4(m,n) \text{ Z}_5(m,n)) \]
\[ \text{Z}_\text{OUT}_{m,n} := \max(\text{Z}(m,n)) \]

\[ \text{PS}(m,n) := (\text{PS}_1(m,n) \text{ PS}_2(m,n) \text{ PS}_3(m,n) \text{ PS}_4(m,n) \text{ PS}_5(m,n)) \]
\[ \text{PS}_\text{OUT}_{m,n} := \max(\text{PS}(m,n)) \]

\[ \text{PB}(m,n) := (\text{PB}_1(m,n) \text{ PB}_2(m,n) \text{ PB}_3(m,n) \text{ PB}_4(m,n) \text{ PB}_5(m,n)) \]
\[ \text{PB}_\text{OUT}_{m,n} := \max(\text{PB}(m,n)) \]

\[ E - 8 \]
Section E: OUTPUT Singleton Function Definitions:

NEG_BIG_SING: 10  NEG_SMALL_SING: 5  ZERO_SING: 1  POS_SMALL_SING: 5  POS_BIG_SING 10

\[
\text{OUTPUT}_{m,n} = \frac{(\text{NB.OUT}_{m,n} \cdot \text{NEG_BIG_SING}) + (\text{NS.OUT}_{m,n} \cdot \text{NEG_SMALL_SING}) + (\text{Z.OUT}_{m,n} \cdot \text{ZERO_SING}) + \text{PS.OUT}_{m,n} \cdot \text{POS_SMALL_SING} + (\text{PB.OUT}_{m,n} \cdot \text{POS_BIG_SING})}{\text{NB.OUT}_{m,n} + \text{NS.OUT}_{m,n} + \text{Z.OUT}_{m,n} + \text{PS.OUT}_{m,n} + \text{PB.OUT}_{m,n}}
\]