



Software development for automatic steering and implement control of agricultural equipment utilizing the global positioning system and a geographic information system
by Brian Thomas Mosdal

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in
Agricultural Engineering
Montana State University
© Copyright by Brian Thomas Mosdal (1994)

Abstract:

This thesis is a report of the software development for automatic steering and implement control of agricultural equipment. The software will use the Global Positioning System for position location and a Geographic Information System to define the input requirements for the machine control functions.

There are several modules contained in the software. These are configuration modules for the geometry of the tractor and hitch, implement, G.P.S. antenna mounting, and field / G.I.S. grid cell interaction. Another module is used to test a field target travel path for continuity and tangency of its travel path segments, before it is used in the main part of the program. The main section of the software deals primarily with determining the steering angle required for a tractor and implement to follow the predrawn, target travel path through the field. This target travel path is constructed using a CADD program or any other method that can produce a DXF output file of the travel path entities.

Provisions have been made for the future development of the software and for incorporating a G.I.S. database, as well as other possibilities. This software development should provide a good basis for future development of automatically controlled agricultural equipment.

SOFTWARE DEVELOPMENT FOR AUTOMATIC STEERING AND
IMPLEMENT CONTROL OF AGRICULTURAL EQUIPMENT UTILIZING THE
GLOBAL POSITIONING SYSTEM AND A GEOGRAPHIC INFORMATION SYSTEM

by

Brian Thomas Mosdal

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Agricultural Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 1994

© COPYRIGHT

by

Brian Thomas Mosdal

1994

All Rights Reserved

N318
m85

APPROVAL

of a thesis submitted by

Brian Thomas Mosdal

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

July 29, 1994
Date

William E. Larsen
Chairperson, Graduate Committee

Approved for the Major Department

29 July 1994
Date

Shirley E. Long
Head, Major Department

Approved for the College of Graduate Studies

8/23/94
Date

R. Brown
Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Signature Brian T. Mosdal

Date July 29, 1994

TABLE OF CONTENTS

	Page
APPROVAL	ii
STATEMENT OF PERMISSION TO USE	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	vi
ABSTRACT	vii
1. INTRODUCTION	1
Project Overview.....	1
Alternative Navigation Systems	2
Role of G.P.S. and G.I.S.....	2
Advantages of a G.P.S. Based Guidance System	3
2. GUIDANCE SYSTEM	6
Overview.....	6
Hardware Requirements	6
Software Overview.....	7
3. DESCRIPTION OF SOFTWARE PROGRAM MODULES	10
Main G.P.S. Module.....	10
G.P.S. Header Module.....	10
General Input Module	11
General Error Module	12
Tractor Configuration Module	15
Implement Configuration Module.....	23
Field Status Configuration Module	29
Field Travel Path Test Module	34
Obtaining DXF Entities	39
Travel Path Data Tests	44
Travel Path Display Module.....	50
Farm Field Module.....	51
4. PROGRAM STATUS AND FUTURE DEVELOPMENT	67
Computer Program Status.....	67
Future Program Development.....	68
Conclusion	76

TABLE OF CONTENTS (CONTINUED)

BIBLIOGRAPHY	77
APPENDICES	80
A. Main Program Module [GPS1.C]	81
B. G.P.S. Header Module [GPSHEADE.H]	87
C. General Input Module [GENINPUT.C]	105
D. General Error Module [GENEROR.C]	112
E. Tractor Configuration Module [TRACTOR.C]	177
F. Implement Configuration Module [IMPLEMEN.C]	202
G. Field Status Configuration Module [FLDSTATS.C]	224
H. Field Travel Path Test Module [TESTFIEL.C]	246
I. Farm Field Module [FARMFIEL.C]	287
J. Tractor .TRC Configuration File	391
K. Implement / GPS .IMP Configuration File	393
L. Field status .FSC Configuration File	395
M. Field Target Travel Path .TPD Output File	397

LIST OF FIGURES

Figure	Page
1. Front Wheel Steering Tractor with Swinging Drawbar	16
2. Swinging Drawbar Tractor Hitch (Detail)	17
3. Implement Geometry	24
4. G.P.S. Receiver Geometry (Detail)	25
5. G.I.S. Grid and Base point Geometry	31
6. Typical Field Target Travel Path	35
7. Tractor / Implement Steering Geometry	61

ABSTRACT

This thesis is a report of the software development for automatic steering and implement control of agricultural equipment. The software will use the Global Positioning System for position location and a Geographic Information System to define the input requirements for the machine control functions.

There are several modules contained in the software. These are configuration modules for the geometry of the tractor and hitch, implement, G.P.S. antenna mounting, and field / G.I.S. grid cell interaction. Another module is used to test a field target travel path for continuity and tangency of its travel path segments, before it is used in the main part of the program. The main section of the software deals primarily with determining the steering angle required for a tractor and implement to follow the predrawn, target travel path through the field. This target travel path is constructed using a CADD program or any other method that can produce a DXF output file of the travel path entities.

Provisions have been made for the future development of the software and for incorporating a G.I.S. database, as well as other possibilities. This software development should provide a good basis for future development of automatically controlled agricultural equipment.

CHAPTER 1

INTRODUCTION

Project Overview

Another revolution in farming is taking place — precision farming!

What is being called 'precision farming,' harnesses such recent space-age developments as global positioning satellites, variable-rate controllers on application machinery, real-time yield monitors, and crop sensors, and powerful computer software to make the science of farming vastly more scientific than it is today. (Gerstner 1994)

This revolution in farming is due primarily to the great advancements in digital computer technology made during the last ten to twenty years. There are many advantages to using high precision farming techniques versus the traditional, manual methods of farming. Some of the major advantages of precision farming will be discussed within this thesis.

Software development for automatic steering and implement control of agricultural machinery utilizing the Global Positioning System (G.P.S.) and a Geographic Information System (G.I.S.) is the subject of this thesis. Major productivity and efficiency gains can be accomplished with an autonomous farm machine — the *ultimate* goal of machine positioning, guidance and control. The purpose of

this software development project is to provide a solid framework for farm machine guidance and implement control, not an autonomous control system per se, but some important components of such a system. This software can be adapted, modified and refined to handle the required inputs and outputs of an agricultural machinery control system.

Alternative Navigation Systems

Several different equipment navigation and guidance systems have been proposed over the years for use in agricultural applications. These systems include, but are not limited to the following methods – laser detection units, LORAN-C, radar beacons, inertial, radio frequency, odometry, computer vision, and “dead reckoning”, as well as the differential, real-time (kinematic) G.P.S. system. The G.P.S. based system appears to have the best potential, with respect to precision, accuracy, standardization, reliability, and future feasibility, for use in precision farming.

Role of G.P.S. and G.I.S.

Very high precision location determination is possible when using a G.P.S. receiver in a differential correction mode. When the G.P.S. signal is corrected in real-time, it is possible to accurately and precisely determine the kinematics of a ground-based vehicle operating at typical farm field travel speeds. This positioning information can be used for navigating the tractor and machinery through a field. The G.P.S. system was developed by the United States Department of Defense and consists of an array of 24 satellites in orbit around the earth, (currently with 2 backup satellites). The system can now be used on a 24 hour per day basis throughout the world.

Field scale G.I.S. databases can be used to store and retrieve data pertinent to a particular field management grid cell. Grid cells can be made to any size, *within* geometrically, technically and economically feasible limits. A number of G.I.S. databases, with different grid sizes, depending on the data type, would be constructed using various sensors to acquire the required data about a field. Such data would include attributes of the soil, terrain and crop condition. These databases would then be combined to create a single field management implement control application grid.

A potential limit is to make the field management grid cells for a particular field operation, *at least* two times as wide, in both directions, as the width of the particular implement used. This ensures that the machine will pass through each cell at least twice, so that a good representation of that particular cell of the field can be acquired. If the cell was only one implement width wide, for example, the implement center (where the G.P.S. receiver is ideally located) could possibly miss that particular G.I.S. grid cell. This would cause the automatic application of the implement control to be based on the neighboring cells, rather than the correct cell.

Advantages of a G.P.S. Based Guidance System

The benefits to be gained from the use of a G.P.S. based guidance system are many. This system would provide a level of machine control, over an extended period of time, that is not possible by even an experienced human operator. A desired performance level for the guidance system should keep the machinery within 5 centimeters of the planned travel path at all times. This software has not been tested to determine if this can be accomplished, since there was no G.P.S. hardware available to the author to test at the time of writing. Computer simula-

tion (Erbach, Choi and Noh 1991) has shown that a tractor *can* be controlled to within a 5 centimeter position error using one-step-ahead deterministic prediction models. This level of precision would be higher than a human operator could sustain over an entire day, especially at "higher" field speeds.

By using the G.P.S. system for navigation, the machinery could automatically be positioned on the same travel path every time it goes through the field. This level of precision would minimize skips and overlaps when travelling through the field.

A major advantage of this type of guidance and control system is that precision farming can contribute to reduced groundwater pollution. The information obtained from a G.I.S. database can be used to tailor the rate of application of fertilizer and other chemical inputs to the specific requirements of a particular "cell" within the field. Only the amounts of material needed by the soil in a specific cell will be applied to the field, and major skips and / or overlaps will be avoided. This would reduce the chance of groundwater contamination by controlling the application of fertilizer or other chemicals, such as herbicides and pesticides.

The input **and** output parameters stored in the G.I.S. database will allow the user to document what has been done in any part of any field, as well as when it was done and why it was done. All of the information that can be measured, within current technological, economic and other practical limits, *can* be available via the input and output G.I.S. database files. The G.I.S. database(s) could be used to keep track of weeds, insects etc., with the information about a certain problem being available to the user long after *visible* signs of the problem have been eliminated, even in later years (Larsen, Tyler, Nielsen 1991).

Wheel track soil compaction is of critical importance in some areas of the country, and the location of wheel tracks can be controlled by precision farming. By using G.P.S. to navigate through a field, the machinery can be programmed to automatically follow the same travel paths through the field each and every time. This would create a permanent roadway and keep the wheel tracks off the rest of the field. Soil compaction due to tire tracks would be minimized, with no extra effort by the machine operator.

The ultimate goal of precision farming is to obtain better control on all farming practices with the goal of reducing inputs of time and money, therefore maximizing productivity and efficiency while also minimizing negative environmental impact. Precision farming will apply only the required amounts of material at the correct location, and in a timely manner. This will result in a savings over the conventional, broadcast farming methods that used manual steering and implement control.

CHAPTER 2

GUIDANCE SYSTEM

Overview

Hardware and software for a guidance system, as outlined in the following sections, are integrated into a framework that, when fully developed, will provide for automatic steering and implement control. The current software is not capable of being used immediately for machine control, as there is no specific code provided where a hardware input or output is required. This can be developed later, when the hardware is available for coding, testing or use.

Hardware Requirements

A number of hardware items must be combined to form a precision farming system. The main hardware consists of a prime mover (tractor), an implement, a portable differentially corrected real-time G.P.S. receiver (with a base station, to provide the differential correction capabilities), and a portable computer system (with its peripherals – monitor and keyboard). Other required components include a steering angle transducer, a steering correction mechanism (servo or step motor), a hitch “bump stop” sensor to indicate that the hitch is at full swing (if equipped with a swinging drawbar or three-point hitch), and the G.P.S. mast angle indicators. Also required are the necessary communication cables, A/D or

D/A converters and other materials required to connect the hardware components and subsystems to the computer system. A more detailed description of the interaction between the hardware and the software is described in the software module sections of this paper.

Software Overview

The software developed consists of several major sections, or modules, of computer code written in the (ANSI) 'C' language; Borland Turbo C, version 3.0 (DOS) was used in the software development. The main function modules of this computer program are:

TRACTOR

IMPLEMENT / G.P.S.

FIELD STATUS

FIELD TEST

TRAVEL PATH DISPLAY

FARM FIELD

There are also support modules to these main modules, these are:

G.P.S. MAIN

GENERAL INPUT

GENERAL ERROR

G.P.S. HEADER

The actual names of these modules (in the computer program) are abbreviated to the DOS name limit of eight characters.

The desired travel path through a field is defined as the target travel path, and the navigation system is designed to follow the target travel path through a field. The target travel path is constructed of lines and arcs, and has the potential to

use points along this path as control points for future program development. This continuous line method is used instead of a series of points with travel from one point to the next. It is easier to construct the travel path from line(s) and arc(s); point(s) can be included along these path segments, if needed. This type of a travel path construction also makes it easier to calculate the turning radii for the path segments. A straight line has an infinite turning radius (with a steering angle of zero), and the turning radius, when the machinery is on an arc segment of the travel path, is constant (with a constant steering angle). Steering angles would be constant only when there is no travel path error, i.e. the machinery followed the travel path perfectly. The method of travel path construction is described in more detail later in the report and suggestions are made for future development and improvement. The program software overview is as follows.

The G.P.S. MAIN module is the main program menu system. It contains the sign on and sign off messages and it sets up the computer display parameters. The G.P.S. HEADER module contains all of the function prototypes, global definitions, type definitions and constants used in this program. The standard libraries required in the program are also called in the header section.

The GENERAL INPUT module contains all program file handling routines for writing to and/or reading configuration files. The appropriate file name extensions are appended to the user specified file name when a file is opened for writing or reading. Other functions, such as the user prompt "Press the <Space Bar> to continue program" and double precision and integer input values are also contained in this module.

The GENERAL ERROR module contains the warnings and errors that are used for most of the program. These error messages are used primarily for the Drawing Exchange (or Interchange) Format (DXF) to Travel Path Data (TPD)

file conversion module (TEST FIELD) and for the tractor configuration input module (TRACTOR). If any error is made by the user during the creation of either type of file, this module warns the user of the problem, and will not save the file until the errors have been corrected.

The TRACTOR, IMPLEMENT and FIELD STATUS modules of the program are where the tractor, implement and field parameter configuration files (respectively) can be created, modified or viewed. These configuration files have the file name extensions of .TRC, .IMP and .FSC respectively.

The FIELD TEST module of the program is where a CADD drawing DXF output file is tested for validity and logic and, upon success, converted to a TPD file for use in the FARM FIELD section of the program. The field target travel path is also displayed for the user in the field test module. A separate module called TRAVEL PATH, is used to display a previously tested target travel path.

The FARM FIELD module of the program is the main module of the program in that it is where the inputs of the entire system are brought together, processed, and outputs to the system are produced. A more detailed description of what is done within each of these modules is given in chapter 3.

CHAPTER 3

DESCRIPTION OF SOFTWARE PROGRAM MODULES

Main G.P.S. Module

The module, GPS1.C is the **main** program function. The C code listing for this module is contained in appendix A. The primary function of this module is to provide the main program menu to the user. The main menu prompts the user to enter the operations they want to perform. Their options are to create, modify or view a tractor, implement or field parameter status configuration file. The user can also test a proposed field travel path, created from a CADD DXF output file, display a target travel path or run the machinery through the field according to the required input software file specifications. The video display colors are set and sign on / sign off messages are displayed in this module as well.

G.P.S. Header Module

The module, GPSHEADE.H includes all standard information that is required for the program to function. The C code listing for this module is contained in appendix B. This module includes the standard program libraries, all function prototypes, all data type definitions and all global constants used in this program.

The global constants used by this program are as follows:

The constant pi (3.14159265359) is used in calculations to convert degrees to radians and vice-versa. The constant Standard_Time_Delay (2000 milliseconds) is used to prevent certain output screens from being cleared immediately. The constant Zero (0.0) is used for clarity, since this "constant" could possibly be changed to a "small" number when used as a tolerance variable in the program. The constant GPS_Update_Interval (set to 0.25 seconds) could be changed to accommodate the position update interval of a particular type of G.P.S. receiver. The constant Dist_Tlrnc (set to 0.01 meters) is used as an error tolerance when calculating distances used in this program. The constant Travel_Path_Err_Tol (set to 0.5 meters) is the maximum distance that implement can be physically separated from the starting point of the field travel path and be considered as being "initialized" to the starting point of the field travel path. The constant Width_Tol (set to 0.05 meters) is the tolerance on the travel path to account for the "weaving" of the implement, about the travel path, as it is being pulled through the field. The constant Max_Impl_Width (set to 50.0 meters) is used as an error check on the user's input of the width of an implement. The constant Max_Impl_Hitch_Length (set to 25.0 meters) is also used as an error check on the user's input to the hitch length.

General Input Module

The module, GENINPUT.C is the program function where file configuration input and output is performed. The C code listing for this module is contained in appendix C. Whenever a configuration file(s) (as described in the tractor, implement and field status sections) is created, saved or modified, the general input module function, Open_File, appends the appropriate file name extension to the user specified file name. The correct file is then opened for writing or reading,

depending on the user's specification. After the information has been read from or written to the file, the file is closed in the function in which it was created.

Other functions of the general input module are to provide a double precision input function, named `Double_Precision`, and an integer number input function, named `Integer_Input`. A small function, `Continue_Program`, that pauses the program and waits for the user to press the space bar before the program will continue is also provided in this module.

General Error Module

The module, `GENERERROR.C` is where all warnings and errors pertaining to the creation of a TPD file from a DXF file, and the tractor configuration creation process, are handled. The functions are called `General_Error` and `Tractor_Configuration_Logic`. The C code listing for this module is contained in appendix D. The incoming codes can either pertain to an error, a warning or a notice to the user. These codes are summarized as follows:

If the incoming error code is a 100 level code, an error has occurred in the DXF to TPD file creation process. An error can occur due to a number of conditions. These include the following:

A DXF output file was a "full" DXF output file when the DXF output file **must** be created using the entity selection mode. This means that the travel path entities have to be selected *individually*, in the order that they are to be followed in the field rather than simply exporting the drawing as an entire (full) DXF file.

The DXF output file is a null file, in other words, no entities were selected for inclusion into a travel path file.

The DXF output file contains no data.

Invalid DXF output file. The output file section of the file must start with a POINT entity type.

Invalid DXF output file. The output file contains only a start POINT, there are no travel path entities in the file; there must be at least one line or arc as a travel path to follow.

An invalid entity type has been detected; valid travel path entities consist solely of POINT(S), LINE(S), and ARC(S).

The error messages coming from the TEST FIELD module pertaining to the creation of a TPD file are due to an invalid DXF output file. If the user does not specify a proposed travel path, from the CADD DXF output file, with a POINT as the starting entity and at least one other valid travel path entity, a LINE or an ARC, the program will return an error message to the user.

If the incoming error code is a 200 level code, a warning is issued to the user during the DXF to TPD file creation process. A warning can occur due to a number of conditions. The following travel path entity / entity interfaces are checked for validity.

POINT / LINE

LINE / POINT

LINE / LINE

POINT / ARC

ARC / POINT

ARC / ARC

LINE / ARC

ARC / LINE

The warnings indicate that the travel path has been constructed invalidly or illogically. This means that either a turning radius was too small for the imple-

ment width specified in the implement configuration file (the implement turning radius must be at least the width of the implement) or that the travel paths were specified in a discontinuous order. This means that there were gaps between the consecutive travel path segments (entities). Other warnings include travel path entities not tangent to each other at their start and end points or two POINT entities in succession. The methods used to determine the validity and logic of the travel path are explained in more detail in the TESTFIEL.C module section of this report.

Upon receiving an error message, the program will halt and return control to the user. If a warning occurs, the user is notified of the problem and the program continues until the entire proposed travel path is processed. The DXF output file is not converted to a TPD file if there are warnings or errors present. The user must first correct the problem(s) before attempting to continue the program.

The `Tractor_Configuration_Logic` function of this program is designed to catch any logical errors in the creation of a tractor configuration file. If there are any logical errors, the error trapping function will alert the user to the problem. There are many logic problems possible when specifying a tractor configuration file. These errors include, but are not limited to, the following items, which are checked in the `Tractor_Configuration_Logic` function of the program.

The error messages for one specific type of tractor configuration, a front wheel steering tractor with a swinging drawbar, are now detailed. A swinging drawbar cannot be located at the front or center of the tractor. The drawbar swing point cannot be located ahead of or behind the front axle of the tractor (it is allowed ahead of the rear axle on all tractor configurations). The tractor's drawbar swing point must be provided for tractors configured with swinging drawbars and three-point hitches. Note that the swing point for a three-point

hitch is the virtual point about which the hitch would pivot if free to swing about that point.

The preceding error messages correspond to one possible type of tractor and hitch type configuration. The user cannot save a tractor configuration file until all error(s) have been corrected. This is due to the fact that the input configuration is looped through the tractor modification function until the user has corrected the problem. Similar errors messages dealing with specific configuration errors can be seen in appendix D.

Tractor Configuration Module

The following explanation describes what the TRACTOR.C program module does and explains each of the functions within the program module. In essence, the module allows for a complete configuration of the geometry of a tractor to be developed, saved, modified and reviewed by the program user.

This is the module, where the user can create a new tractor configuration file, modify an existing file or view an existing file (to check its specifications). There are seven tractor configuration parameters. These parameters are steering type, hitch type, hitch swing point (where applicable), hitch location, wheelbase, drawbar length, hitch swing distance (where applicable), maximum hitch swing angle (where applicable) and the maximum steering angle of the tractor. The geometry of a front wheel steered tractor with a swinging drawbar is presented in figure 1. Figure 2 shows the detailed geometry of the swinging drawbar. The C code listing for this module is contained in appendix E.

The `Configure_Tractor` function permits the user to enter their choice of creating a new file, modifying or viewing an existing tractor configuration file or exiting from this sub-menu back to the main program menu.

FIGURE 1
FRONT WHEEL STEERING TRACTOR WITH SWINGING DRAWBAR
[TOP VIEW]

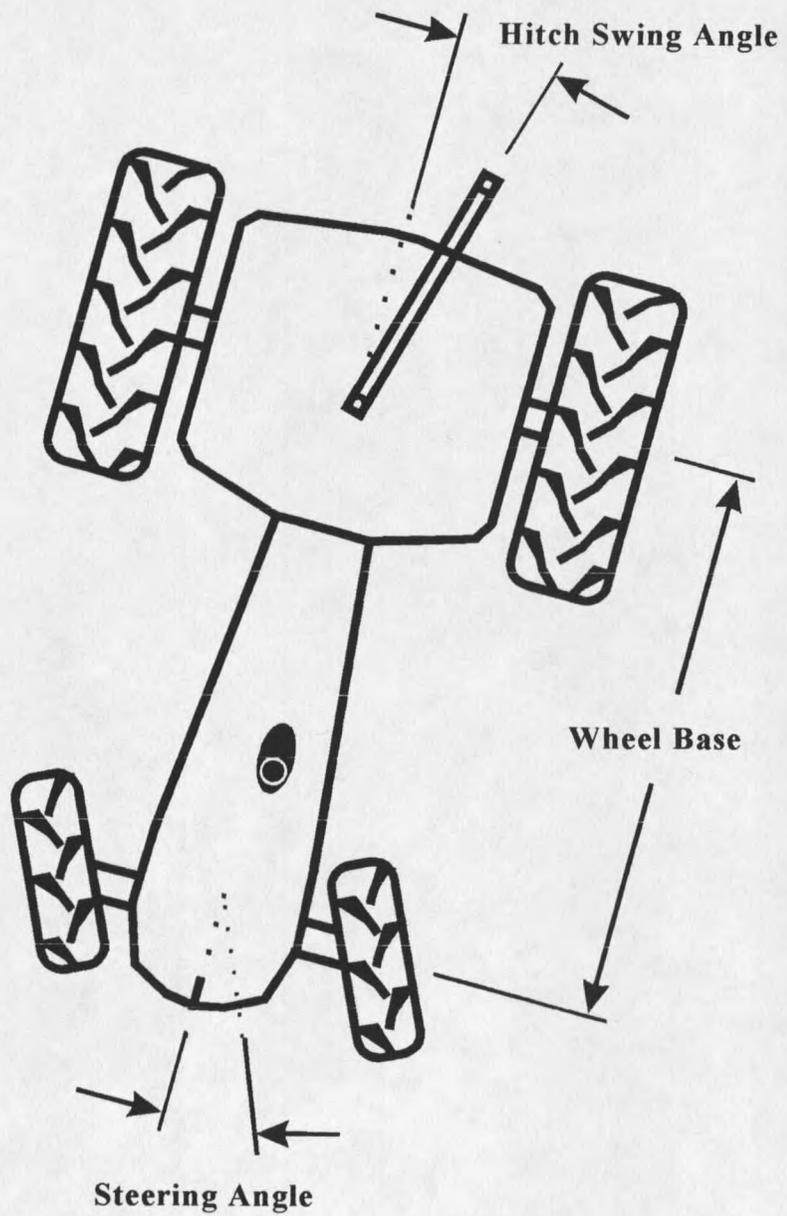
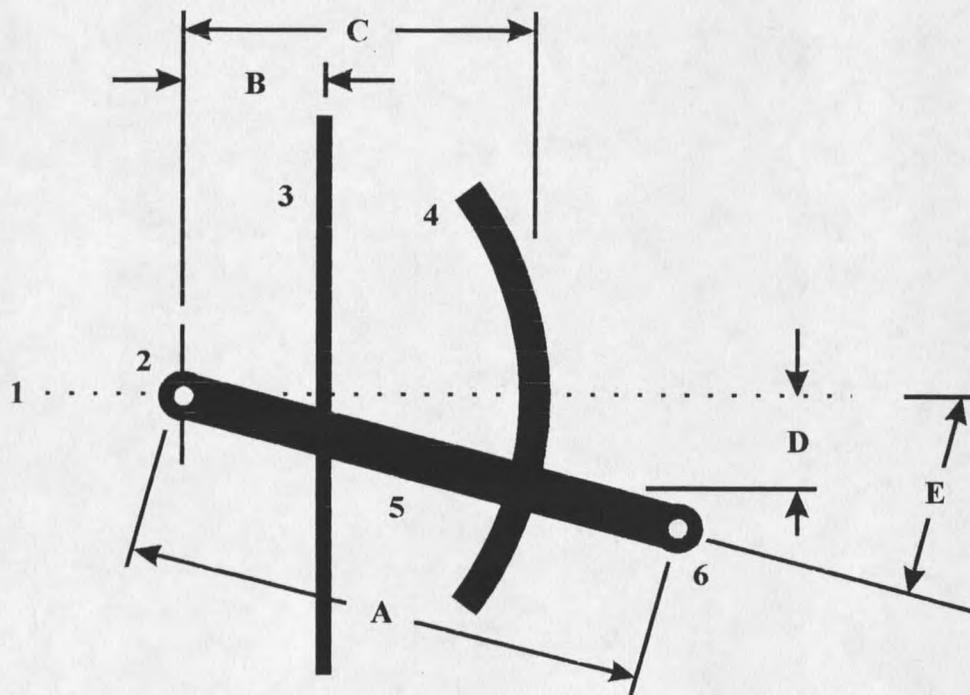


FIGURE 2
 SWINGING DRAWBAR TRACTOR HITCH (DETAIL)
 [TOP VIEW]



Key to Hitch Components:

- | | | | |
|---|------------------------|---|--------------------------------|
| 1 | Tractor centerline | A | Hitch length |
| 2 | Hitch swing point | B | Hitch Swing distance |
| 3 | Tractor rear axle | C | Hitch reference distance |
| 4 | Hitch support frame | D | Hitch swing reference distance |
| 5 | Swinging drawbar | E | Hitch swing angle |
| 6 | Drawbar hitch pin hole | | |

If the user chooses to create a new tractor configuration, the function `Create_Tractor_Configuration` is called. From this function, the steering type, hitch type, hitch location, wheelbase and drawbar length input functions are called. If the hitch type is a swinging drawbar or a three-point hitch, the user is then prompted to enter the hitch swing point, hitch swing distance and the maximum hitch swing angle, otherwise the tractor hitch swing point is set to be not applicable and the hitch swing distance and maximum hitch swing angle are both set to a value of zero. The user is then prompted to input the maximum steering angle of the tractor.

After the values have been input, the new configuration parameters are sent to the `Check_Tractor_Configuration` function. From this function the parameters are sent to the `Tractor_Configuration_Logic` function as described in the GENERAL ERROR module section. Once the values for the tractor configuration are correct, they are sent to the `Tractor_File_Info` function. This function displays the current configuration of the tractor's parameters. The user then has the option of changing the parameter values, saving the file or returning to the previous menu.

If the user chooses to modify a tractor configuration file, the function `Modify_Tractor_Configuration` is called. The user is then presented with a list of the tractor configuration parameters to choose to modify or the user can return to the previous menu. If the user decides to modify a particular configuration parameter, the appropriate parameter input function is called.

If the user decides to view an existing tractor configuration file, the function `Open_Tractor_Config_File` is called to prompt the user to input the file name of the tractor configuration that they wish to review. Then, the `Tractor_File_Info`

function is called to display the tractor configuration. When the user presses the space bar, the program returns to the previous menu.

A description of the tractor configuration parameter options follows this list of the tractor configuration parameters:

- Steering Type
- Hitch Type
- Hitch Location
- Wheelbase
- Hitch Length
- Hitch Swing Point
- Hitch Swing Distance
- Hitch Swing Reference Distance
- Hitch Support Distance
- Maximum Steering Angle

The variable, `Tractor_Steering_Type` input in the function `Steering_Type`, can be front wheel, articulated or frame, skid or controlled differential, all wheel or rear wheel steering.

The variable, `Tractor_Hitch_Type` input in the function `Hitch_Type`, can be swinging drawbar, rigid drawbar, three-point hitch or a semi-integral type of hitch.

The variable, `Tractor_Hitch_Location`, input in the function `Hitch_Location`, can be specified as being located at the front, center or rear of the tractor. Some types of hitches cannot be applied to certain sections of a tractor, as has been described in the GENERAL ERROR module section of the program description.

The variable, `Tractor_Wheelbase`, input in the function `Wheelbase`, is the wheelbase of the tractor. This measurement is from the center of the front axle to the center of the rear axle of the tractor.

The variable, `Tractor_Drawbar_Length`, input in the function `Drawbar_Length`, is the critical length of the particular type of hitch. For a rigid drawbar, this length is the distance from the center of the hitch-pin hole to the center of the rear axle. For a swinging drawbar, this length is the distance from the center of the hitch-pin hole to the center of the pivot pin where the other end of the hitch is attached to the tractor. For a three-point hitch, this length is the distance from the center of the mounting pin holes on the lower links to the (actual or virtual) pivot of the three-point hitch. For a semi-integral hitch, this length is the distance from the center of the pivoting mount points to the center of the reference axle on the tractor. The tractor's reference axle is the one which is located closest to the mounted implement.

The variable, `Tractor_Hitch_Swing_Point` input in the function `Hitch_Swing_Point`, can be, not applicable to this tractor, located ahead of the front axle, behind the front axle, ahead of the rear axle or behind the rear axle. The difference between, behind the front axle or ahead of the rear axle, is only when the hitch swing point is applied to an articulated steering type tractor. With this type of steering system, the hinge point separates the tractor into two distinct halves – front and back. With all other types of steering systems, there is no distinction between ahead of the rear axle or behind the front axle, so the default should be set appropriately relative to where the hitch is located. For example, if a front wheel steering tractor had a swinging drawbar located at the rear, the hitch swing point would be set ahead of the rear axle.

The variable, `Tractor_Hitch_Swing_Distance`, input in the function `Hitch_Swing_Distance`, is the length of the swinging distance between the center of the reference axle on the tractor and the pivoting point(s) on the hitch.

The variable, `Tractor_Max_Hitch_Swing_Angle`, input in the function `Max_Hitch_Swing_Angle`, is used for three-point and swinging drawbar tractor hitches. This angle is the angle between the centerline of the tractor and the centerline of the hitch reference member when the hitch is at its maximum swing. The hitch reference member for a swinging drawbar is the swinging drawbar itself, while the reference member for a three point hitch is either one of the lower links. The maximum hitch angle is computed by taking the arcsine of the variable `Tractor_Swing_Ref_Dist` divided by the variable `Tractor_Hitch_Ref_Dist`. The variable `Tractor_Hitch_Ref_Dist` is the distance from the center of the swinging drawbar pivot point to the centerline of the hitch at the center of the swinging drawbar support frame. For the case of a three-point hitch, this distance is the (real or virtual) distance from the pivot point of the hitch to an arbitrary reference point along the length of one of the lower lift arms. The variable `Tractor_Swing_Ref_Distance` is the perpendicular distance between the tractor centerline and the contact point at the centerline of the swinging drawbar where it rests on its support. For the case of the three-point hitch, this distance is the perpendicular distance from the centerline of the tractor to the centerline of the lower link arm when it is at its maximum swing angle minus the same point when it is at its normal swing angle (the angle when the implement is in its normal position straight behind the tractor). This variance in the angle is "small" enough to negate any explicit travel error, any implicit error due to this method should be taken into account by the steering feedback and control.

The variable, `Tractor_Max_Steering_Angle`, input in the function `Max_Steering_Angle`, is the maximum steering angle for the particular tractor steering type. For a front or rear wheel steer tractor, this angle is the angle between the centerline of the tractor and the centerline of the steering tires when the steering is at its maximum left or right position. Caster action front wheel steering systems are handled the same as a front wheel steer tractor. For an articulated steering tractor, the maximum steering angle is the angle between the centerline of the rearward section of the tractor and the centerline of the forward section of the tractor. The maximum steering angle for an all wheel steer tractor would be the angle between the projected center-lines of the front and rear steered tires when they are steered to their maximum angles. For a skid steered tractor, the maximum steering angle is, in essence, infinite. This steering angle would have to be limited to some maximum amount for practical applications. This angle would vary depending on what type of hitch was being used. For example, any maximum steering angle (up to 180 degrees in either direction) could be used if the implement was mounted on the tractor with a three-point or semi-integral hitch. The maximum steering angle would have to be limited to approximately forty-five degrees if a rigid or swinging drawbar were used. This is to keep the turning radius of the tractor large enough to prevent interference between the tractor and the drawn implement. Similar steering angle limits would have to be placed on other tractor steering methods if the tractor could steer "sharp" enough to cause interference between the tractor and the towed implement.

The function, `Open_Tractor_Config_File` prompts the user for a tractor configuration file name, appends the appropriate file name extension, `.TRC`, and

opens the file. The file is then either written to or read from depending on the file-mode that was chosen by the user.

Implement Configuration Module

The following explanation details what the `IMPLEMENT.C` program module does and explains what each of the functions within the program module do. In essence, the module allows for a complete configuration of the geometry of an implement and G.P.S. receiver to be developed by the program user.

The implement configuration module is quite similar to the tractor configuration module in that it accomplishes the same types of functions for the implement. This is the module, where the user can create a new implement and G.P.S. receiver configuration file, modify an existing file or view an existing file (to check its specifications). The C code listing for this module is contained in appendix F.

There are two implement and configuration parameters and five G.P.S. receiver configuration parameters. These parameters are the implement hitch length and working width, the G.P.S. receiver height, and the distance forward or rearward and left or right of the center of the effective working area of the implement. Figures 3 and 4 show the implement and G.P.S. receiver geometry respectively.

During the implement / G.P.S. receiver configuration creation process function, `Configure_Implement_GPS`, the user is prompted to enter his / her choice of creating, modifying or viewing an implement and G.P.S. receiver configuration file or exiting from this sub-menu back to the main program menu.

If the user decides to create a new file, the function `Create_Implement_GPS_Configuration` is called. This function prompts the user to input the implement hitch

FIGURE 3
IMPLEMENT GEOMETRY
[TOP VIEW]

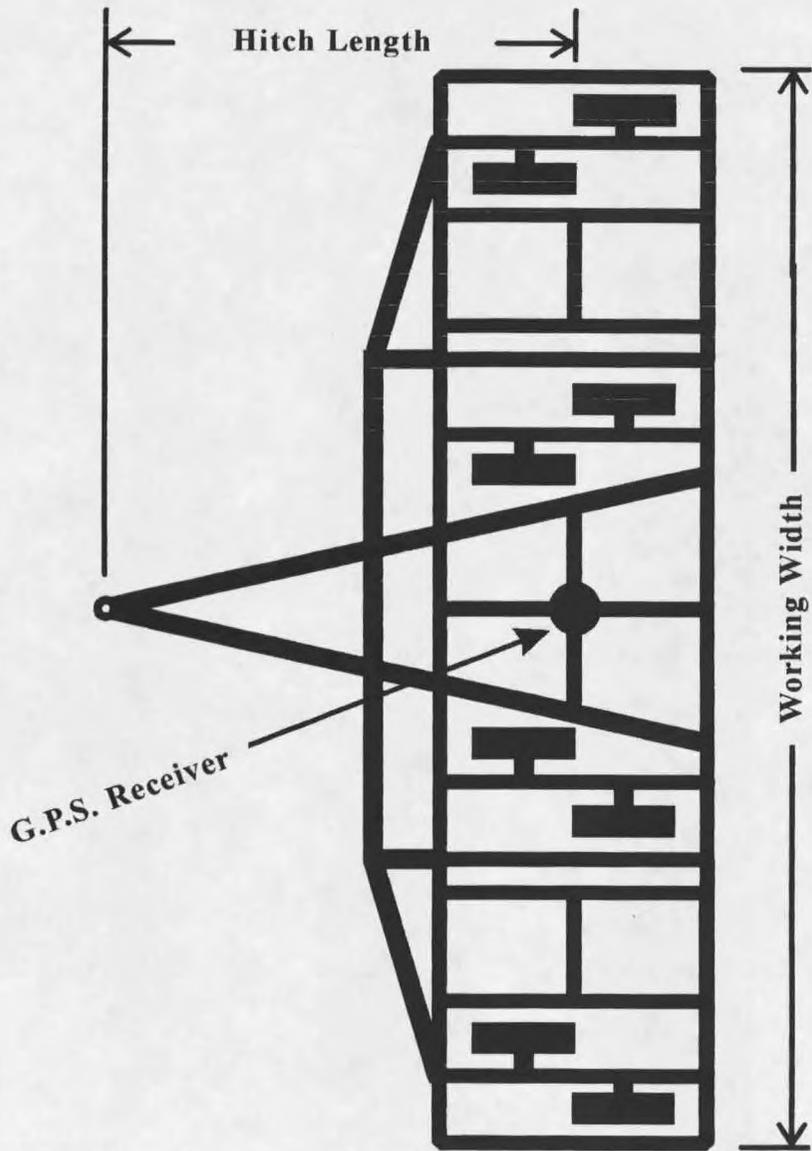
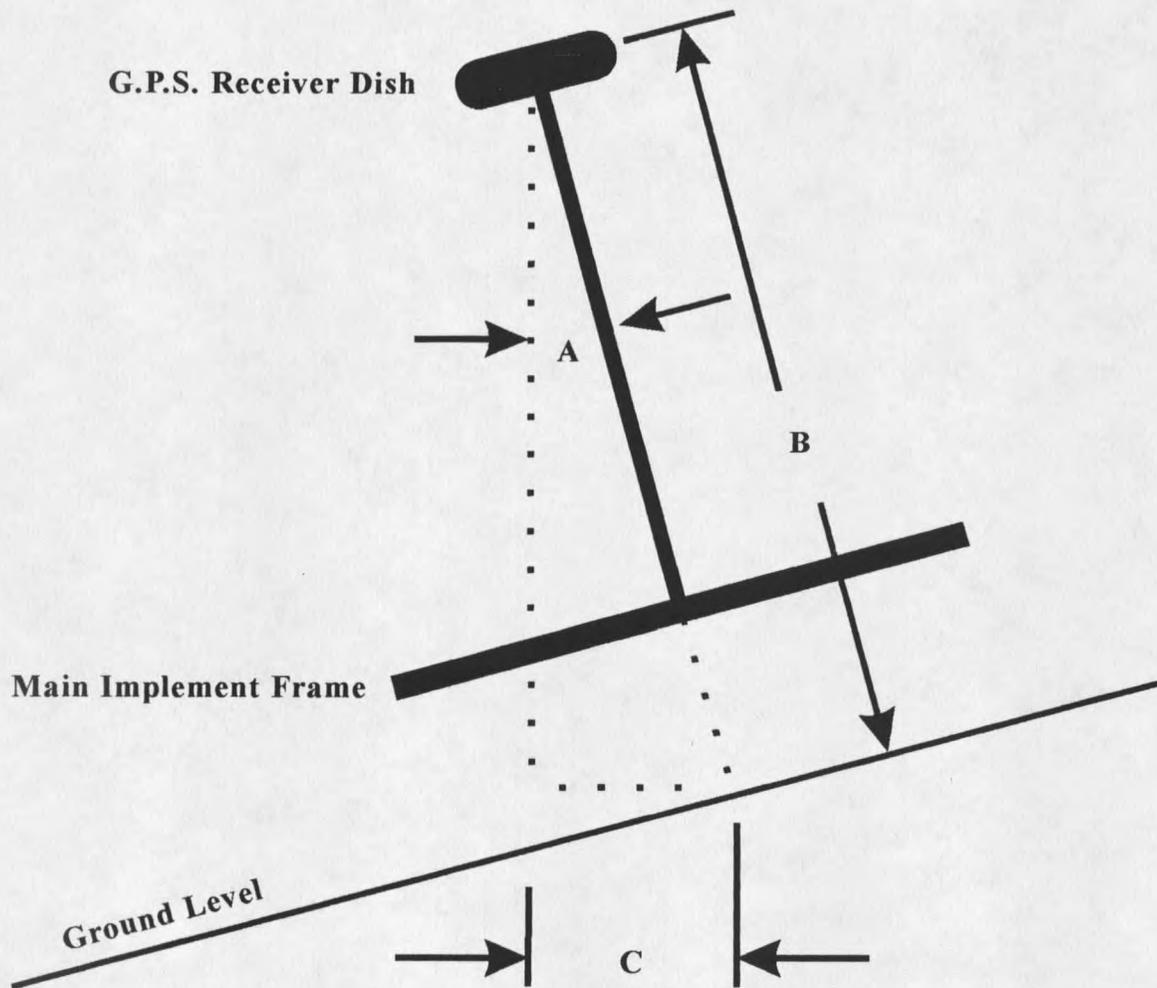


FIGURE 4
G.P.S. RECEIVER GEOMETRY (DETAIL)
[SIDE VIEW]



Key to the G.P.S. receiver geometry dimensions:

- A G.P.S. receiver mast angle
- B G.P.S. receiver mast height (working plane of implement to center of receiver)
- C Horizontal distance between G.P.S. receiver reading and true implement center

length and working width. The user is prompted to enter the height of the G.P.S. receiver above the working plane of the implement. Next the user is prompted to enter whether the G.P.S. receiver is forward or rearward of the center of the effective implement working area. Then, the user is prompted to enter whether the G.P.S. receiver is located to the left or to the right of the effective working area of the implement. After this, the user is prompted to enter the distances forward or backward, left or right of the center of the implement, if the user has indicated that the G.P.S. receiver is not located at the center.

Once all of the appropriate implement / G.P.S. receiver configuration parameters have been entered by the user, they are sent to the `Check_Implement_GPS_Configuration` function. This function sends the parameters to the `Implement_GPS_File_Info` function. This function displays all of the implement / G.P.S. receiver configuration parameters to the user. The user is then prompted to either save this file, modify this file or to return to the previous menu without saving the file.

If the user decides to modify an implement / G.P.S. receiver configuration file, the function `Modify_Implement_GPS_Configuration` is called. From here the user is prompted to enter the letter corresponding to the particular parameter that he / she would like to modify, or the user can return to the previous menu. If the user decides to modify one or more parameters, the appropriate configuration parameter input functions are then called.

If the user decides to view an existing implement / G.P.S. configuration file, the function `Open_Implement_Config_File` is called. This function retrieves the configuration data from a disk file; this function is also used to write the configuration parameters to a file if this is required by the program. The information is then sent to the `Implement_GPS_File_Info` function to display the configuration

parameters to the user. When the user presses the space bar, the program returns to the previous menu.

A description of each of the parameters in the implement / G.P.S. receiver configuration file is as follows:

The variable, `Implement_Hitch_Length` input in the function `Hitch_Length`, is the distance from the center of the effective working area of the implement to the center of the reference connection to the tractor. The effective working area of the implement is the area enclosed by the soil engaging units or an analogous area pertaining to other types of non-contact implements. For the case of a sprayer, this "area", the spray boom, would actually be modeled as a line. The reference connection on the tractor would be the center of the hitch pin hole for rigid and swinging drawbars, or the center of the holes on the lower link arms of a three-point hitch, or the center of the mounting holes on a semi-integral hitch. This length is error checked in the input function to determine that the user has not input a negative value; this value is also checked against a predetermined maximum hitch length value to prevent the user from entering too "huge" a value for the hitch length.

The variable, `Implement_Working_Width` input in the function `Working_Width`, is the distance from one edge of the effective working area of the implement to the opposite edge of the effective working area. These reference edges are the left and right most points of "engagement" of the implement with the field. The implement working width undergoes the same type of negative and maximum value checks as the hitch length.

The variable, `GPS_Receiver_Height` input in the function `Receiver_Height`, is the distance from the working plane of the implement to the center of the actual receiver antenna. The working plane of the implement is defined to be the plane

of the working area of the implement, at ground level. The G.P.S. receiver antenna must be able to stay at the same height relative to the ground whether the implement is raised or lowered. If this cannot be accomplished, other measures will have to be taken to accommodate for this, since the program is not currently set up to handle other cases of receiver mounting.

The variables, `GPS_Forward_Back_Dir` and `GPS_Forward_Backward_Dist` are input in the functions `Forward_Backward_Dir` and `Forward_Backward_Dist` respectively. These variables allow the G.P.S. receiver antenna to be positioned either forward or rearward of the center of the effective working area of the implement.

The variables, `GPS_Left_Right_Dir` and `GPS_Left_Right_Dist` are input in the functions `Left_Right_Dir` and `Left_Right_Dist` respectively. These variables allow the G.P.S. receiver antenna to be positioned either left or right of the center of the effective working area of the implement.

It is highly recommended that the G.P.S. receiver antenna be positioned as close to the center of the effective working area of the implement as possible, to minimize any positioning error due to possible skewing of the implement that might occur while traveling through the field. This skewing, due to steep slopes and or slippery or hard conditions could throw the steering correction off more, due to geometric considerations, than if the receiver is centrally mounted on the implement.

An example of this, taken to an extreme, would be if the G.P.S. receiver antenna was mounted on the implement, near the tractor hitch. If the tractor was traveling through a condition of extreme side slope or mud, but continuing on in a fairly straight line, the center of the effective working area of the implement

may not be following the centerline of the tractor, while the G.P.S. receiver antenna would be "fairly" close to the desired travel path.

Another reason for the G.P.S. receiver to be mounted near the center of the implement's working area is so that the speed of the receiver will not be skewed too high or too low when the implement is following an ARC entity. The edge of the implement on the outside of the curve will be going significantly faster than the edge of the implement on the inside of the curve. Positioning the receiver close to the center of the implement will tend to "average" these speed differences out over the entire implement. The center of the effective working area of the implement is where the steering correction calculations are made, since it is more critical that this point follow the travel path than the centerline of the tractor.

Field Status Configuration Module

The field status configuration module is quite similar to the tractor and implement / G.P.S. receiver configuration modules in that it accomplishes the same types of functions. This is the module, where the user can create a new field status configuration file, modify an existing file or view an existing file (to check its specifications). The C code listing for this module is contained in appendix G.

There are six configuration parameters in the field status configuration module related to the field's reference base point, and six parameters dealing with where the user wants the machinery to start and to stop working the field. These parameters are the field base point reference in degrees, minutes and seconds of latitude and longitude, the local (field) x and y coordinates for the travel path start point and end point, and the travel path starting and ending entity numbers. These starting and ending entity numbers are the number assigned to the particu-

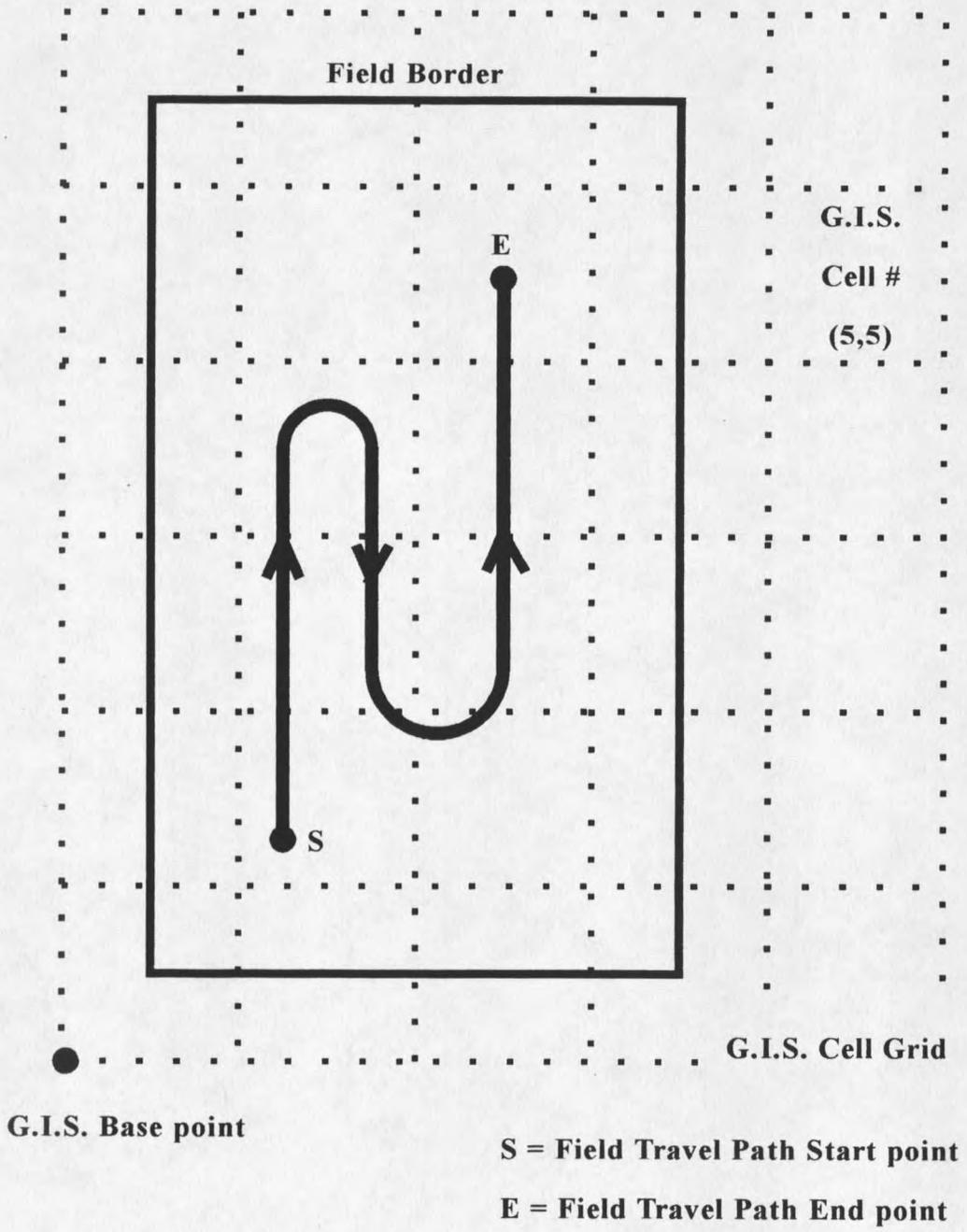
lar entity along the travel path defined by the user. A diagram of the field status parameters and the G.I.S. cell grid system is shown in figure 5. A more detailed explanation of the construction of the travel path and its related parameters is given in the TESTFIEL.C program module.

During the field status configuration creation process function, `Configure_Field_Status_Info`, the user is prompted to enter their choice of creating, modifying or viewing a field status configuration file or exiting from this sub-menu back to the main program menu. If the user decides to create a new file, the function `Create_Field_Status_Info` is called. This function prompts the user to input the field status configuration parameters previously mentioned. Once these parameters have been input, the function `Check_Field_Status_Info` is called. This function first calls the function `Field_Status_Info`, which displays the current field status configuration parameters to the user. Next, the user is prompted to enter whether the information is correct and to save the file, in which case the function `Open_Field_Status_Info` is called and the configuration parameters are saved to a file. If the information is not correct, the user can modify any parameter in the file; the user can also return to the previous menu without saving or modifying the file, if desired.

If the user decides to modify the current field status information file, the function `Modify_Field_Status_Info` is called. This function allows the user to change the particular configuration parameter that is incorrect, by calling the appropriate input parameter function, or the user can return to the previous menu if he / she changes his / her mind.

If the user wishes to just view an existing field status configuration file, the function `Open_Field_Status_Info` is called to read in the desired information. This information is then displayed to the user by a call to the function,

FIGURE 5
G.I.S. GRID AND BASEPOINT GEOMETRY
[TOP VIEW]



Field_Status_Info. When the user has reviewed the configuration file, the space bar is then pressed to return the user to the previous menu.

A description of each of the parameters in the field status configuration file is as follows:

The variable, BasePoint_Degrees_Latitude input in the function Degrees_Latitude, is the degrees component of the local field reference base point latitude.

The variable, BasePoint_Minutes_Latitude input in the function Minutes_Latitude, is the minutes component of the local field reference base point latitude.

The variable, BasePoint_Seconds_Latitude input in the function Seconds_Latitude, is the seconds component of the local field reference base point latitude.

The variable, BasePoint_Degrees_Longitude input in the function Degrees_Longitude, is the degrees component of the local field reference base point longitude.

The variable, BasePoint_Minutes_Longitude input in the function Minutes_Longitude, is the minutes component of the local field reference base point longitude.

The variable, BasePoint_Seconds_Longitude input in the function Seconds_Longitude, is the seconds component of the local field reference base point longitude.

The variable, Startpoint_X_Coord input in the function Startpoint_X, is the local x field coordinate of where the user wants the machinery to start working in the field.

The variable, `Startpoint_Y_Coord` input in the function `Startpoint_Y`, is the local y field coordinate of where the user wants the machinery to start working in the field.

The variable, `Endpoint_X_Coord` input in the function `Endpoint_X`, is the local x field coordinate of where the user wants the machinery to stop working in the field.

The variable, `Endpoint_Y_Coord` input in the function `Endpoint_Y`, is the local y field coordinate of where the user wants the machinery to stop working in the field.

The local, or field, coordinates are determined using an x, y, z axes, orthogonal coordinate system using the right-hand rule. The center of the coordinate system is set so that the x axis is aligned with the east-west latitude axis, with increasing values going east. The y axis is to be aligned north-south, with increasing values in the northern direction; since the lines of longitude converge at the poles of the earth, the local field axes will have to be compensated by the G.P.S. receiver program to take this into account, and return a truly rectangular field coordinate value to this program. The G.P.S. receiver program will have to account for the machinery being in the northern or southern hemisphere due to the lines of latitude values starting at zero at the equator and increasing to ninety degrees in both directions.

The variable, `Field_Travel_Start_Entity` input in the function `Travel_StartEntity`, is the value of the travel path data entity where the user wants the machinery to start working the field. The variable, `Field_Travel_End_Entity` input in the function `Travel_EndEntity`, is the value of the travel path data entity where the user wants the machinery to stop working the field.

The preceding explanation details everything that the FLDSTATS.C program module does and explains what each of the functions within the program module do. In essence, the module allows for a complete configuration of the geometry of a field reference base point and its travel path start and end point locations to be developed, saved, modified and reviewed by the program user.

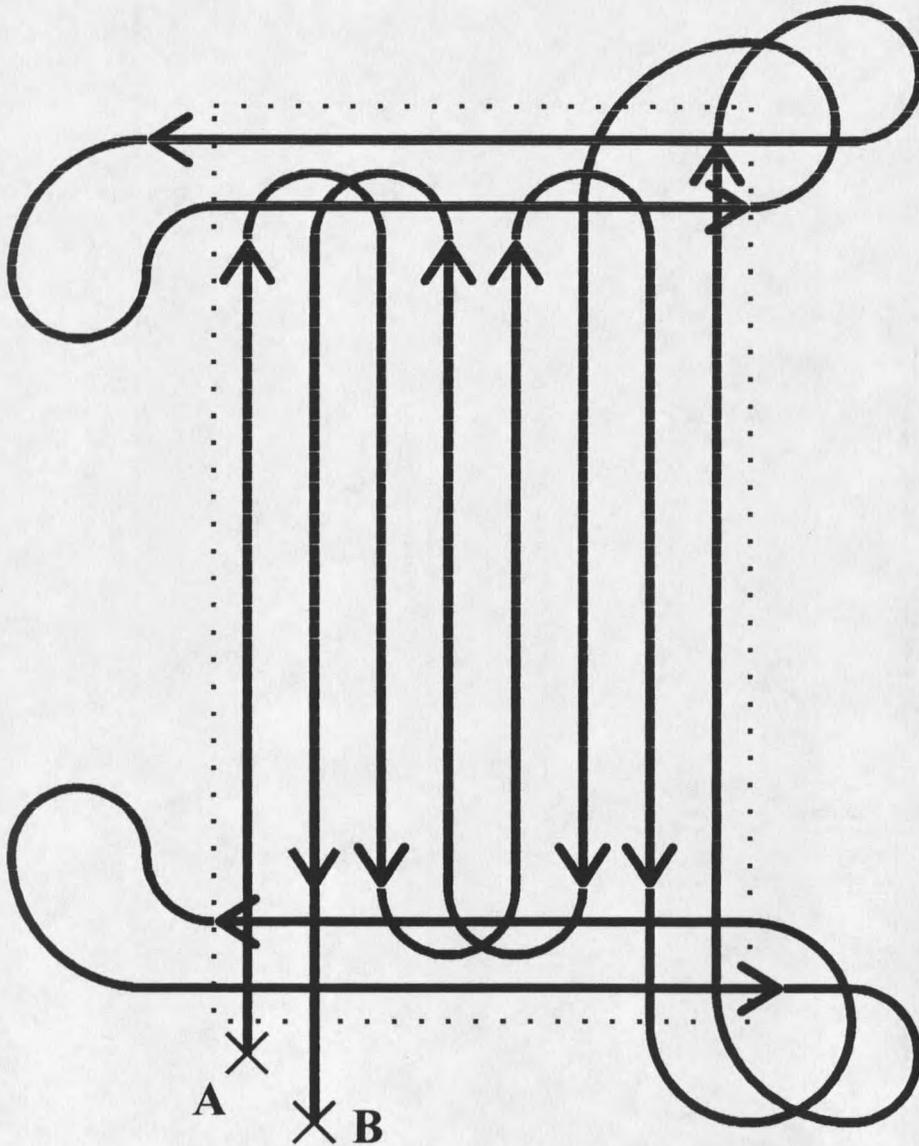
Field Travel Path Test Module

This program module reads entity parameters from a DXF file and checks the logic and validity of the proposed field travel path. The input to this section of the program consists of a .DXF output file, created using a CADD drawing program, and the implement / G.P.S. receiver configuration that is to be used with this particular field travel path. The C code listing for this module is contained in appendix H.

A diagram of a "typical" field target travel path is shown in figure 6. Note the method of travel, utilizing automatic guidance, can be and is different than the travel path used when manually driving through a field. The G.P.S. guided system allows the tractor and implement to work back and forth across the field rather than circling around the field. This travel path minimizes overlapping at the ends of the field compared to the traditional method of farming a field.

As stated above, the user must first create a field travel path for the machinery to follow. This travel path is developed by using a CADD system, or any other desired program so long as a .DXF output file can be generated, to draw the proposed travel path through a field. A simple program could even be written to convert travel path points into a DXF file format. The steps for developing a field travel path are as follows.

FIGURE 6
TYPICAL FIELD TARGET TRAVEL PATH
[TOP VIEW]



Note: The dotted line is the field boundary, the solid lines and arcs represent the target travel path through the field. Point A is the start point and point B is the end point.

First the user will determine the length and width dimensions of a field that is to be farmed as well as determine a base point for the field. The base point should ideally be a point located southwest of the southern and western most point of travel within the field, to keep all field dimensions as positive numbers. This base point should also correspond to the lines of intersection of a set of G.I.S. cells (real or virtual). This base point is used for a reference in determining which G.I.S. cell the machinery is located in. It is also used as the base point for the G.P.S. receiver software when converting the global positioning coordinates into the local coordinates for use in field guidance functions. Therefore it is important that this base point coordinate be carefully chosen so that each use requirement can be met.

Next, the user should set up the particular CADD system used in drawing the proposed field travel path entities, so that the appropriate units of measure and scale are used for length dimensions and angular displacements. The reference axis used for all angle measurements should be set so that it is pointing to the east (3 O'clock position), if this is not already the default value. All angular measurements should be set to degrees; this computer program is set up to convert all input angles to radians for internal use. The user should also be sure that the entity "handles" variable is activated before any entities are drawn. The handles variable is used in the CADD database as a way to keep track of each individual drawing entity. Linear measurements are to be made in the decimal form. The scaled, linear distances drawn in the CADD file should directly correspond to the actual distances of the actual field travel path. There are no provisions currently in this computer program to scale the input dimensions, since this can be much more easily accomplished within the CADD program itself.

Other considerations that should be made are setting the POINT display mode appropriately so that the POINT entity can be easily "picked" when selecting a POINT entity to be placed in a DXF output file. Obviously, the user should also be sure that all travel path entities are drawn so that consecutive entities are connected and tangent to each other. To optimize the travel paths, as many travel paths as possible should be constructed parallel to each other, parallel to the longest side of the field. Sequential paths should also be spaced one implement width minus a "small" distance tolerance apart from each other. This distance tolerance should be set to approximately 10 centimeters, until actual testing of the fully developed system proves another tolerance to be appropriate. This is to prevent the implement from "skipping", or leaving gaps between subsequent travel paths through the field.

The user then selects the travel path entities, one by one, in the order in which the machinery is to be guided through the field. The first point of a proposed field travel path will always be the first entity selected for inclusion into a DXF output file. If the user wishes to have the machinery travel through the field in the opposite direction, another DXF file can be easily produced for this condition by simply selecting the proposed field travel path entities in the opposite direction.

Test_Field_Travel_Path is the main program function for the TESTFIEL.C module. This function prompts the user to enter the file name corresponding to the implement / G.P.S. receiver configuration file to be used when testing a particular, proposed field travel path. The only information that is required from the implement / G.P.S. configuration file is the implement width.

Once the implement / G.P.S. receiver configuration file has been specified, the user is prompted to enter the name of the .DXF file to be tested as a proposed

reference field travel path. Once the file has been successfully opened, a check is made to determine whether the file has any data in it or whether it is a null file. If it is a null file, the program control is sent to the `GENERERROR.C` program module where the appropriate error message is given as described in the general error program module section.

Next, a check is made to determine if the `.DXF` output file that has been specified is a "full" output file. If it is, the appropriate error message is displayed. The user is to specify a `DXF` output file using entity selection only, not a full output file.

Yet another check is then made to determine if there are any entities in the file; the appropriate action is taken depending on the outcome. As soon as the "ENTITIES" section of the `DXF` file occurs, the program enters a loop that will read the first entity of the file. If the first entity is not a `POINT` entity or is an "ENDSEC" line, the appropriate error handling action is taken.

When two proposed travel path entities are successfully obtained, via the `Obtain_Entity_Para_Info` (to be explained in the next section), the program control is sent to the `Check_Travel_Path` function (also to be explained later). Once control returns, the `Error_Warning_Flag` is set, if there was a warning returned from the `Check_Travel_Path` function. If the next entity section is an "ENDSEC" or the next entity is not a `POINT`, `LINE` or `ARC` the appropriate error handling is accomplished.

Next, the entity scan pass counter number is incremented and all of the `Next_...` variables from the `Obtain_Entity_Para_Info` function are copied to the corresponding `Crnt_...` variables. The `Next_...` and `Crnt_...` variables are all `DXF` output file variables corresponding to the different elements associated with a;

particular entity. These parameters will be discussed more in the following section dealing with obtaining the entity information from the DXF output file.

The preceding entity acquire loop is continued until the entire DXF output file has been read and processed. If the DXF output file had any errors in it, no Travel Path Data output file is created; if the DXF output file had warnings, a final reminder warning message is presented to the user, and again no Travel Path Data output file is created. If the DXF file was successfully tested the program is sent to create a Travel Path Data (.TPD) output file via the function, `Create_TPD_File`.

The `Create_TPD_File` copies a 999 (comment) group code and a message, `TPD_TEST_PASS` to the first two lines of the DXF output file. Then, the DXF output file is read in and copied to the .TPD file, line by line. The only other change made to the DXF output file to convert it into a TPD file is to renumber the entity handles. This means that the entities that make up the TPD file will then be numbered in the order in which the user selected them to make up the travel path from start to end. Once the TPD file creation process has been completed, program control returns to the main menu.

Obtaining DXF Entities

The `Obtain_Entity_Para_Info` function is designed to read the parameter information from a DXF output file in a table-driven manner. The first action of this function is to initialize all numerical variables in this section to zero.

At this point the loop used to acquire all parameters of the next entity in the DXF output file is started. The DXF file data is read in pairs. The first parameter of a data pair in the DXF file is the **group code**, which is represented by the variable, `Next_Group_Code` in this program. This variable is a positive, nonzero integer used in the DXF file to indicate what type of data the next entity param-

eter will be. The second parameter of the entity pair is the **group value**, which is represented by the variable, `DXF_Section_Name` in this program. If the `Next_Group_Code` is the number 0 (zero) and the variable `Entity_Flag` is 0 (zero), another entity parameter is scanned into memory, otherwise the entity parameter selection process is completed for this particular entity.

Following is a description of all `DXF_Section_Names` acquired by the `Obtain_Entity_Para_Info` function. The `Next_Group_Code` number will be followed by a brief description of the variable to be obtained and how the particular variable name is used in this program, or how it might potentially be used in later development of this program and what data type the next group value is.

Group Code 0: This identifies the start of an entity, as used in this program. The following value, `DXF_Section_Name`, is a character string.

Group Code 5: This indicates an entity handle, `Next_Entity_Handle`, follows and is input as a hexadecimal string.

Group Code 6: This is the line type name. This particular variable, `Next_Linetype_Impl_UpDown`, is not currently used in this program, but could be used as an implement control code to adjust the implement height or speed relative to its travel path specific position in the field. More detail about ideas for future development of this program will be presented later in this report.

Group Code 8: This is the variable, `Next_Layer_Name`. It corresponds to the layer in which the entity was drawn and is a character string. This particular variable is not used in this program, but could be used as an error check to be sure the user has selected the correct layer corresponding to the particular type of travel path required for a specific implement.

Group Code 10: This is the primary X coordinate (start point of a LINE or Text entity, center of a CIRCLE etc.). This is read in as the floating-point vari-

able, `Next_Primary_X_Coord` and describes the X coordinate of the starting point of a LINE entity as originally drawn in the CADD program. It can also represent the X coordinate of a POINT entity or the X coordinate of the center of radius of an ARC entity.

Group Code 11: This is the secondary X coordinate of an entity. This is read in as the floating point variable, `Next_Secondary_X_Coord` and describes the X coordinate of the end point of a LINE entity as originally drawn in the CADD program.

Group Code 20: This is the primary Y coordinate (start point of a LINE or Text entity, center of a CIRCLE etc.). This is read in as the floating-point variable, `Next_Primary_Y_Coord` and describes the Y coordinate of the starting point of a LINE entity as originally drawn in the CADD program. It can also represent the Y coordinate of a POINT entity or the Y coordinate of the center of radius of an ARC entity.

Group Code 21: This is the secondary Y coordinate of an entity. This is read in as the floating point variable, `Next_Secondary_Y_Coord` and describes the Y coordinate of the end point of a LINE entity as originally drawn in the CADD program.

Group Code 30: This is the primary Z coordinate (start point of a LINE or Text entity, center of a CIRCLE etc.). This is read in as the floating-point variable, `Next_Primary_Z_Coord` and describes the Z coordinate of the starting point of a LINE entity as originally drawn in the CADD program. It can also represent the Z coordinate of a POINT entity or the Z coordinate of the center of radius of an ARC entity. Although the Z (elevation) coordinates of a DXF output file are not specifically utilized by this program, this variable has been implemented into the travel path entity acquire routine. Future development could call for the use

of this variable either as representing elevation or as another auxiliary variable to be used for some type of implement control code.

Group Code 31: This is the secondary Z coordinate of an entity. This is read in as the floating point variable, `Next_Secondary_Z_Coord` and describes the Z coordinate of the end point of a LINE entity as originally drawn in the CADD program.

Group Code 38: This is the entity's elevation if nonzero (fixed). It is output only if the CADD system variable `FLATLAND` is set to 1. For the purposes of this program, the variable is read in as the floating point variable, `Next_CAD_Elevation`. This particular variable has not been implemented into the software development at this time, but again, could be used as an auxiliary variable in future development work.

Group Code 39: This is the entity's thickness if nonzero (fixed). This floating point variable is read in as `Next_Line_Thickness` and is also not specifically implemented into this version of the software development. It should also be considered a useful variable for future development.

Group Code 40: This floating point value is read in as the variable name, `Next_Arc_Radius`. This variable is used in the testing of a field travel path as well as in the many equations relating to the navigation of the tractor.

Group Code 50: This floating point value is read in as the variable name, `Next_Start_Angle`. This refers directly to the starting angle of an ARC drawing (field travel path) entity. The reference axis set for measuring the angles in this program is the compass direction EAST, which is the zero angle reference direction. All angles are measured counterclockwise from this reference direction.

Group Code 51: This floating point value is read as the variable name, `Next_End_Angle`. Again, this refers directly to the ending angle of an ARC

drawing (field travel path) entity. The same rules apply to this angle as for the starting angle described above.

Group Code 62: This value is the color number, when referred to in the context of a CADD drawing. This integer is read in as the variable, `Next_Color`. This is another variable scanned into the program from the DXF output file that is not directly used in this program. This variable could be used in a variety of ways to describe some aspect of the guidance and / or control system in future software developments.

Group Code 999: This is the code for a comment string. If a DXF output file is edited to include a 999 group code, the comment will be printed out on the screen as the file is being processed into a travel path data file.

During the entity parameter data acquisition, whenever any critical parameter of an entity is read into memory, the flag, `Entity_Flag` is set to one. This means that as soon as any variable is read in, other than a comment or a default code, the flag indicating whether or not all entity parameters have been scanned from the file is set to be true. This is checked only when the `Next_Group_Code` is equal to zero, otherwise the parameters continue to be read from the file until the entire set of entity parameters have been read.

The entity parameter acquisition is performed in a table-driven manner so that future development of the DXF system can be easily included into this program, if so desired. No major modifications would have to be made to acquire other types of drawing entities. The `Next_Group_Code` switch statement can be changed to include another case statement corresponding to the group code of the additional drawing entity that is to be acquired from the DXF output file.

Travel Path Data Tests

The function, `Check_Travel_Path` is used to check the validity and logic of a proposed field travel path. There are several basic considerations as to what constitutes a "valid" and "logical" file, with respect to this software development. A valid DXF output file would be required to conform to the following standards for this program:

The DXF output file must consist entirely of POINT(S) and / or , LINE(S) and / or ARC(S). No other drawing entity types are required in this program.

The file must be generated by selecting the travel path (drawing) entities in the order in which the user wants the machinery to be guided through the field.

The path must begin with a POINT entity; this is used as a reference point for the guidance program.

The travel path entities must be connected and tangent from one entity to the next. Connected, as defined in the TEST FIELD module of the program, means that the travel path entity endpoints, from one entity to the next one in sequence, are located within one centimeter of each other. Tangency conditions, as applied in this program module, also have a margin of error geometrically associated with the one centimeter tolerance of the program's global constant, `Dist_Tlrnc`.

Any ARCS used in the travel path are required to have a radius equal to, or greater than, the width of the implement minus an implement width tolerance (variable `Width_Tol`, equal to 0.05 meters), as set in the global constants of the program.

A description of how the entity interfaces are checked for logic and validity is given in the following part of this report:

The first proposed field travel path entity interface type checked in this section of the program is the POINT / LINE interface. If the X and Y coordinates of

a POINT entity are within the Dist_Tlrnc of either end of a LINE entity, the POINT / LINE interface is considered to be valid and logical and the user is notified, via the CRT screen display. The variable, Dist_Tlrnc which is the distance tolerance that was described in the GPSHEAD.H module section earlier in this report. If the POINT / LINE interface is not logical and valid, the program control is sent to the general error / warning handling routine, previously described in this paper.

The next test performed is the LINE / POINT interface. This is essentially the same as the POINT / LINE interface, except that the order of the entities is just the opposite. In other words, a LINE entity is the current travel path entity, while the POINT entity is the next travel path entity in the pair of proposed field travel path entities being checked.

The next check enacted is to determine whether a POINT / POINT interface has been selected by the user. This type of interface is totally invalid and illogical, so program control is always sent to the warning / error handling function.

The next proposed field travel path test is performed on a LINE / LINE interface. Two of the three auxiliary variables required in this section are called, Length_Line_A and Length_Line_B. These line lengths are the length of the current LINE and the next LINE in the paired set of proposed field travel path entities currently being analyzed. The third auxiliary variable is Length_Line_C, which is either the hypotenuse of a triangle consisting of line lengths A and B as the legs, or in the case of a "perfectly logical and valid" proposed field travel path segment, the length of line C would be the distance between the start and / or end points on lines A and B that are located farthest apart from each other. Each of these LINE lengths are computed using the function, Distance_Length. The starting and / or ending point X and Y coordinates of each of the lines, as

the case may be, are sent to this function. This function uses the distance formula between two points to determine the length of each line respectively.

The general formula for the distance between two points (in 2-d Cartesian plane):

$$\text{DISTANCE} = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

If the length of LINE A plus the length of LINE B is equal to the length of LINE C, within the distance tolerance previously described above, and one end or the other of LINE A is connected, within tolerance, to one end or the other of LINE B, the LINE / LINE interface is considered to be logical and valid. The same type of error handling as before is done when the above case is not true.

The next check on the proposed field travel path is to determine if a POINT / ARC interface is valid. The variables, Next_Arc_Startpoint_X_Coord, Next_Arc_Endpoint_X_Coord, Next_Arc_Startpoint_Y_Coord and Next_Arc_Endpoint_Y_Coord, are first computed. These coordinates are determined by the following formulas for each of the four above coordinates respectively:

$$\text{Next_Arc_Startpoint_X_Coord} = \text{Next_Primary_X_Coord} + \text{Next_Arc_Radius} \cdot \cos \left[\text{Next_Start_Angle} \cdot \left(\frac{2 \cdot \text{pi}}{360} \right) \right]$$

$$\text{Next_Arc_Endpoint_X_Coord} = \text{Next_Primary_X_Coord} + \text{Next_Arc_Radius} \cdot \cos \left[\text{Next_End_Angle} \cdot \left(\frac{2 \cdot \text{pi}}{360} \right) \right]$$

$$\text{Next_Arc_Startpoint_Y_Coord} = \text{Next_Primary_Y_Coord} + \text{Next_Arc_Radius} \cdot \sin \left[\text{Next_Start_Angle} \cdot \left(\frac{2 \cdot \text{pi}}{360} \right) \right]$$

$$\begin{aligned} \text{Next_Arc_Endpoint_Y_Coord} = \\ \text{Next_Primary_Y_Coord} + \text{Next_Arc_Radius} \cdot \sin \left[\text{Next_End_Angle} \cdot \left(\frac{2 \cdot \pi}{360} \right) \right] \end{aligned}$$

The next step is to determine if the coordinates of the POINT entity are within the tolerance distance of either the start or end point of the ARC entity and also if the radius of the next ARC entity is greater than or equal to the width of the implement minus the width tolerance. If the preceding conditions are met, the POINT / ARC interface passes the entity interface test.

The next check on the proposed field travel path is to determine if an ARC / POINT interface is valid. The variables, Crnt_Arc_Startpoint_X_Coord, Crnt_Arc_Endpoint_X_Coord, Crnt_Arc_Startpoint_Y_Coord and Crnt_Arc_Endpoint_Y_Coord, are first computed. These coordinates are determined by the following formulas for each of the four above coordinates respectively:

$$\begin{aligned} \text{Crnt_Arc_Startpoint_X_Coord} = \\ \text{Crnt_Primary_X_Coord} + \text{Crnt_Arc_Radius} \cdot \cos \left[\text{Crnt_Start_Angle} \cdot \left(\frac{2 \cdot \pi}{360} \right) \right] \end{aligned}$$

$$\begin{aligned} \text{Crnt_Arc_Endpoint_X_Coord} = \\ \text{Crnt_Primary_X_Coord} + \text{Crnt_Arc_Radius} \cdot \cos \left[\text{Crnt_End_Angle} \cdot \left(\frac{2 \cdot \pi}{360} \right) \right] \end{aligned}$$

$$\begin{aligned} \text{Crnt_Arc_Startpoint_Y_Coord} = \\ \text{Crnt_Primary_Y_Coord} + \text{Crnt_Arc_Radius} \cdot \sin \left[\text{Crnt_Start_Angle} \cdot \left(\frac{2 \cdot \pi}{360} \right) \right] \end{aligned}$$

$$\begin{aligned} \text{Crnt_Arc_Endpoint_Y_Coord} = \\ \text{Crnt_Primary_Y_Coord} + \text{Crnt_Arc_Radius} \cdot \sin \left[\text{Crnt_End_Angle} \cdot \left(\frac{2 \cdot \pi}{360} \right) \right] \end{aligned}$$

The next step is to determine if the coordinates of either the start or end point of the ARC entity are within the tolerance distance of the coordinates of the

POINT entity and also if the radius of the next ARC entity is greater than or equal to the width of the implement minus the width tolerance. If the preceding conditions are met, the ARC / POINT interface passes the entity interface test.

The next check on the validity and logic of a proposed field travel path is to test the ARC / ARC interface. This check requires all eight of the current and next ARC entity start and end point, X and Y coordinate calculations previously mentioned. All of these calculations are made by a call to the function, Calc_Arc_Start_End_points. Next, it is determined if the distance between the center point coordinates of the current and next ARC entities are equal, within the tolerance limits, to the summation or subtraction (or vice-versa as the case may be) of the current and next ARC entity radii. Also the current and next ARC radii must be greater than or equal to the working width of the implement minus the width tolerance on the implement. If the current and next ARC entities pass these checks, the ARC / ARC interface is valid and logic; if not, the error / warning routine handles the appropriate user notification.

The next proposed field travel path check that is done is the LINE / ARC entity interface. First, the length of the LINE is calculated. Then the next arc start and end point X and Y coordinates are calculated using the same procedure as previously described. Note that these ARC start and end point calculations have to be done here as well, since this entity check would be done with a different ARC entity than before. After this, the length of the line squared plus the square of the distance between the "end" of the line and the center of the ARC is checked to see if it is equal, within tolerances, to the square of the distance between the center of the ARC and the "start" of the LINE entity. The "start" and "end" of the LINE entity may or may not be the actual start and end of the

field travel path segment, depending upon how the user drew the LINE entity in the CADD program.

Next, the check to determine whether either the LINE entity's start or end point is connected to the ARC entity's start or end point is performed. Also the check to determine that the radius of the next ARC radius is greater than or equal to the width of the implement minus the width tolerance is performed. If all of the above checks are successful, the LINE / ARC interface is valid and logical.

The next proposed field travel path check that is done is the ARC / LINE entity interface. First, the length of the LINE is calculated. Then the current arc start and end point X and Y coordinates are calculated using the same procedure as previously described. After this, the length of the line squared plus the square of the distance between the "end" of the line and the center of the ARC is checked to see if it is equal, within tolerances, to the square of the distance between the center of the ARC and the "start" of the LINE entity.

Next the check to determine whether either the LINE entity's start or end point is connected to the ARC entity's start or end point is performed. Also the check to determine that the radius of the current ARC radius is greater than or equal to the width of the implement minus the width tolerance is performed. If all of the above checks are successful, the ARC / LINE interface is valid and logical.

This concludes all of the DXF output file proposed field travel path checks that are performed in this program. The logic and validity checks of this program are only basic checks; much more sophisticated checks could be made. These improvements, as well as many others, will be discussed later in this report.

The function `Distance_Length` is the next function in the module `TESTFIELD`. This function receives four parameters. These parameters must be specified as floating point parameters and the parameter order is to pass the first

X and Y coordinate pair then the second X and Y coordinate pair. This function simply uses the distance formula, mentioned earlier, to determine the distance between the two points that were passed to this function.

The next function is `Create_TPD_File`. This function creates a TPD field travel path file only if the DXF output file entities pass all of the required field travel path logic and validity checks previously described in detail. This function places a 999 group code at the start of a new file and the key word `TPD_TEST_PASS` on the second line of the TPD file. Next, the corresponding DXF output file is read into memory, a line at a time. Each line is then written to the TPD file just as it was read from the DXF output file. The only time any changes are made to a line, before being written to the TPD file, is if a `DXF_Group_Code` is equal to 5. If this is the case, the `TPD_Entity_Handle`, on the next line, is written in consecutive order, rather than the order that came from the DXF output file. The entity handle numbering order that comes from a DXF output file may or may not be consecutive and in order depending on how the user drew the proposed field travel path.

Once all of the lines from the DXF output file have been transferred to the TPD file, both files are closed and the user is notified that the proposed field travel path checks were successful. Next, the program control is sent to the function, `Plot_Field_Travel_Path`. A description of this function is given in the following section. When the user is finished visually analyzing the travel path, the program control is then returned to the main program menu.

Travel Path Display Module

The user can choose to graphically display a field travel path from the main menu. In this function the user is prompted to enter the scaling factor required to

graphically display the travel path. The scaling factor is used to scale the actual field coordinates to the monitor resolution; the scaling factor could also be used as a type of "zoom" feature. The entire travel path is then displayed on the monitor with the start point of the entire set of travel path entities being marked as well. When the user is finished analyzing the travel path display they can simply press the <SPACEBAR> to return to the main menu.

Farm Field Module

This module is the main module of the computer program. This function is where all of the inputs are processed and outputs are calculated and processed. The C code listing for this module is contained in appendix I. The first part of this module of the computer program is where the required variables are declared and initialized if needed. Next, the user is prompted to enter the file name of the travel path that they wish to use in a particular field to be farmed. The appropriate extension [.TPD], is automatically appended to the entered file name to eliminate any confusion or chance of error.

Next, the user is prompted to enter his / her choice as to whether he / she would like to create a new field status configuration file, if they have not already done so, or to farm the field using a previously defined field status configuration file. If the user needs to create a new field status configuration file, the program control is sent to the appropriate function to allow the user to do this, otherwise the user is prompted to enter the file name of the field status configuration file that he / she would like to use. Again, the appropriate extension [.FSC], is appended to the entered file name to prevent confusion or the error of trying to open the wrong type of file.

After these files have been opened, the user is then prompted to enter the dimension of the length of a side of the G.I.S. database cells that are to be used in this program. The units of measurement have to be consistent (the same) as the units of measurement used for the other dimensions in this program. Provisions can be made in the future so that any unit of measurement could be used for the different lengths and distances used in this program.

Next the user is prompted to enter the file name to be used for the input G.I.S. database. As usual, the appropriate file name extension [.GIS] is appended to the file name. Then, the user is prompted to enter the file name to be used for the output G.I.S. database. The file name extension [.GOS] is appended to the file name.

After this, the user is prompted to indicate if the travel path data, field configuration status, and the input and output G.I.S. files have been correctly specified. If the user indicates that they aren't, the user can input the correct file names; if the user indicates that the files have been correctly specified, program control continues on.

The next function called is `Init_GPS_Rcvr_Basepoint_Protocol`. This function is currently a "stub", (or null) function. A "stub" function is one that has not been developed, but is still included for future development. This function has not been developed, since there was no G.P.S. hardware available to the author with which to develop and test this program. A description of what this function should accomplish, when developed, is presented in the future program development section of this report.

The next function of the program is the `Info_Tractor_Impl_GPS_Var` function. This function prompts the user to enter the file name of the tractor configu-

ration file to be used for a particular field farming operation. The appropriate extension, [.TRC] is automatically appended to the user specified file name.

The user is then prompted to enter the file name of the implement and G.P.S. configuration file to be used for this particular farming operation. As always, the appropriate file name extension, in this case [.IMP], is automatically appended to the file name.

After the user has specified the tractor and implement configuration file names, each configuration file is opened and displayed on the screen. The user can then see the specified files to double check that all of the information in each file is correct. The user is then prompted to respond as to whether the tractor and implement configuration files are correct or if one or the other needs to be modified. Once the correct configurations have been specified, control returns to the main section of the program.

Next, the user is prompted to manually drive the tractor and machinery to within a preset distance of the specified travel path start point, in this case 1/2 meter. This preset distance is defined (in the G.P.S. header file) as the constant, `Travel_Path_Err_Tol`, and can be easily changed if another distance is required. The travel path data file is read by the program until the ENTITIES section is encountered. The travel path entities are then read in, one by one, until the starting entity is encountered. Next, the G.P.S. position is constantly updated, at the particular update interval, and the distance between the actual position of the machinery and the user specified start point is calculated. As soon as the machinery is within the preset distance (described earlier) of the start point, the user is informed, and control returns to the main section of this program.

Next, the end point of the previous travel path entity is set to the field travel path start point, as specified by the user. This is done so that future travel path

entities can be referenced correctly. The entities may or may not be followed in the same direction as they were originally drawn, so this reference point is necessary to keep track of this condition.

Once all of these initial steps have been taken, the main machinery travel path loop is started. This loop runs until the machinery has been run completely through the field, the machinery comes to the end of the user specified travel path or the user decides to quit farming the field travel path for some other reason. If the last condition is the reason, the program would have to be manually overridden, since there is no provision at this time for the user to quit the travel path loop until one of the first two conditions has been met.

Next, the function `Crnt_Pth_Sgmnt_Strt_End_Pts` is called. This function is used to determine what the current travel path entity's start and end points are, corresponding to the direction of travel, not the direction in which the entity was originally drawn (as explained previously).

If the `Crnt_TPD_Section_Name` (current travel path entity) is a POINT, the X and Y start and end points are set to the current, primary coordinates. If the current travel path entity is LINE and the next entity is an ARC, the start and end points of the LINE entity are determined, relative to the direction of travel. A similar method is used to determine the start and end points of an ARC if it is the current travel path entity and the next entity is a LINE. A similar approach is used if the current travel path entity is a LINE or an ARC and the next entity is a POINT. If there are two ARCS or two LINES in a row, calculations such as for the other possible combinations are used to determine what the current travel path entity start and end point coordinates are.

The above calculations for determining the start and end point coordinates for the different combinations of current and next travel path entities are simply to

determine which end of the current travel path entity is connected to one end of the next travel path entity. The end of the current travel path entity that is connected to an end of the next travel path entity is the current travel path entity end point, and the other end of the current travel path entity is then the start point.

After the current travel path entity start and end point coordinates have been determined, the function `Slope_Input_Data` is called. This function is a "stub" function (undeveloped or not fully developed function), which is described in the future program development section of this paper.

The next function called is `Slope_Corrected_Receiver_Location`. This function is used to calculate the tangential and longitudinal corrections to the G.P.S. receiver location when the implement is tilted due to unlevel / uneven ground. These corrections are then added to or subtracted from the distance that the G.P.S. receiver is located away from the center of the implement, as appropriate. The height of the G.P.S. receiver above the working plane of the implement is also calculated. See the future program development section for more information on this function. The formulas used for these corrections are as shown below:

$$\text{Tangent_Crrctn} = \text{GPS_Receiver_Height} * \sin(\text{Transverse_Mast_Angle})$$

$$\text{Lngtdnl_Crrctn} = \text{GPS_Receiver_Height} * \sin(\text{Longitudinal_Mast_Angle})$$

$$\begin{aligned} \text{GPS_Receiver_Height_Crrctn} &= \text{GPS_Receiver_Height} \\ &* \cos(\text{Transverse_Mast_Angle}) * \cos(\text{Longitudinal_Mast_Angle}) \end{aligned}$$

The `Steering_Angle_Input` function is called next. This function is described in the future program development section. Next, if the tractor hitch is of the swinging drawbar or three-point type, the variable `Hitch_Position` is acquired.

The hitch position comes from the function, `Hitch_Position_Input`. This function is currently a "stub" function and is described in the future program development section.

The next function called is `GPS_Position_Data`. This function is also a "stub" function at the present time and is described in the future program development section.

The next function called is `Travel_Path_Error`. This function is used to determine the value of the variable, `Travel_Path_Dist_Error` via the appropriate function `Point_to_Line_Error` or `Point_to_Arc_Error`. The variables `Path_Point_to_Line_Error` and `Path_Point_to_Arc_Error` come, respectively, from the previously mentioned functions. One of these values is assigned to the value for `Travel_Path_Dist_Error`, depending on whether the current travel path entity is a `LINE` or an `ARC`.

The function, `Path_Point_to_Line_Error` is used to calculate the perpendicular distance between the G.P.S. receiver position and the current travel path `LINE` entity. This calculation is accomplished by using the formulas for the shortest (perpendicular) distance between a `POINT` and `LINE` as follows (which formula is used depends on which direction the travel path is being traversed):

```
Path_Point_to_Line_Error =
(((Crnt_Secondary_X_Coord - X_Field_Coord) *
  (Crnt_Primary_Y_Coord - Crnt_Secondary_Y_Coord)) -
 ((Crnt_Secondary_Y_Coord - Y_Field_Coord) *
  (Crnt_Primary_X_Coord - Crnt_Secondary_X_Coord)))
 / Line_Length)
```

```

Path_Point_to_Line_Error =
((((Crnt_Primary_X_Coord - X_Field_Coord) *
  (Crnt_Secondary_Y_Coord - Crnt_Primary_Y_Coord)) -
  ((Crnt_Primary_Y_Coord - Y_Field_Coord) *
  (Crnt_Secondary_X_Coord - Crnt_Primary_X_Coord)))
/ Line_Length)

```

The function, Path_Point_to_Arc_Error is used to calculate the perpendicular distance between the G.P.S. receiver position and the current travel path ARC entity. This calculation is accomplished by calculating the distance from the center point of the ARC to the G.P.S. position and subtracting the arc radius from this number. The distance formula, as described earlier in this report is used to calculate the distance, while the radius of the ARC entity comes directly from the travel path data file. If this number is positive, the machine centerline is outside of the arc, whereas if this number is negative the machine centerline is inside of the arc. The direction of travel along the arc determines whether the tractor has to be steered left or right to bring the machinery back onto the target travel path. This section of the program not only computes the error distance, but also whether the machinery is located to the left or right of the travel path, depending on the direction of travel of the machinery along the ARC travel path entity.

The next section of the farm field program simply assigns a value of zero to both the look ahead steering angle and the left or right (of the travel path) reference number whenever the current and next travel path entities are both LINES. This is done since the two LINE entities must be connected and parallel to each other in order for the .DXF output file to become a valid travel path data .TPD file.

The next section of the program is the function, `Right_or_Left_of_Travel_Path`. This function is used to determine if the machinery is left or right of the target travel path, when traveling along a `LINE` path entity. This calculation is done for an `ARC` travel path entity in the `Point_to_Arc_Error` function, but must be done separately for a `LINE` travel path entity. The function used to determine if the machinery is left or right of the travel path uses a vector cross product method to calculate whether the center point of the implement is left or right of the target travel path. Using this method, the vector representing the `LINE` travel path entity is crossed with a vector which starts at the start point of the `LINE` travel path entity and ends at the G.P.S. receiver calculated local field coordinate. If this vector cross product is less than zero, the machinery is positioned to the right of the target travel path and the machinery must be steered to the left to correct for this, and vice-versa. The general form of a vector cross product formula is as follows:

$$\text{(2-dimensional) Vector Cross Product [A X B]} = A_x \cdot B_y - A_y \cdot B_x$$

Where:

- A_x is the X component of vector A
- A_y is the Y component of vector A
- B_x is the X component of vector B
- B_y is the Y component of vector B

The next section of the farm field module is the `Compute_Steering_Angle` function. This function is accessed only when the next travel path entity is an `ARC` and when the variable, `Get_Next_Entity` is (Boolean) true. This is where

the "look ahead" part of the program is implemented. The function `Compute_Steering_Angle` calculates what steering angle is required, for a particular tractor and implement configuration, to follow the ARC travel path entity. This steering angle depends on the steering type, hitch type, hitch location and hitch swing point (when applicable) of the tractor as well as the geometric dimensions of both the tractor and the implement.

These geometric dimensions for the tractor are the wheelbase, hitch length etc. as described in the tractor configuration section of this report. The dimensions for the implement are the working width and hitch length as described in the implement / G.P.S. configuration section of this report. Dimensions for the G.P.S. receiver configuration are the height of the G.P.S. receiver above the working plane of the implement as well as the direction and distance of the G.P.S. receiver away from the center of the working plane of the implement, as described in the implement / G.P.S receiver configuration section of this report.

Currently, the program is set up to calculate the steering angle for three *valid* tractor configurations out of 45 possible valid configurations (based on the types of tractor steering and hitch types and hitch locations described earlier in this report). The configurations currently calculated are for a tractor with front wheel steering, a rigid or swinging drawbar located at the rear, and with a swing point either ahead or behind the rear axle (when the tractor has a swinging drawbar attached). The basis for calculating these steering angles comes from steering formulas in the technical papers "Using the GPS Satellites for Precision Navigation" (W.E. Larsen, D. A. Tyler and G. A. Nielsen 1991) and "A Navigation Model using GPS Satellites" (W.E. Larsen and G. A. Nielsen 1991). The reader should refer to these papers to derive all of the steering angle formulas. Refer to appendix E for the formulas required for the three tractor configuration varia-

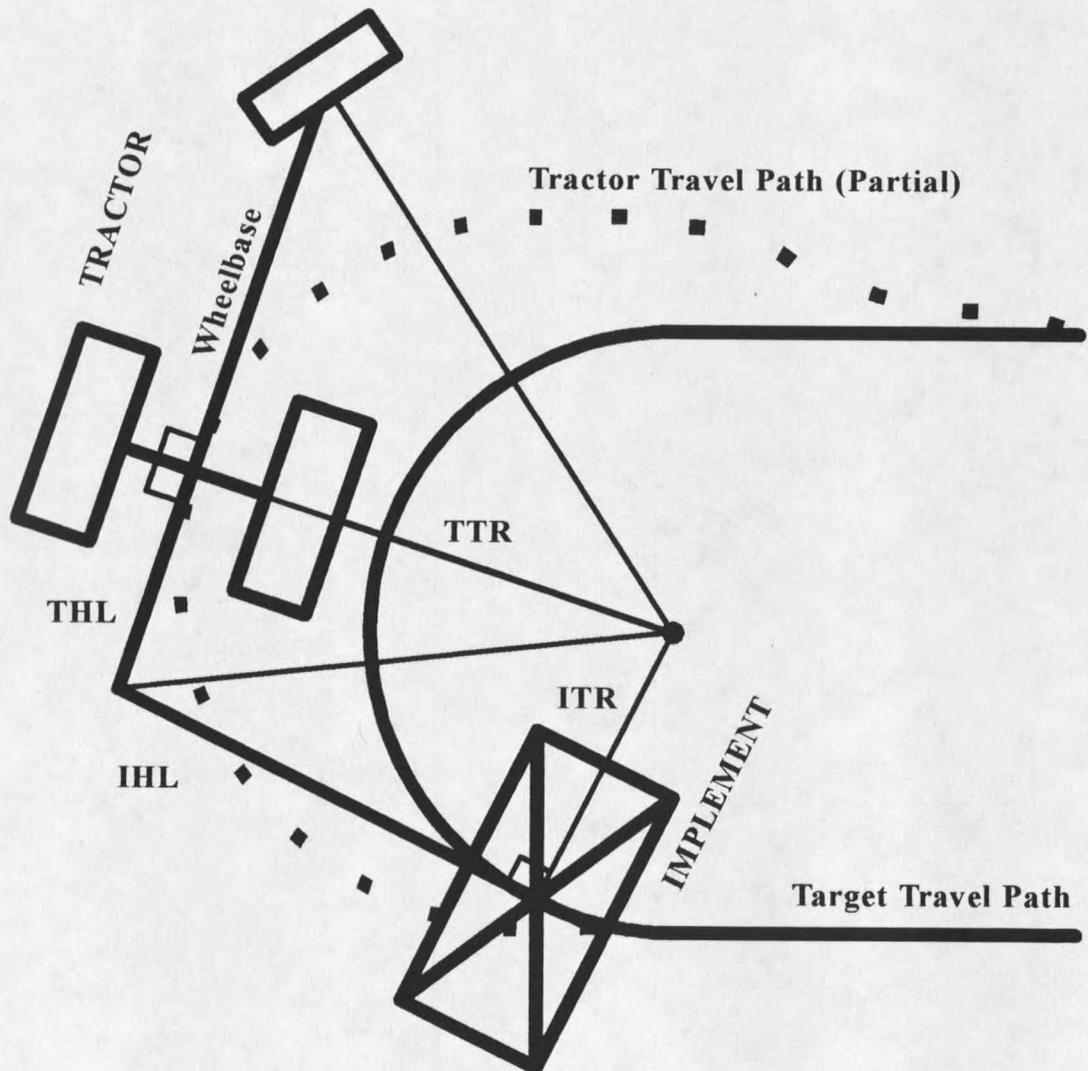
tions used in this program. The formula for calculating the “look ahead” steering angle for a front wheel steering tractor with a swinging drawbar located at the back of the tractor and the swing point ahead of the rear axle (with the drawbar in the freely swinging range of motion) is as follows:

$$\begin{aligned} \text{Look_Ahead_Steer_Angle} = & \\ & (\text{atan} ((\text{Tractor_Wheelbase} / \\ & ((\text{sqrt} ((\text{Implement_Hitch_Length} ^ 2) \\ & + (\text{Tractor_Drawbar_Length} ^ 2) \\ & + (\text{Implement_Turning_Radius} ^ 2) \\ & - (\text{Tractor_Hitch_Swing_Distance})))))) \end{aligned}$$

Now, recall that figures 1, 2, and 3 show the geometries of a front wheel steering tractor, a typical implement and its G.P.S. receiver positioning respectively. A graphical description of the combined tractor and implement steering geometry is shown in figure 7.

The next section of the farm field module is the function `Steer_Left_Right`. This function is used to determine which direction the tractor needs to be steered to follow the next travel path ARC entity. If the current travel path entity is a LINE, the function uses the vector cross product method in a manner similar to that previously described. This time, however, the vectors that are used are one from the start point of the current LINE entity to the end point of this entity, the other vector is from the start point of the ARC entity the end of that entity. When the first vector is crossed into the next vector, and the result is greater than zero, the tractor must be steered to the left to stay on the target travel path, and vice-versa (i.e. when the vector cross product is less than zero, the tractor must be steered to the right to stay on the target travel path).

FIGURE 7
TRACTOR / IMPLEMENT STEERING GEOMETRY
[TOP VIEW]



- THL = Tractor Hitch Length
TTR = Tractor Turning Radius
IHL = Implement Hitch Length
ITR = Implement Turning Radius

If the current travel path entity is an ARC, a different method of calculating whether to steer left or right is used. This method determines whether the current arc and the next arc are curving in the same direction or if they are not, which way the next arc curves relative to the current arc. Once it has been determined which way the next arc is curving, (left or right in the direction of travel relative to the current arc), the appropriate look ahead steering reference direction number is assigned to the variable, `Look_Ahead_Steer_Left_Right`. This number will be plus one (+1) when the tractor must be steered to the left, and minus one (-1) when the tractor must be steered to the right.

The next function in the farm field module is `Error_Corrected_Steering_Angle`. This function is used to correct the theoretical steering angle for travel path error and also for the "look ahead" component of the travel path steering angle computation. This function first calculates the correct travel path error, taking into account the tangential distance of the G.P.S. receiver from the center of the implement as well as the current travel path error. Next, this function calculates the rate of error change from the previous travel path error to the current travel path error. This is the last travel path error, algebraically subtracted from the current error, with this number then being divided by the time between G.P.S. position updates. Next, this function checks to see if the travel path error, the rate of error change and "look ahead" steering angles are "close" to zero. "Close" to zero is determined by comparison of these program variables to the constants `Small_Travel_Path_Error = 0.04`, `Small_Rate_of_Error_Change = 0.05` and `(Straight_Ahead = 0.0 + small = 0.01)`. These constants are all defined in the G.P.S. header file (except the variable `Straight_Ahead`, which is defined at the start of the farm field module) and can be changed to different values if desired. If these conditions are all true, the variable `Actual_Output_Steering_Angle`

is set to zero. Otherwise, the actual output steering angle is computed as follows. The rate of error change is multiplied by the constant `Correction_Factor_1`. The corrected travel path distance error is multiplied by `Correction_Factor_2`. The look ahead steering angle is multiplied `Correction_Factor_3` and also by the reference number, `Look_Ahead_Steer_Left_Right` (which indicates whether the implement is left or right of the target travel path). These terms are then added together and if this number is less than the maximum steering angle of the tractor, this is then the actual steering angle to be used to correct the tractor's travel. If this steering angle exceeds the maximum steering angle of the tractor, the maximum steering angle of the tractor is then assigned to the actual output steering angle. After these calculations have been made, the variables `Previous_Path_Error` and `Prev_Left_Right` are updated to the current values of these variables respectively. Other recommendations for further development of this function of the program are given in the future program development section of this paper.

The next function in the farm field module is `Steering_Output`. This function is currently a "stub" due to lack of hardware availability. Its purpose is outlined in the future development section of this paper.

The `GIS_Data_Input` function is the next part of the farm field module of this program. After this is the `GIS_Data_Output` function followed by the `Determine_Implement_Control` and `Implement_Control_Code` functions. Each of these functions are currently stub functions; the `GIS_Data_Input` function has been *partially* developed. A description of what each of these functions should theoretically accomplish is included in the future development section of this paper.

The final function of the farm field module's main travel path loop is `Check_Acquire_TPD_Entity`. This function is used to determine if another travel path data entity is required. This function is separated into two distinct sections, one is used to calculate the travel path distance remaining along a LINE entity, while the other section is used to calculate the travel path distance remaining along an ARC entity. Both sections of this function use variations on a vector dot product method to determine if another travel path entity is required. Each section is explained individually below, with an explanation of the remaining parts of this function following. The *general* form of a vector dot product is as follows:

(2-dimensional) Vector Dot Product [A * B] = Ax · Bx + Ay · By

Where:

- Ax is the X component of vector A
- Ay is the Y component of vector A
- Bx is the X component of vector B
- By is the Y component of vector B

The section dealing with the LINE travel path entity begins with computing the length of the line. Next, the X and Y direction components of the LINE entity vector are computed. Then, the X and Y vector components of a line from the G.P.S field position to the end of the LINE entity are computed. The vector dot product of the component of the second vector in the direction of the LINE entity vector is computed. Then, the distance that the G.P.S. receiver is located ahead or behind the center of the working plane of the implement is algebraically added to this distance. This length is then the distance of travel left before the end of the LINE travel path entity. This distance is then compared to the distance

that the tractor would travel at the current travel speed multiplied by the amount of time per G.P.S update interval. If the travel path distance remaining is less than the distance that the tractor would travel in the next update interval, another travel path entity would be acquired, otherwise the same travel path entity would be used for the next G.P.S. update time interval.

The next section of this function deals with calculating whether or not another travel path entity would be required before the end of an ARC travel path entity. The method used to calculate whether or not a new travel path entity is required is similar to the LINE travel path entity calculation, except that the remaining angle is used rather than the remaining distance along the travel path entity.

The first calculations are the X and Y components of the vector between the center and the end of the ARC entity. Next, the X and Y components of the vector between the center of the ARC travel path entity and the current G.P.S. local field position are calculated. The variable, `ARCCenter_to_GPS_Pt_Dist` is then corrected, when required, for the distance left or right of center of the working plane of the implement that the G.P.S. receiver is located. Next, the vector dot product of these two vectors is calculated. Then, the angle between these two vectors is computed. The angle that the tractor would travel through in the next G.P.S. update time interval, given the current ARC radius and travel speed, is calculated next. Finally, these two angles are compared, and if the tractor would travel through an angle greater than the angle left before the end of the ARC travel path entity, another travel path entity is acquired, otherwise the same travel path entity would be used for the next G.P.S. update time interval.

If the above calculations show that another travel path entity is required, the function `Obtain_Entity_Para_Info` (described previously) is called. If the end of

the travel path data file is encountered, or the user specified travel path end entity is acquired, the Boolean variable `Field_Travel_Done` is set to true. If a POINT travel path entity is encountered, it is ignored and another travel path entity is acquired. Possibilities for using the POINT entity, as well as other suggestions for improvement, are presented in the future development section of this paper.

Once the function, `Check_Acquire_TPD_Entity` is done, the main machinery travel path loop of the farm field module either continues or ends, depending on whether there are more travel path entities specified to be farmed or not. Once all of the user specified travel path entities have been covered, this module of the program is finished and the user is returned to the main menu of the program. The future development section of this paper contains some recommendations for future development of this travel path looping procedure.

An explanation and listing of one typical tractor configuration file is given in appendix J. An implement / G.P.S. receiver configuration file is shown in appendix K. Appendix L contains a field status configuration file. Refer to the appropriate sections dealing with the tractor, implement / G.P.S. receiver and field status configuration files for detailed explanations of how these configuration files are generally constructed. Finally, the travel path data file, for the field shown in figure 6, is presented in appendix M. Refer to the section, obtaining DXF entities, for a description of the group codes pertaining to a travel path file, or refer to a CADD book that deals with drawing interchange and file formats, such as an AutoCAD reference manual (Autodesk, Inc. 1989).

CHAPTER 4

PROGRAM STATUS AND FUTURE DEVELOPMENT

Computer Program Status

This computer program is a software development intended to be a framework for future development of an automatic guidance system for self-propelled machinery with attached implements. Following is a brief description of the program capabilities and limitations as well as other aspects of the computer program.

This computer program is not complete and therefore could not be used directly to control the steering system of a tractor. It does however, contain the basic program structure required to theoretically determine the required steering angle to correct for travel path error. This program also contains the code required to create, modify and view the configuration files required for the tractor, implement, G.P.S. receiver and field status. The user can also test a proposed travel path developed with a CADD program (as previously described). The user can also view the target travel path within this program. A description of what each of the "stub" (undeveloped or incomplete) functions in this computer program should accomplish, when developed, and other ideas for future program development follow.

Future Program Development

There are many specific and general ideas to improve this program. Following is a description of ideas for future computer program development and improvement.

When fully developed, the function `Init_GPS_Rcvr_Basepoint_Protocol` should send the field base point latitude and longitude of the particular field to be farmed to the computer that is used to process the G.P.S. signals. The G.P.S. receiver processor can then use this information to calculate the local field coordinates (as opposed to the global coordinates) required for use in this program. The G.P.S. receiver should also "lock" onto all available satellites, as well as other required procedures, before returning control to the main computer program.

The function `Slope_Input_Data` is the function of the program that will be used to acquire the variables `Transverse_Mast_Angle` and `Longitudinal_Mast_Angle`. These variables represent the slope of the G.P.S. receiver antenna mast with respect to the ground. These variables will be acquired using digital slope meters located perpendicular to each other on the antenna mast. One slope meter would be mounted parallel (or tangent) to the direction of implement travel, while the other slope meter would be mounted perpendicular to the direction of implement travel. These inputs can then be used later in the computer program to correct the G.P.S. receiver position, with respect to the working plane of the implement, when the implement is not operating on perfectly level ground. This is needed to change the antenna position as determined by the G.P.S. receiver, to a true position on the ground.

The G.P.S. receiver correction for the antenna height above the working plane of the implement is calculated. This correction assumes that the antenna is

mounted on a mast that can follow the contours of the ground. This can be accomplished by mounting the mast so that it will independently ride on top of a wheel and parallel arm assembly attached to the frame of the implement. If the mast is mounted solidly to the frame of the implement, another method of calculating the antenna height would be required. This is because as the implement is raised and lowered, the antenna height would change with the implement frame, rather than with the contour of the ground.

The `Steering_Angle_Input` function will be used to acquire the current steering angle of tractor. This angle will be calculated from the steering angle transducer. The right-hand rule is used to determine whether the tractor is steering left or right. Left turns are positive and right turns are a negative voltage signal. The type of transducer used and its placement on the tractor depend on the type of steering method used in the tractor.

The tractor hitch swing limit function, `Hitch_Position_Input`, is to acquire the position of the swinging drawbar or three-point hitch relative to the tractor. There should be two transducers, one mounted on each side of the hitch frame. The transducer switches should be mounted in such a way so that the switch is normally open, closing only when the hitch contacts the hitch stop. If the hitch is in its freely swinging range of motion, the transducer output should be the value of zero. If the hitch is against the left-hand or right-hand stop, the transducer outputs should have a value of minus and plus one respectively. These integer outputs can then be used later in the program when determining the steering angle required for the tractor and implement to follow the travel path through the field.

The function `GPS_Position_Data` is to be used to acquire the positioning and kinematic data for the machinery. This function will be used to acquire the cur-

rent X, Y and Z local field coordinates from the G.P.S. receiver computer, as well as the current machine speed. These variables are then used later in this computer program for travel path error computation and for use in the "look ahead" feature of the program. The "look ahead" feature is used to begin steering the machinery into position, before an ARC travel path entity is acquired, in a proactive rather than reactive manner.

The function `Error_Corrected_Steering_Angle` could be improved in the future to include more input variables in its calculations to provide even better control of the tractor. Right now this function uses the travel path error, rate of error change and a "look ahead" component of the steering angle to determine what the actual steering angle should be to correct the travel path of the tractor. Future development should also include the current travel speed of the tractor, to prevent the application of sharp steering angles when the tractor is moving at "high" speeds. This function should also take into account the fact that steering response to the input signal is not infinitely fast. Other factors such as possible steering angle slippage, especially in front wheel steer tractors without a powered front wheel drive system, could affect the steering response of the tractor to the steering signal. Some type of actual steering angle feedback should be implemented into the design to take these problem areas into account.

The function `Steering_Output` will be used to send the actual steering angle output to the mechanism(s) that actually would control the steering angle of the tractor. These control mechanisms would most likely consist of some type of electronically controlled servo or stepper motor attached to the steering shaft system, or a linear actuator attached to the steering valves to control the rate and angle of steering.

The GIS_Data_Input function currently is capable of determining the location of the current G.I.S. cell. This calculation is accomplished by dividing the local (field) X and Y coordinate positions of the machinery by the cell size of the G.I.S. grid, typecasting this number to an integer and adding one to the number. By adding one to the number, the lower left G.I.S. grid cell can then be referred to as G.I.S. cell coordinate (1,1) rather than (0,0).

In the future development of this function, the data required for use by the implement control code section of the program would have to be retrieved by the G.I.S. data input function. This data could include any or all of the following variables, as well as others not included in this discussion. These variables are, fertilizer application rate, fertilizer types, chemical types, ratios and application rates, seed types and planting rates and recommended implement depth. These variables would be predetermined for a particular G.I.S. grid cell, based on the analysis of a number of spatially and temporally variable field conditions. These parameters can then be used in the calculation of an implement control code, which will be discussed later in this section of the paper.

The function GIS_Data_Output should be developed to acquire and store the relevant field conditions encountered during a farming operation, for future analysis. These variables could be any or all of the following variables, and other variables determined to be useful in future development of this program. Variables such as soil moisture, type, depth, slope, pH, planting date, rain fall, crop variety, organic content, climatic condition, harvest yield, weed type and concentration, and other such variables determined to be significant (and measurable) should be acquired when possible and / or required. These parameters would then be used as the input to any number of analyses to determine the optimum rates

and types of chemicals, fertilizers (He and Peterson 1991), etc. to be applied to any particular cell of a field.

The G.I.S. output section of this program could also be a file containing the set of location points corresponding to the path actually taken through the field while using a target travel path developed for a theoretically flat field. This set of points could then be converted to a DXF file and input into a CADD program. These points could then be manipulated to form a modified travel path that would eliminate travel path errors due to the varied topography of the field. The user would have to take some measurements of the gaps left after the initial run through the field. The user could also run a simulation of the travel path through the field using the actual travel path data acquired on the first "test" run through the field to determine what adjustments would have to be made to the travel path to correct it to eliminate skips in the field.

By using this method rather than trying to adjust the original travel path file for the three-dimensional effects on the actual travel path, the "object oriented" field travel path originally developed can be easily overlaid onto any other field with the same dimensions. This would be a much more efficient method of creating travel path files for different fields. For example, this would be especially useful for strip crop farming. Most strips have the same basic dimensions for length and width, one travel path file could be produced for a "generic" field. This "prototype" field travel path could then be overlaid onto any strip simply by adjusting the field base point. Then, the travel path data file can be adjusted, as described above, for the particular field.

The function `Determine_Implement_Control`, when developed, will be used to calculate what the implement control code variables are, based on the information from the G.I.S data input function. This function will send its output vari-

ables to the function `Implement_Control_Code`, which would in turn send the actual implement control signals to the implement control system(s). These implement control systems could be any type of linear or rotary (or other) actuator required to adjust the rate of application, depth, attachment speed / feed or mixing action etc. of the particular implement being used.

Suggestions for using POINT entities in implement control are as follow. The POINT travel path entity would always be ignored for purposes of steering calculations, but could be used in the implement control as a way of signaling that the implement should be "toggled" between operation and non-operation. For example, whenever a POINT entity is encountered in a travel path file, the implement may be signalled to raise or lower to preset heights, or the implement may be turned on or off. The POINT entity could also be used for some other type of implement control signal relative to position on the target travel path, as opposed to the G.I.S. database. For example, the POINT entity may be used to signal that there is some obstruction and therefore the implement should be adjusted accordingly.

When the `Check_Acquire_TPD_Entity` function is called, there is no provision at this time to determine whether or not the operator of the tractor has suddenly stopped between the time that another travel path entity was acquired and when the tractor actually passed the end of the current travel path entity. This would need to be checked in a working version of the program, so that the machinery could follow the current travel path entity to its end *before* using the next travel path entity for path error calculations.

The main travel path loop of the farm field module of this program could be improved in several ways. First, the program only checks to see what travel path entities the user wishes to start and stop the field operation on. Provision is made

for the user to input the exact positions along the starting and ending travel path entities where the user would like the machinery to begin and quit farming. These points are not currently checked to see if they actually lie on the starting and ending travel path entities. This should be done in the future development of the program.

Currently, the user must let the tractor and machinery run from the starting entity to the ending entity without stopping the program. The user could stop the tractor along the travel path, but not exit the program if they needed to do so. A method of allowing the user to save the current machinery position along the travel path should be implemented to allow the user to stop farming the field and return to the same point where they left off earlier.

Other improvements that could be made to this program are as follow. This program could be expanded to compute the required steering angle for all of the 45 possible valid tractor steering and hitch types described in the tractor configuration section. Another improvement in this program would be to allow for real-time data acquisition and analyses of other operating parameters, such as tractor throttle and engine speed, drawbar pulling force and fuel consumption. These parameters could be analyzed and used to calculate adjustments to the tractor's speed via the transmission and throttle mechanisms. Other parameters could also be used to develop this program into a fully autonomous system. All tractor and implement functions could be analyzed and the appropriate action taken.

The entity parameters that are used in a target travel path can be modified for use in this program. For example, the line-type, color, and drawing layer parameters in a DXF output file are not currently used by this program. These parameters could be set to represent some type of desired control signal that is specific to the travel path rather than to the G.I.S. grid cells. One example of this would

be to use the line-type to represent whether or not an implement should be raised or lowered, or turned on or off, depending on what part of the travel path the machine was on. One example is that a solid line could be used to represent when the implement should be operating and a dashed or dotted line could be used to indicate when the implement should not be operating.

Another important point about the future development of this computer program also deals with the construction of the target travel path data. Most fields will not be perfectly flat. If this is the case, the travel paths that have been developed for a flat field, will cause the implement to leave gaps in the hilly sections of the field. This is because more implement widths would be required across hilly sections of a field compared to flat sections of a field. A solution to this problem would be to compensate for slope distance when drawing the target travel path. This will require precise topographical data to work with prior to development of the target travel path.

A major problem that must be dealt with in future program development is that of losing satellite "lock". This could be a serious problem, if the amount of time without a G.P.S. receiver update is "long"—more than a few seconds. Another problem is the possibility of the military denying access to the P-code, which is the code used to provide the more precise position readings necessary for use in this system.

The problem of losing satellite lock for "short" periods of time can be overcome. The solution would entail using the "dead reckoning" method of navigation. This method relies on calculation of where the machine would be located at a future time given the current position, steering geometry and speed.

The problem of losing the P-code satellite signal is more complicated, but there are G.P.S. receiver companies bringing out receivers that can use what is

known as the Y-code to provide nearly the same precision as the P-code without actually using the P-code.

This program should be written specifically for a G.U.I. (Graphical User Interface) operating environment. This would greatly increase the user friendliness and ease of use of this program. Writing the code in C++, i.e. object oriented code, would also greatly reduce the size of the source code and therefore the size of the executable program.

Conclusion

There are many complexities to the development of a G.P.S. based guidance system. A control system using a G.I.S. database adds to the complexity. The information inputs and outputs of this type of precision farming system could enable a whole new paradigm of farming. "Agriculture will leave the paradigm of uniformity and go into a small spaced variability (Aurenhammer and Muhr 1991)."

There are many advantages of using these technologies to navigate and control farm machinery: Greater efficiency comes from the operator not having to steer and control the implement, allowing more time to concentrate on other parts of the job. The operator should be able to do the same work with less fatigue, which also increases productivity. More timely application of the required processes will be possible and site specific application of the required processes will lead to less negative environmental impact, more knowledge of the land for future work (from the G.I.S. database) and greater returns for the farmer.

BIBLIOGRAPHY

Aurenhammer, H., and T. Muhr. 1991. GPS in a Basic Rule for Environment Protection in Agriculture. In Automated Agriculture for the 21st Century: Proceedings of the 1991 Symposium in Chicago, Illinois, 16-17 December 1991, by the American Society of Agricultural Engineers, pp 394-402.

AutoCAD Release 10 Reference Manual. 1989. Sausalito, California: Autodesk, Inc.

Beyer, W. H., ed. 1991. CRC Standard Mathematical Tables and Formulae. 29th edition. Boca Raton: Chemical Rubber Company Press.

Erbach, D. C., C. H. Choi, and K. Noh. 1991. Automated Guidance for Agricultural Tractors. In Automated Agriculture for the 21st Century: Proceedings of the 1991 Symposium in Chicago, Illinois, 16-17 December 1991, by the American Society of Agricultural Engineers, pp 182-191.

Faires, J. D., and B. T. Faires. 1983. Calculus and Analytic Geometry. Boston, Massachusetts: Prindle, Weber and Schmidt.

Gerstner, J. 1994. Zeroing in on Precision Farming. JD Journal, Volume 23, Number 1, pp 4-7.

He, B., and C. L. Peterson. 1991. A Comparison of Expert Systems and Simulation Techniques for Control of a Fertilizer Applicator. In Automated Agriculture for the 21st Century: Proceedings of the 1991 Symposium in Chicago, Illinois, 16-17 December 1991, by the American Society of Agricultural Engineers, pp 373-384.

Hurn, J. 1989. GPS A Guide to the Next Utility. Sunnyvale California: Trimble Navigation Ltd.

Larsen, W. E., and G. A. Nielsen. 1991. A Navigation Model using GPS Satellites, ASAE Paper No. PNW91-103.

Larsen, W. E., D. A. Tyler, and G. A. Nielsen. 1991. Using the GPS Satellites for Precision Navigation. In Automated Agriculture for the 21st Century: Proceedings of the 1991 Symposium in Chicago, Illinois, 16-17 December 1991, by the American Society of Agricultural Engineers, pp 201 -208.

Le Bars, J. M., and N. Bachotet. 1993. In-field Location of Agricultural Machinery. Evaluation of Three Techniques, In CEMAGREF, Groupment de Clermont-Ferrad (Montoldre), Domaine des Palaquins, Bulletin Technique du Machinisme et de l'Equipment Agricoles 1993. (No. 69): 13-24 (12 ref.) (Abstract in English).

Palmer, R. J. 1991. Progress Report of a Local Positioning System. In Automated Agriculture for the 21st Century: Proceedings of the 1991 Symposium in Chicago, Illinois, 16-17 December 1991, by the American Society of Agricultural Engineers, pp 403-408.

Scheuller, J.K. 1991. In-Field Site-Specific Crop Production. In Automated Agriculture for the 21st Century: Proceedings of the 1991 Symposium in Chicago, Illinois, 16-17 December 1991, by the American Society of Agricultural Engineers, pp 291-292.

Stafford, J. V., B. Ambler, and M. P. Smith. 1991. Sensing and Mapping Grain Yield Variation. In Automated Agriculture for the 21st Century: Proceedings of the 1991 Symposium in Chicago, Illinois, 16-17 December 1991, by the American Society of Agricultural Engineers, pp 356-365.

VDI / MEG Kolloquium Agrartechnik. 1992. (no. 14) Ortung und Navigation Landwirtschaftlicher Fahrzeuge. Position-Finding and Navigation of Agricultural Vehicles. Proceedings of a Conference Held in Weihenstephan, Germany, 5-6 March 1992. (Various article abstracts from this work)

Von Qualen, K. E., J. W. Hummel, and J. F. Reid. 1991. Machine Vision System for Field Sprayer Guidance. In Automated Agriculture for the 21st Century: Proceedings of the 1991 Symposium in Chicago, Illinois, 16-17 December 1991, by the American Society of Agricultural Engineers, pp 192-200.

APPENDICES

APPENDIX A

MAIN PROGRAM MODULE [GPS1.C]

```
/* Main Program Module [GPS1.C] */
```

```

/*BTM*****
/*      Brian T. Mosdal      */
/*      Global Positioning System of Satellites      */
/*      Automatic Guidance of an Agricultural Tractor      */
/*      & control of Implement using a      */
/*      Geographic Information System Database      */
/*      High Precision Farming      */
/*BTM*****
/* Start of the Main G.P.S. Program */
#include "gpsheade.h"

void main()
{
    /* Declare Variables used in the Program */
    Boolean done = FALSE;      /* check to end loop */
    char c;      /* input control character */

    int Norm_Bckgrnd_Color = BLACK;
    int Norm_Frgrnd_Color = WHITE;

    FILE *TPD_File_Pointer;
    char AUX_FILENAME [25];
    char File_Type;
    char File_Mode [5];
    char FILENAME [25];

    /* Set Display Colors */
    textbackground (Norm_Bckgrnd_Color);
    textcolor (Norm_Frgrnd_Color);
    clrscr ();      /* Clear the Screen */

    /* Main Program Sign on Message */

    printf("\n\n\n\n\n  G.P.S. TRACTOR & IMPLEMENT Control.\n");
    printf("  Global Positioning System of Satellites.\n");

```

```

printf("  Automatic Steering of an Agricultural Tractor.\n");
printf("  Computer control of the tractor and implement.\n");
printf("  High Precision Farming.\n");
printf("  Copyright 1994. \n\n\n\n\n\n\n\n\n\n\n");

```

```

Continue_Program (); /* Allows user to resume program */

```

```

do

```

```

{

```

```

  fflush(stdin);

```

```

  clrscr();

```

```

          /* Main Program Menu */

```

```

printf("\n\n\n");

```

```

printf("          Main Menu\n");

```

```

printf("          Global Positioning System - Automatic Steering Tractor\n");

```

```

printf("          High Precision Farming.\n\n");

```

```

printf(" [T] TRACTOR    [Create, modify or view the Tractor
          configuration]\n\n");

```

```

printf(" [I] IMPLEMENT  [Create, modify or view the Implement]\n");

```

```

printf("    / G.P.S.    [ / G.P.S. Receiver configuration]\n\n");

```

```

printf(" [C] FIELD STATUS [Create, modify or view the Field Status
          Info file]\n\n");

```

```

printf(" [S] FIELD TEST  [Test the Field Travel Path File]\n");

```

```

printf("          [created from a CADD DXFout Drawing]\n\n");

```

```

printf(" [P] TRAVEL PATH [Review a Travel Path Data File (GRAPHICAL
          OUTPUT)]\n\n");

```

```

printf(" [F] FARM FIELD  [Run the Tractor & Implement through the]\n");

```

```

printf("          [field according to the Field Travel Path File.]\n\n");

```

```

printf(" [Q] QUIT        [Exit the program]\n\n");

```

```

printf("          Enter the appropriate letter, then press <Return>.\n");

```

```

printf("          \n ");

```

```

  c = getc (stdin);

```

```
fflush (stdin);
c = toupper(c);
fflush (stdin);

switch (c)
{
    case 'T':
        Configure_Tractor ();
        break;

    case 'I':
        Configure_Implement_GPS ();
        break;

    case 'C':
        Configure_Field_Status_Info ();
        break;

    case 'S':
        Test_Field_Travel_Path ();
        break;

    case 'P':

        clrscr();
        printf("\n\n\n\n\n\n   Travel Path Data File required ");
        printf("is a TRAVEL PATH DATA output File.\n");
        printf("   The extension [.TPD] is automatically appended to
FILENAME.\n");

        File_Type = 'P';
        sprintf(File_Mode,"r");

        Open_File (&TPD_File_Pointer, AUX_FILENAME, File_Type,
                File_Mode, FILENAME);
```



```
normvideo ();    /* Reset Video Display */  
  
clrscr ();  
}  
  
/*BTM*****
```

APPENDIX B

G.P.S. HEADER MODULE [GPSHEADE.H]

```
/* G.P.S. Header Module [GPSHEADE.H] */
```

```
/*BTM*****
```

Brian T. Mosdal

G.P.S. Global Positioning System of Satellites
to Automatically Steer an Agricultural Tractor

```
/*BTM*****/
```

```
/* Include Header Files */
```

```
#ifndef header_incl
```

```
#include <graphics.h>
```

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
#include <math.h>
```

```
#include <stdlib.h>
```

```
#include <ctype.h>
```

```
#include <dos.h>
```

```
#include <string.h>
```

```
/* Type Defenition */
```

```
typedef enum { FALSE, TRUE } Boolean;
```

```
/* Function Prototypes */
```

```
/* Tractor */
```

```
void Configure_Tractor ();
```

```
void Create_Tractor_Configuration ();
```

```
void Modify_Tractor_Configuration (char *, char *, char *, char *,  
double *, double *, double *,  
double *, double *);
```

```
void Tractor_Configuration_Logic (char *, char *, char *, char *,  
double *, double *, double *,  
double *, double *);
```

```
void Check_Tractor_Configuration (char *, char *, char *, char *,  
double *, double *, double *,  
double *, double *);
```

```
void Tractor_File_Info (char, char, char, char, double, double,  
double, double, double);
```

```
void Steering_Type (char *);
```

```
void Hitch_Type (char *);
```

```
void Hitch_Swing_Point (char *);
```

```
void Hitch_Location (char *);
```

```
void Wheelbase (double *);
```

```
void Drawbar_Length (double *);
```

```
void Hitch_Swing_Distance (double *);
```

```
void Max_Hitch_Swing_Angle (double *);
```

```
void Max_Steering_Angle (double *);
```

```
void Open_Tractor_Config_File (char, char [], char *, char *, char *,  
char *, double *, double *, double *, double *, double *);
```

```
        /* Implement */
void Configure_Implement_GPS ();

void Create_Implement_GPS_Configuration ();

void Modify_Implement_GPS_Configuration (double *, double *,
        double *, double *, double *, char *, char *);

void Check_Implement_GPS_Configuration (double *, double *,
        double *, double *, double *, char *, char *);

void Implement_GPS_File_Info (double, double, double, double,
        double, char, char);

void Working_Width      (double *);

void Hitch_Length      (double *);

void Receiver_Height    (double *);

void Forward_Backward_Dist (double *);

void Left_Right_Dist    (double *);

void Forward_Backward_Dir (char *);

void Left_Right_Dir     (char *);

void Open_Implement_GPS_Config_File (char, char [], double *,
        double *, double *, double *, double *, char *, char *);

        /* Field Status Information */
void Configure_Field_Status_Info ();
```

```
void Open_Field_Status_Info (char, char [],  
                             double *, double *, double *, double *,  
                             double *, double *, double *, double *,  
                             double *, double *, int *, int *);
```

```
void Check_Field_Status_Info (double *, double *, double *, double *,  
                              double *, double *, double *, double *,  
                              double *, double *, int *, int *);
```

```
void Create_Field_Status_Info ();
```

```
void Field_Status_Info (double, double, double, double,  
                       double, double, double, double,  
                       double, double, int , int );
```

```
void Modify_Field_Status_Info (double *, double *, double *, double *,  
                               double *, double *, double *, double *,  
                               double *, double *, int *, int *);
```

```
void Degrees_Latitude (double *);
```

```
void Minutes_Latitude (double *);
```

```
void Seconds_Latitude (double *);
```

```
void Degrees_Longitude (double *);
```

```
void Minutes_Longitude (double *);
```

```
void Seconds_Longitude (double *);
```

```
void Startpoint_X (double *);
```

```
void Startpoint_Y (double *);
```

```
void Endpoint_X      (double *);

void Endpoint_Y      (double *);

void Travel_StartEntity (int *);

void Travel_EndEntity (int *);

        /* Test Field Travel Path */
void Test_Field_Travel_Path ();

void Obtain_Entity_Para_Info
(char [], char [], FILE **,
 char [], char [], float *,
 float *, int *, int *,
 float *, float *, float *,
 float *, float *, float *,
 float *, float *, float *);

void Check_Travel_Path
(char [], char [], char [],
 float, float, int, int,
 float, float, float,
 float, float, float,
 float, float, float,
 char [], char [], char [],
 float, float, int, int,
 float, float, float,
 float, float, float,
 float, float, float,
 float, int*);

float Distance_Length (float, float, float, float);

void Create_TPD_File (char []);
```

```
int Plot_Field_Travel_Path (FILE *TPD_File_Pointer, char FILENAME[]);
```

```
    /* Farm Field */
```

```
void Farm_Field ();
```

```
void Info_Tractor_Impl_GPS_Var (Boolean *, Boolean *, Boolean *,
    char *, char *, char *, char *,
    double *, double *, double *,
    double *, double *, double *, double *,
    double *, double *, double *,
    char *, char *);
```

```
void Init_GPS_Rcvr_Basepoint_Protocol (double, double, double,
    double, double, double);
```

```
void Info_Travel_Path_Field_GIS (FILE **, FILE **, FILE **,
    Boolean *, Boolean *, Boolean *,
    double *, double *, double *, double *,
    double *, double *, double *, double *,
    double *, double *, double *, int *, int *);
```

```
void New_Continue_Travel_Path (Boolean *, Boolean *, Boolean *,
    double *, double *, double *, double *,
    double *, double *, double *, double *,
    double *, double *, int *, int *);
```

```
void Initial_Equipment_Positioning (char [], char [], FILE *,
    char [], char [], float *,
    float *, int *, int *,
    float *, float *, float *,
    float *, float *, float *,
    float *, float *, float *,
    double , double , int );
```

```
void Crnt_Pth_Sgmnt_Strt_End_Pts
```

```
(char [], char [],  
float, float, float, float,  
float, float, float, float,  
float, float,  
float *, float *, float *,  
float *, float *, float *);
```

```
void Path_Start_Check (float *, long double, long double, double,  
double);
```

```
void GIS_Data_Input (long double, long double, double);
```

```
void GIS_Data_Output ();
```

```
void Check_Acquire_TPD_Entity
```

```
(Boolean *, char [], FILE *,  
char [], char [], float *,  
float *, int *, int *,  
float *, float *, float *,  
float *, float *, float *,  
float *, float *, float *,  
char [], char [],  
char [], char [],  
float *, float *, int *, int *,  
float *, float *, float *,  
float *, float *, float *,  
float *, float *, float *,  
float, float, float, float,  
long double, long double,  
long double, double,  
double, double,  
char, char, Boolean *, int);
```

```
void TPD_Data
```

```
(char [], char [], FILE **,  
char [], char [], float *,  
float *, int *, int *,  
float *, float *, float *,  
float *, float *, float *,  
float *, float *, float *);
```

```
void Slope_Input_Data (double *, double *);
```

```
void Slope_Corrected_Receiver_Location (double, double,  
double, double, double,  
char, char, double *,  
double *, double *);
```

```
void Steering_Angle_Input (double *);
```

```
void Hitch_Position_Input (int *);
```

```
void GPS_Position_Data (long double *, long double *,  
long double *, long double *);
```

```
void Travel_Path_Error (double *, double, double,  
int, char [25],  
float, float, float, float, float, float, float,  
int, char [25],  
float, float, float, float, float, float, float,  
long double, long double,  
long double, long double);
```

```
void Calc_Arc_Start_End_points (float, float, float, float, float,  
float, float, float, float, float,  
float *, float *, float *, float *,  
float *, float *, float *, float *);
```

```
void Point_to_Line_Error (double *, long double, long double,  
                          float, float, float, float,  
                          char [25], float, float, float,  
                          float, float, float, float);
```

```
void Point_to_Arc_Error  
(double *, long double, long double,  
 float, float, float,  
 float, float, float, float,  
 char [25], float, float, float,  
 float, float, float, float);
```

```
void Right_or_Left_of_Travel_Path  
(int *, long double, long double,  
 float *, float *, float *, float *);
```

```
void Steer_Left_Right  
(int *, char [], float, float,  
 float, float, float, float, float,  
 float, float, float, float, float,  
 float *, float *, float *, float *);
```

```
void Compute_Steering_Angle (long double *, char, char, char, char,  
                             double, double, double, double, double,  
                             double, float, int);
```

```
void Error_Corrected_Steering_Angle  
(double, int, int, double, double, double,  
 float, double, long double *, float);
```

```
void Steering_Output (long double);
```

```
void Determine_Implement_Control ();
```

```
void Implement_Control_Code (double, double, double);
```

```
    /* General */
```

```
void Double_Input (double *);
```

```
void Integer_Input (int *);
```

```
void Open_File (FILE **, char [], char, char [], char []);
```

```
void Continue_Program ();
```

```
void General_Error (int);
```

```
    /* Define Constants */
```

```
#define pi 3.14159265359
```

```
#define Standard_Time_Delay 2000
```

```
#define zero 0.0
```

```
#define small 0.01
```

```
#define Smll_Angle_Tol 0.05
```

```
#define Small_Travel_Path_Error 0.04
```

```
#define Small_Rate_of_Error_Change 0.05
```

```
    /* GPS Receiver Position/Location Tolerance */
```

```
    /* Due to Geometrical/Trigonometric Reasons */
```

```
#define GPS_Recvr_Loactn 1
```

```
    /* Travel Path Error Correction Factors */
```

```
#define Correction_Factor_1 0.75
```

```
#define Correction_Factor_2 0.25
```

```
#define Correction_Factor_3 0.95
```

```
    /* GPS update interval = 1/4 second */
```

```
#define GPS_Update_Interval 0.25
```

```
/* one centimeter err tolerance in the
   distance calculations used in the
   modules TEST FIELD and FARM FIELD */
#define Dist_Tlrnc 0.01

/* Travel Path Error Tolerance */
/* Maximum distance that implement can
   be from field travel path at startpoint of
   field travel */
#define Travel_Path_Err_Tol 0.5

/* five centimeters tolerance on travel path width
   to account for "weaving" of implement in field */
#define Width_Tol 0.05

/* The maximum implement width allowed for use in
   this program (to prevent invalid user input) */
#define Max_Impl_Width 50.0

/* The maximum implement hitch length allowed for
   use in this program (to prevent invalid user input) */
#define Max_Impl_Hitch_Length 25.0

/* GRAPHICS WINDOW and COLOR DEFENITIONS */
# define CLIPOFF 0
# define CLIPON 1

#define Black 0
#define Dark_Blue 1
#define Dark_Green 2
#define Aqua 3
#define Rust 4
#define Medium_Purple 5
#define Brown 6
#define Light_Gray 7
```

```
#define Dark_Gray 8
#define Med_Blue 9
#define Light_Green 10
#define Light_Blue 11
#define Light_Brown 12
#define Light_Purple 13
#define Yellow 14
#define White 15
```

```
#define header_incl 1
#endif
```

```
/* Variables used in the program (reference comment) */
```

```
/*
char  Tractor_Steering_Type;
char  Tractor_Hitch_Type;
char  Tractor_Hitch_Swing_Point;
char  Tractor_Hitch_Location;
double Tractor_Wheelbase;
double Tractor_Drawbar_Length;
double Tractor_Hitch_Swing_Distance;
double Tractor_Max_Hitch_Swing_Angle;
double Tractor_Max_Steering_Angle;
double Implement_Hitch_Length;
double Implement_Working_Width;
double GPS_Receiver_Height;
double GPS_Forward_Backward_Dist;
double GPS_Left_Right_Dist;
char  GPS_Forward_Backward_Dir;
char  GPS_Left_Right_Dir;
double GPS_Receiver_Lngtdnl_Crrctn;
double GPS_Receiver_Tangent_Crrctn;
double GPS_Receiver_Height_Crrctn;
double Transverse_Mast_Angle;
double Longitudinal_Mast_Angle;
```

```
double BasePoint_Degrees_Latitude;
double BasePoint_Minutes_Latitude;
double BasePoint_Seconds_Latitude;
double BasePoint_Degrees_Longitude;
double BasePoint_Minutes_Longitude;
double BasePoint_Seconds_Longitude;
double Startpoint_X_Coord;
double Startpoint_Y_Coord;
double Endpoint_X_Coord;
double Endpoint_Y_Coord;
int Field_Travel_StartEntity;
int Field_Travel_EndEntity;
char Crnt_TPD_Section_Name [25];
char Crnt_Layer_Name [25];
char Crnt_Linetype_Impl_UpDown [25];
float Crnt_CAD_Elevation;
float Crnt_Line_Thickness;
int Crnt_Entity_Handle;
int Crnt_Color;
float Crnt_Primary_X_Coord;
float Crnt_Primary_Y_Coord;
float Crnt_Primary_Z_Coord;
float Crnt_Secondary_X_Coord;
float Crnt_Secondary_Y_Coord;
float Crnt_Secondary_Z_Coord;
float Crnt_Arc_Radius;
float Crnt_Start_Angle;
float Crnt_End_Angle;
char TPD_Section_Name [25];
char Next_TPD_Section_Name [25];
char Next_Layer_Name [25];
char Next_Linetype_Impl_UpDown [25];
float Next_CAD_Elevation;
float Next_Line_Thickness;
int Next_Entity_Handle;
```

```
int Next_Color;
float Next_Primary_X_Coord;
float Next_Primary_Y_Coord;
float Next_Primary_Z_Coord;
float Next_Secondary_X_Coord;
float Next_Secondary_Y_Coord;
float Next_Secondary_Z_Coord;
float Next_Arc_Radius;
float Next_Start_Angle;
float Next_End_Angle;
float Crnt_Arc_Startpoint_X_Coord;
float Crnt_Arc_Endpoint_X_Coord;
float Crnt_Arc_Startpoint_Y_Coord;
float Crnt_Arc_Endpoint_Y_Coord;
float Next_Arc_Startpoint_X_Coord;
float Next_Arc_Endpoint_X_Coord;
float Next_Arc_Startpoint_Y_Coord;
float Next_Arc_Endpoint_Y_Coord;
double Steering_Angle;
long double Look_Ahead_Steer_Angle;
long double Actual_Output_Steering_Angle;
long double X_Field_Coord;
long double Y_Field_Coord;
long double Z_Field_Coord;
long double Travel_Speed;
double Depth;
double Application_Rate;
double Aux_Contrl;
double GIS_Grid_Cell_Size;
int Hitch_Position;
double Travel_Path_Dist_Error;
float Straight_Ahead;
float Previous_Path_Error;
float Crnt_X_Strt;
float Crnt_Y_Strt;
```

```
float Crnt_X_End;
float Crnt_Y_End;
float Previous_X_End;
float Previous_Y_End;
int Left_Right;
float Next_X_Strt;
float Next_X_End;
float Next_Y_Strt;
float Next_Y_End;
int Look_Ahead_Steer_Left_Right;
FILE TPD_File_Pointer;
FILE GOS_File_Pointer;
FILE GIS_File_Pointer;
Boolean Field_Travel_Done;
Boolean TRC_IMP_GPS_File_Done;
Boolean Travel_Path_File_Done;
Boolean Get_Next_Entity;
float Travel_Path_Start_Point_Error;
Boolean Initial_Position;
Boolean Field_Travel_Start;
int Next_Entity_Handle;
double Tangent_Crrctn;
double Lngtdnl_Crrctn;
double Path_Point_to_Line_Error;
double Path_Point_to_Arc_Error;
long double Line_Length;
float Crnt_Arc_Startpoint_X_Coord;
float Crnt_Arc_Startpoint_Y_Coord;
float Next_Arc_Startpoint_X_Coord;
float Next_Arc_Startpoint_Y_Coord;
float Crnt_Arc_Endpoint_X_Coord;
float Crnt_Arc_Endpoint_Y_Coord;
float Next_Arc_Endpoint_X_Coord;
float Next_Arc_Endpoint_Y_Coord;
float X_Coord_Line_Path_Vect;
```

```
float Y_Coord_Line_Path_Vect;
float X_Coord_Arc_Ref_Vect;
float Y_Coord_Arc_Ref_Vect;
float Vector_Cross_Prod;
int Error_Num;
char No_File;
double Implement_Turning_Radius;
double Crctd_Travel_Path_Dist_Error;
float Rate_of_Error_Change;
int Prev_Left_Right;
int X_GIS_Cell_Coord;
int Y_GIS_Cell_Coord;
char c;
char File_Type;
char File_Mode [5];
char AUX_FILENAME [25];
char FILENAME [25];
FILE Temp_TPD_File_Pointer;
FILE Temp_GIS_File_Pointer;
FILE Temp_GOS_File_Pointer;
double TPDist_Remaining;
float Line_Length;
float Field_Coord_to_EndPt_Dist;
int Entity_Scan_Pass_Number;
float X_LINE_Vect;
float Y_LINE_Vect;
float X_Field_to_EndPt_Vect;
float Y_Field_to_EndPt_Vect;
float Vector_Dot_Product;
float X_Coord_ARCenter_End_Vect;
float Y_Coord_ARCenter_End_Vect;
float X_Coord_ARCenter_GPS_Pt_Vect;
float Y_Coord_ARCenter_GPS_Pt_Vect;
float ARCenter_to_EndPt_Dist;
float ARCenter_to_GPS_Pt_Dist;
```

```
float Vector_Check_Angle;
float TPD_Angle_Remaining;
float X_Coord_Err_Dist_Vect;
float Y_Coord_Err_Dist_Vect;
int gdriver;
int gmode;
int gerr;
int Graphics_X_Max;
int Graphics_Y_Max;
float Field_Scale_Factor;
float Radius_Scale_Factor;
Boolean Travel_Path_Done;
float X_ARC_Vect;
float Y_ARC_Vect;
float ARC_Vect_Len;
float Unit_X_Vect;
float Unit_Vect_Len;
float Vector_Angle;
double Tractor_Hitch_Ref_Dist;
double Tractor_Swing_Ref_Dist;
*/
```

```
/*BTM******/
```

APPENDIX C

GENERAL INPUT MODULE [GENINPUT.C]

```
/* General Input Module [GENINPUT.C] */
```

```
#include "gpsheade.h"
```

```
/*BTM*****
```

```
/* Open File */
```

```
void Open_File (FILE **File_Pointer, char AUX_FILENAME [],  
                char File_Type, char File_Mode [], char FILENAME [])
```

```
{
```

```
    char FNAME [9];
```

```
    Boolean done = FALSE;
```

```
do
```

```
{
```

```
    switch (File_Type)
```

```
    {
```

```
        case 'T':
```

```
            scanf("%8s",FNAME);
```

```
            sprintf(AUX_FILENAME,"%s.TRC",FNAME);
```

```
            break;
```

```
        case 'I':
```

```
            scanf("%8s",FNAME);
```

```
            sprintf(AUX_FILENAME,"%s.IMP",FNAME);
```

```
            break;
```

```
        case 'D':
```

```
            scanf("%8s",FNAME);
```

```
            sprintf(AUX_FILENAME,"%s.DXF",FNAME);
```

```
            strcpy(FILENAME, FNAME);
```

```
            break;
```

```
        case 'F':
```

```
            sprintf(AUX_FILENAME,"%s.DXF",FILENAME);
```

```
            break;
```

```
case 'G':
    sprintf(AUX_FILENAME,"%s.TPD",FILENAME);
    break;

case 'N':
    scanf("%8s",FNAME);
    sprintf(AUX_FILENAME,"%s.GIS",FNAME);
    strcpy(FILENAME, FNAME);
    break;

case 'O':
    scanf("%8s",FNAME);
    sprintf(AUX_FILENAME,"%s.GOS",FNAME);
    strcpy(FILENAME, FNAME);
    break;

case 'P':
    scanf("%8s",FNAME);
    sprintf(AUX_FILENAME,"%s.TPD",FNAME);
    strcpy(FILENAME, FNAME);
    break;

case 'S':
    scanf("%8s",FNAME);
    sprintf(AUX_FILENAME,"%s.FSC",FNAME);
    break;

default:
    printf("ERROR in FILENAME");
    break;
}

if ((*File_Pointer = fopen (AUX_FILENAME,File_Mode)) == NULL)
{
```

```

printf("\n\n\n\n\n\n ATTENTION!\n");
printf(" The file %s could not be opened.\n\n",AUX_FILENAME);
printf("\a");
delay (Standard_Time_Delay/2);
clrscr ();
printf("\n\n\n\n\n\n Enter the correct filename.\n\n\n");
printf(" \n");
done = FALSE;
/*normvideo ();
exit (1);*/
}
else (done = TRUE);

} while (done == FALSE);

printf("\n The Filename is : %s\n",AUX_FILENAME);

Continue_Program ();

}

/*BTM*****
/* Continue Program */
void Continue_Program ()
{
char c;
Boolean check = FALSE;

do
{
printf("\n Press the <SpaceBar> to continue program.\n ");

fflush (stdin);
c = getch ();
fflush (stdin);

```

```

        if (c==' ')
            check = TRUE;
        else
            printf("\a");
    } while (check != TRUE);
}

/*BTM*****
/* Double Precision Floating Point Number Input */
void Double_Input (double *number)
{
    Boolean done, check;
    char c;
    char buffer[128];
    float min_num = 0.0;

    done = FALSE;
    check = FALSE;

    do
    {
        fflush (stdin);
        gets (buffer);
        *number = atof (buffer);

        if (((*number) <= min_num) ||
            ((*number) == HUGE_VAL))
        {
            do
            {
                printf("\a");
                printf("  Value entered was not a positive number\n");
                printf("  or it was an invalid number.\n\n");
            }
        }
    }
}

```

```

printf("  Check input value and reenter the number.\n");
printf("  Press the <SpaceBar> to continue.\n\n");

fflush (stdin);
c = getch ();
fflush (stdin);

    if (c==' ')
        {
            printf("Enter a valid number.\n");
            check = TRUE;
        }
    else
        printf("\a");
} while (check != TRUE);

}
else
    done = TRUE;

} while (done != TRUE);
}

/*BTM*****
/* Integer Number Input */
void Integer_Input (int *number)
{
    Boolean done, check;
    char c;
    char buffer[128];
    int min_num = 0;

    done = FALSE;
    check = FALSE;

```

```

do
{
    fflush (stdin);
    gets (buffer);
    *number = atof (buffer);

    if (((*number) <= min_num) ||
        ((*number) == HUGE_VAL))
    {
        do
        {
            printf("\a");
            printf("  Value entered was not a positive number\n");
            printf("  or it was an invalid number.\n\n");
            printf("  Check input value and reenter the number.\n");
            printf("  Press the <SpaceBar> to continue.\n\n");

            fflush (stdin);
            c = getch ();
            fflush (stdin);
            if (c==' ')
            {
                printf("Enter a valid number.\n");
                check = TRUE;
            }
            else
                printf("\a");
        } while (check != TRUE);
    }
    else
        done = TRUE;

} while (done != TRUE);
}
/*BTM***** */

```

APPENDIX D

GENERAL ERROR MODULE [GENERROR.C]


```
break;
```

```
case 101:
```

```
    printf(" The DXF output file is null.\n");  
    printf(" No ENTITIES Section was detected.\n");  
    break;
```

```
case 102:
```

```
    printf(" The DXF output file has no data.\n");  
    printf(" Check file to be sure it contains data.\n");  
    break;
```

```
case 103:
```

```
    printf(" Invalid DXF output file.\n");  
    printf(" The DXF output file section ENTITIES\n");  
    printf(" must have a Starting entity of → POINT.\n");  
    break;
```

```
case 104:
```

```
    printf(" Invalid DXF output file.\n");  
    printf(" There is only a Start POINT in the file.\n");  
    printf(" The DXF file must contain at least one\n");  
    printf(" line or arc as a travel path.\n");  
    break;
```

```
case 105:
```

```
    printf(" An invalid ENTITY Type has been detected.\n");  
    printf(" Valid Entity Types consist of only\n");  
    printf(" LINES, ARCS, or POINTS.\n");  
    break;
```

```
case 106:
```

```
    printf(" \n");  
    break;
```

```
case 200:
    printf(" A Point / Line interface is invalidly constructed. \n");
    break;

case 201:
    printf(" A Line / Point interface is invalidly constructed. \n");
    break;

case 202:
    printf(" Two Points have been selected consecutively. \n");
    break;

case 203:
    printf(" A Line / Line interface is invalidly constructed. \n");
    break;

case 204:
    printf(" A Point / Arc interface is invalidly constructed. \n");
    break;

case 205:
    printf(" An Arc / Point interface is invalidly constructed. \n");
    break;

case 206:
    printf(" An Arc / Arc interface is invalidly constructed. \n");
    break;

case 207:
    printf(" A Line / Arc interface is invalidly constructed. \n");
    break;

case 208:
    printf(" An Arc / Line interface is invalidly constructed. \n");
```

```

        break;

    case 209:
        printf(" \n");
        break;

    case 300:
        printf("\n There is a serious ERROR with the\n");
        printf(" TRACTOR configuration (logic).\n");
        break;

    default:
        printf("\n\n\n\n\n\n\a ERROR!\n");
        break;
}

Continue_Program ();

if (Warning_Error_Flag == 0)
    exit(1);
}

/*BTM*****
/* Check the Tractor configuration setup to determine if it is logical */
void Tractor_Configuration_Logic
(char *Tractor_Steering_Type,
char *Tractor_Hitch_Type,
char *Tractor_Hitch_Swing_Point,
char *Tractor_Hitch_Location,
double *Tractor_Wheelbase,
double *Tractor_Drawbar_Length,

```

```

double *Tractor_Hitch_Swing_Distance,
double *Tractor_Max_Hitch_Swing_Angle,
double *Tractor_Max_Steering_Angle)
{
Boolean Modify_Flag = FALSE;

do
{

switch (*Tractor_Steering_Type)
{
case 'F':

switch (*Tractor_Hitch_Type)
{
case 'S':

switch (*Tractor_Hitch_Location)
{
case 'F':
{
clrscr ();
printf("\n\n\n\n\n\n ATTENTION!\n\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" There can not be a SWINGING DRAWBAR
located\n");
printf(" at the FRONT of the tractor.\n");
printf(" Correct the Tractor Configuration File and
proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'C':
{

```

```

clrscr ();
printf("\n\n\n\n\n ATTENTION!\n");
printf("  The TRACTOR configuration is incorrect.\n");
printf("  There can not be a SWINGING DRAWBAR
located\n");
printf("  at the CENTER of the tractor.\n");
printf("  Correct the Tractor Configuration File and
proceed.\n");
Modify_Flag = TRUE;
}
break;

```

```

case 'R':

```

```

switch (*Tractor_Hitch_Swing_Point)
{
case 'A':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!\n");
printf("  The TRACTOR configuration is incorrect.\n");
printf("  The drawbar SWING POINT can not be
located \n");
printf("  AHEAD of the FRONT AXLE on the tractor.\n");
printf("  Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;

```

```

case 'B':
Modify_Flag = FALSE;
break;

```

```

case 'C':

```

```
{
  clrscr ();
  printf("\n\n\n\n\n ATTENTION!:\n");
  printf(" The TRACTOR configuration is
    incorrect.\n");
  printf(" The drawbar SWING POINT can not be
    located \n");
  printf(" BEHIND the FRONT AXLE on the tractor.\n");
  printf(" Correct the Tractor Configuration File
    and proceed.\n");
  Modify_Flag = TRUE;
}
break;

case 'D':
  Modify_Flag = FALSE;
  break;

case 'N':
  {
  clrscr ();
  printf("\n\n\n\n\n ATTENTION!:\n");
  printf(" The TRACTOR configuration is incorrect.\n");
  printf(" The drawbar SWING POINT must be
    provided.\n");
  printf(" Correct the Tractor Configuration File
    and proceed.\n");
  Modify_Flag = TRUE;
  }
  break;

default:
  {
  clrscr ();
```

```
printf("\n\n\n\n\n\n  ATTENTION!\n");
printf("  The TRACTOR configuration is incorrect.\n");
printf("  Check the .TRC file for the correct\n");
printf("  Tractor_Hitch_Swing_Point.\n");
printf("  Correct the Tractor Configuration File
        and proceed.\n");
Modify_Flag = TRUE;
}
break;

}
break;

}
break;

case 'R':

switch (*Tractor_Hitch_Location)
{
case 'F':
{
clrscr ();
printf("\n\n\n\n\n\n  ATTENTION!\n");
printf("  The TRACTOR configuration is incorrect.\n");
printf("  This program is not compatible with a \n");
printf("  RIGID DRAWBAR mounted on the FRONT of the\n");
printf("  tractor.\n");
printf("  Correct the Tractor Configuration File and
        proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'C':
```

```
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" This program is not compatible with a \n");
printf(" RIGID DRAWBAR mounted on the CENTER of the\n");
printf(" tractor.\n");
printf(" Correct the Tractor Configuration File and
        proceed.\n");
Modify_Flag = TRUE;
}
break;
```

```
case 'R':
```

```
switch (*Tractor_Hitch_Swing_Point)
```

```
{
```

```
case 'A':
```

```
{
```

```
clrscr ();
```

```
printf("\n\n\n\n\n ATTENTION!:\n");
```

```
printf(" The TRACTOR configuration is incorrect.\n");
```

```
printf(" The drawbar basepoint can not be
        located \n");
```

```
printf(" AHEAD the FRONT AXLE on the tractor.\n");
```

```
printf(" Correct the Tractor Configuration File
        and proceed.\n");
```

```
Modify_Flag = TRUE;
```

```
}
```

```
break;
```

```
case 'B':
```

```
{
```

```
*Tractor_Hitch_Swing_Point = 'N';
```

```
*Tractor_Hitch_Swing_Distance = 0.0;
```

```
*Tractor_Max_Hitch_Swing_Angle = 0.0;
Modify_Flag = FALSE;
}
break;

case 'C':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The drawbar basepoint can not be
located \n");
printf(" BEHIND the FRONT AXLE on the tractor.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'D':
{
*Tractor_Hitch_Swing_Point = 'N';
*Tractor_Hitch_Swing_Distance = 0.0;
*Tractor_Max_Hitch_Swing_Angle = 0.0;
Modify_Flag = FALSE;
}
break;

case 'N':
{
*Tractor_Hitch_Swing_Distance = 0.0;
*Tractor_Max_Hitch_Swing_Angle = 0.0;
Modify_Flag = FALSE;
}
break;
```

```
default:
    {
        clrscr ();
        printf("\n\n\n\n\n ATTENTION!:\n");
        printf(" The TRACTOR configuration is incorrect.\n");
        printf(" Check the .TRC file for the correct\n");
        printf(" Tractor_Hitch_Swing_Point.\n");
        printf(" Correct the Tractor Configuration File
            and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

}
break;

}
break;

case 'T':

switch (*Tractor_Hitch_Location)
{
    case 'F':

        switch (*Tractor_Hitch_Swing_Point)
        {
            case 'A':
                {
                    clrscr ();
                    printf("\n\n\n\n\n ATTENTION!:\n");
                    printf(" The TRACTOR configuration is incorrect.\n");
```

```
printf(" The drawbar SWING POINT should not
      be located \n");
printf(" AHEAD the FRONT AXLE on the tractor.\n");
printf(" Correct the Tractor Configuration File
      and proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'B':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The drawbar SWING POINT should not
      be located \n");
printf(" BEHIND the REAR AXLE on the tractor.\n");
printf(" Correct the Tractor Configuration File
      and proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'C':
Modify_Flag = FALSE;
break;

case 'D':
Modify_Flag = FALSE;
break;

case 'N':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
```

```
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The drawbar SWING POINT must be
provided.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;

default:
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" Check the .TRC file for the correct\n");
printf(" Tractor_Hitch_Swing_Point.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;
}
break;

case 'C':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The can not be a THREE-POINT HITCH located\n");
printf(" at the CENTER of the tractor.\n");
printf(" Correct the Tractor Configuration File and
proceed.\n");
Modify_Flag = TRUE;
}
```

```
break;
```

```
case 'R':
```

```
switch (*Tractor_Hitch_Swing_Point)
```

```
{
```

```
case 'A':
```

```
{
```

```
clrscr ();
```

```
printf("\n\n\n\n\n ATTENTION!:\n");
```

```
printf(" The TRACTOR configuration is incorrect.\n");
```

```
printf(" The drawbar SWINGPOINT should not  
be located\n");
```

```
printf(" AHEAD of the FRONT AXLE of the tractor.\n");
```

```
printf(" Correct the Tractor Configuration File  
and proceed.\n");
```

```
Modify_Flag = TRUE;
```

```
}
```

```
break;
```

```
case 'B':
```

```
{
```

```
clrscr ();
```

```
printf("\n\n\n\n\n ATTENTION!:\n");
```

```
printf(" The TRACTOR configuration is incorrect.\n");
```

```
printf(" The drawbar SWINGPOINT should not  
be located\n");
```

```
printf(" BEHIND the REAR AXLE of the tractor.\n");
```

```
printf(" Correct the Tractor Configuration File  
and proceed.\n");
```

```
Modify_Flag = TRUE;
```

```
}
```

```
break;
```

```
case 'C':
```

```
Modify_Flag = FALSE;
break;

case 'D':
    Modify_Flag = FALSE;
    break;

case 'N':
    {
        clrscr ();
        printf("\n\n\n\n\n ATTENTION!:\n");
        printf(" The TRACTOR configuration is incorrect.\n");
        printf(" The drawbar SWING POINT must be
            provided.\n");
        printf(" Correct the Tractor Configuration File
            and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

default:
    {
        clrscr ();
        printf("\n\n\n\n\n ATTENTION!:\n");
        printf(" The TRACTOR configuration is incorrect.\n");
        printf(" Check the .TRC file for the correct\n");
        printf(" Tractor_Hitch_Swing_Point.\n");
        printf(" Correct the Tractor Configuration File
            and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;
}
break;
```

```
    }  
    break;  
  
case 'I':  
  
    switch (*Tractor_Hitch_Location)  
    {  
        case 'F':  
        case 'C':  
        case 'R':  
  
            switch (*Tractor_Hitch_Swing_Point)  
            {  
                case 'A':  
                case 'B':  
                case 'C':  
                case 'D':  
                    {  
                        *Tractor_Hitch_Swing_Point = 'N';  
                        *Tractor_Hitch_Swing_Distance = 0.0;  
                        *Tractor_Max_Hitch_Swing_Angle = 0.0;  
                        Modify_Flag = FALSE;  
                    }  
                break;  
            }  
  
        default:  
            {  
                clrscr ();  
                printf("\n\n\n\n\n\n  ATTENTION!:\n");  
                printf("  The TRACTOR configuration is incorrect.\n");  
                printf("  Check the .TRC file for the correct\n");  
                printf("  Tractor_Hitch_Swing_Point.\n");  
                printf("  Correct the Tractor Configuration File  
                    and proceed.\n");  
                Modify_Flag = TRUE;  
            }  
        }  
    }  
}
```

```
        }
        break;

    }
    break;

}
break;

case 'A':

switch (*Tractor_Hitch_Type)
{
    case 'S':

        switch (*Tractor_Hitch_Location)
        {
            case 'F':
                {
                    clrscr ();
                    printf("\n\n\n\n\n ATTENTION!\n");
                    printf(" The TRACTOR configuration is incorrect.\n");
                    printf(" There can not be a SWINGING DRAWBAR
                        located\n");
                    printf(" at the FRONT of the tractor.\n");
                    printf(" Correct the Tractor Configuration File and
                        proceed.\n");
                    Modify_Flag = TRUE;
                }
                break;

            case 'C':
```

```
{
clrscr ();
printf("\n\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" There can not be a SWINGING DRAWBAR
located\n");
printf(" at the CENTER of the tractor.\n");
printf(" Correct the Tractor Configuration File and
proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'R':

switch (*Tractor_Hitch_Swing_Point)
{
case 'A':
{
clrscr ();
printf("\n\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The drawbar SWING POINT can not be
located \n");
printf(" AHEAD of the FRONT AXLE on the
tractor.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'B':
Modify_Flag = FALSE;
break;
```

```
case 'C':
    {
        clrscr ();
        printf("\n\n\n\n\n\n  ATTENTION!:\n");
        printf("  The TRACTOR configuration is incorrect.\n");
        printf("  The drawbar SWING POINT can not be
            located \n");
        printf("  BEHIND the FRONT AXLE on the tractor.\n");
        printf("  Correct the Tractor Configuration File
            and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

case 'D':
    Modify_Flag = FALSE;
    break;

case 'N':
    {
        clrscr ();
        printf("\n\n\n\n\n\n  ATTENTION!:\n");
        printf("  The TRACTOR configuration is incorrect.\n");
        printf("  The drawbar SWING POINT must be
            provided.\n");
        printf("  Correct the Tractor Configuration File
            and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

default:
    {
        clrscr ();
```

```

        printf("\n\n\n\n\n\n  ATTENTION!\n");
        printf("  The TRACTOR configuration is incorrect.\n");
        printf("  Check the .TRC file for the correct\n");
        printf("  Tractor_Hitch_Swing_Point.\n");
        printf("  Correct the Tractor Configuration File
                and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

}
break;

}
break;

case 'R':

    switch (*Tractor_Hitch_Location)
    {
        case 'F':
            {
                clrscr ();
                printf("\n\n\n\n\n\n  ATTENTION!\n");
                printf("  The TRACTOR configuration is incorrect.\n");
                printf("  This program is not compatible with a \n");
                printf("  RIGID DRAWBAR mounted on the FRONT of the\n");
                printf("  tractor.\n");
                printf("  Correct the Tractor Configuration File and
                        proceed.\n");
                Modify_Flag = TRUE;
            }
            break;

        case 'C':

```

```

{
clrscr ();
printf("\n\n\n\n\n\n  ATTENTION!:\n");
printf("  The TRACTOR configuration is incorrect.\n");
printf("  This program is not compatible with a \n");
printf("  RIGID DRAWBAR mounted on the CENTER of the\n");
printf("  tractor.\n");
printf("  Correct the Tractor Configuration File and
        proceed.\n");
Modify_Flag = TRUE;
}
break;

```

```

case 'R':

```

```

switch (*Tractor_Hitch_Swing_Point)
{
case 'A':
{
clrscr ();
printf("\n\n\n\n\n\n  ATTENTION!:\n");
printf("  The TRACTOR configuration is incorrect.\n");
printf("  The drawbar basepoint can not be l
        ocated \n");
printf("  AHEAD the FRONT AXLE on the tractor.\n");
printf("  Correct the Tractor Configuration File
        and proceed.\n");
Modify_Flag = TRUE;
}
break;

```

```

case 'B':

```

```

{
*Tractor_Hitch_Swing_Point = 'N';
*Tractor_Hitch_Swing_Distance = 0.0;

```

```
*Tractor_Max_Hitch_Swing_Angle = 0.0;
Modify_Flag = FALSE;
}
break;

case 'C':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The drawbar basepoint can not be
located \n");
printf(" BEHIND the FRONT AXLE on the tractor.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'D':
{
*Tractor_Hitch_Swing_Point = 'N';
*Tractor_Hitch_Swing_Distance = 0.0;
*Tractor_Max_Hitch_Swing_Angle = 0.0;
Modify_Flag = FALSE;
}
break;

case 'N':
{
*Tractor_Hitch_Swing_Distance = 0.0;
*Tractor_Max_Hitch_Swing_Angle = 0.0;
Modify_Flag = FALSE;
}
break;
```

```

default:
    {
        clrscr ();
        printf("\n\n\n\n\n  ATTENTION!\n");
        printf("  The TRACTOR configuration is incorrect.\n");
        printf("  Check the .TRC file for the correct\n");
        printf("  Tractor_Hitch_Swing_Point.\n");
        printf("  Correct the Tractor Configuration File
            and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

}
break;

}
break;

case 'T':
    switch (*Tractor_Hitch_Location)
    {
        case 'F':
            {
                clrscr ();
                printf("\n\n\n\n\n  ATTENTION!\n");
                printf("  The TRACTOR configuration is incorrect.\n");
                printf("  There can not be a THREE-POINT HITCH
                    located\n");
                printf("  at the FRONT of the tractor.\n");
                printf("  Correct the Tractor Configuration File and
                    proceed.\n");
                Modify_Flag = TRUE;
            }
    }

```

```
    }  
    break;  
  
case 'C':  
    {  
    clrscr ();  
    printf("\n\n\n\n\n\n  ATTENTION!:\n");  
    printf("  The TRACTOR configuration is incorrect.\n");  
    printf("  The can not be a THREE-POINT HITCH  
    located\n");  
    printf("  at the CENTER of the tractor.\n");  
    printf("  Correct the Tractor Configuration File and  
    proceed.\n");  
    Modify_Flag = TRUE;  
    }  
    break;  
  
case 'R':  
  
    switch (*Tractor_Hitch_Swing_Point)  
    {  
    case 'A':  
    {  
    clrscr ();  
    printf("\n\n\n\n\n\n  ATTENTION!:\n");  
    printf("  The TRACTOR configuration is incorrect.\n");  
    printf("  The drawbar SWINGPOINT should not  
    be located\n");  
    printf("  AHEAD of the FRONT AXLE of the tractor.\n");  
    printf("  Correct the Tractor Configuration File  
    and proceed.\n");  
    Modify_Flag = TRUE;  
    }  
    break;
```

```
case 'B':
{
  clrscr ();
  printf("\n\n\n\n\n ATTENTION!:\n");
  printf(" The TRACTOR configuration is incorrect.\n");
  printf(" The drawbar SWINGPOINT should not
        be located\n");
  printf(" BEHIND the REAR AXLE of the tractor.\n");
  printf(" Correct the Tractor Configuration File
        and proceed.\n");
  Modify_Flag = TRUE;
}
break;

case 'C':
  Modify_Flag = FALSE;
  break;

case 'D':
  Modify_Flag = FALSE;
  break;

case 'N':
{
  clrscr ();
  printf("\n\n\n\n\n ATTENTION!:\n");
  printf(" The TRACTOR configuration is incorrect.\n");
  printf(" The drawbar SWING POINT must be
        provided.\n");
  printf(" Correct the Tractor Configuration File
        and proceed.\n");
  Modify_Flag = TRUE;
}
break;
```

default:

```
{
  clrscr ();
  printf("\n\n\n\n\n ATTENTION!\n");
  printf(" The TRACTOR configuration is incorrect.\n");
  printf(" Check the .TRC file for the correct\n");
  printf(" Tractor_Hitch_Swing_Point.\n");
  printf(" Correct the Tractor Configuration File
        and proceed.\n");
  Modify_Flag = TRUE;
}
```

```
break;
```

```
}
```

```
break;
```

```
}
```

```
break;
```

case 'I':

```
switch (*Tractor_Hitch_Location)
```

```
{
```

```
case 'F':
```

```
case 'C':
```

```
case 'R':
```

```
switch (*Tractor_Hitch_Swing_Point)
```

```
{
```

```
case 'A':
```

```
case 'B':
```

```
case 'C':
```

```
case 'D':
```

```
{
```

```
*Tractor_Hitch_Swing_Point = 'N';
```

```
*Tractor_Hitch_Swing_Distance = 0.0;
```

```
*Tractor_Max_Hitch_Swing_Angle = 0.0;
```

```
        Modify_Flag = FALSE;
    }
    break;

default:
    {
        clrscr ();
        printf("\n\n\n\n\n ATTENTION!:\n");
        printf(" The TRACTOR configuration is incorrect.\n");
        printf(" Check the .TRC file for the correct\n");
        printf(" Tractor_Hitch_Swing_Point.\n");
        printf(" Correct the Tractor Configuration File
            and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

}
break;

}
break;

}
break;

case 'S':

    switch (*Tractor_Hitch_Type)
    {
        case 'S':

            switch (*Tractor_Hitch_Location)
            {
                case 'F':
```

```
{
    clrscr ();
    printf("\n\n\n\n\n ATTENTION!:\n");
    printf(" The TRACTOR configuration is incorrect.\n");
    printf(" There can not be a SWINGING DRAWBAR
        located\n");
    printf(" at the FRONT of the tractor.\n");
    printf(" Correct the Tractor Configuration File and
        proceed.\n");
    Modify_Flag = TRUE;
}
break;

case 'C':
{
    clrscr ();
    printf("\n\n\n\n\n ATTENTION!:\n");
    printf(" The TRACTOR configuration is incorrect.\n");
    printf(" There can not be a SWINGING DRAWBAR
        located\n");
    printf(" at the CENTER of the tractor.\n");
    printf(" Correct the Tractor Configuration File and
        proceed.\n");
    Modify_Flag = TRUE;
}
break;

case 'R':

    switch (*Tractor_Hitch_Swing_Point)
    {
        case 'A':
            {
                clrscr ();
```

```
printf("\n\n\n\n\n ATTENTION!\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The drawbar SWING POINT can not be
located \n");
printf(" AHEAD of the FRONT AXLE on the tractor.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'B':
Modify_Flag = FALSE;
break;

case 'C':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The drawbar SWING POINT can not be
located \n");
printf(" BEHIND the FRONT AXLE on the tractor.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'D':
Modify_Flag = FALSE;
break;

case 'N':
{
```

```
        clrscr ();
        printf("\n\n\n\n\n\n  ATTENTION!:\n");
        printf("  The TRACTOR configuration is incorrect.\n");
        printf("  The drawbar SWING POINT must be
                provided.\n");
        printf("  Correct the Tractor Configuration File
                and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

default:
    {
        clrscr ();
        printf("\n\n\n\n\n\n  ATTENTION!:\n");
        printf("  The TRACTOR configuration is incorrect.\n");
        printf("  Check the .TRC file for the correct\n");
        printf("  Tractor_Hitch_Swing_Point.\n");
        printf("  Correct the Tractor Configuration File
                and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

}
break;

}
break;

case 'R':

    switch (*Tractor_Hitch_Location)
    {
        case 'F':
```

```
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf("  The TRACTOR configuration is incorrect.\n");
printf("  This program is not compatible with a \n");
printf("  RIGID DRAWBAR mounted on the FRONT of the\n");
printf("  tractor.\n");
printf("  Correct the Tractor Configuration File and
        proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'C':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf("  The TRACTOR configuration is incorrect.\n");
printf("  This program is not compatible with a \n");
printf("  RIGID DRAWBAR mounted on the CENTER of the\n");
printf("  tractor.\n");
printf("  Correct the Tractor Configuration File and
        proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'R':

switch (*Tractor_Hitch_Swing_Point)
{
case 'A':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
```

```
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The drawbar basepoint can not be
located \n");
printf(" AHEAD the FRONT AXLE on the tractor.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;
```

```
case 'B':
{
*Tractor_Hitch_Swing_Point = 'N';
*Tractor_Hitch_Swing_Distance = 0.0;
*Tractor_Max_Hitch_Swing_Angle = 0.0;
Modify_Flag = FALSE;
}
break;
```

```
case 'C':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The drawbar basepoint can not be
located \n");
printf(" BEHIND the FRONT AXLE on the tractor.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;
```

```
case 'D':
{
```

```
*Tractor_Hitch_Swing_Point = 'N';
*Tractor_Hitch_Swing_Distance = 0.0;
*Tractor_Max_Hitch_Swing_Angle = 0.0;
Modify_Flag = FALSE;
}
break;

case 'N':
{
*Tractor_Hitch_Swing_Distance = 0.0;
*Tractor_Max_Hitch_Swing_Angle = 0.0;
Modify_Flag = FALSE;
}
break;

default:
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" Check the .TRC file for the correct\n");
printf(" Tractor_Hitch_Swing_Point.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;

}
break;

}
break;
```

```
case 'T':
```

```
switch (*Tractor_Hitch_Location)
{
case 'F':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" There can not be a THREE-POINT HITCH
located\n");
printf(" at the FRONT of the tractor.\n");
printf(" Correct the Tractor Configuration File and
proceed.\n");
Modify_Flag = TRUE;
}
break;
```

```
case 'C':
```

```
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" There can not be a THREE-POINT HITCH
located\n");
printf(" at the CENTER of the tractor.\n");
printf(" Correct the Tractor Configuration File and
proceed.\n");
Modify_Flag = TRUE;
}
break;
```

```
case 'R':
```

```
switch (*Tractor_Hitch_Swing_Point)
```

```
{
case 'A':
{
clrscr ();
printf("\n\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The drawbar SWINGPOINT should not
be located\n");
printf(" AHEAD of the FRONT AXLE of the tractor.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'B':
{
clrscr ();
printf("\n\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The drawbar SWINGPOINT should not
be located\n");
printf(" BEHIND the REAR AXLE of the tractor.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'C':
Modify_Flag = FALSE;
break;

case 'D':
Modify_Flag = FALSE;
```

```
        break;

    case 'N':
        {
            clrscr ();
            printf("\n\n\n\n\n ATTENTION!:\n");
            printf(" The TRACTOR configuration is incorrect.\n");
            printf(" The drawbar SWING POINT must be
                provided.\n");
            printf(" Correct the Tractor Configuration File
                and proceed.\n");
            Modify_Flag = TRUE;
        }
        break;

    default:
        {
            clrscr ();
            printf("\n\n\n\n\n ATTENTION!:\n");
            printf(" The TRACTOR configuration is incorrect.\n");
            printf(" Check the .TRC file for the correct\n");
            printf(" Tractor_Hitch_Swing_Point.\n");
            printf(" Correct the Tractor Configuration File
                and proceed.\n");
            Modify_Flag = TRUE;
        }
        break;
}
break;

}
break;

case 'I':
```

```
switch (*Tractor_Hitch_Location)
{
    case 'F':

        switch (*Tractor_Hitch_Swing_Point)
        {
            case 'A':
            case 'B':
            case 'C':
            case 'D':
                {
                    *Tractor_Hitch_Swing_Point = 'N';
                    *Tractor_Hitch_Swing_Distance = 0.0;
                    *Tractor_Max_Hitch_Swing_Angle = 0.0;
                    Modify_Flag = FALSE;
                }
                break;

            default:
                {
                    clrscr ();
                    printf("\n\n\n\n\n  ATTENTION!:\n");
                    printf("  The TRACTOR configuration is incorrect.\n");
                    printf("  Check the .TRC file for the correct\n");
                    printf("  Tractor_Hitch_Swing_Point.\n");
                    printf("  Correct the Tractor Configuration File
                        and proceed.\n");
                    Modify_Flag = TRUE;
                }
                break;

        }
        break;

    case 'C':
```

```

{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" There can not be a SEMI-INTEGRAL HITCH
located \n");
printf(" at the CENTER of the tractor.\n");
printf(" Correct the Tractor Configuration File and
proceed.\n");
Modify_Flag = TRUE;
}

```

case 'R':

```

switch (*Tractor_Hitch_Swing_Point)
{
case 'A':
case 'B':
case 'C':
case 'D':
{
*Tractor_Hitch_Swing_Point = 'N';
*Tractor_Hitch_Swing_Distance = 0.0;
*Tractor_Max_Hitch_Swing_Angle = 0.0;
Modify_Flag = FALSE;
}
break;

```

default:

```

{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" Check the .TRC file for the correct\n");
printf(" Tractor_Hitch_Swing_Point.\n");

```

```
        printf("    Correct the Tractor Configuration File
                and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

}
break;

}
break;

}
break;

}
break;

case 'L':

switch (*Tractor_Hitch_Type)
{
    case 'S':

        switch (*Tractor_Hitch_Location)
        {
            case 'F':
                {
                    clrscr ();
                    printf("\n\n\n\n\n ATTENTION!:\n");
                    printf("    The TRACTOR configuration is incorrect.\n");
                    printf("    There can not be a SWINGING DRAWBAR
                            located\n");
                    printf("    at the FRONT of the tractor.\n");
                    printf("    Correct the Tractor Configuration File and
                            proceed.\n");
                    Modify_Flag = TRUE;
                }
            }
        }
    }
}
```

```
break;

case 'C':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" There can not be a SWINGING DRAWBAR
located\n");
printf(" at the CENTER of the tractor.\n");
printf(" Correct the Tractor Configuration File and
proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'R':

switch (*Tractor_Hitch_Swing_Point)
{
case 'A':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The drawbar SWING POINT can not be
located \n");
printf(" AHEAD of the FRONT AXLE on the tractor.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;
```

```
case 'B':
    Modify_Flag = FALSE;
    break;

case 'C':
    {
        clrscr ();
        printf("\n\n\n\n\n ATTENTION!\n");
        printf(" The TRACTOR configuration is incorrect.\n");
        printf(" The drawbar SWING POINT can not be
            located \n");
        printf(" BEHIND the FRONT AXLE on the tractor.\n");
        printf(" Correct the Tractor Configuration File
            and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

case 'D':
    Modify_Flag = FALSE;
    break;

case 'N':
    {
        clrscr ();
        printf("\n\n\n\n\n ATTENTION!\n");
        printf(" The TRACTOR configuration is incorrect.\n");
        printf(" The drawbar SWING POINT must be
            provided.\n");
        printf(" Correct the Tractor Configuration File
            and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;
```

```
default:
    {
        clrscr ();
        printf("\n\n\n\n\n ATTENTION!:\n");
        printf(" The TRACTOR configuration is incorrect.\n");
        printf(" Check the .TRC file for the correct\n");
        printf(" Tractor_Hitch_Swing_Point.\n");
        printf(" Correct the Tractor Configuration File
                and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

}
break;

}
break;

case 'R':

switch (*Tractor_Hitch_Location)
{
case 'F':
    {
        clrscr ();
        printf("\n\n\n\n\n ATTENTION!:\n");
        printf(" The TRACTOR configuration is incorrect.\n");
        printf(" This program is not compatible with a \n");
        printf(" RIGID DRAWBAR mounted on the FRONT of the\n");
        printf(" tractor.\n");
        printf(" Correct the Tractor Configuration File and
                proceed.\n");
        Modify_Flag = TRUE;
    }
}
```

```
break;

case 'C':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" This program is not compatible with a \n");
printf(" RIGID DRAWBAR mounted on the CENTER of the\n");
printf(" tractor.\n");
printf(" Correct the Tractor Configuration File and
        proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'R':

switch (*Tractor_Hitch_Swing_Point)
{
case 'A':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The drawbar basepoint can not be
        located \n");
printf(" AHEAD the FRONT AXLE on the tractor.\n");
printf(" Correct the Tractor Configuration File
        and proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'B':
```

```
{
*Tractor_Hitch_Swing_Point = 'N';
*Tractor_Hitch_Swing_Distance = 0.0;
*Tractor_Max_Hitch_Swing_Angle = 0.0;
Modify_Flag = FALSE;
}
break;

case 'C':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" The drawbar basepoint can not be
located \n");
printf(" BEHIND the FRONT AXLE on the tractor.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'D':
{
*Tractor_Hitch_Swing_Point = 'N';
*Tractor_Hitch_Swing_Distance = 0.0;
*Tractor_Max_Hitch_Swing_Angle = 0.0;
Modify_Flag = FALSE;
}
break;

case 'N':
{
*Tractor_Hitch_Swing_Distance = 0.0;
*Tractor_Max_Hitch_Swing_Angle = 0.0;
```

```
Modify_Flag = FALSE;  
}  
break;
```

```
default:
```

```
{  
clrscr ();  
printf("\n\n\n\n\n\n ATTENTION!:\n");  
printf(" The TRACTOR configuration is incorrect.\n");  
printf(" Check the .TRC file for the correct\n");  
printf(" Tractor_Hitch_Swing_Point.\n");  
printf(" Correct the Tractor Configuration File  
and proceed.\n");
```

```
Modify_Flag = TRUE;
```

```
}
```

```
break;
```

```
}
```

```
break;
```

```
}
```

```
break;
```

```
case 'T':
```

```
switch (*Tractor_Hitch_Location)
```

```
{
```

```
case 'F':
```

```
switch (*Tractor_Hitch_Swing_Point)
```

```
{
```

```
case 'A':
```

```
{
```

```
clrscr ();
```

```
printf("\n\n\n\n\n\n  ATTENTION!:\n");
printf("  The TRACTOR configuration is incorrect.\n");
printf("  The drawbar SWING POINT should not
        be located \n");
printf("  AHEAD the FRONT AXLE on the tractor.\n");
printf("  Correct the Tractor Configuration File
        and proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'B':
{
  clrscr ();
  printf("\n\n\n\n\n\n  ATTENTION!:\n");
  printf("  The TRACTOR configuration is incorrect.\n");
  printf("  The drawbar SWING POINT should not
        be located \n");
  printf("  BEHIND the REAR AXLE on the tractor.\n");
  printf("  Correct the Tractor Configuration File
        and proceed.\n");
  Modify_Flag = TRUE;
}
break;

case 'C':
  Modify_Flag = FALSE;
  break;

case 'D':
  Modify_Flag = FALSE;
  break;

case 'N':
{
```

```

    clrscr ();
    printf("\n\n\n\n\n ATTENTION!:\n");
    printf(" The TRACTOR configuration is incorrect.\n");
    printf(" The drawbar SWING POINT must be
        provided.\n");
    printf(" Correct the Tractor Configuration File
        and proceed.\n");
    Modify_Flag = TRUE;
}
break;

default:
{
    clrscr ();
    printf("\n\n\n\n\n ATTENTION!:\n");
    printf(" The TRACTOR configuration is incorrect.\n");
    printf(" Check the .TRC file for the correct\n");
    printf(" Tractor_Hitch_Swing_Point.\n");
    printf(" Correct the Tractor Configuration File
        and proceed.\n");
    Modify_Flag = TRUE;
}
break;
}
break;

case 'C':
{
    clrscr ();
    printf("\n\n\n\n\n ATTENTION!:\n");
    printf(" The TRACTOR configuration is incorrect.\n");
    printf(" The can not be a THREE-POINT HITCH
        located\n");
    printf(" at the CENTER of the tractor.\n");

```

```
printf("  Correct the Tractor Configuration File and  
        proceed.\n");
```

```
Modify_Flag = TRUE;
```

```
}
```

```
break;
```

```
case 'R':
```

```
switch (*Tractor_Hitch_Swing_Point)
```

```
{
```

```
  case 'A':
```

```
    {
```

```
      clrscr ();
```

```
      printf("\n\n\n\n\n  ATTENTION!:\n");
```

```
      printf("  The TRACTOR configuration is incorrect.\n");
```

```
      printf("  The drawbar SWINGPOINT should not  
              be located\n");
```

```
      printf("  AHEAD of the FRONT AXLE of the tractor.\n");
```

```
      printf("  Correct the Tractor Configuration File  
              and proceed.\n");
```

```
      Modify_Flag = TRUE;
```

```
    }
```

```
    break;
```

```
case 'B':
```

```
{
```

```
  clrscr ();
```

```
  printf("\n\n\n\n\n  ATTENTION!:\n");
```

```
  printf("  The TRACTOR configuration is incorrect.\n");
```

```
  printf("  The drawbar SWINGPOINT should not  
          be located\n");
```

```
  printf("  BEHIND the REAR AXLE of the tractor.\n");
```

```
  printf("  Correct the Tractor Configuration File  
          and proceed.\n");
```

```
  Modify_Flag = TRUE;
```

```
    }  
    break;  
  
    case 'C':  
        Modify_Flag = FALSE;  
        break;  
  
    case 'D':  
        Modify_Flag = FALSE;  
        break;  
  
    case 'N':  
        {  
        clrscr ();  
        printf("\n\n\n\n\n\n  ATTENTION!:\n");  
        printf("  The TRACTOR configuration is incorrect.\n");  
        printf("  The drawbar SWING POINT must be  
        provided.\n");  
        printf("  Correct the Tractor Configuration File  
        and proceed.\n");  
        Modify_Flag = TRUE;  
        }  
        break;  
  
    default:  
        {  
        clrscr ();  
        printf("\n\n\n\n\n\n  ATTENTION!:\n");  
        printf("  The TRACTOR configuration is incorrect.\n");  
        printf("  Check the .TRC file for the correct\n");  
        printf("  Tractor_Hitch_Swing_Point.\n");  
        printf("  Correct the Tractor Configuration File  
        and proceed.\n");  
        Modify_Flag = TRUE;  
        }  
}
```

```
        break;
    }
    break;

}
break;

case 'I':

    switch (*Tractor_Hitch_Location)
    {
        case 'F':
        case 'C':
        case 'R':

            switch (*Tractor_Hitch_Swing_Point)
            {
                case 'A':
                case 'B':
                case 'C':
                case 'D':
                    {
                        *Tractor_Hitch_Swing_Point = 'N';
                        *Tractor_Hitch_Swing_Distance = 0.0;
                        *Tractor_Max_Hitch_Swing_Angle = 0.0;
                        Modify_Flag = FALSE;
                    }
                break;

            default:
                {
                    clrscr ();
                    printf("\n\n\n\n\n  ATTENTION!:\n");
                    printf("  The TRACTOR configuration is incorrect.\n");
                    printf("  Check the .TRC file for the correct\n");
                }
            }
        }
    }
}
```

```

printf("  Tractor_Hitch_Swing_Point.\n");
printf("  Correct the Tractor Configuration File
      and proceed.\n");
Modify_Flag = TRUE;
}
break;

}
break;

}
break;

}
break;

case 'R':

switch (*Tractor_Hitch_Type)
{
case 'S':

switch (*Tractor_Hitch_Location)
{
case 'F':
{
clrscr ();
printf("\n\n\n\n\n  ATTENTION!:\n");
printf("  The TRACTOR configuration is incorrect.\n");
printf("  There can not be a SWINGING DRAWBAR
      located\n");
printf("  at the FRONT of the tractor. \n");
printf("  (Front as determined by travel direction).\n");
printf("  Correct the Tractor Configuration File and
      proceed.\n");

```

```
Modify_Flag = TRUE;
}
break;

case 'C':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is incorrect.\n");
printf(" There can not be a SWINGING DRAWBAR
located\n");
printf(" at the CENTER of the tractor.\n");
printf(" Correct the Tractor Configuration File and
proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'R':

switch (*Tractor_Hitch_Swing_Point)
{
case 'A':
{
clrscr ();
printf("\n\n\n\n\n ATTENTION!:\n");
printf(" The TRACTOR configuration is
incorrect.\n");
printf(" The drawbar SWING POINT can not be
located \n");
printf(" AHEAD of the FRONT AXLE on the tractor.\n");
printf(" Correct the Tractor Configuration File
and proceed.\n");
Modify_Flag = TRUE;
}
```

```
    }  
    break;  
  
case 'B':  
    Modify_Flag = FALSE;  
    break;  
  
case 'C':  
    {  
    clrscr ();  
    printf("\n\n\n\n\n\n  ATTENTION!:\n");  
    printf("  The TRACTOR configuration is incorrect.\n");  
    printf("  The drawbar SWING POINT can not be  
        located \n");  
    printf("  BEHIND the FRONT AXLE on the tractor.\n");  
    printf("  Correct the Tractor Configuration File  
        and proceed.\n");  
    Modify_Flag = TRUE;  
    }  
    break;  
  
case 'D':  
    Modify_Flag = FALSE;  
    break;  
  
case 'N':  
    {  
    clrscr ();  
    printf("\n\n\n\n\n\n  ATTENTION!:\n");  
    printf("  The TRACTOR configuration is incorrect.\n");  
    printf("  The drawbar SWING POINT must be  
        provided.\n");  
    printf("  Correct the Tractor Configuration File  
        and proceed.\n");  
    Modify_Flag = TRUE;
```

```
    }
    break;

default:
    {
        clrscr ();
        printf("\n\n\n\n\n\n  ATTENTION!:\n");
        printf("  The TRACTOR configuration is incorrect.\n");
        printf("  Check the .TRC file for the correct\n");
        printf("  Tractor_Hitch_Swing_Point.\n");
        printf("  Correct the Tractor Configuration File
            and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

}
break;

}
break;

case 'R':

    switch (*Tractor_Hitch_Location)
    {
        case 'F':
            {
                clrscr ();
                printf("\n\n\n\n\n\n  ATTENTION!:\n");
                printf("  The TRACTOR configuration is incorrect.\n");
                printf("  This program is not compatible with a \n");
                printf("  RIGID DRAWBAR mounted on the FRONT of the\n");
                printf("  tractor.\n");
            }
        }
    }
}
```

```
printf("  Correct the Tractor Configuration File and
        proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'C':
{
clrscr ();
printf("\n\n\n\n\n  ATTENTION!:\n");
printf("  The TRACTOR configuration is incorrect.\n");
printf("  This program is not compatible with a \n");
printf("  RIGID DRAWBAR mounted on the CENTER of the\n");
printf("  tractor.\n");
printf("  Correct the Tractor Configuration File and
        proceed.\n");
Modify_Flag = TRUE;
}
break;

case 'R':

switch (*Tractor_Hitch_Swing_Point)
{
case 'A':
{
clrscr ();
printf("\n\n\n\n\n  ATTENTION!:\n");
printf("  The TRACTOR configuration is incorrect.\n");
printf("  The drawbar basepoint can not be
        located \n");
printf("  AHEAD the FRONT AXLE on the tractor.\n");
printf("  Correct the Tractor Configuration File
        and proceed.\n");
Modify_Flag = TRUE;
}
```

```
    }  
    break;  
  
case 'B':  
    {  
    *Tractor_Hitch_Swing_Point = 'N';  
    *Tractor_Hitch_Swing_Distance = 0.0;  
    *Tractor_Max_Hitch_Swing_Angle = 0.0;  
    Modify_Flag = FALSE;  
    }  
    break;  
  
case 'C':  
    {  
    clrscr ();  
    printf("\n\n\n\n\n ATTENTION!:\n");  
    printf(" The TRACTOR configuration is incorrect.\n");  
    printf(" The drawbar basepoint can not be  
        located \n");  
    printf(" BEHIND the FRONT AXLE on the tractor.\n");  
    printf(" Correct the Tractor Configuration File  
        and proceed.\n");  
    Modify_Flag = TRUE;  
    }  
    break;  
  
case 'D':  
    {  
    *Tractor_Hitch_Swing_Point = 'N';  
    *Tractor_Hitch_Swing_Distance = 0.0;  
    *Tractor_Max_Hitch_Swing_Angle = 0.0;  
    Modify_Flag = FALSE;  
    }  
    break;
```

```
case 'N':
    {
        *Tractor_Hitch_Swing_Distance = 0.0;
        *Tractor_Max_Hitch_Swing_Angle = 0.0;
        Modify_Flag = FALSE;
    }
    break;

default:
    {
        clrscr ();
        printf("\n\n\n\n\n ATTENTION!\n");
        printf(" The TRACTOR configuration is incorrect.\n");
        printf(" Check the .TRC file for the correct\n");
        printf(" Tractor_Hitch_Swing_Point.\n");
        printf(" Correct the Tractor Configuration File
            and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

}
break;

}
break;

case 'T':

    switch (*Tractor_Hitch_Location)
    {
        case 'F':

            switch (*Tractor_Hitch_Swing_Point)
```

```
{
case 'A':
    {
        clrscr ();
        printf("\n\n\n\n\n ATTENTION!:\n");
        printf(" The TRACTOR configuration is incorrect.\n");
        printf(" The drawbar SWING POINT should not
            be located \n");
        printf(" AHEAD the FRONT AXLE on the tractor.\n");
        printf(" Correct the Tractor Configuration File
            and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

case 'B':
    {
        clrscr ();
        printf("\n\n\n\n\n ATTENTION!:\n");
        printf(" The TRACTOR configuration is incorrect.\n");
        printf(" The drawbar SWING POINT should not
            be located \n");
        printf(" BEHIND the REAR AXLE on the tractor.\n");
        printf(" Correct the Tractor Configuration File
            and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

case 'C':
    Modify_Flag = FALSE;
    break;

case 'D':
    Modify_Flag = FALSE;
```

```
        break;

    case 'N':
        {
            clrscr ();
            printf("\n\n\n\n\n ATTENTION!:\n");
            printf(" The TRACTOR configuration is incorrect.\n");
            printf(" The drawbar SWING POINT must be
                provided.\n");
            printf(" Correct the Tractor Configuration File
                and proceed.\n");
            Modify_Flag = TRUE;
        }
        break;

    default:
        {
            clrscr ();
            printf("\n\n\n\n\n ATTENTION!:\n");
            printf(" The TRACTOR configuration is incorrect.\n");
            printf(" Check the .TRC file for the correct\n");
            printf(" Tractor_Hitch_Swing_Point.\n");
            printf(" Correct the Tractor Configuration File
                and proceed.\n");
            Modify_Flag = TRUE;
        }
        break;
}
break;

case 'C':
{
    clrscr ();
    printf("\n\n\n\n\n ATTENTION!:\n");
    printf(" The TRACTOR configuration is incorrect.\n");
```

```

printf("  The can not be a THREE-POINT HITCH
        located\n");
printf("  at the CENTER of the tractor.\n");
printf("  Correct the Tractor Configuration File and
        proceed.\n");
Modify_Flag = TRUE;
}
break;

```

```
case 'R':
```

```

switch (*Tractor_Hitch_Swing_Point)
{
case 'A':
{
clrscr ();
printf("\n\n\n\n\n  ATTENTION!:\n");
printf("  The TRACTOR configuration is incorrect.\n");
printf("  The drawbar SWINGPOINT should not
        be located\n");
printf("  AHEAD of the FRONT AXLE of the tractor.\n");
printf("  Correct the Tractor Configuration File
        and proceed.\n");
Modify_Flag = TRUE;
}
break;

```

```
case 'B':
```

```

{
clrscr ();
printf("\n\n\n\n\n  ATTENTION!:\n");
printf("  The TRACTOR configuration is incorrect.\n");
printf("  The drawbar SWINGPOINT should not
        be located\n");
printf("  BEHIND the REAR AXLE of the tractor.\n");

```

```
        printf("    Correct the Tractor Configuration File
                and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

case 'C':
    Modify_Flag = FALSE;
    break;

case 'D':
    Modify_Flag = FALSE;
    break;

case 'N':
    {
        clrscr ();
        printf("\n\n\n\n\n  ATTENTION!:\n");
        printf("  The TRACTOR configuration is incorrect.\n");
        printf("  The drawbar SWING POINT must be
                provided.\n");
        printf("  Correct the Tractor Configuration File
                and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;

default:
    {
        clrscr ();
        printf("\n\n\n\n\n  ATTENTION!:\n");
        printf("  The TRACTOR configuration is incorrect.\n");
        printf("  Check the .TRC file for the correct\n");
        printf("  Tractor_Hitch_Swing_Point.\n");
```

```
        printf("    Correct the Tractor Configuration File
                and proceed.\n");
        Modify_Flag = TRUE;
    }
    break;
}
break;

}
break;

case 'I':

    switch (*Tractor_Hitch_Location)
    {
        case 'F':
        case 'C':
        case 'R':

            switch (*Tractor_Hitch_Swing_Point)
            {
                case 'A':
                case 'B':
                case 'C':
                case 'D':
                    {
                        *Tractor_Hitch_Swing_Point = 'N';
                        *Tractor_Hitch_Swing_Distance = 0.0;
                        *Tractor_Max_Hitch_Swing_Angle = 0.0;
                        Modify_Flag = FALSE;
                    }
                break;

            default:
                {
```


APPENDIX E

TRACTOR CONFIGURATION MODULE [TRACTOR.C]

```
/* Tractor Configuration Module [TRACTOR.C] */
```

```
#include "gpsheade.h"
```

```
/*BTM*****
```

```
/* Tractor Configuration (Procedure) */
```

```
void Configure_Tractor()
```

```
{
```

```
char c;
```

```
char File_Type;
```

```
char File_Mode [5];
```

```
Boolean done = FALSE;
```

```
char Tractor_Steering_Type;
```

```
char Tractor_Hitch_Type;
```

```
char Tractor_Hitch_Swing_Point;
```

```
char Tractor_Hitch_Location;
```

```
double Tractor_Wheelbase;
```

```
double Tractor_Drawbar_Length;
```

```
double Tractor_Hitch_Swing_Distance;
```

```
double Tractor_Max_Hitch_Swing_Angle;
```

```
double Tractor_Max_Steering_Angle;
```

```
do
```

```
{
```

```
fflush (stdin);
```

```
clrscr ();
```

```
/* Main Tractor Configuration Menu */
```

```
printf("\n\n\n\n\n\n Tractor Configuration Menu.\n\n\n\n\n\n");
```

```
printf(" [C] CREATE (Make a Tractor Configuration File)\n");
```

```
printf(" [M] MODIFY (Change information in a Tractor Configuration File)\n");  
printf(" [V] VIEW (Examine an existing Tractor Configuration File)\n");  
printf(" [E] EXIT (Return to the Main Menu)\n\n\n\n");
```

```
printf(" Press the letter corresponding to the operation you want to  
do.\n\n");
```

```
c = getc (stdin);  
fflush (stdin);  
c = toupper (c);
```

```
switch (c)
```

```
{
```

```
case 'C':
```

```
    Create_Tractor_Configuration ();
```

```
    done = TRUE;
```

```
    break;
```

```
case 'M':
```

```
    File_Type = 'T';
```

```
    sprintf(File_Mode,"r+");
```

```
    Open_Tractor_Config_File (File_Type, File_Mode,  
                              &Tractor_Steering_Type,  
                              &Tractor_Hitch_Type,  
                              &Tractor_Hitch_Swing_Point,  
                              &Tractor_Hitch_Location,  
                              &Tractor_Wheelbase,  
                              &Tractor_Drawbar_Length,  
                              &Tractor_Hitch_Swing_Distance,  
                              &Tractor_Max_Hitch_Swing_Angle,  
                              &Tractor_Max_Steering_Angle);
```

```
clrscr ();
```

```
Check_Tractor_Configuration (&Tractor_Steering_Type,
```

```
&Tractor_Hitch_Type,  
&Tractor_Hitch_Swing_Point,  
&Tractor_Hitch_Location,  
&Tractor_Wheelbase,  
&Tractor_Drawbar_Length,  
&Tractor_Hitch_Swing_Distance,  
&Tractor_Max_Hitch_Swing_Angle,  
&Tractor_Max_Steering_Angle);
```

```
clrscr ();  
done = TRUE;  
break;
```

```
case 'V':
```

```
clrscr ();
```

```
File_Type = 'T';  
sprintf(File_Mode,"r");
```

```
Open_Tractor_Config_File (File_Type, File_Mode,  
    &Tractor_Steering_Type,  
    &Tractor_Hitch_Type,  
    &Tractor_Hitch_Swing_Point,  
    &Tractor_Hitch_Location,  
    &Tractor_Wheelbase,  
    &Tractor_Drawbar_Length,  
    &Tractor_Hitch_Swing_Distance,  
    &Tractor_Max_Hitch_Swing_Angle,  
    &Tractor_Max_Steering_Angle);
```

```
Tractor_File_Info (Tractor_Steering_Type,  
    Tractor_Hitch_Type,  
    Tractor_Hitch_Swing_Point,  
    Tractor_Hitch_Location,  
    Tractor_Wheelbase,  
    Tractor_Drawbar_Length,
```

```
Tractor_Hitch_Swing_Distance,  
Tractor_Max_Hitch_Swing_Angle,  
Tractor_Max_Steering_Angle);  
  
Continue_Program ();  
  
done = TRUE;  
break;  
  
case 'E':  
done = TRUE;  
break;  
  
default:  
printf("\a");  
break;  
  
}  
} while (!done);  
}  
  
/*BTM*****  
/* Create Tractor Configuration (Procedure) */  
void Create_Tractor_Configuration ()  
  
{  
char Tractor_Steering_Type;  
char Tractor_Hitch_Type;  
char Tractor_Hitch_Swing_Point;  
char Tractor_Hitch_Location;  
  
double Tractor_Wheelbase;  
double Tractor_Drawbar_Length;  
double Tractor_Hitch_Swing_Distance;  
double Tractor_Max_Hitch_Swing_Angle;
```

```
double Tractor_Max_Steering_Angle;
```

```
fflush(stdin);
```

```
clrscr();
```

```
printf("\n\n\n\n\n Create Tractor Configuration\n\n\n\n\n");
```

```
/* Call Tractor Parameter Input Functions */
```

```
Steering_Type      (&Tractor_Steering_Type);
```

```
Hitch_Type        (&Tractor_Hitch_Type);
```

```
Hitch_Location    (&Tractor_Hitch_Location);
```

```
Wheelbase         (&Tractor_Wheelbase);
```

```
Drawbar_Length    (&Tractor_Drawbar_Length);
```

```
if ((Tractor_Hitch_Type == 'S') || (Tractor_Hitch_Type == 'T'))
```

```
{
```

```
    Hitch_Swing_Point    (&Tractor_Hitch_Swing_Point);
```

```
    Hitch_Swing_Distance (&Tractor_Hitch_Swing_Distance);
```

```
    Max_Hitch_Swing_Angle (&Tractor_Max_Hitch_Swing_Angle);
```

```
}
```

```
else
```

```
{
```

```
    Tractor_Hitch_Swing_Point = 'N';
```

```
    Tractor_Hitch_Swing_Distance = 0.0;
```

```
    Tractor_Max_Hitch_Swing_Angle = 0.0;
```

```
}
```

```
Max_Steering_Angle (&Tractor_Max_Steering_Angle);
```

```
Check_Tractor_Configuration (&Tractor_Steering_Type,
```

```
                             &Tractor_Hitch_Type,
```

```
                             &Tractor_Hitch_Swing_Point,
```

```
                             &Tractor_Hitch_Location,
```

```
                             &Tractor_Wheelbase,
```

```

        &Tractor_Drawbar_Length,
        &Tractor_Hitch_Swing_Distance,
        &Tractor_Max_Hitch_Swing_Angle,
        &Tractor_Max_Steering_Angle);
    }

/*BTM*****
/* Check Tractor Configuration File */
void Check_Tractor_Configuration
(char *Tractor_Steering_Type,
char *Tractor_Hitch_Type,
char *Tractor_Hitch_Swing_Point,
char *Tractor_Hitch_Location,
double *Tractor_Wheelbase,
double *Tractor_Drawbar_Length,
double *Tractor_Hitch_Swing_Distance,
double *Tractor_Max_Hitch_Swing_Angle,
double *Tractor_Max_Steering_Angle)

{
    char c;
    char File_Type;
    char File_Mode [5];
    Boolean done = FALSE;

    fflush (stdin);

    do
    {
        Tractor_Configuration_Logic (Tractor_Steering_Type,
        Tractor_Hitch_Type,
        Tractor_Hitch_Swing_Point,
        Tractor_Hitch_Location,
        Tractor_Wheelbase,

```

```
Tractor_Drawbar_Length,  
Tractor_Hitch_Swing_Distance,  
Tractor_Max_Hitch_Swing_Angle,  
Tractor_Max_Steering_Angle);
```

```
Tractor_File_Info (*Tractor_Steering_Type,  
    *Tractor_Hitch_Type,  
    *Tractor_Hitch_Swing_Point,  
    *Tractor_Hitch_Location,  
    *Tractor_Wheelbase,  
    *Tractor_Drawbar_Length,  
    *Tractor_Hitch_Swing_Distance,  
    *Tractor_Max_Hitch_Swing_Angle,  
    *Tractor_Max_Steering_Angle);
```

```
printf("\n\n Is all information correct?\n");  
printf(" Press\n [Y] if YES, to save the file \n [N] if NO, or\n  
[E] to RETURN to ");  
printf("the Tractor Configuration Menu without saving file.\n\n");
```

```
c = getc (stdin);  
fflush (stdin);  
c = toupper (c);
```

```
switch (c)  
{  
    case 'Y':
```

```
        File_Type = 'T';  
        sprintf(File_Mode,"w");
```

```
        Open_Tractor_Config_File (File_Type, File_Mode,  
            Tractor_Steering_Type,
```

```
Tractor_Hitch_Type,  
Tractor_Hitch_Swing_Point,  
Tractor_Hitch_Location,  
Tractor_Wheelbase,  
Tractor_Drawbar_Length,  
Tractor_Hitch_Swing_Distance,  
Tractor_Max_Hitch_Swing_Angle,  
Tractor_Max_Steering_Angle);
```

```
done = TRUE;  
break;
```

```
case 'N':
```

```
Modify_Tractor_Configuration (Tractor_Steering_Type,  
Tractor_Hitch_Type,  
Tractor_Hitch_Swing_Point,  
Tractor_Hitch_Location,  
Tractor_Wheelbase,  
Tractor_Drawbar_Length,  
Tractor_Hitch_Swing_Distance,  
Tractor_Max_Hitch_Swing_Angle,  
Tractor_Max_Steering_Angle);
```

```
done = FALSE;  
break;
```

```
case 'E':
```

```
done = TRUE;  
break;
```

```
default:
```

```
fflush (stdin);  
printf("\a");  
break;
```

```
}
```

```

    } while (!done);
}

/*BTM*****
/* Tractor File Information */
void Tractor_File_Info
(char Tractor_Steering_Type,
char Tractor_Hitch_Type,
char Tractor_Hitch_Swing_Point,
char Tractor_Hitch_Location,
double Tractor_Wheelbase,
double Tractor_Drawbar_Length,
double Tractor_Hitch_Swing_Distance,
double Tractor_Max_Hitch_Swing_Angle,
double Tractor_Max_Steering_Angle)
{
    clrscr ();

        /* Display Tractor File Configuration */

    printf("\n\n\n\n\n The current Tractor Configuration File is as
follows:\n\n");
    printf(" The Tractor Steering Type:      ");
    switch (Tractor_Steering_Type)
    {
        case 'F':
            printf("FRONT WHEEL.\n");
            break;

        case 'A':
            printf("ARTICULATED (or FRAME).\n");
            break;

        case 'S':

```

```
printf("SKID (or CONTROLLED DIFFERENTIAL).\n");
break;

case 'L':
    printf("ALL WHEEL.\n");
    break;

case 'R':
    printf("REAR WHEEL.\n");
    break;

default:
    printf("\a");
    printf("\n  There is an error in the Tractor Configuration File.\n");
    printf("  The Tractor Steering Type is invalid.\n");
    break;
}

printf(" Tractor Hitch Type:      ");
switch (Tractor_Hitch_Type)
{
    case 'S':
        printf("SWINGING DRAWBAR.\n");
        break;

    case 'R':
        printf("RIGID DRAWBAR.\n");
        break;

    case 'T':
        printf("THREE-POINT.\n");
        break;

    case 'I':
        printf("SEMI-INTEGRAL.\n");
```

```

        break;

    default:
        printf("\a");
        printf("\n  There is an error in the Tractor Configuration File.\n");
        printf("    The Tractor Hitch Type is invalid.\n");
        break;
}

printf(" Tractor Hitch Swing Point is located ");
switch (Tractor_Hitch_Swing_Point)
{
    case 'A':
        printf("AHEAD of the FRONT AXLE.\n");
        break;

    case 'B':
        printf("BEHIND the REAR AXLE.\n");
        break;

    case 'C':
        printf("BEHIND the FRONT AXLE.\n");
        break;

    case 'D':
        printf("AHEAD of the REAR AXLE.\n");
        break;

    case 'N':
        printf("(NOT APPLICABLE IN THIS CASE)\n");
        break;

    default:
        printf("\a");
        printf("\n  There is an error in the Tractor Configuration File.\n");

```

```

        printf(" The Tractor Hitch Swing Point is invalid.\n");
        break;
    }

printf(" The Tractor Hitch is located at the ");
switch (Tractor_Hitch_Location)
{
    case 'F':
        printf("FRONT ");
        break;

    case 'C':
        printf("CENTER SECTION ");
        break;

    case 'R':
        printf("REAR ");
        break;

    default:
        printf("\a");
        printf("\n There is an error in the Tractor Configuration File.\n");
        printf(" The Tractor Hitch Location is invalid.\n");
        break;
}

printf("of the Tractor.\n");

printf(" Tractor Wheelbase Length:      %f (meters).\n",
        Tractor_Wheelbase);
printf(" Tractor Drawbar Length:        %f (meters).\n",
        Tractor_Drawbar_Length);
printf(" Tractor Hitch Swing Distance:   %f (meters).\n",
        Tractor_Hitch_Swing_Distance);
printf(" Tractor Maximum Hitch Swing Angle: %f (degrees).\n",
        Tractor_Max_Hitch_Swing_Angle);

```

```

printf(" Tractor Maximum Steering Angle:   %f (degrees).\n",
      Tractor_Max_Steering_Angle);

}

/*BTM*****
/* Modify Tractor Configuration (Procedure) */
void Modify_Tractor_Configuration
(char *Tractor_Steering_Type,
 char *Tractor_Hitch_Type,
 char *Tractor_Hitch_Swing_Point,
 char *Tractor_Hitch_Location,
 double *Tractor_Wheelbase,
 double *Tractor_Drawbar_Length,
 double *Tractor_Hitch_Swing_Distance,
 double *Tractor_Max_Hitch_Swing_Angle,
 double *Tractor_Max_Steering_Angle)

{
char c;
Boolean done = FALSE;

do
{
fflush (stdin);
clrscr ();

/* Modify Tractor Configuration Menu */
printf("\n\n\n\n\n Modify Tractor Configuration.\n\n\n\n\n");

printf(" [S] Tractor Steering Type\n");
printf(" [H] Tractor Hitch Type\n");
printf(" [P] Tractor Hitch Swing Point\n");
printf(" [A] Tractor Hitch Location\n");
printf(" [W] Tractor Wheelbase\n");

```

```
printf(" [L] Tractor Drawbar Length\n");
printf(" [D] Tractor Hitch Swing Distance\n");
printf(" [M] Tractor Maximum Hitch Swing Angle\n");
printf(" [C] Tractor Maximum Steering Angle\n");
printf(" [E] Return to previous menu\n\n\n\n\n");
```

```
printf(" Enter the letter corresponding to the\n");
printf(" Tractor Configuration File item you want to change\n\n");
```

```
c = getc (stdin);
fflush (stdin);
c = toupper (c);
```

```
switch (c)
{
    case 'S':
        Steering_Type (Tractor_Steering_Type);
        break;

    case 'H':
        Hitch_Type (Tractor_Hitch_Type);
        break;

    case 'P':
        Hitch_Swing_Point (Tractor_Hitch_Swing_Point);
        break;

    case 'A':
        Hitch_Location (Tractor_Hitch_Location);
        break;

    case 'W':
        Wheelbase (Tractor_Wheelbase);
        break;
```

```

case 'L':
    Drawbar_Length (Tractor_Drawbar_Length);
    break;

case 'D':
    Hitch_Swing_Distance (Tractor_Hitch_Swing_Distance);
    break;

case 'M':
    Max_Hitch_Swing_Angle (Tractor_Max_Hitch_Swing_Angle);
    break;

case 'C':
    Max_Steering_Angle (Tractor_Max_Steering_Angle);

case 'E':
    done = TRUE;
    fflush (stdin);
    break;

default:
    printf("\a");
    break;
}
} while (!done);

```

```

}

```

```

/*BTM*****

```

```

/* Tractor Steering Type Input */

```

```

void Steering_Type (char *Tractor_Steering_Type)

```

```

{

```

```

    char c;

```

```

    Boolean done;

```

```

    done = FALSE;

```

```

do
{
  clrscr ();
  fflush (stdin);

  printf("\n\n\n\n\n\n  Tractor Steering Type\n\n\n\n\n\n");

  printf("  [F] FRONT WHEEL.\n");
  printf("  [A] ARTICULATED or FRAME.\n");
  printf("  [S] SKID or CONTROLLED DIFFERENTIAL.\n");
  printf("  [L] ALL WHEEL.\n");
  printf("  [R] REAR WHEEL.\n\n\n\n\n\n");

  printf("  Enter the letter corresponding to the Tractor Steering Type.\n\n");

  c = getc (stdin);
  fflush (stdin);
  c = toupper (c);

  if (c=='F' || c=='A' || c=='S' || c=='L' || c=='R')
  {
    *Tractor_Steering_Type = c;
    done = TRUE;
  }
  else
  {
    printf("\a");
    done = FALSE;
  }

} while (done != TRUE);
}

/*BTM*****

```

```
/* Tractor Hitch Type Input */
void Hitch_Type (char *Tractor_Hitch_Type)
{
    char c;
    Boolean done;
    done = FALSE;

    do
    {
        clrscr ();
        fflush (stdin);

        printf("\n\n\n\n\n\n  Tractor Hitch Type.\n\n\n\n\n\n");

        printf("  [S] SWINGING DRAWBAR.\n");
        printf("  [R] RIGID DRAWBAR.\n");
        printf("  [T] THREE-POINT.\n");
        printf("  [I] SEMI-INTEGRAL.\n\n\n\n\n\n");

        printf("  Enter the letter corresponding to the Tractor Hitch Type.\n\n");

        c = getc (stdin);
        fflush (stdin);
        c = toupper (c);

        if (c=='S' || c=='R' || c=='T' || c=='I')
        {
            *Tractor_Hitch_Type = c;
            done = TRUE;
        }
        else
        {
            printf("\a");
            done = FALSE;
        }
    }
}
```

```

    } while (done != TRUE);
}

/*BTM*****
/* Tractor Hitch Swing Point Input */
void Hitch_Swing_Point (char *Tractor_Hitch_Swing_Point)
{
    char c;
    Boolean done;
    done = FALSE;

    do
    {
        clrscr ();
        fflush (stdin);

        printf("\n\n\n\n\n\n  Tractor Hitch Swing Point.\n\n\n\n\n\n");

        printf("  [N] NOT APPLICABLE to this Tractor\n");
        printf("  [A] AHEAD of the Front Axle\n");
        printf("  [B] BEHIND the Rear Axle\n");
        printf("  [C] BEHIND the Front Axle\n");
        printf("  [D] AHEAD of the Rear Axle\n");
        printf("        of the tractor.\n\n\n\n\n\n");

        printf("  Enter the letter corresponding to the Tractor Hitch Swing Point.\n\n");
        c = getc (stdin);
        fflush (stdin);
        c = toupper (c);

        if (c=='A' || c=='B' || c=='C' || c=='D' || c=='N')
        {
            *Tractor_Hitch_Swing_Point = c;
            done = TRUE;
        }
    }
}

```

```

    }
else
    {
        printf("\a");
        done = FALSE;
    }

} while (done != TRUE);
}

/*BTM*****
/* Tractor Hitch Location Input */
void Hitch_Location (char *Tractor_Hitch_Location)
{
    char c;
    Boolean done;
    done = FALSE;

do
{
    clrscr ();
    fflush (stdin);

    printf("\n\n\n\n\n\n  Tractor Hitch Location.\n\n\n\n\n\n");

    printf("  Hitch is located at the\n");
    printf("  [F] FRONT\n");
    printf("  [C] CENTER\n");
    printf("  [R] REAR\n");
    printf("  of the Tractor.\n\n\n\n\n\n");

    printf("  Enter the letter corresponding to the Tractor Hitch
        Location.\n\n");

```

```

c = getc (stdin);
fflush (stdin);
c = toupper (c);

if (c=='F' || c=='C' || c=='R')
{
    *Tractor_Hitch_Location = c;
    done = TRUE;
}
else
{
    printf("\a");
    done = FALSE;
}

} while (done != TRUE);
}

/*BTM******/
/* Tractor Wheelbase Input */
void Wheelbase (double *Tractor_Wheelbase)
{
    clrscr ();

    printf("\n\n\n\n\n Enter the length of the Tractor Wheelbase.\n\n");

    Double_Input (Tractor_Wheelbase);
}

/*BTM******/
/* Tractor Drawbar Length Input */
void Drawbar_Length (double *Tractor_Drawbar_Length)

```

```

{
    clrscr ();

    printf("\n\n\n\n\n\n  Enter the length of the Tractor Hitch.\n\n");

    Double_Input (Tractor_Drawbar_Length);
}

/*BTM*****
/* Tractor Hitch Swing Distance Input */
void Hitch_Swing_Distance (double *Tractor_Hitch_Swing_Distance)
{
    clrscr ();

    printf("\n\n\n\n\n\n  Enter the length of the swing distance \n");
    printf("    of the tractor hitch. (The distance between the \n");
    printf("    center of the axle and the pivot point of the hitch).\n");

    Double_Input (Tractor_Hitch_Swing_Distance);
}

/*BTM*****
/* Tractor Maximum Steering Angle Input */
void Max_Steering_Angle (double *Tractor_Max_Steering_Angle)
{
    clrscr ();

    printf("\n\n\n\n\n\n  Enter the maximum steering angle of the ");
    printf("tractor (in degrees). \n\n");

    Double_Input (Tractor_Max_Steering_Angle);
}

/*BTM*****

```

```

/* Tractor Hitch Swing Angle Input */
void Max_Hitch_Swing_Angle (double *Tractor_Max_Hitch_Swing_Angle)
{
    double Tractor_Hitch_Ref_Dist;
    double Tractor_Swing_Ref_Dist;

    clrscr ();

    printf("\n\n\n\n\n    Enter the distance between the center of the\n");
    printf("    hitch support member and the pivot point of the tractor hitch.\n");
    printf("    Then enter the perpendicular distance from the tractor
           centerline to\n");
    printf("    the centerline of the hitch (at the support) when it is at its\n");
    printf("    maximum swing angle on the tractor.\n");

    Double_Input (&Tractor_Hitch_Ref_Dist);

    Double_Input (&Tractor_Swing_Ref_Dist);

    *Tractor_Max_Hitch_Swing_Angle =
        ((asin(Tractor_Swing_Ref_Dist / Tractor_Hitch_Ref_Dist))*360/(2*pi));
}

/*BTM******/
/* Open Tractor Configuration File */
void Open_Tractor_Config_File
(char File_Type, char File_Mode [],
char *Tractor_Steering_Type,
char *Tractor_Hitch_Type,
char *Tractor_Hitch_Swing_Point,
char *Tractor_Hitch_Location,
double *Tractor_Wheelbase,
double *Tractor_Drawbar_Length,
double *Tractor_Hitch_Swing_Distance,

```

```
double *Tractor_Max_Hitch_Swing_Angle,
double *Tractor_Max_Steering_Angle)
```

```
{
FILE *File_Pointer;
char FILENAME [25];
char AUX_FILENAME [25];
char Dummy_Character;

printf("\n\n\n\n\n\n  Enter the FILENAME of the Tractor Configuration.\n");

Open_File (&File_Pointer, AUX_FILENAME, File_Type, File_Mode,
           FILENAME);

if (!strcmp (File_Mode,"w"))
{
    fprintf (File_Pointer,"%c\n",*Tractor_Steering_Type);
    fprintf (File_Pointer,"%c\n",*Tractor_Hitch_Type);
    fprintf (File_Pointer,"%c\n",*Tractor_Hitch_Swing_Point);
    fprintf (File_Pointer,"%c\n",*Tractor_Hitch_Location);

    fprintf (File_Pointer,"%f\n",*Tractor_Wheelbase);
    fprintf (File_Pointer,"%f\n",*Tractor_Drawbar_Length);
    fprintf (File_Pointer,"%f\n",*Tractor_Hitch_Swing_Distance);
    fprintf (File_Pointer,"%f\n",*Tractor_Max_Hitch_Swing_Angle);
    fprintf (File_Pointer,"%f\n",*Tractor_Max_Steering_Angle);

    clrscr ();
    printf("\n\n\n\n\n\n  Tractor Configuration File saved\n");
    delay (Standard_Time_Delay/2);
}
else if ((!strcmp (File_Mode,"r")) || (!strcmp (File_Mode,"r+")))
{
    fscanf (File_Pointer,"%c",Tractor_Steering_Type);
```


APPENDIX F

IMPLEMENT CONFIGURATION MODULE [IMPLEMEN.C]

```
/* Implement Configuration Module [IMPLEMENT.C] */
```

```
#include "gpsheade.h"
```

```
/*BTM******/
```

```
/* Implement / GPS Receiver Configuration (Procedure) */
```

```
void Configure_Implement_GPS ()
```

```
{
```

```
char c;
```

```
char File_Type;
```

```
char File_Mode [5];
```

```
Boolean done = FALSE;
```

```
double Implement_Hitch_Length;
```

```
double Implement_Working_Width;
```

```
double GPS_Receiver_Height;
```

```
double GPS_Forward_Backward_Dist;
```

```
double GPS_Left_Right_Dist;
```

```
char GPS_Forward_Backward_Dir;
```

```
char GPS_Left_Right_Dir;
```

```
do
```

```
{
```

```
fflush (stdin);
```

```
clrscr ();
```

```
/* Main Implement / GPS Receiver Configuration File Menu */
```

```
printf("\n\n\n\n\n Implement / GPS Receiver Configuration Menu.\n\n");
```

```
printf(" [C] CREATE (Make an Implement / GPS Receiver Configuration File)\n");
```

```

printf(" [M] MODIFY (Change information in an Implement /)\n");
printf("          (GPS Receiver Configuration File)\n");
printf(" [V] VIEW  (Examine an existing Implement /)\n");
printf("          (GPS Receiver Configuration File)\n");
printf(" [E] EXIT  (Return to the Main Menu)\n\n");

```

```

printf(" Press the letter corresponding to the operation you want to do.\n\n");

```

```

c = getc (stdin);
fflush (stdin);
c = toupper (c);

```

```

switch (c)

```

```
{
```

```
  case 'C':
```

```
    Create_Implement_GPS_Configuration ();
```

```
    done = TRUE;
```

```
    break;
```

```
  case 'M':
```

```
    File_Type = 'I';
```

```
    sprintf(File_Mode,"r+");
```

```

    Open_Implement_GPS_Config_File (File_Type, File_Mode,
                                     &Implement_Hitch_Length,
                                     &Implement_Working_Width,
                                     &GPS_Receiver_Height,
                                     &GPS_Forward_Backward_Dist,
                                     &GPS_Left_Right_Dist,
                                     &GPS_Forward_Backward_Dir,
                                     &GPS_Left_Right_Dir);

```

```

clrscr ();

```

```

    Check_Implement_GPS_Configuration (&Implement_Hitch_Length,

```

```
        &Implement_Working_Width,  
        &GPS_Receiver_Height,  
        &GPS_Forward_Backward_Dist,  
        &GPS_Left_Right_Dist,  
        &GPS_Forward_Backward_Dir,  
        &GPS_Left_Right_Dir);  
  
    clrscr ();  
    done = TRUE;  
    break;  
  
case 'V':  
    clrscr ();  
  
    File_Type = 'l';  
    sprintf(File_Mode,"r");  
  
    Open_Implement_GPS_Config_File (File_Type, File_Mode,  
        &Implement_Hitch_Length,  
        &Implement_Working_Width,  
        &GPS_Receiver_Height,  
        &GPS_Forward_Backward_Dist,  
        &GPS_Left_Right_Dist,  
        &GPS_Forward_Backward_Dir,  
        &GPS_Left_Right_Dir);  
  
    Implement_GPS_File_Info (Implement_Hitch_Length,  
        Implement_Working_Width,  
        GPS_Receiver_Height,  
        GPS_Forward_Backward_Dist,  
        GPS_Left_Right_Dist,  
        GPS_Forward_Backward_Dir,  
        GPS_Left_Right_Dir);  
  
    printf("\n\n\n");  
    Continue_Program ();
```

```

        done = TRUE;
        break;

    case 'E':
        done = TRUE;
        break;

    default:
        printf("\a");
        break;
    }
} while (!done);
}

/*BTM******/
/* Create Implement / GPS Receiver Configuration (Procedure) */
void Create_Implement_GPS_Configuration ()
{

    double Implement_Hitch_Length;
    double Implement_Working_Width;
    double GPS_Receiver_Height;
    double GPS_Forward_Backward_Dist;
    double GPS_Left_Right_Dist;

    char GPS_Forward_Backward_Dir;
    char GPS_Left_Right_Dir;

    fflush(stdin);
    clrscr();

    printf("\n\n\n\n\n Create Implement / GPS Receiver
            Configuration\n\n");

```

```

Hitch_Length      (&Implement_Hitch_Length);
Working_Width     (&Implement_Working_Width);
Receiver_Height   (&GPS_Receiver_Height);

Forward_Backward_Dir (&GPS_Forward_Backward_Dir);
Left_Right_Dir     (&GPS_Left_Right_Dir);

if (GPS_Forward_Backward_Dir != 'C')

    Forward_Backward_Dist (&GPS_Forward_Backward_Dist);

if (GPS_Left_Right_Dir != 'C')

    Left_Right_Dist      (&GPS_Left_Right_Dist);

Check_Implement_GPS_Configuration (&Implement_Hitch_Length,
                                   &Implement_Working_Width,
                                   &GPS_Receiver_Height,
                                   &GPS_Forward_Backward_Dist,
                                   &GPS_Left_Right_Dist,
                                   &GPS_Forward_Backward_Dir,
                                   &GPS_Left_Right_Dir);

}

/*BTM*****
/* Check Implement / GPS Receiver Configuration File */
void Check_Implement_GPS_Configuration
(double *Implement_Hitch_Length,
 double *Implement_Working_Width,
 double *GPS_Receiver_Height,
 double *GPS_Forward_Backward_Dist,
 double *GPS_Left_Right_Dist,
 char *GPS_Forward_Backward_Dir,
 char *GPS_Left_Right_Dir)

```

```
{
char c;
char File_Type;
char File_Mode [5];
Boolean done = FALSE;

fflush (stdin);

do
{
    Implement_GPS_File_Info (*Implement_Hitch_Length,
                             *Implement_Working_Width,
                             *GPS_Receiver_Height,
                             *GPS_Forward_Backward_Dist,
                             *GPS_Left_Right_Dist,
                             *GPS_Forward_Backward_Dir,
                             *GPS_Left_Right_Dir);

    printf("\n\n Is all information correct? ");
    printf(" Press\n [Y] if YES,\n [N] if NO, or\n [E] to RETURN to ");
    printf("the Implement / GPS Receiver Configuration\n");
    printf(" Menu without saving file.\n\n");

    c = getc (stdin);
    fflush (stdin);
    c = toupper (c);

    switch (c)
    {
        case 'Y':
            File_Type = 'I';
            sprintf(File_Mode,"w");

            Open_Implement_GPS_Config_File (File_Type, File_Mode,
```

```

        Implement_Hitch_Length,
        Implement_Working_Width,
        GPS_Receiver_Height,
        GPS_Forward_Backward_Dist,
        GPS_Left_Right_Dist,
        GPS_Forward_Backward_Dir,
        GPS_Left_Right_Dir);

        done = TRUE;
        break;

    case 'N':
        Modify_Implement_GPS_Configuration (Implement_Hitch_Length,
            Implement_Working_Width,
            GPS_Receiver_Height,
            GPS_Forward_Backward_Dist,
            GPS_Left_Right_Dist,
            GPS_Forward_Backward_Dir,
            GPS_Left_Right_Dir);

        break;

    case 'E':
        done = TRUE;
        break;

    default:
        fflush (stdin);
        printf("\a");
        break;

    }
} while (!done);
}

/*BTM*****
/* Implement / GPS Receiver File Information */

```

```

void Implement_GPS_File_Info
(double Implement_Hitch_Length,
 double Implement_Working_Width,
 double GPS_Receiver_Height,
 double GPS_Forward_Backward_Dist,
 double GPS_Left_Right_Dist,
 char  GPS_Forward_Backward_Dir,
 char  GPS_Left_Right_Dir)

{
    clrscr ();

        /* Display Implement / GPS Receiver Configuration
File Information */

    printf("\n\n\n\n\n The current Implement / GPS Receiver
Configuration File is as follows:\n\n");

    printf(" Implement Hitch Length:      %f (meters).\n",Implement_Hitch_Length);
    printf(" Implement Working Width:      %f (meters).\n",Implement_Working_Width);
    printf(" GPS Receiver Height:          %f (meters).\n",GPS_Receiver_Height);

    switch (GPS_Forward_Backward_Dir)
    {
        case 'F':
            printf(" The GPS Receiver is %f meter FORWARD of the
Centerline.\n",
GPS_Forward_Backward_Dist);
            break;

        case 'B':
            printf(" The GPS Reciever is %f meter REARWARD of
the Centerline.\n",
GPS_Forward_Backward_Dist);
            break;
    }
}

```

```

case 'C':
    printf(" The GPS Receiver CENTERED (fore/aft) on the
           Implement\n");
    break;

default:
    printf("\a");
    printf("\n There is an error in the Implement / GPS
           Receiver Configuration File.\n");
    printf(" The GPS Receiver Location is invalid.\n");
    break;
}

switch (GPS_Left_Right_Dir)
{
case 'L':
    printf(" The GPS Receiver is %f meter LEFT of the
           Centerline.\n",
           GPS_Left_Right_Dist);
    break;

case 'R':
    printf(" The GPS Reciever is %f meter RIGHT of the
           Centerline.\n",
           GPS_Left_Right_Dist);
    break;

case 'C':
    printf(" The GPS Receiver is CENTERED (left/right) on the
           Implement");
    break;

default:
    printf("\a");

```

```

printf("\n  There is an error in the Implement / GPS
          Receiver Configuration File.\n");
printf("  The GPS Receiver Location is invalid.\n");
break;
}
}

/*BTM*****
/* Modify Implement / GPS Receiver Configuration (Procedure) */
void Modify_Implement_GPS_Configuration
(double *Implement_Hitch_Length,
 double *Implement_Working_Width,
 double *GPS_Receiver_Height,
 double *GPS_Forward_Backward_Dist,
 double *GPS_Left_Right_Dist,
 char *GPS_Forward_Backward_Dir,
 char *GPS_Left_Right_Dir)
{
char c;
Boolean done = FALSE;

do
{
fflush (stdin);
clrscr ();

/* Modify Implement / GPS Receiver Configuration
Menu */

printf("\n\n\n\n\n Enter the letter corresponding to the\n");
printf(" Implement / GPS Receiver Configuration File\n");
printf(" item you want to change\n\n");
printf(" [L] Implement Hitch Length\n");
printf(" [W] Implement Working Width\n");

```

```
printf(" [H] GPS Receiver Height above ground level\n");
printf(" [O] GPS Receiver Forward / Rearward Distance\n");
printf(" [T] GPS Receiver Left / Right Distance\n");
printf(" [F] GPS Receiver Forward / Rearward Direction\n");
printf(" [R] GPS Receiver Left / Right Direction\n");
printf(" [E] Return to menu.\n\n");
```

```
c = getc (stdin);
fflush (stdin);
c = toupper (c);
```

```
switch (c)
{
    case 'L':
        Hitch_Length (Implement_Hitch_Length);
        break;

    case 'W':
        Working_Width (Implement_Working_Width);
        break;

    case 'H':
        Receiver_Height (GPS_Receiver_Height);
        break;

    case 'O':
        Forward_Backward_Dist (GPS_Forward_Backward_Dist);
        break;

    case 'T':
        Left_Right_Dist (GPS_Left_Right_Dist);
        break;

    case 'F':
        Forward_Backward_Dir (GPS_Forward_Backward_Dir);
```

```

        break;

    case 'R':
        Left_Right_Dir (GPS_Left_Right_Dir);
        break;

    case 'E':
        done = TRUE;
        fflush (stdin);
        break;

    default:
        printf("\a");
        break;
}
} while (!done);
}

/*BTM*****
/* Implement Hitch Length Input */
void Hitch_Length (double *Implement_Hitch_Length)
{
    Boolean done;
    done = FALSE;
    do
    {
        clrscr ();

        printf("\n\n\n\n\n\n    Enter the Hitch Length of the Implement.\n\n");

        Double_Input (Implement_Hitch_Length);

        if (*Implement_Hitch_Length >= Max_Impl_Hitch_Length)
        {

```

```

printf("\a The implement hitch length specified is too long.\n");
printf(" It must be less than or equal to %f meters long.
      \n",Max_Impl_Hitch_Length);
done = FALSE;
Continue_Program ();
}

else if (*Implement_Hitch_Length <= zero)
{
printf("\a ERROR: The implement hitch length specified ");
printf("is less than or equal to zero.\n");
done = FALSE;
Continue_Program ();
}

else
{
done = TRUE;
}

}while (!done);

}

/*BTM*****
/* Implement Working Width Input */
void Working_Width (double *Implement_Working_Width)
{
Boolean done;
done = FALSE;
do
{
clrscr ();

printf("\n\n\n\n\n\n Enter the Working Width of the Implement.\n\n");

```

```

Double_Input (Implement_Working_Width);

if (*Implement_Working_Width >= Max_Impl_Width)
{
    printf("\a ERROR: The implement width specified is too wide.\n");
    printf(" It must be less than or equal to %f meters
        wide.\n",Max_Impl_Width);
    done = FALSE;
    Continue_Program ();
}

else if (*Implement_Working_Width <= zero)
{
    printf("\a The implement width specified ");
    printf("is less than or equal to zero.\n");
    done = FALSE;
    Continue_Program ();
}

else
{
    done = TRUE;
}

}while (!done);

}

/*BTM******/
/* GPS Receiver Mounting Height */
void Receiver_Height (double *GPS_Receiver_Height)
{
    clrscr ();
}

```

```
printf("\n\n\n\n\n\n Enter the Height of the GPS Receiver above the\n");
printf(" working plane of the Implement.\n");
```

```
Double_Input (GPS_Receiver_Height);
}
```

```
/*BTM*****
```

```
/* GPS Receiver Forward / Rearward Mounting Distance */
```

```
void Forward_Backward_Dist (double *GPS_Forward_Backward_Dist)
```

```
{
```

```
Boolean done;
```

```
done = FALSE;
```

```
do
```

```
{
```

```
clrscr ();
```

```
printf("\n\n\n\n\n\n Enter the distance from the center of the Implement\n");
```

```
printf(" forward or backward to the center of the GPS Receiver.\n");
```

```
printf(" This distance should be less than 1 meter (geometric reasons)\n");
```

```
Double_Input (GPS_Forward_Backward_Dist);
```

```
if (*GPS_Forward_Backward_Dist >= GPS_Recvr_Loctn)
```

```
{
```

```
printf(" This distance should be less than 1 meter (geometric reasons)\n");
```

```
done = FALSE;
```

```
Continue_Program ();
```

```
}
```

```
else if (*GPS_Forward_Backward_Dist <= zero)
```

```
{
```

```
printf("\a The specified distance ");
```

```
printf("is less than or equal to zero.\n");
```

```

done = FALSE;
Continue_Program ();
}

else
{
done = TRUE;
}

}while (!done);
}

/*BTM******/
/* GPS Receiver Left / Right Mounting Distance */
void Left_Right_Dist (double *GPS_Left_Right_Dist)
{
Boolean done;
done = FALSE;

do
{
clrscr ();

printf("\n\n\n\n\n Enter the distance from the center of the Implement\n");
printf(" left or right to the center of the GPS Receiver.\n");

printf(" This distance should be less than 1 meter (geometric reasons)\n");

Double_Input (GPS_Left_Right_Dist);

if (*GPS_Left_Right_Dist >= GPS_Recvr_Loctn)
{
printf(" This distance should be less than 1. meter (geometric reasons)\n");
done = FALSE;
Continue_Program ();
}
}
}

```

```

    }

else if (*GPS_Left_Right_Dist <= zero)
{
    printf("\a The specified distance ");
    printf("is less than or equal to zero.\n");
    done = FALSE;
    Continue_Program ();
}

else
{
    done = TRUE;
}

}while (!done);
}

/*BTM******/
/* GPS Receiver Forward / Rearward Mounting Direction */
void Forward_Backward_Dir (char *GPS_Forward_Backward_Dir)
{
    char c;
    Boolean done;
    done = FALSE;

    do
    {
        clrscr ();
        fflush (stdin);

        printf("\n\n\n\n\n\n  GPS Receiver Forward / Rearward
                Direction.\n\n\n\n\n\n");

        printf(" [F] FORWARD.\n");
    }
}

```

```
printf(" [C] CENTERLINE.\n");
printf(" [B] REARWARD.\n\n\n\n\n");
```

```
printf(" Enter the letter corresponding to the location of the GPS\n");
printf(" Receiver relative to the center of the implement working area.\n\n");
```

```
c = getc (stdin);
fflush (stdin);
c = toupper (c);
```

```
if (c=='F' || c=='B' || c=='C')
{
    *GPS_Forward_Backward_Dir = c;
    done = TRUE;
}
else
{
    printf("\a");
    done = FALSE;
}
```

```
} while (done != TRUE);
```

```
}
```

```
/*BTM*****
```

```
/* GPS Receiver Left / Right Mounting Direction */
```

```
void Left_Right_Dir (char *GPS_Left_Right_Dir)
```

```
{
```

```
char c;
Boolean done;
done = FALSE;
```

```
do
```

```
{
```

```

clrscr ();
fflush (stdin);

printf("\n\n\n\n\n\n  GPS Receiver Left / Right
      Direction.\n\n\n\n\n\n");

printf("  [L] LEFT.\n");
printf("  [C] CENTERLINE.\n");
printf("  [R] RIGHT.\n\n\n\n\n\n");

printf("  Enter the letter corresponding to the location of the GPS\n");
printf("  Receiver relative to the center of the implement working area.\n\n");

c = getc (stdin);
fflush (stdin);
c = toupper (c);

if (c=='L' || c=='R' || c=='C')
{
  *GPS_Left_Right_Dir = c;
  done = TRUE;
}
else
{
  printf("\a");
  done = FALSE;
}

} while (done != TRUE);

}

/*BTM*****
/* Open Implement GPS Reciever Configuration File */
void Open_Implement_GPS_Config_File

```

```
(char File_Type, char File_Mode [],
double *Implement_Hitch_Length,
double *Implement_Working_Width,
double *GPS_Receiver_Height,
double *GPS_Forward_Backward_Dist,
double *GPS_Left_Right_Dist,
char *GPS_Forward_Backward_Dir,
char *GPS_Left_Right_Dir)
```

```
{
FILE *File_Pointer;
char FILENAME [25];
char AUX_FILENAME [25];
char Dummy_Character;

printf("\n\n\n\n\n  Enter the FILENAME of the Implement /\n");
printf("  GPS Receiver Configuration.\n");

if (*GPS_Forward_Backward_Dir == 'C')
    *GPS_Forward_Backward_Dist = 0.0;

if (*GPS_Left_Right_Dir == 'C')
    *GPS_Left_Right_Dist = 0.0;

Open_File (&File_Pointer, AUX_FILENAME, File_Type, File_Mode,
FILENAME);

if (!strcmp (File_Mode,"w"))
{
    fprintf (File_Pointer,"%f\n",*Implement_Hitch_Length);
    fprintf (File_Pointer,"%f\n",*Implement_Working_Width);
    fprintf (File_Pointer,"%f\n",*GPS_Receiver_Height);
    fprintf (File_Pointer,"%f\n",*GPS_Forward_Backward_Dist);
    fprintf (File_Pointer,"%f\n",*GPS_Left_Right_Dist);
    fprintf (File_Pointer,"%c\n",*GPS_Forward_Backward_Dir);
}
```

```

fprintf (File_Pointer,"%c\n",*GPS_Left_Right_Dir);

clrscr ();
printf("\n\n\n\n\n Implement / GPS Receiver
      Configuration File saved\n");
delay (Standard_Time_Delay/2);

}
else if ((!strcmp (File_Mode,"r")) || (!strcmp (File_Mode,"r+")))
{
fscanf (File_Pointer,"%lf",Implement_Hitch_Length);
fscanf (File_Pointer,"%lf",Implement_Working_Width);
fscanf (File_Pointer,"%lf",GPS_Receiver_Height);
fscanf (File_Pointer,"%lf",GPS_Forward_Backward_Dist);
fscanf (File_Pointer,"%lf",GPS_Left_Right_Dist);

fscanf (File_Pointer,"%c",&Dummy_Character);
fscanf (File_Pointer,"%c", GPS_Forward_Backward_Dir);
fscanf (File_Pointer,"%c",&Dummy_Character);
fscanf (File_Pointer,"%c", GPS_Left_Right_Dir);
}
else
{
clrscr ();
printf("\n\n\n\n\n\ a ERROR.\n");
}

fclose (File_Pointer);
printf("\n The file %s has been closed.\n",AUX_FILENAME);
delay (Standard_Time_Delay/2);

}

/*BTM*****

```

APPENDIX G

FIELD STATUS CONFIGURATION MODULE [FLDSTATS.C]

```
/* Field Status Configuration Module [FLDSTATS.C] */
```

```
#include "gpsheade.h"
```

```
/*BTM*****
```

```
/* Configure Field Information */
```

```
void Configure_Field_Status_Info ()
```

```
{
```

```
char c;
```

```
char File_Type;
```

```
char File_Mode [5];
```

```
Boolean done = FALSE;
```

```
double BasePoint_Degrees_Latitude;
```

```
double BasePoint_Minutes_Latitude;
```

```
double BasePoint_Seconds_Latitude;
```

```
double BasePoint_Degrees_Longitude;
```

```
double BasePoint_Minutes_Longitude;
```

```
double BasePoint_Seconds_Longitude;
```

```
double Startpoint_X_Coord;
```

```
double Startpoint_Y_Coord;
```

```
double Endpoint_X_Coord;
```

```
double Endpoint_Y_Coord;
```

```
int Field_Travel_StartEntity;
```

```
int Field_Travel_EndEntity;
```

```
do
```

```
{
```

```
fflush (stdin);
```

```
clrscr ();
```

```
/* Main Field Status Configuration Menu */
```



```
&BasePoint_Seconds_Longitude,  
&Startpoint_X_Coord,  
&Startpoint_Y_Coord,  
&Endpoint_X_Coord,  
&Endpoint_Y_Coord,  
&Field_Travel_StartEntity,  
&Field_Travel_EndEntity);
```

```
Field_Status_Info (BasePoint_Degrees_Latitude,  
BasePoint_Minutes_Latitude,  
BasePoint_Seconds_Latitude,  
BasePoint_Degrees_Longitude,  
BasePoint_Minutes_Longitude,  
BasePoint_Seconds_Longitude,  
Startpoint_X_Coord,  
Startpoint_Y_Coord,  
Endpoint_X_Coord,  
Endpoint_Y_Coord,  
Field_Travel_StartEntity,  
Field_Travel_EndEntity);
```

```
Continue_Program ();
```

```
done = TRUE;  
break;
```

```
case 'E':  
done = TRUE;  
break;
```

```
default:  
printf("\a");  
break;
```

```
}
```

```

} while (!done);

}

/*BTM*****
/* Open Field Status Information */
void Open_Field_Status_Info
(char File_Type, char File_Mode [],
double *BasePoint_Degrees_Latitude,
double *BasePoint_Minutes_Latitude,
double *BasePoint_Seconds_Latitude,
double *BasePoint_Degrees_Longitude,
double *BasePoint_Minutes_Longitude,
double *BasePoint_Seconds_Longitude,
double *Startpoint_X_Coord,
double *Startpoint_Y_Coord,
double *Endpoint_X_Coord,
double *Endpoint_Y_Coord,
int *Field_Travel_StartEntity,
int *Field_Travel_EndEntity)
{
    FILE *File_Pointer;
    char FILENAME [25];
    char AUX_FILENAME [25];

    printf("\n\n\n\n\n\n  Enter the FILENAME of the Field Status Info
file.\n");

    Open_File (&File_Pointer, AUX_FILENAME, File_Type, File_Mode,
FILENAME);

    if (!strcmp (File_Mode,"w"))
    {
        fprintf (File_Pointer,"%f\n",*BasePoint_Degrees_Latitude);
        fprintf (File_Pointer,"%f\n",*BasePoint_Minutes_Latitude);

```



```

*BasePoint_Seconds_Latitude,
*BasePoint_Degrees_Longitude,
*BasePoint_Minutes_Longitude,
*BasePoint_Seconds_Longitude,
*Startpoint_X_Coord,
*Startpoint_Y_Coord,
*Endpoint_X_Coord,
*Endpoint_Y_Coord,
*Field_Travel_StartEntity,
*Field_Travel_EndEntity);

```

```

printf("\n\n Is all information correct?\n");
printf(" Press\n [Y] if YES, to save the file \n [N] if NO, or\n
[E] to RETURN to ");
printf("the Field Status Info Menu without saving file.\n\n");

```

```

c = getc (stdin);
fflush (stdin);
c = toupper (c);

```

```

switch (c)

```

```

{

```

```

  case 'Y':

```

```

    File_Type = 'S';

```

```

    sprintf(File_Mode,"w");

```

```

    Open_Field_Status_Info (File_Type, File_Mode,
        BasePoint_Degrees_Latitude,
        BasePoint_Minutes_Latitude,
        BasePoint_Seconds_Latitude,
        BasePoint_Degrees_Longitude,
        BasePoint_Minutes_Longitude,
        BasePoint_Seconds_Longitude,
        Startpoint_X_Coord,
        Startpoint_Y_Coord,

```

```
Endpoint_X_Coord,  
Endpoint_Y_Coord,  
Field_Travel_StartEntity,  
Field_Travel_EndEntity);  
  
done = TRUE;  
break;  
  
case 'N':  
do  
{  
    Modify_Field_Status_Info (BasePoint_Degrees_Latitude,  
                               BasePoint_Minutes_Latitude,  
                               BasePoint_Seconds_Latitude,  
                               BasePoint_Degrees_Longitude,  
                               BasePoint_Minutes_Longitude,  
                               BasePoint_Seconds_Longitude,  
                               Startpoint_X_Coord,  
                               Startpoint_Y_Coord,  
                               Endpoint_X_Coord,  
                               Endpoint_Y_Coord,  
                               Field_Travel_StartEntity,  
                               Field_Travel_EndEntity);  
  
    if (*Field_Travel_EndEntity < *Field_Travel_StartEntity)  
    {  
        printf("    The field travel end entity number must be larger than\n");  
        printf("    the field travel start entity number \n");  
        delay (Standard_Time_Delay/2);  
        ok = FALSE;  
    }  
    else  
        ok = TRUE;  
  
} while (ok == FALSE);
```

```

        break;

    case 'E':
        done = TRUE;
        break;

    default:
        fflush (stdin);
        printf("\a");
        break;
}
} while (!done);
}

/*BTM*****
/* Create Field Status Information */
void Create_Field_Status_Info ()
{
    double BasePoint_Degrees_Latitude;
    double BasePoint_Minutes_Latitude;
    double BasePoint_Seconds_Latitude;
    double BasePoint_Degrees_Longitude;
    double BasePoint_Minutes_Longitude;
    double BasePoint_Seconds_Longitude;
    double Startpoint_X_Coord;
    double Startpoint_Y_Coord;
    double Endpoint_X_Coord;
    double Endpoint_Y_Coord;
    int Field_Travel_StartEntity;
    int Field_Travel_EndEntity;
    Boolean done = FALSE;

    fflush(stdin);
    clrscr();

```

```

printf("\n\n\n\n\n Create Field Status Information \n\n\n\n\n");

/* Call Field Status Info Input Functions */

Degrees_Latitude (&BasePoint_Degrees_Latitude);
Minutes_Latitude (&BasePoint_Minutes_Latitude);
Seconds_Latitude (&BasePoint_Seconds_Latitude);

Degrees_Longitude (&BasePoint_Degrees_Longitude);
Minutes_Longitude (&BasePoint_Minutes_Longitude);
Seconds_Longitude (&BasePoint_Seconds_Longitude);

Startpoint_X (&Startpoint_X_Coord);
Startpoint_Y (&Startpoint_Y_Coord);
Endpoint_X (&Endpoint_X_Coord);
Endpoint_Y (&Endpoint_Y_Coord);

do
{
Travel_StartEntity (&Field_Travel_StartEntity);
Travel_EndEntity (&Field_Travel_EndEntity);

if (Field_Travel_EndEntity < Field_Travel_StartEntity)
{
printf(" The field travel end entity number must be larger than\n");
printf(" the field travel start entity number \n");
delay (Standard_Time_Delay/2);
done = FALSE;
}
else
done = TRUE;

}while (done == FALSE);

```

```

Check_Field_Status_Info (&BasePoint_Degrees_Latitude,
                        &BasePoint_Minutes_Latitude,
                        &BasePoint_Seconds_Latitude,
                        &BasePoint_Degrees_Longitude,
                        &BasePoint_Minutes_Longitude,
                        &BasePoint_Seconds_Longitude,
                        &Startpoint_X_Coord,
                        &Startpoint_Y_Coord,
                        &Endpoint_X_Coord,
                        &Endpoint_Y_Coord,
                        &Field_Travel_StartEntity,
                        &Field_Travel_EndEntity);

```

```

}

```

```

/*BTM*****

```

```

/* Modify Field Status Information */

```

```

void Modify_Field_Status_Info
(double *BasePoint_Degrees_Latitude,
 double *BasePoint_Minutes_Latitude,
 double *BasePoint_Seconds_Latitude,
 double *BasePoint_Degrees_Longitude,
 double *BasePoint_Minutes_Longitude,
 double *BasePoint_Seconds_Longitude;
 double *Startpoint_X_Coord,
 double *Startpoint_Y_Coord,
 double *Endpoint_X_Coord,
 double *Endpoint_Y_Coord,
 int *Field_Travel_StartEntity,
 int *Field_Travel_EndEntity)

```

```

{
  char c;
  Boolean done = FALSE;

```

```
do
{
    fflush (stdin);
    clrscr ();

        /* Modify Field Status Information Menu */
    printf("\n\n\n\n\n  Modify Field Status Information.\n\n\n\n\n");

    printf(" [A] Base Point Latitude (Degrees) \n");
    printf(" [B] Base Point Latitude (Minutes) \n");
    printf(" [C] Base Point Latitude (Seconds) \n");

    printf(" [D] Base Point Longitude (Degrees) \n");
    printf(" [E] Base Point Longitude (Minutes) \n");
    printf(" [F] Base Point Longitude (Seconds) \n");

    printf(" [G] Farm Field Travel Startpoint X Coordinate \n");
    printf(" [H] Farm Field Travel Startpoint Y Coordinate \n");

    printf(" [I] Farm Field Travel Endpoint X Coordinate \n");
    printf(" [J] Farm Field Travel Endpoint Y Coordinate \n");

    printf(" [K] Farm Field Travel Start Entity Number \n");
    printf(" [L] Farm Field Travel End Entity Number \n");

    printf(" [R] Return to menu\n\n\n\n\n");

    printf(" Enter the letter corresponding to the\n");
    printf(" Field Status Information File item you want to change\n\n");

    c = getc (stdin);
    fflush (stdin);
    c = toupper (c);
```

```
switch (c)
{
    case 'A':
        Degrees_Latitude (BasePoint_Degrees_Latitude);
        break;

    case 'B':
        Minutes_Latitude (BasePoint_Minutes_Latitude);
        break;

    case 'C':
        Seconds_Latitude (BasePoint_Seconds_Latitude);
        break;

    case 'D':
        Degrees_Longitude (BasePoint_Degrees_Longitude);
        break;

    case 'E':
        Minutes_Longitude (BasePoint_Minutes_Longitude);
        break;

    case 'F':
        Seconds_Longitude (BasePoint_Seconds_Longitude);
        break;

    case 'G':
        Startpoint_X (Startpoint_X_Coord);
        break;

    case 'H':
        Startpoint_Y (Startpoint_Y_Coord);
        break;

    case 'I':
```

```

        Endpoint_X (Endpoint_X_Coord);
        break;

    case 'J':
        Endpoint_Y (Endpoint_Y_Coord);
        break;

    case 'K':
        Travel_StartEntity (Field_Travel_StartEntity);
        break;

    case 'L':
        Travel_EndEntity (Field_Travel_EndEntity);
        break;

    case 'R':
        done = TRUE;
        fflush (stdin);
        break;

    default:
        printf("\a");
        break;
}

} while (done == FALSE);

}

/*BTM*****
/* Field Status Information */
void Field_Status_Info
(double BasePoint_Degrees_Latitude,
double BasePoint_Minutes_Latitude,
double BasePoint_Seconds_Latitude,

```

```
double BasePoint_Degrees_Longitude,
double BasePoint_Minutes_Longitude,
double BasePoint_Seconds_Longitude,
double Startpoint_X_Coord,
double Startpoint_Y_Coord,
double Endpoint_X_Coord,
double Endpoint_Y_Coord,
int Field_Travel_StartEntity,
int Field_Travel_EndEntity)
{
    clrscr ();

    /* Display Field Status Information */

    printf("\n\n\n\n\n The current Field Status Information File is as
follows:\n\n");

    printf(" Base Point Latitude (degrees):   %f \n",
           BasePoint_Degrees_Latitude);
    printf(" Base Point Latitude (minutes):   %f \n",
           BasePoint_Minutes_Latitude);
    printf(" Base Point Latitude (seconds):   %f \n",
           BasePoint_Seconds_Latitude);
    printf(" Base Point Longitude (degrees):   %f \n",
           BasePoint_Degrees_Longitude);
    printf(" Base Point Longitude (minutes):   %f \n",
           BasePoint_Minutes_Longitude);
    printf(" Base Point Longitude (seconds):   %f \n",
           BasePoint_Seconds_Longitude);
    printf(" Farm Field Startpoint X Coordinate: %f \n",
           Startpoint_X_Coord);
    printf(" Farm Field Startpoint Y Coordinate: %f \n",
           Startpoint_Y_Coord);
    printf(" Farm Field Endpoint X Coordinate: %f \n",
           Endpoint_X_Coord);
```

```

printf(" Farm Field Endpoint Y Coordinate: %f \n",
      Endpoint_Y_Coord);
printf(" Farm Field Travel Start Entity Number: %d \n",
      Field_Travel_StartEntity);
printf(" Farm Field Travel End Entity Number: %d \n",
      Field_Travel_EndEntity);
}

/*BTM*****
/* Base Point Degrees Latitude */
void Degrees_Latitude (double *BasePoint_Degrees_Latitude)
{
  clrscr ();

  printf("\n\n\n\n\n  Enter Degrees Latitude of the G.P.S. Basepoint.\n\n");

  Double_Input (BasePoint_Degrees_Latitude);
}

/*BTM*****
/* Base Point Minutes Latitude */
void Minutes_Latitude (double *BasePoint_Minutes_Latitude)
{
  clrscr ();

  printf("\n\n\n\n\n  Enter Minutes Latitude of the G.P.S. Basepoint.\n\n");

  Double_Input (BasePoint_Minutes_Latitude);
}

/*BTM*****
/* Base Point Seconds Latitude */
void Seconds_Latitude (double *BasePoint_Seconds_Latitude)
{

```

```

clrscr ();

printf("\n\n\n\n\n Enter Seconds Latitude of the G.P.S. Basepoint.\n\n");

Double_Input (BasePoint_Seconds_Latitude);
}

/*BTM*****
/* Base Point Degrees Longitude */
void Degrees_Longitude (double *BasePoint_Degrees_Longitude)
{
clrscr ();

printf("\n\n\n\n\n Enter Degrees Longitude of the G.P.S. Basepoint.\n\n");

Double_Input (BasePoint_Degrees_Longitude);
}

/*BTM*****
/* Base Point Minutes Longitude */
void Minutes_Longitude (double *BasePoint_Minutes_Longitude)
{
clrscr ();

printf("\n\n\n\n\n Enter Minutes Longitude of the G.P.S. Basepoint.\n\n");

Double_Input (BasePoint_Minutes_Longitude);
}

/*BTM*****
/* Base Point Seconds Longitude */
void Seconds_Longitude (double *BasePoint_Seconds_Longitude)
{
clrscr ();

```

```

printf("\n\n\n\n\n Enter Seconds Longitude of the G.P.S. Basepoint.\n\n");

Double_Input (BasePoint_Seconds_Longitude);
}

/*BTM*****
/* Field Travel Startpoint X Coordinate */
void Startpoint_X (double *Startpoint_X_Coord)
{
    clrscr ();

    printf("\n\n\n\n\n Enter Farm Field Travel Startpoint X Coordinate.\n\n");

    Double_Input (Startpoint_X_Coord);
}

/*BTM*****
/* Field Travel Startpoint Y Coordinate */
void Startpoint_Y (double *Startpoint_Y_Coord)
{
    clrscr ();

    printf("\n\n\n\n\n Enter Farm Field Travel Startpoint Y Coordinate.\n\n");

    Double_Input (Startpoint_Y_Coord);
}

/*BTM*****
/* Field Travel Endpoint X Coordinate */
void Endpoint_X (double *Endpoint_X_Coord)
{
    clrscr ();

    printf("\n\n\n\n\n Enter Farm Field Travel Endpoint X Coordinate.\n\n");

```

```

    Double_Input (Endpoint_X_Coord);
}

```

```

/*BTM*****
/* Field Travel Endpoint Y Coordinate */
void Endpoint_Y (double *Endpoint_Y_Coord)
{
    clrscr ();

    printf("\n\n\n\n\n Enter Farm Field Travel Endpoint Y Coordinate.\n\n");

    Double_Input (Endpoint_Y_Coord);
}

```

```

/*BTM*****
/* Field Travel StartPoint */
void Travel_StartEntity (int *Field_Travel_StartEntity)
{
    clrscr ();

    printf("\n\n\n\n\n Enter the number of the TRAVEL PATH DATA
entity\n");
    printf(" corresponding to the desired field travel start point.\n\n");

    Integer_Input (Field_Travel_StartEntity);

}

```

```

/*BTM*****
/* Field Travel EndPoint */
void Travel_EndEntity (int *Field_Travel_EndEntity)
{

```

```
clrscr ();
```

```
printf("\n\n\n\n\n Enter the number of the TRAVEL PATH DATA  
entity\n");
```

```
printf(" corresponding to the desired field travel end point.\n\n");
```

```
Integer_Input (Field_Travel_EndEntity);
```

```
}
```

```
/*BTM*****
```

APPENDIX H

FIELD TRAVEL PATH TEST MODULE [TESTFIEL.C]

```
/* Field Travel Path Test Module [TESTFIEL.C] */
```

```
#include "gpsheade.h"
```

```
/*BTM*****
```

```
/* Test Field Travel Path (procedure) */
```

```
void Test_Field_Travel_Path ()
```

```
{
```

```
FILE *IMPL_File_Pointer;
```

```
FILE *DXFile_Pointer;
```

```
char No_File;
```

```
int Error_Num;
```

```
int Error_Warning_Flag = 2;
```

```
int Test_Fail = 2;
```

```
char FILENAME [25];
```

```
char AUX_FILENAME [25];
```

```
char File_Type;
```

```
char File_Mode [5];
```

```
char Data [50];
```

```
char Comment [50];
```

```
int Entity_Scan_Pass_Number =1;
```

```
char DXF_Section_Name [25];
```

```
/**/
```

```
char Next_DXF_Section_Name [25];
```

```
char Next_Layer_Name [25];
```

```
char Next_Linetype_Impl_UpDown [25];
```

```
float Next_CAD_Elevation;
```

```
float Next_Line_Thickness;
```

```
int Next_Entity_Handle;
```

```
int Next_Color;
```

```
float Next_Primary_X_Coord;
```

```
float Next_Primary_Y_Coord;  
float Next_Primary_Z_Coord;  
float Next_Secondary_X_Coord;  
float Next_Secondary_Y_Coord;  
float Next_Secondary_Z_Coord;  
float Next_Arc_Radius;  
float Next_Start_Angle;  
float Next_End_Angle;  
int Next_Group_Code;
```

```
/**/
```

```
char Crnt_DXF_Section_Name [25];  
char Crnt_Layer_Name [25];  
char Crnt_Linetype_Impl_UpDown [50];  
float Crnt_CAD_Elevation;  
float Crnt_Line_Thickness;  
int Crnt_Entity_Handle;  
int Crnt_Color;  
float Crnt_Primary_X_Coord = -1.0;  
float Crnt_Primary_Y_Coord;  
float Crnt_Primary_Z_Coord;  
float Crnt_Secondary_X_Coord;  
float Crnt_Secondary_Y_Coord;  
float Crnt_Secondary_Z_Coord;  
float Crnt_Arc_Radius;  
float Crnt_Start_Angle;  
float Crnt_End_Angle;  
int Crnt_Group_Code;
```

```
/**/
```

```
double Implement_Hitch_Length;  
double Implement_Working_Width;  
  
Boolean Travel_Path_Test_Done = FALSE;
```

```
clrscr();
```

```
printf("\n\n\n\n\n Implement Configuration File information\n");  
printf(" is required. Please type in the FILENAME for the Implement\n");  
printf(" that will be used with this particular Travel Path.\n");  
printf(" The appropriate extension [.IMP]\n");  
printf(" will be appended automatically.\n");
```

```
File_Type = 'I';  
sprintf(File_Mode,"r");
```

```
Open_File (&IMPL_File_Pointer, AUX_FILENAME, File_Type,  
           File_Mode, FILENAME);
```

```
fscanf(IMPL_File_Pointer,"%lf",&Implement_Hitch_Length);  
fscanf(IMPL_File_Pointer,"%lf",&Implement_Working_Width);
```

```
fclose(IMPL_File_Pointer);
```

```
clrscr ();
```

```
printf("\n\n\n\n\n The Implement Working Width is: %f \n",  
       Implement_Working_Width);  
printf(" The above width will be used as the Minimum Turning Radius.\n");
```

```
Continue_Program ();
```

```
clrscr();  
printf("\n\n\n\n\n Travel Path Data File required ");  
printf("is a CAD [DXF] output File.\n");  
printf(" The extension [.DXF] is automatically appended to FILENAME.\n");
```

```
File_Type = 'D';  
sprintf(File_Mode,"r");
```

```
Open_File (&DXFile_Pointer, AUX_FILENAME, File_Type, File_Mode,
FILENAME);
```

```
printf("\n The DXF File %s has been opened successfully.\n",FILENAME);
```

```
delay (Standard_Time_Delay / 2);
```

```
Travel_Path_Test_Done = FALSE;
```

```
No_File = fgetc (DXFile_Pointer);
```

```
if (No_File == EOF)
```

```
{
```

```
fclose(DXFile_Pointer);
```

```
Error_Num = 102;
```

```
General_Error (Error_Num);
```

```
}
```

```
do
```

```
{
```

```
fscanf(DXFile_Pointer,"%s",Next_DXF_Section_Name);
```

```
if (!strcmp(Next_DXF_Section_Name,"HEADER"))
```

```
{
```

```
fclose(DXFile_Pointer);
```

```
Error_Num = 100;
```

```
General_Error (Error_Num);
```

```
}
```

```
else if (!strcmp(Next_DXF_Section_Name,"EOF"))
```

```
{
```

```
fclose(DXFile_Pointer);
```

```
Error_Num = 101;
```

```
General_Error (Error_Num);
```

```
}
```

```
} while ((strcmp(Next_DXF_Section_Name,"ENTITIES")) != 0);
```

```
while ((!Travel_Path_Test_Done) || (Crnt_Primary_X_Coord == -1.0))
```

```
{
```

```
    Obtain_Entity_Para_Info
```

```
        (DXF_Section_Name,  
         Next_DXF_Section_Name,  
         &DXFile_Pointer,  
         Next_Layer_Name,  
         Next_Linetype_Impl_UpDown,  
         &Next_CAD_Elevation,  
         &Next_Line_Thickness,  
         &Next_Entity_Handle,  
         &Next_Color,  
         &Next_Primary_X_Coord,  
         &Next_Primary_Y_Coord,  
         &Next_Primary_Z_Coord,  
         &Next_Secondary_X_Coord,  
         &Next_Secondary_Y_Coord,  
         &Next_Secondary_Z_Coord,  
         &Next_Arc_Radius,  
         &Next_Start_Angle,  
         &Next_End_Angle);
```

```
if ((Entity_Scan_Pass_Number==1) &&  
    (strcmp(Next_DXF_Section_Name,"POINT")))
```

```
{
```

```
    fclose(DXFile_Pointer);  
    Error_Num = 103;  
    General_Error (Error_Num);
```

```
}
```

```
if (((Entity_Scan_Pass_Number == 1) || (Entity_Scan_Pass_Number ==2))  
    && (!strcmp(DXF_Section_Name,"ENDSEC")))
```

```
{
```

```
fclose(DXFile_Pointer);
Error_Num = 104;
General_Error (Error_Num);
}

if ((Entity_Scan_Pass_Number>1) &&
    (strcmp(Next_DXF_Section_Name,"ENDSEC")))
{
    Check_Travel_Path (Crnt_DXF_Section_Name,
                      Crnt_Layer_Name,
                      Crnt_Linetype_Impl_UpDown,
                      Crnt_CAD_Elevation,
                      Crnt_Line_Thickness,
                      Crnt_Entity_Handle,
                      Crnt_Color,
                      Crnt_Primary_X_Coord,
                      Crnt_Primary_Y_Coord,
                      Crnt_Primary_Z_Coord,
                      Crnt_Secondary_X_Coord,
                      Crnt_Secondary_Y_Coord,
                      Crnt_Secondary_Z_Coord,
                      Crnt_Arc_Radius,
                      Crnt_Start_Angle,
                      Crnt_End_Angle,

                      Next_DXF_Section_Name,
                      Next_Layer_Name,
                      Next_Linetype_Impl_UpDown,
                      Next_CAD_Elevation,
                      Next_Line_Thickness,
                      Next_Entity_Handle,
                      Next_Color,
                      Next_Primary_X_Coord,
                      Next_Primary_Y_Coord,
                      Next_Primary_Z_Coord,
```

```

Next_Secondary_X_Coord,
Next_Secondary_Y_Coord,
Next_Secondary_Z_Coord,
Next_Arc_Radius,
Next_Start_Angle,
Next_End_Angle,

```

```

Implement_Working_Width,
&Error_Warning_Flag);

```

```

if (Error_Warning_Flag == 1)
    Test_Fail = 1;
}

```

```

if (!strcmp(DXF_Section_Name,"ENDSEC"))
{
    fclose(DXFile_Pointer);
    Travel_Path_Test_Done = TRUE;
}

```

```

else if ((strcmp(DXF_Section_Name,"POINT")) &&
         (strcmp(DXF_Section_Name,"LINE")) &&
         (strcmp(DXF_Section_Name,"ARC")))
{
    fclose(DXFile_Pointer);
    Error_Num = 105;
    General_Error (Error_Num);
}

```

```

Entity_Scan_Pass_Number ++;

```

```

strcpy(Crnt_DXF_Section_Name , Next_DXF_Section_Name);
strcpy(Next_DXF_Section_Name , DXF_Section_Name);
strcpy(Crnt_Layer_Name , Next_Layer_Name);
strcpy(Crnt_Linetype_Impl_UpDown , Next_Linetype_Impl_UpDown);

```

```

Crnt_CAD_Elevation      = Next_CAD_Elevation;
Crnt_Line_Thickness    = Next_Line_Thickness;
Crnt_Entity_Handle     = Next_Entity_Handle;
Crnt_Color             = Next_Color;
Crnt_Primary_X_Coord   = Next_Primary_X_Coord;
Crnt_Primary_Y_Coord   = Next_Primary_Y_Coord;
Crnt_Primary_Z_Coord   = Next_Primary_Z_Coord;
Crnt_Secondary_X_Coord = Next_Secondary_X_Coord;
Crnt_Secondary_Y_Coord = Next_Secondary_Y_Coord;
Crnt_Secondary_Z_Coord = Next_Secondary_Z_Coord;
Crnt_Arc_Radius        = Next_Arc_Radius;
Crnt_Start_Angle       = Next_Start_Angle;
Crnt_End_Angle         = Next_End_Angle;

```

```

}

```

```

fclose(DXFile_Pointer);
printf("\n\n\n\n\n The DXF File %s has been
        closed.\n",FILENAME);
delay (Standard_Time_Delay/2);

```

```

if (Test_Fail == 1)

```

```

{
  clrscr ();
  printf("\n\n\n\n\n The DXF file had WARNINGS in it. \n");
  printf(" Correct the travel path and test it again.\n");
  Continue_Program ();
}

```

```

else

```

```

  Create_TPD_File (FILENAME);
}

```

```

/*BTM*****

```

```
/* Obtain Entity Parameter Information from DXF Drawing File */
```

```
void Obtain_Entity_Para_Info  
(char DXF_Section_Name [25],  
char Next_DXF_Section_Name [25],  
FILE **DXFFile_Pointer,  
char Next_Layer_Name [25],  
char Next_Linetype_Impl_UpDown [25],  
float *Next_CAD_Elevation,  
float *Next_Line_Thickness,  
int *Next_Entity_Handle,  
int *Next_Color,  
float *Next_Primary_X_Coord,  
float *Next_Primary_Y_Coord,  
float *Next_Primary_Z_Coord,  
float *Next_Secondary_X_Coord,  
float *Next_Secondary_Y_Coord,  
float *Next_Secondary_Z_Coord,  
float *Next_Arc_Radius,  
float *Next_Start_Angle,  
float *Next_End_Angle)  
{  
char Nameless [50];  
int Entity_Flag = 0;  
int Next_Group_Code = 0;  
char Comment [50];  
Boolean Entity_Selection_Done = FALSE;  
  
*Next_Entity_Handle = 0x0;  
*Next_CAD_Elevation = 0.0;  
*Next_Line_Thickness = 0.0;  
*Next_Primary_X_Coord = 0.0;  
*Next_Primary_Y_Coord = 0.0;  
*Next_Primary_Z_Coord = 0.0;  
*Next_Secondary_X_Coord = 0.0;  
*Next_Secondary_Y_Coord = 0.0;
```

```
*Next_Secondary_Z_Coord = 0.0;
*Next_Arc_Radius      = 0.0;
*Next_Start_Angle    = 0.0;
*Next_End_Angle      = 0.0;
*Next_Entity_Handle   = 0;
*Next_Color           = 0;

do
{
    fscanf(*DXFile_Pointer,"%5d",&Next_Group_Code);

    switch (Next_Group_Code)
    {
        case 0:

            fscanf(*DXFile_Pointer,"%s",DXF_Section_Name);

            if (Entity_Flag == 0)
            {
                strcpy(Next_DXF_Section_Name,DXF_Section_Name);
            }

            if ((Entity_Flag == 1) || (strcmp(Next_DXF_Section_Name,"EOF")))
            {
                Entity_Selection_Done = TRUE;
            }
            break;

        case 5:
            fscanf(*DXFile_Pointer,"%x",Next_Entity_Handle);

            Entity_Flag = 1;
            break;

        case 6:
```

```
fscanf(*DXFile_Pointer,"%s",Next_Linetype_Impl_UpDown);
```

```
Entity_Flag = 1;  
break;
```

```
case 8:
```

```
fscanf(*DXFile_Pointer,"%s",Next_Layer_Name);
```

```
Entity_Flag = 1;  
break;
```

```
case 10:
```

```
fscanf(*DXFile_Pointer,"%f",Next_Primary_X_Coord);
```

```
Entity_Flag = 1;  
break;
```

```
case 11:
```

```
fscanf(*DXFile_Pointer,"%f",Next_Secondary_X_Coord);
```

```
Entity_Flag = 1;  
break;
```

```
case 20:
```

```
fscanf(*DXFile_Pointer,"%f",Next_Primary_Y_Coord);
```

```
Entity_Flag = 1;  
break;
```

```
case 21:
```

```
fscanf(*DXFile_Pointer,"%f",Next_Secondary_Y_Coord);
```

```
Entity_Flag = 1;  
break;
```

case 30:

```
fscanf(*DXFile_Pointer,"%f",Next_Primary_Z_Coord);
```

```
Entity_Flag = 1;
```

```
break;
```

case 31:

```
fscanf(*DXFile_Pointer,"%f",Next_Secondary_Z_Coord);
```

```
Entity_Flag = 1;
```

```
break;
```

case 38:

```
fscanf(*DXFile_Pointer,"%f",Next_CAD_Elevation);
```

```
Entity_Flag = 1;
```

```
break;
```

case 39:

```
fscanf(*DXFile_Pointer,"%f",Next_Line_Thickness);
```

```
Entity_Flag = 1;
```

```
break;
```

case 40:

```
fscanf(*DXFile_Pointer,"%f",Next_Arc_Radius);
```

```
Entity_Flag = 1;
```

```
break;
```

case 50:

```
fscanf(*DXFile_Pointer,"%f",Next_Start_Angle);
```

```
Entity_Flag = 1;
```

```
break;
```

```

case 51:
    fscanf(*DXFile_Pointer,"%f",Next_End_Angle);

    Entity_Flag = 1;
    break;

case 62:
    fscanf(*DXFile_Pointer,"%d",Next_Color);

    Entity_Flag = 1;
    break;

case 999:
    fgets(Comment,50,*DXFile_Pointer);

    /* The following line will print out any comments */
    /* put into a CADD DXF output file by the previous */
    /* user, or field travel path drawing developer */

    printf("%s",Comment);
    Entity_Flag = 0;
    break;

default:
    fgets(Nameless,50,*DXFile_Pointer);
    Entity_Flag = 0;
    /* printf("Nameless %s",Nameless);*/
    /* possibly include a send to ERROR handling function here */
    break;
}
} while (!Entity_Selection_Done);
}

/*BTM***** */

```

```
/* Check Field Travel Path Logic */  
void Check_Travel_Path  
(char Crnt_DXF_Section_Name [],  
char Crnt_Layer_Name [],  
char Crnt_Linetype_Impl_UpDown [],  
float Crnt_CAD_Elevation,  
float Crnt_Line_Thickness,  
int Crnt_Entity_Handle,  
int Crnt_Color,  
float Crnt_Primary_X_Coord,  
float Crnt_Primary_Y_Coord,  
float Crnt_Primary_Z_Coord,  
float Crnt_Secondary_X_Coord,  
float Crnt_Secondary_Y_Coord,  
float Crnt_Secondary_Z_Coord,  
float Crnt_Arc_Radius,  
float Crnt_Start_Angle,  
float Crnt_End_Angle,  
  
char Next_DXF_Section_Name [],  
char Next_Layer_Name [],  
char Next_Linetype_Impl_UpDown [],  
float Next_CAD_Elevation,  
float Next_Line_Thickness,  
int Next_Entity_Handle,  
int Next_Color,  
float Next_Primary_X_Coord,  
float Next_Primary_Y_Coord,  
float Next_Primary_Z_Coord,  
float Next_Secondary_X_Coord,  
float Next_Secondary_Y_Coord,  
float Next_Secondary_Z_Coord,  
float Next_Arc_Radius,  
float Next_Start_Angle,  
float Next_End_Angle,
```



```

(Crnt_Primary_X_Coord >= Next_Secondary_X_Coord - Dist_Tlrc)
    &&
((Crnt_Primary_Y_Coord <= Next_Secondary_Y_Coord + Dist_Tlrc) &&
 (Crnt_Primary_Y_Coord >= Next_Secondary_Y_Coord - Dist_Tlrc))))

```

```

{
printf("    Point / Line interface (OK)\n");
delay (Standard_Time_Delay/4);
}

```

```

else
{
    Error_Num = 200;
    *Error_Warning_Flag = 1;
    General_Error (Error_Num);
}

```

```

}

```

```

/*-----*/

```

```

else if ((!strcmp(Crnt_DXF_Section_Name,"LINE")) &&
 (strcmp(Next_DXF_Section_Name,"POINT")))
{
    if (((Crnt_Primary_X_Coord <= Next_Primary_X_Coord + Dist_Tlrc) &&
 (Crnt_Primary_X_Coord >= Next_Primary_X_Coord - Dist_Tlrc))
        &&
        ((Crnt_Primary_Y_Coord <= Next_Primary_Y_Coord + Dist_Tlrc) &&
 (Crnt_Primary_Y_Coord >= Next_Primary_Y_Coord - Dist_Tlrc)))
        ||
        (((Crnt_Secondary_X_Coord <= Next_Primary_X_Coord + Dist_Tlrc) &&
 (Crnt_Secondary_X_Coord >= Next_Primary_X_Coord - Dist_Tlrc))
        &&
        ((Crnt_Secondary_Y_Coord <= Next_Primary_Y_Coord + Dist_Tlrc) &&
 (Crnt_Secondary_Y_Coord >= Next_Primary_Y_Coord - Dist_Tlrc))))

```



```

Next_Secondary_X_Coord,
Next_Secondary_Y_Coord);

```

```

if ((Crnt_Primary_X_Coord <= Next_Primary_X_Coord + Dist_Tlrc) &&
    (Crnt_Primary_X_Coord >= Next_Primary_X_Coord - Dist_Tlrc) &&
    (Crnt_Primary_Y_Coord <= Next_Primary_Y_Coord + Dist_Tlrc) &&
    (Crnt_Primary_Y_Coord >= Next_Primary_Y_Coord - Dist_Tlrc))

```

```

Length_Line_C = Distance_Length (Crnt_Secondary_X_Coord,
                                Crnt_Secondary_Y_Coord,
                                Next_Secondary_X_Coord,
                                Next_Secondary_Y_Coord);

```

```

else if ((Crnt_Primary_X_Coord <= Next_Secondary_X_Coord + Dist_Tlrc) &&
    (Crnt_Primary_X_Coord >= Next_Secondary_X_Coord - Dist_Tlrc) &&
    (Crnt_Primary_Y_Coord <= Next_Secondary_Y_Coord + Dist_Tlrc) &&
    (Crnt_Primary_Y_Coord >= Next_Secondary_Y_Coord - Dist_Tlrc))

```

```

Length_Line_C = Distance_Length (Crnt_Secondary_X_Coord,
                                Crnt_Secondary_Y_Coord,
                                Next_Primary_X_Coord,
                                Next_Primary_Y_Coord);

```

```

else if ((Crnt_Secondary_X_Coord <= Next_Primary_X_Coord + Dist_Tlrc) &&
    (Crnt_Secondary_X_Coord >= Next_Primary_X_Coord - Dist_Tlrc) &&
    (Crnt_Secondary_Y_Coord <= Next_Primary_Y_Coord + Dist_Tlrc) &&
    (Crnt_Secondary_Y_Coord >= Next_Primary_Y_Coord - Dist_Tlrc))

```

```

Length_Line_C = Distance_Length (Crnt_Primary_X_Coord,
                                Crnt_Primary_Y_Coord,
                                Next_Secondary_X_Coord,
                                Next_Secondary_Y_Coord);

```

```

else if ((Crnt_Secondary_X_Coord <= Next_Secondary_X_Coord + Dist_Tlrc) &&
    (Crnt_Secondary_X_Coord >= Next_Secondary_X_Coord - Dist_Tlrc) &&

```

```
(Crnt_Secondary_Y_Coord <= Next_Secondary_Y_Coord + Dist_Tlrc) &&
(Crnt_Secondary_Y_Coord >= Next_Secondary_Y_Coord - Dist_Tlrc))
```

```
Length_Line_C = Distance_Length (Crnt_Primary_X_Coord,
                                Crnt_Primary_Y_Coord,
                                Next_Primary_X_Coord,
                                Next_Primary_Y_Coord);
```

```
if ((Length_Line_C <= (Length_Line_A +
                      Length_Line_B +
                      Dist_Tlrc))
    &&
    (Length_Line_C >= (Length_Line_A +
                      Length_Line_B -
                      Dist_Tlrc)))
```

```
{
printf("    Line / Line interface (OK)\n");
delay (Standard_Time_Delay/4);
}
```

```
else
```

```
{
    Error_Num = 203;
    *Error_Warning_Flag = 1;
    General_Error (Error_Num);
}
```

```
/*-----*/
```

```
else if ((!strcmp(Crnt_DXF_Section_Name,"POINT")) &&
        (!strcmp(Next_DXF_Section_Name,"ARC")))
```

```
{
    Next_Arc_Startpoint_X_Coord = Next_Primary_X_Coord + (Next_Arc_Radius *
```

```
cos(Next_Start_Angle*(2*pi/360)));
```

```
Next_Arc_Endpoint_X_Coord = Next_Primary_X_Coord + (Next_Arc_Radius *
cos(Next_End_Angle*(2*pi/360)));
```

```
Next_Arc_Startpoint_Y_Coord = Next_Primary_Y_Coord + (Next_Arc_Radius *
sin(Next_Start_Angle*(2*pi/360)));
```

```
Next_Arc_Endpoint_Y_Coord = Next_Primary_Y_Coord + (Next_Arc_Radius *
sin(Next_End_Angle*(2*pi/360)));
```

```
if (((Crnt_Primary_X_Coord <= Next_Arc_Startpoint_X_Coord + Dist_Tlrc) &&
(Crnt_Primary_X_Coord >= Next_Arc_Startpoint_X_Coord - Dist_Tlrc) &&
(Crnt_Primary_Y_Coord <= Next_Arc_Startpoint_Y_Coord + Dist_Tlrc) &&
(Crnt_Primary_Y_Coord >= Next_Arc_Startpoint_Y_Coord - Dist_Tlrc))
```

```
||
```

```
((Crnt_Primary_X_Coord <= Next_Arc_Endpoint_X_Coord + Dist_Tlrc) &&
(Crnt_Primary_X_Coord >= Next_Arc_Endpoint_X_Coord - Dist_Tlrc) &&
(Crnt_Primary_Y_Coord <= Next_Arc_Endpoint_Y_Coord + Dist_Tlrc) &&
(Crnt_Primary_Y_Coord >= Next_Arc_Endpoint_Y_Coord - Dist_Tlrc))) &&
(Next_Arc_Radius >= Implement_Working_Width - Width_Tol))
```

```
{
printf("    Point / Arc interface (OK)\n");
delay (Standard_Time_Delay/4);
}
```

```
else
```

```
{
Error_Num = 204;
*Error_Warning_Flag = 1;
General_Error (Error_Num);
}
}
```

```

/*-----*/

else if ((!strcmp(Crnt_DXF_Section_Name,"ARC")) &&
        (!strcmp(Next_DXF_Section_Name,"POINT")))
{
    Crnt_Arc_Startpoint_X_Coord = Crnt_Primary_X_Coord + (Crnt_Arc_Radius *
        cos(Crnt_Start_Angle*(2*pi/360)));

    Crnt_Arc_Endpoint_X_Coord = Crnt_Primary_X_Coord + (Crnt_Arc_Radius *
        cos(Crnt_End_Angle*(2*pi/360)));

    Crnt_Arc_Startpoint_Y_Coord = Crnt_Primary_Y_Coord + (Crnt_Arc_Radius *
        sin(Crnt_Start_Angle*(2*pi/360)));

    Crnt_Arc_Endpoint_Y_Coord = Crnt_Primary_Y_Coord + (Crnt_Arc_Radius *
        sin(Crnt_End_Angle*(2*pi/360)));

    if (((((Next_Primary_X_Coord <= Crnt_Arc_Startpoint_X_Coord + Dist_Tlnc) &&
        (Next_Primary_X_Coord >= Crnt_Arc_Startpoint_X_Coord - Dist_Tlnc) &&
        (Next_Primary_Y_Coord <= Crnt_Arc_Startpoint_Y_Coord + Dist_Tlnc) &&
        (Next_Primary_Y_Coord >= Crnt_Arc_Startpoint_Y_Coord - Dist_Tlnc))
        ||
        ((Next_Primary_X_Coord <= Crnt_Arc_Endpoint_X_Coord + Dist_Tlnc) &&
        (Next_Primary_X_Coord >= Crnt_Arc_Endpoint_X_Coord - Dist_Tlnc) &&
        (Next_Primary_Y_Coord <= Crnt_Arc_Endpoint_Y_Coord + Dist_Tlnc) &&
        (Next_Primary_Y_Coord >= Crnt_Arc_Endpoint_Y_Coord - Dist_Tlnc))) &&
        (Crnt_Arc_Radius >= Implement_Working_Width - Width_Tol))
    {
        printf("    Arc / Point interface (OK)\n");
        delay (Standard_Time_Delay/4);
    }

else
{
    Error_Num = 205;
}
}

```



```

        Next_Primary_Y_Coord)
        + Dist_Tlrc))

    &&
    ((Crnt_Arc_Radius - Next_Arc_Radius) >= (Distance_Length
        (Crnt_Primary_X_Coord,
        Crnt_Primary_Y_Coord,
        Next_Primary_X_Coord,
        Next_Primary_Y_Coord)
        - Dist_Tlrc)))

    ||
    (((Next_Arc_Radius - Crnt_Arc_Radius) <= (Distance_Length
        (Crnt_Primary_X_Coord,
        Crnt_Primary_Y_Coord,
        Next_Primary_X_Coord,
        Next_Primary_Y_Coord)
        + Dist_Tlrc))

    &&
    ((Next_Arc_Radius - Crnt_Arc_Radius) >= (Distance_Length
        (Crnt_Primary_X_Coord,
        Crnt_Primary_Y_Coord,
        Next_Primary_X_Coord,
        Next_Primary_Y_Coord)
        - Dist_Tlrc))))

    &&
    (((Crnt_Arc_Startpoint_X_Coord <= Next_Arc_Endpoint_X_Coord + Dist_Tlrc) &&
    (Crnt_Arc_Startpoint_X_Coord >= Next_Arc_Endpoint_X_Coord - Dist_Tlrc) &&
    (Crnt_Arc_Startpoint_Y_Coord <= Next_Arc_Endpoint_Y_Coord + Dist_Tlrc) &&
    (Crnt_Arc_Startpoint_Y_Coord >= Next_Arc_Endpoint_Y_Coord - Dist_Tlrc))

    ||
    ((Crnt_Arc_Endpoint_X_Coord <= Next_Arc_Startpoint_X_Coord + Dist_Tlrc) &&
    (Crnt_Arc_Endpoint_X_Coord >= Next_Arc_Startpoint_X_Coord - Dist_Tlrc) &&
    (Crnt_Arc_Endpoint_Y_Coord <= Next_Arc_Startpoint_Y_Coord + Dist_Tlrc) &&
    (Crnt_Arc_Endpoint_Y_Coord >= Next_Arc_Startpoint_Y_Coord - Dist_Tlrc))))

    ||
    (((Crnt_Arc_Radius + Next_Arc_Radius) <= (Distance_Length

```

270

```
(Crnt_Primary_X_Coord,  
Crnt_Primary_Y_Coord,  
Next_Primary_X_Coord,  
Next_Primary_Y_Coord)  
+ Dist_Tlrc))
```

&&

```
((Crnt_Arc_Radius + Next_Arc_Radius) >= (Distance_Length  
(Crnt_Primary_X_Coord,  
Crnt_Primary_Y_Coord,  
Next_Primary_X_Coord,  
Next_Primary_Y_Coord)  
- Dist_Tlrc)))
```

&&

```
((Crnt_Arc_Startpoint_X_Coord <= Next_Arc_Startpoint_X_Coord + Dist_Tlrc) &&  
(Crnt_Arc_Startpoint_X_Coord >= Next_Arc_Startpoint_X_Coord - Dist_Tlrc) &&  
(Crnt_Arc_Startpoint_Y_Coord <= Next_Arc_Startpoint_Y_Coord + Dist_Tlrc) &&  
(Crnt_Arc_Startpoint_Y_Coord >= Next_Arc_Startpoint_Y_Coord - Dist_Tlrc))
```

||

```
((Crnt_Arc_Endpoint_X_Coord <= Next_Arc_Endpoint_X_Coord + Dist_Tlrc) &&  
(Crnt_Arc_Endpoint_X_Coord >= Next_Arc_Endpoint_X_Coord - Dist_Tlrc) &&  
(Crnt_Arc_Endpoint_Y_Coord <= Next_Arc_Endpoint_Y_Coord + Dist_Tlrc) &&  
(Crnt_Arc_Endpoint_Y_Coord >= Next_Arc_Endpoint_Y_Coord - Dist_Tlrc))))
```

&&

```
((Crnt_Arc_Radius >= Implement_Working_Width - Width_Tol) &&  
(Next_Arc_Radius >= Implement_Working_Width - Width_Tol)))
```

```
{  
    printf("    Arc / Arc interface (OK)\n");  
    delay (Standard_Time_Delay/4);  
}
```

else

```
{  
    Error_Num = 206;
```

```

*Error_Warning_Flag = 1;
  General_Error (Error_Num);
}
}

/*-----*/

else if ((!strcmp(Crnt_DXF_Section_Name,"LINE")) &&
         (!strcmp(Next_DXF_Section_Name,"ARC")))
{
  Length_Line_A = Distance_Length (Crnt_Primary_X_Coord,
                                   Crnt_Primary_Y_Coord,
                                   Crnt_Secondary_X_Coord,
                                   Crnt_Secondary_Y_Coord);

  Next_Arc_Startpoint_X_Coord = Next_Primary_X_Coord + (Next_Arc_Radius *
                                                         cos(Next_Start_Angle*(2*pi/360)));

  Next_Arc_Endpoint_X_Coord  = Next_Primary_X_Coord + (Next_Arc_Radius *
                                                         cos(Next_End_Angle*(2*pi/360)));

  Next_Arc_Startpoint_Y_Coord = Next_Primary_Y_Coord + (Next_Arc_Radius *
                                                         sin(Next_Start_Angle*(2*pi/360)));

  Next_Arc_Endpoint_Y_Coord  = Next_Primary_Y_Coord + (Next_Arc_Radius *
                                                         sin(Next_End_Angle*(2*pi/360)));

  if ((((((pow(Length_Line_A,2) + pow(Next_Arc_Radius,2)) <=
         (pow ((Distance_Length
              (Crnt_Primary_X_Coord, Crnt_Primary_Y_Coord,
              Next_Primary_X_Coord, Next_Primary_Y_Coord)
              + Dist_Tlrc),2))))
      &&
      ((pow(Length_Line_A,2) + pow(Next_Arc_Radius,2)) >=
       (pow ((Distance_Length

```

```

(Crnt_Primary_X_Coord, Crnt_Primary_Y_Coord,
Next_Primary_X_Coord, Next_Primary_Y_Coord)
- Dist_Tlrnc),2))))
    ||
(((powl(Length_Line_A,2) + powl(Next_Arc_Radius,2)) <=
(powl ((Distance_Length
(Crnt_Secondary_X_Coord, Crnt_Secondary_Y_Coord,
Next_Primary_X_Coord, Next_Primary_Y_Coord)
+ Dist_Tlrnc),2))))
    &&
((powl(Length_Line_A,2) + powl(Next_Arc_Radius,2)) >=
(powl ((Distance_Length
(Crnt_Secondary_X_Coord, Crnt_Secondary_Y_Coord,
Next_Primary_X_Coord, Next_Primary_Y_Coord)
- Dist_Tlrnc),2))))
    &&
(((Crnt_Primary_X_Coord <= Next_Arc_Startpoint_X_Coord + Dist_Tlrnc) &&
(Crnt_Primary_X_Coord >= Next_Arc_Startpoint_X_Coord - Dist_Tlrnc) &&
(Crnt_Primary_Y_Coord <= Next_Arc_Startpoint_Y_Coord + Dist_Tlrnc) &&
(Crnt_Primary_Y_Coord >= Next_Arc_Startpoint_Y_Coord - Dist_Tlrnc))
    ||
((Crnt_Secondary_X_Coord <= Next_Arc_Startpoint_X_Coord + Dist_Tlrnc) &&
(Crnt_Secondary_X_Coord >= Next_Arc_Startpoint_X_Coord - Dist_Tlrnc) &&
(Crnt_Secondary_Y_Coord <= Next_Arc_Startpoint_Y_Coord + Dist_Tlrnc) &&
(Crnt_Secondary_Y_Coord >= Next_Arc_Startpoint_Y_Coord - Dist_Tlrnc))
    ||
((Crnt_Primary_X_Coord <= Next_Arc_Endpoint_X_Coord + Dist_Tlrnc) &&
(Crnt_Primary_X_Coord >= Next_Arc_Endpoint_X_Coord - Dist_Tlrnc) &&
(Crnt_Primary_Y_Coord <= Next_Arc_Endpoint_Y_Coord + Dist_Tlrnc) &&
(Crnt_Primary_Y_Coord >= Next_Arc_Endpoint_Y_Coord - Dist_Tlrnc))
    ||
((Crnt_Secondary_X_Coord <= Next_Arc_Endpoint_X_Coord + Dist_Tlrnc) &&
(Crnt_Secondary_X_Coord >= Next_Arc_Endpoint_X_Coord - Dist_Tlrnc) &&
(Crnt_Secondary_Y_Coord <= Next_Arc_Endpoint_Y_Coord + Dist_Tlrnc) &&
(Crnt_Secondary_Y_Coord >= Next_Arc_Endpoint_Y_Coord - Dist_Tlrnc))))

```

```

        &&
        (Next_Arc_Radius >= Implement_Working_Width - Width_Tol))

    {
        printf ("    Line / Arc interface (OK)\n");
        delay (Standard_Time_Delay/4);
    }

else
    {
        Error_Num = 207;
        *Error_Warning_Flag = 1;
        General_Error (Error_Num);
    }
}

/*-----*/

else if ((!strcmp(Crnt_DXF_Section_Name,"ARC")) &&
        (!strcmp(Next_DXF_Section_Name,"LINE")))
    {
        Length_Line_B = Distance_Length (Next_Primary_X_Coord,
                                         Next_Primary_Y_Coord,
                                         Next_Secondary_X_Coord,
                                         Next_Secondary_Y_Coord);

        Crnt_Arc_Startpoint_X_Coord = Crnt_Primary_X_Coord + (Crnt_Arc_Radius *
                                                                cos(Crnt_Start_Angle*(2*pi/360)));

        Crnt_Arc_Endpoint_X_Coord = Crnt_Primary_X_Coord + (Crnt_Arc_Radius *
                                                             cos(Crnt_End_Angle*(2*pi/360)));

        Crnt_Arc_Startpoint_Y_Coord = Crnt_Primary_Y_Coord + (Crnt_Arc_Radius *
                                                                sin(Crnt_Start_Angle*(2*pi/360)));
    }

```

```
Crnt_Arc_Endpoint_Y_Coord = Crnt_Primary_Y_Coord + (Crnt_Arc_Radius *
    sin(Crnt_End_Angle*(2*pi/360)));
```

```
if ((((((powl(Length_Line_B,2) + powl(Crnt_Arc_Radius,2)) <=
    (powl ((Distance_Length (Crnt_Primary_X_Coord, Crnt_Primary_Y_Coord,
        Next_Primary_X_Coord, Next_Primary_Y_Coord)
        + Dist_Tlrc),2))))
    &&
    ((powl(Length_Line_B,2) + powl(Crnt_Arc_Radius,2)) >=
    (powl ((Distance_Length (Crnt_Primary_X_Coord, Crnt_Primary_Y_Coord,
        Next_Primary_X_Coord, Next_Primary_Y_Coord)
        - Dist_Tlrc),2))))))
    ||
    (((powl(Length_Line_B,2) + powl(Crnt_Arc_Radius,2)) <=
    (powl ((Distance_Length (Crnt_Primary_X_Coord, Crnt_Primary_Y_Coord,
        Next_Secondary_X_Coord,
        Next_Secondary_Y_Coord)
        + Dist_Tlrc),2))))
    &&
    ((powl(Length_Line_B,2) + powl(Crnt_Arc_Radius,2)) >=
    (powl ((Distance_Length (Crnt_Primary_X_Coord, Crnt_Primary_Y_Coord,
        Next_Secondary_X_Coord,
        Next_Secondary_Y_Coord)
        - Dist_Tlrc),2))))))
    &&
    (((Next_Primary_X_Coord <= Crnt_Arc_Startpoint_X_Coord + Dist_Tlrc) &&
    (Next_Primary_X_Coord >= Crnt_Arc_Startpoint_X_Coord - Dist_Tlrc) &&
    (Next_Primary_Y_Coord <= Crnt_Arc_Startpoint_Y_Coord + Dist_Tlrc) &&
    (Next_Primary_Y_Coord >= Crnt_Arc_Startpoint_Y_Coord - Dist_Tlrc))
    ||
    ((Next_Secondary_X_Coord <= Crnt_Arc_Startpoint_X_Coord + Dist_Tlrc) &&
    (Next_Secondary_X_Coord >= Crnt_Arc_Startpoint_X_Coord - Dist_Tlrc) &&
    (Next_Secondary_Y_Coord <= Crnt_Arc_Startpoint_Y_Coord + Dist_Tlrc) &&
    (Next_Secondary_Y_Coord >= Crnt_Arc_Startpoint_Y_Coord - Dist_Tlrc))
    ||
```

```

((Next_Primary_X_Coord <= Crnt_Arc_Endpoint_X_Coord + Dist_Tlrc) &&
 (Next_Primary_X_Coord >= Crnt_Arc_Endpoint_X_Coord - Dist_Tlrc) &&
 (Next_Primary_Y_Coord <= Crnt_Arc_Endpoint_Y_Coord + Dist_Tlrc) &&
 (Next_Primary_Y_Coord >= Crnt_Arc_Endpoint_Y_Coord - Dist_Tlrc))
    ||
((Next_Secondary_X_Coord <= Crnt_Arc_Endpoint_X_Coord + Dist_Tlrc) &&
 (Next_Secondary_X_Coord >= Crnt_Arc_Endpoint_X_Coord - Dist_Tlrc) &&
 (Next_Secondary_Y_Coord <= Crnt_Arc_Endpoint_Y_Coord + Dist_Tlrc) &&
 (Next_Secondary_Y_Coord >= Crnt_Arc_Endpoint_Y_Coord - Dist_Tlrc))))
    &&
(Crnt_Arc_Radius >= Implement_Working_Width - Width_Tol))

{
    printf("    Arc / Line interface (OK)\n");
    delay (Standard_Time_Delay/4);
}

else
{
    Error_Num = 208;
    *Error_Warning_Flag = 1;
    General_Error (Error_Num);
}
}

}

/*BTM*****
/* Distance or Length Calculation */
float Distance_Length (float Primary_X_Coord,
                      float Primary_Y_Coord,
                      float Secondary_X_Coord,
                      float Secondary_Y_Coord)
{
    float Dstnc_Lngth;

```



```
File_Type = 'G';  
sprintf(File_Mode,"w");
```

```
Open_File' (&TPD_File_Pointer, AUX_FILENAME, File_Type,  
            File_Mode, FILENAME);
```

```
TPD_Group_Code = 999;  
fprintf(TPD_File_Pointer,"%d\n",TPD_Group_Code);  
fprintf(TPD_File_Pointer,"TPD_TEST_PASS\n");
```

```
do
```

```
{
```

```
    fscanf(DXFile_Pointer,"%d",&DXF_Group_Code);  
    fscanf(DXFile_Pointer,"%s",&DXF_Data);
```

```
    if (!strcmp(DXF_Data,"EOF"))
```

```
    {
```

```
        fprintf(TPD_File_Pointer,"%d\n",DXF_Group_Code);  
        fprintf(TPD_File_Pointer,"%s\n",DXF_Data);  
        Done = TRUE;
```

```
    }
```

```
    else if (DXF_Group_Code == 5)
```

```
    {
```

```
        fprintf(TPD_File_Pointer,"%d\n",DXF_Group_Code);  
        fprintf(TPD_File_Pointer,"%x\n",TPD_Entity_Handle);  
        TPD_Entity_Handle ++;
```

```
    }
```

```
    else
```

```
    {
```

```
        fprintf(TPD_File_Pointer,"%d\n",DXF_Group_Code);  
        fprintf(TPD_File_Pointer,"%s\n",DXF_Data);
```

```
    }
```

```

    } while (!Done);

fclose(TPD_File_Pointer);
fclose(DXFile_Pointer);

printf("\n\n The DXF and the TPD files have been closed.\n");
printf("\n End of TESTFIELD.\n");

delay (Standard_Time_Delay/2);

clrscr();
File_Type = 'G';
sprintf(File_Mode,"r");

printf("\n\n          Preparing to display the FIELD TRAVEL PATH
\n");

Open_File (&TPD_File_Pointer, AUX_FILENAME, File_Type,
           File_Mode, FILENAME);

printf("\n The TPD File %s has been opened successfully.\n",FILENAME);

Plot_Field_Travel_Path (TPD_File_Pointer, FILENAME);

printf("\n Returning to main menu.\n");
delay (Standard_Time_Delay);

}

/*BTM*****
/* GRAPHICALLY PLOT the TRAVEL PATH DATA FILE */

int Plot_Field_Travel_Path (FILE *TPD_File_Pointer, char FILENAME[])

```

```
{  
  char TPD_Section_Name [25];  
  char Crnt_TPD_Section_Name [25];  
  char Crnt_Layer_Name [25];  
  char Crnt_Linetype_Impl_UpDown [25];  
  float Crnt_CAD_Elevation;  
  float Crnt_Line_Thickness;  
  int Crnt_Entity_Handle;  
  int Crnt_Color;  
  float Crnt_Primary_X_Coord;  
  float Crnt_Primary_Y_Coord;  
  float Crnt_Primary_Z_Coord;  
  float Crnt_Secondary_X_Coord;  
  float Crnt_Secondary_Y_Coord;  
  float Crnt_Secondary_Z_Coord;  
  float Crnt_Arc_Radius;  
  float Crnt_Start_Angle;  
  float Crnt_End_Angle;  
  
  /* Initializing the variables used in the program */  
  /* Select VGA (HI) 640 X 480, 16 COLORS 1 PAGE VIDEO MEMORY  
  */  
  int gdriver = VGA;  
  int gmode = VGAHI;  
  int gerr;  
  int Graphics_X_Max;  
  int Graphics_Y_Max;  
  float Field_Scale_Factor;  
  float Radius_Scale_Factor;  
  
  char No_File;  
  int Error_Num;  
  
  int Entity_Scan_Pass_Number = 1;  
  Boolean Travel_Path_Done = FALSE;
```

```
float Crnt_Arc_Startpoint_X_Coord;
float Crnt_Arc_Endpoint_X_Coord;
float Crnt_Arc_Startpoint_Y_Coord;
float Crnt_Arc_Endpoint_Y_Coord;
```

```
float X_ARC_Vect;
float Y_ARC_Vect;
float ARC_Vect_Len;
float Unit_X_Vect = 1.0;
float Unit_Vect_Len = 1.0;
float Vector_Dot_Product;
float Vector_Angle;
```

```
clrscr();
printf("\n\n\n\n\n\n  Type in the Field Scale Factor; press <ENTER> \n");
printf("    The field scale factor is used to scale the actual field\n");
printf("    coordinates to the monitor display\n");
```

```
scanf("%f",&Field_Scale_Factor);
```

```
printf("\n\n  Initializing GRAPHICS display mode\n");
delay (Standard_Time_Delay/2);
clrscr();
```

```
/* Register the graphics driver */
/* Allows standalone graphics capabilities for this program */
registerbgidriver(EGAVGA_driver);
```

```
/* Load the proper graphics BGI driver files from disk */
initgraph (&gdriver, &gmode, "");
gerr = graphresult ();
```

```
/* Check to see if graphics mode has been
   detected and initialized correctly */
```

```

if (gerr != grOk)
{
    printf("BGI GRAPHICS DISPLAY ERROR!; %s\n", grapherrormsg(gerr));
    exit(gerr);
}

/* Clear the viewport to default settings (background color)*/
graphdefaults ();

/* Set the specified background color */
setbkcolor (Dark_Green);

Graphics_X_Max = getmaxx() + 1;
Graphics_Y_Max = getmaxy() + 1;

/* Draw a thick, solid, white line border around edge of display */
setcolor(White);
setlinestyle(SOLID_LINE,zero,THICK_WIDTH);
rectangle (2,(Graphics_Y_Max - 1), (Graphics_X_Max -1), 2);

Graphics_X_Max = Graphics_Y_Max; /* ← SQUARE VIEWPORT */

/* Set the viewport to the parametric dimensions */
/* Lower left hand corner of viewport CP (Current Point) set at (0,0) */
/* setviewport (int left, int top, int right, int bottom, int clip) */
    setviewport (0, Graphics_Y_Max, Graphics_X_Max, 0, CLIPON);

setcolor(Yellow);
outtextxy(10,10,"FIELD TRAVEL PATH");

/* Set the specified foreground color */
/* All travel path entities Yellow on a Green background */
setcolor(Yellow);

/*_____*/

```

```

/* Check to see if there is data in the file */
No_File = fgetc (TPD_File_Pointer);
if (No_File == EOF)
{
    fclose(TPD_File_Pointer);
    Error_Num = 102;
    General_Error (Error_Num);
}
do
{
    fscanf(TPD_File_Pointer,"%s",Crnt_TPD_Section_Name);

/* Check to see if END OF FILE */
if (!strcmp(Crnt_TPD_Section_Name,"EOF"))
    {
        fclose(TPD_File_Pointer);
        Error_Num = 101;
        General_Error (Error_Num);
    }

} while ((strcmp(Crnt_TPD_Section_Name,"ENTITIES") != 0);

while (!Travel_Path_Done)
{
    Obtain_Entity_Para_Info
    (TPD_Section_Name,
    Crnt_TPD_Section_Name,
    &TPD_File_Pointer,
    Crnt_Layer_Name,
    Crnt_Linetype_Impl_UpDown,
    &Crnt_CAD_Elevation,
    &Crnt_Line_Thickness,
    &Crnt_Entity_Handle,

```

```
&Crnt_Color,  
&Crnt_Primary_X_Coord,  
&Crnt_Primary_Y_Coord,  
&Crnt_Primary_Z_Coord,  
&Crnt_Secondary_X_Coord,  
&Crnt_Secondary_Y_Coord,  
&Crnt_Secondary_Z_Coord,  
&Crnt_Arc_Radius,  
&Crnt_Start_Angle,  
&Crnt_End_Angle);  
  
Crnt_Primary_X_Coord = Crnt_Primary_X_Coord *  
    (Graphics_X_Max / Field_Scale_Factor);  
  
Crnt_Primary_Y_Coord = (Graphics_Y_Max - (Crnt_Primary_Y_Coord *  
*  
    (Graphics_Y_Max / Field_Scale_Factor)));  
  
Crnt_Secondary_X_Coord = Crnt_Secondary_X_Coord *  
    (Graphics_X_Max / Field_Scale_Factor);  
  
Crnt_Secondary_Y_Coord = (Graphics_Y_Max - (Crnt_Secondary_Y_Coord *  
    (Graphics_Y_Max / Field_Scale_Factor)));  
  
/* PLOT THE CURRENT TRAVEL PATH ENTITY */  
if (strcmp(Crnt_TPD_Section_Name,"ENDSEC"))  
{  
    if (!strcmp(Crnt_TPD_Section_Name,"LINE"))  
    {  
        line ((int) Crnt_Primary_X_Coord,  
            (int) Crnt_Primary_Y_Coord,  
            (int) Crnt_Secondary_X_Coord,  
            (int) Crnt_Secondary_Y_Coord);  
    }  
}
```

```

else if (!strcmp(Crnt_TPD_Section_Name,"ARC"))
{
Crnt_Arc_Startpoint_X_Coord = Crnt_Primary_X_Coord + (Crnt_Arc_Radius *
cos(Crnt_Start_Angle * (2 * pi / 360)));

Crnt_Arc_Endpoint_X_Coord = Crnt_Primary_X_Coord + (Crnt_Arc_Radius *
cos(Crnt_End_Angle * (2 * pi / 360)));

Crnt_Arc_Startpoint_Y_Coord = Crnt_Primary_Y_Coord + (Crnt_Arc_Radius *
sin(Crnt_Start_Angle * (2 * pi / 360)));

Crnt_Arc_Endpoint_Y_Coord = Crnt_Primary_Y_Coord + (Crnt_Arc_Radius *
sin(Crnt_End_Angle * (2 * pi / 360)));

X_ARC_Vect = (Crnt_Arc_Endpoint_X_Coord - Crnt_Arc_Startpoint_X_Coord);

Y_ARC_Vect = (Crnt_Arc_Endpoint_Y_Coord - Crnt_Arc_Startpoint_Y_Coord);

ARC_Vect_Len = Distance_Length (zero, zero, X_ARC_Vect, Y_ARC_Vect);

Vector_Dot_Product = (X_ARC_Vect * Unit_X_Vect);

Vector_Angle =
(acos (Vector_Dot_Product / (Unit_Vect_Len * ARC_Vect_Len)))*(360/(2*pi));

Radius_Scale_Factor =
(Graphics_Y_Max - ((Graphics_Y_Max - Graphics_X_Max) *
(Vector_Angle / 90)));

Crnt_Arc_Radius =
Crnt_Arc_Radius * Radius_Scale_Factor / Field_Scale_Factor;

arc ((int) Crnt_Primary_X_Coord,
(int) Crnt_Primary_Y_Coord,
(int) Crnt_Start_Angle,

```

```
        (int) Crnt_End_Angle,  
        (int) Crnt_Arc_Radius);  
    }  
  
    else if (!strcmp(Crnt_TPD_Section_Name,"POINT"))  
    {  
        setcolor(Dark_Blue);  
        outtextxy(Crnt_Primary_X_Coord - 10,  
                 Crnt_Primary_Y_Coord + 10,  
                 "STARTPOINT");  
  
        circle ((int) Crnt_Primary_X_Coord,  
              (int) Crnt_Primary_Y_Coord, 3);  
        setcolor(Yellow);  
    }  
  
    if (!strcmp(TPD_Section_Name,"ENDSEC"))  
    {  
        fclose(TPD_File_Pointer);  
        Travel_Path_Done = TRUE;  
    }  
  
    strcpy(Crnt_TPD_Section_Name , TPD_Section_Name);  
  
    }  
}  
  
setcolor(Yellow);  
outtextxy(400,10,"Press any key to continue");  
getch();  
/* Return to text display */  
closegraph ();  
  
fclose(TPD_File_Pointer);
```


APPENDIX I

FARM FIELD MODULE [FARMFIEL.C]

```
/* Farm Field Module [FARMFIEL.C] */
```

```
#include "gpsheade.h"
```

```
/*BTM*****
```

```
/* Farm Field [High Precision Farming Module] */
```

```
void Farm_Field ()
```

```
{
```

```
/* Declare the variables used in this program module */
```

```
char Tractor_Steering_Type;
```

```
char Tractor_Hitch_Type;
```

```
char Tractor_Hitch_Swing_Point;
```

```
char Tractor_Hitch_Location;
```

```
double Tractor_Wheelbase;
```

```
double Tractor_Drawbar_Length;
```

```
double Tractor_Hitch_Swing_Distance;
```

```
double Tractor_Max_Hitch_Swing_Angle;
```

```
double Tractor_Max_Steering_Angle;
```

```
double Implement_Hitch_Length;
```

```
double Implement_Working_Width;
```

```
double GPS_Receiver_Height;
```

```
double GPS_Forward_Backward_Dist;
```

```
double GPS_Left_Right_Dist;
```

```
char GPS_Forward_Backward_Dir;
```

```
char GPS_Left_Right_Dir;
```

```
double GPS_Receiver_Lngtdnl_Crrctn;
```

```
double GPS_Receiver_Tangent_Crrctn;
```

```
double GPS_Receiver_Height_Crrctn;
```

```
double Transverse_Mast_Angle = 0.0;
```

```
double Longitudinal_Mast_Angle = 0.0;
```

```
double BasePoint_Degrees_Latitude;  
double BasePoint_Minutes_Latitude;  
double BasePoint_Seconds_Latitude;  
double BasePoint_Degrees_Longitude;  
double BasePoint_Minutes_Longitude;  
double BasePoint_Seconds_Longitude;
```

```
double Startpoint_X_Coord;  
double Startpoint_Y_Coord;  
double Endpoint_X_Coord;  
double Endpoint_Y_Coord;
```

```
int Field_Travel_StartEntity;  
int Field_Travel_EndEntity;
```

```
char Crnt_TPD_Section_Name [25];  
char Crnt_Layer_Name [25];  
char Crnt_Linetype_Impl_UpDown [25];  
float Crnt_CAD_Elevation;  
float Crnt_Line_Thickness;  
int Crnt_Entity_Handle;  
int Crnt_Color;
```

```
float Crnt_Primary_X_Coord;  
float Crnt_Primary_Y_Coord;  
float Crnt_Primary_Z_Coord;  
float Crnt_Secondary_X_Coord;  
float Crnt_Secondary_Y_Coord;  
float Crnt_Secondary_Z_Coord;  
float Crnt_Arc_Radius;  
float Crnt_Start_Angle;  
float Crnt_End_Angle;
```

```
char TPD_Section_Name [25];
```

```
char Next_TPD_Section_Name [25];
char Next_Layer_Name [25];
char Next_Linetype_Impl_UpDown [25];
float Next_CAD_Elevation;
float Next_Line_Thickness;
int Next_Entity_Handle;
int Next_Color;

float Next_Primary_X_Coord;
float Next_Primary_Y_Coord;
float Next_Primary_Z_Coord;
float Next_Secondary_X_Coord;
float Next_Secondary_Y_Coord;
float Next_Secondary_Z_Coord;
float Next_Arc_Radius;
float Next_Start_Angle;
float Next_End_Angle;

float Crnt_Arc_Startpoint_X_Coord;
float Crnt_Arc_Endpoint_X_Coord;
float Crnt_Arc_Startpoint_Y_Coord;
float Crnt_Arc_Endpoint_Y_Coord;

float Next_Arc_Startpoint_X_Coord;
float Next_Arc_Endpoint_X_Coord;
float Next_Arc_Startpoint_Y_Coord;
float Next_Arc_Endpoint_Y_Coord;

double Steering_Angle = 0.0;
long double Look_Ahead_Steer_Angle = 0.0;
long double Actual_Output_Steering_Angle = 0.0;

long double X_Field_Coord = 0.0;
long double Y_Field_Coord = 0.0;
```

```
long double Z_Field_Coord = 0.0;  
long double Travel_Speed = 0.0;
```

```
double Depth = 0.0;  
double Application_Rate = 0.0;  
double Aux_Contrl = 0.0;
```

```
double GIS_Grid_Cell_Size;
```

```
int Hitch_Position = 0;  
double Travel_Path_Dist_Error = 0.0;
```

```
/* Default Steering Angle */  
float Straight_Ahead = 0.0;
```

```
float Previous_Path_Error = 0.0;
```

```
float Crnt_X_Strt;  
float Crnt_Y_Strt;  
float Crnt_X_End;  
float Crnt_Y_End;  
float Previous_X_End;  
float Previous_Y_End;  
int Left_Right;
```

```
float Next_X_Strt;  
float Next_X_End;  
float Next_Y_Strt;  
float Next_Y_End;  
int Look_Ahead_Steer_Left_Right;
```

```
FILE *TPD_File_Pointer;  
FILE *GOS_File_Pointer;  
FILE *GIS_File_Pointer;  
Boolean Field_Travel_Done = FALSE;
```

```
Boolean TRC_IMP_GPS_File_Done = FALSE;  
Boolean Travel_Path_File_Done = FALSE;  
Boolean Get_Next_Entity      = FALSE;
```

```
/* Start of Farm Field program module */
```

```
clrscr ();  
printf("\n\n\n\n\n      FARM FIELD\n\n");  
printf("  Globally Positioned Agricultural Machinery\n\n");  
printf("  High Precision Farming Module \n\n\n\n\n");
```

```
Continue_Program ();
```

```
/* Obtain the Field Status information and the GIS input / output  
file names as well as the GIS grid cell size */
```

```
Info_Travel_Path_Field_GIS  
( &TPD_File_Pointer,  
  &GOS_File_Pointer,  
  &GIS_File_Pointer,  
  &Field_Travel_Done,  
  &TRC_IMP_GPS_File_Done,  
  &Travel_Path_File_Done,  
  &BasePoint_Degrees_Latitude,  
  &BasePoint_Minutes_Latitude,  
  &BasePoint_Seconds_Latitude,  
  &BasePoint_Degrees_Longitude,  
  &BasePoint_Minutes_Longitude,  
  &BasePoint_Seconds_Longitude,  
  &Startpoint_X_Coord,  
  &Startpoint_Y_Coord,  
  &Endpoint_X_Coord,  
  &Endpoint_Y_Coord,  
  &GIS_Grid_Cell_Size,  
  &Field_Travel_StartEntity,
```

```
&Field_Travel_EndEntity);

/* Initialize the GPS Receiver Basepoint (for a particular field) */
/* G.P.S. Receiver will also "lock" onto satellites before control */
/* returns to this program */

if ((Field_Travel_Done == FALSE) && (TRC_IMP_GPS_File_Done == FALSE))

{
  Init_GPS_Rcvr_Basepoint_Protocol
  (BasePoint_Degrees_Latitude,
   BasePoint_Minutes_Latitude,
   BasePoint_Seconds_Latitude,
   BasePoint_Degrees_Longitude,
   BasePoint_Minutes_Longitude,
   BasePoint_Seconds_Longitude);

  /* Obtain the Tractor and Implement configuration information */

  Info_Tractor_Impl_GPS_Var
  (&Field_Travel_Done,
   &TRC_IMP_GPS_File_Done,
   &Travel_Path_File_Done,
   &Tractor_Steering_Type,
   &Tractor_Hitch_Type,
   &Tractor_Hitch_Swing_Point,
   &Tractor_Hitch_Location,
   &Tractor_Wheelbase,
   &Tractor_Drawbar_Length,
   &Tractor_Hitch_Swing_Distance,
   &Tractor_Max_Hitch_Swing_Angle,
   &Tractor_Max_Steering_Angle,
   &Implement_Hitch_Length,
   &Implement_Working_Width,
   &GPS_Receiver_Height,
```

```
&GPS_Forward_Backward_Dist,  
&GPS_Left_Right_Dist,  
&GPS_Forward_Backward_Dir,  
&GPS_Left_Right_Dir);
```

```
/* Equipment operator manually driving the machinery to the  
field travel path startpoint to initialize the position */
```

```
Initial_Equipment_Positioning
```

```
(TPD_Section_Name,  
Next_TPD_Section_Name,  
TPD_File_Pointer,  
Next_Layer_Name,  
Next_Linetype_Impl_UpDown,  
&Next_CAD_Elevation,  
&Next_Line_Thickness,  
&Next_Entity_Handle,  
&Next_Color,  
&Next_Primary_X_Coord,  
&Next_Primary_Y_Coord,  
&Next_Primary_Z_Coord,  
&Next_Secondary_X_Coord,  
&Next_Secondary_Y_Coord,  
&Next_Secondary_Z_Coord,  
&Next_Arc_Radius,  
&Next_Start_Angle,  
&Next_End_Angle,  
Startpoint_X_Coord,  
Startpoint_Y_Coord,  
Field_Travel_StartEntity);
```

```
/* Set the Previous End point coordinates equal to the  
Startpoint Coordinates for use in the function
```

```
Crnt_Pth_Sgmnt_Strt_End_Pts */
```

```
Previous_X_End = Startpoint_X_Coord;
```

```
Previous_Y_End = Startpoint_Y_Coord;
}
/*-----*/
/* Main machinery travel path program loop */

while (Field_Travel_Done == FALSE)
{
Crnt_Pth_Sgmnt_Strt_End_Pts
  (Crnt_TPD_Section_Name,
   Next_TPD_Section_Name,
   Crnt_Primary_X_Coord,
   Crnt_Primary_Y_Coord,
   Crnt_Secondary_X_Coord,
   Crnt_Secondary_Y_Coord,
   Crnt_Arc_Radius,
   Crnt_Start_Angle,
   Crnt_End_Angle,
   Next_Primary_X_Coord,
   Next_Primary_Y_Coord,
   Next_Secondary_X_Coord,
   Next_Secondary_Y_Coord,
   Next_Arc_Radius,
   Next_Start_Angle,
   Next_End_Angle,
   &Crnt_X_Strt,
   &Crnt_Y_Strt,
   &Crnt_X_End,
   &Crnt_Y_End,
   &Previous_X_End,
   &Previous_Y_End);

Slope_Input_Data (&Transverse_Mast_Angle, &Longitudinal_Mast_Angle);

Slope_Corrected_Receiver_Location
  (Transverse_Mast_Angle, Longitudinal_Mast_Angle,
```

```
GPS_Receiver_Height, GPS_Forward_Backward_Dist,  
GPS_Left_Right_Dist, GPS_Forward_Backward_Dir,  
GPS_Left_Right_Dir, &GPS_Receiver_Lngtdnl_Crrctn,  
&GPS_Receiver_Tangent_Crrctn, &GPS_Receiver_Height_Crrctn);
```

```
Steering_Angle_Input (&Steering_Angle);
```

```
if ((Tractor_Hitch_Type == 'S') || (Tractor_Hitch_Type == 'T'))  
    Hitch_Position_Input (&Hitch_Position);
```

```
GPS_Position_Data (&X_Field_Coord, &Y_Field_Coord,  
                  &Z_Field_Coord, &Travel_Speed);
```

```
Travel_Path_Error  
(&Travel_Path_Dist_Error,  
GPS_Receiver_Lngtdnl_Crrctn,  
GPS_Receiver_Tangent_Crrctn,  
Crnt_Entity_Handle,  
Crnt_TPD_Section_Name,  
Crnt_Primary_X_Coord,  
Crnt_Primary_Y_Coord,  
Crnt_Secondary_X_Coord,  
Crnt_Secondary_Y_Coord,  
Crnt_Arc_Radius,  
Crnt_Start_Angle,  
Crnt_End_Angle,  
Next_Entity_Handle,  
Next_TPD_Section_Name,  
Next_Primary_X_Coord,  
Next_Primary_Y_Coord,  
Next_Secondary_X_Coord,  
Next_Secondary_Y_Coord,  
Next_Arc_Radius,  
Next_Start_Angle,  
Next_End_Angle,
```

```

X_Field_Coord,
Y_Field_Coord,
Z_Field_Coord,
Travel_Speed);

```

```

if ((!strcmp(Crnt_TPD_Section_Name,"LINE")) ||
    (!strcmp(Next_TPD_Section_Name,"LINE")))

```

```

{
    Look_Ahead_Steer_Angle = Straight_Ahead;
    Look_Ahead_Steer_Left_Right = 0;

```

```

/* Determine if the implement is to the right or left of the
   target travel path (when target travel path is a straight
   LINE. (The Point_to_Arc_Error function determines if the
   implement is right or left of the travel path if the current
   travel path segment is an ARC) */

```

```

Right_or_Left_of_Travel_Path

```

```

(&Left_Right,
 X_Field_Coord,
 Y_Field_Coord,
 &Crnt_X_Strt,
 &Crnt_Y_Strt,
 &Crnt_X_End,
 &Crnt_Y_End);

```

```

}

```

```

/*-----*/

```

```

if ((!strcmp(Next_TPD_Section_Name,"ARC")) &&
    (Get_Next_Entity == TRUE))

```

```

{

```

```

/* ITR is a function of the current travel path entity */

```

```

/* for a LINE the ITR = infinity -> the steering angle = 0 */

```

```

/* for an ARC the ITR = radius of the arc entity */
/* The next procedure will calculate the */
/* tractor steering angle required to follow the ITR arc */

/* Compute the theoretically required steering angle */
/* required to follow the travel path using a particular */
/* Tractor and Implement configuration */

```

Compute_Steering_Angle

```

(&Look_Ahead_Steer_Angle,
 Tractor_Steering_Type,
 Tractor_Hitch_Type,
 Tractor_Hitch_Swing_Point,
 Tractor_Hitch_Location,
 Tractor_Wheelbase,
 Tractor_Drawbar_Length,
 Tractor_Hitch_Swing_Distance,
 Tractor_Max_Hitch_Swing_Angle,
 Tractor_Max_Steering_Angle,
 Implement_Hitch_Length,
 Crnt_Arc_Radius,
 Hitch_Position);

```

```

Next_Arc_Startpoint_X_Coord = Next_Primary_X_Coord + (Next_Arc_Radius *
cos(Next_Start_Angle*(2*pi/360)));

```

```

Next_Arc_Endpoint_X_Coord = Next_Primary_X_Coord + (Next_Arc_Radius *
cos(Next_End_Angle*(2*pi/360)));

```

```

Next_Arc_Startpoint_Y_Coord = Next_Primary_Y_Coord + (Next_Arc_Radius *
sin(Next_Start_Angle*(2*pi/360)));

```

```

Next_Arc_Endpoint_Y_Coord = Next_Primary_Y_Coord + (Next_Arc_Radius *
sin(Next_End_Angle*(2*pi/360)));

```

```

/* Determine if need to steer LEFT or RIGHT
   to keep on next ARC travel path segment */

```

```

Steer_Left_Right (&Look_Ahead_Steer_Left_Right,
                  Crnt_TPD_Section_Name,
                  Next_Arc_Startpoint_X_Coord,
                  Next_Arc_Endpoint_X_Coord,
                  Next_Arc_Startpoint_Y_Coord,
                  Next_Arc_Endpoint_Y_Coord,
                  Next_Primary_X_Coord,
                  Next_Primary_Y_Coord,
                  Next_Secondary_X_Coord,
                  Next_Secondary_Y_Coord,
                  Crnt_Start_Angle,
                  Crnt_End_Angle,
                  Next_Arc_Radius,
                  Next_Start_Angle,
                  Next_End_Angle,
                  Crnt_X_Strt,
                  Crnt_X_End,
                  Crnt_Y_Strt,
                  Crnt_Y_End,
                  &Next_X_Strt,
                  &Next_X_End,
                  &Next_Y_Strt,
                  &Next_Y_End);

```

```

/* The Look_Ahead_Steer_Angle will be adjusted for distance from
   travel path even when following a straight line */

```

```

}

```

```

/* Correct the theoretical steering angle for travel path error
   and also for a "look ahead" component of the travel path */

```

```

Error_Corrected_Steering_Angle ( Travel_Path_Dist_Error,

```

```

    Left_Right,
    Look_Ahead_Steer_Left_Right,
    GPS_Receiver_Tangent_Crrctn,
    GPS_Left_Right_Dist,
    Previous_Path_Error,
    Tractor_Max_Steering_Angle,
    Look_Ahead_Steer_Angle,
    &Actual_Output_Steering_Angle,
    Straight_Ahead);

```

```
Steering_Output (Actual_Output_Steering_Angle);
```

```

/* GIS data used for input into the farm field program module */
/* Specific appropriate data parameters will be passed from this
   function when the actual parameters required are determined */
GIS_Data_Input (X_Field_Coord, Y_Field_Coord, GIS_Grid_Cell_Size);

```

```

/* GIS data used for output from the farm field program module */
/* Specific appropriate data parameters will be passed to this
   function when the actual parameters required are determined */
GIS_Data_Output ();

```

```

/* The Determine_Implement_Control and Implement_Control_Code */
/* functions will have the actual specific parameters passed */
/* to and from them when the required parameters are actually */
/* determined in future program development*/
Determine_Implement_Control ();

```

```
Implement_Control_Code (Depth, Application_Rate, Aux_Contrl);
```

```
/* if... (check to see if need to acquire next entity ("look ahead") */
```

```

Check_Acquire_TPD_Entity
(&Get_Next_Entity,
Next_TPD_Section_Name,

```

TPD_File_Pointer,
Next_Layer_Name,
Next_Linetype_Impl_UpDown,
&Next_CAD_Elevation,
&Next_Line_Thickness,
&Next_Entity_Handle,
&Next_Color,
&Next_Primary_X_Coord,
&Next_Primary_Y_Coord,
&Next_Primary_Z_Coord,
&Next_Secondary_X_Coord,
&Next_Secondary_Y_Coord,
&Next_Secondary_Z_Coord,
&Next_Arc_Radius,
&Next_Start_Angle,
&Next_End_Angle,
TPD_Section_Name,
Crnt_TPD_Section_Name,
Crnt_Layer_Name,
Crnt_Linetype_Impl_UpDown,
&Crnt_CAD_Elevation,
&Crnt_Line_Thickness,
&Crnt_Entity_Handle,
&Crnt_Color,
&Crnt_Primary_X_Coord,
&Crnt_Primary_Y_Coord,
&Crnt_Primary_Z_Coord,
&Crnt_Secondary_X_Coord,
&Crnt_Secondary_Y_Coord,
&Crnt_Secondary_Z_Coord,
&Crnt_Arc_Radius,
&Crnt_Start_Angle,
&Crnt_End_Angle,
Crnt_X_Strt,
Crnt_Y_Strt,

```

Crnt_X_End,
Crnt_Y_End,
Travel_Speed,
X_Field_Coord,
Y_Field_Coord,
Travel_Path_Dist_Error,
GPS_Forward_Backward_Dist,
GPS_Left_Right_Dist,
GPS_Forward_Backward_Dir,
GPS_Left_Right_Dir,
&Field_Travel_Done,
Field_Travel_EndEntity);

/* Loop while the Field Travel Path is NOT Done */
}
printf("    The END of program module FARM FIELD \n");
Continue_Program ();

/* When Field Travel Path is done, close all file pointers */
fcloseall ();
}

/*BTM***** */
/* Initialize the G.P.S. Receiver Basepoint Protocol */
void Init_GPS_Rcvr_Basepoint_Protocol
(double BasePoint_Degrees_Latitude,
double BasePoint_Minutes_Latitude,
double BasePoint_Seconds_Latitude,
double BasePoint_Degrees_Longitude,
double BasePoint_Minutes_Longitude,
double BasePoint_Seconds_Longitude)
{
    /* SOFTWARE OUTPUT TO HARDWARE
    Send the Field Basepoint Latitude and Longitude
    to the GPS Receiver Program for future reference

```

so that the GPS receiver program can calculate the position of the machinery in local (field) coordinates for use in this (FARM FIELD) program module */

/* The Field Basepoint Elevation is not required since the elevation is not used in the Travel Path Data file describing the travel path of the machinery through the field */

/* The G.P.S. Receiver should also "lock" onto the available satellites during this function, before returning to this program section */

}

/*BTM*****

/* Initializing the Machinery to the Travel Path Start Point */

```
void Initial_Equipment_Positioning
(char TPD_Section_Name [25],
char Next_TPD_Section_Name [25],
FILE *TPD_File_Pointer,
char Next_Layer_Name [25],
char Next_Linetype_Impl_UpDown [25],
float *Next_CAD_Elevation,
float *Next_Line_Thickness,
int *Next_Entity_Handle,
int *Next_Color,
float *Next_Primary_X_Coord,
float *Next_Primary_Y_Coord,
float *Next_Primary_Z_Coord,
float *Next_Secondary_X_Coord,
float *Next_Secondary_Y_Coord,
float *Next_Secondary_Z_Coord,
float *Next_Arc_Radius,
float *Next_Start_Angle,
float *Next_End_Angle,
```

```
double Startpoint_X_Coord,  
double Startpoint_Y_Coord,  
int Field_Travel_StartEntity)  
{  
  
long double X_Field_Coord = 0.0;  
long double Y_Field_Coord = 0.0;  
long double Z_Field_Coord = 0.0;  
long double Travel_Speed = 0.0;  
double Transverse_Mast_Angle = 0.0;  
double Longitudinal_Mast_Angle = 0.0;  
  
double GPS_Receiver_Height;  
double GPS_Forward_Backward_Dist;  
double GPS_Left_Right_Dist;  
char GPS_Forward_Backward_Dir;  
char GPS_Left_Right_Dir;  
double GPS_Receiver_Lngtdnl_Crrctn;  
double GPS_Receiver_Tangent_Crrctn;  
char Crnt_TPD_Section_Name [25];  
char Crnt_Layer_Name [25];  
char Crnt_Linetype_Impl_UpDown [25];  
float Crnt_CAD_Elevation;  
float Crnt_Line_Thickness;  
int Crnt_Entity_Handle;  
int Crnt_Color;  
float Crnt_Primary_X_Coord;  
float Crnt_Primary_Y_Coord;  
float Crnt_Primary_Z_Coord;  
float Crnt_Secondary_X_Coord;  
float Crnt_Secondary_Y_Coord;  
float Crnt_Secondary_Z_Coord;  
float Crnt_Arc_Radius;  
float Crnt_Start_Angle;  
float Crnt_End_Angle;
```

```
float Travel_Path_Start_Point_Error = 0.0;
Boolean Initial_Position = FALSE;
Boolean Field_Travel_Start = FALSE;
Next_Entity_Handle = 0;
```

```
clrscr ();
```

```
printf("\n\n\n\n\n  Manually drive the tractor to within 1/2 meter\n");
printf("  of the predetermined travel path startpoint.\n");
printf("  The approximate distance between the GPS receiver and
the\n");
printf("  startpoint will be displayed until \n");
printf("  the tractor is correctly positioned.\n");
```

```
Continue_Program ();
```

```
do
```

```
{
```

```
    fscanf(TPD_File_Pointer,"%s",Next_TPD_Section_Name);
```

```
    } while (strcmp(Next_TPD_Section_Name,"ENTITIES"));
```

```
do
```

```
{
```

```
Obtain_Entity_Para_Info
```

```
(TPD_Section_Name,
```

```
Next_TPD_Section_Name,
```

```
&TPD_File_Pointer,
```

```
Next_Layer_Name,
```

```
Next_Linetype_Impl_UpDown,
```

```
Next_CAD_Elevation,
```

```
Next_Line_Thickness,
```

```
Next_Entity_Handle,
```

```
Next_Color,
```

```
Next_Primary_X_Coord,
```

```

Next_Primary_Y_Coord,
Next_Primary_Z_Coord,
Next_Secondary_X_Coord,
Next_Secondary_Y_Coord,
Next_Secondary_Z_Coord,
Next_Arc_Radius,
Next_Start_Angle,
Next_End_Angle);

```

```

strcpy(Crnt_TPD_Section_Name , Next_TPD_Section_Name);
strcpy(Next_TPD_Section_Name , TPD_Section_Name);
strcpy(Crnt_Layer_Name , Next_Layer_Name);
strcpy(Crnt_Linetype_Impl_UpDown, Next_Linetype_Impl_UpDown);

```

```

Crnt_CAD_Elevation      = *Next_CAD_Elevation;
Crnt_Line_Thickness     = *Next_Line_Thickness;
Crnt_Entity_Handle     = *Next_Entity_Handle;
Crnt_Color              = *Next_Color;
Crnt_Primary_X_Coord    = *Next_Primary_X_Coord;
Crnt_Primary_Y_Coord    = *Next_Primary_Y_Coord;
Crnt_Primary_Z_Coord    = *Next_Primary_Z_Coord;
Crnt_Secondary_X_Coord  = *Next_Secondary_X_Coord;
Crnt_Secondary_Y_Coord  = *Next_Secondary_Y_Coord;
Crnt_Secondary_Z_Coord  = *Next_Secondary_Z_Coord;
Crnt_Arc_Radius         = *Next_Arc_Radius;
Crnt_Start_Angle        = *Next_Start_Angle;
Crnt_End_Angle          = *Next_End_Angle;

```

```

if (Crnt_Entity_Handle == (Field_Travel_StartEntity))
    Field_Travel_Start = TRUE;

```

```

}while (Field_Travel_Start == FALSE);

```

```

Obtain_Entity_Para_Info
(TPD_Section_Name,

```

```
Next_TPD_Section_Name,  
&TPD_File_Pointer,  
Next_Layer_Name,  
Next_Linetype_Impl_UpDown,  
Next_CAD_Elevation,  
Next_Line_Thickness,  
Next_Entity_Handle,  
Next_Color,  
Next_Primary_X_Coord,  
Next_Primary_Y_Coord,  
Next_Primary_Z_Coord,  
Next_Secondary_X_Coord,  
Next_Secondary_Y_Coord,  
Next_Secondary_Z_Coord,  
Next_Arc_Radius,  
Next_Start_Angle,  
Next_End_Angle);
```

```
do
```

```
{
```

```
GPS_Position_Data (&X_Field_Coord, &Y_Field_Coord,  
                  &Z_Field_Coord, &Travel_Speed);
```

```
Path_Start_Check (&Travel_Path_Start_Point_Error,  
                 X_Field_Coord, Y_Field_Coord,  
                 Startpoint_X_Coord, Startpoint_Y_Coord);
```

```
/* Possibly put in an approach angle to the startpoint calculation  
   so that the operator would be able to determine is the machinery  
   was positioned at the correct angle of approach to the travel path */
```

```
clrscr ();
```

```
printf("\n\n\n\n\n The current equipment position is: \n");
```

```

printf("    %f meters from the field travel Startpoint.\n",
       Travel_Path_Start_Point_Error);

delay (Standard_Time_Delay / 8);

if ((Travel_Path_Start_Point_Error < Travel_Path_Err_Tol))
    /* 0.5 Meter Dist. */
{
    Initial_Position = TRUE;
    clrscr ();
    printf("\n\n\n\n\n The Machinery is positioned at the \n");
    printf("    Field Travel Path Startpoint.\n");
    Continue_Program ();
}

}while (!Initial_Position);

}

/*BTM*****
/* Obtain the Implement Slope Meter Signals (in radians) */
void Slope_Input_Data (double *Transverse_Mast_Angle,
                      double *Longitudinal_Mast_Angle)
{
    /* The mast angle data is to come from the slope meter hardware
       One digital slope meter for measuring transverse slopes and
       one to measure the slope of the equipment in the
       longitudinal direction */

    /* HARDWARE INPUT TO SOFTWARE Transverse_Mast_Angle */
    /* HARDWARE INPUT TO SOFTWARE Longitudinal_Mast_Angle */
}

/*BTM*****
/* Slope_Corrected_Receiver_Location */

```

/* Determine the correct location of the center of the G.P.S. receiver relative to the center of the implement (corrected for ground slope)

If the GPS Mast angle is tilting longitudinally FORWARD, in the Direction of travel, the correction distance will be SUBTRACTED from the GPS position, and vice-versa.

If the GPS Mast angle is tilting to the RIGHT, in the tangent reference direction, the correction distance will be SUBTRACTED from the GPS position, and vice-versa. */

```
void Slope_Corrected_Receiver_Location
```

```
(double Transverse_Mast_Angle,
double Longitudinal_Mast_Angle,
double GPS_Receiver_Height,
double GPS_Forward_Backward_Dist,
double GPS_Left_Right_Dist,
char GPS_Forward_Backward_Dir,
char GPS_Left_Right_Dir,
double *GPS_Receiver_Lngtdnl_Crrctn,
double *GPS_Receiver_Tangent_Crrctn,
double *GPS_Receiver_Height_Crrctn)
```

```
{
```

```
double Tangent_Crrctn;
```

```
double Lngtdnl_Crrctn;
```

```
    Tangent_Crrctn = GPS_Receiver_Height * sin(Transverse_Mast_Angle);
```

```
    Lngtdnl_Crrctn = GPS_Receiver_Height * sin(Longitudinal_Mast_Angle);
```

```
    *GPS_Receiver_Height_Crrctn
```

```
        = GPS_Receiver_Height * cos(Transverse_Mast_Angle) *
        cos(Longitudinal_Mast_Angle);
```

```
/* NOTE: If located to RIGHT of travel path (positive + ) value
```

```
    If located to LEFT of travel path (negative - ) value
```

```
    Slope indicators must be set up for proper + / - readings */
```

```
switch (GPS_Forward_Backward_Dir)
{
    case 'F':
        *GPS_Receiver_Lngtdnl_Crrctn =
        - Lngtdnl_Crrctn - GPS_Forward_Backward_Dist;
        break;

    case 'B':
        *GPS_Receiver_Lngtdnl_Crrctn =
        Lngtdnl_Crrctn + GPS_Forward_Backward_Dist;
        break;

    case 'C':
        *GPS_Receiver_Lngtdnl_Crrctn = Lngtdnl_Crrctn;
        break;

    default:
        printf("\n\n\n There is an error in the Implement / ");
        printf("G.P.S. Configuration File. \n");
        break;
}
```

```
switch (GPS_Left_Right_Dir)
{
    case 'L':
        *GPS_Receiver_Tangent_Crrctn =
        Tangent_Crrctn - GPS_Left_Right_Dist;
        break;

    case 'R':
        *GPS_Receiver_Tangent_Crrctn =
        - Tangent_Crrctn + GPS_Left_Right_Dist;
        break;
}
```

```

case 'C':
    *GPS_Receiver_Tangent_Crrctn = Tangent_Crrctn;
    break;

default:
    printf("\n\n\n There is an error in the Implement ");
    printf("/ G.P.S. Configuration File. \n");
    break;
}
}

/*BTM*****
/* Determine the Difference between the actual position and the
   startpoint for the Field Travel Path */
void Path_Start_Check
(float *Travel_Path_Start_Point_Error,
 long double X_Field_Coord,
 long double Y_Field_Coord,
 double Startpoint_X_Coord,
 double Startpoint_Y_Coord)
{
    *Travel_Path_Start_Point_Error = Distance_Length (X_Field_Coord,
                                                    Y_Field_Coord,
                                                    Startpoint_X_Coord,
                                                    Startpoint_Y_Coord);

    /* Possibly include a starting angle check here */

}

/*BTM*****
/* Obtain the Steering Angle Transducer Signal */
void Steering_Angle_Input (double *Steering_Angle)
{

```

```

/* HARDWARE INPUT TO SOFTWARE */
/* The Steering_Angle_Input is to come from the steering angle hardware */
/* Steering_Angle_Input LEFT TURN -> POSITIVE VALUE and vice-versa*/
/* Determined using the right-hand rule */
}

```

```

/*BTM*****
/* Obtain the Hitch Extreme Position Transducer Signal */
void Hitch_Position_Input (int *Hitch_Position)
{
/* HARDWARE INPUT TO SOFTWARE */
/* The Hitch_Position_Input is to come from the hitch position hardware */
/* Hitch_Position LEFT TURN -> POSITIVE VALUE and vice-versa*/
/* Use simple on-off switches (two) */
/* mounted on both sides of the swinging drawbar frame */
/* The variable, Hitch_Position, must be an integer to use in */
/* later conditional statements correctly */
}

```

```

/*BTM*****
/* Calculate the Travel Path Error */
void Travel_Path_Error
(double *Travel_Path_Dist_Error,
double GPS_Receiver_Lngtdnl_Crrctn,
double GPS_Receiver_Tangent_Crrctn,
int Crnt_Entity_Handle,
char Crnt_TPD_Section_Name[25],
float Crnt_Primary_X_Coord,
float Crnt_Primary_Y_Coord,
float Crnt_Secondary_X_Coord,
float Crnt_Secondary_Y_Coord,
float Crnt_Arc_Radius,
float Crnt_Start_Angle,
float Crnt_End_Angle,
int Next_Entity_Handle,

```

```

char  Next_TPD_Section_Name [25],
float  Next_Primary_X_Coord,
float  Next_Primary_Y_Coord,
float  Next_Secondary_X_Coord,
float  Next_Secondary_Y_Coord,
float  Next_Arc_Radius,
float  Next_Start_Angle,
float  Next_End_Angle,
long double X_Field_Coord,
long double Y_Field_Coord,
long double Z_Field_Coord,
long double Travel_Speed)
{

double Path_Point_to_Line_Error;
double Path_Point_to_Arc_Error;

if (!strcmp(Crnt_TPD_Section_Name,"LINE"))
{
    Point_to_Line_Error
    (&Path_Point_to_Line_Error,
    X_Field_Coord, Y_Field_Coord,
    Crnt_Primary_X_Coord, Crnt_Primary_Y_Coord,
    Crnt_Secondary_X_Coord, Crnt_Secondary_Y_Coord,
    Next_TPD_Section_Name,
    Next_Primary_X_Coord, Next_Primary_Y_Coord,
    Next_Secondary_X_Coord, Next_Secondary_Y_Coord,
    Next_Arc_Radius, Next_Start_Angle, Next_End_Angle);

    *Travel_Path_Dist_Error = Path_Point_to_Line_Error;
}

else if (!strcmp(Crnt_TPD_Section_Name,"ARC"))
{
    Point_to_Arc_Error

```

```

    (&Path_Point_to_Arc_Error,
    X_Field_Coord, Y_Field_Coord,
    Crnt_Primary_X_Coord, Crnt_Primary_Y_Coord,
    Crnt_Secondary_X_Coord, Crnt_Secondary_Y_Coord,
    Crnt_Arc_Radius, Crnt_Start_Angle, Crnt_End_Angle,
    Next_TPD_Section_Name,
    Next_Primary_X_Coord, Next_Primary_Y_Coord,
    Next_Secondary_X_Coord, Next_Secondary_Y_Coord,
    Next_Arc_Radius, Next_Start_Angle, Next_End_Angle);

    *Travel_Path_Dist_Error = Path_Point_to_Arc_Error;
}
}

/*BTM*****
/* Calculate the point to line Travel Path error distance */
void Point_to_Line_Error
(double *Path_Point_to_Line_Error,
 long double X_Field_Coord,
 long double Y_Field_Coord,
 float Crnt_Primary_X_Coord,
 float Crnt_Primary_Y_Coord,
 float Crnt_Secondary_X_Coord,
 float Crnt_Secondary_Y_Coord,
 char Next_TPD_Section_Name [25],
 float Next_Primary_X_Coord,
 float Next_Primary_Y_Coord,
 float Next_Secondary_X_Coord,
 float Next_Secondary_Y_Coord,
 float Next_Arc_Radius,
 float Next_Start_Angle,
 float Next_End_Angle)
{

long double Line_Length;

```

```
float Crnt_Arc_Radius;  
float Crnt_Start_Angle;  
float Crnt_End_Angle;  
float Crnt_Arc_Startpoint_X_Coord;  
float Crnt_Arc_Startpoint_Y_Coord;  
float Next_Arc_Startpoint_X_Coord;  
float Next_Arc_Startpoint_Y_Coord;  
float Crnt_Arc_Endpoint_X_Coord;  
float Crnt_Arc_Endpoint_Y_Coord;  
float Next_Arc_Endpoint_X_Coord;  
float Next_Arc_Endpoint_Y_Coord;
```

```
Calc_Arc_Start_End_points  
(Crnt_Arc_Radius,  
Crnt_Start_Angle,  
Crnt_End_Angle,  
Next_Arc_Radius,  
Next_Start_Angle,  
Next_End_Angle,  
Crnt_Primary_X_Coord,  
Crnt_Primary_Y_Coord,  
Next_Primary_X_Coord,  
Next_Primary_Y_Coord,  
&Crnt_Arc_Startpoint_X_Coord,  
&Crnt_Arc_Startpoint_Y_Coord,  
&Next_Arc_Startpoint_X_Coord,  
&Next_Arc_Startpoint_Y_Coord,  
&Crnt_Arc_Endpoint_X_Coord,  
&Crnt_Arc_Endpoint_Y_Coord,  
&Next_Arc_Endpoint_X_Coord,  
&Next_Arc_Endpoint_Y_Coord);
```

```
Line_Length = Distance_Length (Crnt_Primary_X_Coord,  
Crnt_Primary_Y_Coord,  
Crnt_Secondary_X_Coord,
```

Crnt_Secondary_Y_Coord);

```

if ((((!strcmp(Next_TPD_Section_Name,"LINE")) &&
  (((Crnt_Primary_X_Coord <= Next_Primary_X_Coord + Dist_Tlrc) &&
    (Crnt_Primary_X_Coord >= Next_Primary_X_Coord - Dist_Tlrc) &&
    (Crnt_Primary_Y_Coord <= Next_Primary_Y_Coord + Dist_Tlrc) &&
    (Crnt_Primary_Y_Coord >= Next_Primary_Y_Coord - Dist_Tlrc)))
  ||
  ((Crnt_Primary_X_Coord <= Next_Secondary_X_Coord + Dist_Tlrc) &&
    (Crnt_Primary_X_Coord >= Next_Secondary_X_Coord - Dist_Tlrc) &&
    (Crnt_Primary_Y_Coord <= Next_Secondary_Y_Coord + Dist_Tlrc) &&
    (Crnt_Primary_Y_Coord >= Next_Secondary_Y_Coord - Dist_Tlrc))))
  ||
  ((!strcmp(Next_TPD_Section_Name,"POINT")) &&
    ((Crnt_Primary_X_Coord <= Next_Primary_X_Coord + Dist_Tlrc) &&
      (Crnt_Primary_X_Coord >= Next_Primary_X_Coord - Dist_Tlrc) &&
      (Crnt_Primary_Y_Coord <= Next_Primary_Y_Coord + Dist_Tlrc) &&
      (Crnt_Primary_Y_Coord >= Next_Primary_Y_Coord - Dist_Tlrc))))
  ||
  ((!strcmp(Next_TPD_Section_Name,"ARC")) &&
    (((Crnt_Primary_X_Coord <= Next_Arc_Startpoint_X_Coord + Dist_Tlrc) &&
      (Crnt_Primary_X_Coord >= Next_Arc_Startpoint_X_Coord - Dist_Tlrc) &&
      (Crnt_Primary_Y_Coord <= Next_Arc_Startpoint_Y_Coord + Dist_Tlrc) &&
      (Crnt_Primary_Y_Coord >= Next_Arc_Startpoint_Y_Coord - Dist_Tlrc)))
    ||
    ((Crnt_Primary_X_Coord <= Next_Arc_Endpoint_X_Coord + Dist_Tlrc) &&
      (Crnt_Primary_X_Coord >= Next_Arc_Endpoint_X_Coord - Dist_Tlrc) &&
      (Crnt_Primary_Y_Coord <= Next_Arc_Endpoint_Y_Coord + Dist_Tlrc) &&
      (Crnt_Primary_Y_Coord >= Next_Arc_Endpoint_Y_Coord - Dist_Tlrc))))
  {
    *Path_Point_to_Line_Error =
    (((Crnt_Secondary_X_Coord - X_Field_Coord) *
      (Crnt_Primary_Y_Coord - Crnt_Secondary_Y_Coord)) -
      ((Crnt_Secondary_Y_Coord - Y_Field_Coord) *

```

```

        (Crnt_Primary_X_Coord - Crnt_Secondary_X_Coord)))
    / Line_Length);
}

else if ((((!strcmp(Next_TPD_Section_Name,"LINE")) &&
(((Crnt_Secondary_X_Coord <= Next_Primary_X_Coord + Dist_Tlrc) &&
(Crnt_Secondary_X_Coord >= Next_Primary_X_Coord - Dist_Tlrc) &&
(Crnt_Secondary_Y_Coord <= Next_Primary_Y_Coord + Dist_Tlrc) &&
(Crnt_Secondary_Y_Coord >= Next_Primary_Y_Coord - Dist_Tlrc))
    ||
((Crnt_Secondary_X_Coord <= Next_Secondary_X_Coord + Dist_Tlrc) &&
(Crnt_Secondary_X_Coord >= Next_Secondary_X_Coord - Dist_Tlrc) &&
(Crnt_Secondary_Y_Coord <= Next_Secondary_Y_Coord + Dist_Tlrc) &&
(Crnt_Secondary_Y_Coord >= Next_Secondary_Y_Coord - Dist_Tlrc))))
    ||
(!strcmp(Next_TPD_Section_Name,"POINT")) &&
((Crnt_Secondary_X_Coord <= Next_Primary_X_Coord + Dist_Tlrc) &&
(Crnt_Secondary_X_Coord >= Next_Primary_X_Coord - Dist_Tlrc) &&
(Crnt_Secondary_Y_Coord <= Next_Primary_Y_Coord + Dist_Tlrc) &&
(Crnt_Secondary_Y_Coord >= Next_Primary_Y_Coord - Dist_Tlrc))))
    ||
(!strcmp(Next_TPD_Section_Name,"ARC")) &&
(((Crnt_Secondary_X_Coord <= Next_Arc_Startpoint_X_Coord + Dist_Tlrc) &&
(Crnt_Secondary_X_Coord >= Next_Arc_Startpoint_X_Coord - Dist_Tlrc) &&
(Crnt_Secondary_Y_Coord <= Next_Arc_Startpoint_Y_Coord + Dist_Tlrc) &&
(Crnt_Secondary_Y_Coord >= Next_Arc_Startpoint_Y_Coord - Dist_Tlrc))
    ||
((Crnt_Secondary_X_Coord <= Next_Arc_Endpoint_X_Coord + Dist_Tlrc) &&
(Crnt_Secondary_X_Coord >= Next_Arc_Endpoint_X_Coord - Dist_Tlrc) &&
(Crnt_Secondary_Y_Coord <= Next_Arc_Endpoint_Y_Coord + Dist_Tlrc) &&
(Crnt_Secondary_Y_Coord >= Next_Arc_Endpoint_Y_Coord - Dist_Tlrc))))
{
    *Path_Point_to_Line_Error =
    (((Crnt_Primary_X_Coord - X_Field_Coord) *

```

```

        (Crnt_Secondary_Y_Coord - Crnt_Primary_Y_Coord)) -
        ((Crnt_Primary_Y_Coord - Y_Field_Coord) *
        (Crnt_Secondary_X_Coord - Crnt_Primary_X_Coord)))
        / Line_Length);
    }

else
    {
        printf("\a ERROR!\n");
    }
}

```

```

/*BTM*****
/* Calculate the point to arc Travel Path error distance */
void Point_to_Arc_Error
(double *Path_Point_to_Arc_Error,
 long double X_Field_Coord,
 long double Y_Field_Coord,
 float Crnt_Primary_X_Coord,
 float Crnt_Primary_Y_Coord,
 float Crnt_Secondary_X_Coord,
 float Crnt_Secondary_Y_Coord,
 float Crnt_Arc_Radius,
 float Crnt_Start_Angle,
 float Crnt_End_Angle,
 char Next_TPD_Section_Name [25],
 float Next_Primary_X_Coord,
 float Next_Primary_Y_Coord,
 float Next_Secondary_X_Coord,
 float Next_Secondary_Y_Coord,
 float Next_Arc_Radius,
 float Next_Start_Angle,
 float Next_End_Angle)
{

```

```

float Crnt_Arc_Startpoint_X_Coord;
float Crnt_Arc_Startpoint_Y_Coord;
float Next_Arc_Startpoint_X_Coord;
float Next_Arc_Startpoint_Y_Coord;
float Crnt_Arc_Endpoint_X_Coord;
float Crnt_Arc_Endpoint_Y_Coord;
float Next_Arc_Endpoint_X_Coord;
float Next_Arc_Endpoint_Y_Coord;

```

```

Calc_Arc_Start_End_points

```

```

(Crnt_Arc_Radius,
Crnt_Start_Angle,
Crnt_End_Angle,
Next_Arc_Radius,
Next_Start_Angle,
Next_End_Angle,
Crnt_Primary_X_Coord,
Crnt_Primary_Y_Coord,
Next_Primary_X_Coord,
Next_Primary_Y_Coord,
&Crnt_Arc_Startpoint_X_Coord,
&Crnt_Arc_Startpoint_Y_Coord,
&Next_Arc_Startpoint_X_Coord,
&Next_Arc_Startpoint_Y_Coord,
&Crnt_Arc_Endpoint_X_Coord,
&Crnt_Arc_Endpoint_Y_Coord,
&Next_Arc_Endpoint_X_Coord,
&Next_Arc_Endpoint_Y_Coord);

```

```

*Path_Point_to_Arc_Error =

```

```

(sqrt ((powl((Crnt_Primary_X_Coord - X_Field_Coord),2)) +
      (powl((Crnt_Primary_Y_Coord - Y_Field_Coord),2)))) -
Crnt_Arc_Radius;

```

```

if (!strcmp(Next_TPD_Section_Name,"LINE"))

```

```

{
if (((Crnt_Arc_Startpoint_X_Coord <= Next_Primary_X_Coord + Dist_Tlrc) &&
    (Crnt_Arc_Startpoint_X_Coord >= Next_Primary_X_Coord - Dist_Tlrc) &&
    (Crnt_Arc_Startpoint_Y_Coord <= Next_Primary_Y_Coord + Dist_Tlrc) &&
    (Crnt_Arc_Startpoint_Y_Coord >= Next_Primary_Y_Coord - Dist_Tlrc))
    ||
((Crnt_Arc_Startpoint_X_Coord <= Next_Secondary_X_Coord + Dist_Tlrc) &&
    (Crnt_Arc_Startpoint_X_Coord >= Next_Secondary_X_Coord - Dist_Tlrc) &&
    (Crnt_Arc_Startpoint_Y_Coord <= Next_Secondary_Y_Coord + Dist_Tlrc) &&
    (Crnt_Arc_Startpoint_Y_Coord >= Next_Secondary_Y_Coord - Dist_Tlrc)))
{
/* + (pos.) need to steer to right to correct travel */
/* - (neg.) need to steer to left to correct travel */
/* Right Hand rule for travel path steering
   Left turns are positive, so if the value
   required for a left hand turn is negative,
   will have to change sign to conform to the
   right hand rule utilized in this program */

*Path_Point_to_Arc_Error = - *Path_Point_to_Arc_Error;
}

else if
(((Crnt_Arc_Endpoint_X_Coord <= Next_Primary_X_Coord + Dist_Tlrc) &&
    (Crnt_Arc_Endpoint_X_Coord >= Next_Primary_X_Coord - Dist_Tlrc) &&
    (Crnt_Arc_Endpoint_Y_Coord <= Next_Primary_Y_Coord + Dist_Tlrc) &&
    (Crnt_Arc_Endpoint_Y_Coord >= Next_Primary_Y_Coord - Dist_Tlrc))
    ||
((Crnt_Arc_Endpoint_X_Coord <= Next_Secondary_X_Coord + Dist_Tlrc) &&
    (Crnt_Arc_Endpoint_X_Coord >= Next_Secondary_X_Coord - Dist_Tlrc) &&
    (Crnt_Arc_Endpoint_Y_Coord <= Next_Secondary_Y_Coord + Dist_Tlrc) &&
    (Crnt_Arc_Endpoint_Y_Coord >= Next_Secondary_Y_Coord - Dist_Tlrc)))
{

```

```

/* + Steer to left - Steer to right */
/* No Correction Required here */

}

}

else if (!strcmp(Next_TPD_Section_Name,"ARC"))

{

if

(((Crnt_Arc_Startpoint_X_Coord <= Next_Arc_Startpoint_X_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Startpoint_X_Coord >= Next_Arc_Startpoint_X_Coord - Dist_Tlrc) &&
 (Crnt_Arc_Startpoint_Y_Coord <= Next_Arc_Startpoint_Y_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Startpoint_Y_Coord >= Next_Arc_Startpoint_Y_Coord - Dist_Tlrc))
    ||
((Crnt_Arc_Startpoint_X_Coord <= Next_Arc_Endpoint_X_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Startpoint_X_Coord >= Next_Arc_Endpoint_X_Coord - Dist_Tlrc) &&
 (Crnt_Arc_Startpoint_Y_Coord <= Next_Arc_Endpoint_Y_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Startpoint_Y_Coord >= Next_Arc_Endpoint_Y_Coord - Dist_Tlrc)))

{
    /* + Right - Left */
    *Path_Point_to_Arc_Error = - *Path_Point_to_Arc_Error;
}

else if

(((Crnt_Arc_Endpoint_X_Coord <= Next_Arc_Startpoint_X_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Endpoint_X_Coord >= Next_Arc_Startpoint_X_Coord - Dist_Tlrc) &&
 (Crnt_Arc_Endpoint_Y_Coord <= Next_Arc_Startpoint_Y_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Endpoint_Y_Coord >= Next_Arc_Startpoint_Y_Coord - Dist_Tlrc))
    ||
((Crnt_Arc_Endpoint_X_Coord <= Next_Arc_Endpoint_X_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Endpoint_X_Coord >= Next_Arc_Endpoint_X_Coord - Dist_Tlrc) &&

```

```

(Crnt_Arc_Endpoint_Y_Coord <= Next_Arc_Endpoint_Y_Coord + Dist_Tlrc) &&
(Crnt_Arc_Endpoint_Y_Coord >= Next_Arc_Endpoint_Y_Coord - Dist_Tlrc))
{
    /* + Left - Right */
    /* No Correction Required here */
}

}

else if (!strcmp(Next_TPD_Section_Name,"POINT"))
{
    if ((Crnt_Arc_Startpoint_X_Coord <= Next_Primary_X_Coord + Dist_Tlrc) &&
        (Crnt_Arc_Startpoint_X_Coord >= Next_Primary_X_Coord - Dist_Tlrc) &&
        (Crnt_Arc_Startpoint_Y_Coord <= Next_Primary_Y_Coord + Dist_Tlrc) &&
        (Crnt_Arc_Startpoint_Y_Coord >= Next_Primary_Y_Coord - Dist_Tlrc))
    {
        /* + Right - Left */
        *Path_Point_to_Arc_Error = - *Path_Point_to_Arc_Error;
    }

    else if ((Crnt_Arc_Endpoint_X_Coord <= Next_Primary_X_Coord + Dist_Tlrc) &&
        (Crnt_Arc_Endpoint_X_Coord >= Next_Primary_X_Coord - Dist_Tlrc) &&
        (Crnt_Arc_Endpoint_Y_Coord <= Next_Primary_Y_Coord + Dist_Tlrc) &&
        (Crnt_Arc_Endpoint_Y_Coord >= Next_Primary_Y_Coord - Dist_Tlrc))
    {
        /* + Left - Right */
        /* No Correction Required here */
    }
}

}

/*BTM***** */
/* Compute the Look Ahead Steering Left or Right Indicator (for ARC) */
void Steer_Left_Right

```

```

(int *Look_Ahead_Steer_Left_Right,
 char Crnt_TPD_Section_Name [],
 float Next_Arc_Startpoint_X_Coord,
 float Next_Arc_Endpoint_X_Coord,
 float Next_Arc_Startpoint_Y_Coord,
 float Next_Arc_Endpoint_Y_Coord,
 float Next_Primary_X_Coord,
 float Next_Primary_Y_Coord,
 float Next_Secondary_X_Coord,
 float Next_Secondary_Y_Coord,
 float Crnt_Start_Angle,
 float Crnt_End_Angle,
 float Next_Arc_Radius,
 float Next_Start_Angle,
 float Next_End_Angle,
 float Crnt_X_Strt,
 float Crnt_X_End,
 float Crnt_Y_Strt,
 float Crnt_Y_End,
 float *Next_X_Strt,
 float *Next_X_End,
 float *Next_Y_Strt,
 float *Next_Y_End)
{
 float X_Coord_Line_Path_Vect;
 float Y_Coord_Line_Path_Vect;
 float X_Coord_Arc_Ref_Vect;
 float Y_Coord_Arc_Ref_Vect;
 float Vector_Cross_Prod;

 if ((Next_Arc_Startpoint_X_Coord <= Crnt_X_End + Dist_Tlrc) &&
     (Next_Arc_Startpoint_X_Coord >= Crnt_X_End - Dist_Tlrc) &&
     (Next_Arc_Startpoint_Y_Coord <= Crnt_Y_End + Dist_Tlrc) &&
     (Next_Arc_Startpoint_Y_Coord >= Crnt_Y_End - Dist_Tlrc))
 {

```

```

*Next_X_Strt = Next_Arc_Endpoint_X_Coord;
*Next_Y_Strt = Next_Arc_Endpoint_Y_Coord;
*Next_X_End = Next_Arc_Startpoint_X_Coord;
*Next_Y_End = Next_Arc_Startpoint_Y_Coord;
}

else if ((Next_Arc_Endpoint_X_Coord <= Crnt_X_End + Dist_Tlrnc) &&
        (Next_Arc_Endpoint_X_Coord >= Crnt_X_End - Dist_Tlrnc) &&
        (Next_Arc_Endpoint_Y_Coord <= Crnt_Y_End + Dist_Tlrnc) &&
        (Next_Arc_Endpoint_Y_Coord >= Crnt_Y_End - Dist_Tlrnc))
{
    *Next_X_Strt = Next_Arc_Startpoint_X_Coord;
    *Next_Y_Strt = Next_Arc_Startpoint_Y_Coord;
    *Next_X_End = Next_Arc_Endpoint_X_Coord;
    *Next_Y_End = Next_Arc_Endpoint_Y_Coord;
}

/*-----*/

X_Coord_Arc_Ref_Vect = (*Next_X_End - *Next_X_Strt);

Y_Coord_Arc_Ref_Vect = (*Next_Y_End - *Next_Y_Strt);

if (Istrcmp(Crnt_TPD_Section_Name,"LINE"))
{

X_Coord_Line_Path_Vect = (Crnt_X_End - Crnt_X_Strt);

Y_Coord_Line_Path_Vect = (Crnt_Y_End - Crnt_Y_Strt);

Vector_Cross_Prod =
    ((X_Coord_Line_Path_Vect) * (Y_Coord_Arc_Ref_Vect) -
     (Y_Coord_Arc_Ref_Vect) * (X_Coord_Line_Path_Vect));

if (Vector_Cross_Prod > zero + small)

```

```

*Look_Ahead_Steer_Left_Right = 1;

else if (Vector_Cross_Prod < zero - small)
    *Look_Ahead_Steer_Left_Right = -1;

else
    /* Either the ARC entity is extremely close to a straight line
    or there is an error in the .TPD travel path data file */
    *Look_Ahead_Steer_Left_Right = 0;
}

/*-----*/
/* The Right Hand Rule -> Steer Left if +1 ( > zero ) */

if (!strcmp(Crnt_TPD_Section_Name,"ARC"))
{

    if ((Crnt_Start_Angle <= Next_Start_Angle + Smll_Angle_Tol) &&
        (Crnt_Start_Angle >= Next_Start_Angle - Smll_Angle_Tol))
        /* Next steering angle to the LEFT */
        {
            *Look_Ahead_Steer_Left_Right = 1;
        }

    else if ((Crnt_Start_Angle <= Next_End_Angle + Smll_Angle_Tol) &&
            (Crnt_Start_Angle >= Next_End_Angle - Smll_Angle_Tol))
        /* Next steering angle to the RIGHT */
        {
            *Look_Ahead_Steer_Left_Right = -1;
        }

    else if ((Crnt_End_Angle <= Next_Start_Angle + Smll_Angle_Tol) &&
            (Crnt_End_Angle >= Next_Start_Angle - Smll_Angle_Tol))
        /* Next steering angle to the LEFT */
        {

```

```

    *Look_Ahead_Steer_Left_Right = 1;
}

else if ((Crnt_End_Angle <= Next_End_Angle + Sml_Angle_Tol) &&
        (Crnt_End_Angle >= Next_End_Angle - Sml_Angle_Tol))
/* Next steering angle to the RIGHT */
{
    *Look_Ahead_Steer_Left_Right = -1;
}

/* Default Steering Angle = zero (Straight Ahead) */
/* This condition would actually be an error since
   the next travel path entity is an ARC */
else
    *Look_Ahead_Steer_Left_Right = 0;
}
}

/*BTM******/
/* Compute the Steering Angle*/
/* This subroutine is used to compute the required steering angle for*/
/* the particular Tractor / Implement configuration given the desired*/
/* Implement turning radius */
void Compute_Steering_Angle
(long double *Look_Ahead_Steer_Angle,
 char Tractor_Steering_Type,
 char Tractor_Hitch_Type,
 char Tractor_Hitch_Swing_Point,
 char Tractor_Hitch_Location,
 double Tractor_Wheelbase,
 double Tractor_Drawbar_Length,
 double Tractor_Hitch_Swing_Distance,
 double Tractor_Max_Hitch_Swing_Angle,
 double Tractor_Max_Steering_Angle,
 double Implement_Hitch_Length,

```

```
float Crnt_Arc_Radius,
int Hitch_Position)
{
Boolean Err_Flag = FALSE;
/* Err_Flag used to determine if configuration is correct at process time */
int Error_Num;
double Implement_Turning_Radius;
Implement_Turning_Radius = Crnt_Arc_Radius;

switch (Tractor_Steering_Type)
{
case 'F':

switch (Tractor_Hitch_Type)
{
case 'S':

switch (Tractor_Hitch_Location)
{
case 'F':
Err_Flag = TRUE;
/* if Err_Flag == TRUE then the particular
configuration is not a valid setup */
break;

case 'C':
Err_Flag = TRUE;
break;

case 'R':

switch (Tractor_Hitch_Swing_Point)
{
case 'A':
```

```
Err_Flag = TRUE;
break;
```

```
case 'B':
```

```
/* To be included later if actually */
/* implemented. This configuration */
/* is not generally used. */
/*This configuration has not been implemented yet*/
Err_Flag = FALSE;
break;
```

```
case 'C':
```

```
Err_Flag = TRUE;
break;
```

```
case 'D':
```

```
Err_Flag = FALSE;
```

```
if ((Hitch_Position == 1) || (Hitch_Position == -1))
```

```
{
```

```
/* Hitch is at full swing
(left (+1) or right (-1)) */
```

```
*Look_Ahead_Steer_Angle =
```

```
(atan((Tractor_Wheelbase /
((sqrt((pow(Implement_Hitch_Length,2))
+ (pow(Tractor_Drawbar_Length,2))
+ (pow(Implement_Turning_Radius,2))
- (pow(Tractor_Hitch_Swing_Distance,2)))))))));
```

```
}
```

```
else if (Hitch_Position == 0)
```

```
{
```

```
/* Hitch is in freely swinging range of motion */
```

```
*Look_Ahead_Steer_Angle =  
  (atan((Tractor_Wheelbase /  
    ((sqrt((pow(Implement_Hitch_Length  
      + Tractor_Drawbar_Length,2))  
    + (pow(Implement_Turning_Radius,2))  
    - (pow(Tractor_Hitch_Swing_Distance,2)))))))));  
  }  
  
  else  
  {  
    printf("\a ERROR! \n");  
    printf("Hitch position sensor not working.\n");  
  }  
  
  break;  
  
  case 'N':  
    Err_Flag = TRUE;  
    break;  
  
  default:  
    Err_Flag = TRUE;  
    break;  
  
  }  
  break;  
  
  }  
  break;  
  
  case 'R':  
  
    switch (Tractor_Hitch_Location)  
    {  
      case 'F':
```

```
Err_Flag = TRUE;  
break;
```

```
case 'C':  
    Err_Flag = TRUE;  
    break;
```

```
case 'R':
```

```
    switch (Tractor_Hitch_Swing_Point)  
    {
```

```
        case 'A':  
            Err_Flag = TRUE;  
            break;
```

```
        case 'C':  
            Err_Flag = TRUE;  
            break;
```

```
        case 'B':
```

```
        case 'D':
```

```
        case 'N':  
            Err_Flag = FALSE;
```

```
        *Look_Ahead_Steer_Angle =  
            (atan ((Tractor_Wheelbase /  
                ((sqrt ((pow(Implement_Hitch_Length,2))  
                + (pow(Implement_Turning_Radius,2))  
                - (pow(Tractor_Drawbar_Length,2)))))))));
```

```
        break;
```

```
    default:
```

```
        Err_Flag = TRUE;  
        break;
```

```
    }
    break;

}
break;

case 'T':

switch (Tractor_Hitch_Location)
{
    case 'F':

        switch (Tractor_Hitch_Swing_Point)
        {
            case 'A':
                Err_Flag = TRUE;
                break;

            case 'B':
                Err_Flag = TRUE;
                break;

            case 'C':
                Err_Flag = FALSE;
                /*This configuration has not been implemented yet*/
                break;

            case 'D':
                Err_Flag = FALSE;
                /*This configuration has not been implemented yet*/
                break;

            case 'N':
                Err_Flag = TRUE;
```

```
        break;

    default:
        Err_Flag = TRUE;
        break;
}
break;

case 'C':
    Err_Flag = TRUE;
    break;

case 'R':

    switch (Tractor_Hitch_Swing_Point)
    {
        case 'A':
            Err_Flag = TRUE;
            break;

        case 'B':
            Err_Flag = TRUE;
            break;

        case 'C':
            Err_Flag = FALSE;
            /*This configuration has not been implemented yet*/
            break;

        case 'D':
            Err_Flag = FALSE;
            /*This configuration has not been implemented yet*/
            break;

        case 'N':
```



```
        break;

    }
    break;

}
break;

case 'A':

    switch (Tractor_Hitch_Type)
    {
        case 'S':

            switch (Tractor_Hitch_Location)
            {
                case 'F':
                    Err_Flag = TRUE;
                    break;

                case 'C':
                    Err_Flag = TRUE;
                    break;

                case 'R':

                    switch (Tractor_Hitch_Swing_Point)
                    {
                        case 'A':
                            Err_Flag = TRUE;
                            break;

                        case 'B':
                            Err_Flag = FALSE;
                            /*This configuration has not been implemented yet*/
                    }
                }
            }
        }
    }
}
```

```
        break;

    case 'C':
        Err_Flag = TRUE;
        break;

    case 'D':
        Err_Flag = FALSE;
        /*This configuration has not been implemented yet*/
        break;

    case 'N':
        Err_Flag = TRUE;
        break;

    default:
        Err_Flag = TRUE;
        break;

    }
    break;

}
break;

case 'R':

    switch (Tractor_Hitch_Location)
    {
        case 'F':
            Err_Flag = TRUE;
            break;

        case 'C':
            Err_Flag = TRUE;
```

```
break;

case 'R':

switch (Tractor_Hitch_Swing_Point)
{
case 'A':
    Err_Flag = TRUE;
    break;

case 'B':
    Err_Flag = FALSE;
    /*This configuration has not been implemented yet*/
    break;

case 'C':
    Err_Flag = TRUE;
    break;

case 'D':
    Err_Flag = FALSE;
    /*This configuration has not been implemented yet*/
    break;

case 'N':
    Err_Flag = FALSE;
    /*This configuration has not been implemented yet*/
    break;

default:
    Err_Flag = TRUE;
    break;

}
break;
```

```
    }  
    break;  
  
case 'T':  
  
    switch (Tractor_Hitch_Location)  
    {  
        case 'F':  
            Err_Flag = TRUE;  
            break;  
  
        case 'C':  
            Err_Flag = TRUE;  
            break;  
  
        case 'R':  
  
            switch (Tractor_Hitch_Swing_Point)  
            {  
                case 'A':  
                    Err_Flag = TRUE;  
                    break;  
  
                case 'B':  
                    Err_Flag = TRUE;  
                    break;  
  
                case 'C':  
                    Err_Flag = FALSE;  
                    /*This configuration has not been implemented yet*/  
                    break;  
  
                case 'D':  
                    Err_Flag = FALSE;
```

```
        /*This configuration has not been implemented yet*/
        break;

    case 'N':
        Err_Flag = TRUE;
        break;

    default:
        Err_Flag = TRUE;
        break;
}
break;

}
break;

case 'I':
    switch (Tractor_Hitch_Location)
    {
        case 'F':
        case 'C':
        case 'R':

            switch (Tractor_Hitch_Swing_Point)
            {
                case 'A':
                case 'B':
                case 'C':
                case 'D':
                    Err_Flag = FALSE;
                    /*This configuration has not been implemented yet*/
                    break;

                default:
                    Err_Flag = TRUE;
```

```
        break;
    }
    break;
}
break;

case 'S':

    switch (Tractor_Hitch_Type)
    {
        case 'S':

            switch (Tractor_Hitch_Location)
            {
                case 'F':
                    Err_Flag = TRUE;
                    break;

                case 'C':
                    Err_Flag = TRUE;
                    break;

                case 'R':

                    switch (Tractor_Hitch_Swing_Point)
                    {
                        case 'A':
                            Err_Flag = TRUE;
                            break;
```

```
    case 'B':  
        Err_Flag = FALSE;  
        /*This configuration has not been implemented yet*/  
        break;  
  
    case 'C':  
        Err_Flag = TRUE;  
        break;  
  
    case 'D':  
        Err_Flag = FALSE;  
        /*This configuration has not been implemented yet*/  
        break;  
  
    case 'N':  
        Err_Flag = TRUE;  
        break;  
  
    default:  
        Err_Flag = TRUE;  
        break;  
  
    }  
    break;  
  
}  
break;  
  
case 'R':  
  
    switch (Tractor_Hitch_Location)  
    {  
        case 'F':  
            Err_Flag = TRUE;  
            break;
```

```
case 'C':
    Err_Flag = TRUE;
    break;

case 'R':

    switch (Tractor_Hitch_Swing_Point)
    {
        case 'A':
            Err_Flag = TRUE;
            break;

        case 'B':
            Err_Flag = FALSE;
            /*This configuration has not been implemented yet*/
            break;

        case 'C':
            Err_Flag = TRUE;
            break;

        case 'D':
            Err_Flag = FALSE;
            /*This configuration has not been implemented yet*/
            break;

        case 'N':
            Err_Flag = FALSE;
            /*This configuration has not been implemented yet*/
            break;

        default:
            Err_Flag = TRUE;
            break;
```

```
    }  
    break;  
  
}  
break;  
  
case 'T':  
  
    switch (Tractor_Hitch_Location)  
    {  
        case 'F':  
            Err_Flag = TRUE;  
            break;  
  
        case 'C':  
            Err_Flag = TRUE;  
            break;  
  
        case 'R':  
  
            switch (Tractor_Hitch_Swing_Point)  
            {  
                case 'A':  
                    Err_Flag = TRUE;  
                    break;  
  
                case 'B':  
                    Err_Flag = TRUE;  
                    break;  
  
                case 'C':  
                    Err_Flag = FALSE;  
                    /*This configuration has not been implemented yet*/  
                    break;
```

```
    case 'D':
        Err_Flag = FALSE;
        /*This configuration has not been implemented yet*/
        break;

    case 'N':
        Err_Flag = TRUE;
        break;

    default:
        Err_Flag = TRUE;
        break;
}
break;

}
break;

case 'I':

    switch (Tractor_Hitch_Location)
    {
        case 'F':

            switch (Tractor_Hitch_Swing_Point)
            {
                case 'A':
                case 'B':
                case 'C':
                case 'D':
                    Err_Flag = FALSE;
                    /*This configuration has not been implemented yet*/
                    break;
```

```
        default:
            Err_Flag = TRUE;
            break;
    }
    break;

case 'C':
    Err_Flag = TRUE;
    break;

case 'R':

switch (Tractor_Hitch_Swing_Point)
{
    case 'A':
    case 'B':
    case 'C':
    case 'D':
        Err_Flag = FALSE;
        /*This configuration has not been implemented yet*/
        break;

    default:
        Err_Flag = TRUE;
        break;
}
break;
}
break;
}
break;
```

```
case 'L':
```

```
    switch (Tractor_Hitch_Type)
```

```
    {
```

```
        case 'S':
```

```
            switch (Tractor_Hitch_Location)
```

```
            {
```

```
                case 'F':
```

```
                    Err_Flag = TRUE;
```

```
                    break;
```

```
                case 'C':
```

```
                    Err_Flag = TRUE;
```

```
                    break;
```

```
                case 'R':
```

```
                    switch (Tractor_Hitch_Swing_Point)
```

```
                    {
```

```
                        case 'A':
```

```
                            Err_Flag = TRUE;
```

```
                            break;
```

```
                        case 'B':
```

```
                            Err_Flag = FALSE;
```

```
                            /*This configuration has not been implemented yet*/
```

```
                            break;
```

```
                        case 'C':
```

```
                            Err_Flag = TRUE;
```

```
                            break;
```

```
                        case 'D':
```

```
    Err_Flag = FALSE;
    /*This configuration has not been implemented yet*/
    break;

    case 'N':
        Err_Flag = TRUE;
        break;

    default:
        Err_Flag = TRUE;
        break;

    }
    break;

}
break;

case 'R':

    switch (Tractor_Hitch_Location)
    {
        case 'F':
            Err_Flag = TRUE;
            break;

        case 'C':
            Err_Flag = TRUE;
            break;

        case 'R':

            switch (Tractor_Hitch_Swing_Point)
            {
                case 'A':
```

```
Err_Flag = TRUE;  
break;
```

```
case 'B':  
    Err_Flag = FALSE;  
    /*This configuration has not been implemented yet*/  
    break;
```

```
case 'C':  
    Err_Flag = TRUE;  
    break;
```

```
case 'D':  
    Err_Flag = FALSE;  
    /*This configuration has not been implemented yet*/  
    break;
```

```
case 'N':  
    Err_Flag = FALSE;  
    /*This configuration has not been implemented yet*/  
    break;
```

```
default:  
    Err_Flag = TRUE;  
    break;
```

```
    }  
    break;
```

```
    }  
    break;
```

```
case 'T':
```

```
    switch (Tractor_Hitch_Location)
```

```
{
  case 'F':

    switch (Tractor_Hitch_Swing_Point)
    {
      case 'A':
        Err_Flag = TRUE;
        break;

      case 'B':
        Err_Flag = TRUE;
        break;

      case 'C':
        Err_Flag = FALSE;
        /*This configuration has not been implemented yet*/
        break;

      case 'D':
        Err_Flag = FALSE;
        /*This configuration has not been implemented yet*/
        break;

      case 'N':
        Err_Flag = TRUE;
        break;

      default:
        Err_Flag = TRUE;
        break;
    }
  break;

  case 'C':
    Err_Flag = TRUE;
```

```
break;

case 'R':

    switch (Tractor_Hitch_Swing_Point)
    {
        case 'A':
            Err_Flag = TRUE;
            break;

        case 'B':
            Err_Flag = TRUE;
            break;

        case 'C':
            Err_Flag = FALSE;
            /*This configuration has not been implemented yet*/
            break;

        case 'D':
            Err_Flag = FALSE;
            /*This configuration has not been implemented yet*/
            break;

        case 'N':
            Err_Flag = TRUE;
            break;

        default:
            Err_Flag = TRUE;
            break;
    }
    break;
}
```

```
break;

case 'I':

switch (Tractor_Hitch_Location)
{
case 'F':
case 'C':
case 'R':

switch (Tractor_Hitch_Swing_Point)
{
case 'A':
case 'B':
case 'C':
case 'D':
Err_Flag = FALSE;
/*This configuration has not been implemented yet*/
break;

default:
Err_Flag = TRUE;
break;

}
break;

}
break;

}
break;

case 'R':
```

```
switch (Tractor_Hitch_Type)
{
    case 'S':

        switch (Tractor_Hitch_Location)
        {
            case 'F':
                Err_Flag = TRUE;
                break;

            case 'C':
                Err_Flag = TRUE;
                break;

            case 'R':

                switch (Tractor_Hitch_Swing_Point)
                {
                    case 'A':
                        Err_Flag = TRUE;
                        break;

                    case 'B':
                        Err_Flag = FALSE;
                        /*This configuration has not been implemented yet*/
                        break;

                    case 'C':
                        Err_Flag = TRUE;
                        break;

                    case 'D':
                        Err_Flag = FALSE;
                        /*This configuration has not been implemented yet*/
                        break;
```

```
        case 'N':
            Err_Flag = TRUE;
            break;

        default:
            Err_Flag = TRUE;
            break;

    }
    break;

}
break;

case 'R':

    switch (Tractor_Hitch_Location)
    {
        case 'F':
            Err_Flag = TRUE;
            break;

        case 'C':
            Err_Flag = TRUE;
            break;

        case 'R':

            switch (Tractor_Hitch_Swing_Point)
            {
                case 'A':
                    Err_Flag = TRUE;
                    break;
```

```
    case 'B':  
        Err_Flag = FALSE;  
        /*This configuration has not been implemented yet*/  
        break;  
  
    case 'C':  
        Err_Flag = TRUE;  
        break;  
  
    case 'D':  
        Err_Flag = FALSE;  
        /*This configuration has not been implemented yet*/  
        break;  
  
    case 'N':  
        Err_Flag = FALSE;  
        /*This configuration has not been implemented yet*/  
        break;  
  
    default:  
        Err_Flag = TRUE;  
        break;  
    }  
    break;  
  
}  
break;  
  
case 'T':  
  
    switch (Tractor_Hitch_Location)  
    {  
        case 'F':
```

```
switch (Tractor_Hitch_Swing_Point)
{
  case 'A':
    Err_Flag = TRUE;
    break;

  case 'B':
    Err_Flag = TRUE;
    break;

  case 'C':
    Err_Flag = FALSE;
    /*This configuration has not been implemented yet*/
    break;

  case 'D':
    Err_Flag = FALSE;
    /*This configuration has not been implemented yet*/
    break;

  case 'N':
    Err_Flag = TRUE;
    break;

  default:
    Err_Flag = TRUE;
    break;
}
break;

case 'C':
  Err_Flag = TRUE;
  break;
```

```
case 'R':

    switch (Tractor_Hitch_Swing_Point)
    {
        case 'A':
            Err_Flag = TRUE;
            break;

        case 'B':
            Err_Flag = TRUE;
            break;

        case 'C':
            Err_Flag = FALSE;
            /*This configuration has not been implemented yet*/
            break;

        case 'D':
            Err_Flag = FALSE;
            /*This configuration has not been implemented yet*/
            break;

        case 'N':
            Err_Flag = TRUE;
            break;

        default:
            Err_Flag = TRUE;
            break;
    }
    break;
}
break;
```

```
case 'I':

    switch (Tractor_Hitch_Location)
    {
        case 'F':
        case 'C':
        case 'R':

            switch (Tractor_Hitch_Swing_Point)
            {
                case 'A':
                case 'B':
                case 'C':
                case 'D':
                    Err_Flag = FALSE;
                    /*This configuration has not been implemented yet*/
                    break;

                default:
                    Err_Flag = TRUE;
                    break;
            }
            break;
        }
        break;
    }
    break;

default:
    break;
}

if (Err_Flag == TRUE)
{
    Error_Num = 300;
    General_Error (Error_Num);
}
```

```

}
}

/*BTM*****
/* Compute the steering angle actually required to send to the */
/* Tractor steering mechanism (steering angle corrected for the */
/* actual travel path error factor as well as the "look ahead" */
/* steering correction factor */
void Error_Corrected_Steering_Angle
(double Travel_Path_Dist_Error,
 int Left_Right,
 int Look_Ahead_Steer_Left_Right,
 double GPS_Receiver_Tangent_Crrctn,
 double GPS_Left_Right_Dist,
 double Previous_Path_Error,
 float Tractor_Max_Steering_Angle,
 double Look_Ahead_Steer_Angle,
 long double *Actual_Output_Steering_Angle,
 float Straight_Ahead)
{
double Crctd_Travel_Path_Dist_Error;
float Rate_of_Error_Change;
int Prev_Left_Right;

    Crctd_Travel_Path_Dist_Error = Left_Right * Travel_Path_Dist_Error
+
        GPS_Receiver_Tangent_Crrctn;

    Rate_of_Error_Change =
        (Crctd_Travel_Path_Dist_Error - (Prev_Left_Right * Previous_Path_Error)
        / GPS_Update_Interval);

    if ((fabs(Crctd_Travel_Path_Dist_Error) <= Small_Travel_Path_Error ) &&
        (fabs(Rate_of_Error_Change) <= Small_Rate_of_Error_Change) &&
        (fabs(Look_Ahead_Steer_Angle) <= (Straight_Ahead + small )))

```

```

*Actual_Output_Steering_Angle = zero;

else

*Actual_Output_Steering_Angle =
  (Correction_Factor_1 * Rate_of_Error_Change
  + Correction_Factor_2 * Crctd_Travel_Path_Dist_Error)
  + Correction_Factor_3 * Look_Ahead_Steer_Angle
  * Look_Ahead_Steer_Left_Right;

if (abs(*Actual_Output_Steering_Angle) >= Tractor_Max_Steering_Angle)
  *Actual_Output_Steering_Angle = Tractor_Max_Steering_Angle;

Previous_Path_Error = Crctd_Travel_Path_Dist_Error;
Prev_Left_Right = Left_Right;

/* The Look_Ahead_Steer_Angle becomes the current steering angle
   required to follow an ARC travel path segment once the Tractor
   has acquired the next ARC, this is used until another travel
   path entity is acquired, then the Look_Ahead_Steering_Angle
   will be computed for the new travel path entity */
}

/*BTM*****
/* Send a Steering Feedback Output Signal */
void Steering_Output (long double Actual_Output_Steering_Angle)
{
  /* SOFTWARE OUTPUT TO HARDWARE */
  /* The Steering_Output is the signal sent to the steering hardware
  */
  /* Send an output signal to the steering servo mechanism */
}

/*BTM*****

```

```

/* Obtain the G.P.S. Positioning Data */
void GPS_Position_Data
(long double *X_Field_Coord,
 long double *Y_Field_Coord,
 long double *Z_Field_Coord,
 long double *Travel_Speed)
{
  /* HARDWARE AND EXTERNAL (GPS RECEIVER) SOFTWARE INPUT TO SOFTWARE */
  /* Obtain X, Y, Z Local (Field) Coordinates and the Travel Speed */
  /* from the G.P.S. receiver unit */
}

/*BTM***** */
/* Obtain the required G.I.S. Data */
void GIS_Data_Input
(long double X_Field_Coord,
 long double Y_Field_Coord,
 double GIS_Grid_Cell_Size)
{
  int X_GIS_Cell_Coord;
  int Y_GIS_Cell_Coord;

  /* Obtain the G.I.S. Data from the appropriate Database File */

  X_GIS_Cell_Coord = ((int)(X_Field_Coord / GIS_Grid_Cell_Size)) + 1;
  Y_GIS_Cell_Coord = ((int)(Y_Field_Coord / GIS_Grid_Cell_Size)) + 1;

  /* Retrieve data from the (X,Y) GIS_Cell_Coord */
  /* SOFTWARE INPUT */
}

/*BTM***** */
/* Record the new G.I.S. Data */
void GIS_Data_Output ()

```

```

{
    /* Send the newly obtained G.I.S. data to an output file */
    /* SOFTWARE OUTPUT */
}

/*BTM***** */
/* Calculate the Implement Control Code Signal */
void Determine_Implement_Control ()
{
    /* Calculate the determined, actual implement output control code */
}

/*BTM***** */
/* Send an Implement Control Code Signal */
void Implement_Control_Code (double Depth,
                             double Application_Rate,
                             double Aux_Contrl)
{
    /* SOFTWARE OUTPUT TO HARDWARE */
    /* The Implement_Control_Code is sent to the implement control hardware */
    /* Send the appropriate (determined) implement control code(s)
       to the control unit(s) for the implement */
}

/*BTM***** */
/* Obtain the Travel Path data, Field Status and G.I.S. Files */
void Info_Travel_Path_Field_GIS
(FILE **TPD_File_Pointer,
 FILE **GOS_File_Pointer,
 FILE **GIS_File_Pointer,

```

```
Boolean *Field_Travel_Done,  
Boolean *TRC_IMP_GPS_File_Done,  
Boolean *Travel_Path_File_Done,  
double *BasePoint_Degrees_Latitude,  
double *BasePoint_Minutes_Latitude,  
double *BasePoint_Seconds_Latitude,  
double *BasePoint_Degrees_Longitude,  
double *BasePoint_Minutes_Longitude,  
double *BasePoint_Seconds_Longitude,  
double *Startpoint_X_Coord,  
double *Startpoint_Y_Coord,  
double *Endpoint_X_Coord,  
double *Endpoint_Y_Coord,  
double *GIS_Grid_Cell_Size,  
int *Field_Travel_StartEntity,  
int *Field_Travel_EndEntity)
```

```
{
```

```
    char c;
```

```
    char File_Type;
```

```
    char File_Mode [5];
```

```
    char AUX_FILENAME [25];
```

```
    char FILENAME [25];
```

```
    FILE *Temp_TPD_File_Pointer;
```

```
    FILE *Temp_GIS_File_Pointer;
```

```
    FILE *Temp_GOS_File_Pointer;
```

```
do
```

```
{
```

```
clrscr ();
```

```

printf("\n\n\n\n\n\n Enter the name of the [.TPD] file that will be
used\n");
printf(" as the travel path data for the field to be farmed.\n\n\n");
printf(" The appropriate extension [.TPD] will be\n ");
printf(" automatically appended to the filename.\n");

```

```

File_Type = 'P';
sprintf (File_Mode,"r");

```

```

Open_File (&Temp_TPD_File_Pointer, AUX_FILENAME, File_Type, File_Mode,
FILENAME);

```

```

*TPD_File_Pointer = Temp_TPD_File_Pointer;

```

```

New_Continue_Travel_Path
(Field_Travel_Done,
TRC_IMP_GPS_File_Done,
Travel_Path_File_Done,
BasePoint_Degrees_Latitude,
BasePoint_Minutes_Latitude,
BasePoint_Seconds_Latitude,
BasePoint_Degrees_Longitude,
BasePoint_Minutes_Longitude,
BasePoint_Seconds_Longitude,
Startpoint_X_Coord,
Startpoint_Y_Coord,
Endpoint_X_Coord,
Endpoint_Y_Coord,
Field_Travel_StartEntity,
Field_Travel_EndEntity);

```

```

/* Open GIS input FILE */

```

```

clrscr ();
printf("\n\n\n\n\n\n Enter the length dimension of the G.I.S.\n");
printf(" database cells that are to be used in this field\n");

```

```
printf(" The units of measure have to be consistent with the \n");  
printf(" units of measure used in the other measurements of this program\n");
```

```
Double_Input (GIS_Grid_Cell_Size);
```

```
clrscr ();
```

```
printf("\n\n\n\n\n Enter the name of the [.GIS] file used as the\n");  
printf(" Geographic Information data INPUT file for the field.\n\n");  
printf(" The appropriate extension [.GIS] will be\n");  
printf(" automatically appended to the filename.\n");
```

```
File_Type = 'N';  
sprintf (File_Mode,"r");
```

```
Open_File (&Temp_GIS_File_Pointer, AUX_FILENAME, File_Type,  
File_Mode, FILENAME);
```

```
*GIS_File_Pointer = Temp_GIS_File_Pointer;
```

```
/* Open GIS output FILE */
```

```
clrscr ();  
printf("\n\n\n\n\n Enter the name of the [.GOS] file used as the\n");  
printf(" Geographic Information data OUTPUT file for the field.\n\n");  
printf(" The appropriate extension [.GOS] will be\n ");  
printf(" automatically appended to the filename.\n");
```

```
File_Type = 'O';  
sprintf (File_Mode,"w");
```

```
Open_File (&Temp_GOS_File_Pointer, AUX_FILENAME, File_Type,  
File_Mode, FILENAME);
```

```
*GOS_File_Pointer = Temp_GOS_File_Pointer;
```

```
clrscr ();
```

```
printf("\n\n\n\n\n Are the TRAVEL PATH DATA, \n");  
printf(" Field Configuration, and G.I.S. files correctly specified?\n\n");  
printf(" Enter [Y] Yes, or [N] No\n\n\n");  
printf(" ");
```

```
c = getc (stdin);  
fflush (stdin);  
c = toupper(c);
```

```
switch (c)
```

```
{
```

```
case 'Y':
```

```
    *Travel_Path_File_Done = TRUE;  
    *Field_Travel_Done = FALSE;  
    *TRC_IMP_GPS_File_Done = FALSE;  
    break;
```

```
case 'N':
```

```
    clrscr ();  
    printf("\n\n\n\n\n Enter the letter corresponding to the\n");  
    printf(" configuration file to be created.\n");
```

```
    printf(" [S] TRAVEL PATH DATA FILE \n");  
    printf(" [C] FIELD CONFIGURATION FILE \n");  
    printf(" [G] G.I.S. Database input file \n");
```

```
    c = getc (stdin);  
    fflush (stdin);  
    c = toupper(c);
```

```
    switch (c)
```

```
{
```

```
case 'S':
    *Travel_Path_File_Done = TRUE;
    *Field_Travel_Done = TRUE;
    *TRC_IMP_GPS_File_Done = TRUE;
    printf(" Correct the Travel Path Data File, ");
    printf("then proceed.\n");
    delay (Standard_Time_Delay);
    break;

case 'C':
    *Travel_Path_File_Done = FALSE;
    *Field_Travel_Done = FALSE;
    *TRC_IMP_GPS_File_Done = FALSE;
    Create_Field_Status_Info ();
    break;

case 'G':
    *TRC_IMP_GPS_File_Done = TRUE;
    *Travel_Path_File_Done = TRUE;
    *Field_Travel_Done = TRUE;
    printf(" Correct the G.I.S. File, ");
    printf("then proceed.\n");
    delay (Standard_Time_Delay);
    break;

default:
    printf("\a");
    break;
}

clrscr ();
break;

default:
    printf("\a");
```

```
        break;
    }
} while (!*Travel_Path_File_Done);
}
```

```
/*BTM*****
```

```
/* New_Continue_Travel_Path */
```

```
void New_Continue_Travel_Path
(Boolean *Field_Travel_Done,
 Boolean *TRC_IMP_GPS_File_Done,
 Boolean *Travel_Path_File_Done,
 double *BasePoint_Degrees_Latitude,
 double *BasePoint_Minutes_Latitude,
 double *BasePoint_Seconds_Latitude,
 double *BasePoint_Degrees_Longitude,
 double *BasePoint_Minutes_Longitude,
 double *BasePoint_Seconds_Longitude,
 double *Startpoint_X_Coord,
 double *Startpoint_Y_Coord,
 double *Endpoint_X_Coord,
 double *Endpoint_Y_Coord,
 int *Field_Travel_StartEntity,
 int *Field_Travel_EndEntity)
```

```
{
```

```
    char c;
    Boolean Done = FALSE;
    char File_Type;
    char File_Mode [5];
```

```
do
{
    clrscr ();
```

```

printf("\n\n\n\n\n\n      FARM FIELD\n\n\n\n");
printf(" [N] New field status configuration creation required.\n");
printf(" [C] Farm a previously defined field travel path\n");
printf("      with the field status configuration to be specified next");
printf("\n\n\n\n\n\n");

```

```

printf(" Press the letter corresponding to the operation to do.\n\n");
printf("      ");

```

```

c = getc (stdin);
fflush (stdin);
c = toupper(c);

```

```

switch (c)

```

```

{

```

```

    case 'N':

```

```

        clrscr ();

```

```

        delay (Standard_Time_Delay/4);

```

```

        printf("\n\n Create the new field status configuration\n");

```

```

        printf(" file, then proceed.\n\n");

```

```

        Continue_Program ();

```

```

        Create_Field_Status_Info ();

```

```

        *Field_Travel_Done = TRUE;

```

```

        *TRC_IMP_GPS_File_Done = TRUE;

```

```

        *Travel_Path_File_Done = FALSE;

```

```

        Done = TRUE;

```

```

        break;

```

```

    case 'C':

```

```

        clrscr ();

```

```

printf("\n\n\n\n\n\n");

```

```
printf(" Continue farming with a previous field status configuration.\n\n");
```

```
printf("\n\n\n\n\n Enter the name of the [.FSC] file that will be  
used\n");
```

```
printf(" as the field status configuration file.\n\n");
```

```
printf(" The appropriate extension [.FSC] will be\n ");
```

```
printf(" automatically appended to the filename.\n");
```

```
File_Type = 'S';
```

```
sprintf (File_Mode,"r");
```

```
Open_Field_Status_Info (File_Type, File_Mode,  
BasePoint_Degrees_Latitude,  
BasePoint_Minutes_Latitude,  
BasePoint_Seconds_Latitude,  
BasePoint_Degrees_Longitude,  
BasePoint_Minutes_Longitude,  
BasePoint_Seconds_Longitude,  
Startpoint_X_Coord,  
Startpoint_Y_Coord,  
Endpoint_X_Coord,  
Endpoint_Y_Coord,  
Field_Travel_StartEntity,  
Field_Travel_EndEntity);
```

```
clrscr ();
```

```
Field_Status_Info (*BasePoint_Degrees_Latitude,  
*BasePoint_Minutes_Latitude,  
*BasePoint_Seconds_Latitude,  
*BasePoint_Degrees_Longitude,  
*BasePoint_Minutes_Longitude,  
*BasePoint_Seconds_Longitude,  
*Startpoint_X_Coord,  
*Startpoint_Y_Coord,
```



```
double *GPS_Receiver_Height,  
double *GPS_Forward_Backward_Dist,  
double *GPS_Left_Right_Dist,  
char *GPS_Forward_Backward_Dir,  
char *GPS_Left_Right_Dir)  
{  
  
char c;  
char File_Type;  
char File_Mode [5];  
  
while (*TRC_IMP_GPS_File_Done == FALSE)  
{  
  
clrscr ();  
  
printf("\n\n\n\n\n Enter the name of the [.TRC] file corresponding\n");  
printf(" to the TRACTOR configuration used to farm the field.\n\n");  
printf(" The appropriate extension [.TRC] will be\n ");  
printf(" automatically be appended to the filename.\n");  
  
File_Type = 'T';  
sprintf(File_Mode,"r");  
  
Open_Tractor_Config_File (File_Type, File_Mode,  
Tractor_Steering_Type,  
Tractor_Hitch_Type,  
Tractor_Hitch_Swing_Point,  
Tractor_Hitch_Location,  
Tractor_Wheelbase,  
Tractor_Drawbar_Length,  
Tractor_Hitch_Swing_Distance,  
Tractor_Max_Hitch_Swing_Angle,  
Tractor_Max_Steering_Angle);
```

```
clrscr ();
```

```
printf("\n\n\n\n\n Enter the name of the [.IMP] file corresponding\n");  
printf(" to the IMPLEMENT / GPS Receiver configuration\n");  
printf(" that will be used to farm the field.\n\n");  
printf(" The appropriate extension [.IMP] will be\n");  
printf(" automatically appended to the filename.\n");
```

```
File_Type = 'I';  
sprintf(File_Mode,"r");
```

```
Open_Implement_GPS_Config_File (File_Type, File_Mode,  
                                Implement_Hitch_Length,  
                                Implement_Working_Width,  
                                GPS_Receiver_Height,  
                                GPS_Forward_Backward_Dist,  
                                GPS_Left_Right_Dist,  
                                GPS_Forward_Backward_Dir,  
                                GPS_Left_Right_Dir);
```

```
clrscr ();
```

```
Tractor_File_Info (*Tractor_Steering_Type,  
                  *Tractor_Hitch_Type,  
                  *Tractor_Hitch_Swing_Point,  
                  *Tractor_Hitch_Location,  
                  *Tractor_Wheelbase,  
                  *Tractor_Drawbar_Length,  
                  *Tractor_Hitch_Swing_Distance,  
                  *Tractor_Max_Hitch_Swing_Angle,  
                  *Tractor_Max_Steering_Angle);
```

```
Continue_Program ();
```

```
Implement_GPS_File_Info (*Implement_Hitch_Length,
```

```
*Implement_Working_Width,  
*GPS_Receiver_Height,  
*GPS_Forward_Backward_Dist,  
*GPS_Left_Right_Dist,  
*GPS_Forward_Backward_Dir,  
*GPS_Left_Right_Dir);
```

```
Continue_Program ();
```

```
clrscr ();
```

```
printf("\n\n\n\n\n Are the TRACTOR & IMPLEMENT / GPS \n");  
printf(" configuration files correctly specified?\n\n");  
printf(" Enter [Y] Yes, or [N] No\n\n");  
printf(" ");
```

```
c = getc (stdin);  
fflush (stdin);  
c = toupper(c);
```

```
switch (c)
```

```
{
```

```
case 'Y':
```

```
*TRC_IMP_GPS_File_Done = TRUE;  
*Travel_Path_File_Done = TRUE;  
*Field_Travel_Done = FALSE;  
break;
```

```
case 'N':
```

```
clrscr ();  
printf("\n\n\n\n\n Enter the letter corresponding to the\n");  
printf(" configuration file that is to be created\n");
```

```
printf(" [T] TRACTOR \n");  
printf(" [I] IMPLEMENT / GPS Receiver \n\n\n");
```

```
c = getc (stdin);  
fflush (stdin);  
c = toupper(c);
```

```
switch (c)
```

```
{
```

```
    case 'T':
```

```
        Create_Tractor_Configuration ();
```

```
        break;
```

```
    case 'I':
```

```
        Create_Implement_GPS_Configuration ();
```

```
        break;
```

```
    default:
```

```
        printf("\a");
```

```
        break;
```

```
}
```

```
*TRC_IMP_GPS_File_Done = FALSE;
```

```
*Travel_Path_File_Done = TRUE;
```

```
*Field_Travel_Done = FALSE;
```

```
clrscr ();
```

```
break;
```

```
default:
```

```
    printf("\a");
```

```
    break;
```

```
}
```

```
}
```

```
}
```

```

/*BTM*****
/* Arc startpoint and endpoint calculations */
void Calc_Arc_Start_End_points
(float Crnt_Arc_Radius,
 float Crnt_Start_Angle,
 float Crnt_End_Angle,
 float Next_Arc_Radius,
 float Next_Start_Angle,
 float Next_End_Angle,
 float Crnt_Primary_X_Coord,
 float Crnt_Primary_Y_Coord,
 float Next_Primary_X_Coord,
 float Next_Primary_Y_Coord,
 float *Crnt_Arc_Startpoint_X_Coord,
 float *Crnt_Arc_Startpoint_Y_Coord,
 float *Next_Arc_Startpoint_X_Coord,
 float *Next_Arc_Startpoint_Y_Coord,
 float *Crnt_Arc_Endpoint_X_Coord,
 float *Crnt_Arc_Endpoint_Y_Coord,
 float *Next_Arc_Endpoint_X_Coord,
 float *Next_Arc_Endpoint_Y_Coord)

{

*Crnt_Arc_Startpoint_X_Coord = Crnt_Primary_X_Coord + (Crnt_Arc_Radius *
cos(Crnt_Start_Angle*(2*pi/360)));

*Crnt_Arc_Endpoint_X_Coord = Crnt_Primary_X_Coord + (Crnt_Arc_Radius *
cos(Crnt_End_Angle*(2*pi/360)));

*Crnt_Arc_Startpoint_Y_Coord = Crnt_Primary_Y_Coord + (Crnt_Arc_Radius *
sin(Crnt_Start_Angle*(2*pi/360)));

*Crnt_Arc_Endpoint_Y_Coord = Crnt_Primary_Y_Coord + (Crnt_Arc_Radius *
sin(Crnt_End_Angle*(2*pi/360)));

```

```

*Next_Arc_Startpoint_X_Coord = Next_Primary_X_Coord + (Next_Arc_Radius *
    cos(Next_Start_Angle*(2*pi/360)));

*Next_Arc_Endpoint_X_Coord = Next_Primary_X_Coord + (Next_Arc_Radius *
    cos(Next_End_Angle*(2*pi/360)));

*Next_Arc_Startpoint_Y_Coord = Next_Primary_Y_Coord + (Next_Arc_Radius *
    sin(Next_Start_Angle*(2*pi/360)));

*Next_Arc_Endpoint_Y_Coord = Next_Primary_Y_Coord + (Next_Arc_Radius *
    sin(Next_End_Angle*(2*pi/360)));
}

/*BTM*****
/* Check to see if need to acquire another TPD entity */
void Check_Acquire_TPD_Entity
( Boolean *Get_Next_Entity,
  char Next_TPD_Section_Name [],
  FILE *TPD_File_Pointer,
  char Next_Layer_Name [],
  char Next_Linetype_Impl_UpDown [],
  float *Next_CAD_Elevation,
  float *Next_Line_Thickness,
  int *Next_Entity_Handle,
  int *Next_Color,
  float *Next_Primary_X_Coord,
  float *Next_Primary_Y_Coord,
  float *Next_Primary_Z_Coord,
  float *Next_Secondary_X_Coord,
  float *Next_Secondary_Y_Coord,
  float *Next_Secondary_Z_Coord,
  float *Next_Arc_Radius,
  float *Next_Start_Angle,
  float *Next_End_Angle,

```

```
char TPD_Section_Name [],
char Crnt_TPD_Section_Name [],
char Crnt_Layer_Name [],
char Crnt_Linetype_Impl_UpDown [],
float *Crnt_CAD_Elevation,
float *Crnt_Line_Thickness,
int *Crnt_Entity_Handle,
int *Crnt_Color,
float *Crnt_Primary_X_Coord,
float *Crnt_Primary_Y_Coord,
float *Crnt_Primary_Z_Coord,
float *Crnt_Secondary_X_Coord,
float *Crnt_Secondary_Y_Coord,
float *Crnt_Secondary_Z_Coord,
float *Crnt_Arc_Radius,
float *Crnt_Start_Angle,
float *Crnt_End_Angle,
float Crnt_X_Strt,
float Crnt_Y_Strt,
float Crnt_X_End,
float Crnt_Y_End,
long double Travel_Speed,
long double X_Field_Coord,
long double Y_Field_Coord,
double Travel_Path_Dist_Error,
double GPS_Forward_Backward_Dist,
double GPS_Left_Right_Dist,
char GPS_Forward_Backward_Dir,
char GPS_Left_Right_Dir,
Boolean *Field_Travel_Done,
int Field_Travel_EndEntity)

{
double TPDist_Remaining;
float Line_Length;
```

```
float Field_Coord_to_End_Pt_Dist;  
float X_LINE_Vect;  
float Y_LINE_Vect;  
float X_Field_to_EndPt_Vect;  
float Y_Field_to_EndPt_Vect;  
float Vector_Dot_Product;  
float X_Coord_ARCenter_End_Vect;  
float Y_Coord_ARCenter_End_Vect;  
float X_Coord_ARCenter_GPS_Pt_Vect;  
float Y_Coord_ARCenter_GPS_Pt_Vect;  
float ARCenter_to_EndPt_Dist;  
float ARCenter_to_GPS_Pt_Dist;  
float Vector_Check_Angle;  
float TPD_Angle_Remaining;
```

```
/* Get_Next_Entity is assumed FALSE until calculated to be TRUE */  
*Get_Next_Entity = FALSE;
```

```
/* Compute the dist to go, look @ point to line or arc functions */  
/* TPDist_Remaining may also be past the endpoint of the current */  
/* travel path entity, in which case will have to acquire one or */  
/* more travel path entities to "catch up" to the current position */  
/* TPDist_Remaining = ? */  
/* Use [VECTOR DOT PRODUCT METHOD] for LINE "look ahead" determination */
```

```
if (strcmp(Crnt_TPD_Section_Name,"LINE"))  
{
```

```
    Line_Length = Distance_Length (*Crnt_Primary_X_Coord,  
                                   *Crnt_Primary_Y_Coord,  
                                   *Crnt_Secondary_X_Coord,  
                                   *Crnt_Secondary_Y_Coord);
```

```
    X_LINE_Vect = (Crnt_X_End - Crnt_X_Strt);
```

```

Y_LINE_Vect = (Crnt_Y_End - Crnt_Y_Strt);

X_Field_to_EndPt_Vect = (X_Field_Coord - Crnt_X_End);

Y_Field_to_EndPt_Vect = (Y_Field_Coord - Crnt_Y_End);

Vector_Dot_Product = (X_LINE_Vect * X_Field_to_EndPt_Vect)
    + (Y_LINE_Vect * Y_Field_to_EndPt_Vect);

if (GPS_Forward_Backward_Dir == 'F')
    TPDist_Remaining = fabs (Vector_Dot_Product / Line_Length) +
        GPS_Forward_Backward_Dist;
else if (GPS_Forward_Backward_Dir == 'B')
    TPDist_Remaining = fabs (Vector_Dot_Product / Line_Length) -
        GPS_Forward_Backward_Dist;

    if (TPDist_Remaining <= (Travel_Speed * GPS_Update_Interval))
        *Get_Next_Entity = TRUE;

}

else if (!strcmp(Crnt_TPD_Section_Name,"ARC"))
    {

        X_Coord_ARCenter_End_Vect = (Crnt_X_End - *Crnt_Primary_X_Coord);

        Y_Coord_ARCenter_End_Vect = (Crnt_Y_End - *Crnt_Primary_Y_Coord);

        X_Coord_ARCenter_GPS_Pt_Vect = (X_Field_Coord - *Crnt_Primary_X_Coord);

        Y_Coord_ARCenter_GPS_Pt_Vect = (Y_Field_Coord - *Crnt_Primary_Y_Coord);

        ARCenter_to_EndPt_Dist = Distance_Length
            (*Crnt_Primary_X_Coord,
            *Crnt_Primary_Y_Coord,

```

```
Crnt_X_End,
Crnt_Y_End);
```

```
ARCenter_to_GPS_Pt_Dist = Distance_Length
(*Crnt_Primary_X_Coord,
*Crnt_Primary_Y_Coord,
X_Field_Coord,
Y_Field_Coord);
```

```
if (GPS_Left_Right_Dir == 'L')
    ARCenter_to_GPS_Pt_Dist = ARCenter_to_GPS_Pt_Dist +
        GPS_Left_Right_Dist;
else if (GPS_Left_Right_Dir == 'R')
    ARCenter_to_GPS_Pt_Dist = ARCenter_to_GPS_Pt_Dist -
        GPS_Left_Right_Dist;
```

```
Vector_Dot_Product =
(X_Coord_ARCenter_End_Vect * X_Coord_ARCenter_GPS_Pt_Vect)
+
(Y_Coord_ARCenter_End_Vect * Y_Coord_ARCenter_GPS_Pt_Vect);
```

```
Vector_Check_Angle = abs(acos (Vector_Dot_Product /
(ARCenter_to_EndPt_Dist * ARCenter_to_GPS_Pt_Dist)));
```

```
TPD_Angle_Remaining = (Travel_Speed / *Crnt_Arc_Radius) *
(GPS_Update_Interval);
```

```
if (TPD_Angle_Remaining <= Vector_Check_Angle)
    *Get_Next_Entity = TRUE;
}

else
{
*Get_Next_Entity = FALSE;
}
```

```
if (*Get_Next_Entity == TRUE)
```

```
{
```

```
Obtain_Entity_Para_Info
```

```
(TPD_Section_Name,
```

```
Next_TPD_Section_Name,
```

```
&TPD_File_Pointer,
```

```
Next_Layer_Name,
```

```
Next_Linetype_Impl_UpDown,
```

```
Next_CAD_Elevation,
```

```
Next_Line_Thickness,
```

```
Next_Entity_Handle,
```

```
Next_Color,
```

```
Next_Primary_X_Coord,
```

```
Next_Primary_Y_Coord,
```

```
Next_Primary_Z_Coord,
```

```
Next_Secondary_X_Coord,
```

```
Next_Secondary_Y_Coord,
```

```
Next_Secondary_Z_Coord,
```

```
Next_Arc_Radius,
```

```
Next_Start_Angle,
```

```
Next_End_Angle);
```

```
}
```

```
/* The entity type POINT is not currently used, but could  
easily be implemented in the future for reference purposes  
by using the Next_Primary X and Y coordinates obtained  
here, and renaming them to use elsewhere in this program */
```

```
if (!strcmp(Next_TPD_Section_Name,"POINT"))
```

```
Obtain_Entity_Para_Info
```

```
(TPD_Section_Name,
```

```
Next_TPD_Section_Name,
```

```
&TPD_File_Pointer,
```

```
Next_Layer_Name,
```

```

Next_Linetype_Impl_UpDown,
Next_CAD_Elevation,
Next_Line_Thickness,
Next_Entity_Handle,
Next_Color,
Next_Primary_X_Coord,
Next_Primary_Y_Coord,
Next_Primary_Z_Coord,
Next_Secondary_X_Coord,
Next_Secondary_Y_Coord,
Next_Secondary_Z_Coord,
Next_Arc_Radius,
Next_Start_Angle,
Next_End_Angle);

```

```

if (strcmp(Next_TPD_Section_Name,"ENDSEC") != 0 ||
    (*Next_Entity_Handle == Field_Travel_EndEntity))
    *Field_Travel_Done = TRUE;

```

```

else

```

```

{

```

```

strcpy(Crnt_TPD_Section_Name , Next_TPD_Section_Name);
strcpy(Next_TPD_Section_Name , TPD_Section_Name);
strcpy(Crnt_Layer_Name , Next_Layer_Name);
strcpy(Crnt_Linetype_Impl_UpDown, Next_Linetype_Impl_UpDown);

```

```

*Crnt_CAD_Elevation      = *Next_CAD_Elevation;
*Crnt_Line_Thickness     = *Next_Line_Thickness;
*Crnt_Entity_Handle      = *Next_Entity_Handle;
*Crnt_Color              = *Next_Color;
*Crnt_Primary_X_Coord    = *Next_Primary_X_Coord;
*Crnt_Primary_Y_Coord    = *Next_Primary_Y_Coord;
*Crnt_Primary_Z_Coord    = *Next_Primary_Z_Coord;
*Crnt_Secondary_X_Coord  = *Next_Secondary_X_Coord;

```

```

*Crnt_Secondary_Y_Coord    = *Next_Secondary_Y_Coord;
*Crnt_Secondary_Z_Coord    = *Next_Secondary_Z_Coord;
*Crnt_Arc_Radius           = *Next_Arc_Radius;
*Crnt_Start_Angle          = *Next_Start_Angle;
*Crnt_End_Angle            = *Next_End_Angle;
}
}

/*BTM***** */
/* Check to determine if the implement is located right or left
   of the current travel path LINE entity [VECTOR METHOD] */
void Right_or_Left_of_Travel_Path
(int *Left_Right,
 long double X_Field_Coord,
 long double Y_Field_Coord,
 float *Crnt_X_Strt,
 float *Crnt_Y_Strt,
 float *Crnt_X_End,
 float *Crnt_Y_End)

{
float X_Coord_Line_Path_Vect;
float Y_Coord_Line_Path_Vect;
float X_Coord_Err_Dist_Vect;
float Y_Coord_Err_Dist_Vect;
float Vector_Cross_Prod;

X_Coord_Line_Path_Vect = (*Crnt_X_End - *Crnt_X_Strt);

Y_Coord_Line_Path_Vect = (*Crnt_Y_End - *Crnt_Y_Strt);

X_Coord_Err_Dist_Vect = (X_Field_Coord - *Crnt_X_Strt);

Y_Coord_Err_Dist_Vect = (Y_Field_Coord - *Crnt_Y_Strt);

```

```
/* If this Vector Cross Product is < 0 then are RIGHT of the
target travel path and therefore need to
steer LEFT in accordance with the Right Hand Rule */
```

```
Vector_Cross_Prod =
((X_Coord_Line_Path_Vect) * (Y_Coord_Err_Dist_Vect) -
(Y_Coord_Err_Dist_Vect) * (X_Coord_Line_Path_Vect));
```

```
if (Vector_Cross_Prod > zero + small)
    *Left_Right = -1;
```

```
else if (Vector_Cross_Prod < zero - small)
    *Left_Right = 1;
```

```
else
    *Left_Right = 0;
```

```
}
```

```
/*BTM*****
```

```
/* Determine what the current travel path's actual start and end point
coordinates are with respect to the direction of travel,
NOT corresponding to the direction in which the entities
were originally drawn in the CADD program */
```

```
void Crnt_Pth_Sgmnt_Strt_End_Pts
(char Crnt_TPD_Section_Name [],
char Next_TPD_Section_Name [],
float Crnt_Primary_X_Coord,
float Crnt_Primary_Y_Coord,
float Crnt_Secondary_X_Coord,
float Crnt_Secondary_Y_Coord,
float Crnt_Arc_Radius,
float Crnt_Start_Angle,
float Crnt_End_Angle,
```

```
float Next_Primary_X_Coord,  
float Next_Primary_Y_Coord,  
float Next_Secondary_X_Coord,  
float Next_Secondary_Y_Coord,  
float Next_Arc_Radius,  
float Next_Start_Angle,  
float Next_End_Angle,  
float *Crnt_X_Strt,  
float *Crnt_Y_Strt,  
float *Crnt_X_End,  
float *Crnt_Y_End,  
float *Previous_X_End,  
float *Previous_Y_End)  
{  
float Next_Arc_Startpoint_X_Coord;  
float Next_Arc_Endpoint_X_Coord;  
float Next_Arc_Startpoint_Y_Coord;  
float Next_Arc_Endpoint_Y_Coord;  
  
float Crnt_Arc_Startpoint_X_Coord;  
float Crnt_Arc_Endpoint_X_Coord;  
float Crnt_Arc_Startpoint_Y_Coord;  
float Crnt_Arc_Endpoint_Y_Coord;  
  
if (!strcmp(Crnt_TPD_Section_Name,"POINT"))  
  
    {  
        *Crnt_X_Strt = Crnt_Primary_X_Coord;  
        *Crnt_Y_Strt = Crnt_Primary_Y_Coord;  
        *Crnt_X_End = *Crnt_X_Strt;  
        *Crnt_Y_End = *Crnt_Y_Strt;  
    }  
  
if ((!strcmp(Crnt_TPD_Section_Name,"LINE")) &&
```

```

(!strcmp(Next_TPD_Section_Name,"ARC"))
{
  *Crnt_X_Strt = *Previous_X_End;
  *Crnt_Y_Strt = *Previous_Y_End;

  Next_Arc_Startpoint_X_Coord = Next_Primary_X_Coord + (Next_Arc_Radius *
    cos(Next_Start_Angle*(2*pi/360)));

  Next_Arc_Endpoint_X_Coord = Next_Primary_X_Coord + (Next_Arc_Radius *
    cos(Next_End_Angle*(2*pi/360)));

  Next_Arc_Startpoint_Y_Coord = Next_Primary_Y_Coord + (Next_Arc_Radius *
    sin(Next_Start_Angle*(2*pi/360)));

  Next_Arc_Endpoint_Y_Coord = Next_Primary_Y_Coord + (Next_Arc_Radius *
    sin(Next_End_Angle*(2*pi/360)));

  if(((Crnt_Primary_X_Coord <= Next_Arc_Startpoint_X_Coord + Dist_Tlrc) &&
    (Crnt_Primary_X_Coord >= Next_Arc_Startpoint_X_Coord - Dist_Tlrc) &&
    (Crnt_Primary_Y_Coord <= Next_Arc_Startpoint_Y_Coord + Dist_Tlrc) &&
    (Crnt_Primary_Y_Coord >= Next_Arc_Startpoint_Y_Coord - Dist_Tlrc))
    ||
    ((Crnt_Primary_X_Coord <= Next_Arc_Endpoint_X_Coord + Dist_Tlrc) &&
    (Crnt_Primary_X_Coord >= Next_Arc_Endpoint_X_Coord - Dist_Tlrc) &&
    (Crnt_Primary_Y_Coord <= Next_Arc_Endpoint_Y_Coord + Dist_Tlrc) &&
    (Crnt_Primary_Y_Coord >= Next_Arc_Endpoint_Y_Coord - Dist_Tlrc)))
  {
    *Crnt_X_End = Crnt_Primary_X_Coord;
    *Crnt_Y_End = Crnt_Primary_Y_Coord;
  }

  else if
  (((Crnt_Secondary_X_Coord <= Next_Arc_Startpoint_X_Coord + Dist_Tlrc) &&
    (Crnt_Secondary_X_Coord >= Next_Arc_Startpoint_X_Coord - Dist_Tlrc) &&

```

```

(Crnt_Secondary_Y_Coord <= Next_Arc_Startpoint_Y_Coord + Dist_Tlrc) &&
(Crnt_Secondary_Y_Coord >= Next_Arc_Startpoint_Y_Coord - Dist_Tlrc))
    ||
((Crnt_Secondary_X_Coord <= Next_Arc_Endpoint_X_Coord + Dist_Tlrc) &&
(Crnt_Secondary_X_Coord >= Next_Arc_Endpoint_X_Coord - Dist_Tlrc) &&
(Crnt_Secondary_Y_Coord <= Next_Arc_Endpoint_Y_Coord + Dist_Tlrc) &&
(Crnt_Secondary_Y_Coord >= Next_Arc_Endpoint_Y_Coord - Dist_Tlrc))

    {
        *Crnt_X_End = Crnt_Secondary_X_Coord;
        *Crnt_Y_End = Crnt_Secondary_Y_Coord;
    }
}

if ((!strcmp(Crnt_TPD_Section_Name,"ARC")) &&
    (!strcmp(Next_TPD_Section_Name,"LINE")))
{

*Crnt_X_Strt = *Previous_X_End;
*Crnt_Y_Strt = *Previous_Y_End;

Crnt_Arc_Startpoint_X_Coord = Crnt_Primary_X_Coord + (Crnt_Arc_Radius *
    cos(Crnt_Start_Angle*(2*pi/360)));

Crnt_Arc_Endpoint_X_Coord = Crnt_Primary_X_Coord + (Crnt_Arc_Radius *
    cos(Crnt_End_Angle*(2*pi/360)));

Crnt_Arc_Startpoint_Y_Coord = Crnt_Primary_Y_Coord + (Crnt_Arc_Radius *
    sin(Crnt_Start_Angle*(2*pi/360)));

Crnt_Arc_Endpoint_Y_Coord = Crnt_Primary_Y_Coord + (Crnt_Arc_Radius *
    sin(Crnt_End_Angle*(2*pi/360)));

if(((Next_Primary_X_Coord <= Crnt_Arc_Startpoint_X_Coord + Dist_Tlrc) &&
    (Next_Primary_X_Coord >= Crnt_Arc_Startpoint_X_Coord - Dist_Tlrc) &&

```

```
(Next_Primary_Y_Coord <= Crnt_Arc_Startpoint_Y_Coord + Dist_Tlrnc) &&
(Next_Primary_Y_Coord >= Crnt_Arc_Startpoint_Y_Coord - Dist_Tlrnc))
```

```
||
```

```
((Next_Primary_X_Coord <= Crnt_Arc_Endpoint_X_Coord + Dist_Tlrnc) &&
(Next_Primary_X_Coord >= Crnt_Arc_Endpoint_X_Coord - Dist_Tlrnc) &&
(Next_Primary_Y_Coord <= Crnt_Arc_Endpoint_Y_Coord + Dist_Tlrnc) &&
(Next_Primary_Y_Coord >= Crnt_Arc_Endpoint_Y_Coord - Dist_Tlrnc)))
```

```
{
```

```
*Crnt_X_End = Next_Primary_X_Coord;
```

```
*Crnt_Y_End = Next_Primary_Y_Coord;
```

```
}
```

```
else if
```

```
((Next_Secondary_X_Coord <= Crnt_Arc_Startpoint_X_Coord + Dist_Tlrnc) &&
(Next_Secondary_X_Coord >= Crnt_Arc_Startpoint_X_Coord - Dist_Tlrnc) &&
(Next_Secondary_Y_Coord <= Crnt_Arc_Startpoint_Y_Coord + Dist_Tlrnc) &&
(Next_Secondary_Y_Coord >= Crnt_Arc_Startpoint_Y_Coord - Dist_Tlrnc))
```

```
||
```

```
((Next_Secondary_X_Coord <= Crnt_Arc_Endpoint_X_Coord + Dist_Tlrnc) &&
(Next_Secondary_X_Coord >= Crnt_Arc_Endpoint_X_Coord - Dist_Tlrnc) &&
(Next_Secondary_Y_Coord <= Crnt_Arc_Endpoint_Y_Coord + Dist_Tlrnc) &&
(Next_Secondary_Y_Coord >= Crnt_Arc_Endpoint_Y_Coord - Dist_Tlrnc)))
```

```
{
```

```
*Crnt_X_End = Next_Secondary_X_Coord;
```

```
*Crnt_Y_End = Next_Secondary_Y_Coord;
```

```
}
```

```
}
```

```
if(((!strcmp(Crnt_TPD_Section_Name,"LINE")) ||
```

```
!strcmp(Crnt_TPD_Section_Name,"ARC"))
```

```
&&
```

```
!strcmp(Next_TPD_Section_Name,"POINT"))
```

```
{
```

```
*Crnt_X_Strt = *Previous_X_End;
*Crnt_Y_Strt = *Previous_Y_End;
*Crnt_X_End = Next_Primary_X_Coord;
*Crnt_Y_End = Next_Primary_Y_Coord;

}

if ((!strcmp(Crnt_TPD_Section_Name,"ARC")) &&
    (!strcmp(Next_TPD_Section_Name,"ARC")))
{
Calc_Arc_Start_End_points
(Crnt_Arc_Radius,
Crnt_Start_Angle,
Crnt_End_Angle,
Next_Arc_Radius,
Next_Start_Angle,
Next_End_Angle,
Crnt_Primary_X_Coord,
Crnt_Primary_Y_Coord,
Next_Primary_X_Coord,
Next_Primary_Y_Coord,
&Crnt_Arc_Startpoint_X_Coord,
&Crnt_Arc_Startpoint_Y_Coord,
&Next_Arc_Startpoint_X_Coord,
&Next_Arc_Startpoint_Y_Coord,
&Crnt_Arc_Endpoint_X_Coord,
&Crnt_Arc_Endpoint_Y_Coord,
&Next_Arc_Endpoint_X_Coord,
&Next_Arc_Endpoint_Y_Coord);

*Crnt_X_Strt = *Previous_X_End;
*Crnt_Y_Strt = *Previous_Y_End;

if
```

```

(((Crnt_Arc_Startpoint_X_Coord <= Next_Arc_Startpoint_X_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Startpoint_X_Coord >= Next_Arc_Startpoint_X_Coord - Dist_Tlrc) &&
 (Crnt_Arc_Startpoint_Y_Coord <= Next_Arc_Startpoint_Y_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Startpoint_Y_Coord >= Next_Arc_Startpoint_Y_Coord - Dist_Tlrc))
    ||
((Crnt_Arc_Startpoint_X_Coord <= Next_Arc_Endpoint_X_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Startpoint_X_Coord >= Next_Arc_Endpoint_X_Coord - Dist_Tlrc) &&
 (Crnt_Arc_Startpoint_Y_Coord <= Next_Arc_Endpoint_Y_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Startpoint_Y_Coord >= Next_Arc_Endpoint_Y_Coord - Dist_Tlrc)))
{
    *Crnt_X_End = Crnt_Arc_Startpoint_X_Coord;
    *Crnt_Y_End = Crnt_Arc_Startpoint_Y_Coord;
}

else if
(((Crnt_Arc_Endpoint_X_Coord <= Next_Arc_Startpoint_X_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Endpoint_X_Coord >= Next_Arc_Startpoint_X_Coord - Dist_Tlrc) &&
 (Crnt_Arc_Endpoint_Y_Coord <= Next_Arc_Startpoint_Y_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Endpoint_Y_Coord >= Next_Arc_Startpoint_Y_Coord - Dist_Tlrc))
    ||
((Crnt_Arc_Endpoint_X_Coord <= Next_Arc_Endpoint_X_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Endpoint_X_Coord >= Next_Arc_Endpoint_X_Coord - Dist_Tlrc) &&
 (Crnt_Arc_Endpoint_Y_Coord <= Next_Arc_Endpoint_Y_Coord + Dist_Tlrc) &&
 (Crnt_Arc_Endpoint_Y_Coord >= Next_Arc_Endpoint_Y_Coord - Dist_Tlrc)))
{
    *Crnt_X_End = Crnt_Arc_Endpoint_X_Coord;
    *Crnt_Y_End = Crnt_Arc_Endpoint_Y_Coord;
}

}

if ((!strcmp(Crnt_TPD_Section_Name,"LINE")) &&
 (!strcmp(Next_TPD_Section_Name,"LINE")))
{
    *Crnt_X_Strt = *Previous_X_End;

```

```
*Crnt_Y_Strt = *Previous_Y_End;
```

```
if(((Crnt_Primary_X_Coord <= Next_Primary_X_Coord + Dist_Tlrc) &&
    (Crnt_Primary_X_Coord >= Next_Primary_X_Coord - Dist_Tlrc) &&
    (Crnt_Primary_Y_Coord <= Next_Primary_Y_Coord + Dist_Tlrc) &&
    (Crnt_Primary_Y_Coord >= Next_Primary_Y_Coord - Dist_Tlrc)))
```

```
||
```

```
((Crnt_Primary_X_Coord <= Next_Secondary_X_Coord + Dist_Tlrc) &&
 (Crnt_Primary_X_Coord >= Next_Secondary_X_Coord - Dist_Tlrc) &&
 (Crnt_Primary_Y_Coord <= Next_Secondary_Y_Coord + Dist_Tlrc) &&
 (Crnt_Primary_Y_Coord >= Next_Secondary_Y_Coord - Dist_Tlrc)))
```

```
{
```

```
 *Crnt_X_End = Crnt_Primary_X_Coord;
```

```
 *Crnt_Y_End = Crnt_Primary_Y_Coord;
```

```
}
```

```
else if
```

```
((Crnt_Secondary_X_Coord <= Next_Primary_X_Coord + Dist_Tlrc) &&
 (Crnt_Secondary_X_Coord >= Next_Primary_X_Coord - Dist_Tlrc) &&
 (Crnt_Secondary_Y_Coord <= Next_Primary_Y_Coord + Dist_Tlrc) &&
 (Crnt_Secondary_Y_Coord >= Next_Primary_Y_Coord - Dist_Tlrc))
```

```
||
```

```
((Crnt_Secondary_X_Coord <= Next_Secondary_X_Coord + Dist_Tlrc) &&
 (Crnt_Secondary_X_Coord >= Next_Secondary_X_Coord - Dist_Tlrc) &&
 (Crnt_Secondary_Y_Coord <= Next_Secondary_Y_Coord + Dist_Tlrc) &&
 (Crnt_Secondary_Y_Coord >= Next_Secondary_Y_Coord - Dist_Tlrc)))
```

```
{
```

```
 *Crnt_X_End = Crnt_Secondary_X_Coord;
```

```
 *Crnt_Y_End = Crnt_Secondary_Y_Coord;
```

```
}
```

```
}
```

```
}
```

```
/*BTM*****
```

APPENDIX J

TRACTOR

.TRC

CONFIGURATION FILE

TRACTOR.TRC CONFIGURATION FILE**ITEM DESCRIPTION****CONFIGURATION FILE ENTRY**

Front Wheel Steering	F
Swinging Drawbar	S
Swingpoint ahead of Rear Axle	D
Hitch located at Rear of Tractor	R
Wheelbase	2.800000
Hitch Length	1.000000
Hitch Swing distance	0.250000
Maximum Hitch Swing Angle	30.000000
Maximum Steering Angle	52.000000

APPENDIX K

IMPLEMENT / GPS

.IMP

CONFIGURATION FILE

IMPLEMENT / G.P.S. .IMP CONFIGURATION FILE

ITEM DESCRIPTION	CONFIGURATION FILE ENTRY
Hitch Length	3.500000
Working Width	4.000000
G.P.S. Receiver Height	1.500000
G.P.S. Foward / Rearward Distance	0.000000
G.P.S. Left / Right Distance	0.000000
G.P.S. Foward / Rearward Direction	C
G.P.S. Left / Right Direction	C

APPENDIX L

FIELD STATUS

.FSC

CONFIGURATION FILE

FIELD STATUS .FSC CONFIGURATION FILE

ITEM DESCRIPTION	CONFIGURATION FILE ENTRY
Basepoint Latitude (Degrees)	45.000000
Basepoint Latitude (Minutes)	12.000000
Basepoint Latitude (Seconds)	34.000000
Basepoint Longitude (Degrees)	145.000000
Basepoint Longitude (Minutes)	98.000000
Basepoint Longitude (Seconds)	76.000000
Farm Field Startpoint X Coordinate	37.000000
Farm Field Startpoint Y Coordinate	23.000000
Farm Field Endpoint X Coordinate	41.000000
Farm Field Endpoint Y Coordinate	19.000000
Farm Field Travel Start Entity Number	1
Farm Field Travel End Entity Number	24

APPENDIX M

FIELD TARGET TRAVEL PATH

.TPD

OUTPUT FILE

FIELD TARGET TRAVEL PATH .TPD OUTPUT FILE

999	20	0	20
TPD_TEST_PASS	23.0	LINE	33.0
0	30	8	30
SECTION	0.0	0	0.0
2	11	62	40
ENTITIES	37.0	7	4.0
0	21	5	50
POINT	72.0	4	180.0
8	31	10	51
0	0.0	45.0	0.0
62	0	20	0
7	ARC	72.0	LINE
5	8	30	8
1	0	0.0	0
10	62	11	62
37.0	7	45.0	7
20	5	21	5
23.0	3	33.0	6
30	10	31	10
0.0	41.0	0.0	53.0
0	20	0	20
LINE	72.0	ARC	33.0
8	30	8	30
0	0.0	0	0.0
62	40	62	11
7	4.0	7	53.0
5	50	5	21
2	0.0	5	72.0
10	51	10	31
37.0	180.0	49.0	0.0

0	11	10	8
ARC	61.0	67.0	0
8	21	20	62
0	33.0	25.0	7
62	31	30	5
7	0.0	0.0	c
5	0	40	10
7	LINE	6.0	67.0
10	8	50	20
57.0	0	180.0	31.0
20	62	51	30
72.0	7	0.0	0.0
30	5	0	11
0.0	9	ARC	35.0
40	10	8	21
4.0	61.0	0	31.0
50	20	62	31
0.0	33.0	7	0.0
51	30	5	0
180.0	0.0	b	ARC
0	11	10	8
LINE	61.0	67.0	0
8	21	20	62
0	25.0	25.0	7
62	31	30	5
7	0.0	0.0	d
5	0	40	10
8	ARC	6.0	35.0
10	8	50	20
61.0	0	0.0	35.0
20	62	51	30
72.0	7	90.0	0.0
30	5	0	40
0.0	a	LINE	4.0

50	20	62	51
180.0	35.0	7	90.0
51	30	5	0
270.0	0.0	11	ARC
0	40	10	8
ARC	8.0	69.0	0
8	50	20	62
0	180.0	27.0	7
62	51	30	5
7	270.0	0.0	13
5	0	11	10
e	LINE	73.0	73.0
10	8	21	20
27.0	0	27.0	27.0
20	62	31	30
35.0	7	0.0	0.0
30	5	0	40
0.0	10	ARC	8.0
40	10	8	50
4.0	31.0	0	180.0
50	20	62	51
0.0	27.0	7	270.0
51	30	5	0
180.0	0.0	12	LINE
0	11	10	8
ARC	69.0	73.0	0
8	21	20	62
0	27.0	23.0	7
62	31	30	5
7	0.0	0.0	14
5	0	40	10
f	LINE	4.0	65.0
10	8	50	20
31.0	0	270.0	27.0

30	5	0	11
0.0	16	LINE	31.0
11	10	8	21
65.0	73.0	0	78.0
21	20	62	31
72.0	78.0	7	0.0
31	30	5	0
0.0	0.0	18	ARC
0	40	10	8
LINE	8.0	73.0	0
8	50	20	62
0	90.0	78.0	7
62	51	30	5
7	180.0	0.0	1a
5	0	11	10
15	ARC	35.0	31.0
10	8	21	20
65.0	0	78.0	70.0
20	62	31	30
72.0	7	0.0	0.0
30	5	0	40
0.0	17	LINE	8.0
11	10	8	50
65.0	73.0	0	90.0
21	20	62	51
78.0	82.0	7	180.0
31	30	5	0
0.0	0.0	19	ARC
0	40	10	8
ARC	4.0	35.0	0
8	50	20	62
0	270.0	78.0	7
62	51	30	5
7	90.0	0.0	1b

10	8	50	20
27.0	0	270.0	74.0
20	62	51	30
70.0	7	90.0	0.0
30	5	0	11
0.0	1d	ARC	57.0
40	10	8	21
4.0	35.0	0	33.0
50	20	62	31
180.0	74.0	7	0.0
51	30	5	0
0.0	0.0	1f	ARC
0	11	10	8
ARC	67.0	67.0	0
8	21	20	62
0	74.0	74.0	7
62	31	30	5
7	0.0	0.0	21
5	0	40	10
1c	ARC	10.0	53.0
10	8	50	20
35.0	0	90.0	33.0
20	62	51	30
70.0	7	180.0	0.0
30	5	0	40
0.0	1e	LINE	4.0
40	10	8	50
4.0	67.0	0	180.0
50	20	62	51
90.0	79.0	7	0.0
51	30	5	0
180.0	0.0	20	LINE
0	40	10	8
LINE	5.0	57.0	0

62	40
7	4.0
5	50
22	0.0
10	51
49.0	180.0
20	0
33.0	LINE
30	8
0.0	0
11	62
49.0	7
21	5
72.0	24
31	10
0.0	41.0
0	20
ARC	72.0
8	30
0	0.0
62	11
7	41.0
5	21
23	19.0
10	31
45.0	0.0
20	0
72.0	ENDSEC
30	0
0.0	EOF

MONTANA STATE UNIVERSITY LIBRARIES



3 1762 10269567 1

