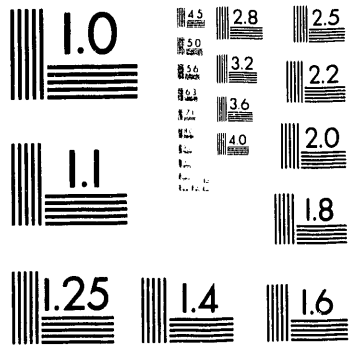

1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

1 of 3

SAND93-0827
Unlimited Release
Printed September 1993

Distribution
Category UC-~~515~~ 706

Global Positioning System Receiver Evaluation Results

Raymond H. Byrne
Advanced Vehicle Development Department
Sandia National Laboratories
Albuquerque, NM 87185, U.S.A.

ABSTRACT

A Sandia project currently uses an outdated Magnavox 6400 Global Positioning System (GPS) receiver as the core of its navigation system. The goal of this study was to analyze the performance of the current GPS receiver compared to newer, less expensive models and to make recommendations on how to improve the performance of the overall navigation system. This paper discusses the test methodology used to experimentally analyze the performance of different GPS receivers, the test results, and recommendations on how an upgrade should proceed. Appendices contain detailed information regarding the raw data, test hardware, and test software.

ACKNOWLEDGMENTS

The author wishes to recognize the technical reviews of this report by Billy Caskey (9616), Kevin Malone (9615), and J. Bryan Pletta (9616) of Sandia National Laboratories. The author also wishes to recognize the efforts of Molly Minahan (editing), Leona C. Tartaglia and Sandra Winter (word processing), and Hawaii Olmstead (illustration), Tech Reps. Inc., during the production of this report.

PREFACE

The intent of this study was to compare the performance of a particular two-channel, sequencing global positioning system (GPS) receiver to that of the newer five- and six-channel parallel receivers. The parallel channel receivers used in this study were selected based upon availability, cost, size, and receiver specifications. Unfortunately, due to budget and time constraints, the entire spectrum of parallel channel receivers could not be included in this study. Also, a variety of comparable GPS receivers have been introduced since this study was begun. The appropriate choice of a receiver is dictated, of course, by the application.

CONTENTS

Figures

Tables

ACRONYMS

ASCII	American Standard Code for Information Interchange
DOP	dilution of precision
FM	frequency modulation
GDOP	geometric dilution of precision
GPS	Global Positioning System
HDOP	horizontal dilution of precision
IC	integrated circuit
IVHS	Intelligent Vehicle Highway Systems
LANL	Los Alamos National Laboratory
LCD	liquid crystal display
MATLAB	Math Library
OEM	original equipment manufacturer
PC	personal computer
PDOP	position dilution of precision
RAM	random access memory
RF	radio frequency
SA	selective availability
SNR	signal-to-noise ratio
STD	standard
SV	satellite vehicle
TDOP	time dilution of precision
3D	three dimensional
2D	two dimensional
TTL	transistor-transistor-logic
VDOP	vertical dilution of precision

1.0 INTRODUCTION

A Sandia project currently uses an outdated Magnavox 6400 Global Positioning System (GPS) receiver as the core of its navigation system. The goal of this paper is to analyze the performance of the current GPS receiver compared to newer, less expensive models and to make recommendations on how to improve the performance of the overall navigation system.

1.1 GPS Basics

This section describes the basic operation of the GPS system for those not familiar with the technology. Those already familiar with GPS might want to skip this section and move on to Section 1.2.

The GPS system was developed by the United States Air Force to allow accurate position location anywhere in the world. GPS uses the principle of triangulation to estimate position. Triangulation involves measuring the distance to three known landmarks to determine position. For the GPS system, these landmarks are satellites orbiting the earth at a height of about 10,900 nautical miles; the satellites, sometimes referred to as Satellite Vehicles (SVs), are depicted in Figure 1-1. The full constellation of satellites will contain 21 active satellites and 3 spares for a total of 24. As of November 1, 1992, there are 19 satellites functioning in orbit. The full constellation should be operational by early 1993.

The range from each satellite is determined by measuring the time of flight of a radio signal transmitted by each satellite and received by the receiver. Distance is calculated using the simple equation shown below:

$$\text{distance} = \text{time} * \text{speed} \qquad (1\text{-}1)$$

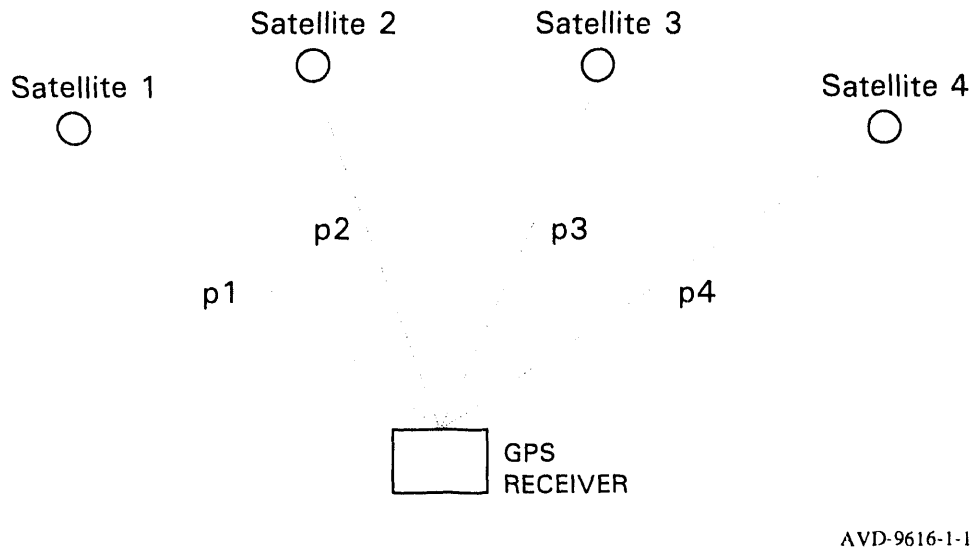


Figure 1-1. GPS System.

A radio signal travels at the speed of light, which is 186,000 miles/second. Because radio signals travel very quickly, very accurate timing is required to measure distance accurately. In one nanosecond (10^{-9} second), a radio signal travels 0.982 feet. Therefore, GPS receivers have very accurate clocks capable of nanosecond resolution. The distance to a satellite is referred to as a pseudo-range because it is an estimate of the range.

In order to measure the time it takes a message to travel from a satellite to the receiver, the start time of the message must be known. This is accomplished by synchronizing the clock on the receiver with the clock on the satellite. All the satellites use expensive atomic clocks to ensure that all of the satellite clocks are synchronized. Atomic clocks use the vibration period of a particular atom (cesium is a popular one) to keep time. All satellites have several atomic clocks to ensure that one is always functioning.

Each satellite broadcasts a pseudo-random code, or sequence of numbers, that repeat periodically. Every satellite has a unique but known pseudo-random code. By generating the same pseudo-random code in the receiver, the time of flight of the radio signal can be estimated by looking at the difference in start times between the received code and the generated code. Because of the nature of the pseudo-random codes, the start of the code doesn't have to be compared, a unique unambiguous section of the code anywhere in the message will suffice. In more technical terms, the time of flight is determined by sliding the generated code over the incoming signal until a peak in the autocorrelation is achieved.

Two different codes are broadcast by each satellite on two different frequencies. The "C/A" code is broadcast on the first frequency, L1, at approximately 1.575 GHz. This is the code intended for civil applications and is available to all users. The military code, or "P" code, is transmitted on the L1 frequency as well as the L2 frequency (1.227 GHz). Once the full GPS constellation is operational, the Air Force intends to encrypt the "P" code into "Y" code and its use will be limited to authorized military users.

Once the distance to three satellites is measured, theoretically one can obtain a position fix. The orbits of the satellites are known very precisely, and the Air Force constantly measures the actual orbit so that the satellites can determine their actual position very accurately. This orbital information, known as the ephemeris, is broadcast to the GPS receiver so that the exact location of the satellite vehicles is known at all times.

If the clock on the GPS receiver was synchronized to the atomic clocks on the satellites, then the distance to three satellites would be sufficient for determining the location of the receiver. However, the clocks on GPS receivers have to be small and inexpensive so they are not as accurate as the satellite clocks. This introduces a time offset error to the measurements. The GPS receiver clock is either ahead or behind the time kept by the atomic clocks by an unknown amount. By measuring the distance to a fourth satellite, this timing error can be eliminated. Measuring the range to four satellites with a clock offset ΔT yields four equations with four unknowns. It is then easy to solve for the local clock offset as well as the x,y,z position of the GPS receiver. If a two-

dimensional location is required (altitude is already known, for example, sea level), then only three satellites are needed.

Many factors exist that influence the accuracy of the GPS receiver's position estimate. The radio signal is refracted as it travels through the earth's atmosphere and therefore does not travel in a straight line. Small variations in the satellite's orbits also affect the accuracy of the GPS fix. Most significantly, the Air Force intentionally sends incorrect clock and ephemeris information on the C/A code to degrade the position accuracy to about 100 meters. This degradation is known as Selective Availability (SA). This intentional degradation is intended to keep unfriendly countries from developing precision guided munitions with GPS. A list of several factors which degrade GPS accuracy appear in Table 1-1 (Brown and Hwang, 1992). The errors in Table 1-1 describe pseudo-range errors that combine with satellite geometry to determine the magnitude of the position error. The particular satellite geometry contributes an error multiplier to the pseudo-range error that will be discussed later.

As seen in Table 1-1, the largest error source is selective availability (SA). The pseudo-range errors caused by SA degrade the position accuracy of GPS to about 100 meters. The effects of SA are illustrated in Figures 1-2 and 1-3. SA was turned off in October for a brief period while the Air Force was conducting tests. The data in Figure 1-2 was taken during this period and shows GPS error as a function of time with SA off. Figure 1-3 shows GPS error as a function of time with SA on. The slowly varying error in Figure 1-3 is caused by SA. Both plots represent data taken as part of this project.

Table 1-1. Typical Pseudo-Range Errors (Brown and Hwang, 1992)

Error Component	Pseudo-range Error Standard Deviation (meters)
Satellite Position	3
Ionosphere refraction with correction	5
Troposphere refraction with correction	2
Multipath (reflections of RF signal)	5
Selective Availability (SA)	30

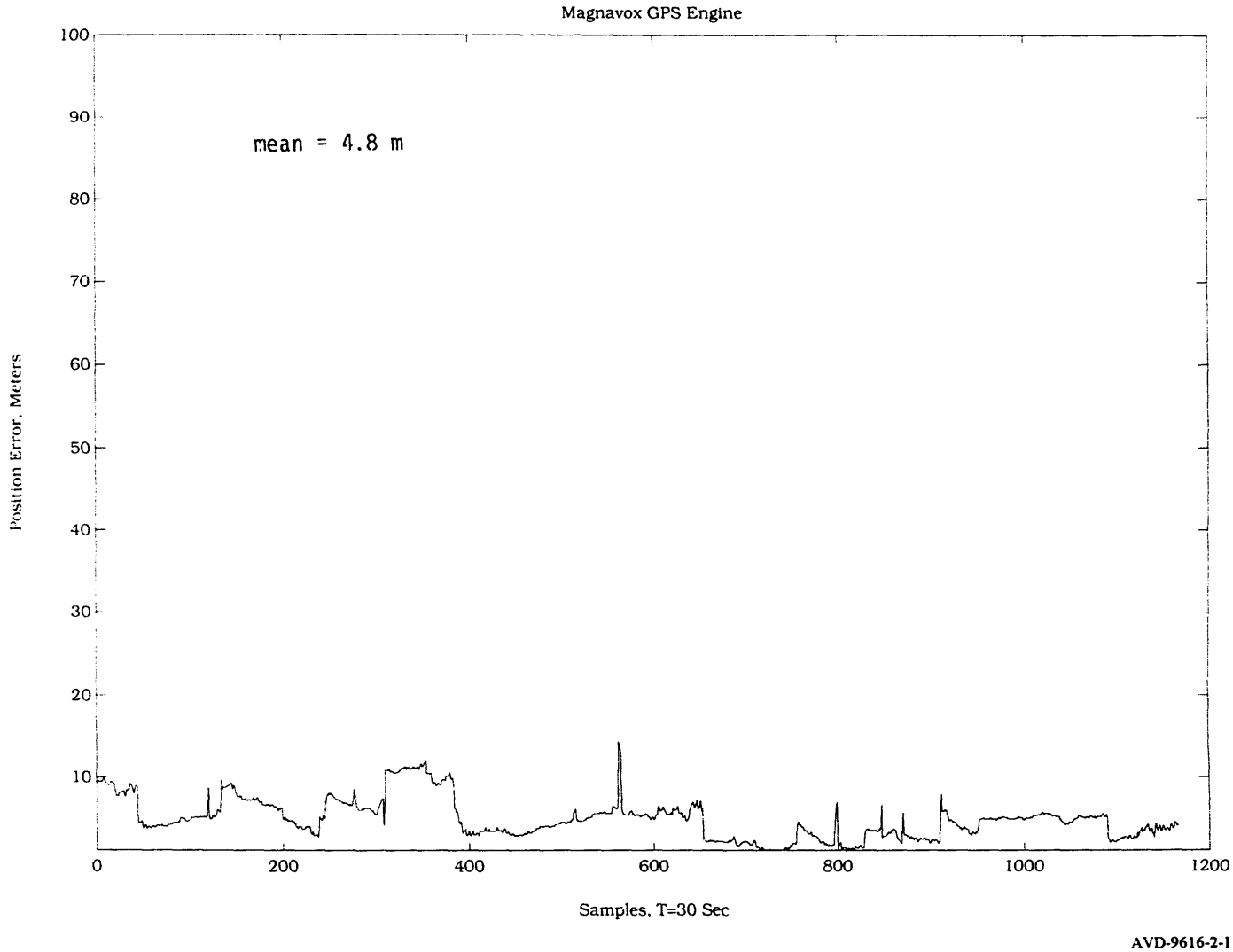


Figure 1-2. Typical GPS Position Error with SA Off, Magnavox GPS Engine.

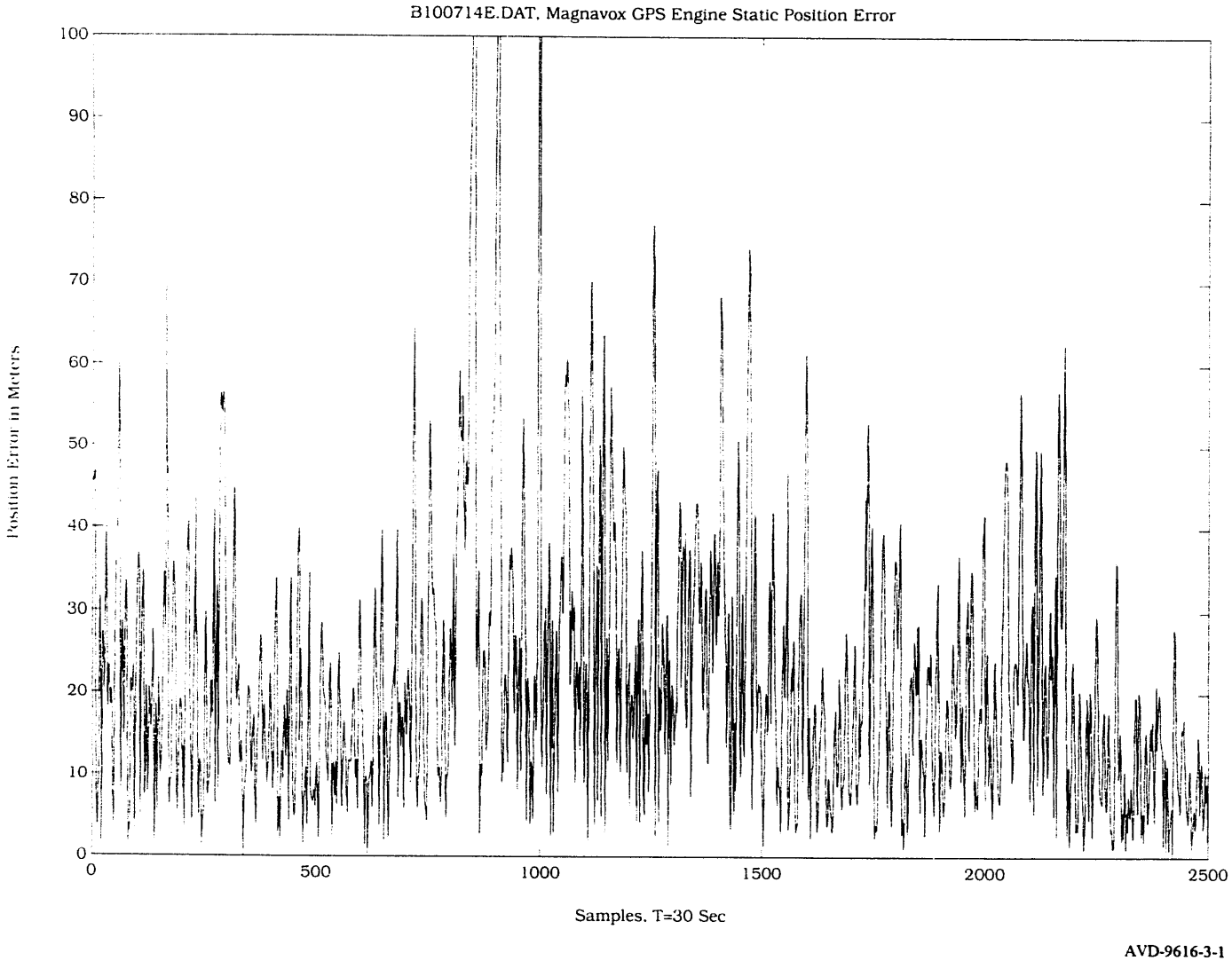


Figure 1-3. Typical GPS Position Static Error with SA On, Magnavox GPS Engine (B100714E.DAT).

The particular geometry of the satellites relative to the receiver also affects the position accuracy. Some geometries are more favorable than others. Terms used to describe the strength of the position fix based on the geometry include: Dilution of Precision (DOP), Horizontal Dilution of Precision (HDOP), Geometric Dilution of Precision (GDOP), Position Dilution of Precision (PDOP), Time Dilution of Precision (TDOP), and Vertical Dilution of Precision (VDOP). The various DOPs are error multipliers that indicate the accuracy of a particular type of position fix based on a certain pseudo-range error. For instance, if the pseudo-range measurements are accurate to 10 meters and the HDOP is equal to 3.5, the horizontal position accuracy would be 35 meters (10 * 3.5). A PDOP of 2 or 3 is fairly good, while a PDOP of 10 is not so good. Certain geometries can cause the DOP to become very large (infinite). Two useful DOP identities are shown in Equations (1-2) and (1-3).

$$PDOP^2 = VDOP^2 + HDOP^2 \tag{1-2}$$

$$GDOP^2 = PDOP^2 + TDOP^2 \tag{1-3}$$

One way to improve GPS accuracy is by taking differential measurements, often referred to as differential GPS (DGPS). This is shown in Figure 1-4.

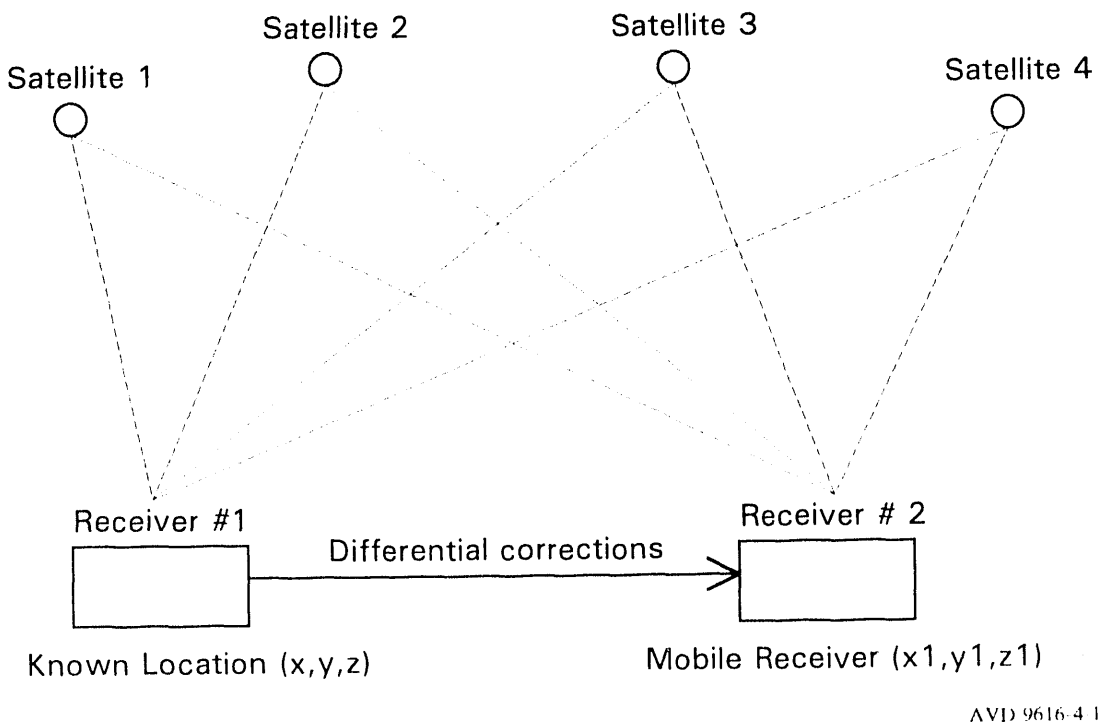


Figure 1-4. Differential GPS.

By placing one GPS receiver in the differential system at a surveyed, known location, the errors from SA, ephemeris, and atmospheric effects can be practically eliminated from the mobile receiver. If the two receivers are not very far apart, they will be using the same satellites as well as the same region of the atmosphere and will be subjected to essentially the same error sources. In practice, it is desirable to keep the receivers within 10 kilometers of each other. The sum of all the error terms can be calculated for the receiver at the known location and then corrections can be sent to other receivers in the area. Differential GPS can easily reduce the position error to less than 10 meters. Other, more complicated methods of obtaining increased accuracy include using differential GPS with carrier phase measurements.

The Coast Guard plans to implement a differential GPS system around the coastal waters of the United States, including the Great Lakes, Hawaii, and Puerto Rico. This system will aide in maritime navigation and will offer a great improvement in positional accuracy over Loran C. Magnavox and CUE Network Corporation also plan to market a nationwide subscriber network that allows users to receive differential corrections over an FM radio link (GPS Report, 1992). Surveyor's currently use differential GPS to obtain positional accuracies of centimeters, but these measurement require post processing of the data and assume a stationary GPS receiver.

GPS receivers are often compared based on the number of channels they have for tracking satellites. As mentioned in the previous section, a GPS receiver must track at least four satellites to perform three-dimensional navigation. Early receivers had only one or two channels for tracking satellites, so they had to sequence from one satellite to another to obtain a position fix. These types of receivers are known as sequencing receivers because they have to switch quickly from one satellite to another to obtain the four necessary pseudo-ranges. More modern receivers have dedicated channels for each satellite being tracked. The number of channels can range from 4 to 12 or more, depending on the cost and accuracy of the receiver. Receivers with 5 or more channels have the advantage of being able to track more than the four necessary satellites simultaneously. This allows incorporating more pseudo-ranges in the position fix for increased accuracy and also lets the receiver continue to navigate if one of the satellites is temporarily obstructed.

In summary, this section has presented an introduction to the basic operation of GPS. GPS uses the concept of triangulation to determine the location of a receiver. The known landmarks from which distances are measured are the GPS satellites orbiting the Earth. The time-of-flight of a pseudo-random code sent synchronously from each satellite is measured at the receiver to determine the distance to each satellite. Ideally, only three satellites would be needed to determine the location of the receiver. However, because of receiver clock errors, a fourth satellite is needed to synchronize the receiver clock with the atomic clocks on the satellites. The largest source of GPS position error is Selective Availability (SA). Most of the position error can be eliminated by using a differential GPS system.

1.2 Project Goals

The current navigation system uses dead reckoning (compass and odometry), updated with position fixes from either GPS or TRANSIT as its position location system. The GPS receiver, a Magnavox 6400 unit, is a 2-channel sequencing receiver that is based on technology more than 10 years old. Newer GPS receivers are less expensive, typically have five or six channels, and offer improved performance. In addition, maintenance and repair of the Magnavox receiver is becoming more expensive as the receiver becomes more obsolete. Therefore, upgrading the current navigation system is a high priority. One goal of this study was to compare the current GPS system with several inexpensive, commercially available systems to quantify the performance improvements gained by switching to a newer receiver. The receivers tested are listed in Table 1-2. The "original equipment manufacturer (OEM) type" receivers are single board GPS receivers that are meant to be integrated into a system or product. The Trimble and Magnavox 6400 receivers are "integrated" commercial products.

Table 1-2. GPS Receivers Tested

Receiver	Description
Magnavox 6400	2-Channel, sequencing receiver, receiver in current system, Integrated System
Magellan OEM GPS Module	5-Channel GPS Receiver, OEM type
Magnavox GPS Engine	6- Channel GPS Receiver, OEM type
Rockwell NavCore V	5- Channel GPS Receiver, OEM type
Trimble Placer	6-Channel Receiver, Integrated System

The current navigation system also uses a fluxgate magnetometer (electronic compass) to determine vehicle heading as part of the dead-reckoning system. A fluxgate magnetometer senses the Earth's magnetic field to determine the direction of magnetic North (the direction a compass would point). The large mass of steel present in the current application interferes with the performance of the magnetometer because the steel (a ferrous material) warps the Earth's magnetic field around the sensor. In general, magnetometers are more accurate when they are not near any induced magnetic fields caused by electric currents or ferrous materials. Therefore, the dead-reckoning performance is degraded because the fluxgate magnetometer heading is not always accurate. Another goal of this study was to suggest methods of improving the dead-reckoning system's performance.

The performance of the current GPS receiver was tested along with four commercially available receivers. Testing included static testing as well as dynamic testing. The results of the testing are outlined in this report.

Recommendations on how to improve the performance of the dead-reckoning system are based on experience with fluxgate magnetometers at Sandia's Robotic Vehicle Range (RVR). This report documents the GPS receiver testing, test results, and recommendations on how to improve the current navigation system.

1.3 Paper Outline

As mentioned in the previous section, the goal of this study was to analyze the performance of several commercial receivers compared to the current GPS system and to make recommendations on how to improve the overall navigation system. This report documents the test methodology used to gather GPS data, the test results, as well as recommendations on how to improve the current navigation system. The test methodology is described in Section 2.0. The parameters tested are discussed in Section 2.1; the test hardware is documented in Section 2.2; and the test software is described in Section 2.3. The test results are discussed in Section 3.0. The static results are presented in Section 3.1, while the dynamic results are described in Section 3.2. Section 3.3 provides a summary of the test results. The recommendations for improving the current navigation system are outlined in Section 4.0. Section 4.1 discusses the selection of an upgraded GPS receiver, while Section 4.2 addresses overall navigation issues. Section 5.0 contains a summary of the report.

Appendices are used to document areas with greater detail than would be suitable for the report. Appendix A contains a detailed description of the test hardware. Appendix B contains descriptions of the serial data communication protocols used by the GPS receivers tested. Appendix C contains a listing of the software used for the GPS receiver testing. This includes the data acquisition software as well as the data post processing software. Appendix D contains raw GPS data. This includes maps made while taking dynamic data and navigation mode as a function of time. Appendix E contains examples of the raw GPS data. Appendix F lists the individual contacts who provided technical support and pricing information for the GPS receivers tested.

2.0 TEST METHODOLOGY

The project goal of this study was to compare the performance of the current GPS system to commercially available GPS receivers, and then to make overall recommendations on how to improve the current navigation system. Many possible parameters exist that may be measured when comparing GPS receivers. These include position error, number of satellites tracked, signal-to-noise ratios, and time-to-first-fix. Section 2.1 discusses the parameters that were chosen to compare the performance of the current GPS receiver to several commercial off-the-shelf units. Section 2.2 describes the test fixture hardware developed to gather GPS data from the five different receivers and Section 2.3 describes the software used to gather the data. Post processing the gathered data is discussed in Section 2.4.

2.1 Parameters Tested

Many parameters exist that may be tested when comparing different GPS receivers. These parameters include receiver sensitivity, static accuracy, dynamic accuracy, number of satellites tracked, and time-to-first-fix. The tests performed to evaluate the five different GPS receivers consisted of both static and dynamic testing. This section discusses the parameters tested and the rationalization for choosing these parameters.

For this navigation application, time-to-first-fix is an important parameter. The current GPS receiver can take up to 30 minutes to initialize and lock onto the satellite signals before it starts navigating. However, all of the newer receivers advertise fast position fixes, usually under one minute, if the receiver knows its position to within several hundred miles. This is often referred to as a "warm start." The difference between a 30-second first fix and a 2-minute first fix is not that important for this application. However, 1-2 minutes is a great improvement over 30 minutes. Although this parameter was not explicitly measured, attention was paid to time-to-first-fix to confirm that the newer receivers were meeting the quoted specification.

The number of satellites tracked and receiver sensitivity are important parameters for this application. The more satellites tracked, the less likely an obstruction of one or more satellites will result in a loss of navigation. Also, a more sensitive receiver is less likely to be affected by foliage and other obstructions that reduce signal strengths. The receiver sensitivity is affected by the type of antenna used and the type of cabling. Some antennas have higher gains than others, different cables have different attenuation characteristics, and the longer the cable the greater the signal attenuation. The navigation mode, two-dimensional (2-D) or three-dimensional (3-D), is affected by the number of satellites visible. Provided that the geometry results in an acceptable DOP, a minimum of four satellites are necessary for 3-D navigation. Additional satellites may be used to achieve a more robust position fix. If four satellites are in view, but the DOP is higher than a certain threshold, many receivers will switch to 2-D navigation.

Ideally, measuring the signal-to-noise ratio (SNR) in the receiver and the number of satellites being tracked would yield the most insight into receiver performance. However, this information is usually buried in several different data packets for any given receiver. For some receivers, this information is not always available (the Trimble Placer does not output SNRs or the number of satellites tracked for example). Therefore, a compromise was made and packets were requested that contained the position fix as well as the navigation mode or number of satellites tracked. Usually this data was contained in the same data packet. This reduced the amount of data stored and simplified the data analysis. The information gathered from each receiver is listed in Table 2-1.

Table 2-1. Summary of Data Analyzed

Receiver	Data Gathered
Magellan	Latitude, Longitude Number of satellites used - implies navigation mode (none, 2-D, or 3-D)
Magnavox GPS Engine	Latitude, Longitude Navigation Mode (none, 2-D, or 3-D)
Rockwell NavCore V	Latitude, Longitude Navigation Mode (none, 2-D, or 3-D) Note: Number of satellites tracked also available from raw data
Magnavox 6400	Latitude, Longitude Number of satellites tracked
Trimble Placer	Latitude, Longitude Navigation Mode (none, 2-D, or 3-D)

Differences in navigation modes can be caused by several factors; these include differences in number of satellites being tracked, differences in the DOP value that cause a switch from 3-D to 2-D navigation, and differences in satellite mask angles and receiver/antenna sensitivity. The DOP settings and mask angles are known for each receiver, so the navigation mode data will allow comparing the number of satellites tracked and receiver/antenna sensitivity as one performance criterion. Although the navigation mode data lumps several factors together, it does give a comparison of overall receiver/antenna performance.

As mentioned in the previous section, the antenna and cable choice affects the performance of the GPS receiver. The antennas used for the GPS testing were supplied with the receiver or OEM evaluation kit. The cabling was also supplied with the exception of the Magnavox GPS Engine. Therefore, the performance of the

antenna and cabling was lumped together with the overall GPS system because each manufacturer recommends (or provides) antennas and cabling.

Other performance factors include the amount of filtering in a GPS receiver. Excessive filtering reduces the amount of variance in the position and velocity data, but also slows the response of the receiver. Excessive filtering will cause a receiver to output incorrect positions when starting, stopping, or turning sharply. This type of error is not very important for this application because the position data is sent over a radio frequency (RF) link to a command center. The delay and sampling introduced by the communication link will probably be much greater than the delay introduced by filtering in the receiver. Additional parameters that were not analyzed are velocity and heading accuracy. Accurate velocity information is already available from odometry, and heading information that would be required for dead reckoning is not needed while GPS is functional (GPS is more accurate than dead reckoning over long periods of time).

Another easy-to-measure performance criterion is static position accuracy. This parameter was measured by placing the GPS receivers at a surveyed location and taking data for approximately 24 hours. Although in this application, the receivers will be moving most of the time, the static accuracy does give a good idea of the receivers' position accuracy capabilities.

The parameters measured and the performance insights gained from these measurements are summarized in Table 2-2.

Table 2-2. Summary of Parameters Measured and Performance Areas Evaluated

Parameter Measured	Performance it Evaluates
Time-to-first-fix	How quickly a receiver starts navigating. Not explicitly measured, but qualitatively considered.
Static Position Accuracy	Static accuracy and insight into overall accuracy.
Static Navigation Mode - Number of Satellites Tracked	Taking into account DOP switching, gives insight into receiver/antenna sensitivity.
Dynamic Position Plots	Some accuracy information is obtained by comparing different data plots taken while driving down the same section of road. Most of this analysis is qualitative though because there is no ground-truth data for comparison.
Dynamic Navigation Mode	Taking DOP switching into account gives insight into the sensitivity of the receiver/antenna and the rate with which the receiver recovers from obstructions.

In summary, the GPS testing performed for this project consisted of storing position and navigation mode data from five different GPS receivers for both static and dynamic tests. The static testing provides information about the static position accuracy as well as the sensitivity of the receiver and antenna if DOP switching is taken into account. The dynamic testing mostly provides information about the receiver/antenna sensitivity and the receiver's ability to recover from temporary obstructions (taking into account DOP switching). The dynamic testing also provides some qualitative information about position accuracy by comparing plots of the data points from the various receivers.

2.2 Test Hardware

This section describes the test fixture developed to gather GPS data from the five different GPS receivers tested. A more detailed description of the test fixture hardware appears in Appendix A.

The GPS receivers tested use a serial interface for communicating position information. The Magnavox 6400 receiver communicates using RS-422 serial communications, while the other four receivers use the RS-232 communications standard. The RS-422 and RS-232 standards for data transmission are described and compared in more detail in Table 2-3.

Table 2-3. Comparison of RS-232 and RS-422 Serial Communications

RS-232 Communications	RS-422 Communications
Single-ended data transmission	Differential data transmissions
Relatively slow data rates (usually < 20K baud), short distances up to 50 feet, most widely used.	Very high data rates (up to 10M baud), long distances (up to 4000 feet at 100K baud), good noise immunity.

For the short distances involved in transmitting GPS data from the receiver to a computer, the type of serial communications is not important. In fact, even though RS-232 communications are inferior in some ways to RS-422, RS-232 is easier to work with because it is a more common standard (especially for PC-type computers).

The GPS receivers tested use serial communications to send information to another computer as well as to receive configuration commands. Typically, the GPS receivers output a position update at a fixed rate, once per second for example. With five GPS receivers "talking" at their own fixed rate, a data acquisition system was needed that could gather serial data from five serial ports simultaneously, without losing information. Serial ports may be monitored in two ways, either in polled or interrupt operation. In polled operation, software periodically "looks" at the serial port to determine if a new character, or byte, has been received. If the software is too slow in

checking the port, some data may be lost. However, under interrupt operation, an interrupt is generated whenever a new character is received, allowing the software to be doing something else when there are no new characters to process.

For this application, interrupt operation was necessary to be able to quickly gather serial data from five ports without losing information if all of the receivers happened to talk at the same time. A PC typically has two interrupt capable serial ports. Additional serial ports may be added via an expansion card, but these additional serial ports can usually only operate in a polled mode. A PC expansion card would not work for the data rates and number of ports that could potentially be talking simultaneously in this application. Therefore, a Standard (STD) Bus computer was selected as the data acquisition system. PC-compatible STD Bus computers are readily available, and serial cards are also available that support interrupt operation. PC compatibility simplifies software development and data transfer. Also, STD Bus computers are well suited for portable data acquisition because of their small form factor and low power consumption.

The STD Bus data acquisition hardware used to gather GPS data is outlined in Table 2-4. The overall system is best described as an embedded PC because all of the functional features of a PC are available in a reduced form factor. A 286-compatible processor allowed software development using Borland's Turbo C. An interrupt-driven serial card yielded a total of six interrupt-driven serial channels. A hard disk was used for data storage, while a 3.5-inch drive was used to transfer data between a PC and the data acquisition computer. A video/keyboard card was necessary to interface to a video monitor and standard IBM XT keyboard. A small, low power liquid crystal display (LCD) was used for the dynamic testing where low power consumption was critical.

A block diagram of the overall GPS test system is shown in Figure 2-1. Figure 2-1 depicts the system used for dynamic testing where power was supplied from a 12-volt battery. For the static testing, AC power was available with an extension cord. Therefore, the computer supply was connected directly to AC, while the +12 volts for the GPS receivers was generated using a Hewlett Packard supply for the static test.

The Magnavox GPS Engine requires interface circuitry to convert from transistor-transistor-logic (TTL) levels (0,5 volts) to RS-232 levels (-12, +12 volts). The Magnavox GPS Engine and the Magellan OEM module also required an external battery for the battery-backed random access memory (RAM). In addition, the Magnavox GPS Engine requires regulated power at +5 volts and + 7 volts. These voltages were generated using linear regulator integrated circuits (ICs) and the raw battery supply.

The GPS test fixture was set up in a Chevrolet van with an extended rear for additional room. The GPS antennas were mounted on aluminum plates that where attached to the van with magnets. The Rockwell antenna came with a magnetic mount so it was attached directly to the roof. The five antennas were within one meter of each other near the rear of the van and mounted at the same height so that no antenna obstructed the others.

Table 2-4. STD Bus Computer System

STD Bus Card or Peripheral	Description
ZT 8901 (Ziatech Corp.)	Low power CMOS, 286 compatible processor, 3 serial ports, 640 K RAM, 380 K battery-backed RAM disk, 48 parallel I/O lines
ZT 88CT41 (Ziatech Corp.)	Quad serial card, low power CMOS, supports interrupt operation, RS-232 or RS-422 operation
ZT-8980/ZT-1030 (Ziatech Corp.)	Video/Keyboard Support, Driver for LCD Display
ZT-8950 (Ziatech Corp.)	3.5-Inch Disk Drive
ZT-8952-40 (Ziatech Corp.)	40 MByte Hard Disk
IBM XT Keyboard	Keyboard

For the dynamic testing, power was supplied from a 60 Amp-Hour lead acid battery. The battery was used to power the AC-DC inverter as well as the five receivers. The van's electrical system was tried at first, but noise caused the computer to lock up occasionally. Using an isolated battery solved this problem. An AC-powered computer monitor was used for the static testing because AC power was available. For the dynamic testing, the low power LCD display was used.

A more detailed description of the test fixture hardware appears in Appendix A. This includes interface circuit schematics as well as serial cable pinouts.

2.3 Test Software

This section describes the software developed to gather the GPS data from the five different GPS receivers.

As mentioned in the previous section, the five GPS receivers tested use serial communications to communicate with another computer. Serial communications require a defined protocol, so that both computers know the allowed commands and data formats. The serial communications protocols for the different GPS receivers are listed in Appendix B. The description in Appendix B is limited to the packets used.

The baud rates and serial port configurations of the different GPS receivers are listed in Table 2-5.

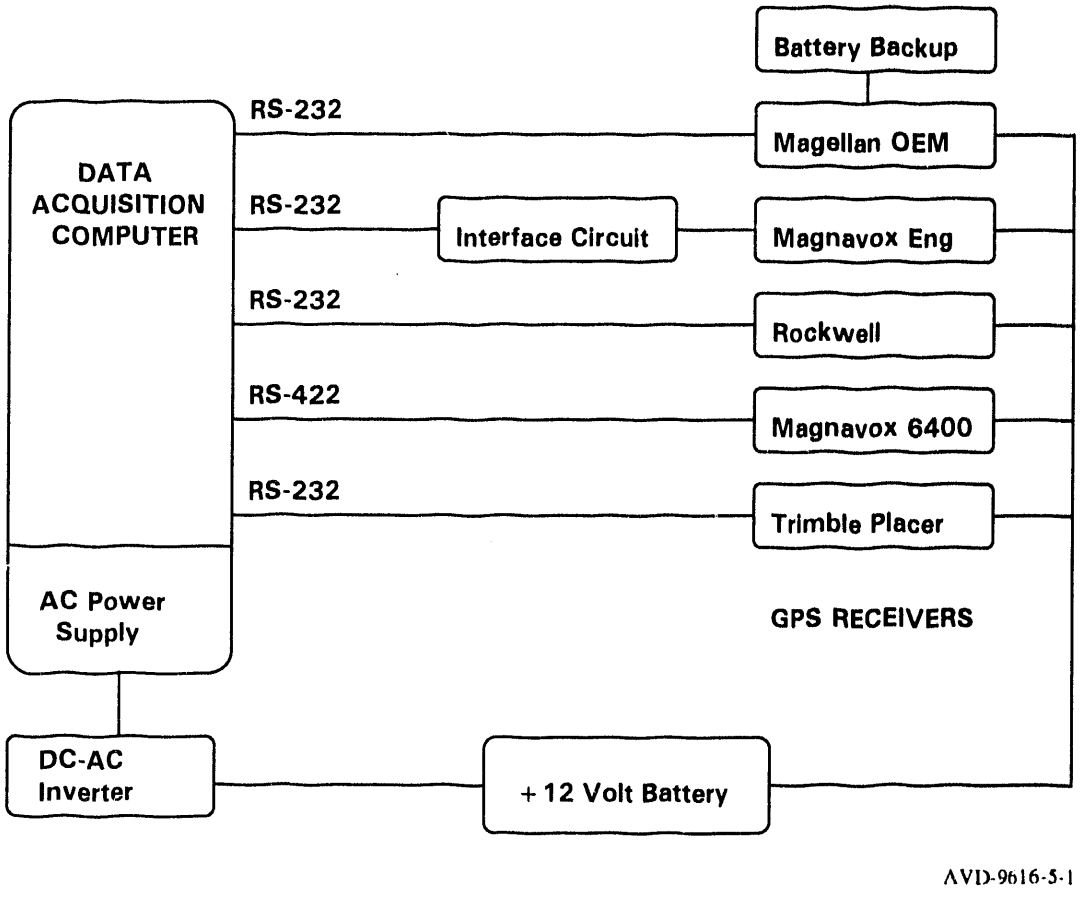


Figure 2-1. Block Diagram of GPS Test Fixture (Dynamic Testing).

Table 2-5. GPS Receiver Serial Port Configurations

Receiver	Serial Port Configuration
Magellan	9600 baud, 8 bit, 1 stop, no parity
Magnavox GPS Engine	4800 baud, 8 bit, 1 stop, no parity
Rockwell NavCore V	9600 baud, 8 bit, 1 stop, odd parity
Magnavox 6400	2400 baud, 7 bit, 1 stop, odd parity
Trimble Placer	9600 baud, 8 bit, 1 stop, odd parity

The data acquisition software is outlined in Figure 2-2. The data acquisition program first reads the clock to determine the time and date. Then, it creates and opens a data file for each of the GPS receivers. The filename created for each receiver is unique and contains information about the GPS receiver as well as the date and time of the test. The filename format is shown in Figure 2-3.

The letter at the beginning of the data filename corresponds to one of the five GPS receivers. This relationship is outlined in Table 2-6.

Table 2-6. Data File Letter Assignments

Receiver	File Letter
Magellan OEM Module	A
Magnavox GPS Engine	B
Rockwell NavCore V	C
Magnavox 6400	D
Trimble Placer	F

After creating and opening the data files for storing the serial GPS data, the software then initializes the interrupt controllers on the processor card and the serial card. Then the serial ports are initialized and configured for the correct baud rate. Once the serial ports are configured, the software initializes the Magellan, Magnavox GPS Engine, and Trimble Placer to output data every 30 seconds. This reduces the amount of data stored and allows taking data over a longer period of time. The Magnavox 6400 outputs data at once per second, and this is

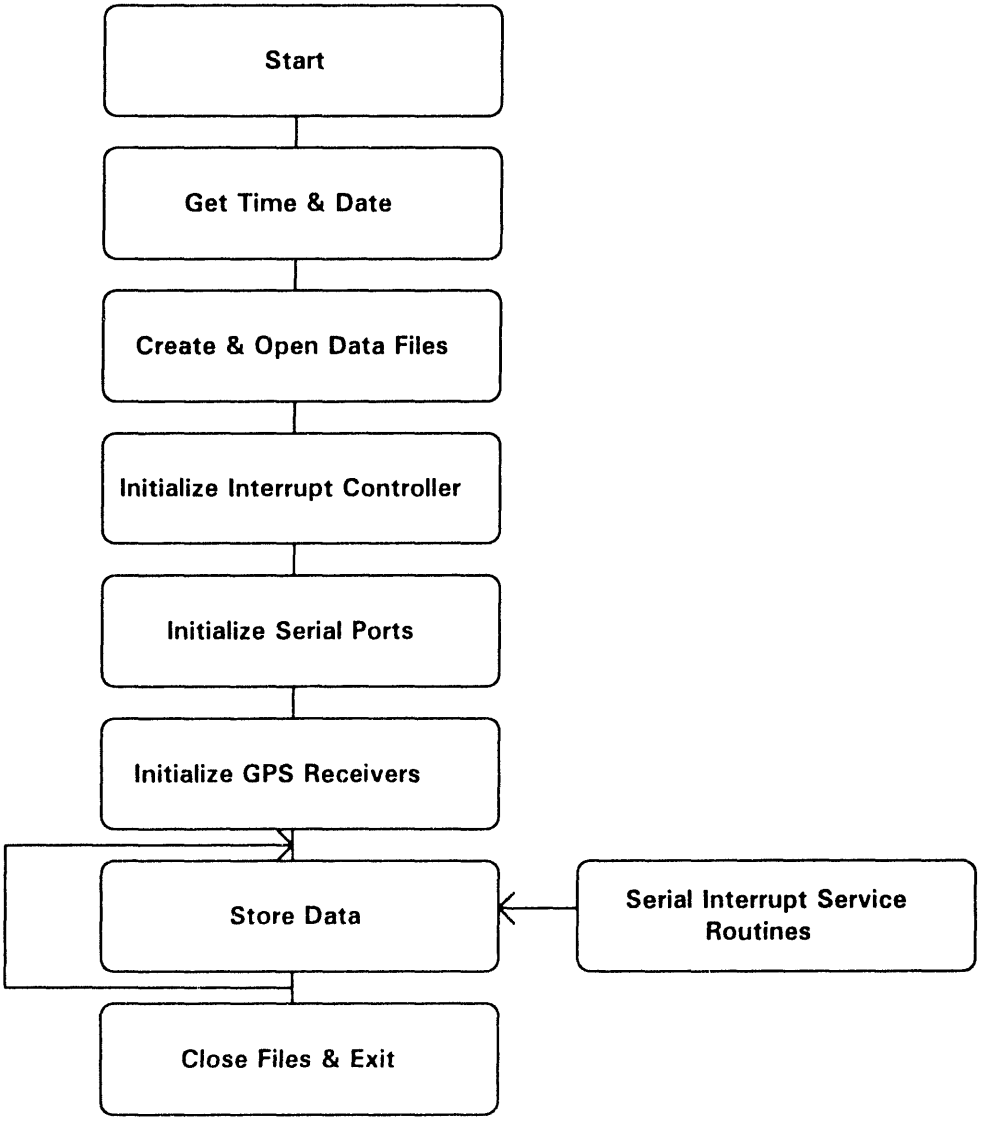


Figure 2-2. Software Flow Diagram.

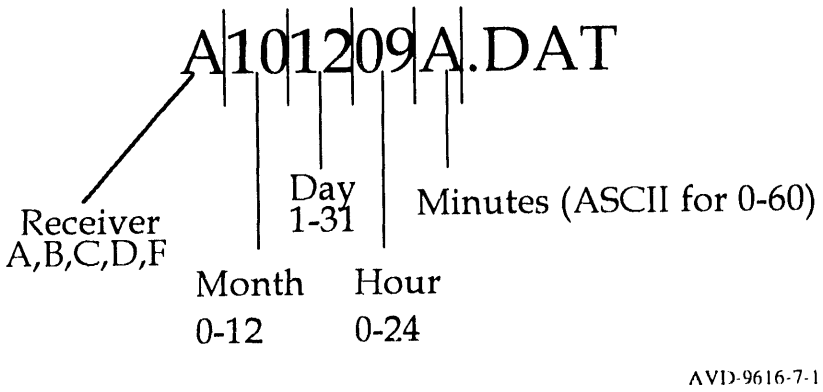


Figure 2-3. Data Filename Format.

not easily changed. Therefore, all of the 6400 data was stored. The Rockwell receiver also outputs data once per second, and is not configurable for other data rates. Therefore, the software only logged every 30th message. This yields an effective data rate of once every 31 seconds (a slight bug in the software).

Once everything is initialized, the data acquisition software uses interrupt service routines to place incoming serial data into buffers. The main loop of the program empties these buffers into the appropriate data files as time permits. Because all of the data acquisition is done in fast interrupt service routines, no serial data is lost even if all of the receivers "talk" at the same time. The program is exited by hitting the <enter> key on the keyboard. This closes all of the data files, turns off the interrupt service routines, and exits the program. Some error checking is used in the program to display communications errors, like overflow and parity errors, to the screen should they occur. These errors were never seen during normal operation.

Some software routines were provided by Ziatech Corporation for the serial card. These routines initialize the serial ports and interrupt controller, and also contain the interrupt service routines for putting the serial data into a buffer. These routines greatly simplified the software development. A description of these routines appears in Appendix C.4.

2.4 Data Post Processing

As mentioned in the previous section, the GPS data was stored in raw form and post processed to extract position and navigation data. This was done so that the raw data could be analyzed again if there were any questions with the results. Also, storing the data as it came in from the serial ports required less computational effort and reduced the chance of overloading the data acquisition computer. This section describes the software used to post process the data.

The GPS serial data from each receiver was stored in an ASCII file on the hard disk of the data acquisition computer. After taking data, the ASCII files were transferred to a PC using the BACKUP and RESTORE DOS commands and 3.5-inch floppy disks. A 50-MHz 486 computer was used to post process and analyze the GPS data. The raw data files were converted into new files with latitude, longitude, and navigation mode columns. The file conversion software is listed in Appendix C.1. A typical post processed file is shown in Figure 2-4.

The latitude and longitude are represented in decimal degrees. For the navigation mode, 4 corresponds to 3-D navigation, 3 corresponds to 2-D navigation, and -1 corresponds to not navigating. This columnar format was chosen because the data can be easily loaded into MATLAB for processing. MATLAB is a software package for analyzing and processing matrices and vector data. Therefore, it is well suited for analyzing vectors of position data.

The number of significant figures of valid data is a function of the data format of the different GPS receivers. The data formats for latitude and longitude are outlined in Table 2-7.

Looking at Table 2-7, the minimum resolution of the different GPS receivers varies greatly. However, the resolution of all of the receivers is still orders of magnitude less than the typical position error of up to 100 meters. Therefore, this parameter will not be an issue in the data analysis.

```
35.4081523  106.5217254   3
35.4081523  106.5217254   3
35.4081524  106.5217254   3
35.4081523  106.5217254   3
35.4081623  106.5217254   4
35.4081524  106.5217254   3
35.4081523  106.5217254   -1
35.4081624  106.5217254   3
35.4081624  106.5217254   3
35.4081623  106.5217254   4
```

AVD 9616.8.1

Figure 2-4 Typical post-processed data file showing latitude, longitude, and Nav mode.

Table 2-7. Accuracy of Receiver Data Formats

Receiver	Data Format Resolution	Minimum Resolution (meters)
Magellan	10^{-7} degrees	0.011
Magnavox GPS Engine	1.7×10^{-6} degrees	0.19
Rockwell NavCore V	10^{-11} radians (5.73×10^{-10} degrees)	6.36×10^{-5}
Magnavox 6400	10^{-8} radians (5.73×10^{-7} degrees)	6.36×10^{-2}
Trimble Placer	10^{-5} degrees	1.11

Once the raw data was converted to files with latitude, longitude, and navigation mode in columnar form, the data was loaded into MATLAB and analyzed. Simplified examples of the different manipulations performed on the data are summarized in Table 2-8. Other MATLAB features including axis labeling, axis sizing, and histogram bin definition were used to make the appropriate plots for this paper.

Degrees of latitude and longitude were converted to meters using the conversion factors listed below. The conversion factors are from the survey performed by Santiago Romero, Jr., a registered professional surveyor in the state of New Mexico (see Section 3.1 of this report).

Latitude Conversion Factor 11.0988×10^4 meter/degree latitude

Longitude Conversion Factor 9.126×10^4 meter/degree longitude

In summary, the data post processing consisted of massaging the raw data to obtain latitude, longitude, and the navigation mode into a form suitable for entry into MATLAB. Then, MATLAB was used to analyze the data. Data manipulations included obtaining the position error from a surveyed location, generating histograms of position error and navigation mode, and plotting dynamic position data. The mean and variance of the position errors were also obtained.

Table 2-8. MATLAB Data Manipulations

Data Manipulation	MATLAB Commands

Load data file	load C:\SF\A102209A.OUT
	lat=A102209A(:,1);
	long=A102209A(:,2);
	sats=A102209A(:,3);

Determine position error as a function of time	lat=(lat-RVR_lat)*11.0988e4;
	long=(long-RVR_long)*9.126e4;
	e=sqrt(lat.*lat + long.*long);
	plot(e)

Generate histogram of position error	hist(e)
Determine mean and variance of position error	mean(e) cov(e)
Plot navigation mode as a function of time	plot(sats)
Generate histogram of navigation mode	hist(sats)

3.0 TEST RESULTS

Sections 3.1 and 3.2 discuss the test results for the static and dynamic tests. The results of the static and dynamic tests provide different information about the overall performance of the GPS receivers. The static test compares the accuracy of the different receivers as they navigate at a surveyed location. The static test also provides some information about the receiver/antenna sensitivity by comparing navigation modes (2D, 3D, or not navigating) of the different receivers over the same time period. Differences in navigation mode may be caused by several factors. One is that the receiver/antenna operating in 2D may not be able to track a satellite close to the horizon. This reflects receiver/antenna sensitivity. Another reason is that different receivers have different DOP limits that cause them to switch to 2-D navigation when four satellites are in view but the DOP becomes too high. This merely reflects the designer's preference in setting DOP switching masks that are somewhat arbitrary.

Dynamic testing was used to compare relative receiver/antenna sensitivity and to determine the amount of time that navigation was not possible because of obstructions. By driving over different types of terrain, ranging from normal city driving to deep canyons, the relative sensitivity of the different receivers was observed. The navigation mode (2D, 3D, or not navigating) was used to compare the relative performance of the receivers. In addition, plots of the data taken give some insight into the accuracy by qualitatively observing the scatter of the data. Because of the qualitative nature of this observation, this data is found in Appendix D and is not analyzed in detail in this report.

The static and dynamic testing results are summarized in Section 3.3.

3.1 Static Test Results

Static testing was conducted at a surveyed location at Sandia National Laboratories' Robotic Vehicle Range (RVR). The position of the surveyed location is described in Table 3-1. The survey was performed by Santiago Romero, Jr., a registered professional surveyor in the State of New Mexico.

Table 3-1. Location of Surveyed Point at Robotic Vehicle Range

Surveyed Latitude	Surveyed Longitude
35 02 27.71607 (deg min sec)	106 31 16.14169 (deg min sec)
35.0410322 (deg)	106.5211505 (deg)

The data taken for this final report was gathered on October 7-8, 1992, from 2:21 p.m. to 2:04 p.m. Although this is the only static data analyzed in this report, a significant amount of additional data was gathered when all of the receivers were not functioning simultaneously. This previously gathered data supported the trends found in the October 7-8 test.

The plots of the static position error for each receiver are shown in Figures 3-1 to 3-5. A summary of the mean and standard deviation (σ) of the position error for the different receivers appears in Table 3-2.

Table 3-2. Summary of Static Position Error Mean and Variance for Different Receivers

Receiver	Mean Position Error (meters)	Position Error STD Deviation (meters)
Magellan	33.48	23.17
Magnavox GPS Engine	22.00	16.06
Rockwell NavCore V	30.09	20.27
Magnavox 6400	28.01	19.76
Trimble Placer	29.97	23.58

As evidenced in Table 3-2, the Magnavox GPS Engine was noticeably more accurate when comparing static position error. The Magellan, Rockwell, Magnavox 6400, and Trimble Placer all exhibit comparable, but larger, average position errors. This trend was also observed when SA was turned off. However, a functioning Rockwell receiver was not available for this test so the data will not be presented. It is interesting to note that the Magnavox 6400 unit compares well with the newer receivers when looking at static accuracy. This is expected. Because the receiver only has two channels, it will take longer to reacquire satellites after blockages and will also have more difficulty with dynamic situations. However, in a static test, the weaknesses of a sequencing receiver are not noticed.

The error distributions for the data taken during the static test are shown in Figures 3-6 to 3-10. Looking at the error distributions, the Magnavox GPS Engine has the most data points within 20 meters of the surveyed position. This corresponds with the smallest mean position error exhibited by the Magnavox receiver. The error distributions for the other four receivers are fairly similar. The Magnavox 6400 unit has slightly more data points in the 10-20 meter error bin, but otherwise there are no unique features. The Magnavox GPS Engine is the only receiver of the five tested that had a noticeably superior static position error distribution.

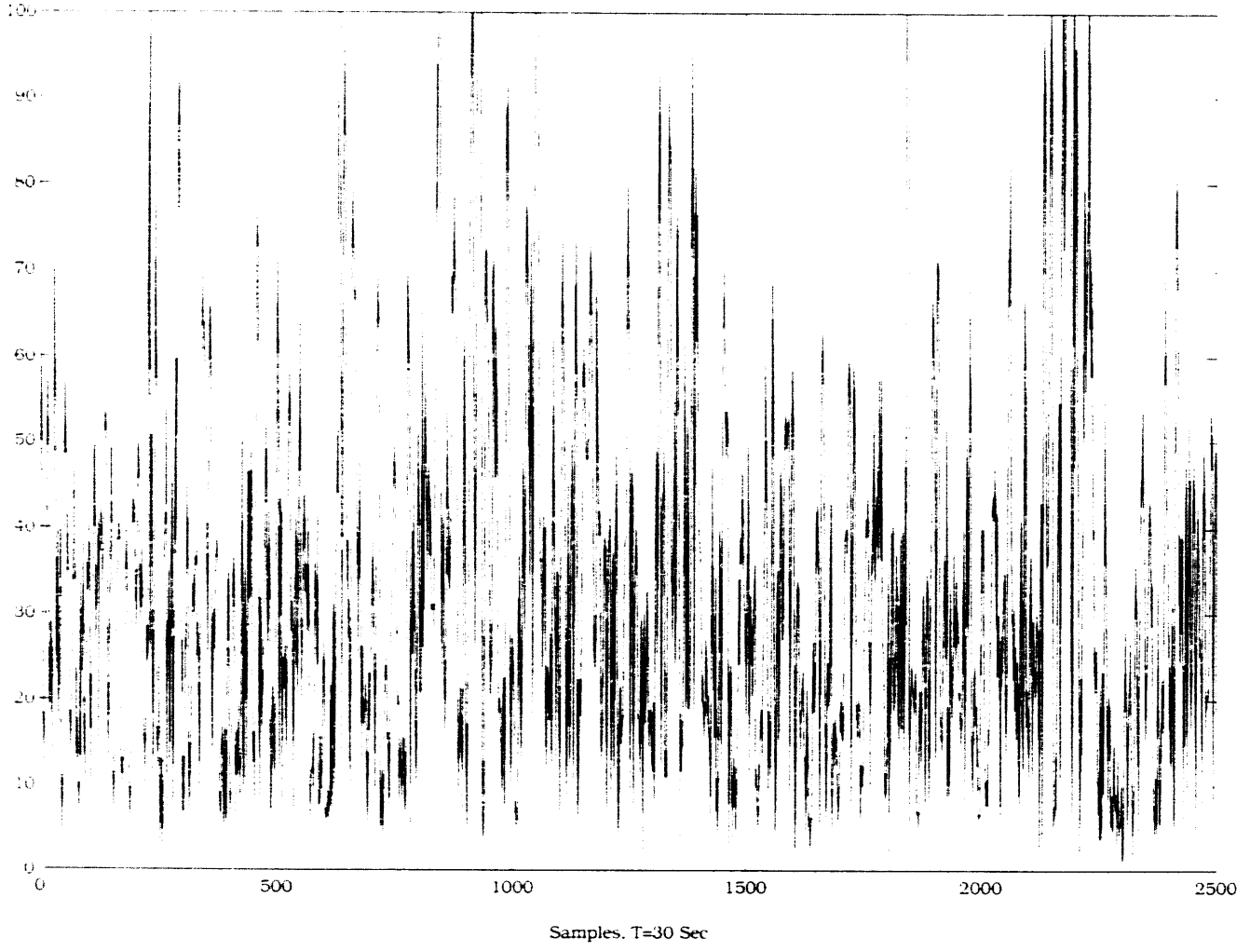


Figure 3-1. Magellan Static Position Error (A100714E.DAT).

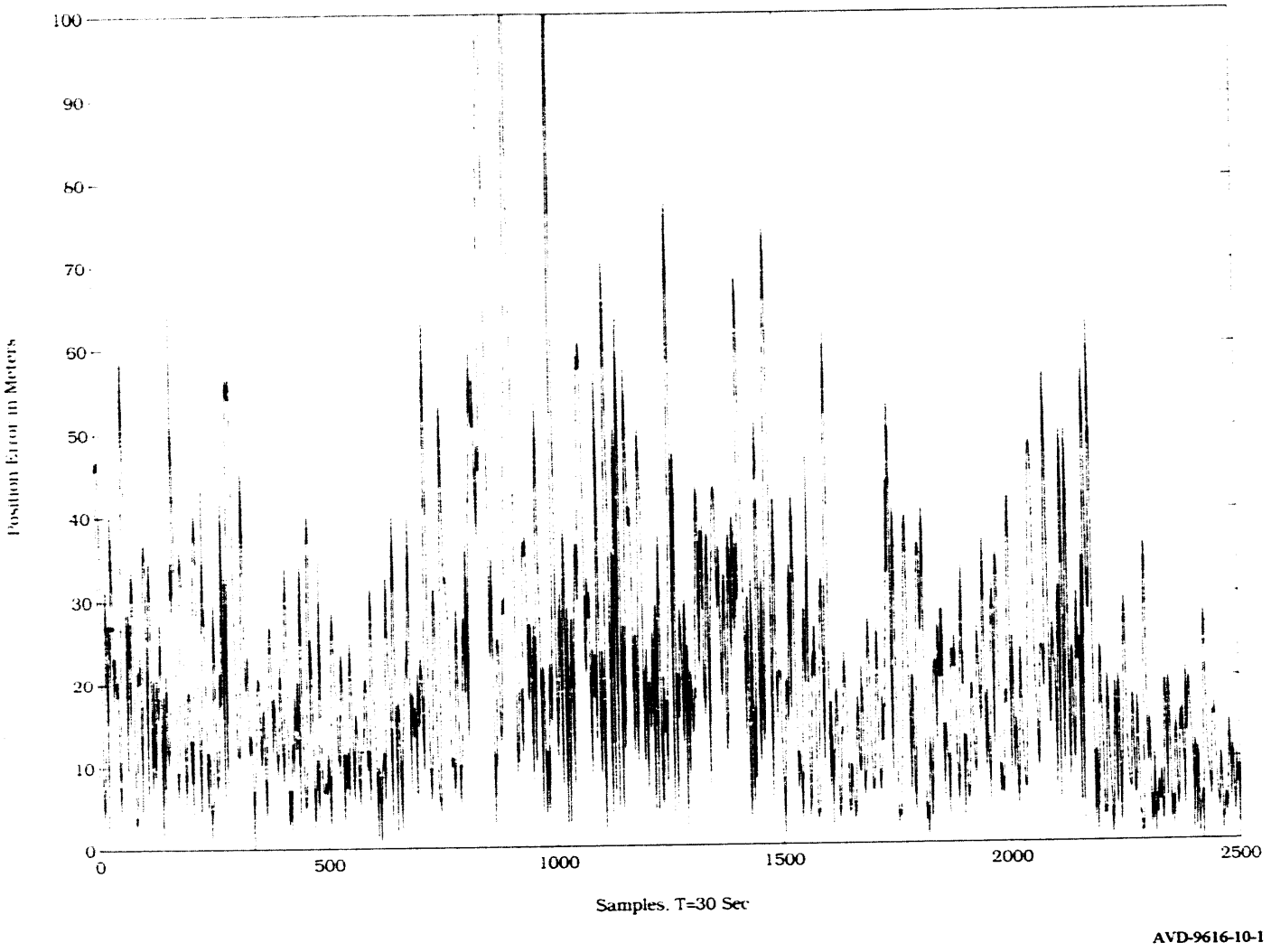


Figure 3-2. Magnavox GPS Engine Static Position Error (B100714E.DAT).

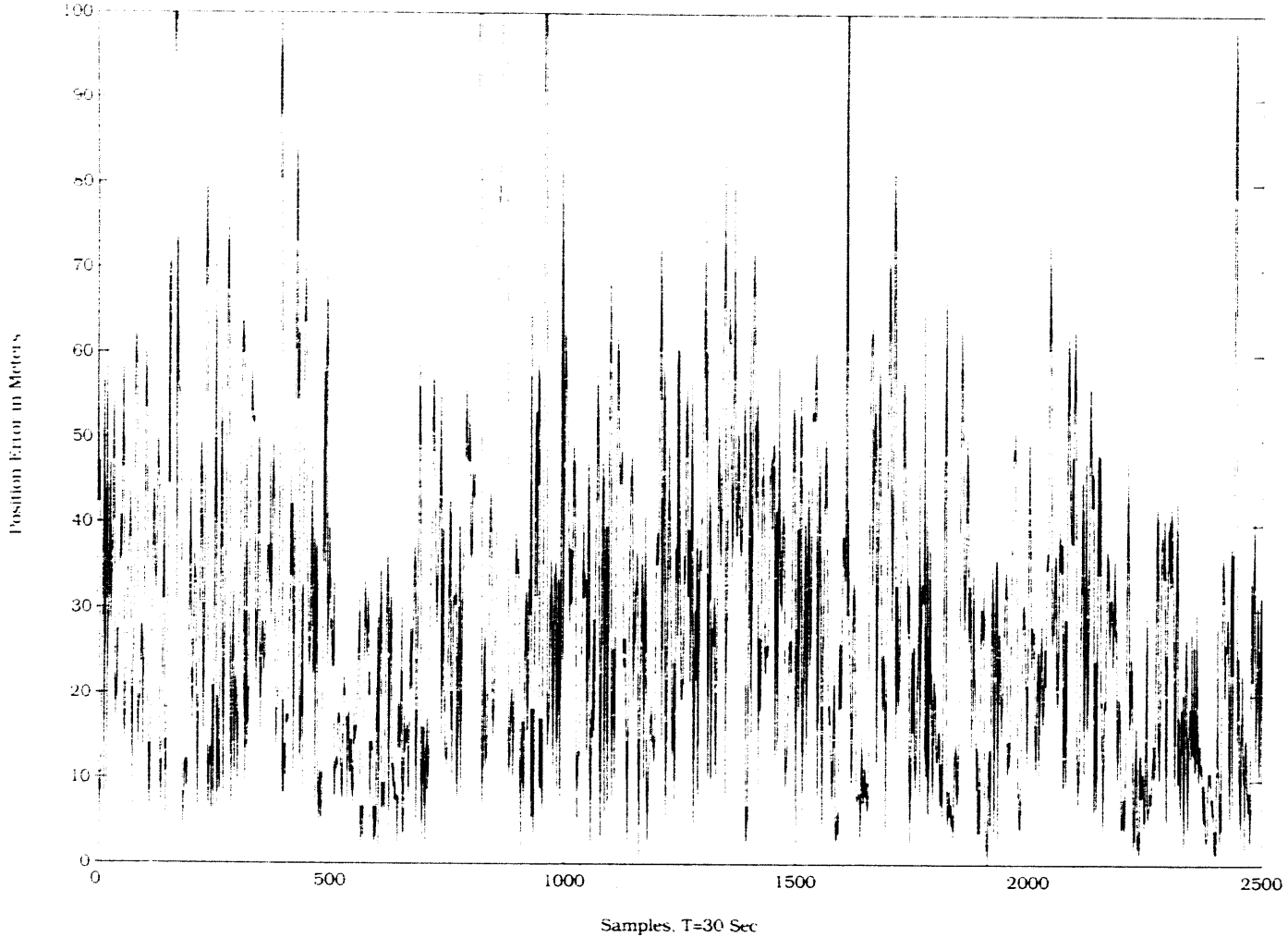


Figure 3-3. Rockwell NavCore V Static Position Error (C100714E.DAT).

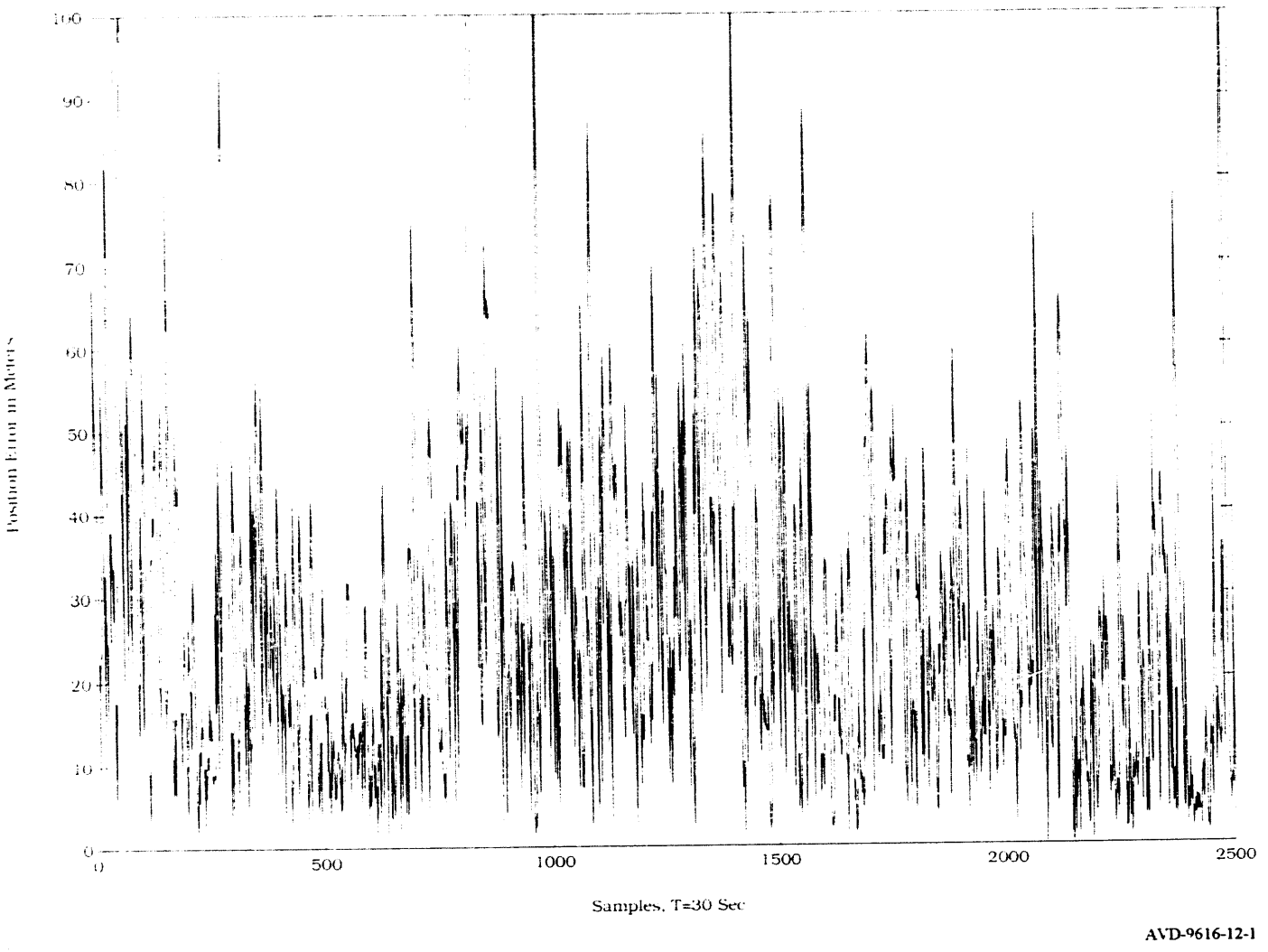


Figure 3-4. Magnavox 6400 Static Position Error (D100714E.DAT).

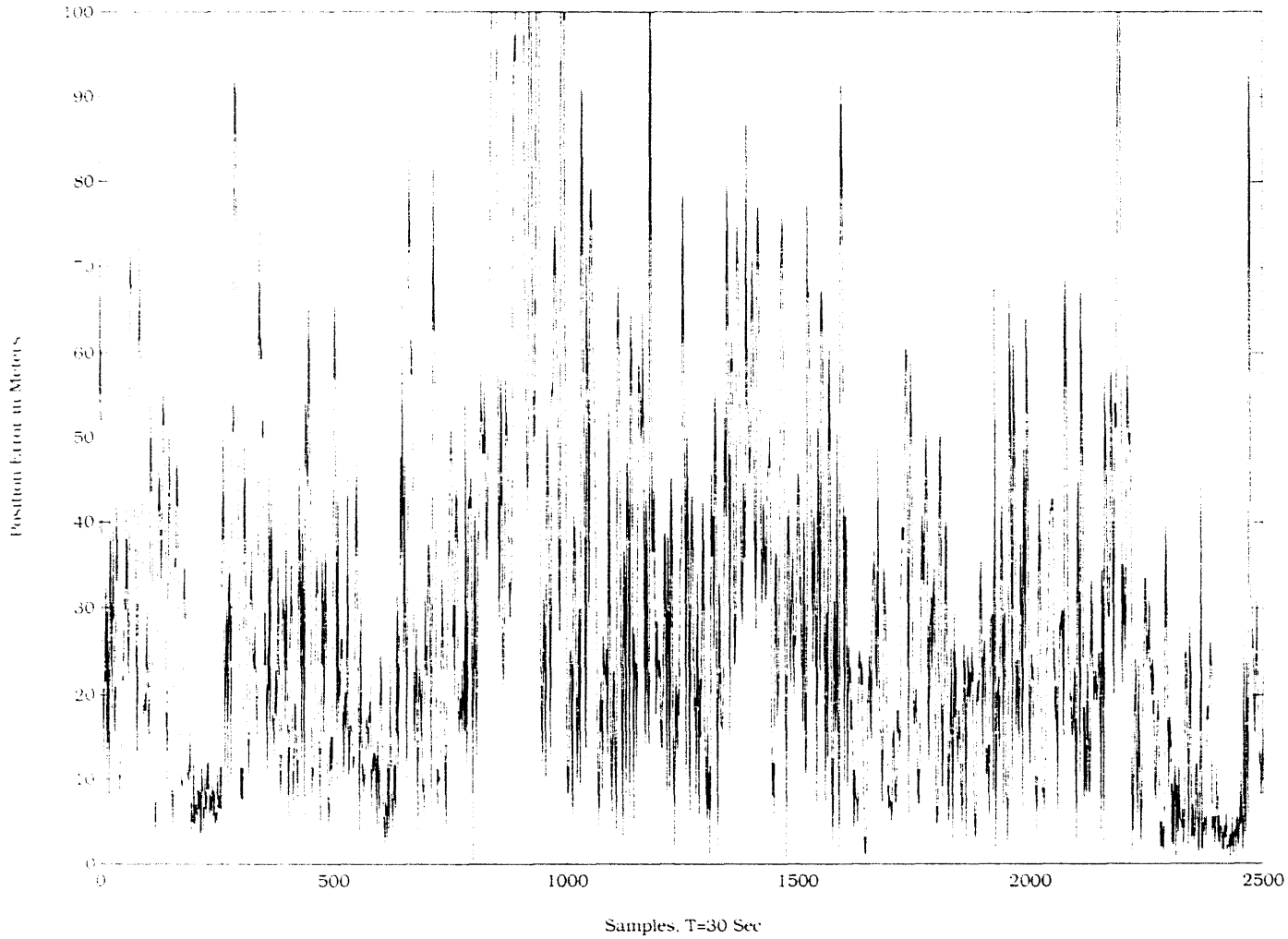


Figure 3-5. Trimble Placer Static Position Error (F100714E.DAT).

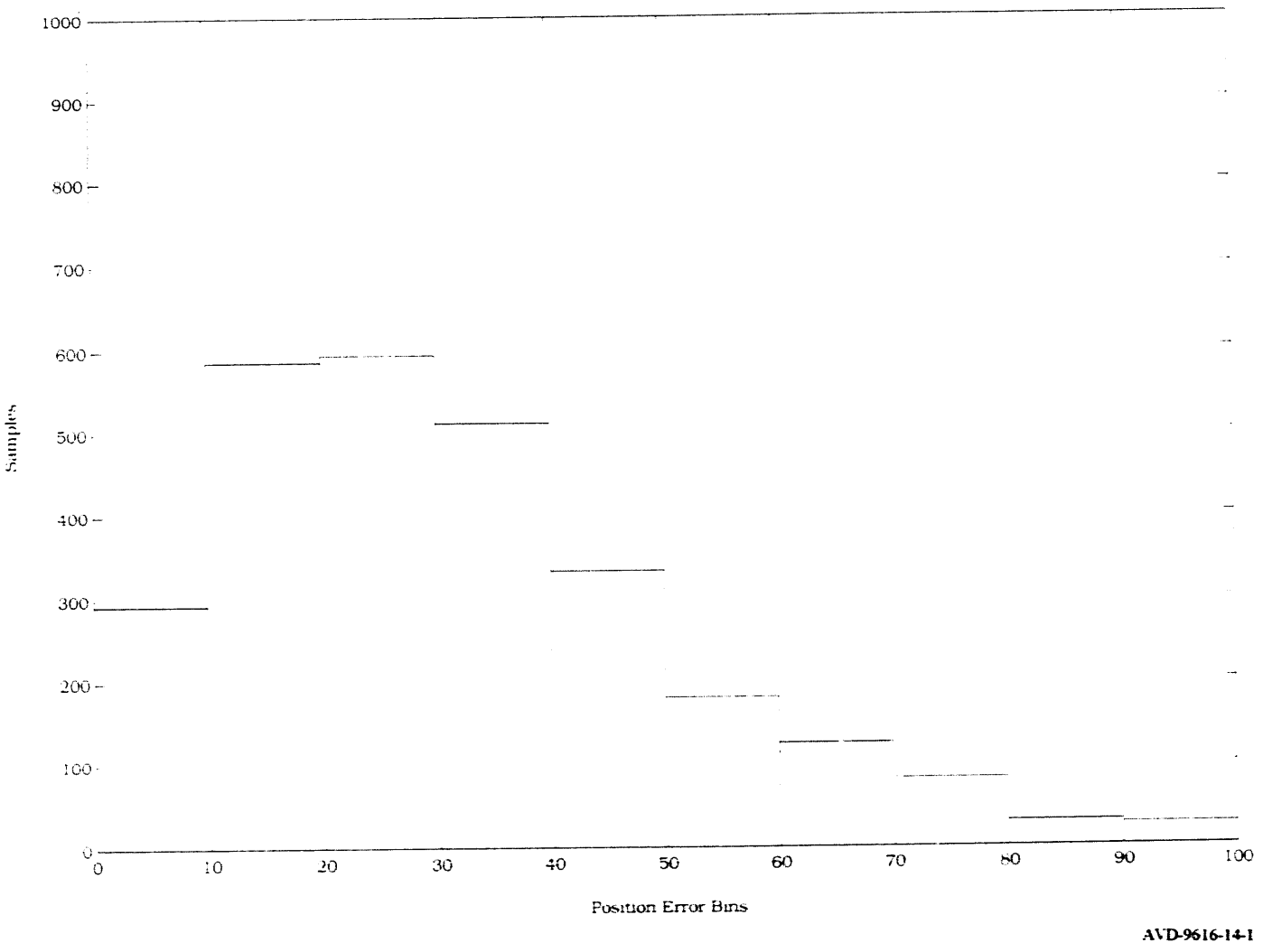


Figure 3-6. Magellan Static Error Distribution (A100714E.DAT).

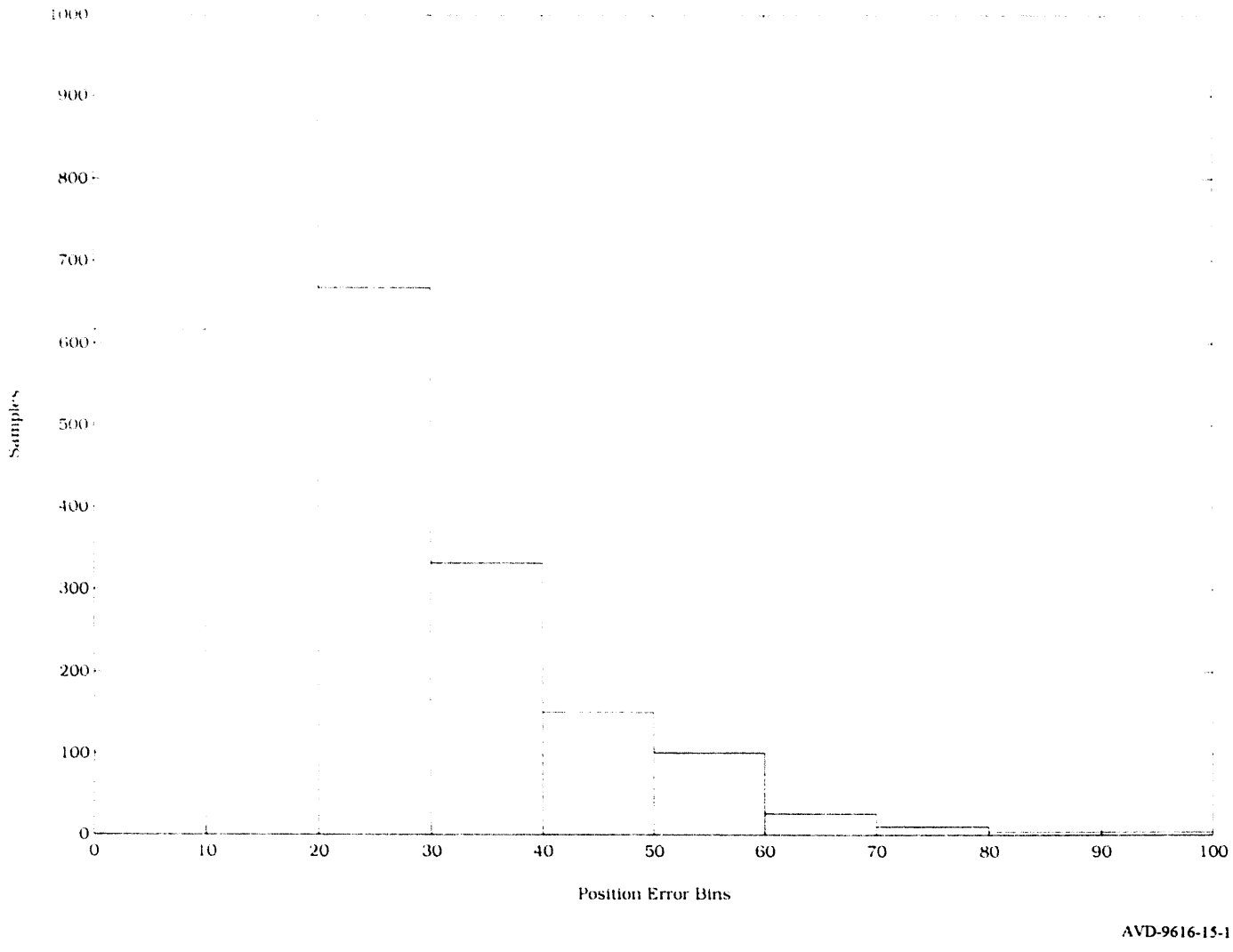


Figure 3-7. Magnavox GPS Engine Static Error Distribution (B100714E.DAT).

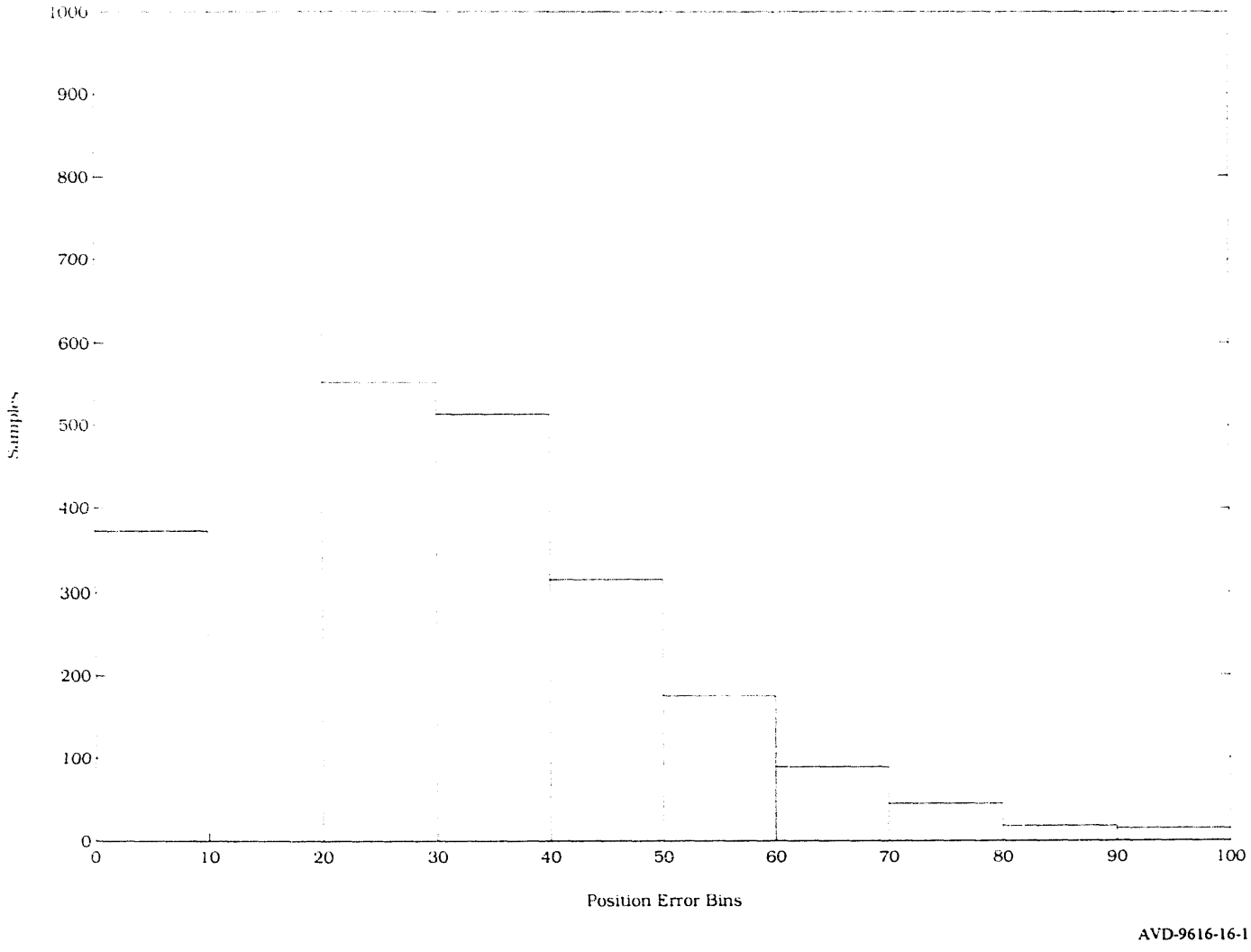


Figure 3-8. Rockwell NavCore V Static Error Distribution (C100714E.DAT).

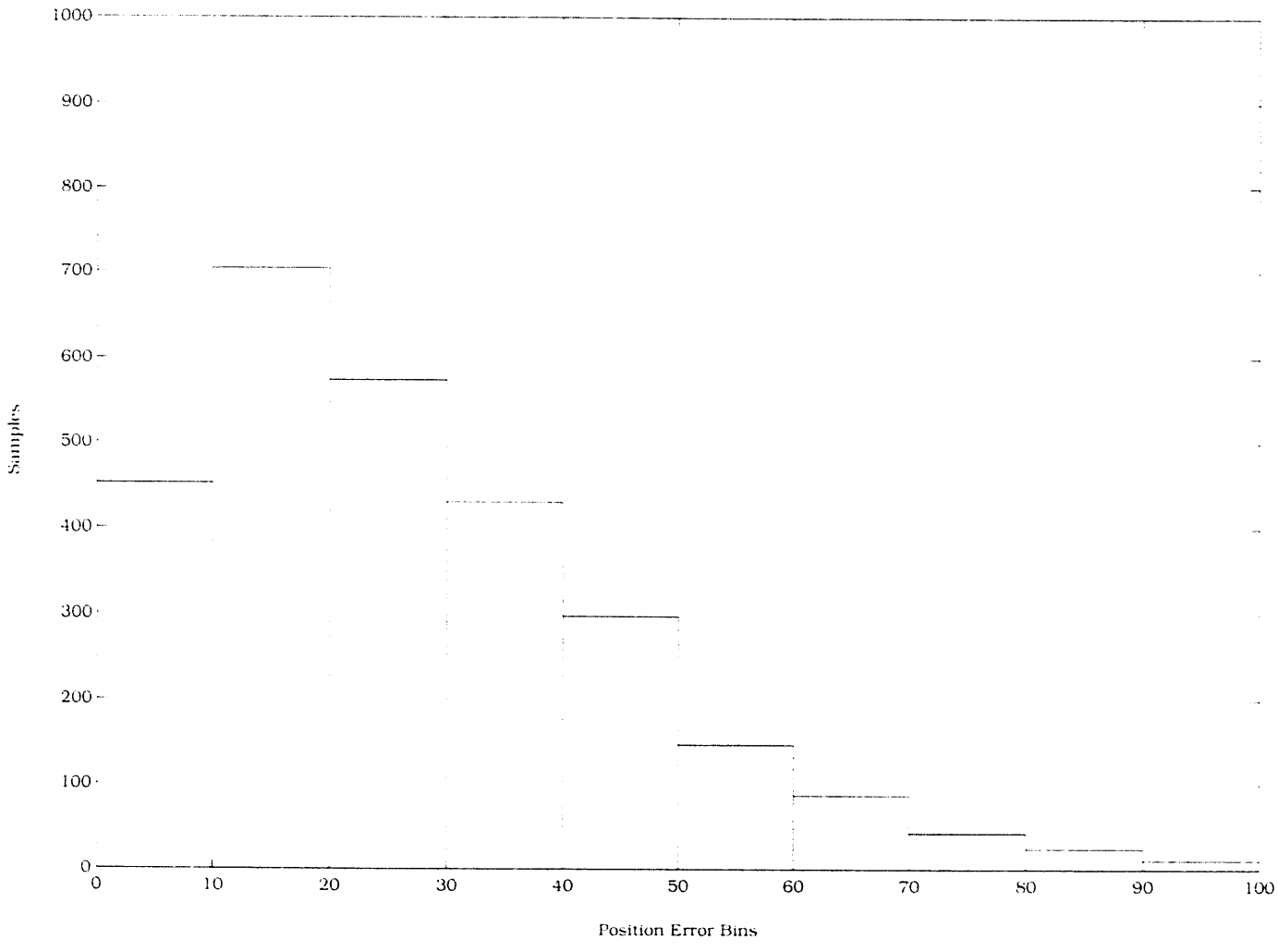


Figure 3-9. Magnavox 6400 Static Error Distribution (D100714E.DAT).

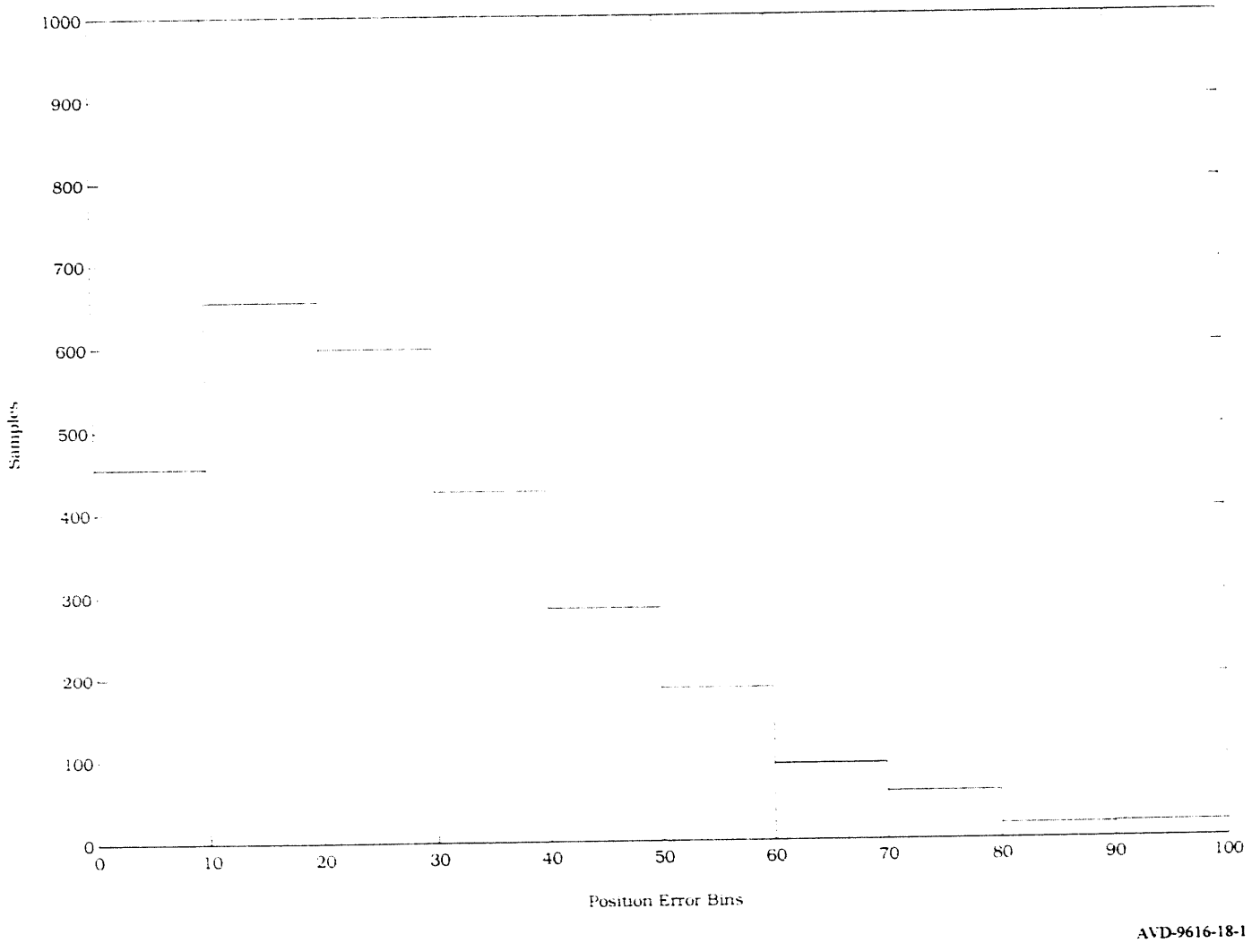
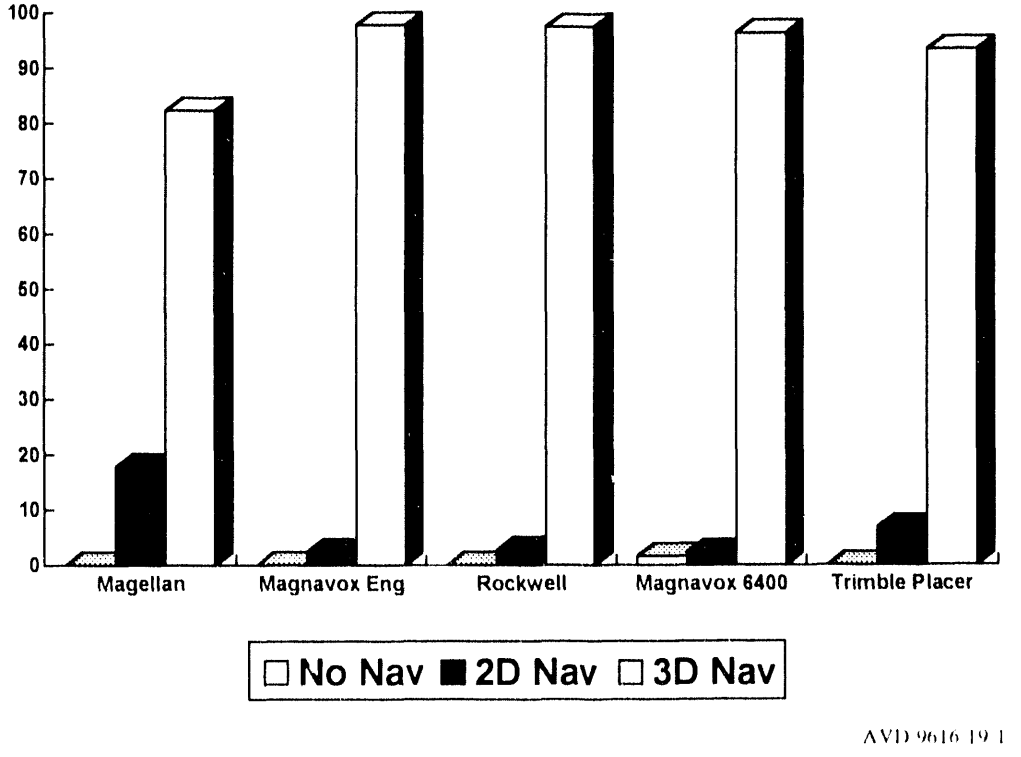


Figure 3-10. Trimble Placer Static Error Distribution (F100714E.DAT).

Navigation mode data for the different receivers is summarized in Table 3-3 for the static test. This data is also presented graphically in Figure 3-11.

Table 3-3. Summary of Navigation Mode Data for Static Test

Receiver	% No Navigation	% 2-D Navigation	% 3-D Navigation
Magellan	0	17.78	82.22
Magnavox Engine	0	2.35	97.65
Rockwell NavCore	0	2.66	97.34
Magnavox 6400	1.55	2.22	96.23
Trimble Placer	0	6.72	93.28



AVD 9616.19.1

Figure 3-11. Navigation Mode Data for Static Test.

In order to analyze the data in Figure 3-11, one needs to take into account the DOP criterion for the different receivers. As mentioned previously, some receivers switch from 3-D navigation to 2-D navigation if four satellites are visible but the DOP is above a predetermined threshold. The DOP switching criterion for the different receivers are outlined in Table 3-4. As seen in Table 3-4, the different receivers use different DOP criterion. However, by taking advantage of Equations (3-1) and (3-2), the different DOP criterions can be compared.

$$PDOP^2 = VDOP^2 + HDOP^2 \tag{3-1}$$

$$GDOP^2 = PDOP^2 + TDOP^2 \tag{3-2}$$

Table 3-4 Summary of DOP - Navigation Mode Switching Criterions

Receiver	2-D/3-D DOP Criterion	PDOP Equivalent
Magellan	If 4 satellites visible and VDOP >7, will switch to 2-D navigation.	$PDOP \geq (HDOP^2 + 7^2)^{1/2}$
	Enters 3-D navigation when VDOP<5.	$PDOP \leq (HDOP^2 + 5^2)^{1/2}$
Magnavox GPS Engine	If 4 satellites visible and VDOP>10, switches to 2-D navigation.	$PDOP \geq (HDOP^2 + 10^2)^{1/2}$
	If HDOP>10, suspends 2-D navigation	
Rockwell NavCore V	If 4 satellites visible and GDOP>13, switches to 2-D navigation.	$PDOP \geq (13^2 - TDOP^2)^{1/2}$
Magnavox 6400	Data Not Available in MX 5400 Manual Provided	
Trimble Placer	If 4 satellites visible and PDOP>8, switches to 2-D navigation. If PDOP>12, receiver stops navigating	$PDOP \geq 8$

Table 3-4 relates all of the different DOP criterions to the PDOP. Based on the information in Table 3-4, several comments can be made about the relative stringency of the various DOP criterions. First, the Magnavox GPS Engine VDOP criterion is much less stringent than the Magellan VDOP criterion (these two can be compared directly). The Magellan unit also incorporates hysteresis, which makes the criterion even more stringent. Comparing the Rockwell to the Trimble Placer, the Rockwell criterion is much less stringent. A TDOP of 10.2 would be required to make the two criterions equivalent. The Rockwell and Magnavox GPS Engine have the least stringent DOP requirements.

Taking into account the DOP criterions of the different receivers, the significant amount of 2-D navigation exhibited by the Magellan receiver might be attributed to a more stringent DOP criterion. However, this did not

improve the horizontal (latitude-longitude) position error. The Magnavox GPS Engine still exhibited the most accurate static position performance. The same can be said for the Trimble Placer unit. Although is has a stricter DOP requirement than the Magnavox Engine, its position location accuracy was not superior. The static navigation mode results don't conclusively show that any receiver has superior sensitivity. However, the static position error results do show that the Magnavox GPS Engine is clearly more accurate than the other receivers tested. The superior accuracy of the Magnavox receiver in the static tests might be attributed to more filtering in the receiver. It should also be noted that the Magnavox 6400 unit was the only receiver that did not navigate for some time period during the static test.

3.2 Dynamic Test Results

The dynamic test data was obtained by driving the instrumented van over different types of terrain. The different types of terrain are summarized in Table 3-5. The various routes were chosen so that the GPS receivers would be subjected to a wide ensemble of obstructions. These include buildings, underpasses, signs, and foliage for the city driving. Rock cliffs and foliage were typical for the mountain and canyon driving. Large trucks, underpasses, highway signs, buildings, foliage, as well as small canyons were found on the interstate and rural highway driving routes.

Table 3-5. Summary of Terrain Used for Dynamic Testing

Terrain Description	Road Sections Used for Testing
City Driving	Central Avenue in Albuquerque, NM from Tramway to 7th Street
Mountain Driving	Tijeras Canyon, I-40, East and West I-40 Tramway Exit (Albuquerque, NM) to Sedillo (NM) Exit
Interstate Highway Driving	I-25, North and South I-25 Tramway Exit (Albuquerque, NM) to NM 285 Exit in S[illegible]e, NM
Rural Highway Driving	NM 285 and NM 4 Intersection of NM 285 and I-25 (Santa Fe, NM) to intersection (Los Alamos, NM) where LANL* Truck Route (East Jemez Road) begins (at traffic light)
Canyon Driving	LANL* Truck Route (East Jemez Road) in Los Alamos, NM from traffic light to traffic light, climbing from Sandia Canyon up to the South Mesa

* Los Alamos National Laboratory

The dynamic testing was carried out over the course of several days. The dates of the different dynamic test routes are summarized in Table 3-6.

Table 3-6. Summary of Dynamic Test Dates

Terrain Description	Dates Testing Performed
City Driving	10/22 and 10/25/92
Mountain Driving	10/25 and 10/26/92
Interstate Highway Driving	10/27 and 10/28/92
Rural Highway Driving	10/27 and 10/28/92
Canyon Driving	10/27 and 10/28/92

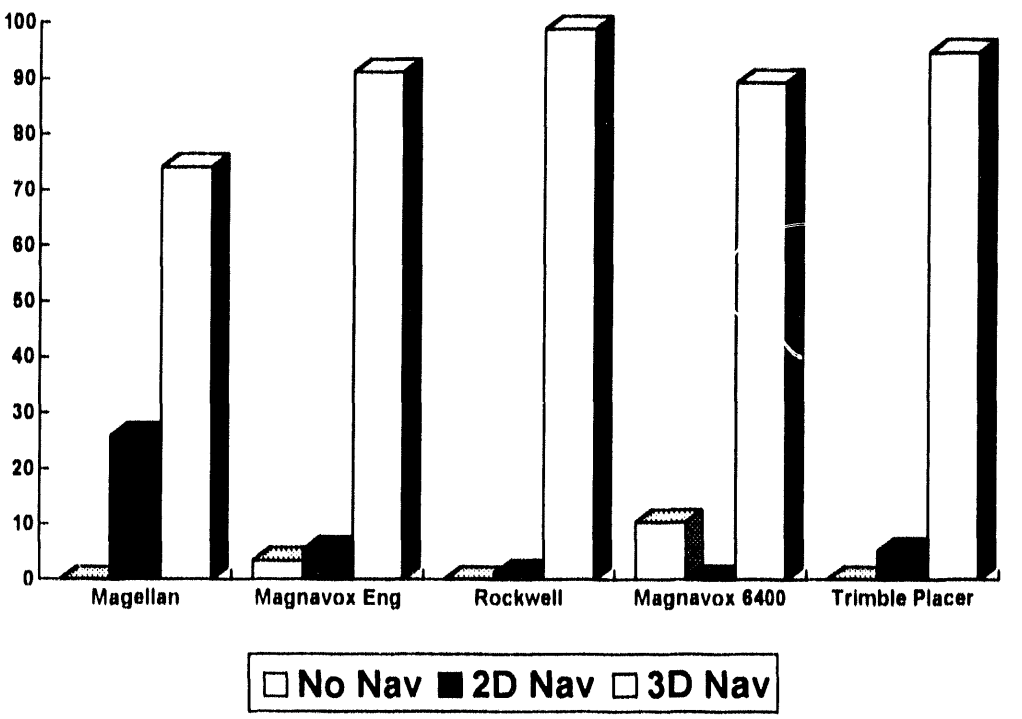
The results of the dynamic testing are presented in two formats. First, a table is used to summarize the percentage of 2-D navigation, 3-D navigation, and no navigation (Tables 3-7 through 3-11). Then this information is presented graphically in the form of a bar graph (Figures 3-12 through 3-16). The raw data, including plots of the actual GPS position data, appears in Appendix D.1. The dynamic test results as well as a discussion of the results appear on the following pages.

Several noticeable differences exist between Figure 3-11 (Static Navigation Mode) and Figure 3-12. The Magnavox 6400 unit is not navigating a significant portion of the time. This is because sequencing receivers do not perform as well in dynamic environments with periodic obstructions. The Magellan GPS receiver also navigated in 2D a larger percentage of the time compared with the other receivers. The Rockwell unit was able to navigate in 3D the largest percentage of the time. Although this is also a result of the Rockwell DOP setting discussed in the previous section, it does seem to indicate that the Rockwell receiver might have slightly better sensitivity (Rockwell claims this is one of the receiver's selling points). The Magnavox GPS Engine also did not navigate a small percentage of the time. This can be attributed to the small period of time when the receiver was obstructed and the other receivers (which also were obstructed) might not have been outputting data (caused by asynchronous sampling).

The mountain driving actually yielded less obstructions than the city driving. This might be a result of better satellite geometries during the test period. However, the Magnavox 6400 unit once again did not navigate for a significant portion of the time. The Magellan receiver navigated in 2D a significant portion of the time, but this can be attributed to some degree to the stricter DOP limits. The performance of the Rockwell Nav Core V, Trimble Placer, and Magnavox GPS Engine are comparable.

Table 3-7 Summary of City Driving Results

Receiver	% No Navigation	% 2-D Navigation	% 3-D Navigation
Magellan	0	25.82	74.18
Magnavox Engine	3.42	5.34	91.24
Rockwell Nav V	0	1.12	98.88
Magnavox 6400	10.34	0.22	89.44
Trimble Placer	0	5.18	94.82



AVD 9616.20.1

Figure 3-12. Summary of City Driving Results

Table 3-8. Summary of Mountain Driving Results

Receiver	% No Navigation	% 2-D Navigation	% 3-D Navigation
Magellan	0	12.29	87.71
Magnavox Engine	0	0.97	99.03
Rockwell Nav V	0	0	100
Magnavox 6400	4.55	0	95.45
Trimble Placer	0	1.3	98.70

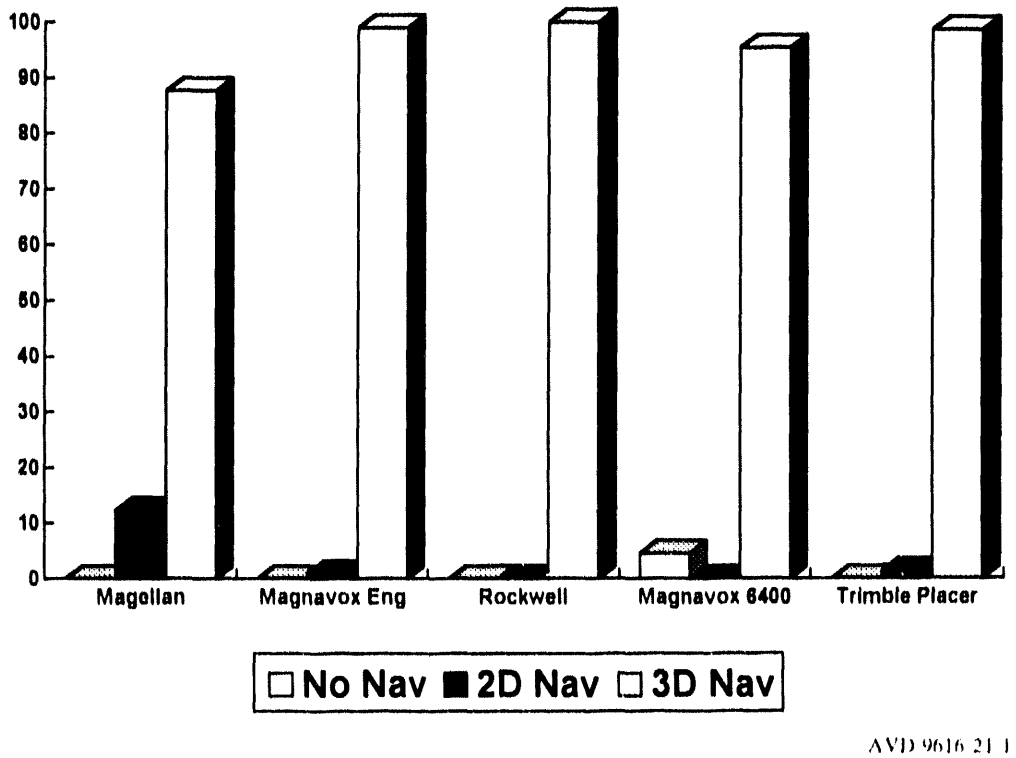
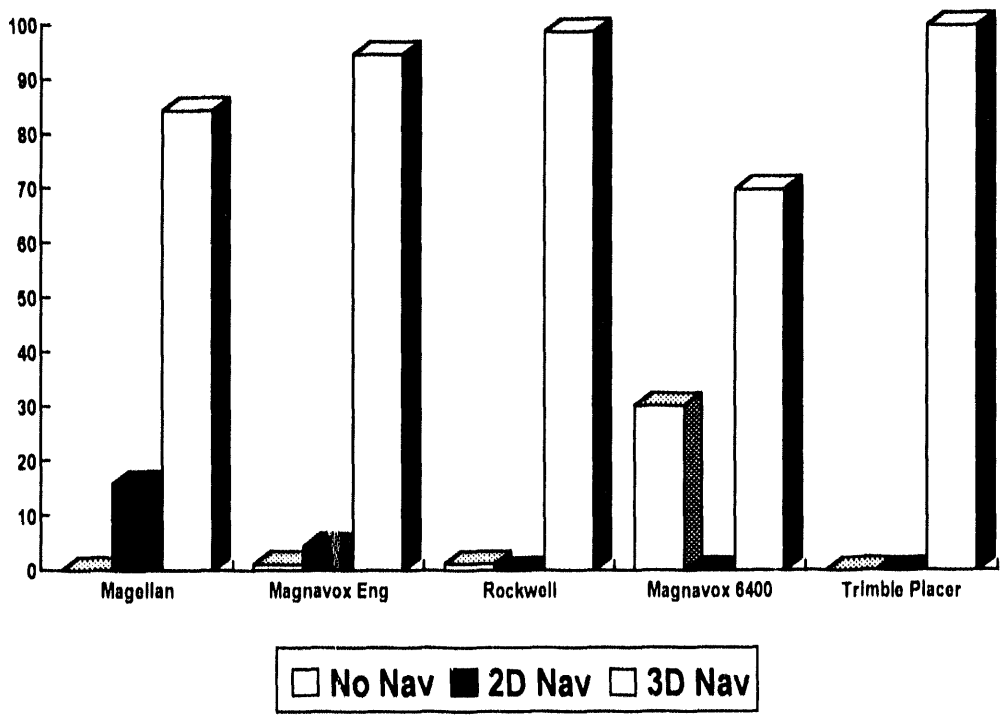


Figure 3-13. Summary of Mountain Driving Results.

Table 3-9. Summary of Canyon Driving Results

Receiver	% No Navigation	% 2-D Navigation	% 3-D Navigation
Magellan	0	15.73	84.27
Magnavox Engine	1.09	4.35	94.57
Rockwell Nav V	1.18	0	98.82
Magnavox 6400	30.17	0	69.83
Trimble Placer	0	0	100



AVD 9616.22.1

Figure 3-14. Summary of Canyon Driving Results.

Table 3-10. Summary of Interstate Highway Results

Receiver	% No Navigation	% 2-D Navigation	% 3-D Navigation
Magellan	0	32.82	67.18
Magnavox Engine	0.38	0.38	99.25
Rockwell Nav V	0.20	0.20	99.61
Magnavox 6400	20.08	0	79.92
Trimble Placer	0	4.15	95.85

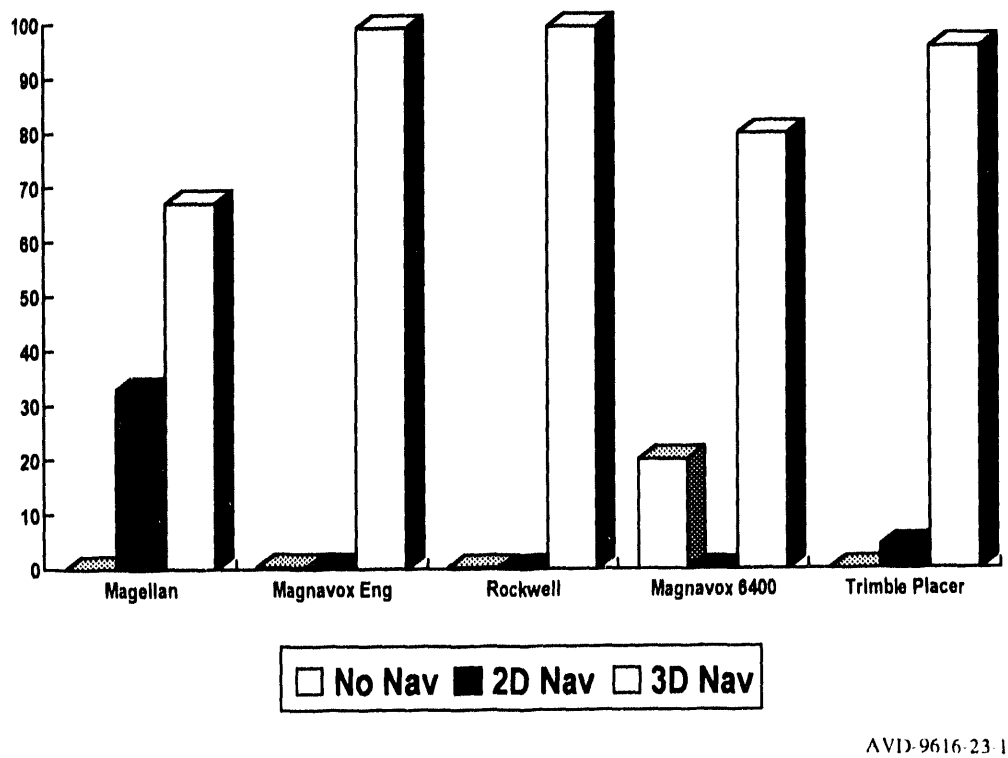


Figure 3-15. Summary of Interstate Highway Results.

Table 3-11. Summary of Rural Highway Results

Receiver	% No Navigation	% 2-D Navigation	% 3-D Navigation
Magellan	0	7.35	92.65
Magnavox Engine	0.26	1.28	98.47
Rockwell Nav V	1.63	0.54	97.83
Magnavox 6400	10.39	1.82	87.79
Trimble Placer	0	3.89	96.11

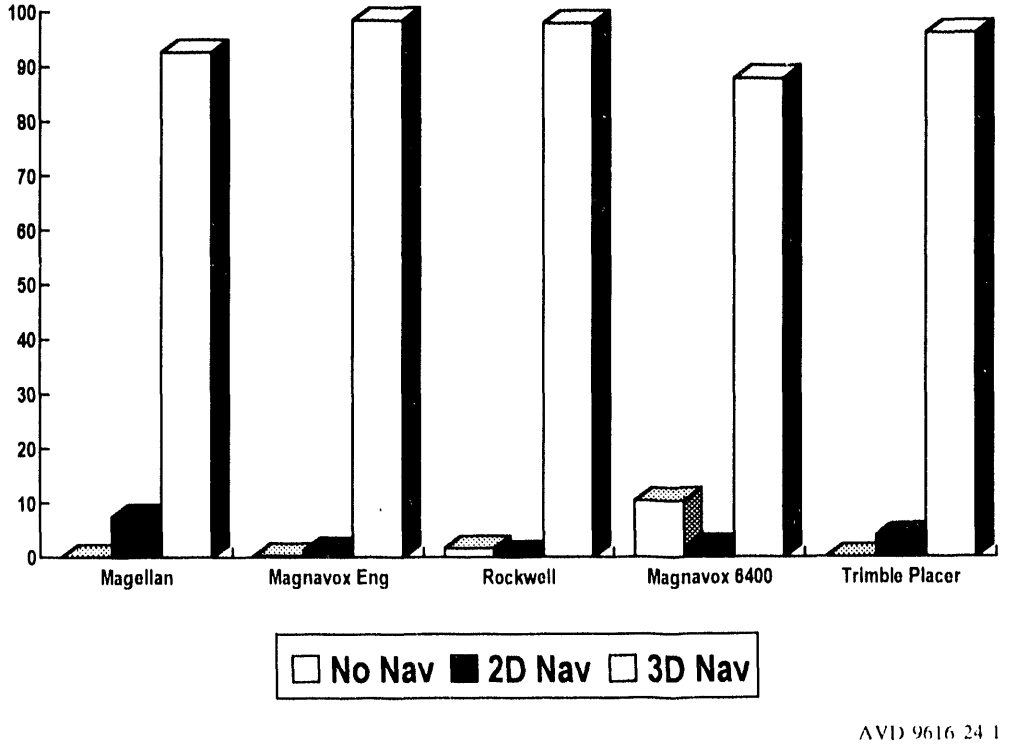


Figure 3-16. Summary of Rural Highway Results.

The LANL Truck Route Canyon testing exposed the GPS receivers to the most obstructions. The steep canyon walls and abundant foliage stopped the current receiver from navigating over 30 percent of the time. The Magnavox GPS Engine and Rockwell receiver were also not navigating a small percentage of the time. This particular test clearly shows the superiority of the newer receivers over the older sequencing receiver. Because the newer receivers are able to track extra satellites and recover more quickly from obstructions, they are better suited for operation in dynamic environments with periodic obstructions. The Trimble Placer and Rockwell receiver performed the best in this particular test, followed closely by the Magnavox GPS Engine.

During the Interstate Highway driving tests, the Magnavox 6400 unit did not navigate over 20 percent of the time. This is consistent with the sometimes poor performance exhibited by the current navigation system. The other newer receivers did quite well, with the Trimble Placer, Magnavox GPS Engine, and Rockwell Nav Core V exhibiting similar performance. Once again, the Magellan unit navigated in 2D a significant portion of the time. This can probably be attributed to the stricter DOP limits.

During the Rural Highway testing, the Magnavox 6400 unit once again did not navigate a significant portion of the time. All of the newer receivers had similar performance results. The Magellan receiver navigated in 2D considerably less in this test compared to the other dynamic tests.

3.3 Summary of Test Results

Both static and dynamic tests were used to compare the performance of the five different GPS receivers. The static test results showed that the Magnavox GPS Engine was the most accurate (for static situations). The other four receivers were slightly less accurate and exhibited similar static position error performance. The static navigation mode results did not differentiate the sensitivity of the various receivers significantly. The Magellan unit navigated in 2D much more frequently than the other receivers, but some of this can be attributed to stricter DOP limits. However, the stricter DOP limits of the Magellan receiver and Trimble Placer did not yield better static position accuracies. All four of the newer GPS receivers obtained a first fix under one minute, which verifies the time to first-fix specifications stated by the manufacturers.

The dynamic tests were used to differentiate receiver sensitivity and the ability to recover quickly from periodic obstructions. As expected, the Magnavox 6400 unit did not perform very well in the dynamic testing. The Magnavox 6400 was unable to navigate for some period of each dynamic test. This was most noticeable in the LANL Truck route canyon, where the receiver did not navigate over 30 percent of the time. The newer receivers performed much better in the dynamic testing, navigating almost all of the time. The Magnavox GPS Engine, Rockwell Nav Core V, and Trimble Placer exhibited comparable receiver/antenna sensitivity during the dynamic testing based on the navigation mode data. The Magellan unit navigated in 2D significantly more than the other receivers in the dynamic tests. Most of this can probably be attributed to a more stringent DOP requirement. It

should also be noted that the Magellan receiver was the only receiver to navigate (2D or 3D) 100 percent of the time in all of the dynamic tests.

Overall, the four newer receivers performed significantly better than the Magnavox 6400 unit in the dynamic tests. In the static test, all of the receivers performed satisfactorily, but the Magnavox GPS Engine exhibited the most accurate position estimation. Recommendations on choosing a GPS receiver are outlined in the next section.

4.0 RECOMMENDATIONS

The next two sections outline recommendations for choosing a GPS receiver as well as an integrated navigation system. Section 4.1 discusses the choice of a GPS receiver, while Section 4.2 discusses the requirements of the overall navigation system.

4.1 GPS Receiver

In order to discuss some of the integration issues involved with GPS receivers, a list of the problems encountered with the receivers tested is outlined in Table 4-1. The problems encountered with the Magnavox 6400 unit (there were several) are not listed because the Magnavox 6400 unit is not comparable to the newer receivers in performance.

Based on the problems experienced testing the GPS receivers as well as the requirements of the current application, a list of critical issues is outlined in Table 4-2.

One critical integration issue not mentioned in Table 4-2 is price. Almost any level of performance can be purchased, but at a significantly increased cost. This issue will be addressed further in the next section. Overall, the Magellan OEM Module, the Magnavox GPS Engine, Rockwell Nav Core V, and Trimble Placer are good receivers. The Magnavox GPS Engine exhibited superior static position accuracy. During dynamic testing, all of the receivers were able to navigate a large percentage of the time, even in hilly wooded terrain. Based on the experimental results, other integration issues such as price, software flexibility, technical support, size, power, and differential capability are probably the most important factors to consider when choosing a GPS receiver.

4.2 Overall Navigation System

The current navigation system uses dead reckoning (compass and odometry) and TRANSIT to supplement GPS. Because of the high value of the assets, a redundant navigation system is required. If the assumption is made that GPS is going to be operational 100 percent of the time for the long term, then the design of the navigation system is greatly simplified. Assuming full GPS coverage, a redundant navigation system must be designed that can handle short periods of time when GPS is not available because of obstructions. The dead-reckoning system only needs to be accurate for up to several minutes, which makes the design of the dead-reckoning system much easier (and less expensive).

Table 4-1. Summary of Problems Encountered with GPS Receivers Tested

Receiver	Problems Encountered
Magellan OEM Module	No problems, unit functioned correctly out of the box. However, the current drain on the battery for the battery backed RAM seemed high. A 1 Amp-Hour 3.6 volt Lithium battery only lasted a few months. The Binary position packet was used because of the increased position resolution. Sometimes the receiver outputs a garbage binary packet (about 1 percent of the time).
Magnavox GPS Engine	The first unit received was a pre-production unit. It had a difficult time tracking satellites. On one occasion it took over 24 hours to obtain a first fix. This receiver was returned to Magnavox. Magnavox claimed that upgrading the software fixed the problem. However, the EEPROM failed when trying to load the oscillator parameters. A new production board was shipped and it functioned flawlessly out of the box. The RF connector for the Magnavox GPS Engine was also difficult to obtain. The suppliers recommended in the back of the GPS Engine Integration Guide have large minimum orders. A sample connector was finally requested. It never arrived and a second sample had to be requested.
Rockwell Nav Core V	The first Rockwell receiver functioned for a while, and then began outputting garbage at 600 baud (9600 baud is the only selectable baud rate). Rockwell claims that a Gallium Arsenide IC that counts down a clock signal was failing because of contamination from the plastic package of the IC (suppliers fault). This Rockwell unit was returned for repair under warranty. The second Rockwell unit tested output data but did not navigate. Power was applied to the unit with reverse polarity (Sandia's fault) and an internal rectifier bridge allowed the unit to function, but not properly. Applying power in the correct manner (positive on the outside contact) fixed the problem.
Trimble Placer	No problems, unit functioned correctly out of the box.

Table 4-2. Summary of Critical Integration Issues

Critical Issue	Rationalization
Flexible Software Interface	Having the flexibility to control the data output by the receiver is important. This includes serial data format (TTL, RS-232, RS-422), baud rates, and packet data rates. It is desirable to have the receiver output position data at fixed data rate, that is user selectable. It is also desirable to be able to request other data packets when needed All of the receivers with the exception of the Rockwell unit were fairly flexible. The Rockwell unit on the otherhand outputs position data at a fixed 1-Hz rate and fixed baud rate of 9600 baud. The format of the data packets is also important. ASCII formats are easier to work with because the raw data can be stored and then analyzed visually. The Rockwell unit uses an IEEE floating point format. Although Binary data formats and the Rockwell format might be more efficient, it is much easier to troubleshoot a problem when the data does not have to be post processed just to take a quick look.
Differential Capability	The capability to receive differential corrections is important if increased accuracy is desired. Although a near-term fielded system might not use differential corrections, the availability of subscriber networks that broadcast differential corrections in the future will probably make this a likely upgrade.
Time to first fix	A fast time-to-first-fix is important. However, all newer receivers usually advertise a first fix in under one minute when the receiver knows its approximate position. The difference between a 30-second first fix and a one-minute first fix is probably not that important. This parameter also affects how quickly the receiver can reacquire satellites after blockages.
Memory Back Up	Different manufacturers use different approaches for providing power to back up the static memory (which stores the last location, almanac, ephemeris, and receiver parameters) when the receiver is powered down. These include an internal lithium battery, an external voltage supplied by the integrator, and a large capacitor. The large capacitor has the advantage of never needing replacement. This approach is taken on the Rockwell Nav Core V. However, the capacitor charge can only last for several weeks. An internal lithium battery can last for several years, but will eventually need replacement. An external voltage supplied by the integrator can come from a number of sources, but must be taken into account when doing the system design.
Size, Power, and Packaging	Low power consumption and small size are advantageous for vehicular applications. Some manufacturers also offer the antenna and receiver integrated into a single package. This has some advantages, but limits antenna choices.

Table 4-2. Summary of Critical Integration Issues (continued)

Critical Issue	Rationalization
Active/Passive Antenna	Active antennas with built in amplifiers allow longer cable runs to the receiver. Passive antennas require no power but can not be used with longer cabling because of losses.
Cable length and number of connectors	The losses in the cabling and connectors must be taken into account when designing the cabling and choosing the appropriate antenna.
Receiver/Antenna Sensitivity	Increased receiver/antenna sensitivity will reduce the affects of foliage and other obstructions. The sensitivity is affected by the receiver, the cabling, as well as the antenna used.
Position Accuracy	Both static and dynamic position accuracy are important. However, the affects of SA reduce the accuracy of all receivers significantly. Differential accuracy will become an important parameter in the future.
Technical Support	Good technical support, including quick turn around times for repairs, is very important. Quick turn around for failed units can also be accomplished by keeping spares in stock.

The main integration issue is the level of coupling between the dead-reckoning system and the GPS receiver. A highly integrated system will be defined as a system where the dead reckoning and GPS are supplied by a single manufacturer as a complete navigation system. The current navigation system supplied by Magnavox would be an example of a highly integrated system. A loosely integrated system will be defined as a system where the dead reckoning and GPS system are purchased or designed separately and then integrated by Sandia. The highly integrated system can actually be broken down further into two categories. One, a tightly coupled system, can be defined as a system where the dead-reckoning or inertial data is used in the same filter as the GPS data to estimate the current position. The other highly integrated system, a loosely coupled system, will refer to an integrated navigation system from a single manufacturer that does not include the inertial or dead-reckoning data in the GPS filter. These different configurations are depicted in Figures 4-1 and 4-2.

The tightly coupled configuration can use a centralized filter or cascade filter design, depending on the level of integration (Institute of Navigation, 1992). The loosely coupled system in Figure 4-2 results in a sub-optimal configuration because of the cascading of filters. However, it is often difficult to obtain raw GPS data. The loosely coupled system also has some advantages. These include easier fault detection and isolation as well as economy of operation (several reduced state filters rather than one large filter) (Institute of Navigation, 1992).

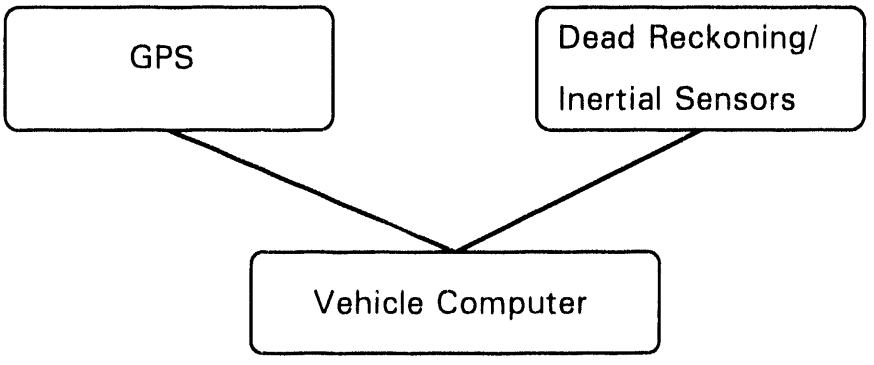


Figure 4-1. Loosely Integrated System.

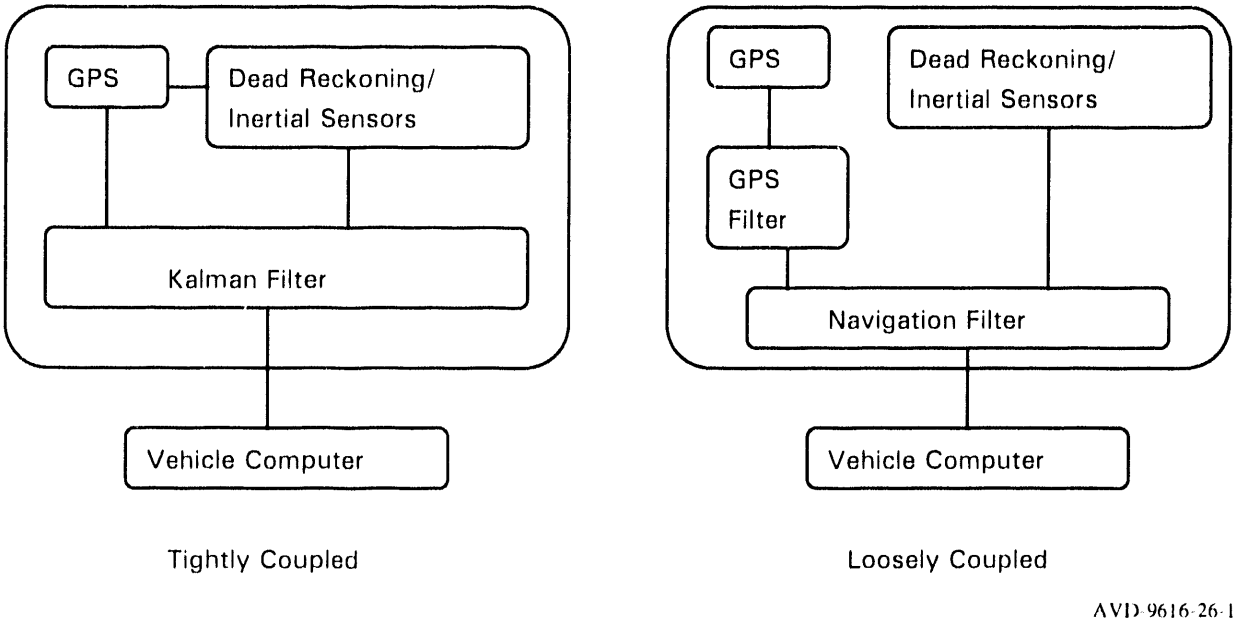


Figure 4-2. Highly Integrated Systems.

The only difference between the loosely integrated system in Figure 4-1 and the loosely coupled highly integrated system in Figure 4-2 is that the systems integration is already done for the highly integrated system, while Sandia (or a contractor) must do the integration for the loosely integrated system.

Tightly coupled GPS-Inertial Systems are being developed by several companies. However, no one has developed a tightly coupled commercial system where dead-reckoning information, combined with a vehicle model, is used with GPS in a single Kalman Filter. Such a system is depicted in Figure 4-3. The advantages of this type of configuration include optimal position estimation. However, a considerable amount of time would have to be spent developing a vehicle model as well as characterizing the sensor noise. In addition, many of the functions already performed by a GPS receiver would not be utilized because the inputs to the filter would be the pseudo range estimates. Although this type of highly integrated position location system would provide an accurate and optimal position estimate, the time spent developing a good vehicle model and characterizing sensors would make the overall system very expensive.

Because of the expense involved in developing the tightly coupled position location system in Figure 4-3, the most economical approach would be to purchase a loosely coupled highly integrated system or to integrate a loosely integrated system. There are advantages and disadvantages to both approaches. These are outlined in Table 4-3.

As seen in Table 4-3, the only real disadvantage of integrating a loosely integrated system is that Sandia must take the time to do the integration. The prices of several different systems are outlined in Table 4-4 to compare the cost of the two options.

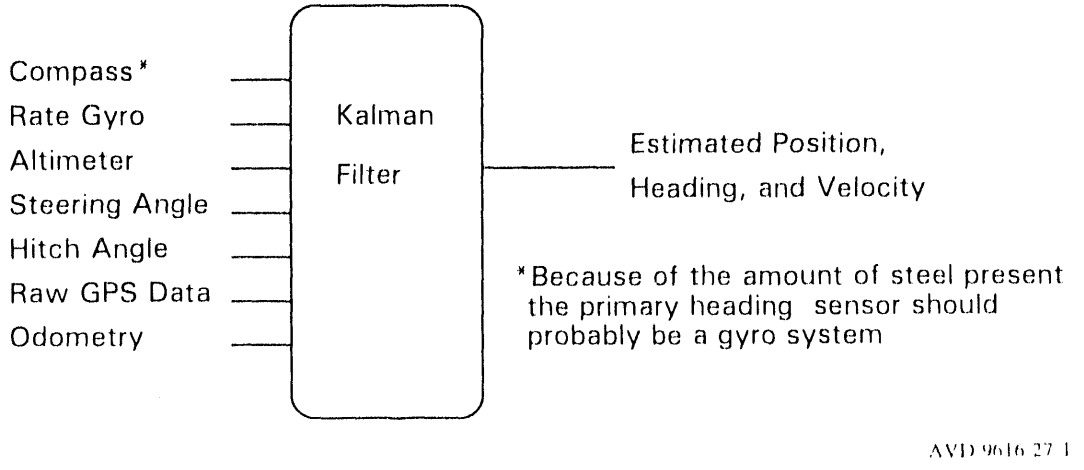


Figure 4-3. Tightly Coupled GPS Navigation System.

Table 4-3. Purchasing a Highly Integrated Navigation System vs. Performing the System Integration In-House

Purchase a Loosely Coupled Highly Integrated System	Integrate a Loosely Integrated System
Advantages	Advantages
• Complete navigation system is an off-the-shelf item. • Simplified maintenance because off-the-shelf system.	• Sandia is not locked into a particular manufacturer for the GPS system. • GPS receivers can be updated as new technologies become available without affecting the dead reckoning system. • Sandia maintains vehicle navigation tech base that can be applied to IVHS research.
Disadvantages	Disadvantages
• The number of off-the-shelf integrated systems is limited. These include the Trimble Placer DR and a semi-custom Magnavox system. • Sandia will be locked into a particular manufacturer for the complete navigation system.	• Sandia has to develop the dead reckoning system (might not really be a disadvantage).

In order to compare purchasing a loosely coupled highly integrated system to integrating a system in-house, two different cases will be studied. Both cases assume that 200 systems will be purchased. Case I describes the costs involved with a loosely coupled highly integrated system, while Case II covers the costs associated with integrating a system in-house.

CASE I - Purchase a Loosely Coupled Highly Integrated System

Assume a Trimble Placer DR system is purchased. The Magnavox integrated dead-reckoning GPS system is not an off-the-shelf item. Also, the Trimble system uses a rate gyro that should perform better than an electronic compass. The dead-reckoning accuracy of the Trimble Placer DR is specified as 3-5% of distance traveled up to 10 kilometers. The cost for 200 Trimble Placer systems is calculated below.

200 units x $3183 per unit = $636.6K

Total Cost for 200 systems = $636.6K

Table 4-4. Summary of Pricing for Different Navigation Systems

GPS/Navigation System	Cost (12/92)
Trimble Placer DR - patch antenna (Qty 1-24)	$3,745
Trimble Placer DR - patch antenna (Qty 25-74)	$3,633
Trimble Placer DR - patch antenna (Qty 75-149)	$3,445
Trimble Placer DR - patch antenna (Qty 150-299)	$3,183
Trimble Placer DR - patch antenna (Qty 300-499)	$2,921
Trimble Placer DR - flange antenna (Qty 1-24)	$3,895
Trimble Placer 300 w/differential (Qty 1)	$1195
Magellan OEM Module (Qty 1-99) no antenna, w/differential	$445
Magellan OEM Module (Qty 400+) no antenna, w/differential	$345
Antenna for Magellan OEM Module (Avg Price)	$135
Magnavox GPS Engine (Qty 1-24) no antenna, w/differential	$1,870
Magnavox GPS Engine (Qty 25-49) no antenna, w/differential	$1,360
Magnavox GPS Engine (Qty 50-99) no antenna, w/differential	$1,150
Magnavox GPS Engine (Qty 100-249) no antenna, w/differential	$1,000
Magnavox GPS Engine (Qty 250-499) no antenna, w/differential	$950
Antenna for Magnavox GPS Engine (Qty 1)	$325- $600
Rockwell Nav Core V (Qty 100) not differential capable	$500

CASE II - Integrate a Loosely Integrated System

Assume, as calculated below, that the least expensive differential GPS system is purchased—the Magellan OEM Module. Then determine how much money would be available for development of the dead-reckoning system.

Receiver cost	200 units x $445 per unit	= $ 89K
Antenna cost	200 units x $135 per unit	= $ 27K
Total GPS cost		= $116K

Thus, the following are the funds left to develop dead-reckoning system:

$$\$636.6K - \$116K = \$520.6K.$$

And then, assuming one-man year of effort (0.5 for odometry, 0.5 for gyro system) at $150K, the following funds are left for hardware:

$$\$520.6K - \$150K = \$370.6K$$

Thus, the cost per vehicle for gyro hardware is calculated below:

$$\$370.6K/200 \text{ units} = \$1,853$$

These calculations assume that existing computing power could be used for the GPS dead-reckoning data fusion.

Although this is a rough cost comparison, because of the large number of systems involved, integrating the navigation system in-house might be justified by the economics alone. Other reasons might include contributing to Sandia's Intelligent Vehicle Highway Systems (IVHS) tech base as well as keeping sensitive information in-house. In summary, a decision has to be made whether to purchase a fully integrated navigation system with a newer GPS receiver or to integrate a GPS receiver with a dead-reckoning system developed by Sandia or a contractor. The expertise for developing the dead-reckoning system is available at Sandia in Departments 9616 (odometry and vehicle modeling) and 2334 (gyros and inertial navigation).

5.0 SUMMARY AND CONCLUSIONS

In summary, this report analyzes the performance of the current GPS receiver compared to newer, less expensive models and makes recommendations on possible options for upgrading the current system. The test fixture and test methodology used to perform both static and dynamic tests are documented. Static tests were used to determine the static position accuracy of the different receivers. Of the receivers tested, the Magnavox GPS Engine exhibited superior static position accuracy. All of the other receivers demonstrated similar, but less accurate static position accuracy. Dynamic tests were used to determine how well the various receivers handle intermittent obstructions as well as to determine the receiver/antenna sensitivity. Because of different DOP limits, exact comparisons of the different receivers was difficult. However, taking into account DOP criterions and the navigation mode data, all of the newer receivers performed satisfactorily. The performance of the newer receivers was far superior to the current Magnavox 6400 system being used. Based on the similar performance of the newer receivers in the dynamic tests, other integration issues like price, software flexibility, technical support, size, power, and differential capability are probably the most important factors to consider when choosing a GPS receiver.

The current navigation system uses dead reckoning (compass and odometry) and TRANSIT to supplement GPS. Because of the high value of the assets, a redundant navigation system is required. If the assumption is made that GPS is going to be operational 100 percent of the time for the long term, then the design of the current navigation system is greatly simplified. Assuming full GPS coverage, a redundant navigation system must be designed that can handle short periods of time when GPS is not available because of obstructions. The dead-reckoning system only needs to be accurate for up to several minutes, which makes the design of the dead-reckoning system much easier (and less expensive).

The main integration issue is the level of coupling between the dead-reckoning system and the GPS receiver. A tightly coupled GPS navigation system that combines GPS data with inertial and dead-reckoning sensors in a single Kalman filter would offer optimal performance. However, the cost of developing such a system would be fairly high (this approach should be taken if the 'best possible navigation system' is desired). A more economical approach would be to purchase a completely integrated GPS dead-reckoning system (like the Trimble Placer DR) or to develop the dead-reckoning system in-house and purchase a differential capable GPS receiver. Both options have advantages and disadvantages. These are summarized in Table 5-1.

A rough cost comparison of the two options in Table 5-1 was conducted in the previous section. The cost of integrating the system at Sandia might be justified on economics alone if enough units are purchased. Other reasons for doing the dead-reckoning navigation in-house include keeping the navigation tech base for IVHS related projects. Once a decision has been made whether to purchase an integrated system or to develop the dead-

Table 5-1. Comparison of Integration Options

Purchase a Loosely Coupled Highly Integrated System	Integrate a Loosely Integrated System
Advantages	Advantages
• Complete navigation system is an off-the-shelf item.	• Sandia is not locked into a particular manufacturer for the GPS system.
• Simplified maintenance because off-the-shelf system.	• GPS receivers can be updated as new technologies become available without affecting the dead-reckoning system.
	• Sandia maintains vehicle navigation tech base that can be applied to IVHS research.
Disadvantages	Disadvantages
• The number of off-the-shelf integrated systems is limited. These include the Trimble Placer DR and a semi-custom Magnavox system.	• Sandia has to develop the dead-reckoning system (might not really be a disadvantage).
• Sandia will be locked into a particular manufacturer for the complete navigation system.	

reckoning system in-house, no major technical hurdles exist in implementing either system. Therefore, a decision has to be made, based on the data presented in this paper as well as the goals of the program, as to which option is most desirable.

6.0 REFERENCES

Brown, R.G., and P.Y.C. Hwang. 1992. *Introduction to Random Signals and Applied Kalman Filtering.* 2nd ed. New York, NY: John Wiley and Sons.

GPS Report. November 5, 1992. Potomac, MD: Phillips Business Information.

Institute of Navigation. 1992. Class notes from "Introduction to GPS/INS Integration," *Institute of Navigation GPS-92 Conference, Tutorials, Albuquerque, NM, September 14-15, 1992.* Arlington, VA: Navtech Seminars, Inc.

APPENDIX A: TEST FIXTURE HARDWARE - CABLE PINOUTS

Developed by:
Raymond H. Byrne
Advanced Vehicle Development Department, 9616
Sandia National Laboratories
Albuquerque, NM 87185

APPENDIX A: TEST FIXTURE HARDWARE - CABLE PINOUTS

CABLE PINOUTS

Connector Assignments

Magellan	ZT 8901 J5 (COM1)
Magnavox Engine	ZT 8901 J4 (COM2)
Rockwell NavCore V	ZT 8841 J1
Magnavox 6400	ZT 8841 J2
Trimble Placer	ZT 8841 J4

Magellan

Signal	DB-25 (Magellan)	14 Pin (ZT 8901)
TX	3	3
RX	2	5
GND	7	13

Rockwell Nav Core V

Signal	DB-9 (Rockwell Unit)	14 Pin (ZT 8841)
TX	2	5
RX	3	3
GND	5	13

Magnavox 6400

Signal	DB-25 (Magnavox 6400)	10 Pin (ZT 8841)
GND	1	8
GND	7	7
TX(A)	14	14
TX(B)	2	13
RX(A)	15	1
RX(B)	3	2

Trimble Placer

Signal	DB-25 (Trimble Placer)	14 Pin (ZT 8841)
TX	3	5
RX	2	3
RTS	5	9
CTS	4	7
GND	22	13

Magnavox GPS Engine Serial Interface
- TTL to RS-232 conversion done with a Maxim MAX235 chip
- A LM340-5 chip was used to generate +5V from the battery supply
- A LM317 chip was used to generate +7V from the battery supply

APPENDIX B.1: SERIAL DATA FORMATS - MAGELLAN OEM MODULE

Excerpts (pages 3-1, 3-2, 4-1, and 4-2) from the 1992
User Guide for the Magellan OEM GPS Module.

Reprinted with permission from:
Magellan Systems Corporation
960 Overland Court
San Dimas, CA 91773

APPENDIX B.1: SERIAL DATA FORMATS - MAGELLAN OEM MODULE

CHAPTER 3
OEM MESSAGE DEFINITIONS

PORT CONFIGURATIONS

Baud Rate options: 1200, 2400, 4800, 9600 baud

Data Bits: 8

Parity: None

Stop Bits: 1

The baud rate on the OEM is set by jumpers on the PCB. (See Default PCB Configuration in Chapter 6.) Once in operation, firmware protocol will allow the baud rate to be changed. The change will be retained with memory backup, but if all power is removed the baud rate resets to the default as set by the jumpers.

MESSAGE FORMATS

Both binary and ASCII formats are provided for most of the sentences. The exceptions are listed below:

Binary only: J00 – Differential corrections
T01 – Almanac data
T02 – Ephemeris data

ASCII only: PMGLI – Data flow control

Binary

The BINARY sentence starts with "$$" (two HEX 24) followed by a one-byte sentence identifier (from A to Z in ASCII) and a one-byte binary subindex (00 to FF HEX), followed by the binary data field. All of the binary data are in integers with defined precision. The sentence is terminated with a line feed (HEX 0A). The checksum of the sentence is one byte before the line feed, and is required in binary format. The checksum is calculated by XORing the 8 binary data bits of each byte in the sentence with each other, starting with the first byte after the "$$" and ending with the first byte before the checksum. For example:

```
$$R100000000CL
  xxxxxxxxxx
```

("x" denotes data field to be XORed, "C" denotes checksum, "L" denotes line feed)

ASCII

The ASCII sentences conform to the NMEA (National Marine Electronics Association) 0183 software protocol. Two types of sentences are used. Where the 0183 protocol has a pre-defined sentence for specific data, this sentence is used. This includes GPGGA, GPZDA, GPVTG, GPBOD, GPBWC, GPHDM, GPHDT, GPAPA, and so forth. For information that does not have a predefined sentence, we used the method defined in the protocol for designing our own proprietary sentences so they will be compatible with the standard sentences. The headers

for proprietary sentences use 'P' for proprietary, 'MGL' for Magellan as manufacturer's identification, and a primary index of 'A' through 'Z'. The first data field is a subindex, which has a range of from 00 to 99. The checksum is the 2-byte hexadecimal ASCII of the binary byte, and is calculated by XORing each successive byte in the sentence, between the '$' and the '*'.

All data output sentences have the checksum calculated. The board's firmware does not require a checksum on ASCII input, but without one the integrity of the data cannot be guaranteed. If you do not want to send the checksum, omit the '*' and the two following bytes.

Example with checksum:

```
$PMGLI,00,U03,1,A,02*CKRL
 xxxxxxxxxxxxxxxxxxx
```

('x' denotes data field to be XORed, 'CK' denotes 2 bytes checksum, 'R' denotes carriage return - hex 0D, 'L' denotes line feed – hex 0A)

Example without checksum:

```
$PMGLI,00,U03,1,A,02RL
```

CHAPTER 4
OEM FIRMWARE PROTOCOL

OUTPUT MESSAGES

Output messages are from the module to the host. Also, the ASCII reference included in each message title is the one used in the PMGLI (data flow control sentence) for that message.

Time and Date — A00

BINARY

```
$$A0xxxxxxxCL
   1234567
```

1: 1 byte, subindex
2: 1 byte, UTC hour
3: 1 byte, UTC minute
4: 1 byte, UTC second
5: 1 byte, day
6: 1 byte, month
7: 2 bytes, year

ASCII

```
$GPZDA,xxxxxx,xx,xx,xxxx,*CKRL
       1      2  3  4
```

1: 6 bytes, UTC hhmmss
2: 2 bytes, day
3: 2 bytes, month
4: 4 bytes, year

Position and Altitude — B00

BINARY

```
$$B0xxxxxxxxxxxxxxxxCL
   12   3   4   5
```

1: 1 byte, subindex
2: 4 bytes, timetag in seconds, offset from the beginning of the GPS week (00:00 Sunday GMT)
3: 4 bytes, latitude in 10^{-7} degree (two's complement)
4: 4 bytes, longitude in 10^{-7} degree (two's complement)
5: 4 bytes, altitude in 0.01 meter/feet

ASCII

```
$GPGGA,xxxxxx,xxxx.xx,N,xxxxx.xx,W,x,x,xxx,xxx,M,xxxx,M*CKRL
       1      2       3 4         5 6 7 8   9  10 11   12
```

1: 6 bytes, UTC timetag of position fix (hhmmss)
2: 7 bytes, GPS latitude (DDMM.HH where D = degrees, M = minutes, H = hundredths of minutes)
3: 1 byte, latitude N or S
4: 8 bytes, GPS longitude (DDDMM.HH)
5: 1 byte, longitude E or W
6: 1 byte, GPS availability
   0 = GPS not available (last fix more than 10 seconds ago)
   1 = GPS available
7: 1 byte, number of satellites being used (3 or 4)
8: 3 bytes, HDOP (recalculated every 3 minutes or with every fix if PDOP > 8; see message G00 for a brief explanation of DOP's)
9-10: 3 bytes, antenna height, meters/feet
   (height above sea level)
11-12: 4 bytes, geoidal height, meters/feet
   (difference between antenna height and geoidal height in the WGS84 map datum)

Position Only — B01

BINARY

Same as $$B0

ASCII

```
$GPGLL,xxxx.xx,N,xxxxx.xx,W*CKRL
       1       2 3         4
```

1: 7 bytes, last fix latitude (DDMM.HH)
2: 1 byte, latitude N or S
3: 8 bytes, last fix longitude (DDDMM.HH)
4: 1 byte, longitude E or W

Extended Altitude — B02

BINARY

```
$$B2xxxxCL
   12
```

1: 1 byte, subindex
2: 4 bytes, altitude in 0.01 meters/feet

APPENDIX B.2: SERIAL DATA FORMATS - MAGNOVOX GPS ENGINE

Excerpts (pages A-1 to A-4) from the 1991
Magnovox GPS Engine Integration Guide.

Reprinted with permission from:
Magnavox Advanced Products and Systems Company
2829 Maricopa Street
Torrance, CA 90503

APPENDIX B.2: SERIAL DATA FORMATS - MAGNAVOX GPS ENGINE

CONTROL PORT RECORDS

NOTE

The structure of the control port records is based on the NMEA-0183 Standard for Interfacing Marine Electronics Navigation Devices (Version 1.5). For details beyond those provided in this appendix, refer to the Standard document.

Reserved characters are used to indicate the beginning and the end of records in the data stream, and to delimit data fields within a record. Only printable ASCII characters (Hex 20 through 7F) may be used in the records. Table A-1 lists the reserved characters and defines their usage. Figure A-1 illustrates the general Magnavox proprietary NMEA record format

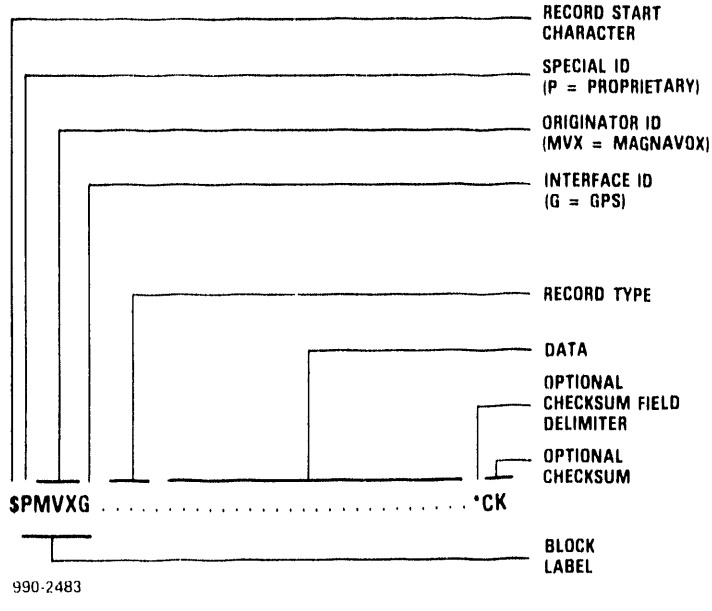


Figure A-1. Magnavox Proprietary NMEA Record Format

CONTROL PORT RECORDS

Table A-1. NMEA Record Reserved Characters

Character	Hex Value	Usage
$	24	Start of Record Identifier
{CR}{LF}	OD OA	End of Record Identifier
,	2C	Record Field Delimiter
*	2A	Optional Checksum Field Delimiter

Following the start character ($), the first five characters constitute the block label of the record. For the Magnavox proprietary records, this label is always PMVXG. The next field after the block label is the record type consisting of the three decimal digits.

The data, delimited by commas, follows the record type. The data is defined for each record in Tables A-2 through A-37. Note that the GPS Engine™ uses a free-format parsing algorithm so, you need not send the exact number of characters shown in the examples. You will need to use the commas to determine how many bytes of data need to be retrieved.

The notation *CK shown in the tables symbolically indicates the optional checksum in the examples. The checksum is computed by exclusive-ORing all of the bytes between the $ and the * characters. The $, * and the checksum are not included in the checksum computation.

*Table A-2. Record Type **$PMVXG,000***

Function: Initialization/Mode Control - Part A. Initializes the time, position and antenna height of the GPS Engine™ .

```
          1  2  3    4         5              6 7              8 9       A B
          |  |  |    |         |              | |              | |       | |
Format: DD,MM,YYYY,HHMMSS,DDMM.MMMM,N,DDDMM.MMMM,W,HHHH.H,X*CK
```

where:

Field	Field Contents		Default	Range
1: DD	Day of month	-	1 - 31	
2: MM	Month of year	-	1 - 12	
3: YYYY	Year	1992	1991 - 9999	
4: HHMMSS	GMT	-	235959	
5: DDMM.MMMM	Latitude (degrees, minutes)	00.000	0 - 89.9999	
6: N	North (S = south)	N	N or S	
7: DDDMM.MMMM	Longitude (degrees, minutes)	000.000	0 - 179.999	
8: W	West (E = east)	E	E or W	
9: HHHHH.H	Altitude (height above Mean Sea Level) in meters	00.0	-	
A: X	Not used	-	-	
B: *CK	Delimiter and checksum	-	-	

EXAMPLE MESSAGE:

$PMVXG,000,04,04,1991,181115,3350.5287,N,11820.2131,W,42.7 ,*75

CONTROL PORT INPUT RECORDS

Table A-3. Record Type ***$PMVXG,001***

Function: Initialization/Mode Control - Part B. Specifies various navigation parameters: Altitude aiding, acceleration, maximum DOP limits, and satellite elevation limits.

```
        1 2 3   4 5    6    7 8 9       A
        | | |   | |    |    | | |       |
Format: X,X,X.XX,X,XXXX,XXXX,XX,X,HHHMM*CK
```

where:

Field	Field Contents	Default	Range
1: X	Constrain Altitude: 0 = Never constrained. 3-D nav only when 4 sats available. Determine lat, lon, time, altitude. 1 = Auto. 3-D nav when 4 sats available, and determines lat, lon, time, altitude. Switches to 2-D nav when only 3 sats available, and determines lat, lon, time using manually entered altitude. 2 = Always constrained. 2-D nav when 3 sats available. Determine lat, lon, time, using manually entered altitude. 3 = Coast. 3-D nav when 4 sats available, and determines lat, lon, time, altitude. Switches to 2-D nav when only 3 sats available, and determines lat, lon, time using last calculated GPS altitude.	3	0 - 3
2: X	Not used	-	-
3: X.XX	Horizontal acceleration factor (m/sec**2)	1.0	0.50 - 10.0
4: X	Not used	-	-
5: XXXX	VDOP limit (3-D nav suspended when limit is exceeded)	10	1 - 9999
6: XXXX	HDOP limit (2-D nav suspended when limit is exceeded)	10	1 - 9999
7: XX	Elevation limit (degrees) (sat not used if elevation angle is below limit)	5	0 - 90
8: X	Time output mode: U = UTC L = Local time	U	U, L
9: HHHMM	Local time offset (±) from GMT	-	± 0 - 2359
A: *CK	Delimiter and checksum	-	-

EXAMPLE MESSAGE:

$PMVXG,001,1,,.5,0,5,10,5,U,-0800*0E

APPENDIX B.3: SERIAL DATA FORMATS - ROCKWELL NAVCORE V

Excerpts (pages 4-1 to 4-15 and 4-21 to 4-23) from the March 1992
NavCore® V - Rockwell Global Positioning System
Receiver Engine Designer's Guide.

APPENDIX B.3: SERIAL DATA FORMATS - ROCKWELL NAVCORE V

4. SERIAL DATA INTERFACE

4.1 Serial Data Protocol

4.1.1 Data Bit Rate

The serial data bit rate is determined by the state of pin J2-14 of the NavCore® V at application of "Prime" power. If the pin is held high, the data will be transmitted and received at 9600 bps. If the pin is held low, the NavCore V will receive data at 19,200 bps and will transmit data at 76,800 bps. Once the data rate has been established, it will remain unchanged throughout the current power cycle. (Normal operation is 9600 bps.)

4.1.2 Data Encoding and Format

Each byte of information is transmitted through the serial port as a sequence of 11 bits. The serial data port will be configured for odd parity, so that each byte of serial data is encoded into 11 bits for serial transmission - one start bit, eight data bits, one odd parity bit, and one stop bit (Figure 4-1). The data bits are ordered from least significant bit to most significant bit. For each data word, the least significant byte is transmitted first, followed by the most significant byte. Integer and floating point data types consisting of more than one word are transmitted from the lowest numbered word to the highest numbered word as indicated in the data type format descriptions below.

DATA FORMATS

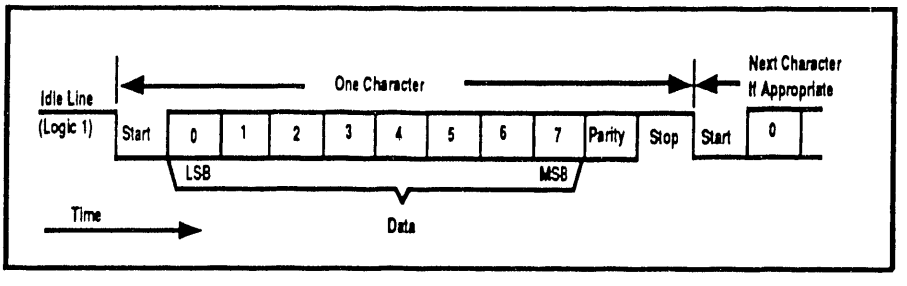


Figure 4-1. Serial Data Timing Diagram.

Single Precision Integer Format

Single precision integer data types are allocated one 16-bit word. The most significant bit is the sign bit. Negative numbers are represented in two's complement form, where the sign bit is one (Figure 4-2).

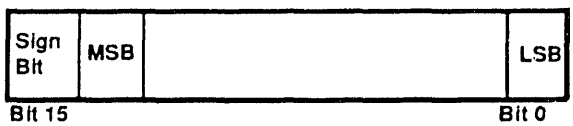


Figure 4-2. Single Precision Integer Format.

Double Precision Integer Format

Double precision integer data types are allocated two 16-bit words. The most significant bit is the sign bit. Negative numbers are represented in two's complement form, where the sign bit is one (Figure 4-3).

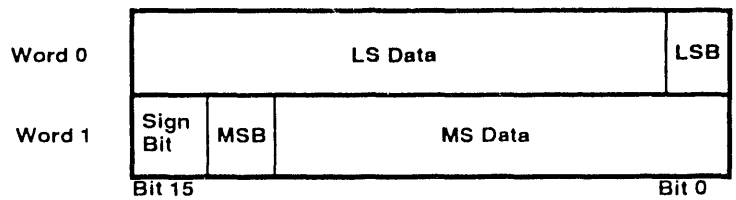


Figure 4-3. Double Precision Integer Format.

Floating Point Format

Floating point data types are allocated two 16-bit words (Figure 4-4). The floating point representation consists of a mantissa, exponent, and sign bit, where the absolute value of the floating point number equals the mantissa multiplied by $2^{Exponent}$, and where a sign bit of 0 indicates a positive number.

The mantissa is normalized so that its most significant bit (the 2^{-1} place) is a 1, with the exception of the value 0.0, in which case all bits in the words are 0. Thus, the most significant bit is hidden, and the mantissa has one extra bit of precision more than its actual number of allocated bits. The mantissa field width is 23 bits for floating point data types, which allows 24 bits of precision. For all floating point types, the exponent field is eight bits wide. Exponents are represented in excess 128 notation. Since all bits are 0 for the value 0.0, floating point representations with a zero exponent and a non-zero mantissa or sign bit are not allowed.

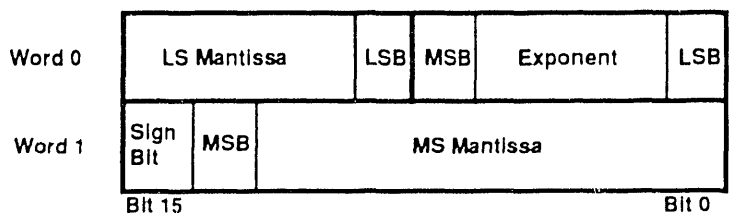


Figure 4-4. Floating Point Format.

Extended Floating Point Format

Extended floating point data types are allocated three 16-bit words (Figure 4-5). The extended floating point representation consists of a mantissa, exponent, and sign bit, where the absolute value of the extended floating point number equals the mantissa multiplied by $2^{Exponent}$, and where a sign bit of 0 indicates a positive number.

The mantissa is normalized so that its most significant bit (the 2^{-1} place) is a 1, with the exception of the value 0.0, in which case all bits in the words are 0. Thus, the most significant bit is hidden, and the mantissa has one extra bit of precision more than its actual number of allocated bits. The mantissa field width is 39 bits for extended floating point, which allows 40 bits of precision. For all floating point types, the exponent field is eight bits wide. Exponents are represented in excess 128 notation. Since all bits are 0 for the value 0.0, extended floating point representations with a zero exponent and a non-zero mantissa or sign bit are not allowed.

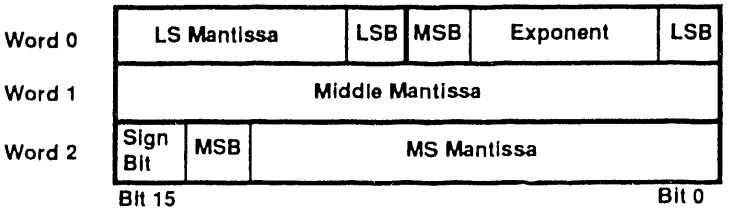


Figure 4-5. Extended Floating Point Format.

Further discussion of the conversion routines "from" and "to" the NavCore V formats in IEEE 754 format is included in Section 11.1.

4.1.3 Message Structure

Each message will consist of a message header, which identifies the message, and message data. A Checksum word will accompany each component for error detection. Figure 4-6 shows a simplified message structure illustrating where the components are placed to form a message.

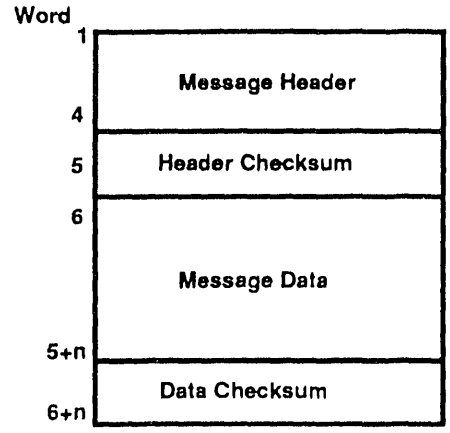


Figure 4-6. Message Structure.

The message will consist of two parts: a header and a data portion. The organization for a word will be such that the least significant byte arrives first. The header portion will have a fixed length of five 16-bit words, consisting of a DEL-SOH byte pair, message ID, Word Count, flags, and the Header Checksum. The length of the data portion will be defined in the header and, if present, will consist of *n* data words and a Data Checksum, where *n* will be less than or equal to 100 (Figure 4-7).

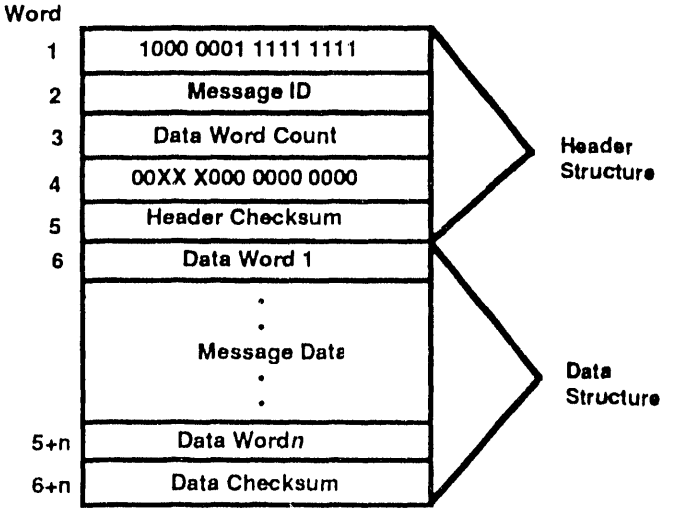


Figure 4-7. Complete Message Structure.

4.1.3.1 Header Structure

All messages have a message header that consists of five words as follows: a word indicating the start of the message, equal to 129 decimal in the high order byte and 255 decimal in the low order byte; an integer value equal to the message ID; an integer value specifying the number of words of data in the message; and a word containing protocol related flags. Examples are given for headers "to" NavCore V (Figure 4-8) and "from" NavCore V (Figure 4-9).

Word	
1	1000 0001 1111 1111
2	Message ID
3	Data Word Count
4	000R 0000 0000 0000
5	Header Checksum

Figure 4-8. Header for Message "To" NavCore V.

(R) = Acknowledge Request

Word	
1	1000 0001 1111 1111
2	Message ID
3	Data Word Count
4	0000 AN00 0000 0000
5	Header Checksum

Figure 4-9. Header for Message "From" NavCore V.

(A) = Acknowledgment Flag
(N) = Negative Acknowledgment Flag

4.1.3.1.1 Header Format Word 1 of 5 (Start of Message)

Word		SOH / DEL	
1	*	1000 0001 / 1111 1111	*
2		Message ID	
3		0000 0000 0000 0000	
4		000R AN00 0000 0000	
5		Header Checksum	

Figure 4-10. Word 1 Start of Message.

The first word of any header indicates the start of a message (Figure 4-10). This word is divided into two bytes. The higher order byte (SOH) is equal to 129 decimal. The lower order byte (DEL) is equal to 255 decimal.

DEL = A constant byte valued at 255 decimal (all ones). DEL forms the first eight bits of Word 1 of the header.
SOH = A constant byte valued at 129 decimal. SOH forms the second eight bits of Word 1 of the header.

4.1.3.1.2 Header Format Word 2 of 5 (Message ID)

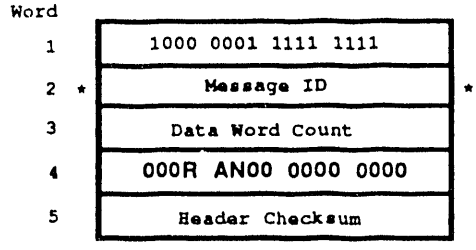


Figure 4-11. Word 2 Message ID.

The message ID (Figure 4-11) is a binary representation of the "Message ID Number" located in Figures 4-12 and 4-13.

For example, ID #203 is "0000 0000 1100 1011."

MESSAGE ID = A 16-bit field which uniquely defines the message type and origin. This field comprises Word 2 of the header and is a right justified integer.

Message ID	Message Description	Rate	Source
101	Built-In Test Results	On request	GPSRE
102	Visible Satellite Azimuth(s) and Elevation(s)	2 minute intervals	GPSRE
103	NavCore V Navigation Solution and Status (Time Mark Solution)	1 Hz	GPSRE
104	Almanac Data	On request (max. 1 Hz)	GPSRE
105	Age of Almanac	On request	GPSRE

Figure 4-12. Status (Output) Blocks to the Application Processor (AP).
*GPSRE = GPS Receiver Engine.

Message ID	Message Description	Rate	Source
201	Position, Velocity, and Time Initialization	As required (max. 0.1 Hz)	AP
202	Built-In Test Command	As required (max. 0.1 Hz)	AP
203	Altitude Hold Enable Command	As required (max. 1 Hz)	AP
204	Amended Altitude Hold Command	As required (max. 1 Hz)	AP
205	Externally Supplied Almanac Data	As required (max. 1 Hz)	AP
206	Almanac Data Request	As required (max. 1 Hz)	AP
207	Age of Almanac Request	As required (max. 1 Hz)	AP
208	Manual Satellite Selection	As required (max. 0.1 Hz)	AP

Figure 4-13. Command (Input) Blocks from the Application Processor (AP).

Figures 4-12 and 4-13 are described in more detail in Sections 4.2 and 4.3.

4.1.3.1.3 Header Format Word 3 of 4 (Data Word Count)

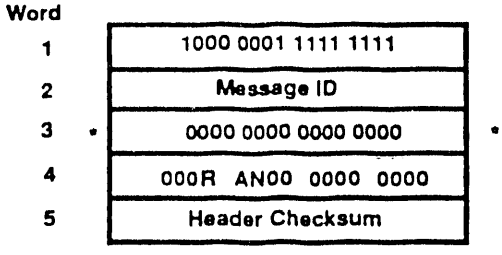


Figure 4-14. Word 3 Data Word Count.

WORD COUNT = A 16-bit field which specifies the number of words contained in the data field (Figure 4-14). The Word Count does not include the Data Checksum word. The value zero is used to indicate a message containing no data, in which case the message is a header-only message and will terminate with the header Checksum word. The word is a right justified integer.

4.1.3.1.4 Header Format Word 4 of 5 (Flags)

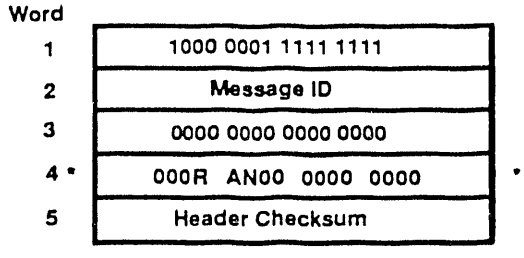


Figure 4-15. Flag Bits.

FLAGS = A 16-bit field allocated to protocol and message related flags (Figure 4-15). These flag bits provide for the control of the protocol operation. The NavCore V only requires bits 10, 11, 12 for message acknowledgement. These messages are described in more detail in the following sections.

4.1.3.1.5 Header Format Word 5 of 5 (Header Checksum)

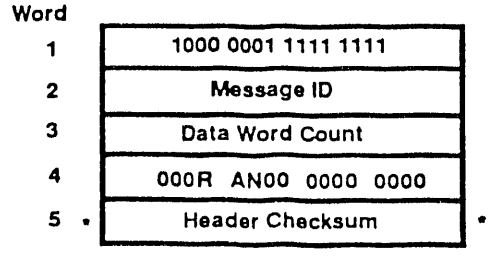


Figure 4-16. Word 5 Header Checksum.

HEADER CHECKSUM = A 16 bit Checksum is used to validate the header portion of the message (Figure 4-16). It is computed by summing (modulo 2**16) all words (including the word containing DEL and SOH) contained in the header and then complementing that sum. The computation of Header Checksum may be expressed mathematically as:

$$\text{Header Checksum} = - \text{ Mod } 2^{16} \sum_{I=1}^{4} \text{Word(I)}$$

where:

(a) Unary negation is computed as the two's complement of some 16-bit data word.
(b) Mod 2^{16} indicates the least 16 bits of an arithmetic process. That is, carry bits from the bit position 16 are ignored.
(c) The summation is the algebraic binary sum of the words indicated by the subscript (I).

4.1.3.2 Data Structure

The data structure (Figure 4-17) consists of formatted message data followed by a data Checksum. Data formats are unique for each message, and are shown in Section 4.2 for status messages and Section 4.3 for command messages. Once the message data is formatted, a Data Checksum is calculated as described in Section 4.1.3.2.2.

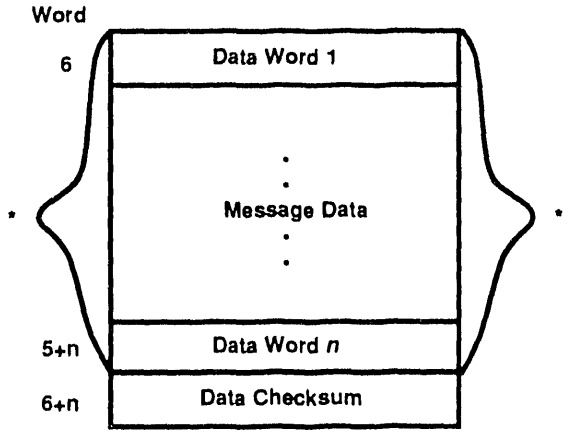


Figure 4-17. Data Structure.

4.1.3.2.1 Data Word Format (Words 6 through 5+n)

DATA = The message data field. This field is totally transparent to the protocol and has no restrictions on bit patterns or character groupings. The field will contain the number of words specified in the Word Count field. When the Word Count field is zero, the data field does not exist. Data fields are formatted as shown in Figure 4-17. Messages transmitted and received by the NavCore V have a maximum length of 106 words, so that up to 100 words of data, plus the header and Checksums, can be transmitted in a single message.

4.1.3.2.2 Data Checksum Format (Word 6+n)

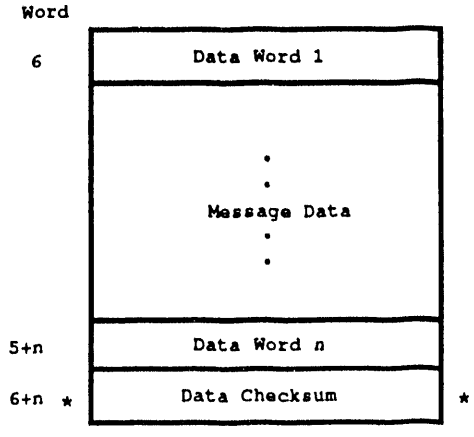


Figure 4-18. Data Checksum.

DATA CHECKSUM = A 16-bit Checksum is used to validate the data portion of the message. It is transmitted as the last word of any message containing data (Figure 4-18). When the Word Count field is zero, the Data Checksum does not exist. It is computed by summing (modulo 2**16) all words in the data portion of the message and then complementing that sum. The mathematical expression for the Data Checksum is:

$$\text{Data Checksum} = - \text{ Mod } 2^{16} \sum_{i=6}^{5+N} \text{Word}(i)$$

where:

(a) Unary negation is computed as the two's complement of some 16-bit data word.
(b) Mod 2^{16} indicates the least 16 bits of an arithmetic process. That is, carry bits from the bit position 16 are ignored.
(c) The summation is the algebraic binary sum of the words indicated by the subscript (i).

4.1.4 Message Acknowledgment

The NavCore V software will respond to requests from the OEM Application Processor to acknowledge received messages. A request for message acknowledgment will only be recognized if the Acknowledge Request flag (Figure 4-19) is asserted in the header of the received message, and the header was received without errors. Upon receiving an Acknowledge Request, an Acknowledgment Message (Figure 4-20) or Negative Acknowledgment Message (Figure 4-21) will be transmitted back to the OEM Application Processor within 250 milliseconds.* Serial data messages transmitted to the OEM Application Processor will not require acknowledgment.

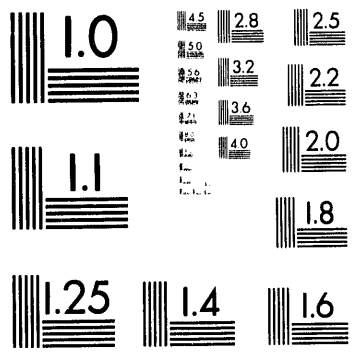
***400 milliseconds at the lower, 9600 bps serial rate.**

If an Acknowledge Request was sent and neither an Acknowledgment message nor Negative Acknowledgment message was received after 250 mS*, then the OEM Application Processor must assume that a transmission error occurred. And, since messages received in error by the NavCore V are discarded, the Application Processor must resend the message if it is still desired that the message take effect.

A new message with Acknowledge Request should not be sent until either an Acknowledgment Message or Negative Acknowledgment Message has been received in response to the previous message with Acknowledge Request, or until 250 mS* have elapsed.

The Application Processor must be prepared to receive a periodic message (such as Message 102 or 103) before receiving the Acknowledgment or Negative Acknowledgment to a message with Acknowledge Request.

***400 milliseconds at the lower, 9600 bps serial rate.**


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

2 of 3

4.1.4.1 Acknowledge Request ("To" NavCore V)

1000 0001 1111 1111
Message ID
Data Word Count
000R 0000 0000 0000
Header Checksum
Data Word 1
• • Message Data • •
Data Word *n*
Data Checksum

Figure 4-19. Acknowledge Request Flag.

(R) = ACKNOWLEDGE REQUEST FLAG (BIT 12) (Figure 4-19). Indicates that the message block bein transmitted requires acknowledgment by the NavCore V. This bit (BIT 12) is set to a logical 1 for a request

4.1.4.2 Acknowledge Message Description ("From" NavCore V)

1000 0001 1111 1111
Message ID
0000 0000 0000 0000
0000 A000 0000 0000
Header Checksum

Figure 4-20. Bit 11 Acknowledge/Flag.

(A) = ACKNOWLEDGE (Bit 11) (Figure 4-20).

Acknowledge (ACK) is transmitted only in response to a message, requiring acknowledgment, which wa received without parity or Checksum errors.

The ACK message is constructed in the following manner:

(a) Set Flag bit 12 to 0.
(b) Set Flag bit 11 (ACKNOWLEDGE FLAG) to 1.
(c) Set Word Count (Word 3) to 0.
(d) Recompute the Header Checksum.
(e) Transmit the new header (No data words are transmitted for an ACK.)

4.1.4.3 Negative Acknowledge Message Descriptions ("From" NavCore V)

1000 0001 1111 1111
Message ID
Data Word Count
0000 0N00 0000 0000
Header Checksum

Figure 4-21. Bit 10 Negative Acknowledgment Flag.

(N) = NEGATIVE ACKNOWLEDGE (Bit 10) (Figure 4-21).
Negative Acknowledge (NAK) is transmitted only in response to a message requiring acknowledgment whose header was received correctly, but whose data portion was received with a parity or Checksum error. A NAK message also indicates that none of the data in the message requiring acknowledgment was accepted by the NavCore V. Header-only messages requiring acknowledgment that are received with errors cannot be negatively acknowledged with a Negative Acknowledge Message. The absence of an ACK message serves as Negative Acknowledgment for header-only messages.

The message is constructed in the following manner:
(a) Set Flag bit 12 to 0.
(b) Set Flag bit 10 (NEGATIVE ACKNOWLEDGE FLAG) to 1.
(c) Set Word Count (Word 3) to 0.
(d) Recompute the Header Checksum.
(e) Transmit the new header (No data words are transmitted with a NAK.)

4.1.5 Error Handling

Serial data messages received by the NavCore V are validated by detecting parity and framing errors in received serial data byte transmissions, detecting erroneous data bit transitions, and verifying the Header and Data Checksum words according to the Checksum calculations of the message protocol. Received messages that contain errors, as well as messages with unassigned message IDs, are discarded. Received messages that have errors in the data portion, but whose headers were received correctly, may be negatively acknowledged.

4.2 NavCore V to OEM Application Processor Messages

In the following paragraphs, abbreviations for data types are as follows:

I	Integer
DI	Double Precision Integer
FP	Floating Point
EFP	Extended Floating Point
B	Boolean

Boolean data types are represented as integers, where a value of zero indicates false and a value of one indicates true. All bits that are designated as spare or reserved within the bit descriptions of binary data will be set to logic zero by the NavCore V. These may be used in the future so the OEM is advised NOT to rely on their being zero, but should mask these unused bits prior to processing the defined data bits.

4.2.3 NavCore V Series Position, Velocity, Time Solutions and Status (Time Mark Message) (Message ID 103)

This message contains time, position, and velocity data that is valid for the current Time Mark pulse. It also contains status information valid for the current Time Mark pulse, including error information, channel tracking status, signal-to-noise ratios, Altitude Hold status and Receiver Engine Mode information.

Message ID: 103

Rate: 1 Hz

Word Length: 64

Word No.	Name	Type	Units	Range	Resolution	
1-4	Message Header					
5	Header Checksum					
6-8	GPS Time of Week	EFP	sec	0..604,800	10^{-6} sec	
9	GPS Week Number	I	weeks	0..32,767		
10	UTC Validity	B				
11-13	UTC Time of Day	EFP	sec	0..86,400	10^{-6} sec	
14	UTC Day	I	days	1..31		
15	UTC Month	I	months	1..12		
16	UTC Year	I	year	1980..2079		
17-18	Set Time	DI	20 ms	$0..2^{31}$	20 ms	
19-21	Position X	EFP	meters	±9,000,000	10^{-4} m	Note 1
22-24	Position Y	EFP	meters	±9,000,000	10^{-4} m	Note 1
25-27	Position Z	EFP	meters	±9,000,000	10^{-4} m	Note 1
28-29	Velocity East	FP	meters/sec	±1,000	10^{-4} m/s	Note 2
30-31	Velocity North	FP	meters/sec	±1,000	10^{-4} m/s	Note 2
32-33	Velocity Up	FP	meters/sec	±1,000	10^{-4} m/s	Note 2
34-36	Latitude	EFP	radians	$\pm\pi/2$	10^{-11} rad	Note 3
37-39	Longitude	EFP	radians	$\pm\pi$	10^{-11} rad	Note 3
40-41	Altitude	FP	meters	±50.000	0.01 m	Note 3
42	Horizontal Dilution of Precision	I	0.01	1.0..327.67	0.01	
43	Vertical Dilution of Precision	I	0.01	1.0..327.67	0.01	
44	Position Dilution of Precision	I	0.01	1.0..327.67	0.01	
45	Time Dilution of Precision	I	0.01	1.0..327.67	0.01	

Word No.	Name	Type	Units	Range	Resolution	
46	Geometric Dilution of Precision	I	0.01	1.0..327.67	0.01	
47	Chan.1 Measurement State	Binary				Note 4
48	Channel 1 C/N_o	Binary				Note 5
49	Chan.2 Measurement State	Binary				Note 4
50	Channel 2 C/N_o	Binary				Note 5
51	Chan.3 Measurement State	Binary				Note 4
52	Channel 3 C/N_o	Binary				Note 5
53	Chan.4 Measurement State	Binary				Note 4
54	Channel 4 C/N_o	Binary				Note 5
55	Chan.5 Measurement State	Binary				Note 4
56	Channel 5 C/N_o	Binary				Note 5
57	Expected Horiz. Position Error	I	meters	0..32,767	1 meter	
58	Expected Vert. Position Error	I	meters	0..32,767	1 meter	
59	Expected Time Error	I	meters	0..32,767	1 meter	Note 6
60	Expected Horizontal Velocity Error	I	0.01 meters/sec	0..327.67	0.01 m/s	
61-62	Reserved					
63	Receiver Engine Status	Binary				Note 7
64	Data Checksum					

Note 1:
WGS-84 Earth-Centered/Earth-Fixed coordinate.

Note 2:
Local tangent plane coordinate referenced to the WGS-84 ellipsoid, where velocities are referenced to true north.

Note 3:
Referenced to WGS-84 map datum.
North latitude is positive, South latitude is negative.
East longitude is positive, West longitude is negative.

Note 4:
Channel Measurement State Words are formatted as follows:

bits 0-5	Satellite PRN Code, integer range 1-32. A value of 0 indicates that no satellite is assigned to this channel.
bits 6-8	Spare
bits 9-11	Channel Activity 000 - Idle 001 - C/A code search 101 - C/A code tracking
bits 12-14	Channel ID, integer range 1-5.
bit 15	Spare

Note 5:
Signal-to-noise values are formatted as follows:

bits 0-7	Spare
bits 8-13	Integer value of C/N_o, range 0-63, in units of dBHz
bits 14-15	Spare

Note 6:
The expected time error can be converted to units of seconds by dividing by the speed of light.

Note 7:
The Receiver Engine Status word contains Boolean indicators and integer data fields formatted as follows:

bit 0	Altitude Accept - Logic 1 indicates entered or automatic Altitude Hold measurements will be accepted (if Altitude Hold is enabled) to produce a navigation solution.
bit 1	Altitude Used - Logic 1 indicates Altitude Hold measurements were used in the navigation solution.
bit 2	Altitude Hold Enabled.
bit 3	Navigation Mode Logic 1 indicates that the Receiver Engine is in Navigation Mode. Logic 0 indicates that the Engine is in Acquisition Mode.
bits 4-6	Integer value, range 0 to 5, indicating the number of satellite measurements being incorporated.
bit 7	Spare
bits 8-11	An integer value indicating the Figure of Merit, which is the estimated position error, where: 1 => less than 26 meters 2 => 26 to 50 meters 3 => 51 to 75 meters 4 => 76 to 100 meters 5 => 101 to 200 meters 6 => 201 to 500 meters 7 => 501 to 1000 meters 8 => 1001 to 5000 meters 9 => Greater than 5000 meters
bits 12-14	Integer value, range 0 to 5, indicating the channel designated as the utility channel, which is used to acquire and track secondary satellites and to collect satellite data while in Navigation Mode. A value of 0 indicates that no utility channel is currently assigned.
bit 15	Spare.

APPENDIX B.4: SERIAL DATA FORMATS - MAGNAVOX 6400

Excerpts (pages D-1 and D-3) from the 1988
MX 5400 GPS Satellite Navigation System - Technical Reference Manual

Reprinted with permission from:
Magnavox Advanced Products and Systems Company
2829 Maricopa Street
Torrance, CA 90503

APPENDIX B.4: SERIAL DATA FORMATS - MAGNAVOX 6400

EXT INTFC CONNECTOR PORT A (COMMUNICATION CHANNEL 1)
RAW DATA FORMATS

This appendix describes the available format options for the raw data transferred from the MX 5400 to a user-supplied data device via port A. Data communication is asynchronous, with the format and transfer rate selectable with KYBD function 3 (IO-PORT A), as described in Chapter 3. The modes selectable with KYBD function 3 relate to the data in this appendix as follows:

KEYBOARD FUNCTION PORT A OUTPUT OPTION	APPENDIX REFERENCE
NAV1	Data record 29 only
NAV	Data records 21 and 31
NAV+	Data records 21, 31, 903, 904 and all 400-series
ALL	All data records
DBUG	All data records plus test messages
RAW	Data blocks Q, P, and V
NONE	No data output
NMEA	Selectable records in NMEA-0183 format

The data records, as listed in table D-1, are in ASCII format. The RAW output data blocks, as listed in table D-2, are in binary format. The three raw data blocks are quality, position, and velocity. The quality data block is always transmitted once per second. The position and velocity blocks are also transmitted once per second, but only if the MX 5400 is in the navigation mode. A typical output from port A is shown in figure D-1. The NMEA record formats are described in Appendix E.

Table D-1. Data Record Formats (Continued)

Field Name	Units	Type	Byte Size	Range
POSITION RECORD (TYPE 29)				
Record ID	N/A	Integer	4	Always 29
Time of Position (UTC)	Seconds	Real	11	0-604,800
Latitude	Radians	Real	13	±Π/2
Longitude	Radians	Real	13	±Π
Height Above Ellipsoid	Meters	Real	13	0-99999.9
Sat. Selected	N/A	Integer	3	0-32
Sat. Selected	N/A	Integer	3	0-32
Sat. Selected	N/A	Integer	3	0-32
Sat. Selected	N/A	Integer	3	0-32
Ndop	N/A	Real	5	0-99.9
Edop	N/A	Real	5	0-99.9
Vdop	N/A	Real	5	0-99.9
Mode ID	--	--	2	--
Terminator (CR,LF)	--	--	2	--
Total Bytes			85	

Example Type 29 Position Record:

```
29  273196.875    1.49123125    -2.14523122    531.1
    12  3  9  11  1.4 9.1  11.3   D
```

NOTE: The last character D denotes DGPS mode. A blank denotes no DGPS.

Field Name	Units	Type	Byte Size	Range
VELOCITY RECORD (TYPE 31)				
Record ID	N/A	Integer	4	Always 31
Velocity North	Meters/Sec	Real	8	±100
Velocity East	Meters/Sec	Real	8	±100
Velocity Up	Meters/Sec	Real	8	±100
Local Frequency Offset	Meters/Sec(2)	Real	9	±100
Terminator (CR,LF)	--	--	2	--
Total Bytes			39	

Example Velocity Record:

```
30    0.043    0.016   -0.074    -1.035
```

Note (2):
To convert local frequency offset to parts per million, divide by C/1,000,000 (C/1,000,000 = 299.792458).

APPENDIX B.5: SERIAL DATA FORMATS - TRIMBLE PLACER

Excerpts (pages 1 to 4 of Appendix A,
Trimble ASCII Interface Protocol [TAIP] Version 1.01)
from the 1991
Placer ™ GPS Intallation and Operator's Manual

Reprinted with permission from:
Trimble Navigation
645 North Mary Avenue
Sunnyvale, CA 94088-3642

APPENDIX B.5: SERIAL DATA FORMATS - TRIMBLE PLACER

Trimble ASCII Interface Protocol (TAIP)
Version 1.01

1.0 General

Trimble ASCII Interface Protocol is a Trimble-specified digital communication interface, based on printable ASCII characters over a serial data link. It supports scheduled and polled responses. Messages may be scheduled for output at a user specified rate starting on a given epoch from top of the hour. For communication robustness, the protocol optionally supports checksum on all messages. It also provides the user with the option of tagging all messages with the unit's previously specified id. This greatly enhances the functional capability of the unit in a network environment. Additionally, given the printable ASCII format of all communication, TAIP is ideal for use with mobile data terminals, seven bit modems and portable computers. Although, sensors incorporating this protocol are shipped from the factory with a specific serial port setting, the port characteristics are fully programmable through TAIP messages.

2.0 Message Format

All communication is done using printable ASCII characters. The interface provides the means to configure the unit to output various sentences in response to query or on a scheduled basis. Each sentence has the following general format:

>*A**BB*{*C*}[;ID=*DDDD*][;**JJ*]<

where

>	start of new message,
A	message qualifier,
BB	a two character message ID,
C	data string,
DDDD	optional 4 character vehicle ID,
JJ	optional 2 character checksum,
<	the delimiting character.

Notation:

{x}	signifies that x can occur any number of times.
[x]	signifies that x may optionally occur once.

Start of new message

A '>' is used to specify the start of a new sentence.

Message Qualifier

A one character message qualifier is used to describe the action to be taken on the message. The following table lists the valid qualifiers:

Qualifier	Action
Q	Query for a single sentence.
R	Response to a query or scheduled report
F	Scheduled reporting frequency interval in seconds
S	Set command

Message ID

A unique two character message ID consisting of letters of alphabet is used to identify different type messages. Valid message IDs are presented in the tables of section 3.

Data String

The format and length of the data string are dictated by the message qualifier and the message ID. It can consist of any printable ASCII character with the exception of ('>', '<', and ';'). See section 3 for detailed descriptions of message formats. Most messages are length sensitive and unless otherwise specified, field separators including space are not used.

Vehicle ID

A vehicle ID may optionally be used in all the communications with the unit. Each vehicle may be assigned a four digit ID and be forced to use that in all correspondence. The default vehicle ID is '0000' and the default mode of operation is not to use vehicle ID.

CheckSum

The checksum field provides for an optional two digit hex checksum value, which is computed as XOR of all characters from the beginning of the sentence up to and including the '*'. If provided, the checksum is always the last element of the sentence before the message delimiter. The default mode of operation is to include checksum in sentences.

Message Delimiter

A '<' signifies end of a sentence and is used as message delimiter.

3.0 Message Formats

The following table lists all the messages currently supported:

Message ID	Message Name
CP	Compact Position Solution
DC	Differential Corrections
DD	Delta Differential Corrections
ID	Vehicle ID
PT	Port characteristic
PV	Position/Velocity Solution
RM	Reporting Mode
VR	Version Number

CP Compact Position Solution

Data String Format:

AAAAABBBCCCCDDDDEEEEFG

Item	# of Char	UNITS	Format
GPS Time of day	5	Sec	*AAAAA*
Latitude	7	Deg	*BBB.CCCC*
Longitude	8	Deg	*DDDD.EEEE*
Source	1	n/a	*F*
Possible Values:			
GPS 2D	0		
GPS 3D	1		
DGPS 2D	2		
DGPS 3D	3		
Age of Data Indicator	1	n/a	*G*
Possible Values:			
Fresh(< 10 sec)	2		
Old (>= 10 sec)	1		
Unknown	0		
Total	22		

Position is in latitude (positive north) and longitude (positive east) WGS-84. The GPS time of day is the time of fix rounded to the nearest second.

PV Position/Velocity Solution

Data String Format:

AAAAABBBCCCCDDDDEEEEFFFGGGHI

Item	# of Char	UNITS	Format
GPS Time of day	5	Sec	*AAAAA*
Latitude	8	Deg	*BBB.CCCCC*
Longitude	9	Deg	*DDDD.EEEEE*
Speed	3	MPH	*FFF*
Heading	3	Deg	*GGG*
Source	1	n/a	*H*
Possible Values:			
GPS 2D	0		
GPS 3D	1		
DGPS 2D	2		
DGPS 3D	3		
Age of Data Indicator	1	n/a	*I*
Possible Values:			
Fresh(< 10 sec)	2		
Old (>= 10 sec)	1		
Unknown	0		
Total	30		

Position is in latitude (positive north) and longitude (positive east) WGS-84. Heading is in degrees from True North increasing eastwardly. The GPS time of day is the time of fix rounded to the nearest second.

ID _____ Vehicle ID

Data String Format:

AAAA

Item	# of Char	UNITS	Format
Numeric vehicle ID	4	n/a	*AAAA*

Vehicles unique four digit ID. The default at start-up is '0000'.

RM _____ Reporting Mode

Data String Format:

[;ID_FLAG=*A*][;CS_FLAG=*B*][;EC_FLAG=*C*]

APPENDIX C.1: SOFTWARE LISTING - GPS CONVERSION PROGRAM

Developed by:
Frank Wunderlin*
Technadyne Engineering Consultants, Inc.
P.O. Box 1328
Albuquerque, NM 87192

* Work performed under Contract No. 87-2050 for the Advanced Vehicle Development Department (9616), Sandia National Laboratories.

APPENDIX C.1: SOFTWARE LISTING - GPS CONVERSION PROGRAM

Converts raw GPS data into latitude, longitude, and navigation mode columns.

```
/* ---------------------------------------------------------------------
*    FILE: GPS11.C
*
*    Developed by Frank Wunderlin
*
*    This file contains functions used to read and convert raw
*    GPS data files.
*
*    Functions included in this file are:
*
*    ConvertAfile( char *name )
*    ConvertBfile( char *name )
*    ConvertCfile( char *name )
*    ConvertDfile( char *name )
*    ConvertFfile( char *name )
*    ConvertToFloatingPt( unsigned char buff[] )
*    DisplayLineNum( int num )
*    MakeNames( char filetype, char rawname[], char outname[], char name[] )
*    NextDataset( int start, char *datasetname )
*    OutputData( FILE *fpptr, double latitude, double longitude, int Sats )
*    QueryAbort()
*    ReportDataError( char *name, int linenum )
*    ReportFileError( char *name )
*    Update_llerrs( int err, char fname[] )
*
*/
#include <stdio.h>
#include <conio.h>
#include <io.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <ctype.h>
#include <math.h>

union
   {
   unsigned char bd[8];
   int      iv;
   long int lv;
   float    fv;
   double   dv;
   } valu;

#define GPSdriverFile "FILES.GPS"
#define TTMINUTES_TO_FRACTION_DEG ( 1.0 / 600000.0 )
#define RADS_TO_DEGS  57.29577951
#define MAXERROR  10000
#define FALSE 0
#define TRUE 1

int ConvertAfile( char * );
int ConvertBfile( char * );
int ConvertCfile( char * );
int ConvertDfile( char * );
int ConvertFfile( char * );
double ConvertToFloatingPt( unsigned char [] );
void DisplayLineNum( int  );
void MakeNames( char, char [], char [], char [] );
int NextDataset( int, char * );
int OutputData( FILE *, double, double, int );
int QueryAbort( void );
int ReportDataError( char *, int );
int ReportFileError( char * );
void Update_llerrs( int, char[] );
```

```
/* ------------------------------------------------------------ */
void main()

{
char rootname[10];
int err, first;

clrscr();
first = TRUE;
err = FALSE;
while( NextDataset( first, rootname ) )
   {
   gotoxy( 5, 12 );
   printf( "Processing data set: %s", rootname );
   first = FALSE;
   err = ConvertAfile( rootname );
   if ( ! err )
      err = ConvertBfile( rootname );
   if ( ! err )
      err = ConvertCfile( rootname );
   if ( ! err )
      err = ConvertDfile( rootname );
   if ( ! err )
      err = ConvertFfile( rootname );
   if ( err )
      break;
   }
}

/* ------------------------------------------------------------
*   int NextDataSet( int start, char *datasetname )
*
*   This function open and reads the driver file returning the dataset ID
*   of the next data set to process.
*
*   Args:  start - an integer that should be set to TRUE the first time
*                  this function is called, otherwise FALSE.
*          datasetname - a pointer to a character array which will be
*                        filled with the name of the next data set to
*                        process.  This name is valid only when the
*                        function returns TRUE.
*
*  Returns:  more - TRUE if additional data needs to be processed.
*
*/

int NextDataset( int start, char *datasetname )

{
static FILE *fptr;
int more, stat;
char dum1[100], dum2[100];

more = TRUE;
if ( start )
   {
   fptr = fopen( GPSdriverFile, "r" );
   if ( fptr == NULL )
      more = FALSE;
   }
if ( more )
   {
   stat = fscanf( fptr,"%[^\n]%[\n]", dum1, dum2 );
   if ( stat != EOF  )
      strcpy( datasetname, dum1 );
```

```
   else
      {
      more = FALSE;
      fclose( fptr );
      }
   }
return( more );
}

/* ------------------------------------------------------------------*/
int ConvertAfile( char *name )

{
FILE *outptr;
double latitude, longitude, asciilat, asciilong, FractionalDegrees;
long int binlong, binlat, tmplongint;
int linenum, stop, stat, ErrsInData, NumSats, filehandle, good;
unsigned char tmpbyte, str1[100];
char rawfilename[30], outfilename[30], strid[10];
char DegreesLatitude[5], DegreesLongitude[5], minutestr[5], hundredthstr[5];

ErrsInData = 0;
stop = FALSE;
MakeNames( 'A', rawfilename, outfilename, name );
Update_llerrs( FALSE, rawfilename );
filehandle = open( rawfilename, O_RDONLY | O_BINARY );
if ( filehandle == -1 )
   stop = ReportFileError( rawfilename );
else
   {
   outptr = fopen( outfilename, "w" );
   linenum = 0;
   stat = read( filehandle, &str1[0], 1 );
   if ( stat == 0 || stat == -1 )
      stat = EOF;
   while( stat != EOF )
      {
      while( str1[0] != '$' && stat != EOF )
         {
         stat = read( filehandle, &str1[0], 1 );
         if ( stat == 0 || stat == -1 )
            stat = EOF;
         }
      if ( stat != EOF )
         {
         stat = read( filehandle, &str1[0], 2 );
         if ( stat == 0 || stat == -1 )
            stat = EOF;
         }
      if ( stat != EOF )
         {
         strncpy( strid, (char *)&str1[0], 2 );
         strid[2] = 0;
         if ( ! strcmp( strid, "GP" ) )
            {
            stat = read( filehandle, &str1[0], 38 );
            if ( stat == 0 || stat == -1 )
               stat = EOF;
            if ( stat != EOF )
               {
               NumSats = str1[34] - 48;
               strncpy( DegreesLatitude, (char *)&str1[11], 2 );
               DegreesLatitude[2] = 0;
               asciilat = atof( DegreesLatitude );
               strncpy( minutestr, (char *)&str1[13], 2 );
               minutestr[2] = 0;
               strncpy( hundredthstr, (char *)&str1[16], 2 );
```

```
        hundredthstr[2] = 0;
        FractionalDegrees = atof( hundredthstr ) + ( 10000.0 *
                            atof( minutestr ) );
        FractionalDegrees *= 100.0;
        FractionalDegrees = FractionalDegrees *
                            TTMINUTES_TO_FRACTION_DEG;
        asciilat = asciilat + FractionalDegrees;
        strncpy( DegreesLongitude, (char *)&str1[21], 3 );
        DegreesLongitude[3] = 0;
        asciilong = atof( DegreesLongitude );
        strncpy( minutestr, (char *)&str1[24], 2 );
        minutestr[2] = 0;
        strncpy( hundredthstr, (char *)&str1[27], 2 );
        hundredthstr[2] = 0;
        FractionalDegrees = atof( hundredthstr ) + ( 10000.0 *
                            atof( minutestr ) );
        FractionalDegrees *= 100.0;
        FractionalDegrees = FractionalDegrees *
                            TTMINUTES_TO_FRACTION_DEG;
        asciilong = asciilong + FractionalDegrees;
        }
    }
if ( ! strcmp( strid, "$B" ) )
    {
    stat = read( filehandle, &str1[0], 15 );
    if ( stat == 0 || stat == -1 )
        stat = EOF;
    if ( stat != EOF )
        {
        binlat = 0L;

        tmpbyte = str1[5];
        tmplongint = (long)tmpbyte << 24;

        tmpbyte = str1[6];
        tmplongint |= ( (long)tmpbyte << 16 );

        tmpbyte = str1[7];
        tmplongint |= ( (long)tmpbyte << 8 );

        tmpbyte = str1[8];
        tmplongint |= tmpbyte;
        binlat = tmplongint - 1;
        binlat = ~binlat;
        latitude = (double)( binlat / 10000000.0 );
        binlong = 0L;

        tmpbyte = str1[9];
        tmplongint = (long)tmpbyte << 24;

        tmpbyte = str1[10];
        tmplongint |= ( (long)tmpbyte << 16 );

        tmpbyte = str1[11];
        tmplongint |= ( (long)tmpbyte << 8 );

        tmpbyte = str1[12];
        tmplongint |= tmpbyte;

        binlong = tmplongint - 1;
        binlong = ~binlong;
        longitude = (double)( binlong / 10000000.0 );
        good = TRUE;
        if ( fabs( longitude ) > 110.0  || fabs( longitude ) < 102.0 )
            good = FALSE;
        if ( fabs( latitude ) > 40.0  || fabs( latitude ) < 30.0  )
            good = FALSE;
```

```
                if ( ! good )
                   {
                   longitude = asciilong;
                   latitude = asciilat;
                   }
                ErrsInData += OutputData(outptr, latitude, longitude, NumSats);
                if ( ErrsInData > MAXERROR )
                   {
                   stat = EOF;
                   stop = ReportDataError( rawfilename, linenum );
                   }
                }
             }
          DisplayLineNum( linenum++ );
          }
       }
    fprintf( outptr, "\n" );
    close( filehandle );
    fclose( outptr );
    }
return( stop );
}

/* ---------------------------------------------------------------- */
int ConvertBfile( char *name )

{
FILE *rawptr, *outptr;
double latitude, longitude, FractionalDegrees;
int linenum, stat, found, ErrsInData, NumSats, stop;
char rawfilename[30], outfilename[30], str1[100], str2[5], strid[15];
char DegreesLatitude[5], DegreesLongitude[5], minutestr[5],
tenthousandthstr[5];

ErrsInData = 0;
stop = FALSE;
MakeNames( 'B', rawfilename, outfilename, name );
rawptr = fopen( rawfilename, "r" );
Update_llerrs( FALSE, rawfilename );
if ( rawptr == NULL )
   stop = ReportFileError( rawfilename );
else
   {
   outptr = fopen( outfilename, "w" );
   linenum = 0;
   stat = fscanf( rawptr, "%[ -~]%c%[\n]", str1, str2, str2 );
   do
      {
      DisplayLineNum( linenum++ );
      strncpy( strid, &str1[1], 9 );
      strid[9] = 0;
      if ( strcmp( strid, "PMVXG,021" ) )
         found = FALSE;
      else
         found = TRUE;
      if ( found )
         {
         NumSats = -1;
         if ( str1[75] == '5' )
            {
            if ( str1[76] == '1' )
              NumSats = -1;
            }
         else
            {
            if ( str1[76] == '2' )
               NumSats = 3;
```

```
            if ( str1[76] == '3' )
               NumSats = 4;
            }
         strncpy( DegreesLatitude, &str1[21], 2 );
         DegreesLatitude[2] = 0;
         latitude = atof( DegreesLatitude );
         strncpy( minutestr, &str1[23], 2 );
         minutestr[2] = 0;
         strncpy( tenthousandthstr, &str1[26], 4 );
         tenthousandthstr[4] = 0;
         FractionalDegrees = atof( tenthousandthstr ) +
                             ( 10000.0 * atof( minutestr ) );
         FractionalDegrees = FractionalDegrees * TTMINUTES_TO_FRACTION_DEG;
         latitude = latitude + FractionalDegrees;
         strncpy( DegreesLongitude, &str1[33], 3 );
         DegreesLongitude[3] = 0;
         longitude = atof( DegreesLongitude );
         strncpy( minutestr, &str1[36], 2 );
         minutestr[2] = 0;
         strncpy( tenthousandthstr, &str1[39], 4 );
         tenthousandthstr[4] = 0;
         FractionalDegrees = atof( tenthousandthstr ) +
                             ( 10000.0 * atof( minutestr ) );
         FractionalDegrees = FractionalDegrees * TTMINUTES_TO_FRACTION_DEG;
         longitude = longitude + FractionalDegrees;
         ErrsInData += OutputData( outptr, latitude, longitude, NumSats );
         if ( ErrsInData > MAXERROR )
            {
            stat = EOF;
            stop = ReportDataError( rawfilename, linenum );
            }
         }
      stat = fscanf( rawptr, "%[ -~]%c%[\n]", str1, str2, str2 );
      } while ( stat != EOF );

   fprintf( outptr, "\n" );
   fclose( rawptr );
   fclose( outptr );
   }
return( stop );
}

/* ------------------------------------------------------------------*/
int ConvertFfile( char *name )

{
FILE *rawptr, *outptr;
double latitude, longitude;
int linenum, stat, found, ErrsInData, NumSats, stop;
char rawfilename[30], outfilename[30], str1[100], str2[5], strid[15];
char DegreesLatitude[15], DegreesLongitude[15], FractionalDegrees[10];

ErrsInData = 0;
stop = FALSE;
MakeNames( 'F', rawfilename, outfilename, name );
rawptr = fopen( rawfilename, "r" );
Update_llerrs( FALSE, rawfilename );
if ( rawptr == NULL )
   stop = ReportFileError( rawfilename );
else
   {
   outptr = fopen( outfilename, "w" );
   linenum = 0;
   stat = fscanf( rawptr, "%[^\n]%[\n]", str1, str2 );
   do
      {
      DisplayLineNum( linenum++ );
```

```
      strncpy( strid, &str1[1], 3 );
      strid[3] = 0;
      if ( strcmp( strid, "RPV" ) )
         found = FALSE;
      else
         found = TRUE;
      if ( found )
         {
         NumSats = -1;
         if ( str1[32] == '0' )
            NumSats = 3;
         if ( str1[32] == '1' )
            NumSats = 4;
         strncpy( DegreesLatitude, &str1[10], 2 );
         DegreesLatitude[2] = 0;
         latitude = atof( DegreesLatitude );
         strncpy( FractionalDegrees, &str1[12], 5 );
         FractionalDegrees[5] = 0;
         latitude = latitude + ( atof( FractionalDegrees ) *
                    ( 1 / 100000.0 ));
         strncpy( DegreesLongitude, &str1[18], 3 );
         DegreesLongitude[3] = 0;
         longitude = atof( DegreesLongitude );
         strncpy( FractionalDegrees, &str1[21], 5 );
         FractionalDegrees[5] = 0;
         longitude = longitude + ( atof( FractionalDegrees) *
                     ( 1 / 100000.0 ) );
         ErrsInData += OutputData( outptr, latitude, longitude, NumSats );
         if ( ErrsInData > MAXERROR )
            {
            stat = EOF;
            stop = ReportDataError( rawfilename, linenum );
            }
         }
      stat = fscanf( rawptr, "%[^\n]%[\n]", str1, str2 );
      } while ( stat != EOF );

   fclose( rawptr );
   fclose( outptr );
   }
return( stop );
}

/* ------------------------------------------------------------------*/
int ConvertDfile( char *name )

{
FILE *rawptr, *outptr;
double latitude, longitude, RadsLatitude, RadsLongitude;
double newtime, oldtime;
int linenum, i, stat, found, ErrsInData, NumSats, satnum[4], stop;
char rawfilename[30], outfilename[30], str1[100], str2[5], strid[15];

newtime = oldtime = 0.0;
ErrsInData = 0;
stop = FALSE;
MakeNames( 'D', rawfilename, outfilename, name );
Update_llerrs( FALSE, rawfilename );
rawptr = fopen( rawfilename, "r" );
if ( rawptr == NULL )
   stop = ReportFileError( rawfilename );
else
   {
   outptr = fopen( outfilename, "w" );
   linenum = 0;
   stat = fscanf( rawptr, "%[^\n]%[\n]", str1, str2 );
   do
```

```
        {
        strncpy( strid, &str1[0], 4 );
        strid[4] = 0;

        if ( strcmp( strid, "  29" ) )
           found = FALSE;
        else
           found = TRUE;
        if ( found )
           {
           sscanf( &str1[5], "%lf", &newtime );
           if ( newtime > ( oldtime + 30.0 ) )
              {
              oldtime = newtime;
              found = TRUE;
              }
           else
              found = FALSE;
           }

        if ( found )
           {
           DisplayLineNum( linenum++ );
           NumSats = 0;
           sscanf( &str1[50], "%d %d %d %d", &satnum[0], &satnum[1],
                        &satnum[2], &satnum[3] );
           for( i = 0; i < 4; i++ )
              if ( satnum[i] > 0 )
                 NumSats++;
           sscanf( &str1[18], "%lf %lf", &RadsLatitude, &RadsLongitude );
           latitude = RadsLatitude * RADS_TO_DEGS;
           longitude = RadsLongitude * RADS_TO_DEGS;
           ErrsInData += OutputData( outptr, latitude, longitude, NumSats );
           if ( ErrsInData > MAXERROR )
              {
              stat = EOF;
              stop = ReportDataError( rawfilename, linenum );
              }
           }
        stat = fscanf( rawptr, "%[^\n]%[\n]", str1, str2 );
        } while ( stat != EOF );

     fclose( rawptr );
     fclose( outptr );
     }
  return( stop );
  }

  /* ------------------------------------------------------------- */
  int ConvertCfile( char *name )

  {
  FILE *rawptr, *outptr;
  double latitude, longitude;
  int linenum, i, stat, dummyval, rawval, NumSats, rawSats, stop;
  int ready, id1, id2, id3, ErrsInData;
  unsigned char bytes[8];
  char rawfilename[30], outfilename[30], str1[500], str2[5];

  ErrsInData = 0;
  stop = FALSE;
  MakeNames( 'C', rawfilename, outfilename, name );
  rawptr = fopen( rawfilename, "r" );
  Update_llerrs( FALSE, rawfilename );
  if ( rawptr == NULL )
     stop = ReportFileError( rawfilename );
  else
```

```
    {
    outptr = fopen( outfilename, "w" );
    linenum = 0;
    do
       {
       DisplayLineNum( linenum++ );
       stat = fscanf( rawptr, "%d", &id1 );
       stat = fscanf( rawptr, "%d", &id2 );
       stat = fscanf( rawptr, "%d", &id3 );
       ready = FALSE;
       if ( id1 == 255 && id2 == 129 && id3 == 103 )
         ready = TRUE;
       if ( ! ready )
         stat = fscanf( rawptr, "%[^\n]%[\n]", str1, str2 );
       else
         {
         for( i = 0; i < 63; i ++ )
            fscanf( rawptr, "%d", &dummyval );
         for( i = 0; i < 6; i ++ )
            {
            fscanf( rawptr, "%d", &rawval );
            bytes[i] = rawval & 0xFF;
            }
         latitude = ConvertToFloatingPt( bytes ) * RADS_TO_DEGS;
         for( i = 0; i < 6; i ++ )
            {
            fscanf( rawptr, "%d", &rawval );
            bytes[i] = rawval & 0xFF;
            }
         longitude = ConvertToFloatingPt( bytes ) * RADS_TO_DEGS;
         for( i = 0; i < 47; i ++ )
            fscanf( rawptr, "%d", &rawval );
         rawSats = ( ( rawval & 0x0A )  );
         switch( rawSats )
           {
           case 0:              // Bits 1 and 3 low
           case 2:              // Bit 1 low, bit 3 high
             NumSats = -1;
             break;
           case 8:             // Bit 1 high, bit 3 low
             NumSats = 4;
             break;
           case 10:           // Bits 1 and 3 high
             NumSats = 3;
             break;
           }
         ErrsInData += OutputData( outptr, latitude, longitude, NumSats );
         if ( ErrsInData > MAXERROR )
            {
            stat = EOF;
            stop = ReportDataError( rawfilename, linenum );
            }
         else
            stat = fscanf( rawptr, "%[^\n]%[\n]", str1, str2 );
         }
       } while ( stat != EOF );
    fclose( rawptr );
    fclose( outptr );
    }
return( stop );
}

/* ------------------------------------------------------------*/
void MakeNames( char filetype, char rawname[], char outname[], char name[]

{
int len;
```

```
rawname[0] = filetype;
strcpy( &rawname[1], name );
len = strlen( rawname );
strcpy( outname, rawname );
strcpy( &rawname[len], ".DAT" );
strcpy( &outname[len], ".OUT" );
gotoxy( 5, 15 );
printf( "Processing File: %s line:", rawname );
}

/* ------------------------------------------------------------*/
double ConvertToFloatingPt( unsigned char buff[] )

{
if( (float)buff[0] == 0.0 )
   valu.dv = 0.0;
else
   {
   valu.bd[7] = (buff[5] & 0x80 );
   valu.bd[7] |= ( ( ( buff[0] ^ 0x80 ) + 0x7E ) >> 4 ) & 0x0F;

   if (( valu.bd[7] & 0x04) && ( valu.bd[7] & 0x08 ))
      valu.bd[7] |= 0x70;
   if (( valu.bd[7] & 0x04 ) && ( ! ( valu.bd[7] & 0x8 )))
      valu.bd[7] |= 0x3C;
   if (!(valu.bd[7] & 0x04) && (valu.bd[7] & 0x08 ))
      valu.bd[7] ^= 0x48;


   valu.bd[6] = (((buff[0] ^ 0x80 ) + 0x7E ) << 4 ) & 0xF0;
   valu.bd[6] |= (buff[5] & 0x78 ) >> 3;
   valu.bd[5] = buff[5] << 5;
   valu.bd[5] |= buff[4] >> 3;
   valu.bd[4] = buff[4] << 5;
   valu.bd[4] |= buff[3] >> 3;
   valu.bd[3] = buff[3] << 5;
   valu.bd[3] |= buff[2] >> 3;
   valu.bd[2] = buff[2] << 5;
   valu.bd[2] |= buff[1] >> 3;
   valu.bd[1] = buff[1] << 5;
   valu.bd[0] = 0;
   }
return( valu.dv );
}

/* ------------------------------------------------------------*/
int OutputData( FILE *fpptr, double latitude, double longitude, int Sats )

{
int good, err;
char dummy[3];

good = TRUE;
if ( fabs( longitude ) > 110.0  || fabs( longitude ) < 102.0 )
   good = FALSE;
if ( fabs( latitude ) > 40.0  || fabs( latitude ) < 30.0  )
   good = FALSE;
if ( good )
   {
   err = FALSE;
   fprintf( fpptr, "%10.6lf    %10.6lf  %2d\n", latitude, longitude, Sats );
   }
else
   {
   err = TRUE;
   Update_llerrs( err, dummy );
```

```
    }
return( err );
}

/* ---------------------------------------------------------------*/
void Update_llerrs( int err, char fname[] )

{
static int numerrs = 0, linenum = 0;
char fileid;

if ( err )
    {
    numerrs++;
    gotoxy( 57, linenum );
    printf( "%3d", numerrs );
    }
else
    {
    fileid = fname[0];
    switch( fileid )
       {
       case 'A':
          linenum = 2;
          break;
       case 'B':
          linenum = 3;
          break;
       case 'C':
         linenum = 4;
          break;
       case 'D':
          linenum = 5;
          break;
       case 'F':
          linenum = 6;
          break;
       }
    DisplayLineNum( 0 );
    gotoxy( 5, linenum );
    printf( "%s", fname );
    numerrs = 0;
    gotoxy( 20, linenum );
    printf( "Number of Latitude/Longitude errors: %3d", numerrs );
    }
}

/* ---------------------------------------------------------------*/
int ReportDataError( char *name, int linenum )

{
int finished;

gotoxy( 5, 21 );
printf(  "Too many errors in file %s, last error was at line number: %d.",
name, linenum );
finished = QueryAbort();
gotoxy( 5, 21 );
printf( "
" );
gotoxy( 5, 22 );
printf( "                                                                " );
gotoxy( 5, 23 );
printf( "                                                                " );
return( finished );
}
```

```
/* ---------------------------------------------------------------*/
int ReportFileError( char *name )

{
int finished;

gotoxy( 5, 21 );
printf(  "Error, cannot find input file %s !", name );
finished = QueryAbort();
gotoxy( 5, 21 );
printf( "                                                          " );
gotoxy( 5, 22 );
printf( "                                                          " );
gotoxy( 5, 23 );
printf( "                                                          " );
return( finished );
}

/* ---------------------------------------------------------------*/
int QueryAbort()
{
int response, finished;

gotoxy( 5, 22 );
printf( "Enter 'P' to abort program, or 'F' to abort " );
gotoxy( 5, 23 );
printf( "processing of this file only. [P/F]" );
do
   {
   response = toupper( getch() );
   } while( response != 'P' && response != 'F' );
if ( response == 'P' )
   finished = TRUE;
else
   {
   finished = FALSE;
   }
return( finished );
}

/* ---------------------------------------------------------------*/
void DisplayLineNum( int num )

{
gotoxy( 41, 15 );
printf( "%5d", num );
}

/* ---------------------------------------------------------- EOF */
```

APPENDIX C.2: SOFTWARE LISTING - GPS DATA ACQUISITION PROGRAM

Developed by:
Raymond Byrne
Advanced Vehicle Development Department, 9616
Sandia National Laboratories
Albuquerque, NM 87185

APPENDIX C.2: SOFTWARE LISTING - GPS DATA ACQUISITION PROGRAM

Gathers serial GPS data and logs data to a file. Uses the two serial ports on the ZT 8901 processor card as well as the four serial ports on the ZT88CT41 Quad Serial Card.

```
/*****************************************************************
 *                                                               *
 *   FILE GPSLOG.C                                               *
 *                                                               *
 *   Developed by Ray Byrne, Department 9616                     *
 *                                                               *
 *   This program gathers GPS serial data from the 2 serial ports*
 *   on the Ziatech 8901 processer card as well as from the 4    *
 *   serial ports on the ZT 8841 Quad Serial Board               *
 *                                                               *
 *****************************************************************/

#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>

#include <ddp.h>
#include <pnc.h>
#include <ser8250.h>

#define     buffer_size     200
#define     half_full       100
#define     timeout         0

#define     ch_0            0
#define     ch_1            1
#define     ch_2            2
#define     ch_3            3
#define     ch_4            4
#define     ch_5            5
#define     ch_6            6

#define   TRUE    1
#define   FALSE   0


/* ------------FUNCTION PROTOTYPES---------- */

void get_time (     unsigned char   *month,
                    unsigned char   *date,
                    unsigned char   *hour,
                    unsigned char   *ten_minutes);

void init_filenames   (  unsigned char   month,
                         unsigned char   date,
                         unsigned char   hour,
                         unsigned char   ten_minutes,
                         char            A[13],
                         char            B[13],
                         char            C[13],
                         char            D[13]);
```

```
                        char        E[13],
                        char        F[13]);

void create_files       (char       A[13],
                        char        B[13],
                        char        C[13],
                        char        D[13],
                        char        E[13],
                        char        F[13]);

void Init_PIC (void);

void close_files (void);

void Init_Serial_Ports (void);

void Save_A (void);

void Save_B (void);

void Save_C (void);

void Save_D (void);

void Save_E (void);

void Save_F (void);

void Set_Up_Rec (void);

void Req_Magellan (void);

/* --------------------VARIABLE DECLARATIONS---------------------- */

FILE        *FA, *FB, *FC, *FD, *FE, *FF;

unsigned char   eom_value = 0x0a;

unsigned char   Rockwell_count = 0,
                Record_Rockwell = FALSE;

unsigned char   month,
                date,
                hour,
                ten_minutes;

char  A[13], B[13], C[13], D[13], E[13], F[13];

unsigned char   buffer_A[2 * buffer_size],
                buffer_B[2 * buffer_size],
                buffer_C[2 * buffer_size],
                buffer_D[2 * buffer_size],
                buffer_E[2 * buffer_size],
```

```
                buffer_F[2 * buffer_size];


/* --------------------FUNCTIONS---------------------- */

void get_time  (unsigned char   *month,
                unsigned char   *date,
                unsigned char   *hour,
                unsigned char   *ten_minutes)
{

      unsigned char   dummy;

      outportb( 0xFDC0, (inportb ( 0xFDC0) & 0x3F) );   /* clear ARS, APS */


      outportb (0xFDDD, 0);    /* clear month */
      outportb (0xFDDC, 0);    /* clear days */
      outportb (0xFDDB, 0);    /* clear hours */
      outportb (0xFDDA, 0);    /* clear minutes */

       outportb (0xFDC4, 0x80);       /* track current time */
      outportb (0xFDC4, 0x00);        /* latch current time */


      dummy = inportb (0xFDDD);
      *month = (dummy & 0x0F) + (dummy >> 4) * 10;

      dummy = inportb (0xFDDC);
      *date = (dummy & 0x0F) + (dummy >> 4) * 10;

      dummy = inportb (0xFDDB);
      *hour = (dummy & 0x0F) + (dummy >> 4) * 10;

      dummy = inportb (0xFDDA);
      *ten_minutes = (dummy & 0x0F) + (dummy >> 4) * 10;

      return;

}     /* end of get_time() */


void init_filenames     (   unsigned char   month,
                            unsigned char   date,
                            unsigned char   hour,
                            unsigned char   ten_minutes,
                            char            A[13],
                            char            B[13],
                            char            C[13],
                            char            D[13],
                            char            E[13],
```

```
                        char            F[13])
{

    unsigned int        month_tens,
                        month_ones,
                        date_tens,
                        date_ones,
                        hour_tens,
                        hour_ones;


    month_tens = month / 10;
    month_ones = month % 10;
    date_tens = date / 10;
    date_ones = date % 10;
    hour_tens = hour / 10;
    hour_ones = hour % 10;


    sprintf (A, "A%d%d%d%d%d%d%c.DAT",month_tens, month_ones,
                date_tens, date_ones, hour_tens, hour_ones, ten_minutes + 48);

    sprintf (B, "B%d%d%d%d%d%d%c.DAT",month_tens, month_ones,
                date_tens, date_ones, hour_tens, hour_ones, ten_minutes + 48);

    sprintf (C, "C%d%d%d%d%d%d%c.DAT",month_tens, month_ones,
                date_tens, date_ones, hour_tens, hour_ones, ten_minutes + 48);

    sprintf (D, "D%d%d%d%d%d%d%c.DAT",month_tens, month_ones,
                date_tens, date_ones, hour_tens, hour_ones, ten_minutes + 48);

    sprintf (E, "E%d%d%d%d%d%d%c.DAT",month_tens, month_ones,
                date_tens, date_ones, hour_tens, hour_ones, ten_minutes + 48);

    sprintf (F, "F%d%d%d%d%d%d%c.DAT",month_tens, month_ones,
                date_tens, date_ones, hour_tens, hour_ones, ten_minutes + 48);

    return;


}   /* end of init_filenames() */


void create_files   (   char    A[13],
                        char    B[13],
                        char    C[13],
                        char    D[13],
                        char    E[13],
                        char    F[13])
{
    FA = fopen(A,"a");
    FB = fopen(B,"a");
```

```
    FC = fopen(C,"a");
    FD = fopen(D,"a");
    FE = fopen(E,"a");
    FF = fopen(F,"a");

    return;

}   /* end of create_files() */


void Init_PIC (void)
{
    unsigned int    ercode,
                    version,
                    cpu_type = DDP_ZT_8901,
                    save_mask = DDP_TRUE,
                    addr0 = 0xA8,
                    addr1 = 0xA9,
                    base_vector = 0x80,
                    master_ir_level = 2;


    ercode = PicInitSw (&version);
    if (ercode) printf ("Error initializing PIC Software\n");

    ercode = PicInitHw (cpu_type, save_mask);
    if (ercode) printf ("Error initializing PIC Hardware\n");

    ercode = PicInitSlave (addr0, addr1, base_vector, master_ir_level);
    if (ercode) printf ("Error initializing Slave PIC\n");

    return;

}   /* end of Init_PIC */


void close_files (void)
{

    fclose (FA);
    fclose (FB);
    fclose (FC);
    fclose (FD);
    fclose (FE);
    fclose (FF);

    return;

}   /* end of close_files() */
```

```
void Init_Serial_Ports (void)
{
      unsigned int     ercode,
                       version;


      ercode = Ser8250InitSw(&version);
      if (ercode) printf("Error initializing Serial Software\n");



      /* initialize COM1, 9600 Baud, 8 bit, 1 stop, no parity */
      /* Magellan OEM GPS Module on J5, ZT8901 */

      ercode = Ser8250InitHw (ch_0, 0x3f8, buffer_A, buffer_size, 0x01c3,0x0c,
                                        DDP_ZT_8901, eom_value);
      if (ercode) printf("Error initializing COM1 port\n");



      /* initialize COM2, 4800 Baud, 8 bit, 1 stop, no parity */
      /* Magnavox GPS Engine on J4, ZT8901 */

      ercode = Ser8250InitHw (ch_1, 0x2f8, buffer_B, buffer_size, 0x0183, 0x0b,
                                          DDP_ZT_8901, eom_value);
      if (ercode) printf("Error initializing COM2 port\n");



      /* initialize 8841-COM1, 9600 Baud, 8 bit, 1 stop, odd parity */
      /* Rockwell GPS Unit on J1, ZT8841 */

      ercode = Ser8250InitHw (ch_2, 0x3e0, buffer_C, buffer_size, 0x01cb, 0x84,
                                          DDP_ZT_8901, eom_value);
      if (ercode) printf ("Error initializing ch_2\n");



      /* initialize 8841-COM2, 2400 Baud, 7 bit, 1 stop, odd parity */
      /* Magnavox SST GPS System */

      ercode = Ser8250InitHw (ch_3, 0x2e0, buffer_D, buffer_size, 0x014a, 0x83,
                                          DDP_ZT_8901, eom_value);
      if (ercode) printf ("Error initializing ch_3\n");



      /* initialize 8841-COM3, 2400 Baud, 7 bit, 1 stop, odd parity */

      ercode = Ser8250InitHw (ch_4, 0x3e8, buffer_E, buffer_size, 0x014a, 0x86,
                                          DDP_ZT_8901, eom_value);
      if (ercode) printf ("Error initializing ch_4\n");
```

```
        /* initialize 8841-COM4, 9600 Baud, 8 bit, 1 stop, odd parity */
        /* Trimble Placer GPS on J4, ZT8841 */

        ercode = Ser8250InitHw (ch_5, 0x2e8, buffer_F, buffer_size, 0x01eb, 0x85,
                                                DDP_ZT_8901, eom_value);
        if (ercode) printf ("Error initializing ch_5\n");


}       /* end of Init_Serial_Ports() */



void Save_A (void)
{
        unsigned int    inchar,
                        status,
                        ercode;

        ercode = Ser8250Recv (ch_0, &inchar, timeout);
        if (ercode)
        {
                printf ("Error %i on Channel 0\n", ercode);
                ercode = Ser8250Status (ch_0, &status);
        }
        fprintf (FA, "%c", inchar);

}       /* end of Save_A() */



void Save_B (void)
{
        unsigned int    inchar,
                        status,
                        ercode;

        ercode = Ser8250Recv (ch_1, &inchar, timeout);
        if (ercode)
        {
                printf ("Error %i on Channel 1\n", ercode);
                ercode = Ser8250Status (ch_1, &status);
        }
        fprintf (FB, "%c", inchar);

}       /* end of Save_B() */



void Save_C (void)
{
```

```
unsigned int    inchar,
                status,
                ercode;

static unsigned int flag = FALSE;


ercode = Ser8250Recv (ch_2, &inchar, timeout);

if (ercode)
{
      printf ("Error %i on Channel 2\n", ercode);
      ercode = Ser8250Status (ch_2, &status);
}
switch (inchar)
{
      case 255:       flag = TRUE;
                      break;

      case 129:       if (flag == TRUE)
                      {
                            Rockwell_count++;
                            if (Rockwell_count == 30)
                            {
                                  fprintf (FC, "\n%i %i ", 255, 129);
                                  flag = FALSE;
                                  Record_Rockwell = TRUE;
                            }
                            if (Rockwell_count > 30)
                            {
                                  Record_Rockwell = FALSE;
                                  Rockwell_count = 0;
                            }
                      }     /* end of if */
                      else
                      {
                            if (Record_Rockwell == TRUE)
                                        fprintf (FC,"%i ", 129);
                      }
                      break;

      default:        if (flag == TRUE)
                      {
                            if (Record_Rockwell == TRUE)
                                        fprintf (FC, "%i ", 255);
                            flag = FALSE;
                      }
                      if (Record_Rockwell == TRUE)
                                  fprintf (FC, "%i ", inchar);
                      break;

}     /* end of switch */
```

```
}     /* end of Save_C() */


void Save_D (void)
{
      unsigned int     inchar,
                       status,
                       ercode;

      ercode = Ser8250Recv (ch_3, &inchar, timeout);
       if (ercode)
      {
            printf ("Error %i on Channel 3\n", ercode);
            ercode = Ser8250Status (ch_3, &status);
      }
      fprintf (FD, "%c", inchar);

}     /* end of Save_D() */


void Save_E (void)
{
      unsigned int     inchar,
                       status,
                       ercode;

      ercode = Ser8250Recv (ch_4, &inchar, timeout);
      if (ercode)
      {
            printf ("Error %i on Channel 4\n", ercode);
            ercode = Ser8250Status (ch_4, &status);
      }
      fprintf (FE, "%c", inchar);

}     /* end of Save_E() */


void Save_F (void)
{

/* Saves data to file F*******.DAT from Trimble GPS unit */

      unsigned int     inchar,
                       status,
                       ercode;

      ercode = Ser8250Recv (ch_5, &inchar, timeout);
      if (ercode)
      {
```

```
            printf ("Error %i on Channel 5\n", ercode);
            ercode = Ser8250Status (ch_5, &status);
        }
        if (inchar == '>')
        {
            fprintf (FF, "\n");
            Req_Magellan();
        }
        fprintf (FF, "%c", inchar);

}   /* end of Save_D() */


void Set_Up_Rec ( void )
{
        char Magnavox[] = "$PMVXG,007,021,1,1,,30,,,";
        char Trimble[] = ">FPV00300000<";


        unsigned int ercode, status, i;


        for(i=0;i<strlen(Magnavox);i++)
        {
            ercode = Ser8250Xmit(ch_1,Magnavox[i],0);
            if (ercode)
            {
                printf ("Error %i on Channel 1\n", ercode);
                ercode = Ser8250Status (ch_1, &status);
            }

        }

        Ser8250Xmit(ch_1,0x0d,0);
        Ser8250Xmit(ch_1,0x0a,0);

        for(i=0;i<strlen(Trimble);i++)
        {
            ercode = Ser8250Xmit(ch_5,Trimble[i],0);
            if (ercode)
            {
                printf ("Error %i on Channel 5\n", ercode);
                ercode = Ser8250Status (ch_5, &status);
            }
        }

}   /* end of */



void Req_Magellan (void)
{
        char Magellan[] = "$PMGLL,00,B00,1,B,";
```

```
        char Magellan1[] = "$PMGLL,00,B00,1,A,";
        unsigned int i, ercode, status;


         for(i=0;i<strlen(Magellan);i++)
        {
              ercode = Ser8250Xmit(ch_0,Magellan[i],0);
              if (ercode)
              {
                    printf ("Error %i on Channel 0\n", ercode);
                    ercode = Ser8250Status (ch_0, &status);
              }

        }

        Ser8250Xmit(ch_0,0x0d,0);
        Ser8250Xmit(ch_0,0x0a,0);


        for(i=0;i<strlen(Magellan1);i++)
        {
              ercode = Ser8250Xmit(ch_0,Magellan1[i],0);
              if (ercode)
              {
                    printf ("Error %i on Channel 0\n", ercode);
                    ercode = Ser8250Status (ch_0, &status);
              }

        }

        Ser8250Xmit(ch_0,0x0d,0);
        Ser8250Xmit(ch_0,0x0a,0);


}       /* end of Req_Magellan */



main()
{

        unsigned int    ercode,
                        count,
                        status,
                        com_count;

        unsigned int    ch;

        unsigned int    inchar;

        get_time (&month, &date, &hour, &ten_minutes);

        init_filenames (month, date, hour, ten_minutes, &A[0], &B[0], &C[0]
```

```
                                , &D[0], &E[0], &F[0]);

Init_PIC();

Init_Serial_Ports();

create_files(A, B, C, D, E, F);

Set_Up_Rec();



while (!kbhit())
{

      ercode = Ser8250RecvChk (ch_0, &count);
      if (ercode) printf ("Error: Channel 0 Buffer");
      if (count >= 1) Save_A();


      ercode = Ser8250RecvChk (ch_1, &count);
      if (ercode) printf ("Error: Channel 1 Buffer");
      if (count >= 1) Save_B();


      ercode = Ser8250RecvChk (ch_2, &count);
      if (ercode) printf ("Error: Channel 2 Buffer");
      if (count >= 1) Save_C();


      ercode = Ser8250RecvChk (ch_3, &count);
      if (ercode) printf ("Error: Channel 3 Buffer");
      if (count >= 1) Save_D();


      ercode = Ser8250RecvChk (ch_4, &count);
      if (ercode) printf ("Error: Channel 4 Buffer");
      if (count >= 1) Save_E();


      ercode = Ser8250RecvChk (ch_5, &count);
      if (ercode) printf ("Error: Channel 5 Buffer");
      if (count >= 1) Save_F();


}     /* end of while (!kbhit()) */

ercode = Ser8250Exit();
if (ercode) printf ("Error closing serial ports\n");

close_files();
```

```
}    /* end of main */
```

APPENDIX C.3: SOFTWARE LISTING - ASCII TO BINARY CONVERSION PROGRAM

Developed by:
Frank Wunderlin*
Technadyne Engineering Consultants, Inc.
P.O. Box 1328
Albuquerque, NM 87192

* Work performed under Contract No. 87-2050 for the Advanced Vehicle Development Department (9616), Sandia National Laboratories.

APPENDIX C.3: SOFTWARE LISTING - ASCII TO BINARY CONVERSION PROGRAM

This program converts the raw Rockwell GPS data, stored in ASCII, to a binary file so that the Rockwell provided programs can be used to analyze the Rockwell data.

```
/*
*     FILE CONVERT.C
*
*     Developed by Frank Wunderlin.
*
*     This file contains functions used to read and convert processed GPS data
*     files to binary data.
*
*     Functions included in this file are:
*
*     CConvert( char *name )
*     MakeNames( char rawname[], char outname[], char name[] )
*     ReportFileError( char *name )
*     NextFile( char filename[] )
*/

#include <stdio.h>
#include <conio.h>
#include <io.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <ctype.h>
#include <math.h>

#define FALSE 0
#define TRUE 1

int CConvert( char * );
void MakeNames( char rawname[], char outname[], char name[] );
int ReportFileError( char *name );
int NextFile( char[] );

/*                                                                    */
void main()

{
char rootname[10];

clrscr();
while( NextFile( rootname ) )
   {
   if ( rootname[0] != 0 )
     {
     clrscr();
     gotoxy( 5, 1 );
     printf( "Processing file: %s", rootname );
     CConvert( rootname );
     }
   }
}

/*                                                                    */
int CConvert( char *name )

{
FILE *rawptr;
int handle, dummyint;
unsigned char byte;
int stat, numvals;
char rawfilename[50], outfilename[30], str[5];

MakeNames( rawfilename, outfilename, name );
rawptr = fopen( rawfilename, "r" );
if ( rawptr == NULL )
   ReportFileError( rawfilename );
```

```
else
   {
   if ((handle = open( outfilename, O_CREAT | O_WRONLY | O_BINARY)) == -1)
      {
      ReportFileError( outfilename );
      exit( 1 );
      }
   numvals = fscanf( rawptr, "%d", &dummyint );
   do
      {
      if ( numvals == 1 )
       {
       byte = dummyint & 0xFF;
       write( handle, &byte, 1 );
       numvals = fscanf( rawptr, "%d", &dummyint );
       }
      else
       stat = fscanf( rawptr, "%[\n]", str2 );
      } while ( stat != EOF );
   fclose( rawptr );
   close( handle );
   }
return( 1 );
}

/* ---------------------------------------------------------------------*/
void MakeNames( char rawname[], char outname[], char name[] )

{
int len;

strcpy( &rawname[0], name );
len = strlen( rawname );
strcpy( outname, rawname );
strcpy( &rawname[len], ".DAT" );
strcpy( &outname[len], ".BIN" );
}

/* ---------------------------------------------------------------------*/
int ReportFileError( char *name )

{
gotoxy( 5, 21 );
printf(  "Error, opening file %s !", name );
gotoxy( 5, 23 );
printf( "Press any key to continue." );
while( ! kbhit() );
gotoxy( 5, 21 );
printf( "                                                              " );
gotoxy( 5, 22 );
printf( "                                                              " );
gotoxy( 5, 23 );
printf( "                                                              " );
return( 1 );
}

/* ---------------------------------------------------------------------------*/
int NextFile( char filename[]  )

{
int done, KeepWorking;
char inputstr[40];

done = FALSE;
KeepWorking = TRUE;
filename[0] = 0;
while( ! done )
```

```
    {
    clrscr();
    gotoxy( 5, 5 );
    printf( "Extension of file to convert to binary is assumed to be '.DAT' " );
    gotoxy( 5, 6 );
    printf( "converted file will have extension '.BIN'." );
    gotoxy( 5, 10 );
    printf("Input name of file ( do not enter extension ) to convert to binary"
);
    gotoxy( 5, 11 );
    printf( "followed by a CR or enter 'Q' to quit: " );
    scanf( "%s", inputstr );
    if ( strlen( inputstr ) == 1 )
      {
      if ( toupper( inputstr[0] ) == 'Q' )
        {
        KeepWorking = FALSE;
        done = TRUE;
        }
      }
    else
      {
      strcpy( filename, inputstr );
      done = TRUE;
      }
    }
return( KeepWorking );
}

/* ---------------------------------------------------------------------------- EOF
*/
```

APPENDIX C.4: SOFTWARE LISTING - DESCRIPTION OF ZIATECH SOFTWARE ROUTINES

Reprinted with permission from:
Ziatech Corporation
3433 Roberto Court
San Luis Obispo, CA 93401

APPENDIX C.4: SOFTWARE LISTING - DESCRIPTION OF ZIATECH SOFTWARE ROUTINES

8250 SERIAL DRIVER VERSION 3.21

The Ser8250 driver supports any serial channel that uses an 8250 UART (or compatible) and an 8259 interrupt controller (single or cascaded). This includes support for Ziatech's ZT 8840, ZT 8841, and most Ziatech CPU boards. The following list summarizes the driver's significant features:

- Fully interrupt driven
- Supports bi-directional communication
- Supports user-defined End-Of-Message (EOM) character detection
- Full support for character flow control (XON/XOFF)
- Full support for hardware flow control (RTS/CTS)
- Support for cascaded and directly connected interrupts
- User configurable buffer sizes
- Supports up to eight channels (can be recompiled for more)
- Supports baud rates up to 38.4K baud

The Ser8250 driver implements a fully interrupt driven interface to 8250 compatible UARTs. Characters to be transmitted are queued into a transmit buffer and are transferred to the desired UART as soon as the UART is ready to receive them. Characters received are queued into a receive buffer when they are received. The size of the transmit and receive buffers are configurable on a channel basis.

The Ser8250 driver also can optionally check for a user-defined End-Of-Message (EOM) characters. As EOM characters are received an internal counter is incremented in addition to storing the EOM character in the receive buffer. This internal counter can be checked with the Ser8250RecvEomChk procedure. As the EOM characters are removed from the buffer by calling Ser8250Recv, the internal counter is decremented. The EOM character counter is especially useful for determining when a complete variable length message has been received.

Two forms of flow control are provided by the Ser8250 driver: character flow control and hardware flow control. Character flow control and hardware flow control can be enabled or disabled in any combination on a channel basis. For the following discussion of these two forms of flow control, assume we are dealing with one end of an RS-232 channel.

Hardware flow control is implemented using the RTS output and the CTS input of the UART. When RTS is active, the driver is ready to receive data. When there is room for only 10 more characters in the receive buffer, the driver sets RTS to inactive. When 10 characters are left in the receive buffer, the driver sets RTS to active. Any characters received when RTS is inactive are buffered if room is available, otherwise, they are lost and a buffer overflow error is posted. CTS is monitored by the driver. When CTS is active, the driver transmits data. Otherwise data is queued until CTS becomes active, at which time data transmission is resumed. The actual pin numbers for RTS and CTS vary depending upon whether the particular channel is configured for DCE (Data Communication Equipment) or DTE (Data Terminal Equipment). Refer to your hardware manual for details on jumper configurations and pin numbers.

Character flow control is implemented using the standard XON and XOFF

characters. The driver monitors the data stream for XON and XOFF characters. When the receive buffer has room for 20 more characters, the driver sends XOFF. When only 10 characters are left in the receive buffer and XOFF has been sent, the driver sends XON. Any characters received after XOFF has been sent and before XON has been sent are buffered if room is available; otherwise, they are lost and a buffer overflow error is posted. The driver transmits data until XOFF is received. After XOFF is received, data is queued until XON is received, at which time data transmission resumes.

The Ser8250 driver provides applications with ten functions:

Ser8250InitSw - initializes the Ser8250 driver

Ser8250InitHw - initializes an 8250 (or compatible) UART

Ser8250Xmit - transmit a character on a channel

Ser8250Recv - receive a character on a channel

Ser8250Status - check the status of a channel

Ser8250RecvChk - check the number of characters in the receive buffer for channel

Ser8250RecvEomChk - check the number of EOM characters in the receive buffer for channel

Ser8250XmitChk - checks the number of characters free in the transmit buffer for a channel

Ser8250BuffFlush - flush the transmit and/or receive buffers for a channel

Ser8250Exit - disable all channels

1) Ser8250InitSw is called once to initialize the Ser8250 driver. This must be the first call to the Ser8250 driver. Ser8250InitSw requires one parameter for output:

version - the variable in which to return the driver version number. The version is stored in BCD. For version 3.00 a value of 0x0300 is returned. This parameter is passed by reference.

Ser8250InitSw returns:

DDP_ERR_NOERROR(0) - No error

Ser8250InitSw calls should use the following format:

```
unsigned ret_errcode;          /* error code returned */
unsigned int version           /* driver version returned */
```

```
ercode = Ser8250InitSw(&version);
.
```

2) Ser8250InitHw initializes one serial I/O channel. This procedure must be called once for each serial channel to be used. The buffers are initialized, the interrupt handler is installed, and the UART is configured. PicInitSw and PicInitHw must be called prior to Ser8250InitHw. Eight parameters are required for input.

chan - the channel number to initialize (0-7). This number is used to reference a particular serial channel. This parameter is passed by value

uart - the base address of the UART for this channel. This parameter is passed by value

buffer - the address of the data buffer. This is a pointer to a character array that is to be divided evenly into two buffers. This parameter is passed by value.

buf_size - The size of each buffer. The character array pointed to by 'buffer' must be 'buf_size' * 2 characters long. This parameter is passed by value

mode - specifies the operational characteristics for this channel. The table below describes the bit definitions for this parameter. This parameter is passed by value

bit(s)	description
15-12	Reserved (leave as zero)
11	Echo input 0 - disabled 1 - enabled
10	Hardware flow control (RTS/CTS) 0 - disabled 1 - enabled
9	Character flow control (XON/XOFF) 0 - disabled 1 - enabled
8-6	Baud rate: 8 7 6 ----- 0 0 0 - 19.2K baud 0 0 1 - 38.4K baud 0 1 0 - 300 baud 0 1 1 - 600 baud 1 0 0 - 1200 baud 1 0 1 - 2400 baud 1 1 0 - 4800 baud 1 1 1 - 9600 baud
5-3	Parity 5 4 3 ----- X X 0 - No parity 0 0 1 - Odd parity 0 1 1 - Even parity

1 0 1 - Mark (High) parity
1 1 1 - Space (Low) parity

2 Stop bits:
0 - one stop bit
1 - two stop bits (1.5 for 5 data bits)

1-0 Data bits:
1 0
---
0 0 - 5 data bits
0 1 - 6 data bits
1 0 - 7 data bits
1 1 - 8 data bits

vector - the interrupt vector number to use for this channel. This number must correspond to the interrupt vector generated by the 8259 interrupt controller for this UARTs interrupt level. This parameter is passed by value.

cpu_type - the CPU type as defined in "DDP.H." This parameter is passed by value.

eom_value - the End-Of-Message (EOM) character value. Values of 0 thru 255 are supported as valid EOM characters. To disable this feature, use a value greater than 255. This parameter is passed by value.

Ser8250InitHw returns:

DDP_ERR_NOERROR (0) - No error
DDP_ERR_BADARG1 (11) - Invalid value for chan
DDP_ERR_BADARG1 (14) - Invalid value for buf_size

Ser8250InitHw calls should use the following format:

```
unsigned int ercode;            /* error code returned */
unsigned int chan = 0;          /* initialize channel 0 */
unsigned int uart = 0x3f8;      /* uart mapped at 0x3f8 */
unsigned char buffer[100 * 2]; /* space for 2 buffers */
unsigned int buf_size = 100;    /* single buffer size */
unsigned int mode = 0x1c3;      /* 9600 baud, no parity, 1 stop bit,
                                   8 data bits, no handshaking */
unsigned int vector = 0x0c;     /* channel is on interrupt vector
                                   0x0c */
unsigned int cpu_type = ZT_8910;
                                /* CPU is a ZT 8910 */
unsigned int eom_value = '\n'; /* EOM character is \n */

.

ercode = Ser8250InitHw(chan,uart,buffer,buf_size,mode,vector,
                       cpu_type,eom_value);

.
```

3) Ser8250Xmit transmits a character on a channel. The character is queued in the transmit buffer and transmitted as soon as the channel is ready to transmit data. Three parameters are required for input:

chan - the channel number used to transmit (0-7). This parameter is passed by value.
ch - the character to transmit. This parameter is passed by value.
timeout - the value to use for a timeout if the transmit buffer is full. If zero, this procedure waits indefinitely until room is available in the transmit buffer. Non-zero values are used as a maximum time to wait for room in the transmit buffer. This value depends on CPU speed. This parameter is passed by value.

Ser8250Xmit returns:

DDP_ERR_NOERROR (0) - No error
DDP_ERR_TIMEOUT (3) - Timeout waiting for room in the transmit buffer
DDP_ERR_LINE_ERROR (4) - A line error has occurred on this channel

Ser8250Xmit calls should use the following format:

```
unsigned int ercode;          /* error code returned */
unsigned int chan = 0;        /* transmit to channel 0 */
unsigned int ch = 'A';        /* transmit a 'A' */
unsigned int timeout = 0;     /* no timeout */
.
.
ercode = Ser8250Xmit(chan,ch,timeout);
.
.
```

4) Ser8250Recv receives a character from a channel. The character is taken from the receive buffer. Three parameters are required - two for input (chan and timeout) and one for output (ch):

chan - the channel number from which you receive (0-7). This parameter is passed by value.
ch - the variable in which to return the received character. This parameter is passed by reference.
timeout - the value to use for a timeout if the receive buffer is empty. If zero, this procedure waits indefinitely until a character is available. Non-zero values are used as a maximum time to wait for a character to be available. This value depends on CPU speed. This parameter is passed by value.

Ser8250Recv returns:

DDP_ERR_NOERROR (0) - No error
DDP_ERR_TIMEOUT (3) - Timeout waiting for a character
DDP_ERR_LINE_ERROR (4) - A line error has occurred on this channel

Ser8250Recv calls should use the following format:

```
unsigned int ercode;          /* error code returned */
unsigned int chan = 0;        /* receive from channel 0 */
unsigned int ch;              /* character returned */
unsigned int timeout = 0;     /* no timeout */
.
.
ercode = Ser8250Recv(chan,&ch,timeout);
.
.
```

5) Ser8250Status checks a channel's line status. Two parameters are required - one for input (chan) and one for output (status):

chan - the channel number whose status you are checking (0-7). This parameter is passed by value.

status - the variable in which to return the channel status. The table below describes the bit definitions for this parameter. This parameter is passed by reference.

bit(s)	description
15-13	Reserved
12	CTS active
11-9	Reserved
8	CTS has changed
7-6	Reserved
5	Software receive buffer overrun
4	Received break indicator
3	Receive framing error
2	Receive parity error
1	UART receive buffer overrun
0	Reserved

Ser8250Status returns:

DDP_ERR_NOERROR (0) - No error

Ser8250Status calls should use the following format:

```
unsigned int ercode;          /* error code returned */
unsigned int chan = 0;        /* read status from channel 0 */
unsigned int status;          /* status returned */
.
.
ercode = Ser8250Status(chan,&status);
.
.
```

6) Ser8250RecvChk determines the number of characters available in the receive buffer for a channel. Two parameters are required - one for input (chan) and one for output (count):

chan - the number of the channel whose receive buffer is to be checked (0-7). This parameter is passed by value.

count - the variable in which to return the number of characters
available in the receive buffer. This parameter is passed by
reference.

Ser8250RecvChk returns:

DDP_ERR_NOERROR (0) - No error

Ser8250RecvChk calls should use the following format:

```
unsigned int ercode;          /* error code returned */
unsigned int chan = 0;        /* check number of characters in
                              receive buffer for channel 0 */
unsigned int count;           /* number of characters in receive
                              buffer returned */
.
.
ercode = Ser8250RecvChk(chan,&count);
.
.
```

7) Ser8250RecvEomChk determines the number of EOM characters available in the receive buffer for a channel. Two parameters are required - one for input (chan) and one for output (count):

chan - the number of the channel whose receive buffer is to be
checked (0-7). This parameter is passed by value.
count - the variable in which to return the number of EOM characters
available in the receive buffer. This parameter is passed by
reference.

Ser8250RecvEomChk returns:

DDP_ERR_NOERROR (0) - No error

Ser8250RecvEomChk calls should use the following format:

```
unsigned int ercode;          /* error code returned */
unsigned int chan = 0;        /* check number of EOM characters in
                              receive buffer for channel 0 */
unsigned int count;           /* number of EOM characters in receive
                              buffer returned */
.
.
ercode = Ser8250RecvEomChk(chan,&count);
.
.
```

8) Ser8250XmitChk determines the number of characters free in the transmit buffer for a channel. Two parameters are required - one for input (chan) and one for output (count):

chan - the number of the channel whose transmit buffer is to be
checked (0-7). This parameter is passed by value

count - the variable in which to return the number of characters free
in the transmit buffer. This parameter is passed by reference.

Ser8250XmitChk returns:

DDP_ERR_NOERROR (0) - No error

Ser8250XmitChk calls should use the following format:

```
unsigned int ercode;          /* error code returned */
unsigned int chan = 0;        /* check number of free bytes in
                              transmit buffer for channel 0 */
unsigned int count;           /* number of free bytes in transmit
                              buffer returned */
.
.
ercode = Ser8250XmitChk(chan,&count);
.
.
```

9) Ser8250BuffFlush flushes the transmit and/or receive buffers for a channel. Three parameters are required for input:

chan - the channel number to flush for buffers (0-7). This parameter is passed by value.
in - specifies whether or not to flush the receive buffer. A non-zero value causes the receive buffer to be flushed. This parameter is passed by value.
out - specifies whether or not to flush the transmit buffer. A non-zero value causes the transmit buffer to be flushed. This parameter is passed by value.

Ser8250BuffFlush returns:

DDP_ERR_NOERROR (0) - No error

Ser8250BuffFlush calls should use the following format:

```
unsigned int ercode;          /* error code returned */
unsigned int chan = 0;        /* flush channel 0 buffers */
unsigned int in = 0;          /* don't flush receive buffer */
unsigned int out = 1;         /* flush transmit buffer */
.
ercode = Ser8250BuffFlush(chan,in,out);
.
.
```

10) Ser8250Exit disables all channels. No parameters are used.

Ser8250Exit returns:

DDP_ERR_NOERROR (0) - No error

Ser8250Exit should use the following format:

```
unsigned int ercode;          /* error code returned */
.
.
ercode = Ser8250Exit();
.
.
```

USING THE SER8250 DRIVER WITH MICROSOFT C

The /Gs compiler option MUST be specified for DOS systems. Otherwise, stack overflows can occur during interrupt service routines.

USING THE SER8250 DRIVER WITH BORLAND'S TURBO C OR TURBO C++

The -N compiler option MUST NOT be specified for DOS systems. Otherwise, stack overflows can occur during interrupt service routines.

USING THE SER8250 DRIVER WITH MICROSOFT'S QUICK BASIC

The Ser8250 driver is not compatable with the Quick BASIC environment. It is, however, compatable with stand-alone Quick BASIC programs provided that the buffer is allocated using the STATIC metacommand. For example:

```
rem $include: 'Ser8250.bas'

rem $static
dim buffer%(100)

ercode% = Ser8250InitSw(version%)
ercode% = Ser8250InitHw%(0,&H3F8,buffer%(0),100,&H1C3,&H0C,ZT_8901)
```

THE SER8250 SOURCE CODE FILES

The Ser8250 driver uses a generic interrupt driven serial handler (SERIAL.C). This generic handler relies on a collection of UART specific macro definitions. These macro definitions are located in SER8250.C.

APPENDIX D.1: GPS DATA - MAPS MADE DURING DYNAMIC TESTING

Dynamic Testing Performed
from October 22 - 28, 1992

APPENDIX D.1: GPS DATA - MAPS MADE DURING DYNAMIC TESTING

The following tables summarize the test dates (Table D.1-1) and the terrain (Table D.1-2) used for the dynamic testing of the GPS systems. Figures D.1-1 through D.1-25 present the mapping of the GPS data taken during the dynamic testing.

Table D.1-2. Summary of Dynamic Test Dates

Terrain Description	Dates Testing Performed
City Driving	10/22 and 10/25/92
Mountain Driving	10/25 and 10/26/92
Interstate Highway Driving	10/27 and 10/28/92
Rural Highway Driving	10/27 and 10/28/92
Canyon Driving	10/27 and 10/28/92

Table D.1-2. Summary of Terrain Used for Dynamic Testing

Terrain Description	Road Sections Used for Testing
City Driving	Central Avenue in Albuquerque, NM — from Tramway to 7th Street
Mountain Driving	Tijeras Canyon, I-40, East and West — I-40 Tramway Exit (Albuquerque, NM) to Sedillo (NM) Exit
Interstate Highway Driving	I-25, North and South — I-25 Tramway Exit (Albuquerque, NM) to NM 285 Exit in Santa Fe, NM
Rural Highway Driving	NM 285 and NM 4 — Intersection of NM 285 and I-25 (Santa Fe, NM) to intersection (Los Alamos, NM) where LANL* Truck Route (East Jemez Road) begins (at traffic light)
Canyon Driving	LANL* Truck Route (East Jemez Road) in Los Alamos, NM — from traffic light to traffic light, climbing from Sandia Canyon up to the South Mesa

* Los Alamos National Laboratory

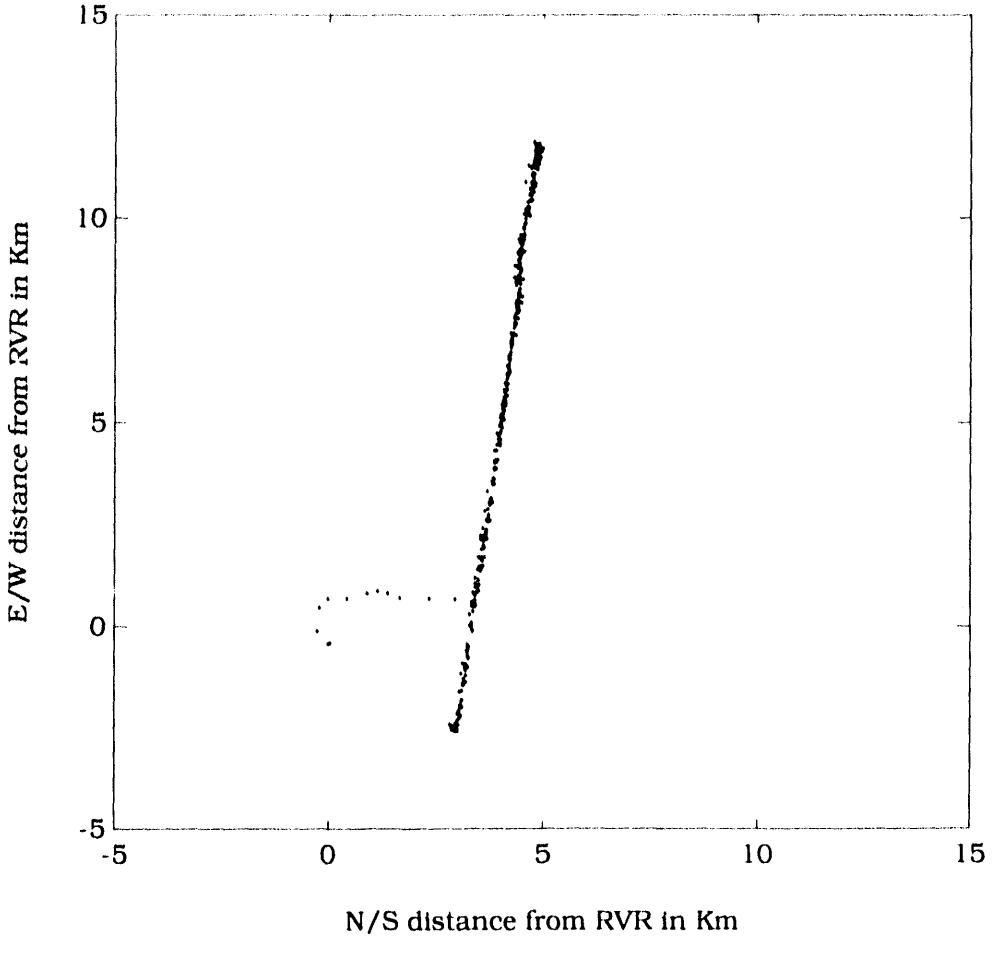


Figure D.1-1. City Driving Data, Magellan.

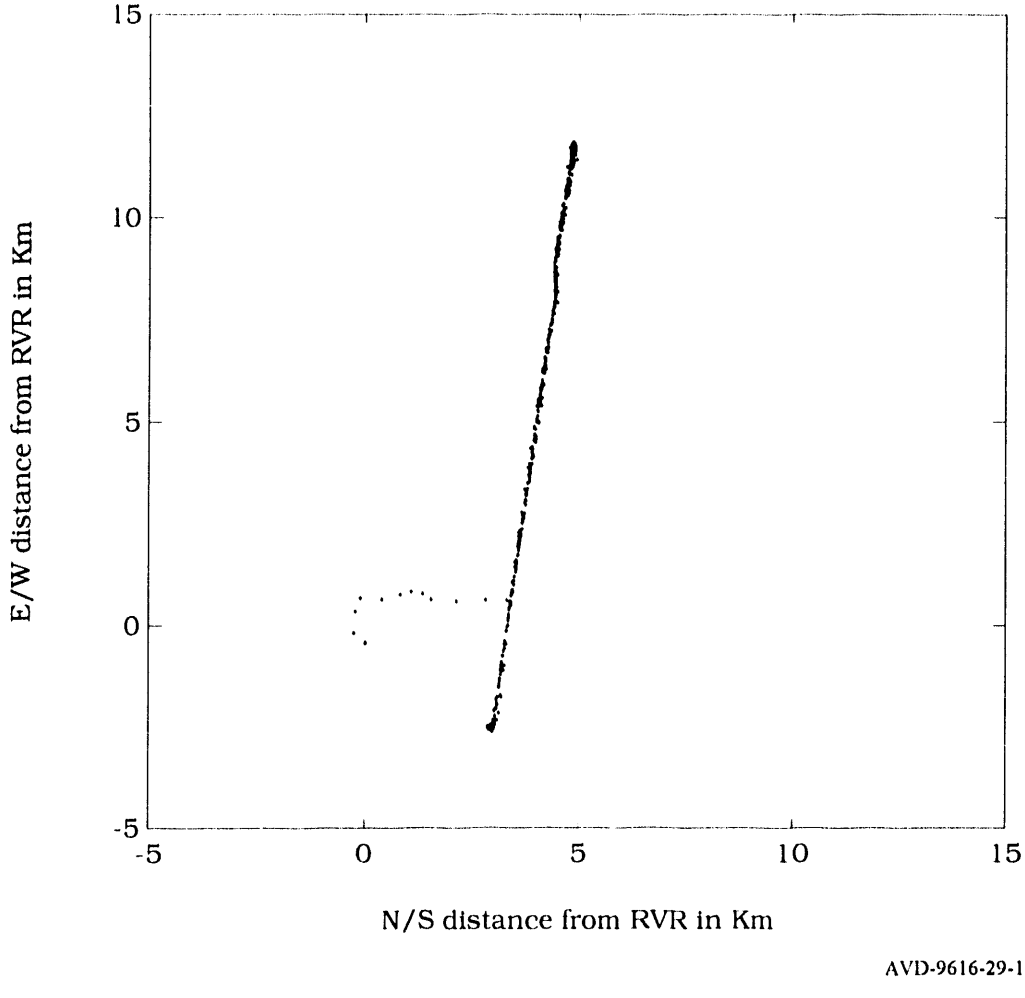


Figure D.1-2. City Driving Data, Magnavox GPS Engine.

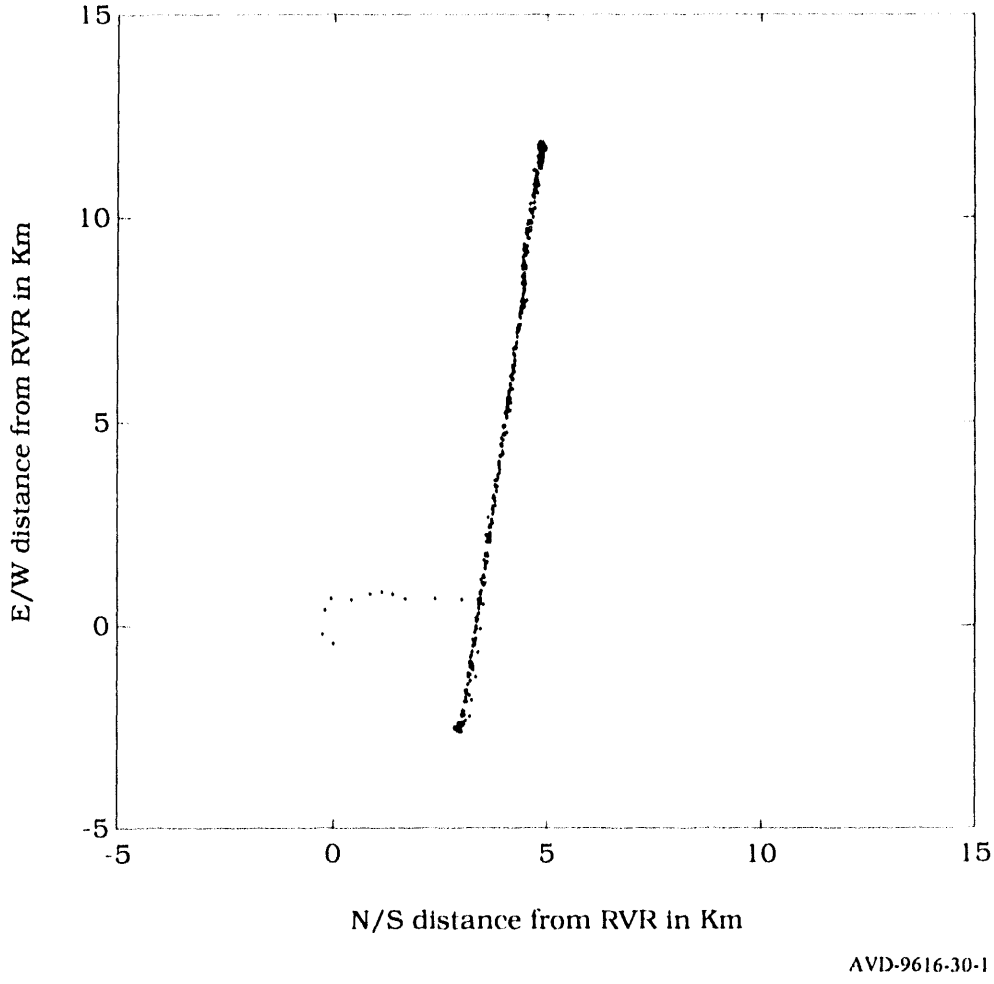


Figure D.1-3. City Driving Data, Rockwell NavCore V.

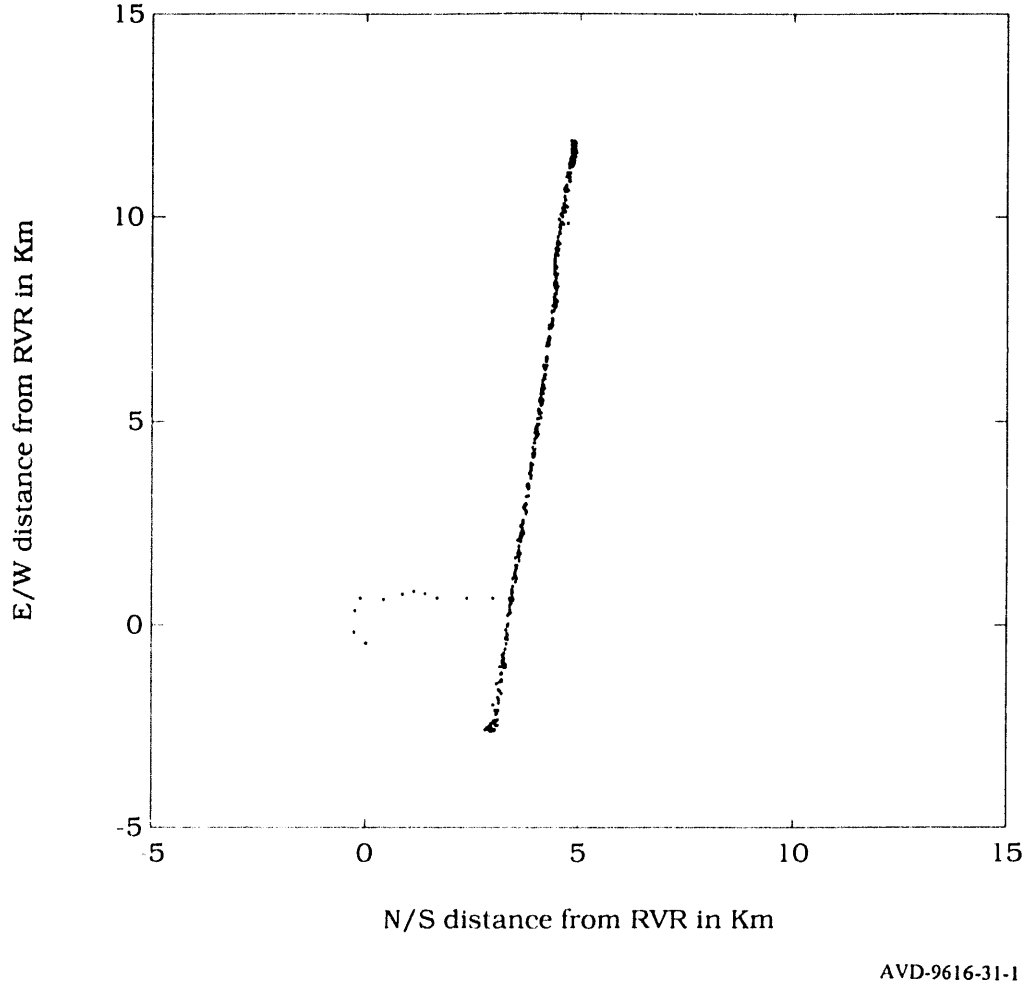


Figure D.1-4. City Driving Data, Magnavox 6400.

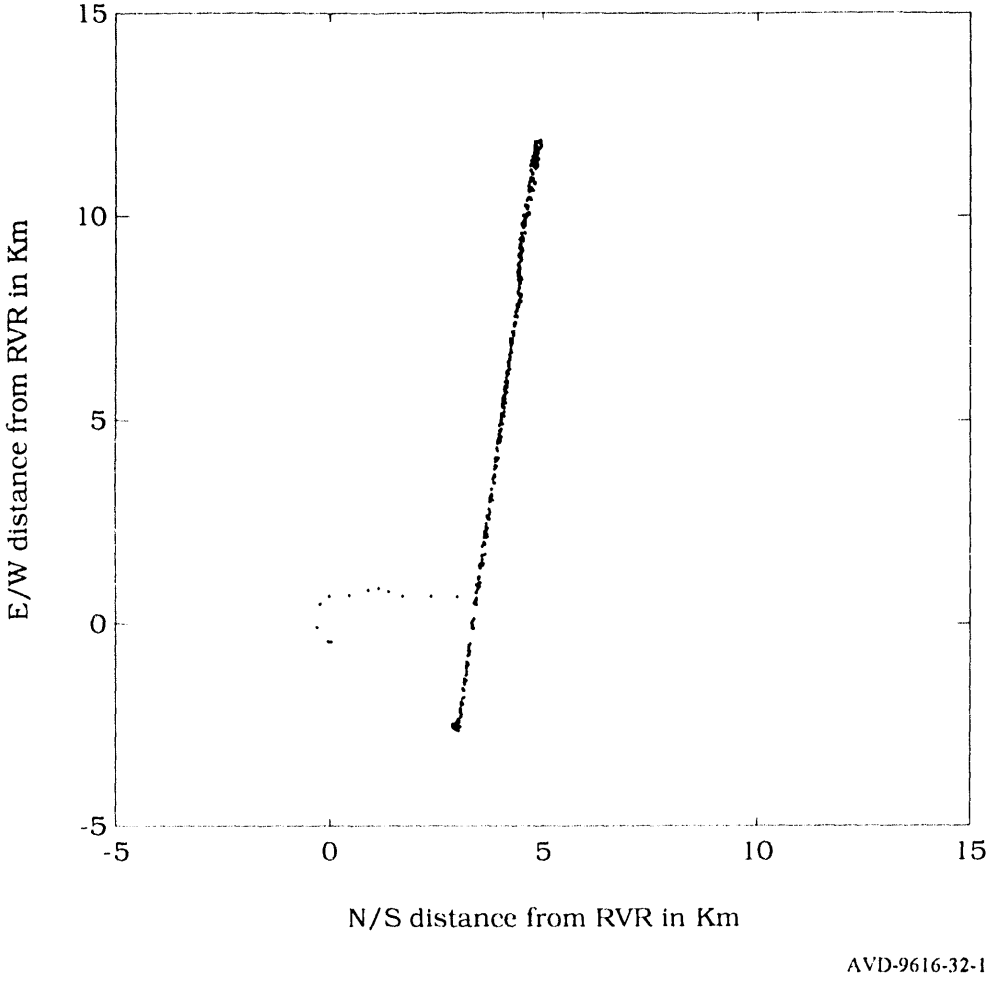


Figure D.1-5. City Driving Data, Trimble Placer.

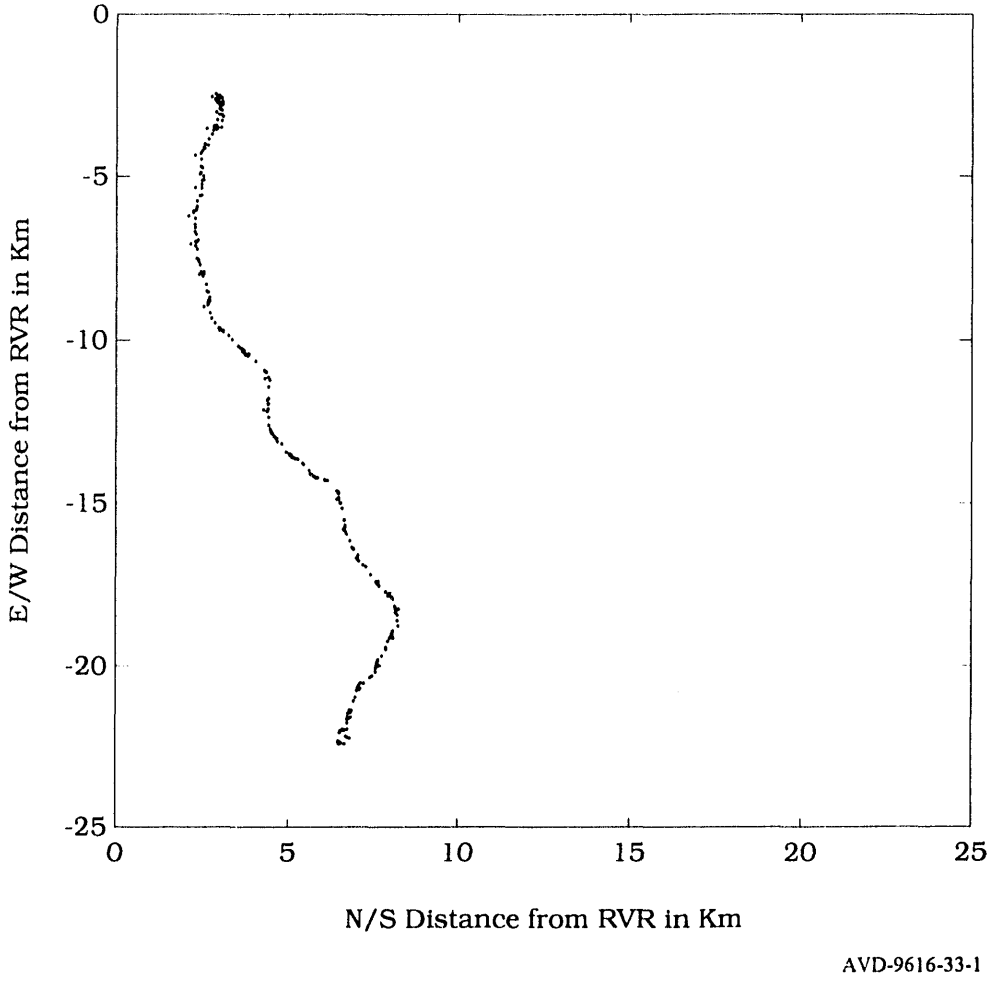


Figure D.1-6. Tijeras Canyon Data, Magellan.

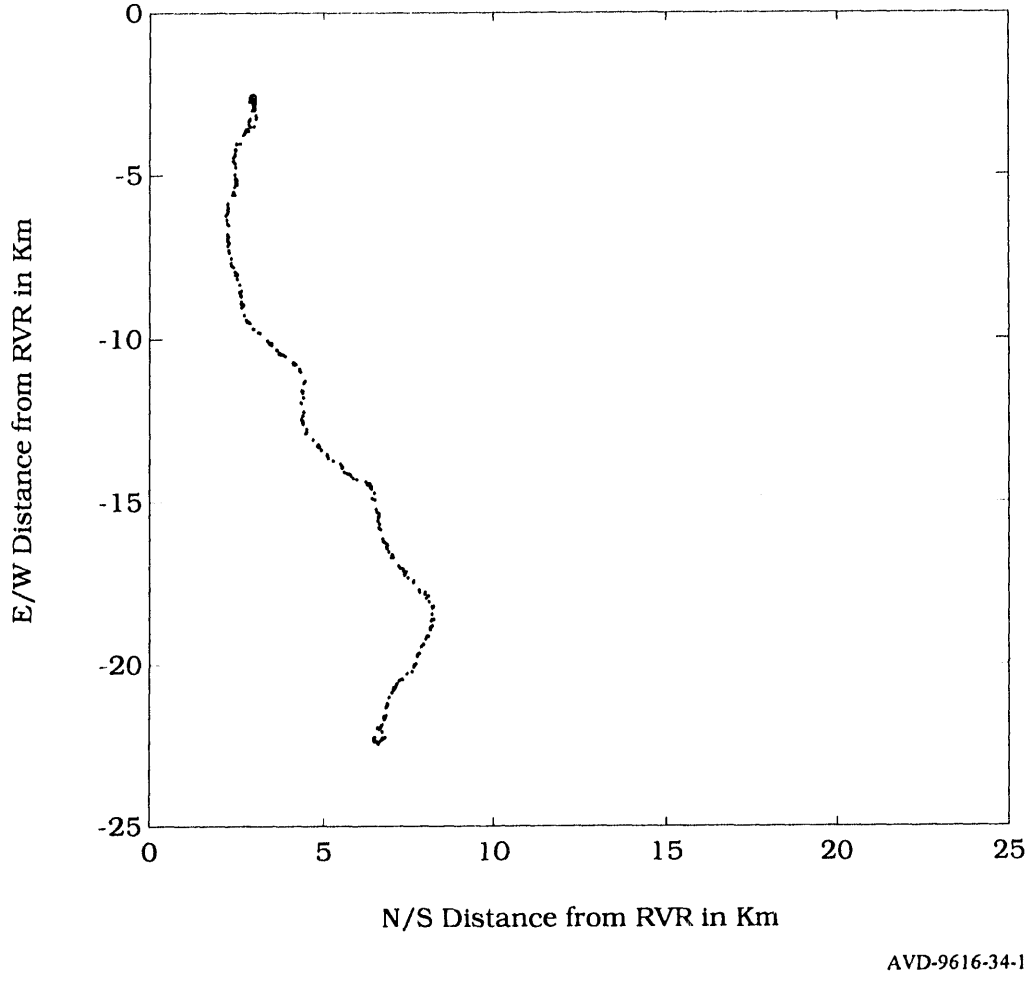


Figure D.1-7. Tijeras Canyon Data, Magnavox GPS Engine.

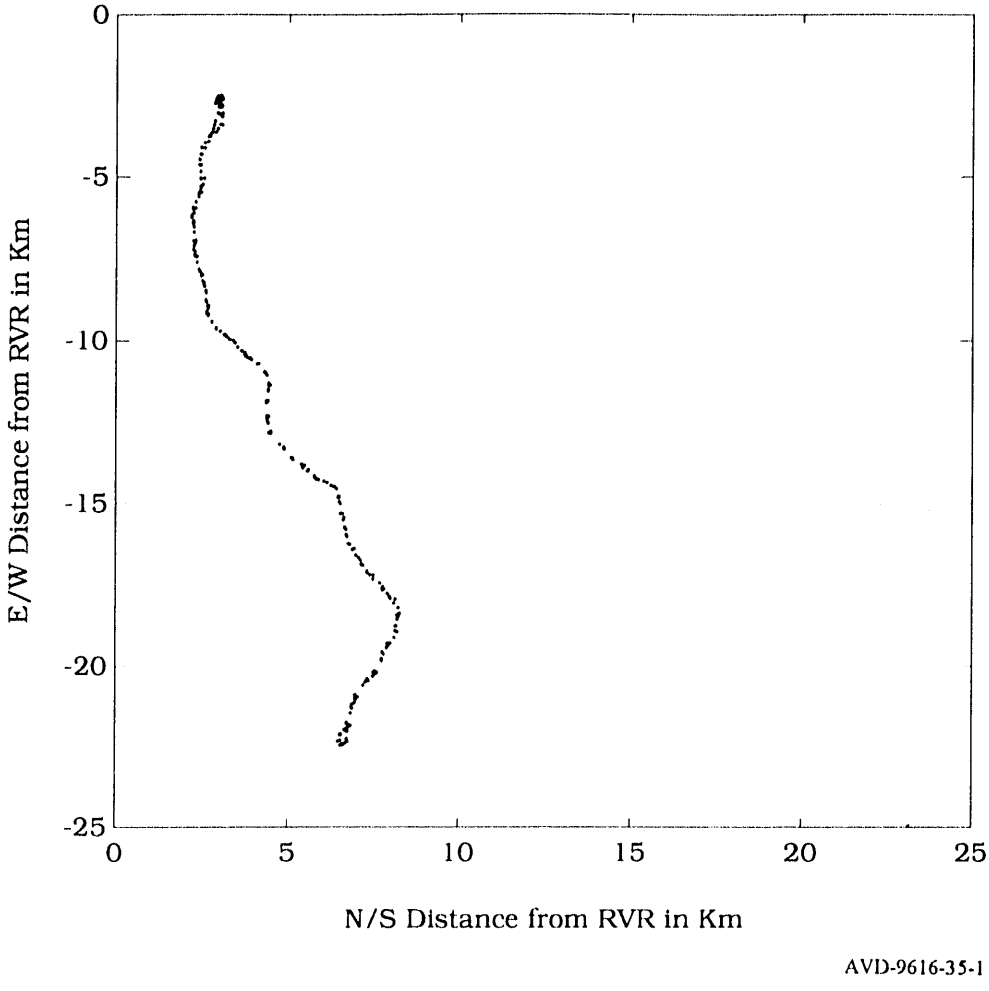


Figure D.1-8. Tijeras Canyon Data, Rockwell NavCore V.

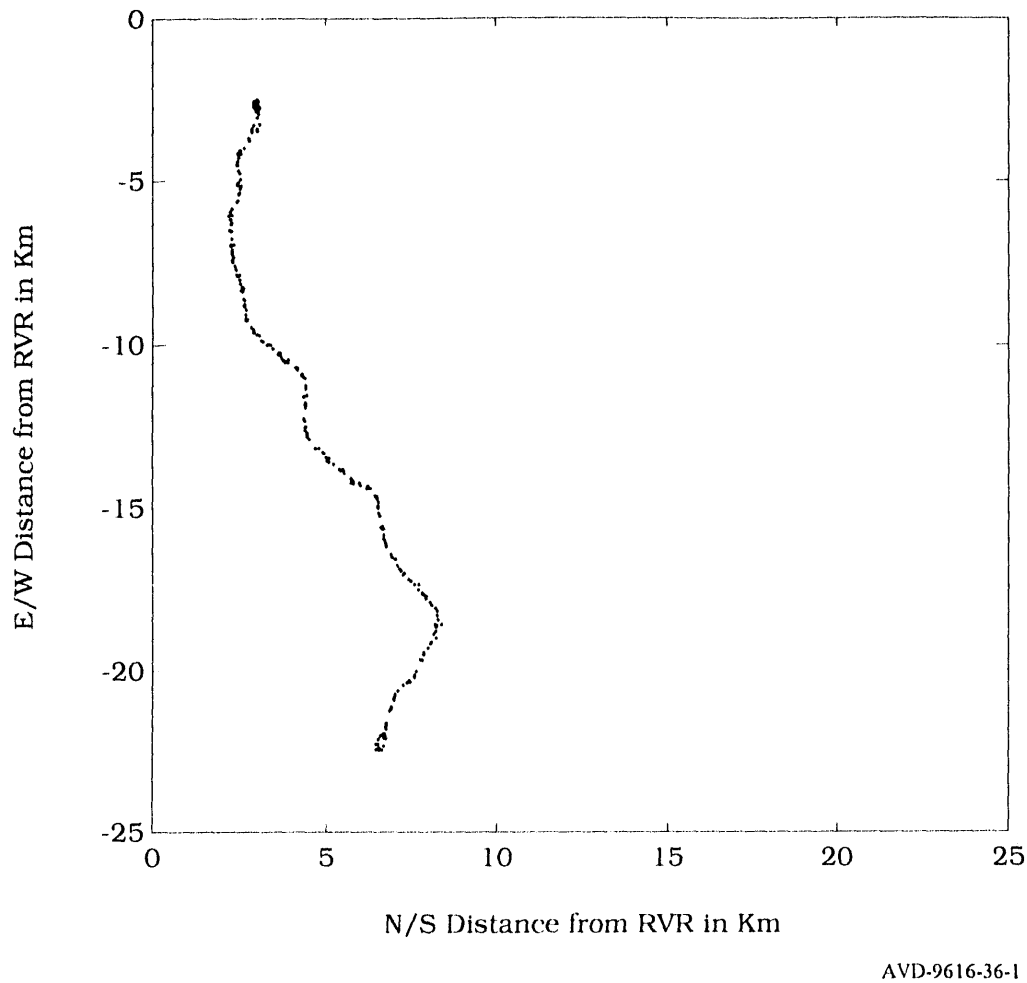


Figure D.1-9. Tijeras Canyon Data, Magnavox 6400.

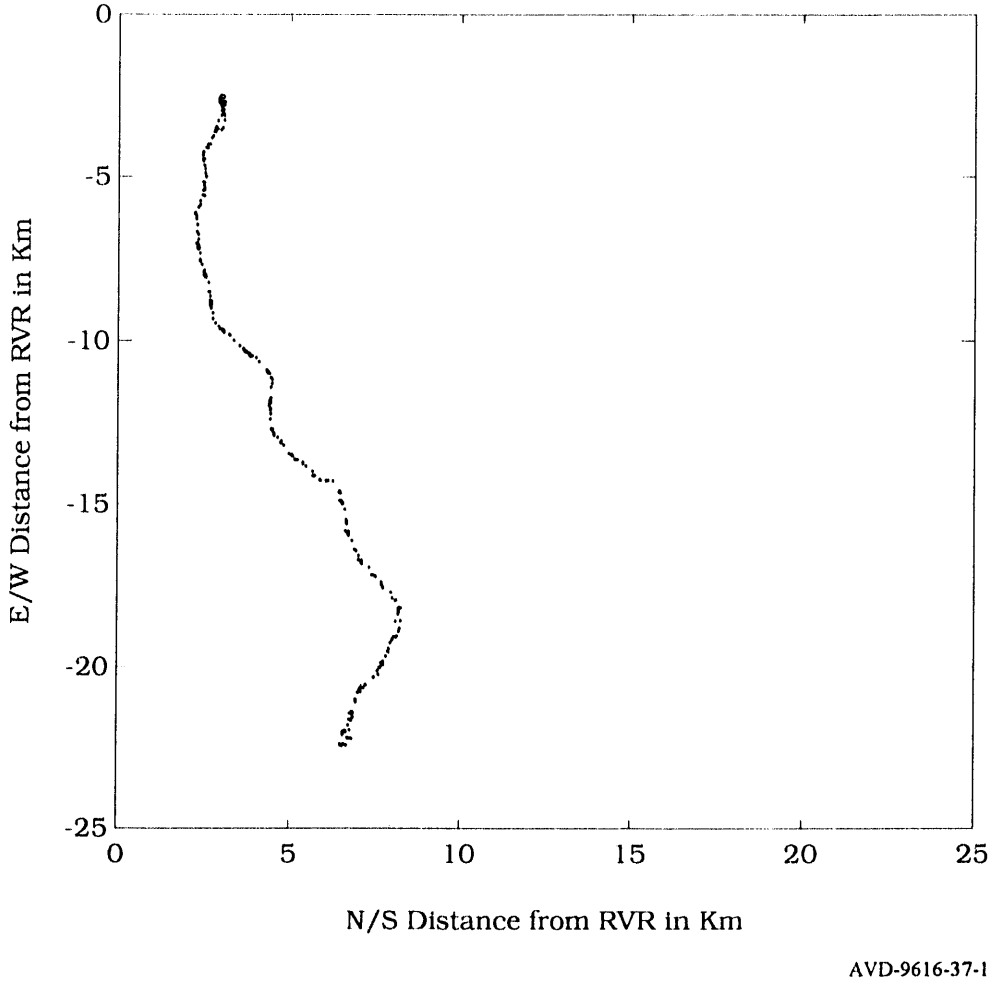


Figure D.1-10. Tijeras Canyon Data, Trimble Placer.

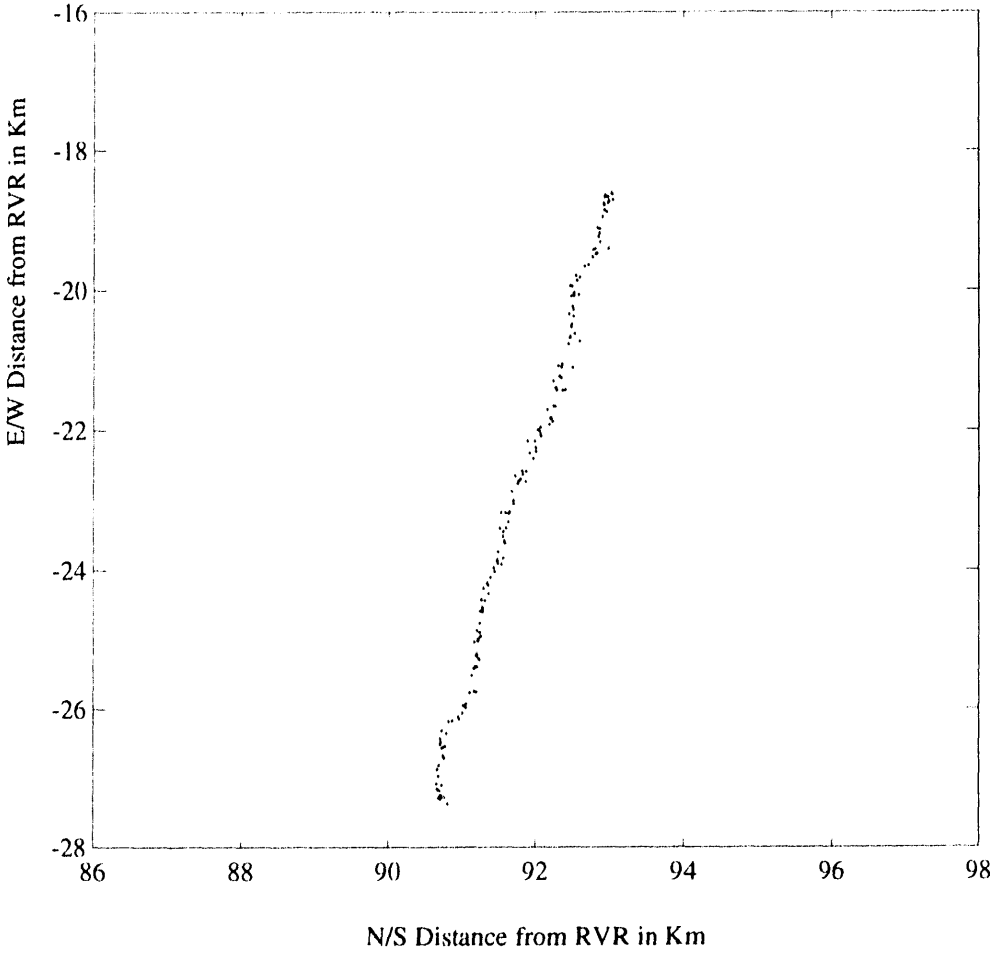


Figure D.1-11. LANL Truck Route Canyon Data, Magellan.

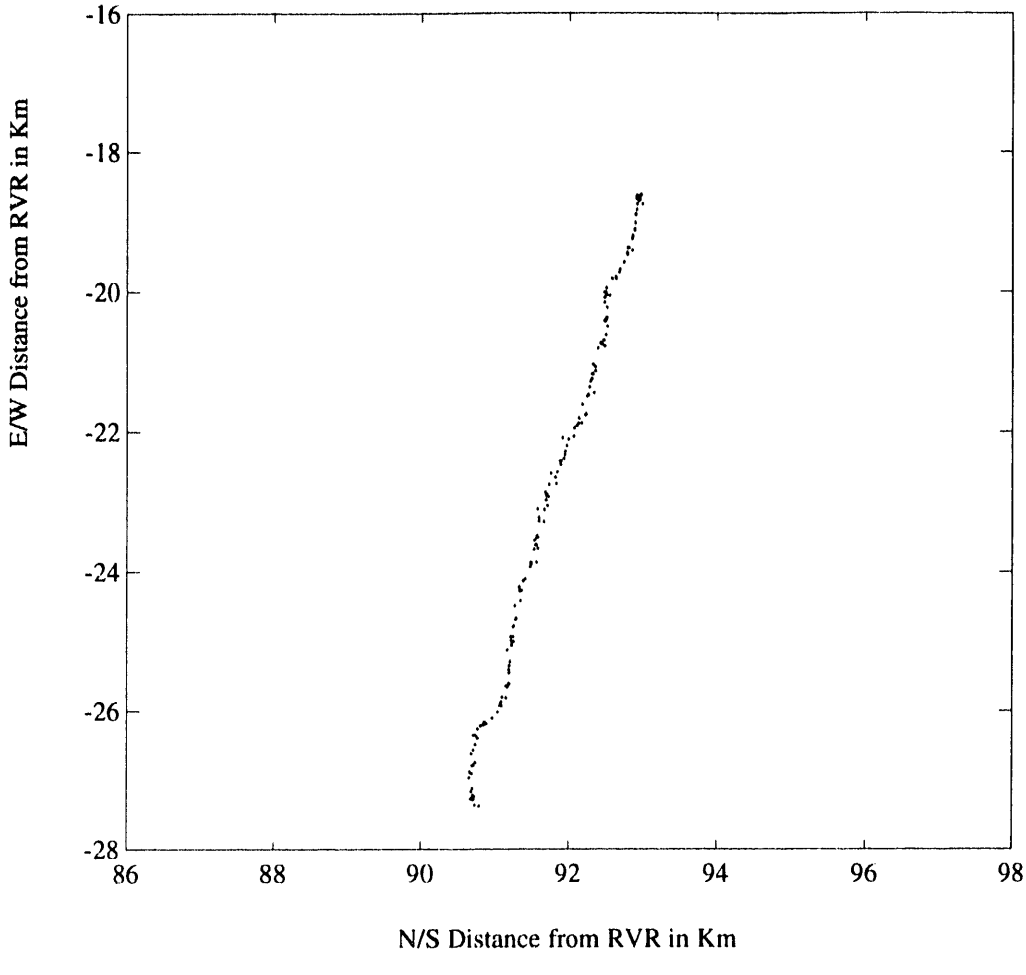


Figure D.1-12. LANL Truck Route Canyon Data, Magnavox GPS Enginc.

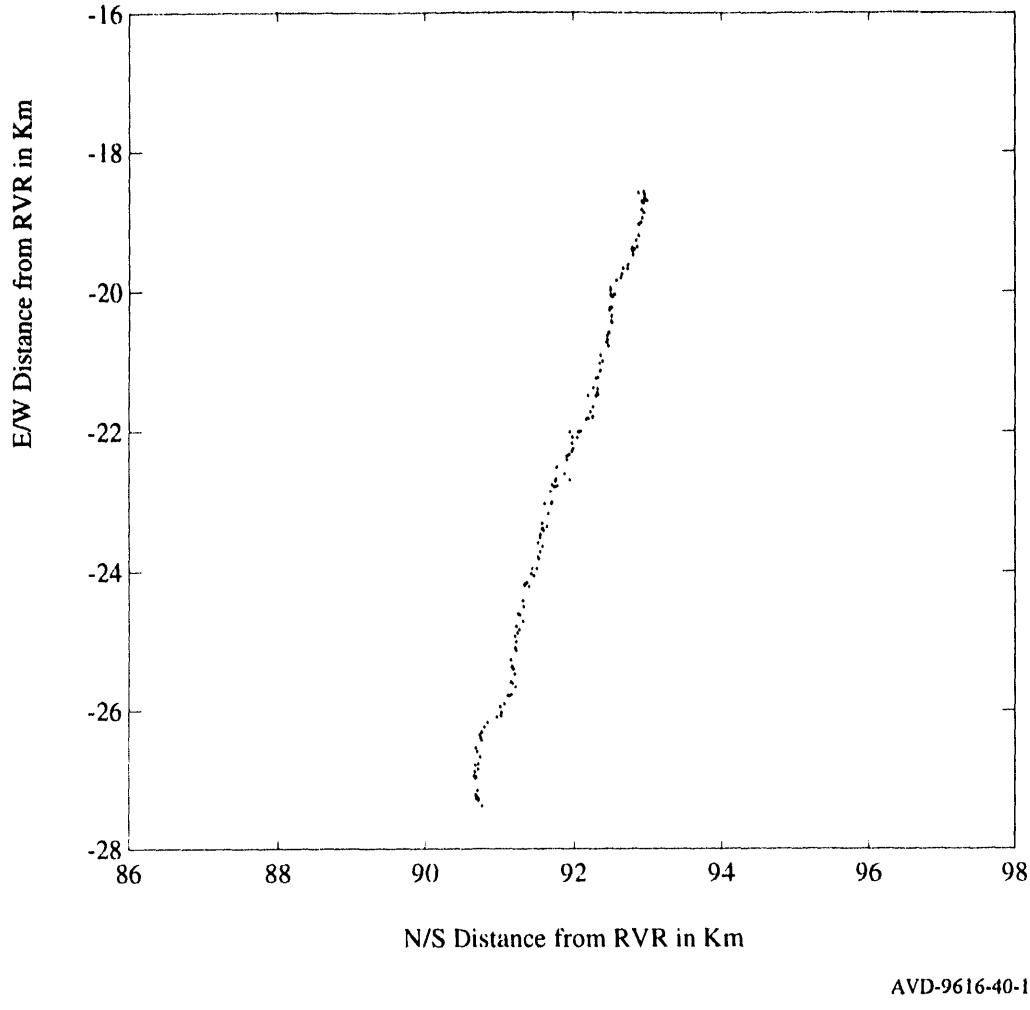


Figure D.1-13. LANL Truck Route Canyon Data, Rockwell NavCore V.

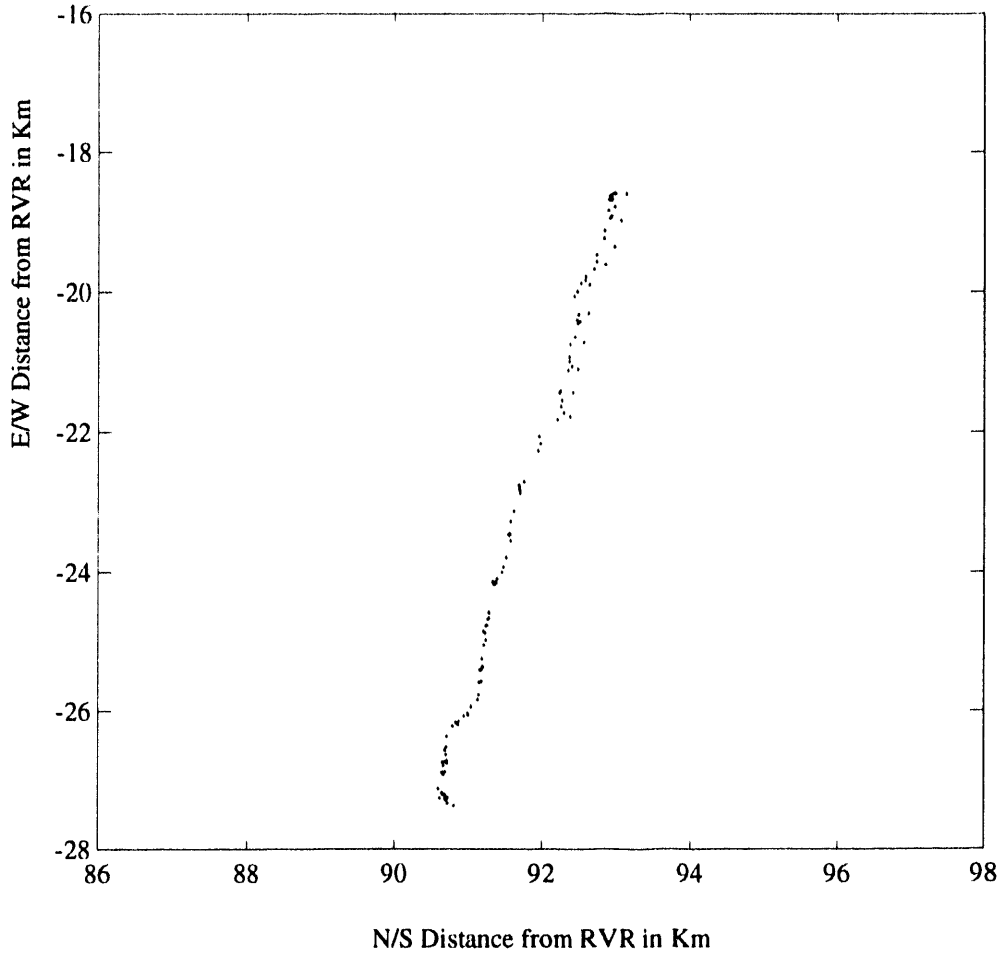


Figure D.1-14. LANL Truck Route Canyon Data, Magnavox 6400.

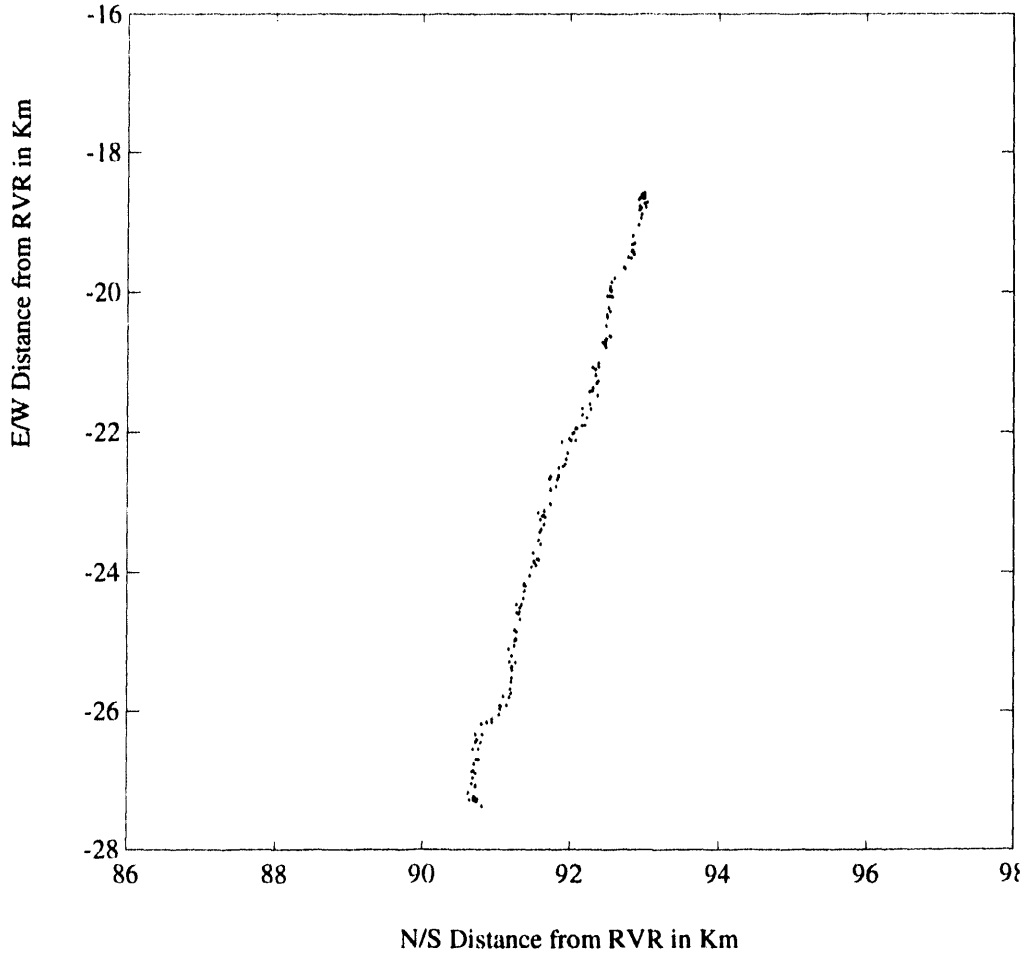


Figure D.1-15. LANL Truck Route Canyon Data, Trimble Placer.

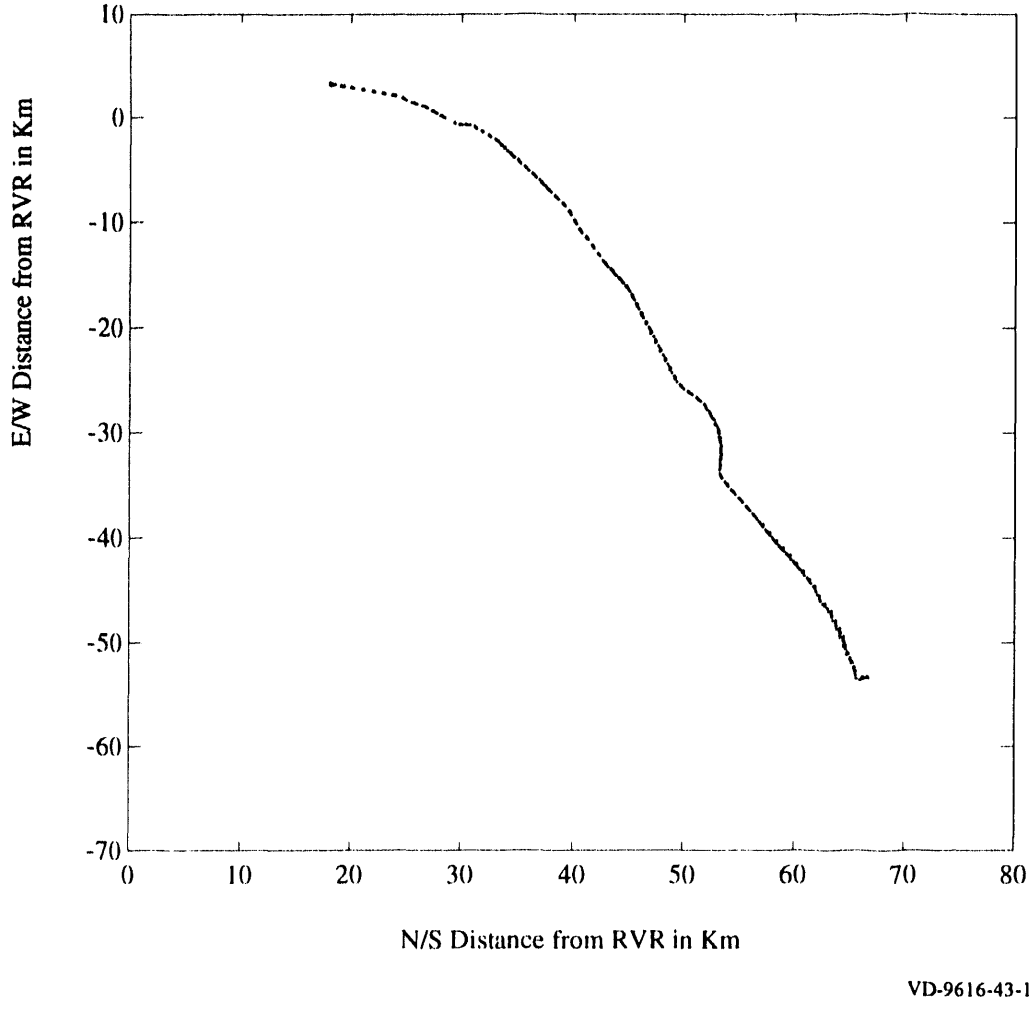


Figure D.1-16. I-25 Data, Magellan.

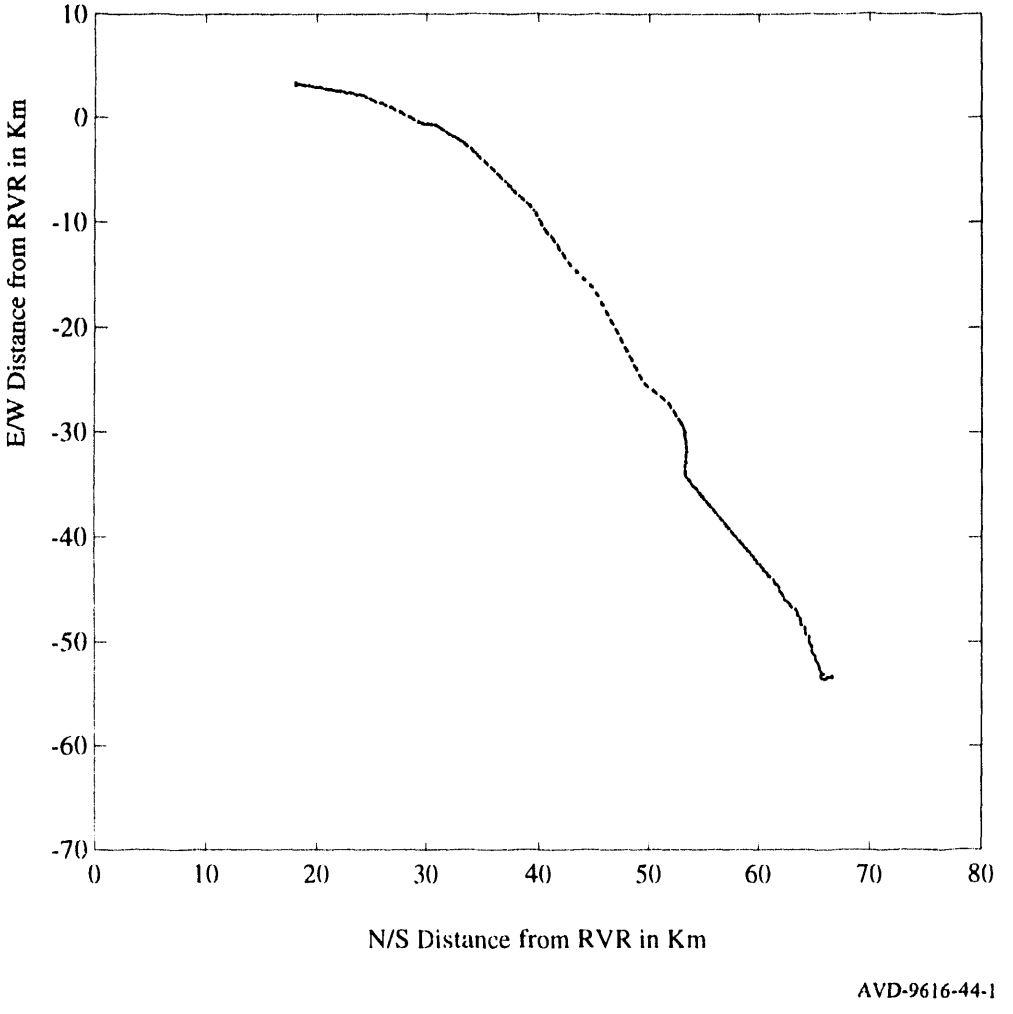


Figure D.1-17. I-25 Data, Magnavox GPS Engine.

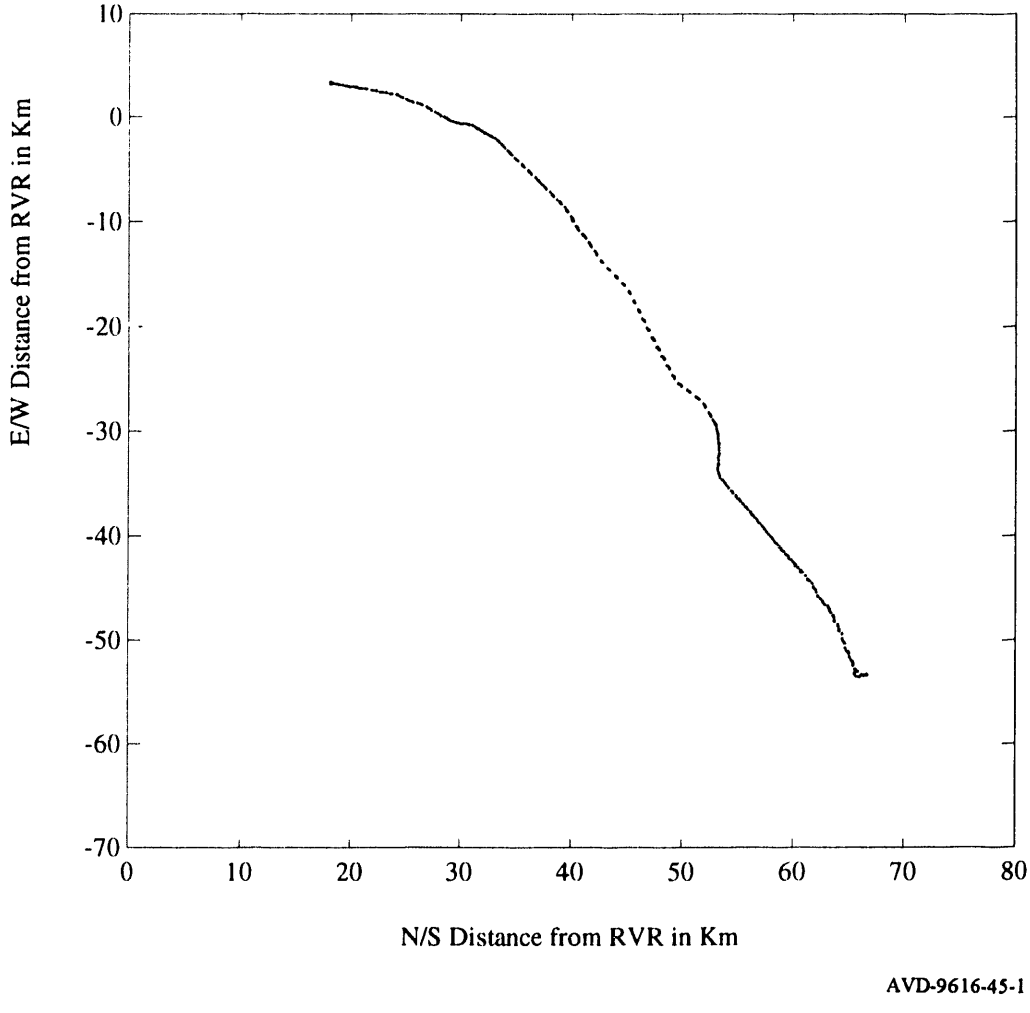


Figure D.1-18. I-25 Data, Rockwell NavCore V.

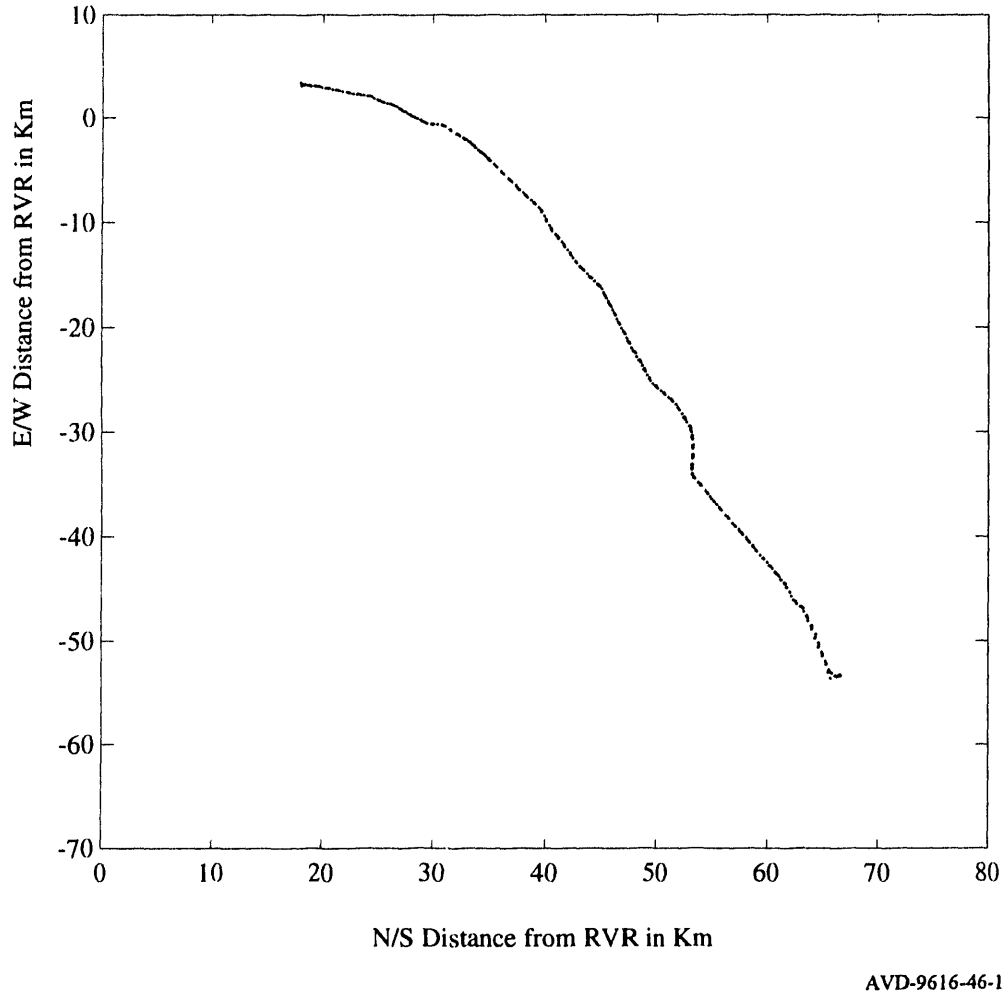


Figure D.1-19. I-25 Data, Magnavox 6400.

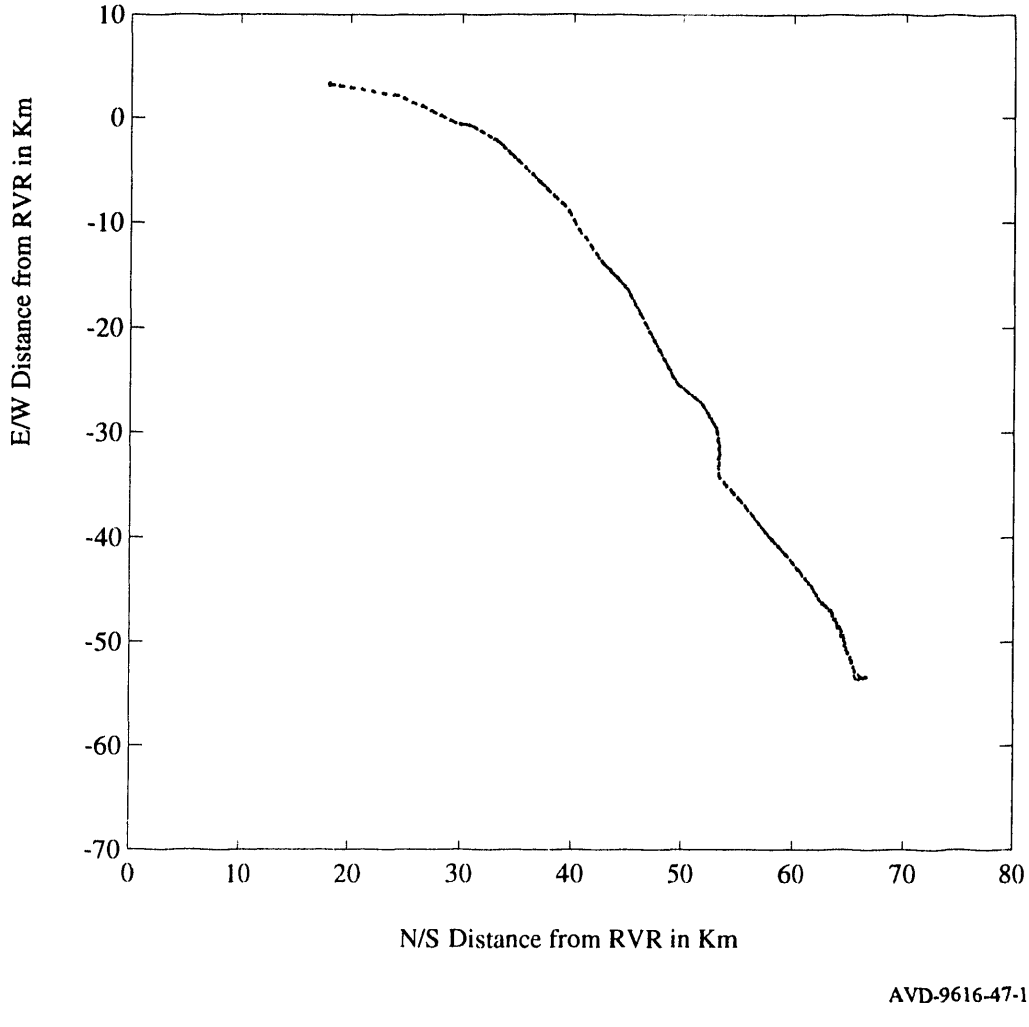


Figure D.1-20. I-25 Data, Trimble Placer.

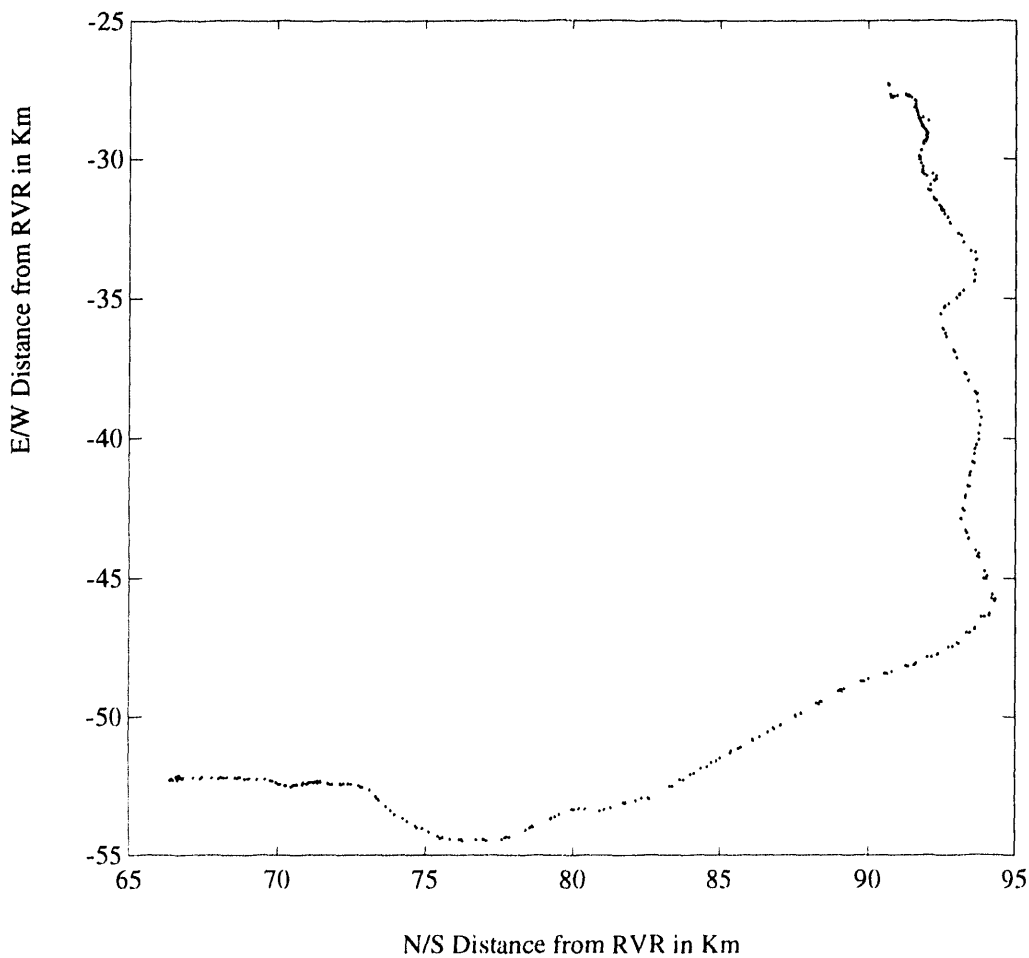


Figure D.1-21. Santa Fe to LANL Data, Magellan.

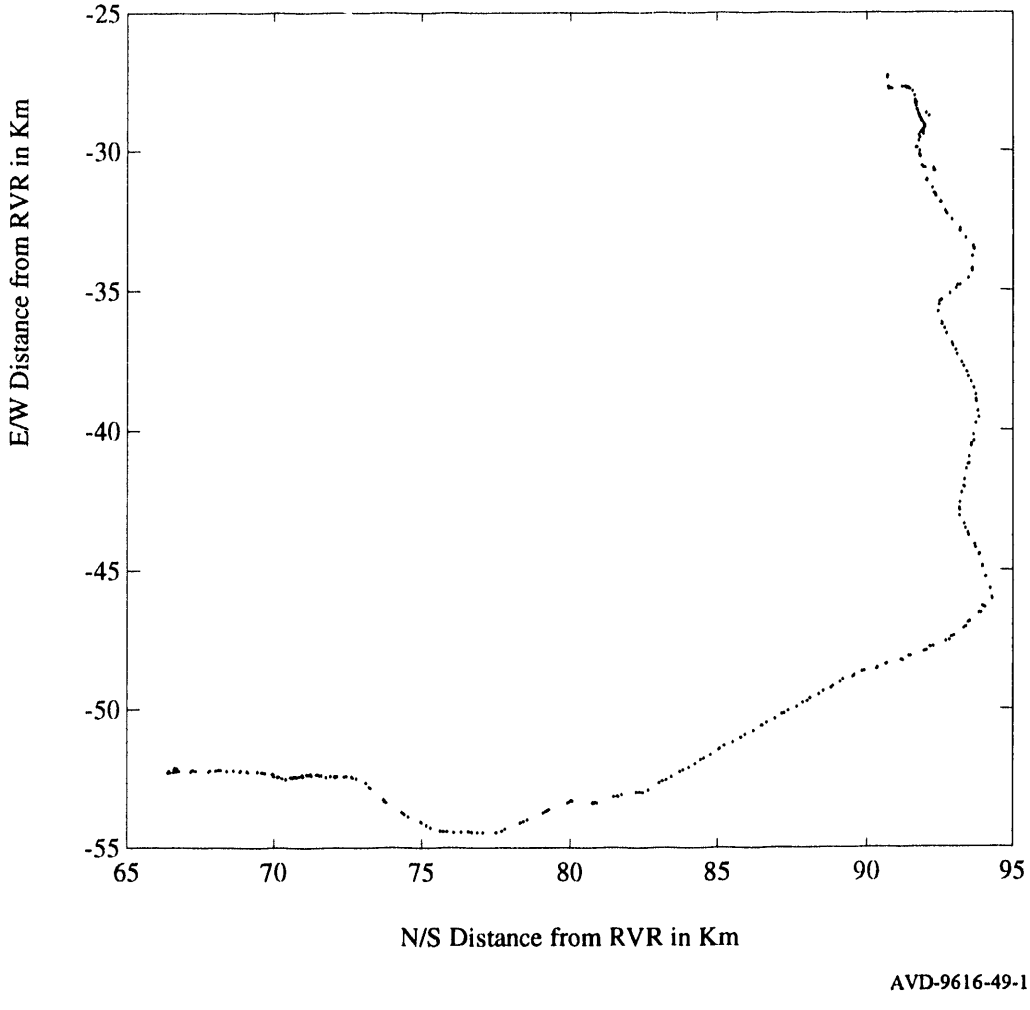


Figure D.1-22. Santa Fe to LANL Data, Magnavox GPS Engine.

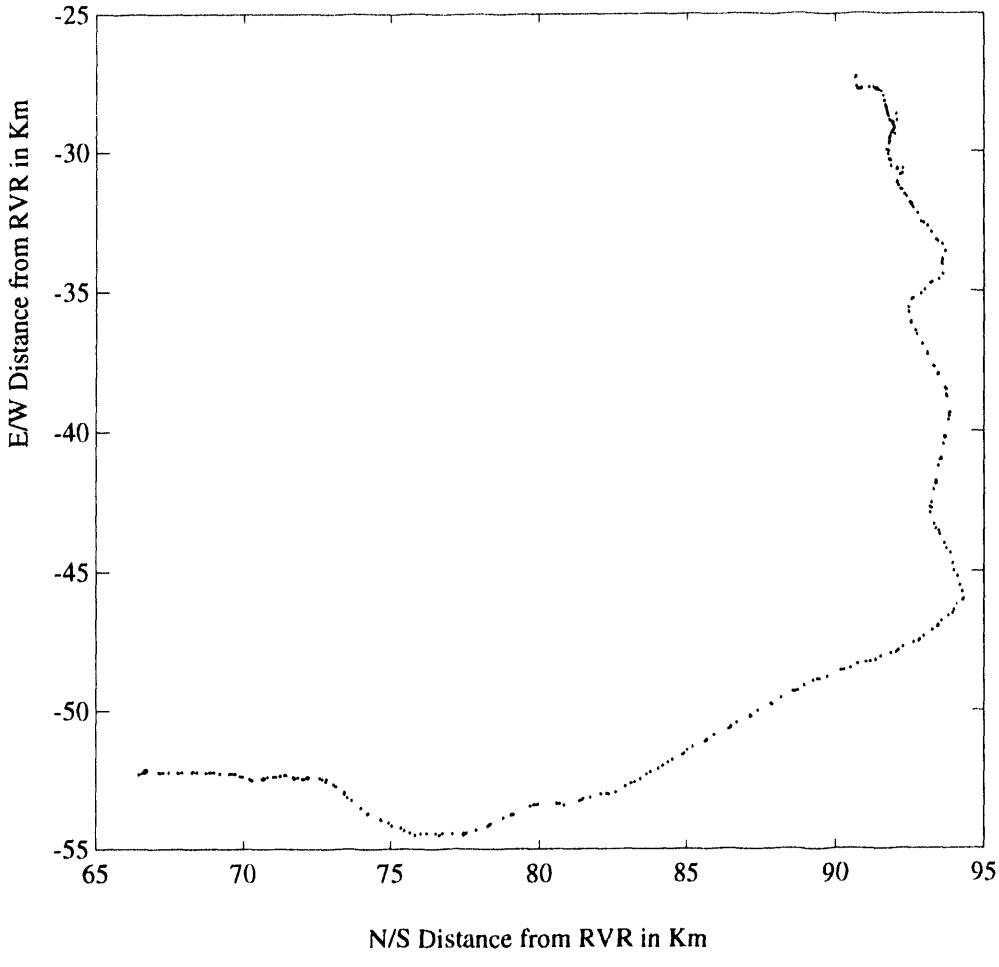


Figure D.1-23. Santa Fe to LANL Data, Rockwell NavCore V.

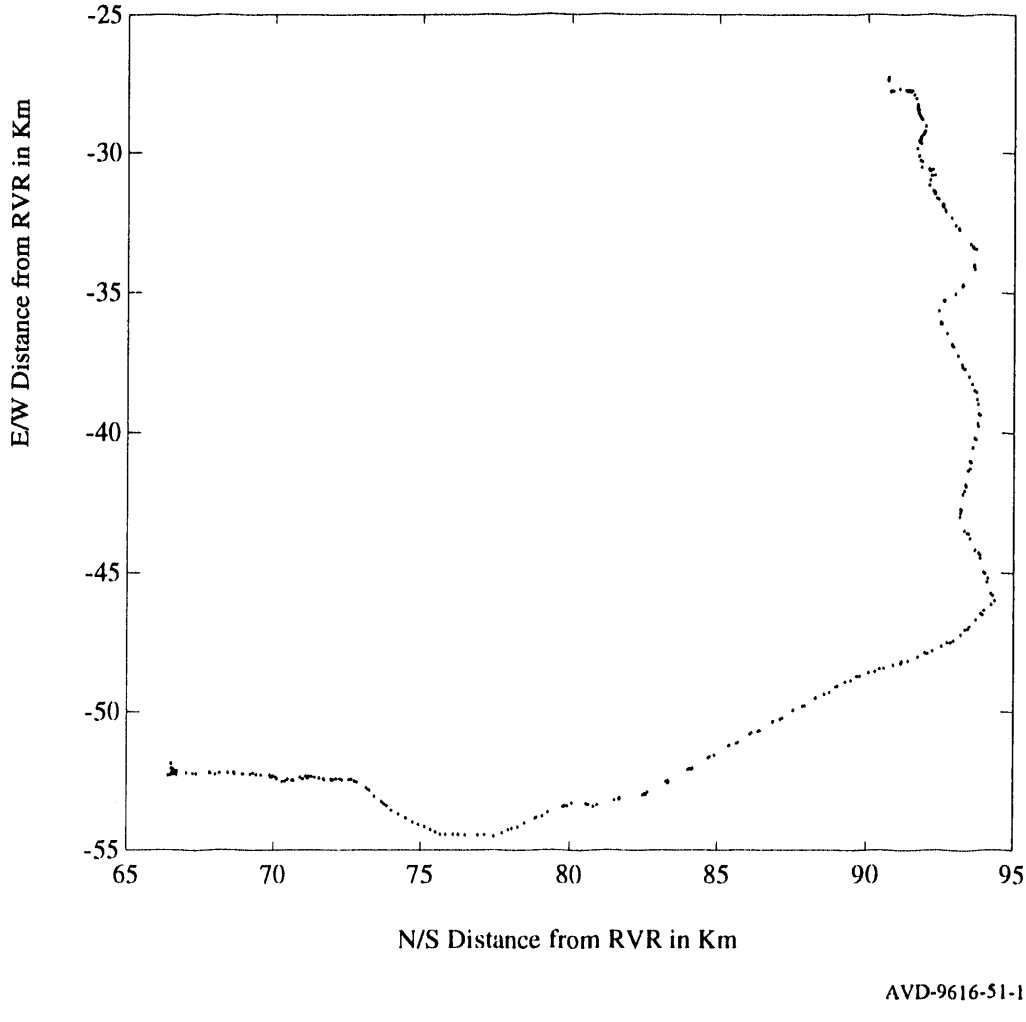


Figure D.1-24. Santa Fe to LANL Data, Magnavox 6400.

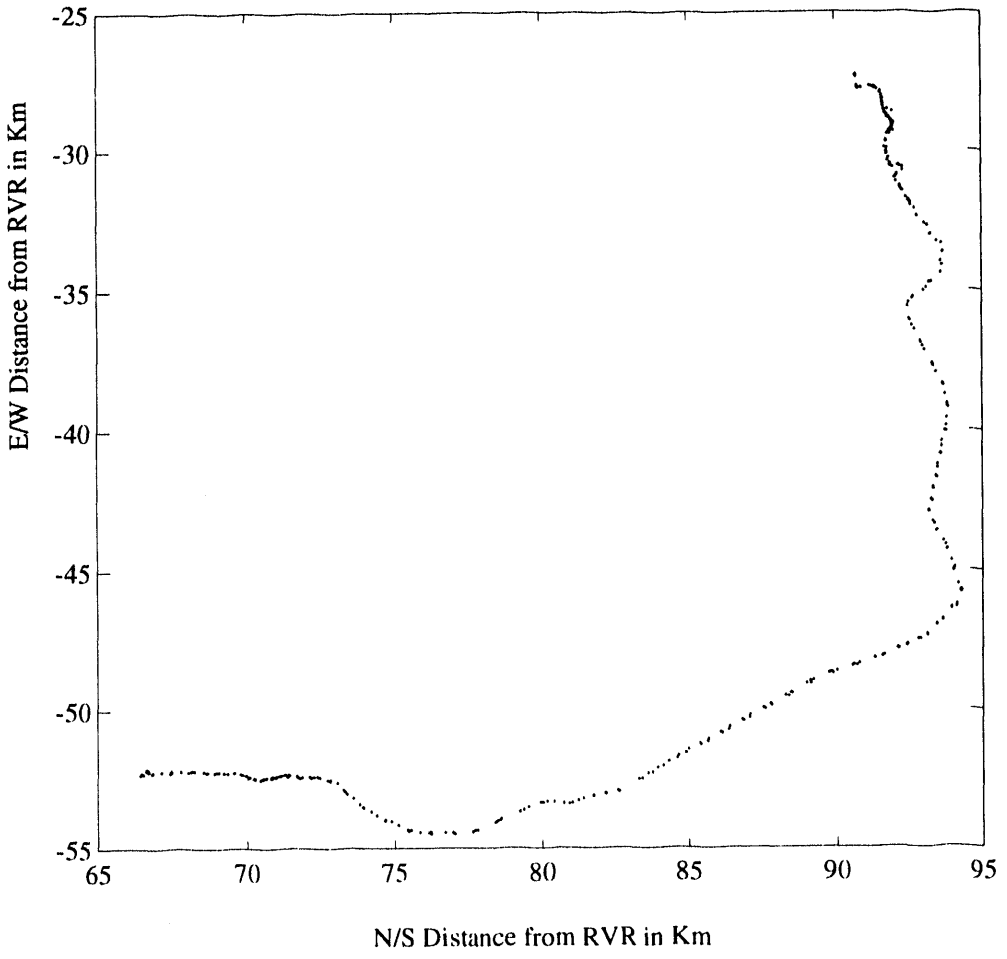


Figure D.1-25. Santa Fe to LANL Data, Trimble Placer.

APPENDIX D.2: GPS DATA - STATIC TEST NAVIGATION MODE AS A FUNCTION OF TIME

APPENDIX D.2: GPS DATA - STATIC TEST NAVIGATION MODE AS A FUNCTION OF TIME

The following pages show static data in graph form.

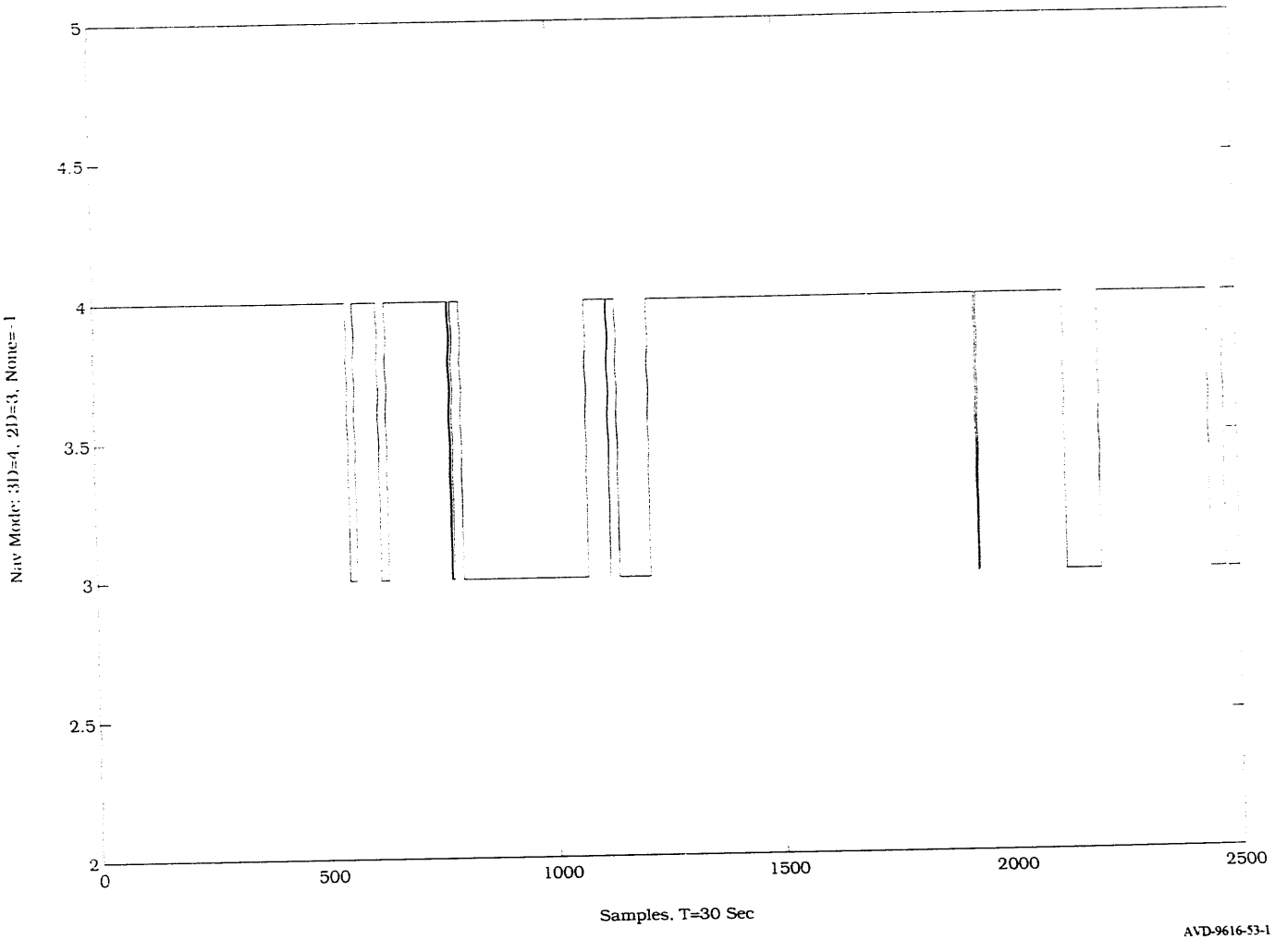


Figure D.2-1. Static Test Navigation Mode as a function of time, Magellan Navigation (A100714E.DAT).

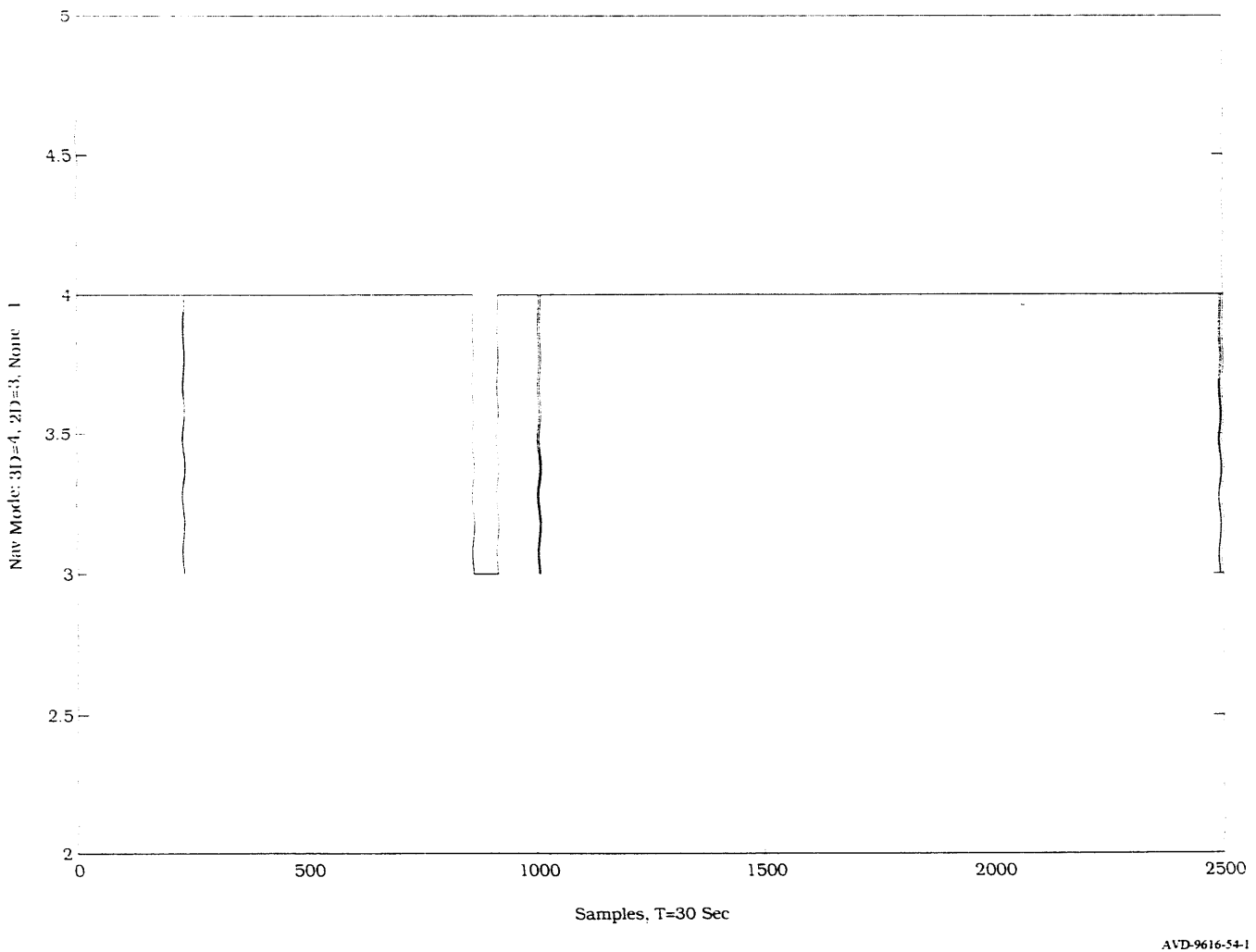


Figure D.2-2. Static Test Navigation Mode as a function of time, Magnavox GPS Engine Navigation (B100714E.DAT).

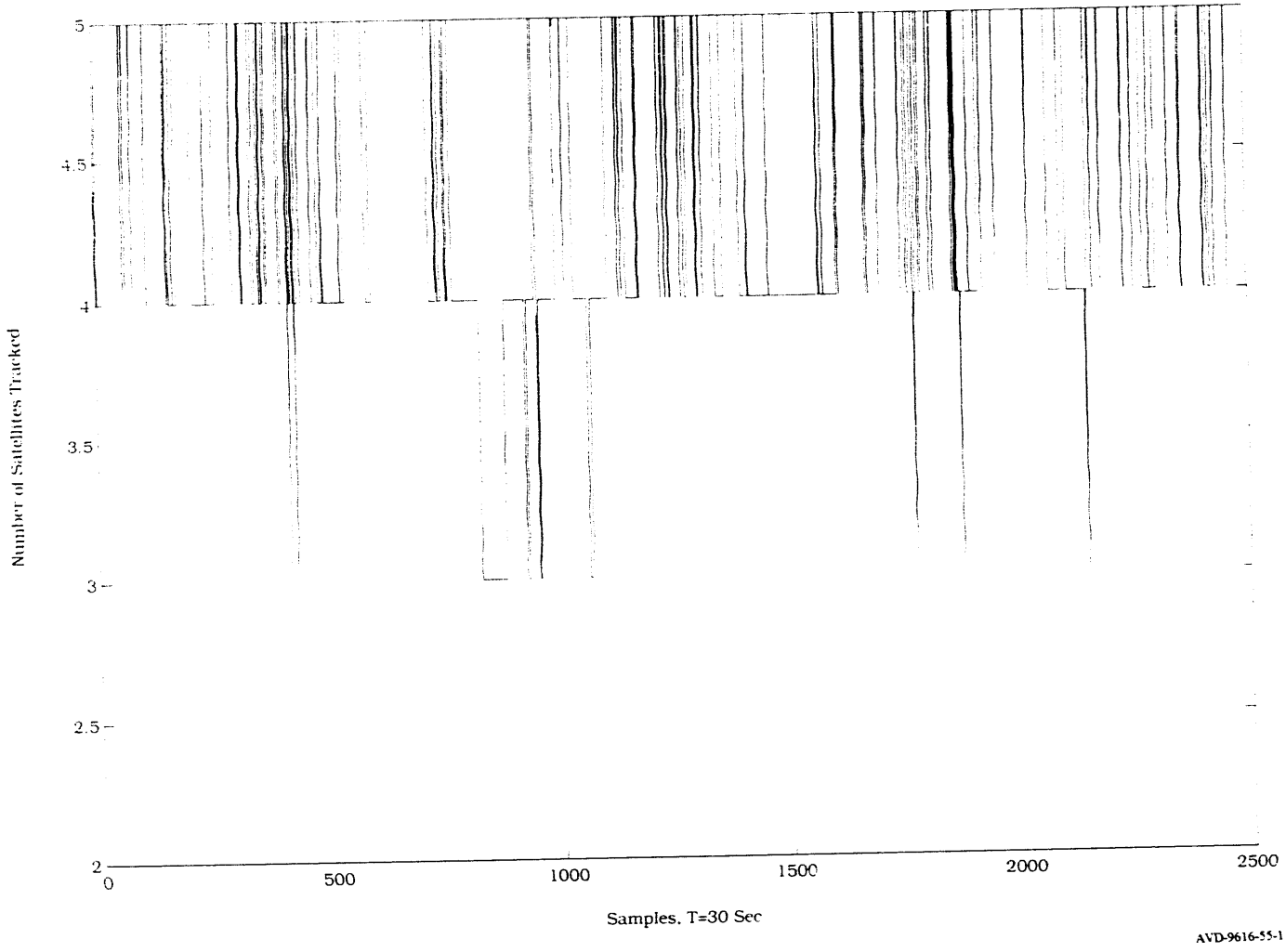
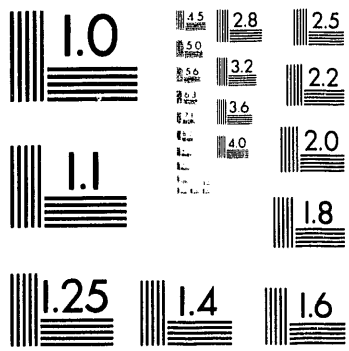


Figure D.2-3. Static Test Navigation Mode as a function of time, Rockwell NavCore V Navigation modes (C100714E.DAT).


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

3 of 3

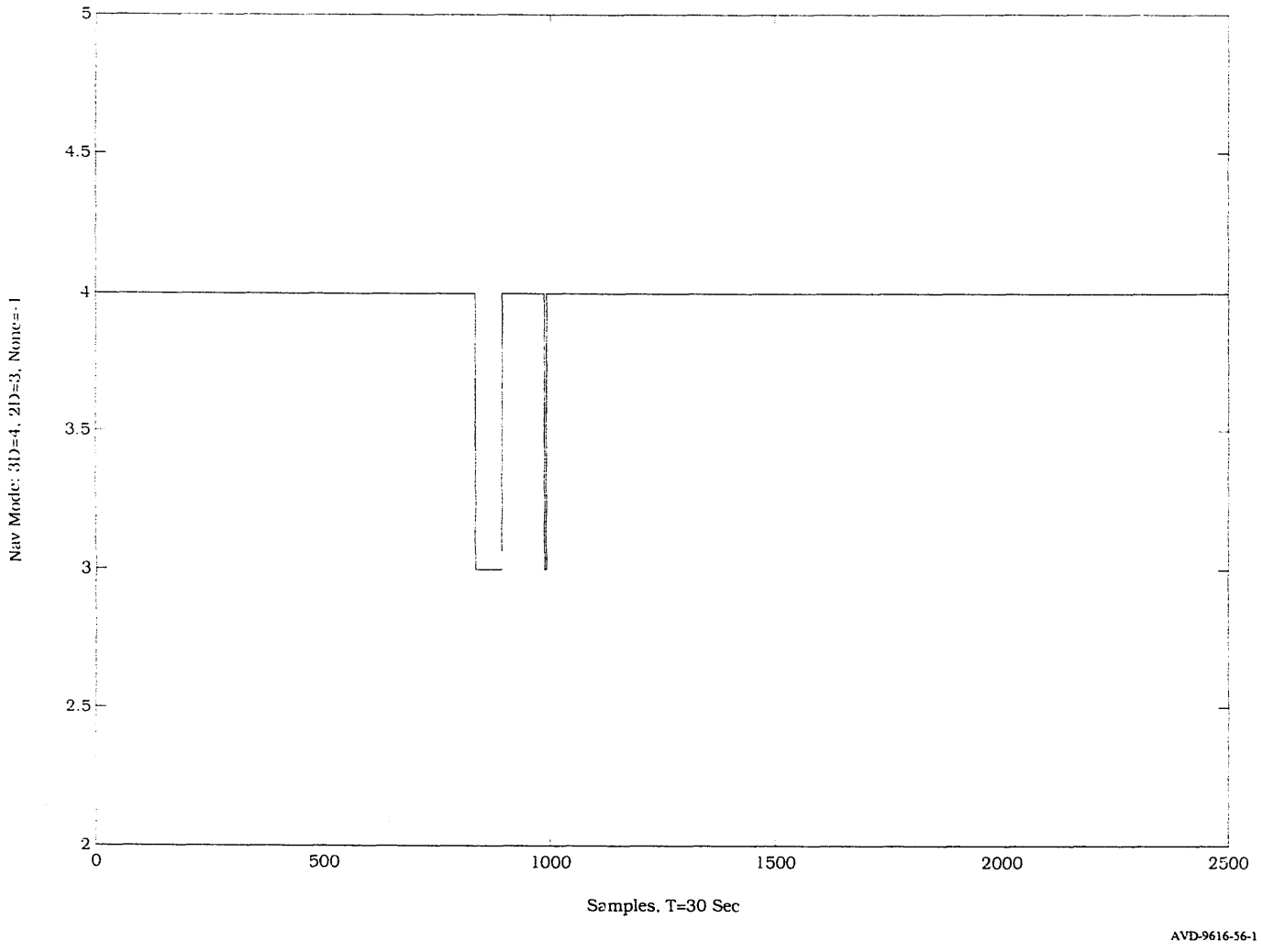


Figure D.2-4. Static Test Navigation Mode as a function of time, Magnavox 6400 Engine Navigation modes (D100714E.DAT).

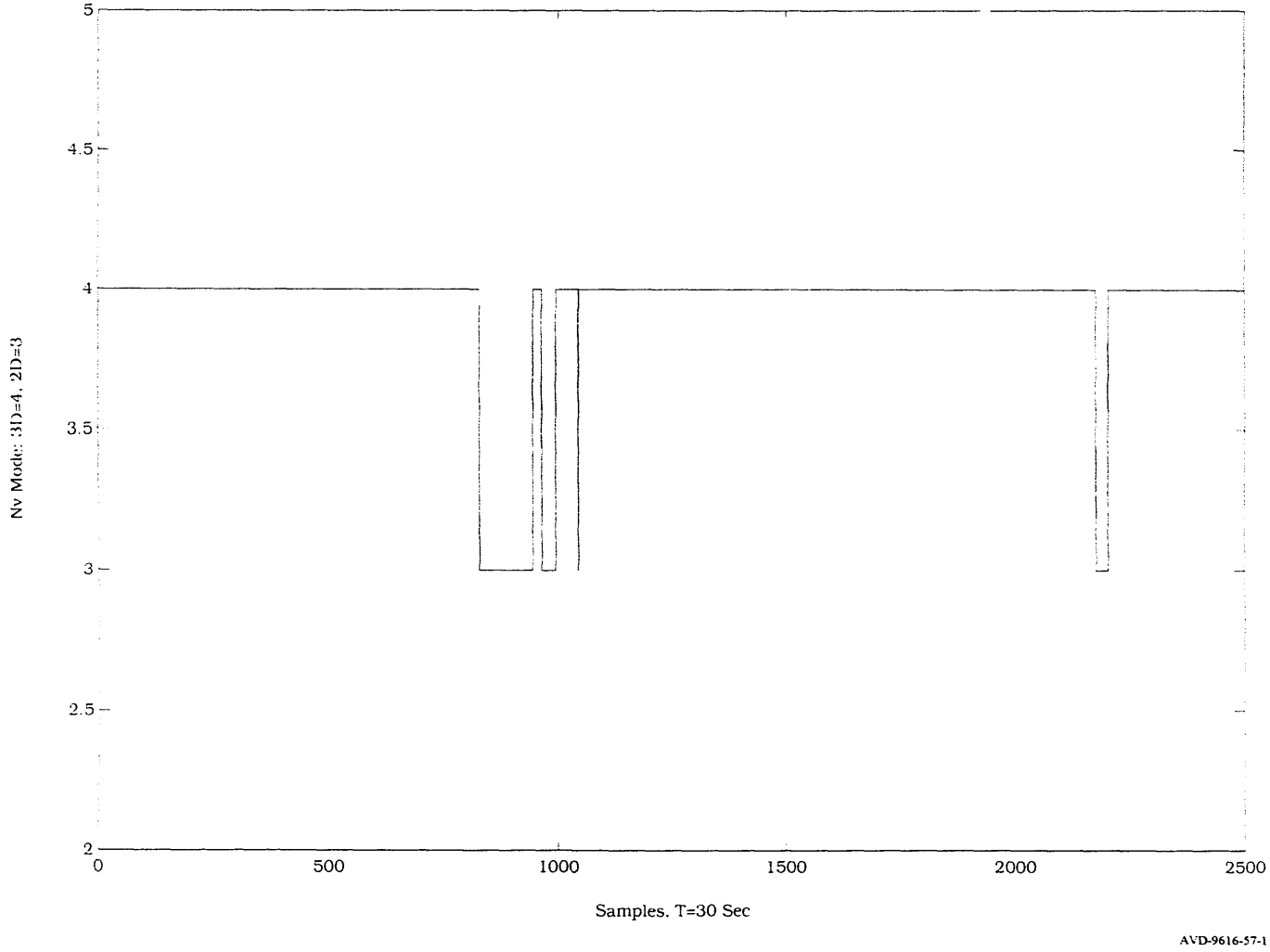


Figure D.2-5. Static Test Navigation Mode as a function of time, Trimble Placer Navigation (F100714E.DAT).

APPENDIX E: SAMPLES OF RAW DATA

Samples of Test Data taken during
the GPS receiver experimental period
(July to October 1992)

APPENDIX E: SAMPLES OF RAW DATA

Magellan

$GPGGA,201920,3502.43,N,10631.29,W,1,4,002,999,M,-023,M*68
$$BP[illegible]Â[illegible]À,[illegible]Ý%
$GPGGA,201950,3502.43,N,10631.28,W,1,4,002,999,M,-023,M*6E
$$B[illegible]Ã4À,ë[illegible]x

Magnavox GPS Engine

$PMVXG,021,332333.00,3502.4450,N,10631.2908,W,01642.8,-023.0,0000.1,-000.1,03*71
$PMVXG,021,332363.00,3502.4423,N,10631.2873,W,01627.9,-023.0,0000.3,-000.2,03*7E
$PMVXG,021,332393.00,3502.4394,N,10631.2815,W,01615.2,-023.0,0000.3,-000.2,03*70
$PMVXG,021,332423.00,3502.4371,N,10631.2751,W,01607.3,-023.0,0000.3,-000.1,03*79

Rockwell NavCore V

255 129 103 0 58 0 0 0 96 125 147 35 180 129 74 34 153 2 1 0 145 140 208 6 230 14 7 0 10 0 200 7 147 101 79 0 149 166 209 42 134 181 151 172 113 84 254 152 150 140 158 208 81 94 124 238 217 44 123 120 37 175 127 220 86 200 128 246 212 252 143 28 129 215 215 159 248 237 139 251 38 76 233 0 44 1 124 1 223 0 184 1 3 26 0 34 21 42 0 41 17 58 0 39 26 74 0 43 28 82 0 0 5 0 4 0 2 0 53 2 0 0 0 0 76 81 155 147

255 129 103 0 58 0 0 0 96 125 147 213 179 97 78 34 153 2 1 0 145 84 207 134 245 14 7 0 10 0 200 7 161 107 79 0 149 187 204 194 133 181 151 61 237 166 254 152 150 3 108 100 81 94 127 25 126 3 125 146 15 56 128 6 9 151 128 221 23 140 143 28 129 71 8 121 248 237 139 11 225 77 233 0 44 1 124 1 223 0 184 1 3 26 0 28 21 42 0 41 17 58 0 42 26 74 0 42 28 90 0 31 5 0 4 0 2 0 53 2 0 0 0 0 76 81 193 134

255 129 103 0 58 0 0 0 96 125 147 136 179 65 82 34 153 2 1 0 145 32 206 6 5 15 7 0 10 0 200 7 175 113 79 0 149 229 10 107 133 181 151 85 154 253 253 152 150 18 63 170 81 94 127 118 222 97 127 26 85 154 126 172 156 160 128 148 128 49 144 28 129 114 200 142 248 237 139 121 129 70 188 0 251 0 57 1 181 0 106 1 28 26 0 31 21 42 0 41 17 58 0 42 26 74 0 41 3 90 0 27 5 0 4 0 2 0 52 2 0 0 0 0 76 81 250 181

255 129 103 0 58 0 0 0 96 125 147 62 179 1 86 34 153 2 1 0 145 248 204 6 20 15 7 0 10 0 200 7 139 119 79 0 149 128 33 184 132 181 151 223 34 249 253 152 150 39 231 122 81 94 127 93 15 57 123 80 226 5 125 49 32 211 128 126 207 36 144 28 129 21 104 109 248 237 139 75 201 68 191 0 0 1 63 1 185 0 113 1 28 26 0 31 21 42 0 41 17 58 0 42 26 74 0 41 3 90 0 28 5 0 4 0 2 0 52 2 0 0 0 0 92 81 42 199

Magnavox 6400

29 332334.479 0.61157228 -1.85914969 1645.0 21 3 17 26 1.3 2.0 3.0
:O11911DB018EAEEB4405E000000000000006840005C000E9800290041
29 332335.607 0.61157220 -1.85914972 1645.6 21 3 17 26 1.3 2.0 3.0
29 332336.728 0.61157210 -1.85914977 1645.8 21 3 17 26 1.3 2.0 3.0

Trimble Placer

>RPV73170+3504044-1065214900003912;*78<
>RPV73200+3504046-1065213900003912;*79<
>RPV73230+3504052-1065213100003912;*77<
>RPV73260+3504058-1065211700103912;*7D<
>RPV73290+3504069-1065211200103912;*75<

APPENDIX F: GPS COMPANY CONTACTS

List of the individual contacts,
who provide technical support and pricing
information for the GPS receivers tested.

APPENDIX F: GPS COMPANY CONTACTS

The following is a list of the individual contacts, who provide technical support and pricing information for the GPS receivers tested.

Magellan

Emile Yakoup
Magellan Systems Corp.
960 Overland Court
San Dimas, CA 91773
909-394-6062

Magnavox Advanced Products and Systems

Eric Furlong
Magnavox Advanced Products and Systems
2829 Maricopa Street
Torrance, CA 90503
310-618-1200, Ext. 3056

Rockwell International

Larry Creech
Rockwell International
3200 East Renner Road
MS 461-235
Richardson, TX 75082
214-705-1704

Trimble Navigation

Jeff Jacobs (Qty Pricing)
408-481-2865
Joel Avey (Tech Support)
408-481-8927
Trimble Navigation
645 North Mary Avenue
Building 5
Sunnyvale, CA 94088-3642

DISTRIBUTION

Company Contacts

1 Magellan Systems Corp.
Attn: Emile Yakoup
960 Overland Court
San Dimas, CA 91773

3 Magnavox Advanced Products and Systems
Attn: Eric Furlong (3)
2829 Maricopa Street
Torrance, CA 90503

1 Rockwell International
Attn: Larry Creech
3200 East Renner Road
MS 461-235
Richardson, TX 75082

1 Rockwell International
Attn: Bob Knott
Commercial GPS Business
3200 East Renner Road
Richardson, TX 75082

1 Trimble Navigation
Attn: Joel Avey
645 North Mary Avenue
Building 5
Sunnyvale, CA 94088-3642

1 Ziatech Corporation
Attn: Dave Rennie
3433 Roberto Court
San Luis Obispo, CA 93401

Additional Distribution

1 Lt. Col. Mark Swinson
USMC
UGV/JPO
U.S. Army Missile Command (MICOM)
Attn: AMSMI-RD-UG (AMC-PM-UG)
Redstone Arsenal, Bldg. 5410
Huntsville, Alabama 35898

1 Russell Garnsworthy
Australian Associated Technologies
Ptg Limited
Level 32
600 Bourke St.
Melbourne, Australia 3000

2 Tech Reps, Inc.
Attn: M. Minahan
B. Grant
5000 Marble NE, Suite 222
Albuquerque, NM 87110

Sandia Internal

1	1602	R.W. Harrigan
1	2346	A.L. Schauer
1	9600	J.R. Kelsey
1	9602	D.L. Caskey
1	9603	E.R. Hoover
1	9604	S.C. Roehrig
1	9613	R.C. Wahlberg
1	9615	R.C. Ghormley
5	9615	K.T. Malone
1	9615	L. Scott
1	9615	L. Riblett
1	9616	B.C. Caskey
20	9616	R.H. Byrne
1	9616	J.B. Pletta
1	9616	R.K. Cover
1	9616	F.K. Wunderlin
3	8523-2	Central Technical Files
3	7151	Technical Publications
10	7613-2	Document Processing for DOE/OSTI
5	7141	Technical Library

☆ U.S. GOVERNMENT PRINTING OFFICE 1993—774-122/80245

DATE
FILMED
12/27/93
END