An Enhanced Reality System for the Great Outdoors

by

Ona Wu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science

and

Master of Science in Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

© Ona Wu, MCMXCIV. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part, and to grant others the right to do so.

Author .....
Department of Electrical Engineering and Computer Science
May 16, 1994

Certified by .....
Alex P. Pentland
Associate Professor of Media Arts and Sciences
Thesis Supervisor

Certified by .....
Michael Potmesil
Member of Technical Staff, AT&T Bell Laboratories
Thesis Supervisor

Accepted by .....
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

WEISNER
Eng.
MASSACHUSETTS INSTITUTE
OF TECHNOLOGY
MIT LIBRARIES
JUL 13 1994

An Enhanced Reality System for the Great Outdoors

by

Ona Wu

Submitted to the Department of Electrical Engineering and Computer Science
on May 16, 1994, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science
and
Master of Science in Electrical Engineering

Abstract

Computers and communication technology can be very effective in enhancing our ability to visualize 3D environments where we live and work. This thesis proposes a prototype system that not only allows viewing but also real-time interaction with various 2D and 3D models of our environs. Implementing such a system required the development of hardware that tracks a user's spatial position and orientation anywhere on the planet and software utilizing this information. Towards this end, a navigation unit was designed which outputs the user's 3D location (in Earth-Centered Earth-Fixed coordinates) as well as 3D orientation (azimuth, elevation, and roll angles) with respect to the earth's magnetic North Pole and local gravity vector. This unit was interfaced to a portable computer which reads the navigation information and provides the user's position via a server to various client graphics programs which were developed to demonstrate the potential of a portable enhanced-reality system.

Thesis Supervisor: Alex P. Pentland
Title: Associate Professor of Media Arts and Sciences

Thesis Supervisor: Michael Potmesil
Title: Member of Technical Staff, AT&T Bell Laboratories

Acknowledgements

I would like to thank David Doyle and Dr. Dennis Milbert of NOAA for giving me advice on where to start my research on GPS. My thanks also go to Mike Slavin and Paul Braisted at Trimble Navigation for technical support. I would also like to thank Boris Pevzner for his advice, and especially for the use of his personal computer in generating many of the figures.

I would like to thank my faculty advisor at MIT, Sandy Pentland, for his patience and words of advice.Finally, I would like to express my gratitude to everyone in the Machine Perception Department at AT&T Bell Laboratories for their help, advice and especially camaraderie. My especial heartfelt thanks goes to my advisor, Michael Potmesil, for his support and guidance all these years. Most of all, I must thank my parents for their unmitigated support of their daughter and her dreams.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Taking advantage of improved computer and communications technology, we designed and implemented a portable system which not only viewed but also interacted with 2D and 3D models of the world as a precursor to what can lead ultimately to a portable "virtual reality" system.

1.1 Virtual Reality

In the words of Jaron Lanier [17], the coiner of the term "Virtual Reality", "Virtual reality is a technique for creating simulated experience, or rather an experience of a simulated external world." Previously confined to research labs, virtual reality has found popularity in recent years due to media hype, movies such as "Lawnmower Man" and "Virtuality" centers pervasively found in shopping malls across America.[1] Although virtual reality has gained popular recognition, it is not yet ready for consumer use primarily due to the necessity of users donning embarrassing and restrictive body attachments. One of the more burdensome equipment a would-be virtual reality user are sometimes forced to wear is an eye-phone consisting of two Sony NTSC television sets mounted on a helmet placed over the user's eyes with stereoscopic images displayed on the screens. Some other systems require the user to wear a Data Glove, a glove with many wires and cables connected to a processor which tracks the hand's movement and motion. Other systems use a joystick which tracks the hand's movement and allows commands to be executed by pressing a button or a series of buttons. With all of these body-mounted and expensive electronics, the potential consumer could be put off. However, due to improving technology some of the aforementioned difficulties could be overcome.

One of the essential components in virtual reality (VR) systems is the necessity of tracking a user's position in 3D space. In many of the existing VR systems, the user is forced to stay in a restricted area where electromagnetic sensors can

[1] These centers consist of video games in which the user is immersed in a computer generated playing ground. In one game, the person is driving a car through a computer generated road. In another game, "Dactyl Nightmare", the user shoots at enemies and dodges pterodactyls. These centers are discussed in [17].

track her location and motion. To create a truly portable virtual reality system, a method of tracking the user's six degrees of freedom (x, y, z location and azimuth, elevation and roll orientation) without relegating the person to a restricted area must be utilized. Thanks to the research of the United States Department of Defense, we have available to us a system known as the Global Positioning System (GPS) which has the capability of pinpointing a person's location under normal operation, to an accuracy within ten to twenty meters.[2] Through the use of a series of compasses and tilt sensors, the user's local orientation can be determined. Combining these two technologies, portable virtual reality can become a *physical* reality.

There are many potential applications for a portable VR system besides the obvious entertainment ones such as video games. One can look up at the sky and see an annotated map of stars and planets based on one's current location and time. Another application would be for maintenance engineers who could look down and view the underground infrastructure of a particular site and view its tunnels, pipes and cables and locate accurately the area where repairs were needed, eliminating the need for costly excavations. For architects, who already use virtual reality as a design aid, one can look at a future construction site and see its finished version.[3] For history students, by turning towards the site of an ancient civilization, the ruins of past civilizations could be reconstructed to their previous glory before the students' eyes. Nature lovers could use such a system when hiking or bicycling to zoom in on a potential hiking site or to avoid getting lost. One can also use such systems to help him navigate through a shopping center or around town. These are just some of the possible applications that a portable VR system will open up to society.

The examples just mentioned could also be applications of "Enhanced Reality" Systems in addition to Virtual Reality systems. "Enhanced Reality" Systems would enable the user to retain control of her senses by allowing the user to retain the ability to interact with *real* reality. Instead of an eye-phone, the user could use the *Virtual Vision Sport* from Virtual Vision, Inc. (a heads-up head-mounted display that projects a TV image in front of a user's eyes but does not completely cover them). We implemented a portable "enhanced reality" system rather than a portable VR system since we felt that because we are developing a *portable* system, it would be extremely dangerous for a user be submerged in another reality while walking about in *real* reality. We rely on our senses to protect ourselves from foolhardily endangering ourselves and if they were blocked off, as what true virtual reality systems would do, we would be susceptible to tripping, bumping and falling into things that exist in natural surroundings.

[2]GPS will be discussed in more detail in the second chapter.

[3]This is much like Matsushita's Electric Works in Tokyo where consumers go to a special area, don the heavy equipment and design their own kitchen by viewing how the kitchen would look. This is describe in [17] and [46].

1.1.1 System Overview

The implementation of our enhanced-reality system can essentially be divided into two components: hardware and software. The hardware part of the system consisted of the design and implementation of a "navigation unit" that outputs one's spatial information to a computer interface. The unit contains a GPS receiver, two compasses, nine tilt sensors and an interface module. The hardware implementation will be discussed in Chapter 3.

The application software interface involved developing a multithreaded server which provides data from the navigation unit to numerous client application programs that utilized the data to render position dependent graphics. The server/client software aspect of the project will be discussed in Chapter 4. The graphics software package is discussed in Chapter 5.

Once the system was designed, we developed a few demonstration programs to exhibit the capabilities of such a system. Before we could use the navigation box, we conducted experiments to determine the reliability of the system, concentrating especially on the GPS receiver. The outcome of the experiments is discussed in Chapter 6. Chapter 7 discusses a few proof-of-concept demonstrations. Finally, the concluding chapter explores future upgrades and areas of research to make the system more robust and truly portable.

Chapter 2

Global Positioning System

Portable enhanced reality can exist only if there exists a capability to track the user's position with respect to some global coordinate system. The system that we have developed has been designed for only terrestrial use, that is for use only on this planet. With this in mind, the user's position anywhere on the planet will suffice as the user's translational (x, y, z) location with his orientation (azimuth, elevation and roll) angles calculated with respect to his horizon. The Global Positioning System (GPS), developed and maintained by the United States Department of Defense (DOD) provides us with the capability to track the user's translational position. Compasses and tilt sensors, to be discussed in the next chapter, provide the local orientation information.

The Global Positioning System, an all-weather 3D radionavigation system, operates on the principle of triangulation. Using a constellation of satellites, GPS receivers can pinpoint their 3D position to within a few meters in single receiver mode, and to within centimeters if using a network of receivers. In addition, the GPS system provides 3D velocity and precise time information. The system was designed to serve an unlimited number of users on a global basis. Currently, there is no user fee for using the system, however, this situation may change in the near future. GPS technology has already come into use in a wide variety of applications ranging from the obvious military ones to less obvious ones such as ambulance tracking or AT&T network synchronization [8]. Because of its world-wide coverage, GPS applications are popular globally, as was exhibited by the formation of the Japan GPS Council (JGPSC) in November 1992, a Japanese counterpart to the United States GPS Industry Council [29].

The Global Positioning System was developed in response to DOD request for an improved navigation and timing system in 1973 [27]. In 1978, the first satellite was launched. By 1983, the DOD announced the NAVSTAR (Navigation Satellite Timing and Ranging) GPS policy for non-military users. This was detailed in the first Federal Radionavigation Plan (FRP) in 1986. The FRP continues to be updated and published every two years. GPS positioning capabilities are provided at two levels of service – Standard Positioning Service (SPS) and Precise Positioning Service (PPS). SPS is the service provided to civilians worldwide at no cost with a horizontal

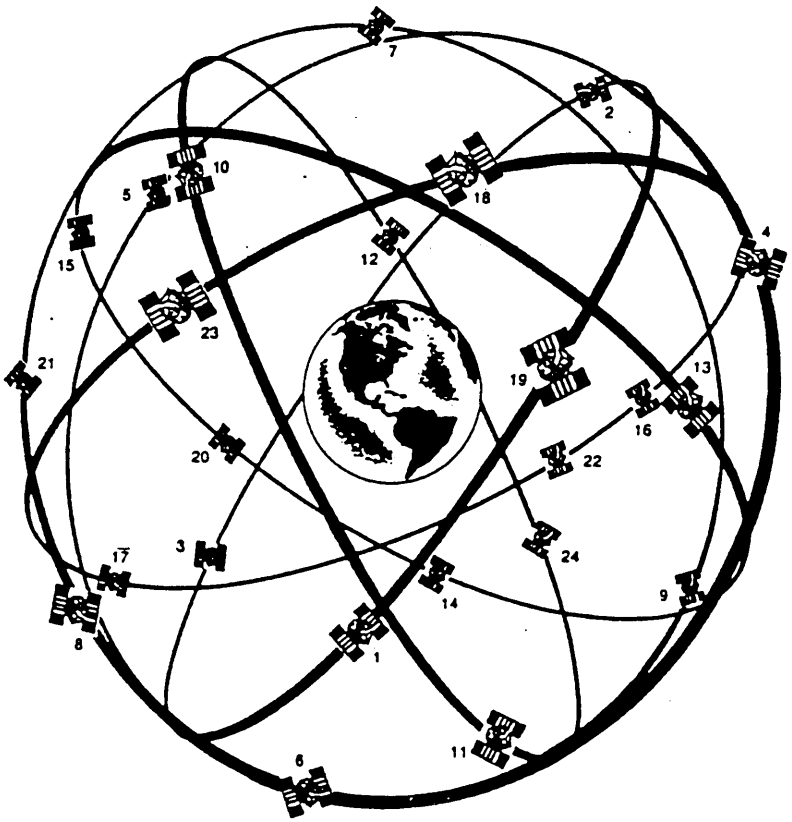
positioning accuracy of less than 100 meters [7] and a UTC[1] (Universal Coordinate Time) time transfer accuracy of 337 nanoseconds with a 95% probability. This is the equivalent of 156 meters 3D positioning accuracy. SPS velocity accuracy is 0.2 m/s. The velocity accuracy in degraded service mode is classified [27, p. 1-13]. The Precise Positioning Service, on the other hand, is for military applications. It provides extremely accurate position, velocity and time information. It is specified to provide 16 meters Spherical Error Probability (SEP) (3D, 50%) positioning accuracy and 100 nanoseconds (one sigma) UTC time transfer accuracy. This is equivalent to 30 meters of accuracy and 197 nanoseconds time accuracy. In terms of velocity, it has an accuracy of 0.2 m/s [27, p. 1-13]. This service is restricted to the United States government and its allies.

2.1 GPS Segments

The Global Positioning System can essentially be divided into three basic components – the User Segment, the Space Segment and the Control Segment. The User Segment is composed of the receivers and antennas used to determine position, velocity and precise time. There are many vendors of GPS equipment, some who claim various wondrous things about their equipment, which may or may not be believed. The particular choice of a receiver depends on the user's requirements, i. e., expense, accuracy, data format, and so forth.

The Space Segment consists of a constellation consisting of a minimum of 21 operational satellites vehicles (SV) and three spare vehicles. With this combination, 8 satellites or less will be in view more than 99% of the time [34, p. 3-1]. Currently, the satellite constellation is a combination of Block I and Block II satellites. Eleven Block I satellites and 28 Block II satellites were commissioned to be built by by Rockwell International. Their life-spans are approximately 7.5 years. Block I satellites are older satellites used for developmental purposes. When full GPS Operational Capability (FOC) GPS is reached, worldwide 3D coverage will be available 24 hours daily and the constellation will consist of only Block II satellites. This is not expected to occur until sometime in 1995. Currently, GPS is only in Intermittent Operational Capability (IOC) mode due to the fact only 23 of the 24 Block II satellites have been launched. In addition, three of the orbiting satellites are Block I satellites, resulting in a total constellation of 26 satellites. The final constellation will at a minimum consist of six (ABCDEF) planes with four Block II satellites in each plane at an inclination angle of 55 degrees from one another [Figure 2-1]. Eighteen of the Block II satellites are designated as Block IIA. They have the feature that they can operate without contact with the Control Segment for 180 days. In 1989, General Electric was awarded a contract to build twenty Block IIR satellites with option of six more. These satellites can also operate for up to 180 days without contact with the Control Segment using cross-link ranging techniques [27, p. 1-7]. The orbital altitude of each

[1]UTC time or broadcast time is the time used for civilian applications. It is considered a hybrid time scale that relates highly stable atomic time to the Earth's rotational time [15].



From **NAVSTAR GPS User Equipment Introduction** [27, p. 1-3]

Figure 2-1: GPS Satellite Constellation

satellite is approximately 20,200 km or 10,900 nautical miles (nm), the equivalent of approximately 3 Earth diameters. It should be noted that the satellites are not geosynchronous but rather that they complete an orbit every 12 sidereal hours. In terms of UTC time, they appear twice a day four minutes earlier than the previous day.

The Control Segment involves monitoring and updating the satellites of the GPS Space Segment. It consists of five monitor stations, three ground antennas and the Master Control Station (MCS). The five monitor stations, located at Hawaii, Kwajalein, Ascension Island, Diego Garcia and Colorado Springs, track the SV positions as they traverse their respective orbits. The MCS, operated by the USAF 2nd Satellite Control Squadron (2SCS) at the Falcon Air Force base in Colorado, processes information from the monitor stations to predict the satellites' ephemerides and update the SVs' navigation message. The ground antennas at Ascension Island, Diego

Garcia and Kwajalein transmit the updated information back to the satellites [7].

Although our project primarily involves the User Segment, in order to fully understand the limitations on GPS positioning capabilities, one should understand roughly the makeup of the system. Therefore, before proceeding to the implementation of an enhanced reality system utilizing GPS receivers, we will first discuss the basic theory behind GPS, its limitations and solutions to try to overcome them. The majority of the information in this chapter regarding the Global Positioning System is based upon information from the government, primarily the Institute of Navigation papers, GPS specification sheets and from on-line information available from the U.S. Naval Observatory and daily updates from the `CANSPACE@UNBVM1.BITNET` newsgroup.

2.2 GPS Satellite Signals

As mentioned previously, the Global Positioning System operates on the principle of triangulation. By measuring the distance between the receiver and at least three satellites, the user can calculate the 3-D position (U_x, U_y, U_z) of the receiver. To calculate the distance between a satellite and a receiver, the travel time of a signal between the two is measured and multiplied by the speed of light $c = 2,979,258$ m/s. Because an error in the synchronization between the clocks on the SV and on the receiver as little as one nanosecond can lead to an error of about 0.3 meters, very accurate timing systems are required. Very stable atomic clocks are used on-board the satellites with a predictable offset from GPS time.[2] Clock correction parameters are periodically transmitted to the satellites. However, if such accuracies were demanded on the users' clocks, it would make receivers inordinately expensive. To offset the cost, measurements from at least 4 satellites are required where the fourth unknown is the receiver clock bias. The measured ranges to the satellites are therefore called pseudo ranges since each measurement contains a clock bias contribution in addition to range information.

Two radiofrequency (RF) right-handed circularly polarized signals at carrier frequencies of L1=1575.42 MHz and L2=1227.6 MHz are transmitted by the satellites. The L1 signal is bi-phase shift key modulated (BPSK) by two pseudo-random noise (PRN) codes – precision P-code and C/A coarse/acquisition code. The L2 signal is modulated only by one code, either P-code or C/A code – which one being determined by the Control Segment. Typically, L2 is modulated by the P-code. C/A code has a chipping rate of 1.023 MHz while the P-code is at 10.23 MHz. In addition, both signals are also modulated by a 50 baud navigation message. Both transmitted signals are derived from the fundamental frequency of 10.23 MHz. Due to relativistic effects, the on-board SV oscillator is offset to 10.22999999543 MHz. All signals are synchronized with the P-code. The minimum power level requirements for the signals

[2]GPS time is not the same as UTC time due to the need for UTC to be adjusted by leap seconds. To avoid disruptions that continual adjustments would cause in GPS, the difference between GPS and UTC to within 100 μs is maintained and published regularly for users for are interested in accurate timing [25, p. 5]. The difference in integer leap seconds between GPS and UTC is provided by the satellite navigation messages.

Channel	Signal	
	P(Y)	C/A
L1	-163.0 dBW	-160.0 dBW
L2	-166.0 dBW	-166.0 dBW

Table 2.1: Received Minimum RF Signal Strength

at a GPS receiver is specified by Table 2.1 [13, p. 13]. PPS users can use P-code or C/A code or both. SPS users are relegated to using only C/A once GPS becomes FOC. Although SPS users currently have access to the P-code, once GPS becomes FOC, the anti-spoofing (A-S) feature will be activated which will encrypt the P-code to Y-code. PPS users will have access to the encryption key while SPS users will not. Therefore, civilians should limit their applications to relying on a purely C/A code solution.

Each satellite transmits its own unique PRN codes which allows all 24 satellites to broadcast at the same frequencies. The period of the P-code is 37 weeks. This allows the assignment of unique one week length segments to each satellite. The PRN number that typically identify satellites, is determined by which segment of the PRN code which it broadcasts. Because the code length is seven days, the code is reset once a week at Saturday-to-Sunday midnight. The GPS week, which serves as the basic unit of time, begins at the start of the code cycle. GPS zero time, week zero is at midnight of January 5, 1980 to January 6, 1980 [13, p. 32]. The P-code is the modulo-2 sum of two subsequences known as X1 of length 15,345,000 chips and X2 of length 15,345,037 chips. A shifted X2 is combined with X1 to generate the 37 unique codes. For more details on the generation of the P-code, see the Interface Control Document [13, pp. 16-26].

The C/A code, which is synchronized with P-code, has a period of only 1 millisecond (ms). Each satellite also broadcasts its own unique C/A code. The C/A code is the exclusive-or of two Gold codes, G1 and G2, of length of 1023 chips. The G2 sequence is delayed by from 5 to 950 chips resulting in 36 unique C/A codes [13, p. 7]. The C/A code is used by PPS users to synchronize onto the P-code since the 7-day period P-code is quite difficult to lock onto, while the 1 ms C/A code is not as long a search interval. Through the use of the handover word (HOW) in the navigation message, once lock onto C/A code has occurred, the receiver can then switch to P-code. There will be difficulty in locking onto the C/A code if the receiver's clock offset is greater than 1 ms.

On board GPS receivers is a copy of the PRN codes. Once the receiver has successfully locked onto the L1 signal, which will be different from the specified carrier frequency due to a Doppler shift dependent on receiver dynamics and positioning, the receiver enters a code tracking loop which tries to match up the received signal with the internally generated signal. The phase difference between the received code and the generated code is approximately the travel time. The signal is then demodulated using a BPSK demodulator leaving the navigation message and noise on the carrier.

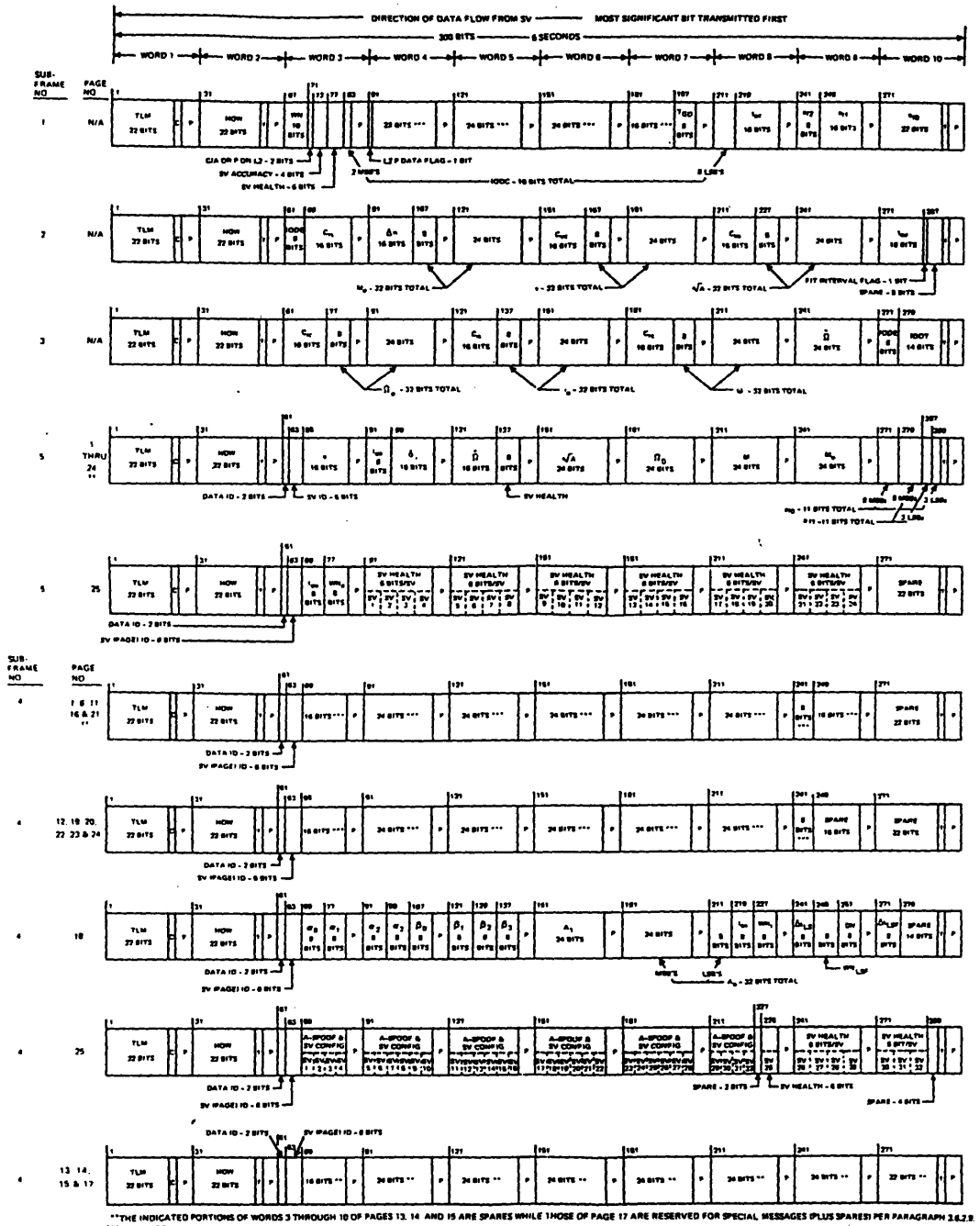
The message is recovered using bit synchronization and a data detection filter. See Spilker [36] for more details.

2.2.1 Navigation Message

The navigation message contains the broadcasting satellite's almanac and ephemeris information, timing characteristics, operational status and clock behavior. All of this information is necessary for position determination. The message is maintained and updated by the Control Segment. In this section, we will briefly discuss the contents of the navigation message. For more details see the Interface Control Document [13].

The navigation message is a 1500-bit long data stream subdivided into 5 subframes. Each subframe is made up of 30 10-bit length words. The first subframe consists of clock correction parameters. Subframes 2 and 3 consist of the ephemeris parameters of the broadcasting satellite. Subframe 4 and 5 rotate among 25 different versions called pages. Subframe 4 contains ionospheric correction terms for single frequency users, conversion from GPS to UTC time parameters and special messages. Subframe 5 consists of the health data and almanac information of all 24 satellites. The almanac information contains the less precise ephemerides for the satellites. It shortens the user time-to-first fix (TTFF) by helping the receiver acquire signals from the other satellites once it has received 25 full frames from one satellite since it now has a rough estimate of the position of the other satellites. The message is transmitted at 50 baud, resulting in a total transmission time of 30 seconds for one frame of data resulting in a total of 750 seconds to receive 25 full frames. After 25 full frames have been received the message repeats itself. Figure 2-2 shows the navigation message structure [13, pp. 60-61].

Each subframe begins with the same two words – the telemetry word (TLM) and the handover word (HOW). The 30-bit TLM word has the first 8 bits as a preamble that is used for subframe synchronization, 14 bits of TLM message, 2 bits reserved and the remaining 6 bits for parity check. The first 17 bits of the 30-bit HOW are the 17 most significant bits (MSB) of the time of week (TOW) count. The TOW is the 19 least significant bits of the 29 bit Z-count which counts the number of X1 epochs (the length of an X1 epoch is 1.5 seconds) that have passed. The TOW count ranges from 0 to 403,199 and is reset to zero at the start of a GPS week. The 10 MSB of the Z-count is the GPS week (modulo 1024) which ranges from 0 to 1023 and is referenced to January 5, 1980 [13, p. 33]. Bit 18 has a dual purpose – it is the roll momentum dump flag or an alert flag warning unauthorized users that the system may be degraded. Bit 19 also has two purposes – it is either a synchronization flag or an indicator of whether AS has been invoked. The purpose of bits 18 and 19 depends on the state of the configuration code in page 25 of subframe 4 [13, p. 65]. The remaining 3 bits, not counting parity bits, are used for subframe identification and have values from 1 to 5. The last 6 bits of every 30-bit word are used for parity checking using a (32,26) Hamming Code [45, p. 57].



From **ICD-GPS-200** [13, pp. 60-61].

Figure 2-2: Navigation Message Data Format

2.2.2 Parity Check

To check the parity of the received data, one must first recover the raw data bits. This is done by checking if the last parity bit of the previous word of the subframe is set to 1. If it is, then the first 24 received bits, $D_1 \cdots D_{24}$ are complemented to obtain the raw bits $d_1 \cdots d_{24}$. Parity bits D_{25} through D_{30} are then computed using the following algorithm and compared to the received parity bits:

$$\begin{aligned}
D_{25} &= D^{\star}_{29} \oplus d_1 \oplus d_2 \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_{10} \oplus d_{11} \oplus \\
&\quad d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{17} \oplus d_{18} \oplus d_{20} \oplus d_{23} \\
D_{26} &= D^{\star}_{30} \oplus d_2 \oplus d_3 \oplus d_4 \oplus d_6 \oplus d_7 \oplus d_{11} \oplus d_{12} \oplus \\
&\quad d_{13} \oplus d_{14} \oplus d_{15} \oplus d_{18} \oplus d_{19} \oplus d_{21} \oplus d_{24} \\
D_{27} &= D^{\star}_{29} \oplus d_1 \oplus d_3 \oplus d_4 \oplus d_5 \oplus d_7 \oplus d_8 \oplus d_{12} \oplus \\
&\quad d_{13} \oplus d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{19} \oplus d_{20} \oplus d_{22} \\
D_{28} &= D^{\star}_{30} \oplus d_2 \oplus d_4 \oplus d_5 \oplus d_6 \oplus d_8 \oplus d_9 \oplus d_{13} \oplus \\
&\quad d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{17} \oplus d_{20} \oplus d_{21} \oplus d_{23} \\
D_{29} &= D^{\star}_{30} \oplus d_1 \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_7 \oplus d_9 \oplus d_{10} \oplus \\
&\quad d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{17} \oplus d_{18} \oplus d_{21} \oplus d_{22} \oplus d_{24} \\
D_{30} &= D^{\star}_{29} \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_8 \oplus d_9 \oplus d_{10} \oplus d_{11} \oplus \\
&\quad d_{13} \oplus d_{15} \oplus d_{19} \oplus d_{22} \oplus d_{23} \oplus d_{24}
\end{aligned} \tag{2.1}$$

where $D^{\star}_{29}$ and $D^{\star}_{30}$ are the last 2 bits of the previous word of the subframe. If the computed parity equals the received parity, the data passes the parity check, otherwise, it fails.

2.2.2.1 Contents of Navigation Message Subframes

The third word of Subframe 1 consists of the 10 MSB's of the Z-count, also known as the GPS week. The remaining bits indicate which code is being used on the L2 channel, the user range accuracy (URA), and SV health. The remaining words contain the ionospheric and clock correction parameters. The Issue of Data, Clock (IODC) indicates the time of last upload and signals the user to changes in the correction parameters. The remaining bits are the clock correction parameters $t_{oc}, a_{f_2}, a_{f_1}, a_{f_0}$ and T_{GD}. T_{GD} is the estimated group delay differential for use by single frequency users. The T_{GD} is the difference in propagation time between L1 and L2 through the SV processing hardware prior to transmission. Dual frequency users do not need this term since a_{f_0} was calculated assuming the user uses two frequencies. For the actual algorithm used to correct the time received from the SV for clock errors, relativistic effects and group delay differential, see ICD [13, pp. 72-76].

Subframe 2 and 3 contain the ephemeris information of the transmitting SV. The parameters are only Keplerian in appearance but are actually the predicted ephemeris parameters using a least-squares curve fit over a period of time as indicated by the "fit interval" flag which indicates whether the curve fit interval was less than 4 hours or greater than 4 hours. In addition, there is an issue of ephemeris data (IODE) term

that indicates if there is a change in the ephemeris parameters. The receiver can then calculate the position of the satellite with respect to the center of the Earth using modified Kepler formulas. The parameter definitions are given in Table 2.2 [15, p. 61]. The scale factor for each parameter is in [13, pp. 80-81]. The calculation of

M_o	Mean Anomaly at Reference Time
Δn	Mean Motion Difference from Computer Value
e	Eccentricity
$\sqrt{A}$	Square Root of the Semi-Major Axis
Ω_0	Longitude of Ascending Node of Orbit Plane at Weekly Epoch
i_0	Inclination Angle at Reference Time
ω	Argument of perigee
$\dot{\Omega}$	Rate of Right Ascension
IDOT	Rate of Inclination Angle
C_{uc}	Amplitude of the Cosine Harmonic Correction Term to the Argument of Latitude
C_{us}	Amplitude of the Sine Harmonic Correction Term to the Argument of Latitude
C_{rc}	Amplitude of the Cosine Harmonic Correction Term to the Orbit Radius
C_{rs}	Amplitude of the Sine Harmonic Correction Term to the Orbit Radius
C_{ic}	Amplitude of the Cosine Harmonic Correction Term to the Angle of Inclination
C_{is}	Amplitude of the Sine Harmonic Correction Term to the Angle of Inclination
t_{oe}	Ephemeris Reference Time
IODE	Issue of Data (Ephemeris)

Table 2.2: Ephemeris Parameterization of the Navigation Message

satellite positions using these parameters will be discussed in more detail later in this chapter in the section covering Kepler's formula.

Subframes 4 and 5 are subcommutated to 25 pages. Pages 1, 6, 11, 12, 16, 19, 20, 21, 22, 23, and 24 are reserved. Pages 2, 3, 4, 5, 7, 8, 9 and 10 contain almanac data for SV 25 through SV 32 with an option to be used for other information which is indicated by a satellite health code of all ones in the last page of subframe 4. Pages 13, 14 and 15 are spares. Page 17 is used for special messages. Page 18 contains ionospheric and UTC data. Page 25 contains AS flags, SV configuration code for the 32 SVs and SV health for SV 25 through 32. Pages 1 through 24 of Subframe 5 contain almanac data for SV 1 through 24. Page 25 contains the SV health code for SV 1 through 24, almanac reference time and almanac reference week number.

The pages containing almanac information in Subframe 4 and 5 follow the same

format. The first 6 bits, starting at the third word is the SV identification number (SV ID) which is equal to the PRN signal code being used for SV ID 1 through 32. SV ID 0 indicates a dummy SV. SV ID 51 through 63 indicate that pages are not being used for almanac information. The remaining codes are currently unassigned. The almanac data contains a rough estimate of the clock and ephemeris data for a satellite. It is valid for up to 180 days and is used to aid the user in estimating satellite visibility and pseudorange to satellite [27, p. 3-7]. The parameters received are e, almanac reference time t_{oa}, inclination perturbation δ_i, $\dot{\Omega}$, $\sqrt{A}$, Ω_o, ω, M_o, and clock correction parameters a_{f_0} and a_{f_1}. The calculation of the estimated orbit position is simplified by assuming all parameters, except the seven Keplerian parameters $M_o, t_{oa}, e, \sqrt{A}, \Omega_o, i_o + \delta_i, \omega$ to be zero with a nominal inclination i_o of 60 degrees. The almanac time parameters are used to calculate the time from the reference epoch which is used in the calculation of the satellite orbital position. The algorithm is similar to the one used to adjust the time using parameters from Subframe 1 with the exception that the second order term, a_{f_2} is set to zero and relativistic effects are ignored. GPS time, t, for the almanac can be calculated as follows:

$$t = t_{sv} - \Delta t_{sv} \tag{2.2}$$

$$\Delta t_{sv} = a_{f_0} + a_{f_1}(t - t_{oa}) \tag{2.3}$$

In Equation 2.3, t is the time calculated from Subframe 1 clock correction parameters. The calculation of the orbit paths will then be the same as for the ephemeris. This will be discussed in a later section.

In addition to almanac information, Subframe 4 also contains UTC Parameters and ionospheric models. The UTC parameters relate UTC time with GPS time. They are given in Table 2.3. The algorithm to relate UTC to GPS time is given by ICD [13, pp. 104-106a]. The ionospheric parameters in Subframe 4 is meant to aid

A_0	Constant term of polynomial
A_1	First order term of polynomial
Δt_{LS}	Delta time due to leap seconds
t_{ot}	Reference time for UTC data
WN_t	UTC Reference Week Number
Δt_{LSF}	Scheduled future of recent past value of delta time due to leap seconds
WN_{LSF}	Week number leap second becomes effective
DN	Day number leap second becomes effective

Table 2.3: UTC Parameters

single frequency users to effectively model the propagation delay of the signal.[3] The parameters α_n are coefficients of the cubic equation representing the vertical delay

[3]Dual frequency users, i.e. those that track both the L1 and L2 frequency, can calculate the

and β_n are coefficients of a cubic equation representing the period of the model where $n = 0, 1, 2, 3$. The algorithm is given in ICD [13, pp. 107-108b].

Under normal operations, Block I satellites transmit Subframes 1, 2, and 3 for periods of one hour with a curve-fit interval of four hours. Block I SV's can only retain 3-4 days of upload data from the Control Segment. Under normal operation, Block II satellites transmit Subframes 1, 2, and 3 for periods of two hours with a curve-fit interval of two hours. Block II SV's can retain up to 182 days of upload data [13, p. 110a]. For the ephemeris information, the data is valid for approximately two hours. The almanac information is valid for about 3.5 days but can be extended by considering crossovers in time in the calculation of the orbit path.

2.2.2.2 Kepler's Law

To solve for the orbital position of a satellite, one needs to use Kepler's Formula to calculate the satellite position. Six Keplerian elements $\{\Omega, \omega, i, a, e, f\}$ can be used to characterize the orbit of the satellite. The parameters are given in Table 2.4. Alternate sets of Kepler Elements are $\{\Omega, \omega, i, a, e, M\}$ or $\{\Omega, \omega, i, a, e, E\}$ where M

Ω	Right Ascension of ascending node
ω	Argument of perigee
i	Inclination Angle
a	Semi-Major Axis
e	Eccentricity
f	True anomaly

Table 2.4: Six Kepler Elements

is the Mean Anomaly and E is the eccentric anomaly. The first five parameters are constant for normal orbits with the anomalies functions of time. Normal orbits are specified by treating the Earth as a point mass with no other external forces acting upon the satellite. Unfortunately, satellites in real orbits undergo various external disturbances such as variations in the Earth's gravity potential, gravitational force of the sun and moon and solar radiation pressure. Therefore, all Keplerian elements are functions of time for real satellites. See Leick [15, pp. 32-51] for more details.

The transmitted ephemeris information is only Keplerian in appearance since it contains correction factors for the Earth's gravitational potential and is a least-squares curve fit of the predicted orbit. The algorithm for calculating the ephemeris, shown in Table 2.5, is specified by the ICD [13, pp. 84-84b]. The algorithm to calculate the orbit position using almanac information is the same as for ephemeris information except that parameters not specified in the almanac data should be set to zero.

propagation delay due to the fact that the delay experienced by a signal going through the ionosphere is inversely proportional to the frequency of the signal squared. Therefore, the delay can be calculated by measuring the time delay differential in receiving the two signals. See [13, p. 75] for more details.

μ	=	$3.986005 \star 10^{14}$ meter^3/sec^2	WGS84 value of the Earth's Universal Gravitational Parameter
$\dot{\Omega}_e$	=	$7.2921151467 \star 10^{-5}$	WGS84 value of the Earth's rotation rate
A	=	$(\sqrt{A})^2$	Semi-major axis
n_0	=	$\sqrt{\frac{\mu}{A^3}}$	Computed mean motion in radians/sec
t_k	=	$t - t_{oe}$	Time from ephemeris reference epoch
n	=	$n_0 + \Delta n$	Corrected mean motion
M_k	=	$M_0 + nt_k$	Mean anomaly
M_k	=	$E_k - e \sin E_k$	Kepler's equation for eccentric anomaly (Solved by iteration)
ν_k	=	$\arctan(\frac{\sin \nu_k}{\cos \nu_k})$	True anomaly
$\sin \nu_k$	=	$\frac{\sqrt{1-e^2} \sin E_k}{1-e \cos E_k}$	
$\cos \nu_k$	=	$\frac{\cos E_k - e}{1-e \cos E_k}$	
E_k	=	$\arccos \frac{e+\cos \nu_k}{1+e \cos \nu_k}$	Eccentric Anomaly
Φ_k	=	$\nu_k + \omega$	Argument of latitude
δu_k	=	$C_{us} \sin 2\Phi_k + C_{uc} \cos 2\Phi_k$	Argument of latitude correction
δr_k	=	$C_{rc} \cos 2\Phi_k + C_{rs} \sin 2\Phi_k$	Radius correction
δi_k	=	$C_{ic} \cos 2\Phi_k + C_{is} \sin 2\Phi_k$	Inclination Correction
		$\delta u_k, \delta r_k, \delta i_k$	Second harmonic perturbations
u_k	=	$\Phi_k + \delta u_k$	Corrected argument of latitude
r_k	=	$A(1 - e \cos E_k) + \delta r_k$	Corrected radius
i_k	=	$i_0 + \delta i_k + (IDOT)t_k$	Corrected inclination
x'_k	=	$r_k \cos u_k$	Position in orbital plane
y'_k	=	$r_k \sin u_k$	Position in orbital plane
Ω_k	=	$\Omega_0 + (\dot{\Omega} - \dot{\Omega}_e)t_k - \dot{\Omega}_e t_{oe}$	Corrected longitude of ascending node
x_k	=	$x'_k \cos \Omega_k - y'_k \cos i_k \sin \Omega_k$	Earth-fixed X coordinate
y_k	=	$x'_k \sin \Omega_k + y'_k \cos i_k \cos \Omega_k$	Earth-fixed Y coordinate
z_k	=	$y'_k \sin i_k$	Earth-fixed Z coordinate

where t is GPS system time at time of transmission corrected for transit time (range/speed of light). Furthermore, t_k shall be the actual total time difference between the time t and the epoch time t_{oe}, and must account for beginning or end of week crossovers. That is, if t_k is greater than 302,400 seconds, subtract 604,800 seconds from t_k. If t_k is less than 302,400 seconds, add 604,800 seconds to t_k.

Table 2.5: Ephemeris Algorithm

2.3 Navigation Solution

Once 25 complete frames of the navigation message have been decoded, the user can use the almanac information to start searching for other visible satellites to use in computing the position. As mentioned earlier, to do point positioning, the receiver requires the range information for at least four satellites as shown in Figure 2-3. The four equations resulting from these four measurements are:

$$\mathrm{PR}_i = [(S_{xi} - U_x)^2 + (S_{yi} - U_y)^2 + (S_{zi} - U_z)^2]^{\frac{1}{2}} + b_u \tag{2.4}$$

where:

PR_i	=	measured pseudorange to the ith satellite
S_{xi}, S_{yi}, S_{zi}	=	position coordinates of the ith satellite, computed from the navigation message
U_x, U_y, U_z	=	three coordinates of the user position to be solved
b_u	=	contribution to the pseudorange caused by the user clock offset error to be solved

However, besides the constant clock offset error, there exist independent errors in each range measurement. Even after correcting for the SV clock drifts for ionospheric delays, using information from the navigation message, there exist independent errors in the range measurement that cannot effectively be modeled. As a result, we cannot solve the two sets of four equations as deterministic equations, but rather, need to model them as noisy probabilistic signals. To solve for the four unknowns, we use a linear least squares solution. From Figure 2-3, we see that:

$$\vec{R}_i = \vec{S}_i - \vec{U} \tag{2.5}$$

for the ith satellite and $\vec{R}_i = (S_{xi} - U_x, S_{yi} - U_y, S_{zi} - U_z)$. The magnitude of the range R_i is equal to:

$$R_i = \vec{e_i} \cdot \vec{R}_i \tag{2.6}$$

where $\vec{e_i} = \vec{R_i}^T \vec{R}_i / R_i$. Therefore, PR_i, can be represented as:

$$\mathrm{PR}_i = \vec{e_i} \cdot \vec{R}_i + b_u + B_i \tag{2.7}$$

where b_u and B_i represents the range equivalent of the user and satellite clock biases. Taking the dot product of Equation 2.5 with $\vec{e_i}$ and substituting Equation 2.7, we get:

$$\mathrm{PR}_i - b_u - B_i = \vec{e_i} \cdot [\vec{S}_i - \vec{U}] \tag{2.8}$$

Rearranging terms we get:

$$\vec{e_i} \cdot \vec{U} - b_u = \vec{e_i} \cdot \vec{S}_i - \mathrm{PR}_i + B_i \tag{2.9}$$

To solve for n equations where $n \geq 4$, we represent Equation 2.9 in matrix representation as:

$$B = \mathbf{A}X \tag{2.10}$$

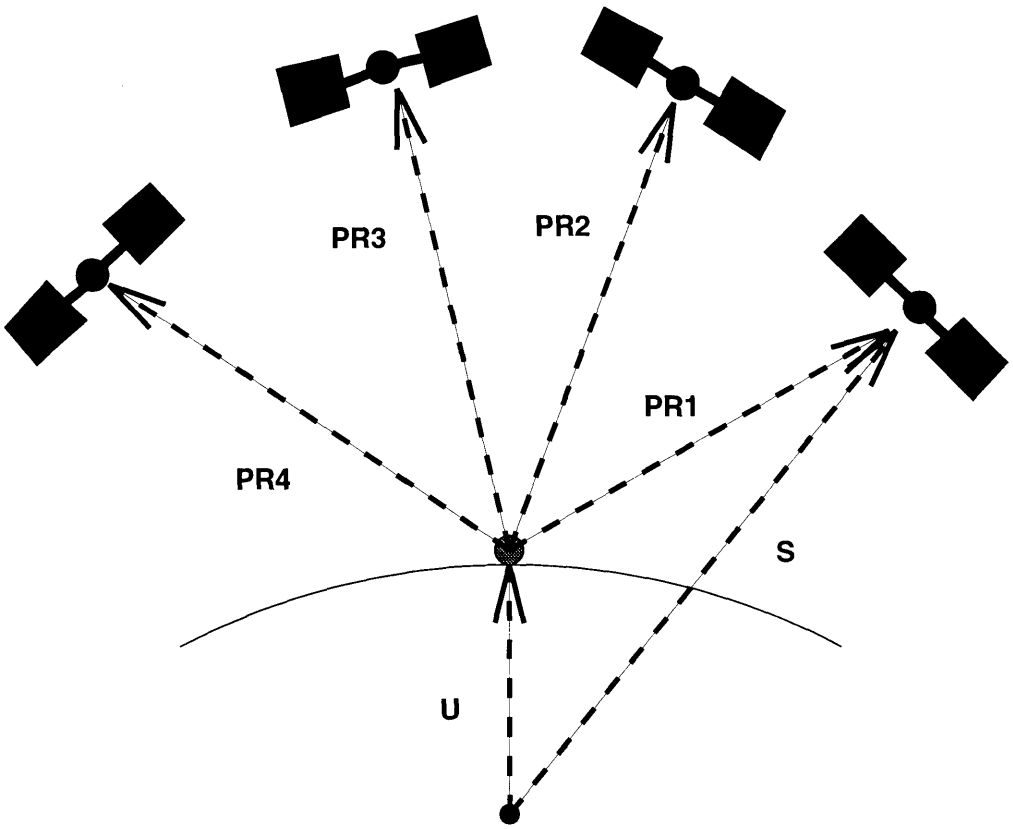


Figure 2-3: Geometry for GPS Measurement

where $\mathbf{A}$ is an $n \times 4$ matrix of the form:

$$\mathbf{A} = \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \\ \Gamma_3 \\ \vdots \\ \Gamma_n \end{bmatrix} \tag{2.11}$$

$$\Gamma_i = [e_{ix}, e_{iy}, e_{iz}, 1]^T \tag{2.12}$$

and $\vec{X}$ is an 4×1 vector of the form $[U_x, U_y, U_z, -b_u]$. B is an $n \times 1$ vector, whose ith element is

$$B_i = \vec{e_i}\vec{S_i} - \mathrm{PR}_i + B_i \tag{2.13}$$

Therefore, B can be represented as:

$$B = \mathbf{H}\vec{S} - \mathrm{PR} \tag{2.14}$$

$$\mathbf{H} = \begin{bmatrix} \Gamma_1^T & \vec{0} & \vec{0} & \cdots & \vec{0} \\ \vec{0} & \Gamma_2^T & \vec{0} & \cdots & \vec{0} \\ \vec{0} & \vec{0} & \Gamma_3^T & \cdots & \vec{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \vec{0} & \vec{0} & \cdots & \vec{0} & \Gamma_n^T \end{bmatrix} \tag{2.15}$$

$$\begin{aligned} \vec{S} = {} & [S_{1x}, S_{2x}, S_{3x}, B_1 \cdots \\ & \cdots S_{nx}, S_{ny}, S_{nz}, B_n] \end{aligned} \tag{2.16}$$

and PR is the vector of pseudorange measurements. From the above equations, one notices that $\mathbf{A}$ and $\vec{H}$ are both dependent on knowledge of the direction cosines between the user and the satellites. Unless, the receiver is initialized with its position, an iterative solution, refining the direction cosines, and therefore matrices $\mathbf{A}$ and $\mathbf{B}$, is used to determine the position. One method is to use a Kalman filter, a recursive estimator that minimizes the covariance matrix. In addition to the position of the receiver, the velocity can be calculated in a similar fashion using relative velocities rather than pseudoranges. The relative velocities can be calculated by measuring the Doppler shift in the carrier frequencies of the signals from the four satellites. This can be done in the carrier tracking loop. The three unknowns, (V_x, V_y, V_z) plus the receiver clock frequency error, f_u can be solved. The the state variables of the Kalman filter can then be $\vec{x_t} = \{U_x, U_y, U_z, b_u, V_x, V_y, V_z, f_u\}$. The observation vector is represented by $\vec{z_t} = (\mathrm{PR}_i, \mathrm{RR}_i)$ where RR_i is the observed relative velocities or range rates. For more detailed explanations of the navigation solution, see [27, pp. 8-1 to 8-7] and Milliken & Zoller [25].

2.3.1 Dilution of Precision

Since the navigation solution provides the user only with an estimate of the receiver's position, the answers are not exact. A measurement of the accuracy of the calculated position is given in the covariance matrix minimized by the Kalman filter:

$$\begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 & \sigma_{xz}^2 & \sigma_{xt}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 & \sigma_{yz}^2 & \sigma_{yt}^2 \\ \sigma_{zx}^2 & \sigma_{zy}^2 & \sigma_{zz}^2 & \sigma_{zt}^2 \\ \sigma_{tx}^2 & \sigma_{ty}^2 & \sigma_{tz}^2 & \sigma_{tt}^2 \end{bmatrix} \tag{2.17}$$

Typical indications of the accuracy are the dilution of precision (DOP) factors which are derived from the diagonal terms of the covariance matrix. The Geometric Dilution of Precision (GDOP) indicates the effect of geometry on user position error which includes the 3-D positioning error plus time error. PDOP is the dilution of precision in 3-D. The HDOP is the dilution of precision in the horizontal plane. VDOP is the DOP in the vertical dimension. TDOP is the DOP in the range equivalent of the receiver clock offset. They are related to the terms in the covariance matrix by the following formulae:

$$\text{GDOP} = \sqrt{\sigma_{xx}^2 + \sigma_{yy}^2 + \sigma_{zz}^2 + \sigma_{tt}^2} \tag{2.18}$$

$$\text{PDOP} = \sqrt{\sigma_{xx}^2 + \sigma_{yy}^2 + \sigma_{zz}^2} \tag{2.19}$$

$$\text{HDOP} = \sqrt{\sigma_{xx}^2 + \sigma_{yy}^2} \tag{2.20}$$

$$\text{VDOP} = \sigma_{zz} \tag{2.21}$$

$$\text{TDOP} = \sigma_{tt} \tag{2.22}$$

Some literature gives the HDOP and VDOP in terms of geodetic coordinates, that is northings, eastings and height [15]. In this case, using a transformation matrix, one can convert the covariance matrix from xyz coordinates to geodetic coordinates and $HDOP = \sqrt{\sigma_{nn}^2 + \sigma_{ee}^2}$ and $VDOP = \sigma_{hh}$. By multiplying the DOP's by the user equivalent range error (UERE), the positioning error is calculated:

PDOP	×	UERE	=	Radial error in position in 3-D
HDOP	×	UERE	=	Radial error in position in the horizontal plane
VDOP	×	UERE	=	Vertical error in position
TDOP	×	UERE	=	Error in range equivalent of user clock offset

The determination of the UERE will be discussed in the next section discussing sources of error. In normal operation, UERE is about 10.8 to 13.9 meter 1σ. Typical DOP errors are PDOP < 6, HDOP < 4, VDOP < 4.5 and TDOP < 2 [27, pp. 3-3 to 3-5]. In selecting the four satellites to use for the navigation solution, the receiver picks the four satellites that will minimize the PDOP. The PDOP is approximately equal to the inverse of the volume of the tetrahedron formed by connecting the user position to each of the four satellites. By choosing the 4 satellites which generates the tetrahedron with the maximum value, the receiver will select the best 4 satellites.

The configuration that will maximize volume is one satellite at zenith (90 degrees elevation angle) and the other three satellites 120 degrees apart at a low elevation angle. See Spilker [36] for more details.

2.4 Errors in Solution

Sources of error in the navigation solution can arise due to numerous factors. However, this can be overcome since the propagation of the signal through the ionosphere can typically be modeled. The ionospheric delay is dependent upon the elevation angle of the satellite – the lower the elevation angle the larger the delay. It is also dependent upon atmospheric conditions resulting in less delay at nighttime. When there are large atmospheric disturbances, however, difficulty can arise in generating suitable modeling parameters. If one was using a dual carrier receiver, the propagation delay can be computed as mentioned in the last section. Otherwise, single frequency users can use the ionospheric correction parameters to correct for the delay. In addition to the ionosphere, the carrier signal also experiences a delay in propagation through the troposphere which increases as elevation angle decreases but is independent of frequencies. The delay can be modeled as a function of elevation angle and, if available, as a function of local weather conditions. Therefore, although DOP measurements favor very low elevation angles, lower angles result in longer propagation delays. The signal can be interfered with as well by blocking structures which reflect the GPS signal resulting in a longer travel time and a longer pseudorange measurement. This error is commonly referred to as multipath error which is a major problem in cities, woody areas and over water.

Another source of error is due to ephemeris errors. Due to the nonidealities in the satellite orbits previously mentioned, the broadcast ephemeris parameters are only least-squares curve fit and therefore inherently has some errors. Also, there may be some perturbations in the satellite orbit due to unmodeled occurrences. There are also some SV clock errors that have not been modeled by the correction parameters, but these are very slight. In addition to errors arising on the SV, there can be some errors in the receiver hardware due to noise and inaccuracies. Table 2.6, from [27, p. 3-3] lists the UERE budget.

In addition to physical errors, there are ones that are man made. Errors can arise inadvertently from receiver or satellite hardware failure. However, the greatest error in the Navigation Solution, is an artificial one introduced by the control stations. By the FRP, Selective Availability (SA) was activated. As of this date, there are no plans to turn it off. SA degrades GPS position determination capabilities by altering navigation message data and by dithering the SV's clock frequency. Most of the error is due to clock dither since the noise varies more greatly and therefore is not as systematic and predictable. After FOC, AS will be implemented on all Block II Satellites. SA and AS are not implemented on Block I satellites. However, since Block I satellites are being phased out, any potential benefits from avoiding AS/SA from the remaining Block I satellites are inconsequential. With S/A degradation, GPS performance is reduced to only within 100 meters (95%). This is the equivalent

Segment Source	Error Source	P-code Pseudo Range Error 1σ [meters]	C/A-code Pseudo Range Error 1σ [meters]
Space	Clock and Navigation Sub-system Stability	3.0	3.0
	Predictability of Satellite Perturbations	1.0	1.0
	Other	0.5	0.5
Control	Ephemeris Prediction Model Implementation	4.2	4.2
	Other	0.9	0.9
User	Ionospheric Delay	2.3	5.0 - 10.0
	Tropospheric Delay Compensation	2.0	2.0
	Receiver Noise and Resolution	1.5	7.5
	Multipath	1.2	1.2
	Other	0.5	0.5
System UERE 1σ [meters]	Total (RSS)	6.6	10.8 - 13.9

Table 2.6: The GPS System Range Error Budget

of 31.3 meters UERE, or equivalently 104 ns TDOP [27, p. 12-17]. The velocity degradation due to S/A is classified. Requests for "SA/A-S free vehicles" have been denied by the U.S. Coast Guard which administers the civilian interface to GPS service for the Department of Transportation[7].

2.5 Differential GPS

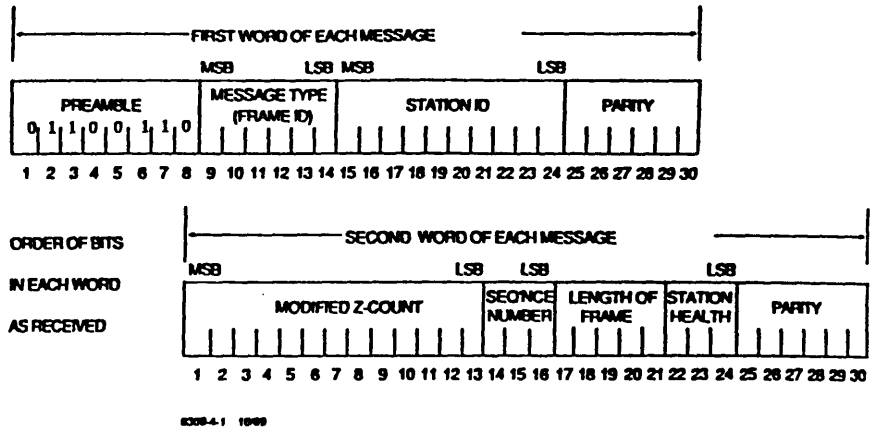
To overcome the errors introduced by Selective Availability, C/A users who need better than 100 meters of accuracy can use differential GPS to correct for the errors. Differential GPS (DGPS) uses a stationary reference receiver which precisely knows its position at a pre-surveyed site. This receiver calculates the errors in the pseudorange measurements and their rate of change for each visible satellite and broadcasts the information to differentially-capable receivers in the area. Besides correcting for S/A errors, differential GPS can also compensate for common error sources such as that from the Space and Control Segment. Also, if the receivers are in the same area as the differential base station, atmospheric and ionospheric errors can also be removed. The further the users are from the reference station, the less accurate will the differentially-corrected positions be. When DGPS starts to degrade, the difference is about 250 km between the reference station and the DGPS receivers, but corrections can still be used effectively by receivers as far as 800 km [27, pp. 11-1 to 11-7]. It has been recommended that the data format for the broadcasted corrections follow a standard data format in order to allow compatibility between receivers and different manufacturers of differential reference stations.

2.5.1 RTCM-104 Standard–Version 2.0

The recommended data format for differential corrections, RTCM SC-104 was set by the Radio Technical Commission for Maritime Services Special Committee-104. It was designed to follow the same format as the GPS satellite navigation messages. It has the same 50-baud rate, parity algorithms, 30-bit word size and word format. However, unlike the GPS messages of fixed 10-word subframe message sizes, RTCM SC-104 allow for variable subframe length. This is due to the fact that the length of the correction message depends on the number of currently visible satellites.

There are two versions of RTCM SC-104 standards, the second version providing some slight improvements over the first release. The following paragraphs describe version 2.0 of RTCM SC-104. At the end of this section, the differences between the two versions will be described briefly. For more details regarding the standard refer to [34] and Kalafus, Van Dierendonck, Pealer[14].

There are a total of 64 possible message types generated by a differential reference receiver. However, only 21 are currently defined. They are listed in Table 2.7 [34, p. 4-5]. The first two words in each message are used for synchronization and message identification. The format of the two words is shown in Figure 2-4 [34]. The first 8 bits, the preamble, serve the same synchronization purpose that they did in the header of GPS navigation messages. However, for differential corrections the 8-bit



From **RTCM Recommended Standards** [34, p. 4-3].

Figure 2-4: Two-Word Header for All RTCM SC-104 Messages

preamble is set to be `0x66`. The next 6 bits are used for identifying the type of message in the subframe. The next 10 bits are station identification bits so that in case the corrections are being broadcast at the same frequencies as other differential base stations in the area are broadcasting on, receivers have some way of discriminating between base stations. The last 6 bits of the first word is, as in the navigation message, parity bits. The first 13 bits of the second word is the modified Z-count, which indicates the time of the start of the next frame of data as in the navigation message, and serves as a reference time for the time the corrections were generated. There is a difference between the Z-count in the RTCM-104 message and that of the GPS navigation message in that the scale factor for the former is 0.6 seconds as opposed to 6 seconds for the GPS navigation message to compensate for the variable frame length. Its range is also only one hour in order to conserve bits since the Z-count. The next 3 bits represent a sequence number between 0 and 7. The sequence number is used for synchronization when the differential receiver is not transmitting corrections in pseudolite mode. The sequence number for each message should be in numerical order. Therefore, if the first message has a sequence number of one the next message should follow sequentially with the number two. If not, it means the data was somehow corrupted. The next five bits represent the length of the frame $N + 2$ where N represents the total number of words which follows. The next 3 bits represent the station health. (See RTCM SC-104 Recommended Standards [34] for states.) The last 6 bits serve as parity bits. Table 2.8 (modified from [34]) summarizes the contents of the first two words.

Synchronization is done in a similar manner as for the GPS navigation message. The user searches the message until the preamble is located. To avoid a false positive indication of the start of the frame, the user must verify parity with the first two

Message Type No.	Current Status	Title
1	Fixed	Differential GPS Corrections
2	Fixed	Delta Differential GPS Corrections
3	Fixed	Reference Station Parameters
4	Tentative	Surveying
5	Tentative	Constellation Health
6	Fixed	Null Frame
7	Tentative	Beacon Almanacs
8	Tentative	Pseudolite Almanacs
9	Fixed	High Rate DGPS Corrections
10	Reserved	P-code Differential Corrections
11	Reserved	C/A-code L1, L2 Delta Corrections
12	Reserved	Pseudolite Station Parameters
13	Tentative	Ground Transmitter Parameters
14	Reserved	Surveying Auxiliary Message
15	Reserved	Ionosphere (Troposphere) Mesg
16	Fixed	Special Message
17	Tentative	Ephemeris Almanac
18-59		Undefined
60-63	Reserved	Differential Loran-C Mesgs

Table 2.7: RTCM SC-104 Version 2.0 Message Types

Word	Content	Bits	Scale Factor	Range
First	Preamble	8	NA	NA
	Message Type	6	1	1-64[4]
	Station Id	10	1	0-1023
	Parity	6	NA	NA
Second	Modified Z-count	13	0.6 seconds	0-3599.4 sec
	Sequence Number	3	1	0-7
	Frame Length (N+2)	5	1 word	2-33 words
	Station Health	3	NA	8 states
	Parity	6	NA	NA

Table 2.8: Content of First and Second Words of RTCM-104 Message Frames

words using the algorithm described in Section 2.2.2. Once parity has been verified, the user decodes the sequence number and length of frame to determine the location of the next frame for further verification. Once the frame data has been verified, the receiver is ready to decode the message.

2.5.1.1 Message Types

As seen by Table 2.7, only six of these messages, types 1, 2, 3, 6, and 16, have been defined in final fixed form and are the messages typically broadcast by reference stations. Therefore, we will only examine these six message types. See RTCM SC-104 Recommended Standards document for a discussion of the other messages.

Message Type 1, shown in Figure 2-5, contain the differential GPS corrections parameters PRC and RRC, where $\text{PRC}(t_0)$ is the pseudorange correction and RRC is the range rate correction. The total correction to be applied is:

$$\text{PRC}(t) = \text{PRC}(t_0) + \text{RRC} \times (t - t_0) \tag{2.23}$$

where t_0 is the 13-bit modified Z-count of the second word. The pseudorange can then be corrected as follows:

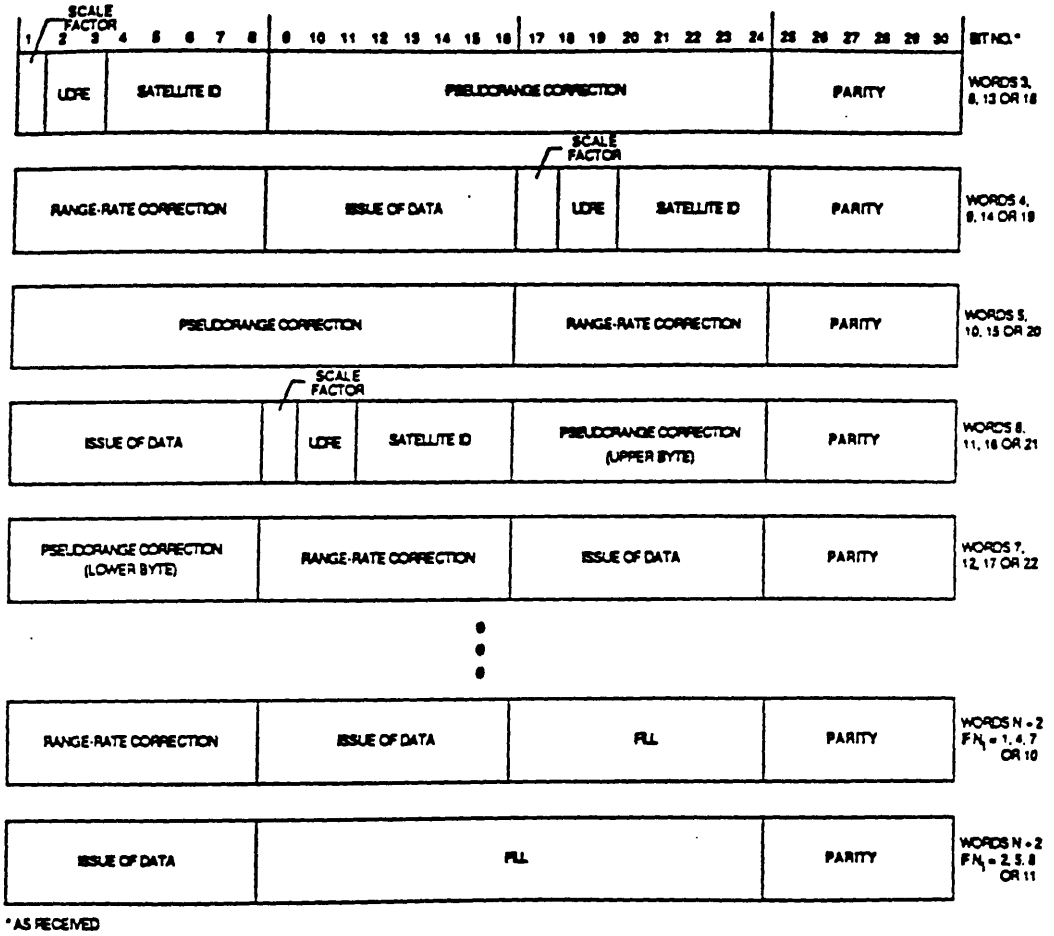
$$\text{PR}(t) = \text{PRM}(t) + \text{PRC}(t) \tag{2.24}$$

where $\text{PRM}(t)$ is the receiver's measured pseudorange, t is the time of the pseudorange measurement, and $\text{PR}(t)$ the corrected pseudorange. The $\text{PRC}(t_0)$ value is a predicted one whose validity decreases with time. The RRC is used to compensate for this. The basic contents of the message are summarized in Table 2.9 [34]. These 40 bits are repeated N times where N is the number of satellites ranges in the corrections. The remaining bits are filler bits and the 6 bits of parity for each word.

Parameter	No. of Bits	Code	Scale Factor & Units
Scale Factor	1	0	0.02
		1	0.32
UDRE	2	00	Error $\leq$ 1 meter
		01	Error $\leq$ 4 meters
		10	Error $\leq$ 8 meters
		11	Error $>$ 8 meters
SV ID	5		1
$\text{PRC}(t_0)$	16		0.02 or 0.32 by above
RRC	8		0.002 or 0.032 by above
IODE	8		See ICD [13]

Table 2.9: Contents of Type 1 Message

Message Type 2 follows a similar format as in Message Type 1. (See Figure 2-6.)



From **RTCM Recommended Standards** [34, p. 4-6].

Figure 2-5: Type 1 Message Format

This message contains the delta differential corrections which are used to maintain high levels of accuracy when new satellite navigation data has become available. It contains the difference in the calculated pseudorange corrections using the old navigation message and the new message. For more details on how this is used in adjusting the measured pseudorange in conjunction with Type 1 messages see [34] since we are predominantly concerned about the makeup of the message and not the implementation of it in the receivers. The contents of Type 2 message is similar to that of Type 1, except that Delta PRC and Delta RRC data is transmitted in place of PRC(t_0) and RRC data respectively.

Message Type 3 contains the reference station's Earth-Centered Earth-Fixed coordinates. This message is of a fixed size of 4 words (not including the two word header). 32 bits are used for each X,Y and Z coordinate with the remaining 24 bits used for parity.

Message Type 6 is the null frame message containing no relevant information. It is used as a filler. There is either 0 or 1 word containing an alternating sequence of 1's and 0's (this is the filling pattern) following the two word header of this message depending upon the type of fill required.

Message Type 9 is identical to Message Type 1 in format and content. It is used when there is a high rate of change in corrections. Typically the number of satellite ranges in this message is much less than that in Type 1, usually on the order of 1 or 2. It contains the corrections for the 1 or 2 satellites whose pseudorange is rapidly changing.

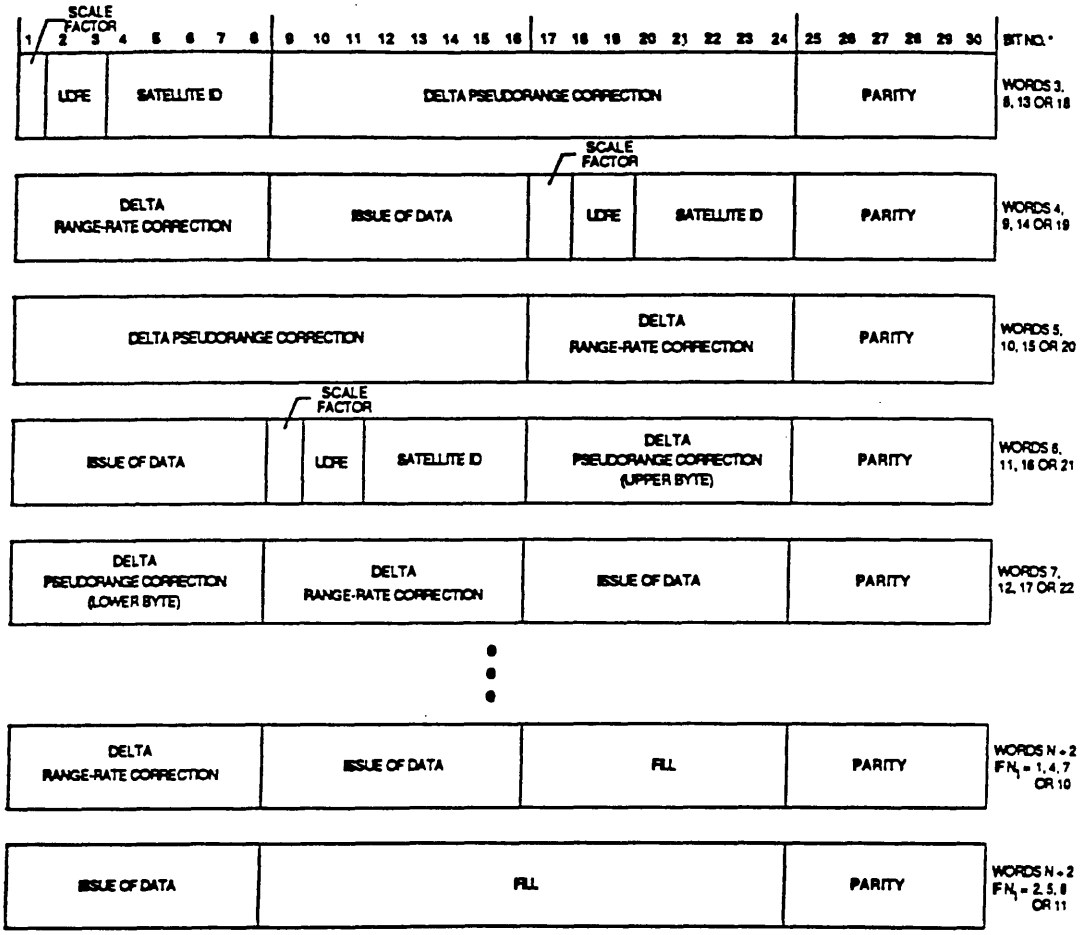
Message Type 16 is used for transmitting ASCII messages. Fill bits are zero in this case to avoid misinterpretation of the alternating sequence as an ASCII character.

2.5.2 RTCM SC-104 Version 1.0

Details of version 1.0 of RTCM SC-104 recommendations is given in Kalafus, et al. [14]. Primarily we are concerned with the differences between the two versions. One difference is that the number of possible messages was increased from 16 in Version 1.0 to 64 in Version 2.0 giving it more flexibility. The preamble was also changed from 0x8B in Version 1.0 to 0x66 in Version 2.0.

In Message Type 1, in Version 1.0, there is no scale factor. Instead, the scaling of the PRC(t_0) and RRC term was determined by the UDRE which consisted of 3 bits instead of 2 bits. The scale factor for the PRC(t_0) term is equal to $0.02 \times (2)^{\text{UDRE}}$ when UDRE has a value from 0 to 5. The scale factor for the RRC term is $0.002 \times (2)^{\text{UDRE}}$. In addition, the UDRE codes changed due to the greater range available to the UDRE. The new codings is shown in Table 2.10 from [14]. All other terms perform the same function. Changes in Message Type 9 from Version 2.0 is the same as changes to Message Type 1 since Message Type 9 performs the same function of providing additional pseudorange corrections when there is rapidly changing pseudoranges.

Message Type 2 in Version 1.0 does not have a Scale Factor term, UDRE term or the satellite id number. In addition, it does not have a delta range rate correction term. The scale factor for the delta pseudorange correction used is from the UDRE



From **RTCM Recommended Standards** [34, p. 4-11].

Figure 2-6: Type 2 Message Format

Code	Indication
000	Error $\leq$ 0.5 meters
001	Error $>$ 0.5 meters
010	Error $>$ 1 meter
011	Error $>$ 2 meters
100	Error $>$ 4 meters
101	Error $>$ 8 meters
110	See Kalufas, et al [14]
111	Do not use

Table 2.10: Satellite Health and UDRE for Version 1.0

term of the most recently received Message Type 1.

Message Type 3, 6, and 16 remain unchanged.

2.5.3 Equipment Interface

In addition to the message formats, the RTCM SC-104 recommended standard also specifies the user equipment interface. One recommended interface involves utilizing a data link to download differential corrections to the receivers. Messages are required to be transmitted at a rate of 50 baud. However, this could be continuous or in burst mode and therefore can be transmitted using Electrical Industry Association (EIA) RS-232-C or RS-422-A/RS-449 standards at a serial baud rate rate from 300 to 9600 baud with an 8 bit character structure.

Because of the 30-bit word size of the message, it means that the message needs to be reformatted before being transmitted over the serial communication line. Two formats are available to the user. One is an "8 of 8" format. The "8 of 8" format is not required of all equipment. It utilizes all 8 bits of data available. The other is an "6 of 8" format which is required on all equipment. In this format, only the 6 least significant bits (LSB) contains information leaving bit 7 set to "marking" or binary 1, and bit 8 set to "space" or binary 0. In addition the data is sent MSB first while the ANSI standard requires the LSB to be sent first. The data link can be a transmitter which broadcasts corrections to mobile receivers equipped with another antenna to pick up the corrections and extra hardware to do the processing.

An alternative method of downloading corrections is by using a pseudolite technique in which the reference base station behaves like a satellite. The reference station will broadcast differential corrections at the same frequencies as the GPS carrier frequencies. Its advantage is that no new hardware or antenna is needed since it is at the same frequency as the GPS signals. However, this method has the disadvantage that the receiver is limited to line of sight applications as if it was using a real satellite. See [34] for more details.

Another method is through the use of cellular phones and modems. It is similar to the data link method, except the means of transmission of the corrections is through

use of a cellular phone and modems. Corrections for a particular area will be available on a computer. The user connects to the computer via a cellular phone that is connected to a modem through a conversion box.[5] Once connected, the user can then download the corrections over the cellular phone link. The advantage of this system is that no special broadcasting licenses are required for transmitting the corrections, as it would be if one were to use a transmitter. The disadvantage is that the means of communication, by cellular phone, is not available in all areas and occasionally the carrier is dropped requiring the user to navigate without differential corrections. In addition, it requires a computer and modem to be connected to the GPS receiver for transferring the corrections. However, many applications are now integrating GPS receivers with computers that this requirement does not pose too much of a problem. Because of these reasons, we implemented our system by downloading differential corrections using this method. By broadcasting the differential corrections over the ethernet, a user, once logged on, accesses the corrections as a client program. Ethernet is simulated over the serial line through the use of PPP from Morning Star Technologies. Installation details are given in [26]. The differential corrections interface will be discussed in more details in the software chapter.

2.6 Geoids

All of the previous calculations were performed assuming an ellipsoidal model of the Earth which provided the user position in an Earth-Centered Earth-Fixed (ECEF) coordinate system. Unfortunately, most applications use a physical rather than mathematical model of the Earth. The physical model is represented as a geoid as opposed to the mathematically regular ellipsoid model and is based on the physical makeup of the Earth. This model is closest to that measured utilizing the standard levelling rod [24] which is based on local geographical variations which include topographic and gravitational variations. Therefore, areas measured to be the same level essentially lies on the same geopotential surface – that is where the gravitational field exerts the same force. There are many geopotential surfaces about the Earth (the gravitational force exerted is the gradient of the potential). However, the geoid is the equipotential surface having an average potential which approximates the mean global ocean surface the best. The reason the geoid is not an ellipsoid and instead is an irregularly shaped object is due to variations in the Earth's density (potential $= k \times m/d$ where m is a point mass, however the Earth is not a point mass and the potential at a particular point must be solved for by integrating over the volume) and the contributions to the potential from centrifugal force due to the Earth's angular velocity. See [15] for a more rigorous mathematical derivation of the geopotential equations.

The GPS, on the other hand, utilizes a mathematical representation of the Earth as an ellipsoid or geodetic datum. A geodetic datum is specified by 8 parameters – 2 for ellipsoid, 3 for location of the origin and 3 for the orientation of the ellipsoid.

[5]The conversion box is necessary to convert cellular phone signals to the standard phone RJ11 signals.

The geodetic datum used is the World Geodetic System 1984 (WGS84) developed by the U.S. Department of Defense as a global geodetic system as opposed to the regional datums which are valid over only a small region.[6] In the case of WGS-84, the semimajor axis is specified as 6378.137 kilometers and 1/f=298.257223563 where f represents the flattening of the ellipsoid. There exists formulas for conversion between the WGS-84 coordinate system and other geodetic datums. See [16] for more detail. Fortunately for the U.S., the North American Datum 1983 is very close to WGS84 model. The errors resulting in discrepancies between the two datums are very small in comparison to other errors which tend to arise when using GPS. WGS84 is approximately the same as the Geodetic Reference System 1980 (GRS80) adopted by the International Union of Geodesy and Geophysics (IUGG) in 1979 as the closest approximation to the Earth.

WGS-84 is centered at the Earth's center of mass and its z-axis is coincident with the Conventional Terrestrial Pole (CTP), the Earth's average pole position[7], and whose x-axis is orthogonal and passes through the Greenwich Meridian. Positions in WGS84 coordinates can be given in terms of Earth-Centered Earth-Fixed (XYZ) coordinates or in terms of geodetic coordinates – longitude, latitude, and geodetic or ellipsoidal height (λ, φ, h). Ellipsoidal height is the position of the point above the mathematically defined surface. Figure 2-7 shows how the two coordinate systems relate to one another. The conversion from geodetic coordinate system to ECEF is given by the following formulas from Leick [15, p. 184-185]:

$$X = (N+h)\cos\varphi\cos\lambda \tag{2.25}$$
$$Y = (N+h)\cos\varphi\sin\lambda \tag{2.26}$$
$$Z = (N(1-e^2)+h)\sin\varphi \tag{2.27}$$

where:

$$N = \frac{a}{\sqrt{1-e^2\sin^2\varphi}} \tag{2.28}$$
$$e^2 = 2f - f^2 \tag{2.29}$$

and a is the length of the semimajor axis.

The conversion from ECEF to geodetic coordinates is given by:

$$\tan\lambda = Y/X \tag{2.30}$$
$$\tan\varphi = \frac{(N+h)\sin\varphi}{\sqrt{X^2+Y^2}} \tag{2.31}$$

As you can see, Equation 2.31 is nonlinear and must be solved iteratively. By substi-

[6]These datums were optimized for the area of interest such as NAD27, European 1950, South American 1969, etc. [16] and were therefore only valid in these regions.

[7]The Earth's axis of rotation moves as a result of polar motion and therefore the average position of the pole is used as the z-axis.

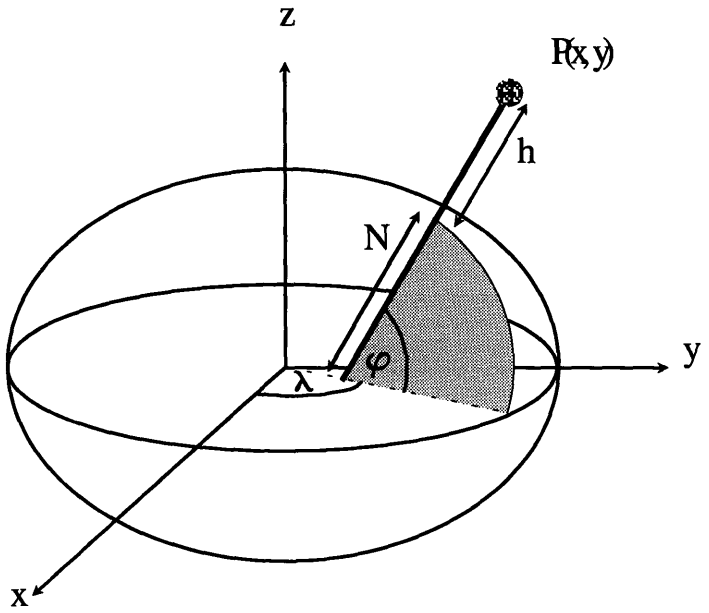


Figure 2-7: Relationship between ECEF and Geodetic Coordinates

tuting in Equation 2.27 and rearranging we get:

$$\tan\varphi = \frac{Z + e^2 N \sin\varphi}{\sqrt{X^2 + Y^2}} \tag{2.32}$$

By initializing the height, h, to zero and substituting in Equation 2.27, we can rewrite Equation 2.31 as:

$$\tan\varphi = \frac{1}{1 - e^2} \frac{Z}{\sqrt{X^2 + Y^2}} \tag{2.33}$$

and solve for φ. Then the calculated value of φ is substituted into and::

$$h = \frac{X^2 + Y^2}{\cos\varphi} - N \tag{2.34}$$

The computed value of φ is then used in the $\sin\varphi$ calculation in Equation 2.32 to calculate a new value of φ. With this new value, a new value of h is calculated. This process is repeated until h converges and the difference between the old h value and the new h value is less than some ϵ value. In our calculations, we used $\epsilon = 0.001$.

The above formulas give the transformation from ECEF coordinates to geodetic coordinates. However, the height that is calculated is the ellipsoidal height. Most geographical databases use altitude above mean sea level or the orthometric height. The orthometric height, H, of a point is the height measured along the normal from a level or equipotential surface, in this case the geoid. The measurement depends upon the topography and geological makeup of the region. The geoidal height or undulation is the distance of the geoid from the ellipsoid. Therefore, to calculate one's mean sea level altitude from GPS coordinates, assuming knowledge of the geoid height:

$$H = h - N \tag{2.35}$$

All that is required is to get an accurate model of the geoid height. An equipotential model of the Earth was computed by Richard Rapp and Nikolaos Pavlis at Ohio State University (OSU89A and OSU89B) from the geopotential Goddard Earth Model (GEM-T2). Its resolution is 0.5 degrees or 50 kilometers which is not enough resolution when dealing with GPS data. However, a high resolution model of geoid heights for conterminous United States was developed by Dennis Milbert of the National Geodetic Survey called GEOID90 by combining the high frequency components of the OSU89B model with 1.5 million terrestrial and water gravity data. For detailed explanation on the calculation of the GEOID90 model, see [23]. GEOID90 is referenced to GRS80 which as mentioned previously is very similar to WGS-84. It consists of three height grids:

Region	Latitude Range (N)	Longitude Range (W)
Eastern	24 - 50	66 - 90
Central	24 - 50	83 - 107
Western	24 - 50	101 - 125

The total dataset consists of 581 rows by 1281 columns. Its resolution is 3 minutes by 3 minutes which is approximately 5 kilometers. For finer resolution, the program, provided by NGS called GEOID, does interpolation on the bounding geoid heights to calculate the geoid height at a particular geodetic longitude and latitude. One must be careful when using these corrections since there still exists a systematic error often in the corrections. Therefore, in calculations of orthometric height, the resulting height will not be very accurate due to errors in OSU89B and biases in the measurements. However, these errors are small and need not be considered since the accuracy that is required for our application is on the order of meters as opposed to centimeters. The interfacing of the geoid height corrections with our system will be discussed in detail in the software chapter.

Chapter 3

The Navigation Unit

Our enhanced reality system consists of a network of portable and stationary computers. The portable computers connected to navigation tracking devices compose the mobile units. The stationary computer broadcasting differential GPS corrections is referred to as the central computer. Implementing an enhanced reality system, as mentioned earlier, can essentially be divided into two tasks – developing a *navigation unit* and developing an application programming interface (API) to the system. In this section, we are primarily concerned with the navigation unit (NAVunit). In the next section, we will discuss the API to the hardware.

An overall system block diagram of the mobile unit is given in Figure 3-1. The computing engine in the system is a portable computer. From the diagram, one can see that the portable computer serves a variety of functions. It drives the graphical display of information, provides means of communication with the navigation unit and communicates over the network with other computers. The navigation unit provides 3D position information to the portable computer. The notebook computer, processes this information and then, either displays it on the screen or broadcasts it to other computers, acting as a filter of the information from the NAVunit.

It is, however the NAVunit that is the heart of the enhanced reality system. It is the navigation unit that provides the user with the spatial information. The NAVunit consists of basically four components:

- GPS Receiver to provide a user's global XYZ coordinates.
- Compasses to provide the user's azimuth orientation angle.
- Tilt Sensors to provide the user's local elevation and roll orientation angles.
- Microcontroller to control the flow of information from the other three components to the computing platform

In designing the NAVunit, our prime design consideration was the issue of modularity. That is, we wanted to be able to consider the navigation unit as a black box that simply output the user's positional parameters. The computer only needs to know the output format and input format. The advantage of this approach lies in ease of upgradability. The computing platform and navigation unit can therefore be

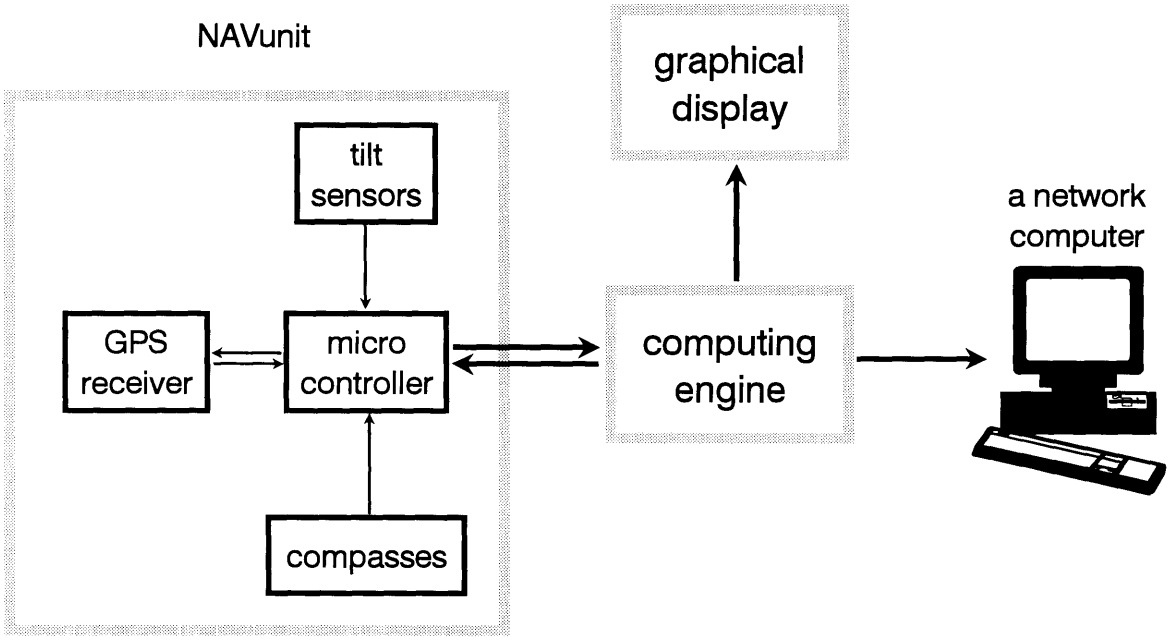


Figure 3-1: The Navigation Unit

upgraded independently of one another. In addition, a single navigation unit could also be easily swapped from one computing platform to another without changing the NAVunit's firmware.

Another design criterion was I/O format. In deciding the NAVunit's output format, we could design the unit to broadcast only the user's six position parameters. This would have the advantage of being more modular in design. However, this design has the major disadvantage of losing much of the information from the GPS receiver which provides more than just the user's XYZ location, such as the satellites' ephemerides. We decided the access to additional information was worth the loss in modularity. Since we want more information from the unit than just the XYZ location, we needed to be able to communicate to our unit in order to request the additional information. In addition, we also needed to be able to download RTCM SC-104 differential corrections to the GPS receiver in the unit.

We had two design considerations for the data controller. We could have used a bus design to send the data from the compasses, tilt sensors and GPS receiver as it is and leave the processing to the computing engine to format the information appropriately. Alternately, we could use a microcontroller which reformats the data first and then transmits it to the processing unit. We decided to use the latter scheme due to the fact that the computing platform is already involved in many other tasks such as communicating with the central computer, running location based application programs, communicating with the navigation unit, generating 3D graphics and/or performing Text-to-Speech Synthesis. In addition, the ease of translating preformatted data received from the navigation unit allows us to abstract the unit as a cohesive whole and utilize the output as it is without additional dithering of the algorithms. We decided to reformat our compass and tilt sensor information into TSIP format. Thus, our unit will behave as though it merely has an extended library of TSIP routines.

In the end, we decided that the best approach would be to make the navigation unit behave as a "super" GPS receiver which not only "knows" its XYZ position, but its local azimuth, elevation and roll angles as well. Communication between the unit and the computing platform follow EIA Standard RS-232-C interface since most computers have at least one RS-232-C compatible serial port. The sensors – compasses, tilt sensors and antenna – are in a separate unit from the logic circuitry. This is due to the fact that the logic circuitry at this stage is bulkier than the sensors. Since eventually we want to head-mount the sensors, and therefore keep the heavier logic circuitry separate such as in a backpack, we will separate the sensors from the unit even in the prototype.

3.1 Computing Platform

The decision of which computing platform to use for our computing engine was based primarily on code portability issues. All of the development software was developed on a Sun workstation. For this reason, we chose to use a SPARCbook 2 as the portable computer in the mobile unit. Because the SPARCbook 2 is binary compatible with

the workstation, we did not need to develop two sets of code for the stationary and mobile units. In addition, the SPARCbook 2 also has several positive features such as an 8-bit frame buffer and an active color LCD display for graphics, a built-in modem and Ethernet interface for communication, and speaker output, and microphone input for audio. There was one major disadvantage in using the SPARCbook 2 and that was the fact that it only had one RS-232-C compatible serial port. This required that we have the capability of sending RTCM SC-104 corrections and receiver configuration commands over the same serial line. Another disadvantage is that the SPARCbook 2 has a very short battery life of only about two hours and weighs over 7 lbs. In future implementations of the mobile units, we are considering using penpad and wearable computers for our computing engine.

3.2 GPS Receiver

The basic theory behind GPS technology was described in the previous chapter. Rather than expending resources in developing our own GPS receiver, we decided to use a commercial one. Currently there is a multitude of receivers available on the market, so much so that it prompted one person who deals with GPS to call GPS salesmen the equivalent of used car salesmen who will promise you centimeter resolution but neglect to tell you that it will take five minutes of post processing to do so. We decided ultimately on utilizing the Trimble Advanced Navigation Systems (TANS) `6-Channel` family of receivers due to Trimble's reputation and the fact that they provide one of the smallest receivers available on the market. Distinguishing Trimble from many of the various receivers commercially available was their user interface. Many of the other receivers provide the user only with the longitude, latitude and altitude coordinates on an LCD display, the better ones providing an RS-232-C compatible output port option. The Trimble receivers, on the other hand, provided a convenient serial I/O line with the ability to communicate to the receiver. In other words, other information, such as the contents of the navigation message, could be requested. The receiver operation mode could also be customized. Other features of the Trimble receivers are RTCM SC-104 differential operating modes, access to satellite ephemeris information, 0.6 second update rate, and use of purely C/A-code positioning algorithm.[1]

In the first stage in the development of the NAVunit, `6-Channel Trimble GPS Sensor Boardset SPS` receivers were used. The mobile units, however, consisted of SVeeSix Core Module (CM) SPS receivers which were much smaller (46.5 mm$\times$ 82.5 mm$\times$ 17.7 mm) than the Boardsets (122 mm $\times$ 202mm$\times$ 22 mm). Because the CM receivers follow the same communication protocol as the `6-Channel Boardset`, we were able to develop the test software on the more reliable and sturdy Boardset receivers first and then later switch to the CM GPS receivers with minimal effort.

To communicate with the boardset and the CM receiver, we utilized the TANS protocol or Trimble Standard Interface Protocol (TSIP).[2] With the CM we could have

[1] Many companies cheat by using the soon to be obsolete P-code to get a more precise positioning.

[2] They are identical protocols but since the CM is not considered a member of the TANS family,

also have used either an ASCII or NMEA-0183 interface. However, the TSIP interface was chosen due to the discovery that the CM module could run in differential mode while still allowing the user to retain input control only using this communication protocol. In order to implement differential GPS on the CM receivers using only one port, RTCM SC-104 corrections are downloaded to the receiver while the user would be receiving position information over the transmit line of RS-232-C port. However, since corrections are constantly being downloaded to the GPS, we would forsake any control requests from the user to the receiver since raw RTCM SC-104 data cannot be mixed with GPS command codes. Fortunately, we discovered an undocumented feature in the TSIP protocol which enables a user to transmit differential corrections along with control commands. This is done by reformatting the RTCM SC-104 corrections into TSIP format. This method will be discussed more fully in the software section since it was a software based solution.

The TSIP protocol is based on a binary packet protocol. Each message sent to and by the receiver must be sent in a packet which starts with `<DLE>` (`0x10`) followed by an 8-bit packet identification tag. Data is then transmitted over the serial line in 8-bit bytes, odd parity, one stop bit, LSB first. The end of the packet is marked by the sequence of `<DLE><ETX>` where `<ETX>` is the hexadecimal number `0x03`. In order to allow the value `0x10` to be transmitted, it must be followed by a second `0x10`. This protocol is used for transmitting to and receiving from the receiver.

The previously described TSIP protocol is currently followed by both the CM and Boardset receiver which enabled software development on the boardset. Data is transmitted at a rate of 9600 baud at TTL levels. It outputs position datum in WGS-84 Datum format at a rate of approximately once every 0.6 s. Running off 5V, the CM dissipates 1.5 watts with the antenna connected. The CM board weighs 32 grams. The antenna weighs 60 grams.

3.3 Compass and Tilt Sensors

The GPS receiver only provides the user's global position in ECEF coordinates. However, to determine the user's viewing direction, need a means of extracting our local orientation angles , (φ, θ, ϕ), at a known translational position [Figure 3-2]. For the first model of our system, we used combinations of digital compasses to extract one's azimuth and combinations of tilt sensors to extract the elevation and roll angles.

3.3.1 Compass

The digital compasses, 1" $\times$ 1" $\times$ 0.5" flux-gate magnetometers designed by International Navigation Inc. (INI), output the N (numerator) and D (denominator) components of the tangent of the azimuth angle at a given point with respect to the Earth's magnetic pole. These components are proportional, respectively, to the magnitude of the East and North components of the Earth's magnetic field at that point. The

Trimble renamed the TANS protocol to Trimble Standard Interface Protocol.

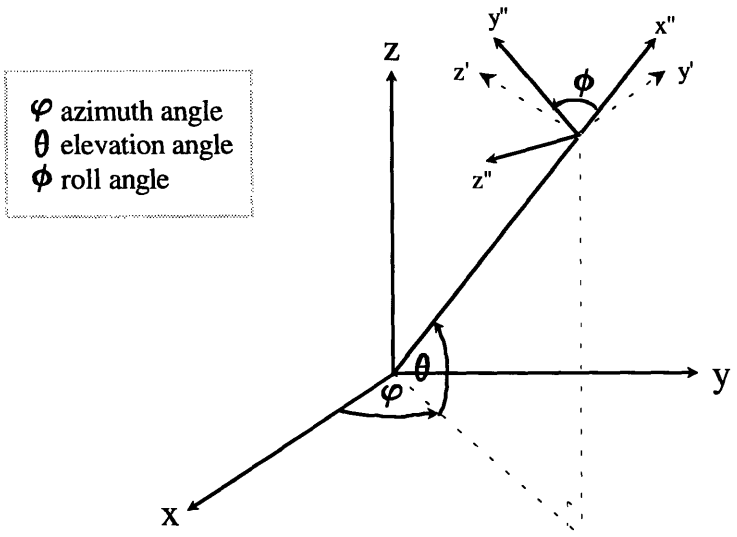


Figure 3-2: Six Parameters of Freedom

compass operates by driving periodic signals through two amorphous magnetostrictive cores arranged orthogonal to one another. Sensing circuits then measure the time between reentrant induced pulses in each core. The time between these pulses is a function of the magnitude of the Earth's magnetic field component. (See Farrell [4] for more details on the operation of the compass.) These timing values, minus a bias term, are then averaged and output as the N and D components over a serial line at a rate of 20 kbaud with the most significant bit first (MSB). A total of five bytes are sent 34 times/second. The first two bytes represent the N component in two's complement integer format. The following two bytes consist of the two's complement integerized D component. The last byte contains the status bits in the first four bits and the checksum in the last four bits. In addition to the serial data line, the digital compass generates a 34-Hz clock pulse output line whose rising edge is synchronized with the start of a data message. Figure 3-3 shows the timing characteristics of the serial line with respect to the pulse line.

One can calculate their azimuth position, the number of degrees that they are tilted away from true north in a clockwise direction, by taking the arctangent of the ratio of N to D. This is assuming that core axis #1 lies along the North axis and core axis #2 along the East-axis components of the horizontal intensity component of the Earth's magnetic field. This is because D is the filtered measured time value for core axis #1 and N is the filtered measured time value for core axis #2. The calculated angle will then be the number of degrees the Earth's horizontal field is away from axis #1.

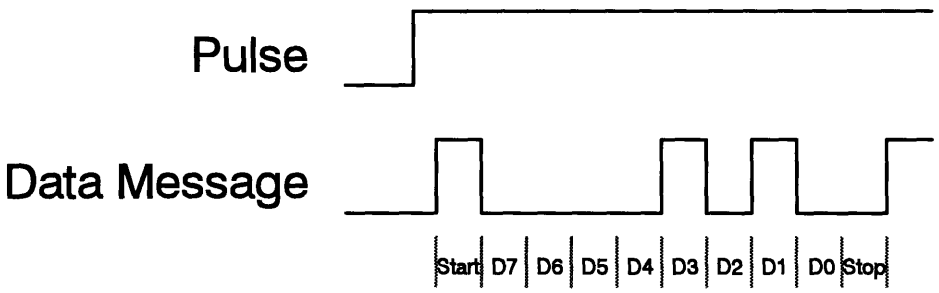


Figure 3-3: Timing Diagram for Compass Output

However, the measurement is accurate only if the compass axes lie in a horizontal plane tangent to the Earth's surface. If the compass is tilted perpendicularly, it would be unable to properly resolve the user's orientation due to the fact that one of its primary coils would be perpendicular to the Earth's magnetic field and therefore the measured magnitude would be close to zero. To overcome this difficulty, we utilized a combination of 2 digital compasses orthogonally aligned [Figure 3-4]. The first compass is aligned in the horizontal plane parallel to the keyboard of the notebook computer since the primary mode of operation of the unit will be on a flat or on a close to flat plane parallel to the Earth's surface. The second compass is placed perpendicular to the first one such that if one were to orient the SPARCbook 2 in a vertical position, the second compass would lie in the horizontal plane that was expected to be the second primary mode of operation.

By using two compasses arranged as in Figure 3-4, one is able to resolve the magnitude of the three components of the total magnetic field at a particular point since we receive four measurements of N_1, D_1, N_0 and D_0, where one set of measurements, N_0 and N_1, are redundant. We can then resolve the magnetic force vector from this information. Knowing the magnetic force vector in addition to the tilt of the compasses from the plane horizontal to the Earth from the tilt sensors, one can then project the vector to the horizontal plane and then calculate the number of degrees the computer is facing from true north. Therefore, only using the compasses in conjunction with the tilt sensors can the user's azimuth angle be resolved.

Letting $\vec{F}$ represent the magnetic force vector,

$$\vec{F} = \{N_0, D_0, -D_1\} \tag{3.1}$$

Using our elevation and roll angle information, we can generate two rotation matrices and multiply $\vec{F}$ by the matrix to get the new $\vec{F}$. Let's assume the following:

$$\begin{aligned} \varphi &= \text{azimuth rotation angle about the z-axis} \\ \theta &= \text{elevation rotation angle about the y-axis} \\ \phi &= \text{roll rotation angle about the y-axis} \end{aligned}$$

We use Euler angles to describe our orientation with respect to the global reference frame where rotation is done in the order where rotation about the z-axis is done

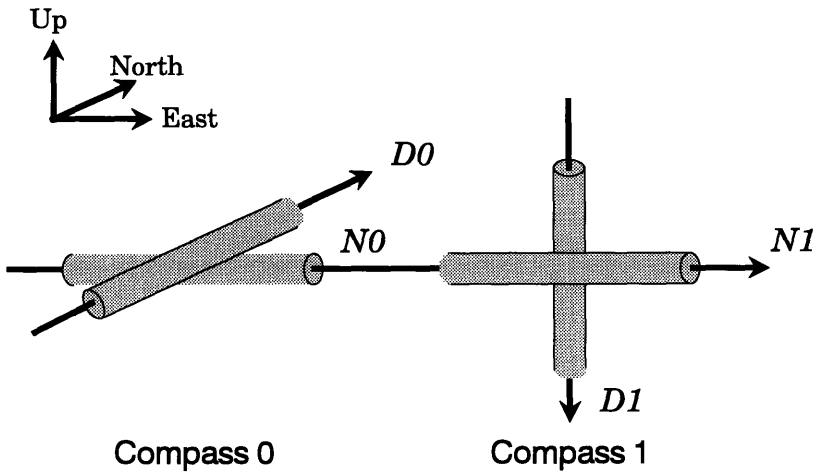


Figure 3-4: Compass Configuration

first, followed by rotation about the y-axis and then x-axis.

Assuming that θ and ϕ will be positive for counterclockwise rotation about their respective rotation axes, the transformation matrices needed to transform the magnetic force vector into a coordinate system parallel to the horizontal plane are:

$$\Theta = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.2}$$

where Θ represents the transformation matrix to compensate for the elevation angle and:

$$\Phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{3.3}$$

where Φ represents the matrix compensation for the roll angle. To calculate the magnetic force vector in a coordinate system parallel to the horizontal plane, we perform the following transformation:

$$\hat{F} = \Theta\Phi\vec{F} \tag{3.4}$$

resulting in:

$$\begin{bmatrix} \hat{F}_x \\ \hat{F}_y \\ \hat{F}_z \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta\sin\phi & -\sin\theta\cos\phi \\ 0 & \cos\phi & -\sin\phi \\ \sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \begin{bmatrix} N_0 \\ D_0 \\ -D_1 \end{bmatrix} \tag{3.5}$$

Therefore, to calculate the azimuth angle from the magnetic force vector components, we take:

$$\varphi = \arctan \frac{\hat{F}_x}{\hat{F}_y} \tag{3.6}$$

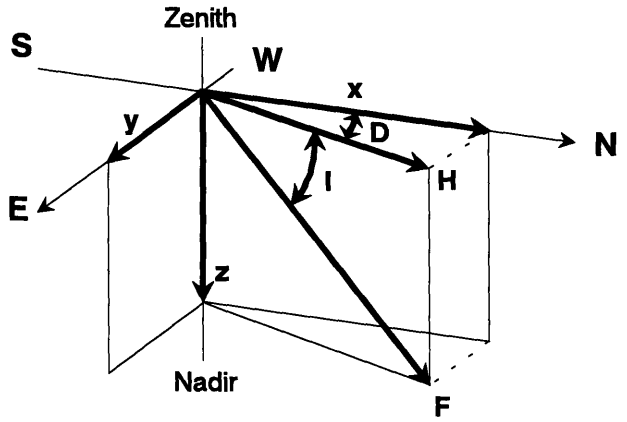
The above equations only hold true if the N_1 and N_0 components were identical. Unfortunately, this is not the case due to differences in the two compasses in both physical composition and in their compensation which can lead to differences on the order of 1000 out of a range of -32768 to 32767 which is about a 1.5% error margin. To compensate for the discrepancies we use the average value of N_0 and N_1 for the x-component of the magnetic field.

These equations provide the magnetic force vector at the user's location. Therefore, the azimuth angle calculated from the ratio of the north-south component to the east-west component is not the azimuth from true north celestial terrestrial pole (CTP) but from the geomagnetic north which is not at the same point as true north. Because the Earth's magnetic poles vary over time, a model of the magnetic field must be used which reflects variations over time in addition to position. Corrections to the magnetic force vector can be then obtained and applied to obtain the user's rotation angle from true north. The corrections are calculated using routines in the program GEOMAG provided by the National Geophysical Data Center which uses the International Geomagnetic Reference Field (IGRF) model of the Earth's magnetic field. These routines were extrapolated from GEOMAG and made into functions which generated and applied corrections to the compass measurements.

Given geodetic coordinates, the GEOMAG routines return the seven parameters of the Earth's magnetic field at that particular location. These consist of the declination, inclination, horizontal intensity, vertical intensity, total intensity and the north and east components of the horizontal intensity [Figure 3-5] For our particular application, we are particularly interested in the declination, the angle between the horizontal intensity of the Earth's magnetic field and true north. To calculate the user's orientation from true north, the declination is added to the calculated azimuth. More details of the interfacing of the corrections to the compass measurements will be discussed in the next chapter.

3.3.2 Tilt Sensors

The tilt sensors and their electrical interfaces were constructed by Robert W. Jebens of AT&T Bell Laboratories. The nine tilt sensors consist of small glass bulbs filled with electrolytic fluid with three output terminals. The tilt of the sensor is determined by measuring the resistance across the terminals which is dependent upon the amount of fluid between them. Effectively the three terminals behave like the three terminals



D	Declination	x	North Component
I	Inclination (or dip)	y	East Component
H	Horizontal Intensity	z	Vertical Intensity
F	Total Intensity		

Figure 3-5: Seven Parameters of the Magnetic Field

of a balanced bridge. For example, if the sensor is leveled, the fluid is balanced and therefore the resistance at the terminals is completely balanced. If the terminals are tilted, the resistance between the terminals will be different. The tilt sensor is connected to operational amplifiers which generate a voltage between 2.5 volts and 5.0 volts if the sensor is tilted in one direction and a voltage between 0 and 2.5 volts if the sensor is tilted the other way. The voltage response can be approximated as a nonlinear cubic function of the tilt angle of the sensor. The analog circuitry interface consists of an analog output line for each tilt sensor whose voltage roughly corresponds to its tilt. To calculate the tilt of each sensor, an inverse cubic function of the voltage reading is used. For our purposes, it is sufficient to model the tilt sensors as a black box that outputs voltages that swing about 2.5 volts. Since the midrange voltage reference point depends highly on the quality of the power supply, we have an additional midrange voltage reference line from the tilt sensor circuitry that should be 2.5 volts for a 5-volt power supply.

The range of the tilt sensors is from -70 to $+70$ degrees with respect to the gravity vector. To resolve local orientation, 360 degrees range is needed in all directions. Using a configuration of 3 orthogonal planes each with 3 sensors set 120 degrees apart from one another for a total of 9 sensors, one should be able to measure 360 degrees of movement in 3D. Figure 3-6 displays the configuration of the tilt sensors where the up arrows represent the upright position of an individual sensor. Each sensor is labelled 1 through 9. Next to each plane of tilt sensors are the orientation axes of the horizontal plane of the entire tilt sensor unit. The upright position of the tilt sensors ($\theta = 0$, $\phi = 0$) is considered to be when tilt sensor 7 is upright. When tilt sensors 1 and 4 are upright, the board plane is orthogonal to its normal position with the y-axis pointing down.

The algorithm to decode the nine tilt sensor readings is simplified due to the fact that only two of the nine tilt sensors are relevant for any particular orientation, one corresponding to the elevation and the other to the roll angle. Using an algorithm developed by Robert P. Lyons of AT&T Bell Laboratories, the two relevant sensor readings are extracted, the others discarded. Using these two values, the algorithm produces the gravity vector components with respect to the tilt sensor plane. Assuming θ and ϕ represent elevation and pitch angles respectively, we can use the same matrices Θ and Φ that were given in Section 3.3.1 where the rotation angles are in the opposite directions. When the tilt sensor plane is unrotated, as shown in Figure 3-7, the gravity vector is $\vec{G} = (0, 0, -1)$. The values provided by Lyons' algorithm is then:

$$\vec{G'} = \Phi(-\phi)\, \Theta(-\theta)\, \vec{G} \tag{3.7}$$

Therefore, the vector components are:

$$G'_x = -\sin\theta \tag{3.8}$$

$$G'_y = -\sin\phi\cos\theta \tag{3.9}$$

$$G'_z = -\sin\phi\cos\theta \tag{3.10}$$

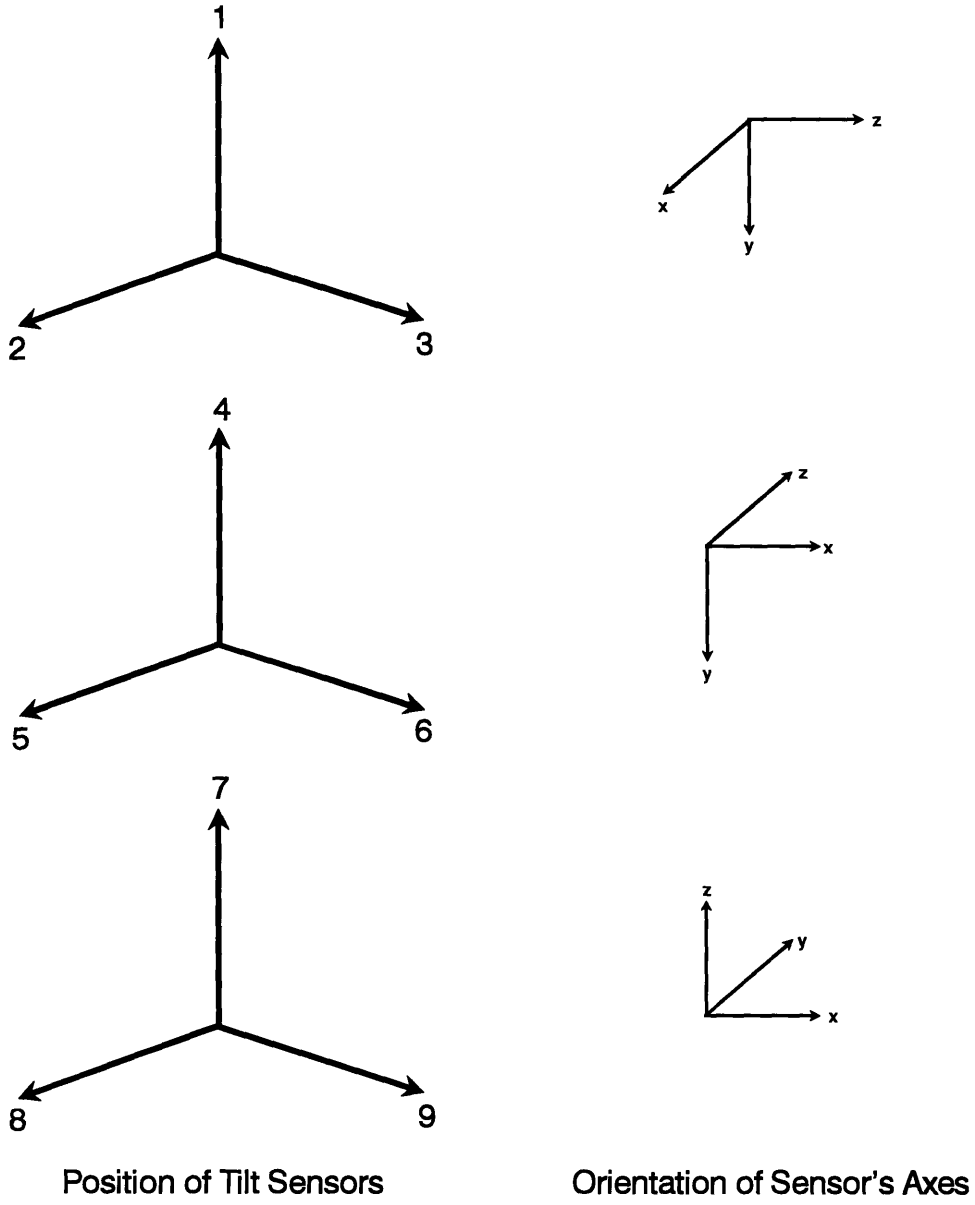


Figure 3-6: The Tilt-Sensor Layout

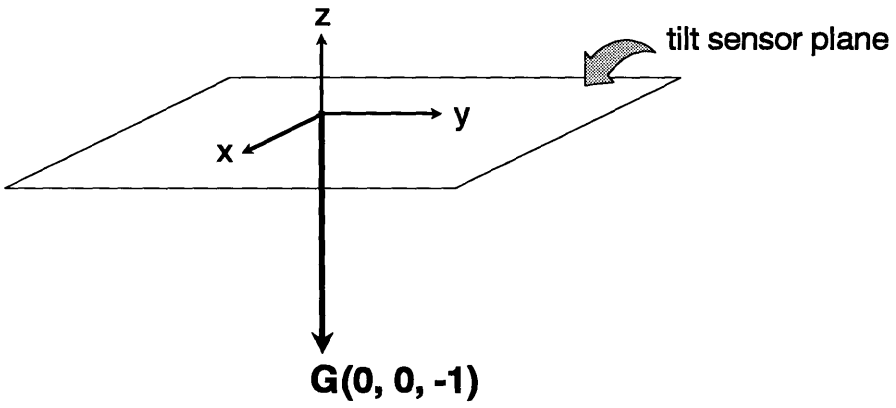


Figure 3-7: Gravity Vector

To compute the elevation and roll angles, we solve:

$$\theta = \arcsin -G'_x \tag{3.11}$$

$$\phi = \arctan \frac{-G'_y}{-G'_z} \tag{3.12}$$

3.4 Microcontroller and Associated Hardware

Controlling the data flow between the GPS receiver, compasses and tilt sensors is a microcontroller. We are using a NMIS-0021B CPU board based on an F68HC11E9 microcontroller running in expanded multiplexed mode. This particular microcontroller was chosen for its sophisticated on-chip peripherals and ease of adding external peripheral devices. The microcontroller board was purchased from New Micros, Inc. (NMIS) which manufactures peripherals specifically for the 68HC11 chip family on Vertical Stacking boards of size 2" $\times$ 4" at a stacking height of 0.75" between adjacent peripheral boards.

The F68HC11E9 is an 8-bit microcontroller with five 8-bit communication ports. The ports are labelled ABCDE respectively. In expanded multiplexed mode, two ports (BC) are not available for use as communication ports but are used instead for addressing lines. Port A can be used for four input capture units and three output compare units or four output compare units and three input compare units with one pulse accumulator input or fifth output compare unit. In the design of the navigation unit, the first mode of operation for Port A was utilized. Port D provides the Serial Communications Interface (SCI) and serial peripheral interface (SPI). The SCI was used for communications between a terminal and the board

during program development but was not used afterwards. We did not use the SPI. Port E is used for static inputs and/or as 8 A/D channels. This port was also not used. There are also 7 8-bit CPU registers and two 8-bit accumulator registers that can be used in conjunction for 16-bit operations. In addition, there are 512 bytes of on-chip EEPROM memory and 512 bytes of RAM memory. See Figure 3-8 for a memory-map of the CPU.

The microcontroller was not the most user-friendly in terms of programmability. The development of program code was straightforward due to the availability of the on-chip SCI which enabled the use of one serial RS-232 tty line at 9600 baud rate to download code into memory. The microcontroller could be programmed in either the high-level language MAX-FORTH V3.5, provided by NMIS, or in machine language.[3] In the navigation unit, the microcontroller was programmed in FORTH for non-time critical subroutines. However, for the time critical routines, each routine had to be machine coded by hand for optimal performance due to the inefficiencies in the FORTH stack language.

In expanded multiplexed mode, the code to be executed by the microcontroller is typically stored externally where more memory is usually available. The code was developed first using an SRAM for external memory. Once the code is downloaded, the execution of the program is started from another terminal using the SCI. However, the SRAM eventually loses the data (in this case the data is the program) that is stored in memory over time. Even if the memory was battery-backed, due to the delay associated with the battery-backup device, the program would not be able to autostart the desired code upon a power cycle or a reset. To overcome this problem, two external data storage components – one an SRAM and the other an EEPROM – were used. Storing the program on the EEPROM was the best way to insure that the program would stay in memory over extended periods of time and be able to activate fast enough for the microcontroller to enter autostarting mode.

The microcontroller (MCU) has two start-up operating modes – autostart and interactive. Autostart mode causes the program upon the detection of a specific sequence of bytes upon reset or power on to jump to the memory address in the next byte and to start execution from there. If these bytes are not detected, the program will default to its default start up state which is to be in interactive mode using the SCI. The autostart bytes, `0xA44A` were set in address location `0x8000` which is the starting address in the external EEPROM. Upon startup, when these bytes are detected, the microcontroller resumes execution starting at the address stored in the two bytes immediately following it. This was done by storing the address of the code to be executed in memory location `0x8002`. SRAM memory is also needed for the use of fast-access memory since the EEPROM does not allow the storing of data during real-time execution.

[3]By machine language, we literally mean the hexadecimal representation of each instruction and data since there was no assembly language interface to the microcontroller.

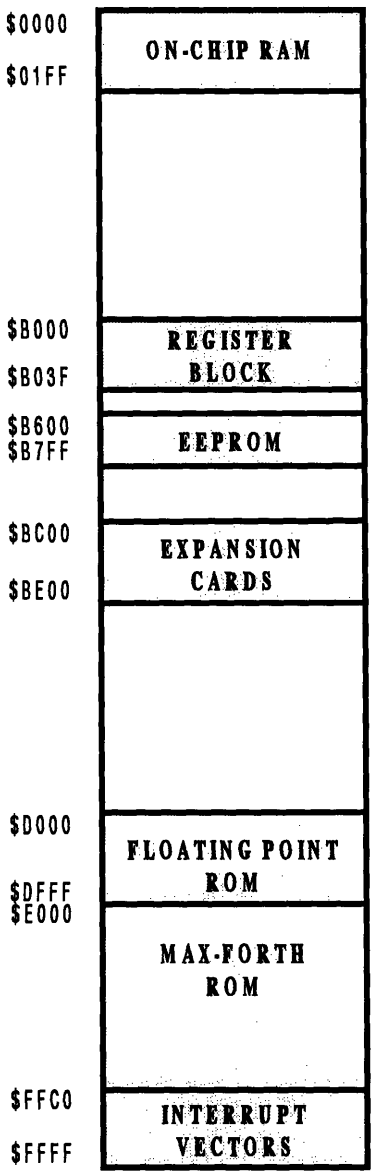


Figure 3-8: Memory Map of Microcontroller Unit

3.4.1 Compass Interface

In order to read in the compass information, there existed two possible approaches – serial and parallel. In a serial implementation, the compass information would be read in by the CPU, preempting other operations from continuing until all 50 bits have been sampled. By parallel, the data will be read by an external device that does not require any CPU time but will signal the MCU that data has been read in. The serial approach can be implemented by using a PIO port. Through a PIO port, the 34 Hz clock pulse would generate interrupt events to the MCU who would then enter a loop sampling the lines. The problem with this design is that 2500 μs of processing time would be consumed, most of it idling in a loop waiting for data bytes. Because data are coming from many sources, this would not be an optimal design. The parallel approach can be implemented utilizing a UART. The disadvantage to this approach is that it involves external circuitry. However, the trade off in less software complexity for the extra circuit complexity was considered to be worth the savings in CPU time. Consequently, UART's were used to read data in from the compass data line.

Each data bit in the compass data line is 50 μs long resulting in the constraint that the maximum total timing error that can be tolerated before losing synchronization is $\pm 25\,\mu$s. The total transmission time for the five bytes consisting of 10 bits each (8 bits of data plus one start and one stop bit) is 2500 μs. Thus, the crystal safety factor is $\frac{25}{2500} = 0.01\%$. Therefore, effectively, in order to avoid loss of synchronization, the rate of sampling the data can vary at most by 0.01% which means the sampling of the data line can be off by at most 200 Hz leading a baud rate range between 19.8 kbaud and 20.2 kbaud.

For the serial to parallel data conversion, the NMI-5002 DACIA card which is based on the Rockwell 65C52 Dual Asynchronous Communications Interface Adapter (DACIA) was used. Unfortunately, the range of valid baudrates for the compass are not within the range of standard baud rates. As a result, an external clock was used to drive the external receive clock line RxC (pin 25) on the DACIA. Since the DACIA does an internal divide by 16, a clocking frequency of $16 \times 20 = 320$ kbaud is needed on RxC [Figure 3-9]. Using an 8 MHz crystal oscillator to clock the 74HC390 Dual 4 Bit Decade Counter in Bi-Quinary mode [Figure 3-10], a divide-by-25 counter was generated. Figure 3-11 shows the clock signal generated. From the diagram, one notices that RxC does not have a 50% duty cycle. This would not cause a problem in our application since the DACIA is rising edge sensitive and the edges occur at 320 kHz. TxC (pin 15) was grounded since communication to the digital compasses was not used.

In addition to problems as a result of the odd baud rate required to sample the data, difficulties result from the fact that the serial data line does not really follow any accepted protocol (i.e. NRZ or LSB). The DACIA assumes the incoming data is NRZ format as shown in Figure 3-12, where the start bit is low and the stop bit(s) are high. However, as was shown in Figure 3-3, the start bit is high and the stop bit is low. In order to read in the signal, the data line from the compasses must be inverted first. But this results in inverted data as well. Undoing the inversion of the incoming data was accomplished by the microcontroller firmware. In addition, the bits need to

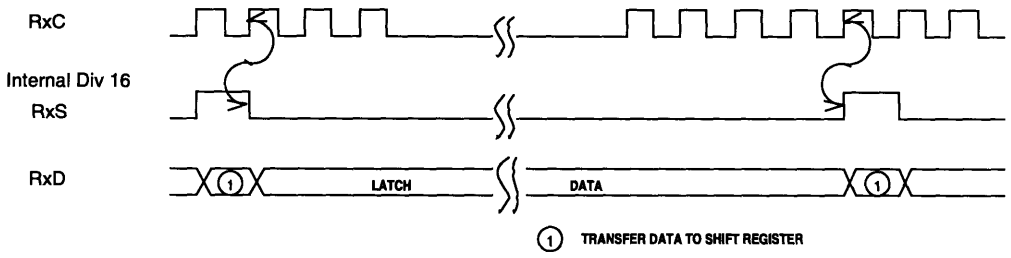


Figure 3-9: DACIA External Clock Timing – Receive Data

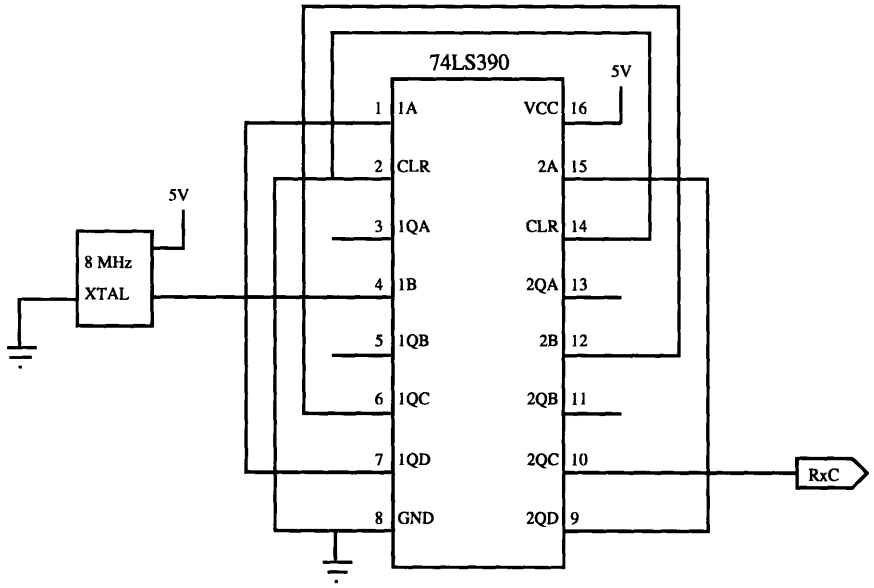


Figure 3-10: Circuitry Generating Compass Clock Signal

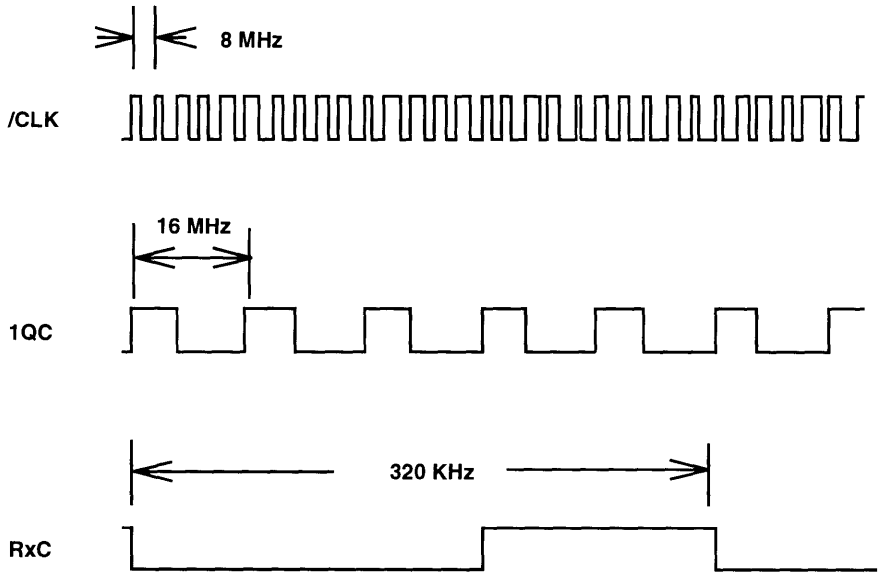


Figure 3-11: Timing Diagram for RxC Signal

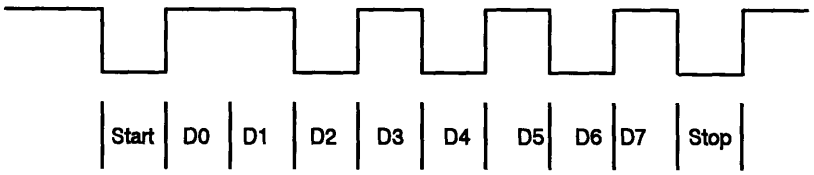


Figure 3-12: NRZ Data Format

be reversed since typically serial data is sent least significant bit first. The reversal of bits was also accomplished through software. The preprocessing of the compass data will be discussed in the next section on the microcontroller firmware.

3.4.2 Tilt Sensor Interface

The interface of the nine tilt sensors to the microcontroller is relatively straightforward. Taking the voltage outputs from each sensor, we connected the lines to two NMIS-4004 12-bit 8-channel A/D boards using an HADC574Z A/D converter. The converter had a maximum conversion time of 25 μs and power consumption of 150 mW. The outputs from the tilt sensors are in the range [0V, 5V]. However, the A/D board accepts only ranges [0V, 10V], [-5V, +5V], or [-10V, 10V]. Therefore, for maximum resolution, we set the A/D operating mode for inputs in the range [0V, 10V] and amplified the incoming voltages by a gain factor of two. In addition, since we are using 9 voltage readings, we need at least two A/D boards because each board can only multiplex up to 8 analog lines. To account for the characteristics of the A/D converters on each board, we connected the 2.5V reference voltage line to each and scaled each voltage reading by $\frac{2.5}{V_{ref}}$ for each board. The voltage reference lines are read along with the tilt sensors.

All nine tilt sensors plus the two reference voltages are transmitted in the same packet as the compass readings since the three orientation angles are coupled. However, since all eleven A/D conversions must be completed before the compass readings can be transmitted, our update rate will be reduced. This can possibly be overcome by having the A/D conversion done in parallel rather than multiplexed and serialized. To do this in parallel one would need 9 A/D converters and 9 data registers to store the digitized readings. The conversion cycle can be tied with the 34 Hz clock from the compasses. Then, when the readings are needed one could cycle through the registers rather than wait for the conversion of all 9 voltages. However, this is at the expense of bulk and power consumption. As a compromise, only two A/D's were used in parallel since we needed at a minimum 2 boards to begin with. We divided the 11 signals into 6 lines on one board and 5 lines on the other board rather than an 8 and 3 division in order to allow maximum parallel conversion.

3.5 Microcontroller Firmware

The microcontroller firmware was written in MAX-FORTH V3.5 provided by New Micros, Inc. In addition, much of the machine dependent code, such as setting interrupt vectors was hand written in machine language. The main routine in the microcontroller is an infinite loop which is entered after a few initialization instructions. At startup, the microcontroller enters a delay loop before the interrupt vectors can be set due to the fact that the digital compasses has an initialization delay. The interrupt vectors should only be set after the compasses are ready to generate interrupts to the microcontroller. After the delay, the MCU resets all flags, variables and registers, initializes the interrupt-vector lookup table, and initializes the RS-232 ports.

Once initialization is completed, the microcontroller cycles in an infinite loop waiting for an interrupt to occur. The system was designed to be run on interrupt basis, thereby, creating a pseudo multitasking operating system. Although there are four basic tasks, there are only three interrupt routines – one to handle data from the host to the MCU, one (actually two) to handle data from the compasses and one to handle data from the GPS. The last task is the transmission of information to the computing engine from the MCU. This is not an interrupt routine because it is a routine in the infinite main loop. This allows data to be transmitted only when there is no data waiting to be read in which allows the maximum amount of CPU time to be spent trying to read in data which is arriving asynchronously from many sources. This routine in the main loop will be discussed later in this section. All interrupt routines were written in machine language.

3.5.1 Computer to Microcontroller Communication

A second NMI-5002 DACIA card was used with one port for communication between the GPS CM module and the MCU and the other port for communication between the computing engine and the MCU. In this section, the routines for transmitting data from the computer to the MCU and for transmitting data to the GPS CM receiver from the MCU will be discussed. The routine to transmit data from the MCU to the computing engine will be discussed in the end of this section. Data from the CM is received at 9600-baud rate, 8 data-bit words, odd parity and one stop bit at TTL levels. Although RS-232 levels were available, they would dissipate more energy since it requires data to be transmitted at levels of $\pm$12V. Since data transmission was over a distance of only a few centimeters, noise would not be much of a concern and therefore 5V levels would provide us with enought reliability. Since the NMI-5002 card was equipped already with level shifters, the card was modified to conserve energy. The data format between the computing engine and the MCU was set to be compatible with the data output of the GPS CM receiver. In this manner, the software interface written for the navigation unit can be used directly with a CM receiver. This assisted in the debugging of the software interface and also made the system more flexible.

The interrupt routine handling the data transfer from the computing register is invoked when the ACIA's /IRQ2 is asserted. In the initialization routine, along with

data format, the second DACIA was programmed to generate an interrupt when the RDRF in the interrupt status register (ISR) is set, indicating that data is ready to be read from the receive data register. The interrupt routine reads in the received byte, and clears the interrupt flag. Any data from the host computer is meant only for the GPS receiver because none of the other devices are commandable. Therefore, any and all bytes are directly transmitted to the GPS receiver. In addition, because there are no other devices communicating with the GPS receiver, there is no need to worry about corruption of data from other devices. It should be noted that an interrupt is also generated when the ACIA detects a framing or receive overrun error (bit 1 in the ISR). By checking if bit 1 has been set in the ISR, we only transmit data to the receiver if it has not been set. Otherwise, we clear the flag by reading the receive data register (RDR) and then returning to the main routine.

3.5.2 GPS to Microcontroller Communication

The transfer of data from the GPS receiver to the MCU is similar as for transmitting data from the computing engine to the MCU. An interrupt routine is invoked when the ACIA /IRQ1 is asserted. As above, at initialization, the first ACIA in the DACIA was programmed to generate an interrupt when the RDRF in the ISR is asserted. The routine to handle transfer of GPS information to the host computer is entered upon the signalling of RDRF on the GPS DACIA. Instead of transferring the data directly to the computing engine, the data is stored in a memory buffer of 127 bytes.[4] Ideally, the data should be transmitted directly to the computing engine as in the case of above where data was transmitted directly to the GPS. However, the algorithm is not as simple when transmitting to the computing engine due to the fact that the compasses and tilt sensors are also trying to send data to the host. The difficulties involved will be discussed later in the section dealing with the routine which transmits data to the computing engine.

3.5.3 Compass to Microcontroller Communication

As mentioned previously, a NMI-5002 DACIA card was used for communicating between the compasses and the MCU. There are two interrupt routines to handle signals from the two compasses but since they are identical, it is sufficient to discuss only one routine. The compasses produce readings 34 times/sec. In the average operation of the navigation box, the user will not require this much time resolution. In addition, there may be timing difficulties in terms of the serial I/O port on the SPARCbook 2 in that if it is running very computationally intensive applications, the SIO port on the SPARCbook 2 can become backlogged, creating SILO overflow errors. This in turn will lead to losing bytes of data which would cause the entire data packet to become corrupted and discarded. Therefore, sampling the compasses at a much lower rate of about 3 or 4 times per second should solve the problem of data overflow

[4] 127 bytes were chosen since the F68HC11E9 is an 8-bit processor and it is easiest to detect when 127 bytes have been stored.

without significantly reducing resolution. This was achieved through counting the number of interrupts generated by the compasses and performing the actual parsing of data on the ninth interrupt.

Interrupts are generated by a low-to-high transition on two of the input capture lines on Port A which is connected to 34-Hz clock lines from the compasses. There is approximately 39 μs between the rising edge of the signal and the first start bit of the first start byte. When a byte is received the word is placed in a 5-byte buffer. At initialization, one of the interrupt lines is disabled in order to avoid contention between the two compasses and to prevent any one compass from dominating the other by constantly generating interrupts a few microseconds before the other compass. As a result, the two compasses are coupled. Once a compass's information has been read, its interrupt line is disabled and the interrupt line of the other compass is then enabled. When the other compass's information has been parsed, its interrupt line is disabled and the tilt sensors are then sampled. The cycle of the compasses then repeats upon reenabling of the first compass's interrupt line which is done on completion of sampling the tilt sensors.

Parsing compass data is not as simple as for the data from the computing engine. At initialization, the compass DACIA was programmed to have an external RxC clock, 8 bits of data, 1 stop and 1 start bit with no parity bits. Once the compass data has been read in from the RDR, the data must be inverted and then bit reversed. The inversion of the 8 bits is done in a single command in the MCU. However, to do the bit reversal, a lookup table was used in which the table address added to the number to be reversed would contain the 8 bits reversed. The byte is then stored in a buffer in SRAM memory. This process is repeated 5 times until all five bytes have been read in and stored.

The five bytes are then passed through a checksum test. Figure 3-13 shows an example checksum test. If the data has been read in properly, the last four bits of the fifth byte should equal the exclusive or of the first four bits with the last four bits of the exclusive or of the first four bytes with the status nibble (first four bits of the fifth byte padded with four zeroes). If the checksum does not equal the transmitted checksum the 5 bytes are discarded. If the checksums are equal, then the first compass is disabled, the second compass enabled and the process repeated. A flag is set upon completion of parsing the second compass line. This flag will notify the main routine that compass data is has been read in and to start sampling the tilt sensors.

3.5.4 MCU to Computing Engine Communication

Once compass and GPS data information has been received and stored by the MCU, the MCU must transmit the information back to the computing engine where the data will be processed and distributed among several application clients. Because there are asynchronous devices trying to communicate to a single port, the MCU needs to maintain the data in a format in which the computing engine would be able to translate the data back to its original sources. We decided that GPS data should be given precedence in transmittal due to the fact that GPS data can be several bytes of information, whereas compass and tilt sensor information is a set size of 30 bytes for

	00001001	N High Byte
XOR	01110110	N Low Byte
	01111111	
XOR	11111001	D High Byte
	10000110	
XOR	11111111	D Low Byte
	01111001	
XOR	01010000	Status Nibble
	00101001	Transfer High
XOR	0010	Nibble
	1011	CHECKSUM

Figure 3-13: Checksum Example

the compass data and tilt readings. In addition, we have limited control over when to sample the compass information, whereas the GPS data comes independently. For these reasons, the main transmission routine loops infinitely checking to see if the transmit-data register to the computing engine is empty [Figure 3-14].

Once the transmit data register is empty, the MCU checks to see if there is new data to be read. This is done by maintaining pointers, one to the last data to be stored, GPS_CNT and another to the next data to be transmitted, GPS_OUT. If GPS_OUT equals GPS_CNT, it means effectively that all the data in the memory buffer has been transmitted. If they are not equal, the MCU then outputs the byte stored in the address pointed to by GPS_OUT. In this manner data is transmitted as soon as it is received by the MCU to the computing engine.

However, we run into problems if we try to output compass and tilt data in the same fashion. This is due to the fact that the MCU may be engaged in transmitting data from the GPS data memory buffer when the compass information is ready to be transmitted. For example, if compass and tilt data were to be transmitted directly to the computing engine, there is a risk that they would collide with GPS receiver data that is currently being transmitted. If GPS data is currently being transmitted, the compass information could become corrupted [Figure 3-15]. To prevent such a collision from happening, the compass and tilt data is packetized into a single message by waiting for all 10 bytes of compass data and all 22 bytes of tilt information to be received before we transmit the data back to the computing engine.

When all 10 bytes of compass data have been read in, COMP_FLAG is set, signalling the start of sampling of the tilt sensors. The nine tilt sensors plus two reference voltage readings are sampled in the main loop. In order to optimize performance, we effectively steal idle cycles from the MCU. While the MCU waits for the transmit data register empty (TDRE) flag on the ACIA to the computing engine, we sample one tilt sensor from each board. This is possible since the transmission of one byte of data at 9600 baud takes approximately 1 ms whereas sampling of the tilt sensors requires only 25 μs delay. Upon completion of sampling the sensors, we go back and check the TDRE flag. As long as the TDRE flag is not true, we sample the tilt sensors, keeping track of how many sensors have been read. When all sensors and voltage reference lines have been read,[5] a flag, TFLAG, is set indicating the completion of sampling. When TFLAG is set, we loop infinitely until the transmit data register is empty.

In addition to the GPS data possibly corrupting the compass information, we also have the possibility that an entire compass data packet will be broadcast in the middle of a GPS data packet [Figure 3-16]. Thus, we must also packetize the GPS receiver's information. The packetization of GPS data is done by preventing the transmission of compass information while GPS data is currently being transmitted. However, the GPS receiver data packets have variable length. Therefore, we need to check for the end of a packet before allowing compass and tilt information to be transmitted. This can be done by checking for the sequence <DLE> <ETX>. First, we check if the previous byte was of value <DLE> by checking if the DLE_FLAG was asserted. If it

[5]Since we have 11 analog lines to sample and we are sampling two at a time, the last voltage reading for the second A/D board will contain garbage information and was therefore ignored.

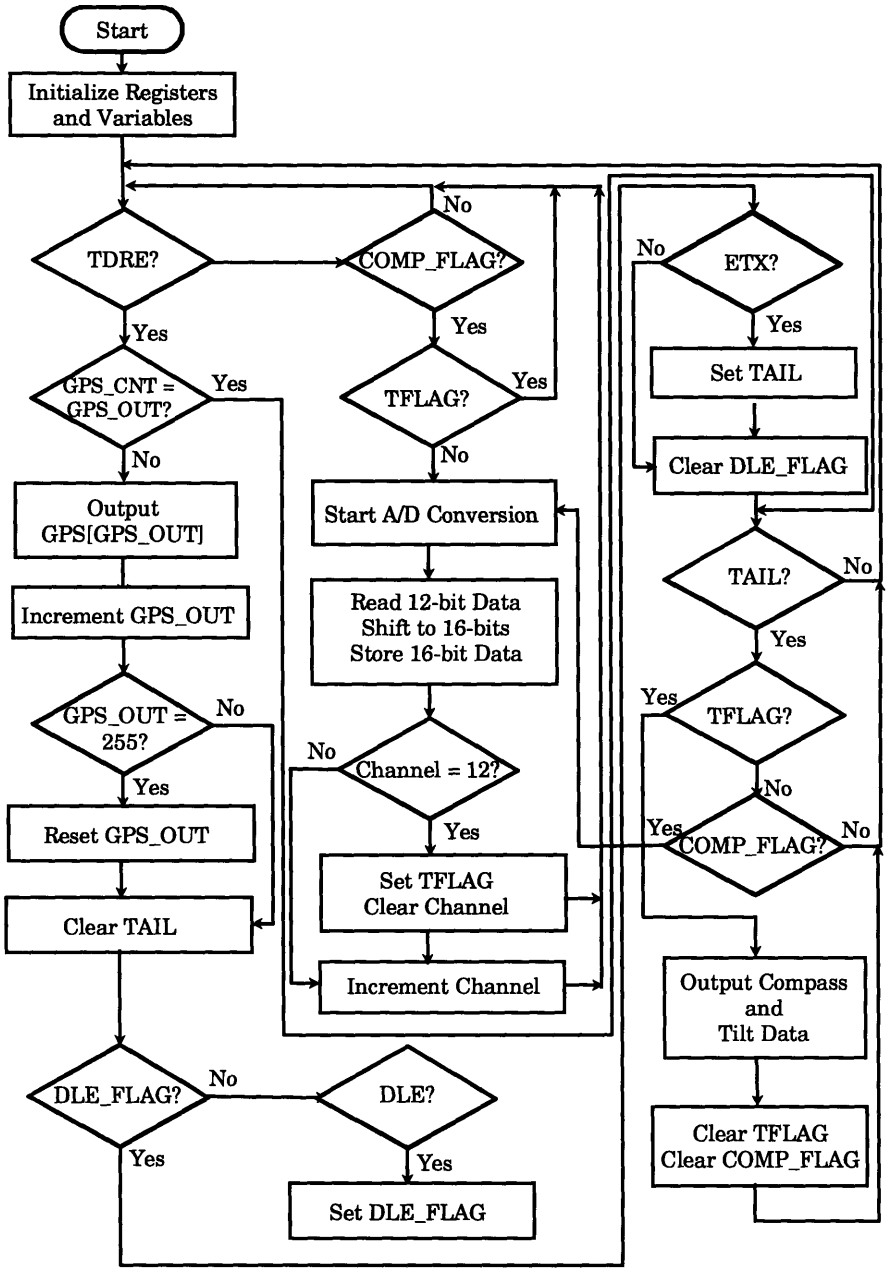


Figure 3-14: Algorithm to Transmit Tilt and Compass Information

Output Byte Stream	GPS Rcvr Data	Compass and Tilt Sensor Data
0	<DLE>	
1		N0 - High Byte
2		N0 - Low Byte
3	<opcode>	
4		D0 - Low Byte
•		•
•		•
•		•
n	<data>	
n+1	<data>	
•	•	
•	•	
•	•	

Figure 3-15: The Effect of GPS Data on Unpacketized Compass Messages

Output Byte Stream	GPS Rcvr Data	Compass and Tilt Sensor Data
0	<DLE>	
1	<opcode>	
2	<data>	
3		N0 - High Byte
4		N0 - Low Byte
5		D0 - Low Byte
•		•
•		•
•		•
n		Tilt 11 - High Byte
n+1		Tilt 12 - Low Byte
n+2	<data>	
•	•	
•	•	
•	•	

Figure 3-16: The Effect of Compass Messages on Unpacketized GPS Messages

has not been asserted, we check if the current byte's value is equal to <DLE>. If it is, the DLE_FLAG is asserted. Otherwise, it checks if the end of a packet had already been detected. If the DLE_FLAG had been already set and the current byte to be transmitted was <ETX>, the MCU indicates that the end of a packet has been reached by asserting the TAIL flag. Otherwise, the DLE_FLAG is cleared and the MCU proceeds to check if the end of a packet had already been reached. The TAIL flag is cleared as soon as a byte from the GPS receiver has been transmitted [Figure 3-14].

After each byte is transmitted, the MCU checks to see if the end of a packet has been set by determining if the TAIL flag has been set or not. If it has not, that means a packet is in the process of being transmitted to the computing engine and so returns to the main loop, first clearing TAIL flag. However, if the TAIL flag is set, it means that a complete packet has been set. The MCU then checks to see if the TFLAG

is enabled which signifies that sampling of the tilt sensors have finished. If the flag is set, the program proceeds to output the compass and tilt information. Otherwise, the MCU checks if the COMP_FLAG has been set indicating that new compass data has been read in. If COMP_FLAG is asserted, the program jumps to the A/D sampling routines described above. If neither flag is set, indicating that the compass and tilt data is old, the program loops back to the beginning [Figure 3-14].

To output the compass and tilt data, we transmit the information following TSIP format in one packet where the first 4 bytes represent the N and D components of the two axes of the compass lying in the horizontal plane, the next four bytes being the N and D components of the second orthogonal compass and the remaining 22 bytes being the voltage readings of the eleven lines for a total of 30 bytes being transmitted. These 30 bytes are preceded with the heading of <DLE> <opcode>, where the opcode is the hexadecimal number 0xB0[6], and followed by the sequence <DLE> <ETX>, signifying the end of a packet. Once transmission of a compass data has been completed, TFLAG and COMP_FLAG are cleared.

3.6 Navigation Box

Having developed the necessary circuitry and firmware, we need to put the entire system into a portable unit. The unit is constructed from black acrylic material. Figure 3-17 shows the front panel of the unit. It consists of an ON/OFF switch and a reset button for a soft reset of the system. There are also two ports, one for communicating with the SPARCbook 2 and the other to the the sensor unit containing the compasses and tilt sensors. The GPS antenna can be placed on the knapsack and does not need to be part of the sensor unit since a difference of a few inches between the sensor unit and the circuitry unit would not make a significant difference in the XYZ position since the resolution of the receiver is not that high to begin with. The antenna is connected to the GPS receiver antenna connection on the receiver board through a hole in the side of the NAVunit.

The I/O port communicating to the SPARCbook 2 is a standard male DB-25 RS-232-C compatible connection. The Sensor Port has two lines for compass data and two lines for the 34-Hz clocking signals from the two compasses, 9-tilt sensor voltage lines, plus two reference voltage lines, one 5V power line, and three ground lines. One ground line is connected to the tilt sensor board and the other two go to the compasses. Although one ground line would have been sufficient, it never hurts to have more than one ground line connection. The Sensor Port is a female DB-25 port connected through a ribbon cable to the sensor unit. The sensor unit consists of one board that has mounted upon it the 9 tilt sensors with their electrical interface and the two compasses.

[6]The opcode was arbitrarily chosen.

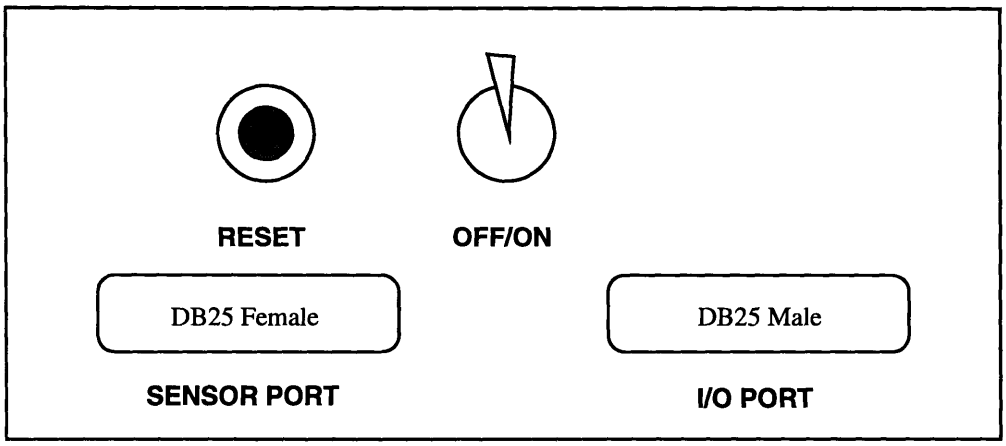


Figure 3-17: Navigation Box Front Panel

3.6.1 Power Considerations

In the design of the NAVunit, the power consumption of the unit was very important since in the design of a portable system it determines how long the system can be used without being recharged. The approximate power consumption of the components used in the Navigation Unit is shown in the following chart:

Component	Power (mW)
GPS CM receiver	1500
Compasses (2)	500
MCU and Peripherals	500
A/D (2)	300
Tilt Sensors	100
Total Power Dissipation	2900

Since many of these are approximate ratings and measured when the system was quiescent, a conservative estimate of the energy dissipation of the unit can be made to be approximately 0.25 amps. As mentioned in the earlier section, because the A/D converters require additional supply voltages of $\pm$12V, a triple-output DC/DC switching regulator is needed. This was the primary reason a 12V battery was chosen to power the system since it is more efficient to step down in voltage than to step up voltages. A DATEL TWR-5/1000-12/210-D12 DC/DC power converter was used to provide 5V and $\pm$12V at 81% efficiency resulting in a total draw of approximately 0.3 amps.

This amout of current is more than can safely be delivered by the SPARCbook 2 ethernet port, necessitating the need for a separate power source. Experience with the SPARCbook 2 demonstrated that at best, the computer could only operate 2 hours off its batteries. One could have included a separate power supply for the SPARCbook 2 in the Navigation Unit as well, however that would have violated abstraction barriers and destroyed the modularity of the design. In the end, a separate battery only for the Navigation Unit was decided upon. Using the very generous approximation for energy dissipation from above, a 12V 1.2 Amp-hour battery was chosen on the assumption that it should be able to power the system for about 4 hours reliably before the necessitating a recharge. Considering that the lifespan of the SPARCbook 2 on battery power is typically only about 2 hours, this battery life for the navigation unit is tolerable. The batteries are gel cel or sealed lead-acid batteries. These were chosen instead of Nickel-Cadmium batteries due to the greater ease of recharging gel-cel and lead-acid batteries. In addition, the interface of the digital circuitry to the battery was much more simpler as well.

Chapter 4

Enhanced-Reality API

The hardware aspects of an enhanced-reality system was discussed in the previous chapter. In this chapter, we will discuss the design and development of a software interface to the system. In the design of the interface, two primary considerations needed to be dealt with. First, we wanted an interface capable of simultaneously distributing position and orientation information from the navigation unit to an unlimited number of application programs. Second, each application program should be able to run independently and asynchronously of all the others.

In dealing with the two questions, we need to take into consideration that the computing engine has only one input port. As a result, the multiplexing of the information from the unit to many processes must be performed in software. Since most serial line drivers allow only one user to connect to it at a time, information is instead stored in buffers accessible by other programs. Questions involving the distribution of limited resources among several users are typically dealt with by multithreaded servers. Multithreaded servers are programs or processes that service many clients by creating new server instances or threads upon detection of connection requests. A daemon process is responsible for handling client requests while the new threads are responsible for actually servicing the clients. Multithreaded servers are different from single threaded servers in that single threaded servers handle only one client request at a time and in sequential order. Since one of our design constraints was the ability to run clients asynchronously and independently, a single threaded server model was ruled out.

Using a multithreaded server model, we were able to develop an interface which distributed information received from the navigation unit to several client application programs. In addition, a multithreaded server model is used for distributing differential corrections to local receivers. By using a multithreaded server as the data link, we are able to distribute corrections to an unlimited number of differentially capable GPS receivers, allowing greater positioning precision.

The basic algorithm for both servers follows essentially the same format. The differences lie mainly in the type of processing the main parent programs does and the type of services the client programs want performed. Both programs require the distribution of information from limited resources: the navigation unit and the reference base station. In addition, they both use the same techniques for shared-

memory allocation. For network information distribution, they spawn one or more child processes that behave as daemon processes whose purpose is to detect connection requests by hopeful clients. The number of processes that were forked off depends on the number of different types of services that each server provides. Upon detection of a client request, the daemon processes spawn other processes that are responsible for servicing the client requests. Meanwhile, the original processes are independently processing data that will eventually be used by the client programs. In the remainder of the chapter, we will first discuss the algorithm used by both servers and then we will go into details regarding the differences between the two programs. This discussion will assume that the user has some familiarity with network systems and the UNIX operating system. If the reader is unfamiliar with either one, see Stevens' *Unix Network Programming* [40] for more information.

4.1 Network Communication

Before plunging into the hairy implementation details of the software interface for the navigation unit and differential reference station, we will discuss briefly network communication. By network, we are referring to a communication system which connects two or more computers. This is essential to our server programs since these networks provide the means that will enable us to broadcast differential corrections from the reference base station that is connected to our central computer to the navigation units that are connected to the portable computers. In addition, the same protocols that enable two computers to communicate to one another enable two processes to communicate to one another, whether the processes are on two separate computers or on a single multitasking machine.

Networks are typically described by the International Standards Organization (ISO) open systems interconnection (OSI) model. This model divides the network protocol into seven layers to allow modular development as well as better abstraction for each layer. The seven layers are shown in Table 4.1 [40, 3]. For most of our applications, the first two layers, which constitute the hardware interface, will be utilizing an ethernet interface. However, we will also be utilizing PPP, a protocol using a serial line connection as the data link layer, when communicating to the mobile units. The network layer protocol used is the Internet Protocol (IP) which is responsible for routing datagrams from one machine to the other. However, it is an unreliable and connectionless protocol. If there is an error in the delivery of a datagram, the datagram is discarded, relying on the higher layers in the OSI model to perform error recovery. It should be noted that IP is not the only protocol used for the network layer, but in the implementation of our servers, this is the protocol that is used.

IP is the network layer for protocols utilizing TCP and UDP as the transport layers. Transmission Control Protocol (TCP) is a connection oriented protocol that provides reliable ordered delivery of messages. User Datagram Protocol (UDP), on the other hand, is connectionless and therefore less reliable than TCP. UDP is different from IP in that it has port numbers and has an optional checksum verification of

Layer	Name
7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data Link
1	Physical

Table 4.1: The OSI 7-Layer Network Model

the message. UDP is used when delivery speed of messages is more important than having the data actually arrive. TCP is slightly slower than UDP due to the overhead needed for keeping track of messages and for recovery from lost datagrams. TCP is used for long lived network connections where ordered delivery of packets is required while UDP is used for applications with repeated short operations [38, page 10]. Both protocols utilize port numbers which allows more than one communication link to use TCP or UDP at a time. The port numbers are essentially addresses which help the protocols resolve where to deliver data, allowing more than one process to send data at a time to different addresses.

The remaining three layers in the seven layer OSI model do not really concern us here. They are used for standardizing session management, data presentation and various application programs that should be able to run using the described network protocol. We are more concerned with the actual transfer of data among different processes using the transport layer. For our servers, we used TCP over UDP because of TCP's more reliable system which guaranteed delivery of messages. UDP users, on the other hand, have to check for themselves if the message arrives. Although UDP is faster, we were concerned more with the reliable delivery of data packets than in the speed of the delivery since data was typically only ready to be delivered once every 0.6 seconds due to the current GPS sampling rate. In addition, since we are using a binary packet protocol in sending TSIP packets, if a few bytes were lost, the entire message would become invalidated.

Although there are more than one application program interfaces to TCP/IP, we used Berkeley sockets for the simple reason that the author is more familiar with them than other API. Utilizing TCP/IP protocol, we implemented our servers. The main procedure in both programs forks off child processes for each different type of service we want to offer in each server. Each of the child services behaves as a daemon process since it runs in the background listening to connections on sockets that it opened. It is important to remember that the server must first initialize the socket and be listening for requests before a client program can connect to it. Since we are using TCP/IP protocol we must specify that we are using full-duplex connection oriented streams. Once the sockets have been established, the daemon processes, upon detection of a request, make new copies of the sockets that were being listened

on, fork children processes that close the original sockets and then proceed to service the client requests. (See Figure 4-1.) The original socket is closed in the child process since its purpose is to listen for connections and therefore, the child process has no need for it anymore. The child process will use the new socket created for data transfer.

In some implementations, daemon processes block until a connection has been made on the listening socket. However, we utilized a no-wait implementation in order to prevent the creation of zombie processes. Zombie processes are processes that have terminated but whose parents did not wait for them. Although the resources associated with such processes have been released by the kernel, the exit status and process ID's are still retained in order to inform the parent of the child's exit status. If the daemon processes blocked until a client connection request was detected and one of the child processes spawned to service a previous request had died, the terminated process exit status will remain in kernel memory until a new client requests a connection. Because we can have any number of clients[1], we can also have any number of clients all terminating before another client requests service. The terminated clients will then be consuming kernel memory. Therefore, to circumvent such a problem from occurring, we utilized the `select` system call rather than do our own polling of the socket which could waste resources. With the `select` system call, the program waits until the socket's file descriptor is ready for reading which indicates that a client connection request has been received by the system and returns setting a bit that indicates the socket is ready to be read. After one second of waiting, the `select` system call returns without setting any indicators.

4.2 Interprocess Communication

Sockets are a form of interprocess communication (IPC) that allow the exchange of information between processes, whether the processes are on a single computer system or on two or more computer systems. In order to implement our servers, we needed to utilize other forms of IPC's to allow the sharing of information from a single resource among several processes. For both the navigation unit and the reference station, the single resource consists of data arriving over a single serial line port. We need some means of distributing the data from this single source to many users. Sockets provided us the means to transport the data from the daemon processes to the client programs. However, the child programs, which actually service the clients, need a means of obtaining the information from the main parent process which has exclusive access to the serial line, first. One method would have the parent process make a copy of the original data for each grandchild process. However, this is rather slow and inefficient due to the extra copying time and extra memory involved. We decided that the optimal method would be through the use of shared-memory buffers. The information from the serial line is stored, either the raw data or after

[1]The number of clients is limited by TCP/IP protocol in how many unique addresses can be generated for each client.

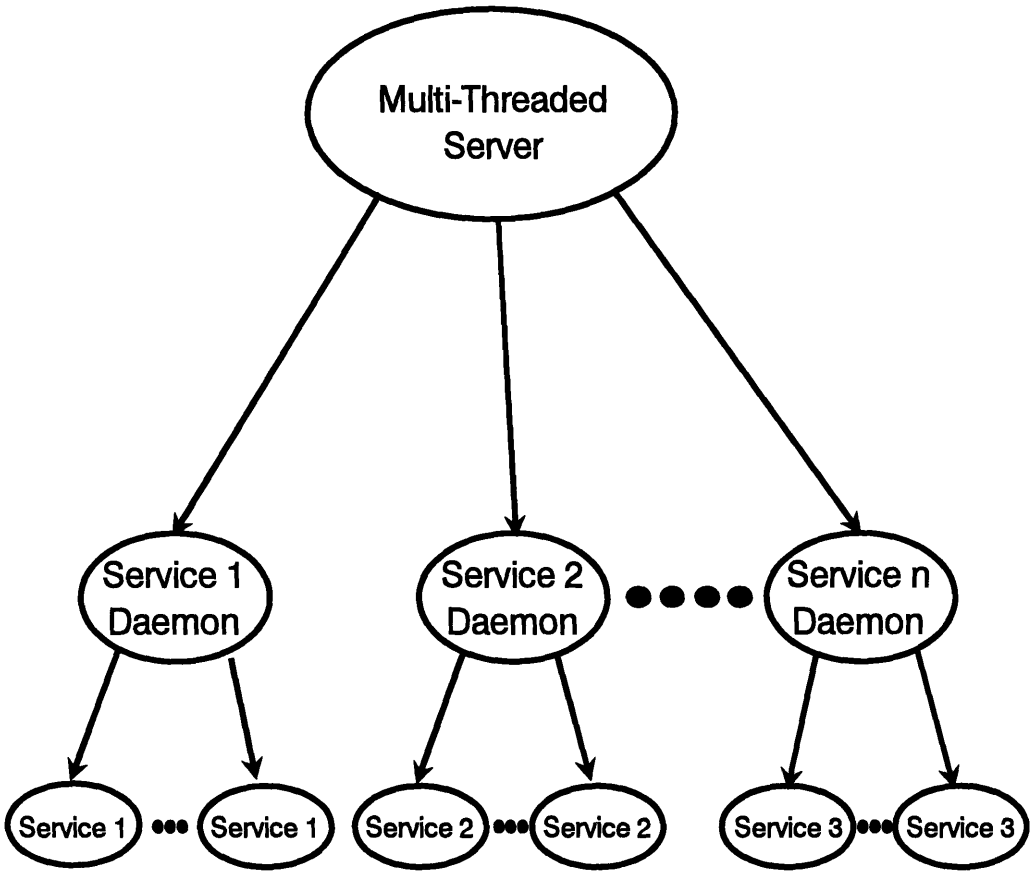


Figure 4-1: Basic Multi-Service Multithreaded Server Model

some reformatting, in a shared-memory buffer. Shared-memory allows more than one process to access a common buffer if the processes have the proper shared-memory keys. However, when more than one process share a common resource, we have to deal with problems of synchronizing access to the memory buffers. We require some means of preventing one process from writing, and possibly corrupting data, into a buffer while other processes are reading from it. This synchronization can be done on UNIX systems with semaphores.

Semaphores are synchronization primitives whose values are stored in the kernel. Operations on the semaphore are atomic operations which perform several instructions as single operations, thereby avoiding race conditions which could arise if an interrupt occurred between operations. Before storing data into a buffer, the writing process waits until it receives a signal indicating that no client is currently reading from the buffer. This is done by locking a read-in-progress semaphore. As soon as the process that is trying to write detects the read-in-progress semaphore is unlocked, it sets a write-in-progress semaphore, effectively locking other processes from reading the possibly corrupted file. This operation must be done in the same system call otherwise a race condition can occur. Such a condition would result if the reading lock had released the lock and then locked the file again before the writing process detected the second lock who would then proceed writing to the buffer. However, since the detection of the release and the locking were done in one atomic operation, there is no room for the reading process to possibly access data that is being written to. Once the writing process is finished writing to the buffer, it decrements the write-in-progress semaphore, effectively unlocking the memory. The reverse is true for the reading process. Locking of semaphores is performed by incrementing the semaphore value. Access would be denied to a buffer until the semaphore value is set to zero. Unlocking would consist of decrementing the semaphore value. By using two semaphores, one for reading and one for writing and locking/unlocking each respectively, we allow for multiple processes to read simultaneously from the same buffer. Reads are allowed once the write-in-progress semaphore has been decremented to zero. Writes, on the other hand, are not allowed until all the reading processes, which had incremented the read-in-progress semaphore once they detected the write-in-progress was released, are done and have each decremented the read-in-progress semaphore once again. Each reading client process use the same read-in-progress semaphore.

For accessing the shared memory buffers for both server programs, we only permit one process, the master process, to write to the buffer. However, that is not to say that every server can only allow one process to write. For our servers, only one process can communicate from the navigation unit and reference station at a time. However, for the navigation unit, we also would like the ability to have more than one process communicating to the server in order to request client specific information. This is done by using another semaphore which prevents writes until its value has reached zero. Once a writing process detects that the semaphore has been released by another process, it simultaneously increments the semaphore again to prevent other writing processes from gaining control of the lock.

In addition to ensuring that data is delivered uncorrupted, another important consideration in multithreaded servers is the avoidance of deadlock or a livelock con-

dition that may arise as a result of improperly locking and unlocking semaphores. In cases where two processes are trying to access a resource whose access is controlled by semaphores, deadlock can arise if both processes believe the resource is locked resulting in both processes blocking and not proceeding and the resource never being unlocked. In the case of livelock, one process monopolizes the resource never properly signalling the other process that it is done reading resulting in the other process blocking. For more than two processes, deadlock and livelock can occur from the same conditions as described above. In addition, one must make sure that each client gets the data and that not only one process is consuming all the resources. However, given our design constraints that each client process can run asynchronously from one another and the master process and can connect and disconnect asynchronously, our algorithm is further complicated.

The master process must ensure that each client process receives a copy of the information in the buffer before it overwrites it.[2] As a result, we need to keep track of the total number of clients connected at any given time on the server side and the currency of the data on the client side. This was done through the use of two semaphores – one for keeping track of the clients and for keeping track of the data.

We must make sure each client reads the buffer only once otherwise one client can utilize all the resources of a server or read the same information more than once. Although this may not seem to be too much of a problem for clients using server connected to the NAVunit, for clients of the reference station server this could cause errors since many differentially capable GPS receivers perform a frame sequence check on the RTCM SC-104 messages received (see Section 2.5.1.1). The current-data-stored semaphore's value was incremented each time a new message was stored in the buffer. The client programs kept track of the last message read's value and compare its value with the current-data-stored semaphore value of the memory buffer when the write-in-progress semaphore is released. Only if the two values are different will the client process read the data.

To keep track of the current number of clients connected is a bit more complicated since clients can connect and disconnect asynchronously. For a newly connected client to lose some information on startup is tolerable. However, if a client disconnects while the server were distributing information among its clients, the server could end up hanging waiting for a client that is no longer connected to read the information. To solve this problem, the server loops, keeping track of the number of clients connected through a client-count semaphore which is incremented each time a client connects. The semaphore was initialized so that when a client disconnects, which causes its associated server on the other end of the socket (the grandchild process) to exit as well, the semaphore value is automatically decremented. The server loops until the reading-clients semaphore is unlocked by all its clients. At the end of writing, the server sets

[2]By client process, we mean the process spawned by the daemon child process of the master program. This process actually services the client request by giving information to the client that it had copied from the parent process. Rather than referring to it as the grandchild process, we will refer to it as the client process, although the client programs are completely separate and do not need any information regarding the implementation details of the servers.

the reading-clients semaphore to equal the number of clients currently attached. As each client finishes reading only new data, reading-clients is decremented. However, because of the possibility of a client disconnecting during this loop, and therefore making the reading-clients semaphore invalid, in the loop, we keep track of the number of clients connected. If the current number of clients connected differs from the old number of clients connected, we change reading-clients semaphore value by the difference whether positive (meaning new clients have added) or negative (a client has died). In addition, during this loop, we must lock the reading-clients semaphore from being changed by a client process. This is done by waiting until the read-in-progress semaphore is unlocked and then setting the write-in-progress semaphore. Even though no writes to the memory buffer is occurring, this effectively blocks client programs from changing the reading-clients semaphore which is done after every read. The write-in-progress semaphore is unlocked after the comparison in order to allow clients still waiting to read the memory buffer to do so. At the end of the loop, the server finally writes to memory after double checking that no reads are occurring and the number of clients waiting for data is zero.

4.3 Reference Server

A Trimble 4000 RL II differential reference base station is connected to the central computer. The 4000 RL II transmits RTCM SC-104 message types 1, 2, 3, 6, and 16, computing corrections using purely C/A code algorithm at an update rate of approximately once every 0.6 seconds. The corrections are transmitted over an RS-232-C serial line to the computer at 9600 baud. These corrections are distributed via a multithreaded Reference Server that runs on a server computer, named `alfalfa`, in our AT&T Holmdel lab. A DGPS reference station with an antenna on the roof of the AT&T Holmdel building is connected to this computer via a tty line. provides a means of distributing information from the differential reference base station to numerous users, without the necessity of transmission of corrections over radio frequencies.

The Reference Server creates two children processes. The main parent process receives RTCM SC-104 Version 2.0 corrections over a serial line and broadcasts the information over the network as either raw data bytes following the "6 of 8" protocol discussed earlier or as TSIP packets. This is done by using two shared memory buffers, one for each service, and semaphores for resource synchronization. The parent process is responsible for reading information from the receiver, processing it and then storing them in the appropriate format in both shared memory buffers.

The processing of the information by the parent process is complicated due to the "6 of 8" protocol mentioned earlier. It essentially involves taking the message, and decoding it bit by bit. The bit by bit parsing involves reading in a byte of information, taking the six LSB and then byte swapping. This was necessary due to the fact the RTCM SC-104 corrections were transmitted MSB first while ANSI standards expect LSB first, resulting in swapped bits. As described in Section 2.5.1.1, frame synchronization is achieved by searching for the preamble bit pattern, `0x66` or its complement `0x99`. The complement preamble pattern is also searched for in case

the last parity bit of the previous word was 1, because then all the bits of the current word, except the parity bits but including the preamble, were complemented before transmission by the reference station. To recover the raw data, the bits have to be complemented. Once the first two words of the frame are recovered, providing us with message type, station identification, Z-count, sequence number, length of message and health, a parity check is performed for verification. In addition, the sequence number of the second word is also checked. Once verification has passed, the rest of the bits in the frame are parsed depending on message type. If the message is of Type 1, 2 or 9[3], which all follow the same message format, we parse the data using the same complement and swap technique just described, and parse in the data bit-by-bit into a byte format, leaving the reformatting into TSIP protocol for the grandchild processes. If the message is Type 3, the parsing is simpler, since the message is of fixed size (see Section 2.5.1.1). For message Type 6, the following byte, if any, is read in and nothing is done since this is a filler message. Message Type 16 is ignored since it is an ASCII message. All of the messages are stored in the TSIP shared-memory buffer with the first two words decoded and the rest in byte format. The processing for the raw RTCM SC-104 data service merely copies whatever has been transmitted by the 4000 RL-II into its shared-memory buffer. The parsing of the reference receiver data for both services is done at the same time by storing information into two different buffers.

The child processes forked by the parent process are daemon processes that handle all incoming connection requests from client programs. When the child process detects such a request, it spawns a new child process that will service the request and continues to listen for connection requests over a TCP/IP socket. Meanwhile, the spawned-off process waits until the first process completed saving information. Utilizing the IPC techniques described in the previous section, the spawned off process locks the shared-memory buffer until it finishes copying the data. When done, it unlocks the buffer and signals the main process that it has read the data. When the grandchild process completes copying the data, it then processes the information to prepare for transmission.

The processing of data to be broadcast as raw RTCM SC-104 Version 2.0 data is relatively straightforward since no formatting is required. It simply sends all the data that it has copied from the shared memory buffer over the socket. The processing required for the service that broadcasts corrections in TSIP format is a bit more complicated. It essentially involves taking the decoded RTCM SC-104 message, and reformatting it to the TSIP packet format shown in Table 4.3. However, the real difficulty is the result of the fact that the TSIP format is for RTCM SC-104 Version 1.0 while the differential reference station broadcasts Version 2.0. Since we know the difference between the two messages as described in Section 2.5.2, we take that into account in the reformatting of the data. Once the data has been reformatted, it is transmitted to the client programs via sockets.

[3]Our receiver does not broadcast Type 9 messages, but since it is identical to Type 1 messages in format, it does not cost us anything in terms of code length.

ID (hex)	Description	Byte	Item	Type
60	Message Type 1 and 9	0	Modified Z-count	Word
		2	Station Health	Byte
		3	Number of SV's in Packet	Byte
	The next 5 bytes are repeated for each SV			
		4+(N*5)	UDRE & SV PRN	Byte
		5+(N*5)	PRC	Word
		7+(N*5)	RRC	Byte
		8+(N*5)	IODE	Byte
61	Message Type 2	0	Modified Z-count	Word
		2	Station Health	Byte
	The next 3 bytes are repeated for each SV			
		3+(N*3)	SV PRN	Byte
		4+(N*3)	Delta range correction	Word

Table 4.2: TSIP Format of RTCM Corrections

4.4 Navigation Server

The Navigation Server is configured in a similar manner as the Reference Server except that it is connected to the NAVunit and provides six types of services. It spawns six child daemon processes, rather than just two. The parent process is responsible for obtaining information from the NAVunit via a tty line and then parsing it into various formats that will be stored into different shared-memory segments to be accessed by client programs. In one service, all packets received from the unit are saved into a shared-memory segment with the TSIP header and trailer removed. If the packet is position information, i.e. ECEF or LLA coordinates or compass or tilt sensor readings, the packet is again saved in another shared-memory buffer. If the information is time data, it is stored in another shared-memory segment. For the Navigation Server, we therefore need a total of three discrete shared-memory segments.

The semaphore operations necessary to prevent race conditions or corruption of data for the Position Server buffer and TSIP packet buffer were described in the previous section on semaphore synchronization. However, the semaphore used for the time service is handled differently. The parent process waits until it can get a lock on the shared-memory segment, which it can't if another time client is currently accessing the data. As soon as the semaphore is unlocked, the parent process saves the data into shared memory and then signals the time client that it is done saving. In addition, packets are only saved into the time service shared-memory buffer if a client is currently connected. This is done to insure that the time returned is when

the user requested the information as opposed to old time data that was still residing in the time service buffer.

The child processes forked by the Navigation Server are intermediate daemon processes set up in a similar manner as the Reference Server. In and of themselves they do not do any processing. Their main function is to listen for connection requests and fork processes to service the requests. Five different sockets are established: TIMEsocket, POSsocket, DEMANDsocket, VIEWsocket, and CONTROLsocket. The purpose of these sockets is described later. The child processes of the daemon processes then operate on the data in the shared-memory segments.

4.4.1 Time Service

The time server routine generates a request for time information to the NAVunit. Meanwhile, it opens up and attaches shared-memory segments that had been created by the grandparent process. The server signals the grandparent process through a semaphore that a client is waiting for data. The routine blocks until the grandparent process signals that time data has been stored in a buffer. When the time server (the grandchild process) receives the semaphore signal, it locks the data so that the grandparent process cannot alter the data in case another client attaches, requesting the time information. After copying the information, it unlocks the semaphore and detaches, but does not remove the shared-memory segment. The copied data is then reformatted from TSIP packets to an ASCII character string containing time of day of week, day, month, date and year. The ASCII string is transmitted over the TIMEsocket to the client program.

4.4.2 Position Service

The next service spawned by Navigation Server is a position server that spawns child processes to provide position information. As described above, the parent process stores position related information into the Position Service shared-memory buffer. The XYZ coordinates are all in LLA coordinate system. We decided to use an LLA coordinate system since more databases use LLA than ECEF. If the data from the NAVunit was received in ECEF coordinates, we converted the measurements to LLA using the formulae given in Section 2.6. However, as discussed earlier, the altitude measurement is ellipsoidal height and not the height above mean-sea level (MSL). Since the majority of map databases deal with altitude as MSL, we adjusted the altitude measurement with the geodetic undulation at the measured longitude latitude position using the modified GEOID90 program discussed in Section 2.6. The modifications involved taking the critical FORTRAN subroutine and calling with the LLA position coordinates. The resulting correction is then subtracted from the geodetic height and transmitted with the longitude and latitude coordinates as the altitude information.

If the information was compass data, we calculate the angle from the transmitted parameters as described in Section 3.3.1. However, this information is slightly skewed due to the nature of the Earth's magnetic field. We correct this error by calculating

the magnetic declination at a given position and time of year using subroutines from the GEOMAG program described in Section 3.3.1. The declination is then added to the calculated azimuth. The elevation and roll angles are calculated according to formulas given in Section 3.3.2. The user's LLA position and orientation angles are transmitted to client programs over the POSsocket.

4.4.3 Demand Service

The Demand Service is a service which behaves the same as the Position Service. However, the difference between the two services rests in the fact that demand service only outputs information after receiving a request from the client program for data. It also utilizes the Position Service shared-memory buffer. However, communication is done over the DEMANDsocket.

4.4.4 View Service

The View Service broadcasts all the information that the navigation unit is currently outputting over the VIEWsocket. It utilizes the TSIP shared-memory buffer and follows the semaphore conventions required by it. Clients that connect to this port do so on a purely voyeuristic basis. The clients reformat TSIP packets into user-friendly ASCII messages. No communication from the client to the navigation unit is possible through this service.

4.4.5 Control Service

The Control Service behaves in a similar fashion as the View Service with one important exception; clients connected to the Control Service can communicate with the navigation unit over the CONTROLsocket. Using as a client a modified version of PKTMON, a program distributed by Trimble Navigation, Inc., to interface with their equipment, we can send commands to the navigation unit, specifically the GPS receiver, since none of the other components in the unit can receive input. In addition, clients have access to all the information that the View Service clients have access to since it also utilizes the TSIP shared-memory buffer.

Clients connected to the Control Service have access to most of the information that was described in Section 2. Besides position, velocity and time (PVT) information, clients can access ephemeris, almanac, ionospheric correction coefficients, and other data in the navigation message by request.

4.4.6 DGPS client

The last child process that is spawned off by the server is not a service, but is a client to another server program. This process is a client to the Reference Server that receives differential corrections over the DGPSsocket in TSIP format to the NAVunit. The navigation unit in turn then transmits the information to the GPS receiver.

4.5 Interfacing Mobile Navigation Units to the Network

Therefore, our reference server, running on our server computer `alfalfa` and using the GPS reference station and antenna on the Holmdel roof can broadcast differential-corrections to mobile and stationary GPS receivers on the Internet network via the navigation server. The applicability of the corrections is determined by the GPS satellite orbit since the reference as well as the mobile solutions should be tracking the same satellites for differential-corrections to be of use. In addition, the software supporting programs on the mobile units is also written using server/client models in which the server program continuously broadcasts position information from the NAVunit over the network. Thus, the central computer and/or other mobile units can obtain the spatial information of any other unit and track it as demonstrated in Figure 4-2, where `alfalfa` is broadcasting corrections while simultaneously receiving spatial information from `amalfi` and `amadeus`, two SPARCbooks each with its own NAVunit.

The link supporting communications between the mobile unit and the central computer is a cellular line over which Morning Star's PPP is installed. Using Motorola's MicroTacLite and CellularConnection modem, we established a serial link with our central computer over a modem. Once the connection has been established, we initialize PPP, which allows the transmission of data using IP protocols over serial lines. This cellular connection is used not only for broadcasting differential corrections but also for broadcasting the mobile unit's position. If there comes a time when civilian GPS has achieved an accuracy such that differential corrections are no longer needed to be broadcast, the cellular line can serve further uses such as for communicating between units and possibly as a communication link for database access.

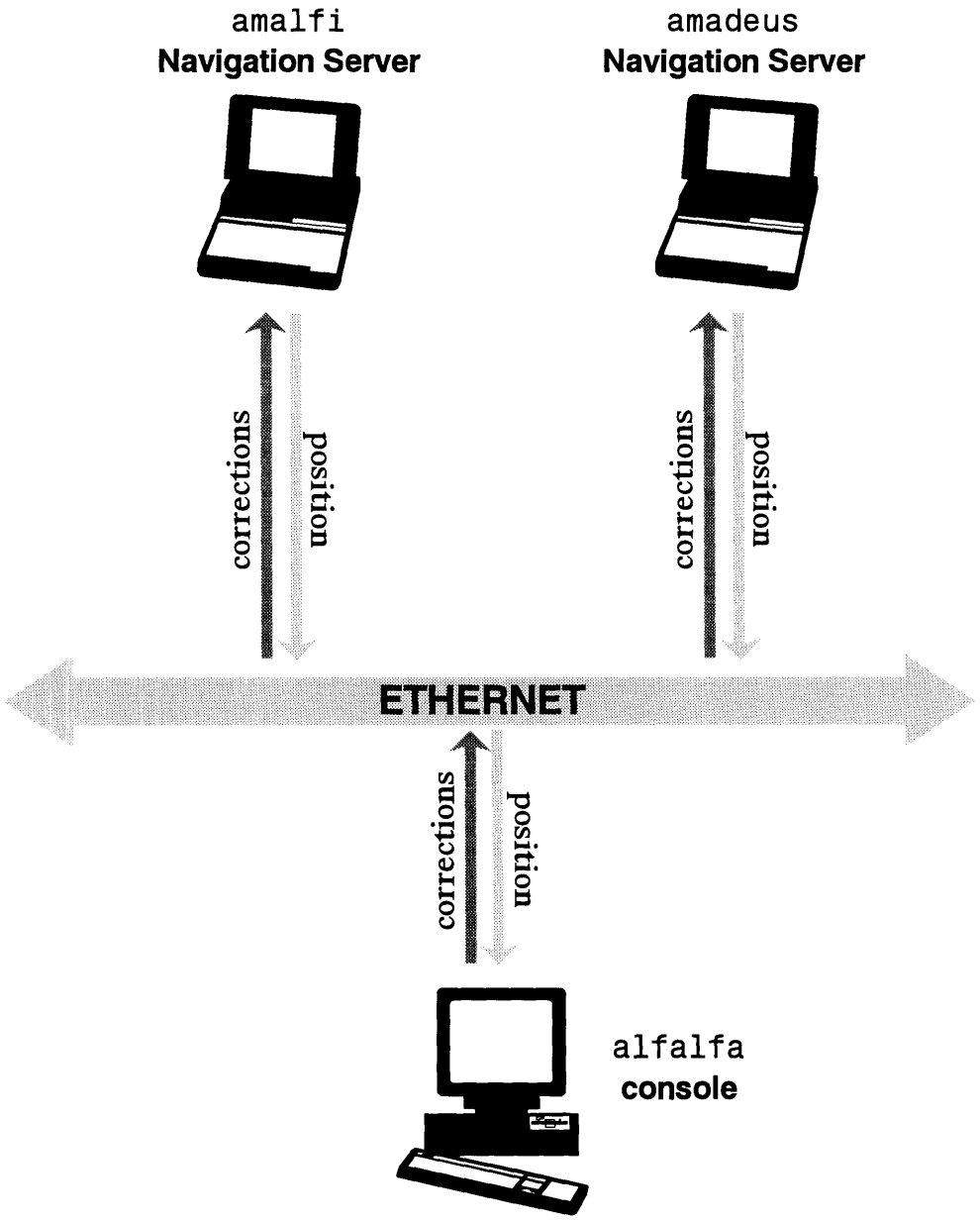


Figure 4-2: Networked Mobile Navigation Units

Chapter 5

Display Software

The enhanced-reality system's ability to display 3D spatial information as opposed to merely two dimensional data based on longitude-latitude-altitude (LLA) coordinates distinguishes this system from other "personal navigators" currently under development[1]. To achieve this, two 3D graphics libraries were developed – one based on the SPOTlight Graphics SPOTlib library [37] and the other using the SCULPT modeling tools developed by Bruce Naylor of AT&T Bell Laboratories [28]. In addition to these two libraries, some programs were developed using the X Window Systems Xlib drawing package for rendering line drawings. The programs developed using these graphics libraries were clients to the navigation server discussed in the previous chapter, and therefore utilized the six parameters from POSservice to update the viewer's vantage point accordingly. Using the client/server abstraction discussed previously, we were able to develop the graphical interface separate from the API developed in the previous section. This chapter provides a brief overview of the SPOTlib and SCULPT graphics libraries. For more information regarding Xlib, the reader is referred to the O'Reilly and Associates manuals on X11 and X Windows programming [30] and [31].

5.1 SPOTlib Graphics Library

The SPOTlib graphics library uses a z-buffer or depth-buffer algorithm for rendering images. (See Foley, van Dam, Feiner, and Hughes[5][pages 668-672) for details on this algorithm.) Given a list of objects, represented as a set of polygons, and their locations in a scene, a photographic quality image of the desired scene is generated. In addition to polygons, lines and points can also be rendered using the z-buffer algorithm. Each pixel of the image is represented by 4 floating-point numbers where the first 3 numbers are the intensities of the red, green and blue components of the

[1] AT&T, GO, PenStuff and Trimble Navigation recently announced the development of the PenPoint Global Positioning System [21]. CMU is extending its VuMan portable computing project to a navigation project called Navigator that will use GPS positioning information to do some navigation programs [35].

image and the last number represents the alpha channel.[2] Images are updated by performing a series of transformations on the original polygonal representation of the objects in a scene, rendering the image representation and then displaying it. The polygon representation of a scene was scene dependent and must be done beforehand. The real-time rendering of the image was handled by SPOTlib graphics commands. Once rendered, we now need to display the image.

5.1.1 Problems in Displaying Full Color Images on Frame Buffer Displays

A difficulty encountered in displaying the images produced by the SPOTlib graphics package was that the images were in full 24-bit color format. To display a raw full color digitized image, typically 24 bits are used for each pixel where each color component is quantized to 8 bits. Therefore, such an image has the potential of containing $2^8 \times 2^8 \times 2^8$ unique colors or approximately 16 million colors. This high intensity resolution results in the generation of realistic color images. However, the color displays generally available on notebook computers utilize, at best, an 8-bit frame buffer. Frame buffers can be considered lookup tables that contain the intensity levels for the three color components of any 256 colors out of the possible combination of 16 million. Thus, in order to display full color images on 8-bit frame buffer displays, a method to fairly map 16 million colors down to 256 in a manner that will still preserve much of the information from the original image needed to be developed.

One solution that has been proposed and is quite popular was one based on an image dependent colormap. Heckbert [11] presents a method which is based upon the median cut algorithm, a modification of the popularity algorithm. The median cut algorithm calculates the maximum and minimum color component values of the image, and then divides the colormap along the median color component value creating two "sub-bins". This step is repeated for each sub-bin until the desired number of colors are generated. However, in order to calculate this median color a histogram of the color distribution must be made first. Although this method tends to produce very good results, it is computationally expensive and, perhaps more seriously, image dependent. Since the image generated for each frame does not necessarily have the same color distributions, the histogram of color distributions would need to be calculated at the start of each frame, thereby slowing the update rate considerably. Therefore, we decided that better methods for dynamic full color reproduction of images would be those that are image independent. This in turn led us to decide that the best colormap to use would be one that is image independent such that if reading in more than one full color image (as would be the case in generating a sequence of video frames), one would not have to recompute the entire colormap for the new image since allocating the colormap would add extra delays to our display system. Thus, in order to do dynamic display of full color images, we felt that it would be most efficient to use a static colormap that would be computed and loaded once at runtime.

[2]For our application, we ignore the alpha channel.

5.1.2 Colormap

In deciding a colormap selection strategy, one of our main design criterion, besides being image independent, was that it should be computationally simple to map a 24 bit color value to its equivalent frame buffer value. As a result, we implemented a pseudostandard `XStandardRGBColormap` where the index to the frame buffer location provided information about the color in the bin. Thus, since the values that are stored in the color map are known a priori, we can generate the index based on the 24 bit pixel value. In standard `XStandardRGBColormap`, typically the first 3 bits of the color map index is the red component, the next 3 is the green component and the last 2 represent the blue component. Effectively, this will quantize 256 gradations of red and green intensities down to 8 gradation levels and 256 levels of blue down to 4 levels. The blue channel was roughly quantized as compared to the red and green components since the human eye is typically less sensitive to variations in the blue channel spectrum. This choice of quantization levels leads to a total of 256 colors.

However, we are using a pseudostandard colormap because we are using less than the full 256 available colors in order to enable users to run other color programs simultaneously with our enhanced reality system application programs. Therefore, in our implementation, we have the option of using only 2 bits/pixel for each channel or 1 bit/pixel, leading to a total palette of 64 or 8 colors respectively for the graphics library programs to utilize.

The formulae used for generating an index into the color map were:

$$\text{color table index} = \begin{cases} 00|rr|gg|bb & n = 2 \\ 00|00|0r|gb & n = 1 \end{cases} \tag{5.1}$$

where each position represents its bit location with MSB first.

We now have to set decision and quantization levels before generating the colormap, i.e. we must decide what value a red component value quantized to 3 should be reconstructed as. However, this in turn will depend heavily on which method we implement to do our image reproduction since different techniques require different quantization reconstruction values.

5.1.3 Halftoning

Dithering can improve the image quality by redistributing much of the low frequency noise due to quantization error to the higher, less visually disturbing, frequencies. After halftoning, the low frequency components of the quantization error are of much lower amplitudes. The low frequency components also depend on the number of intensity levels being used. Based on experiments with various halftoning techniques, we decided that the optimal halftoning method for our particular application would be one utilizing a dispersed dot ordered dither. This was based on two prime considerations - ease of computation and perceived distortion. Although other methods exist that provide better perceptual results, i.e. Heckbert's median cut algorithm and Floyd-Steinberg Error Diffusion algorithm, they require a large amount of computational time. There are also faster techniques than a dispersed-dot ordered dither,

such as uniform quantization and Robert's Pseudo Random Noise technique, however, the halftoned images are not as perceptually pleasing due to the false contours and graininess appearing in the new image. Since the dispersed-dot ordered dither technique was a pixel-by-pixel operation that distributed noise into the less perceptually disturbing high frequencies, it was selected.

5.1.3.1 Dispersed-Dot Ordered Dither

For monochrome images, if the image value is less than the threshold array, a value of 0 is given to the pixel. If the value is greater than the pixel, a value of 1 (or the highest representable intensity value) is displayed. A mathematical formulae describing this mapping is:

$$\hat{f}(n_1, n_2) = int\{f(n_1, n_2) + D(n_1, n_2)\} \tag{5.2}$$

where $D(n_1, n_2)$ is the dither signal consisting of the threshold array values [41][pp. 340-341].

However, this is only true if the desired output format is for binary displays. Suppose we wish to halftone an image onto an M-ary display where M represents the number of levels we wish to represent. Ulichney [41] demonstrates that this could be done by modifying Equation 5.2 to:

$$\hat{f}_M(n_1, n_2) = \frac{1}{2^M - 1} int\{(2^M - 1)f(n_1, n_2) + D(n_1, n_2)\} \tag{5.3}$$

for an M-bit display with 2^M colors. This algorithm assumes that the the user's output level contains 0 and 1, with 1 being the maximum value, and the remaining $2^M - 1$ levels equitably distributed in between. Because our displays map minimum and maximum intensity values to integers between 0 and 255, while Equation 5.3 assumes that all values are between 0 and 1, input pixel values were first scaled down by 255 before processing and then rescaled up by 255 after processing.

Having decided on the quantization and reconstruction levels, we needed to select the order of the threshold array. Therefore, we also experimented with various orders to see which would be optimal for the images we were displaying on our computing platforms. If the order of the threshold array is too small, false contours occur. If too large, texture patterns are generated in areas of slowly varying luminance. With most images, a threshold array of order seven should be used. However, because our system deals with synthetic images, consisting of many smooth polygon surfaces, as opposed to digitized "real" ones, a threshold array of order seven generated too much texture artifacts on surfaces that should appear smooth. For a threshold of order three, too many false contours were now being generated. We selected odd period orders since even order arrays produced horizontal and vertical artifacts to which the human eye is sensitive [41][page 84]. Using a threshold array of order five appeared to provide the least amount of distortion for synthesized images. Therefore, each image generated by the SPOTlib graphics library was dithered with a fifth order threshold array according to Equation 5.3 before being displayed since it provided a perceptually pleasing image at a low computational cost. The threshold array used

was one specified in [41][p. 135] and is included in the Appendix A.

5.2 SCULPT

The ability to render 3D images in real-time was a feature desired by our system. In SPOTlib, this performance criterion was not met due to the computational complexity level of rendering a z-buffered image is quite high mostly due to overhead from the scan-conversion of the polygons and a per pixel comparison operation for the sorting [5][p. 671]. In addition, the need to dither every frame generated further reduced performance by an additional $O(n^2)$ operations for $n \times n$ images. To solve this problem, for applications in which realism was not as an important consideration as rendering speed, we utilized the `sculpt` graphics toolkit developed by Bruce Naylor of AT&T Bell Laboratories.

`sculpt` is an interactive program that allows the user to interact with models of 3D objects in real-time. This is done using binary space partitioning (BSP) tree representations of objects rather than the traditional boundary representation (BREP). The BSP tree representation of an object is a continuous space representation that is scene-independent. In the partitioning tree, regions (the faces of the object) are segmented into areas of uniform texture. An algorithm described by Thibault and Naylor [47] converts the BREP of an object to its BSP tree representation. The BSP tree representation of the object is operated on when updating the user's viewport and then displayed. By using bsp trees, we manipulate regions or faces rather than individual points, allowing the manipulation of models of 3D objects in real-time. Segmentation occurs a priori to the rendering and display of the object, thereby, negating the overhead of converting BREP to BSP representations. For more details on BSP trees see [47] or [5][pp. 675-680].

In order to use the `sculpt` package, the boundary representation of an object has to have been processed previously to generate the BSP tree representation. The BSP tree representation, that was stored as an ASCII file, is then loaded upon startup. Once loaded, various manipulations on the object can be done. Since BSP trees can be manipulated as sets, operations with other BSP tree objects are possible such as merging and subtracting. The object created as a result of the operation can then be saved as a new BSP tree object. Other operations that can be done to the object is changing various attributes such as its color, how it was rendered and and how it was shaded. However, for our purposes, the most important operation that can be done to the object is the ability to alter an object's orientation in space. In Naylor's original implementation, the object's position is controlled by tracking the mouse position to control rotation and pressing the mouse buttons for zooming in and out. In addition, the viewer's viewpoint and the object's translational position can be changed by using menu options and the mouse. To work with our system, we altered the program to also accept inputs from the navigation unit. The object's rotation angles were then also controlled by the azimuth, elevation and roll angles of the NAVunit.

Chapter 6

Experimental Results

The first five chapters have discussed the implementation of the hardware and software needed in an enhanced-reality system. Before our system can be used, a few details still need to be considered. Primarily, we still need to measure the accuracy of our GPS receiver to determine if its errors are within our error tolerances. To do this we ran several experiments. The nature of these experiments are to take a large set of samples from the receiver positioned at a fixed point. One set of our sample data is shown in Figure 6-1 taken October 23-24, 1992 (GPS Week 667). From this data, in which a stationary point was measured, the samples fluctuated more than three hundred meters from its mean. This error was not tolerable for our prototype system which required errors of standard deviation of less than ten meters. As a result of our initial experiments at the start of the project, we realized that differential corrections were needed to meet the accuracy requirements of our system.

As mentioned in Chapter 2, differential corrections are, at best, only as accurate as the base station's reference position. If the reference position of the base station is offset by any amount, the mobile receiver's differentially calculated positions will also contain this systematic error regardless of the receiver's accuracy. As a result, determining the initial position of the antenna of the reference station is critical to the accuracy of the system. We had a choice of either paying a professional to survey the site for us, using a topological map to estimate the receiver's position or relying on data gathered from a GPS receiver to calculate the mean position. The latter was decided upon since it seemed to be an economically feasible solution that would provide fairly accurate results.

6.1 Data Analysis

Over a period of 7 weeks, approximately 1.25 million sample points were collected. The sample sets were taken over periods of about 24 hours at a sampling rate of approximately once per second. Position fixes were generated by the receiver at a rate of once every 0.6 seconds. Samples obtained when less than 4 pseudorange measurements were used for the navigation solution were excluded. Based on experimental observation, using less than 4 satellites to calculate the user's 3D position

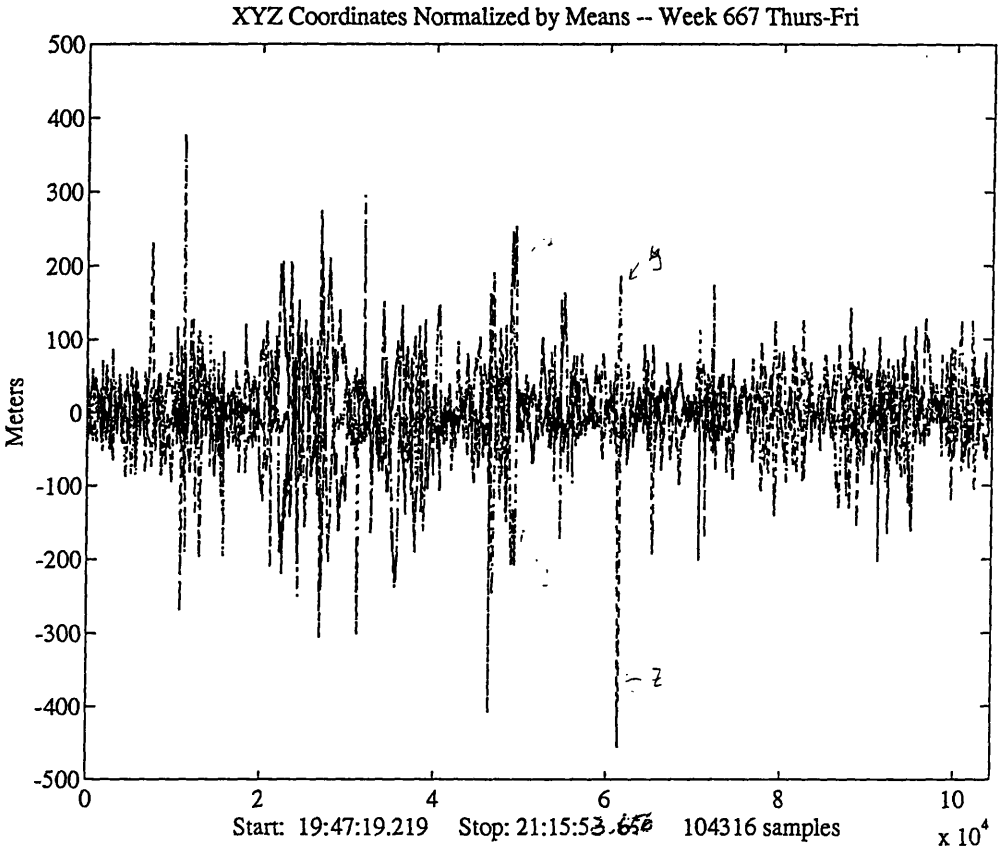


Figure 6-1: Non-corrected GPS Measurements Taken at the Holmdel Building Roof

resulted in data that fluctuated, on average, more than three standard deviations from the expected values. Figure 6-2 displays raw data collected and normalized by its means with all three ECEF coordinates superimposed. The data was taken September 20, 1992. Figure 6-3 displays the number of satellites used for the positioning. One notices that there is a direct correlation between how noisy the data is to the number of satellites used. Removing samples that were calculated using only 3 satellites in the solution resulted in cleaner data as shown in Figure 6-4.[1]

Figure 6-5 displays the number of visible overhead satellites during each fix for a sample data set collected November 18-19, 1993. The number of satellites are centered about 6 with the maximum number of satellites being 8 and the minimum 4.[2] From Figure 6-5, one sees that the data fluctuated least when there were the greatest number of satellites visible. Also, by examining the corresponding PDOPs, we also notice that the PDOPs tend to be the lowest when there are the greatest number of overhead satellites. Figure 6-6 displays the inverse PDOP as a function of time. These observations are consistent with what we expected since a larger number of visible satellites results in more of a choice for the receiver when selecting the four satellite combination that will produce the lowest PDOP.

For determining the position of the reference station's antenna, we utilized a `6-Channel Boardset` receiver since this was the only type of receiver that we had available at the time. The raw data was collected using our program **`gather`** which communicated directly to the receiver in TSIP, requesting position fixes in addition to the calculated DOPs associated with each measurement, and the number of overhead satellites during the fix. The receiver's operating parameters were configured before each run to be in stationary mode producing only 3D[3] double precision fixes in ECEF coordinates. This data was converted from TSIP format and then written into three output files compatible with MATLAB input format [19]. The first file consisted of the three double precision position ECEF coordinates calculated by the receiver. The next file is a list of the corresponding PDOPs for each measurement. The last file contained the number of satellites that were visible for each fix. This file was used to view the accuracy of the position fix as a function of the number of visible satellites as was shown in Figure 6-5.

The three files were loaded into MATLAB for processing. The three position parameters were independently processed and averaged by the total number of samples. An example data set is shown in Figure 6-7 taken during January 14-15, 1993 (GPS Week 679) with a total of 138,223 samples. The calculated mean positions were The standard deviations were $\sigma_x = 23.7, \sigma_y = 45.6$ and $\sigma_z = 41.5$. To smooth data, we removed any outliers of sample points that were greater than three standard deviations away from the calculated mean. Using the same sample set shown in Figure 6-7, Figure 6-8 shows the same data with outliers removed leaving in 134,507 samples.

Rather than using a simple average, a weighted average of the sample points

[1]This set of data was not used in the calculation of the final position due to the use of 3 satellites in some of the fixes. The Trimble receivers have the capability of defaulting to 2-D position fixes if the overhead satellite configuration does not allow for 3-D positioning.

[2]The receiver was configured to do Manual 3D fixes, that is, producing a solution only when

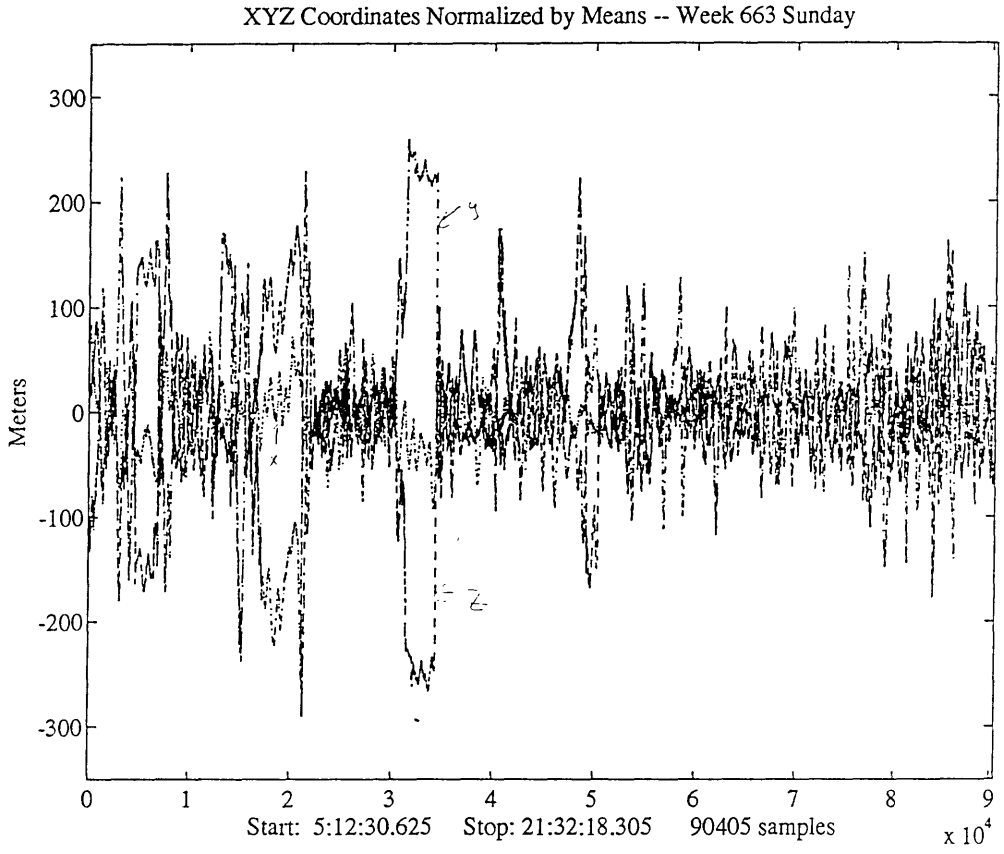


Figure 6-2: Effects of Number of Satellites on the Navigation Solution

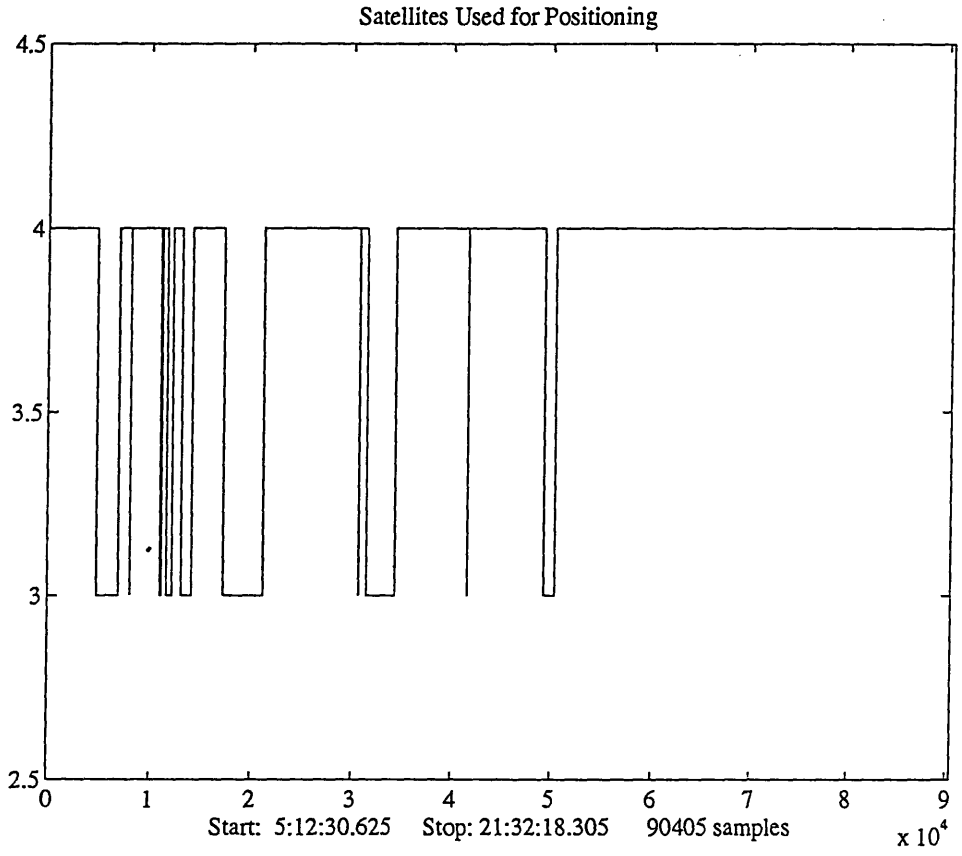


Figure 6-3: Number of Satellites Used for Positioning

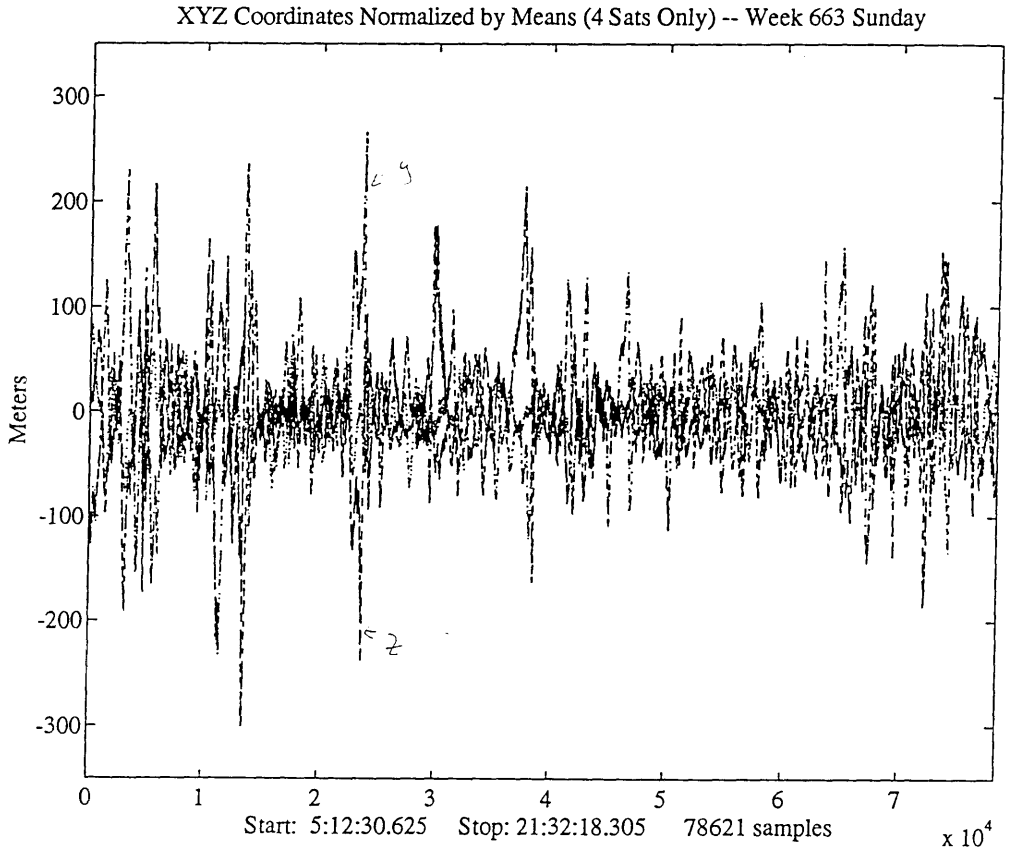


Figure 6-4: Position Fixes with Samples Taken with 3 Satellites Removed

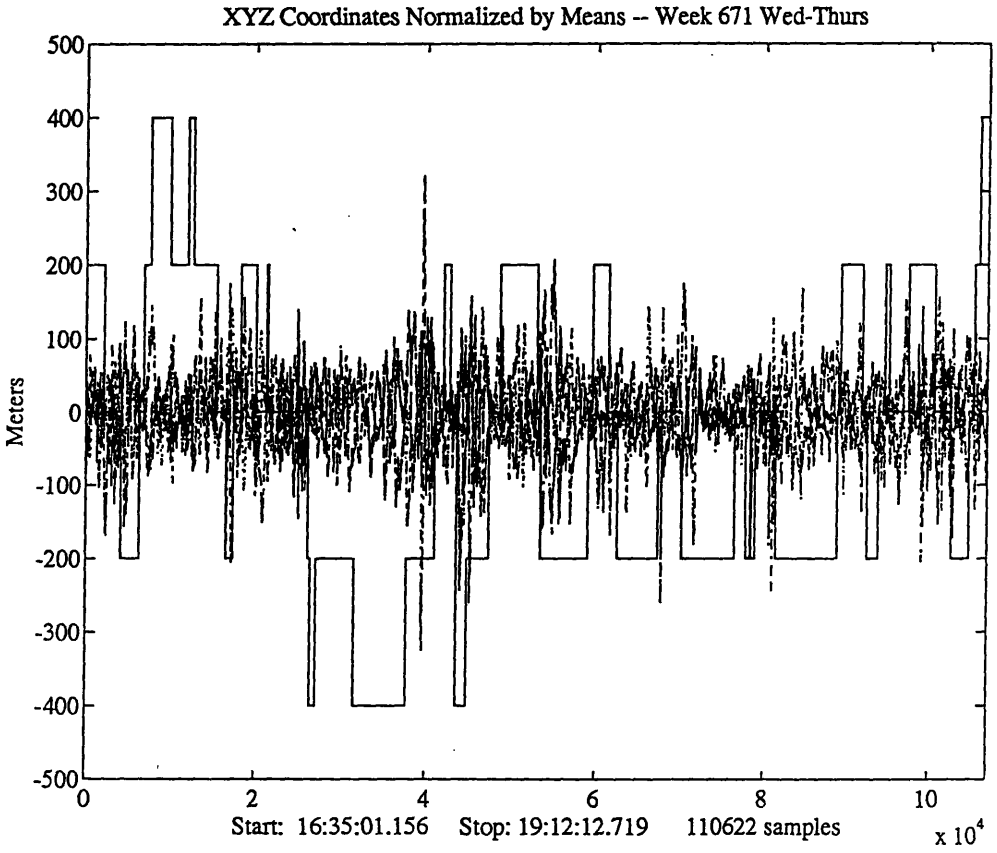


Figure 6-5: Effects of Number of Visible Satellites on Position Precision

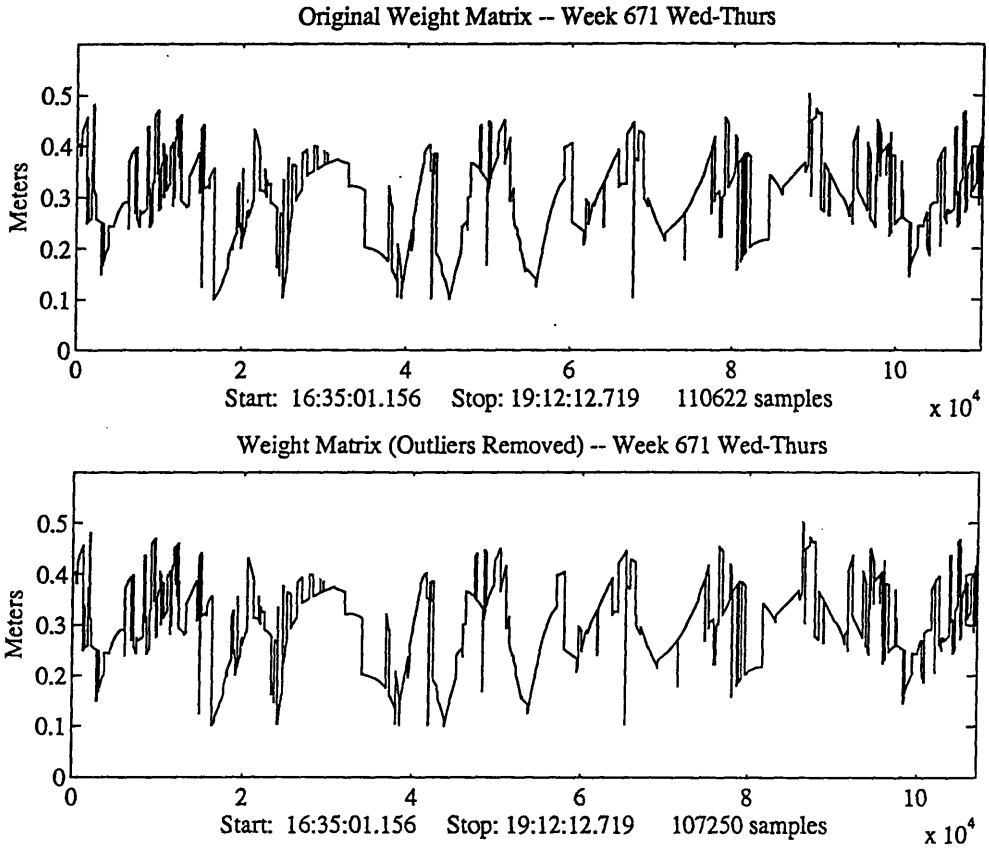


Figure 6-6: Effects of Number of Visible Satellites on PDOP

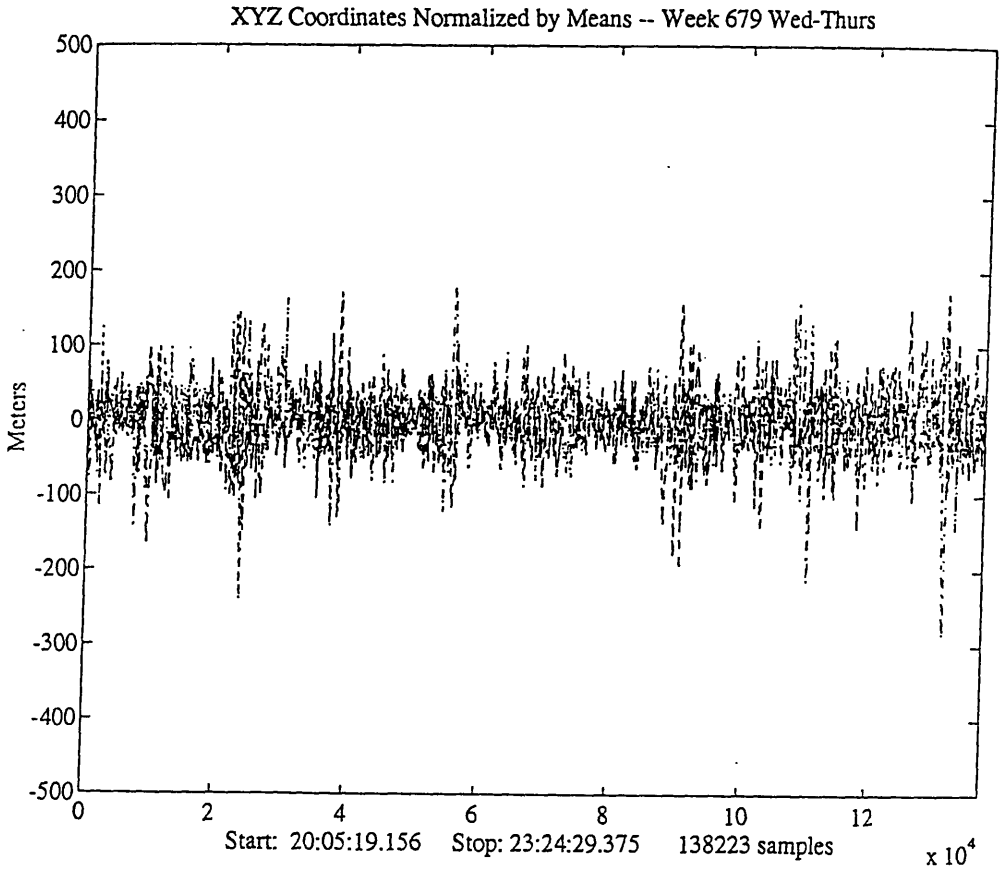


Figure 6-7: Subset of GPS Measurements Used to Calculate Position of Base Station

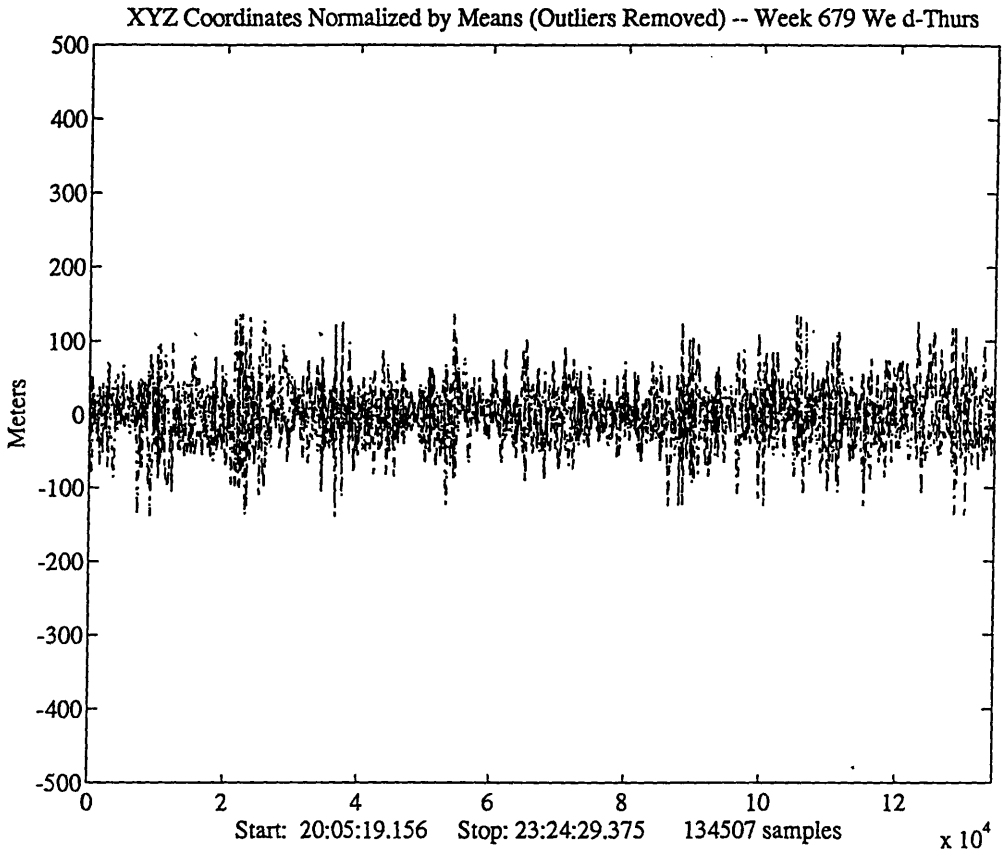


Figure 6-8: Smoothed Subset of GPS Measurements

was used since each data does not have the same precision. This is due to the fact that position precision greatly depends on time varying conditions as was discussed in Section 2.4 regarding sources of errors. Ideally, we would use σ_{xx}, σ_{yy} and σ_{zz} as given by the calculated covariance matrix used in the navigation solution (see Section 2.3.1) as the weights for each X,Y, and Z sample. However, only the DOPs were available to us. In the end, the inverse of the PDOP of each measurement was used as the sample's weighting function since, as noted earlier, the PDOPs, being the dilution of precision in 3D space, provide an indication of the accuracy of each measurement. The inverse of the PDOP was used as the weight since a smaller PDOP implies greater precision in the calculated position. Figure 6-9 shows the weights for both the raw and smoothed data shown in Figure 6-7 and Figure 6-8 respectively. Using Equation 6.1, the weighted mean position was calculated for each coordinate.

$$M_x = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i} \tag{6.1}$$

where M_x represents the weighted mean of N elements in the sample set X and W is the $N \times 1$ weight matrix whose elements are the inverse of the PDOPs for each sample point in X.

The above experiment was repeated several times resulting in a total sample set of 1,247,849 points. Each run consisted of roughly 100,000 sample points. The runs were then summed together after multiplying the calculated weighted average of each set by the number of sample points in the set. The sum was then divided by the total number of samples resulting in the estimated coordinates of the *6-Channel Boardset* receiver antenna to be 1327631.29 m, -4681967.76 m, 4108988.50 m in ECEF coordinates. This translates to a longitude of 40^o 21.9145' N, latitude of 74^o 10.1186' W at an ellipsoidal height of 28.970 m. The geodetic undulation at this point was calculated to be -32.464 m, resulting in an orthonormal height of 61.434 m or 201.60 ft. Since the altitude measurement from the GPS is typically the noisiest of the three measured LLA parameters (for reasons discussed in Chapter 2), the accuracy of the measured altitude was compared to that given by the architectural plans and drawings of the Holmdel building as an indication of the overall precision in the calculated position. According the the plans, the MSL altitude of the building's roof is approximately 196 feet . Using a yard stick, the height of the antenna from the base of the roof was measured to be roughly 37.5 inches or 3.1 feet. Therefore, summing the two, the approximate mean altitude we expect to calculate was about 199.1 feet. Thus, the calculated average position had an accuracy of approximately 2.6 feet out of 26,200 km![4]

In our earliest experiments, the standard deviation of out sample data sets were typically around sixty to eighty meters as shown in Figure 6-1. However, towards the end of the experiment, the government had launched more satellites. As mentioned

there were at least 4 usable overhead satellites.

[3]The receiver had the capability to default to 2D mode if not enough satellites were visible.

[4]This is the approximate orbiting radius of the satellites from the Earth's center.

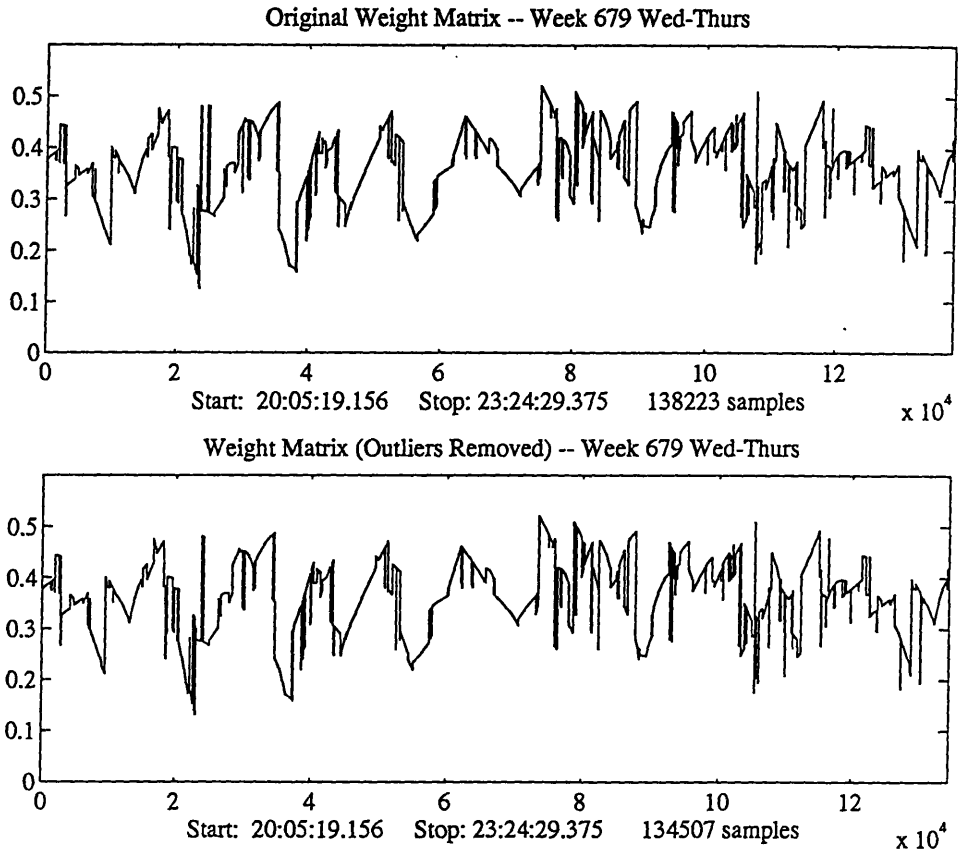


Figure 6-9: Weights of GPS Measurements

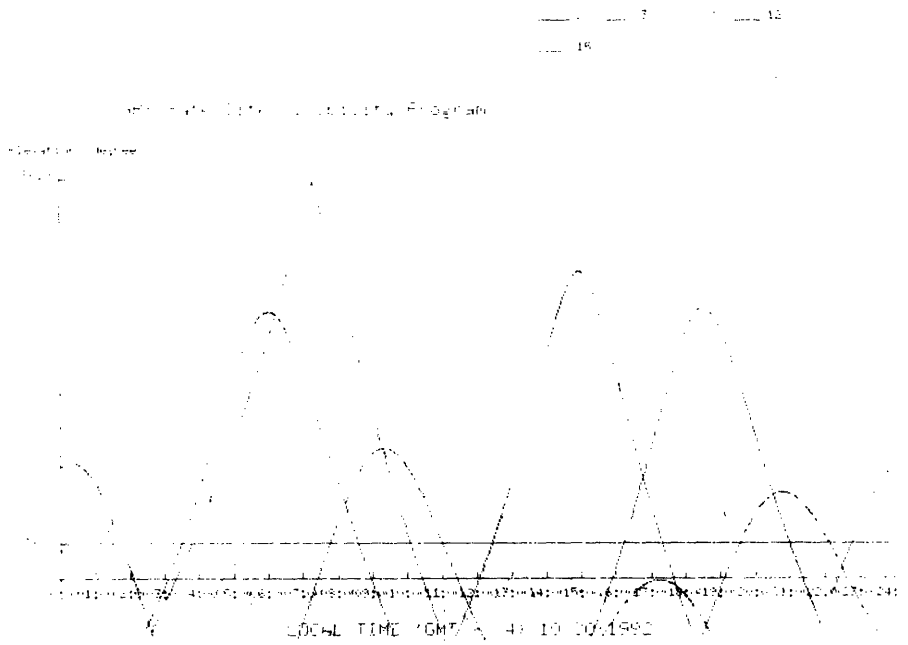


Figure 6-10: Satellite Visibility Plot for GPS Week 667

previously, the number of visible satellites greatly improves positioning. At the start of the data gathering in October, 1992 there were only 18 satellites in the constellation, providing approximately only 23.5 hours of 3D coverage per day. However, by the end of the experiment in January 1993, there were 21 satellites in the constellation. Plots of the elevation of visible satellites with respect to the `6-Channel Boardset` receiver over a 24-hour period for the same time period as the data shown in Figure 6-1 and Figure 6-7 is shown in Figure 6-10 and Figure 6-11 respectively. From the figures, we note that the hourly coverage of the satellites appears to have improved greatly over the weeks the experiments were conducted. Figure 6-12 shows a more recent elevation plot for the April 19, 1994 constellation consisting of 26 satellites. Our final data sets showed a marked improvement. The standard deviation in these last experiments cases was roughly only thirty to sixty meters. Although significant, this improvement in positioning was still not sufficient to provide the positioning accuracy that is needed by our system. This is further shown by Figure 6-13 which was taken the week of May 9, 1994 (GPS Week 718). As a result, differential corrections are still needed to improve the positioning precision provided by GPS.

6.2 Differential GPS

Using the data from our experiments, the fixed position of the antenna of our differential base station, a Trimble 1000 RL II receiver, was entered as 40° 21.9145' N,

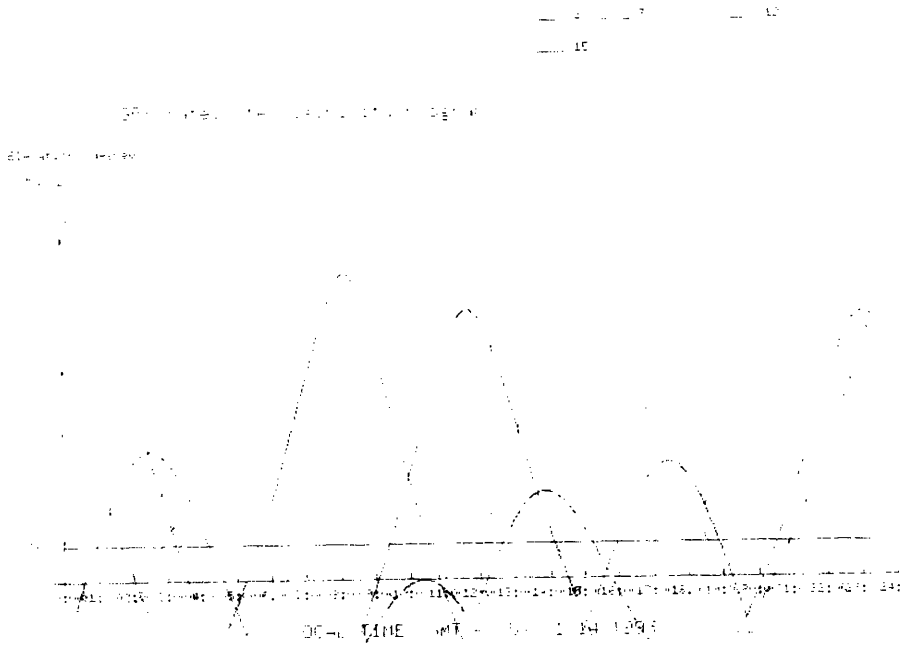


Figure 6-11: Satellite Visibility Plot for GPS Week 679

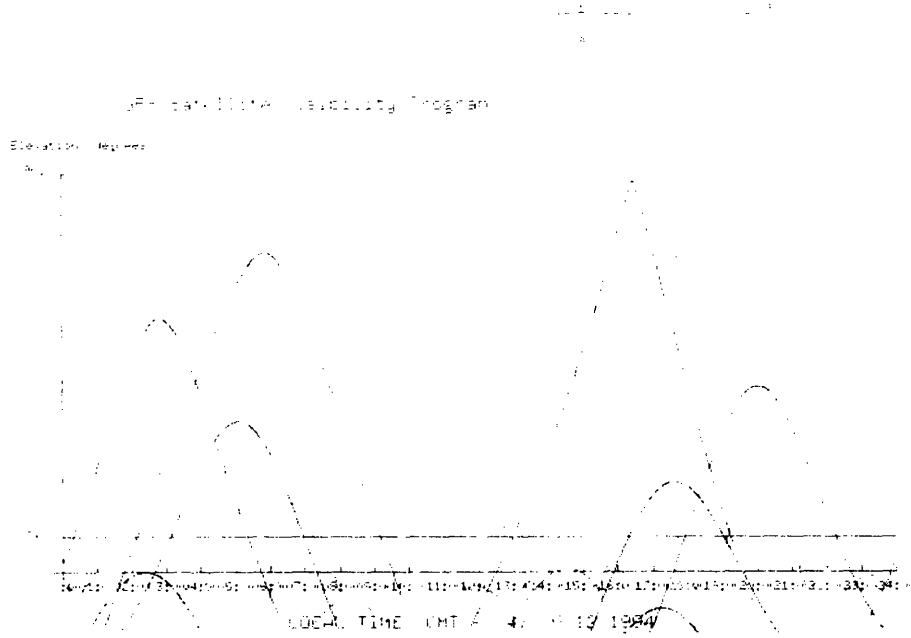


Figure 6-12: Satellite Visibility Plot for GPS Week 718

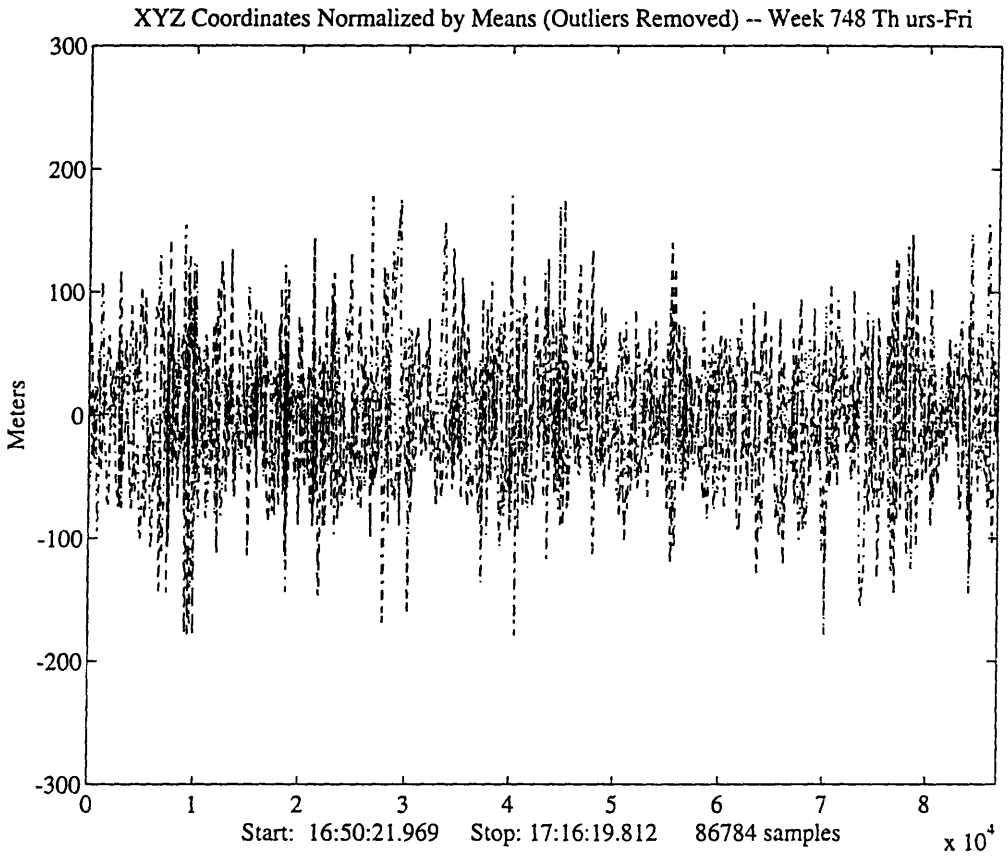


Figure 6-13: GPS Measurements taken May 1994

74° 10.1186' W with the ellipsoidal height set to 29.0 meters.[5] Although the location of the antenna of the base station and the antenna of the `6-Channel Boardset` receiver used in the calculations were not physically the same, but rather, offset in the horizontal plane by approximately one foot, we felt that since the precision of the system even with differential corrections would not achieve higher than a feet of resolution, we could tolerate this offset with negligible ill effects.

To test the system's performance in differential mode, we conducted several static measurements using the `6-Channel Boardset` again. However, this time the calculations were performed using differential mode, receiving corrections from our `reference` server. Not only would this test the accuracy of the differential GPS, and thereby justify the costs of a reference base station to our department head, but it would also test the server software described in Chapter 4. These experiments were conducted over a period of seven non-consecutive days in order to insure that the data was uncorrelated to any conditions arising during a particular week. The data was then post-processed in the same manner as we did with the uncorrected samples. However, the post-processing program, `gather`, was modified to communicate with the GPS receiver in the unit through the `navigation` server. An example of the differentially corrected data collected February 9-10, 1993 is shown in Figure 6-14. Figure 6-15 shows the smoothed data with outliers removed. The weight matrix for both the raw and smoothed data is shown in Figure 6-16. The calculated mean was the same as the previously computed reference position, which is what we expected. As you can see the variance is much less with differential corrections than without. The standard deviation of our data was approximately 3 meters which is much improved from data collected using regular GPS positioning. Figure 6-17 shows the differentially corrected raw data collected on the week of May 9, 1994 (GPS Week 748).

6.3 Kinematic GPS

In our previous testing of our system, the conditions were quite different from what they would be when operating in the field. The data was taken for one stationary point allowing us the opportunity to average the sample points to produce improved position fixes. However, under normal operating conditions, the user, and thus the receiver, will be moving and data will have to be available in real time. The fact the user will no longer be stationary will actually improve the positioning calculations since at high velocities, the higher doppler shift in the signals can be used to provide improved fixes. In addition, if the user is on the road, the line of sight of the antenna is typically less obstructed than in a city environment and therefore, the signals freer of multipath errors (unless the user is travelling under a tree-lined road). However, if the user is travelling with very slow dynamics, any gain in precision by utilizing velocity is lost. If the user moves slowly in a car or is walking, errors in GPS positioning become more prominent. The real-time restriction relegates post-processing or smoothing of

[5]The Trimble 4000 RL II has only one decimal place accuracy for the ellipsoidal height parameter.

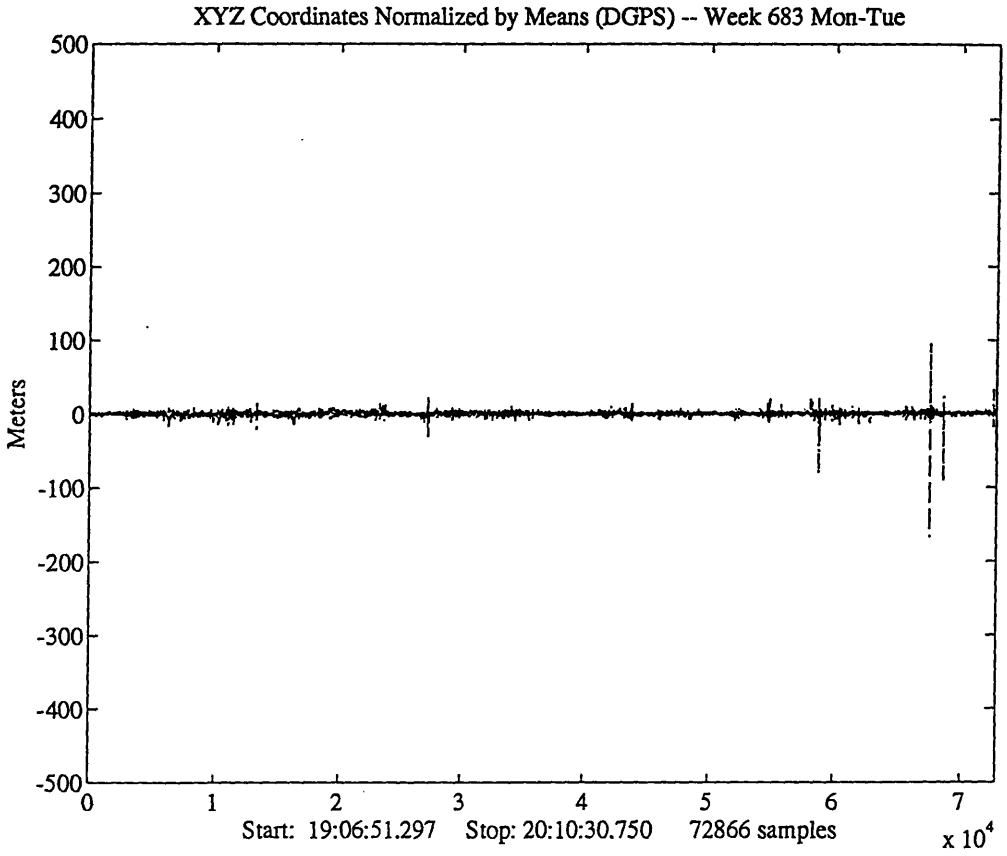


Figure 6-14: Differentially-Corrected GPS Measurements

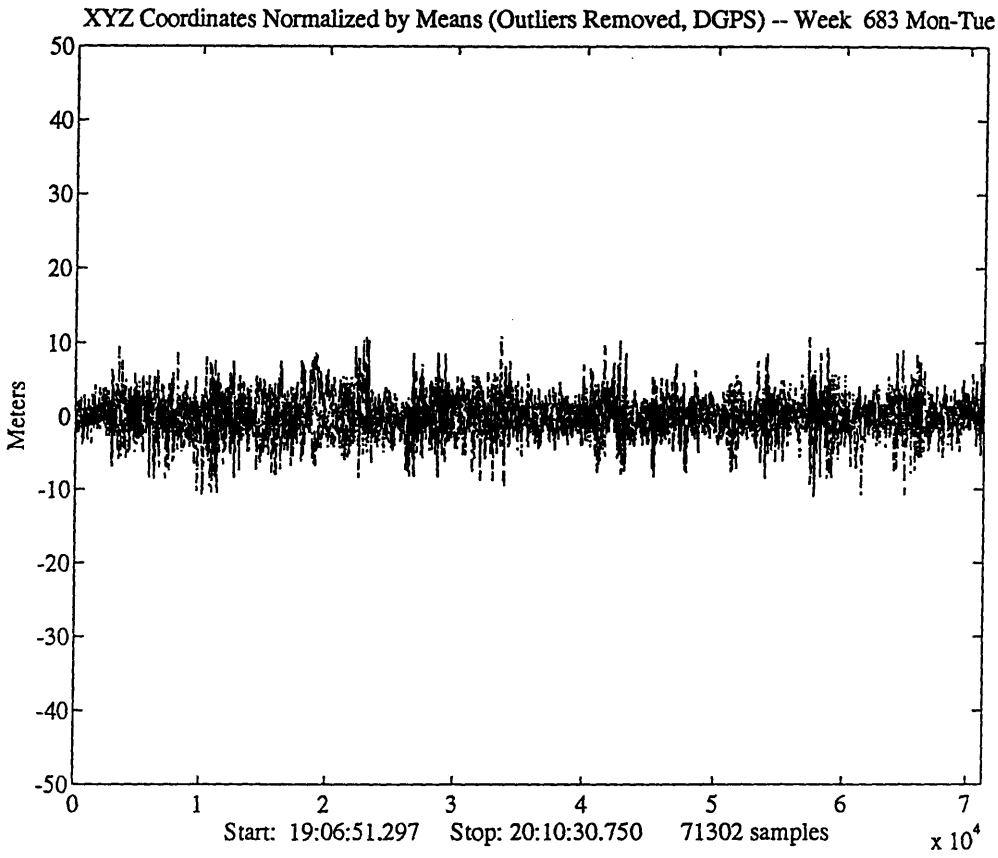


Figure 6-15: Smoothed Differentially-Corrected GPS Measurements

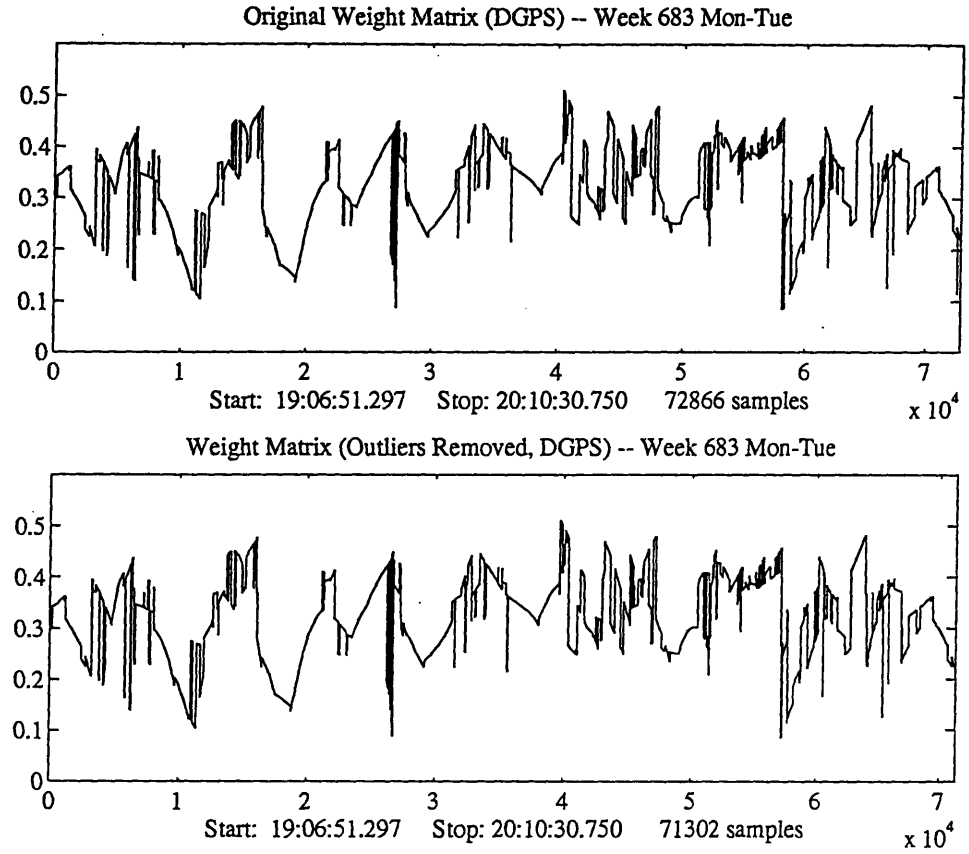


Figure 6-16: Weights of Differentially-Corrected GPS Measurements

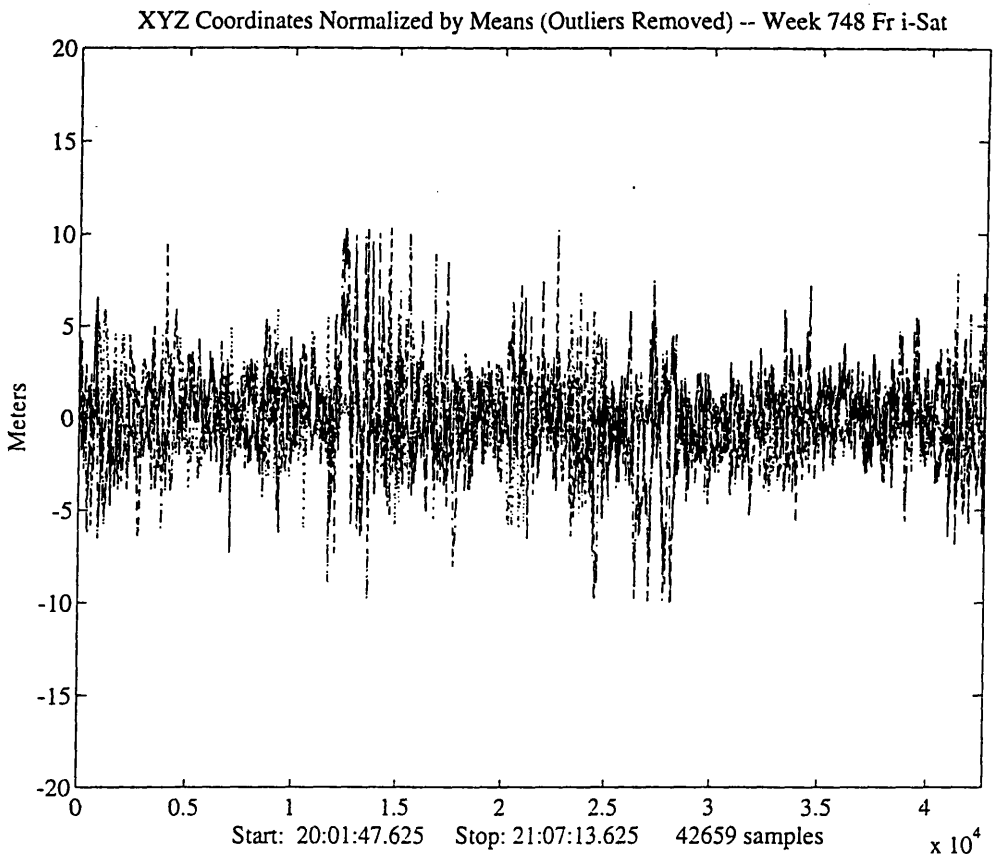


Figure 6-17: Differentially-Corrected GPS measurements GPS Week 748

data to a minimum, thereby requiring high precision in the raw position fixes.

For the following experiments, we did not use any quantitative measurements to indicate the accuracy of the data gathered. This was primarily due to the lack of accurate map information. For a quantitative error measurement, the sum of the errors of each calculated position from the actual trajectory traveled should be calculated. However, since the "real" trajectory traveled is not available to us, we will rely on qualitative judgments of smoothness of the trajectory of the computed positions to assess the system's kinematic performance.

An example of high dynamic kinematic positioning, Figure 6-18 is a recording of Michael Potmesil's trip from AT&T Bell Laboratories' Holmdel site to the location of our yearly lab picnic in July 1992. The data was recorded using a program that stored TSIP data into an output file. The data was then processed and superimposed on map data from the U. S. Census Bureau. Details of the implementation of the display code is given in the next chapter.[6] From the image, one notices that the receivers appears to produce fairly accurate results. However, zooming in on the Holmdel site, one notices significant errors (see Figure 6-19). The offset in the ovals are due to errors in the census data.[7] However, the drift was due to GPS errors since otherwise the driver would have to have been driving on the lawns of the Bell Laboratories' site. Other than the drift, the data was satisfactory.

Figure 6-20 is an example of data gather when the driver was driving rather slowly in order to gather as much sample points as possible. From the figure, we notice wild fluctuations in the trajectory. Believing the driver's assertions that he did not drive so recklessly, as the recorded data suggests, we concluded that the errors were probably GPS induced. We repeated the experiment several months later with differential corrections. The corrections were transmitted to the receiver using Morning Star's PPP as described in Section 4.5. The result is shown in Figure 6-21 As one can see, this time it appears that the driver stayed on the road and returned to the same point as he started on. The differentially-corrected data was much more stable around the Holmdel building than the uncorrected error.

Even when the user is not moving, differentially-corrected data is much improved over uncorrected data as was shown earlier in this chapter. Because the calculated positions still contain errors, we decided that in operation it would be best to average samples if time permitted between updates of the application program. However, this option is left to the application client developer since we wanted the designer to have as much flexibility as possible which implies access to as much information

[6]Some of the images have been colormap modified and will be appear lighter than other images generated from the same program. This is to provide higher contrast between the GPS data (the darker lines) and the database information (the lighter lines).

[7]At first, we had believed the errors to be GPS related. However, upon examining the Holmdel architectural files, we discovered that the tilt of the oval matched closely to our GPS data rather than the Census data. The Census data was then compared to a topological map from the Defense Mapping Agency and the error in the Census data was discovered. Because the error was not noticeable at a large scale, this leads one to suspect that this may have been due to digitization or human operator error. It is errors such as these in the current available digital map databases that makes GPS so attractive to those in the GIS industry.

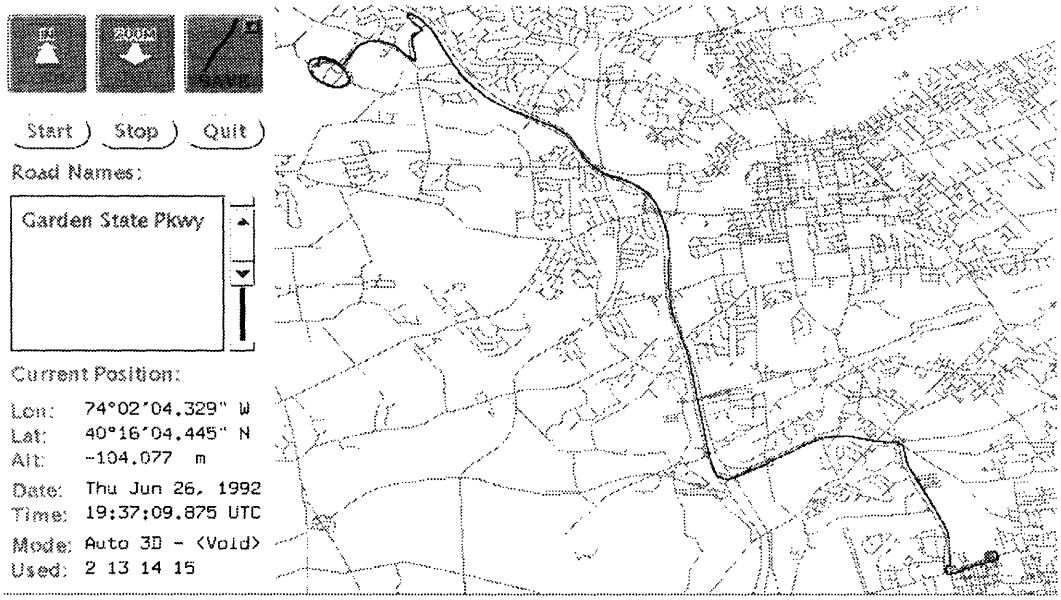


Figure 6-18: Travelling on the Garden State Parkway

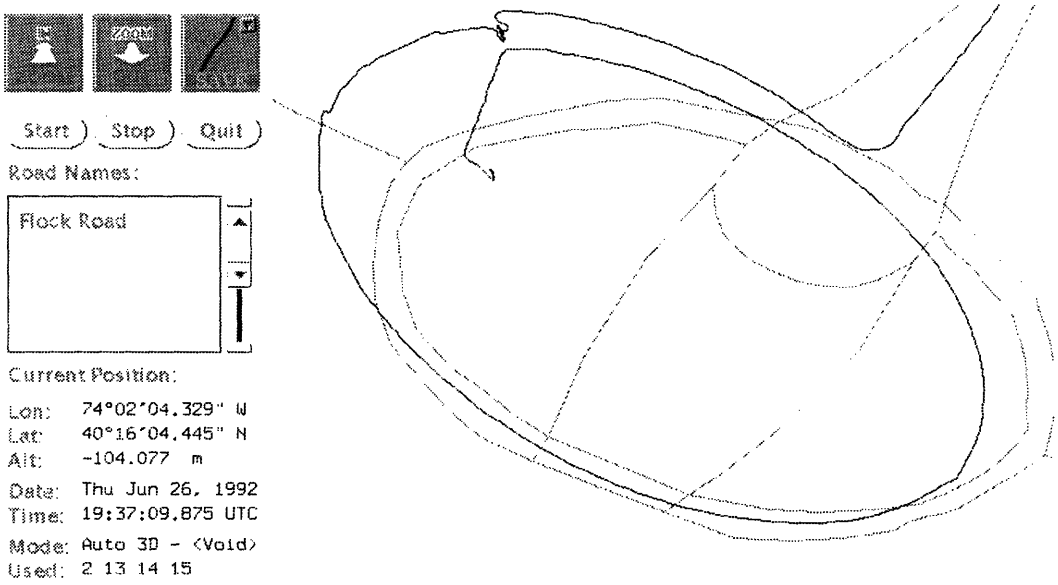


Figure 6-19: Holmdel Site High-Dynamics Measurements

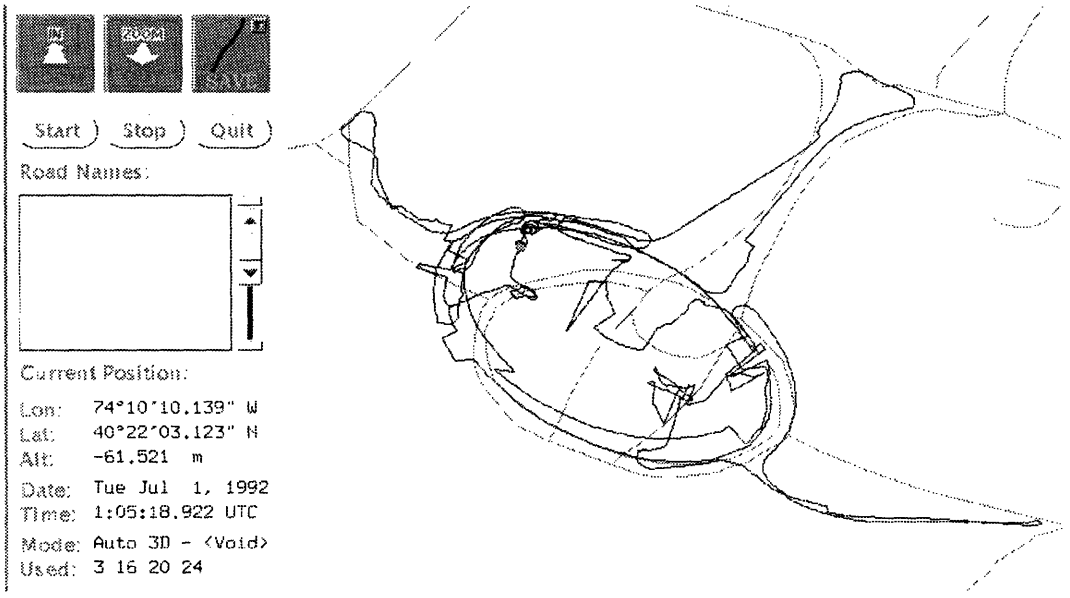


Figure 6-20: Kinematic Measurements Around the Holmdel Site

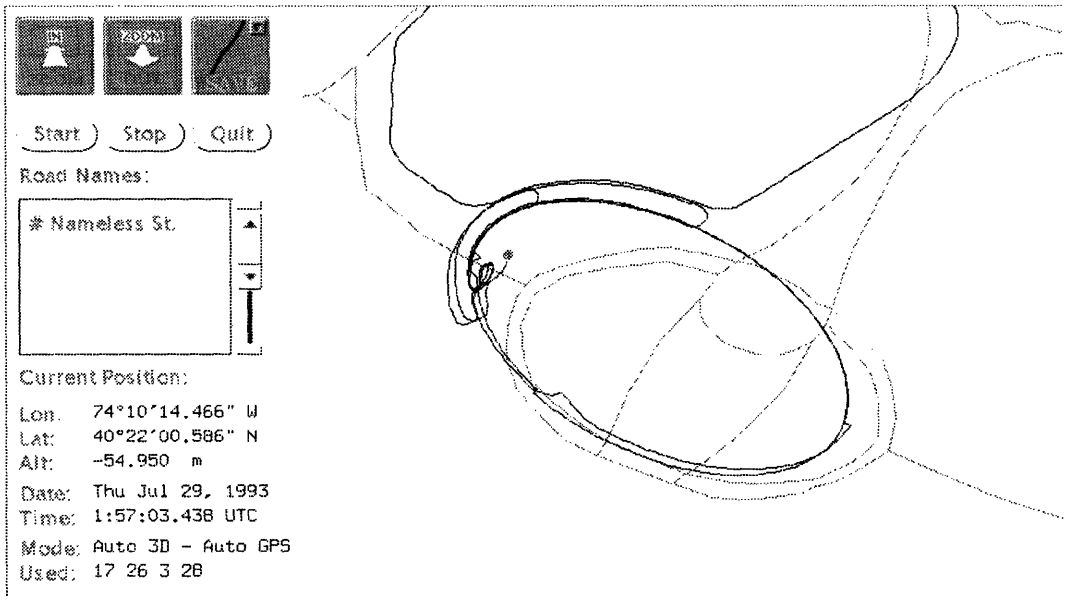


Figure 6-21: DGPS Kinematic Measurements Around the Holmdel Site

as possible. In the next chapter, we will describe a few sample programs that were developed to exploit the functions provided by our enhanced-reality system.

Chapter 7

Enhanced-Reality in the Great Outdoors

In the previous chapters, the implementation and testing of our enhanced-reality system was discussed. In order to exhibit some of the potential applications and uses of such a system, we developed a few proof-of-concept demonstrations that utilize our system's portable tracking capabilities. To that effect, we developed two types of programs that emphasize different aspects of our system. The first type of programs emphasizes the GPS positioning and local orientation tracking capabilities of our system. The second type of programs emphasizes the 3D graphics libraries described in Chapter 5 utilizing the tracking capabilities of our system to update the viewing perspective of the user. In these second type of programs, the programs either exploit the system's GPS positioning capabilities or its orientation-tracking capabilities. The following sections will provide a more detailed description of our testbed applications.

7.1 Somebody Is Watching You

To demonstrate the system's GPS positioning capabilities, a program called `mapper` was developed to not only track a user's global position and azimuth angle but also display the data graphically by overlaying a trajectory of the path traveled on a map of the local area. `mapper` is a program that primarily tests our GPS and server algorithms. It is a client to the navigation server described in Chapter 4. A program developed using the Xlib graphics library routines, `mapper` utilizes position information from the NAVunit and draws a line segment between the current LLA position point and the last sample point superimposed on a map generated with information from the U.S. Census Bureau's TIGER database. In our program, a modified TIGER database for Monmouth County, New Jersey consisting of two files, one containing the names of the roads and points and the second contains the actual latitude and longitude coordinates for those points. The original data was a modification of the original TIGER database into a format suitable for use with LISP machines. The ordering, and therefore the rendering, of the line segments was not in any particular sequence. Dave Weimer of AT&T Bell Laboratories was responsible for much of the

modifications to the database and the algorithm for accessing the information for display. We modified some of the user interfaces and added the navigation server interface to his code. The precursor to the `mapper` program was originally developed as a diagnostic tool to examine the precision of our GPS system. It was a simple overlay of connected GPS data points rendered in one color over the TIGER database map information which was drawn in another color. With the modifications from Weimer's code, we were able to improve access time of the data and also improve the aesthetics of the user interface. Using this program, we were able to generate the plots shown in the end of Chapter 6. Data from our runs was gathered first and then later overlaid on a map of the local area. In newer data sets, the data was simultaneously being gathered and displayed on `mapper` since in addition to displaying prerecorded data, `mapper` has the capability to display data in real-time. It can also save data in a compressed format consisting of only POSservice data packets. In addition, the choice of map locale is also settable as command-line arguments, with the restriction that the two designated input data files are in the appropriate modified TIGER database format.

The basic strategy used by the `mapper` program in displaying the user's current position was to keep the user's updated position displayed at all times. The user's current position is represented by a red dot 16 pixels in diameter. As the user changed her position, the dot's position was updated accordingly. In addition to the user's current position, the user's old position is displayed as connected line segments beginning at the point the user activates the `mapper` program up to her current position.

One feature of the `mapper` program is its ability to zoom in and to zoom out at varying resolutions. The top level image is a map of the entire Monmouth County centered in the viewport (see Figure 7-1). This is the maximum that a user can zoom out. However, zooming in is controlled by clicking the left mouse button and dragging the mouse to select the area of interest. The user is relegated to a selection window with the same width to height ration that is the program's default. The user can zoom in a maximum of 31 different levels. However, the resolution of the windows is only limited by how small of a window the user can draw using a mouse. Figure 7-2 shows a zoomed in image centered around the region of the AT&T Holmdel site.

Having introduced the ability to zoom, a number of questions arising from this feature must be dealt with. First, by zooming in on a particular area of the map image, we will end up clipping part of the image. Fortunately, the Xlib drawing routines handle clipping. However, in order to improve the computation time in rendering the map, we reduced the number of points needed to be rendered by not drawing any line segments whose endpoints rested outside the boundary of the image. In addition, zooming complicates our strategy in displaying the user's current position. Before zooming was introduced, the basic strategy used by the `mapper` program in displaying the user's current position was to keep the user's updated position displayed at all times. The user's current position was represented by a red dot 16 pixels in diameter. The coordinate system was in terms of longitude and latitude where longitude was the x-coordinate and latitude was the y-coordinate.[1] As the user changed her position,

[1]To be rigorous, we should have converted the longitude and latitude coordinates into a meters

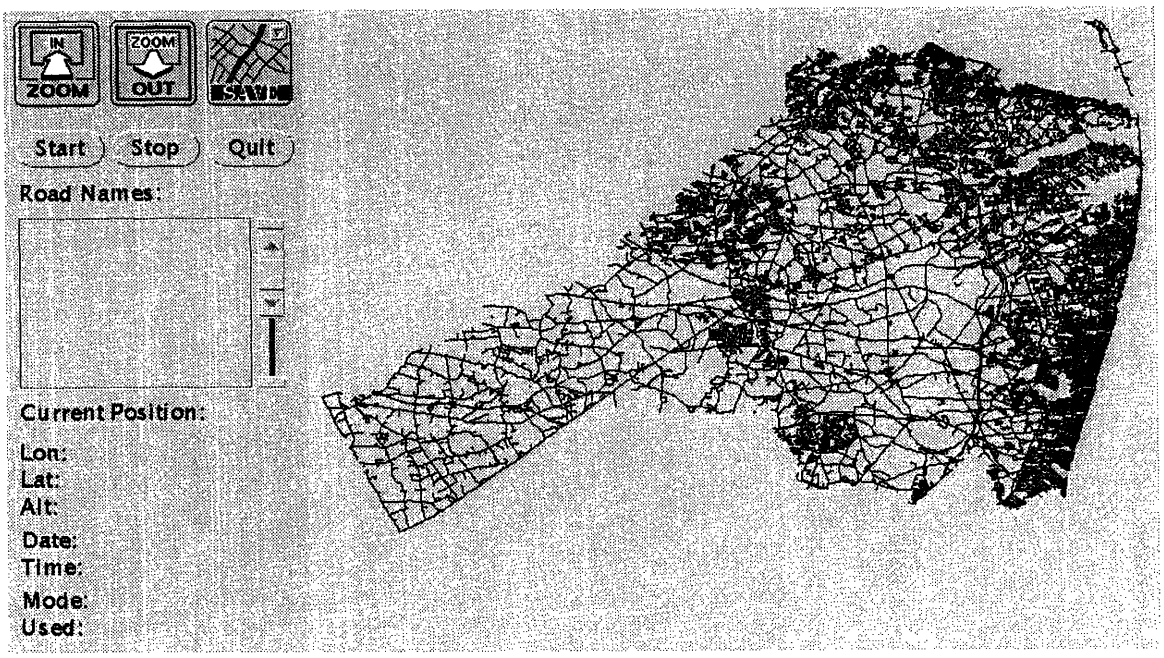


Figure 7-1: Monmouth County, NJ

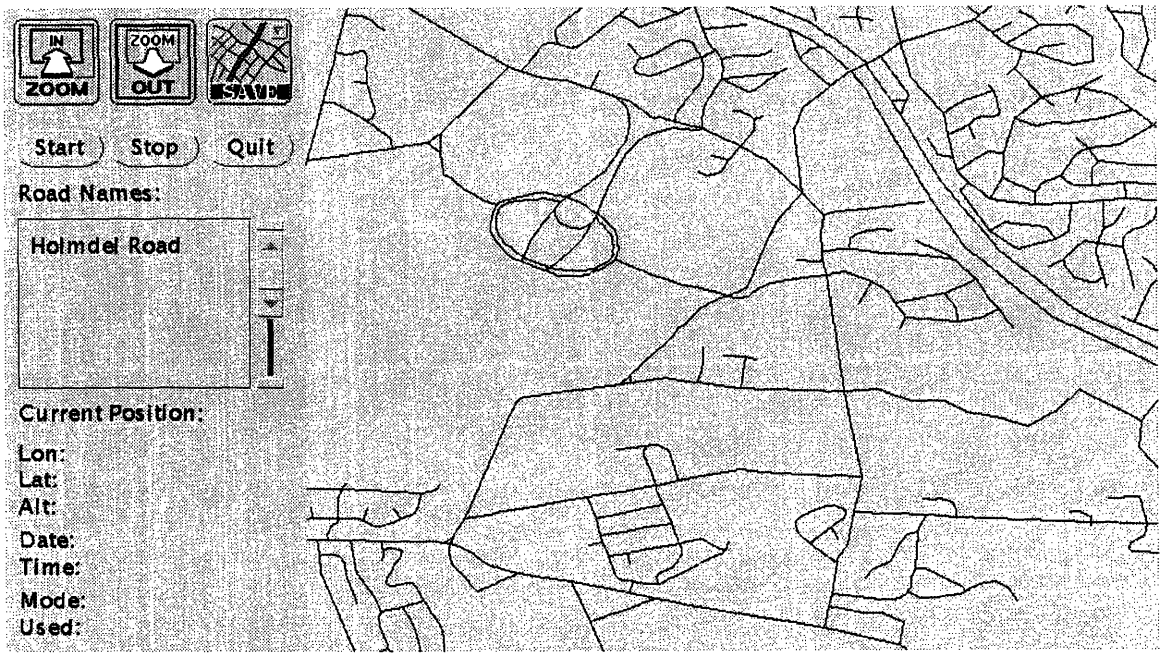


Figure 7-2: Zoomed Region of Holmdel, NJ

the dot's position was updated accordingly. In addition to the user's current position, the user's old position is displayed as connected line segments beginning at the point the user activates the **mapper** to her current position. However, the algorithm needs to take into consideration the possibility that the user can now conceivably move out of the area displayed by a zoomed in window.[2] To account for a user's restlessness, the algorithm was modified to update the section of the map displayed in the viewport such that the user's new position is in the center of the map. Since the overhead in redrawing the map is higher than the overhead in drawing the user, as represented by a dot of 16 pixels[3], we removed from consideration the option of having the underlying map database scroll while the user position is stable, but opted instead to update the user's cursor position. As the user changed her position, her cursor's position would be updated appropriately as well. However, if the user moved, such that her position is no longer visible on the displayed map, a new viewport into the map will be displayed such that the user's new position is centered. The scale of the map remains the same. The user's position is then updated as before, until she moves out of view, and then the process of displaying a new map repeats. This process reduces the amount of redraws that is necessary by a restless user while still providing the user updated information about herself. Figure 7-3 displays the position of the stationary GPS receiver's antenna on the Holmdel roof in conjunction with differential corrections. Figure 7-4 displays the same information but at a higher resolution where the drift inherent in GPS positioning is now evident..

A similar algorithm is used if the user's azimuth angle changes. In updating a user's position due to a change in azimuth angle, the main criterion was to try to maintain the user's heading to be toward the top of the window. Since the computation time required for rendering the map is quite high even with our improvements, the image is not rotated until the user's azimuth angle has changed by more than a threshold angle. The threshold angle was chosen arbitrarily to be thirty degrees. If the user's azimuth angle changes by more than this threshold, the map and user data are rotated accordingly. Equations 7.2 and 7.2 give the formulae that were used to transform the $[x, y]$ coordinates or the longitude, latitude respectively to the new coordinates $[x_p, y_p]$ of both the map and user data.

$$x_p = x\cos\varphi - y\sin\varphi \tag{7.1}$$

$$y_p = x\sin\varphi + y\cos\varphi \tag{7.2}$$

representation of distance from 0^o longitude and latitude. However, since the distances that we are dealing with are relatively small compared to the Earth's surface area, the differences in the results from the two projections of data were negligible. Therefore, the extra per point calculation that would have been required to convert from a longitude to latitude coordinate system was not worth the gained precision.

[2]This can also happen if the user was in an unzoomed state as well. However, this would require the user to move out of the loaded file, in this case Monmouth County database, which would make the display of updated data difficult since a new database would have to be loaded in. In the current implementation of **mapper**, a user is relegated to travelling within the database that was loaded in at runtime.

[3]The dot was a bitmap image of a red dot. Therefore, the cursor can also be represented by a more complicated cursor, such as gumby, without any extra rendering overhead.

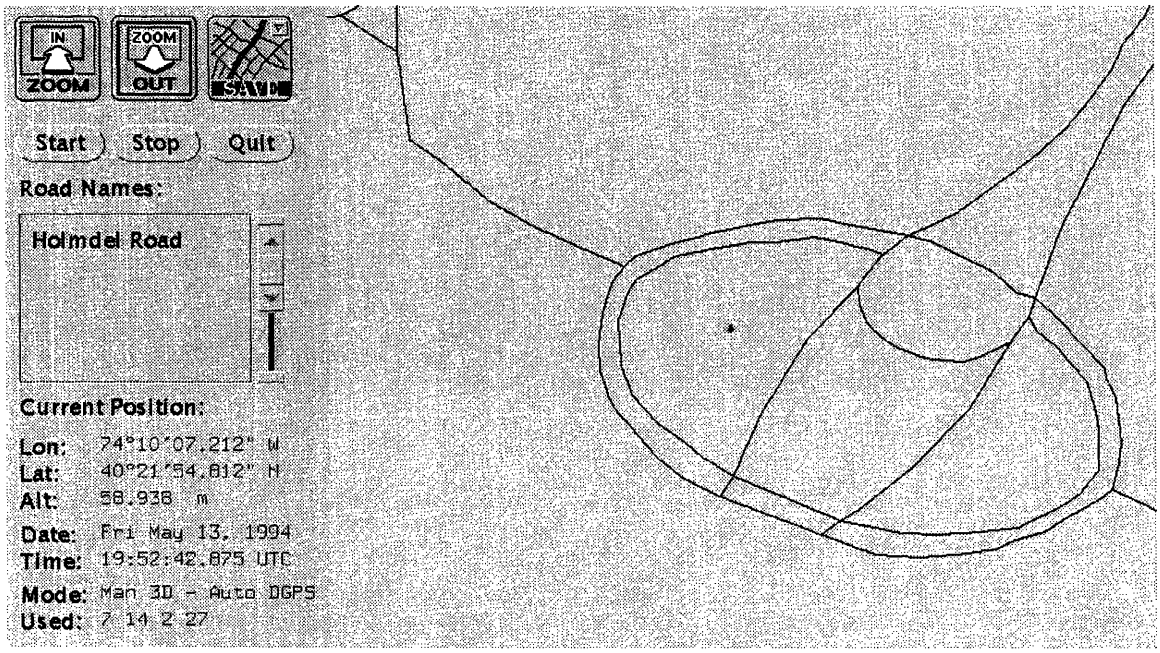


Figure 7-3: Roads on AT&T Holmdel Site

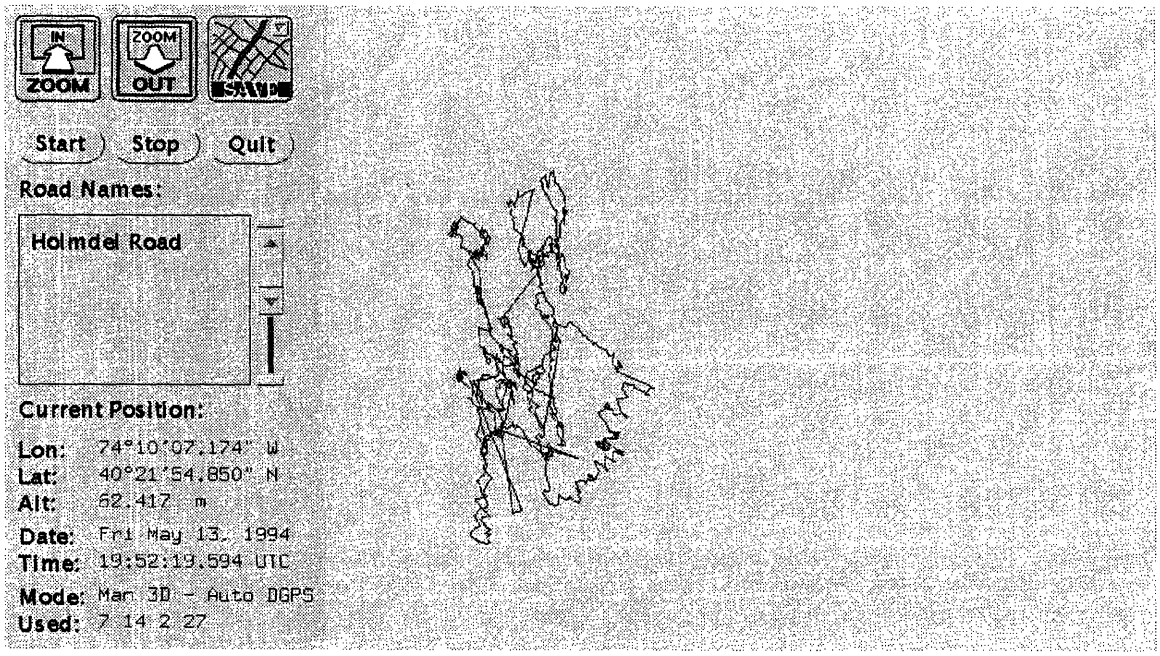


Figure 7-4: Drift on Holmdel Roof

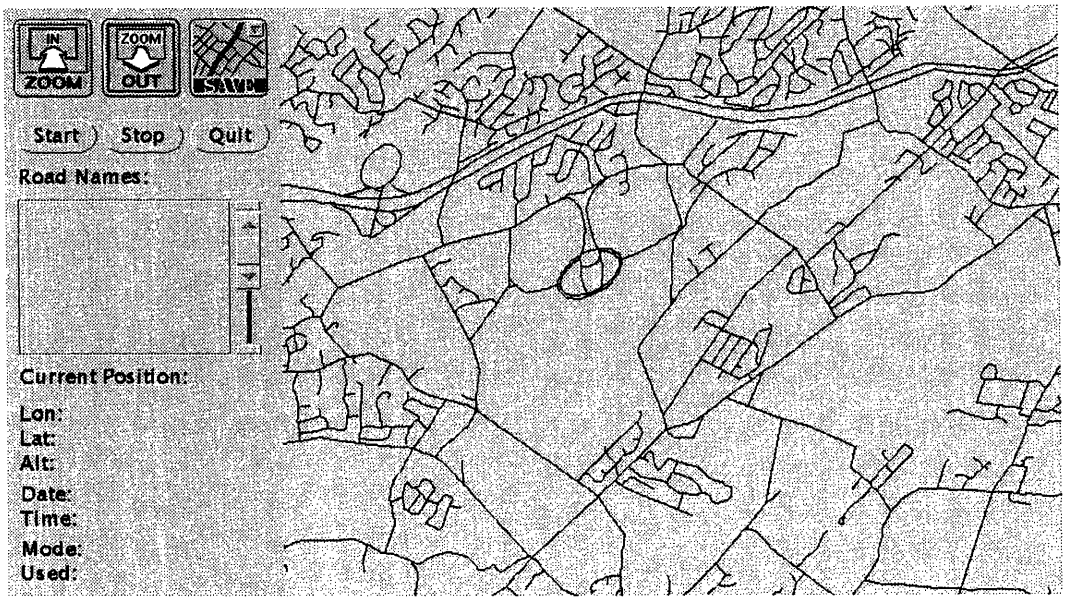


Figure 7-5: Updating a User's Azimuth Angle

Due to the overhead in calculating the rotated positions, the ability to update the map by the user's azimuth angle is left as a command-line option for the user. Figure 7-5 shows part of Monmouth County rotated by approximately 45 degrees.[4]

Another feature of the system is the ability to specify the hostname of the computer on which the navigation server is currently running. Because of the manner in which the navigation server was implemented (see Chapter 4), this feature allows a user to track other users' position trajectories in addition to her own. For example, with the `mapper` program running on a mobile and stationary system, both users would be able to see the current position of the mobile user updated in real-time on their respective displays. Although this feature can be construed as an invasion of privacy, it should rather be considered as a benefit that will allow two or more travelling parties to find and meet one another more easily. In addition, another use of this feature would be to guide a user to a new location since the other party will know where the mobile party is located without the necessity of the lost traveler having to give detailed descriptions of her locale. The ability of tracking another user's NAVunit can easily be extended to cases with more than two users.

Through text-to-speech synthesis routines that were developed by Mark Beutnagel of the Linguistics Research Department at AT&T Bell Laboratories, `mapper` also has the capability of announcing roads as a user approaches road intersections. The

[4]The angular rotation was simulated via a command line argument. However, rotation of the image has been tested and proven to work in real-time.

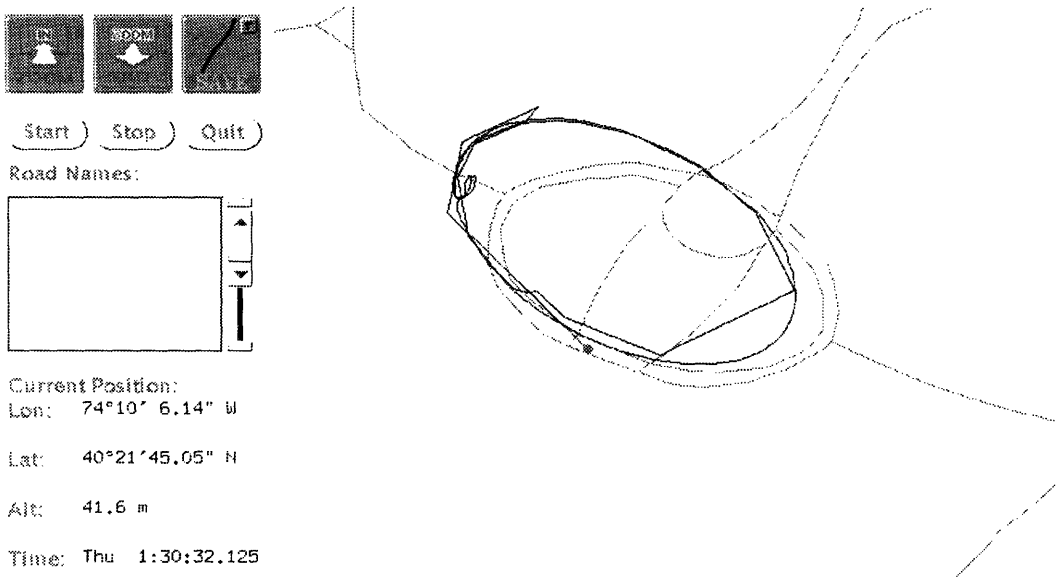


Figure 7-6: Tracking a Mobile Unit's User

criterion for intersection detection is a crude one based on the assumption that GPS position information is not in general very accurate and therefore the calculated position would lie off the actual road. Using a nearest road name algorithm which had originally been implemented to retrieve the name of the road nearest to the mouse pointer upon a mouse click, the program was altered to also pipe the string output through the audio port on either `alfalfa` and/or `amalfi`. Because text-to-speech synthesis requires a significant amount of computation and can also be a possible irritant to some users, it was also made an option to the user. Figure 7-6 demonstrates `mapper`'s ability with differential positioning enabled. The similarity to the figures in the end of Chapter 6 is not simply a coincidence. Those figures were generated using `mapper` and prerecorded data. They are also an example of time travel since they display data that was gathered in another time and yet displays the information as though it was being gathered in real-time. In Figure 7-6, which was originally displayed in real-time, Michael Potmesil, using the mobile unit consisting of the navigation unit and a SparcBook 2, which we affectionately termed `amalfi`, reported his position to our in-door server `alfalfa` which tracked him as he drove around the AT&T Bell Laboratories Holmdel site. One should notice the irregularities in the trajectory curve as compared to Figure 6-21. This is due to the loss of data over the cellular link. Hopefully, with improved cellular technology, our system's performance will improve.

7.2 Satellites and Other Heavenly Bodies

7.2.1 Satellites

Having demonstrated the global positioning capabilities of our system, a second program, `el-az-gps`, was developed which emphasizes the system's ability to determine local orientation and update its display graphics accordingly. Similar to `mapper`, the `el-az-gps` program was originally developed as a diagnostic tool. The original code from Trimble Navigation was used to generate the satellite visibility charts shown in Chapter 6. The code was modified to display graphics in real-time and in a different coordinate system. Therefore, in using `el-az-gps`, the user gains insight on which satellites are visible to the GPS receiver's antenna at any particular time and can visualize the relation of the satellites positions to one another which resulted in the combination of 4 satellites that was used in the navigation solution. In addition, since the satellites are not stationary with respect to the Earth, but instead make two revolutions about the Earth daily, the ability of our system to display moving objects will also be demonstrated. For `el-az-gps`, the display graphics was also written using Xlib graphics library routines.

`el-az-gps` was a demonstration program which displays the principles of GPS and the ability of our system to show local orientation. The trajectories of the satellites are plotted by obtaining the overhead satellites' Keplerian elements. Since the almanac and ephemeris information of all satellites are broadcast in the navigation message, the trajectory of any satellite, even those not visible, can be plotted with respect to any given point once the navigation message is received and decoded using the formulae given in Table 2.5 in Chapter 2. For `el-az-gps`, although the ephemeris information provides more accuracy in calculating a satellite's position, the almanac information was used instead since it requires less acquisition time to obtain requiring less bits to transmit as was discussed in Section 2.2.2.1. Because almanac information is valid for approximately 24 hours (as opposed to ephemeris information which is a least squares curve fit for a period of typically only two hours), we only need to request almanac information once every twenty-four hours, thereby consuming very little of the GPS receiver's resources.

Once the Keplerian elements of the satellites and the user's location have been obtained, `el-az-gps` plots the flight paths of the overhead satellites. Each satellite is a color coded bitmap. The trajectories are polar plots of the satellites elevation and azimuth angles over time. The elevation angle is the angle from the horizon. The azimuth angle determines the angle from the axis pointing North. The perspective is that of a viewer centered in the origin and looking up. A satellite is considered visible only if the elevation angle of the satellite is above the elevation mask of the receiver. Using these criteria, `el-az-gps` plots the satellites trajectories into an X window as shown in Figure 7-7. In addition, the user also has the option not to draw the trajectories of the satellites but to draw only the satellites. This has the advantage of using less memory and less time to refresh the image. Figure 7-8 is an example of `el-az-gps` without the trajectories plotted.

Just as the `mapper` program has the capability to rotate the map image based on

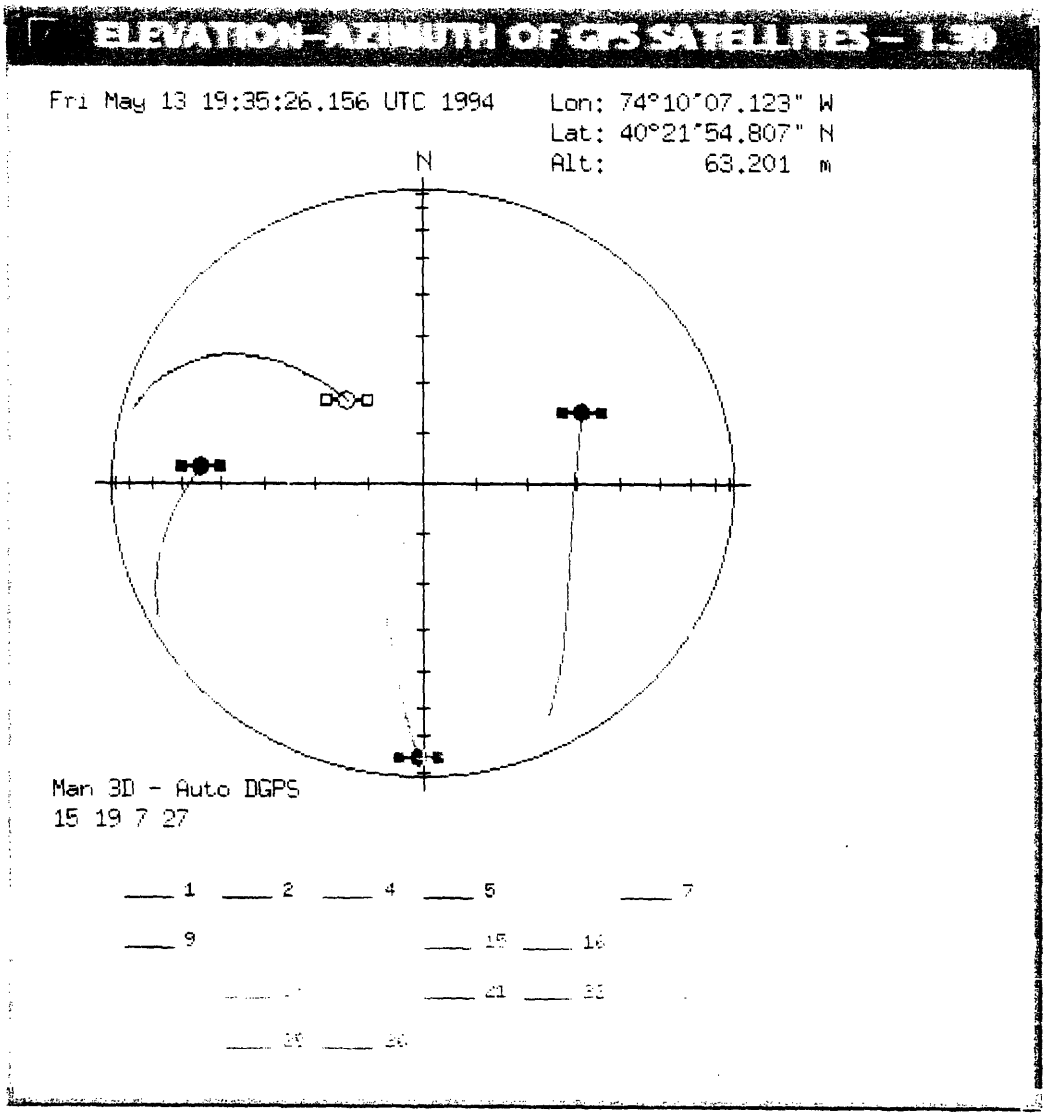


Figure 7-7: Overhead Satellite Trajectories on May 13, 1994

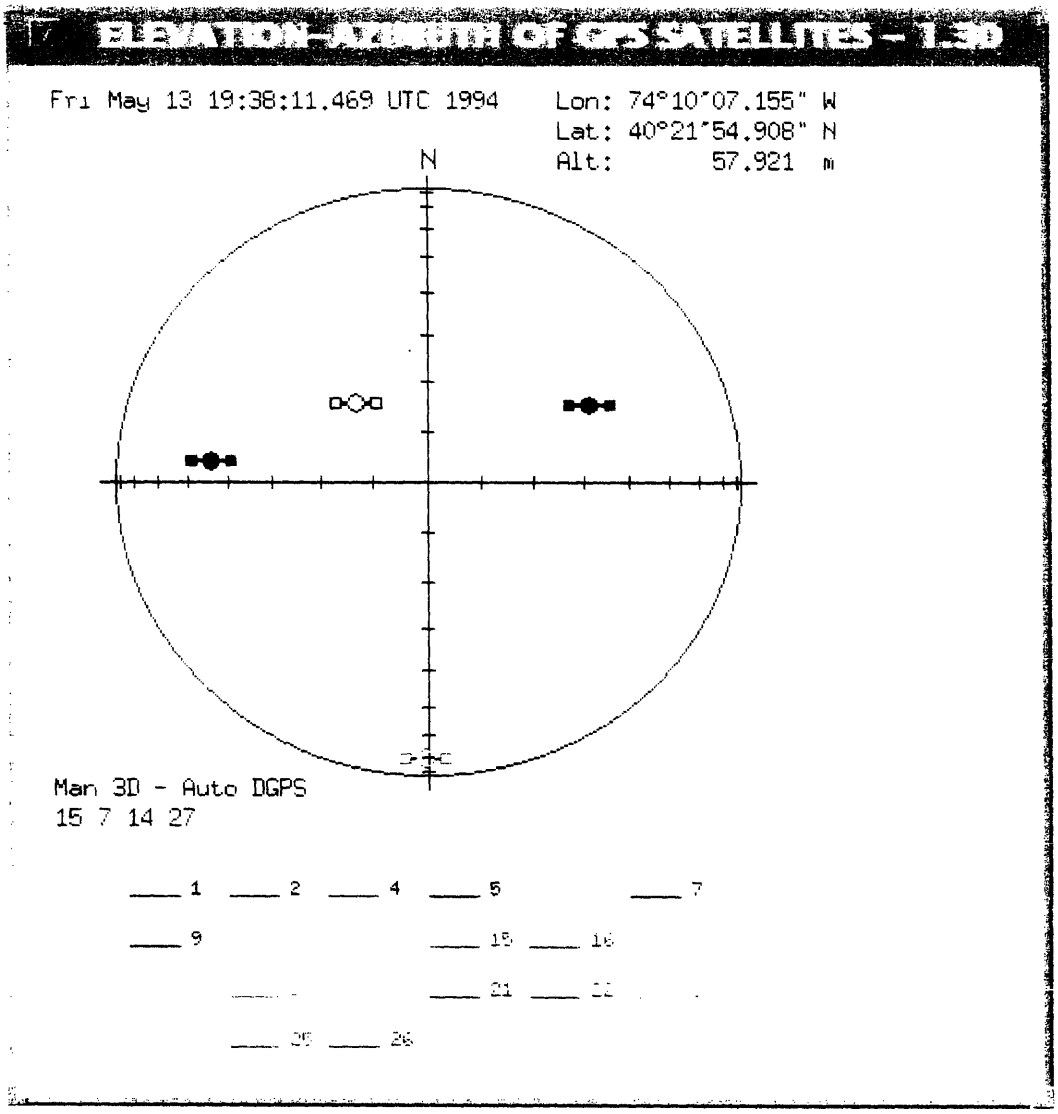


Figure 7-8: Overhead Satellites without Trajectories on May 13, 1994

the user's azimuth angles, the `el-az-gps` program can also rotate its image based on the azimuth angle. As the user's azimuth angle changes, the picture is also rotated. For example, if the mobile unit is rotated 20 degrees clockwise, the trajectory plots will be rotated 20 degrees counterclockwise as shown in Figure 7-9. In addition, even if the user is stationary, the paths displayed will be rotated to account for the user's orientation with respect to magnetic north as determined by the compasses. If the user moves her physical location, changing her GPS coordinates, the `el-az-gps` image will be appropriately translated. If either rotation or translation of the mobile unit is done rapidly, `el-az-gps` keeps up with the shifts and updates its display accordingly. The latency involved with `el-az-gps` is not as high as for `mapper` since it has significantly less number of lines to draw. Even if the user is stationary, if one waits long enough, one can see the satellites move along their trajectories. The satellite positions are updated at a rate specified by the user on the command-line. If no rate is set, the system defaults to once per second.

Since the `el-az-gps` program is a client of the navigation server, many users can see which satellites are above other users in the same manner as described previously for `mapper`. This can provide a convenient method to test how far away a user can be from another user before the overhead satellite configuration changes significantly.

7.2.2 Heavenly Bodies

Similar to the `el-az-gps` program, an astronomy program `sun-moon-gps` was developed, which displays the sun's and moon's positions at any given time of day from the user's position. The display methodology is very similar to that used in `el-az-gps`. Instead of satellite icons, a sun and a moon icon are used. The trajectories of the sun and moon are plotted with respect to the user's centered position. Once the sun has set, the icon changes color from a sunny yellow to a sunset red. The same holds true for the moon. When the moon is above the horizon, the moon is white. When it is below, it becomes blue. Thus, this program differs slightly from `el-az-gps` in that it emphasizes timing capabilities of the GPS system in addition to positioning. There is also a simulation version of the program which can simulate time passing quickly and one can see how the sun's and moon's positions change over the course of a year. As in `el-az-gps`, the sun and moon are plotted in polar coordinates in terms of their elevations and azimuth. Figure 7-10 shows a snapshot of `sun-moon-gps`. Figure 7-11 shows results of the simulation version of the program, `sun-moon-sim`, displaying the position of the sun and moon during the eclipse on May 10, 1994 from the vantage point of the roof of the Holmdel building. Notice that the moon and the sun overlay one another.

The implementation of `sun-moon-gps` was done using algorithms from Duffett-Smith [3]. The algorithms, meant for calculators, were easily converted into library routines to be used by programs. In dealing with astronomical questions, a primary consideration is how to measure time. There are various timing systems in use such as UTC, TAI, GMT, LST, GST, and so forth. The difference is often only a few minutes or dependent on the user's longitude position. Another consideration is time of year. Currently, the United States is using a Gregorian Calendar. However, the

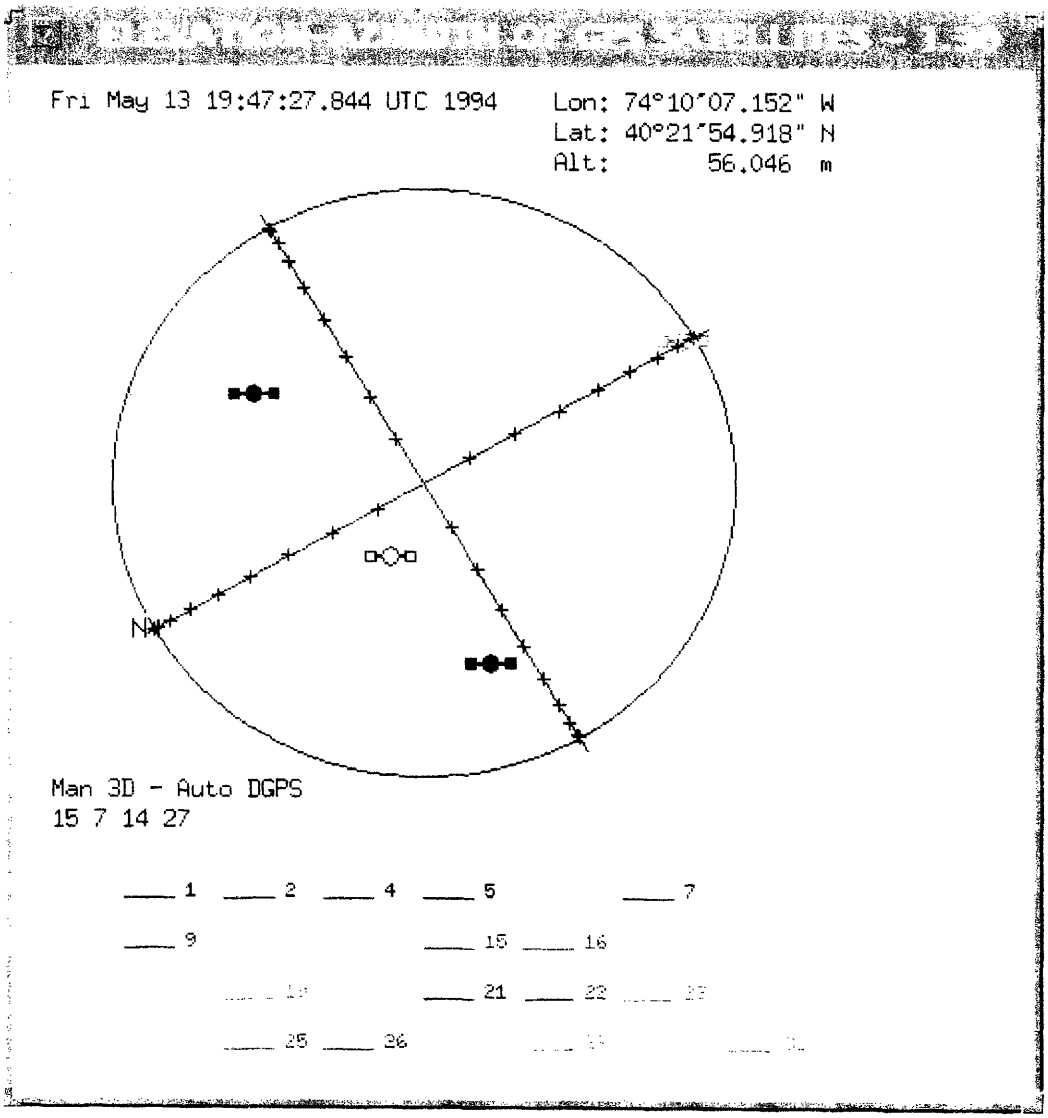


Figure 7-9: `el-az-gps` Rotated

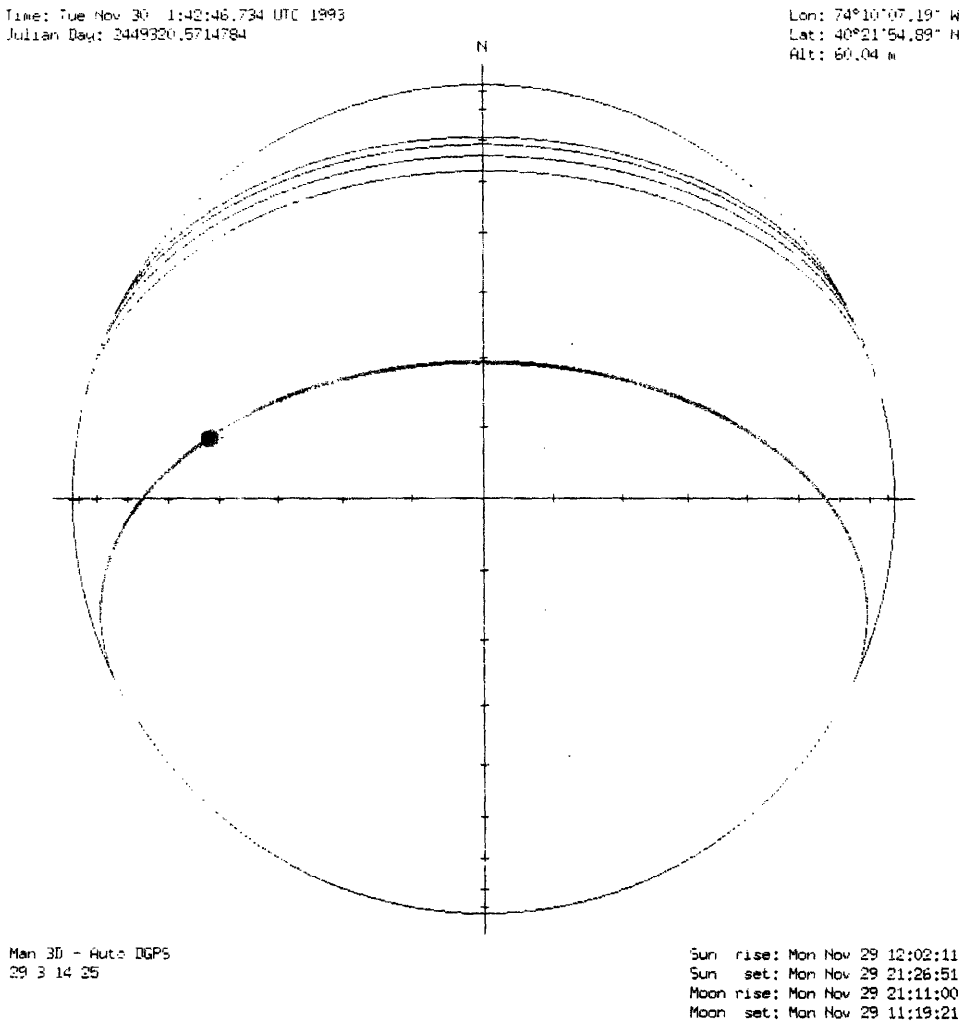


Figure 7-10: Sun and Moon

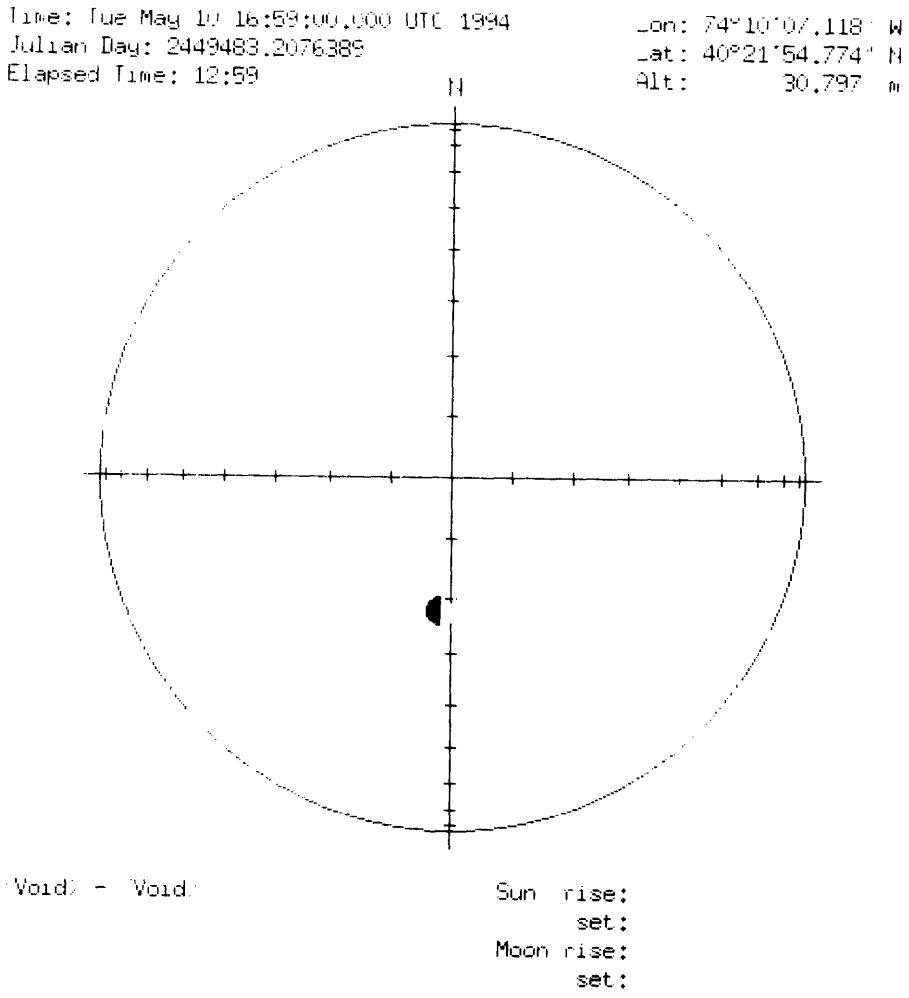


Figure 7-11: Simulated May 10, 1994 Eclipse Sun and Moon Positions

calculations used in determining the sun's and moon's position is performed using the Julian Calendar. Therefore, before calculating the sun's position, GPS time and date needs to be converted to its equivalent Julian day (JD) where time of day is converted to fractions of a JD.

In addition to varying time systems, there are also various coordinate systems to express the position of celestial bodies. We are concerned primarily with ecliptic, equatorial and horizontal coordinate systems. The ecliptic coordinate system is typically used when comparing positions of objects in the solar system. The ecliptic is the Earth's orbital plane around the Sun. The coordinates are in terms of ecliptic longitude and ecliptic latitude. The equatorial coordinate system is given with respect to the Earth's equatorial plane. The coordinates are in terms of right ascension and declination angles. The declination angle is the number of degrees the object is away from the equatorial plane. The right ascension angle is the angle away from the vernal equinox or first point of Aries. The horizontal coordinate system is the elevation and azimuth coordinates that we have been using all along. There are formulae to convert from ecliptic to equatorial and equatorial to horizontal coordinates. As mentioned previously, these were given by Duffett-Smith.

In order to calculate the sun's position, one only needs to know the Julian day. From the JD, the sun's ecliptic and therefore equatorial position can be calculated. However, the horizontal position of the sun can be calculated only if the user's latitude is known. The plotted $[x, y]$ coordinates can then be determined as follows:

$$x = r \cos\theta \sin\varphi \tag{7.3}$$

$$y = r \cos\theta \cos\varphi \tag{7.4}$$

where θ is the elevation angle and φ is the azimuth angle. A similar algorithm was implemented to display the moon position. An additional feature in the moon display was the ability to display phases of the moon. This was done by calculating the moon phase and displaying the appropriate bitmap for the moon icon. Only four phases – new moon, first quarter, full moon and third quarter – are displayed. Figure 7-12 displays a new moon where the moon is represented by a hollow ring since the new moon is invisible. The blue color of the moon indicates that it is below the horizon. The viewpoint is from the perspective of an observer on the South Pole in Spring 1989. Because we can see moons even when they're new and we can visualize where the sun and moon are with respect to our position, even when they are below the horizon, `sun-moon-gps` is a prime example of enhanced reality which allows the user to "see" things that otherwise is not visible to her. In addition, it demonstrates the user's ability to simulate travel in time and space.

7.3 SCULPT

The two previous sections dealt primarily with rendering 2D line drawings of physical models using Xlib library routines. However, 2D representations of information would not be enough to justify the terming of our system as an *enhanced reality* system.

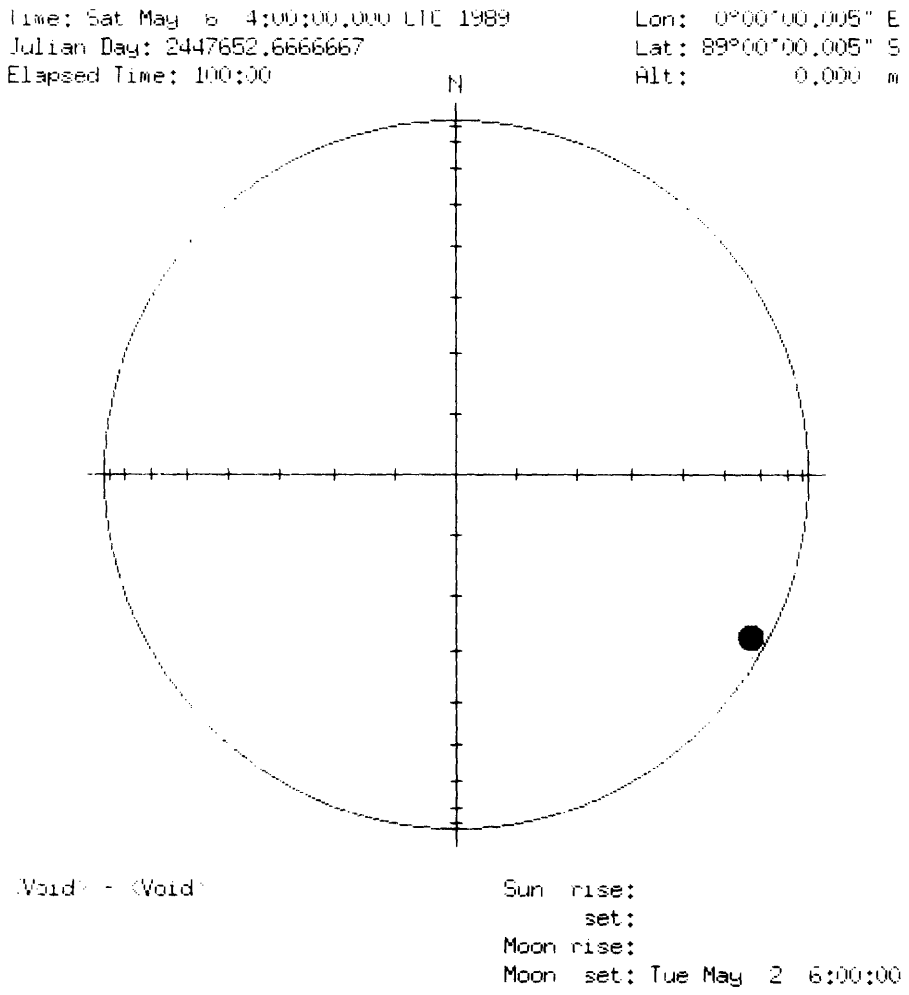


Figure 7-12: New Moon that Has Set in the South Pole

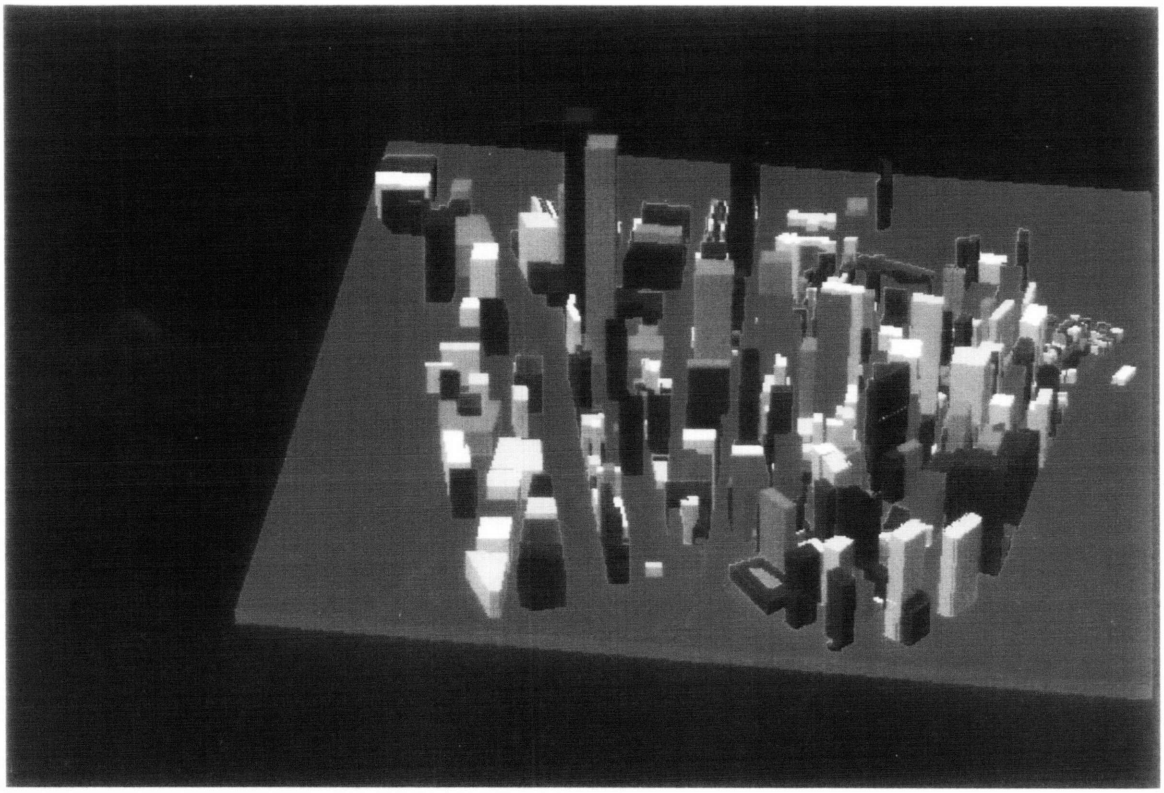


Figure 7-13: BSPT Representation of Lower Manhattan

Towards adding more realism in our system, we made use of our 3D graphics libraries previously described in Chapter 5. `sculpt` is an interactive program that lets the user manipulate models of 3D objects rapidly in real-time. In our current implementation, `sculpt` is used only to demonstrate the fast rotation of objects using the navigation unit. It emphasizes the ability of our system to track the local orientation angles. In the current implementation, `sculpt` has not been used to demonstrate the NAVunit's GPS capabilities. This was somewhat due to the position independence of the BSP tree representation. In addition, because all images displayed by `sculpt` need to be objects initially represented by a BREP, all objects needed to be connected regions. We were able to load in a BSP tree representation of an AutoCAD file of lower Manhattan. However, the results looked very artificial, as shown in Figure 7-13, with all of the entire Manhattan area buildings connected to a plane in order to create a Manhattan BREP. In addition, the sense of one's position in the image was lost although we were still able to rotate the object. Possibly in future upgrades of the system, the `sculpt` tools can be used for greater effect.

7.4 AutoCAD

For the final demonstration, we developed a program, called `Xdxfdraw` which performs in a similar fashion as `mapper` except that GPS information is overlaid on AutoCAD files rather than a map of Monmouth County. A reason for the desirability of such a feature is that it allows the user to view architectural information, such as underlying roads and changes that have been done to the site, that is otherwise unavailable to the viewer. In addition, site-specific maps often contain more detail and precision than generic databases, such as the TIGER database, which contains little information on a lot of topics. It was only by comparison of the AutoCAD files with the TIGER database that the errors in the database was localized.

The nucleus of the `Xdxfdraw` program consists of a DXF parser which converts the default format file for AutoCAD files into a more suitable format. Michael Potmesil created a parser which converted DXF files into a format suitable for input by the Pixel Machine. In writing `Xdxfdraw`, we cannibalized his parsing program while modifying his rendering routines to utilize Xlib drawing commands. The image consisted primarily of line segments between two point, with curve fitting being required depending on the image. The curve fitting, or handling of the *bulge* data type, was a bit more complicated than connecting a line between two consecutive points in that the *bulge* structure consisted of the enpoints of a curve and the angle subtended by a chord connecting the two points. Rendering the bulge involved calculating the radius and perpendicular distance to the subtending chord from the point that would be the center of the circle that generated the arc segment. From these parameters, the center of the circle used to draw the arc segment can be determined and therefore the arc segment drawn.

The algorithm for displaying the user's current position is very similar to the one utilized by `mapper`. However, for `Xdxfdraw`, we also need to concern ourselves with transforming either the GPS data into the coordinates of the AutoCAD file or the AutoCAD file into GPS coordinates since, unfortunately, most site plans are done in a local coordinate system independent of geodetic longitude and latitude. For the data to be useful, we must translate coordinate systems. To do this, a transformation matrix needs to be calculated which will transform data from one coordinate system into the other.

7.4.1 Determining the Transformation Matrix

In order to calculate a transformation matrix to adjust between the coordinate system of the building and holmdel facilities and the geodetic longitude, and latitude coordinates[5] that we have been using, we want to solve a series of equations of the form:

$$T_{lonx}Lon + T_{latx}Lat + Cx = x \tag{7.5}$$

[5]The altitude and height coordinates can be ignored since for this application we will be dealing primarily with the 2D projection of data onto the horizontal plane.

$$T_{lony}Lon + T_{laty}Lat + Cy = y \tag{7.6}$$

Equation 7.5 and Equation 7.6 can be represented in terms of matrix multiplication as:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} T_{lonx} & T_{latx} & C_x \\ T_{lony} & T_{laty} & C_y \end{bmatrix} \begin{bmatrix} Lon \\ Lat \\ 1 \end{bmatrix} \tag{7.7}$$

Since we have six unknowns, we need at least 3 points to solve the system of equations. However, this is assuming that our measured data is precise which typically will not be the case. Thus, by overdetermining our solution, a better estimate of the necessary transformation matrix may be obtained by taking measurements m distinct points where $m \geq 3$. The solution then becomes a linear least squares estimation problem.

To solve for the transformation matrix, let $\vec{p_i} = [lon, lat, 1]^T$ be the position vector representing the ith measurement's 2D coordinates. We then represent our samples vector $\mathbf{A}$ as:

$$\mathbf{A} = \begin{bmatrix} \vec{p_1} \\ \vec{p_2} \\ \vdots \\ \vec{p_m} \end{bmatrix} \tag{7.8}$$

where $m \geq 3$ and $\mathbf{A}$ is an m x 3 matrix. Letting $\vec{r_i} = [x, y]^T$ represent the ith data point in the Holmdel building coordinate system, $\mathbf{B}$ containing the set of data points can then be represented as:

$$\mathbf{B} = \begin{bmatrix} \vec{r_1} \\ \vec{r_2} \\ \vdots \\ \vec{r_m} \end{bmatrix} \tag{7.9}$$

where $m \geq 3$ and $\mathbf{B}$ is an $m \times 2$ matrix. If we represent the desired transformation matrix to be the matrix X where:

$$X^T = \begin{bmatrix} T_{lonx} & T_{lony} \\ T_{latx} & T_{laty} \\ C_x & C_y \end{bmatrix} \tag{7.10}$$

an equivalent solution for Equation 7.7 can be then be represented as:

$$\mathbf{B} = \mathbf{A}X^T \tag{7.11}$$

The transformation matrix can therefore be computed as follows:

$$X^T = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{B} \tag{7.12}$$

To calculate the transformation matrix, we need the geodetic latitude and longitude coordinates of at least 3 points whose equivalent Holmdel building coordinates are known. Although the AutoCAD files contain the coordinates of the points in the Holmdel building coordinate system and the geodetic latitude and longitude co-

ordinates of all the same points in the Holmdel building is determinable, registering these points to the precise position where the geodetic latitude and longitude coordinates were measured is not easily done unless the measurements are done on a marked building whose position in the Holmdel building is known. For these reasons, four points – the four corners of the building – were used to calibrate the transformation matrix. These four points were selected due to the ease of calculating their position in the Holmdel building coordinate system. Although more sample points would have improved the precision of the transformation matrix coefficients, points in building coordinates other than at the corners of the building would have been would have been very difficult to calculate. This is due to the fact that we were given explicitly only the Holmdel building coordinates of the center of the building ($[\ x, y\ ] = [\ -11826.777, -12069.407\ ]$) and the building's dimensions (length = 1031.583, width = 360.75). From this information, the coordinates of the four buildings in the Holmdel building coordinate system is readily calculable.

In order to obtain the geodetic coordinates of the four corners of the building, we performed similar experiments as was done in Chapter 6 to determine the position of the the GPS reference station's antenna. Utilizing `amalfi` and the MicroTacLite and CellularConnection modem and the modified `gather` program described in Chapter 6, differentially corrected data was collected over a period of approximately half an hour for each of the corners. The data was collected on the roof of the Holmdel building in order to improve the satellite visibility of our antenna. The results of the data collection with outliers removed are shown in Figure 7-14, Figure 7-15, Figure 7-16 and Figure 7-17.

The data for each corner was then averaged in the same manner as described in Section 6.1 and the answers converted to LLA representation. Because of safety concerns, the GPS receiver's antenna could not be placed precisely on the corners of the building without going over the guard rails on the roof. Instead, the antenna was placed on the corner formed by the guard rails. To adjust for this offset, we measured the distance of the antenna from the actual corner and adjusted the Holmdel building coordinates accordingly. Table 7.1 summarizes the GPS measured positions and Holmdel building coordinates for each corner. The data points in the building coordinates and in the geodetic coordinates are shown respectively in Figure 7-18 and Figure 7-19. The coordinates shown in Table 7.1 were then substituted into Equation 7.12 to obtain the transformation matrix. The matrix to transform from geodetic coordinates to Holmdel building coordinates is:

$$X = \begin{bmatrix} -2.1893 \times 10^5 & 2.3352 \times 10^5 & -2.5675 \times 10^7 \\ -1.8362 \times 10^5 & -2.7704 \times 10^5 & -2.4478 \times 10^6 \end{bmatrix} \tag{7.13}$$

Although the matrix to transform from Holmdel building to geodetic coordinates can also be calculated, we opted not to do so since the matrix would not be used in our application. This is because the computation overhead to convert all the points in the DXF coordinates into geodetic coordinates would be too high. However, another transformation needed to be derived as well since the architectural files of the Holmdel facilities was stored in two different formats. In the format used above,

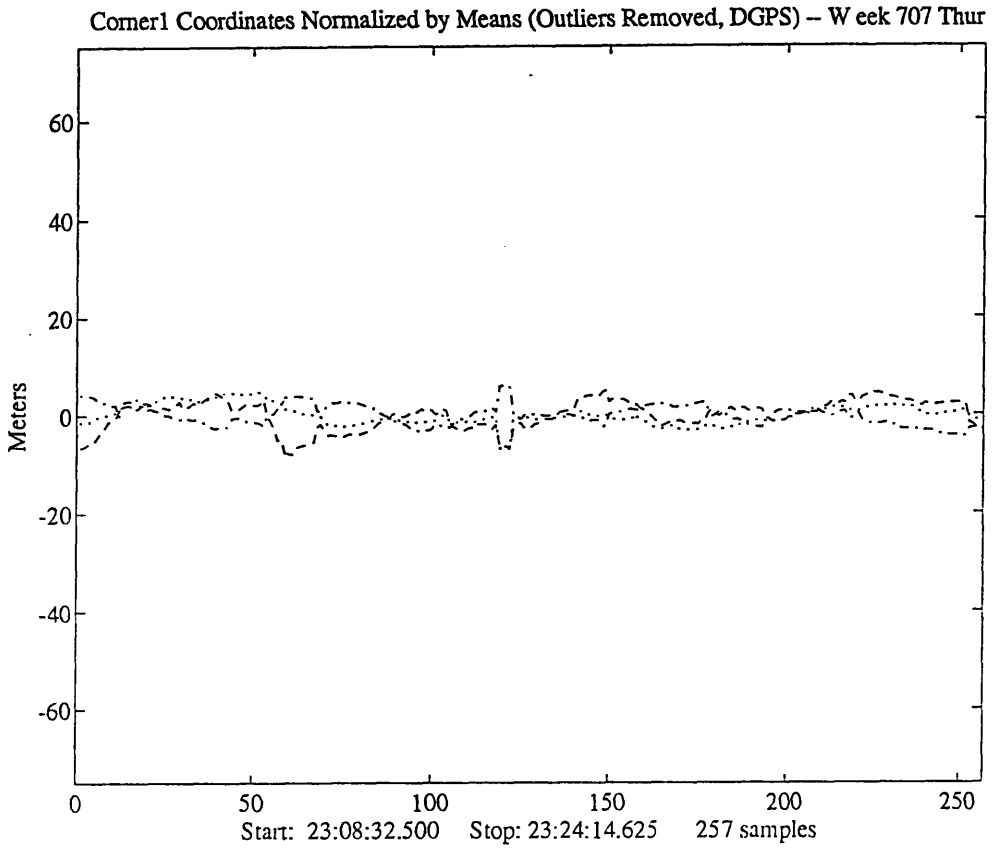


Figure 7-14: Data Collected in Corner 1 of Holmdel Building

Corner	Geodetic		Building Plans	
	Longitude (degrees)	Latitude (degrees)	X (feet)	Y (feet)
Corner 1	-74.1685	40.3663	-11321.777	-12240.240
Corner 2	-74.1658	40.3646	-12331.735	-12240.365
Corner 3	-74.1665	40.3638	-12331.652	-11898.553
Corner 4	-74.1693	40.3656	-11321.902	-11898.532

Table 7.1: Coordinates of the Four Corners of the Holmdel Building

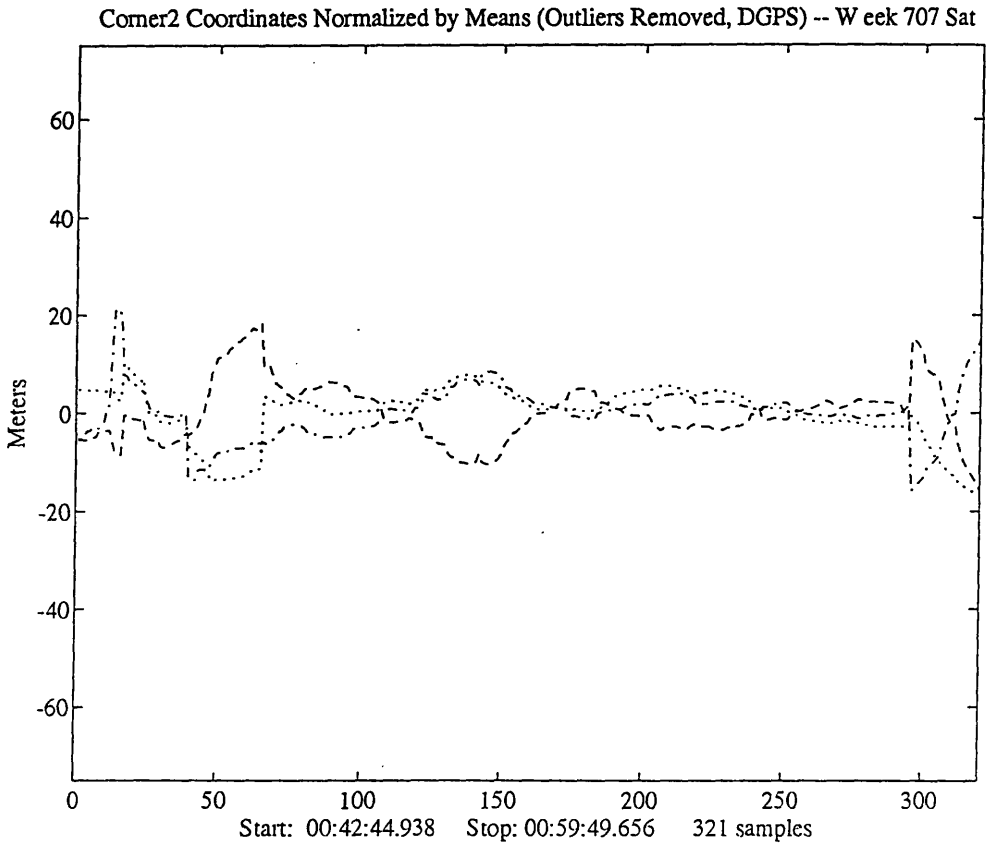


Figure 7-15: Data Collected in Corner 2 of Holmdel Building

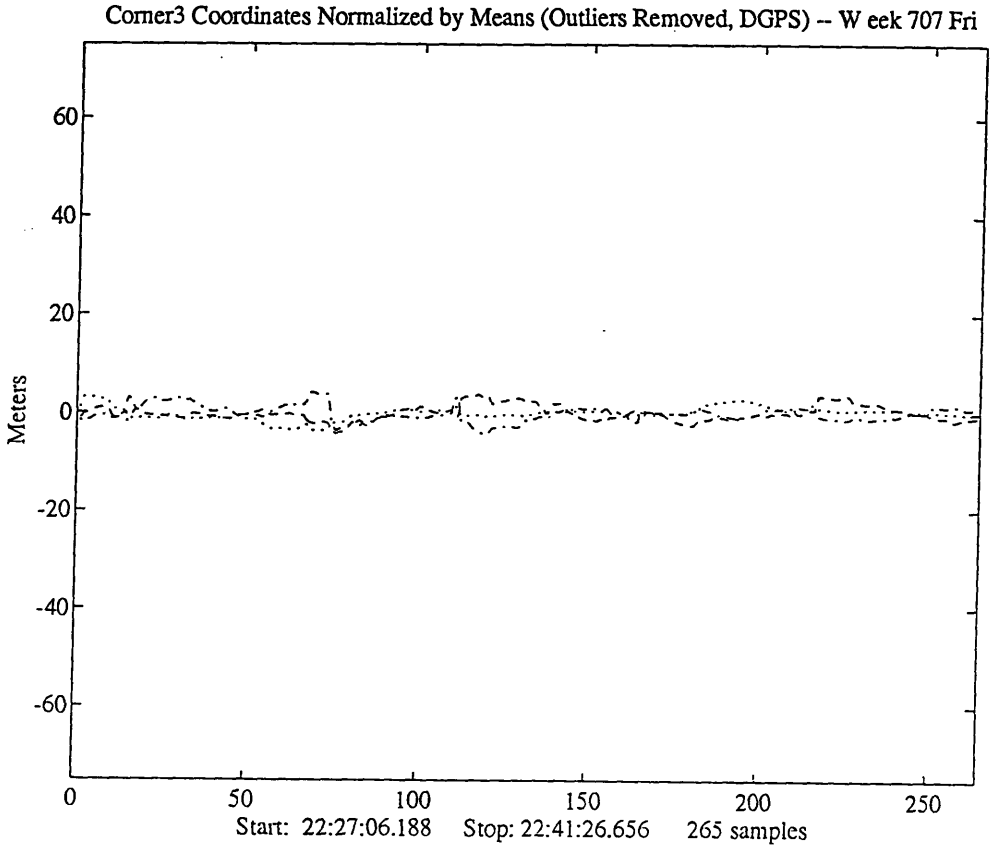


Figure 7-16: Data Collected in Corner 3 of Holmdel Building

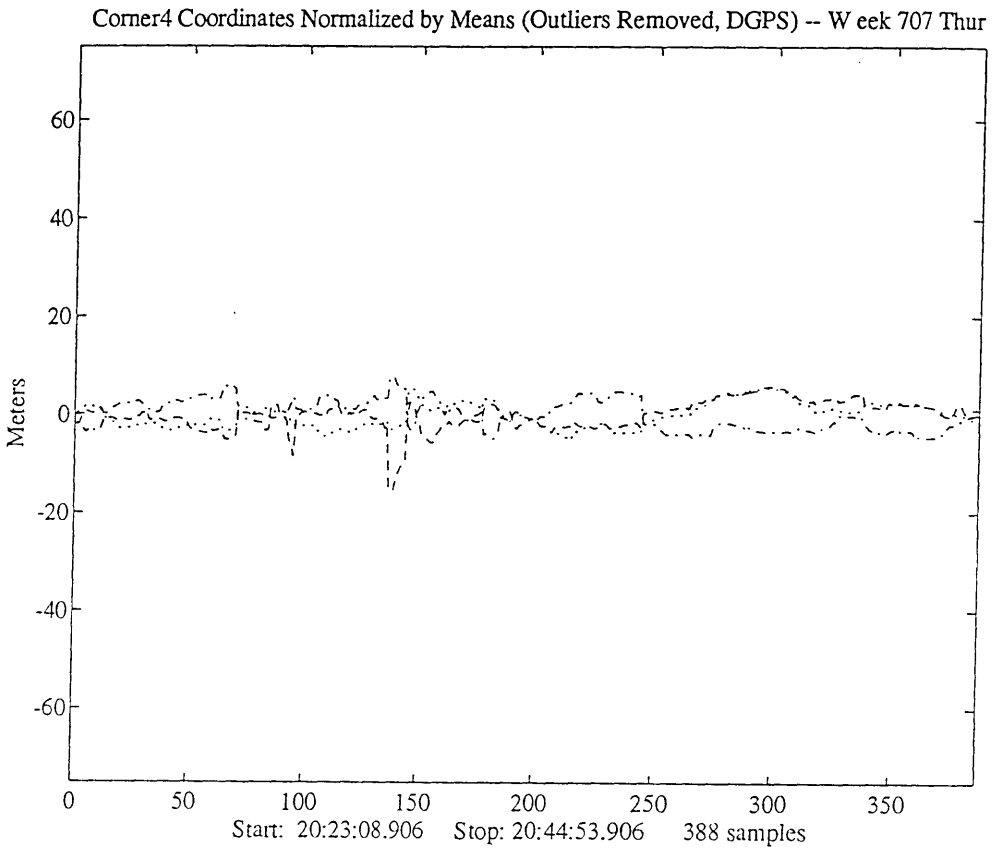


Figure 7-17: Data Collected in Corner 4 of Holmdel Building

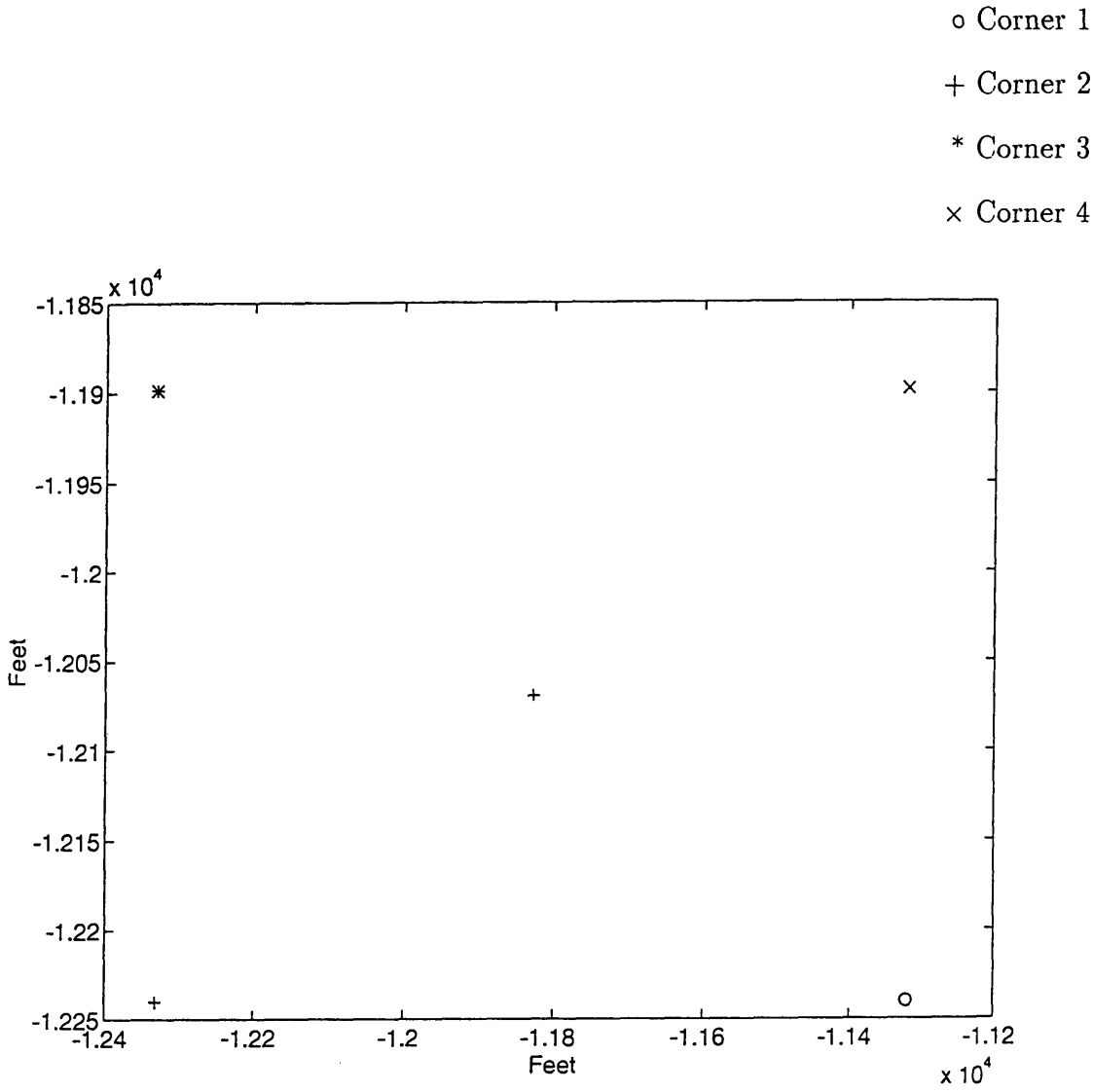


Figure 7-18: Measured Corners in Holmdel Building Coordinate System

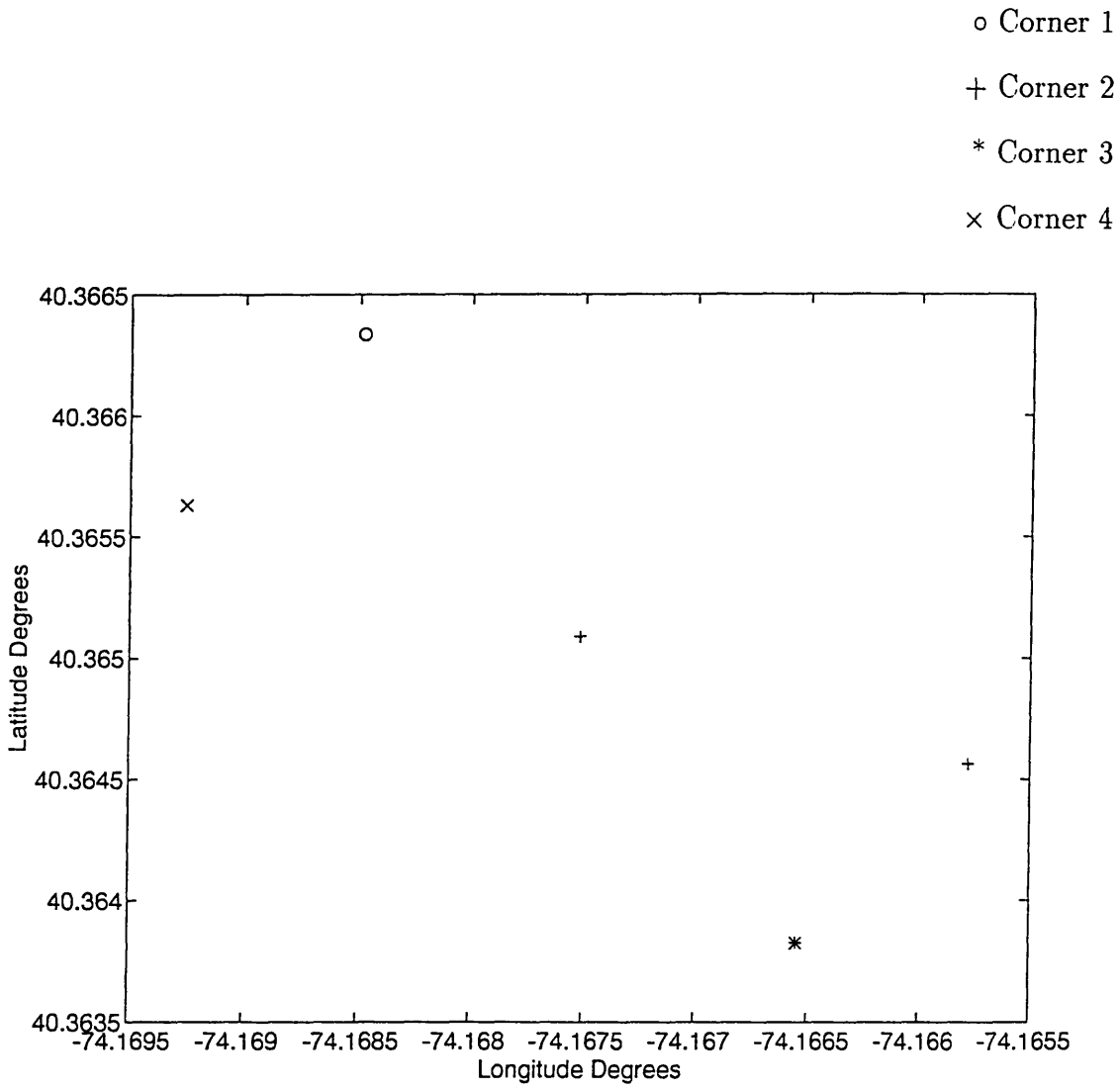


Figure 7-19: Measured Corners in Geodetic Coordinate System

the Holmdel building lies parallel to the Holmdel building coordinate system's axes as shown in Figure 7-18. In the alternate representation used to display the entire site's infrastructure information, the coordinate axes are rotated 232.525 degrees clockwise from the x-axes of the original coordinate system. Therefore, to transform coordinates in the Holmdel building coordinate system into the Holmdel site coordinate system, one needs to translate the building coordinates origin to [0, 0], rotate by an angle $\theta = 232.525$ and then retranslate back to the origin. This can be represented as a series of multiplications by three transformation matrices:

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{7.14}$$

If we represent the product of the three transformation matrices to be the single matrix H, the transformation matrix to convert directly from geodetic coordinates to Holmdel site coordinate system can be represented by the 3×3 matrix K as:

$$K = H \cdot \begin{bmatrix} T_{lonx} & T_{latx} & C_x \\ T_{lony} & T_{laty} & C_y \\ 0 & 0 & 1 \end{bmatrix} \tag{7.15}$$

The result of applying K on the measured longitude and latitude coordinates of the four corners is shown in Figure 7-20.

Using these transformation matrices, we were able to successfully navigate around the Holmdel facilities. Just as `mapper` was able to zoom in and increase the resolution of a particular area on the map, so can `Xdxfdraw`. Unfortunately, `Xdxfdraw`, besides a not very user friendly interface, does not have the ability to zoom in and out more than once. However, probably the most important difference between `mapper` and `Xdxfdraw` is the inability of `Xdxfdraw` to track the user once she moves out of the viewing region nor when she alters her orientation angles. This is due to the very high overhead of parsing the DXF files. Upon a change of rendering scale, the entire DXF file is reparsed. Because of the latency from the time a user zooms in and the program finishes drawing the change, the socket communicating with the navigation server is shut down and reactivated once the program has finished parsing the files. This problem can easily be alleviated by more intelligent data accessing routines. However, since the priority of this program was to demonstrate a practical application of our enhanced reality system, that is to be able to register points on architectural plans with a common geodetic coordinate system, it was sufficient for our purposes. Figure 7-21 shows the same differentially corrected data, already displayed in Figure 6-21, but this time superimposed on AutoCAD files of the Holmdel site. Figure 7-22 displays GPS data superimposed on the Holmdel building.

7.4.2 3D AutoCAD

In addition to the previously described 2D visualization of AutoCAD information, we also developed a program that worked in 3D. This program utilizes the SPOTlib

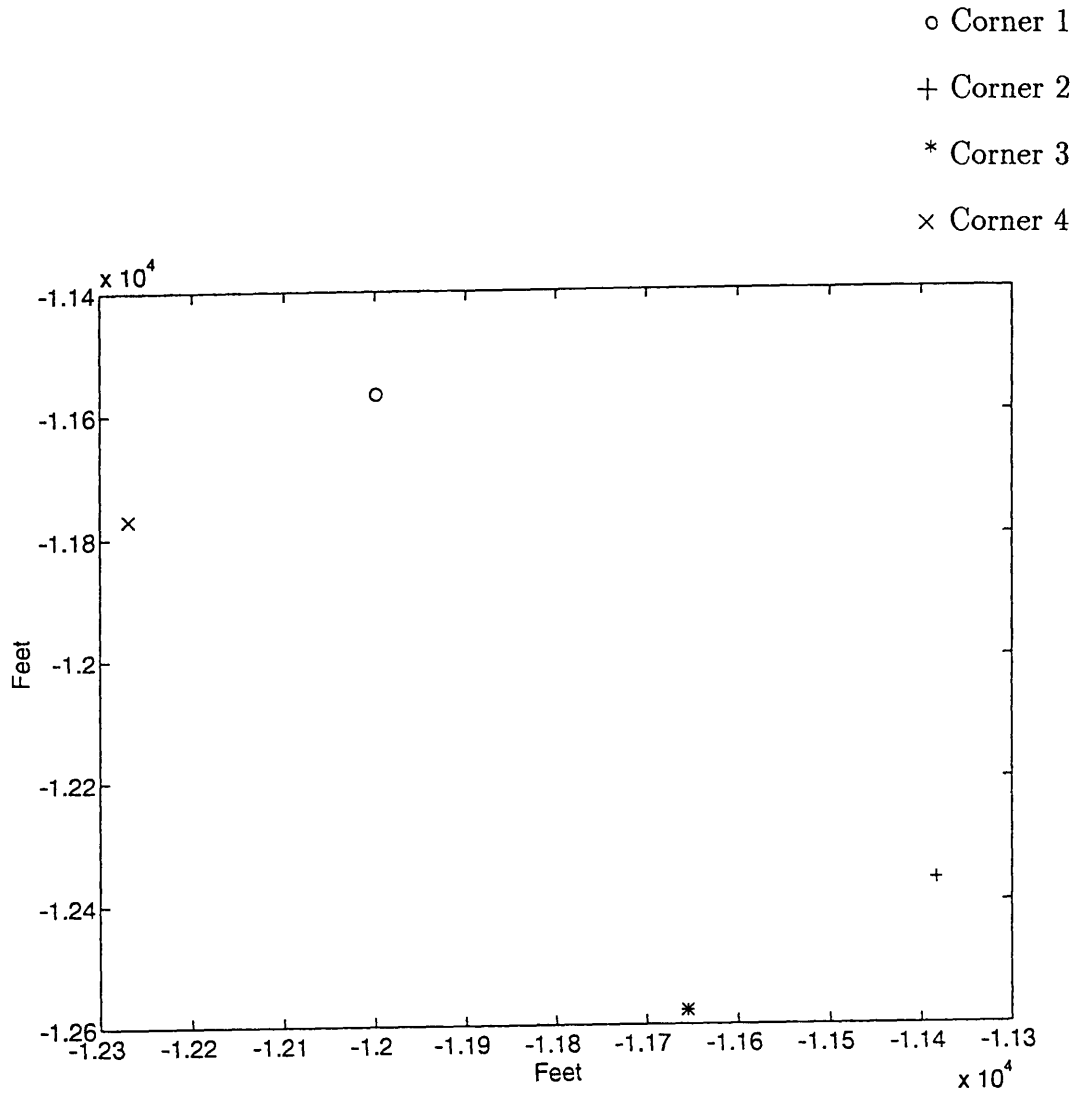


Figure 7-20: Measured Corners in Geodetic Coordinate System

Figure 7-21: GPS Data Superimposed on the Holmdel Site Plans

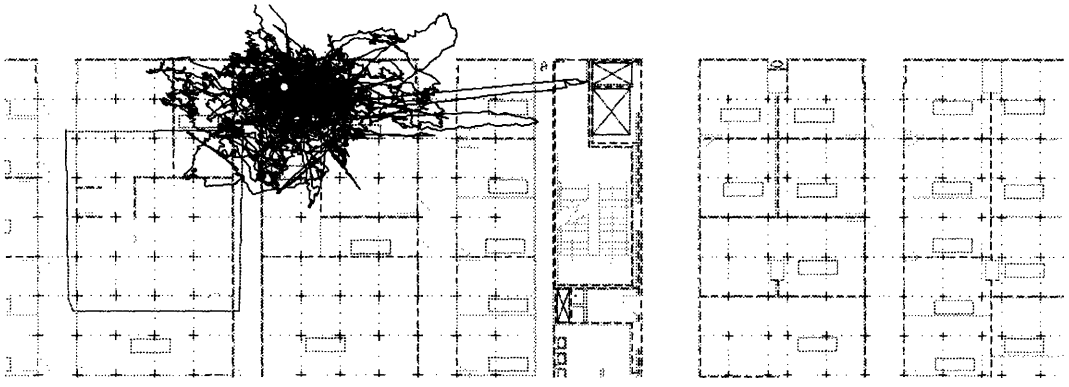


Figure 7-22: GPS Data Superimposed on the Holmdel Building Plans

library routines which were discussed in Chapter 5 in order to do the 3D rendering. We added some interfacing routines to SPOTlib and renamed the complete library `xartlib`. A 3D model of an AutoCAD file of a building may be generated by projecting wall layers to the given height of the walls in a building.

A common demonstration in the Machine Perception Department at AT&T Bell Laboratories, created by Michael Potmesil, involves the display of a virtual lab generated by the Pixel Machine. We transformed the commands suitable for the Pixel Machine to that which were suitable for `xartlib`. It should be noted, however, that we did not have to change any of the models for the lab since SPOTlib's input format is very similar to SIMlib's format for the Pixel Machine.[6] In addition, the input consisted of a list of polygons that needed to be rendered and therefore the input data did not need to be modified since the same objects will still need to be rendered using either library.

Combining the lab model used in Potmesil's demonstration with the AutoCAD files, we generated the view that one would see while looking down through the roof into the lab area. Ideally, we would want to be able to take our system on the roof of the Holmdel building and be able to rotate the unit and have the underlying image change accordingly. Although this is all within the capabilities of our system, unfortunately our software is not implemented efficiently enough to allow for fast real-time display of 3D graphics. Not only is there the overhead from `xartlib`, we also have to deal with the start-up delay resulting from the parsing of the AutoCAD data files. There are problems that can be solved and can be possible additions in future implementations of our system. In the next chapter we will discuss other areas where our system can be improved upon.

[6] In fact, one of the developers of SPOTlib worked on the development the Pixel Machine.

Chapter 8

Conclusion

The first seven chapters have described the development and implementation of an enhanced-reality system. Some potential applications of such a system have been demonstrated. One must remember that the system, we have developed, is an *enhanced* reality system and not a *virtual* reality system. Although the difference between the two are slight, they are significant. An enhanced-reality system does not attempt to replace reality for the user, as virtual reality does. Instead, it provides the user with more information than is readily available to her from her senses. For example, `mapper` and `Xdxfdraw` provide the user the ability to know where she is with respect to a given area through a mixture of visual and audio feedback. `el-az-gps` can display satellites that are currently overhead and visible to the GPS receiver but are not visible to the human eye. `sun-moon-gps` can show us the position of the sun and moon even when they are below the horizon.

Naturally, the aforementioned example programs are just glimpses of a whole plethora of potential uses of a fully working enhanced reality system. "Fully working" is the key to the future popularity of the system. We have laid the basic, but essential, foundations for future research and development in the area of enhanced-reality. There exist many areas for improvements in our fledgling system. One aspect that can be immediately improved is to custom-design the electronics used for interfacing the GPS receiver with the tilt sensors and compasses, thereby reducing the bulk and power consumption. However, Trimble Navigation, Ltd. is now offering GPS receivers that can communicate using the standard PCMCIA interface. In addition, Tadpole Technologies has also announced the release of a SPARCbook 3 which will have a PCMCIA slot and two RS-232 compatible serial ports, thereby allowing us to simplify our hardware which will in turn lead to a higher throughput.

Future models of our system should be hands free consisting of a head-mounted display, a small head-mounted tilt sensor and compass unit, and a small wearable, but powerful computing platform that has a low power consumption. There already exist head-mounted displays such as the Private-Eye from Reddy Information Systems and Virtual Vision Sport from Virtual Vision. A head-mounted tilt sensor unit has also already been built consisting of only 3 sensors due to the expected reduced freedom in movement of a head-mounted system. The azimuth orientation angle is tracked by a single compass.

Our desire for a small compact yet powerful computing engine is the most difficult component to obtain. A primary limitation of our system was that it is computationally power hungry, as most graphics-intense applications are. For example, with the power of an SGI Reality Engine, the application programs can be more sophisticated with a much more realistic graphical interface. Unfortunately, SGI does not have any immediate plans to make a portable Indigo, let alone a portable Onyx. Nonetheless, the trend in manufacturing of mobile computers already consists of increased horsepower, less power consumption and improved communication with external devices. With this overall trend in mind, it is a matter of time before we will get the computing power we desire.

Another area where our system was limited by technology was in the area of wireless communications. In our examples, we were limited to databases stored on our portable computer. Perhaps we can improve the range of data that we can display by retrieving information from a CD-ROM player or pre-loading information from an electronic bulletin board service or from the Home Shopping Network through Interactive TV. Although this will give us access to more information, the user will still be limited to data on CD-ROM's that she had thought a priori to her trip to bring with her. The necessity of methodically planning one's movements removes spontaneity from the user's actions. The answer can lie potentially in remotely accessing data on a call by need basis. With improved cellular communication, the databases can possibly be retrieved over a cellular line in real-time.

The last two upgrades depend on the improving technology in other areas of the computing industry. While waiting for the hardware to improve, on the software side many improvements can also be made. One particular region is in the generation of 3D graphics. Although `sculpt` and `xartlib` provided some means of rendering objects in 3D, their performance in displaying real areas in real-time was not satisfactory. A possible alternative drawing package would be OpenGL. Another problem in software is the manner in which data was being acquired. For the AutoCAD files, the structures could be obtained and stored in a much more orderly fashion such that if only a small region of the entire database was being displayed, only visible points will need to be processed. Improvements in the database handling routines would then also improve our rendering time since the majority of the computation time is absorbed searching and retrieving the necessary points for rendering rather than in the rendering itself (excluding programs using `xartlib`).

As one can see, there is much that remains to be done to make enhanced-reality systems a potential consumer product. Our thesis demonstrated that such a system is indeed viable with potential educational, in addition to practical, applications. The foundations set, enhanced reality waits for technology to make it a common reality.

Appendix A

Dithering Array

The following is the threshold array used in the dithering of the images produced by `SPOTlib`:

$$\begin{bmatrix}
1 & 236 & 59 & 220 & 15 & 232 & 29 & 216 & 3 & 234 & 57 & 218 & 13 & 230 & 53 & 214 \\
130 & 65 & 188 & 124 & 144 & 79 & 184 & 119 & 132 & 67 & 186 & 121 & 142 & 77 & 182 & 117 \\
33 & 194 & 17 & 152 & 47 & 208 & 31 & 248 & 35 & 196 & 19 & 250 & 45 & 206 & 55 & 246 \\
162 & 97 & 146 & 81 & 176 & 111 & 160 & 95 & 164 & 99 & 148 & 83 & 174 & 109 & 158 & 93 \\
9 & 226 & 49 & 210 & 5 & 240 & 63 & 224 & 11 & 228 & 51 & 212 & 7 & 238 & 61 & 222 \\
138 & 73 & 178 & 113 & 134 & 69 & 192 & 127 & 140 & 75 & 180 & 115 & 136 & 71 & 190 & 125 \\
41 & 202 & 25 & 242 & 37 & 198 & 21 & 255 & 43 & 204 & 27 & 244 & 39 & 200 & 23 & 254 \\
170 & 105 & 154 & 89 & 166 & 101 & 150 & 85 & 172 & 107 & 156 & 91 & 168 & 103 & 152 & 87 \\
3 & 234 & 57 & 218 & 13 & 230 & 53 & 214 & 1 & 236 & 59 & 220 & 15 & 232 & 29 & 216 \\
132 & 67 & 186 & 121 & 142 & 77 & 182 & 117 & 130 & 65 & 188 & 124 & 144 & 79 & 184 & 119 \\
35 & 196 & 19 & 250 & 45 & 206 & 55 & 246 & 33 & 194 & 17 & 152 & 47 & 208 & 31 & 248 \\
164 & 99 & 148 & 83 & 174 & 109 & 158 & 93 & 162 & 97 & 146 & 81 & 176 & 111 & 160 & 95 \\
11 & 228 & 51 & 212 & 7 & 238 & 61 & 222 & 9 & 226 & 49 & 210 & 5 & 240 & 63 & 224 \\
140 & 75 & 180 & 115 & 136 & 71 & 190 & 125 & 138 & 73 & 178 & 113 & 134 & 69 & 192 & 127 \\
43 & 204 & 27 & 244 & 39 & 200 & 23 & 254 & 41 & 202 & 25 & 242 & 37 & 198 & 21 & 255 \\
172 & 107 & 156 & 91 & 168 & 103 & 152 & 87 & 170 & 105 & 154 & 89 & 166 & 101 & 150 & 85
\end{bmatrix}$$

Bibliography

[1] Canadian GPS Associates, *Guide to GPS Positioning.* 2nd printing. New Brunswick, Canada: Canadian GPS Associates, 1987.

[2] Department of Defense, *World Geodetic System 1984 - Its Definition and Relationships with Local Geodetic Systems.* DMA TR 8350.2. Second Printing. Revised. Washington D.C., 1 March 1988.

[3] Duffett-Smith, P., *Practical astronomy with your calculator.* Second Edition. New York: Cambridge University Press, 1981.

[4] Farrar, J. G., "Electronic Digital Compass." U.S. Patent 4,918,824. October, 1988.

[5] Foley, J.D., van Dam, A., Feiner, S.K., and Hughes, J.F., *Computer Graphics: Principles and Practice.* Second Edition. New York: Addison-Wesley Publishing Company, 1990.

[6] `ftp://tycho.usno.navy.mil/pub/gps/gpsb1.txt`

[7] `ftp://tycho.usno.navy.mil/pub/gps/gpssy.txt`

[8] "The First Annual *GPS WORLD* Applications Contest: Results." *GPS World.* vol. 3. no. 5. pp. 21-39. May 1992.

[9] Getting, I. A., "The Global Positioning System." *IEEE Spectrum.* Decemer 1993.

[10] Goad, C. C., "Precise Positioning with the Global Positioning System: An Overview". Workshop 13, ASPRS/ACSM/RT 92 Convention, Washington, D.C., 10 August 1992.

[11] Heckbert, P. S., *Color Image Quantization for Frame Buffer Display.* Cambridge. 1980.

[12] Institute of Navigation, *Global Positioning System Papers*, vol. I-III. Washington, D.C.: Institute of Navigation, 1980-1986.

[13] Interface Control Document, *NAVSTAR GPS Interface Control Document.* ICD-GPS-200. 3 July 1991. Navstar GPS Space Segment/Navigation User Interfaces. Public Release Version.

[14] Kalafus, R. M., Van Dierendonck, A. J., and Pealer, N. A., "Special Committee 104 Recommendations for Differential GPS Service." *Global Positioning System Papers.* vol III. pp 101-116. Washington D.C.: Institute of Navigation, 1986.

[15] Leick, A., *GPS Satellite Surveying.* New York: John Wiley & Sons, 1990.

[16] Langley, R. B., "Basic Geodesy for GPS". *GPS World.* pp. 44-49. February 1992.

[17] Lanier, J., "A Brave New World: Virtual Reality Today." *Virtual Reality Special Report*, pp. 11-17, July 1992.

[18] Lim, J. S., *Two-Dimensional Signal and Image Processing.* Englewood Cliffs: Prentice Hall. 1990.

[19] The Math Works, Inc., *MATLAB User's Guide for UNIX Workstations.* Natick: The Math Works Inc., 1992.

[20] McNamara, J. E., *Technical Aspects of Data Communication.* 2nd Edition. Bedford,MA: Digitttal Equipment Corporation, 1982.

[21] Messmer, E., "API to give mobile devices access to positioning service." *Network World.* vol 10. issue 30. p. 83. 26 July 1993.

[22] Milbert, D. G., "Computing GPS Derived Orthometric Heights with the GEOID90 Geoid Height Model." *Technical Papers of the 1991 ASPRS Fall Convention.* pp. A46-55. Atlanta, Georgia. 1991.

[23] Milbert, D. G., "GEOID90: A High-Resolution Geoid for the United States." *EOS, Transactions, American Geophysical Union.* vol. 72. no. 49. December 3, 1991.

[24] Milbert, D. G., "GPS and GEOID90 – The New Level Rod". *GPS World.* pp. 38-43. February 1992.

[25] Milliken, R. J. and Zoller, C.J., "Principle of Operation of NAVSTAR and System Characteristics." *Global Positioning System Papers.* vol I. pp 3-14. Washington D.C.: Institute of Navigation, 1980.

[26] Morning Star Technologies, Inc., *Morning Star PPP – Version 1.4β.* Revision 1.134. Columbus,Ohio: Morning Star Technologies, Inc., June 1993.

[27] NAVSTAR GPS User Equipment. Public Release Version, Los Angeles Air Force Base: NAVSTAR-GPS Joint Program Office, February 1991.

[28] Naylor, Brucen "Interactive Solid Geometry Via Partitioning Trees." *Graphics Interface '92.* pp. 11-18, May 1992.

[29] "Newsfront." *GPS World.* vol 4. no 1. p. 20. January 1993.

[30] Nye, Adrian, *Xlib Programming Manual.* Vol 1. Sebastopol: O'Reilly & Associates, Inc., 1992.

[31] Nye, Adrian, *Xlib Reference Manual.* Vol 2. Sebastopol: O'Reilly & Associates, Inc., 1992.

[32] Ohio State University, "The GPS/IMAGING/GIS Project – Application of the Global Positioning System for Transportation Planning: A Multi-State Project". 1 December 1991.

[33] Polhemus Kaiser Aerospace & Electronics Company, *3SPACE Digitizer/Tracker User's Manual.* July 1991.

[34] Radio Technical Commission for Maritime Services, *RTCM Recommended Standard for Differential NAVSTAR GPS Service. Version 2.0.* RTCM Special Committee No. 104. RTCM Paper 134-89/SC 104-68. 1 January 1990.

[35] Siewiorek, D. P., "Wearable Infostations," *Electronic Engineering Times*, March 8, 1993.

[36] Spilker, J.J., Jr., "GPS Signal Structure and Performance Characteristics." *Global Positioning System Papers.* vol I. pp 29-54. Washington D.C.: Institute of Navigation, 1980.

[37] Spotlight Graphics, Inc., *SPOTlib User Manual.* 1993.

[38] Stern, H., *Managing NFS and NIS.* Sebastopol: O'Reilly & Associates, Inc., 1991.

[39] Trimble Navigation, Inc., "Guide to the Next Utility." 1992.

[40] Stevens, W. R., *UNIX Network Programming.* New Jersey: PTR Prentice Hall, 1990.

[41] Ulichney, R., *Digital Halftoning.* Cambridge: The MIT Press. 1990.

[42] U.S. Department of Commerce National Oceanic and Atmospheric Administration, *North American Datum of 1983.* Rockville: National Geodetic Survey. NOAA Professional Paper NOS 2. 1989.

[43] U.S. Department of Transportation, *Global Positioning System Information Center Users Manual.* Washington D.C.: United States Coast Guard, September 1992.

[44] United States Geological Survey, "Sample Runs for GEOMAG Program." National Geomagnetic Informaton Center. 1992.

[45] Van Dierendonck, A.J., Russell, S.S., Kopitzke, E.R., and Birnbaum, M., "The GPS Navigation Message." *Global Positioning System Papers.* vol I. pp 55-73. Washington D.C.: Institute of Navigation, 1980.

[46] Sheridan, T.B. and Zeltzer, D., "Virtual Reality Check." *Technology Review.* vol. 96, no. 7, pp. 20-28, October 1993.

[47] Thibault, W. and Naylor, B., "Set Operations on Polyhedra Using Binary Space Partitioning Trees." *Computer Graphics.* vol. 21 no. 4. pp. 153-162. July 1987.