High Precision Short-Baseline Pointing System Using GPS Interferometry,

A Thesis Presented to
The Faculty of the Russ College of Engineering and Technology

Ohio University

In Partial Fulfillment
of the Requirements for the Degree

Master of Science

by

Lap Ho

March, 1995
Acknowledgements

The research presented in this thesis was funded by the Federal Aviation Administration (FAA) through Research Grant 92-G-023, and was also jointly funded by the FAA and the National Aeronautics and Space Administration (NASA) under Grant NGR-36-009-017.

Dr. Michael Braasch, Assistant Professor of Electrical and Computer Engineering, is thanked for being the chairman of the thesis committee. His advice, recommendations, and proofreading were important factors for the success of this thesis.

Dr. Jeff Dill, Associate Professor of Electrical and Computer Engineering, Dr. Trent Skidmore, Research Scientist with the Avionics Engineering Center, and Dr. Tiao J. Chang, Professor of Civil Engineering, are thanked for their help in reviewing this document and serving as members of the thesis committee.

The author would like heartfully to thank Dr. Frank van Graas, Associate Professor of Electrical and Computer Engineering for his unceasing encouragement, support and help throughout this research.

The author would also like to note his profound appreciation to his parents for their love, support, understanding, encouragement, and constant prayer throughout all these years.

Most of all, thanks to our heavenly Father for his eternal love and blessings.
# Tables of Contents

1. Introduction ........................................................................... 1

2. The Global Positioning System ........................................... 3
   2.1 Basic Concept .................................................................. 3
   2.2 Signal Structure .............................................................. 5
   2.3 Ranging Measurement ....................................................... 7
   2.4 Ephemerides ................................................................... 7

3. GPS Interferometers ............................................................. 10
   3.1 Introduction .................................................................... 10
   3.2 Point positioning ............................................................ 12
   3.3 Carrier Phase Measurement ............................................. 15
   3.4 Single-Difference Formation .......................................... 19
   3.5 Double-Difference Formation ......................................... 22
   3.6 Short-Baseline Implementation ...................................... 24

4. Implementation ...................................................................... 32
   4.1 System Set Up ............................................................... 32
   4.2 Software Implementation ................................................. 33
   4.3 Data Collection ............................................................... 35

5. Test Result and Data Analysis .............................................. 38
   5.1 Zero baseline results ....................................................... 38
   5.2 Length, Azimuth Angle, and Elevation Angle .................. 46
   5.3 Results ........................................................................... 48

6. Conclusion ............................................................................ 58

7. Appendix ............................................................................... 59
   7.1 Appendix 1 ..................................................................... 59
   7.2 Appendix 2 ..................................................................... 106

8. References ............................................................................ 132
List of Figures

Figure 1: Biphase Modulation. .............................................. 6
Figure 2: GPS interferometer ........................................... 10
Figure 3: Integrated Doppler range and Pseudorange. .................. 17
Figure 4: Interferometer for double-difference. .......................... 22
Figure 5: Short baseline interferometer with incident waves. ......... 24
Figure 6: Short baseline interferometer with only fractional part of incident signal. 25
Figure 7: Interferometer with baseline length less than 1 wavelength. .... 26
Figure 8: Interferometer with 7 satellites. ............................. 27
Figure 9: Interferometer with four satellites. .......................... 29
Figure 11: Set up interrupt services. ...................................... 34
Figure 12: Flow chart of baseline solution. ............................. 34
Figure 13: Implemented interferometer. .................................. 36
Figure 14: Complete system set up. ...................................... 36
Figure 15: Pseudorange double-difference residual using satellite 20 and 24. . . . 39
Figure 16: Pseudorange double-difference residuals using satellite 20 and 4. . . . 40
Figure 17: Pseudorange double-difference residuals using satellite 20 and 5. . . . 41
Figure 18: Integrated Doppler double-difference residuals using satellite 20 and 24. 42
Figure 19: Integrated Doppler double-difference residuals using satellite 20 and 4. 43
Figure 20: Integrated Doppler double-difference residuals using satellite 20 and 5. 44
Figure 21: Vector in East North Up coordinates ........................ 47
Figure 22: Length of short baseline vector points at azimuth angle North 60 and elevation angle 0 degree. ................................................................. 49

Figure 23: Azimuth angle for short baseline points at azimuth angle North 60 and elevation angle 0 degree. ................................................................. 50

Figure 24: Elevation angle of short baseline points at azimuth angle North 60 and elevation angle 0 degree. ................................................................. 51

Figure 25: Length of short baseline points at azimuth angle North -30 and elevation angle 0 degree. ................................................................. 52

Figure 26: Azimuth angle of short baseline points at azimuth angle North -30 and elevation angle 0 degree. ................................................................. 53

Figure 27: Elevation angle of short baseline vector points at azimuth angle North -30 and elevation angle 0 degree. ................................................................. 54
List of Tables

Table 1: Ephemerides ................................. 8
Table 2: Standard deviation of zero baseline double-difference using pseudorange and integrated Doppler. .............................................. 45
Table 3: Statistical results for baseline. ......................... 55
1 Introduction

The NAVSTAR Global Positioning System (GPS) is a space-based radionavigation aid. Developed, owned and operated by the United States Department of Defense (DOD), its primary purpose is the determination of platform position and velocity as well as the dissemination of time. More recently, however, interferometric principles have been applied to GPS for the real-time determination of attitude and heading.

The central theme in GPS interferometry is the determination of position of an antenna/receiver relative to another antenna/receiver. This may also be viewed as solving for the three-dimensional vector from one antenna to the other. This vector is sometimes referred to as a baseline vector or simple as the baseline.

If the two antennas are mounted on, for example, an aircraft fuselage, knowledge of the baseline vector can be exploited to determine aircraft pitch and heading. If additional antennas are mounted on the wings, the resulting baselines can be used to determine roll angle as well. This thesis will focus exclusively on the determination of a single baseline between a pair of antennas. The antennas, receivers, and computational hardware and software required to perform this task are collectively referred to as a pointing system.

In general, GPS interferometric systems require a long (i.e., several minutes) initiation procedure prior to providing high precision vector coordinates. Instantaneous baseline determination can be achieved if the length of the baseline is less than one carrier wavelength (approximately 19 centimeters for GPS L1). The price paid is reduced angular precision. It has been suggested [van Graas, 1993], however, that a short baseline
pointing system could be used as a rough initialization for a longer baseline system. It is envisioned that the overall system could be initialized in a fraction of the time required by current state-of-the-art techniques.

The detailed discussion of the principles of using interferometry to determine the baseline vector will be described in later a chapter. In chapter 2, the Global Positioning System will be introduced. In chapter 3, the details about the interferometer will be illustrated. Chapter 4 describes the implementation of the pointing system. The results are discussed in chapter 5. Finally, conclusions and recommendations for future work are given in chapter 6.
2 The Global Positioning System

2.1 Basic Concept

In 1973, the Joint Program Office (JPO) was directed by the U.S. Department of Defense (DOD) to establish and develop a spaceborne positioning system [B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, 1993]. The present NAVigation System with Timing And Ranging (NAVSTAR) Global Positioning System (GPS) is the result of this initial directive.

GPS was designed as a world wide, all-weather, space-based navigation and position determination system. In addition to position determination, GPS provides near instantaneous velocity and precise coordination of time. It was originally conceived as a ranging system from known positions of satellites in space to unknown positions in sea, land, in air and even in space. The basic concept of point position determination using GPS relies on the solution of simultaneous sets of equations. In two dimensions, a range measurement from a transmitter at a known location provides a circular line-of-position. Position determination requires multiple measurements and thus determination of the intersection of multiple circles. In three dimensions, the concept of point positioning is the same except that one is finding the intersection of spheres of position. GPS employs the same concept in that the space vehicles (satellites) are the transmitters at known three dimensional coordinates. Conceptually then, any point in the three dimensional space covered by the satellite signals can be determined precisely if the ranges between the point
and at least three satellites are provided. The satellite position can be obtained from the ephemeris data transmitted by the space vehicles themselves. The computation of satellite position using ephemeris data will be discussed later. Range determination is based on the relationship of the speed of the signal and the length of time the signal travels. Range is simply equal to the multiplication of the speed of the signal (speed of light) by the signal travel time. For effectively providing the signal travel time, the satellite signal is continually marked with its own transmission time. Thus, the signal transit period can be measured with a synchronized receiver. However, the clocks of receivers are never perfectly synchronized with the clocks of satellites, therefore the range calculated is not the true range. This measurement is called the pseudorange since it is the true range plus or minus a small distance resulting from the receiver clock bias. In practice, then, four pseudorange measurements are used to solve for three dimensional position and clock bias. As will be discussed later, a biased range measurement can also be derived from the Doppler shift of the signal.

The GPS system consists of three segments: the space segment, control segment, and user segment. The space segment includes satellites. There is a total of 24 GPS satellites that orbit at an altitude of 20,200 km and have an inclination of 55 degrees with respect to the equator [Börje Forssell, 1991]. Each satellite is a radio transceiver station. It broadcasts a message which contains information allowing the user to determine the position of satellites at any arbitrary instant.

The control segment consists of a master control station, worldwide monitor stations, and ground control station. The control segment is the brain of the entire system.
The main task of the control segment is to keep track of satellite orbits, compute and correct satellite clock drifting, and upload messages to the satellites. The master control station is responsible for satellite control, system operation, collecting tracking data from the ground monitoring stations and calculating satellite orbit and clock parameters. The results are then transmitted to one of three ground stations. The ground station will eventually upload the data to the satellite. There is a total of five monitor stations spread over the world. Each of these stations is equipped with a precise time standard and satellite signal receiver which measures pseudoranges to all satellites in view at a rate of 1.5 seconds per sample. All measurement data are sent from the monitor stations to the master control station for calculation of satellite clock and orbital parameters. This information is then uploaded to the satellites. The final segment is the user segment. This segment consist of the user receivers mounted on the various platforms (airplanes, boats, ground vehicles, etc).

2.2 Signal Structure

GPS operates on two carrier signals in the L-band (L-band signals range from 950 MHz to 1600 MHz). These two L-band signals are defined as link 1 and link 2 which are commonly referred to as L1 and L2. The fundamental frequency \( f_0 \) of L1 and L2 is 10.23 MHz. L1 and L2 are generated by integer multiplications of the fundamental frequency. The frequency of L1 is 1575.42 MHz, \( f_0 \) times 154, with an approximate wavelength of 19.0 cm. The frequency of L2 is 1227.6 MHz, \( f_0 \) times 120, with an
approximate wavelength of 24.4 cm.

The carrier frequencies L1 and L2 are modulated by codes to provide satellite signal time-of-transmission to the receiver. The codes are digital signals with states of +1 and -1. In the transmission data link, the signal is represented in binary form with 1 and 0. Biphase modulation (Figure 1) is employed in the analog transmission. It is transmitted by performing a 180 degree phase shift in the carrier signal whenever a change in the state occurs.

![Biphase Modulation](image)

Figure 1: Biphase Modulation.

There are two kinds of codes. The coarse / acquisition code (C/A-code) has a frequency of 1.023 Mbits/sec which is $f_0/10$. The precision code (P-code) has a frequency of 10.23 Mbits/sec. The frequency of the P-code is equal to $f_0$. L2 is modulated with the P-code only. L1 is modulated with both the P-code and the C/A code in such a way that the C/A code is offset 90 degrees the from the P-code. In addition to the P-code and C/A code, the carrier is modulated with navigation (NAV) data which contains satellite orbit
information (ephemeris). The signals can be expressed mathematically as in equation (1) [Spilker, 1980]:

\[
\begin{align*}
L1 &= a_P(t)D(t)\cos(2\pi f_1 t) + a_{C/A}(t)D(t)\sin(2\pi f_1 t) \\
L2 &= a_P(t)D(t)\cos(2\pi f_2 t)
\end{align*}
\]  

(1)

Where \(a_{c}\cos(f_1 t)\) is the unmodulated carrier. \(P(t), C/A(t),\) and \(D(t)\) are the P-code, C/A code, and the navigation message, respectively.

2.3 Ranging Measurement

For positioning purposes, distance between the satellite and user must be measured. As mentioned before, pseudorange is calculated by multiplying the signal travel time by the speed of light. For a state-of-the-art receiver, pseudorange tracking noise is on the order of 1 meter [Braasch, M. and F. van Graas, 1991]. In some applications, tolerable deviations can only be in the centimeter range. As such, the pseudorange is not sufficient for the task. Measurement and accumulation of the signal Doppler shift provides a relatively noise free, yet biased, range measurement. More detail on the range measurement and processing are provided in the following chapter.

2.4 Ephemerides

As mentioned in the first section of this chapter, point positioning relies on range measurements from known satellite positions. Satellite position is calculated using the
ephemeris data (which is part of the broadcast NAVdata). Ephemeris data provide all information necessary to compute satellite position at any time within a window of data applicability.

The broadcast ephemeris data are generated by the control segment based on observations at the five monitor stations. It contains parameters of the orbit of the satellite and the age of the parameters. The parameters have been precisely computed to account for the external forces, such as the deviation of the earth from spherical shape and the lunar and solar gravitation forces which vary depending on the position of the satellites, influencing the satellite orbits [B. Hofmann-Wellenhof, 1993]. The accuracy of satellite orbits (calculated from the broadcast ephemeris) is approximately 5 meters assuming three upload of ephemerides per day. With a single daily update of ephemerides, the accuracy is about 10 meters. The Master Control Station is responsible for the computation of the ephemerides and the upload to the satellites. The parameters listed in Table 1 are broadcast every hour and should only be used within the next four hours.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AODE</td>
<td>Age of ephemerides data</td>
</tr>
<tr>
<td>(t_e)</td>
<td>Ephemerides reference epoch</td>
</tr>
<tr>
<td>(\sqrt{a}, e, Mo, \omega_o, i_o, l_o)</td>
<td>Keplerian parameters at (t_e)</td>
</tr>
</tbody>
</table>
The deviation of the accuracy of the satellite orbit using ephemerides is called ephemeris errors. The relationship between ephemeris errors and position errors are geometry-dependent. The maximum error occurs when the user antenna is in the line containing the correct and erroneous positions of the satellite. This is due to the fact that the user experiences the full effect of the orbital error vector rather than just a partial component.

<table>
<thead>
<tr>
<th>$\Delta \eta$</th>
<th>Mean motion difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>Rate of inclination angle</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>Rate of node's right ascension</td>
</tr>
<tr>
<td>Cuc, Cus</td>
<td>Correction coefficients (argument of perigee)</td>
</tr>
<tr>
<td>Cic, Cis</td>
<td>Correction coefficients (inclination)</td>
</tr>
<tr>
<td>Crs, Crs</td>
<td>Correction coefficients (geocentric distance)</td>
</tr>
</tbody>
</table>
3 GPS Interferometers

3.1 Introduction

In a given space, any two points can be joined together as a line. From one point to another, a vector can be formed and can be described in a 3-dimensional space in terms of Cartesian coordinates. Consider a simple GPS interferometer as depicted in figure 2.

![GPS interferometer diagram]

**Figure 2:** GPS interferometer

where:

A: Antenna A.

B: Antenna B.

$\overrightarrow{b}$: The vector pointing from antenna A to antenna B.

$\overrightarrow{u}$: The vector pointing from the interferometer to the satellite.
\( i_1 \): The incident signal from satellite to antenna A.

\( i_2 \): The incident signal from satellite to antenna B.

\( \Delta \Phi \): Range Difference.

In this GPS interferometer, the incident waves \( i_1 \) and \( i_2 \) come from the same satellite. Because the distance between the interferometer and the satellite is large (on the order of 20,000,000.0 meters), the incident wave can be considered as a plane wave. As such, the incident signals \( i_1 \) and \( i_2 \) are parallel to each other. Expressing the relationship between the variables of the interferometer in mathematical form (equation (2)):

\[
\vec{b} \cdot \hat{u} = \Delta \Phi
\]

(2)

In the equation above, \( \vec{b} \) is the unknown, \( \hat{u} \) (unit vector of \( \vec{u} \)) and \( \Delta \Phi \) are the known quantities. \( \vec{u} \) can be calculated from the satellite position (based on the broadcast ephemeris data) and a rough interferometer position obtained using the pseudorange measurements from the GPS receiver. \( \Delta \Phi \) can be obtained from the accumulated (integrated) carrier Doppler frequency from the GPS receiver. The theory and use of the accumulated Doppler frequency will be discussed later in this chapter.

In order to determine the vector \( \vec{u} \) exactly, positions of both antenna A and antenna B must be known. In addition, satellite position must be given. One question may be asked at this point. Which antenna should be used as a reference to form the vector \( \vec{u} \)? The answer is either antenna will suffice. Comparing the length of baseline vector \( \vec{b} \) (
0.19 cm or less for a short interferometer) and the length of vector $\vec{u}$ (about 20,000,000.0 m), the change of $\vec{u}$ is insignificant whether A or B is used. In this project, antenna A is used as the reference.

### 3.2 Point positioning

As mentioned in the previous section, pseudorange-based position determination is used in the process of calculating the vector, $\vec{u}$. Determining the point position requires the distance between the point and the satellite and the satellite position. Distance between the satellite and the point (in this project, the point is the antenna) is given by the pseudorange measurement made by the GPS receiver. As mentioned in the last chapter, the time of transmission is encoded on the GPS signal. The GPS receiver decodes this time and stores the time of reception. As such, the travel time of the signal is determined. The time of reception determined by the receiver is not absolutely accurate because of the clock offset of the receiver. In the point positioning process, the effect of this clock offset has to be taken into account.

Satellite position can be calculated using the broadcast ephemeris data. The process of calculating satellite position using ephemeris data is given in the appendix which covers the programs developed for this research. With the satellite position and distance between satellite and antenna given, their relationship to the point position may be expressed in equation (3).
where:

D: Distance from satellite to antenna.

\(x_s, y_s, z_s\): Satellite position.

\(x_p, y_p, z_p\): Antenna position.

There are three unknowns in equation (3): \(x_p, y_p\) and \(z_p\). Thus, three equations are needed to solve for all three unknowns. In order to have two more equations, two more satellites are needed. Rewriting the equation with three satellites:

\[
D = \sqrt{(x_s-x_p)^2 + (y_s-y_p)^2 + (z_s-z_p)^2} 
\] (3)

Equation (3) is the key relationship used in point positioning. However, as discussed, the pseudorange measurement is biased by the receiver clock offset. In state-of-the-art receivers, pseudorange measurements to multiple satellites are made simultaneously. The clock offset is, therefore, the same for all measurements made at a given moment in time. Clock offset is an additional unknown which then requires at least 4 satellites be tracked by the receiver simultaneously in order to solve for the point position plus clock offset. The complete pseudorange point positioning equation is then written as (equation (5)):
where:

PRi: Pseudorange to the i" satellite.

b: Clock offset of the receiver.

c: speed of light.

Equation (5) is a non-linear set of equations which require a linearization procedure in order to solve for the unknowns [B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, 1993]. The software required to solve this set of equations is included in the appendix.

It must be emphasized that using pseudoranges to solve for point position is not the final goal. Pseudorange-based point positioning is subject to a variety of error sources with the resulting accuracy being on the order of 100 meters, horizontal [Federal Radionavigation Plan, 1992]. Most of these error sources cancel in the differencing procedures used to solve for the baseline vector. However, multipath and thermal noise errors do not cancel and thus baseline vector determination using pseudoranges is only accurate on the order of 1 to 2 meters. For centimeter-level accuracy, a low-noise, low multipath range measurement, namely the carrier phase (also known as integrated Doppler ), is used for the baseline solution.
3.3 Carrier Phase Measurement

In this project, the L1 carrier frequency (1575.42 MHz) is used. According to the principle of Doppler frequency, the relative movement between receiver and satellite can be obtained. Starting from the Doppler shift of electromagnetic waves [Halliday, 1988]:

\[
f' = \sqrt{\frac{1 - \frac{v_{\text{los}}^2}{c^2}}{1 - \frac{v_{\text{los}}}{c}}} f_0
\]

(6)

where:
- \( f' \): Doppler shift frequency.
- \( f_0 \): Original frequency.
- \( v_{\text{los}} \): Line-of-sight (LOS) velocity.
- \( c \): Speed of light.

Since the line-of-sight velocity is much less than the speed of light, the Doppler shift can be approximated as:

\[
f' = f_0 \left( 1 - \frac{v_{\text{los}}}{c} \right)^{-1}
\]

(7)

Performing a binomial expansion on equation (7) yields equation (8).
Again, since the speed of light is much larger than the line-of-sight velocity, equation (8) can be reduced to equation (9).

\[ f' = f_0 \left( 1 + \frac{v_{\text{los}}}{c} + \frac{v_{\text{los}}}{c^2} + \ldots \right) \quad (8) \]

Solving for \( v_{\text{los}} \):

\[ v_{\text{los}} = \frac{c}{f_0} \left( f' - f_0 \right) = \frac{c}{f_0} (f' - f_0) \quad (10) \]

Let:

\[ f = f' - f_0 \] (the Doppler shift).

\[ v = v_{\text{los}}. \]

\[ \lambda = \frac{c}{f_0}. \]

Thus, the relationship between Doppler frequency, line-of-sight velocity and carrier wavelength can be expressed as (equation (11)) [Braasch, 1994 Spring]:

\[ f \cdot \lambda = v \quad (11) \]
The signal velocity is the speed of light, frequency is 1575.42 MHz and the wavelength is 0.19 meters. The frequency of the signal received by the receiver is not constant. It is affected by the relative motion of antennas and satellites. Since the wavelength of the signal remains constant, (refers to equation (11)), change of the frequency is the result of a change of velocity. Take the integral on both sides of equation (11), yielding equation (12).

\[ \lambda \int_{t_0}^{t} \omega f dt = \int_{t_0}^{t} \omega v dt \]  

(12)

On the right hand side of the equation, the integral of velocity is equal to a change-of-range. As such, the left hand side, being the integral of the Doppler frequency shift, will also yield change-of-range. In equation (12), frequency is the instantaneous Doppler frequency shift. The integral of instantaneous Doppler frequency is accumulated Doppler frequency, also called integrated Doppler. Integrated Doppler is measured by the GPS receiver. In the GPS receiver used in this project, the integrated Doppler is measured in units of carrier cycles. Thus, it has to be converted to normal distance units before it can be used.

The difference between pseudorange and integrated Doppler range can be graphically illustrated as in Figure 3.
Figure 3: Integrated Doppler range and Pseudorange.

As is indicated in figure 3, the integrated Doppler measurement is much less noisy than the pseudorange measurement. The trend of the curve is the same for both the pseudorange and the integrated Doppler. However, the offsets labeled $D_1$ and $D_2$ are missing in the integrated Doppler. $D_1$ and $D_2$ are the true distances between the satellite and the antenna at the time Doppler integration was started. The GPS receiver is not capable to provide this information at the time when the receiver has just been turned on. In the case of $D_2$, pseudorange measurements were made for $t_1$ seconds prior to the start of integrated Doppler measurements. In this case, a negative integrated Doppler was measured by the receiver. When the receiver starts to track a satellite, the integrated
Doppler is zero. This, obviously, is not the right range because the distance between the satellite and the antenna is not zero. The integrated Doppler is thus a biased range measurement. The initial offset between the pseudorange and the integrated Doppler is referred to as range ambiguity. This range ambiguity is different for each satellite being tracked. As a result, the ambiguity is not eliminated in the differencing procedure used to determine the baseline vector. Ambiguity resolution is the initialization procedure mentioned in the first chapter. In this paper, detailed discussion of ambiguity resolution is not the intention. Much research has been done on the ambiguity resolution problem. The main obstacle remaining in the ambiguity resolution is its processing time. The time required to resolve the ambiguities is on the order of 5 to 10 minutes. For navigation purposes, this processing time often is not acceptable. In this project, the short-baseline is used to eliminate the ambiguity and thus the need for a lengthy resolution process. The baseline vector can be therefore be determined within one second (the typical GPS measurement interval).

3.4 Single-Difference Formation

The baseline vector solution is based on equation (2). Equation (2) contains the baseline vector: \( \mathbf{b} \), unit vector from antenna to satellite: \( \hat{u} \), the range from antenna A to the satellite minus the range from antenna B to the satellite: \( \Delta \Phi \). However, equation (2) only holds in the ideal case. In practice, the GPS signal is affected by many error sources. Taking all error sources into consideration, the range from an antenna to the
satellite can be written as (equation (13)):

\[ \Phi = f\Phi + i\Phi + n\Phi + \varepsilon_i + \varepsilon_t + sc + rc + \varepsilon n \]  

(13)

where:

\( \Phi \): Measured integrated Doppler.

\( f\Phi \): Fractional portion of integrated Doppler measured by the receiver.

\( i\Phi \): Integer portion of integrated Doppler measured by the receiver.

\( n\Phi \): Range ambiguity.

\( \varepsilon_i \): Signal propagation delay due to the ionosphere.

\( \varepsilon_t \): Signal propagation delay due to the troposphere.

\( sc \): Satellite clock offset.

\( rc \): Receiver clock offset.

\( \varepsilon n \): noise sources (eg. multipath and thermal noise).

For the baseline vector in figure 2, the integrated Doppler measured by antenna A from the satellite is:

\[ \Phi_A = f\Phi_A + i\Phi_A + n\Phi_A + \varepsilon_i A + \varepsilon_t A + sc_A + rc_A + \varepsilon n_A \]  

(14)

The integrated Doppler measured by antenna B from satellite is:

\[ \Phi_B = f\Phi_B + i\Phi_B + n\Phi_B + \varepsilon_i B + \varepsilon_t B + sc_B + rc_B + \varepsilon n_B \]  

(15)

Because antenna A and antenna B are placed extremely close together, both experience the same effects from the ionosphere and troposphere. Therefore, \( \varepsilon_i A \) is equal
to $\varepsilon_i$, and $\varepsilon_t$, equal to $\varepsilon_t$. By the same token, both receiver A and receiver B track the same satellite. The satellite clock offset of equation (14) is equal to the satellite clock offset of equation (15). Thus equation (14) - equation (15) gives:

$$\Delta \Phi = \Phi_A - \Phi_B$$

(16)

where:

$$\Delta \Phi = (f\Phi_A - f\Phi_B) + (i\Phi_A - i\Phi_B) + (n\Phi_A - n\Phi_B)$$

+ ($-\varepsilon_i$ + $\varepsilon_i$) + ($\varepsilon_t$ - $\varepsilon_t$) + ($sc_A - sc_B$)

+ ($rc_A - rc_B$) + ($\varepsilon n_A - \varepsilon n_B$)

(17)

As discussed:

$$(-\varepsilon_i + \varepsilon_i) = (\varepsilon_t - \varepsilon_t) = (sc_A - sc_B) = 0.0.$$

Thus:

$$\Delta \Phi = (f\Phi_A - f\Phi_B) + (i\Phi_A - i\Phi_B) + (n\Phi_A - n\Phi_B) + (rc_A - rc_B) + (\varepsilon n_A - \varepsilon n_B)$$

(18)

The ionosphere error, troposphere error and satellite clock offset were eliminated in equation (18). The process of forming $\Delta \Phi$ by subtracting the integrated Doppler of one antenna from the integrated Doppler at another antenna is called the Single-difference.

Although, the single-difference has eliminated three error sources from the system, the receiver clock offset, a single-difference ambiguity, and noise error still remain. To eliminate the receiver clock offset, further processing, known as the double-difference, is
used. The double-difference will eliminate the receiver clock offset. The trade-off for using it is that more satellites have to be tracked in order to obtain the double-difference.

3.5 Double-Difference Formation

Only one satellite is needed in the formation of the single-difference. Unlike the single-difference, formation of the double-difference requires at least two satellites. Thus, the GPS receiver has to be able to track two satellites simultaneously. Consider the figure 4.

The difference between figure 4 and figure 2 is that one more satellite is tracked by the receivers. Both receiver A (connected to antenna A) and receiver B (connected to antenna B) are tracking satellite 1 and satellite 2 at the same time. The single-difference equation of each satellite will be:
The differenced receiver clock offset is the same in equation (19) and equation (20). This is because all measurements were formed simultaneously. The noise error of equation (19) and equation (20) are not the same, and do not cancel. For the purpose of illustrating the remaining processing, however, the noise error will be left out. Subtracting equation (19) from equation (20) eliminates the receiver clock offset term. The subtraction is performed on two single-difference equations, thus it is called the Double-difference. The double-difference equation is written as:

\[ \Delta \Phi 1 = (f\Phi_A^1 - f\Phi_B^1) + (i\Phi_A^1 - i\Phi_B^1) + (n\Phi_A^1 - n\Phi_B^1) + (rc_A - rc_B) + (en_A - en_B) \]  

\[ \Delta \Phi 2 = (f\Phi_A^2 - f\Phi_B^2) + (i\Phi_A^2 - i\Phi_B^2) + (n\Phi_A^2 - n\Phi_B^2) + (rc_A - rc_B) + (en_A - en_B) \]  

Equation (21) has three parts, the fractional, the integer portion of the integrated Doppler, and the integer ambiguity. \( n \) is the only unknown in equation (21). \( f \) and \( i \) are components of the integrated Doppler measurement. As mentioned, ambiguity resolution typically requires 5 to 10 minutes.
Much research has been done in the ambiguity resolution techniques to reduce the processing time. However, current methods still cannot achieve the resolution in less than a minute. In order to obtain the baseline result instantly, the unknown integer ambiguity has to be eliminated from the processing. One way achieve the goal is by employing the technique of the short-baseline.

3.6 Short-Baseline

Consider the baseline depicted in figure 5:

Figure 5: Short baseline interferometer with incident waves.

The incident signals at receiver A and receiver B have been divided into two parts, the fractional part and integer part. The single difference is:
To understand the advantage of a short baseline, consider figure (6). Because of geometrical constraints, the single-difference must be less than or equal to the length of baseline. Thus, if the baseline is less than one wavelength, the single-difference will be a fraction of a cycle and thus will have no ambiguity. Note that this is only true for the ideal case when the receiver clock offsets are zero. Since the clock offsets are not zero in reality, double-differences are required as an additional constraint to eliminate ambiguity.

\[
\Phi_A - \Phi_B = f\Phi_A + n_A\lambda - f\Phi_B - n_B\lambda
\] (22)

Figure 6: Short baseline interferometer with only fractional part of incident signal.

By doing the single difference, the satellite clock offsets are cancelled. Furthermore, the double difference will cancel the receiver clock offsets. However, the
double difference still has ambiguity. Even for the short baseline, the subtraction of two single differences can be larger than 1 wavelength. Consider figure 7 depicted as follows:

Figure 7: Interferometer with baseline length less than 1 wave length.

The single difference for each satellite can be expressed as:
The double difference is:

\[
\Delta \Phi_1 = 0.18 \cos 150 = -0.156 \text{ m} \\
\Delta \Phi_2 = 0.18 \cos 20 = 0.169 \text{ m}
\] (23)

The magnitude of the result is larger than 1 wavelength (0.19 m). Note this is the ideal case with a known baseline. In reality, the baseline is unknown and ambiguous integrated Doppler measurements are used to form the double-difference. Since the true double-difference is larger than one wavelength, ambiguity exists when forming the double-differences with the measurements.

There are several methods which could be used to solve this problem. One is to use the ambiguity resolution techniques and this is the most time consuming. Another method is to reduce the baseline length to less than half of a wavelength. In this case the true double-difference will never exceed 1 wavelength regardless of sign changes. The disadvantage of placing two antennas less than a half wavelength apart is that the antennas will experience mutual coupling effects. In this project, the distance between antenna A and antenna B is 18 cm, which is larger than one half wavelength (9.5 cm). In order to eliminate the ambiguity and avoid using the methods discussed above, a judicious set of satellites were selected to form the double differences. Consider the following figure. Seven satellites were tracked simultaneously by the receiver.
Satellite 4 was chosen as the base satellite used in all double-differencing. It is almost directly above the interferometer. Satellites located near satellite 4 where chosen such that the angle between the base satellite and the other single difference satellite would not be larger than 90 degrees. By examining the previous illustration, one can see that the resulting true double-difference will not be larger than 1 wavelength. Using this method, the ambiguity inherent in the double-difference is eliminated.

Before describing the actual implementation of the system, the technique involved in solving the baseline equation should be discussed. The baseline is described in 3 dimensional space in ECEF (Earth Center Earth Fixed) coordinates. This essentially consists of a Cartesian coordinate system with origin at the center of the Earth. The x-axis intersects the prime meridian at the equator, the z-axis is roughly coincident with the North pole and the y-axis is orthogonal to the x and z axes [Hoffmann-Wellenhof, 1993]. One equation is not sufficient to solve for the three unknowns. Three equations are required.
Each double-difference requires data from two satellites and therefore, at least 4 satellites must be tracked to solve the equations. A complete interferometer using 4 satellites can be described as follows:

Figure 9: Interferometer with four satellites.

\[
\begin{align*}
\bar{b} \cdot \hat{u}_1 &= \Delta \Phi_1 \\
\bar{b} \cdot \hat{u}_2 &= \Delta \Phi_2 \\
\bar{b} \cdot \hat{u}_3 &= \Delta \Phi_3 \\
\bar{b} \cdot \hat{u}_4 &= \Delta \Phi_4
\end{align*}
\]  

(25)

Rewrite the equation (25) in matrix form:
Using the single difference equation above to obtain double difference equation, a satellite has to be chosen to subtract from the others. The way to chose the base satellite is to find the one that has the highest elevation angle. Here, satellite 1 was assumed the to have the highest elevation angle. The double difference equation is:

\[
\begin{bmatrix}
    u_{1x} & u_{1y} & u_{1z} \\
    u_{2x} & u_{2y} & u_{2z} \\
    u_{3x} & u_{3y} & u_{3z} \\
    u_{4x} & u_{4y} & u_{4z}
\end{bmatrix}
\cdot
\begin{bmatrix}
    bx \\
    by \\
    bz
\end{bmatrix}
=
\begin{bmatrix}
    \Delta \Phi_1 \\
    \Delta \Phi_2 \\
    \Delta \Phi_3 \\
    \Delta \Phi_4
\end{bmatrix}
\]  \hspace{1cm} (26)

Using the single difference equation above to obtain double difference equation, a satellite has to be chosen to subtract from the others. The way to chose the base satellite is to find the one that has the highest elevation angle. Here, satellite 1 was assumed the to have the highest elevation angle. The double difference equation is:

\[
\begin{bmatrix}
    u_{1x} - u_{2x} & u_{1y} - u_{2y} & u_{1z} - u_{2z} \\
    u_{1x} - u_{3x} & u_{1y} - u_{3y} & u_{1z} - u_{3z} \\
    u_{1x} - u_{4x} & u_{1y} - u_{4y} & u_{1z} - u_{4z}
\end{bmatrix}
\cdot
\begin{bmatrix}
    bx \\
    by \\
    bz
\end{bmatrix}
=
\begin{bmatrix}
    \Delta \Phi_1 - \Delta \Phi_2 \\
    \Delta \Phi_1 - \Delta \Phi_3 \\
    \Delta \Phi_1 - \Delta \Phi_4
\end{bmatrix}
\]  \hspace{1cm} (27)

Let the baseline vector be represented by \([\vec{b}]\), the unit vector matrix be represented by \([\vec{u}]\), and the integrated Doppler double-difference matrix be represented by \([\Delta \Phi]\).

Thus,

\[
\begin{bmatrix}
    \vec{u} \\
\end{bmatrix}
\cdot
\begin{bmatrix}
    \vec{b}
\end{bmatrix}
=
\begin{bmatrix}
    \Delta \Phi
\end{bmatrix}
\]  \hspace{1cm} (28)

The ordinary least squares solution for the baseline vector is then [Freund. J, 1992]:

\[
\begin{bmatrix}
    \vec{b}
\end{bmatrix}
=
\left(\begin{bmatrix}
    \vec{u} \\
\end{bmatrix}
\cdot
\begin{bmatrix}
    \vec{u}
\end{bmatrix}
\right)^{-1}
\cdot
\begin{bmatrix}
    \vec{u}
\end{bmatrix}
\cdot
\begin{bmatrix}
    \Delta \Phi
\end{bmatrix}
\]  \hspace{1cm} (29)

Equation (29) is the final form used in the baseline determination in this project.

In order to determine successfully the baseline vector using double differences, at least 4 satellites have to be tracked by both receivers at the same time. Most GPS receivers can
track more than 4 satellites simultaneously. For the receivers used in this project, the maximum number of satellites which can be tracked is 10. Thus, the dimension of the matrix being used in equation (29) varies from time to time. The change of matrix dimension does not change the form of equation (29). Additional measurements simply allow error reduction via the least-squares principle.

The process of obtaining the baseline vector has been discussed in this chapter. The actual implementation of the system will be discussed in the next chapter.
4 Implementation

This chapter discusses the hardware and software used to implement the GPS interferometric pointing system.

4.1 System Set Up

The interferometer includes two GPS antennas, two GPS receivers, and a computer. It was set up in two configurations as follows (the Zero-baseline set up will be explained later):

Figure 10: (A) Short baseline interferometer. (B) Zero baseline interferometer.
where:

Antennas: NovAtel GPS Antenna Model 511.

Receivers: NovAtel GPS receivers Model 2151R.

Many pointing systems have been developed by various researchers to perform attitude determination [Brown, 1990], [Neira, 1990], [van Graas, 1991], [Cohen, 1992], [Cohen, 1993], and [Ferguson, 1991]. Unlike these other systems, the length of the pointer is only 18 cm in this project.

The communication link between the computer and receivers are RS232 cables which connect receiver input/output and the serial ports of the computer. It is a 2-way communication link between the receivers and the PC. Commands were sent from the PC to the receivers as well as having data transmitted from the receivers to the PC. In order to achieve a fast communication rate for the data link as well as make provisions for eventual real time data processing, interrupt driven communication with the serial port is required. In the In/Out port of the receiver, there is a built-in interrupt driven port. All of the incoming commands will be stored in the In/Out buffer. The receiver will interrupt with the command when a carriage return signal (byte) is detected. On the PC side, the author has written and implemented interrupt driven serial communications for the port. Thus, the port services the data transmission in the background while the main data processing program is executing. The routines were required to handle 10 measurement blocks per second [GPSCard, 1993]. Testing of this system at such a rate for 20 hours resulted in zero lost data bits.
4.2 Software Implementation

There are two parts of the software. One handles the communication between the PC and receiver. The other handles the data manipulation which includes decoding GPS data sent from the receiver along with calculating the baseline solution. The source code implemented by the author is given in the appendix. The flow chart of the program is depicted in figure 11 and figure 12.

\[\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{flowchart.png}
\caption{Set up interrupt services.}
\end{figure}\]
4.3 Data Collection

As mentioned earlier, two configurations were used: short baseline and zero baseline. Zero baseline tests allow for the examination of system noise characteristics [ 
Nolan, 1991]. For the short baseline tests, the two antennas were mounted on a piece of 1/4 inch plywood as shown in figure 13.

![Figure 13: Implemented interferometer.](image)

Test Date: Afternoon August 11, 1994.

Test Location: Ohio University Airport.

The complete system setup is shown in figure 14.

![Figure 14: Complete system set up.](image)
Overall, the data was collected under good weather conditions. The presence of the large metal roof has been shown to be a strong multipath error source [Braasch, Michael, 1994]. In next the chapter, the data will show the evidence of the multipath.
5 Test Result and Data Analysis

5.1 Zero baseline results

As mentioned earlier, a zero length baseline has double-differences in which most of the error sources, such as multipath, ionosphere, troposphere, satellite clock offset and receiver clock offset cancel out. But the zero baseline test does not eliminate thermal noise. Because of this characteristic of the zero baseline, it is a good tool to do system analysis. It provides the noise level of the system itself. With this information, one can evaluate the results produced by the system more accurately. In this project, a zero baseline was implemented to investigate the noise level. In order to demonstrate pseudorange as well as integrated Doppler noise, double-differences were formed using the pseudorange and the integrated Doppler. Since the baseline is zero length, the double-difference is simply composed of residual error due to thermal noise.

This set of data was collected at a rate of 1 sample per second for 80 seconds. The satellites chosen were 20, 24, 4, and 5. The reason for choosing these four satellites was due to the fact that they had the highest elevation angles. Recall from chapter 3 that a judicious choice of satellites eliminates ambiguities in short baseline double-differences. Satellite 20 is the base satellite used to form double-differences with the other satellites. The result of the zero baseline residuals generated using pseudorange measurements are depicted in figures 15, 16, and 17, respectively. The result of zero baseline residuals generated by the integrated Doppler measurements are depicted in figures 18, 19, and 20,
Figure 15: Pseudorange double-difference residual using satellite 20 and 24.
Figure 16: Pseudorange double-difference residuals using satellite 20 and 4.
Figure 17: Pseudorange double-difference residuals using satellite 20 and 5.
Figure 18: Integrated Doppler double-difference residuals using satellite 20 and 24.
Double-Difference Residuals, SV 20-4

Integrated Doppler, zero baseline

Figure 19: Integrated Doppler double-difference residuals using satellites 20 and 4.
Double-Difference Residuals, SV 20-5

Integrated Doppler, zero baseline

Figure 20: Integrated Doppler double-difference residuals using satellite 20 and 5.
respectively. As will be shown later in this chapter, the system noise directly impacts the accuracy of the pointing system. According to the double-difference residuals using the integrated Doppler, the noise level is on the order of 0.5 cm. This implies that the raw integrated Doppler noise level is on the order of millimeters. Again, all of this noise will be seen in the next section in the results of the short baseline determination. Comparing the double-difference of the integrated Doppler to the double-difference of the pseudorange, the pseudorange has significantly higher noise level. Referring to the pseudorange figures, the largest residual (figure 16) is about 50 cm. The noise level of the pseudorange is almost 100 times larger than for the integrated Doppler. This verifies the fact that the integrated Doppler is much less corrupted by noise than the pseudorange. Table 2 is generated to compare the noise level between pseudorange and integrated Doppler.

**Table 2:** Standard deviation of zero baseline double-difference using pseudorange and integrated Doppler.

<table>
<thead>
<tr>
<th>Satellite number</th>
<th>Double-difference residual Pseudorange Standard deviation (m)</th>
<th>Double-difference residual Integrated Doppler Standard deviation (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SV 20 and SV 24</td>
<td>0.091</td>
<td>2.07e-03</td>
</tr>
<tr>
<td>SV 20 and SV 4</td>
<td>0.143</td>
<td>2.45e-03</td>
</tr>
<tr>
<td>SV 20 and SV 5</td>
<td>0.097</td>
<td>2.22e-03</td>
</tr>
</tbody>
</table>
For the example of the double-difference with satellite 20 and 24, the standard deviation of the integrated Doppler is 2.07e-3 meters, but the standard deviation of the pseudorange is 0.091 meters. Concluding the results of the zero baseline: first, noise exists in the system at the half-centimeter level for integrated Doppler double-differences. Second, the pseudorange noise is much larger than the integrated Doppler noise.

5.2 Length, Azimuth Angle, and Elevation Angle

In the rest of this chapter, the short baseline testing results will be posted, analyzed and discussed. There are three types of results that have been generated from two different sets of data. These data were collected at different times but under the same environment and error source conditions. The three types of data can be categorized as follows:

1. Length of baseline.
2. Azimuth angle of the baseline vector.
3. Elevation angle of the baseline vector.

For these three types of results, only the length of the baseline is known. That is, an exact truth reference is available for baseline length but not azimuth or elevation. It was mentioned in the previous chapter that the length of the baseline was 18 cm. The length is used as the reference to compare to the results generated by the pointing system. The pointing system calculates the baseline in Earth Centered Earth Fixed (ECEF) coordinates. Using the x, y, z coordinates of the baseline vector, the length of the baseline can be calculated with the usual distance formula.
In addition to the length of the baseline, the azimuth angle of the baseline vector was also calculated. Unlike the ECEF system, azimuth angle is defined in the local level plane which is also called the East North Up (ENU) coordinate system. The three axes of the ENU coordinate system are East, North, Up. East axis points to East. North axis points to North, and Up axis points upward to the local vertical. Thus, prior to computing azimuth, the baseline vector coordinates must be translated from ECEF to ENU. The method of transferring ECEF coordinates into ENU coordinates is explained in the software in the appendix. It is important to know the usefulness of ENU coordinates. In most pointing systems, the interest is focused on the system orientation with respect to the local plane. This is especially true when performing attitude determination using the pointing system. Azimuth angle is also defined in ENU coordinates. It is positive in the clockwise direction from North, and negative in the counter clockwise direction. Elevation angle is also defined in the local level system with positive angle above the local level plane and negative angles below it. See figure 21 for example.

![Figure 21: Vector in East North Up coordinates](image)

Azimuth angle 30 degree
Elevation angle 45 degree

\[ \text{Azimuth angle 30 degree} \]
\[ \text{Elevation angle 45 degree} \]
As mentioned earlier while setting up the pointing system for data collection, an exact truth reference was not available for azimuth angle or elevation angle. Rough estimates of azimuth angle and elevation angle of the pointing system were approximated by mounting the system parallel to well known landmarks. One is the Ohio University airport runway and the other is the metal plate which the antennas were mounted on. The runway is oriented in the direction of North 70° and the metal plate is assumed to be parallel with the local level plane. The pointing system was manually lined up with the airport runway and placed on top of the metal plate. Thus, the pointer is roughly in line with North 70° and level with the local level plane. In other words, the azimuth angle of the pointer is 70° and the elevation angle is 0°. The second data collection effort involved mounting the pointer in the same location but rotated by 90°. In other words, the azimuth angle of the second pointer is -20° and the elevation angle is 0°.

5.3 Results

The first set of data was taken during a 10 minute interval at a rate of 10 Hz. The length of the baseline, azimuth angles and elevation angles of the first pointer are illustrated in figures 22, 23 and 24, respectively. The length of the baseline fluctuates between 17 cm and 20 cm with mean 17.88 cm and standard deviation 0.766 cm. Azimuth angle varies between 52° and 62° with mean 56.763° and standard deviation 1.934°. The elevation angle varies from +10° to -10° with mean -4.972° and standard deviation 6.761°.
Length of baseline
Points at azimuth N 60, elevation 0

Figure 22: Length of short baseline vector points at azimuth angle North 60 and elevation angle 0 degree.
Azimuth angle of baseline vector

Points at azimuth N 60, elevation 0

Figure 23: Azimuth angle for short baseline points at azimuth angle North 60 and elevation angle 0 degree.
Elevation angle of baseline vector

Points at azimuth N 60, elevation 0

Figure 24: Elevation angle of short baseline points at azimuth angle North 60 and elevation angle 0 degree.
Figure 25: Length of short baseline points at azimuth angle North -30 and elevation angle 0 degree.
Azimuth angle of baseline vector

Points at azimuth N-30, elevation 0

Figure 2.6: Azimuth angle of short baseline points at azimuth angle North -30 and elevation angle 0 degree.
Elevation angle of baseline vector

Points at azimuth N -30, elevation 0

Figure 27: Elevation angle of short baseline vector points at azimuth angle North -30 and elevation angle 0 degree.
For the second set of data, the length of the baseline, azimuth angle and elevation angle are plotted in figures 25, 26 and 27, respectively. The baseline length varies from 17 cm to 20 cm with mean 19.28 cm and standard deviation 0.73 cm. The value of azimuth angle is around $-30^\circ$ with very little fluctuation. It has a mean value of $-30.3^\circ$ and standard deviation of $0.668^\circ$. The elevation angle varies from $+10^\circ$ to $+3^\circ$ with mean $8.0^\circ$ degree and standard deviation $1.9^\circ$. The summation of all the graphic information is listed in Table 3.

**Table 3:** Statistical results for baseline.

<table>
<thead>
<tr>
<th></th>
<th>Type of data</th>
<th>Case 1</th>
<th>Case 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length (cm)</td>
<td>mean</td>
<td>17.888</td>
<td>19.277</td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>0.766</td>
<td>0.73</td>
</tr>
<tr>
<td>Azimuth (degree)</td>
<td>mean</td>
<td>56.76</td>
<td>-30.33</td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>1.93</td>
<td>0.66</td>
</tr>
<tr>
<td>Elevation (degree)</td>
<td>mean</td>
<td>-4.97</td>
<td>8.00</td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>6.76</td>
<td>1.9</td>
</tr>
</tbody>
</table>

For both sets of data, the noise errors are similar. The first set of data has a mean value of azimuth angle at $56^\circ$. Thus, it would appear that the pointer is not exactly lined up with the airport runway. This is undoubtedly due to the lack of a truth reference while mounting the pointing system. The accuracy was confirmed by the second set of data.
The second set of data has a mean azimuth angle of -30.33°, which is very nearly 90° offset from the first set of data.

The difference between these two sets of data is the average computed elevation angle. This difference likely stems from the fact that the metal plate is not complete flat and thus may have created non-zero elevation angles which differed depending upon the pointer orientation. Since there is 90° of azimuth rotation between the two pointers, the elevation angle may be different.

An additional error source is the presence of multipath. The pointer was mounted above a large metal roof. It was therefore susceptible to signal degradation due to undesired reflections and diffraction. In the trace of the curves, the evidence of multipath can be seen. The fluctuation of each small section of data is only 0.2 cm. This is due to the system noise. However, the sinusoidal pattern of the overall curves shows the effect of the multipath [Braasch, M., 1994,b].

Besides multipath, there is another factor which amplifies error in the system. It is the geometry of the satellites. In chapter 3, the highest elevation angle satellites were selected for the double difference to avoid the ambiguity. This creates a poor dilution of precision (DOP) condition while solving the simultaneous equation. In chapter 3, the double difference equation was shown to be:

\[
[DD] = [u] \cdot [b]
\]  

(30)

Where DD, u and b are all in matrix form. If Gaussian white noise error (such as thermal noise) corrupts the double-difference measurements, the accuracy of the baseline solution
is given by [Hoffmann-Wellenhof, 1993]:

\[
DD - DOP = \sqrt{Q_{11} + Q_{22} + Q_{33}}
\]

[ \begin{bmatrix} Q \end{bmatrix} = ( \begin{bmatrix} u \end{bmatrix}^T \begin{bmatrix} u \end{bmatrix} )^{-1} 
\sigma_b = (DD - DOP)(\sigma_{DD})
\]

where:

\( \sigma_b \): RMS error of calculated baseline coordinates.
\( \sigma_{DD} \): RMS error of double-difference measurements.
\( Q_{11}, Q_{22}, Q_{33} \) are the diagonal elements of the Q matrix.

DD-DOP is the dilution of precision of the baseline determined with double-difference processing.

The DD-DOP is 6.45 for pointer one and 7.86 for pointer two. The ideal case is to have DD-DOP less than or equal to 1. Under such circumstances, satellite geometry is not amplifying the double-difference measurement errors. As shown in section 5.1, the integrated Doppler double-difference noise level is on the order of 0.5 cm and the high value of DD-DOP are multiplying this in the propagation to baseline component errors. Noise is not the only double-difference error source. As mentioned, multipath serves to degrade the results as well.

As all the results show, the short baseline pointing system is feasible. It is a system which could contribute in a variety of applications, especially as an initialization aid in real time attitude determination using long baselines. The system by itself can be used as a medium accuracy stand alone attitude determination system. In the next chapter, the discussion will be focused on the use of the short baseline and further application.
6 Conclusion

In this paper, a short baseline GPS interferometric pointing system was successfully implemented. The results show the accuracy of the short pointing system. Under the bad DOP conditions, the precision of results is still within what was expected. With the DD-DOP values of 6 and 7, the standard deviation of the azimuth angle and elevation are within a few degrees, the standard deviation of the length of the baseline less than 1 cm. System accuracy will improve significantly in applications with less severe multipath environments. In the beginning of this thesis, the short pointing system was emphasized because of its time efficient algorithms. For long baselines requiring ambiguity resolution, the short pointer will provide a better initial estimate which will reduce the search time for the unknown integer ambiguities.

In this project, the software was designed to allow for real-time attitude determination [van Graas, 1992]. For the data communication link between the PC and the GPS receiver, assembly language routines were developed and tested to handle the high baud rate data transmit/receive. The data transmission rate can reach as high as 10 measurement blocks per second. At such a data rate, the interrupt driven serial port showed its reliability by running 20 hours without losing a single bit.

The entire set of processing software was written in C. With the unique capability of C, such as, structure, pointer and dynamic memory allocation, the efficiency of the software is maximized. It is expected that many of the software modules written for the short baseline processing may be exploited for long baseline systems as well.
7 Appendix

7.1 Appendix 1

C code for baseline determination.

```c
/* Gps.h */
/* Description: Header file for Short Baseline Determination. */
/* This file contains the names of all the C functions for the short baseline */
/* determination. Most of the function name and variable name are self explained. */
/* Input: none: */
/* Output: none: */
/* Author: Lap Ho */
/* Date: November, 1994 */

/**************************************************************/

/* C library functions */
#include <stdio.h>
#include <math.h>
#include <float.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>

/* Define constants */
#define BUFF 100 /* Comm port buffer */
#define TRUE 0 /* Boolean */
#define FALSE 1 /* Boolean */
#define TINY 1.0e-20
#define LIGHT_SPEED 2.99792458e+8f

/* Macro function */
/* Swaps two values, a and b */
#define SWAP( a, b) {double temp=(a);(a)=(b);(b)=temp;}

/* External functions ( assembly functions for serial port communication ) */
```
/* See Assembly language for detail description */
extern setint();
extern setpbs();
extern clsint();
extern go_stop();
extern append_b();
extern append_2();
extern takeb();
extern takeb2();
extern go_stop();
extern outcom1();
extern outcom2();
extern setintr();
extern clsint();
extern display_1();
extern display_2();
extern go_stop();
extern tranb();
extern deltab();
extern cl_ring();

/* Structure of the ID name for NovAtel receiver command: rqgb */
/* For more information, refer to GPScard book */
struct rqgb_id_word
{
  unsigned long int length_word;
  int number_of_obs;
  int week_number;
  double second_of_week;
  long int receiver_status;
}id_name;
struct rqgb_id_word id_name2;

/* Structure of ranging data for Novatel receiver command: rqgb */
struct rqgb_range_record
{
  int lock_time;
  int channel_number;
  int tracking_state;
  int PRN;
  int phase_parity_known_flag;
  int phase_lock_flag;
  double accumulated_doppler_range;
double doppler_frequency;
double initial_accu;
double pseudorange;
double trans_time;
};
/* Assign ranging structure to comm port 1 and 2 */
struct rqgb_range_record channel[40];
struct rqgb_range_record channel2[40];

/* Structure for the quantities of ephemeris data, NovAtel receiver command 'repb' */
struct repb_data
{
    long int prn;
    double toc,tgd,af2,af1,af0,crs,deltan,mo,cuc,eccen,cus,sqrsma,Toe,cic,
    omegao,cis,io,crc,omega,omgdot,idot;
}data_repb[40];

/* Function which decodes ranging data from comm port 1 */
void decode(int onsight_sv[40],int checksum,struct rqgb_id_word *id_pt, struct
rqgb_range_record *channel);

/* Function which decodes ranging data from comm port 2 */
void decode2(int onsight_sv2[40],int checksum2,struct rqgb_id_word *id_pt2, struct
rqgb_range_record *channel2);

/* Function which decodes ephemeris data from comm port 1 */
void decode_ephemeris(int checksum, struct repb_data *data_repb);

/* Function which calculates satellite position using ephemeris [Press, 1990] */
void ephcal(double *dtsv,int onsight_sv[40],double satellite_position[40][4],
struct rqgb_id_word *id_pt, struct repb_data data_repb[40]);

/* Function which calculates distance between satellites and point */
void pseudorange(int onsight_sv[40],double estimated_xyzb[4],
double svecef_matrix[40][4],struct rqgb_id_word *id_pt,
double estimated_pseudorange[40]);

/* Function which transpose a matrix */
void transpose_matrix(double h_matrix[20][20],double temp[20][20], int row,
int column4);

/* Function which multiply 2 matrix */
void matrix_multiply(double temp[20][20],double h_matrix[20][20],
double multi[20][20], int n, int m, int o);

/* Function which invers a matrix [ Rockwell, 1985]*/
void inverse_matrix(double a[20][20], int n, double b[20][20], int m);

/* Function which output error message */
void nerror(char error_text[]);

/* Function for single-difference formation */
void single_difference(int onsight_sv[40], struct rqgb_id_word *id_pt,
struct rqgb_range_record channel[40], struct rqgb_range_record channel2[40],
double single[10]);

/* Function for double-difference formation */
void double_difference(int onsight_sv[4], struct rqgb_id_word *id_pt,
double single[10], double double_d[10], double azimuth[10]);

/* Function which calculates unit vector */
void unit_vector(double user_position[4], double satellite_position[40][4],
struct rqgb_id_word *id_pt, struct rqgb_range_record channel[40],
double u_vector[10][3], int onsight_sv[20]);

/* Function for unit vector for double-difference */
void double_unit_vector(struct rqgb_id_word *id_pt, double u_vector[10][3],
double d_unit[10][3]);

/* Function which output commands to comm ports */
it send_out();

/* Function searching header of data from comm port 1 */
long int header1(char *checksum);

/* Function searching header of data from comm port 2 */
long int header2(char *checksum2);

/* Function calculates point position */
point_position(struct rqgb_id_word *id_pt, struct rqgb_range_record channel[40],
double user_position[4], double satellite_position[40][4], int *onsight_sv);

/* Function for baseline solution */
short_baseline(struct rqgb_id_word *id_pt, double d_unit[10][3], double double_d[10]);

/* Function which converts ECEF coordinates to latitude, longitude, and height */
void ec2llh(double xyz[4], double plh[3]);

/* Function which converts ECEF coordinates to East, North, Up */
void ec2enu(double xyz[3], double enu[3], double orgece[3], double orgllh[3]);

/* Function which calculates azimuth and elevation angle */
double enu2ae(double enu[3]);
#include "gps.h"
main()
{
    /* Variables */
    char checksum, checksum2, header;
    int c, i, j, iterations, port1_bps, port2_bps, onsight_sv[20], onsight_sv2[20];
    double single[10], double_d[10], d_unit[10][3], u_vector[10][3];
    double dtsv;
    double user_position[4], satellite_position[40][4];
    int delta;
    int doppler_flag;
    double xyz[3], enu[3], ORGEC[3], plh[3];
    double elevation[10], azimuth[10];

    /* Estimate user position */
    user_position[0] = 669402.860; /* 600000.0e0; */
    user_position[1] = -4903252.614; /* -4000000.0e0; */
    user_position[2] = 4010599.46; /* 4000000.0e0; */
    user_position[3] = 100.0e0;

    /* Serial port configuration */
    printf("\n baud rate (bps) for comm1 = ");
    scanf("%u", &port1_bps);
    printf("\n baud rate (bps) for comm2 = ");
    scanf("%u", &port2_bps);
    setintr();
    setbps(port1_bps, port2_bps);

    /* Initialize ephemeris data, and ranging data */
    printf("Updating ranging and ephemeris data...\n");
    printf("Press Tab to continue and to terminate program \n");
    while(send_out() == TRUE)
    {
        /* Update data from comm port 1 */
        header = header1(&checksum);
        /* header = 24, range data; header = 14, ephemerides data */
if(header == 24) decode(onsight_sv, checksum, &id_name, channel);
else if (header == 14) decode_ephemeris(checksum, data_rep);
header = 0;

/* Update data from comm port 2 */
header = header2(&checksum2);
if(header == 24) decode2(onsight_sv2, checksum2, &id_name, channel2);
header = 0;
}

/* baseline determination */
while(send_out() == TRUE)
{
  /* update new ephemeris and ranging data */
  header = header1(&checksum);
  if(header == 24) decode(onsight_sv, checksum, &id_name, channel);
  else if (header == 14) decode_ephemeris(checksum, data_rep);
  header = 0;
  header = header2(&checksum2);
  if(header == 24) decode2(onsight_sv2, checksum2, &id_name, channel2);
  header = 0;

  /* Calculates satellites position */
  ephcal(&dtsv, onsight_sv, satellit_pos, &id_name, data_rep);

  /* Calculates user position */
  point_position(&id_name, channel, user_pos, satellit_pos, onsight_sv);

  /* Converts point position from ECEF coordinates to Latitude, Longitude and height */
  ec2llh(user_pos, plh);
  for(i = 0; i < id_name.number_of_obs; i++)
  {
    xyz[0] = satellit_pos[onsight_sv[i]][0];
    xyz[1] = satellit_pos[onsight_sv[i]][1];
    xyz[2] = satellit_pos[onsight_sv[i]][2];
    orgece[0] = user_pos[0];
    orgece[1] = user_pos[1];
    orgece[2] = user_pos[2];

    /* Calculate Elevation angle of each satellite */
    ec2llh(orgece, plh);
    ec2enu(xyz, enu, orgece, plh);
    elevation[i] = enu2ae(enu);
} 

/ * Single difference formation */
  single_difference(onsight_sv,&id_name,channel,channel2,single);

/ * Double difference formation */
  double_difference(onsight_sv,&id_name,single,double_d,azimuth);

/ * Calculates unit vector */
  unit_vector(user_position,satellite_position,&id_name,channel,u_vector,onsight_sv);

/ * Calculate unit vector for double-difference */
  double_unit_vector(&id_name,u_vector,d_unit);

/ * Determine base-line coordinates */
  short_baseline(&id_name,d_unit,double_d);

/ * reset serial port */
  clsint();
}
#include "gps.h"

void decode(int doppler_flag, int onsight_sv[40], int checksum, struct rqgb_id_word *id_pt, struct rqgb_range_record *channel)
{
    /* local variable */
    int ibyte_temp, prn;
    long libyte_shift, second_of_week, libyte_temp;
    unsigned long lpseudorange;
    long double pseudorange;
    long laccumulated_doppler_range;
    double accumulated_doppler_range;
    unsigned long ldoppler_frequency;
    double dummy, doppler_frequency;
    int range_decode_loop;

    /* Takes bytes from comm port ring buffer */
    /* Decode ID information of each set of data */
    id_pt->length_word = takeb();
    libyte_shift = takeb();
    libyte_shift = libyte_shift << 8; /* left shift 8 bits */
    id_pt->length_word = id_pt->length_word | libyte_shift;
    libyte_shift = takeb();
    libyte_shift = libyte_shift << 16;
    id_pt->length_word = id_pt->length_word | libyte_shift;
    libyte_shift = takeb();
    libyte_shift = libyte_shift << 24;
    id_pt->length_word = id_pt->length_word | libyte_shift;

    id_pt->number_of_obs = takeb();
    libyte_shift = takeb();
    libyte_shift = libyte_shift << 8;
    id_pt->number_of_obs = id_pt->number_of_obs | libyte_shift;

    id_pt->week_number = takeb();
byte_shift = takeb();
byte_shift = byte_shift << 8;
id_pt->week_number = id_pt->week_number | byte_shift;

second_of_week = takeb();
byte_shift = takeb();
byte_shift = byte_shift << 8;
second_of_week = second_of_week | byte_shift;
byte_shift = takeb();
byte_shift = byte_shift << 16;
second_of_week = second_of_week | byte_shift;
byte_shift = takeb();
byte_shift = byte_shift << 24;
second_of_week = (second_of_week | byte_shift);
dummy = second_of_week * 1.0;
id_pt->second_of_week = dummy / 100;

id_pt->receiver_status = takeb();
byte_shift = takeb();
byte_shift = byte_shift << 8;
id_pt->receiver_status = id_pt->receiver_status | byte_shift;
byte_shift = takeb();
byte_shift = byte_shift << 16;
id_pt->receiver_status = id_pt->receiver_status | byte_shift;
byte_shift = takeb();
byte_shift = byte_shift << 24;
id_pt->receiver_status = id_pt->receiver_status | byte_shift;

/* Decode ranging data */
if (id_pt->number_of_obs < 40)
{
    for(range_decode_loop = 0; range_decode_loop < id_pt->number_of_obs; range_decode_loop++)
    {
        channel[0].lock_time = takeb();
        byte_shift = takeb();
        byte_shift = byte_shift << 8;
        channel[0].lock_time = channel[0].lock_time | byte_shift;

        ibyte_temp = takeb();
        channel[0].channel_number = (ibyte_temp >> 4) + 1;

        channel[0].tracking_state = ibyte_temp & 0x0007;
ibyte_temp = takeb();
prn = ibyte_temp >> 2;
(channel+prn) -> PRN = prn;
onsight_sv[range_decode_loop] = prn;

(channel+prn) -> phase_parity_known_flag = (ibyte_temp >> 1) & 1;
(channel+prn) -> phase_lock_flag = ibyte_temp & 0x0001;
(channel+prn) -> lock_time = channel[0].lock_time;
(channel+prn) -> channel_number = channel[0].channel_number;
(channel+prn) -> tracking_state = channel[0].tracking_state;

laccumulated_doppler_range = takeb();
libyte_shift = takeb();
libyte_shift = libyte_shift << 8;
laccumulated_doppler_range = laccumulated_doppler_range | libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 16;
laccumulated_doppler_range = laccumulated_doppler_range | libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 24;
laccumulated_doppler_range = laccumulated_doppler_range | libyte_shift;
accumulated_doppler_range = laccumulated_doppler_range * 1.0;
(channel+prn) -> accumulated_doppler_range = accumulated_doppler_range / 256;

libyte_temp = takeb();
lDoppler_frequency = libyte_temp >> 4;
libyte_shift = takeb();
libyte_shift = libyte_shift << 4;
lDoppler_frequency = lDoppler_frequency | libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 12;
lDoppler_frequency = lDoppler_frequency | libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 20;
lDoppler_frequency = lDoppler_frequency | libyte_shift;
if((libyte_shift & 0x08000000ul) != 0)
    {
        doppler_frequency = ((~lDoppler_frequency & 0xFFFFFFFul) + 1) * (-1.0);
    }
else
    {
        doppler_frequency = lDoppler_frequency * 1.0;
    }
(channel + prn) -> doppler_frequency = doppler_frequency / 256;

lpseudorange = takeb();
libyte_shift = takeb();
libyte_shift = libyte_shift << 8;
lpseudorange = lpseudorange | libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 16;
lpseudorange = lpseudorange | libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 24;
lpseudorange = lpseudorange | libyte_shift;

libyte_temp = libyte_temp & 0x0000000FUL;
if(( libyte_temp >> 3 ) == 1)
  {
    libyte_shift = ~libyte_temp;
    libyte_shift = libyte_shift & 0x0000000FUL;
    pseudorange = (libyte_shift & 1) * pow(2, 32)
      + ((libyte_shift >> 1) & 1) * pow(2, 33)
      + ((libyte_shift >> 2) & 1) * pow(2, 34);
    lpseudorange = ~lpseudorange;
    pseudorange = (pseudorange + lpseudorange + 1) * (-1.0);
  }
else
  {
    pseudorange = (libyte_temp & 1) * pow(2, 32)
      + ((libyte_temp >> 1) & 1) * pow(2, 33)
      + ((libyte_temp >> 2) & 1) * pow(2, 34);
    pseudorange = pseudorange + lpseudorange;
  }

channel + prn) -> pseudorange = pseudorange / 128;

(channel + prn) -> trans_time = id_pt -> second_of_week -
((channel + prn) -> pseudorange / LIGHT_SPEED);
#include "gps.h"

void decode2(int doppler_flag, int onsight_sv2[40], int checksum2, struct rqgb_id_word *id_pt2, struct rqgb_range_record *channel2)
{
    /* Local variables */
    int libyte_temp;
    int prn;
    long libyte_shift, second_of_week;
    long libyte_temp;
    unsigned long lpseudorange;
    long double pseudorange;
    long accumulated_doppler_range;
    double accumulated_doppler_range;
    unsigned long Idoppler_frequency;
    double dummy, doppler_frequency;
    int range_decode_loop;

    /* Decode ID information */
    id_pt2->length_word = takeb2();
    libyte_shift = takeb2();
    libyte_shift = libyte_shift << 8;
    id_pt2->length_word = id_pt2->length_word | libyte_shift;
    libyte_shift = takeb2();
    libyte_shift = libyte_shift << 16;
    id_pt2->length_word = id_pt2->length_word | libyte_shift;
    libyte_shift = takeb2();
    libyte_shift = libyte_shift << 24;
    id_pt2->length_word = id_pt2->length_word | libyte_shift;

    id_pt2->number_of_obs = takeb2();
    libyte_shift = takeb2();
    libyte_shift = libyte_shift << 8;
    id_pt2->number_of_obs = id_pt2->number_of_obs | libyte_shift;
id_ptr2->week_number = takeb2();
libyte_shift = takeb2();
libyte_shift = libyte_shift << 8;
id_ptr2->week_number = id_ptr2->week_number | libyte_shift;

second_of_week = takeb2();
libyte_shift = takeb2();
libyte_shift = libyte_shift << 8;
second_of_week = second_of_week | libyte_shift;
libyte_shift = takeb2();
libyte_shift = libyte_shift << 16;
second_of_week = second_of_week | libyte_shift;
libyte_shift = takeb2();
libyte_shift = libyte_shift << 24;
second_of_week = (second_of_week | libyte_shift);
dummy = second_of_week * 1.0;
id_ptr2->second_of_week = dummy / 100;

id_ptr2->receiver_status = takeb2();
libyte_shift = takeb2();
libyte_shift = libyte_shift << 8;
id_ptr2->receiver_status = id_ptr2->receiver_status | libyte_shift;
libyte_shift = takeb2();
libyte_shift = libyte_shift << 16;
id_ptr2->receiver_status = id_ptr2->receiver_status | libyte_shift;
libyte_shift = takeb2();
libyte_shift = libyte_shift << 24;
id_ptr2->receiver_status = id_ptr2->receiver_status | libyte_shift;

/* Decode range data from comm port 2 */
if(id_ptr2->number_of_obs < 40)
{
    for(range_decode_loop = 0; range_decode_loop < id_ptr2->number_of_obs;
        range_decode_loop++)
    {
        channel2[0].lock_time = takeb2();
        libyte_shift = takeb2();
        libyte_shift = libyte_shift << 8;
        channel2[0].lock_time = channel2[0].lock_time | libyte_shift;

        ibyte_temp = takeb2();
        channel2[0].channel_number = (ibyte_temp >> 4) + 1;

        channel2[0].tracking_state = ibyte_temp & 0x0007;
ibyte_temp = takeb2();
prn = ibyte_temp >> 2;
(channel2+prn)->PRN = prn;
onsight_sv2[range_decode_loop] = prn;

(channel2+prn)->phase_parity_known_flag = (ibyte_temp >> 1) & 1;
(channel2+prn)->phase_lock_flag = ibyte_temp & 0x0001;

(channel2+prn)->lock_time = channel2[0].lock_time;
(channel2+prn)->channel_number = channel2[0].channel_number;
(channel2+prn)->tracking_state = channel2[0].tracking_state;

laccumulated_doppler_range = takeb2();
libyte_shift = takeb2();
libyte_shift = libyte_shift << 8;
laccumulated_doppler_range = laccumulated_doppler_range | libyte_shift;
libyte_shift = takeb2();
libyte_shift = libyte_shift << 16;
laccumulated_doppler_range = laccumulated_doppler_range | libyte_shift;
libyte_shift = takeb2();
libyte_shift = libyte_shift << 24;
laccumulated_doppler_range = laccumulated_doppler_range | libyte_shift;
accumulated_doppler_range = laccumulated_doppler_range * 1.0;
(channel2+prn)->accumulated_doppler_range = accumulated_doppler_range / 256;

libyte_temp = takeb2();
l_doppler_frequency = libyte_temp >> 4;
libyte_shift = takeb2();
libyte_shift = libyte_shift << 4;
l_doppler_frequency = l_doppler_frequency | libyte_shift;
libyte_shift = takeb2();
libyte_shift = libyte_shift << 12;
l_doppler_frequency = l_doppler_frequency | libyte_shift;
libyte_shift = takeb2();
libyte_shift = libyte_shift << 20;
l_doppler_frequency = l_doppler_frequency | libyte_shift;
if((libyte_shift & 0x08000000ul) != 0)
{
    doppler_frequency =((-l_doppler_frequency & 0xFFFFFFF邬) + 1) * (-1.0);
}
else
{
    doppler_frequency = l_doppler_frequency * 1.0;
}
(channel2+prn)->doppler_frequency = doppler_frequency / 256;

lpseudorange = takeb2();
libyte_shift = takeb2();
libyte_shift = libyte_shift << 8;
lpseudorange = lpseudorange | libyte_shift;
libyte_shift = takeb2();
libyte_shift = libyte_shift << 16;
lpseudorange = lpseudorange | libyte_shift;
libyte_shift = takeb2();
libyte_shift = libyte_shift << 24;
lpseudorange = lpseudorange | libyte_shift;

libyte_temp = libyte_temp & 0x0000000Ful;
if((libyte_temp >> 3) == 1)
{
    libyte_shift = ~libyte_temp;
    libyte_shift = libyte_shift & 0x0000000Ful;
    pseudorange = (libyte_shift & 1) * pow(2,32)
        + ((libyte_shift >> 1) & 1) * pow(2,33)
        + ((libyte_shift >> 2) & 1) * pow(2,34);
    lpseudorange = ~lpseudorange;
    pseudorange = (pseudorange + lpseudorange + 1) * (-1.0);
}
else
{
    pseudorange = (libyte_temp & 1) * pow(2,32)
        + ((libyte_temp >> 1) & 1) * pow(2,33)
        + ((libyte_temp >> 2) & 1) * pow(2,34);
    pseudorange = pseudorange + lpseudorange;
}

(channel2+prn)->pseudorange = pseudorange / 128;
(channel2+prn)->trans_time = id_pt2->second_of_week -
((channel2+prn)->pseudorange/LIGHT_SPEED);
}
#include "gps.h"

void decode_ephemeris(int checksum, struct repb_data *data_repb)
{
    unsigned int uibyte_temp, uibyte_shift, i, number_of_bytes, num;
    unsigned long int ulibyte_shift, ulibyte_temp;
    int ibyte_temp, ilibyte_shift;
    long int prn, ilibyte_shift, second_of_week;
    long int ilibyte_temp;
    long int length_word, id;

    length_word = takeb();
    ilibyte_shift = takeb();
    ilibyte_shift = ilibyte_shift << 8;
    length_word = length_word | ilibyte_shift;
    ilibyte_shift = takeb();
    ilibyte_shift = ilibyte_shift << 16;
    length_word = length_word | ilibyte_shift;
    ilibyte_shift = takeb();
    ilibyte_shift = ilibyte_shift << 24;
    length_word = length_word | ilibyte_shift;

    prn = takeb();
    ilibyte_shift = takeb();
    ilibyte_shift = ilibyte_shift << 8;
    prn = prn | ilibyte_shift;
    ilibyte_shift = takeb();
    ilibyte_shift = ilibyte_shift << 16;
    prn = prn | ilibyte_shift;
    ilibyte_shift = takeb();
    ilibyte_shift = ilibyte_shift << 24;
    prn = prn | ilibyte_shift;

    data_repb[prn].prn = prn;
for(i=0;i<20;i++) ibyte_shift = takeb();

ibyte_shift = takeb();
(data_repb+prn)->tgd = (ibyte_shift << 8) * (double) pow(2,-39);
ibyte_shift = takeb();

ubyte_shift = takeb();
ubyte_shift = ubyte_shift << 8;
ubyte_temp = takeb();
ubyte_shift = ubyte_shift | ubyte_temp;
(data_repb+prn)->toc = ubyte_shift * (double) pow(2,4);

ibyte_shift = takeb();
(data_repb+prn)->af2 = (ibyte_shift << 8) * (double) pow(2,-63);

ibyte_shift = takeb();
ibyte_shift = ibyte_shift << 8;
ibyte_temp = takeb();
ibyte_temp = ibyte_temp | ibyte_shift;
(data_repb+prn)->af1 = ibyte_temp * (double) pow(2,-43);

ubyte_shift = takeb();
ubyte_shift = ubyte_shift << 16;
ubyte_temp = ubyte_shift;
ubyte_shift = takeb();
ubyte_shift = ubyte_shift << 8;
ubyte_temp = ubyte_temp | ubyte_shift;
ubyte_shift = takeb();
ubyte_temp = ubyte_temp | ubyte_shift;
(data_repb+prn)->af0 = (ubyte_temp << 10) * (double) pow(2,-41);

for(i=0;i<7;i++) ibyte_shift = takeb();

ibyte_shift = takeb();
ibyte_shift = ibyte_shift << 8;
ibyte_temp = takeb();
ibyte_temp = ibyte_temp | ibyte_shift;
(data_repb+prn)->crs = ibyte_temp * ((double) pow(2,-5));

ibyte_shift = takeb();
ibyte_shift = ibyte_shift << 8;
ibyte_temp = takeb();
ibyte_temp = ibyte_temp | ibyte_shift;
(data_repb+prn)->deltan = ibyte_temp * ((double) pow(2,-43));

libyte_shift = takeb();
libyte_shift = libyte_shift << 24;
libyte_temp = libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 16;
libyte_temp = libyte_temp | libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 8;
libyte_temp = libyte_temp | libyte_shift;
libyte_shift = takeb();
libyte_temp = libyte_temp | libyte_shift;
(data_repb+prn)->mo = libyte_temp * ((double) pow(2,-31));

ibyte_shift = takeb();
ibyte_shift = ibyte_shift << 8;
ibyte_temp = takeb();
ibyte_temp = ibyte_temp | ibyte_shift;
(data_repb+prn)->mo = libyte_temp * ((double) pow(2,-29));

ulibyte_shift = takeb();
ulibyte_shift = ulibyte_shift << 24;
ulibyte_temp = ulibyte_shift;
ulibyte_shift = takeb();
ulibyte_shift = ulibyte_shift << 16;
ulibyte_temp = ulibyte_temp | ulibyte_shift;
ulibyte_shift = takeb();
ulibyte_shift = ulibyte_shift << 8;
ulibyte_temp = ulibyte_temp | ulibyte_shift;
ulibyte_shift = takeb();
ulibyte_temp = ulibyte_temp | ulibyte_shift;
(data_repb+prn)->eccen = ulibyte_temp * ((double) pow(2,-33));

ibyte_shift = takeb();
ibyte_shift = ibyte_shift << 8;
ibyte_temp = takeb();
ibyte_temp = ibyte_temp | ibyte_shift;
(data_repb+prn)->cus = ibyte_temp * ((double) pow(2,-29));

ulibyte_shift = takeb();
ulibyte_shift = ulibyte_shift << 24;
ulibyte_temp = ulibyte_shift;
ulibyte_shift = takeb();
ulibyte_shift = ulibyte_shift << 16;
ulibyte_temp = ulibyte_temp | ulibyte_shift;
ulibyte_shift = takeb();
ulibyte_shift = ulibyte_shift << 8;
ulibyte_temp = ulibyte_temp | ulibyte_shift;
ulibyte_shift = takeb();
ulibyte_temp = ulibyte_temp | ulibyte_shift;
(data_repb+prn)- > sqrsma = ulibyte_temp * ((double) pow(2,-19));

ulibyte_shift = takeb();
ulibyte_shift = ulibyte_shift << 8;
ulibyte_temp = takeb();
ulibyte_temp = ulibyte_temp | ulibyte_shift;
(data_repb+prn)- > toe = ulibyte_temp * ((double) pow(2,4));

for(i=0;i<7;i++) {ibyte_shift = takeb();
ibyte_shift = takeb();
ibyte_shift = ibyte_shift << 8;
ibyte_temp = takeb();
ibyte_temp = ibyte_temp | ibyte_shift;
(data_repb+prn)- > cic = ibyte_temp * ((double) pow(2,-29));
}

libyte_shift = takeb();
libyte_shift = libyte_shift << 24;
libyte_temp = libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 16;
libyte_temp = libyte_temp | libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 8;
libyte_temp = libyte_temp | libyte_shift;
libyte_shift = takeb();
libyte_temp = libyte_temp | libyte_shift;
(data_repb+prn)- > omegao = libyte_temp * ((double) pow(2,-31));

ibyte_shift = takeb();
ibyte_shift = ibyte_shift << 8;
ibyte_temp = takeb();
ibyte_temp = ibyte_temp | ibyte_shift;
(data_repb+prn)- > cis = ibyte_temp * (pow(2,-29));
libyte_shift = takeb();
libyte_shift = libyte_shift << 24;
libyte_temp = libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 16;
libyte_temp = libyte_temp | libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 8;
libyte_temp = libyte_temp | libyte_shift;
libyte_shift = takeb();
libyte_temp = libyte_temp | libyte_shift;
(data_rep+prn)->io = libyte_temp * (double) pow(2,-31);

libyte_shift = takeb();
libyte_shift = libyte_shift << 8;
libyte_temp = takeb();
libyte_temp = libyte_temp | libyte_shift;
(data_rep+prn)->crc = libyte_temp * (double) pow(2,-5);

libyte_shift = takeb();
libyte_shift = libyte_shift << 24;
libyte_temp = libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 16;
libyte_temp = libyte_temp | libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 8;
libyte_temp = libyte_temp | libyte_shift;
libyte_shift = takeb();
libyte_temp = libyte_temp | libyte_shift;
(data_rep+prn)->omega = libyte_temp * (double) pow(2,-31);

libyte_shift = takeb();
libyte_shift = libyte_shift << 16;
libyte_temp = libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 8;
libyte_temp = libyte_temp | libyte_shift;
libyte_shift = takeb();
libyte_temp = libyte_temp | libyte_shift;
(data_rep+prn)->omgdot = (libyte_temp << 8) * (double) pow(2,-51);

ibyte_shift = takeb();
ibyte_shift = ibyte_shift << 8;
ibyte_temp = takeb();
ibyte_temp = ibyte_temp | ibyte_shift;
(data_repb+prn)->idot = (ibyte_temp < 2) * ((double) pow(2,-45));
}
#include "gps.h"

void ephcal(double *dtsv, int onsight-sv[40], double satellite_position[40][4],
struct rqgb_id_word *id_pt, struct repb_data data_repb[40])
{
    double grav, eanvel, fclk, epsi, pi;
    double k, tmod, smx, no, n, mk, ea, eadiff, ek, oneme2, coea, xvk, yvk, phik, phik2, ds, dc,
    delik, delrk, deluk, uk, ik, rk, vk, xkprim, ykprim, dt, omegak, eatemp;
    int i, j, itmax;
    long int iters, ierr;

    /* wgs84 parameters are used for the grav and eanvel. */
    grav = 3.986005e+14;
    /* earth angular velocity */
    eanvel = 7.2921151467e-5;
    /* earth's universal gravitational constant */
    fclk = -4.442807633e-10;
    /* accuracy limit */
    epsi = 1.0e-13;
    /* max number if iterations */
    itmax = 8;
    pi = 3.1415926535898e0;

    for(i=0; i < id_pt->number_of_obs; i + +)
    {
        ierr = 0;
        /* check for begin and/or end of week cross-over */
        tk = channel[onsight_sv[i]].trans_time - data_repb[onsight_sv[i]].toe;
        if(tk > 302400.0e0) tk = tk - 604800.0e0;
        if(tk < -302400.0e0) tk = tk + 604800.0e0;
        tmod = channel[onsight_sv[i]].trans_time - data_repb[onsight_sv[i]].toc;
        if(tmod > 302400.0e0) tmod = tmod - 604800.0e0;
        if(tmod < -302400.0e0) tmod = tmod + 604800.0e0;
        smx = data_repb[onsight_sv[i]].sqrsmas * data_repb[onsight_sv[i]].sqrsmas;
        if(smx != 0.0e0)
{ 
  no = (double)sqrt(grav/((double)pow(smx,3)));
  n = no + (data_repb[onsight_sv[i]].deltan * pi);
  mk = (data_repb[onsight_sv[i]].mo * pi) + (n * tk);
  /* compute the eccentric anomaly from the mean anomaly and the */
  /* eccentricity. use up to itmax iterations, stop at desired */
  /* accuracy (espi) is reached. ( 3 iterations will reach it. */
  iters = 0;
  eatemp = mk;
  ea = mk + (data_repb[onsight_sv[i]].eccen * (double)sin(eatemp));
  iters = iters + 1;
  eadiff = ea - eatemp;
  while((fabs(eadiff) >= epsi) && (iters <= itmax))
  {
    ea = mk + (data_repb[onsight_sv[i]].eccen * (double)sin(eatemp));
    iters = iters + 1;
    eadiff = ea - eatemp;
    eatemp = ea;
  }
  ek = ea;

  /* compute the true anomaly */
  oneme2 = (double)sqrt(1.0e0 - (data_repb[onsight_sv[i]].eccen *
                           data_repb[onsight_sv[i]].eccen));
  coea = (double)cos(ek);
  xvk = (coea - (data_repb[onsight_sv[i]].eccen) / (1.0e0 -
                          ((data_repb[onsight_sv[i]].eccen) * coea));
  yvk = (oneme2 * (double)sin(ek)) / (1.00e0 - (data_repb[onsight_sv[i]].eccen *
                           coea));
  vk = (double) atan2(yvk,xvk);
  phik = vk + (data_repb[onsight_sv[i]].omega * pi);
  phik2 = phik + phik;

  /* compute 2nd harmonic perturbations */
  ds = (double) sin((double) phik2);
  dc = (double) cos((double) phik2);
  deluk = (data_repb[onsight_sv[i]].cus * ds) + (data_repb[onsight_sv[i]].cuc * dc);
  delrk = (data_repb[onsight_sv[i]].crc * dc) + (data_repb[onsight_sv[i]].crs * ds);
  delik = (data_repb[onsight_sv[i]].cic * dc) + (data_repb[onsight_sv[i]].cis * ds);
  uk = phik + deluk;
  ik = (data_repb[onsight_sv[i]].io * pi) + delik + (data_repb[onsight_sv[i]].idot * tk
        * pi);
  rk = smx * (1 - (data_repb[onsight_sv[i]].eccen * coea)) + delrk;
  
  /* ... */
xkprim = rk * (double) cos((double) uk);
ykprim = rk * (double) sin((double) uk);

/* corrected longitude of the ascending node */
omegak = (data_repb[onsight_sv[i]].omegao * pi)
    + (((data_repb[onsight_sv[i]].omgdot * pi) - eanvel) * tk)
    - (eanvel * data_repb[onsight_sv[i]].toe);

/* satellite ECEF coordinates (meters) */
ds = (double) sin((double) omegak);
dc = (double) cos((double) omegak);
satellite_position[onsight_sv[i]][0] = (xkprim * dc) - (ykprim * (double) cos((double) ik) * ds));
satellite_position[onsight_sv[i]][1] = ((xkprim * ds) + (ykprim * (double) cos((double) ik) * dc));
satellite_position[onsight_sv[i]][2] = (ykprim * (double) sin((double) ik));

/* compute clock offset */
dtr = fclk * data_repb[onsight_sv[i]].eccen
    * data_repb[onsight_sv[i]].sqrsmas * (double)sin(ek);
dtsv = data_repb[onsight_sv[i]].af0
    + (data_repb[onsight_sv[i]].af1 * tmod)
    + (data_repb[onsight_sv[i]].af2 * tmod * tmod) + dtr;
dtsv = *dtsv - data_repb[onsight_sv[i]].tgd;
}
else
{
printf("\n ephcal: sqrsmas = 0.0.");
}
}
}
#include "gps.h"
void pseudorange(int onsight_sv[40], double estimated_xyzb[4],
                 double svecef_matrix[40][4], struct rqgb_id_word *id_pt,
                 double estimated_pseudorange[40])
{
    int i;
    for(i=0;i < id_pt->number_of_obs;i++)
    {
        estimated_pseudorange[onsight_sv[i]] =
            (double) sqrt((double) pow((estimated_xyzb[0] - svecef_matrix[onsight_sv[i]][0]),2) +
                            (double) pow((estimated_xyzb[1] - svecef_matrix[onsight_sv[i]][1]),2) +
                            (double) pow((estimated_xyzb[2] - svecef_matrix[onsight_sv[i]][2]),2) +
                            (estimated_xyzb[3]));
    }
}
#include "gps.h"
void transpose_matrix(double matrix[20][20], double matrix_t[20][20], int row, int column)
{
    int i, j;
    for(i=0; i < column; i++)
    {
        for(j=0; j < row; j++)
        {
            matrix_t[i][j] = matrix[j][i];
        }
    }
}
#include "gps.h"
void matrix_multiply(double m1[20][20], double m2[20][20], double mr[20][20],
int n, int m, int o)
{  
  int i, j, k;
  for(i = 0; i < 20; i++)
    for(j = 0; j < 20; j++)
    {
      mr[i][j] = 0.0;
    }
  for(i = 0; i < n; i++)
  {
    for(j = 0; j < o; j++)
    {
      for(k = 0; k < m; k++) mr[i][j] = mr[i][j] + m1[i][k]*m2[k][j];
    }
  }
// Description: Inverse a square matrix

/* Input format: inverse_matrix(a,n,b,m)
   Output: b
   Argument: 1st matrix = function matrix
   2nd matrix = solution matrix, m = dimension of matrix
   Answer stores in solution matrix
   Author: Lap Ho
   Date: November, 1994
*/

#include "gps.h"
void inverse_matrix(double a[20][20],int n,double b[20][20],int m)
{
    int indxc[20],indxr[20],ipiv[20];
    int i,icol,irow,j,k,l,ll;
    double tem1,big,dum,pivinv,temp;

    for(j=0;j<n;j++) ipiv[j]=0;
    for(i=0;i<n;i++)
    {
        big=0.0;
        for(j=0;j<n;j++)
            if(ipiv[j] != 1)
                for(k=0;k<n;k++)
                {
                    if(ipiv[k] == 0)
                    {
                        if(fabs(a[j][k]) >= big)
                        {
                            big=fabs(a[j][k]);
                            irow=j;
                            icol=k;
                        }
                    } else if (ipiv[k] > 1) nerror("singular matrix 1");
                    ++(ipiv[icol]);
                    if(irow != icol)
                    {
                        for(l=0;l<n;l++) SWAP(a[irow][l],a[icol][l])
                        for(l=0;l<m;l++) SWAP(b[irow][l],b[icol][l])
                    }
    }
indx[i] = irow;
indx[i] = icol;
if (a[icol][icol] == 0.0) nerror("singular matrix 2");
pivinv = 1.0/a[icol][icol];
a[icol][icol] = 1.0;
for (l = 0; l < n; l++) a[icol][l] *= pivinv;
for (l = 0; l < m; l++) b[icol][l] *= pivinv;
for (l = 0; l < n; l++)
  if (l != icol)
    {
      dum = a[l][icol];
      a[l][icol] = 0.0;
      for (l = 0; l < n; l++) a[l][l] -= a[icol][l]*dum;
      for (l = 0; l < m; l++) b[l][l] -= b[icol][l]*dum;
    }
for (l = n; l > 0; l--)
  {
    if (indx[l] != indx[l]) SWAP(a[k][indx[l]], a[k][indx[l]]);
  }
free((char*)ipiv + 1);
free((char*)(indx + 1));
free((char*)(indx + 1));
#include "gps.h"
void nerror(char *error_text)
{
    void exit();
    printf("run-time error...
");
    printf("%s\n", error_text);
    printf("...now exiting to system...
");
    exit(1);
}
#include "gps.h"

void single_difference(int onsight_sv[40], struct rqgb_id_word *id, struct rqgb_range_record channel[40], struct rqgb_range_record channel2[40], double single[10])
{
    int i;
    double acc1,acc2;

    for(i=0; i < id->number_of_obs; i++)
    {
        single[i] = (channel[onsight_sv[i]].accumulated_doppler_range
                      - channel2[onsight_sv[i]].accumulated_doppler_range);
    }
}
/** Description: Double-difference formation */
/** Input format: double_difference(onsight_sv,&id_name,single,double_d */
/** Output: Double_d */
/** Author: Lap Ho */
/** Date: November, 1994 */
/*
#include "gps.h"
void double_difference(int onsight_sv[40],struct rqgb_id_word *id_pt,
double single[10],double double_d[10])
{
int i;
double p;

/* in this case the first singale different is pick */
/* should use DOP to fine the best one as the base single differenced */
for(i=0;i < id_pt->number_of_obs -1;i++)
{
    double_d[i] = (single[0] - single[i+1]) ;

    if(double_d[i] > 0)
    {
        p = floor(double_d[i]);
        double_d[i] = (double_d[i] - p) * 0.190293673;
    }
    else
    {
        p = ceil(double_d[i]);
        double_d[i] = (double_d[i] - p) * 0.190293673;
    }
}
*/
#include "gps.h"

void unit_vector(double user_position[4], double satellite_position[40][4],
                 struct rqgb_id_word *id_pt, struct rqgb_range_record channel[40],
                 double u_vector[10][3], int on sight_sv[20])
{
  int i;
  double x, y, z;

  for(i=0;i<id_pt->number_of_obs;i++)
  {
    x = satellite_position[on sight_sv[i]][0] - user_position[0];
    y = satellite_position[on sight_sv[i]][1] - user_position[1];
    z = satellite_position[on sight_sv[i]][2] - user_position[2];

    u_vector[i][0] = x/sqrt(pow(x,2)+pow(y,2)+pow(z,2));
    u_vector[i][1] = y/sqrt(pow(x,2)+pow(y,2)+pow(z,2));
    u_vector[i][2] = z/sqrt(pow(x,2)+pow(y,2)+pow(z,2));
  }
}
I* ID unit.c I*

DESCRIPTION: Double difference unit vector
I*

INPUT FORMAT: double_unit_vector(&id_name,u_vector,d_unit)
I*

OUTPUT: d_unit
I*

AUTHOR: Lap Ho
I*

DATE: November, 1994
I*

#include "gps.h"

void double_unit_vector(struct rqgb_id_word *id, double u_vector[10][3],
double d_unit[10][3])
{
int i;

for(i=0; i < id->number_of_obs - 1; i++)
{
    d_unit[i][0] = u_vector[0][0] - u_vector[i+1][0];
    d_unit[i][1] = u_vector[0][1] - u_vector[i+1][1];
    d_unit[i][2] = u_vector[0][2] - u_vector[i+1][2];
}
}
#include "gps.h"

int send_out()
{
    int stop_run;
    stop_run = 0;
    stop_run = go_stop();
    if(stop_run != 0)
    {
        outcom1(stop_run);
        outcom2(stop_run);
    }

    /* if Ctrl-c or tab was pressed, terminate the loop */
    if((stop_run == 9) || (stop_run == 3)) return FALSE;
    else return TRUE;
}
#include "gps.h"
long int header1(char *checksum)
{
    int seek, head, seekdelta, count;
    unsigned long int seek_shift, id_word, libyte_shift;

    /* make sure at least a complete set of data is reside in ring buffer */
    seekdelta = deltab();

    /* if the input pointer is less than output pointer, SIZE of
    ring buffer must add to input pointer */
    if(seekdelta < 0) seekdelta = seekdelta + 10000;

    /** the max bytes of one set of data is less than 200, should wait until buffer filled **/
    /** with one set of data than decode. **/
    /** reason: 1 save time **/
    /** 2 prevent hard drive over work **/
    if(seekdelta >= 200)
    {
        seek_shift = 0;
        count = 0;
        while((seek_shift != 0x00AA4411ul) && (count < 1000))
        {
            seek = takeb();
            seek_shift = seek_shift << 8;
            seek_shift = seek_shift | seek;
            seek_shift = seek_shift & 0x00FFFFFFul;
            count += 1;
        }
        if(count == 1000) printf("Endless loop\n\n");

        *checksum = takeb();
        id_word = takeb();
        libyte_shift = takeb();
libyte_shift = libyte_shift << 8;
id_word = id_word | libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 16;
id_word = id_word | libyte_shift;
libyte_shift = takeb();
libyte_shift = libyte_shift << 24;
id_word = id_word | libyte_shift;
return id_word;
/* Header2.c */
/* Description: Search for header of messages from comm2 */
/* Input format: header2(checksum2) */
/* Output: Identity number of each set of data */
/* Author: Lap Ho */
/* Date: November, 1994 */

#include "gps.h"

long int header2(char *checksum2)
{
    int seek2, head2, seekdelta2;
    long int seek_shift2, id_word2, libyte_shift;

    /* start to search for header only one complete set of data has come into comm port */
    seekdelta2 = deltab2();
    if(seekdelta2 < 0) seekdelta2 = seekdelta2 + 10000;

    /* 200 for 2ab */
    if(seekdelta2 >= 200)
    {
        seek_shift2 = 0;
        while(seek_shift2 != 0x00AA4411ul)
        {
            seek2 = takeb2();
            seek_shift2 = seek_shift2 << 8;
            seek_shift2 = seek_shift2 | seek2;
            seek_shift2 = seek_shift2 & 0x00FFFFFFul;
        }
    }
    *checksum2 = takeb2();
    id_word2 = takeb2();
    libyte_shift = takeb2();
    libyte_shift = libyte_shift << 8;
    id_word2 = id_word2 | libyte_shift;
    libyte_shift = takeb2();
    libyte_shift = libyte_shift << 16;
    id_word2 = id_word2 | libyte_shift;
    libyte_shift = takeb2();
    libyte_shift = libyte_shift << 24;
    id_word2 = id_word2 | libyte_shift;
    return id_word2;
#include "gps.h"

point_position(struct rqgb_id_word *id_pt, struct rqgb_range_record channel[40],
double user_position[4],double satellite_position[40][4], int *onsight_sv)
{
  int i,j,row,column4,column1,column3;
  double estimated_pseudorange[40];
  double soluvector[20][20];
  double delta_pseudorange[40];
  double h_matrix[20][20];
  double temp[20][20],multi[20][20];
  column4 = 4;
  column1 = 1;
  column3 = 3;

  for(i=0;i<id_pt->number_of_obs;i++)
  {
    estimated_pseudorange[onsight_sv[i]] =
      (double)sqrt((double)pow((user_position[0] - satellite_position[onsight_sv[i]][0]),2)
    + (double)pow((user_position[1] - satellite_position[onsight_sv[i]][1]),2)
    + (double)pow((user_position[2] - satellite_position[onsight_sv[i]][2]),2))
    + (user_position[3]);
  }

  for(i=0;i<id_pt->number_of_obs;i++)
    delta_pseudorange[onsight_sv[i]] = channel[onsight_sv[i]].pseudorange
      - estimated_pseudorange[onsight_sv[i]];  

  for(i=0;i<id_pt->number_of_obs;i++)
    {
      for(j=0;j<3;j++)
        h_matrix[i][j] = (user_position[j] - satellite_position[onsight_sv[i]][j])
          ...
for(i=0;i<id_table.number_of_obs;i++) h_matrix[i][3] = 1.0;
row = id_table->number_of_obs;
transpose_matrix(h_matrix,temp,row,column4);
matrix_multiply(temp,h_matrix,multi,column4,row,column4);
for(i=0;i<20;i++)
{
    for(j=0;j<20;j++)
    {
        if(i == j)
            soluvector[i][j] = 1.0;
        else
            soluvector[i][j] = 0.0;
    }
}
inverse_matrix(multi,column4,soluvector,column4);
matrix_multiply(soluvector,temp,multi,column4,column4,row);
for(i=0;i<id_table->number_of_obs;i++)
    soluvector[i][0] = delta_pseudorange[onsight_sv[i]];
matrix_multiply(multi,soluvector,temp,column4,column4,1);
for(i=0;i<column4;i++) user_position[i] = user_position[i] + temp[i][0];
}
/*
 * Baseline.c
 * Description: Using least square solution to obtain baseline vector.
 * Input format: short_baseline(&id_name,d_unit,double_d)
 * Output: Baseline coordinates. store in temp array
 * Author: Lap Ho
 * Date: November, 1994
 */

#include "gps.h"
short_baseline(struct rqgb_id_word *id_pt, double d_unit[10][3], double double_d[10])
{
    FILE *fpt;
    int i,j,row,column3,column4,column1;
    double soluvector[20][20];
    double h_matrix[20][20];
    double temp[20][20],multi[20][20];
    double orgerce[3],plh[3],xyz[4],enu[3];

    column4 = 4;
    column1 = 1;
    column3 = 3;

    for(i=0;i<20;i++)
    {
        for(j=0;j<20;j++)
        {
            h_matrix[i][j] = 0.0;
            temp[i][j] = 0.0;
            multi[i][j] = 0.0;
        }
    }

    /* set up H matrix */
    for(i=0;i<(id_pt->number_of_obs -1);i++)
    {
        for(j=0;j<3;j++)
        {
            h_matrix[i][j] = d_unit[i][j];
        }
        row = id_pt->number_of_obs - 1;
    }
/* baseline = ((Ht*H)-1 * Ht) * DD */
transpose_matrix(h_matrix,temp,row,column3);

matrix_multiply(temp,h_matrix,multi,column3,row,column3);

for(i=0;i < 20;i + +)
{
    for(j =0;j < 20;j ++)
    {
        if(i == j)
            {
                soluvector[i][j] = 1.0e0;
            }
        else
            {
                soluvector[i][j] = 0.0e0;
            }
    }
}

inverse_matrix(multi,column3,soluvector,column3);

matrix_multiply(soluvector,temp,multi,column3,column3,row);

for(i=0;i < (id_pt->number_of_obs -1);i ++)
    soluvector[i][0] = double_d[i];

matrix_multiply(multi,soluvector,temp,column3,row,1);
/* Description: Converts ECEF to Latitude, Longitude and Height */
/* Input format: ec2llh(xyz, plh) */
/* Output: plh */
/* Author: Lap Ho */
/* Date: November, 1994 */

#include "gps.h"

void ec2llh(double xyz[4], double plh[3])
{
    double a, cl, cp, e, el, esq, gsq, h, p, r, rsq, sl, sp, x, y, z, zp;
    int i;
    double epsilon;
    i = 0;
    a = 6378137.0;
    e = 8.1819190842622e-2;
    esq = e*e;
    x = xyz[0];
    y = xyz[1];
    z = xyz[2];
    rsq = (x*x)+(y*y);
    h = esq*z;
    zp = z + h;
    r = sqrt(rsq + (zp*zp));
    sp = zp/r;
    gsq = 1.0 - (esq*sp*sp);
    en = a/sqrt(gsq);
    p = en*esq*sp;
    epsilon = fabs(h-p);
    while( (epsilon > 5.0e-4) && (i < 6) )
    {
        h = p;
        zp = z + h;
        r = sqrt(rsq + (zp*zp));
        sp = zp/r;
        gsq = 1.00 - (esq*sp*sp);
        en = a/sqrt(gsq);
        p = en*esq*sp;
        epsilon = fabs(h-p);
        i++;
    }
\[ p = \text{atan2}(zp, \sqrt{rsq}); \]
\[ h = r - en; \]
\[ \text{plh}[0] = p; \]
\[ \text{plh}[1] = \text{atan2}(y, x); \]
\[ \text{plh}[2] = h; \]

\[ \text{plh1} = \text{plh}[0] \times 180.0 / 3.141592654; \]
\[ \text{plh2} = \text{plh}[1] \times 180.0 / 3.141592654; \]
Description: Calculate one point respect to other point in ENU coordinate.

Input format: ec2enu(xyz, enu, orgence, orgllh)

Output: enu.

Author: Lap Ho

Date: November, 1994

#include "gps.h"

void ec2enu(double xyz[3], double enu[3], double orgence[3], double orgllh[3])
{
    double difece[3];
    double sla, cla, slo, clo;
    long int i;
    for(i=0; i<3; i++)
    {
        difece[i] = xyz[i] - orgence[i];
    }
    sla = sin(orgllh[0]);
    cla = cos(orgllh[0]);
    slo = sin(orgllh[1]);
    clo = cos(orgllh[1]);

    enu[0] = (-slo * difece[0]) + (clo * difece[1]);
    enu[1] = (-sla * clo * difece[0]) + (-sla * slo * difece[1])
            + (cla * difece[2]);
    enu[2] = (cla * clo * difece[0]) + (cla * slo * difece[1])
            + (sla * difece[2]);
}
#include "gps.h"

double enu2ae(double enu[3])
{
    double x, angle;
    double az, ele;
    x = sqrt(pow(enu[0], 2) + pow(enu[1], 2));
    if (x != 0.0) ele = atan2(enu[2], x);
    if (enu[1] != 0.0) az = atan2(enu[0], enu[1]);
    az = az * (180 / 3.141592654);
    ele = ele * (180 / 3.141592654);
    return(ele); /* if return azimuth angle, return(az)*/
}
7.2 Appendix 2

Assembly code for serial port communication.

```assembly
page 60,132
title interrupt driven serial ports
comment

 قائلاً: Lap Ho
* date: November, 1994
* serial port communication(interrupt driven)
* subroutines and its function
* for compiler Microsoft MASM 6.0
* - setint(): set com1 receive com2 sending
* - serint(): IRQ polling routine for setint()
* - s_null(): send character to com2 to initiate empty THR interrupt
* - setinr(): set both com1 & com2 receive only
* - serinr(): IRQ polling routine for setinr()
* - clsint(): close com1 and com2 to its original status
* - go_stop(): bios to read keyboard input without waiting
* - sum_by(): return sum of byte value received from com1
* - cnt_by(): return # of byte received from com1
* - sum_by2(): return sum of byte value received from com1
* - cnt_by2(): return # of byte received from com1
* - count_1(): count total byte value & # of byte received from com1
* - count_2(): count total byte value & # of byte received from com2
* - setbps(rate1,rate2) set baud rate of comports
* - putb(al): put byte received from comport to ring buffer1
* - putb_2(al): put byte received from comport to ring buffer2
* - getb(al): get byte from ring buffer1 & send it out to com2
* - append_b(char): put one character into ring buffer1
* - append_2(char): put one character into ring buffer2
* - outcom1(): sent one character to comport 1
* - outcom2(): sent one character to comport 2
* - tranb(): transfer one byte from ring buffer 1 to C
* - takeb(): transfer one byte from ring buffer 1 to c with check
* - takeb2(): transfer one byte from ring buffer 2 to c with check
* - display_1(): display content of ring buffer 1
* - display_2(): display content of ring buffer 2
* - deltab(): number of bytes reside in ring buffer 1
* - deltab2(): number of bytes resides in ring buffer 2
```
* - cl_ring(): clear ring buffer 1
* - ci_ring2(): clear ring buffer 2
* ring buffer 1 for data comes in from comport 1
* ring buffer 2 for data comes in from comport 2


| ;--- com port addresses
| recb_1 equ 03f8h ;com1 receive buffer
| recb_2 equ 02f8h ;com2 receive buffer
| thr_1 equ 03f8h ;com1 transmit holding register
| thr_2 equ 02f8h ;com2 transmit holding register
| inte_1 equ 03f9h ;com1 interrupt enable register
| inte_2 equ 02f9h ;com2 interrupt enable register
| iir_1 equ 03fah ;com1 interrupt identification register
| iir_2 equ 02fah ;com2 interrupt identification register
| lsr_1 equ 03fdh ;com1 line status register
| lsr_2 equ 02fdh ;com2 line status register
| msr_1 equ 03feh ;com1 modem status register
| msr_2 equ 02feh ;com2 modem status register
| mcr_1 equ 03fch ;com1 modem control register
| mcr_2 equ 02fch ;com2 modem control register
| bufsize equ 10000 ;ring buffer size

.286 ;run on 286 cpu
.model large,c ;interfacing with C, memory model large
.stack ;size of stack (default)
data ;data area
buff db bufsize dup("!") ;ring buffer1 usually for com1
bufst dd buff
bufend dd buff+bufsize
dumy2 dd 1 dup("_")
bufin dw 00 ;buffer1 input pointer
bufout dw 00 ;buffer1 output pointer
buff_2 db bufsize dup("!") ;ring buffer2 usually for com2
bufst_2 dd buff_2
bufend_2 dd buff_2+bufsize
dumy2_2 dd 1 dup("_")
bufin_2 dw 00 ;buffer2 input pointer
bufout_2 dw 00 ;buffer2 output pointer
sav_4 dw 2 dup(?) ;reserve for com1 interrupt vector
sav_3 dw 2 dup(?) ;reserve for com2 interrupt vector
i_max dd 115200 ;max of baud rate, use as a divisor
;total value of byte received from com1
sum_1 dd 00

;total byte received from com1
cnt_1 dd 00

;total value of byte received from com2
sum_2 dd 00

;total byte received from com2
cnt_2 dd 00

.code

;code area

section .data
page 60,132

section .text

public s_null

(comment|

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

* function: sent the first character to com2 to turn on transmit holding register empty *
* interrupt *
* call: s_null() *
* return: none *

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

| s_null proc far
pusha
cli ; disable interrupt
mov ax, bufin ; check buffer1 if it is empty
mov bx, bufout
cmp ax, bx
je no_s ; yes, nothing to send.
mov dx, lsr_2 ; check line status register if
in al, dx ; holding register is empty
and al, 20h
jz no_s ; no, do not sent
mov ax, bx ; is output pointer reaching the end
cmp ax, bufsize ; of buffer?
jne next_5
push ds ; yes, get the character in last space
mov ax, seg buff ; of buffer
mov ds, ax
mov dx, recb_2 ; get address of holding register
mov al, ds: [bx + buff]
out dx, al ; sent the character out
mov bx, 0 ; reset the pointer to the beginning of
jmp nex_4 ; buffer
next_5:
push ds ; no, get the character from buffer
mov ax, seg buff
mov ds, ax
mov dx, recb_2 ; get address of holding register
|
mov    al,ds:[bx+buff]
out    dx,al ;send the character out
inc    bx ;increment the output pointer by 1
nex_4: mov    bufout,bx ;reset the output pointer
        pop    ds
no_s:   sti   ;enable interrupt
        popa
        ret
s_null endp

page 60,132
public setint

comment

*******************************************************************************
* function: initialize serial port          *
* -set transmit rate, parity, stop bit and # of bits/char       *
* -save IRQ4(com1) and IRQ3(com2) interrupt vectors             *
* -replace interrupt IRQ4 and IRQ3 vectors within ours          *
* -enable interrupt, data available interrupt for com1 and empty transmit holding *
* register interrupt for com2 on UART 8250                      *
* -enable modem general interrupt bit                         *
* -enable interrupt controller for IRQ4 & IRQ3(active low)      *
*******************************************************************************

setint proc far
pusha
cli

;--- set speed, parity, stop bit and # of bits per char
mov    dx,0 ;com1
mov    al,73h ;9600 bps
mov    ah,0 ;int severe 0
int    14h ;call int 14h to set it
mov    dx,1 ;com2
mov    al,73h ;9600 bps
mov    ah,0 ;int severe 0
int    14h ;call int 14h to set it

;--- save original interrupt vectors
xor    ax,ax
mov    si,ax ;make si to 0 for index addressing mode
mov    es,si ;use es to access low memory
mov    ax,es:[si+2ch] ;get IRQ3 offset
mov    [si+sav_3],ax ;save it
mov    ax,es:[si+2eh] ;get IRQ3 segment address
mov [si+sav_3+2],ax ;save it
mov ax,es:[si+30h] ;get IRQ4 offset
mov [si+sav_4],ax ;save it
mov ax,es:[si+32h] ;get IRQ4 segment address
mov [si+sav_4+2],ax ;save it

;--- place our own interrupt vectors
mov ax,offset serint ;get offset of serint
mov es:[si+2ch],ax ;put it in both IRQ3 and IRQ4
mov es:[si+30h],ax
mov ax,seg serint ;get segment address of serint
mov es:[si+2eh],ax ;put it in both IRQ3 and IRQ4
mov es:[si+32h],ax

;--- enable modem general interrupt bit
mov dx,mcr_1 ;get com1 modem control register
in al,dx
or al,8 ;set interrupt bit on
out dx,al
mov dx,mcr_2 ;get com2 modem control register
in al,dx
or al,8 ;set interrupt bit on
out dx,al

;--- enable UART 8250
mov dx,inte_1 ;get com1 interrupt enable register
mov al,1 ;set data available interrupt on
out dx,al
mov dx,inte_2 ;get com2 interrupt enable register
mov al,2 ;set empty transmit holding register
out dx,al ;interrupt on

;--- enable 8259 interrupt controller
in al,21h
mov ah,0e7h ;set IRQ4 and IRQ3 interrupt on
and al,ah
out 21h,al
sti
popa
ret

setint endp

page 60,132
public serint

comment

* interrupt service routine for intset(). rec/send *
* -identify what kind of interrupt has occurred
* -call appropriate service routine for the interrupt
* -turn off interrupt flag
* -send EOI. end of interrupt to CPU and IRQ controller

```
serint proc far
pusha
mov dx,iir_1 ;get com1 interrupt identification
in al,dx ;register
mov ah,al
cmp ah,1 ;has any interrupt occurred in com1?
je inexint ;no, it MUST com2
mov dx,recb_1 ;yes, get com1 receiver buffer
in al,dx ;input char in al
call far ptr putb ;call putb subroutine to put it to ring
;jmp iendint ;buffer
inexint: mov dx,iir_2 ;get com2 interrupt identification
in al,dx ;register
mov ah,al ;interrupt occurred?
cmp ah,1
je iendint ;no, exit
call far ptr getb ;yes, call getb subroutine to send char
iendint: mov al,20h ;to ring buffer to serial port
out 20h,al ;send end of interrupt to controller
popa
iret ;inform /CPU interrupt service is over
serint endp
```

```
page 60,132
public go_stop ;public, can be called externally
```

```
* function: get a byte from input device, usually keyboard it will not wait for input.
* good for "while" loop doing something else while getting key input
* call: go_stop()
* return: go_stop(); itself is a value. normally store in ax
```

```
go_stop proc far ;return address in seg:offset form
push dx ;4 bytes has been pushed to stack
mov dl,255 ;255:input only
mov ah,06 ;severce 06
```
```assembly
int 21h
mov ah,0 ; make sure return value is 0xal
pop dx ; the byte get will store in al but
ret ; return value is ax

; make sure return value is Oxal
; the byte get will store in a1
; return value is ax

; function: return the value of sum_1. it is a double word
; return value in dx:ax from. a double word value, 2nd word store in upper 2 bytes
; 1st word store in lower 2 bytes
; call: sum_by()
; return: sum_by() dx:ax

sum_by proc far
    mov dx,word ptr sum_1+2 ; lower word in dx
    mov ax,word ptr sum_1   ; upper owrd in ax
    ret
sum_by endp

; function: return the value of cnt_by
; call: cnt_by()
; return: cnt_by() in form dx:ax

cnt_by proc far
    mov dx,word ptr cnt_1+2
    mov ax,word ptr cnt_1
    ret
cnt_by endp

; function: return the value of sum_by2
```
* call: sum_by2()  
* return: sum_by2() in the form dx:ax

sum_by2 proc far
mov dx,word ptr sum_2+2
mov ax,word ptr sum_2
ret
sum_by2 endp

page 60,132
public cnt_by2

comment|  
* function: return the value of cnt_by2  
* call: cnt_by2  
* return: cnt_by2 in the form dx:ax

cnt_by2 proc far
mov dx,word ptr cnt_2+2
mov ax,word ptr cnt_2
ret
cnt_by2 endp

page 60,132
public count_1

comment|  
* function: count total number of bytes (cnt_1) and vaule (sum_1) received from  
* com1. it does that by count the byte in ring buffer.  
* call: count_1()  
* return: none

count_1 proc far
pusha ;save all internal register
ctrps: mov ax,bufin ;check buffer1 if it is empty
        mov bx,bufout
        cmp ax,bx
        je ctno_s ;yes, exit
        mov ax,bx ;no,is output pointer reaching the end
        cmp ax,bufsize ;of buffer?
        ...
jne ctnext_5
push ds ;yes, get the character in last space
mov ax,seg buff ;of buffer
mov ds,ax
mov al,ds:[bx+buff]
mov ah,0
add word ptr sum_1,ax ;add its value to lower word of sum_1
adc word ptr sum_1+2,0 ;add carry to upper word
add word ptr cnt_1,1 ;add 1 to lower word of cnt_1
adc word ptr cnt_1+2,0 ;add carry to upper word
mov bx,0 ;reset the pointer to the beginning of
jmp ctnex_4 ;buffer
ctnext_5: push ds
push bx ;no, get the character from buffer
mov ax,seg buff
mov ds,ax
mov al,ds:[bx+buff]
mov ah,0
add word ptr sum_1,ax ;add its value to lower word of sum_1
adc word ptr sum_1+2,0 ;add carry to upper word
add word ptr cnt_1,1 ;add 1 to lower word of cnt_1
adc word ptr cnt_1+2,0 ;add carry to upper word
pop bx
inc bx ;increment the output pointer by 1
ctnex_4: mov bufout,bx ;reset the output pointer
pop ds
jmp ctrps ;go back to add till buffer empty
ctno_s: popa
ret
count_1 endp

Page 60,132
Public count_2

Comment

*************************************************************************************************************
* function: count total number of bytes (cnt_2) and vaule (sum_2) received from *
* com2. it does that by count the byte in ring buffer. *
* call: count_2() *
* return: none *
*************************************************************************************************************

count_2 proc far
pusha ;save all internal register
ctrps2:           mov  ax,bufin_2               ;check buffer if buffer is empty
        mov  bx,bufout_2
        cmp  ax,bx
        je  ctno_s2                    ;yes, exit
        mov  ax,bx                    ;no, is output pointer reaching the end
        cmp  ax,bufsize                ;of buffer?
        jne  ctnext_52
        push ds                        ;yes, get the character in last space
        mov  ax,bufout_2               ;of buffer
        mov  ds,ax
        mov  al,ds:[bx+buff_2]
        mov  ah,0
        add  word ptr sum_2,ax         ;add its value to lower word of sum_2
        adc  word ptr sum_2+2,0        ;add carry to upper word
        add  word ptr cnt_2,1          ;add 1 to lower word of cnt_2
        adc  word ptr cnt_2+2,0        ;add carry to upper word
        mov  bx,0                      ;reset the pointer to the beginning of
        jmp  ctnext_42                 ;buffer
        ctnext_52: push ds
        push bx                        ;save bx, using it for index addressing
        mov  ax,seg buff_2             ;no, get the character from buffer
        mov  ds,ax
        mov  al,ds:[bx+buff_2]
        mov  ah,0
        add  word ptr sum_2,ax         ;add its value to lower word of sum_2
        adc  word ptr sum_2+2,0        ;add carry to upper word
        add  word ptr cnt_2,1          ;add 1 to lower word of cnt_2
        adc  word ptr cnt_2+2,0        ;add carry to upper word
        pop  bx
        inc  bx                        ;increment the output pointer by 1
        ctnext_42: mov  bufout_2,bx    ;reset the output pointer
        pop  ds
        jmp  ctrps2                   ;go back to add till buffer empty
        ctno_s2: popa                 ;pop all internal registers from stack
        ret                            ;leave stack as it was before calling
        count_2 endp                  ;the subroutine

page 60,132
public setbps

comment |  
*******************************************************************************  
* function: set bit rate in bits per second(bps).                               *
* bps_1: bit rate for com1.  bps_2: bit rate for com2.                          *
* suggested bit rate: 57600, 38400, 28800, 23040, 19200, 14400
* 12800, 11520, 9600, 7680, 7200 (any number is a factor of 115200). 115200 bps
* may cause loss of data.
* the baud rate divisor (BRD) is calculated as follows:
* BRD = 115200/tare
* e.g. bps of 300 => BRD = 384 = 180h BRDH = 01h BRDL = 80h
* call: setbps(rate1,rate2). rate1/2 must 2 bytes long
* return: none

```
suggested bit rate: 57600, 38400, 28800, 23040, 19200, 14400
* 12800, 11520, 9600, 7680, 7200 (any number is a factor of 115200). 115200 bps
* may cause lose of data.
* the baud rate divisor (BRD) is calculated as follows:
* BRD = 115200/tare
* e.g. bps of 300 => BRD = 384 = 180h BRDH = 01h BRDL = 80h
* call: setbps(rate1,rate2). rate1/2 must 2 bytes long
* return: none

| setbps proc far bps_1:word, bps_2:word
pusha
;;; set com1
  mov   dx,word ptr i_max+2 ;get i_max value. it is a double word
  mov   ax,word ptr i_max  ;value. must divide in such format when
  div   bps_1              ;divided by a single word value
    ;result store in ax
  mov   cx,ax              ;load cs with BRD
  mov   dx,03f8h           ;access com1 line control register
  in    al,dx              ;select divisor latch
  out   dx,al              ;write to LCR
  mov   dx,03f8h           ;access com1 BRDL
  mov   ax,cx              ;get the BRD
  out   dx,al              ;write BRDL
  add   dx,1               ;access BRDH
  mov   al,ah              ;copy ah to al
  out   dx,al              ;write BRDH
  add   dx,2               ;access LCR
  in    al,dx              ;get content of LCR
  and   al,7fh             ;deselect the divisor latch
  out   dx,al              ;write to LCR
;;; set com2
  mov   dx,word ptr i_max+2 ;4 byte value divided by 2 byte value
  mov   ax,word ptr i_max  ;must in such format
  div   bps_2              ;result store in ax
  mov   cx,ax              ;load cx with BRD
  mov   dx,02f8h           ;access com2 line control register
  in    al,dx              ;select divisor latch
  out   dx,al              ;write to LCR
  mov   dx,02f8h           ;access com2 BRDL
  mov   ax,cx              ;get the BRD
```

out  dx,al ;write BRDL
add  dx,1 ;access BRDH
mov  al,ah ;copy ah to al
out  dx,al ;write BRDH
add  dx,2 ;access LCR
in   al,dx ;get content of LCR
and  al,7fh ;deselect the duvusor latch
out  dx,al ;write to LCR
popa
ret
setbps endp

page 60,132
public outcom2
comment|
******************************************************************************************************************
* function: sent the one character to comport 2. Used by c or other language.  *
* call:   outcom2(char)    *
* return: none             *
******************************************************************************************************************
|
outcom2 proc far bout:byte ;character from C in size of byte
    mov  dx,02f8h ;get comport 2 address
    mov  al,bout ;get value sent by C
    out  dx,al ;sent out to serial port
    ret
outcom2 endp

page 60,132
public outcom1
comment|
******************************************************************************************************************
* function: sent the one character to comport 1. Used by c or other language.  *
* call:   outcom1(char)    *
* return: none             *
******************************************************************************************************************
|
outcom1 proc far bout:byte ;character from C in size of byte
    mov  dx,03f8h ;get comport 1 address
    mov  al,bout ;get value sent by C
    out  dx,al ;sent out to serial port
    ret
outcom1 endp
comment

* initialize serial port
* -set transmit rate, parity, stop bit and # of bits/char
* -save IRQ4(com1) and IRQ3(com2) interrupt vectors
* -replace interrupt IRQ4 and IRQ3 vectors within ours
* -enable interrupt, data available interrupt for com1 and
* com2 on UART 8250
* -enable modem general interrupt bit
* -enable interrupt controller for IRQ4 & IRQ3(active low)

| setintr proc far
pusha
cli

;--- set speed, parity, stop bit and # of bits per char
mov dx,0 ;com1
mov al,73h ;9600 bps
mov ah,0 ;int severe 0
int 14h ;call int 14h to set it
mov dx,1 ;com2
mov al,73h ;9600 bps
mov ah,0 ;int severe 0
int 14h ;call int 14h to set it

;--- save original interrupt vectors
xor ax,ax
mov si,ax ;make si to 0 for index addressing mode
mov es,si ;use es to access low memory
mov ax,es:[si+2ch] ;get IRQ3 offset
mov [si+sav_3],ax ;save it
mov ax,es:[si+2eh] ;get IRQ3 segment address
mov [si+sav_3+2],ax ;save it
mov ax,es:[si+30h] ;get IRQ4 offset
mov [si+sav_4],ax ;save it
mov ax,es:[si+32h] ;get IRQ4 segment address
mov [si+sav_4+2],ax ;save it

;--- place our own interrupt vectors
mov ax,offset serintr ;get offset of serint
mov es:[si+2ch],ax ;put it in both IRQ3 and IRQ4
mov es:[si+30h],ax
mov ax,seg serintr ;get segment address of serint
mov es:[si+2eh],ax ; put it in both IRQ3 abd IRQ4
mov es:[si+32h],ax

;--- enable modem general interrupt bit
mov dx,mcr_1 ; get com1 modem control register
in al,dx
or al,8 ; set interrupt bit on
out dx,al
mov dx,mcr_2 ; get com2 modem control register
in al,dx
or al,8 ; set interrupt bit on
out dx,al

;--- enable UART 8250
mov dx,inte_1 ; get com1 interrupt enable register
mov al,1 ; set data available interrupt on
out dx,al
mov dx,inte_2 ; get com2 interrupt enable register
mov al,1 ; set data available interrupt on
out dx,al

;--- enable 8259 interrupt controller
in al,21h
mov ah,0e7h ; set IRQ4 and IRQ3 interrupt on
and al,ah
out 21h,al
sti
popa
ret

setintr endp

page 60,132
public serintr

comment | 
*************************************************************************************************
| * interrupt service routine for receiving only                              *
* - identify what kind of interrupt has occurred                            *
* - call appropriate service routine for the interrupt                    *
* - turn off interrupt flag                                                *
* - send EOI. end of interrupt to CPU and IRQ controller                   *
************************************************************************************************* 

serintr proc far
pusha
mov dx,iir_1 ; get com1 interrupt identification
in al,dx ; register
mov ah,al
cmp ah,1 ;has any interrupt occurred in com1?
je nexint ;no, it MUST com2
mov dx,recb_1 ;yes, get com1 receiver buffer
in al,dx ;input char in al
call far ptr putb ;call putb subroutine to put it to ring
jmp endint ;buffer1

nexint: mov dx,iir_2 ;get com2 interrupt identification
in al,dx ;register
mov ah,al ;has any interrupt occurred in com2?
cmp ah,1 ;no, exit
je endint
mov dx,recb_2 ;yes, get com2 receiver buffer
in al,dx ;input character in al
call far ptr putb_2 ;call putb_2 to put it to ring buffer2

endint: mov al,20h ;send end of interrupt to controller
out 20h,al ;inform I/O interrupt service is over
popa
iret

serintr endp

page 60,132
public putb

comment

*****************************************************************************
* function: place one character in buffer1
* call: putb
* return: none
*****************************************************************************

putb proc far
push ds
mov cx,seg buff ;get segment address of buffer1
mov ds,cx ;put it in data segment register
mov bx,bufin ;get buffer input pointer
mov cx,bx ;make a copy
cmp cx,bufsize ;is it pointing to end of buffer?
jne putb2
mov ds:[bx+buff],al ;yes, get the character
mov bx,0 ;reset input pointer points to beginning
jmp putb3 ;of buffer

putb2: mov ds:[buff+bx],al ;no, put the character
inc bx ;increment pointer by 1
putb3:  mov  bufin,bx
           pop  ds
           ret
putb   endp

page 60,132
public  putb_2

comment
******************************************************************************
* function: place one character in buffer2
* call: put_2
* return: none
******************************************************************************

putb_2   proc far
           push  ds
           mov  cx,seg buff_2 ;get segment address of buffer2
           mov  ds,cx         ;put it in data segment register
           mov  bx,bufin_2    ;get buffer input pointer
           mov  cx,bx         ;make a copy
           cmp  cx,bufsize    ;is it pointing to end of buffer?
           jne  putb2-2
           mov  ds:[bx+buff_2],al ;yes, get the character
           mov  bx,0           ;reset input pointer points to beginning
           jmp  putb3_2        ;of buffer
putb2_2:  mov  ds:[buff_2+bx],al ;no, put the character
           inc  bx             ;increment pointer by 1
putb3_2:  mov  bufin_2,bx
           pop  ds
           ret
putb_2   endp

page 60,132
public  getb

comment
******************************************************************************
* function: get a character from ring buffer1 and send it to com2
* call: getb
* return: none
******************************************************************************

getb   proc far
           push  ds
mov ax, seg buff ; get segment address of buffer1
mov ds, ax ; put it in data segment register
mov bx, bufout ; get buffer output pointer
mov cx, bx ; make a copy
cmp cx, bufin ; is the buffer empty?
je out_1 ; yes, out of service routine
mov cx, bx ; no, make a copy of output pointer
cmp cx, bufsize ; is it pointing to end of buffer?
jne nex_2
mov dx, recb_2 ; yes, get com2 transmit holding register
mov al, ds:[bx + buff] ; get the character from ring buffer
out dx, al ; send it to transmit holding register
mov bx, 0 ; reset pointer points to beginning of
jmp nex_2 ; buffer

nex_2: mov dx, recb_2 ; no, get com2 transmit holding register
mov al, ds:[bx + buff] ; get the character from ring buffer
out dx, al ; send it to transmit holding register
inc bx ; increment output pointer by 1

nex_1: mov bufout, bx
out_1: pop ds
ret
getb endp

public tranb

comment |

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

* function: transfer a character from ring buffer1 to C
* call: tranb
* return: ax
***********************************************************************************************************************************************

tranb proc far
push ds
mov ax, seg buff ; get segment address of buffer1
mov ds, ax ; put it in data segment register
mov bx, bufout ; get buffer output pointer
mov cx, bx ; make a copy
cmp cx, bufin ; is the buffer empty?
je tr_out ; yes, out of service routine
mov cx, bx ; no, make a copy of output pointer
cmp cx, bufsize ; is it pointing to end of buffer?
jne tr_2

takeb proc far
push ds
     mov ax,seg buff  ;get segment address of buffer1
mov ds,ax         ;put it in data segment register
mov bx,bufout     ;get buffer output pointer
mov cx,bx         ;make a copy
cmp cx,bufin      ;is the buffer empty?
je   ta_re        ;yes, out of service routine
mov cx,bx         ;no, make a copy of output pointer
cmp cx,bufsize    ;is it pointing to end of buffer?
jne  ta_2         ;if it is not
mov al,ds:[bx+buff] ;get the character from ring buffer
mov ah,0
mov bx,0         ;reset pointer points to beginning of
jmp ta_1         ;buffer

        ta_2:  mov al,ds:[bx+buff]  ;get the character from ring buffer
mov ah,0         ;make sure ah = 0
inc bx            ;increment output pointer by 1

        ta_1:  mov bufout,bx
jmp ta_out

        ta_re: mov ah,7    ;make it not 0
ta_out:    pop    ds
            ret
takeb    endp

page 60,132
public takeb2

comment

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

* function: transfer a character from ring buffer2 to C
* if buffer empty return ah = 0xFF. otherwise ah = 0
* call: takeb2
* return: ax
************************************************************************

| takeb2  proc  far
push    ds
mov     ax,seg buff_2 ;get segment address of buffer2
mov     ds,ax          ;put it in data segment register
mov     bx,bufout_2    ;get buffer output pointer
mov     cx,bx          ;make a copy
cmp     cx,bufin_2     ;is the buffer empty?
je      ta_re2         ;yes,out of service routine
mov     cx,bx          ;no, make a copy of output pointer
cmp     cx,bufsize     ;is it pointing to end of buffer?
jne     ta_22          ;buffer
mov     al,ds:[bx+buff_2] ;get the character from ring buffer
mov     ah,0
mov     bx,0           ;reset pointer points to beginning of
jmp      ta_12          ;buffer
ta_22:   mov     al,ds:[bx+buff_2] ;get the character from ring buffer
mov     ah,0            ;make sure ah=0
inc     bx             ;increment output pointer by 1
ta_12:   mov     buffout_2,bx
        jmp      ta_out2
ta_re2:   mov     ah,7
ta_out2:   pop    ds
            ret
takeb2    endp

page 60,132
public append_b

comment

************************************************************************
* function: append a byte to buffer1. usually called by main program. can be used  
* when want to send something through serial port.  
* call: append_b(bvalue). bvalue = byte value  
* return: none

```
append_b proc far chat:byte
pusha
mov cx,seg buff ;get segment address of buffer1
mov ds,cx ;put it in data segment register
mov bx,bufin ;get buffer input pointer
mov cx,bx ;make a copy
cmp cx,bufsize ;is it pointing to end of buffer?
jne app2
mov al,chat
mov ds:[bx+buf2],al ;yes, put the character
mov bx,0 ;reset input pointer points to beginning
jmp app3 ;of buffer
app2: mov al,chat
mov ds:[buff+bx],al ;no, put the character
inc bx ;increment pointer by 1
app3: mov bufin,bx
popa
ret
append_b endp
```

---

append_2 proc far chat:byte
pusha
mov cx,seg buff_2 ;get segment address of buffer2
mov ds,cx ;put it in data segment register
mov bx,bufin_2 ;get buffer input pointer
mov cx,bx ;make a copy
cmp cx,bufsize_2 ;is it pointing to end of buffer?
```
jne    app22
mov    al,chat
mov    ds:[bx+buff_2],al  ;yes, put the character
mov    bx,0              ;reset input pointer points to beginning
jmp    app32            ;of buffer

app22: mov    al,chat
        mov    ds:[buff_2+bx],al  ;no, put the character
        inc    bx                ;increment pointer by 1

app32:  mov    bufin_2,bx
        popa
        ret
append_2  endp

page 60,132
public  display_1

comment|
******************************************************************************
* function: display content of com1 buffer *
* call: display_1()               *
* return: none                    *
******************************************************************************
|
display_1  proc    far
    pusha
    ;cli       ;disable interrupt
    rps:      mov    ax,bufin       ;check buffer if it is empty
    mov    bx,bufout
    cmp    ax,bx
    je     no_s                      ;yes, nothing to send.
    mov    ax,bx                     ;is output pointer reaching the end
    cmp    ax,bufsize                ;of buffer?
    jne    next_5
    push    ds                        ;yes, get the character in last space
    mov    ax,seg buff               ;of buffer
    mov    ds,ax
    mov    al,ds:[bx+buff]
    mov    bh,0
    mov    bl,3
    mov    ah,14
    int    10h
    mov    bx,0                       ;reset the pointer to the beginning of
    jmp    nex_4                      ;buffer

next_5:   push    ds
push bx ; no, get the character from buffer
mov ax, seg buff
mov ds, ax
mov al, ds: [bx + buff]
mov bh, 0
mov bl, 3
mov ah, 14
int 10h
pop bx
inc bx ; increment the output pointer by 1
nex_4:
mov bufout, bx ; reset the output pointer
pop ds
jmp rps
no_s: ; sti ; enable interrupt
popa
ret
display_1 endp

page 60, 132
public display_2

comment

display_2 proc far
    pusha
    cli ; disable interrupt
rps_2: mov ax, bufin_2 ; check buffer if it is empty
    mov bx, bufout_2
    cmp ax, bx
    je no_s_2 ; yes, nothing to send.
    mov ax, bx ; is output pointer reaching the end
    cmp ax, bufsize ; of buffer?
    jne next_5_2
    push ds ; yes, get the character in last space
    mov ax, seg buff_2 ; of buffer
    mov ds, ax
    mov al, ds: [bx + buff_2]
    mov bh, 0
    mov bl, 3
mov ah,14
int 10h
mov bx,0 ;reset the pointer to the beginning of
jmp nex_4_2 ;buffer

next_5_2: push ds
push bx ;no, get the character from buffer
mov ax,seg buff_2
mov ds,ax
mov al,ds:[bx+buff_2]
mov bh,0
mov bl,3
mov ah,14
int 10h
pop bx
inc bx ;increment the output pointer by 1

nex_4_2: mov bufout_2,bx ;reset the output pointer
pop ds
jmp rps_2

no_s_2: ;sti ;enable interrupt
popa
ret
display_2 endp

deltab proc far
mov ax,bufin
mov bx,bufout
sub ax,bx
ret
deltab endp

deltab2 proc far
mov ax,bufin
mov bx,bufout
sub ax,bx
ret
deltab2 endp

page 60,132
public deltab
comment|

* function: computes bytes left in ring buffer 1 by subtract out pointer by in pointer  *
* call: deltab().
* return: al  *

page 60,132
public deltab2
comment|
* function: computes bytes left in ring buffer 2 by subtract out pointer by in pointe
* call: deltab2().
* return: al

```
function: computes bytes left in ring buffer 2 by subtract out pointer by in pointe

| deltab2 proc far
    mov ax,bufin_2
    mov bx,bufout_2
    sub ax,bx
    ret
| deltab2 endp

page 60,132
public deltab2
```

* function: clear ring buffer 1 by setting out pointer equal to in pointer
* call: cl_ring().
* return: none

```
function: clear ring buffer 1 by setting out pointer equal to in pointer

| cl_ring proc far
    mov ax,0
    mov bufin,ax
    mov bufout,ax
    ret
| cl_ring endp

page 60,132
public cl_ring
```

* function: clear ring buffer 2 by setting out pointer equal to in pointer
* call: cl_ring2().
* return: none

```
function: clear ring buffer 2 by setting out pointer equal to in pointer

| cl_ring2 proc far
    mov ax,0
    mov bufin_2,ax
    mov bufout_2,ax
    ret
| cl_ring2 endp
```
public clsint

* close IRQ service (MUST be called before exit program)
* -reinstall original IRQ4 & IRQ3 interrupt vectors
* -turn off modem general interrupt enable bit
* -turn off UART 8250 interrupt register
* -turn off 8259 interrupt controller

| clsint proc far
pusha
cli
xor ax,ax
mov si,ax
mov es,ax

;--- reinstall IRQ4 and IRQ3 interrupt vectors
mov ax,[si+sav_4] ;get offset of IRQ4
mov es:[si+30h],ax ;put it in its place
mov ax,[si+sav_4+2] ;get segment address of IRQ4
mov es:[si+32h],ax ;put in its place
mov ax,[si+sav_3] ;get offset of IRQ3
mov es:[si+2ch],ax ;put in its place
mov ax,[si+sav_3+2] ;get segment address of IRQ3
mov es:[si+2eh],ax ;put it in its place

;--- turn off modem general interrupt bit
mov dx,mcr_1 ;get com1 modem control register
in al,dx
and al,0f7h ;turn general IRQ off
out dx,al ;set it
mov dx,mcr_2 ;get com2 modem control register
in al,dx
and al,0f7h ;turn general IRQ off
out dx,al ;set it

;--- turn off interrupt enable register in UART 8250
mov dx,inte_1 ;get com1 interrupt enable register
xor al,al ;turn all interrupt off
out dx,al ;set it
mov dx,inte_2 ;get com2 interrupt enable register
xor al,al ;turn all interrupt off
out dx,al ;set it

;--- turn of IRQ4 and IRQ3 interrupt in 8259 interrupt controller
in al,21h ;get controller register
mov ah,18h
or al,ah ;turn IRQ4 & IRQ3 off(negative logic)
out 21h,al
sti
popa
ret
clsint endp
8 References


Braasch, Michael (1994), Private communication with the author.


van Graas, F. (1993), Private communication with the author.