NOAA Technical Memorandum OAR ARL-248

COMBINING CONDITIONED LASER ALTIMETER DATA AND GPS ALTITUDE DATA TO OBTAIN ACCURATE AIRCRAFT SENSOR HEIGHT MEASUREMENTS

Tamara K. Grimmett
NRC Postdoctoral Associate

Field Research Division
Idaho Falls, Idaho

Air Resources Laboratory
Silver Spring, Maryland
March 2003

UNITED STATES DEPARTMENT OF COMMERCE

Donald L. Evans
Secretary

NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION

VADM Conrad C. Lautenbacher, Jr.
Under Secretary for Oceans and Atmosphere/Administrator

Oceanic and Atmospheric Research Laboratories

Daniel L. Albritton
Acting Director

Notice

This document was prepared as an account of work sponsored by an agency of the United States Government. The views and opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government. Neither the United States Government, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, product, or process disclosed, or represents that its use would not infringe privately owned rights. Mention of a commercial company or product does not constitute an endorsement by NOAA/OAR. Use of information from this publication concerning proprietary products or the tests of such products for publicity or advertising purposes is not authorized.

Contents

List of Figures

List of Tables

Abstract

For low level applications, the height of a given sensor above a surface or vegetation canopy is often a critical measurement if vertical gradients are significant (i.e. within the lower boundary layer.) This is particularly true for low flying aircraft (10-20 m altitude) where, under stable conditions, vertical gradients are significant over a few meters. Data from two different altitude measuring systems (GPS and laser altimeters) can be combined to produce an accurate sensor altitude measurement. This report outlines an algorithm developed to condition laser altimeter data and combine it with Global Positioning System (GPS) altitude data to arrive at a more accurate sensor altitude measurement for low-level flights.

1 Introduction

In an effort to increase our knowledge of the air-sea interface, the Coupled Boundary Layers Air-Sea Transfer (CBLAST) for light-wind (Low) research program was created. The objectives of the CBLAST-Low program are

- to measure vertical fluxes of momentum and heat in the lower atmospheric boundary layer and in the ocean surface layer;
- to identify the processes that influence these fluxes (e.g., shear, convection, surface wave breaking, Langmuir cells);
- to close budgets for heat and momentum;
- to test parameterizations of fluxes; and
- to obtain other measurements (e.g., horizontal variability of pressure and temperature sufficient to provide boundary conditions for a large eddy simulation or local application of a regional-scale simulation).

The initial CBLAST-Low pilot study was conducted during a three week period in July and August 2001 over the Atlantic ocean south of Martha's Vineyard Island, Massachusetts. The LongEZ (registration N3R) research aircraft acquired high-resolution *in situ* atmospheric turbulent fluxes in the marine atmospheric boundary layer (MABL) and simultaneously documented the characteristics of the surface wave field with various remote sensors [Crescenti et al. (2001)]. This highly instrumented aircraft has proven to be an especially powerful tool for studying the spatial variability of air-sea interaction [Crawford et al. (1993), Vogel and Crawford (1999), Crescenti et al. (1999), Mahrt et al. (2001), Mourad (1999), Sun et al. (2001), Vogel and Crawford (1999), Vandemark et al. (2001), French et al. (2000), Mourad et al. (2000), Vickers et al. (2001)].

The purpose of this technical report is to outline a methodology for deriving highly accurate sensor altitude measurements above the sea surface. Precise sensor altitude is necessary to correct or extrapolate wind speed to a 10-meter reference height. Corrections may be significant under stable conditions where strong vertical gradients may introduce significant changes in wind speed over a few meters. We have observed uncertainties in GPS-derived altitudes up to 4 meters even after differential corrections have been applied.

2 Instrumentation

A dual-frequency Ashtech GPS is used on N3R to determine ground velocity and aircraft position (x,y,z) relative to a fixed point on earth. By using differential GPS (DGPS) correction techniques, aircraft position can be measured to within several centimeters and ground velocity can be computed to an accuracy of roughly 2 cm/s in the horizontal and 2.5 cm/s in the vertical. (DGPS is not always available such as for flights over the remote ocean or the arctic sea-ice where a fixed base station is not possible.) These data are acquired at a rate of 5 Hz. Three

orthogonally-mounted accelerometers are used to augment the GPS position and velocity data to a frequency of 50 Hz.

An array of laser altimeters is designed to measure the sea surface profile and the one- and two-dimensional slopes of intermediate scale waves on the order of 1 to 10 m from three independent altitude measurements. The laser array consists of three Riegl downward-looking lasers mounted on the vertices of an equilateral triangle with 0.95-m separation. Two are mounted under either wing (model LD90-3100VHS) while the third (model LD90-3100EHS) is situated in an instrument pod mounted underneath the aircraft fuselage. The circular footprint of each laser is about 7.5 mm in diameter at an altitude of 15 m. The two lasers mounted in either wing operate with a pulse repetition frequency of 2 KHz while the third laser in the pod operates at a frequency of 12 KHz. The individual pulses are averaged down to a rate of 150 Hz to reduce noise resulting in 13 pulses per data point for the 2-KHz lasers and 80 pulses per data point for the 12-KHz laser. The lasers output distance and number of valid returns. The focal length of the lasers are set to 15 m providing a nominal accuracy of ± 2 mm. The maximum altitude for which the lasers provide useful data is about 50 m. For more information on N3R and its instrumentation, see Crescenti et al. (2001).

Even after differential corrections are applied, the GPS-derived altitude above sea level has uncertainties which are due to the geometry of the constellation of satellites, the geoid model used to represent the earth's surface, the actual location of the sea surface compared to the geoid surface, atmospheric refraction, multipath effects and the number of satellites. This causes the altitude measurement of the aircraft to be off by as much as 4 meters even though the reported accuracy of the GPS is several centimeters. The most likely source of this large error is the fact that the surface of the ocean is assumed to be the surface of the geoid. This error in altitude of up to 4 meters may not be significant at high altitudes but at lower altitudes (10-20 meters), where gradients in meteorological variables are larger, can be a signficant error. The signal is 'clean' though in that it has no outliers or dropouts unlike the laser measurements which are quite accurate below 50 m, but exhibit a significant number of outliers and dropouts. The lasers require some surface roughness to get a return. Therefore, at very low wind speeds (< 2 m/s) when the ocean is relatively smooth, the lasers may not provide reliable data. By conditioning the laser data and combining it with the Ashtech altitude data, an accurate sensor altitude measurement can be obtained. This report outlines the procedure developed to detect outliers in the laser data, how values are interpolated to fill in the missing laser data, and the method of merging the GPS data with the conditioned laser data.

3 Data and Data Processing

All of the data presented in this report are from the first day's flight (21 July 2001) where wind speeds ranged from about 2 to 4.5 m/s providing conditions such that the lasers should return good data. However, there were many drop outs and outliers. Drop outs were possibly due to small slicks on the ocean surface which cause the lasers to receive no returns. Outliers might be due to interference between the lasers. For example, the ocean surface could be such that laser 1 might receive laser 3's signal. Also the plane may not always be in level flight which could be another cause of outliers.

The data post-processing and graphics generation is done with Research Systems, Inc. (www.rsinc.com) IDL (Interactive Data Language) program. In IDL, the laser data is stored as a two dimensional array. The first dimension is data rate, and the second is the number of scans. One scan is one second's worth of data. For the lasers which are sampled at 150 Hz, the first dimension is 150 and the second dimension, for example, of a 60 second record would be 60. IDL converts the two-dimensional array to a one-dimensional array when plotting. Therefore when plotting the laser data, the numbers that would be seen on the x-axis would appear quite large because they would be the actual number of data points. However, to make the plots more readable, the x-axis is presented as number of scans. It must be remembered that each scan consists of 150 data points. Therefore, the actual number of points between scans 3100 and 3200 is (3200-3100+1)*150=15150.

4 Methodology

Figure 1 shows data for all three lasers from a flux leg of the first day's flight. The flux leg (scans 2955 to 3547) consists of 593 scans (nearly 10 minutes) which corresponds to 88950 data points. It is easy to see the good signal in the data, but many outliers are evident, particularly for lasers 1 and 3. Laser 2, with its higher sampling rate, exhibits fewer outliers.

4.1 Outlier Detection

The first problem is to identify the invalid data points and outliers in the laser data. This is done by applying three different detection methods. The first method, dubbed the 'returns' method, identifies all invalid data points which are defined as data points that have less than 3 returns. The second is a distance-based (DB) outlier detection algorithm. The third is to apply Grubbs' test [GraphPad.com (2000)] for outliers. A box plot method was tried in place of the distance-based method and Grubbs' test initially, but because of the erratic nature of the data it was unable to satisfactorily detect the outliers. The so called 'returns' method identifies bad points, not just outliers. The DB method and Grubbs' test are designed to detect outliers. Throughout the remainder of this report, the terms drop outs and outliers are referred to as bad points since in the end all these points are rejected and new data are interpolated at their locations. For all of these methods, one scan at a time is examined.

Returns As mentioned in section 2, 13 valid returns are possible for lasers 1 and 3 and a possible 80 returns for laser 2. If less than 3 valid returns for any laser are received, the data point is considered invalid. This filter will reject points that are outliers or dropouts as well as data that appears to be valid. If in one scan (150 data points) there are less than 4 good data points, the whole scan is marked as bad and values will be interpolated for it. Figure 2 shows, for flux leg 2 (88950 data points), the data points that are identified as bad by this filter. For laser 1, 28.6% (25467 data points) of the total points are identified as bad. For laser 2, 20.1% (17863 data points) are identified as bad and for laser 3, 29.9% (26564 data points) are bad. (Table 1 summarizes the cumlative number of points identified as bad with each successive detection method.)

Distance-Based Method Many outliers still remain after the 'returns' filter is applied, so the distance-based (DB) outlier detection scheme based on work of Knorr and Ng [Knorr and Ng (1998)] is employed. This method was intended for multi-dimensional databases so the basic concept is simplified for the one-dimensional case. The algorithm is as follows:

Laser	Returns	DistBased	Grubbs
1	25467 (28.6%)	27165 (30.5%)	33519 (37.7%)
2	17863 (20.1%)	19940 (22.4%)	26304 (29.6%)
3	26564 (29.9%)	28634 (32.2%)	34875 (39.2%)

Table 1: Summary of cumulative number of bad points identified by different detection methods. Each column indicates the number and percentage of bad points found by each method and any proceeding methods for scans 2955 to 3547 (88950 data points).

1. Determine how many 'good' points are left after the 'returns' filter.
2. Find the first good point in the scan.
3. Compute the Lagrangian distances ($l = \sqrt{x^2 + y^2}$) from the first good point to every other good point in the scan.
4. If a distance exceeds some value, D, then mark the point as an outlier. (A data point is marked bad by using NaN.)
5. Add these outliers to the bad points found by the 'returns' filter.

The first good point is determined by computing 'max' and 'min' values which are defined as $\pm$ one standard deviation from the mean of the remaining good points, and the first point that falls within this range is selected as the first good point. A value of $D = 1.2$ was determined empirically to give good results when less than 143 good points remain after the 'returns' test. If 143 or more points are good, then D is set to 0.5. The value 0.5 was also empirically determined. It was found that if more than 143 points in the scan were good, then outliers were generally closer to the good data so the distance measure had to be decreased. Figure 3 shows all the points that are detected as bad from the returns filter and the distance-based outlier detection scheme.

Grubbs' Test A very careful examination of Figure 3 shows some few outliers still remain after the DB method is applied. A smaller section (scans 3150 to 3200) of the laser 3 data of Figure 3 is shown in Figure 4 which shows more clearly some of these remaining outliers. As a final step to detect these outliers, Grubbs' test [GraphPad.com (2000)] is used. Grubbs' test is only done if at least 6 good data points remain in the scan. If this is true, then the following algorithm is followed otherwise Grubbs' test is not performed.

1. Compute the mean (μ) and standard deviation (σ) of the remaining good points.
2. Compute the Grubbs' test statistic as
$$Z = \frac{|\mu - value|}{\sigma}$$
3. Compute the critical value of Z, Z_c.
$$Z_c = \frac{(N-1)}{\sqrt{N}} * 0.15$$

where N is the number of values. The criteria for outliers is $Z > Z_c$. Z_c can never be larger than $(N-1)/\sqrt{N}$, but since the outliers sought in this step are very near the good data, the critical value was reduced significantly by the factor 0.15 which was empirically determined.

4. Mark outliers with NaN and add to other bad points.

Figure 5 shows results (same as Figure 4) after Grubbs' test is used.

4.2 Interpolation of Data

After all the bad data points have been determined for each laser, data is interpolated at the locations of these bad points to provide a good continuous signal. Simple linear interpolation is used. The entire flux leg is treated at once versus a scan at a time. The reason for this is that there are some scans with no good points with which to start the interpolation. By using the full flux leg there are always good points to use in the interpolation scheme. The interpolation algorithm is as follows:

1. Sort the bad points in ascending order.
2. If there are no bad points, return.
3. Note the location of a bad point.
4. Search to the left until a good point is found; note as L. If the end of the flux leg is reached, set the left point to be the first point of the flux leg.
5. Search to the right until a good point is found; note as R. If the end of the flux leg is reached, set the right point to the last point of the flux leg.
6. Determine how many points need to be interpolated $(R - L + 1)$.
7. Linearly interpolate for the bad points using IDL's built in functions.
8. Check if the interpolated point is within 3% of the original point. If the interpolated value is less than 3% different from the original, then use the original value, else use the interpolated value.
9. Repeat from step 3 until no outliers are left.

The check to see if the interpolated value is within 3% of the original value is done to ensure that any original data, if it appears to be good, is used. This uncertainty can arise for the case when the original point was marked bad because of too few returns. In some of these cases, an interpolated value can be very close to the original value, and it was deemed better to use the original data rather than the interpolated value.

If no good data points are found to the left or right of a given bad point, then the interpolation is done from zero. For example, if there is not good data point found to the left in the search to the

beginning of the flux leg, then that first point is assumed to be zero and is used in the interpolation. This is discussed further in section 4.4.

After the three lasers have been corrected to obtain three good continuous signals, the vertical distance from a given sensor (for example, the wind measurement sensor) to the ocean surface is derived. This involves converting the data from aircraft based coordinates to earth-based coordinates. The conversion to earth-based coordinates, explained more fully in Vogel and Crawford (1999), involves taking into account the attitude of the aircraft and the sensor locations.

4.3 Merging GPS and Laser Data

As mentioned earlier, the GPS altitude measurement tends to drift. Figure 6 shows the laser 1 data ($L1Dist$) with the GPS ($NAlt$) measurement. The GPS altitude is several meters lower than the laser measurement and is evidence of the drift in the Ashtech GPS. The merging of the GPS data and the laser data is done as follows:

1. Average the three laser readings together to get one reading.
2. Compute the median difference between the average laser reading and the GPS signal in 60 second (scan) increments. Consideration is given to the fact that the laser data is taken at 150 Hz while the GPS data is 50 Hz.
3. Add this value to the GPS data.
4. Handle in a similar manner the last interval of the flux leg that is less than the 60 second averaging time in length.

This method gives reasonable results as seen in Figure 7. The variable $L1Out$ is the conditioned laser 1 data; variable $L123Avg$ is the average of the 3 conditioned (in earth-based coordinates) laser signals; and $NAltc$ is the offset GPS signal which is taken as the true sensor altitude. Since the aircraft is flying nearly level most of the time the conversion to earth-based coordinates makes little difference.

4.4 Known Problems

The above outlined method gives good results for the middle portion of flux legs where the attitude of the plane is relatively level. But at the beginning and end of flux legs, there are scans where the aircraft is pulling out of or going into a profile event. The plane is not completely level during this time which would cause the laser signal to glance off the water surface at an oblique angle and not return to the antenna causing drop outs. For example, a careful look at the very left edge of Figure 7 shows that $L123Avg$ appears to go to zero. A closer look at just the first scan (scan 2955) for the flux leg, clearly shows the problem. This is shown in Figure 8 where the corrected laser 3 data (before the conversion to earth-based coordinates), the three-laser averaged signal and the corrected GPS value are plotted. The corrected GPS value $NAltc$ gives a seemingly correct answer because of the offset added to the raw GPS based on a 60-second average of the combined laser signal.

Figure 9 shows the raw laser data and the corrected laser data for the three lasers. It can be seen that the corrected signal for laser 2 and 3 goes to zero at the very beginning of the scan. This is because there were no good data points until later in the scan, thus when the interpolation was done there was no 'left' side data point except zero. The interpolated signal ramps from 0 to the first good data point. In calculating fluxes from these flights, the very beginning and ending of each flux leg is neglected to ensure only a good signal is being used.

5 Discussion

There is probably more 'tweaking' that could be done to this method, but the steps as they now stand give satisfactory results in a reasonable amount of time. Therefore it was deemed unnecessary to try to refine the method any further. However, many of the parameters in this method are arbitrarily chosen. For example, the minimum number of points required to determine that a scan is good was chosen as four after looking at many scans, but the number is still rather arbitrary. And deciding that six points were sufficient to use Grubbs' test was arbitrarily chosen also though based on minimums suggested for the method. These types of parameters would need adjusting to apply this method to data from another experiment. This method would therefore be more useful if the adjustable parameters could be chosen as a percentage of the number of data points or by some other statistical test. Determining these parameters for each experiment empirically (trial and error) is time consuming and very arbitrary.

Other outlier detection schemes might also work in place of the distance-based scheme and/or Grubbs' test, but these worked well for the CBLAST-Low data. Repeated applications of Grubbs' test with decreasing critical values might also work just as well, but this was not tried. Repeated applications of the DB method did not work. It was not possible to find a second value for D that would detect all remaining outliers after an initial run of the DB method.

Determining how well this algorithm performs is difficult. Checking to see if all bad points are identified in all scans for all flights by visual means is time consuming and tedious. Therefore, a check of every scan was not done, but random ranges of scans were visually checked. A better checking method would be desirable, but if this could be devised a better detection scheme would then immediately be available.

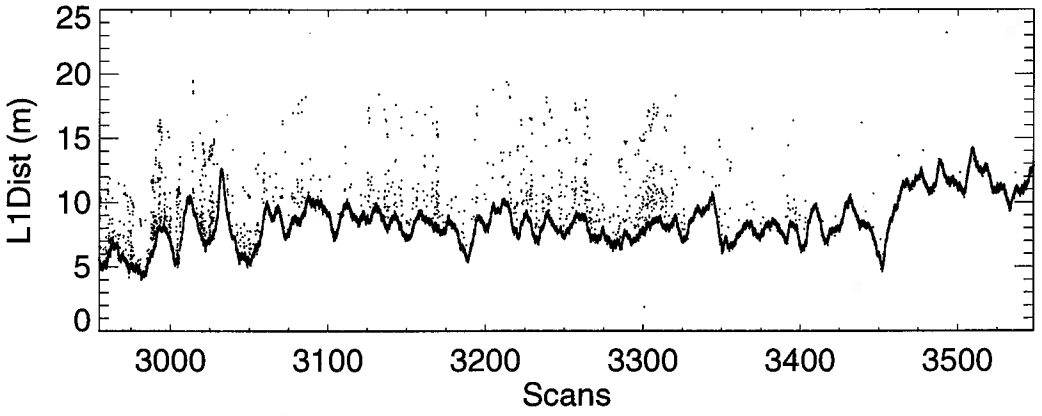
This method could also be used with only one laser altimeter measurement. In this case, there would be no averaged laser signal but one signal to correct the GPS altitude.

Comments from a reviewer were gratefully received. It was suggested that the work be extended to include height assessment over a forest. However, that was beyond the scope of the present work. But in that case, the algorithm would be revised to calculate sensor heights above both the ground and the canopy. Also a future application might be a real-time height algorithm developed for unmanned aerial vehicle (UAV) autopilot software. However, this has probably already been addressed more rigorously elsewhere.

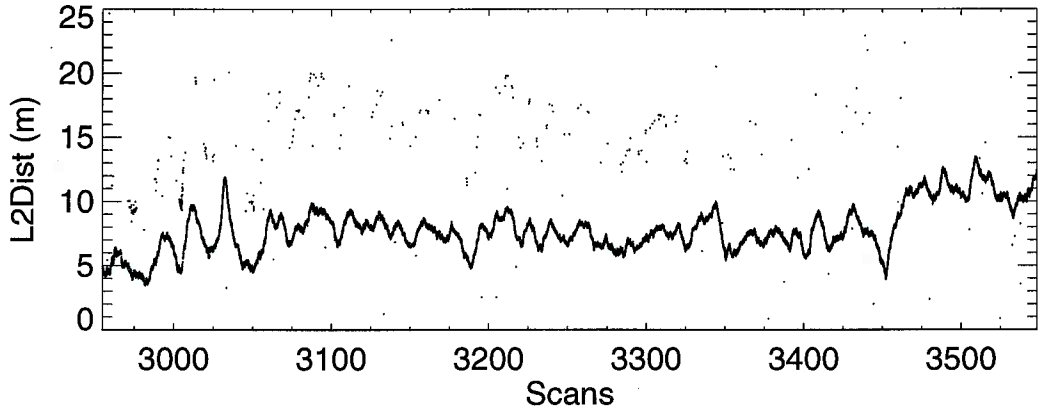
6 Acknowledgments

This work was performed while the author held a National Research Council Research Associateship Award at NOAA's Air Resources Laboratory/Field Research Division in Idaho

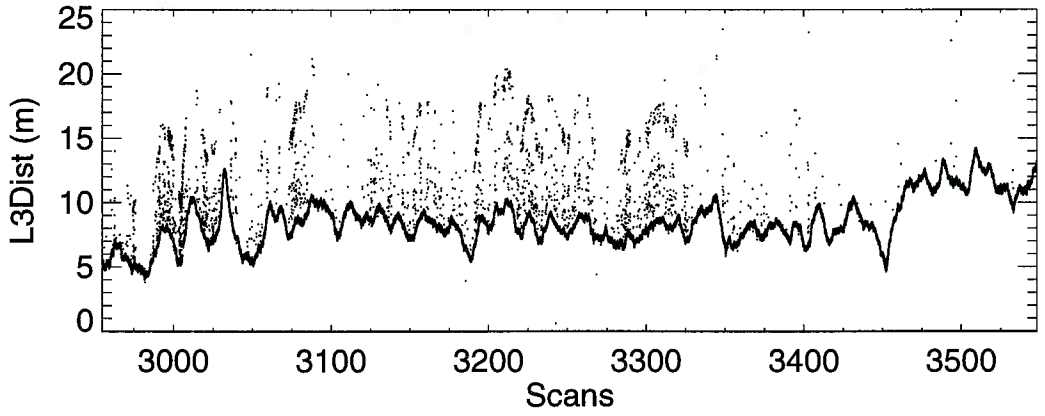
Falls, Idaho. The CBLAST-Low work was supported by a contract from the Office of Naval Research (N00014-01-F-0008).



(a) Laser 1 (left wing)

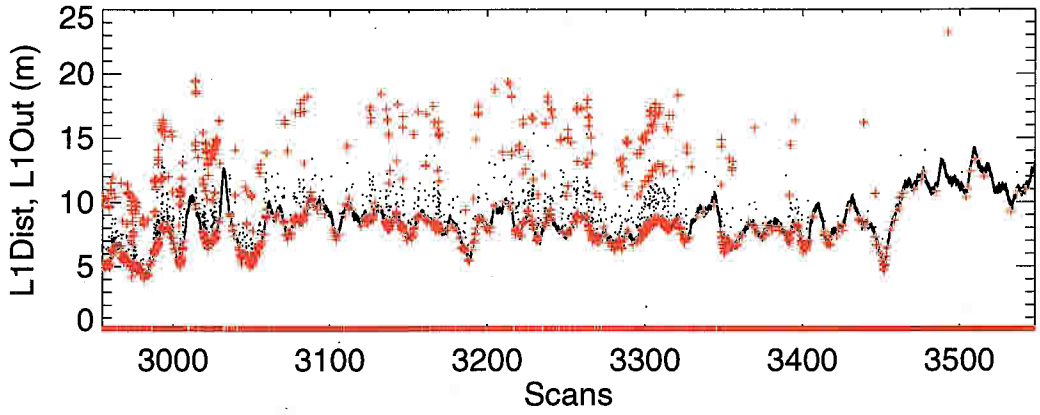


(b) Laser 2 (pod)

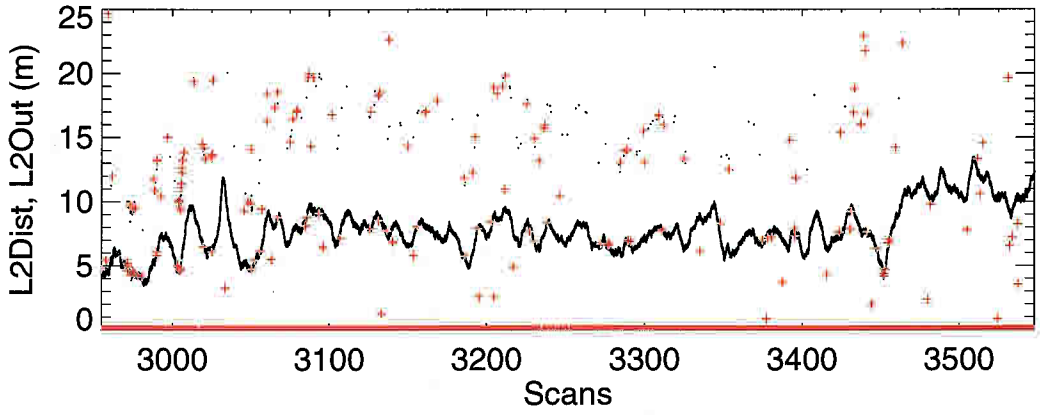


(c) Laser 3 (right wing)

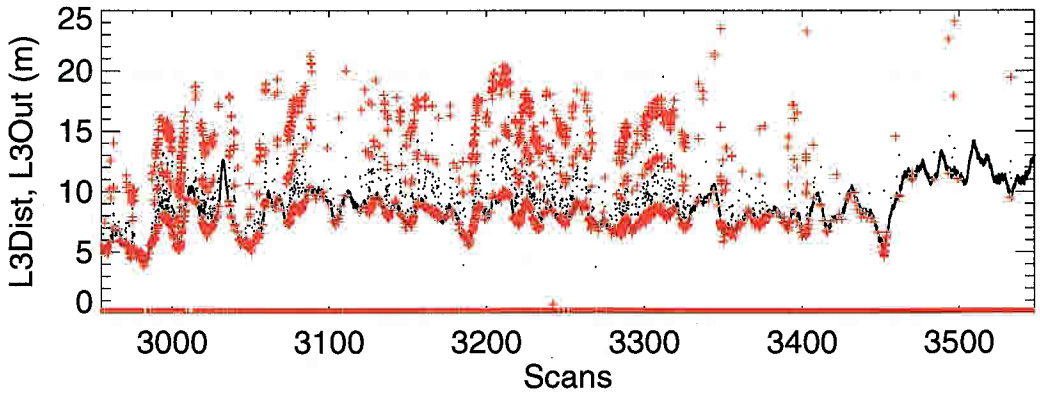
Figure 1: Raw laser data ($LXDist$) for Flight 1 (July 21, 2002), second flux leg (scans 2955 to 3547). (a) Laser 1 (meters), (b) Laser 2 (meters) (c) Laser 3 (meters).



(a) Laser 1 invalid returns

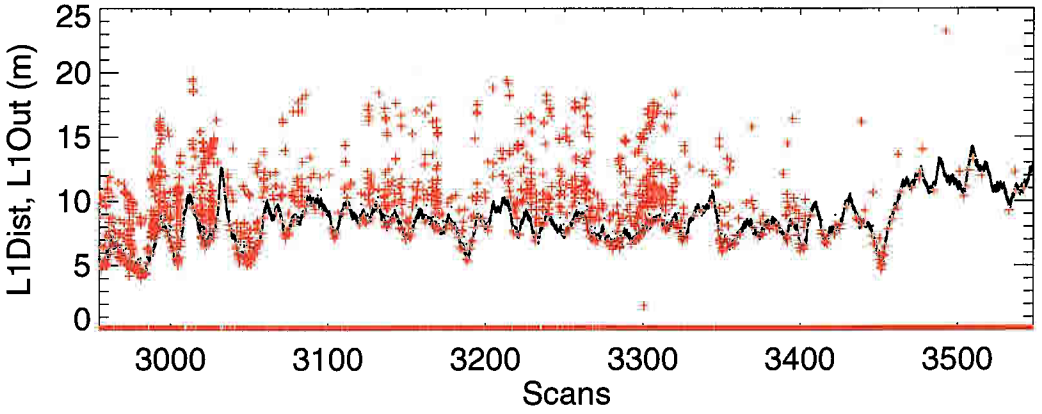


(b) Laser 2 invalid returns

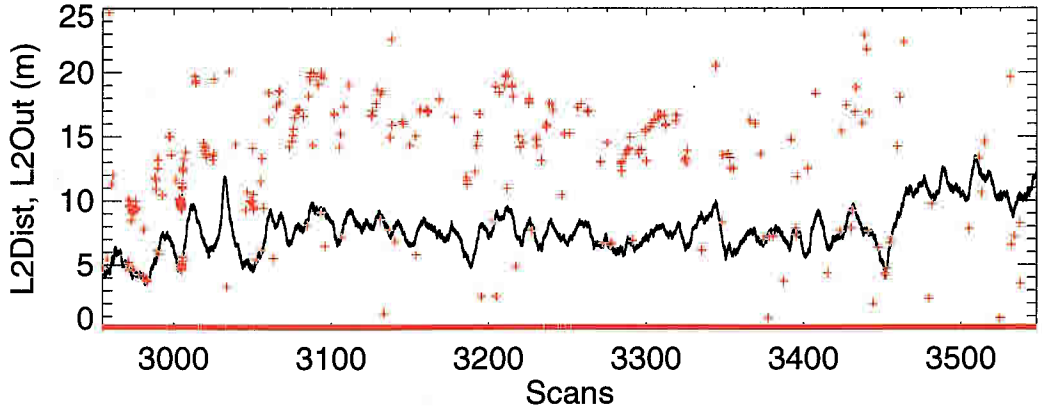


(c) Laser 3 invalid returns

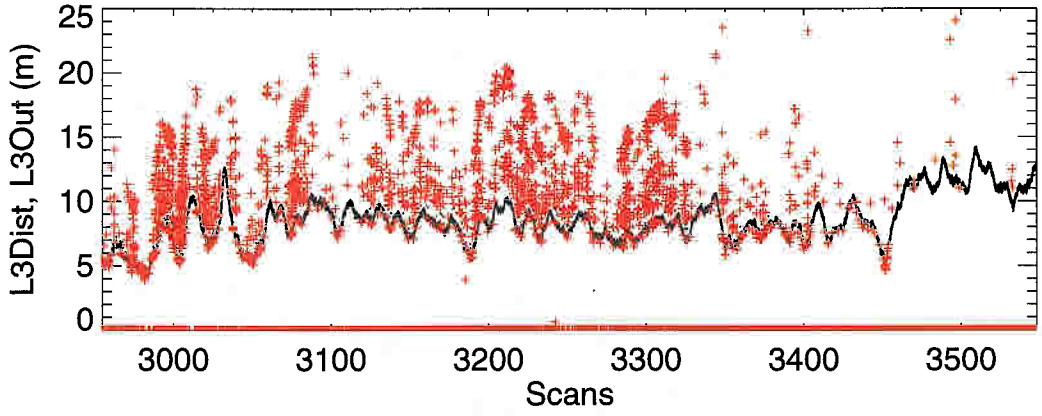
Figure 2: Original signal ($LXDist$; black) and data points identified as bad ($LXOut$; red +) due to too few returns (a) Laser 1, (b) Laser 2, (c) Laser 3.



(a) Laser 1



(b) Laser 2



(c) Laser 3

Figure 3: Original signal ($LXDist$; black) and data points identified as bad ($LXOut$; red +) due to too few returns and by distance-based outlier detection scheme (scans 2955 to 3547). (a) Laser 1, (b) Laser 2, (c) Laser 3.

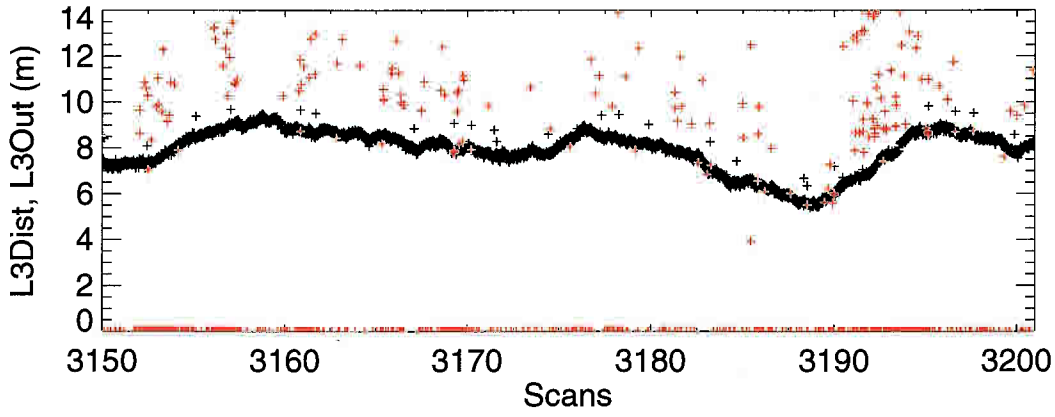


Figure 4: Laser 3 original signal ($L3Dist$; black +) and data points identified as bad ($L3Out$; red +) due to too few returns and the distance-based outlier detection scheme (scans 3150 to 3200). The clearly distinguished black '+' are the remaining outliers.

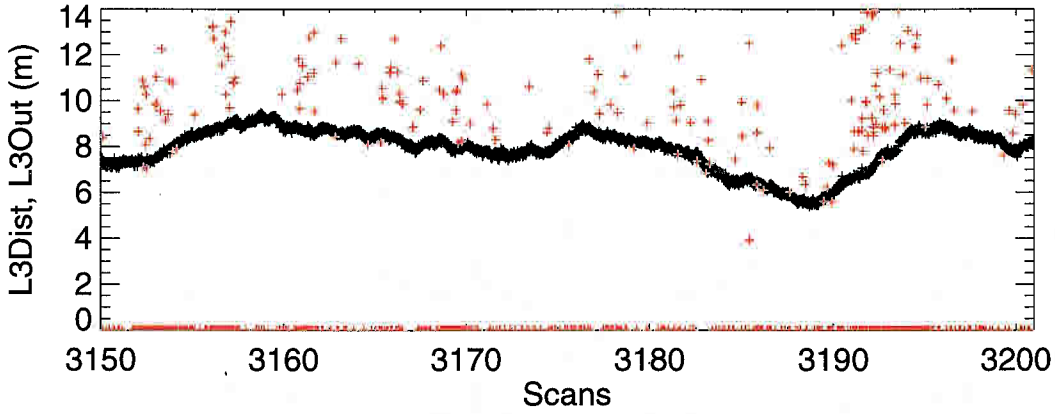


Figure 5: Laser 3 original signal ($L3Dist$; black +) and data points identified as bad ($L3Out$; red +) due to too few returns, the distance-based outlier dectection scheme, and Grubbs' test (scans 3150 to 3200).

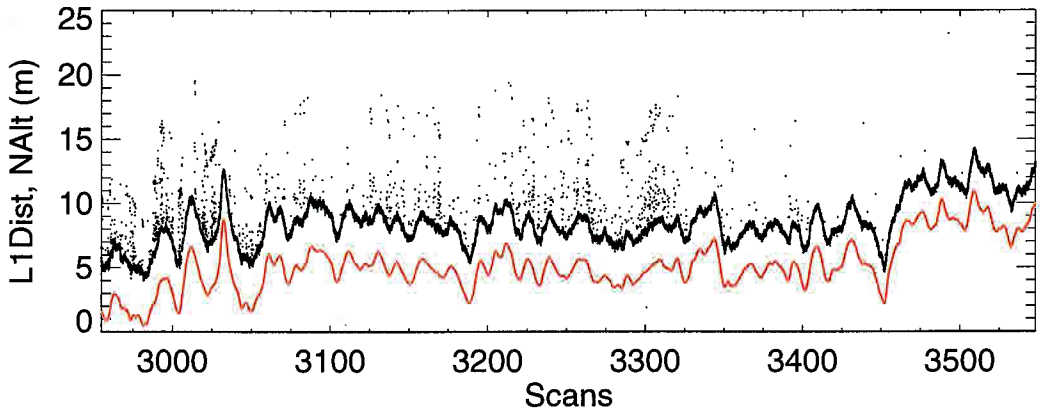


Figure 6: $L1Dist$ (black) and $NAlt$ (red) for flight 1, second flux leg (scans 2955 to 3547).

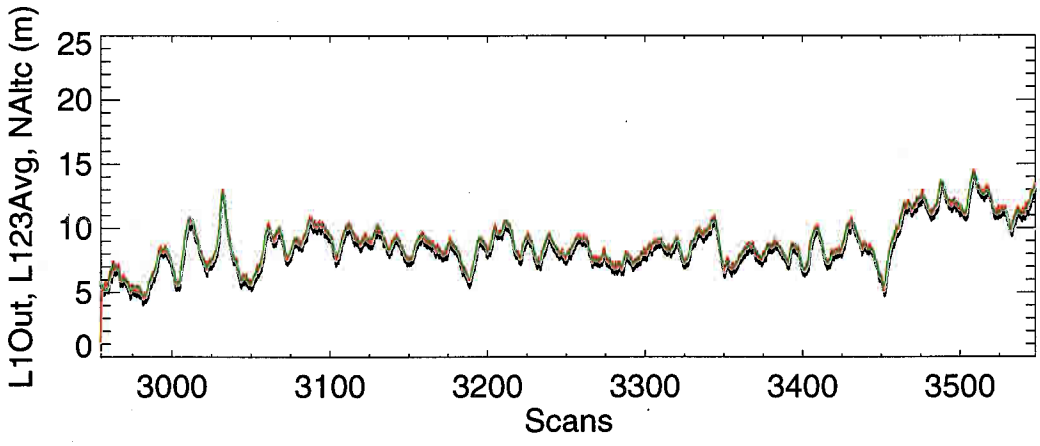


Figure 7: $L1Out$, $L123Avg$, $NAltc$ for flight 1, second flux leg (scans 2955 to 3547). Black line ($L1Out$) is the corrected laser 1 data; red ($L123Avg$) is the three laser signals averaged together; green ($NAltc$) is the offset GPS signal.

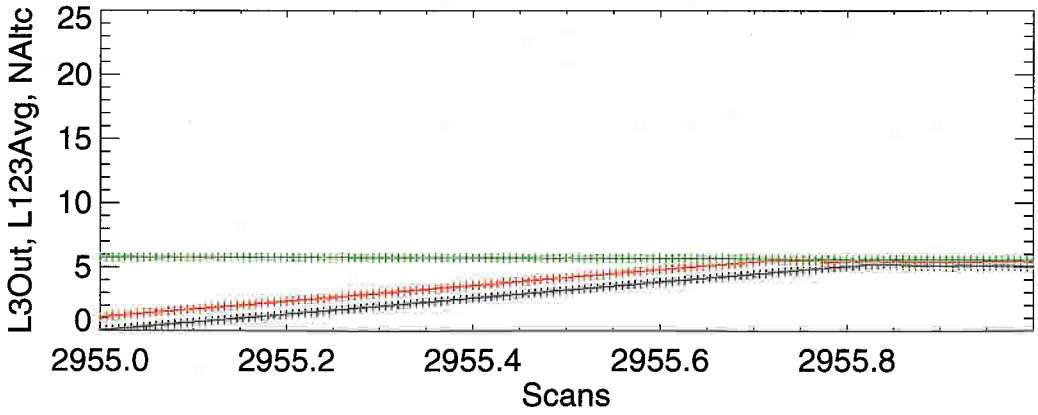
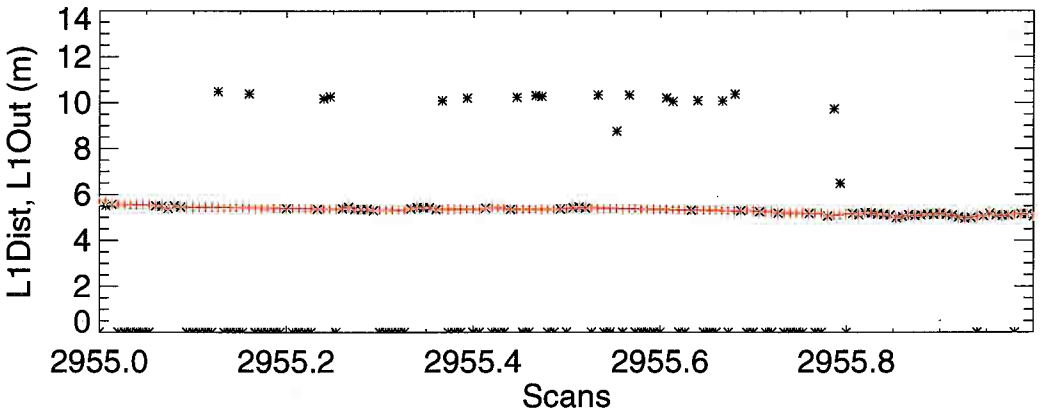
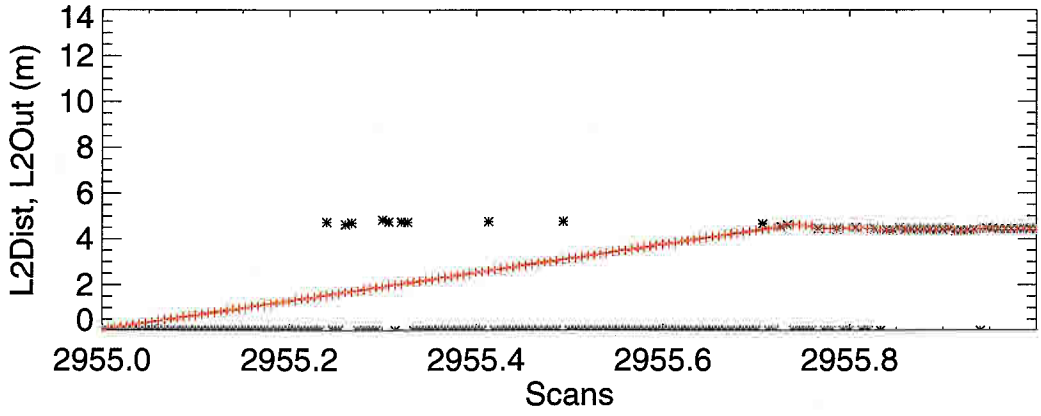


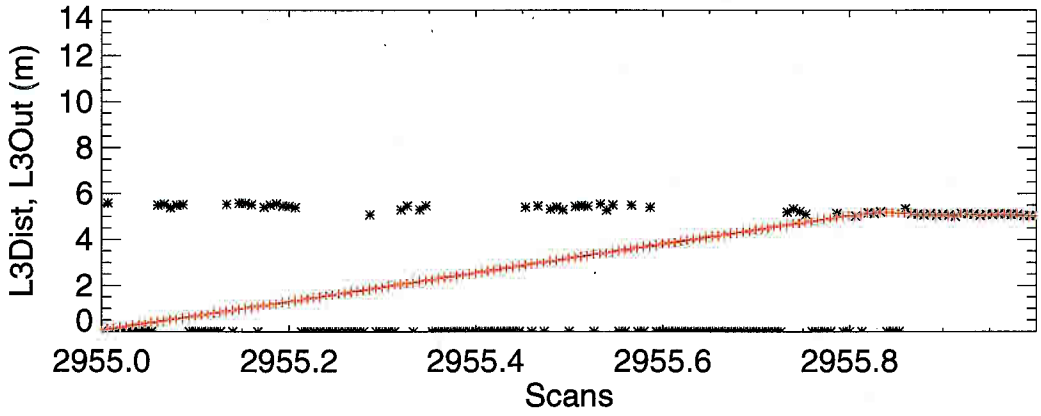
Figure 8: $L3Out$, $L123Avg$, $NAltc$ (in meters) for flight 1, second flux leg (scan 2955). Black line is the corrected laser 3 data; red is the three laser signals averaged together; green is the offset GPS signal.



(a) Laser 1



(b) Laser 2



(c) Laser 3

Figure 9: Original laser signal ($LXDist$; black) and the corresponding corrected signal ($LXOut$; red) for scan 2955. (a) Laser 1, (b) Laser 2, (c) Laser 3.

References

Crawford, T., R. McMillen, and R. Dobosy, 1993: Correcting airborne flux measurements for aircraft speed variation. *Boundary-Layer Meteorol.*, **66**, 237–245.

Crescenti, G., T. Crawford, and E. Dumas: 1999, Data report: LongEZ (N3R) participation in the 1999 Shoaling Waves Experiment (SHOWEX) spring pilot study. NOAA Technical Memorandum ERL ARL-232, Silver Spring, MD, 86 pp.

Crescenti, G., J. French, and T. Crawford: 2001, Aircraft measurements in the Coupled Boundary Layers Air-Sea Transfer (CBLAST) light wind pilot field study. NOAA Technical Memorandum OAR ARL-241, Silver Spring, MD, 82 pp.

French, J., G. Crescenti, T. Crawford, and E. Dumas: 2000, LongEZ (N3R) Participation in the 1999 Shoaling Waves Experiment (SHOWEX) Spring Pilot Study. NOAA Technical Memorandum ARL-20, Silver Spring, MD, 51 pp.

GraphPad.com: 2000, Grubbs test for detecting outliers. Internet: www.graphpad.com/calculators/GrubbsHowTo.cfm.

Knorr, E. and R. Ng: 1998, Algorithms for mining distance-based outliers in large datasets. *Proceedings of the 24th VLDB Conference*, Very Large Data Base Endowment.

Mahrt, L., D. Vickers, J. Sun, T. Crawford, G. Crescenti, and P. Frederickson, 2001: Surface stress in offshore flow and quasi-frictional decoupling. *J. Geophys. Res.*, **106**, 20629–20639.

Mourad, P.: 1999, Footprints of atmospheric phenomena in synthetic aperature radar images of the ocean surface: A review. *Air-Sea Exchange: Physics, Chemistry and Dynamics*, G. Geernaert, ed., Kluwer Academic Publishers, 269–290.

Mourad, P., D. Thompson, and D. Vandemark, 2000: Extracting fine-scale wind fields from synthetic aperture radar images of the ocean surface. *Johns Hopkins Univ., Appl. Phys. Lab. Tech. Dig.*, **21**, 108–115.

Sun, J., D. Vandemark, L. Mahrt, D. Vickers, T. Crawford, and C. Vogel, 2001: Momentum transfer over the coastal zone. *J. Geophys. Res.*, **106**, 12437–12448.

Vandemark, D., P. Mourad, S. Bailey, T. Crawford, C. Vogel, J. Sun, and B. Chapron, 2001: Measured changes in ocean surface roughness due to atmospheric boundary layer rolls. *J. Geophys. Res.*, **106**, 4639–4654.

Vickers, D., L. Mahrt, J. Sun, and T. Crawford, 2001: Structure of offshore flow. *Mon. Wea. Rev.*, **129**, 1251–1258.

Vogel, C. and T. Crawford: 1999, Exchange measurements above the air-sea interface using an aircraft. *Air-Sea Exchange: Physics, Chemistry and Dynamics*, G. Geernaert, ed., Kluwer Academic Publishers, 231–245.

A Code

This appendix contains listings of the code written for IDL Version 5.5. The variable 'datapath' is 'hard wired' in the code. This contains the directory name where the data is kept. But the 'rootname' variable must be added to 'datapath' to get the full directory specification.

```
;
; NAME: Alt    (FILE: alt.pro)
;
;-------------------------------------------------------------------------
;
; Tami Grimmett
; NRC Postdoctoral Research Associate
; NOAA/Air Resources Laboratory
; Field Research Division
; 1750 Foote Drive
; Idaho Falls, ID 83402
; Office: 208-526-2743
; Fax:    208-526-2549
; tami@noaa.inel.gov
;
;-------------------------------------------------------------------------
;
; PURPOSE:
;
;   The purpose of this procedure is to compute a good altitude for
;   the flux legs and profile manuevers.
;
; CALLING SEQUENCE:
;
;   naltc = Alt(rootname, nalt, pitch,roll,heading, $
;               L1Dist,L2Dist,L3Dist, L1Retn,L2Retn,L3Retn, $
;               L1e, L2e, L3e)
;
;
;
; EXTERNAL FUNCTIONS, PROCEDURES, AND FILES:
;
;   Error_message (from www.dfanning.com)
;   Readmkr
;   HandleBadPoints
;     DBOutliers
;     Interp
;   InterpFastData
;   CorrNAlt
;
;
; NAMED STRUCTURES:
;
;   marker
;
;
; INPUTS:
;      rootname:                root filename for data
```

```
;      nalt:                    GPS altitude data
;      pitch, roll, heading
;      L1Dist, L2Dist, L3Dist:  Uncorrected laser data
;      L1Retn, L2Retn, L3Retn:  Number of returns for each datapoint
;      L1e, L2e, L3e:           Laser data converted to earth-based coordinates
;      L1Out, L2Out, L3Out:     Raw laser data with bad points identified
;      Hz150:                   Flag to indicate if the GPS data needs
;                               to be converted to 150 Hz.
;
;
; OUTPUTS:
;
;      L1e:   Laser 1 data corrected to earth-based coordinates
;      L2e:   Laser 2 data corrected to earth-based coordinates
;      L3e:   Laser 3 data corrected to earth-based coordinates
;      naltc: Corrected altitude data  (function return)
;
; VARIABLES:
;
;
; MODIFICATION HISTORY
;
;    17 Dec 2001   Started
;    05 Mar 2002   Completed & code clean up
;
;-------------------------------------------------------------------------
Function Alt, rootname, nalt, pitch, roll, heading, $
         L1Dist, L2Dist, L3Dist, L1Retn, L2Retn, L3Retn, $
         L1e, L2e, L3e, L123avg, L1Out,L2Out,L3Out, Hz150=nHz150
;
;    Note start time of this procedure
   strtTime=systime(/seconds)
;
;    Set constants
;
   nHz50 = 50.
   lsize = SIZE(L1Dist)


;-------------------------------------------------
;
;  Interpolate slow data (50 Hz) into fast data (150 Hz)
;
   pitch     = REFORM(pitch,    [nHz50,lsize[2]])
   roll      = REFORM(roll,     [nHz50,lsize[2]])
   heading   = REFORM(heading,  [nHz50,lsize[2]])
   nalt      = REFORM(nalt,     [nHz50,lsize[2]])

   pitch_f   = InterpFastData(pitch, nHz150) ; 50 Hz
   roll_f    = InterpFastData(roll, nHz150) ; 50 Hz
   heading_f = InterpFastData(heading, nHz150) ; 50 Hz
   IF KEYWORD_SET(nHz150) THEN BEGIN
      nalt_f = InterpFastData(nalt, nHz150)
      nalt   = nalt_f
```

```
   ENDIF


;-------------------------------------------------
;
;  Read marker file
;  Check if there are any flux legs to correct
;
   marker = ReadMKR(rootname,nflux=nflux)

   IF nflux LE 0 THEN GOTO, endit

;-------------------------------------------------
;
;  Create arrays to hold the data transformed to earth based coordinates
;
   L1e = FLTARR(Lsize[1],Lsize[2])
   L2e = FLTARR(Lsize[1],Lsize[2])
   L3e = FLTARR(Lsize[1],Lsize[2])

;
;  Compute the matrix that transforms airplane coordinates to earth coordinates
;
;   matrix = Transmatrix(roll_f,pitch_f,heading_f)


   FOR l = 0L, 2 DO BEGIN                    ; for each laser
      Learth = FLTARR(LSize[1],LSize[2])


;-------------------------------------------------
;
;  Create arrays to hold data with outliers removed
;  Get rid of bad data (dropouts and outliers)
;
      IF l EQ 0 THEN LOut = L1Dist
      IF l EQ 1 THEN LOut = L2Dist
      IF l EQ 2 THEN LOut = L3Dist
;

      PRINT, '   Laser ',l+1
      FOR m=0L,nflux-1 DO BEGIN
         fluxStrt = marker.flux[m,0]
         fluxEnd = marker.flux[m,1]
         PRINT, 'Alt: flux leg #', m+1
         IF l EQ 0 THEN BEGIN
            HandleBadPoints, L1Dist, LOut, L1Retn, nHz150, fluxStrt, fluxEnd
         ENDIF ELSE IF l EQ 1 THEN BEGIN
            HandleBadPoints, L2Dist, LOut, L2Retn, nHz150, fluxStrt, fluxEnd
         ENDIF ELSE IF l EQ 2 THEN BEGIN
            HandleBadPoints, L3Dist, LOut, L3Retn, nHz150, fluxStrt, fluxEnd
         ENDIF

         PRINT, ' '
```

```
      ENDFOR


;-------------------------------------------------
;
;  Correct altitudes to earth-based coordinates
;  1,2, & 3 refer to lasers 1,2, & 3
;
;     Define displacement matrix
;     (These are the distances from the bat probe to the lasers.)
      LD = [ [-3.6871, -2.8829, -3.6871], $
             [ 0.4707,  0.0000, -0.4707], $
             [-0.3048, -1.0148, -0.3048] ]
;
;     Translate each lasers measurement to the bat probe
;
      Learth[*,*] = LOut[*,*] - LD[1,2] - $
                    LD[1,0] * tan(pitch_f/!radeg) - $
                    LD[1,1] * tan(roll_f/!radeg)

;
;     Do the transformation for the z-coordinate only
;
      FOR m=0L,nflux-1 DO BEGIN
         fluxStrt = marker.flux[m,0]
         fluxEnd = marker.flux[m,1]

         Learth[*,fluxStrt:fluxEnd] = TEMPORARY(Learth[*,fluxStrt:fluxEnd]) * $
                         cos(pitch_f[*,fluxStrt:fluxEnd]/!radeg) * $
                         cos(roll_f[*,fluxStrt:fluxend]/!radeg)

      ENDFOR
;
;     Store corrected and transformed laser values
;
      IF l EQ 0 THEN L1e = Learth
      IF l EQ 1 THEN L2e = Learth
      IF l EQ 2 THEN L3e = Learth

      IF l EQ 0 THEN L1Out = LOut
      IF l EQ 1 THEN L2Out = LOut
      IF l EQ 2 THEN L3Out = LOut
   ENDFOR

;-------------------------------------------------
;
;  Average the three laser readings together to get one reading
;
   IF KEYWORD_SET(nHz150) THEN BEGIN
      L123avg = DBLARR(nHz150,lsize[lsize[0]])
      nscans = lsize[lsize[0]]
      FOR j=0L,nscans-1 DO BEGIN
         FOR i=0L,nHz150-1 DO BEGIN
            L123avg[i,j] = MEDIAN( [L1e[i,j], L2e[i,j], L3e[i,j]] )
```

```
      ENDFOR
   ENDFOR

ENDIF ELSE BEGIN
   L123avg = DBLARR(nHz50 ,lsize[lsize[0]])
   nscans = lsize[lsize[0]]
   FOR j=0L,nscans-1 DO BEGIN
      FOR i=0L,nHz50-1 DO BEGIN
         i1 = i*3.
         i2 = i*3. + 2.

         tmp = [ L1e[i1:i2,j], L2e[i1:i2,j], L3e[i1:i2,j] ]
         L123avg[i,j] = MEDIAN(tmp)
      ENDFOR
   ENDFOR
ENDELSE

;-------------------------------------------------
;
;  Compute the mean offset between L123avg (the average laser signal) and
;  NAlt (the GPS signal) to apply as a correction to the GPS signal
;  Apply the correction
;
;
;  Define averaging length
;
   nSecMean = 60
;
;  Define structure to hold offsets
;
   struct = {offset_str, nOffsets:0L, values:FLTARR(30)}
   onestr = {offset_str}
   offset = REPLICATE(onestr, nflux)
;
;  Correct the altitude
;
   naltc = CorrNAlt( nSecMean, marker, nalt, L123avg, offset, Hz150=nHz150 )


;-------------------------------------------------
;
;  Apply a linearly weighted correction for the profile
;  If the profile is in between two flux legs, the correction
;  to the profile is weighted by the corrections for the two
;  flux legs.  For example, on the left side, the first point
;  of the profile is given a correction which is 100% of the
;  offset for the flux leg on that side and 0% from the right flux leg.
;
   sizepro = SIZE(marker.profile)
   nprof = sizepro[1]
;  Check if there are any profiles
   IF nprof LE 0 THEN BEGIN
;    If no, print a message indicating there are no profiles
      PRINT, 'ALT: There are no profiles to correct, nprof=',nprof
```

```
;    If yes,


   ENDIF ELSE BEGIN
;        print a message indicating how many profiles there are
      PRINT, 'ALT: There are', nprof, ' profiles to correct'
;           determine if there are any flux legs
      IF nflux LE 0 THEN BEGIN
;             If no, print a message indicating there are no flux legs
         PRINT, 'ALT: There are no flux legs to correct profiles, nflux=',nflux
;             If yes,


      ENDIF ELSE BEGIN
;               for each profile
         FOR ip=0L,nprof-1 DO BEGIN
            profStrt = marker.profile[ip,0]
            profEnd = marker.profile[ip,1]


;                determine if profile is before the first flux leg
;                  If yes, add 1st flux leg's offset to profile
;                  If no, continue
            IF marker.profile[ip,1] LE marker.flux[0,0] THEN BEGIN
               naltc[*,profStrt:profEnd] = nalt[*,profStrt:profEnd] + $
                                           offset[0].values[0]


;                else determine if profile is after the last flux leg
;                  If yes, add last flux leg's offset to profile
;                  If no, continue
            ENDIF ELSE IF marker.profile[ip,0] GE marker.flux[nflux-1,1] THEN BEGIN
               noffsets = offset[nflux-1].noffsets
               naltc[*,profStrt:profEnd] = nalt[*,profStrt:profEnd] + $
                                           offset[nflux-1].values[noffsets-1]



            ENDIF ELSE BEGIN
;                else determine if profile is between flux legs
;                  If no, print a message indicating there is something
;                  wrong with this profile
;                  If yes, add a linear offset to the profile
               Catch, theError
               IF theError NE 0 THEN BEGIN
                  Catch, /Cancel
                  ok = Error_Message(/Traceback)
               ENDIF

               ix =0L
               locFound = 0
               WHILE (locFound NE 1) DO BEGIN
                  fluxStrt = marker.flux[ix,0]
                  fluxEnd = marker.flux[ix,1]
```

```
                  IF (marker.profile[ip,0] GE marker.flux[ix,1]) AND $
                    (marker.profile[ip,1] LE marker.flux[ix+1,0]) THEN BEGIN
                     locfound = 1
                     npts = (profEnd - profStrt + 1) * nHz150
                     nStrt = profStrt * nHz150
                     nEnd = profEnd * nHz150


                     FOR pt = 0L, npts-1 DO BEGIN
                        n = nStrt + pt
                        noffsets = offset[nflux-1].noffsets
                        totoffset = (n - nStrt)/npts*offset[ix+1].values[0]+ $
                          (nEnd - n)/npts*offset[ix].values[noffsets-1]
                        naltc[n] = nalt[n] + totoffset
                     ENDFOR
                  ENDIF

                  ix = ix+1
               ENDWHILE
            ENDELSE          ; end of check for profile location in relation to flux legs
         ENDFOR              ; end of each profile loop
      ENDELSE                ; end of flux check
   ENDELSE                   ; end of nprof check (i.e. there were profiles to correct)


endit:
;
;    Compute the execution time of this procedure
;
   endTime=systime(/seconds)
   elaps = endTime-strtTime
   minut = floor(elaps/60.)
   sec = elaps - (minut*60.)
   PRINT, format = '("ALT: Elapsed Execution Time (mm:ss):  ", i2, ":", i2)', minut, se

   RETURN, naltc
END
```

```
;+
; NAME:  ReadMKR
;
;
; PURPOSE:
;
;   This function reads a marker file (.mkr) and returns
;   a structure containing the marker tag and start/stop
;   scans for marker pairs.
;
;
; CALLING SEQUENCE:
;
;   ReadMKR, rootname, nflux=nflux, help=help, nprofile=nprof, nevent=nevnt
;
; INPUTS:
;
;   rootname     The root name for the flight data
;
; OUTPUTS:
;
;   nflux:       Number of flux legs in flight
;   nprof:       Number of profiles in flight
;   nevent:      Number of other events in flight
;   marker:      A structure
;
; NAMED STRUCTURES:
;
;    marker
;       flux       LONG   Array[*,2]
;       profile    LONG   Array[*,2]
;       event      LONG   Array[*]
;
; MODIFICATION HISTORY:
;
;   05 Mar 2002  TKG  Updated  (based on version from Jeff French)
;-
;-----------------------------------------------------------------
FUNCTION ReadMKR, rootname, nflux=nflux, help=help, nprofile=nprof, nevent=nevnt

IF KEYWORD_SET(help) THEN BEGIN
  print,'SYNTAX'
  print,'  Result = ReadMKR(rootname)'
  print,'ARGUMENTS'
  print,'  rootname'
  print,'    a string containing the name (and path if'
  print,'    not in current directory) of the marker file'
  print,'    to read'
  print,' '
  print,'EXAMPLE'
  print,'  Create a structure containing the start/stop'
  print,'  scans for flux & profile runs, and scan numbers'
  print,'  for event switches.  If there are no events (or'
  print,'  flux/profiles) the tag is returned equal to -1'
```

```
  print,' '
  print,'    marker = ReadMKR(rootname)'
  print,' '
  print,'    help,marker,/struc'
  print,'    ** Structure <81b492c>, 3 tags, length=48, data length=48, refs=1'
  print,'       FLUX            LONG      Array[4, 2]'
  print,'       PROFILE         LONG      Array[1, 2]'
  print,'       EVENT           LONG      Array[2]'
  print,' '
  print,'    print,marker.event'
  print,'         2491        2622'
  return,-1
endif

;
; define data path and marker file absolute filename
;
 datapath = '/home/tgrimmet/Research/Air-Sea/CBLAST-Low/Flights/'
 datafile_mkr = datapath + rootname + '.mkr'

; open file
  OPENR, fnum, datafile_mkr, /get_lun


; begin reading in marker file, line by line searching for 'OPENED'
  line=''
  REPEAT begin
    READF, fnum, line
  ENDREP UNTIL (EOF(fnum) OR (STRPOS(line,'OPENED') gt 0))

; now we are into the events, fluxes, etc....
  proon = 0 && flxon = 0 && ipro = 0 && iflx = 0 && ievt = 0
  flux = lonarr(100,2) && profile = lonarr(100,2) && event = lonarr(100)
  REPEAT begin
    READF, fnum, line

    ; look for marker switch turning on/off or blank spaces
    CASE (STRMID(line,0,3)) OF

      'EVT' : begin
        event[ievt] = LONG(STRMID(line,7,5))
        ievt = ievt + 1
      end

      'FLX' : begin
        flux[iflx,0] = LONG(STRMID(line,7,5))
        flxon = 1
        proon = 0     ; sanity check for messed up markers
      end

      'PRO' : begin
        profile[ipro,0] = LONG(STRMID(line,7,5))
        proon = 1
        flxon = 0     ; sanity check for messed up markers
```

```
      end

      '    ' : begin
        if (STRMID(line,4,2) eq ' 0') then begin
          if (flxon) then begin
            flux[iflx,1] = LONG(STRMID(line,7,5))
            flxon = 0
            iflx = iflx + 1
          endif
          if (proon) then begin
            profile[ipro,1] = LONG(STRMID(line,7,5))
            proon = 0
            ipro = ipro + 1
          endif
        endif
      end

    ELSE : tmp=0   ; do nothing

    ENDCASE

  ENDREP UNTIL (EOF(fnum) OR (STRPOS(line,'CLOSED') gt 0))


  if (iflx gt 0) then $
        flux = flux[0:iflx-1,*] $
  else flux = -1

  if (ipro gt 0) then $
        profile = profile[0:ipro-1,*] $
  else profile = -1

  if (ievt gt 0) then $
        event = event[0:ievt-1,*] $
  else event = -1

  nflux = iflx
  nprof = ipro
  nevnt = ievt

; close file & deallocate file unit
  FREE_LUN, fnum


  marker = { flux:flux, profile:profile, event:event}


  RETURN, marker

end
```

```
;+
; NAME:  HandleBadPoints
;
;
; PURPOSE:
;
;   Find dropout points and outliers using a distance-based method
;   Interpolate new values using linear interpolation
;
;
; CALLING SEQUENCE:
;
;   HandleBadPoints, LDist, LOut, LRetn, Hz, scan1, scan2, xOutTot
;
; INPUTS:
;
;   LDist
;   LOut
;   LRetn
;   Hz
;   scan1, scan2
;
; OUTPUTS:
;
;   xOutTot
;
; MODIFICATION HISTORY:
;
;   05 Mar 2002  TKG  Original
;-
;---------------------------------------------------------------------
PRO HandleBadPoints, LDist, LOut, LRetn, Hz, scan1, scan2

   xOutliers = 0
   xOutTot = 0

;
;  Find outliers
;
   FOR j = scan1,scan2 DO BEGIN
      noutl = DBOutliers(LDist, LRetn, LOut, Hz, j, xOutliers)

      IF noutl GT 0 THEN BEGIN

         IF (SIZE(xOutTot))[0] EQ 0 THEN BEGIN
            xOutAbs = (j*Hz) + xOutliers  ; calculate absolute address
            xOutTot = xOutAbs

         ENDIF ELSE BEGIN
            xOutAbs = (j*Hz) + xOutliers  ; calculate absolute address
            xOutTot = [xOutTot, xOutAbs]  ; concatenate arrays

         ENDELSE
```

```
      ENDIF
   ENDFOR

;
;  Put arrays into the [datapoint*scan] format for interpolation
;
   szLDist = SIZE(LDIST)
   IF szLDist[0] EQ 2 THEN BEGIN
      nscans = szLDist[2]
      LDist = REFORM(LDist,[Hz*nscans])
      LOut = REFORM(LOut,[Hz*nscans])
   ENDIF

;
;  Interpolate new values for the bad points
;
   Interp, LDist, LOut

;
;  Put arrays back into the [datapoint, scan] format
;
   szLDist = SIZE(LDIST)
   IF szLDist[0] EQ 1 THEN BEGIN
      nscans = szLDist[1] / Hz
      LDist = REFORM(LDist,[Hz,nscans])
      LOut = REFORM(LOut,[Hz,nscans])
   ENDIF

END
```

```
;-------------------------------------------------
;+
; NAME:  DbOutliers
;
;
; PURPOSE:
;
;    To determine and mark all invalid datapoints and outliers in a
;    scan.  Invalid datapoints are defined as those having fewer than
;    3 returns. Outliers are defined as points that lie at a greater
;    distance than some specified distance from a selected point in the
;    distribution that is defined as good.
;
;
; CALLING SEQUENCE:
;
;    noutliers = DBOutliers(LDist, LRetn, LOut, Hz, scan, xoutliers)
;
; INPUTS:
;
;    LDist:    'Raw' laser data
;    LRetn:    Number of laser returns
;    LOut:     Laser data with bad points identified
;    Hz:       Rate data was acquired (Hz)
;    scan
;
; OUTPUTS:
;
;    xoutliers:  array index of bad points
;    noutliers:  number of bad points found (function return)
;
; MODIFICATION HISTORY:
;
;    05 Mar 2002  TKG  Original
;-
;-------------------------------------------------
FUNCTION DBOutliers, LDist, LRetn, LOut, Hz, scan, xoutliers

;-------------------------------------------------
;
;  Initialize outliers variable
;
   xOutliers = -1


;-------------------------------------------------
;
;  Mark those points that have fewer than 3 returns as outliers
;
   xOutliers = where(LRetn[*,scan] LT 3)
   remaining = where(LRetn[*,scan] GE 3)


;-------------------------------------------------
```

```
;
;  Check for scans that have only a few good points (in this case, LE
;  3 pts).  If this is the case, set the whole scan as being bad and
;  all points are outliers and return.
;
   IF (SIZE(remaining))[1] LE 3 OR (SIZE(remaining))[0] EQ 0 THEN BEGIN
      LOut[*,scan]=!Values.F_NaN
      xOutliers = where(FINITE(LOut[*,scan]) GT -1000.)
      RETURN, N_ELEMENTS(xOutliers)
   ENDIF


;-------------------------------------------------
;
;  Set the distance from a point that will be considered an outlier.
;  If there were many points with less than 3 returns,
;  probably most of the scatter in the data is removed, and a larger distance must
;  be used to not identify good points.
;
;
   sr = SIZE(remaining)
   IF sr[1] GE 143 THEN BEGIN
      D=0.5
   ENDIF ELSE BEGIN
      D=1.2
   ENDELSE


;-------------------------------------------------
;
;  Determine the number of remaining points
;
   npts = sr[sr[0]+2]

;
;  Create array to hold the computed distances
;
   dist = FLTARR(npts)


;-------------------------------------------------
;
;  Compute distances
;

   mn = moment(ldist[remaining,scan])
   sd = sqrt(mn[1])
   min = mn[0]-sd
   max = mn[0]+sd

;  Find the first good point.  The first points in a scan may be outliers.
   q=0
   WHILE LDist[remaining[q],scan] LT min OR LDist[remaining[q],scan] GT max DO BEGIN
      q = q+1
```

```
   ENDWHILE


   dist = SQRT( (remaining/Hz-remaining[q]/Hz)^2 + $
                (LDist[remaining,scan]-LDist[remaining[q],scan])^2 )


;-------------------------------------------------
;
;  Determine outliers.  Concantenate all outliers together.  Mark the
;  outliers with NaN.
;

   xo = where(dist[*] GT D)

   IF xoutliers[0] NE -1 THEN BEGIN
      IF xo[0] NE -1 THEN BEGIN
         xoutliers = [ xoutliers, remaining[xo[*]] ]
      ENDIF
   ENDIF ELSE BEGIN
      IF xo[0] NE -1 THEN BEGIN
         xoutliers = xo
      ENDIF
   ENDELSE
   xoutliers=xoutliers[sort(xoutliers)]

   IF (SIZE(xoutliers))[0] NE 0 THEN LOut[xoutliers,scan]=!Values.F_NaN


;-------------------------------------------------
;
;  Do a second pass through the data
;  Grubbs Test

   remaining = WHERE( FINITE(LOut[*,scan]) NE FINITE(!Values.F_NaN))
   sr = SIZE(remaining)

   IF sr[1] GT 6 THEN BEGIN
      mn = moment(ldist[remaining,scan])
      sd = sqrt(mn[1])
      z = ABS(mn[0]-LOut[remaining,scan])/sd

      siglevl = ((sr[1]-1)/SQRT(sr[1])) * 0.15
      iz = WHERE(z GT siglevl)

;-------------------------------------------------
;
;  Determine outliers.  Concantenate all outliers together.  Mark the
;  outliers with NaN.
;

      IF xoutliers[0] NE -1 THEN BEGIN
         IF iz[0] NE -1 THEN BEGIN
            xoutliers = [ xoutliers, remaining[iz] ]
```

```
         ENDIF
      ENDIF ELSE BEGIN
         IF xo[0] NE -1 THEN BEGIN
            xoutliers = iz
         ENDIF
      ENDELSE
      xoutliers=xoutliers[sort(xoutliers)]

      IF (SIZE(xoutliers))[0] NE 0 THEN LOut[xoutliers,scan]=!Values.F_NaN
   ENDIF

;-------------------------------------------------
;
;  Check if there are enough good points left
;

   IF (SIZE(xoutliers))[1] GE 144 THEN LOut[*,scan]=!Values.F_NaN

;
;  Return the number of outliers found
;
   RETURN, N_ELEMENTS(xoutliers)

END
```

```
;+
; NAME:  Interp
;
;
; PURPOSE:
;
;   To linearly interpolate new values for points marked as bad.
;
; CALLING SEQUENCE:
;
;   Interp, LDist, LOut, xOutliers
;
; INPUTS:
;
;   LDist:      'Raw' laser data
;   LOut:       Laser data with bad points identified
;   xOutliers:  array index of bad points
;
; OUTPUTS:
;
;   LOut:       Corrected laser data
;
; REFERENCES:
;
;   "Algorithms for Mining Distance-Based Outliers in Large Datasets"
;   Edwin M. Knorr and Raymond T. Ng, University of British Columbia
;   Proceedings of the 24th VLDB Conference, New York, USA, 1998
;
; MODIFICATION HISTORY:
;
;   05 Mar 2002  TKG  Original
;-
;-------------------------------------------------------------------------
PRO Interp, LDist, LOut, xOutliers
;
;  Find all the outliers
;
   xOutliers = WHERE(FINITE(LOut) EQ FINITE(!Values.F_NaN))
;
;  Make sure the xOutliers array is sorted
;
   sortxOut = xOutliers[SORT(xOutliers)]


;-------------------------------------------------
;
;  Set constants
;
   pcntDiff = 0.03
   nOutliers = N_ELEMENTS(xOutliers)

;
;  Check if there are any outliers
;
```

```
   IF xoutliers[0] EQ -1 THEN return

;-------------------------------------------------
;
;  One point on either side of 'j', which is the location
;  where we need an interpolated value, is needed.  The points are
;      L = left
;      R = right
;
   FOR j = 0L, nOutliers-1 DO BEGIN
      jj = LONG( xOutliers[j] )

;    Find L [ideally this would be (j-1)]
      IF jj EQ 0 THEN BEGIN
         L = jj
      ENDIF ELSE BEGIN
         L = jj-1
         WHILE (FINITE(LOut[L]) EQ FINITE(!Values.F_NaN) AND L GT 0) DO L=L-1
      ENDELSE

;    Find R [ideally this would be (j+1)]
      R = jj+1
      WHILE (FINITE(LOut[R]) EQ FINITE(!Values.F_NaN)) DO BEGIN
         IF  R EQ  (N_ELEMENTS(LOut)-1) THEN BREAK
         R=R+1
      ENDWHILE


;-------------------------------------------------
;
;   After finding good values on either side of the bad point
;   or outlier, linear interpolation is done.
;
      npts = R - L + 1
      xGood = [L, R]
      yGood = LOut[xGood]
      yInterp = INTERPOL(yGood,npts)


;-------------------------------------------------
;
;   Now we need to check whether the interpolated value is better
;   or not.  If the interpolated point is within 3% of the original
;   data point, the original data point is retained.  Otherwise
;   the interpolated value is used.
;
      IF  yInterp[jj-L] LT LDist[jj] THEN BEGIN
         IF LDist[jj] NE 0. THEN BEGIN
            IF yInterp[jj-L]/LDist[jj] LT 1.-pcntDiff THEN BEGIN
               LOut[jj] = yInterp[jj-L]
            ENDIF ELSE BEGIN
               LOut[jj] = LDist[jj]
            ENDELSE
         ENDIF ELSE IF LDist[jj] EQ 0. THEN BEGIN
```

```
            LOut[jj] = yInterp[jj-L]
         ENDIF


      ENDIF ELSE IF yInterp[jj-L] GT LDist[jj] THEN BEGIN
         IF LDist[jj] NE 0. THEN BEGIN
            IF yInterp[jj-L]/LDist[jj] GT 1.+pcntDiff THEN BEGIN
               LOut[jj] = yInterp[jj-L]
            ENDIF ELSE BEGIN
               LOut[jj] = LDist[jj]
            ENDELSE
         ENDIF ELSE IF LDist[jj] EQ 0. THEN BEGIN
            LOut[jj] = yInterp[jj-L]
         ENDIF


      ENDIF ELSE IF yInterp[jj-L] EQ LDist[jj] THEN BEGIN
         LOut[jj] = LDist[jj]
      ENDIF


   ENDFOR   ; end of loop for each outlier

   RETURN
END
```

```
;+
;  NAME: InterpFastData
;
;  PURPOSE:
;
;    To create 150 Hz data from 50 Hz data
;
; CALLING SEQUENCE:
;
;   InterpFastData, slowvar, fastHz
;
; INPUTS:
;
;   slowvar:    Data to be interpolated to faster rate
;   Hz:         Rate data is to be interpoated to
;
; OUTPUTS:
;
;   fastvar:    Data interpolated to faster rate
;
; MODIFICATION HISTORY:
;
;   05 Mar 2002  TKG  Original
;-
;-------------------------------------------------
FUNCTION InterpFastData, slowvar, fastHz

   nsize = SIZE(slowvar)
;
;     For 1 Hz data
;
   IF nsize[0] EQ 1 THEN BEGIN

      nScan = nSize[1]
      fastvar = DBLARR(fastHZ, nScan)
      dt = 1./fastHz

      FOR j=0L,nScan-1 DO BEGIN
         fastvar[0,j] = slowvar[j]
         FOR i=1,fastHz-1 DO BEGIN
            fastvar[i,j] = fastvar[i-1,j] + dt
         ENDFOR
      ENDFOR
;
;       Greater than 1 Hz data
;
   ENDIF ELSE IF nsize[0] EQ 2 THEN BEGIN
      nScan = nSize[2]
      fastvar = FLTARR(fastHZ, nScan)
      FOR j=0L,nScan-1 DO BEGIN
         fastvar[*,j] = INTERPOL(slowvar[*,j], fastHz)
      ENDFOR

   ENDIF
```

```
   RETURN, fastvar
END
```

```
;-------------------------------------------------
;+
; NAME: CorrNAlt
;
;
; PURPOSE:
;
;  Compute the mean offset between zavg (the average laser signal) and
;  NAlt (the GPS signal) to apply as a correction to the GPS signal.
;  Apply the correction.
;
;
; CALLING SEQUENCE:
;
;   naltc = CorrNAlt(nSecMean, marker, nalt, zavg, offset, Hz150=nHz150)
;
; INPUTS:
;
;   nSecMean:    Length of time for which the offset will be calculated
;   marker:      A structure
;   nalt:        GPS data
;   zavg:        Averaged laser data
;
; KEYWORD PARAMETERS:
;
;   nHz150:
;
; OUTPUTS:
;
;   offset:      The computed difference between zavg and nalt
;   naltc:       The corrected GPS signal (function return)
;
; NAMED STRUCTURES:
;
;    marker
;       flux        LONG    Array[*,2]
;       profile     LONG    Array[*,2]
;       event       LONG    Array[*]
;
;
; MODIFICATION HISTORY:
;
;    05 Mar 2002  TKG  Original
;-
;-------------------------------------------------
FUNCTION CorrNAlt, nSecMean, marker, nalt, zavg, offset, Hz150=nHz150
;
;  Determine how many offsets there will be for the averaging length
;  nSecMean
;
   noffsets = 0
   sizemkr = SIZE(marker.flux)
   nflux = sizemkr[1]            ; no. of flux legs
   FOR m=0L,nflux-1 DO BEGIN
```

```
      fluxStrt = marker.flux[m,0]
      fluxEnd = marker.flux[m,1]
      noffsets = noffsets + (fluxEnd - fluxStrt + 1)/nSecMean
      offset[m].noffsets = (fluxEnd - fluxStrt + 1)/nSecMean
   ENDFOR

   naltc = nalt


   FOR m=0L,nflux-1 DO BEGIN
      fluxStrt = marker.flux[m,0]
      fluxEnd = marker.flux[m,1]

      i1 = fluxStrt
      i2 = i1 + nSecMean
      ict = 0

      FOR j = 0L, offset[m].noffsets-2 DO BEGIN

         offset[m].values[ict] = MEDIAN(zavg[*,i1:i2] - nalt[*,i1:i2])
         naltc[*,i1:i2] = nalt[*,i1:i2] + offset[m].values[ict]

         ict = ict + 1
         i1 = i2
         i2 = i1 + nSecMean
      ENDFOR
;
;     Handle last interval which will be greater than nSecMean in
;     length
;
      i2 = fluxEnd
      offset[m].values[ict] = MEAN(zavg[*,i1:i2] - nalt[*,i1:i2])
      naltc[*,i1:i2] = nalt[*,i1:i2] + offset[m].values[ict]

   ENDFOR

   RETURN, naltc
END
```

```
;+
; NAME:
;    ERROR_MESSAGE
;
; PURPOSE:
;
;    The purpose of this function  is to have a device-independent
;    error messaging function. The error message is reported
;    to the user by using DIALOG_MESSAGE if widgets are
;    supported and MESSAGE otherwise.
;
; AUTHOR:
;
;   FANNING SOFTWARE CONSULTING
;   David Fanning, Ph.D.
;   1645 Sheely Drive
;   Fort Collins, CO 80526 USA
;   Phone: 970-221-0438
;   E-mail: davidf@dfanning.com
;   Coyote's Guide to IDL Programming: http://www.dfanning.com/
;
; CATEGORY:
;
;    Utility.
;
; CALLING SEQUENCE:
;
;    ok = Error_Message(the_Error_Message)
;
; INPUTS:
;
;    the_Error_Message: This is a string argument containing the error
;       message you want reported. If undefined, this variable is set
;       to the string in the !Error_State.Msg system variable.
;
; KEYWORDS:
;
;    NONAME: If this keyword is set the name of the calling routine
;       is not printed along with the message.
;
;    TRACEBACK: Setting this keyword results in an error traceback
;       being printed to standard output with the PRINT command.
;
;    In addition, any keyword appropriate for the MESSAGE or DIALOG_MESSAGE
;    routines can also be used.
;
; OUTPUTS:
;
;    Currently the only output from the function is the string "OK".
;
; RESTRICTIONS:
;
;    The "Warning" Dialog_Message dialog is used by default. Use keywords
;    /ERROR or /INFORMATION to select other dialog behaviors.
;
; EXAMPLE:
;
;    To handle an undefined variable error:
;
;    IF N_Elements(variable) EQ 0 THEN $
;       ok = Error_Message('Variable is undefined', /Traceback)
;
; MODIFICATION HISTORY:
;
;    Written by: David Fanning, 27 April 1999.
;    Added the calling routine's name in the message and NoName keyword. 31 Jan 2000. DWF.
;    Added _Extra keyword. 10 February 2000. DWF.
;    Forgot to add _Extra everywhere. Fixed for MAIN errors. 8 AUG 2000. DWF.
```

```
;    Adding call routine's name to Traceback Report. 8 AUG 2000. DWF.
;    Switched default value for Dialog_Message to "Error" from "Warning". 7 OCT 2000. DWF.
;-
;#############################################################################
;
; LICENSE
;
; This software is OSI Certified Open Source Software.
; OSI Certified is a certification mark of the Open Source Initiative.
;
; Copyright © 1999-2000 Fanning Software Consulting
;
; This software is provided "as-is", without any express or
; implied warranty. In no event will the authors be held liable
; for any damages arising from the use of this software.
;
; Permission is granted to anyone to use this software for any
; purpose, including commercial applications, and to alter it and
; redistribute it freely, subject to the following restrictions:
;
; 1. The origin of this software must not be misrepresented; you must
;    not claim you wrote the original software. If you use this software
;    in a product, an acknowledgment in the product documentation
;    would be appreciated, but is not required.
;
; 2. Altered source versions must be plainly marked as such, and must
;    not be misrepresented as being the original software.
;
; 3. This notice may not be removed or altered from any source distribution.
;
; For more information on Open Source Software, visit the Open Source
; web site: http://www.opensource.org.
;
;#############################################################################


FUNCTION ERROR_MESSAGE, theMessage, Traceback=traceback, NoName=noName, _Extra=extra

On_Error, 2

   ; Check for presence and type of message.

IF N_Elements(theMessage) EQ 0 THEN theMessage = !Error_State.Msg
s = Size(theMessage)
messageType = s[s[0]+1]
IF messageType NE 7 THEN BEGIN
   Message, "The message parameter must be a string.", _Extra=extra
ENDIF

   ; Get the call stack and the calling routine's name.

Help, Calls=callStack
callingRoutine = (Str_Sep(StrCompress(callStack[1]),"("))[0]

   ; Are widgets supported? Doesn't matter in IDL 5.3 and higher.

widgetsSupported = ((!D.Flags AND 65536L) NE 0) OR Float(!Version.Release) GE 5.3
IF widgetsSupported THEN BEGIN
   IF Keyword_Set(noName) THEN answer = Dialog_Message(theMessage, _Extra=extra) ELSE BEGIN
      IF StrUpCase(callingRoutine) EQ "$MAIN$" THEN answer = Dialog_Message(theMessage, _Extra=extra) ELSE $
         answer = Dialog_Message(StrUpCase(callingRoutine) + ": " + theMessage, _Extra=extra)
   ENDELSE
ENDIF ELSE BEGIN
      Message, theMessage, /Continue, /NoPrint, /NoName, /NoPrefix, _Extra=extra
      Print, '%' + callingRoutine + ': ' + theMessage
      answer = 'OK'
ENDELSE
```

```
   ; Provide traceback information if requested.

IF Keyword_Set(traceback) THEN BEGIN
   Help, /Last_Message, Output=traceback
   Print,''
   Print, 'Traceback Report from ' + StrUpCase(callingRoutine) + ':'
   Print, ''
   FOR j=0,N_Elements(traceback)-1 DO Print, "      " + traceback[j]
ENDIF

RETURN, answer
END
```