GPS Navigation in a Visibility Challenged Environment

Ryan D. Riveland and Johnathan Clouse*
University of Colorado, Boulder, CO, 80309-0429, USA

The purpose of this experiment was to investigate the accuracy of a hand-held GPS receiver in an environment where signal visibility was limited. A route was chosen west of Castle Rock, CO where the first section was expected to have ample signal availability. The second section was in a canyon where signal availability was expected to be limited due to terrain. As expected, position error in the second section was degraded relative to the first section. It was concluded that terrain has a strong influence on GPS navigation accuracy.

*Graduate Students, Aerospace Engineering Sciences, 1111 Engineering Drive, Boulder, CO, 80309-0429

Contents

I. Objectives

The objective of this experiment was to investigate how varying terrain would impact the ability to navigate using GPS. In mountainous regions, signal visibility may be obstructed by the terrain leading to potential dilution of precision (DOP) of the solution. Several factors contributed to the extent to which the GPS signals may be obstructed. A few of these factors were the height of the terrain relative to the user, time of day and latitude. The experiment attempted to account for all of these factors in the results.

Since two hand-held GPS receivers were available, the Wide Area Augmentation System (WAAS) capability on one of the receivers was turned off to determine if using WAAS could mitigate navigation difficulties due to limited signal availability from the nominal GPS constellation. WAAS is a GPS augmentation system completed in 2003 by the FAA.[1]

The results of this experiment are of interest to those that may want to rely on GPS navigation for mountain recreation, to include hiking or any off road recreation. It was hypothesized that GPS navigation would be more accurate in wide-open terrain where more GPS signals were available for processing. Additionally, it was expected that the WAAS-enabled GPS receiver would be more accurate and precise than the GPS receiver where WAAS had been disabled.

II. Method

A 15-mile route along US Highway 85/ CO State Highway 67 west of Castle Rock, CO was chosen due to its wide variety of terrain. The first section of the route was wide-open with low elevation terrain. Signal availability was expected to be ample in this area. The second section was in a Rocky Mountain canyon where signal availability was expected to be limited due to the higher terrain. During the experiment, GPS tracks were recorded using both a Garmin GPS60 and a Delorme Earthmate PN-40. For consistency, both receivers were set to collect track points at 1 second intervals (which was also the fastest they could record).

While driving the route, azimuth and elevation measurements were taken of the terrain at various waypoints. Every effort was made to choose waypoints with varying terrain, however safe turn-off locations were limited. The azimuth measurements were taken using a compass and the elevation was measured using a clinometer at 45 degree azimuth intervals. In order to account for azimuth directions without measurements, the elevation was linearly interpolated between measurements. Digital photos of the signal strength screen were taken at each waypoint to correlate the signal strength vs. the visibility.

At each waypoint the time, latitude, longitude and height were extracted from the waypoint file output from the receiver. These data points were used to compute PRN azimuth and elevation. The azimuth and elevation points were plotted along with the linearly interpolated horizon mask to determine which PRN signals were actually available for navigation processing.

To determine the accuracy of the position measurements the tracks from each receiver were output to a .gpx file. This file is a standardized format recognized by a variety of GPS visualization software. For this experiment Google Earth was used for visualization. The highway on the Google Earth maps were used for qualitative truth. In addition to the position-performance truth, the visualization was used as a qualitative comparison between relative receiver performance.

In addition to qualitative comparisons, a few quantitative techniques were used to determine accuracy. Built into Google Earth, is a ruler tool that allowed the user to determine the distance between two points on a map. For this experiment, the road on the map was considered the truth. To determine position accuracy at various points, the distance could be measured between the superimposed track and the road. Secondly, data could be extracted from the .gpx files to determine the relative accuracy between the two receivers. The relative difference and root sum square (RSS) of latitude and longitude could be computed and plotted.

At the end of the route, the tracks were reset and data was collected along the path to the starting location

using the same route. Because of this, and the redundancy of using two different receivers, only one trial was conducted.

III. Experiment

The experiment was conducted on the afternoon of October 4, 2014. The weather conditions were sunny with a light wind. Since the method of travel was via car, the GPS receivers were set on the dashboard. Ideally, the GPS receivers would be held with the antenna facing upwards but this was not possible for safety reasons. As expected, locations to stop and collect horizon mask data were limited but every effort was made to take these measurements in locations with varying terrain. Despite from the previously mentioned limitations, all data collection activities were performed successfully.

IV. Data and Results

The resulting GPS data from the trip were post-processed to find the relative error between the two receivers. The relative errors were plotted with waypoint information so that they could be cross referenced with the topographic elevation measurements. The relative error between the receivers can be seen in Fig. 1 and Fig. 2. The tabular topographic measurements are shown in Table 1.

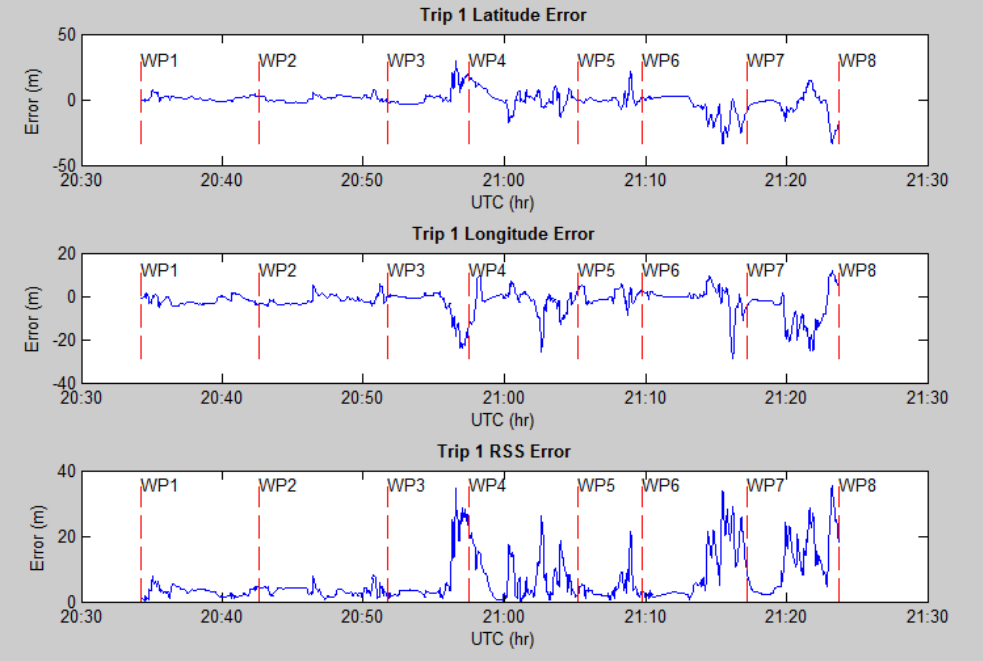

Figure 1. Relative receiver error observed during trip 1.

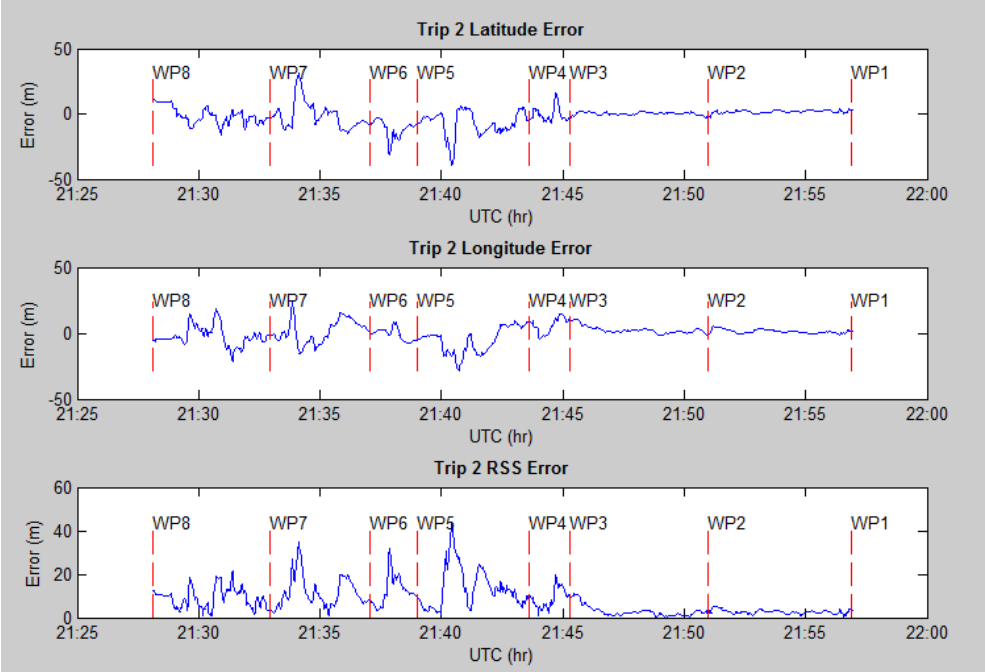

Figure 2. Relative receiver error observed during trip 2.

Table 1. Horizon mask for all waypoints.

Az.	El. WP1	El. WP2	El. WP3	El. WP4	El. WP5	El. WP6	El. WP7	El. WP8
0°	5°	2°	9°	18°	8°	11 °	21°	12°
45°	4°	0°	1°	24°	3°	9°	15°	6°
90°	15°	3°	0°	23°	0°	3°	8°	7°
135°	3°	6°	0°	13°	0°	6°	6°	13°
180°	7°	5°	2°	20°	3°	7°	14°	6°
225°	3°	3°	15°	26°	3°	4°	7°	10°
270°	2°	4°	20°	23°	6°	9°	6°	11°
315°	2°	3°	10°	15°	6°	9°	14°	20°

Another set of data gathered were the computed altitude of each device, the relative errors of which are shown in Fig. 3.

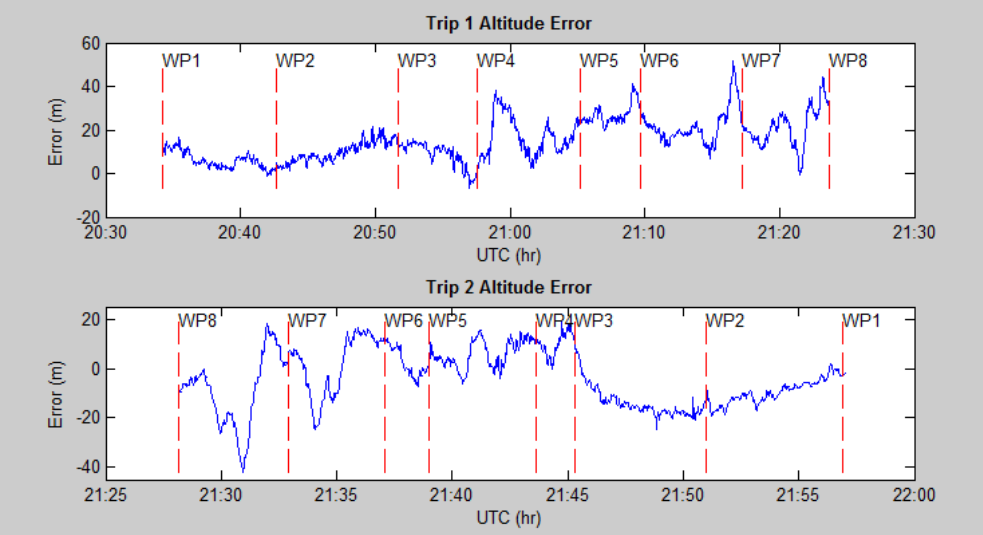

Figure 3. Relative receiver altitude error observed during trips 1 and 2.

V. Analysis

The error plots show that there was much greater relative error between the devices in the canyon than in the clear segment. An interesting artifact the authors found was that most of the waypoints where elevation measurements were taken did not show much error compared to the trip between them. The reason that most of the waypoints had more visibility, and less relative error, is most likely due to the condition of the canyon road. Much of the canyon had steep terrain with no shoulder or turn-off area, so taking waypoints at such places was nearly impossible.

The route between waypoints 1 and 3 (clear segment) was in clear terrain, with waypoint 3 observing the most topographic elevation to the west at 20 degrees (see Appendix A Fig. 11). The authors considered waypoint 3 to be the entrance to the canyon, with the latter points being in the canyon (canyon segment). The data were first checked against a map of the road in Google Earth to ensure the data were close to the road, as hypothesized. Figure 4 shows a representative example for each trip in the clear segment. Also, Fig. 7 shows a representative PRN signal strength in the clear segment.

Figure 4. Representative examples of position accuracy in the clear segment. The colored lines are the path travelled.

Clock synchronization between the receivers was initially a concern to the authors. The data were only recorded at integer-second intervals, which could produce a large error for a given epoch. This concern was abated when the clear segment's relative errors were analyzed. Between waypoints 1 and 2, the vehicle was traveling at 60 miles/hr (∼27 m/s). A representative measurement from each receiver at the same time on the clear segment was taken to be 2.6 meters, as seen in Fig. 5. If the difference was completely due to clock biases, that difference would be 0.1 s. However, pseudorange errors could be within this range of error, and the Garmin receiver had WAAS capability disabled. Thus, clock synchronization error was deemed low enough not to be an issue compared to other error sources, especially since the canyon segment had a top speed of 30 mph.

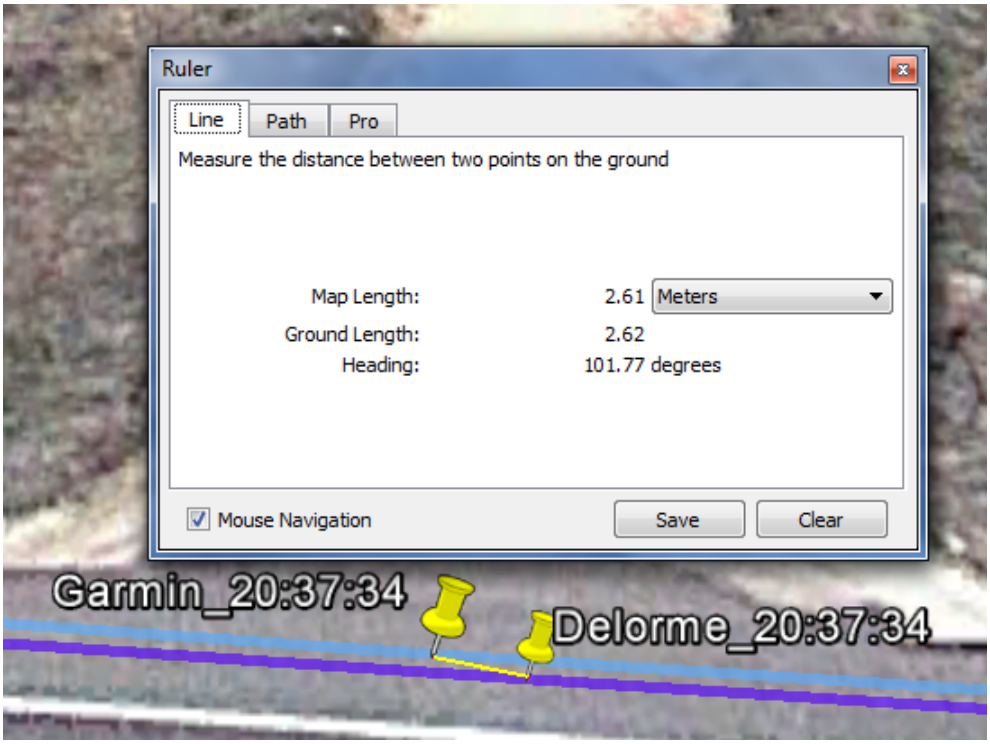

Figure 5. Difference in receiver positions with the same timestamps.

Representative canyon data are visualized in Fig. 6. The purple line is track data from the Delorme receiver and the blue line is the track data from the Garmin receiver. The Delorme receiver, with WAAS enabled, stays close to the road in Google Earth. On the other hand, the receiver without WAAS on varies wildly throughout the canyon. Using Google's Street View capability, one can see that this magnitude of error happens when there is much less visibility of the sky. Appendix A Figure 12 shows the visibility plot near this location; Appendix A Fig. 8 shows a representative PRN signal strength.

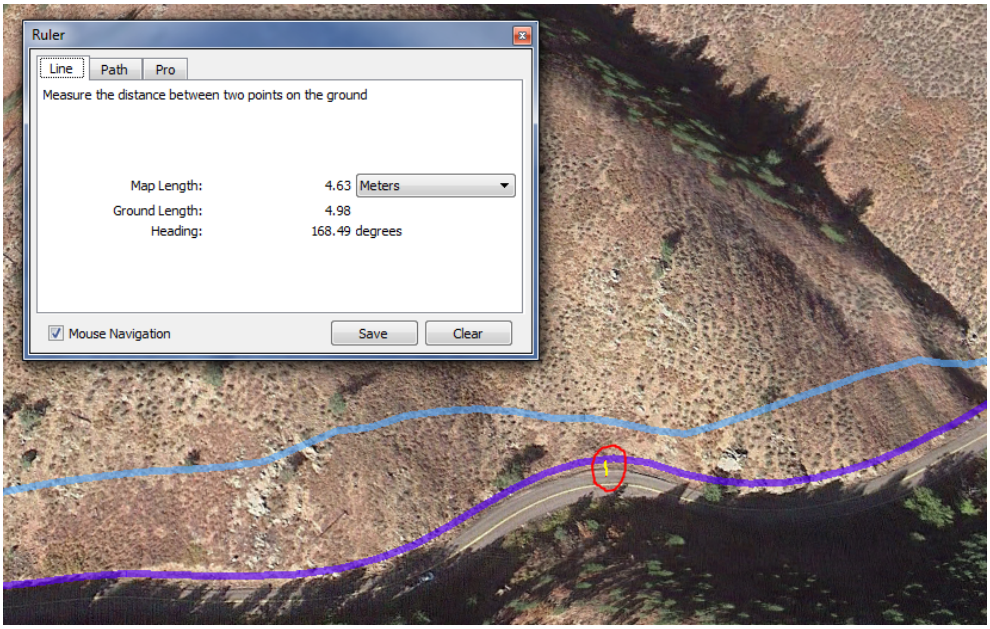

Figure 6. Representative example of position accuracy in the canyon segment. The purple line is the Delorme receiver, and the blue line is the Garmin receiver.

The altitude difference between the receivers is more substantial than the latitude/longitude differences. The error in the canyon segment is large, as expected. However, the altitude error varied by as much as 20 m in the clear segment, almost ten times that of the latitude and longitude relative errors. Despite this large difference, the tracks were overlaid on the road quite well. For this reason, the altitude was not further considered in the experiment, although it should be explored further.

VI. Conclusion and Recommendations

The data from both trips, and both receivers, were unambiguous: GPS navigation was better with fewer topographic features, and worse in the canyon. With the WAAS enabled, the Delorme receiver was consistently closer to the road throughout the canyon segment. The Garmin, without WAAS, was much more error-prone. With these results, the hypothesis was confirmed.

The experiment could have benefited from more control in a few areas. Receivers of the same model and firmware versions would eliminate any potential differences between the performed position calculations. The receivers were also sitting on the dashboard of the car, which still had obstruction potential. Being able to set the receivers above the cabin would allow for maximal visibility (topography notwithstanding). An especially enterprising team could combine a camera system and accelerometers to constantly take pictures and measurements to be used in topographic elevation measurements throughout the route, instead of a few discrete points made at safe turn-offs.

This experiment generated some questions the authors think could be explored. First, how accurate are the satellite images on Google Earth? They were assumed to be truth for the experiment, but error in the image placement would affect the perceived performance of the receivers. Additionally, are there survey data for the road that could be used to statistically fit the empirical data? This kind of analysis would yield more quantitative results than the use of Google Earth's ruler at various points. Finally, why were the height measurements so different between the receivers, compared to the latitude/longitude measurements?

The experiment successfully demonstrated that open terrain allowed for better GPS receiver accuracy than terrain with much topographic masking. However, there were areas identified to improve its quality. The authors also recommend to explore the questions identified previously, as they may lead to more insight into the data. Qualitatively, both receivers appeared to perform well in the canyon, despite the reduced satellite visibility. Even the Garmin with WAAS disabled could perform crude navigation of the canyon if needed, although it would not do well for precision applications. The experiment should be repeated in an area with higher, and more consistent, topographic features to further mask visibility and potentially draw sharper conclusions.

References

[1]Misra, P. and Enge, P. *Global Positioning System Signals, Measurements and Performance Revised Second Edition*, Ganga-Jamuna Press, 2012.

VII. Appendix A

This appendix contains supplemental pictures to further support the conclusions drawn previously. Figs. 7 and 8. Figs 9 - 16 show the elevation mask on sky visibility plots.

Figure 7. Representative example of PRN signal strength in the clear segment.

Figure 8. Representative example of PRN signal strength in the canyon segment.

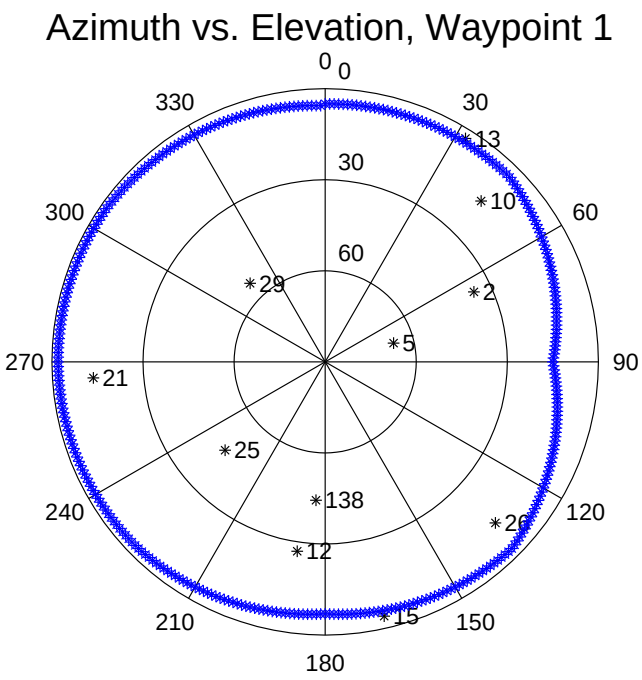

Figure 9. Plot for horizon mask for Waypoint 1.

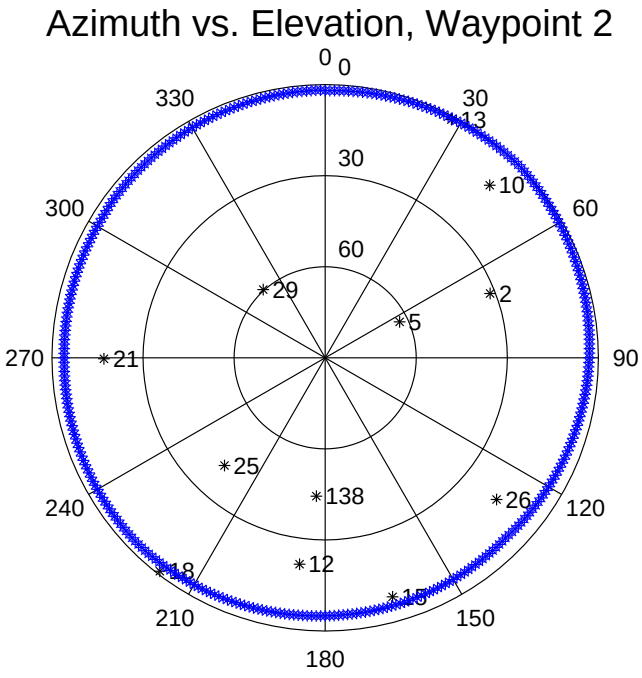

Figure 10. Plot for horizon mask for Waypoint 2.

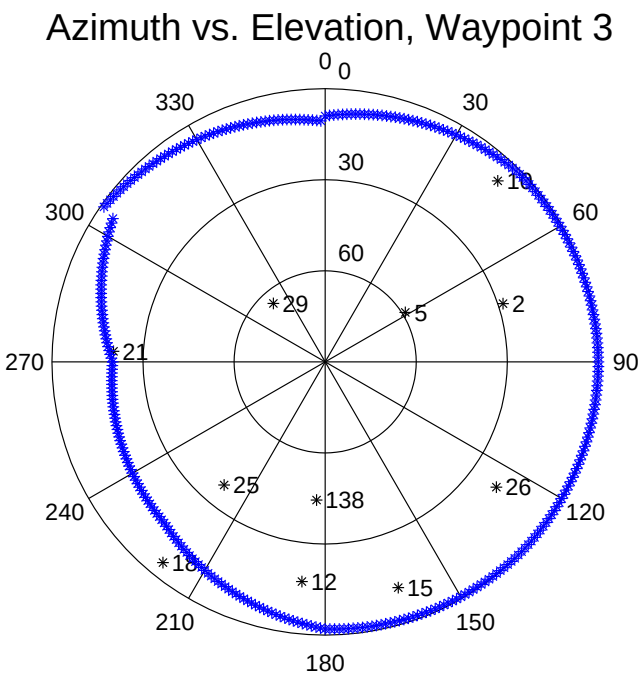

Figure 11. Plot for horizon mask for Waypoint 3.

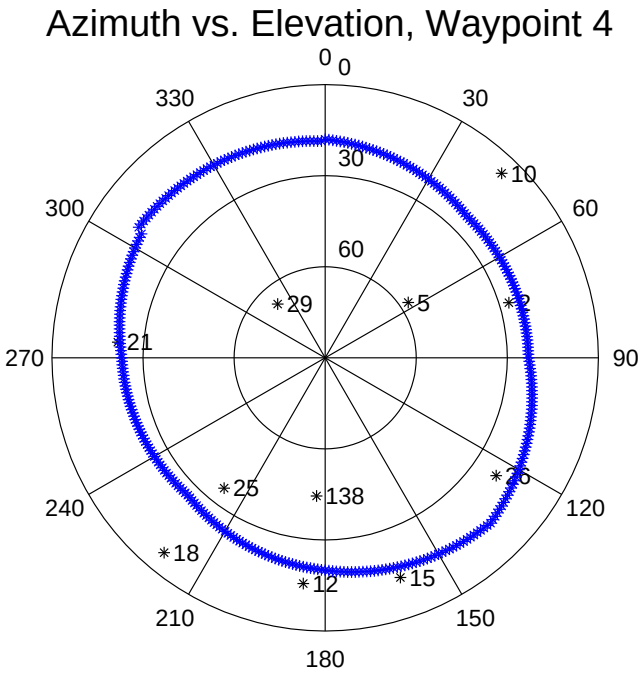

Figure 12. Plot for horizon mask for Waypoint 4.

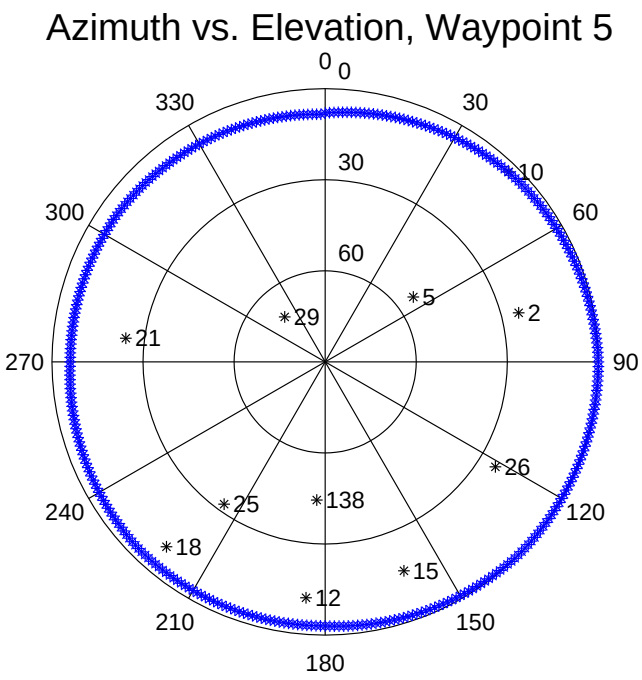

Figure 13. Plot for horizon mask for Waypoint 5.

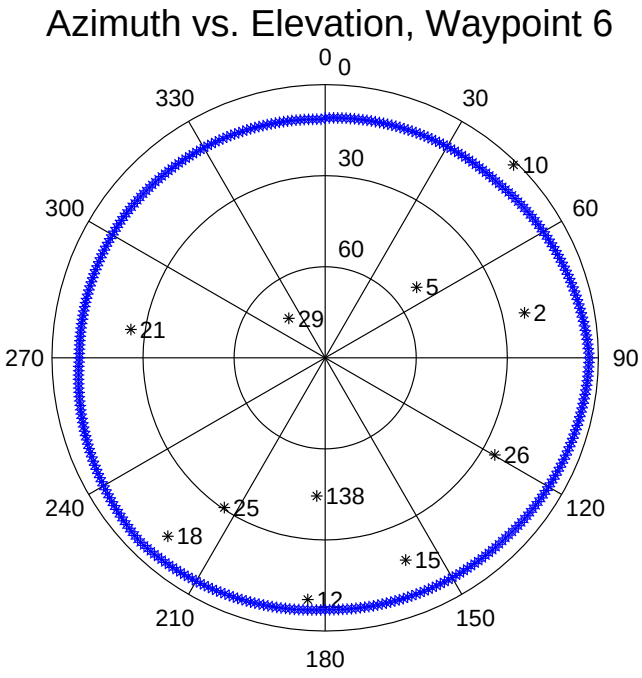

Figure 14. Plot for horizon mask for Waypoint 6.

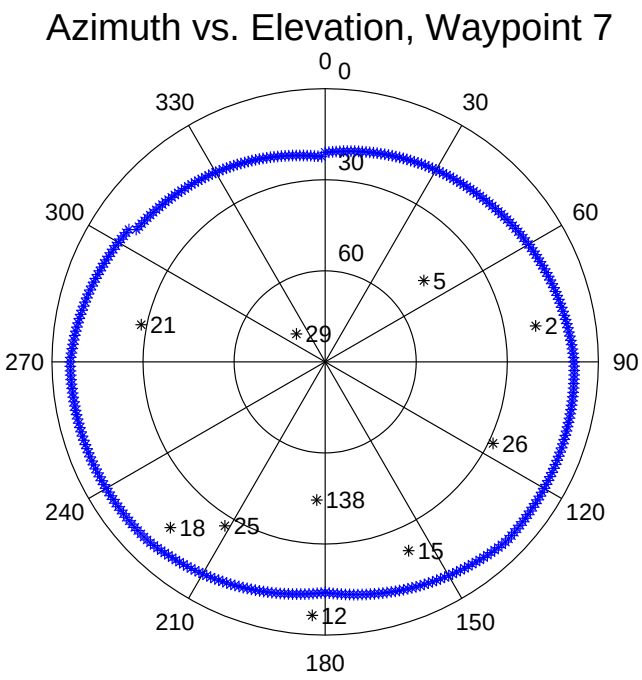

Figure 15. Plot for horizon mask for Waypoint 7.

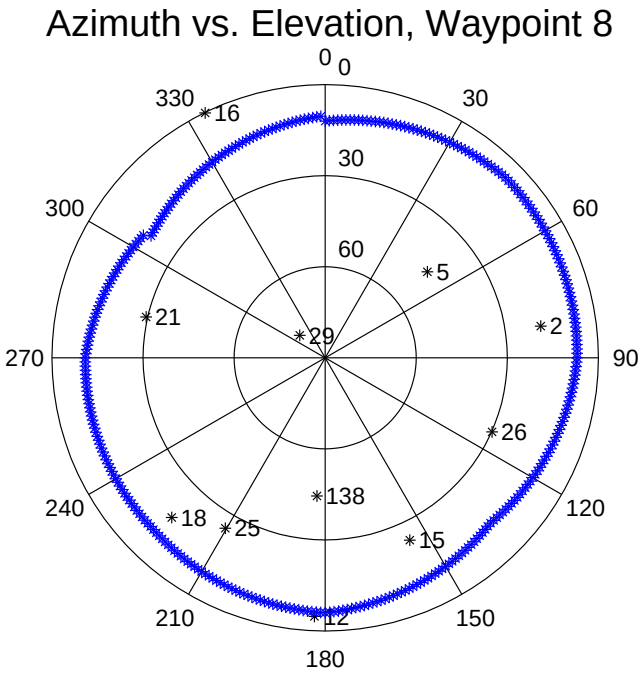

Figure 16. Plot for horizon mask for Waypoint 8.

VIII. Appendix B

This appendix contains all Matlab code used by the authors to analyze their data.

```
%% Initialize
clear
close all

%% Read in GPS data in CSV format
% Ryan had the Delorme, with WAAS on
% John had the Garmin, WAAS off
[ryan_in, john_in] = import_gps_data('RyanInMat.csv', 'JohnInMat2.csv');
[ryan_out, john_out] = import_gps_data('RyanOut.csv', 'JohnOut.csv');

%% Convert errors...
% Compute arc lengths along the ellipsoid
% WGS84, http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf
a = 6378137.0; % m
b = 6356752.3142; % m
e = sqrt(a*a-b*b)/a;
lat = ryan_in.lat(1); % rad, just a representative latitude
lon2m = a*cos(lat)/sqrt(1-e*e*sin(lat)*sin(lat));
lat2m = a*(1-e*e)/(sqrt(1-e*e*sin(lat)*sin(lat))*(1-e*e*sin(lat)*sin(lat)));

err.lat = (john_in.lat-ryan_in.lat)*lat2m;
err.lon = (john_in.lon-ryan_in.lon)*lon2m;
err.alt = (john_in.alt-ryan_in.alt);
% err.rss = sqrt(err.lat.*err.lat+err.lon.*err.lon+err.alt.*err.alt);
err.rss = sqrt(err.lat.*err.lat+err.lon.*err.lon);
err.time = john_in.time;

%Waypoints
in_wps = ['04-OCT-14 20:28:09'
     '04-OCT-14 20:42:37'
     '04-OCT-14 20:51:42'
     '04-OCT-14 20:57:31'
     '04-OCT-14 21:05:10'
     '04-OCT-14 21:09:44'
     '04-OCT-14 21:17:12'
     '04-OCT-14 21:23:44'];

out_wps = ['04-OCT-14 21:23:44'
     '04-OCT-14 21:32:55'
     '04-OCT-14 21:37:04'
     '04-OCT-14 21:39:01'
     '04-OCT-14 21:43:37'
     '04-OCT-14 21:45:19'
     '04-OCT-14 21:50:58'
     '04-OCT-14 21:56:54'];

in_wps_times = zeros(7,1);
out_wps_times = zeros(7,1);
for ii = 1:8
     in_wps_times(ii) = datenum(in_wps(ii,:));
     out_wps_times(ii) = datenum(out_wps(ii,:));
end
```

```
in_wps_times(1)=err.time(1);

subplot(3,1,1)
lab_err_plots(err.time,err.lat,'UTC (hr)', 'Error (m)', 'Trip 1 Latitude Error', in_wps_times)

subplot(3,1,2)
lab_err_plots(err.time,err.lon,'UTC (hr)', 'Error (m)', 'Trip 1 Longitude Error', in_wps_times)

subplot(3,1,3)
lab_err_plots(err.time,err.rss,'UTC (hr)', 'Error (m)', 'Trip 1 RSS Error', in_wps_times)


errout.lat = (john_out.lat-ryan_out.lat)*lat2m;
errout.lon = (john_out.lon-ryan_out.lon)*lon2m;
errout.alt = (john_out.alt-ryan_out.alt);
% errout.rss = sqrt(errout.lat.*errout.lat+errout.lon.*errout.lon+errout.alt.*errout.alt);
errout.rss = sqrt(errout.lat.*errout.lat+errout.lon.*errout.lon);
errout.time = john_out.time;
out_wps_times(1) = errout.time(1);
out_wps_times = flip(out_wps_times); %so the waypoint plots show up correctly

figure
subplot(3,1,1)
lab_err_plots(errout.time,errout.lat,'UTC (hr)', 'Error (m)', 'Trip 2 Latitude Error', out_wps_times)

subplot(3,1,2)
lab_err_plots(errout.time,errout.lon,'UTC (hr)', 'Error (m)', 'Trip 2 Longitude Error', out_wps_times)

subplot(3,1,3)
lab_err_plots(errout.time,errout.rss,'UTC (hr)', 'Error (m)', 'Trip 2 RSS Error', out_wps_times)

figure
subplot(2,1,1)
lab_err_plots(err.time,err.alt,'UTC (hr)', 'Error (m)', 'Trip 1 Altitude Error', in_wps_times)
subplot(2,1,2)
lab_err_plots(errout.time,errout.alt,'UTC (hr)', 'Error (m)', 'Trip 2 Altitude Error', out_wps_times)
```

```
function [struct1, struct2] = import_gps_data(file1, file2)
% import_gps_data Import two data that have similar time series to compare
% against eachother
% lat/lon output in rad

% csv data is expected to be lat, lon, alt, ISO 8601 time string
[struct1.lat, struct1.lon, struct1.alt, struct1.time] = ...
    textread(file1, '%f%f%f%s', 'delimiter', ',');
struct1.time = convert_times(struct1.time);
```

```
10
11 [struct2.lat, struct2.lon, struct2.alt, struct2.time] = ...
12     textread(file2, '%f%f%f%s', 'delimiter', ',');
13 struct2.time = convert_times(struct2.time);
14
15 begin_time = max([struct1.time(1), struct2.time(1)]);
16 end_time = min([struct1.time(end), struct2.time(end)]);
17
18 % Trim both times series to common beginning/end times
19 trim = @(x,y,t1,t2) x(y >= t1 & y <= t2); %anon fcn for consistency
20 struct1.lat = trim(struct1.lat,struct1.time, begin_time, end_time);
21 struct1.lon = trim(struct1.lon,struct1.time, begin_time, end_time);
22 struct1.alt = trim(struct1.alt,struct1.time, begin_time, end_time);
23 struct1.time = trim(struct1.time,struct1.time, begin_time, end_time);
24 struct2.lat = trim(struct2.lat,struct2.time, begin_time, end_time);
25 struct2.lon = trim(struct2.lon,struct2.time, begin_time, end_time);
26 struct2.alt = trim(struct2.alt,struct2.time, begin_time, end_time);
27 struct2.time = trim(struct2.time,struct2.time, begin_time, end_time);
28
29 % Convert lat/lon to rads
30 struct1.lat = struct1.lat*pi/180;
31 struct1.lon = struct1.lon*pi/180;
32 struct2.lat = struct2.lat*pi/180;
33 struct2.lon = struct2.lon*pi/180;
34
35 % Deal with lack of data for some times
36 % Find the common data and stick them in an array
37 cnt = 1;
38 series1=zeros(1,4);
39 series2=zeros(1,4);
40 for ii = 1:length(struct2.time)
41     cond = struct1.time == struct2.time(ii);
42     if struct1.time(cond) == struct2.time(ii)
43         series1(cnt,:) = [struct1.lat(cond) struct1.lon(cond) struct1.alt(cond
              ) struct1.time(cond)];
44         series2(cnt,:) = [struct2.lat(ii) struct2.lon(ii) struct2.alt(ii)
              struct2.time(ii)];
45         cnt = cnt + 1;
46     end
47 end
48 for ii = 1:length(struct1.time)
49     cond = struct2.time == struct1.time(ii);
50     if struct2.time(cond) == struct1.time(ii)
51         series1(cnt,:) = [struct1.lat(ii) struct1.lon(ii) struct1.alt(ii)
              struct1.time(ii)];
52         series2(cnt,:) = [struct2.lat(cond) struct2.lon(cond) struct2.alt(cond
              ) struct2.time(cond)];
53         cnt = cnt + 1;
54     end
55 end
56
57 % Remove duplicates, sort by time
58 series1 = unique(series1, 'rows', 'stable');
59 series2 = unique(series2, 'rows', 'stable');
60 series1 = sortrows(series1, 4);
```

```
series2 = sortrows(series2, 4);

% Reassign data structs.
struct1.lat = series1(:,1);
struct1.lon = series1(:,2);
struct1.alt = series1(:,3);
struct1.time = series1(:,4);
struct2.lat = series2(:,1);
struct2.lon = series2(:,2);
struct2.alt = series2(:,3);
struct2.time = series2(:,4);
end

function time_nums = convert_times(input)
len = length(input);
time_nums = zeros(len,1);
for ii = 1:len
    time_nums(ii) = datenum8601(input{ii},'*ymdHMS');
end
end
```

```
function [DtN,Spl,TkC] = datenum8601(Str,Tok)
% Convert an ISO 8601 formatted Date String (timestamp) to a Serial Date
    Number.
%
% (c) 2014 Stephen Cobeldick
%
%### Function ###
%
% Syntax:
%  DtN = datenum8601(Str)
%  DtN = datenum8601(Str,Tok)
%  [DtN,Spl,TkC] = datenum8601(...)
%
% By default the function automatically detects all ISO 8601 timestamp/s in
% the string, or use a token to restrict detection to only one particular
    style.
%
% The ISO 8601 timestamp style options are:
% - Date in calendar, ordinal or week-numbering notation.
% - Basic or extended format.
% - Choice of date-time separator character ( @T_).
% - Full or lower precision (trailing units omitted)
% - Decimal fraction of the trailing unit.
% These style options are illustrated in the tables below.
%
% The function returns the Serial Date Numbers of the date and time given
% by the ISO 8601 style timestamp/s, the input string parts that are split
% by the detected timestamps (i.e. the substrings not part of  any ISO 8601
% timestamp), and string token/s that define the detected timestamp style/s.
%
% Note 1: Calls undocumented MATLAB function "datenummx".
% Note 2: Unspecified month/date/week/day timestamp values default to one (1).
% Note 3: Unspecified hour/minute/second timestamp values default to zero (0).
% Note 4: Auto-detection mode also parses mixed basic/extended timestamps.
```

```
33 %
34 % See also DATESTR8601 DATEROUND CLOCK NOW DATENUM DATEVEC DATESTR NATSORT
     NATSORTROWS NATSORTFILES
35 %
36 % ### Examples ###
37 %
38 % Examples use the date+time described by the vector [1999,1,3,15,6,48.0568].
39 %
40 % datenum8601('1999-01-03 15:06:48.0568')
41 %  ans = 730123.62972287962
42 %
43 % datenum8601('1999003T150648.0568')
44 %  ans = 730123.62972287962
45 %
46 % datenum8601('1998W537_150648.0568')
47 %  ans = 730123.62972287962
48 %
49 % [DtN,Spl,Tok] = datenum8601('A19990103B1999-003C1998-W53-7D')
50 %  DtN = [730123,730123,730123]
51 %  Spl = {'A','B','C','D'}
52 %  Tok = {'ymd','*yn','*YWD'}
53 %
54 % [DtN,Spl,Tok] = datenum8601('1999-003T15')
55 %  DtN = 730123.6250
56 %  Spl = {'',''}
57 %  Tok = {'*ynTH'}
58 %
59 % [DtN,Spl,Tok] = datenum8601('1999-01-03T15','*ymd')
60 %  DtN = 730123.0000
61 %  Spl = {'','T15'}
62 %  Tok = {'*ymd'}
63 %
64 % ### ISO 8601 Timestamps ###
65 %
66 % The token consists of one letter for each of the consecutive date/time
67 % units in the timestamp, thus it defines the date notation (calendar,
68 % ordinal or week-date) and selects either basic or extended format:
69 %
70 % Input     | Basic Format                | Extended Format (token prefix '*')
71 % Date      | In/Out | Input Timestamp | In/Out   | Input Timestamp
72 % Notation:| <Tok>: | <Str> Example:  | <Tok>:   | <Str> Example:
73 % ==========|========|=================|==========|=========================
74 % Calendar  |'ymdHMS'|'19990103T150648'|'*ymdHMS'|'1999-01-03T15:06:48'
75 % ----------|--------|-----------------|----------|-------------------------
76 % Ordinal   |'ynHMS' |'1999003T150648' |'*ynHMS' |'1999-003T15:06:48'
77 % ----------|--------|-----------------|----------|-------------------------
78 % Week      |'YWDHMS'|'1998W537T150648'|'*YWDHMS'|'1998-W53-7T15:06:48'
79 % ----------|--------|-----------------|----------|-------------------------
80 %
81 % Options for reduced precision timestamps, non-standard date-time separator
82 % character, and the addition of a decimal fraction of the trailing unit:
83 %
84 % Omit trailing units (reduced precision), eg:                  | Output->
     Vector:
```

```
85 %
   ========|========|==================|========|==================|====================
86 %         |'Y'     |'1999W'           |'*Y'    |'1999-W'
   |[1999,1,4,0,0,0]
87 %
   --------|--------|------------------|--------|------------------|--------------------
88 %         |'ymdH'  |'19990103T15'     |'*ymdH' |'1999-01-03T15'
   |[1999,1,3,15,0,0]
89 %
   --------|--------|------------------|--------|------------------|--------------------
90 % Select the date-time separator character (one of ' ','@','T','_'), eg:
91 %
   ========|========|==================|========|==================|====================
92 %         |'yn_HM' |'1999003_1506'    |'*yn_HM'|'1999-003_15:06'
   |[1999,1,3,15,6,0]
93 %
   --------|--------|------------------|--------|------------------|--------------------
94 %         |'YWD@H' |'1998W537@15'     |'*YWD@H'|'1998-W53-7@15'
   |[1999,1,3,15,0,0]
95 %
   --------|--------|------------------|--------|------------------|--------------------
96 % Decimal fraction of trailing date/time value, eg:
97 %
   ========|========|==================|========|==================|====================
98 %         |'ynH3'  |'1999003T15.113'  |'*ynH3' |'1999-003T15
   .113'|[1999,1,3,15,6,46.80]
99 %
   --------|--------|------------------|--------|------------------|--------------------
100 %         |'YWD4'  |'1998W537.6297'   |'*YWD4' |'1998-W53
   -7.6297'|[1999,1,3,15,6,46.08]
101 %
   --------|--------|------------------|--------|------------------|--------------------
102 %         |'y10'   |'1999.0072047202'|'*y10'
   |'1999.0072047202'|[1999,1,3,15,6,48.06]
103 %
   --------|--------|------------------|--------|------------------|--------------------
104 %
105 % Note 5: This function does not check for ISO 8601 compliance: user beware!
106 % Note 6: Date-time separator character must be one of ' ','@','T','_'.
107 % Note 7: Date notations cannot be combined: note upper/lower case characters.
108 %
109 % ### Input & Output Arguments ###
110 %
111 % Inputs (*default):
```

```
112 %  Str = Date String, possibly containing one or more ISO 8601 dates/
        timestamps.
113 %  Tok = String token, to select the required date notation and format (*[]=
        any).
114 %
115 % Outputs:
116 %  DtN = Numeric Vector of Serial Date Numbers, one from each timestamp in
        input <Str>.
117 %  Spl = Cell of Strings, the strings before, between and after the detected
        timestamps.
118 %  TkC = Cell of Strings, tokens of each timestamp notation and format (see
        tables).
119 %
120 % [DtN,Spl,TkC] = datenum8601(Str,Tok*)
121
122 % Define "regexp" match string:
123 if nargin<2 || isempty(Tok)
124     % Automagically detect timestamp style.
125     MtE = [...
126         '(\d{4})',... % year
127         '((-(?=(\d{2,3}|W)))?)',... % -
128         '(W?)',...     % W
129         '(?(3)(\d{2})?|(\d{2}(?=($|\D|\d{2})))?)',... % week/month
130         '(?(4)(-(?=(?(3)\d|\d{2}))))?)',...    % -
131         '(?(4)(?(3)\d|\d{2})?|(\d{3})?)',... % day of week/month/year
132         '(?(6)([ @T_](?=\d{2})))?)',... % date-time separator character
133         '(?(7)(\d{2})?)',...  % hour
134         '(?(8)(:(?=\d{2})))?)',...  % :
135         '(?(8)(\d{2})?)',...  % minute
136         '(?(10)(:(?=\d{2})))?)',... % :
137         '(?(10)(\d{2})?)',... % second
138         '((\.\d+)?)']; % trailing unit decimal fraction
139     % (Note: allows a mix of basic/extended formats)
140 else
141     % User requests a specific timestamp style.
142     assert(ischar(Tok)&&isrow(Tok),'Second input <Tok> must be a string.')
143     TkU = regexp(Tok,'(^\*?)([ymdnYWD]*)([ @T_]?)([HMS]*)(\d*$)','tokens','
            once');
144     assert(~isempty(TkU),'Second input <Tok> is not recognized: ''%s''',Tok)
145     MtE = [TkU{2},TkU{4}];
146     TkL = numel(MtE);
147     Ntn = find(strncmp(MtE,{'ymdHMS','ynHMS','YWDHMS'},TkL),1,'first');
148     assert(~isempty(Ntn),'Second input <Tok> is not recognized: ''%s''',Tok)
149     MtE = dn8601Usr(TkU,TkL,Ntn);
150 end
151 %
152 assert(ischar(Str)&&size(Str,1)<2,'First input <Str> must be a string.')
153 %
154 % Extract timestamp tokens, return split strings:
155 [TkC,Spl] = regexp(Str,MtE,'tokens','split');
156 %
157 [DtN,TkC] = cellfun(@dn8601Main,TkC);
158 %
159 end
```

```
%------------------------------------------------------------------------END:
    datenum8601
function [DtN,Tok] = dn8601Main(TkC)
% Convert detected substrings into serial date number, create string token.
%
% Lengths of matched tokens:
TkL = cellfun('length',TkC);
% Preallocate Date Vector:
DtV = [1,1,1,0,0,0];
%
% Create token:
Ext = '*';
Sep = [TkC{7},'HMS'];
TkX = {['ymd',Sep],['y*n',Sep],['YWD',Sep]};
Ntn = 1+(TkL(6)==3)+2*TkL(3);
Tok = [Ext(1:+any(TkL([2,5,9,11])==1)),TkX{Ntn}(0<TkL([1,4,6,7,8,10,12]))];
%
% Convert date and time values to numeric:
Idx = [1,4,6,8,10,12];
for m = find(TkL(Idx));
    DtV(m) = sscanf(TkC{Idx(m)},'%f');
end
% Add decimal fraction of trailing unit:
if TkL(13)>1
    if Ntn==2&&m==2 % Month (special case not converted by "datenummx"):
        DtV(3) = 1+sscanf(TkC{13},'%f')*(datenummx(DtV+[0,1,0,0,0,0])-
            datenummx(DtV));
    else % All other date or time values (are converted by "datenummx"):
        DtV(m) = DtV(m)+sscanf(TkC{13},'%f');
    end
    Tok = {[Tok,sprintf('%.0f',TkL(13)-1)]};
else
    Tok = {Tok};
end
%
% Week-numbering vector to ordinal vector:
if Ntn==3
    DtV(3) = DtV(3)+7*DtV(2)-4-mod(datenummx([DtV(1),1,1]),7);
    DtV(2) = 1;
end
% Convert out-of-range Date Vector to Serial Date Number:
DtN = datenummx(DtV) - 31*(0==DtV(2));
% (Month zero is a special case not converted by "datenummx")
%
end
%------------------------------------------------------------------------END:
    dn8601Main
function MtE = dn8601Usr(TkU,TkL,Ntn)
% Create "regexp" <match> string from user input token.
%
% Decimal fraction:
if isempty(TkU{5})
    MtE{13} = '()';
else
    MtE{13} = ['(\.\d{',TkU{5},'})'];
```

```
end
% Date-time separator character:
if isempty(TkU{3})
    MtE{7} = '(T)';
else
    MtE{7} = ['(',TkU{3},')'];
end
% Year and time tokens (year, hour, minute, second):
MtE([1,8,10,12]) = {'(\d{4})','(\d{2})','(\d{2})','(\d{2})'};
% Format tokens:
if isempty(TkU{1}) % Basic
    MtE([2,5,9,11]) = {'()','()','()','()'};
else % Extended
    MtE([2,5,9,11]) = {'(-)','(-)','(:)','(:)'};
end
% Date tokens:
switch Ntn
    case 1 % Calendar
        Idx = [2,5,7,9,11,13];
        MtE([3,4,6]) = {'()', '(\d{2})','(\d{2})'};
    case 2 % Ordinal
        Idx = [2,7,9,11,13];
        MtE([3,4,5,6]) = {'()','()','()','(\d{3})'};
    case 3 % Week
        Idx = [2,5,7,9,11,13];
        MtE([3,4,6]) = {'(W)','(\d{2})','(\d{1})'};
end
%
% Concatenate tokens into "regexp" match token:
MtE(Idx(TkL):12) = {'()'};
MtE = [MtE{:}];
%
end
%------------------------------------------------------------------------------END:
    dn8601Usr
```

```
function lab_err_plots(x,y,x_label, y_label, ptitle, wpts)

plot(x, y)
hold on
ylims = [min(y) max(y)];
hold on
for ii = 1:length(wpts)
    plot([wpts(ii),wpts(ii)],ylims,'r--')
    text(wpts(ii),ylims(2),sprintf('WP%d',ii))
end
hold off
datetick('x', 'HH:MM')
title(ptitle, 'FontWeight', 'bold')
ylabel(y_label)
xlabel(x_label)
```