GPS Parsing PreLab
ME597D

Spring 2011

For this PreLab, we will be working with GPS data in the NMEA 0183 format. There are many different sentences output. For the files provided here, three are presented: $GPRMC, $GPGGA, and $GPGSV. As you can see, the sentence begins with a ‘$’ and can be no longer than 80 characters of visible text (plus the line terminators). The data is contained within this single line with data items separated by commas. The data below provides you with how information is presented in these two sentence formats.

$GPGGA,170834,4124.8963,N,08151.6838,W,1,05,1.5,280.2,M,-34.0,M,,,*75

<table>
<thead>
<tr>
<th>170834</th>
<th>17:08:34 UTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>4124.8963, N</td>
<td>41d 24.8963’ N or 24’ 54” N</td>
</tr>
<tr>
<td>08151.6838,W</td>
<td>81d 51.6838’ W or 81d 51’ 41” W</td>
</tr>
<tr>
<td>1</td>
<td>Fix Quality: 0 = Invalid, 1 = GPS fix, 2 = DGPS fix</td>
</tr>
<tr>
<td>05</td>
<td>5 Satellites are in view</td>
</tr>
<tr>
<td>1.5</td>
<td>HDOP (Horizontal Dilution of Precision)</td>
</tr>
<tr>
<td>280.2,M</td>
<td>280.2 meters above MSL (mean sea level)</td>
</tr>
<tr>
<td>-34.0, M</td>
<td>-34.0 meters below the WGS84 ellipsoid</td>
</tr>
<tr>
<td>Blank</td>
<td>No last update</td>
</tr>
<tr>
<td>Blank</td>
<td>No station id</td>
</tr>
<tr>
<td>*75</td>
<td>Checksum to verify sentence</td>
</tr>
</tbody>
</table>

$GPRMC,225446,A,4916.45,N,12311.12,W,000.5,054.7,191194,020.3,E*68

<table>
<thead>
<tr>
<th>225446</th>
<th>22:54:46 UTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Navigation receiver warning, A = Valid, V = warning</td>
</tr>
<tr>
<td>4916.45, N</td>
<td>49d 16.45’ N</td>
</tr>
<tr>
<td>12311.12,W</td>
<td>123d 11.12’ W</td>
</tr>
<tr>
<td>000.5</td>
<td>Speed over ground, Knots</td>
</tr>
<tr>
<td>054.7</td>
<td>Course Mad Good, degrees true</td>
</tr>
<tr>
<td>191194</td>
<td>UTC Date of fix, 19 November 1994</td>
</tr>
<tr>
<td>020.3, E</td>
<td>Magnetic variation, 20.3d East</td>
</tr>
<tr>
<td>*68</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

If you open the provided data file (GPStest.txt) you can see a sample log of collected data. For more information about other sentences interpretation, here are some helpful websites:

http://home.mira.net/~gnb/gps/nmea.html
Parsing Real GPS Data and Plotting the Data on a Given Map

The goal of this PreLab is to read in NMEA formatted data, find the sentences of interest, parse the data, and then plot the locations on a map. The text file “GPStest.dat” contains real GPS data we collected around campus. When you open it up, you will find out that there are four sentences GPRMC, GPGGA, GPGSA and GPGSV. You can open the website mentioned above to find more information about these sentences. Again, in this part, you need to use data represented in GPGGA format.

**Step 1:** In the real world, the connection between GPS device and satellite sometimes is bad which means the data is invalid. If you look at the text file really carefully, you can find some data is not complete. Fortunately, we have one bit in GPGGA sentence representing the data validity. First we want to use the fopen() command in MATLAB to open the data as an object. We can do this with the following example.

```matlab
obj1 = fopen('GPStest.dat');
```

As a side note, this same procedure could be used to obtain “live” GPS data from a serial port.

**Step 2:** We now want to go through the file and look for the sentence format ($GPGGA) that we want. The MATLAB command strncmpi('s1','s2',n) compares the first n characters of s1 and s2 and returns a logical: 1 if they are the same and 0 if they are not.

```matlab
(strncmpi(data, '$GPGGA', 6))
```

**Step 3:** Once we have identified the sentence that we want, we need to go through and find the delimiters, which are commas in our format. This can be done with the following code:

```matlab
ii=findstr(data, '','');
```

ii will now contain all of the locations within the sentence where a comma appears.

**Step 4:** Now that we know where the commas are located, we need to use our knowledge of the sentence format to determine where the information we want is located in the sentence. For example, we know that the time is the first entry in the GPGGA format, so we need to look between the first and second comma. This can be accomplished with the following code.

```matlab
UTC = data(ii(1)+1:ii(2)-1);
```

We now have the time, but as a string of hours, minutes, and seconds. To convert this from a string to a number, we can use the MATLAB command str2num. However, if applied to the entire string, it will not parse the hours minutes and seconds automatically. We need to individually split out the hours, minutes,
and seconds. For this PreLab, the time should be represented as number of seconds since midnight in UTC (convert everything to seconds).

\[
\text{Time} = \text{str2num}(	ext{UTC}(1:2)) \times 3600; \quad \% \text{Convert hours to secs}
\]
\[
\text{Time} = (\text{str2num}(	ext{UTC}(3:4)) \times 60) + \text{Time}; \quad \% \text{Convert mins to secs and add}
\]
\[
\text{Time} = \text{str2num}(	ext{UTC}(5:end)) + \text{Time}; \quad \% \text{add seconds to converted secs}
\]

**Step 5:** Our next step is to find the latitude and longitude in a similar manner. Latitude is the next piece of information in the sentence. It includes 9 characters denoting the degrees and minutes. The first two digits are the degrees measurement. The minutes (60 per degree) are represented as the next two digits followed by a decimal point followed by four digits. The next field gives a character that represents whether the latitude is given north or south. ‘N’ or ‘n’ is used to indicate North and ‘S’ or ‘s’ is used to indicate south. If the data is given to the South, we want to represent this as negative latitude.

\[
\text{if} \quad (\text{NS} == \text{'S'}) \quad \text{||} \quad (\text{NS} == \text{'s'})
\]
\[
\text{Lat} = -\text{Lat};
\]
\text{end}

Longitude is similar to latitude, except three digits are given for the degree measurement. Also, the data can be given either East or West. If the data is given to the west, it needs to be multiplied by negative 1.

**Step 6:** The last piece of information that we will use is the data fix. If the GPS receiver is unable to collect pseudoranges (measurements) from enough satellites, it is unable to estimate position. This will be displayed as a 0 in the data fix field. If a location fix is not present, the provided position estimate should not be used nor plotted.

**Step 7:** Now that we know how to read one line of data, we need to make a loop that will repeat this process until the end of the file. The MATLAB commandfeof will return a logical determining if the end of the file is reached: 1 if we are at the end of the file and 0 if we are not.

feof(obj1)

**Step 8:** Now that we are done with the file, we want to close the object (file) used. Use the MATLAB commandfclose()

fclose(obj1);

**Step 9:** Plot the latitude and longitude coordinates in the Reber Map provided. Remember computer defines original (0,0) as the up-left corner of a picture. The imagesc command scales and translates the image to the correct coordinates. Use the following code to plot the satellite image:

\[
\text{reber} = \text{imread('reber.png')}; \quad \% \text{a satellite pic of the Reber area}
\]

\[
\text{figure(1)}
\]
\[
\text{imagesc([-77.866405 -77.861577], [40.794692 40.792272],reber)};
\]
\[
\text{axis image};
\]
\[
\text{set(gca,'YDir','normal');}
\]
\[
\text{hold on;}
\]
The numbers used in the imagesc command provided are the LATLON coordinates for the upper left and lower right corners of the image (-77.866405, 40.794692), (-77.861577, 40.792272). The axis image command sets the step size on the x and y axes to be the same. The set(gca,'YDir','normal') sets the axis to display in the traditional form (with the origin towards the lower left of the plot).

You can now plot your data on top of the map.

**Deliverable:** Turn in your coordinate trajectory on Reber map and MATLAB code.

For further assistance, the following link provides an example code. This code will load the data file and will look through the file to find the first matching NMEA sentence. It then parses the Time and Latitude components. Note, this code performs parsing on the $GPRMC sentence and will ONLY find and parse one line. Should you use this code, you need to modify it to parse over the entire file and use the $GPGGA sentence.

[http://www.personal.psu.edu/sqm110/ME597GPS/MATLAB_GPS_example.m](http://www.personal.psu.edu/sqm110/ME597GPS/MATLAB_GPS_example.m)