CS22510 Assignment 1: "Where Have I Been?" - Processing GPS

James Euesden - jee22

Introduction

The assignment brief provided stated that I must write an application in Java, C and C++. This application must be able to read in 2 .dat files that would represent 2 'Streams' of GPS data. With these Streams, they must be 'synced' to the same timestamp as each other, and a GPX file must be produced that shows the route recorded by the GPS devices. In particular, the application must choose the 'best' signal from between the two Streams for each Timestamp, based upon the signal-to-noise ratio of the satellites providing the gps data. This document breifly describes the actions I took while writing the applications for the CS22510 assignment 'Where have I been?'. It will also include any assumptions I made and any simplications made on the data and request of the assignment brief.

Analysis

When presented with the assignment, my first step was to try and understand exactly what the NMEA standard for GPS sentences was, and what each individual one represented. For this I used the website http://www.gpsinformation.org/dale/nmea.htm to try and understand them. Once I had come to the realisation that each sentence provided only a portion of data, while many added up to a whole, I knew that I would need some sort of data structure in a class or struct that would have the most recent items of relevant data. On each sentence read-in, if the sentence contained data I wanted, it could be used to update the variables needed in the class made to hold the Stream data. On each update of the Timestamp, I knew that would be the point to write out the data to GPX, or at least store it in a list for writing out at the end of the read-in.

I found that getting this basic understanding and abstraction comprehension for the sentences was instrumental in designing my applications. Once I understood this, I had a much clearer path in my mind of how best to tackle the issue. Knowing that I didn't need all of the data from sentences was useful too. Based on this, I decided to use purely the GPRMC, GPGGA and GPGSV sentences. This gave me the Latitude and Longitude, Elevation and a Timestamp. While some other sentences also included a Timestamp, I felt that the GPGGA and GPRMC would be enough to update my Timestamp appropriately to give a clear view of the route taken with the devices. It is possible that the application could be extended to include all relevant data from the other sentences in the future, should this be required.

Another assumption was made, that in order to determine whether the signal was good or not, my applications would only focus upon whether or not the Signal-to-Noise Ratio (SNR) values were good enough. As stated by the assignment brief, this was considered to be good at 3 or more satellites at 35 or above, acceptable with satellites SNR signal between 30 and 35 and unacceptable any lower or with any less satellites. If the main stream had 3 or more satellites between 30 and 35, and the second stream also had 3 or more in this range, but more satelites at or above 35 than stream 1 (while it has less than 3), it was decided that stream 2 would have a better signal. Likewise, if stream 1 had 3 or more satellites with an SNR of 35 or above, stream 1 would be used as output. However, if it had 2 or less satellites with 35 or more SNR, and 2 or less satellites with SNR between 30 and 35, stream 2 would be used regardless.

Since I had decided that I only really needed to know which SNR values were within a certain range, I simplified the data sent by the sentences into two integers. 'Above35' and 'Between 30 and 35'. Whenever an SNR value was found, it would be checked for it's range, and an appropriate integer would be incremented by one, and then discarded. I realized there would be no need to save any sort of list about what the satellite's PRN was, or what SNR was connected to what satellite. This saved on space and on processing power when computing the difference between the two Streams and their signals. These values were to be reset in the Stream class/struct whenever a new GSV sentence would be read in.

To calculate the offsets/differences between the two Streams, I decided a simple calculation of deducting the difference between the Latitude and Longitude of Stream 1 with those of Stream 2 was enough as an offset. Then whenever Stream 2 was needed to be output, I would add the Latitude and Longitude offset values onto the Latitude and Longitude of Stream 2, to give a placement close to Stream 1. For the sake of my assignment, I considered these as 'Primary' (Stream 1) and 'Secondary' (Stream 2) streams.

Programming

I decided to begin programming the first version of the application in Java, as I had created more applications in that language than in C, and was not too comfortable with C++ yet. During the coding of each language, I made notes on those things that hindered or helped me with each language, that I have set aside on a note pad. These are for the next portion of the assignment, when comparing the languages and documenting my experiences with them. The general structure and flow of data between each application was practically identical in the end.

My first step in writing the Java code was planning out my data structure. I considered using a different class for each sentence, along with an abstract class that each of these classes could extend from that contained shared methods. I soon cast aside this idea though when I realized how simple getting and storing the data of the sentences was with just a few methods within a Stream class. A Stream class could receive a sentence read in by a File reader as a string, see what type it was, and then use methods from within the class itself to set it's own variables. I found this was an easy process, and the result was a nice flow of data: From File Reader to a Coordinator to the Stream class to an XML tagger to a File outputter.

Once I had built my Java application, I tested it by putting the GPX file into a website, this was: http://www.gpsvisualizer.com/. The site takes a GPX file and plots it onto a map to be seen visually. Using this tool helped me greatly in seeing if there were any areas where my application may have lost precision on the Latitude and Longitude points, or the offsets had not been applied to Stream 2 properly. Once I had sufficiently tested my application and commented it, I moved onto programming in C.

I knew that my C program would be very similar to the Java application, in such that instead of classes, I would just have C source files, with a struct to represent the data of a Stream. I found the conversion from Java to C relatively simple. It took some time to find the appropriate C functions and ways of doing the things, that were easy in Java, in C but the resulting application has much the same structure and functionality. After implementing the application in C, I moved on to programming in C++.

C++ was interesting to code, as I found myself using the same structure as my Java application, but writing the methods in much the same was as the C functions. Indeed, many of the methos I wrote for the C++ application were almost identical to those functions from my C application, just using the standard C++ way of handling things rather than those of C. Since the C++ application was similar to that of Java and C, I found I wrote the code for it in less time than it took for either of the other applications, and was in fact probably the easiest to write for. This was helped by the fact that I already knew the structure of my application and the logical ordering of my function/method calls and where my conditional statements should be placed. Again, upon completion of the C++ application, I tested the route on gpsvisualizer to ensure that it looked similar to those from the other applications.

The final steps were to record my screen casts for each application, which, aside from finding a decent screen recorder, went without a hitch.