CS22510 assignment 1
Where have I been? — processing GPS data

Version 2

Fred Labrosse Jon Bell

March 18, 2014

1 Introduction

This is the first of two assignments in CS22510. It will count for 40% of the total marks for the module. You should therefore spend roughly 40 hours of work on this assignment. **The (extended) deadline for this assignment is Friday 28th March 2014 at 5pm and must be submitted via blackboard.** The second assignment is worth 20% of the module's mark.

The first assignment is about implementing a given task using the three languages covered by the module while the second assignment is about writing a short essay discussing the merits of the three languages with respect to the task, highlighting where each language helped or hindered you.

2 The task

GPS (Global Positioning System) receivers output data in a format called NMEA (National Marine Electronics Association). This format can handle more than just GPS data, but this is what concerns us for this assignment. The format is essentially a comma-separated file. Usually the data is directly streamed from, in our case, a GPS receiver for as long as the receiver is connected. For this assignment however, we will use a file containing such data. This means that you cannot assume the number of lines of data and should therefore use dynamically allocated data structures.

Each line, or sentence in NMEA speak, gives information about the GPS receiver and the data it calculates from reception from the satellites it can see. Such data includes position of the receiver, number of satellites used and fix quality. The NMEA standard we will use is 0183[1] and an exhaustive description can be found at `http://www.gpsinformation.org/dale/nmea.htm`.

GPS receivers are not perfect, and will fail for a number of reasons that depend on the environment the receiver is in and on the quality of the receiver. Different receivers may fail in different ways. In particular, two receivers at the same position will not give exactly the same answer. In other cases, one of the receivers may loose its fix (or get a too low quality fix, see Section 3) or fail to provide data altogether (for example with sophisticated receivers that need very good satellite coverage). It is therefore useful to build up some redundancy in systems where getting a position is crucial. In this assignment, we will use two receivers (and therefore two files) moving together and assumed to be on the same point on the vehicle carrying them. The aim of the assignment is to output a single record of where the vehicle went from the two receivers.

A simple (simplistic) strategy is to consider that one of the receivers (GPS_1), when it returns a good quality fix, is correct (returns the correct position) and that the other (GPS_2) can be aligned to that one by introducing an offset in the value returned. Then, when GPS_1 fails (fix quality too low or no data), then its values can be replaced with that returned by GPS_2 appropriately

[1] `http://en.wikipedia.org/wiki/NMEA_0183`

aligned. For this assignment, you should expect a position every second. So if a time gap is larger than that, then some data is missing and must be replaced.

The alignment of one stream to the other must be calibrated. This should be done as the data is acquired using the valid pairs of positions (same time and fix good enough). When one of the streams fails, the last valid pair should be used to offset the position from the stream. More complicated schemes could be used (such as interpolation of the offset between valid pairs before and after the time gap), but this will suffice for the assignment.

The output of the program must be a GPX (GPS eXchange Format)[2] file of the corrected stream. You need to provide **three** implementations of the same program, one in each of the three languages covered by the module (Java, C and C++). Obviously, all three implementations should reach the same (or similar) outcomes. This is a good way for you to check that your implementation is correct.

Data files containing typical logs of running two different GPS receivers can be obtained from Blackboard (as `gps_1.dat` and `gps_2.dat`). Your programs must be able to deal with these files but we may want to run your programs with other such files. It is up to you to figure out which NMEA sentences are relevant and what the format of a GPX file is.

3 GPS fix

For a fix to be obtained, at least three satellites must be visible from the GPS receiver. Moreover, the signal quality received from the satellites must be good enough.

GSA sentences give some measure of fix quality in the form of "no fix", "2D fix" or "3D fix". However, most modern GPS receivers will have a 3D fix even in very poor conditions, and the position returned will not be very good. A better indication is the Signal to Noise Ratio (SNR) of transmissions from the satellites the receiver is getting information from. This can be obtained from GSV sentences and the rule of thumb is that a fix obtained from at least 3 satellites with a SNR above 35 will be good enough. The strict minimum for an acceptable fix would be at least three satellites, all of them with an SNR of 30 or above. More satellites and/or better SNR is better!

For this assignment, we will consider that a fix obtained with fewer than three satellites with an SNR value above 30 has to be obtained from the other receiver. SNR values between 30 and 35 are acceptable, but positions obtained from three satellites with an SNR in that range or above (and fewer than three with an SNR above 35) could be replaced by positions from the other stream.

4 What to hand in

The hand in must be done using Blackboard. You must hand in the following files:

- **A README file** that describes what is what. In particular this must give the names of the various files you hand in. This must also describe how to build your code, including a mention of standard libraries/packages and the versions you used.
- **Your programs**. Make sure that all the necessary files are included. Do not include standard libraries.
- **An example of the output**. This must be the GPX file (one file if they are all the same, several if the files you produce are different)
- **Three screencasts**, one for each program/language, showing:
    1. the missing GPX file before execution (using for example `ls output.gpx`, or whatever file name you used);

[2] `http://en.wikipedia.org/wiki/GPS_eXchange_Format`

2. the building of your code, with all errors (hopefully none) and warnings (hopefully not too many);
3. the running of your code, clearly showing the output;
4. the beginning of the GPX file (using for example `more output.gpx`).

The screencasts should probably include voice to explain what is going on.

- **A document** (maximum two pages long) that explains what you have done.

5 Assessment

This assignment will be assessed using the Assessment Criteria for Development. This can be found in the Student Handbook, Appendix AA, at `http://www.aber.ac.uk/compsci/Dept/Teaching/Handbook/AppendixAA.pdf`. The usual requirements for coding projects apply, namely that programs should be well commented with comments that add real value and do not just, in essence, duplicate code. Programs should have good layout and must use meaningful names for variables, classes and other identifiers.

More specifically, the detailed marking scheme for this particular assignment is given in Table 1.

Table 1: Assessed aspects of your work and their value

What	Value
Code layout	5%
Identifier names	5%
Comments	10%
Readability	10%
Documentation	10%
For each of the three implementations:	
Program quality	5%
Program compilation	5%
Program success	10%

6 Final remarks

You are more than welcome to use existing libraries to help you in your task. This could include generic libraries such as boost in C++ or more specific ones to help with the parsing of NMEA or the generation of GPX files. Make sure you acknowledge that in the documentation of your code.

This is an individual assignment and that it must therefore be completed as a one person effort by the student submitting the work as their own. In particular, your attention is drawn to Section 5 of the student handbook[3]. If in doubt, make sure that you properly acknowledge material (including code that is reused or closely derived from others) in order to avoid any suspicion that you are trying to cheat and ask for advice if you are not sure. By submitting your work to Blackboard, you agree to the statement about the Declaration of Originality. This statement is the same as that shown on the Anonymous Marking Sheet Cover. You do not need to submit a separate declaration form.

[3] `http://www.aber.ac.uk/~dcswww/Dept/Teaching/Handbook/`

7 Final, final remarks

Following questions from various students, by email or during lectures, here are additional comments. They all correspond to simplifications of the task to be done.

The two GPS data streams commence and finish at slightly different times. You may, if that helps, manually remove the extraneous sentences that do not have a time equivalent in the other file. Similarly, I do not believe there is any missing data, i.e. once the few first and last sentences are removed (or specifically processed), all times should be there in both files. It is however safer to check that when you read sentences the times match. I am happy for you to only check, and make your programs gracefully fail if one time point is missing. A better approach, but not necessary, it to treat such cases as if there was a bad fix, i.e. create the missing position from the other, corrected, position.

The sections above mention that the data should be considered as being streamed. What that means is that you should not read the whole files first and save the data in, e.g., an array, but process one sentence at a time, read directly from the file. This is not any more difficult. Also, you do not need several threads to do that. You can read fron one file, then the from the other in the same thread.

The text above mentions an offset between positions. A valid pair (good enough SNR) of positions from the two streams will be off by some offset. This is inherent to the way GPS works. This offset is not constant, but changes slowly with time. If a newly read in position is judged not good enough (based on the SNR of the satellite), then the bad reading can be corrected from the good reading and the offset calculated from the previous valid pair of positions. The position that is then saved in the GPX file is the corrected position. To calculate the offset, you only need to consider latitude and longitude, and the offset is simply the difference in latitude and longitude between the two positions. You can ignore altitude.

There are many types of sentences in NMEA files. All you need to extract is time, latitude and longitude (from either RMC or GGA sentences) and SNR values (from GSV sentences). You can ignore all the other sentences. GSV sentences do not contain time. It is likely that the official specification (proprietary) states to which RMC or GGA sentences they apply, but for the purpose of this assignment, you can either use the last one before or the first one after. You decide (based on how your algorithm works).

As stated in Section 6, you are free to use any library that you may find that does part of the job. Make sure you document that in your code and discuss that in Assignment 2.

NMEA files are simple! They are essentially CSV files, with variable length lines. The first 6 characters tell you what type the sentence is. Based on that you decide to skip the sentence (some information that you don't need) or to carry on reading it. The NMEA specification exactly defines how to read the remaining parts of each sentence, basically a list of strings, characters and numbers separated by commas. Google "CSV scanf" for hints on how to do that in C. There are similar results for C++. Java will probably need a string tokaniser

There is no specific environment on which you need to run your code, or C or C++ specific standard to use. Just make sure you specify the environment you used in the README file.

The files initially posted on Blackboard have Unix line endings. You can either use tools like `unix2dos` to convert the files, tell which ever `readline()` function you use to use the correct line termination, or use the "dos" files I sent by email.

GPX is a specific file format, that has constraints on how latitudes and longitudes are written. That obviously needs to be obeyed to. On the other hand, you do not need to include any other info (than the corrected latitude and longitude) in the GPX file.

Use the (max) two pages long document that you need to submit to describe assumptions and simplifications you made.

If you have any more questions, please contact Fred Labrosse (`ffl@aber.ac.uk`).