E-AGLE Trento
Racing Team

Ubx parser

Eugenio Berretta eugenio.berretta@studenti.unitn.it

Academic year 2019/2020

Contents

1 Concept

1.1 Introduction

The telemetry uses two high-precision GPS, one as base-station and another in the car as a rover. The rover GPS data are parsed by the telemetry and saved in a local mongodb. The base-station saves instead a raw ubx log file. The purpose of this project is making easy parsing the wanted data (latitude, longitude timestamp altitude and speed) and save it in an easy-handable format such as json and csv.

1.2 History

On Sunday 23th February 2020 some tests have been taken place. The main purpose was testing the new GPS, that consisted in a base and a rover. The rover was plugged directly to the telemetry and its data was read from the serial port, parsed and saved. The base was still, it was plugged on a pc and its data was gathered by a pre-made program and all its messages were saved in an ubx file. We needed only some messages and in order to use that data, they should also be parsed in json and csv.

2 Decision

2.1 Why nodejs?

There were no performance reasons to choose a low level language such as C to write this project. We decided to write this parser during the tests, so we needed to have it quickly. A good choice could have been using python, but I decided to write it in Javascript because I knew it better. The parser is a simple script that runs over nodejs and takes the ubx inputs in order to generate the json or the csv parsed outputs.

2.2 Why a command line program?

Like with the telemetry exporter, we could have written this parser with a GUI, developing a webapp or something. But this project was less important then the exporter and would have been used by only one or two members of the team. Hence we decided to stay with the simpler solution that was a normal command line program.

2.3 Why GGA, GGL and RMC messages?

Only the gps messages GGA, GGL and RMC are parsed, all the other messages are ignored by the parser. Initially only the car coordinates were needed, in order to plot the track on Google Earth and get a first raw evidence that they worked properly. After that, the analisys team needed the coordinates and the speed of the car. The GGA, GGL and RMC messages are the ones that together provide these values and the other messages were consequently ignored. Also the message VTG provided the speed of the car, but we discarded it because it had no timestamp.

2.4 Why NaN on some values?

The values that are extracted are latitude longitude timestamp altitude speed and course. The coordinates are converted in a more suitable format. Not all messages contain every parameter and initially every parameter that is not contained in a message was set to NaN. Later we decided to change it with NaN, becuase it was easier to handle with MatLab.

3 Implementation

The code of the exporter can be found in this github repo: https://github.com/eagletrt/eagletrt-ubx-parser

3.1 Dependencies

This only external used module is chalk, to a coloured and pretty log. All the others (fs and path) are already provided by NodeJS.

3.2 main.js

This is the root file of the script. It simply reads the config.json file in order to get the inputs file and the format of the output, parses them and write the generated files.

3.3 utils

The /utils folder contains the principal functions that parse the three types of messages (GGA, GGL, RMC).

3.4 config.json

It should be the only file modified by the user in order to configure it. It is a simple object with two keys. TYPE is the desired output format, it can be "JSON" or "CSV". INPUTS is an array containing the name without extension of the ubx input files.

4 User usage

To use this parser:

1. Clone the repo https://github.com/eagletrt/eagletrt-ubx-parser
2. Install nodejs
3. Navigate to the root folder of the repo
4. Execute npm i
5. Move the ubx file which are to be exported in the /inputs folder
6. Open the config.json file, it will contain two properties:
   - TYPE is the output format. It can be CSV or JSON
   - INPUTS is an array containing the name (without extension) of the file in the inputs folder that are to be parsed
7. Change the config.json file as wanted
8. Execute npm start, the parsed file will be in the outputs/csv or outputs/json folder

5 Conclusion

5.1 Summary

This project is a good example of "Do one thing and do it well". During a test we needed to parse some ubx files and we developed a solution that was very simple, written with a very hight-level language, but worked and was ready in a matter of minutes. The parser has been used also after the tests.

5.2 Future

There are no changes planned to this parser. It works and is used by only one or two persons. In the future, if there will be time, it can be developed with a GUI. It could for instance be re-written in another language or as a Desktop app written as a site and built with Electron.