Flight recorder simulator in Python

Project for the Python in the Enterprise by T.Szumlak

Rafal Sikora

April 8, 2016

Abstract

This is the report explaining basic concepts and features of the "black box" simulator - an assesment project for the Python in the Enterprise, implemented under python in version 2.7. It contains description of all components of the program and user tutorial, as well as discussion of possible future extensions that have been considered during development of described software.

1 Project objectives

General requirements which implemented flight recorder should fulfil were provided by the teacher in the following PDF file: http://home.agh.edu.pl/∼szumlak/lects/pite_2016_l/sproject_1.pdf. The main objective of the project was to implement a software behaving as a flight recorder - container storing some data which could be accessed after the flight.

Given wide freedom in the capabilities that implemented software should present I decided to focus on the graphical part of the project. Such choice was motivated by the fact of lack of any experience with graphical user interface implementation in any of known programming languages. This project was an excellent motivation to gain such skill.

Sub-objectives which I initially set for the project:

1. Creating user-friendly graphical interface
2. Reading parameters of the real flights from some open-access database (www)
3. Usign BSON (binary JSON) for the data storage
4. Using pyROOT for graphical presentation of the data

2 Implementation

Created software consists of the following files:

1. flightRecorderSimulator.py - main program file
2. AirportsList.py - module responsible for reading a list of all registered airports in the world (with their geographical coordinates etc.)
3. AirPlane.py - abstract base class representing an airplane
4. Boeing747.py - subclass derived from AirPlane representing Boeing747 airplane
5. BlackBox.py - class representing a flight-recorder
6. MainWindow.py - GUI module
7. DataViewer.py - class responsible for analysing and drawing data from black-box
8. Coordinates.py - module with method translating geographical coordinates to coordinates on the flat surface

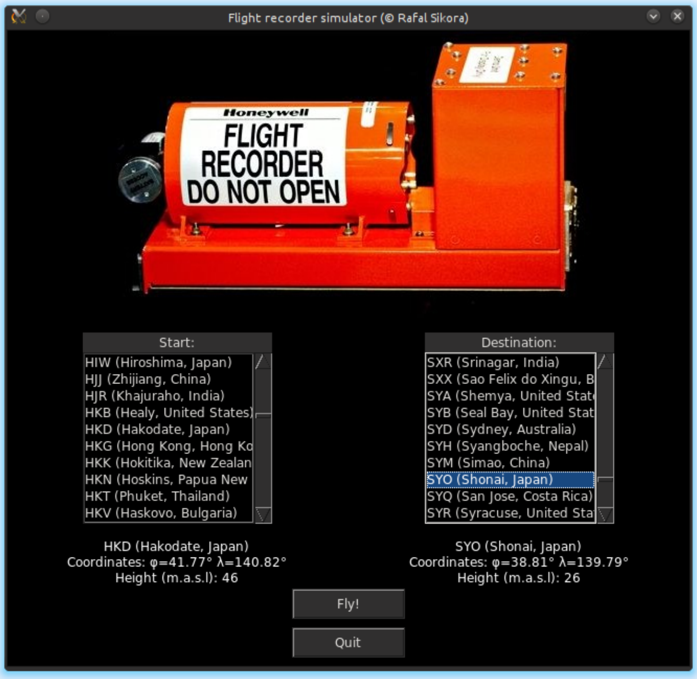
At the beginnin of project implementation I was forced to change one of the objectives, namely the method of obtaining the flight data. I encountered difficulty in finding online database with the parameters of flights, therefore I had to create an algorithm which produce values of pitch, height, etc. based on random number generation.

I decided to let user choose the starting and ending flight point. For this purpose I used a database of all world airports available at http://openflights.org/data.html.

In order to allow further customisation of flights, an abstract class AirPlane was introduced. So far only one instantiation of AirPlane was created, Boeing747.

A BlackBox class directly representing the flight recorder was implemented in such a way, that it is straightforward to extend number/type of data which it stores. It was achieved by reading a single data package in a form of dictionary, which does not have predefined number of keys(elements). This is very useful feature since the same black-box can be used in any type of an airplane (each airplane may record different parameters).

BlackBox data was stored in a text file using JSON format. This is unfortunately not primary-design solution (first idea was to use BSON format). Binary files are much more leightweight than text files, however limited time resources did not allow to overcome difficulties with BSON format handling.



HKD (Hakodate, Japan)
Coordinates: φ=41.77° λ=140.82°
Height (m.a.s.l): 46

SYO (Shonai, Japan)
Coordinates: φ=38.81° λ=139.79°
Height (m.a.s.l): 26

Figure 1: Main program window.

Graphical user interface, presented in the Fig. 1, was implemented using Tkinter package. It allows user to choose flight start point and destination from two separate listboxes. Once a list item is clicked, short info about chosen airport is printed below the listbox. In the central bottom part of GUI there are "Fly!" and "Quit" buttons. If items from both listboxes are selected, it is possible to run a simulation via "Fly!" button.

Results of the flight simulation are read from BlackBox and passed to DataViewer class, where the data is converted and drawn using pyROOT (python version of the C++ ROOT library). Output is saved as graphics (PNG files) in the program directory. There are two output files: one contains world map with a flight path drawn (Fig. 2), the other presents graphs of time dependence of parameters recorded by the BlackBox in

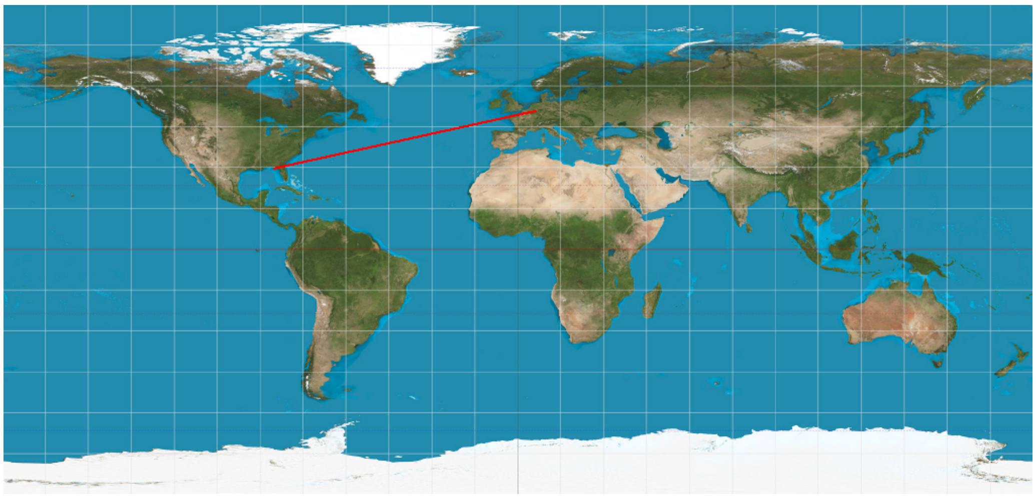


Figure 2: Sample of "flightPath.png" output file. Path of flight is approximately a straight line because of equirectangular projection used for drawing.

particular AirPlane (ommiting φ and λ).

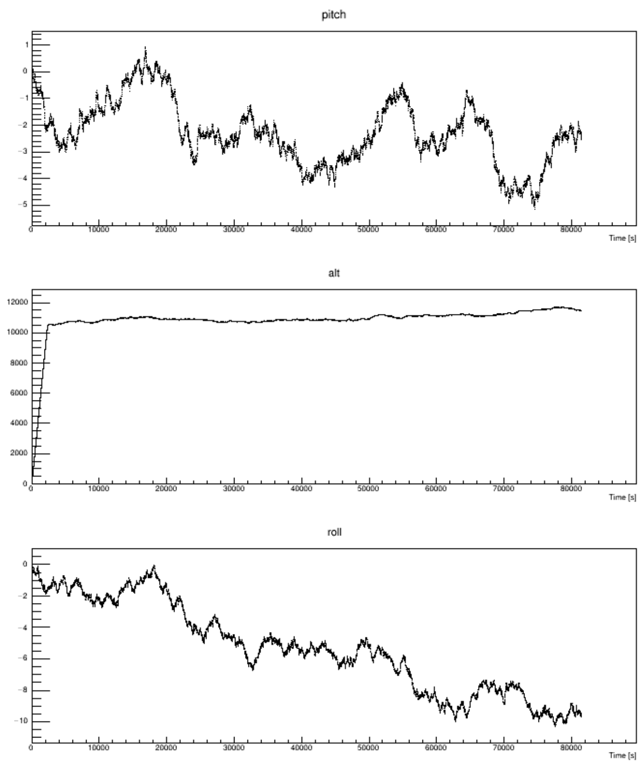


Figure 3: Sample of "flightParameters.png" output file.

3 Program usage

1. Download flight recorder package from GitHub repository:
   https://github.com/rafalsikora/PITE_FlightRecorder
2. Start "flightRecorderSimulator.py" with a standard command
   :~$ python flightRecorderSimulator.py.
   GUI window should pop up on the screen.
3. Select start and destination airports from two listboxes.
4. Click "Fly!" button. Wait until you see text information in console about finished flight simulation.
5. Output files will appear in program directory.

4 Known program limitations

- Parameters of airplanes (so far only of Boeing 747) are not verified in terms of their correspondence with real values
- Simulation of flight dynamics is incompleted (mainly solving the equations of motion)
- Software does not verify if the distance between two selected airports is reachable for any passenger airplane in a single flight (without refuel)

5 Conclusions

Presented version of flight recorder simulator meets a larger fraction initial objectives. Basic functionalities which have been implemented can be relatively easily extended. During development of this software one could gain valuable skills such us GUI implementation, usage of JSON format and pyROOT libraries.