UNIVERSITÀ DEGLI STUDI DI PISA

FACOLTÀ DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA AEROSPAZIALE

TESI DI LAUREA SPECIALISTICA

DEVELOPMENT AND VALIDATION OF A PRE-POST PROCESSOR FOR A ROTORCRAFT SIMULATION CODE

Relatori:

Prof. Ing. **Eugenio Denti**

Prof. **Attilio Salvetti**

Ing. **Riccardo Bianco Mengotti**

Candidato:

Stefano Mariotti

ANNO ACCADEMICO 2011 - 2012

Ai miei Genitori

Ringraziamenti

Un sincero riconoscimento e ringraziamento ai miei relatori; per primo il Prof. Ing. Eugenio Denti che mi ha affiancato e supportato durante lo svolgimento di questo lavoro e al quale devo questa splendida esperienza in AgustaWestland. Il Prof. Attilio Salvetti che allo stesso modo mi ha affiancato nello svolgimento del presente lavoro. L'Ing. Riccardo Bianco Mengotti che mi ha dato la possibilità di svolgere lo stage in AW seguendomi costantemente per tutto il suo svolgimento.

Un ringraziamento sincero per i miei genitori che mi hanno sempre sostenuto e incoraggiato nel raggiungimento dei miei obiettivi. Grazie a mio fratello Alfredo, per me rappresenta sempre il giusto esempio da seguire! Grazie a mia sorella Azzurra, Claudio e i miei due bellissimi nipotini Mattia e Matilde, che presto diventeranno tre con Ettore, per tutti i vostri incoraggiamenti!

Un ringraziamento speciale alla mia Annina, che mi ha sempre sostenuto nei momenti di difficoltà, e ce ne sono stati molti ultimamente!! Grazie Piccola!

Grazie a tutti i miei cari amici con i quali ho condiviso questo percorso, senza di voi non sarebbe stata la stessa cosa! Grazie; Marco, Gerardo, Giuà, Giuseppe e tutti gli altri che non cito, ma ringrazio. Grazie agli amici della chiesa che hanno sempre cercato di sviarmi dai miei momenti di studio, spesso senza riuscirci. Grazie; Perpe, Simone, Roberto, Pucci, Rosi, e tutti gli altri che non cito perché siete troppi! Grazie a tutti!

Grazie ai ragazzi della meccanica del volo che mi hanno sempre aiutato e incoraggiato. Grazie ai Senatori; F.Scorcelletti e P.Sabato per avermi ammesso al senato con i suoi membri Raffo e Fresta. Grazie ai barbari; Di Trocchio, Bicego e Bosk e agli astenuti dagli schieramenti; Andrea, Del Grande, Gianca, Fabio e Luca per le tutte le risate di questi giorni, senza di voi sarebbe stato tutto più difficile!

Sommario

L'obiettivo del presente lavoro di tesi è sviluppare un pre-post processor (R-GUI) per un codice di simulazione di dinamica del volo di velivoli ad ala rotante sviluppato internamente al dipartimento di Flight Mechanics di AgustaWestland.
Il lavoro è suddiviso in una prima fase di definizione dei requisiti in cui si è proceduto alla stesura della specifica di progetto basandosi sulle richieste degli specialisti di meccanica del volo in lavoro presso il suddetto dipartimento e confrontando altri software di pre-post processing, sia commerciali che non, già in utilizzo. In base alla specifica si è passati allo sviluppo della piattaforma di analisi, tendendo conto sia delle richieste di elaborazione dati che di quelle di tipo grafiche di interfaccia. Il codice sviluppato è stato poi validato, con semplici test cases, comparando i risultati con quelli ottenuti da metodi di utilizzo convenzionali del codice di simulazione.
Nella seconda parte del lavoro si è passati all'implementazione della funzione di linearizzazione del modello, prima applicata al codice di simulazione e poi al pre-post processor. La funzione sviluppata è stata poi validata confrontando le risposte lineari con quelle non lineari relative ai test cases utilizzati nella meccanica del volo.
In fine è stato eseguito lo studio dei poli del sistema elicottero al variare della velocità di avanzamento arrivando a tracciare il relativo luogo delle radici al fine di valutare il comportamento dell'elicottero e la validità del modello utilizzato nonché del codice di simulazione.

Abstract

The scope of this thesis is to develop a pre-post processor (R-GUI) for a flight dynamics rotorcraft simulation code which has already developed in the Flight Mechanics department of AgustaWestland.

The work can be divided in two parts. At First is defined the specific of the pre-post processor following the requests of the specialists working in the Flight Mechanics Department of AW and comparing, both commercial and open source software, in order to define the best solution. According to the specific, the platform has been developed taking into account both data processing and graphic interface demands. Hence, R-GUI has been validated using simple test cases and comparing its results with those obtained by conventional methods of analysis.

In the second part the model linearization function has been developed, first to the simulation code and then to the pre-post processor. The function is validated using simple test cases and comparing the linear responses with the non-linear one. At the end the poles of the helicopter system have been studied as function of speed, plotting the root locus and analyzing the helicopter behaviour, making some consideration about the validity of the model as well as the simulation code.

Index

List of figure

List of Table

List of Acronyms

FMI	Flight Mechanics Input
FMO	Flight Mechanics output
FMR	Flight Mechanics rotorcraft

Introduzione

La tesi è strutturata in cinque capitoli.

Il primo capitolo si prefigge lo scopo di fornire le motivazioni che hanno portato allo sviluppo di R-GUI e l'ambiente in cui esso si colloca, presentato il codice di simulazione (RSim) rispetto al quale è stato sviluppato. Sono presentate a titolo di esempio alcune interfacce grafiche utilizzate da programmi simili di pre-post processing utilizzati nel dipartimento HSD di AgustaWestland. Vengono inoltre fissati i requisiti base a cui R-GUI deve soddisfare, in termini di gestione e elaborazione dati, dovuti a RSim. In fine è presentata una panoramica dei linguaggi di programmazione utilizzabili per creare R-GUI e più nello specifico quello con il quale è stato implementato.

Nel secondo capitolo viene riportato il percorso concettuale che ha portato allo sviluppo di R-GUI. Partendo dalla definizione della specifica di progetto sono definite le funzioni di interesse e il modo in cui queste sono state interpretate e implementate sia dal punto di vista di gestione dati che di rappresentazione grafica.

Nel terzo capitolo sono presentati i risultati relativi all'utilizzo di R-GUI, prendendo in esempio alcuni semplici test cases, rappresentanti varie condizioni di volo. Le risposte sono state comparate con quelle ottenute in modo convenzionale, andando a considerare non solo il risultato numerico, ma anche la facilità di uso ed i benefici in termini di tempo e di utilizzo apportati da R-GUI.

Il quarto capitolo, invece, riguarda la definizione di una nuova funzione implementata nel codice di simulazione con lo scopo di definire il modello linearizzato dell'elicottero. Sono quindi definiti i concetti base e i modi operativi di calcolo utilizzati per arrivare alla sua definizione. Si è passati quindi alla sua validazione, andando a comparare le risposte lineari del modello con quelle ottenute nel caso non lineare per alcuni test cases. In ultimo è presentato uno studio sui poli della macchina, definiti in funzione della velocità di avanzamento, arrivando a tracciare il relativo luogo delle radici.

Il quinto capitolo riporta le conclusioni che possono essere tratte dal presente lavoro. Per prime quelle riguardanti l'utilizzo di R-GUI, e in ultimo quelle relative alla funzione di linearizzazione deducibili dallo studio delle risposte linearizzate. Altre importanti conclusioni riguardanti il modello lineare e il codice di simulazione sono state dedotte dall'analisi del luogo delle radici.

Introduction

This thesis is divided into five chapters.

The first chapter aims to provide the reasons which have led to the development of R-GUI and the environment in which it is located, presenting the simulation code (RSim) to which it has been developed. As examples given, some graphical user interfaces took by similar pre-post processors used in the HSD Department of AgustaWestland have been presented. In additional the basic requirements which R-GUI must satisfy are set out in terms of processing and data management defined by RSim. At least, there is an overview on the programming languages may be used to create R-GUI and more specifically the one used to develop it.

In the second chapter the conceptual scheme of the R-GUI development have been defined. Starting from the specific, the functions and their implementations are defined in graphic and data management way.

The third chapter presents some test cases about typical flight condition, which are resolved both using R-GUI and the conventional way. The responses are compared not only in numerical way but analyzing the advantages brought by it.

The fourth chapter regards the definition of a new function applied to the simulation code and then to R-GUI. Its goal is the evaluation of the linear model of the rotorcraft. In this chapter, the concepts and the methods used to define it have been presented. Therefore, it is validated using some typical test cases through the comparison between the linear and non linear responses. At the end the behaviour of the vehicle are studied through its poles which have been defined as function of speed and represented in the root locus.

The last chapter reports the results of this thesis; at first those brought by the use of R-GUI and secondly those deduced by the comparison between the linear and non linear responses. Also, other important conclusions, about the model and the simulation code, are defined from the rotorcraft poles and the root locus analysis.

1. Requirements and Specification

The aim of this chapter is to define the framework in which the R-GUI has been developed. The principal aspects treated are: state of art, which reasons have led to the R-GUI development, which requirements R-GUI have to satisfy. At the end, it is proposed a short outline of the programming language used to develop it.

1.1. State of Art

R-GUI is a pre-post processor developed to improve the use a flight dynamics rotorcraft simulator code called RSim. From the conceptual point of view, R-GUI uses specific function of RSim through graphical representation of objects, hence using the graphical language.

A language is first of all a communication system which allows to express concepts and ideas. The philosophy of language classifies languages in two main categories natural and artificial languages. A natural language is any language which arises in an unpremeditated way as the result of the innate facility for language possessed by the human intellect: the rules and the structure of the language have not been established by anyone, but are the result of a consolidation process of usage and procedure. A natural language is typically used for communication, and may be spoken, signed, or written. On the other side, an artificial language, is any language whose structure and rules have been consciously devised or modified by an individual or group, instead, of having evolved naturally.

The graphic language is an “artificial” language that use graphic elements to present functions or specific operations which are performed. This is the basic concept on which are developed the graphical interfaces. Indeed R-GUI is also a graphical interface which has to help the user in the use of RSim. In the same way, every high level software needed of a pre-post processor to perform and present the functions implemented. Some of the main software, both

commercial and open source, used in the HSD of AgustaWestland are: Adams, Ideas, Revis, Giaguaro and each one have a specific pre-post processor that allows the user to use correctly and quickly their functions. In the follow, some examples of the graphical interfaces implementation for some of them are presented.

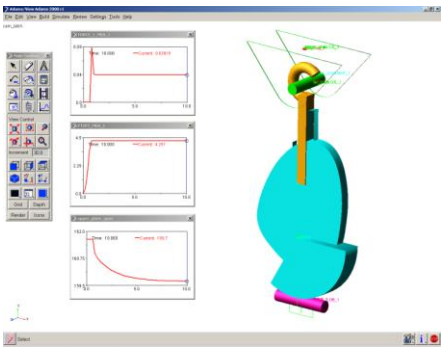

Figure 1.1 - Adams

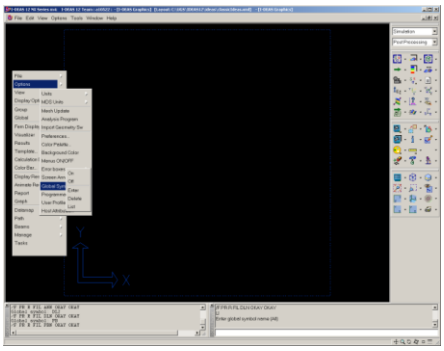

Figure 1.2 - Ideas

1.2. R-GUI background and motivation

R-GUI is born in the Flight mechanics department of AgustaWestland and it is designed to improve the use of Rotorcraft Simulation code, RSim (see RSim code simulation).

1.2.1. RSim code simulation

RSim is a routine collection for real-time vertical-lift simulation. Its final target is the development of flight dynamic simulation model that represent correctly the mechanic and dynamic of flight of a rotary wing aircraft, both in conventional (Main/Tail Rotor) or unconventional (Tilt Rotor) configuration. RSim is developed to produce a suitable reconfigurable real-time simulation system. A simplified methodology is used (rigid helicopter and algebraic rotor), that would result in a minor risk for the development/maintenance of the code, in a more affordable real-time use and in an easier model automatic tuning ([2]).

The architecture of RSim is sketched in Figure 1.3 where the conceptual work scheme of the software is represented.

The input and output data are organized as ASCII file in FMI and FMO format, respectively (see RSim data section). The input file, in FMI format, is created from the user that is going to run a specific simulation, while the software will generate the output file, in FMO format, which contained the results of the simulation performed.

The software will start executing the application file called “RSim Caller.exe” from Command Prompt of Windows®. As sketched in Figure 1.3 this application file read the input file and perform the simulation requested calling the specifics functions which are organized in two levels; high and low.

The high level functions are grouped in a library called “RSim_Analysis.dll”. They are used to perform the analysis case requested in the input file, as showed in Figure 1.3. Instead the low level library is called “RSim_Model.dll” and its functions are used to perform the basic operations executed by the high level one. In additional this library load, through another set of functions which have been previously developed in AW, the model requested for the analysis as FMR file (see RSim data section). These additional functions are get together in the library called “AW_FMR.dll” showed in Figure 1.3.

The special organization of these libraries allows the republication of the lower level functions than the present one. For example, the library “RSim_Model.dll” publishes its functions and the functions of the previous levels, and so on…

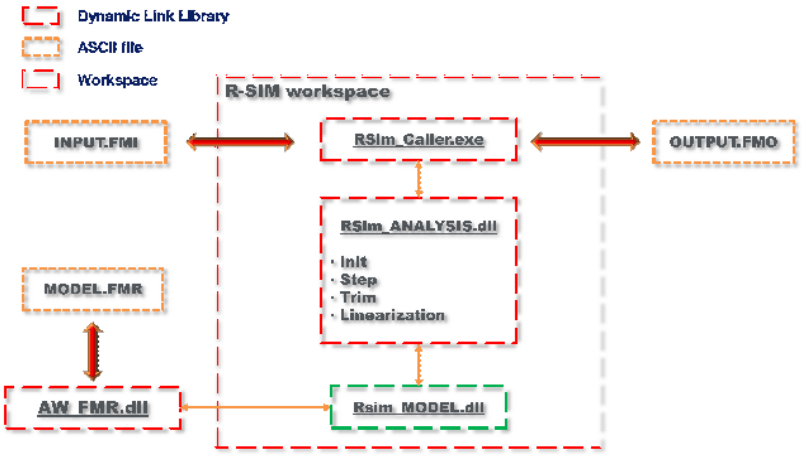

Figure 1.3 - RSim software architecture ([2])

1.2.2. RSim data format

How defined in the previous section the file types used by RSim are in ASCII format. These file are the input, output and model, organized in fixed format and listed in follow:

- FMR (Flight Mechanics Rotorcraft)
- FMI (Flight Mechanics Input)
- FMO (Flight Mechanics Output)

The **FMR** structure data contained all of the parameters of the model sorted in namelist ([1]).

The formats of the FMR structure are:

- standard (normal format of model);
- dump (extended format);
- crypt.

The **FMI** structure regarded the input data format. This is sorted in a single row with several areas (about eighty columns) that contained all of the inputs for the analysis ([3]) ([2]).

The **FMO** structure regarded the output data format. This is similar to the FMI but with more areas ([3]) ([2]).
Different from FMI, the FMO format can be:

- standard (with all parameters);
- compact (with only necessary parameters).

1.2.3. R-GUI goals

The mean task of the R-GUI project is the development of a pre-post processor that is able to replace the application file "RSim_Caller.exe", referred to Figure 1.3. In additional several others functions must be implemented in it, for example the managing of the ASCII file used by RSim, such as input, output and model, the real time analysis and others which will be define in the specific, see **Errore. L'origine riferimento non è stata trovata.** section. In order to improve the use of RSim, R-GUI must be able to call every function published from each librarie. Therefore is developed a cross linking between the program language of RSim (C++ or Fortran) and the program language used to developed R-GUI (FreePascal with Lazarus, see Lazarus: The Development Tool section) in order to call correctly every function published by RSim.

1.3. Lazarus: The Development Tool

The choice of the programming language used to develop R-GUI has been driven mainly by two requirements: the need of programming a GUI and the program execution speed. In addition, it is better to use a freeware licence programming tool, and the cross-platform capability is a desirable feature. A platform is a

combination of hardware and software used to run software applications. A platform can be described simply as an operating system or
computer architecture, or it can be the combination of both; hence, a crossplatform software is able to run on and operate on multiple platforms. This feature in the case of a compiler for a programming language, allows to write the program once and choosing the platform on which compile it later on. Starting from the same code, it is possible to compile an application for any Operating Systems and hardware architecture: for example, for Microsoft Windows on the x86 architecture, Linux on the x86 architecture and Mac OS X on either the PowerPC or x86 based Apple Macintosh systems, even if the architecture and the operating system used to write the program are different from those on which the program will have to run. R-GUI is first of all a GUI, hence the best choice to manage, in an easy way, all the aspects and all the problems linked to programming a GUI is to use an IDE. The acronym IDE stands for Integrated Development Environment, i.e. a software that helps programmers in the code development covering most aspects, such as editing, compiling, debugging and interface creation. Generally, an IDE is characterized by:

- Graphical Interface Editor;
- Code Editor;
- Compiler/Interpreter;
- Debugger.

The use of a visual IDE is perfect for the Graphical User Interface (GUI) development thanks to simplicity and speed with which it is possible to realize graphical interfaces in addition to usual programming functions. In fact, the main strength in using an IDE to realize a GUI is that an IDE provides to the programmes a Visual Component Library, in short VCL, which contains all those component most used in the graphical interfaces development, like buttons, check box, radio button, canvas, textbox, label etc. Moreover, thanks to the Graphical Interface Editor provided by the IDE, the GUI can be visually drawn by

the programmer, dragging and dropping on the form the desired components and writing the event handler routines, for the management of each GUI component. From a practical point of view, the IDE helps the programmer in writing the text file which will be given to the compiler/interpreter to build the program. Clearly, it is possible to project and manage a GUI using directly the programming language and the API of windows, bypassing the IDE. However ,the direct programming of a GUI is more difficult, complex and long than using an IDE. So the use of IDE in the realizing graphical interfaces is by now a standard. To satisfy the real time requirement, a certain speed in the execution of the program is needed. The programming languages can be divided in two main categories: the interpreted ones and the compiled ones. An interpreted language is a programming language in which programs are indirectly executed, i.e. interpreted, by an interpreter program, while in a compiled language the program is converted into machine code and then 'directly' executed by the host CPU. Even if an interpreted language offers some flexibilities with respect a compiled one, from the point of view of the execution performances of the implemented program, the main difference between these programming approaches is that a compiled program is faster than its interpreted correspondent. Hence, if for the considered program, the execution speed is crucial the best choice is to use a compiled programming language.

Typically IDE are related to a specific programming language, hence there are IDE for both compiled and interpreted languages. The most known interpreted languages are Visual Basic, Java, Lisp, Perl, while the compiled ones are C,C++/ Visual C,C++, Fortran, Pascal-Delphi. The satisfaction of the real time requirements leads towards the choice of a compiled programming language, neglecting the interpreted ones. Hence, now a short outlines on the main used compiled programming language is presented, pointing out their main features with the aim of understanding if they are easily usable or not for the creation of a GUI. Fortran is a general-purpose, procedural, programming language that is suited to scientific computing. It was developed in the 1950s for scientific and 14

Requirements and Specifications engineering applications, and for the next fifty years dominated the area of programming in many ambits like the structural finite element analysis, computational fluid dynamics, computational physics and computational chemistry. It was one of the most popular languages in the area of high performance computing. Successive versions have added support for processing of character-based data (FORTRAN 77), array programming, modular programming and object-based programming (Fortran 90 / 95), and object oriented

and generic programming (Fortran 2003). Unfortunately, none of this versions has a visual programming functionality, hence Fortran is not suitable to project GUI. C is a general-purpose computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system. Although C was designed for implementing system software, it is also widely used for developing portable application software. C is one of the most popular programming languages of all time. C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language derived from C. It was developed by Bjarne Stroustrup with the aim of adding new features to C like classes, in particular, virtual functions, operator overloading, multiple inheritance, and templates. C++ is widely used in the software industry. Some of its application domains include systems software, application software, device drivers, embedded software, high-performance server and client applications, and entertainment software such as video games. C/C++ have visual versions, designed to make easy the creation of GUI, the most known are Microsoft Visual C or Embarcadero C++ Builder, which are both commercial non free product respectively of Microsoft and Embarcadero. Notice that if for C/C++ compiler both open source and commercial products are available, there is not any open source visual IDE for C/C++.

Pascal is an influential imperative and procedural programming language, developed in 1970 by Niklaus Wirth as a small and efficient language intended to encourage good programming practices using structured programming and data

structuring. Successively in the 80s a derivatives version of Pascal for the object oriented programming was developed: the Object Pascal, mostly known Embarcadero Delphi[1]. Embarcadero Delphi is also a visual IDE for the GUI project, which mix the easiness in projecting GUIs with the main characteristic of all Pascal compilers: of high execution speed while producing highly optimized code. Moreover, close to Delphi, which is a commercial non-free product of Embarcadero, there is also a free IDE available: Lazarus developed by the Larazus Project Team for the FreePascal compiler. A short overview on Lazarus is presented in the next paragraph. Neglecting Fortran, for which any specific IDE for the GUI project is available, the choice is between C/C++ and Pascal/Delphi/Lazarus. Since both the visual IDE for C/C++ are commercial and the chosen programming language is Pascal, in particular the open source FreePascal compiler in conjunction with its IDE, Lazarus, which is the only with crossplatform capabilities.

1.3.1. An overview on Lazarus Ide and FreePascal

Lazarus is an open source multiplatform integrated interface programming environment, IDE, released and developed by the Free Pascal Team. As it has been underlined in the previous chapter, an IDE is made of a Graphical Interfaces Editor, a Code Editor, a Compiler and a Debugger. In Lazarus, both the Graphical Interfaces Editor and the Code Editor are proprietary, while the compiler is the FreePascal compiler and the Debugger is GDB, ie GNU Debugger. The main scope of Lazarus is to create an open-source alternative to the well known Delphi ObjectPascal compiler, but with more functionalities, among which the most important is the cross-platform capability. With this regard, being based on Free Pascal, Lazarus was the first IDE open source environment for the Windows 64 platform. The Lazarus Project started in 1999 and at present the stable available

[1] Both Delphi and C++ Builder were developed respectively in 1997 and in 1999 by Borland. In 2009 Embarcadero technologies, which is an American based database tools and application development tools software company, bought Borland, becoming the owner of its products

version (considered in this document) is the 0.9.28.2 with Free Pascal 2.24 version, even if the latest version is the 0.9.29 with Free Pascal 2.3.1 released on the end of August but at the moment is still to be considered under development. Free Pascal is an open source cross platform compiler for 32/64 bits processors. The compiler at base implements the Borland Pascal Language, Turbo Pascal and Delphi (Object Pascal), but it has also typical characteristics of the MacPascal, which can be quickly ported or adapted. The compiler supports most used processors such as, i386 (80386, 80486, Athlon, Pentium, Atom, CoreDuo etc), x86 64 bit family (Xeon, Amd64 etc), PowerPC, PowerPC64, Sparc, Arm and is available for the most common operative systems. Hence, with Lazarus it is possible to create applications for several operating system, including:

- Windows 32bit (Win95, 98, NT, Me, 2000, WinXP, Win Vista 32, Win 7 32);
- Windows 64bit (WinXP x64, Win Vista 64, Win 7 64);
- Windows CE;
- Linux (Debian, Ubuntu etc);
- Mac Os X;
- FreeBsd;
- PalmOS;
- Symbian;
- Amiga;
- Netware;
- Beos;
- Solaris;
- OS/2

Moreover, thanks to versions for Linux and Windows CE, it is possible to make Lazarus functioning or write programs for some new tablet devices or PDAs.

In addition to the cross-platform capability, Lazarus is able to read and compile, directly or with some limitations, Object Pascal (or Delphi) code and has a fairly complete database support. Lazarus also has a class library, LCL(Lazarus Component Library), in strong growth and always updated. Both Lazarus and Free Pascal have an official documentation and a wide base of online reference material, often written by the users themselves. The main source of documentation for the Free Pascal compiler is the website ([7]) which contains the official Freepascal manual as well as many free downloadable PDF and online html files. The main source of Lazarus documentation is ([8]) where the Lazarus manuals and other valuable reference material can be downloaded or used directly online.

At last, thank to its open source nature, Lazarus is completely free, in fact Lazarus is released under the GPL licence.