Implementation of the pointing model for the 40m OAN radiotelescope

T. Alonso Albi, P. de Vicente,
V. Bujarrabal

Informe Técnico IT-OAN 2010-3

Change Record

Version	Date	Author	Remarks
1.0	20-Jan-2008	T. Alonso Albi	First version

Contents

1 Introduction

We analyze the software implementation of the pointing model for the 40m radiotelescope. The aim of this application is to assist in the process of solving the equations of the pointing model as described in *IT OAN 2003-7* [1], *IT OAN 2007-26* [2], and *IT OAN 2008-7* [3]. First we will describe how to compile, run, and use the program. Next we will introduce the technical specification of the application, and we will continue with a detailed analyze of the code.

2 User operation

In this section we will introduce how to compile, run, and use the program from a user point of view. It is asumed that the user is running the Linux Operating System.

2.1 Generating the program binaries

The application source files can be downloaded from the CVS repository at Yebes. *OAN 40m Pointing Model* is a Java program that requires at least a *Java Runtime Environtment* (JRE) version 1.5 to be executed. However, to compile the code a *Java Development Kit* (JDK) version 1.5 or above is required. This JDK can be downloaded at *java.sun.com* [4]. The recommended steps to follow before proceeding with the compilation are as follows:

- To check for a possible previous installation of a Java JRE/JDK. You can type in a console *java -version*, and also *javac <some file>* to check for the *javac* program that allows to compile the code. If the version is 1.5 or above and the *javac* program exists (so a JDK is installed), then it is not necessary any update, and you can skip the following points.
- To check for a possible previous non-complete installation (the JDK itself is installed, but the previous commands links to an old JRE/JDK version). You can navigate through */usr/java* and */user/lib/java* or */user/lib/jvm*. If you find a jdk directory version 1.5 or higher, check the *./bin/java* and *./bin/javac* programs to exist. If you find them, you can skip the following step.
- Otherwise, root privileges are required for the installation. However, if any problem arise during the installation of a new version it is possible to copy the JDK directory of any other computer with the same operating system installed, and copy the entire JDK directory in some path of your user home directory. In this case, some little modificacions will be required to execute the application (instead of command *java*, use the path to your local program */home/user/.../jdk../bin/java*).

Once everything is ready to compile, you can check the *make.sh* script provided with the distribution. This script must be edited to update the variables DEV_ROOT with the current path, and $JAVA\_HOME$ with the path of the jdk directory found or installed in the previous steps. Note a file separator string / must not be present at the end of any of these variables.

Now you can ensure the script is executable (*chmod 666 make.sh* will enable it), and you can execute the script. No error should arise, but perhaps some warning/s. When finished, go

to subdirectory make and execute the *pointingModel.sh* script. The application window will appear. If an error occurs, possibly you will have to modify the script to point to the correct *java* program (including the full path) if you decided to copy the JDK directory from another computer.

The script creates a */obj* and */doc* subdirectories with the program binaries and the Java documentation. The Java documentation (properly known as Javadoc) can be visualized with any web browser starting from the */doc/index.html* file. It will be described in detail in section 3.2.

If you plan to develop or modify the code, the use of a Java IDE (Integrated Development Environtment) is strongly recommended. The IDE used in the development was *Eclipse SDK*, downloadable from *www.eclipse.org* [5], which is the one mostly used, and the distribution provides with specific configuration files to create an *Eclipse* project. The Java development is beyond the scope of this document.

2.2 Description of the *OAN 40m Pointing Model* application

The application can be optionaly executed with an special command-line option *enable fits*, adding these words at the end of the execution script. The end of the file should read something like
org.oan.swing.LaunchApplication enable fits. This is not the default in the distribution, so it should be enabled manually. This option allows the user to reduce and manage optical images in .fits format that shows the position of some star at some instant. The telescope is pointed to the theorical position of the star in azimuth and elevation, but the star is not located at the center of the CCD camera (i.e. the .fits image) due to the pointing errors. The date and the position of the antenna can be read from the .fits header, that contains this information in an special format in addition to the image itself. The format of the header will be described in detail in section 3.3. All images presented in this document are shown with this specific feature enabled, although it is now obsolete and substituted by a more sofisticate treatment of the antenna data in radio wavelengths, using Gildas software.

The first window that appears when the program is started in shown in figure 1. At the right a black square shows the picture region where some charts with the predicted and observed pointing errors can be visualized. At the left the current model parameters are listed in arcseconds. Each parameter has a check box that can be checked or unchecked to fix a given parameter and maintain it constant. The mouse cursor can be moved to any of these parameters and left there during a second to show a pop-up message with the meaning of this parameter. The current implementation of the equations (see [3] for example) discards parameter 6.

Below, the residual parameter box informs about the mean geometric variance of each parameter in a given fit. This is the expected (mean) error of each fitted parameter. This value will not be shown if a fit is performed with any of the parameters fixed.

The mean deviation shown at the bottom provides the current mean geometric variance in the current chart shown at the right. Depending on wether the chart shows azimuth errors or elevation errors and the goodness of the fit, the value can vary substantially. However, the residual parameter should be in the same range of those values.

A menu is visible at the top with common main options. These options are summarized below, sorted in the same order as any user should use them provided that an input file is

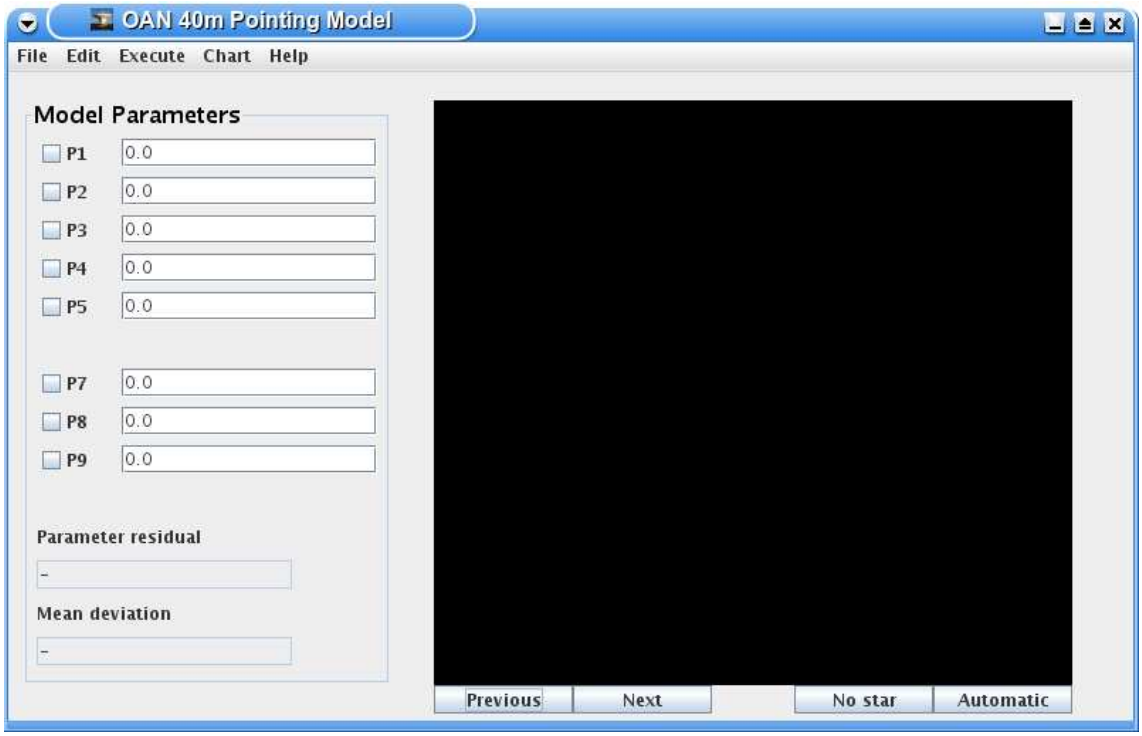


Figure 1: *The program initial window.*

available.

- Reading an input file.

  An *ASCII*/binary input file can be imported into the program by clicking *File - Read File*. The *ASCII* file should contain azimuth, elevation (degrees), azimuth error, elevation error (arcseconds) as the first four fields (and optionally the star position x, y offsets from the center of each image (pixels) and the file name, to complete 7 fields), separated by blank space/s. The binary file contains a whole set of observations with data like azimuth and elevation, azimuth and elevation pointing errors, and also the information contained in each header of the .fits files. The program will first try to read the file as an *ASCII* one, and in case of error it will be treated as binary. Both the *ASCII* and the binary files are created by the .fits reduction process that is detailed later.

  The current version also supports ASCII files generated by Gildas software.

  After the file is imported, a chart with the observed azimuths and elevations will be shown.

- Executing the Model.

  The *Execute* menu and option allows to fit the observations according to the implemented model. The model parameters (P1, ... P9) will be fitted by least-squares using the math library *LAPACK* (more precisely the subroutine called *dgells*). The parameters will appear in the corresponding text boxes, as well as the residual parameter.

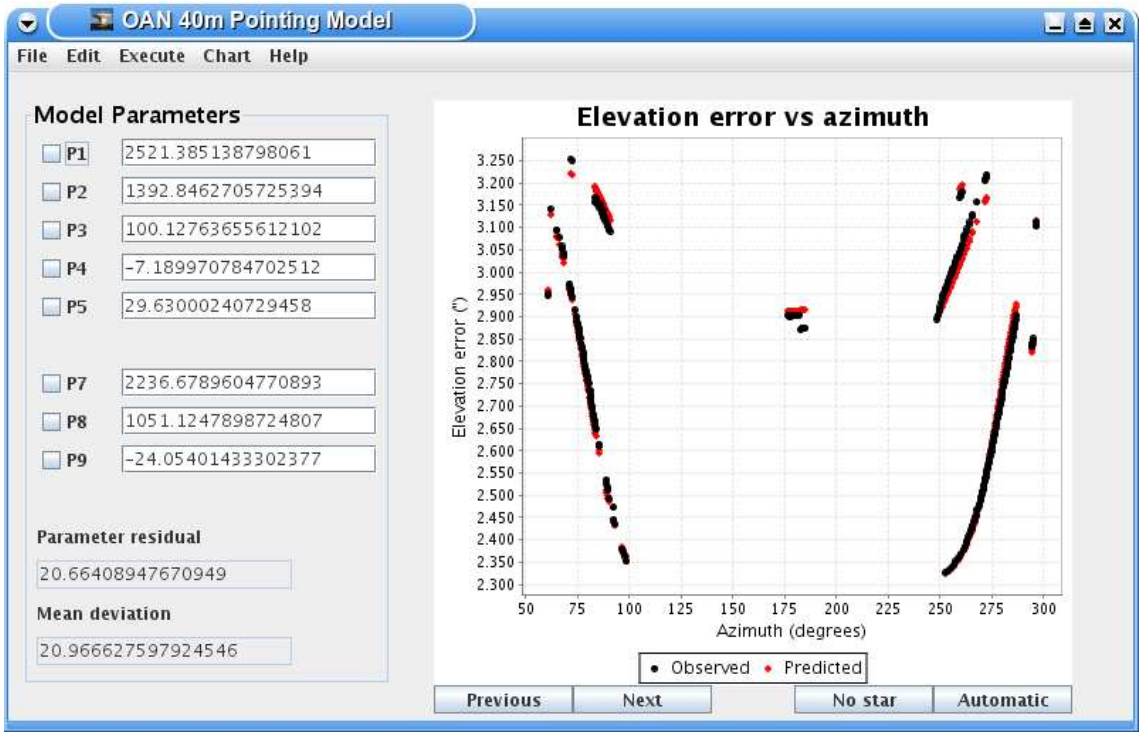


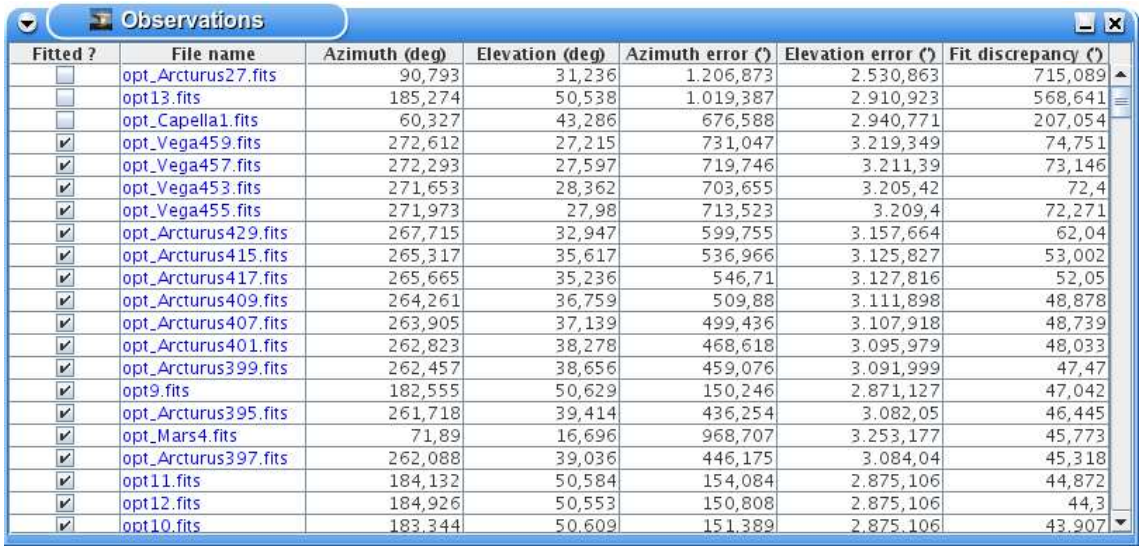
Figure 2: *Example of a fit showing a chart with the pointing errors in elevation.*

- Editing the observations.

  The Chart menu provides access to some useful charts that can be used to evaluate the quality of the fit. When selecting charts that show both the observed and the theorical pointing errors, the mean deviation of the fit will be also displayed. This deviation represents a mean (geometric) variance of each observation. The usual criteria for a good fit is that this deviation should be lower than *beam/4*, where *beam* is the beam of the telescope in arcseconds.

  Bad observations can be edited and removed with *Edit - Observations* (see figure 3). All calculations will be automatically updated. The observations can be sorted by any of the columns. It is specially useful to sort in decreasing order of the discrepancy between the predicted and the observed pointing errors. This allows to easily spot bad observations and remove them. A given .fits file can be visualized to check a specific observation by clicking on the name of the file.

  In the main window, there is one checkbox for each of the parameters of the model. They are useful to fix some parameters and study the dependence of the goodness of the fit when varying each of them. Simply check the box to fix some parameter, modify its value, and execute the model again. The fixed parameter will remain constant. It is also possible to modify a given parameter without neither checking the box nor executing the model. Simply change the value and select the chart to display again to update it. The current implementation does not allow to re-fit the parameters by least squares after a



Fitted ?	File name	Azimuth (deg)	Elevation (deg)	Azimuth error (')	Elevation error (')	Fit discrepancy (')
☐	opt_Arcturus27.fits	90,793	31,236	1.206,873	2.530,863	715,089
☐	opt13.fits	185,274	50,538	1.019,387	2.910,923	568,641
☐	opt_Capella1.fits	60,327	43,286	676,588	2.940,771	207,054
☑	opt_Vega459.fits	272,612	27,215	731,047	3.219,349	74,751
☑	opt_Vega457.fits	272,293	27,597	719,746	3.211,39	73,146
☑	opt_Vega453.fits	271,653	28,362	703,655	3.205,42	72,4
☑	opt_Vega455.fits	271,973	27,98	713,523	3.209,4	72,271
☑	opt_Arcturus429.fits	267,715	32,947	599,755	3.157,664	62,04
☑	opt_Arcturus415.fits	265,317	35,617	536,966	3.125,827	53,002
☑	opt_Arcturus417.fits	265,665	35,236	546,71	3.127,816	52,05
☑	opt_Arcturus409.fits	264,261	36,759	509,88	3.111,898	48,878
☑	opt_Arcturus407.fits	263,905	37,139	499,436	3.107,918	48,739
☑	opt_Arcturus401.fits	262,823	38,278	468,618	3.095,979	48,033
☑	opt_Arcturus399.fits	262,457	38,656	459,076	3.091,999	47,47
☑	opt9.fits	182,555	50,629	150,246	2.871,127	47,042
☑	opt_Arcturus395.fits	261,718	39,414	436,254	3.082,05	46,445
☑	opt_Mars4.fits	71,89	16,696	968,707	3.253,177	45,773
☑	opt_Arcturus397.fits	262,088	39,036	446,175	3.084,04	45,318
☑	opt11.fits	184,132	50,584	154,084	2.875,106	44,872
☑	opt12.fits	184,926	50,553	150,808	2.875,106	44,3
☑	opt10.fits	183,344	50,609	151,389	2.875,106	43,907

Figure 3: *The window to edit observations. In this example, three observations are identified as wrong and deleted from a total of about two hundred.*

given parameter is modified and fixed. This means that any change on the parameters will have no effect in the other parameters if they are fitted again.

- Saving Options.

  The current fit and the observations can be exported to a binary file with option *File - Save state*. This binary file can be imported later. Not only the observations, but also any other state of the program will be recovered exactly as when it was saved. This is specially useful to reduce the observations in different days or different computers, or to save a given set of bad observations that were removed from the fit with option *Edit - Observations*. It is recommended to save the current state of the program frequently.

  The current chart can be saved in .eps (Encapsulated Postscript), .jpg, .bmp, .gif, and .png picture formats. A special format .graphic is available to export the chart as an script to be later executed with the *GILDAS* package.

- Exporting Options.

  There are some export options in the *Edit* menu. The value of the parameters, as well as the current chart shown can be copied to the clipboard to paste them later in another program. The chart copy-paste process is known to fail in OpenOffice, while it works in any picture editor tested so far.

  An special export option is also available as *File - Export results*. The output is a *LATEX* file that can be compiled to .pdf and opened directly through this application. The .tex file will contain a table with the fitted parameters, all charts, a description of the model implementation, and possibly some other tables with a summary of the processed scans and possible warnings thrown during the reduction process. This useful option provides

a quick, detailed, and user-friendly output for the current results of the fit. The version of the program that creates an specific output report can be checked in the first line of the source code of the .tex file.

If an input file is not available, but a set of observations in the form of .fits files are locally available, the user should reduce these observations to create an input file. Here is when the command-line *enable fits* is required. The *File* menu will provide with a *Process fits* option for such task.

Next the basic steps of the .fits reduction process are summarized.

- Importing .fits files.

  Once you click on *File - Process fits* the program will prompt for an input directory to import the .fits files from.

  The .fits files contain optical CCD images taken through a refractor telescope that was located at the subreflector of the radiotelescope. As stated before, this obsolete feature was a provisional test tool and the program was designed to accept this particular setup.

- Reducing the .fits files.

  Once imported, the window will show the first (alphabetically sorted) .fits file (see figure 4). It is recommeded to enlarge the application window close to full screen before working with these files.

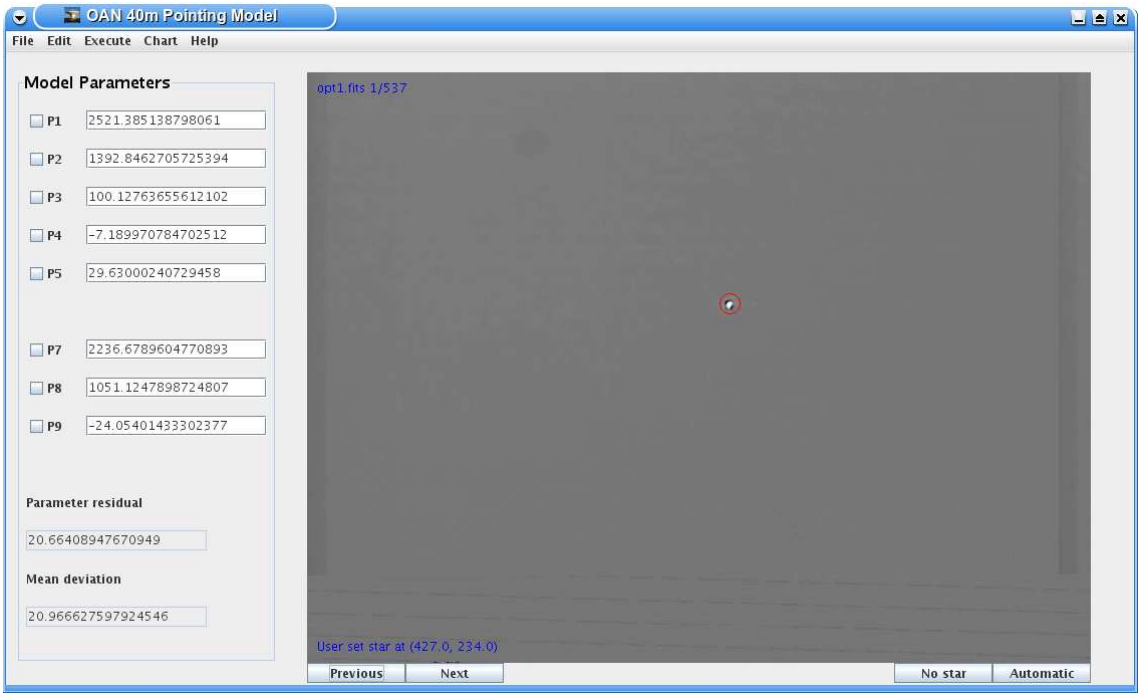


Figure 4: *The .fits reduction process showing and image where the user set the star position.*

In the upper - left corner you will see the name of the current image, and the number of this image from a given total number of them to reduce. In the bottom - left corner you will see a message with the position of the star detected in this image, or a red message indicating that no star was automatically detected.

The buttons provided below the image allows to change to the next or the previous image. A "No star" button is provided if a wrong star is detected automatically. The program contains a simple star detection algorithm that works well most of the times, but, if a star is not centered correcly you can click with the mouse on the star position to correct it. To move the star position just one pixel, click with the mouse at a position close to a corner of the image with the key *SHIFT*, *ALT*, or *CTRL* pressed. The right button of the mouse can also be pressed before the left click (at the left or right corners) to navigate faster through the input files. You can go to the 10th next image, or 100th next image using a double click. The reduction process can be canceled by clicking the mouse with two of the three modifiers (*ALT*, *CTRL*, *SHIFT*) pressed at the same time.

An automatic reduction button is provided. Once clicked, the rest of the reduction process will continue in a fully automatic way. It is not recommeded to use this feature unless you want to quickly test the results and you have good observations with a dark background and point-like stars, with nothing or little pixel saturation.

The process will continue until the next button is clicked in the last image. Then three files: the binary *pointing.bin*, the corresponding *ASCII pointing.dat* (with just 7 fields: azimuth, elevation (degrees), their pointing errors (arcseconds), x and y offsets of the star from the center (pixels), and the file name) and the *ASCII* file *pointingWarning.dat* will be created in the same folder of the .fits files. The first two files will be ready to be fitted.

- Re-processing the .fits files.

  The .fits reduction process can be started again at any time provided that the state of the program is saved. It is recommended to load the generated pointing.bin file before doing that to ensure that all the important data is loaded and saved. If the path of the .fits files changes (because a change in the user or the computer for example) the program will prompt for a new path with the same .fits files. If new .fits files are detected when re-processing, the program will ask if those files should be added to the current ones.

The application also contains a *Help* menu that includes most of the information provided in this document so far (figure 5).

2.3 Final remarks

The implementation is expected to be accurate for relatively low corrections up to one or two degrees. The pointing errors in azimuth and elevation are measured as celestial angles, not coordinate angles. As a consecuence, the equation of the pointing error in azimuth implemented in the code is multiplied by the cosine of the elevation in comparison with the one found in any of the cited references (see [3] for the last up-to-date equations). This means that you would not expect the azimuth correction to get higher and higher when pointing close to the cenit.

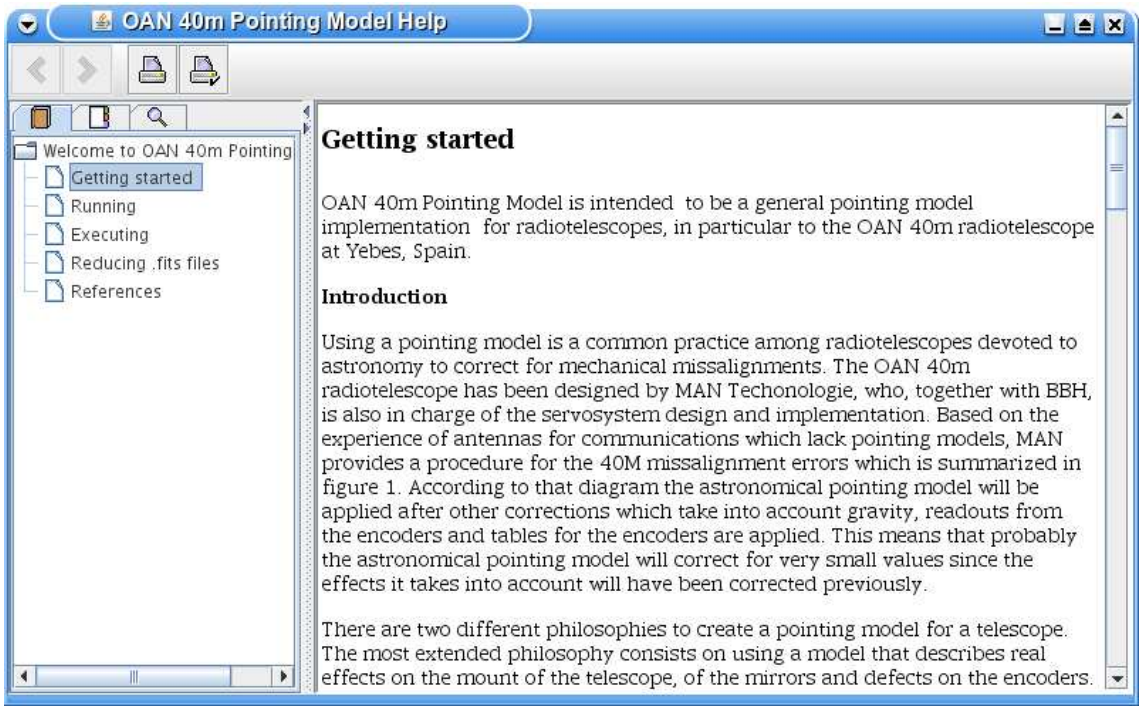


Figure 5: *The help window with the getting started page that introduces the user into the problem of the pointing models.*

By maintaining all the azimuth pointing errors in the same range, a better fit is obtained with *LAPACK*, since all the points are treated with a similar weight.

The pointing errors produced during the .fits reduction process are true angles on the celestial sphere. This means that the azimuth (and elevation) corrections are performed following a curve for points close to the cenit. This is known to improve the residuals for high elevation points (by about 1 arcsecond or 5% in the examples shown in the previous charts), and modifies the resulting P parameters by approximately 10%.

The example provided in the charts shows a final deviation of 20" in the fit, both in azimuth and elevation. However, this value should be taken with caution since some issues were found and corrected in the process of creating the .fits files and measuring the field of view and inclination angle of the CCD camera after this example was created. The expected error in the fit to new observations is much lower. In addition to that, some other improvements in the model are subject to discuss and implement, such a correction for differential refraction or an improvement in the accuracy of the pointed position and the date in the header of the .fits files.

We would like to stress on the importance of saving the state of the program everytime a given task is finished, and to load this file everytime the application is started. As stated before, some of features are in a provisional status and they could require further development or improvements as long as other tasks in progress for the 40m radiotelescope are completed.

3 Technical implementation

In this section we will describe the technical implementation of the model from a developer point of view. This part of the document assumes the reader is familiarized with the basics in the development of (Java) applications.

3.1 Design and development considerations

The *OAN 40m Pointing Model* application is implemented in pure Java 1.5, using the standard encoding *ISO-8859-1*. In its first version, it is organized in 7 packages containing 30 Java source files (Java classes) with about 15000 lines of code (2000 of them belonging to documentation). The application is written [6] and documented [7] according to the standards of quality established and recommended by *Sun Microsystems*.

The development of the first version were carried on between November and December, 2007. All the code was written during one month, mainly because most of the Java classes in the code are based on previous work by T. Alonso. Only 8 Java classes in 3 packages are in fact new work, among of them the .fits methods to interoperate with the .fits external library, the *Graphical User Interface* (GUI) as an user front-end to the application, and the mathematical implementation of the pointing model using the Java version of *LAPACK*.

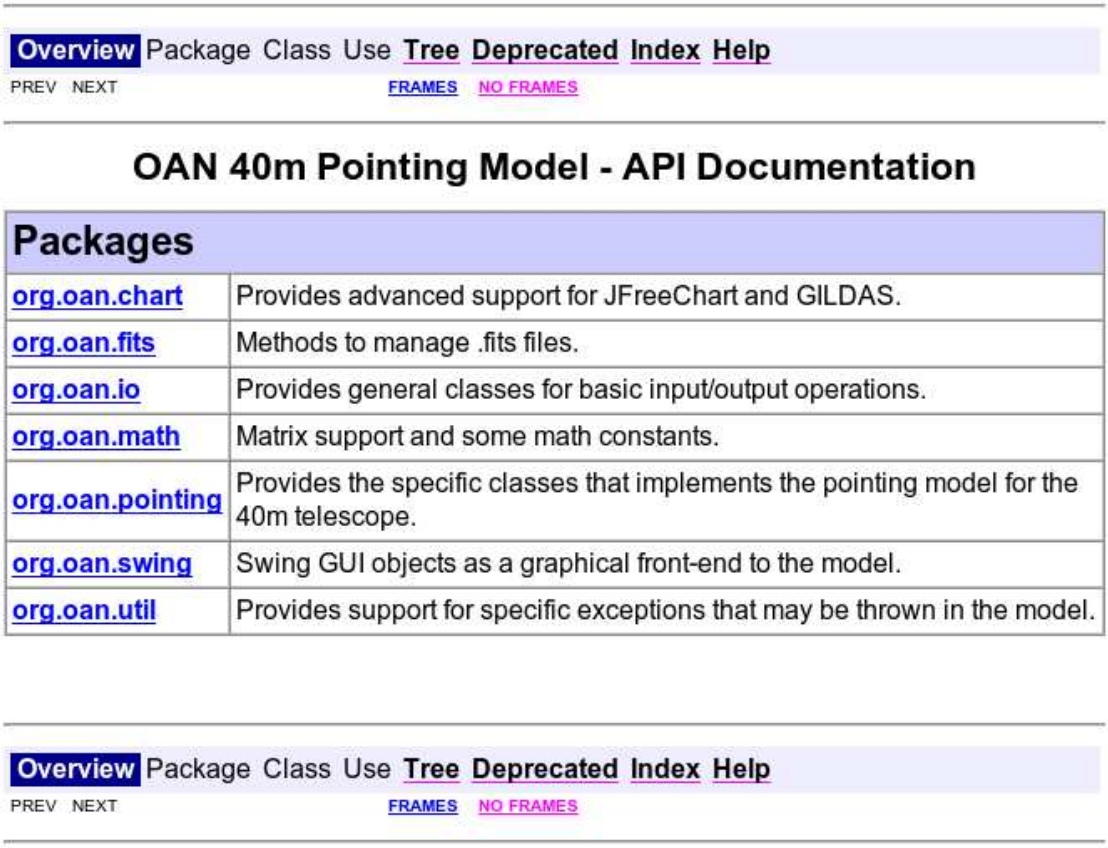
The program was designed according to a previous implementation in *C++* by P. de Vicente. Mathematically speaking, both implementations are almost identical. In terms of the GUI, both implementations are quite similar. In this Java version further development was consider to give more flexibility to the user in some common tasks, like reducing the .fits files, saving and reading the work in progress, or exporting the results.

This new implementation in Java solves some issues of the previous one. The availability and compatibility of some libraries in *C++* with the ones actually in use in the new versions of *KDE* was a problem. Also, this implementation can be executed either in Unix/Linux, Windows, Mac, or even in a mobile phone with some minor modifications. In the last few years Java have grown in such way that is now the main programming language in consideration for astronomical projects. In this way, it will be possible to integrate this piece of software into the contributions to the *ALMA Common Software* (ACS), which is being developed in Java.

The application will be distributed under the terms of the LGPL license. This is in agreement with the distribution policy of the dependence libraries and provides rights to third party users to freely distribute the source code in its original or modified versions, even in the scope of possible merchandising, provided that the conditions in the LGPL license agreement remains unviolated.

3.2 The Javadoc

In the Javadoc it is possible to find the full *Application Program Interface* (API). This includes all the public methods and fields used within the application, including the input and output parameters for each method with a description of the purpose of each of them. The Java documentation is automatically generated by the *javadoc* tool already integrated in the JDK using



Overview Package Class Use Tree Deprecated Index Help

PREV NEXT FRAMES NO FRAMES

OAN 40m Pointing Model - API Documentation

Packages	
org.oan.chart	Provides advanced support for JFreeChart and GILDAS.
org.oan.fits	Methods to manage .fits files.
org.oan.io	Provides general classes for basic input/output operations.
org.oan.math	Matrix support and some math constants.
org.oan.pointing	Provides the specific classes that implements the pointing model for the 40m telescope.
org.oan.swing	Swing GUI objects as a graphical front-end to the model.
org.oan.util	Provides support for specific exceptions that may be thrown in the model.

Overview Package Class Use Tree Deprecated Index Help

PREV NEXT FRAMES NO FRAMES

Figure 6: *The main Javadoc window showing the summary of the packages in the application.*

the comments that the developer introduces in the code following certain rules [7]. This information is intended to be used as a reference source for developers (already integrated in a given IDE when a project is configured) and is subject to change in possible future releases. A considerable amount of private methods and fields not visible in the documentation are also present, but it is not necessary to take care about them, even from a developer point of view, unless an important change is made to the code. Any further modificacion of the code should continue the standards of quality of this implementation to ensure that this Javadoc can continue to be generated and used as a reference.

3.3 Packages description

The application is organized in packages (directories) that are responsible of different blocks in the execution of the program. All modern applications are organized in a similar way, separating blocks like input/output operations, the math operations, the charts or graphic generation, the error reporting and catching, or the *Graphical User Interface*. This modular design is the best way to maintain a high level of independence between different blocks in order to improve or correct errors, perform a further development by the same or any other team of developers, or for re-utilization of the resources in different projects. There are 7 packages and 30 Java classes in this application which are analized here shortly. Additional information can be found in the Javadoc documentation, available after compiling the source code, or through the reference pages in the application help system.

- *org.oan.chart*.

  This package contains the classes developed to create charts using the *JFreeChart* package [8], to export charts as a script for *GILDAS*, and to read and write pictures in several graphic formats.

  - *ChartElement*. This class allows to create Java objects that contains the properties of a *JFreeChart* chart like the data series, the chart type, the labels for the axes, and so on.
  - *ChartSeriesElement*. This class provides secondary objects for the *ChartElement* class that represents the data series to be displayed, including the x and y datasets, the shape and size of the marker for the points, possible line strokes or regressions, and so on.
  - *CreateChart*. This is the core of the package. The *ChartElement* object, hold in a developer friendly way, is translated to a *JFreeChart* object, ready to be displayed, using the constructor of this class. This class also provides output to .eps and .graphic formats, among others. *JFreeChart* supports x-y charts, pie charts and bar charts, but here a simplified version of the original *CreateChart* class has been implemented, supporting only x-y charts.
  - *DataSet*. This class provides methods to transform the x and y datasets between different formats, to extract subdatasets, to obtain the maximum and minimum values, or to sort them in different ways.

  - *Draw*. A class to draw pictures, lines, or text with easy to create some panels. It was used to create the About box.
  - *Picture*. This class reads and writes picture in a wide variety of formats.

- *org.oan.fits*.

  This package provides the tools to manage .fits files.

  - *CCDElement*. This class holds the properties of the CCD camera and provides tools to transform the horizontal offsets into sky angles.
  - *FitsHeaderElement*. This object holds the header of a .fits files and provides methods to access the data that contains.

    The current format of the .fits header includes information about the size of the image, the color resolution, the telescope and date, the source and the position of the antenna with possible pointing corrections applied. Since the field of view of the CCD is about 40 arcminutes, which could be lower than the pointing errors, in some cases it is necessary to perform a first correction of the pointing in order to be able to catch the star inside the field of the CCD. An example table with the current format of the header is shown in table 1.
  - *FitsIO*. The core of the package. It reads a .fits file providing both the Picture and the *FitsHeaderElement*.
  - *FitsProgram*. A class that sorts a set of .fits files and contains a method to automatically calculate the position of a star in a .fits image. It was used as a program at the very beginning to automatically reduce the .fits files, but soon it was clear that a fully automatic reduction was not acceptable.

- *org.oan.io*.

  This package provides input/output support for several tasks in the program.

  - *ApplicationLauncher*. Provides general methods to execute programs with independence on the operating system the user is running, and, in the case of Linux, with independence of the desktop (KDE, GNOME) in execution.
  - *IOConstant*. General constants for input/output like the path separator, which depends on the operating system, or the separator used in the *ASCII* files.
  - *LATEXReport*. A special class to write documents to Latex with easy.
  - *ReadFile*. A class to read files, access the different fields or columns, and get a list of files or directories in a given path.
  - *Serialization*. A special class to write and read binary files that can virtually save any abstract object or data in the Java language. This feature of the Java language is used to save and recover the state of the application at any time.
  - *SystemClipboard*. Methods to access the system clipboard to copy or get contents like text or images.

Table 1: Example header of a .fits file.

HEADER ENTRY	VALUE	COMMENT
SIMPLE	T	
BITPIX	8	
NAXIS	2	
NAXIS1	760	
NAXIS2	570	
DATAMIN	0	
DATAMAX	255	
TELESCOP	ARIES21	Telescope Name
ORIGIN	OAN	Organisation or Institution
CREATOR	Manual pyfits	Software (including version)
TIME	2007_11_13 15:43:4	Time
AZIM	253.984	Azimuth
ELEV	84.462	Elevation
SOURCE	Vega	Source
RA	18.616	RA
DEC	38.784	DEC
P1	0.000	Az Encoder Point
P7	-2700.000	El Encoder Point
AZCORR	-0.278	az correction
ELCORR	0.000	el correction
HISTORY	Created by ImageMagick.	

  - *WriteFile*. Provides a way to write simple text files.

- *org.oan.math*. A package the provides useful math operations.
  - *Constant*. Holds math constants like conversion from degrees to radians and viceversa, among others. The same math constants are used everywhere in the code.
  - *Functions*. Provides math functions like degrees/radians normalization, or formatting the coordinates.
  - *Matrix*. Provides support for matrices, including operations like solving linear systems.

- *org.oan.pointing*. This class solves the pointing model itself.
  - *ModelElement*. An object that holds the parameters of the model and provides methods to obtain the predicted pointing error in azimuth and elevation.
  - *PointingModel*. The class that solves the pointing model using *JLAPACK* [9].
  - *ScanElement*. An object that holds a given scan or observation. This includes the azimuth, elevation, x and y offsets, the file name, the *CCDElement* with the properties of the CCD camera, and the *FitsHeaderElement* with the header of the .fits file.

- *org.oan.swing*. This package provides the swing (graphical) implementation of the application.
  - *DoubleBuffer*. Provides double buffer support, that allows to draw a chart for example without seeing how every line is drawn.
  - *GraphLayout*. Provides a layout to arrange different components in a panel with double buffering support.
  - *GUI*. The core of the application is the *Graphical User Interface* that creates the program window and responds to the user operation.
  - *LaunchApplication*. The class that is called in the application launch. It is merely used to create a splash window and to call to the GUI.
  - *SplashWindow*. The class to create a very simple splash window, just an image which is displayed while the application in starting and allocating it resources.

- *org.oan.util*. A class to catch and respond to possible errors during the execution, and to hold the program version.
  - *PointingModelException*. The class which is called whenever a given predicted error occurs. For example, an error occurs when the user reads a binary file which is first treated as *ASCII*, but no error is thrown since this is catched and treated as 'predicted'.
  - *Version*. Holds the name of the application, the version, and (hopefully in the future) a log of changes.

3.4 The HelpSet

The helpset are the contents displayed in the *Help* menu. These contents are mainly web pages with some special support files organized according to the *JavaHelp 2.0* specification [10]. The detailed analyze of *JavaHelp* is beyond the scope of this document, but it must be noted that any further modification or development of the help menu should take use of the help authoring tool *JHelpDev* [11].

3.5 Dependencies

The application needs 13 .jar files (external libraries) to be executed. All these libraries are original and unmodified work from their respective authors distributed under LGPL or compatible licenses. They are described here shortly.

- *JFreeChart*. The *JFreeChart* library [8] dependencies includes the *jfreechart-1.0.3.jar* and the *jcommon-1.0.6.jar*. Updated or older versions of these packages could work without problems, but it cannot be guarranteed.
- *JLAPACK*. The *JLapack* library [9] includes the files *lapack.jar*, *xerbla.jar*, *f2jutil.jar*, and *blas.jar*.
- The *JavaHelp*. The .jar file *jhall.jar* [10] is necessary to show the help menu.
- .fits files. The library fits.jar [12] is used in this implementation.
- *FreeHEP*. The *FreeHEP* library [13] provides output to .eps (Postscript) format in Java. It includes *freehep-graphics2d-2.0.jar, freehep-graphicsio-2.0.jar, freehelp-graphicsio-ps-2.0.jar, freehep-io-2.0.1.jar*, and *freehep-util-2.0.1.jar*.

3.6 Application Flow

The program flow is shown in figure 7. There are three main arms in the picture representing the three main activities of the program: the .fits reduction process, the process of reading an input file and fitting the observations, and the process of exporting the results. The output from the .fits reduction process can be used as an input for the pointing model solving process. The same happends with the state of the program saved to a binary file. In this case, and only in this case, it is possible to re-process the .fits files if the state was saved after the reduction process finished and the file was not overwritten later. In the process of solving the pointing model the observations can be edited to select which points are to be fitted.

References

[1] P. de Vicente, *Algorithms to be implemented in the Antenna Control Unit*, IT-OAN 2003-7.

[2] P. de Vicente, A. Barcia, *Deconstructing a pointing model for the 40m OAN radiotelescope*, IT-OAN 2007-26.

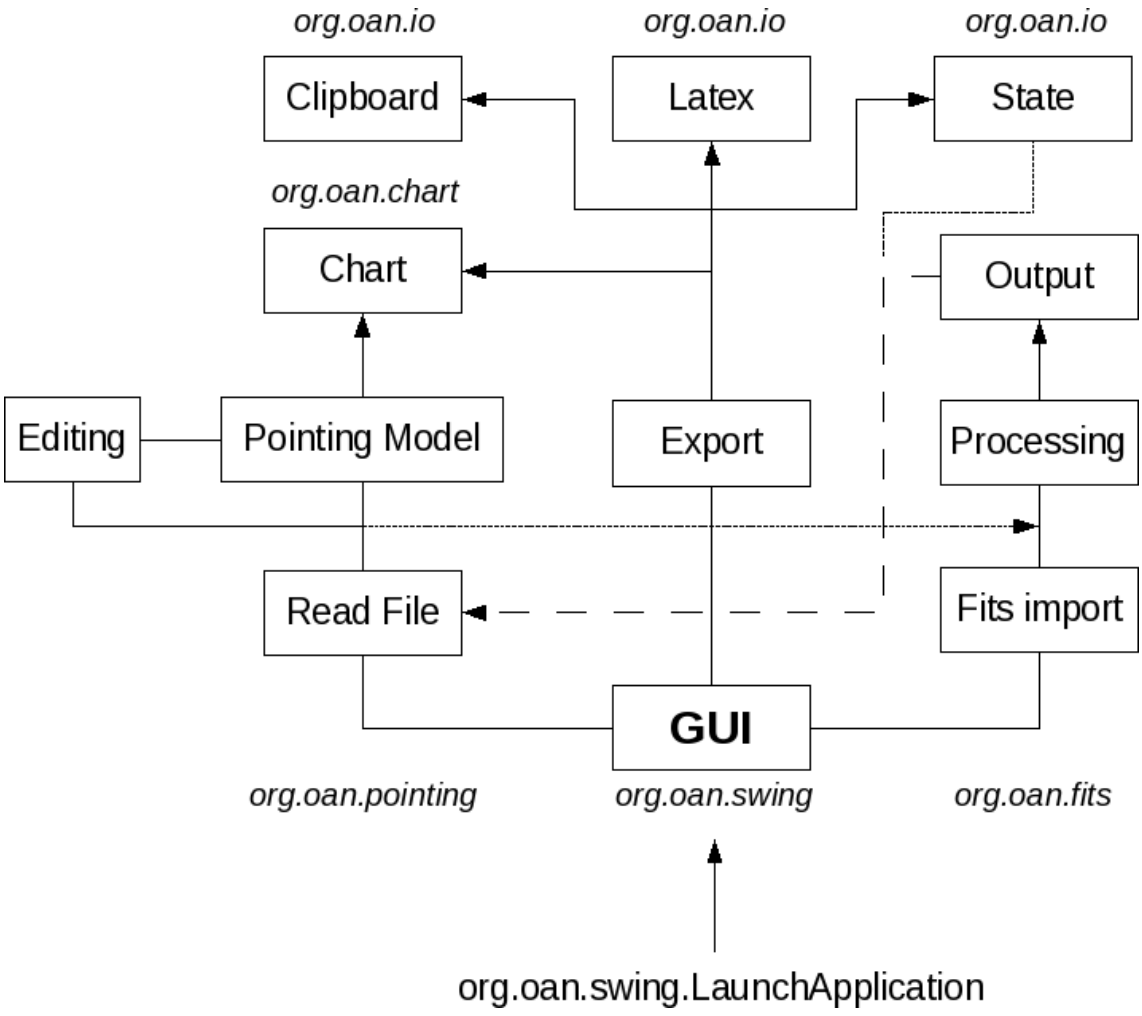


Figure 7: *Program flow.*

[3] P. de Vicente, *Checking the pointing corrections implementation for the 40m radiotelescope*, IT-OAN 2008-7.

[4] The Java JRE/JDK 1.5 can be downloaded at java.sun.com.

[5] The *Eclipse SDK* project, downloadable at www.eclipse.org.

[6] Article on code conventions in Java, http://java.sun.com/docs/codeconv/.

[7] Article on how to write comments for the *javadoc* tool, available at http://java.sun.com/j2se/javadoc/writingdoccomments/.

[8] *JFreeChart* can be downloaded at www.jfree.org/jfreechart/.

[9] *JLapack* can be downloaded at http://www.netlib.org/java/f2j/.

[10] The *JavaHelp* reference page is available at http://java.sun.com/products/javahelp/.

[11] *JHelpDev* can be downloaded at http://jhelpdev.sourceforge.net/.

[12] Reference page for fits utilities at http://fits.gsfc.nasa.gov/. Package nom.tam.fits can be downloaded at http://heasarc.gsfc.nasa.gov/docs/heasarc/fits/java/v0.9/.

[13] *FreeHEP* project page available at http://java.freehep.org/.