ISOCAM Interactive Analysis User's Manual
Version 4.0

Document Reference Number SAI/96-5226/Dc

Edited by:
Matt Delaney
Stockholm University

with contributions from
ISOCAM Interactive Analysis Team
ESA, CEA, IAS, IPAC & RAL

April 7, 2000

Acknowledgements

Contributions to the *CIA User's Manual* were made by all members of the ISOCAM Interactive Analysis Team and in particular members of the ISOCAM Instrument Dedicated Team, VilSpa and the Service d' Astrophysique, Saclay. Some sections of this document are extracted from internal documentation – acknowledgments can be found in footnotes to these sections.

Alain Abergel (IAS)
Babar Ali (IPAC/UR)
Bruno Altieri (ESA)
Jean-Louis Auguéres (CEA)
Herve Aussel (CEA)
Jean-Philippe Bernard (IAS)
Andrea Biviano (OAT/ESA)
Joris Blommaert (ESA)
Olivier Boulade (CEA)
Francois Boulanger (IAS)
Catherine Cesarsky (ESO, ISOCAM PI)
Diego Cesarsky (IAS)
Pierre Chanial (CEA)
Vassilis Charmandaris (CEA)
Ranga-Ram Chary (UCLA)
Arnaud Claret (CEA)
Alain Coulais (IAS)
Matt Delaney (ESA/UCD/SO)
Christophe Delattre (CEA)
Thomas Deschamps (CEA)
Francois-Xavier Désert (IAS)
Pierre Didelon (CEA)
David Elbaz (CEA)
Yaeï Fuchs (CEA)
Pascal Gallais (ESA/CEA)
Ken Ganga (IPAC)
René Gastaud (CEA)
Steve Guest (ESA/RAL)
George Helou (IPAC)
Martin Heemskerk (Univ. of Amsterdam)
Mih-seh Kong (IPAC)
Francois Lacombe (DESPA)
David Landriu (CEA)
Olivier Laurent (CEA)
Patricia Le Coupanec (DESPA)
Jing Li (IPAC)
Leo Metcalfe (ESA, responsible for flight calibration)
Marc-Antoine Miville-Deschǹes (IAS)
Koryo Okumura (ESA/IAS)
Stephan Ott (ESA, responsible for IA system)
Michel Pérault (IAS, responsible for ground calibration)
Sibylle Peschke (ESA)
Andy Pollock (ESA)
Daniel Rouan (DESPA)
Pilar Roman (ESA)
Michael Rupen (NRAO)
Jaqui Sam Lone (ESA/CEA)

Marc Sauvage (CEA, responsible for the French ISO Data Centre)
Ralf Siebenmorgen (ESA)
Jean-Luc Starck (CEA)
Richard Tuffs (MPIK)
Dan Tran (CEA)
Dave Van Buren (IPAC)
Laurent Vigroux (CEA, responsible for IA algorithms)
Florence Vivares (IAS)
Herve Wozniak (Observatoire de Marseille)

Contents

List of Figures

List of Tables

Chapter 1

About the CIA User's Manual

1.1 Organization of the CIA User's Manual

The *CIA User's Manual* is split into four parts – Part I: *Quick Start Guide*, Part II: *CIA Basic Guide*, Part III: *Data Management* and Part IV: *Advanced Use of CIA*. To further understand this organization read the introduction that can be found at the beginning of each of these parts. To get maximum benefit from the *CIA User's Manual*, novice users of CIA may find it helpful to read each chapter sequentially, beginning obviously with the *CIA Basic Guide*, while more advanced users may wish to jump in at the sections of interest to them. In any case it is hoped that adequate cross-references exist to make the *CIA User's Manual* useful wherever you choose to begin reading.

1.2 What you need to begin

Check that you have the following before continuing:

1. Some knowledge of **IDL**[1].
2. **Data products** – Either retrieved from ISO's Post Mission Archive (IDA) at Villafranca, a CD-ROM issued to you by ESA (see Chapter C), or an archive at your site.

Additional documentation which you may need as you progress with CIA data analysis is described here.

1. **CIA documentation** – CIA documentation is distributed with CIA. Documentation is generally in postscript format and can be found in the directory *doc* just below the top of the CIA distribution.

   This documentation includes:

   Technical reports These are referred to in the *CIA User's Manual* where relevant and are also listed in Appendix I.

   User guides These are guides to individual routines and written by the authors of the routines. Again, they are referred to throughout the *CIA User's Manual* and are also listed in the bibliography.

[1] IDL is a registered trademark of Research Systems Inc.

Note that the documentation is only as up-to-date as the installed version of CIA. The ISO web site is the best source of current documentation:

`http://www.iso.vilspa.esa.es`

The ISO Explanatory Library contains full documentation on ISO and the ISO instruments:

`http://www.iso.vilspa.esa.es/users/expl_lib/expl_lib.html`

The CAM Instrument Page contains specific information on CAM:

`http://www.iso.vilspa.esa.es/users/expl_lib/CAM_top.html`

2. **ISO documents** – Throughout the *CIA User's Manual*, references are made to the documents below. They are retrievable/browsable from the ISO web site and IDA (address above) or found on an ISO CD-ROM (see Chapter C) along with the data products.

   The ISO handbooks, especially the

   - *ISOCAM Handbook*
   - *ISO Satellite Handbook*

   The Observer's Manuals.

   - *ISOCAM Observer's Manual*
   - *ISO Observer's Manual*

   The IDPD.

   - *ISO Data Product Document*

1.3 Reporting comments on the *CIA User's Manual*

Suggestions for improving this manual may be submitted to the address in Section H.2.

Chapter 2

About CIA

2.1 History and Purpose of CIA

Once upon a time there was a prototype system that went by the name of ICE, or ISOCAM Calibration Environment. Soon thereafter CIA, CAM Interactive Analysis, came into existence, inheriting some modules from ICE. CIA was to be an evolution from

- the *minimum system*, completed 15th April '95
- over the *operational system* used during the operations of ISO
- to the *astronomical data-processing system* used during the post-operational phase of ISO

It may be instructive to list the functional requirements of the operational system (which was to run on VMS):

- calibrate ISOCAM
- monitor the health of ISOCAM
- perform any sort of investigation requested for problem diagnostics
- assess the quality of ISOCAM data products
- debug, validate and refine the pipeline
- provide a test-bed for algorithmic developments

It should be clear that given the driving forces behind the design, the evolution into a different kind of beast altogether – an astronomical data-processing system – would not be without its difficulties. Some users, unaware of the operational *raison d'être*, are puzzled by certain features of the system.

CIA has now acquired a wealth of astronomical data processing routines and user-friendly widget-based programs. Though the end of ISO's operational lifetime has passed, CIA will continue to develop as an astronomical data analysis tool well into the future.

2.2 System requirements

The CIA system requires the following resources:

- IDL must be installed on your system. Versions earlier than IDL 5.0 are *not* supported by CIA 4.0.
- CIA needs disk space of 375 MB. Additional disk space of 450 MB is required for the optional theoretical PSFs (available from the CIA server).
- Users should have access to 250 MB of swap space, bigger data-sets might require up to 1GB of swap space.
- CIA is supported to run on a Sun Sparc Solaris 2.5 (or later) and VMS Alpha 6.2 (or later), and is known to run on DEC ALPHA OSF1 (Digital Unix), HP/UX (Hewlett-Packard Unix) and x86 Debian Linux 2.0. Users are invited to build the CIA executables and modify IDL coded modules so as to enable CIA to run on alternative platforms.

2.3 Getting started

Before proceeding make sure that CIA is properly installed on your system and that your system meets the necessary requirments for running CIA (see Section 2.2).

2.3.1 How to Start CIA

This section is by its nature dependent on system configuration which can vary. In case of problems consult your system manager, though Section 2.3.4 describes CIA customization and may provide pertinent information on.

On VMS:

```
$ idl
```

If this fails try typing `idl5`.

On Unix:

```
> cia [version]
```

Here there is an optional parameter 'version' in the form `MMMYY` (*eg* `APR00`). The default will usually be the latest version.

2.3.2 Using the Online Help: cia_help

The CIA help system uses the old style IDL help. This is invoked by entering **widget_olh** on the CIA command line. The alias **ciainfo** will also invoke **widget_olh**, provided of course that the alias list is compiled...

```
CIA> .r alias
```

The organisation of the headers within **ciainfo** is slightly different on VMS and UNIX platforms. In the VMS version, the routines are organised into two groups: *CIA USER* and *CIA HELP*. The former contains just those routines of most interest to the user, while the latter contains the full suite of routines (see Figure 2.2. In UNIX you have also *CIA USER* and *CIA PRGM*, but additionally the routines are split into several groups (see Figure 2.1). The titles of each should be self-explanatory.

In addition to **ciainfo** there is a dedicated help, invoked by typing **cia_help** on the command line (see Figure 2.3). (Note that this is an IDL widget based program, so use of the command line is suspended under VMS during its operation.) In the **cia_help** opening widget the CIA routines are organised into eight *supergroups*. These are listed in the top half of the widget and accompanied by a brief description in the bottom half of the widget. To get deeper into the help choose a *supergroup* you are interested in and click on the appropriate button. A *supergroup widget* will then appear and present you with a list of *groups* from the supergroup you chose. Again, choose the *group* you want and a widget containing a list of routines will appear.

Each **cia_help** widget contains a set of buttons. These provide the following functions:

- *Done* button of any of the widgets will close that widget and its dependents.
- *Find* button will invoke the find function, which returns a list of routines from the current *supergroup* or *group* widget that contain the supplied string.
- *Quit* button quits **cia_help**.
- *Help* button invokes a description of the current widget.

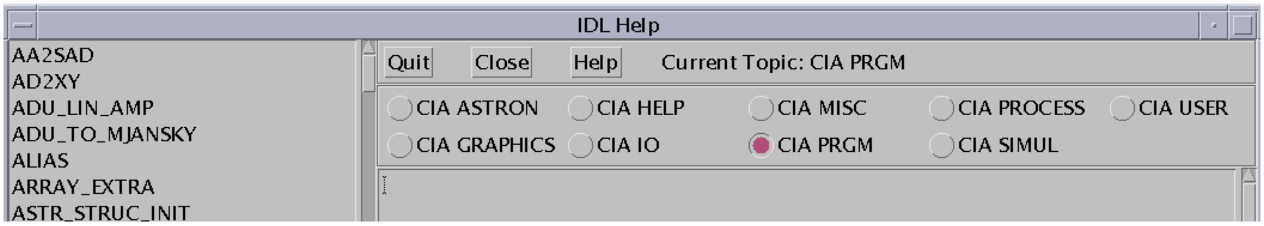

Figure 2.1: typical **ciainfo** display for UNIX.

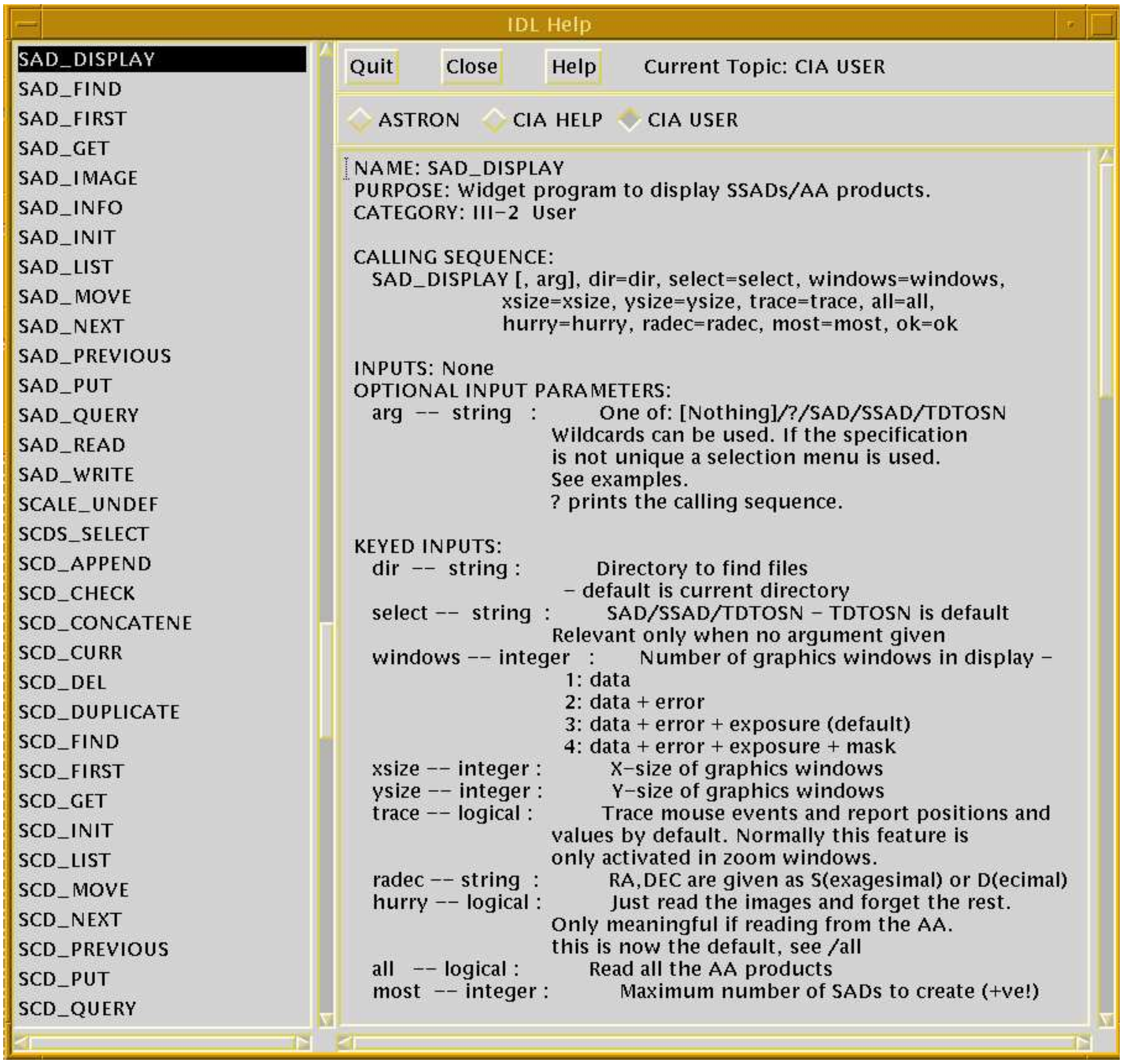

Figure 2.2: typical **ciainfo** display for VMS.

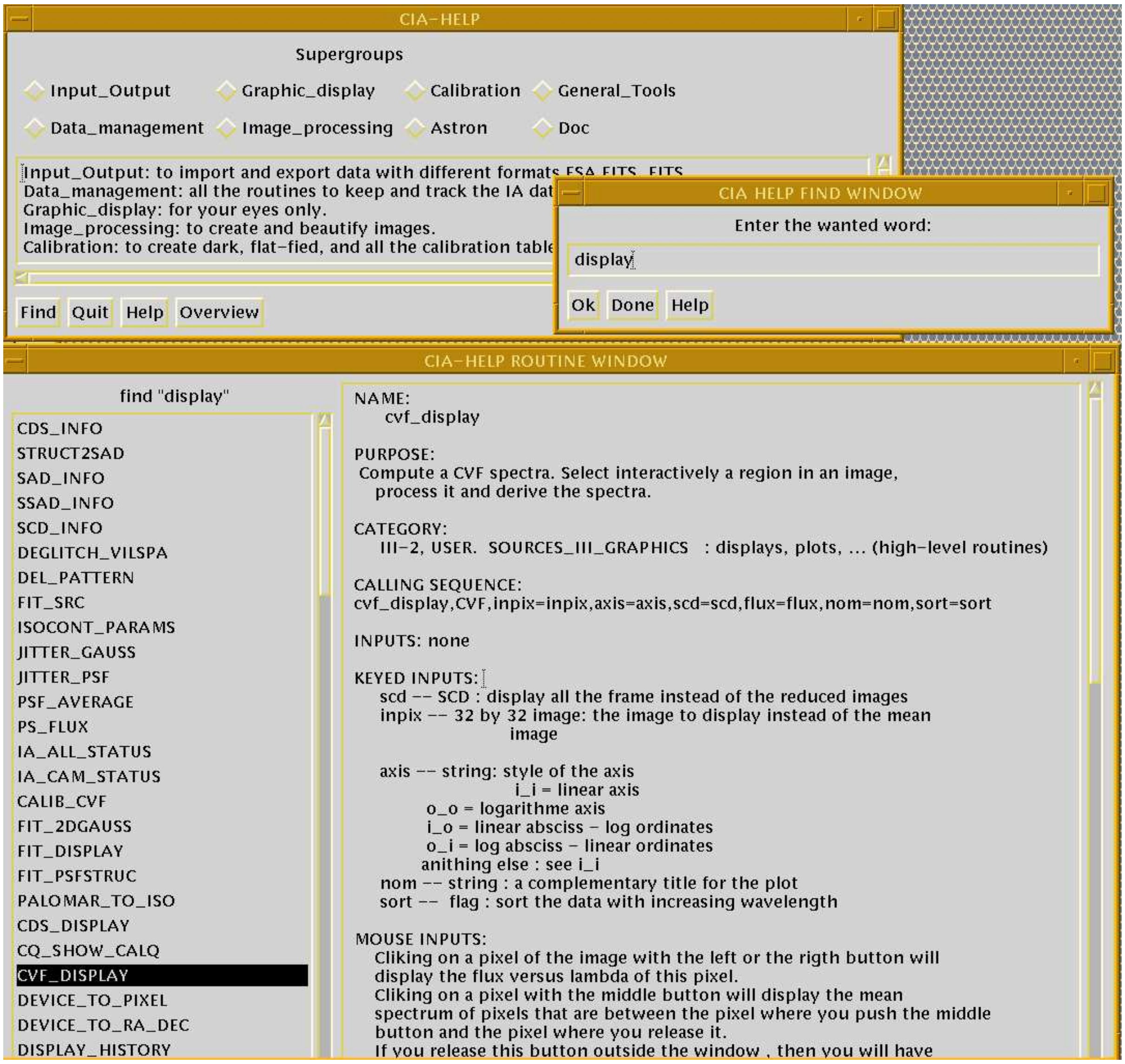

Figure 2.3: typical **cia_help** display.

2.3.3 So you want to see some ISOCAM images ASAP?

Part I is a quick start guide to analysing CAM data products. It should get you up and running. If want to have a look at your Auto-Analysis data products (data which has been delivered after standard automatic processing) then see Section 10.2.

2.3.4 Customizing your CIA session

First-time users and anyone not interested should skip this section. The default configuration should be adequate for the novice user.

You can customize the way CIA is set up with your own CIA startup file `user_init.pro`. This file should follow the rules for IDL batch files. You should copy the default startup file, delivered with CIA, modify it yourself, and then instruct CIA to use it. Currently, the Unix version of this file has the following content:

```
;
; startup file for running CIA ith IDL
; can be modified at willing but must call the CIA_start file
;

@$cia_vers/unix/cia_start

; DO NOT CHANGE LINES ABOVE

; if you really need logfiles, nodify this line
; SET_LOGFILE, /noerror, /nosession, /nomaster, newsite="mysite"

; define whether you would like to have informational messages printed
 !quiet = 1

; define how many many lines you would like to have in the command buffer
; !edit_input = 100

; define the display order
; !order = 0

; CIA's mask (easy or complex)
; !mask = 0

; change CIA's warning level
; !ciaerr.xwarn = 0

; add the contributions to the path for VMS
if (!version.os_family eq 'vms') then $
   !path = !path + ","+ EXPAND_PATH("+CIA_DIR:[CONTRIB]")
if (!version.os_family eq 'vms') then print, ' '
if (!version.os_family eq 'vms') then $
   print, 'Contrib routines have been added.'
```

```
; decide whether you like to have the user's or the full help for cia_help
;if (!version.os_family eq 'unix') then $
;    setenv, 'cia_help_dir=$cia_vers/ia_help/help_usr' else $
;    setlog, 'cia_help_dir', 'CIA_DIR:[HELP_USR]'
;if (!version.os_family eq 'unix') then $
;    setenv, 'cia_help_dir=$cia_vers/ia_help/help_prg' else $
;    setlog, 'cia_help_dir', 'CIA_DIR:[HELP_PRG]'

; decide if you want to run the alias file
; .run alias
```

Generally, the things you may want to configure are

- The default CIA version (if more than one version is installed). This only applies to VMS CIA. For same in UNIX see Section 2.3.1.
- CIA's logfile behaviour. By default, no logfiles are produced.
- !QUIET system variable which defines how informational messages are handled
- !EDIT_INPUT system variable which enables keyboard line editing
- !ORDER system variable. Since CIA 2.0 !ORDER defaults to 0 (the IDL default), though in CIA 1.0 it defaults to 1. If you want later versions of CIA to display images in the same manner as CIA 1.0 then change !ORDER to 1.
- MASK configuration (see Section 15.2.2.18). The MASK can be set to be *simple* or *complex* by setting the system variable !MASK to 0 or 1 respectively. !MASK defaults to 0.
- Very advanced users can configure CIA logging and debugging verbosity with the system variable !CIAERR.

```
CIA> help, !ciaerr, /str
** Structure <7d9788>, 5 tags, length=12, refs=2:
   SCREEN          BYTE         2
   LOG             BYTE         3
   XWARN           BYTE         0
   DEBUG           BYTE         0
   PRINT           STRING    'PRINT'
```

PRINT Defines the routine used by the CIA's **cia_print** to print information. Can be used to redirect output of **cia_print**.

DEBUG Flag indicating debugging level. Set to 0 for silent output and 1 for verbose output. For example,

```
CIA !ciaerr.debug=1
```

will cause CIA to output debugging information.

LOG Log file reporting verbosity. Ranging from 1 to 3 for increasing verbosity.

SCREEN Screen reporting verbosity. Ranging from 1 to 3 for increasing verbosity.

XWARN CIA uses IDL's keyword inheritance. Setting this flag will alert you if a supplied keyword parameter is not known by the called routine. If a warning appears, you have either deliberately specified a keyword used by a low level routine or mistyped the keyword (in which case it will be ignored). For example:

```
CIA> !ciaerr.xwarn=1

CIA> corr_dark, pds, /goodbye, /hello
CORR_DARK: Undefined keywords: /GOODBYE/HELLO
Dark correction with model and no scaling
```

- inclusion of contributed routines to CIA's path. (Only for VMS, under Unix these routines are included by default).
- setting the level of the on-line help (see section 2.3.2). Either the complete help for all routines (Programmer's Help) or only the help for high-level routines (User's Help) is displayed.
- defining whether the alias file is run.
- inclusion of your routines in IDL's search path.
- Some users have reported problems running CIA on 24 bit X-Windows displays. One suggested solution is to force IDL to use an 8 bit PseudoColor. To do this, add the following line to your CIA startup file:

  ```
  device, pseudo_color=8
  ```

  For more on this topic look for information on the procedure **DEVICE** in the IDL online help.

  Another solution to this problem, that does not involve the CIA startup file, is suggested in Section 2.4.

  As already stated, all of the above can be configured in a CIA user startup file. The following sections describe how you can create your own CIA startup file on both the UNIX and VMS platforms.

2.3.4.1 VMS

Normally your system administrator should have set up the global symbol `CIA_ENV` – this should point to the generic command-file `CIA_VMS_ENV.COM`. It should be set to something like

```
$ sh sym CIA_ENV
CIA_ENV = "@SAPI01$DKC200:[CIA]CIA_VMS_ENV.COM"
```

This command-file sets up the environment for a CIA session. It must be executed *before* CIA is started. For convenience you can include a line in your `login.com` to this automatically upon login.

Before any modifications to your CIA set-up can be made you must do the following:

1. Make a personal copy of the generic CIA startup script `user_init.pro` – this should be found in the CIA installation directory. In your `LOGIN.COM` file change the logical `idl_startup` to point to your personal `user_init.pro`. Them make sure that your definition of the logical `idl_startup` is used. For example, include the following lines in your login.com:

   ```
   $ CIA_ENV
   $ DEFINE IDL_STARTUP SYS$LOGIN:USER_INIT.PRO
   ```

If you want to modify CIA's logging directory, then you also have to create your personal `CIA_VMS_ENV.COM`, which has to be executed instead of the general one.

To specfiy the directory to which CIA log files are to written, in your `CIA_VMS_ENV.COM` change the logical `logfile_dir`. For example:

```
$ DEFINE LOGFILE_DIR SAPI01$DKA200:[DELANEY.CIA_LOGFILES]
```

However, for most users switching off CIA's logging is the preferred option. You can do so by placing the following command in your `CIA_VMS_ENV.COM` file.

```
$ DEFINE/NOLOG LOGFILE_DIR NL:
```

To use your `CIA_VMS_ENV.COM` instead of the general one, include the following lines in your login.com:

```
$ @SYS$LOGIN:CIA_VMS_ENV.COM
```

If you also wish to have your private `user_init.pro`, then change the logical `idl_startup` within your `CIA_VMS_ENV.COM` file as described above.

2.3.4.2 Unix

You can find out what the command `cia` is by typing:

```
> which cia
```

This will probably be a link. You can use the `ls -l` command to see what it is pointing to if you wish, though it is not particularly important. Inspect this file to see what `cia_vers` is defined as. Alternatively, this information will probably be printed when CIA is started. Now you can copy the file `$cia_vers/unix/user_init.pro`. In order for CIA to find it you should put it in your `$HOME/bin` directory.

To change the default version you can either copy and modify the `cia` file or alias `cia` to include the version, though remember you can also specify it on the command line.

By default, no log files are written. You can turn on the logging by reinstating the commented-out line `set_logfile` in `user_init.pro`. Then they are written to the directory specified by the environment variable `logfile_dir` which in turn is set to the environment variable `cia_logfile_dir`. These variables are set in, respectively, `cia` and `setup/cshrc.camia` if it exists. If it does exist and sets `cia_logfile_dir`, you will need your own version of `cia` to change the logfile directory, otherwise you can simply define `cia_logfile_dir` yourself.

2.4 CIA caveats

Some points to note about IDL and CIA:

- CIA should only be run in a directory where the user has write permission.
- Run only one CIA session per directory. Some CIA routines create temporary files in the current directory. Running more than one CIA session in the same directory can cause a file I/O conflict.
- **xv_raster** might crash for some North-oriented rasters. It is recommended that you use **ximage** instead.
- **ximage** will crash if you use it together with **ciainfo** or **widget_olh**.
- **corr_flat** will fail to flat correct a PDS that contains data with an optical configuration that is not constant.
- There has been reports from users that MIDAS has problems reading CIA generated FITS files. It appears that MIDAS does not implement the full FITS standard, and therefore can not read CIA astrometry.
- There is a known problem with the CIA routine **sscd_del**. Deleting an SSCD may corrupt those SSCDs which have been derived or extracted from the original. For example:

```
CIA> spdtoscd, 'cisp02600506.fits', sscd, dir='$cia_vers/test', /nowrite

CIA> cleaned_sscd=sscd_clean(sscd)

CIA> sscd_del, sscd
```

  Any manipulation of the derived SSCDs (as named in *cleaned_sscd*) will now fail:

```
CIA> sscd_write, cleaned_sscd[0]
ARRAY_HANDLE    LONG      =           72
 3-Feb-2000 14:14:19.00  IA_make_array V.1.0
<First argument (array_handle) is not an handle - E>

etc...

CIA> raster_pds = get_sscdraster( cleaned_sscd[0] )
ARRAY_HANDLE    LONG      =           70
 3-Feb-2000 14:05:23.00  IA_make_array V.1.0
<First argument (array_handle) is not an handle - E>
% HANDLE_CREATE: Invalid handle identifier: 72.

etc...
```

  It is recommended that you do not use **sscd_del** in this manner.

- When CIA is processing a CIA data structure, for example using **sscd_info** on an SSCD, it is not recommended to interrupt the processing by typing Ctrl-C. This can cause a corruption of the data structure in a similar way to the **sscd_del** problem detailed above.

- Since CIA V4.0 the widget are no-blocking for Unix and VMS/Development Environment. While this means that the command line can be used to perform other tasks, most of the widget routines are not re-entrant and therefore can not run in multiple instances.

```
CIA> x3d, raster1
x3d is free
; first instance of x3d runs correctly

CIA> help, raster2, /struct
** Structure <8c878>, 54 tags, length=2941368, refs=3:
   RASTERCOL       INT              4
   RASTERLINE      INT              4
   M_STEPCOL       FLOAT           84.0000
   N_STEPLINE      FLOAT           84.0000
; command line is free to perform other tasks

CIA> x3d, raster2
; incorrectly, a second instance of x3d is invoked.
; x3d continues to work, however on the first dataset.

; After quitting x3d, the following message appears
CIA>
% WSET: Window is closed and unavailable.
% Execution halted at:  X3D_EVENT         625 /spia/APR00/graphics/x3d.pro
%                       WIDGET_PROCESS_EVENTS
%                       $MAIN$

; the session has to be recovered by

CIA> xmanager
CIA> retall
```

- to dark-correct data of the revolutions 12 and 32, only the “library” or “calg” methods can be used.

- There are problems running IDL on display that are 8 bit. If you can force your display into 8 bit mode then it is recommended to do so.

  For Solaris systems with the M64 Graphics Accelerator this can be done with the Solaris command **m64config**:

  ```
  % /usr/sbin/m64config -depth 8
  ```

  Now log out of CDE or whichever windows manager you run. When you log back in the display should be set to 8 bits. To confirm the display configuration:

```
% /usr/sbin/m64config -prconf
```

You might also want to try adding the following line to your *user_init.pro* – see Section 2.3.4 for details of customizing CIA. This command forces IDL to use an 8 bit PseudoColor.

```
% device, pseudo=8, decomposed=0
```

- There are some incompatibilities between different implementations of X Windows. This can cause some widgets to be imcompletely rendered on screen. This situation can occur when CIA is running on remote machine and a PC X-Windows server is used for display.
- The IDL Astronomy User's Library routine **getrot** returns the ROLL angle and not the rotation angle as specified. See Appendix E.
- CIA creates log files when a session is initiated. Information about your session, and any errors which may occur, are recorded in these files. You can use the CIA routine **error_level** to set the level of verbosity of error reporting, both to the screen and the to log files. See the on-line help or **cia_help** (Section 2.3.2) for usage.
- CIA uses IDL's READ and RESTORE to save CIA data structures to file. Such a data file cannot be restored by a version of IDL pre-dating the version which saved the file. However, the converse is not true. IDL can always restore data saved by a preceding version.
- As is usual with IDL, in the event of a crash you may not automatically return to the main IDL level and so your variables will seem to have disappeared. Generally, you can recover by typing RETALL on the command line.
- Following a widget crash you may find that all subsequently called widgets appear dead on your screen. This is a problem with the IDL widget manager, XMANAGER. Usually, invoking it manually (type XMANAGER on the CIA command line) will reactivate your widget.
- Please be aware that the use of netscape (or another X-windows resource hog) might get your CIA session stuck when using widget routines like **xdisp** or **xv_raster**.
- **x_cia**, when running on VMS, is not able to load SSCDs or data structure files if they are not in the current directory.
- Data products from ISO revolutions 12 and 32 contain data that will be imperfectly dark corrected by the 'model' and 'vilspa' dark correction methods. Use an alternative method.

2.5 Acknowledging CIA in publications

CIA is a joint development by the ESA Astrophysics Division and the ISOCAM Consortium. The ISOCAM Consortium is led by the ISOCAM PI, C. Cesarsky. Contributing ISOCAM Consortium institutes are Service d'Astrophysique (SAp, Saclay, France) and Institut d'Astrophysique Spatiale (IAS, Orsay, France) and Infrared Processing and Analysis Center (IPAC, Pasadena, U.S.A.).

When publishing ISOCAM Data reduced with this analysis package, please mention this in the acknowledgement the following way:

> The ISOCAM data presented in this paper was analysed using 'CIA', a joint development by the ESA Astrophysics Division and the ISOCAM Consortium. The ISOCAM Consortium is led by the ISOCAM PI, C. Cesarsky.

If you want to cite CIA in your bibliography, please refer to:

> "Design and Implementation of CIA, the ISOCAM Interactive Analysis System", Ott S., et al, 1997, in ASP Conf. Ser. Vol. 125, Astronomical Data Analysis Software and Systems (ADASS) VI, ed. G. Hunt & H.E.Payne, (San Francisco: ASP), 34

2.6 Reporting problems with CIA

Considering the complexity and size of CIA, it is not unlikely that you may find bugs in some routines. If you do encounter what you suspect is a bug please check it first with your local CIA expert. If (s)he can't help you either, then you are encouraged to submit a report of your problem – or, even better, if you have managed to solve the problem yourself then send us the fixes. Please keep in mind that we can only *try* to solve your problem on a best effort basis...Unfortunately, this means that if the bug is of a trivial nature we may never find the time to fix it.

Instructions for the submission of both Software Problem Reports and Software Change Requests can be found in Section H.1.

Part I

Quick Start Guide

Introduction

The *Quick Start Guide* contains real-life examples of data analysis with CIA for different AOTs, i.e. observation types. It is written for the novice user, and attempts to avoid unnecessary details of the CIA system. However, some CIA and ISOCAM concepts will be introduced: STATE, CONFIGURATION, AOT, SCD, SSCD, IMAGE, EXPOSURE, MOSAIC. See the glossary (Appendix A) for definitions of these terms.

Chapter 3

Raster observation (CAM01)

3.1 Description of the observation

The data used here is from a CAM calibration raster observation of the Antenna galaxy. This observation is comprised of four CONFIGURATIONs. These four CONFIGURATIONs together make up the entire raster observation or AOT. Each CONFIGURATION has a fixed raster size of 2×4 pointings and a PFOV of $3.0''$. However, other parameters do change (indeed this is what gives rise to the different CONFIGURATIONs) such as the filter wheel, integration time, PFOV and gain. Table 3.1 summarizes the parameters for each CONFIGURATION.

3.2 Data analysis

1. Start your CIA session

   ```
   # cia
   ```

   If you work on a VMS system you may have to type `IDL` to begin your CIA session.

2. The data products which are delivered to you on the CD-ROM are in the format of extended FITS files. The first thing you want to do is load all this data into your CIA session. We will use **spdtoscd** to do this. Firstly, be sure that the directory you are working in is writable as CIA needs to be able to write and delete files on the current

CONFIGURATION	raster size	filter wheel	PFOV (arcseconds)	integration time (seconds)	gain
1	2×4	LW2	3.0	5.04	1
2	2×4	LW2	3.0	2.10	2
3	2×4	LW3	3.0	5.04	2
4	2×4	LW3	3.0	2.10	2

Table 3.1: The CONFIGURATION parameters of the raster observation of the Antenna galaxy.

directory (for file saving, sharing data with external executables etc...). Identify the location of the CISP data product. The actual CISP data product name is the first argument to **spdtoscd** and the keyword *dir* is set to the directory containing the CISP file. If you are working from the CD-ROM you will need to copy your data products to disk space – see Section 10.1.

In our example the CISP file is called *cisp02600506.fits* and it is located in the subdirectory *test/* of the CIA installation directory:

```
CIA> spdtoscd, 'cisp02600506.fits', sscd, dir='$cia_vers/test', /nowrite
```

When **spdtoscd** is finished we have created a multitude of data structures in CIA's memory. There is a simple bi-level hierarchy to these structures. At the bottom level there is one data structure for all the data from a CAM STATE, these data structures are called SCDs. In the case of a raster observation, a STATE is time spent in each pointing of CAM. So in our example, there are at least 32 SCDs: 4 CONFIGURATIONs, each of a 2×4 raster, adding up to 32. In order to keep track of all these SCDs there is a top level data structure – this is called an SSCD. It holds relatively few data as its primary function is to catalogue its component SCDs. The variable *sscd* returned by **spdtoscd** contains the unique SSCD name.

```
CIA> print, sscd
CSSC026005060101_98052614571745
```

This name is used to address the data in CIA. Note that the SSCD and SCD are not regular IDL structures, but are implemented in CIA using handles.

3. Remember from Section 3.1 that there is more than one CONFIGURATION in our example data set. This can be illustrated with the routine **sscd_info**. This routine operates on the SSCD and lists the characteristics of its component SCDs. In doing so it is also listing the characteristics of all the STATEs in the AOT.

```
CIA> sscd_info, sscd, /deg
          43 SCDs in the SSCD: CSSC026005060101_98052614571745
 seq channel mode fltrwhl pfov tint gain offset size    ra       dec
   0  LW IDLE     LW2  6.0 25.20   1     512     1 ******* *******
   1  LW IDLE     LW2  6.0  2.10   2     512     1 ******* *******
   2  LW  OBS     LW2  3.0  5.04   1     512    46 180.498 -18.849
   3  LW  OBS     LW2  3.0  5.04   1     512    24 180.490 -18.871
   4  LW  OBS     LW2  3.0  5.04   1     512    24 180.482 -18.893
   5  LW  OBS     LW2  3.0  5.04   1     512    24 180.473 -18.915
   6  LW  OBS     LW2  3.0  5.04   1     512    24 180.450 -18.908
   7  LW  OBS     LW2  3.0  5.04   1     512    23 180.458 -18.886

etc...
```

If you look at the output of **sscd_info** you will see more than the expected 32 SCDs. These others correspond to STATEs where CAM is busy doing other things other than observing. Our next step will be to discard those states. **sscd_clean** will do this for us. It will perform another important task: at the moment there is only one SSCD cataloging all the SCDs; clearly it would be neater if we could divide all these SCDs into four distinct groups, where each group contains all the data from a single CONFIGURATION.

So let's run **sscd_clean** on our SSCD.

```
CIA> cleaned_sscd=sscd_clean(sscd)
Out of 43 SCDs:
11 are rejected due to mode
10 are rejected due to csh flag
9 are rejected due to qla flag
In total 32 are accepted
26-May-1998 15:06:54.00 SSCD_CLEAN v.2.0
<Splitting SSCD into 4 segments - I>
```

The variable *cleaned_sscd* is an array containing the 4 names of the SSCDs cataloging our reorganised SCDs. (Because the SSCD name is derived from your computer's system clock, when you reproduce this example your SSCD names will differ.)

```
CIA> print, cleaned_sscd
 CSSC026005060001_98052615065472 CSSC026005060002_98052615070078
 CSSC026005060003_98052615070752 CSSC026005060004_98052615071446
```

Let's concern ourselves with the first clean SSCD. Using **sscd_info**, compare the characteristics of its component SCDs with those of CONFIGURATION 1 listed in Table 3.1. They should be the same. As you might expect the first SSCD returned by **sscd_clean** contains all the data from the first CONFIGURATION.

```
CIA> sscd_info, cleaned_sscd[0], /deg
           8 SCDs in the SSCD: CSSC026005060001_98052615065472
 seq channel mode fltrwhl pfov tint gain offset size    ra       dec
   0  LW  OBS     LW2  3.0  5.04   1     512   46 180.498 -18.849
   1  LW  OBS     LW2  3.0  5.04   1     512   24 180.490 -18.871
   2  LW  OBS     LW2  3.0  5.04   1     512   24 180.482 -18.893
   3  LW  OBS     LW2  3.0  5.04   1     512   24 180.473 -18.915
   4  LW  OBS     LW2  3.0  5.04   1     512   24 180.450 -18.908
   5  LW  OBS     LW2  3.0  5.04   1     512   23 180.458 -18.886
   6  LW  OBS     LW2  3.0  5.04   1     512   24 180.467 -18.864
   7  LW  OBS     LW2  3.0  5.04   1     512   23 180.475 -18.842
```

4. So we have split our data into four clean SSCDs, one for each CONFIGURATION. Continuing with the first clean SSCD, we now need to gather all the data in its component SCDs into one regular IDL structure. **get_sscdraster** will do this for us.

```
CIA> raster_pds = get_sscdraster( cleaned_sscd[0] )
```

Note that *raster_pds* is a regular IDL structure – in CIA we call this class of structure a prepared data structure, or PDS, and in particular a PDS holding raster obervation data is called a *raster* PDS. We can use IDL's **HELP** to view the contents of our *raster* PDS.

```
CIA> help, raster_pds, /str
** Structure <b09c0>, 54 tags, length=1399800, refs=1:
   RASTERCOL       INT              4
   RASTERLINE      INT              2
   M_STEPCOL       FLOAT          84.0000
   N_STEPLINE      FLOAT          84.0000
   RA_RASTER       DOUBLE          180.47414
   DEC_RASTER      DOUBLE         -18.878420
   ANGLE_RASTER    DOUBLE          109.43000
   RASTER_ROTATION DOUBLE          199.43000
   RASTER_ORIENTATION
                   STRING    = 'SPACECRAFT Y_AXIS'
   ASTR            STRUCT    -> ASTR_STRUC Array[1]
   NX_RASTER       INT            116
   NY_RASTER       INT             60
   RASTER          FLOAT     Array[116, 60]

etc...
```

5. Now it is time to do some actual calibration. However, before we proceed let's take a look at some of the effects we want to eliminate from our data. At the moment the most substantial portion of our data is held in a *raster_pds.cube*. This is a cube of raw CAM IMAGEs taken from all the SCDs of the first clean SSCD. We will use **x3d** to take a look at this cube of IMAGEs – see Figure 3.1.

```
CIA> x3d, raster_pds
Cube(1,9,106) =       134.000
```

x3d is a tool for browsing through the frames of a cube – in our case the frames are CAM IMAGEs. One IMAGE is displayed at a time. The slider bar to the right of the displayed IMAGE can be used to flick through the cube. A plot above the IMAGE displays the value of a selected pixel (selection can be done by right-clicking on the IMAGE) throughout the cube. In effect, this is the history of that pixel during the entire CONFIGURATION.

Upon calling, **x3d** displays the center frame of the cube, i.e. *raster_pds.cube[*, *, 106]*. The are several effects evident in this IMAGE. Dark current causes the alternately dark and bright horizontal lines. The bright pixels at the bottom left of the IMAGE is a cosmic ray glitch – this is clear from the sharp spike in the history of the currently selected pixel, *raster_pds.cube[1, 9, 106]*. Additional effects which are not obvious in Figure 3.1 are: (i) pixel to pixel non-unformity which need to be corrected by flat-fielding, (ii) instability in the detector which can be corrected by a variety of fitting and masking routines. We will now attempt to remove all these effects.

There is a set of CIA calibration routines that we will use to do all the necessary data calibration and correction – Chapter 20 contains more detail on these routines. Almost

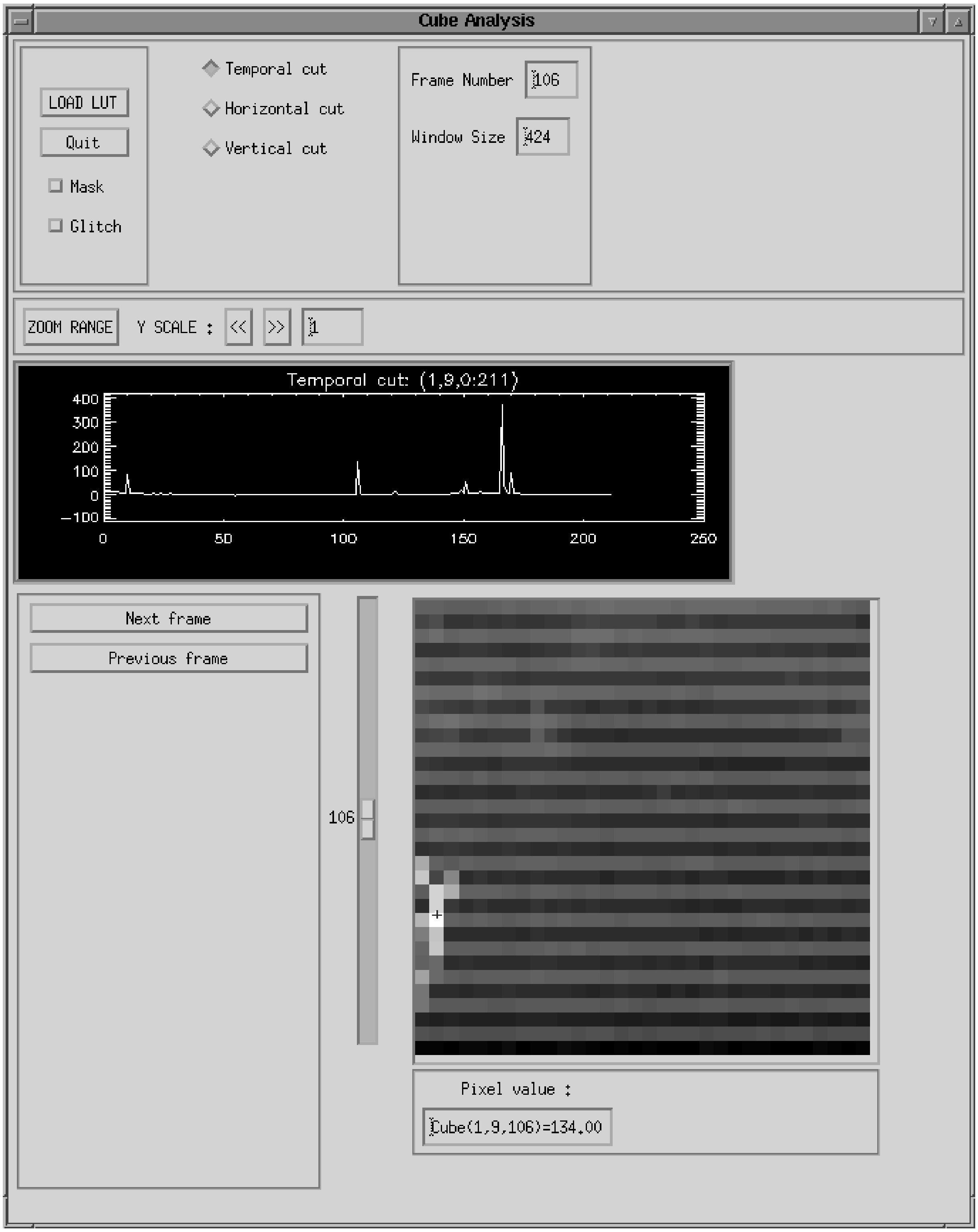

Figure 3.1: A raster IMAGE before calibration. Dark current causes the alternately dark and bright horizontal lines. The bright pixels at the bottom left of the IMAGE are due to a glitch – this is clear from the sharp spike in the history of the currently selected pixel, *raster_pds.cube[1, 9, 106]*. (Displayed by **x3d**.)

all of these routines accept any flavour of PDS, the one exception is the raster MOSAIC creation routine. This is of course specific to a *raster* PDS. All these routines have a choice of different methods, though for simplicity we will just use the default here. Now we will perform the following corrections: (i) dark correction, (ii) deglitching, (iii) stabilization.

```
CIA> corr_dark, raster_pds

CIA> deglitch, raster_pds

CIA> stabilize, raster_pds
```

After some time, and many messages printed to the screen, you will have a fully calibrated *raster* PDS. Now take a look at the cube of IMAGEs again.

```
CIA> x3d, raster_pds
Cube(1,9,106) =       4.03450
```

You will see that the IMAGE displayed by **x3d** (Figure 3.2) has been greatly transformed. The dark current bright and dark lines have gone and so has the glitch. In fact a previously obscured source is now visible in the top half of the IMAGE. You may notice also that the data values of the pixel have decreased. This is because before dark correction all the IMAGEs are normalised to ADUs/gain/second.

We can complete the calibration be reducing the IMAGEs to EXPOSUREs and flat-fielding the EXPOSUREs.

```
CIA> reduce, raster_pds

CIA> corr_flat, raster_pds
```

You can use **x3d** to take a look at the EXPOSUREs.

```
CIA> x3d, raster_pds.image
```

Finally we can create the raster MOSAIC. All the EXPOSUREs will be projected on to the raster field of view.

```
CIA> raster_scan, raster_pds
```

You can view this MOSAIC with **tviso** (a convenient modified version of IDL's **TVSCL**).

```
CIA> tviso, raster_pds.raster
```

The window in Figure 3.3 will appear. It contains the raster MOSAIC that is stored in *raster_pds*.raster.

6. You may wish to save the results of the data analysis. You can do this with IDL's SAVE.

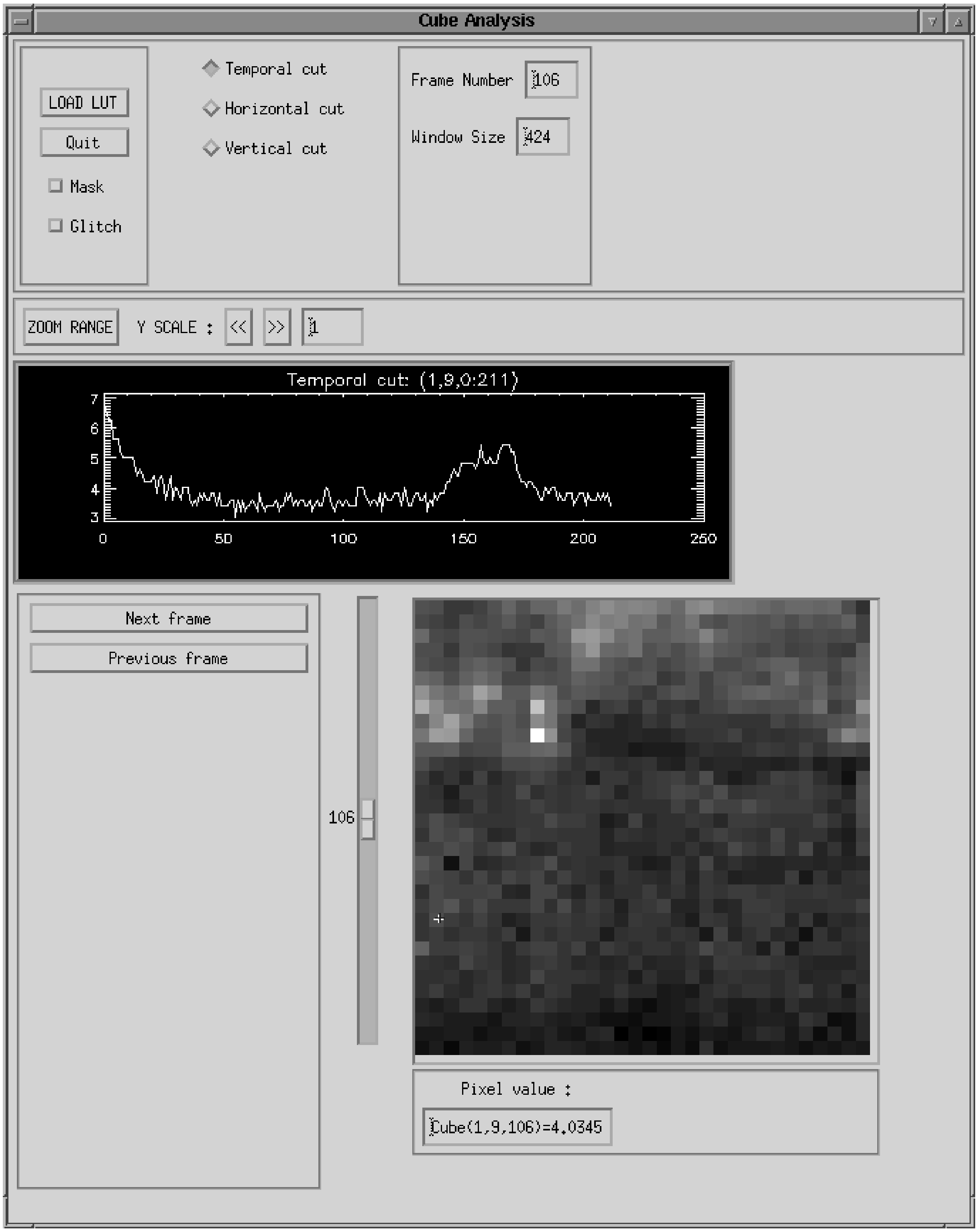

Figure 3.2: A raster IMAGE after dark correction, deglitching and stabilization. The effects of the dark current have been removed and the glitch which was very apparent in Figure 3.1 is no longer visible. A source, which was previously obscured by the glitch, has now become visible in the upper half of the IMAGE. (Displayed by **x3d**.)

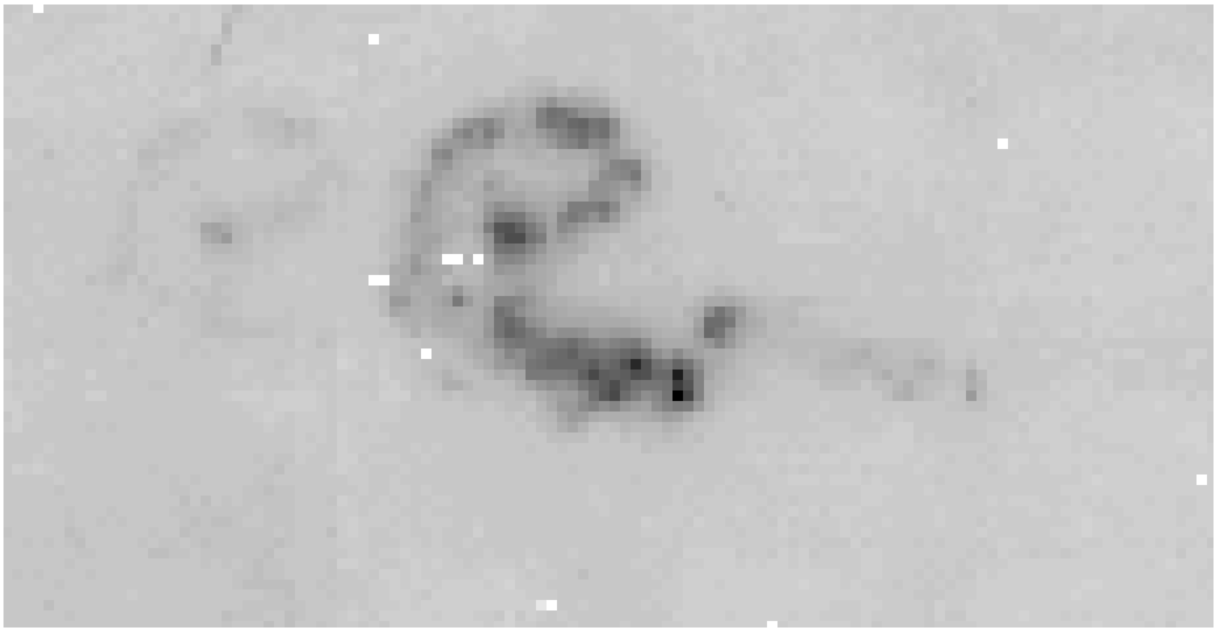

Figure 3.3: The calibrated raster MOSAIC. Some pixels have been masked for the entire duration of this CONFIGURATION. This leads to the appearance of undefined pixels in the raster MOSAIC – here they seem like white 'holes' in the image. (Displayed by **tviso**.)

```
CIA> save, file='raster_pds.xdr', raster_pds
```

Alternatively you can export the data to a FITS file. This is more useful if you intend to perform further analyses with other analysis packages. The following will export *raster_pds* to an IRAF FITS file (see also Section 18.2).

```
CIA> raster2fits, raster_pds, name='raster.fits', /iraf
```

7. You can calibrate the other CONFIGURATIONs in a similiar manner. Start again at Step 4 and use a command like:

```
CIA> another_raster_pds=get_sscdraster( cleaned_sscd[1] )
```

Chapter 4

Staring observation (CAM01)

4.1 Description of the observation

The data used here is from a CAM staring calibration observation of HIC 73005. A staring observation is the simplest application of CAM01 – there is only one pointing in the observation and hence only one STATE per CONFIGURATION. Though you can have several CONFIGURATIONS per observation, the data presented in this chapter is from an observation with a single CONFIGURATION. The relevant parameters are: LW7 filter, 1.5″ PFOV, gain 2 and integration time 2.1 s.

4.2 Data analysis

It is assumed in this section that you have read Chapter 3. Generally concepts described in that section will not be re-described here.

1. Start a CIA session.

```
# cia
```

2. Convert your CISP data product into SCDs with **spdtoscd**.

```
CIA> spdtoscd, 'cisp03001209.fits', sscd, dir='$cia_vers/test', /nowrite
```

3. Get an overview of the SCDs:

```
CIA> sscd_info, sscd, /deg
          4 SCDs in the SSCD: CSSC030012090001_98052617190596
 seq channel mode fltrwhl pfov tint gain offset size    ra       dec
   0  LW IDLE     LW5  1.5  2.10   2    512   12 223.797  53.680
   1  LW  OBS     LW7  1.5  2.10   2    512    7 223.796  53.680
   2  LW  OBS     LW7  1.5  2.10   2    512  101 223.796  53.680
   3  LW IDLE     LW7  1.5  2.10   2    512   22 223.796  53.680
```

Compared to other observations there are very few SCDs / STATEs. This is typical of a staring observation.

4. Remove unwanted SCDs with **sscd_clean**.

```
CIA> cleaned_sscd = sscd_clean( sscd )
Out of 4 SCDs:
2 are rejected due to mode
1 is rejected due to csh flag
3 are rejected due to qla flag
In total 1 is accepted
```

   We are left with only one SSCD and one corresponding SCD.

5. Now we must place the contents of the SSCD into a PDS. For a staring observation we use a *general* PDS. This is created with **get_sscdstruct**.

```
CIA> staring_pds = get_sscdstruct( cleaned_sscd )
```

   You might be interested to see the data at this stage. **x3d** can be used to do this:

```
CIA> x3d, staring_pds
```

6. Now we can proceed with the calibration. We will perform the standard calibration steps on the cube, i.e. *staring_pds*.CUBE. In this observation the data does not need stabilization correction – however feel free to experiment with this.

```
CIA> corr_dark, staring_pds
```

```
CIA> deglitch, staring_pds
```

   Now we have a nicely calibrated PDS. You might want to check this with **x3d**. Use the same calling sequence as above. This time you can click on the button *mask* to see which pixel have been masked by the calibration routines.

7. Finally we make the single EXPOSURE from the cube

```
CIA> reduce, staring_pds
```

   and perform flat field correction on the EXPOSURE

```
CIA> corr_flat, staring_pds
```

8. So to view the results of your calibration use **xdisp** or **tviso** to take a look at the single EXPOSURE.

```
CIA> tviso, staring_pds.image
```

   You should see the same image as in Figure 4.1.

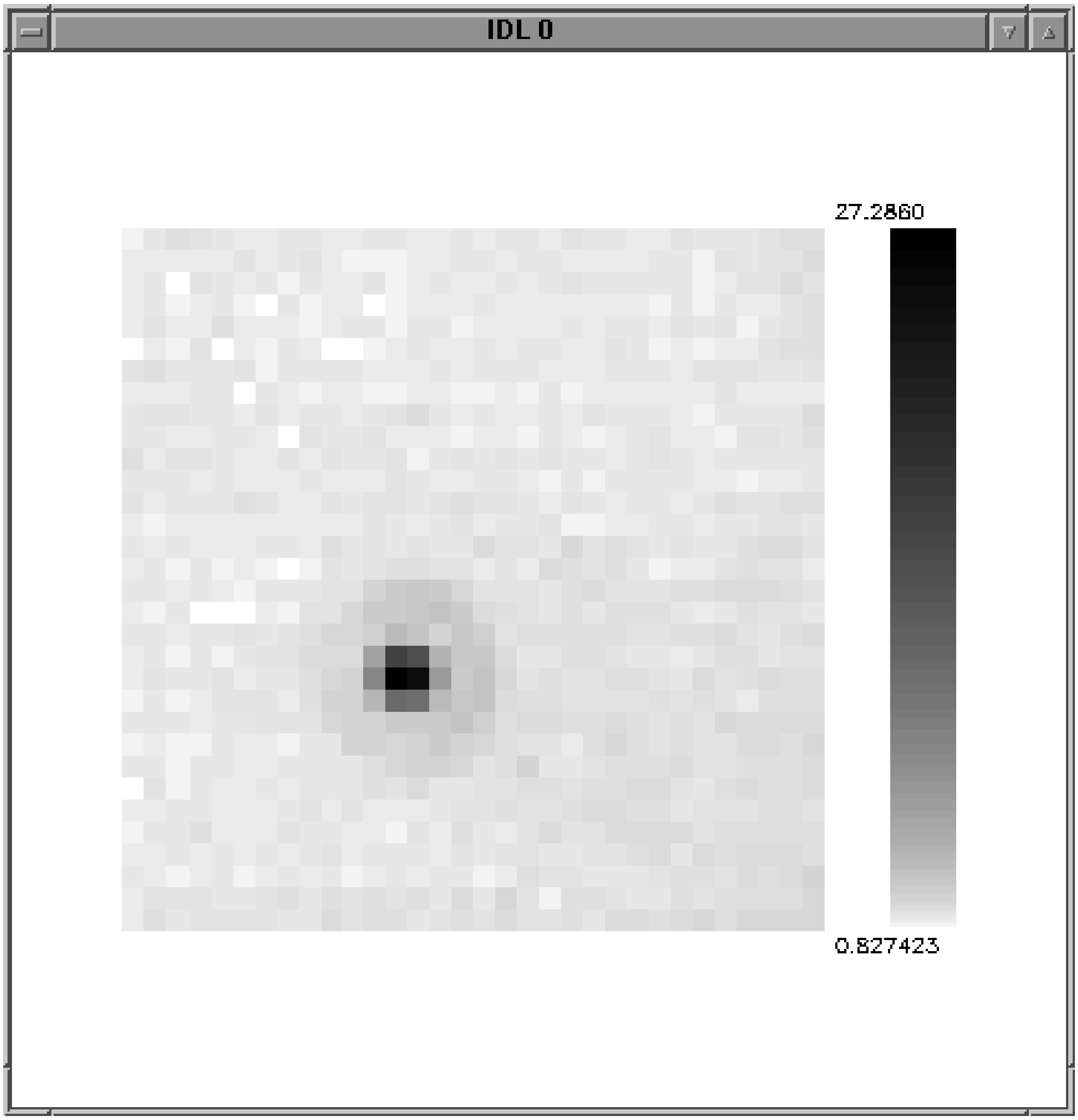

Figure 4.1: **tviso** display of an EXPOSURE from a staring observation.

9. As in the end of Section 3.2 you can save the data using IDL's SAVE.

```
CIA> save, file='staring_pds.xdr', staring_pds
```

And export to a FITS file with CIA's **imagette2fits**. This routine will place the data in the PDS field .IMAGE into the primary array of the FITS file.

```
CIA> imagette2fits, staring_pds, name='staring.fits'
```

Chapter 5

Solar Sytem Object observation (CAM01)

5.1 Description of the observation

The data used here is from a CAM solar system object observation of the comet Tempel-Tuttle. This observation is comprised of four CONFIGURATIONs. These four CONFIGURATIONs together make up the entire observation or AOT. Each CONFIGURATION has roughly the same duration of 180 seconds. In order to keep the object in ISOCAM's FOV, multiple (5 – 6) repointings of ISO had to be made during each CONFIGURATION. To accommodate this repointing of the satellite, solar system observations are implemented as a one-dimensional raster, with the first raster-point identification (M_RASTER) being incremented for each repointing.

5.2 Data analysis

It is assumed in this section that you have read Chapter 3. Generally concepts described in that section will not be re-described here.

1. Start your CIA session

   ```
   # cia
   ```

2. Convert your CISP data into SCDs with **spdtoscd**

   ```
   CIA> spdtoscd, 'cisp81200202.fits', sscd, dir='$cia_vers/test', /nowrite
   ```

   During the slicing, CIA will print the following warning:

   ```
   pixel [14,17] is affected by saturation at SCD 3 with the average
   value 4093.00 (value for End-of-Integration, 3 readouts)
   ```

   This indicates that pixel [14,17] was saturated, and its photometry has to be carefully assessed.

3. Get an overview of the SCDs:

```
CIA> sscd_info, sscd, parameter=['ENTWHL', 'mode', 'fltrwhl', 'pfov', $
CIA> 'tint', 'gain', 'size', 'm_raster', 'n_raster']
  40 SCDs in the SSCD: CSSC812002020101_99121614032101
 seq ENTWHL mode fltrwhl pfov tint gain size m_raster n_raster
   0 HOLE   IDLE     LW2  6.0 25.20    1    1    1      1
   1 HOLE   IDLE     LW2  6.0  2.10    1    1    1      1
   2 HOLE   IDLE     LW2  6.0  2.10    2    1    2      1
   3 HOLE    OBS     LW2  6.0  2.10    2    3    2      1
   4 HOLE    OBS     LW2  6.0  2.10    2    1    2      1
   5 HOLE    OBS     LW2  1.5  2.10    2    1    2      1
   6 HOLE    OBS     LW7  1.5  2.10    2   10    2      1
   7 HOLE    OBS     LW7  1.5  2.10    2   18    3      1
   8 HOLE    OBS     LW7  1.5  2.10    2   18    4      1
   9 HOLE    OBS     LW7  1.5  2.10    2   18    5      1
  10 HOLE    OBS     LW7  1.5  2.10    2   19    6      1
  11 HOLE    OBS     LW7  1.5  2.10    2    9    7      1
  12 HOLE   IDLE     LW7  1.5  2.10    2    3    7      1
  13 HOLE    OBS     LW7  1.5  2.10    1    1    7      1
  14 HOLE    OBS     LW8  1.5  2.10    1    3    7      1
  15 HOLE    OBS     LW8  1.5  2.10    1   18    8      1
  16 HOLE    OBS     LW8  1.5  2.10    1   18    9      1
  17 HOLE    OBS     LW8  1.5  2.10    1   18   10      1
  18 HOLE    OBS     LW8  1.5  2.10    1   18   11      1
  19 HOLE    OBS     LW8  1.5  2.10    1   18   12      1
  20 HOLE    OBS     LW8  1.5  2.10    1    2   13      1
  21 HOLE   IDLE     LW8  1.5  2.10    1    3   13      1
  22 HOLE    OBS     LW8  1.5  2.10    2    1   13      1
  23 HOLE    OBS     LW6  1.5  2.10    2   10   13      1
  24 HOLE    OBS     LW6  1.5  2.10    2   18   14      1
  25 HOLE    OBS     LW6  1.5  2.10    2   18   15      1
  26 HOLE    OBS     LW6  1.5  2.10    2   19   16      1
  27 HOLE    OBS     LW6  1.5  2.10    2   18   17      1
  28 HOLE    OBS     LW6  1.5  2.10    2   10   18      1
  29 HOLE   IDLE     LW6  1.5  2.10    2    3   18      1
  30 HOLE    OBS     LW6  1.5  2.10    1    1   18      1
  31 HOLE    OBS     LW9  1.5  2.10    1    2   18      1
  32 HOLE    OBS     LW9  1.5  2.10    1   18   19      1
  33 HOLE    OBS     LW9  1.5  2.10    1   18   20      1
  34 HOLE    OBS     LW9  1.5  2.10    1   18   21      1
  35 HOLE    OBS     LW9  1.5  2.10    1   18   22      1
  36 HOLE    OBS     LW9  1.5  2.10    1   18   23      1
  37 HOLE    OBS     LW9  1.5  2.10    1    1   24      1
  38 HOLE   IDLE     LW9  1.5  2.10    1    6   24      1
  39 HOLE    OBS     LW2  6.0 25.20    1    1   24      1
```

Depending on the apparent velocity of the target, the number of SCDs / STATEs for each configuration can vary significantly.

4. Remove unwanted SCDs with **sscd_clean**.

```
CIA> cleaned_sscd = sscd_clean( sscd )
Out of 40 SCDs:
7 are rejected due to mode
13 are rejected due to csh flag
16 are rejected due to qla flag
In total 22 are accepted
16-Dec-1999 14:30:22.00  SSCD_CLEAN v.2.5
<Splitting SSCD into 4 segments - I>
```

We are left with four SSCDs, corresponding to the four configurations.

Let's concern ourselves with the first clean SSCD. Using **sscd_info**, compare the characteristics of its component SCDs with those of the whole observation. As expected the first SSCD returned by **sscd_clean** contains all the data from the first CONFIGURATION, an observation using the LW7 filter. Also, it can be seen that 5 pointings were performed to track the movement of the comet.

```
CIA> sscd_info, cleaned_sscd[0], parameter=['channel', 'ENTWHL', 'mode', $
CIA> 'fltrwhl', 'pfov', 'tint', 'gain', 'size', 'm_raster', 'n_raster']
          5 SCDs in the SSCD: CSSC812002020001_99121614325400
 seq channel ENTWHL mode fltrwhl pfov tint gain size m_raster n_raster
   0      LW HOLE   OBS     LW7  1.5  2.10    2   18     3      1
   1      LW HOLE   OBS     LW7  1.5  2.10    2   18     4      1
   2      LW HOLE   OBS     LW7  1.5  2.10    2   18     5      1
   3      LW HOLE   OBS     LW7  1.5  2.10    2   19     6      1
   4      LW HOLE   OBS     LW7  1.5  2.10    2    9     7      1
```

5. Now we must place the contents of the SSCD into a PDS. For an SSO observation we use a *general* PDS. This is created with **get_sscdstruct**.

```
CIA> sso_pds = get_sscdstruct( cleaned_sscd[0] )
```

6. Now we can proceed with the calibration. We will perform the standard calibration steps on the cube, i.e. *sso_pds.cube*. In this observation the data does not need stabilization correction – however feel free to experiment with this.

```
CIA> corr_dark, sso_pds

CIA> deglitch, sso_pds
```

Now we have a nicely calibrated PDS. You might want to check this with **x3d**. This time you can click on the button *mask* to see which pixel have been masked by the calibration routines.

7. The final step in the data reduction is to create the MOSAIC image. This is really just the average of *all* the IMAGEs in *sso_pds.cube* and we can easily derive it with CIA's **reduce_cube**. Before doing so we must do a 'fake' reduce as we normally do when reducing a *raster* PDS, *BS* PDS or *CVF* PDS. This is really just to keep the PDS consistent with the data reduction step by making sure important fields, such as *sso_pds.image_unit*, are updated.

```
CIA> reduce, sso_pds
```

Now you can take a look at the EXPOSURE obtained for each SCD or pointing of the spacecraft. These images should look very much alike – all will be slightly smeared. This smearing is due to steps ISO makes as it repoints, i.e. the tracking is of course not smooth. You will also notice that the position of the object may be slightly different from EXPOSURE to EXPOSURE. This second effect is due to the inaccuracies of ISO's pointing.

```
CIA> x3d, sso_pds
```

Now we will do the 'real' reduce manually. The MOSAIC image is stored in the first frame of *sso_pds.image*:

```
CIA> sso_pds.image[*,*,0] = reduce_cube(sso_pds.cube, mask=sso_pds.mask)
```

Flat correction finishes off this reduction step:

```
CIA> corr_flat, sso_pds
```

8. So to view the results of your calibration use **xdisp** or **tviso**, specifying the first frame of *sso_pds.image* as input:

```
CIA> tviso, sso_pds.image[*,*,0]
```

You should see the same image as in Figure 5.1. The effects that we saw earlier in Step 7 with **x3d** (smearing and pointing inaccuracies) will have combined to make the comet look somewhat blurred in this final MOSAIC image.

9. Now we can convert the MOSAIC image from ISOCAM units into milli-janskys (mJy):

```
CIA> conv_flux, sso_pds
```

10. As in the end of Section 3.2 you can save the data using IDL's SAVE:

```
CIA> save, file='sso_pds.xdr', sso_pds
```

Export to FITS may be performed with CIA's **imagette2fits**. This routine will place the data in the field *sso_pds.image*, and hence the MOSAIC image, into the primary array of the FITS file:

```
CIA> imagette2fits, sso_pds, name="tempel-tuttle.fits", rank=0
```

We use the keyword *rank* to specify the MOSAIC image, i.e. the first frame of *sso_pds.image*. This image is written to the file *tempel-tuttle1.fits*.

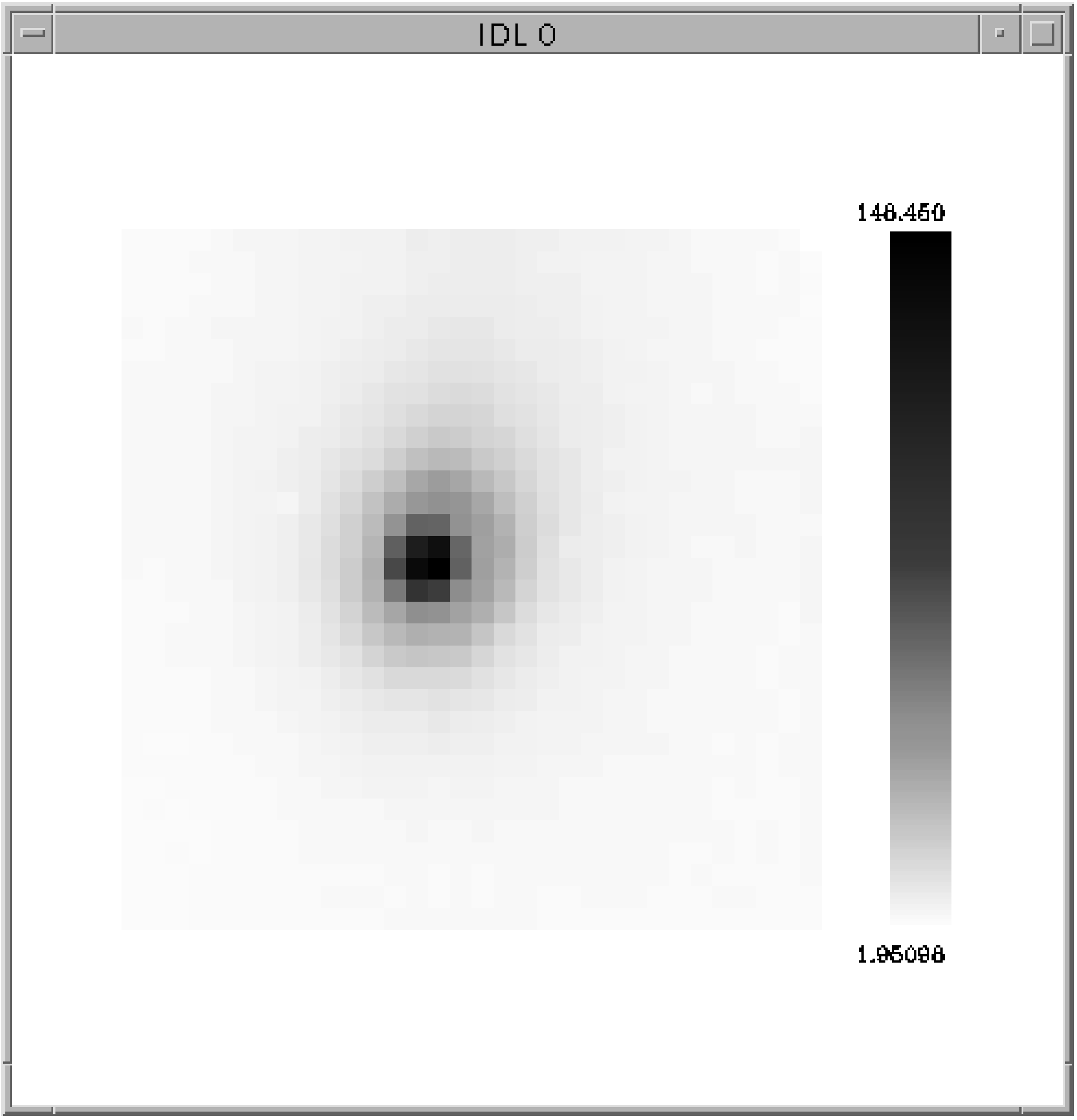

Figure 5.1: **tviso** display of the MOSAIC image from a solar system object observation.

Chapter 6

Beam-switch observation (CAM03)

6.1 Description of the observation

The data used here is from a CAM beam-switch calibration observation of HIC 96901. A beam-switch observation comprises of a cycle of ISO pointings – an on source pointing followed by an off source or reference pointing. There can be one or several cycles in a single observation. Since each pointing is a STATE, we will have at least two STATEs per CONFIGURATION. The CONFIGURATION (and in this case the STATE) parameters are: LW10 filter, 1.5″ PFOV, integration time of 2.10 s and gain 2. In the observation analysed here there are 4 cycles.

6.2 Data analysis

It is assumed in this section that you have read Chapter 3. Generally concepts described in that section will not be re-described here.

1. Start a CIA session.

```
# cia
```

2. Convert your CISP data product into SCDs with **spdtoscd**.

```
CIA> spdtoscd, 'cisp05804610.fits', sscd, dir='$cia_vers/test', /nowrite
```

3. Get an overview of the SCDs:

```
CIA> sscd_info, sscd, /deg
          23 SCDs in the SSCD: CSSC058046100101_98052615192743
 seq channel mode fltrwhl pfov tint gain offset size    ra       dec
   0  LW IDLE     LW2  6.0 25.20   1    512    1 ******* *******
   1  LW IDLE     LW2  6.0  2.10   2    512    1 ******* *******
   2  LW  OBS     LW2  6.0  2.10   2    512    1 ******* *******
   3  LW  OBS     LW2  6.0  2.10   2    512    1 ******* *******
   4  LW  OBS    LW10  1.5  2.10   2    512   14 295.462  50.518
   5  LW  OBS    LW10  1.5  2.10   2    512   26 295.463  50.518
   6  LW  OBS    LW10  1.5  2.10   2    512   21 295.462  50.518
```

```
  7  LW  OBS    LW10  1.5  2.10   2    512    27 295.437  50.547
  8  LW  OBS    LW10  1.5  2.10   2    512    21 295.437  50.547
  9  LW  OBS    LW10  1.5  2.10   2    512    26 295.463  50.518
 10  LW  OBS    LW10  1.5  2.10   2    512    22 295.462  50.517
 11  LW  OBS    LW10  1.5  2.10   2    512    26 295.417  50.501
 12  LW  OBS    LW10  1.5  2.10   2    512    21 295.417  50.501
 13  LW  OBS    LW10  1.5  2.10   2    512    26 295.463  50.518
 14  LW  OBS    LW10  1.5  2.10   2    512    22 295.463  50.517
 15  LW  OBS    LW10  1.5  2.10   2    512    26 295.488  50.488
 16  LW  OBS    LW10  1.5  2.10   2    512    21 295.488  50.488
 17  LW  OBS    LW10  1.5  2.10   2    512    27 295.463  50.518
 18  LW  OBS    LW10  1.5  2.10   2    512    21 295.463  50.518
 19  LW  OBS    LW10  1.5  2.10   2    512    26 295.508  50.534
 20  LW  OBS    LW10  1.5  2.10   2    512     2 295.508  50.534
 21  LW IDLE    LW10  1.5  2.10   2    512    24 295.463  50.518
 22  LW IDLE     LW2  6.0 25.20   1    512     1 ******* *******
```

There are quite a few SCDs here. Some of these we will discard.

For each source and reference pointing there are a pair of SCDs. The first SCD contains on target data and the second SCD contains data from an intermediate step when ISO is slewing. This SCD is considered to contain invalid data, though in reality the data may be quite usable. Section 19.5.1 describes how such data may be incorporated into the data reduction.

4. We normally can not use **sscd_clean** on a beam-switch observation (see Section 19.5.1 for a work-around). However, it is a simple manner of looking at the above list and deleting the undesirable SCDs. STATEs that are IDLE and STATEs where the filter wheel was not set to LW10 should be discarded:

```
CIA> scds = sscd_elem( sscd )
CIA> scd_del, scds[ 0:3 ]
CIA> scd_del, scds[ 21:22 ]
```

5. Now we must place the contents of the SSCD into a PDS. For a beam-switch observation we use a *BS* PDS. This is created with **get_sscdbs**.

```
CIA> bs_pds = get_sscdbs( sscd )
```

Note that there is one irregularity which sometimes arises in beam-switch observations: some observers have programmed their observations in reverse! Section 19.4 tells you how **get_sscdbs** can be used to deal with this problem.

6. Now we can proceed with the calibration. We will perform the standard calibration steps on the cube, i.e. *bs_pds.cube*.

```
CIA> x3d, bs_pds
```

When you are satisfied we can perform the first calibration steps of DARK correction and deglitching. As in Section 4.2, the data does not need stabilization correction – if you are not convinced you can check this with **x3d** before progressing.

```
CIA> corr_dark, bs_pds
```

```
CIA> deglitch, bs_pds
```

Now the PDS contains a nicely calibrated cube. Again, you might want to check this with **x3d** – see Figure 6.1. Use the same calling sequence as above.

7. Now we can create the EXPOSUREs

   ```
   CIA> reduce, bs_pds
   ```

   and perform flat field correction

   ```
   CIA> corr_flat, bs_pds
   ```

8. You may be curious as to which EXPOSUREs are source and which are reference:

   ```
   CIA> print, bs_pds.src_image
          1       5       9      13
   CIA> print, bs_pds.ref_image
          3       7      11      15
   ```

   You can use **x3d** to look at the EXPOSUREs:

   ```
   CIA> x3d, bs_pds.image
   ```

   Use the slider to select the desired EXPOSURE, e.g. for the second source EXPOSURE set the slider to 5.

9. Finally we can create the beam-switch MOSAIC. This is simply done.

   ```
   CIA> reduce_bs, bs_pds
   ```

   reduce_bs adds up all the source EXPOSUREs and subtracts the reference EXPOSUREs (taking the MASK into account of course). To display this MOSAIC:

   ```
   CIA> tviso, bs_pds.raster
   ```

   You should see the same image as in Figure 6.2. Note that for historical reasons the beam-switch MOSAIC is placed in the field .RASTER.

10. You may wish to save the results of the data analysis. You can do this with IDL's SAVE.

   ```
   CIA> save, file='bs_pds.xdr', bs_pds
   ```

   Alternatively you can export the data to a FITS file. Again we take advantage of the *BS* PDS compatibility with the *raster* PDS and use **raster2fits**:

   ```
   CIA> raster2fits, bs_pds, name='bs.fits'
   ```

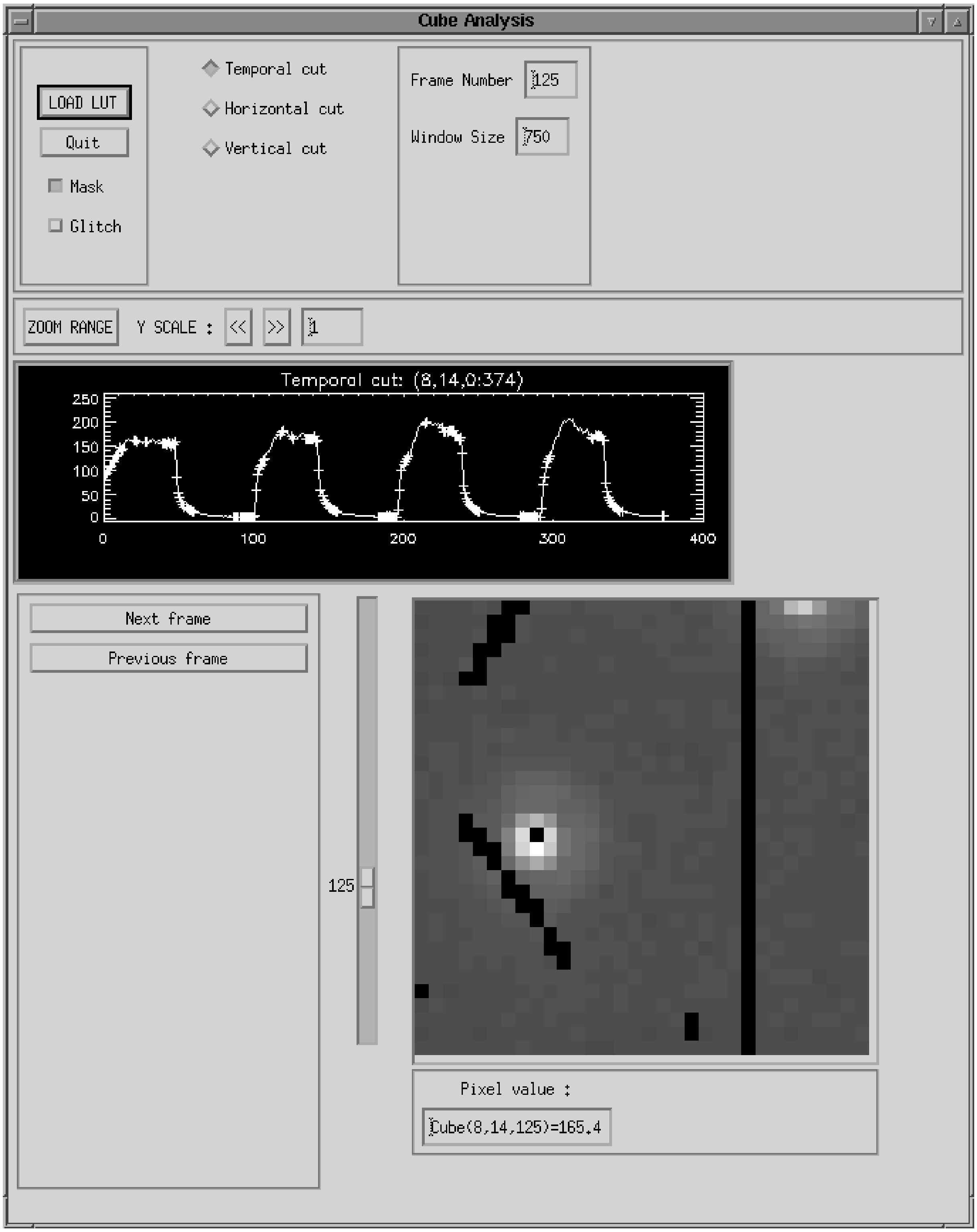

Figure 6.1: **x3d** display of a calibrated *BS* PDS.CUBE. The button *mask* has been activated. Undefined image pixels are blank. In the plot window crosses mark where pixels are undefined in the their time history.

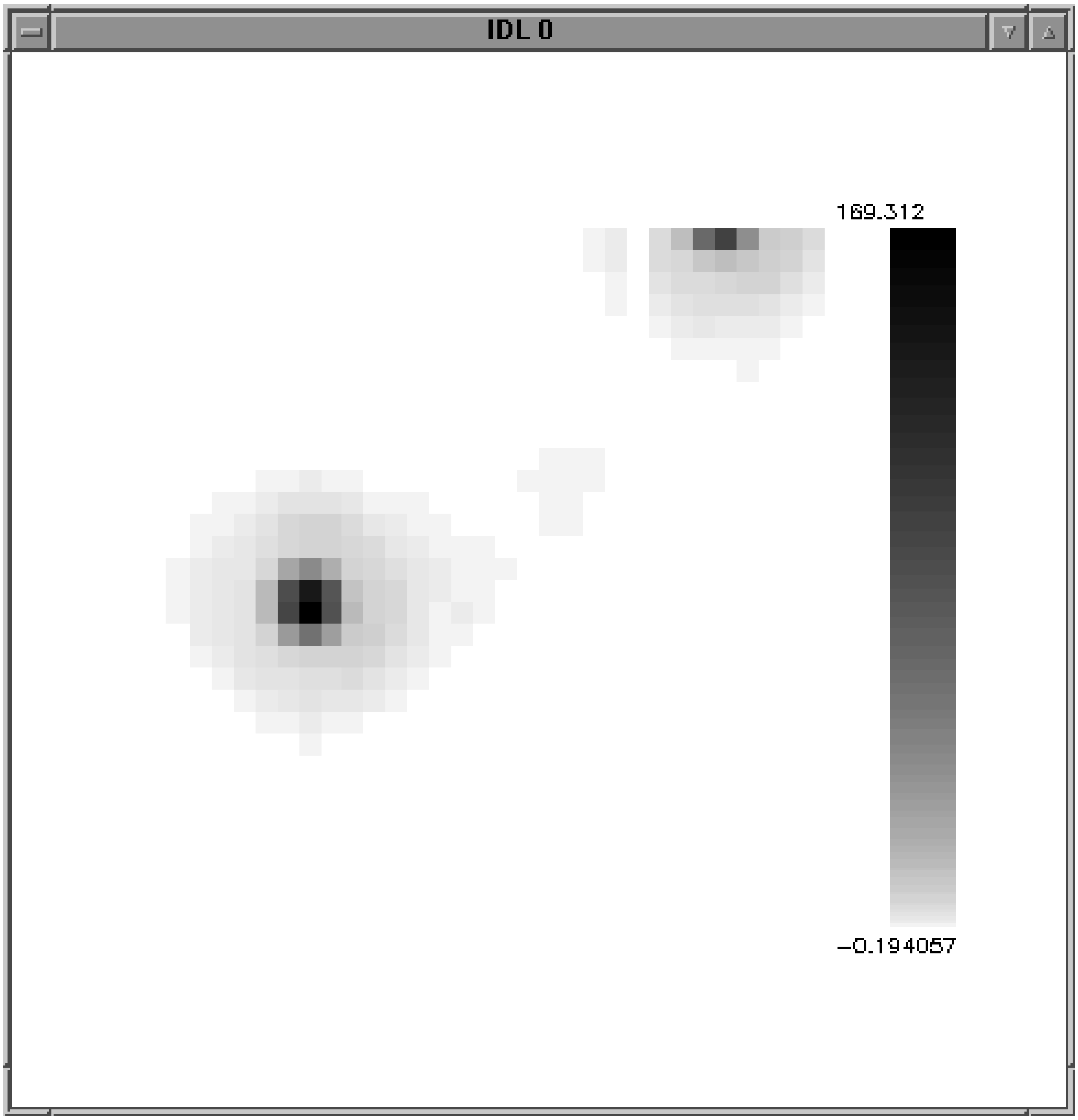

Figure 6.2: **tviso** display of beam-switch MOSAIC. Undefined pixels in column 24 are blank.

Chapter 7

CVF observation (CAM04)

7.1 Description of the observation

The data used here is from a CAM calibration raster observation of the object HIC94890. This observation is comprised of only one CONFIGURATION – this makes it simpler than the raster obervation described in Section 3. The CONFIGURATION parameters are: LW channel, CVF1 filter wheel, 6″ PFOV and a gain of 2. As we will later see this CONFIGURATION has 20 STATEs where meaningful observation data was accumulated by CAM. In a raster observation the pointing changes from STATE to STATE, but as we will see later, in a CVF observation the CVF wheel position changes.

7.2 Data analysis

It is assumed in this section that you have read Chapter 3. Generally concepts described in that section will not be re-described here.

1. Start a CIA session.

   ```
   # cia
   ```

2. Convert your CISP data product into SCDs with **spdtoscd**.

   ```
   CIA> spdtoscd, 'cisp05805004.fits', sscd, dir='$cia_vers/test', /nowrite
   ```

3. Remove unwanted SCDs with **sscd_clean**.

   ```
   CIA> cleaned_sscd = sscd_clean( sscd )
   Out of 22 SCDs:
   2 are rejected due to mode
   0 are rejected due to csh flag
   2 are rejected due to qla flag
   In total 20 are accepted

   CIA> print, cleaned_sscd
   CSSC058050040001_98052617314573
   ```

In contrast to the raster observation in Section 3 we here we have only one CONFIGURATION and consequently only one SSCD.

4. As in Section 3 we need to create a PDS from the SSCD. For CVF data, the equivalent routine to do this is **get_sscdcvf**. Note that the resulting *CVF* PDS differs somewhat from a *raster* PDS.

```
CIA> cvf_pds = get_sscdcvf( cleaned_sscd )
```

5. Now we can calibrate the data in *cvf_pds*. Mostly everthing that was discussed in the previous data analysis examples apply here, with the exception that there is no MOSAIC for the *CVF* PDS. As in Section 4.2, this data does not need stabilization correction. Again, as in the previous data analysis examples, you don't have to execute all the commands below at once – you may like to examine the data with **x3d** between each command.

```
CIA> corr_dark, cvf_pds

CIA> deglitch, cvf_pds

CIA> stabilize, cvf_pds

CIA> reduce, cvf_pds

CIA> corr_flat, cvf_pds
```

An additional step that we will perform is the conversion of the EXPOSUREs from ADU to milli-jansky (mJy).

```
CIA> conv_flux, cvf_pds
```

To view the results of your calibration use **cvf_display**.

```
CIA> cvf_display, cvf_pds
```

The **cvf_display** (Figure 7.1) window displays an EXPOSURE (co-added IMAGE) and a plot of the values of a selected pixel throughout the cube of EXPOSUREs. As stated earlier, the EXPOSURE is derived from the data in a STATE or SCD, and since the CVF wheel position changed as the STATE changed, then the EXPOSUREs are pictures taken over a range of wavelengths. This means the plot is a CVF spectrum of the sky covered by a selected pixel. You can select pixels by right-clicking on the EXPOSURE. The CVF spectrum is calibrated to flux in mJy per square pixel against wavelength in microns.

6. You may wish to save the results of the data analysis. You can do this with IDL's SAVE.

```
CIA> save, file='cvf_pds.xdr', cvf_pds
```

Alternatively you can export each of the CVF EXPOSUREs in .IMAGE to individual FITS files (see also Section 18.2).

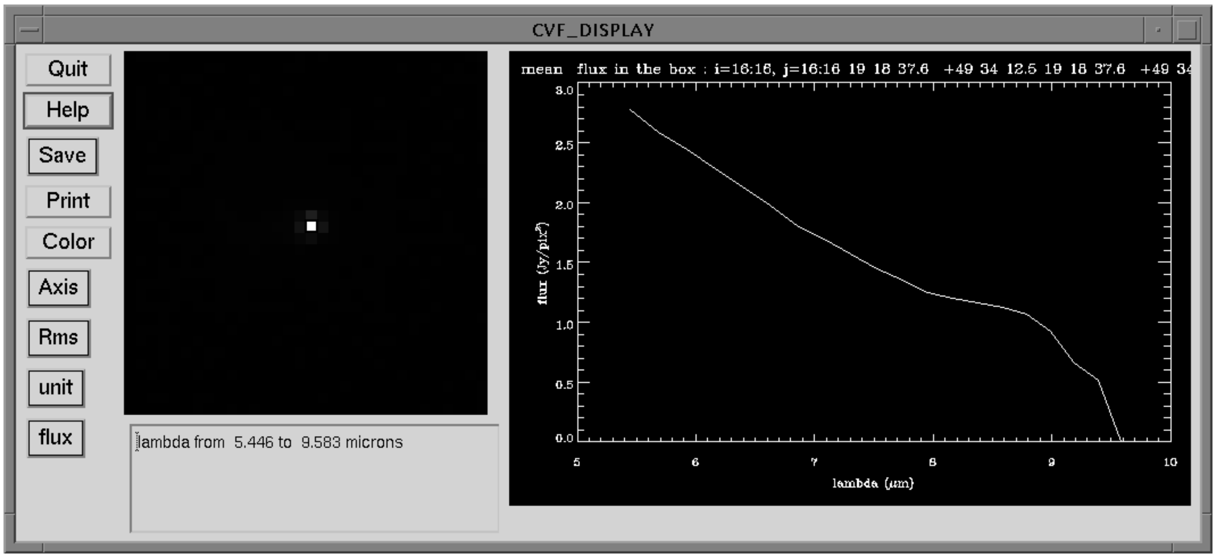

Figure 7.1: The left panel of the **cvf_display** window displays an EXPOSURE from *cvf_pds*. The right pane displays a plot of the CVF spectrum of a pixel selected from that EXPOSURE.

```
CIA> imagette2fits, cvf_pds, name='cvf.fits'

CIA> $ls cvf*.fits
cvf1.fits   cvf13.fits  cvf17.fits  cvf20.fits  cvf6.fits
cvf10.fits  cvf14.fits  cvf18.fits  cvf3.fits   cvf7.fits
cvf11.fits  cvf15.fits  cvf19.fits  cvf4.fits   cvf8.fits
cvf12.fits  cvf16.fits  cvf2.fits   cvf5.fits   cvf9.fits
```

Chapter 8

Polarization observation (CAM05/dedicated CAM99)

8.1 Description of the observation

The data used here is from a CAM polarization observation of the object HIC085371. This observation is basically a 2×2 raster observation, though at each raster pointing the entrance wheel cycles through the 3 polarizors 3 times. Hence there are many SCDs for this observations: $2 \times 2 \times 3 \times 3$ in addition to several IDLE and other uninteresting SCDs. This chapter provides a good example of the technique of calibrating an SSCD.

8.2 Data analysis

It is assumed in this section that you have read Chapter 3. Generally concepts described in that section will not be re-described here. In this example, it is necessary to calibrate the SSCD, and not the PDS. This is because the polarization data will no longer be contiguous when it is frozen in a PDS – see Section 13.1.1 for details of the limitations of a PDS. This will become more obvious later in the chapter when we get to the SSCD cleaning stage. For more on calibration of SSCDs see Section 20.3.

8.2.1 Overview of calibration steps

The steps involved in the calibration of data from this polarization observation may be summarized as:

1. Slice in the usual way using one of the slicing routines, e.g. **spdtoscd** or **x_slicer**. Perform dark, deglitching and transient correction on the SSCD using the routines **corr_dark**, **deglitch** and **stabilize**.

2. After the core calibration steps are complete we manually clean and combine SSCDs and SCDs before attempting to create a PDS. This step is very much dependent on how the observation was programmed.

3. Now freeze the SSCD into a *raster* PDS with **get_sscdraster** and reduce the IMAGEs to EXPOSUREs with **reduce**.

4. Flat-field the EXPOSUREs. Firstly, identify the entrance wheel associated with each EXPOSURE. Restore the appropriate flat-field image from the set of polarization flat-fields distributed with CIA and supply as input to **corr_flat**.

5. Convert the pixel values to milli-janskys (mJy) with **conv_flux** and perform photometry on the EXPOSUREs and/or MOSAIC images. One may use **xphot** for this purpose.

6. Use **get_polar_weight** to determine the polar weight factors and **comp_stokes** to calculate the Stokes parameters.

8.2.2 Slice and perform core calibration

Slicing is straightforward:

```
CIA> spdtoscd, 'cisp35600501.fits', sscd, dir='$cia_vers/test', /nowrite
```

Core calibration may be done with the usual routines except we work directly on the SSCD rather than on a PDS. Later you will see that the PDS will contain SCDs extracted from different points within the SSCD or observation, i.e. the PDS will not contain contiguously acquired data. Because core calibration works best on contiguous data we must postpone the freezing of the data in a PDS until *after* the core calibration.

```
CIA> corr_dark, sscd
CIA> deglitch, sscd
CIA> stabilize, sscd
```

8.2.3 Clean the SSCD

Cleaning the SSCD is a difficult task that must be done manually. A single polarization observation may be a combination of multiple rasters with changing entrance wheel at each raster point id or each raster position. The steps for this observation are as follows.

1. First list all the SCDs in the observation:

```
CIA> sscd_info, sscd, /pol
          50 SCDs in the SSCD: CSSC356005010001_00020117264901
 seq      entwhl mode fltrwhl pfov  tint gain size m_raster n_raster
   0        HOLE  OBS     LW2  6.0 25.20    1    1        1        1
   1        HOLE IDLE     LW2  6.0  2.10    1   11        1        1
   2        HOLE  OBS     LW2  3.0  2.10    1  102        1        1
   3        HOLE  OBS     LW2  3.0  2.10    1   92        2        1
   4        HOLE  OBS     LW2  3.0  2.10    1   92        2        2
   5        HOLE  OBS     LW2  3.0  2.10    1   97        1        2
   6        HOLE  OBS     LW2  3.0  2.10    1   13        1        1
   7 POLARIZOR 1  OBS     LW2  3.0  2.10    1   78        1        1
   8 POLARIZOR 2  OBS     LW2  3.0  2.10    1   77        1        1
   9 POLARIZOR 3  OBS     LW2  3.0  2.10    1    1        1        1
  10 POLARIZOR 3  OBS     LW2  3.0  2.10    1   78        1        1
  11 POLARIZOR 1  OBS     LW2  3.0  2.10    1   78        1        1
  12 POLARIZOR 2  OBS     LW2  3.0  2.10    1   77        1        1
```

```
  13 POLARIZOR 3  OBS     LW2  3.0  2.10    1   78         1         1
  14 POLARIZOR 1  OBS     LW2  3.0  2.10    1   78         1         1
  15 POLARIZOR 2  OBS     LW2  3.0  2.10    1   78         1         1
  16 POLARIZOR 3  OBS     LW2  3.0  2.10    1   78         1         1
  17 POLARIZOR 1  OBS     LW2  3.0  2.10    1    4         1         1
  18 POLARIZOR 1  OBS     LW2  3.0  2.10    1   74         2         1
  19 POLARIZOR 2  OBS     LW2  3.0  2.10    1   77         2         1
  20 POLARIZOR 3  OBS     LW2  3.0  2.10    1   78         2         1
  21 POLARIZOR 1  OBS     LW2  3.0  2.10    1   78         2         1
  22 POLARIZOR 2  OBS     LW2  3.0  2.10    1   78         2         1
  23 POLARIZOR 3  OBS     LW2  3.0  2.10    1   78         2         1
  24 POLARIZOR 1  OBS     LW2  3.0  2.10    1   78         2         1
  25 POLARIZOR 2  OBS     LW2  3.0  2.10    1   78         2         1
  26 POLARIZOR 3  OBS     LW2  3.0  2.10    1   78         2         1
  27 POLARIZOR 1  OBS     LW2  3.0  2.10    1    8         2         1
  28 POLARIZOR 1  OBS     LW2  3.0  2.10    1   70         2         2
  29 POLARIZOR 2  OBS     LW2  3.0  2.10    1   77         2         2

etc...
```

We need to extract and concatenate the correct SCDs in order to make a *raster* PDS containing data acquired with the same entrance wheel parameter. First we clean the SSCD and eliminate all SCDs that have obviously too few IMAGES to be valid. In this example, **cleaned_sscd** splits the SSCD into two segements –an unnecessary sideeffect due to limitations in the design of the SSCD and the CAM05 observing mode. We can however easily recombine the SSCD segements:

```
CIA> cleaned_sscd = sscd_clean( sscd, minimum=20 )
CIA> sscd_del, sscd
CIA> sscd_concatene, cleaned_sscd[0], cleaned_sscd[1]
```

2. The next step is to create a new SSCDs containing SCDs from a single entrance wheel parameter setting. This we can do with **scds_select**:

```
CIA> sscd_polar1 = $
  scds_select(cleaned_sscd[0], 'polar1', 'entwhl', 'POLARIZOR 1' )

CIA> sscd_info, sscd_polar1, scds, /pol
          12 SCDs in the SSCD: CSSC000000POLAR1_00020117462500
 seq      entwhl mode fltrwhl pfov  tint gain size m_raster n_raster
   0 POLARIZOR 1  OBS     LW2  3.0  2.10    1   78         1         1
   1 POLARIZOR 1  OBS     LW2  3.0  2.10    1   78         1         1
   2 POLARIZOR 1  OBS     LW2  3.0  2.10    1   78         1         1
   3 POLARIZOR 1  OBS     LW2  3.0  2.10    1   74         2         1
   4 POLARIZOR 1  OBS     LW2  3.0  2.10    1   78         2         1
   5 POLARIZOR 1  OBS     LW2  3.0  2.10    1   78         2         1
   6 POLARIZOR 1  OBS     LW2  3.0  2.10    1   70         2         2
   7 POLARIZOR 1  OBS     LW2  3.0  2.10    1   78         2         2
```

```
   8 POLARIZOR 1  OBS     LW2  3.0  2.10    1   78        2        2
   9 POLARIZOR 1  OBS     LW2  3.0  2.10    1   64        1        2
  10 POLARIZOR 1  OBS     LW2  3.0  2.10    1   78        1        2
  11 POLARIZOR 1  OBS     LW2  3.0  2.10    1   78        1        2
```

3. Clearly from the above listing *sscd_polar1* contains only those SCDs with the entrance wheel set to *POLARIZOR 1*. However, for each of the 4 raster positions we have 3 SCDs. To merge these 4 triplets we can use **scd_concatene**:

```
CIA> tmp = scd_concatene( scds[0], scds[1] )
CIA> scd_del, scds[0] & scd_del, scds[1]
CIA> scd0 = scd_concatene( tmp, scds[2] )
CIA> scd_del, tmp & scd_del, scds[2]

CIA> tmp = scd_concatene( scds[3], scds[4] )
CIA> scd_del, scds[3] & scd_del, scds[4]
CIA> scd1 = scd_concatene( tmp, scds[5] )
CIA> scd_del, tmp & scd_del, scds[5]

CIA> tmp = scd_concatene( scds[6], scds[7] )
CIA> scd_del, scds[6] & scd_del, scds[7]
CIA> scd2 = scd_concatene( tmp, scds[8] )
CIA> scd_del, tmp & scd_del, scds[8]

CIA> tmp = scd_concatene( scds[9], scds[10] )
CIA> scd_del, scds[9] & scd_del, scds[10]
CIA> scd3 = scd_concatene( tmp, scds[11] )
CIA> scd_del, tmp & scd_del, scds[11]
```

And now we have a more usefully arranged SSCD.

```
CIA> sscd_info, sscd_polar1, /pol
            4 SCDs in the SSCD: CSSC000000POLAR1_00020118111300
 seq      entwhl mode fltrwhl pfov  tint gain size m_raster n_raster
   0 POLARIZOR 1  OBS     LW2  3.0  2.10    1  234        1        1
   1 POLARIZOR 1  OBS     LW2  3.0  2.10    1  230        2        1
   2 POLARIZOR 1  OBS     LW2  3.0  2.10    1  226        2        2
   3 POLARIZOR 1  OBS     LW2  3.0  2.10    1  220        1        2
```

For other entrance wheel settings, e.g. *POLARIZOR 2*, return to Step 2 and create another SSCD, e.g. *sscd_polar2*.

8.2.4 Freeze the data in a PDS

The SSCD *sscd_polar1* appears to contain data from a regular raster, hence we use a *raster* PDS:

```
CIA> pds_polar1 = get_sscdraster( sscd_polar1 )
```

Reduce the IMAGEs to EXPOSUREs in the usual way:

```
CIA> reduce, pds_polar1
```

8.2.5 Flat-field correction

For the flat-field correction we must use special flat-fields. These are distributed with CIA as IDL savesets and must be restored manually:

```
CIA> help, pds_polar1.pfov, pds_polar1.fltrwhl
<Expression>    FLOAT     =       3.00000
<Expression>    STRING    = 'LW2'

CIA> restore, file='$cia_vers/data/cds/fl_lw2_p1_6.xdr'
```

Now use **corr_flat** to do the flat correction, supplying the restored flat-field image:

```
CIA> corr_flat, pds_polar1, inflat=flat
```

8.2.6 Photometry

Before performing astrometry we should convert the pixel values to mJy in both the EXPOSUREs and the MOSAIC image:

```
CIA> conv_flux, pds_polar1, /image

CIA> raster_scan, pds_polar1
```

Now perform photometry on each reduced EXPOSURE – use **xphot** or any other program of your choice.

```
CIA> xphot, pds_polar1.image[*,*,0]
```

You can also perform photometry on the raster MOSAIC. The results should be consistent with flux measurements on the EXPOSUREs.

8.3 Calculate Stoke parameters

At this stage you need to have a calibrated PDS for *each* of the polarizors and photometric measurements for the source as seen through each polarizor. To calculate the Stokes parameters we first need to calculate the polar weights with **get_polar_weights**:

```
CIA> fltrwhli = convert_wheel_back('FLTRWHL'+pds_polar1.channel, $
CIA> pds_polar1.fltrwhl)

CIA> get_polar_weight, fltrwhli, pds_polar1.pfov, w1, w2, w3, dw1, dw2, dw3
 not exact matching for pfov, I take the first complete set of data for
 polarizors for the pfov
     360

CIA> print, w1, w2, w3, dw1, dw2, dw3
     0.986200     0.993700      1.00000   0.00100000   0.00100000      0.00000
```

Now we supply the MOSAIC images for each of the three polarizors, along with their associated RMS images and the polar weights to the routine **comp_stokes** (note that you may also supply flux measurements):

```
comp_stokes, roll=pds_polar1.angle_raster, $
  pds.polar1.raster, $
  pds.polar1.rmsraster, $
  pds.polar2.raster, $
  pds.polar2.rmsraster, $
  pds.polar3.raster, $
  pds.polar3.rmsraster, $
  w1, w2, w3, dw1, dw2, dw3, $
  stokes_i, sigma_i, stokes_q, sigma_q, stokes_u, sigma_u, $
  polar_rate, rmspolar_rate, $
  polar_angle, polar_angle_north, rmspolar_angle
```

The last three lines of the above command contains the required output. For further details on the meaning of these parameters see the online help on **comp_stokes**.

Part II

CIA Basic Guide

Introduction

The purpose of Part II: *CIA Basic Guide* is to allow you to view reduced images from the Auto Analysis processing of your ISOCAM data and make a first attempt at doing your own analyses. No previous knowledge of Cam Interactive Analysis is needed to use the *CIA Basic Guide*.

- **Chapter 9** introduces you to your data products and how they are managed in CIA.
- **Chapter 10** introduces you to a CIA session and shows you how to display your Auto-Analysis Results data.
- **Chapter 11** overviews the CIA analysis process.
- **Chapter 12** describes the first step in the analysis process: data slicing.
- **Chapter 13** introduces you to the data calibration routines of CIA.
- **Chapter 14** describes CIA's image analysis and display routines.

Chapter 9

The data products and CIA data structures

In this chapter we will look at how the data are stored in the data products as FITS files, how these data products are named and how they relate to CIA Data Structures (see Part III for a more detailed account of the CIA Data Structures).

9.1 Data product filename convention

Observers who have not obtained their data via IDA will have to deal with the filename convention that has been applied to the data product files on the ISO CD-ROM. This is due to a community requirement that the length of the filenames comply with the DOS convention of a maximum of eight characters, with a three character extension. The file *datalist.txt* (see Section C.2.2) lists the *official* names of the files on the CD-ROM along with the actual abbreviated name used. Note that in the *CIA User's Manual*, so-called observation data product files (found in the CD-ROM directory */products/pmmmmmmm/nnnxxxyy*) are always referred to by their filename root, even though the *official* name is the root with *nnnxxxyy* appended to it, e.g. CIER refers to CIER*nnnxxxyy*.

9.2 Data products as FITS files

All ISO data products, whether retrieved from IDA or delivered on an ISO CD-ROM, are FITS files. The features of these FITS file are described in the *ISOCAM Handbook*[1]. The *ISO Data Product Document* lists the header and binary table keywords for all ISO files. From the table of contents of the *ISO Data Product Document* you can see that the files are grouped in a particular way. Section 9.3 will help you understand the grouping and find the FITS file details you want in the *ISO Data Product Document*.

9.3 Relating data product types to filenames

You can consider the data products to be organised into five groups. Raw Data, Standard Processed Data, Automatic Analysis Results, Calibration Data and Auxiliary Data. Further

[1]Also a good reference is Harten R.H., 1988, *The FITS tables extension*, A&A Suppl. Ser. 73, 365-372.

explanation of these data types and the files where actual data may be found follows. This may not be an exhaustive review[2] of the data products but should suffice at this stage.

9.3.1 Raw data products

The ISOCAM raw data products consist of CAM Compact Status Data and CAM Edited Raw Data.

Compact Status Data (CSTA) Data on the status of ISOCAM during the observation. It is delivered in the CSTA file. ISO CD-ROM users can find this file in the directory */products/pmmmmmmm/nnnxxxyy*. Though used by some routines[3] these data are not essential.

CAM Edited Raw Data (CIER) ERD refers to raw CAM data. These are all the data from and associated with an observation, including internal calibration and housekeeping data. These data are delivered to you in the CIER file[4]. ISO CD-ROM users can find this file in the directory */products/pmmmmmmm/nnnxxxyy*.

9.3.2 Standard Processed Data (SPD)

SPD are data which have had some processing. Housekeeping data has been removed and IMAGEs are computed from the RESET and End-Of-Integration (EOI) FRAMEs that are present in CIER. These data are delivered in the CISP file[5]. ISO CD-ROM users can find this file in the directory */products/pmmmmmmm/nnnxxxyy*.

9.3.3 Automatic Analysis Results (AAR)

The AA processing produces several data products. CCIM, CMAP and CMOS files primarily contain processed images. The remaining files contain *by-products* of the AA, for example CGLL contains a list of glitches detected by AA processing. ISO CD-ROM users can find this file in the directory */products/pmmmmmmm/nnnxxxyy*.

- *Primary* AAR level data products containing ISOCAM images:

  CCIM Contains AA computed EXPOSUREs in detector coordinates.

  CMAP Contains AA computed EXPOSUREs in astronomical coordinates.

  CMOS CAM Mosaic. Contains MOSAICS constructed by AA from EXPOSUREs, contained in the CMAP data product, of the same CONFIGURATION.

- *By-products* of AA processing:

  CUFF CAM User-friendly File. Contains a log of messages from AA processing.

  CGLL CAM Glitch List. Contains a list of AA detected glitches.

  CJAM CAM Jitter, Memory and Stabilisation information.

  CPSL Contains a catalogue of AA detected point sources.

  CSSP Contains the measured spectrum for each point source detection. Used in multi-filter observations only.

[2] Refer to the *ISOCAM Handbook* and the *ISO Data Product Document* for a comprehensive account.
[3] Currently **x_slicer** (see Section 12.3) uses the CSTA data product though it will work without it.
[4] CAM parallel ERD data are delivered in the CPER file and CAM Diagnostic ERD data in the CDER file.
[5] CAM parallel SPD data are delivered in CPSP.

9.3.4 Auxiliary data products

Additional information, such as pointing and spacecraft position and velocity, are also provided on the CD. ISO CD-ROM users can find IRPH and IIPH in the directory */products/-pmmmmmmm/nnnxxxyy* and ORBIT in */products/pmmmmmmm/others*.

IIPH Instrument Instantaneous Pointing History. Contains instantaneous pointing information for CAM during your AOT. Note that this is an essential file for calibrating your data. Users have experienced difficulty with very old IIPH files, i.e. generated by OLP prior to version 4.0, see Section E.2 for details.

The IIPH equivalent for CAM parallel mode is CIPH.

IRPH Instrument Reference Pointing History. Contains reference pointing information for CAM during an AOT. It contains similar, though less comprehensive information as the IIPH. CIA uses the IRPH when analysing CAM parallel mode data.

The IRPH equivalent for CAM parallel mode is CRPH.

ORBIT Contains information on orbital parameters for all ISO revolutions up to and at the very least including the revolution during which your AOT is performed.

Strictly speaking this is a CAL-G file, but to follow the organisation of the *ISO Data Product Document* it is placed in this section.

9.3.5 Calibration Data Products

Calibration data are available in two formats, CIA's Calibration Data Structures (CDS) and CAL-G FITS files.

9.3.5.1 CIA's Calibration Data Structures

All the necessary calibration data are delivered with CIA in the form of CDSs (see Section 9.4.3 and Section 15.3), with notifications of updates available by email. Access to the CDSs is usually handled by CIA routines and so is mainly transparent to the user. For CIA users this is the best way to deal with calibration data. It is certainly the most convenient and has the added advantage that CDSs will contain the most up-to-date calibration data.

9.3.5.2 CAL-G FITS files

Alternatively, calibration data in the form of CAL-G FITS files can be sourced from an ISO CD-ROM or the IDA at

`http://www.iso.vilspa.esa.es/`

Of course the ISO CD-ROM files are current when the CD is pressed while the archives will always be up-to-date. Note however that CAL-G files from both these sources are official ESA releases of calibration data. There will usually be some time lag between the release of ESA official CAL-G files and the ISOCAM consortium official CDSs that are part of CIA. Again, another reason why CIA users generally only use CIA's CDSs.

On the ISO CD-ROM the CAL-G files are found in the directory */products/pmmmmmmm-/others* and in subdirectories below this. This should be clear from *datalist.txt*. Generally, the names of the CAL-G files begin with *CCG**, where * refers to the either the SW or LW detector.

These files contain dark current exposure, detector flat image, optical flat image, dead pixel map and PSF libraries. A listing of the files is provided below and full details can be found in the *ISOCAM Handbook*, Chapter 3, and Section 2.3.5. The files are listed here by their *official* name, again *datalist.txt* will help you associate these names with the files on the CD-ROM (see Section 9.1).

- *Basic* calibration files:

File	Description
CSCGCROSS	CAM SW noise/cross talk decorrelation matrices
CHCGCONV	house keeping interpolation values
CCG*DEAD	CAM * dead pixel map
CCG*DARK	CAM * dark current exposure
CCG*DFLT	CAM * detector flat field library
CCG*OFLT	CAM * optical flat field library
CCG*SPEC	CAM * filter & CVF spectral data
CSWCVF	CAM SW CVF description
CLWCVF1	CAM LW CVF1 description
CLWCVF2	CAM LW CVF2 description
CCG*SLP	CAM * CVF spectral line profile
CCG*PSF	CAM * point spread function library
IFPG	focal plane geometry
CWHEELS	CAM wheels information
ORBIT	ISO orbital parameters

($* =$ *SW* or *LW*.)

- *Higher level* calibration libraries:

File	Description of contents
CCGLWDMOD	CAM LW parameters for the time dependent dark corrections
CCG*TRANS	CAM * model transients
CCG*LINEAR	CAM * linearity correction library
CCG*FRAME	CAM * detector astrometric calibration
CCG*GLITCH	CAM * glitch model
CCG*STRAY	CAM * non-dark local light model

($* =$ *SW* or *LW*.)

9.4 Relating Data Product Types to CIA Data Structures.

In Section 9.3 we have described the data products, or files, on the CD-ROM and and how the data products relate to the data product types. Here we shall discuss how the data product types relate to the data structures employed by CIA.

9.4.1 What is a CIA Data Structure?

The CIA Data Structures have been specially designed to manage ISOCAM data within a CIA session. There are several types of structures, each of which are used to hold a specific type of data product, or data from different stages of CIA processing.

The CIA data structures are quite complex structures, containing many data. These include images (FRAMEs, EXPOSUREs and MOSAICs) and CAM parameters (CVF position, RA and DEC of images, etc...). Where possible the data are presented in fields of the structure in a user-friendly format – telemetry coded parameters have been converted to string information by CIA. With the exception of PDSs (see Section 9.4.4) these structures are not regular IDL structures. Unfortunately, due to IDL memory management restrictions these structures do not have a user-friendly user interface and may well be the most difficult hurdles for CIA novices. You *can't* do something like[6]:

```
CIA> help, cia_struct, /str
```

However, they do offer great flexibility. They can handle data from the most 'exotic' of observations (observations with telemetry drop-outs etc...) and allow the advanced CIA user the possibility to perform a more sophisticated calibration. Most users will only briefly encounter CIA data structures as a means to prepare, or slice, their data before placing the data in the more user-friendly PDS and proceeding with calibration.

CIA data structures are accessed via IDL handles. However, CIA does provide routines that make their manipulation transparent, so for the most part you never need to dig deeply into the nature of these structures. A similar suite of such routines exists for each data structure. For example, you may use such routines as **scd_get** to extract data from an SCD, **sad_get** to extract data from an SAD and so on. Part III serves as a reference guide to these data structures and manipulation routines. It also intended to be a companion to the *CIA Basic Guide* and to help you should you have difficulty with understanding references to CIA data structures that appear here.

We will broadly group the CIA structures into those containing observation data (see Section 9.4.2) and those that contain calibration data (see Section 9.4.3). For a more detailed look at these structures see Chapter 15.

9.4.2 Structures containing Observation Data

These structures are used to contain actual images from an observation and data about an observation.

Science CAM Data (SCD) – ERD/SPD Level The SCD structure has two flavours, one is used to hold data of ERD product type, the *ERD* SCD, and the other of SPD product type, the *SPD* SCD. The *ERD* SCD contains all the EOI and RESET FRAMEs from a single STATE and in addition parameters describing that STATE, e.g. coordinates, lens, filter, etc.... The *SPD* SCD differs primarily in that it holds IMAGEs which have been computed from the EOI and RESET FRAMEs. These IMAGEs are either directly taken from the CISP data product or computed by CIA from the FRAMEs in the *ERD* SCD.

Set of Science CAM Data (SSCD) The SSCD is primarily designed to catalogue a set of SCDs (either *ERD* SCDs or *SPD* SCDs, but not both together) belonging to the same CONFIGURATION and variables which describe that CONFIGURATION. However, it may be used to catalogue any number of STATEs: from all the STATEs in a AOT down to a single STATE.

[6]For an indirect way of treating a CIA structure as a regular IDL structure see Section 16.1.2.

Science Analysed Data (SAD) – AAR Level Contains two EXPOSUREs, one calibrated in detector coordinates (from the CCIM data product) and the other calibrated in celestial coordinates (from the CMAP data product). The SAD is also used to hold EXPOSUREs (from the CMOS data product). In addition to AAR data products, the SAD may be used to hold CIA calibrated EXPOSUREs and MOSAICs.

Set of Science Analysed Data (SSAD) Contains a catalogue of a set of SADs. Its function is analogous to that of the SSCD. In general, best use of the SSAD is made when it catalogues SADs that belong to a single CONFIGURATION. However, it may catalogue any subset of SADs from an AOT.

Diagnostic Specific Data (DSD) Contains physical parameters of the camera: temperatures, voltages, wheel positions, etc. It is directly used only by instrument experts for in-depth investigation of ISOCAM behaviour. Such data that is of interest to the normal user is also held in the SCD. Documentation of the DSD structure is beyond the scope of the *CIA User's Manual*[7].

9.4.3 Calibration Data Structure (CDS)

One generic structure is used to contain all of the different types of calibration data. This is the Calibration Data Structure. It contains a dynamic substructure or field named DATA and a standard set of fields. DATA holds the actual calibration data: dark images, flat images, point spread images, etc. The standard fields hold information related to the nature of the CDS structure itself, e.g. the CDS name and size.

The CDS is almost a direct conversion from FITS format data to IDL data structure, the important difference being that the actual image data are scaled, so the BZERO and BSCALE keywords are discarded. So the CDS differs from the SCDs/SADs in that the CAM parameters are NOT presented in the CIA user friendly format, but as raw values taken from the CAL-G FITS files.

9.4.4 Regular IDL structures

Unlike the structures described in Sections 9.4.2 and 9.4.3 the structures described here are regular IDL structures.

Prepared Data Structure (PDS) Currently, three flavours of the PDS exist: a CVF PDS, a raster PDS (also known as a raster data structure) and a general PDS. This data structure is formed from an SSCD (and its *SPD* SCDs) and is used to hold all the sliced (i.e. prepared) data that you need to perform calibration. Usually, the data in a PDS correspond to a single CONFIGURATION.

See Section 15.5 for more on details of PDSs.

[7] Further details on the DSD may be found in the *ISO Data Product Document* or the *CAM Parameter Characteristic Document*, 1991, ref ISOCAM ST110.

Chapter 10

First look at the data

This chapter will introduce to a CIA session and show you how to display AA computed images. Reading the earlier chapters of the *CIA Basic Guide* may be helpful for fully understanding the text here, though not necessary if you are impatient for a glimpse of your images and not worried too much about the details at this time.

10.1 Copying data products from ISO CD-ROM to hard disk

If you are working with ISO CD-ROMs rather than data obtained from the IDA then it is recommended that the data products are copied to hard disk prior to analysis, though if you just intend at this time to view the AAR data with **sad_display** then skip to Section 10.2. Some of the CIA routines strictly require a naming convention for the data products and the filenames of the products on the CD-ROM use only an abbreviated version of the full name *official* name (see Section 9.1). Routines described below will copy your data products to hard disk and give them the correct *official* name.

10.1.1 Copying on VMS

READ_CDROM is a DCL script distributed with CIA. It will automatically copy/rename the CD-ROM data products to disk space[1]. Before using it, ensure that your CD-ROM drive is *not* already mounted and then insert your CD-ROM into the drive. To execute READ_CDROM, change to the directory in which it resides and call it with:

```
W40 @READ_CD.COM
```

Initially, the program prompts you for an (existing) destination directory. It then mounts the CD-ROM drive and reads the file *datalist.txt* directly form your CD.

```
----------------------------------------------------------------------
You'll be asked now where you want your files copied to.
Name of target directory (<RET> = ISOW40$DKA200:[SOTT]): OLP1:[IA.STEPHAN.TMP]
----------------------------------------------------------------------
```

[1]READ_CDROM contains some lines which have to be edited for the particular system you are working on, e.g. the CD-ROM device-name must be inserted. These are indicated at the beginning of the file. It is expected that your CIA administrator will have taken care of this.

```
Temporary mount to read contents of CD-ROM
%MOUNT-I-WRITELOCK, volume is write locked
%MOUNT-I-CDROM_ISO, I0058016:I0058016 (1 of 1) , mounted on  _ISOW40$DKA600:
%COPY-S-COPIED, ISOW40$DKA600:[000000]DATALIST.TXT;1 copied to
OLP1:[IA.STEPHAN.TMP]DLIST.TMP;1 (14 blocks)
```

It then dismounts the drive and prompts for the data you wish to copy – you can choose to copy data by instrument. Type a single letter (*l*,*s*,*c* and *p* for LSW, SWS, CAM and PHOT respectively) for each instrument's data that you want copied. If you are unsure, choose *A* for all. The drive is then re-mounted and the transfer to disk is executed.

```
Now dismounting before final mount to read FITS data
%MOUNT-I-WRITELOCK, volume is write locked
%MOUNT-I-CDROM_ISO, I0058016:I0058016 (1 of 1) , mounted on  _ISOW40$DKA600:
Ready for data transfers...
--------------------------------------------------------------------
    Enter from 1 to 4 characters, one per instrument you want to copy,
    for instance LS for LWS and SWS. Alternatively, you may copy all
    data (example LPC, or CLPS; enter just A for all data): a
--------------------------------------------------------------------
--------------------------------------------
[Product sets]
--------------------------------------------
--------------------------------------------
Product set NR: P0087982, Proposer ID : LMETCALF
--------------------------------------------
==> Product: aocs58302914
 =======> Creating aocs58302914.fits on OLP1:[IA.STEPHAN.TMP]
%COPY-S-COPIED, ISOW40$DKA600:[PRODUCTS.P0087982.58302914]AOCS.FIT;1 copied to
OLP1:[IA.STEPHAN.TMP]AOCS58302914.FITS;1 (1350 blocks
)
==> Product: eoha583
 =======> Creating eoha583.fits on OLP1:[IA.STEPHAN.TMP]

...

==> Product: psta58303015
 =======> Creating psta58303015.fits on OLP1:[IA.STEPHAN.TMP]
%COPY-S-COPIED, ISOW40$DKA600:[PRODUCTS.P0087984.58303015]PSTA.FIT;1 copied to
OLP1:[IA.STEPHAN.TMP]PSTA58303015.FITS;1 (34 blocks)
DATALIST.TXT finds 86 FITS files
```

10.1.2 Copying on UNIX

fitsname is a general purpose routine for managing FITS data products[2] in the UNIX environment. To copy/rename all the data products from an ISO CD-ROM to the directory ˜*/products*:

```
CIA> fitsname,dir='/cdrom',dest='~/products',/copy
```

Alternatively, you can create links – each link will have the *official* name – from your hard disk to the data products on the CD-ROM. In this case invoke **fitsname** without any keywords:

```
CIA> fitsname,dir='/cdrom',dest='~/products'
```

Note that if you are accessing a CD-ROM jukebox then you may have to mount your CD-ROM before using **fitsname**[3]. This is simply done by doing an `ls` on the desired CD-ROM, eg.

```
CIA> $ls /jukebox/P038/03802817/
AOCS.FIT*  CIER.FIT*  CSTA.FIT*  IIPH.FIT*
CDER.FIT*  CISP.FIT*  CUFF.FIT*  IRPH.FIT*
```

Now you may execute **fitsname** as described previously.

10.1.3 Manual copying

To manually copy your data products to disk space follow the steps below:

1. Copy the files from */products/pmmmmmmm/nnnxxxyy*[4] to a convenient directory, for example */data*. Now, copy the calibration products from */products/pmmmmmmm/nnnxxxyy/others* to */data/others*. Note that these directories are only examples and no convention exists for the actual directory names.

2. Look in the file *datalist.txt* for the *official* names of the files. Rename the files accordingly. Mostly this involves appending *nnnxxxyy* to the root, e.g. *cier.fit* to *cier14300601.fit*. Finally, rename all the files so as to have the extension *.fits* rather then *.fit* .

10.2 Examining the AAR Data Products

This section shows you some basic things you can do with your AAR data in a CIA session.

The most useful CIA routine for displaying the AAR products is **sad_display**. Simply type,

```
CIA> sad_display, windows=1
```

to invoke it. A pop-up window will appear requesting a CCIM[5] file. After some processing time – during which SADs (see Section 9.4.2) are created in memory from the CMAP, CCIM and CMOS files – a widget will appear with an image window and a control panel (see Figure 10.1).

[2] In fact will work with any FITS file that contains the key FILENAME in its primary header.

[3] **fitsname** uses a unix `find` program to compile lists of files – you may not be allowed to execute `find` on your jukebox.

[4] Note that UNIX notation is used in the above instructions. In VMS, */products/pmmmmmmm/nnnxxxyy* would be something like *DKA600:[000000.PRODUCTS.PMMMMMMM.NNNXXXYY]*.

[5] Found on the ISO CD-ROM in the directory */products/pmmmmmmm/nnnxxxyy*. See Section 9.3.3.

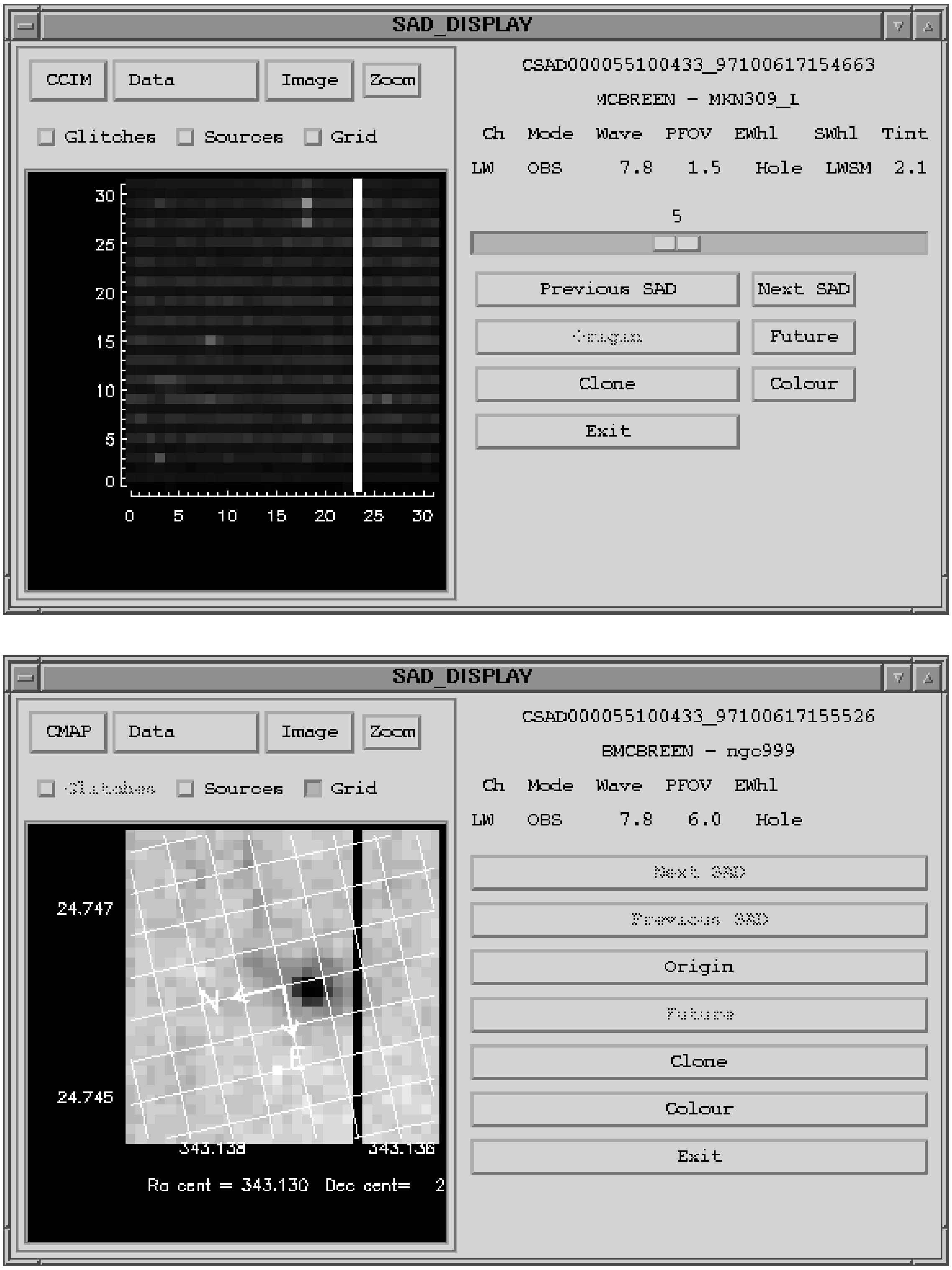

Figure 10.1: **sad_display** windows. The upper window displays an EXPOSURE from an *origin* SAD. The lower window displays the MOSAIC from a *future* SAD. A grid indicating image orientation and coordinates is overlayed on the MOSAIC.

All the images that you will now view with **sad_display** are held in SADs within IDL's memory. These images are either EXPOSUREs or MOSAICs: the former is taken from the CMAP file and the latter from the CMOS file. Within CIA, *origin* SADs hold EXPOSUREs and *future* SADs hold MOSAICs. If you think that a MOSAIC is built from a set of EXPOSUREs then this terminolgy makes more sense.

Now try browsing through your data. Initially, the **sad_display** image window will display an EXPOSURE from the CMAP.

- You can flick through the EXPOSUREs in your AOT by clicking on the button *Next SAD* or by using the slider.
- You can also display a MOSAIC from the CMOS file by clicking on the *future* button. After the *future* button is selected, clicking on the button *Next SAD* allows you to flick through all the MOSAICs in the CMOS.
- Clicking on the button *colour* will load the colour table.
- By clicking on the button *Grid* a grid is displayed indicating the astronomical coordinates of the current image.
- Glitches and point sources detected by AA may be displayed by clicking on the buttons *Glitches* and *Sources*. These will be marked on the displayed image. Note that this functionality will only be activated when the keyword */all* is set when invoking **sad_display**.
- Buttons at the top of the window allow you to toggle between the CCIM and CMAP EXPOSUREs, zoom the image in the window, and toggle between the data image of the EXPOSURE, its RMS error and an image representing the integration time per pixel in the EXPOSURE (also called the weight of the EXPOSURE).
- Clicking on the zoom buttons activates mouse tracking to display pixel values with coordinates.

Pop-up windows with error messages may appear when you try to attempt an undefined action (you may regard these as more warning messages than errors) but just hit *O.K.* and ignore them.

Chapter 11

Introduction to CIA data analysis

This chapter provides an overview of the CIA data analyses processes *vis-à-vis* data structures and data products.

11.1 CIA Processing Overview

Figure 11.1 provides an overview of the general analysis procedure you will follow to reduce your data products to presentable images. The three processing steps are data preparation, data calibration and image analysis & display – subsequent chapters describe each of these steps. Note that the level of processing depends on the data products you begin with, e.g. you don't need to perform data calibration on the images in the AAR data products. Also remember that CIA data structures are the kernel of CIA processing. Each processing step is performed on a particular data structure and the structures evolve as the data are further reduced. Hopefully, all this will become clear as you read on:

1. **Data preparation** (also known in CIA as data slicing) refers to the process of extracting data from the ERD or SPD data products, translating telemetry coded parameters into a user-friendly format, and placing all these data in an *SPD* SCD. There are two paths to producing *SPD* SCDs from data products:

   (a) **Automatic data slicing:**

   - ERD data products are converted into *ERD* SCDs with **erdtoscd** and then the *ERD* SCDs are converted into *SPD* SCDs with **erd2spd**.
   - SPD data products can be directly converted into *SPD* SCDs with **spdtoscd**.

   These methods are referred to as automatic data slicing methods (see Section 12.2).

   (b) **Data slicing with x_slicer:**
   ERD data products can be directly converted into *SPD* SCDs with the widget-based interface **x_slicer**. (see Section 12.3).

   x_slicer requires more effort to use than the automatic slicers, but has greater flexibility and a nice user interface. However, regardless of which way you choose to create *SPD* SCDs, the end result should be the same. See the referenced sections, for the method of your choice.

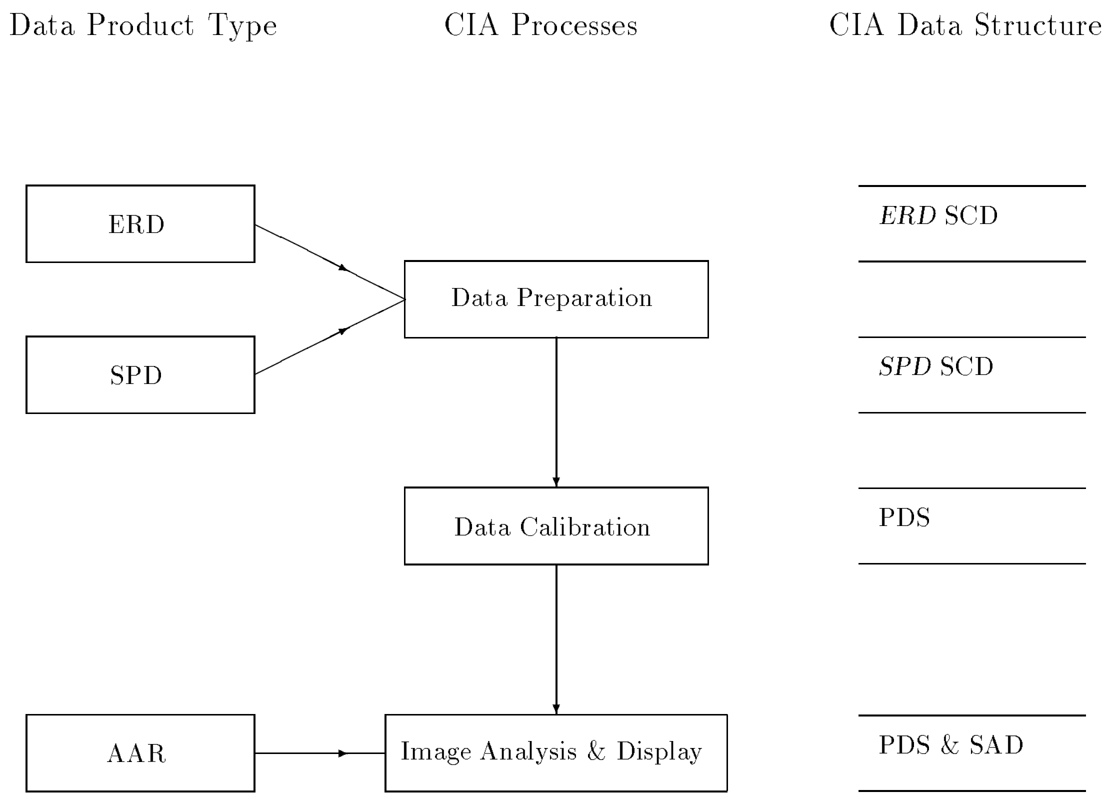

Figure 11.1: Overview of the processing steps in CIA, beginning with either ERD, SPD or AAR data product type. The data structure employed at each level of processing is given in the right-most column.

2. **Data calibration** refers to the process of dark correction, deglitching, stabilisation and flat-fielding of your ISOCAM images. Though this is a highly complex process, CIA handles it transparently via a set of high level routines. The basic process of calibration is:

   (a) After slicing your data you make a PDS from the set of *SPD* SCDs you have created in the slicing process. This is done with one of the CIA routines, **get_sscdraster**, **get_sscdstruct** or **get_sscdcvf**, depending on your observation type.

   (b) You perform the calibration on the PDS with the core calibration routines (see Section 13.2.1). Note that it is possible (and sometimes desirable) to directly perform calibration on the *SPD* SCDs. All the core calibration routines are capable of accepting an SSCD as input.

   Chapter 13 describes in more detail the process of calibration. When your data are calibrated you are ready to move on to the final step of image display.

3. **Image analysis & display** refers to the presentation of your calibrated ISOCAM images. (You can also display the images in the AA calibrated images in the AAR data product directly - this you may have already done in Chapter 10.)

   CIA provides routines to do the following:

   - Display images from the AAR data products with **sad_display**. (See Section 10.2.)
   - Examine the temporal behaviour of pixels with **xcube** and **x3d**. (See Section 14.4.3 and Section 14.4.4 respectively.)
   - Overlay your CAM images with optical images (or indeed any counterpart images) **isocont**. (See Section 14.6.2.)
   - Interactively choose contours for contour plots with **xcontour**. (See Section 14.7.1).
   - Create postscript plots.

Chapter 12

Data slicing

This chapter introduces you to CIA data preparation or data slicing (see Figure 11.1). It follows logically from Chapter 11 and it is assumed that you have read that chapter.

12.1 Data slicing methods

As already stated in the Chapter 11 you have a choice of starting with ERD or SPD data products and as you might expect the data preparation is different for each data product. You may use CIA's conversion routines to automatically create CIA data structures from either data products or you may use the widget-based program **x_slicer** to slice the ERD data. Both these methods are described below in Section 12.2 and Section 12.3 respectively.

12.2 Automatic data slicing

Guidelines and examples of slicing data from different CAM observations can be found in the following sections. In these sections you will be introduced to several new routines, the most important of which are **erdtoscd** and **spdtoscd** – these routines are used to convert ERD data products to *ERD* SCDs and SPD data products to *SPD* SCDs respectively. A general guide to these routines may be found in Chapter 17. All other CIA routines used in automatic slicing are described in Chapter 16. However, it is hoped that from the examples in the following sections the function of these routines should be clear.

12.2.1 General slicing tips

When using the automatic slicers the following steps can be applied to data of all types of observations.

1. Make *ERD* SCDs from the ERD data products with **erdtoscd** and then convert them to *SPD* SCDs with **erd2spd**.

   OR

   Make *SPD* SCDs from the SPD data product with **spdtoscd**.

2. Save a copy of the unsliced *SPD* SSCD and its *SPD* SCDs. They can be discarded when you are finished slicing.

3. At this stage you can choose to use the fast and less flexible **sscd_clean**

   (a) Use **sscd_clean** to split the unsliced SSCD into its component sliced SSCDs, one per CONFIGURATION.

   (b) Save each of the sliced SSCDs.

OR

the slow and flexible **scd_find** and **scd_del** to slice the *SPD* SCDs.

   (a) Identify the *SPD* SCDs which contain STATEs that you don't want to keep, i.e. when CAM is not in OP-MODE OBS. Use **sscd_info** or **scd_find**. Delete the unwanted SCDs with **scd_del**.

   (b) If more than one CONFIGURATION exists in the AOT, identify STATEs of a configuration you want to keep. Delete the SCDs corresponding to the remaining STATEs. Again, use **sscd_info** or **scd_find** to help you do this.

   (c) Save your sliced *SPD* SCDs by saving their SSCD with **sscd_write**, though in a different directory to where the original SCDs are stored. (You don't want to overwrite the old SSCD with the modified SSCD.)

   (d) Restore the original SSCD and slice another CONFIGURATION (if there is one).

12.2.2 Slicing a raster observation (AOT#1)

An example procedure which illustrates the use of both automatic slicers follows: begin the procedure at Step 5 if you are only interested in slicing SPD data products and skip that step if you are only interested in slicing ERD data products.

1. Start a CIA session.

2. Assign useful directory paths to IDL variables:

   In VMS...

```
CIA> product_dir = 'DKA200:[MDELANEY.PRODUCTS]'
CIA> scd_dir = 'DKA200:[MDELANEY.SCDS]'
CIA> sad_dir = 'DKA200:[MDELANEY.SADS]'
```

   ...and in UNIX...

```
CIA> product_dir = '/home/mdelaney/products/'
CIA> scd_dir = '/home/mdelaney/scds/'
CIA> sad_dir = '/home/mdelaney/sads/'
```

3. Convert the ERD data products on disk to *ERD* SCDs in CIA. Use the CIA routine **erdtoscd** to perform this conversion and automatically save the *ERD* SCDs to disk.

```
CIA> erdtoscd, 'cier14300601.fits', erd_sscd, dir=product_dir, $
CIA> scd_dat=scd_dir, ack=ack
```

The *ERD* SCDs are both in memory and saved on disk. This is useful if you wish to end your CIA session – with **sscd_read** you can easily recover the *ERD* SCDs from disk in a later session.

Let's take a look at the contents of memory as follows:

```
CIA> print, sscd_list()
CSSC143006010001_96082811465989

CIA> print, erd_sscd
CSSC143006010001_96082811465989
```

As expected the only SSCD in memory is *erd_sscd*. Now take a look at what SCDs are in memory.

```
CIA> print, scd_list()
CSCD143006010001_96082811470062 CSCD143006010002_96082811473689
etc...

CIA> print, sscd_elem( erd_sscd )
CSCD143006010001_96082811470062 CSCD143006010002_96082811473689
etc...
```

Again as expected the only SCDs in memory are those that correspond to *erd_sscd*.

For convenience, assign the names of the SCDs in memory to an IDL variable.

```
CIA> erd_scds = sscd_elem( erd_sscd )
```

4. Remember that *erd_scds* is an array of names of *ERD* SCDs, i.e. SCDs which contain EOI and RESET frames.

   ```
   CIA> help, scd_get( 'eoi', erd_scds[0] )
   <Expression> INT = Array(32, 32)
   ```

   This means that we must convert them to *SPD* SCDs using **erd2spd**. There are two ways of doing this, the first uses the routine **frames_to_image** and the second **erd2psd** .

   (a) Using **frames_to_image** is probably the simplest way to convert *ERD* SCDs to *SPD* SCDs...

   ```
   CIA> spd_sscd = frames_to_image( erd_sscd, ack=ack )
   ```

   Since a new *SPD* SSCD, *spd_sscd*, is created we can delete the old *ERD* SSCD and its SCDs.

   ```
   CIA> sscd_del, erd_sscd
   ```

   (b) Find the number of *ERD* SCDs that we have created:

```
CIA> nscd = sscd_get( 'nscd', erd_sscd )

CIA> print, nscd
      41
```

Using a simple IDL loop we can convert all the *ERD* SCDs to *SPD* SCDs in one line:

```
CIA> spd_scds = strarr( nscd )

CIA> for i = 0, nscd - 1 do spd_scds[i] = erd2spd( erd_scds[i], $
CIA> ssid=erd_sscd, /del)
```

Note that the *del* keyword is set to delete the *ERD* SCDs from memory as the *SPD* SCDs are created.

Finally, for convenience create a new variable containing the name of the *SPD* SSCD. Since the name of SSCD is unchanged this is achieved by copying the variable *erd_sscd*:

```
CIA> spd_sscd = erd_sscd
```

Now we can remove the old *ERD* SSCD and its *ERD* SCDs from disk. We can use **sscd_remove** to do just that.

```
CIA> sscd_remove, erd_sscd, dir=scd_dir
```

Finally we can save the *SPD* SSCD with its new *SPD* SCDs.

```
CIA> sscd_write, spd_sscd, dir=scd_dir
```

5. As an alternative to the above steps using **spdtoscd** we can directly create *SPD* SCDs in memory (and on disk) with the SPD data product:

```
CIA> spdtoscd, 'cisp14300601.fits', spd_sscd, dir=product_dir, $
CIA> scd_dat=scd_dir, ack=ack
```

6. Our next step is to find the data we require from the SSCD. The whole AOT is contained within the SSCD, but we may be only interested in a single CONFIGURATION. In our example here, we have a raster observation with two CONFIGURATIONs, one for filter-wheel LW6 and one for LW3, and in addition several STATEs corresponding to internal calibrations and detector stabilisation.

   We can get an overview of the contents of the SSCD with **sscd_info** – it lists important parameters of all the SCDs in an SSCD. (These parameters are simply values extracted from fields within the SCD structure.) The contents of the SCDs are in fact a history of the observation. Reading down through the list below you can see the behaviour of CAM through out the AOT. In particular, look out for which filter (the field FLTRWHL) and which OP-MODE (the field MODE) is given for each SCD. Remember, in this observation the filter defines the difference between the two CONFIGURATIONs.

```
CIA> sscd_info, spd_sscd, /deg
          41 SCDs in the SSCD: CSSC143006010101_98060117121857
 seq channel mode fltrwhl pfov tint gain offset size    ra       dec
   0  LW IDLE     LW2  6.0  2.10   1    512    1 ******* *******
   1  LW IDLE     LW2  6.0  2.10   2    512    1 ******* *******
   2  LW  OBS     LW3  6.0  5.04   2    512   31 161.244  55.967
   3  LW  OBS     LW3  6.0  5.04   2    512   20 161.282  55.983
   4  LW  OBS     LW3  6.0  5.04   2    512   20 161.319  56.000
   5  LW  OBS     LW3  6.0  5.04   2    512   20 161.356  56.016
   6  LW  OBS     LW3  6.0  5.04   2    512   20 161.386  55.995
   7  LW  OBS     LW3  6.0  5.04   2    512   20 161.349  55.979
   8  LW  OBS     LW3  6.0  5.04   2    512   20 161.311  55.962
   9  LW  OBS     LW3  6.0  5.04   2    512   19 161.274  55.946
  10  LW  OBS     LW3  6.0  5.04   2    512   20 161.303  55.925
  11  LW  OBS     LW3  6.0  5.04   2    512   20 161.341  55.941
  12  LW  OBS     LW3  6.0  5.04   2    512   20 161.378  55.958
  13  LW  OBS     LW3  6.0  5.04   2    512   20 161.415  55.974
  14  LW  OBS     LW3  6.0  5.04   2    512   20 161.445  55.953
  15  LW  OBS     LW3  6.0  5.04   2    512   19 161.407  55.937
  16  LW  OBS     LW3  6.0  5.04   2    512   20 161.370  55.920
  17  LW  OBS     LW3  6.0  5.04   2    512   19 161.333  55.904
  18  LW IDLE     LW3  6.0  5.04   2    512    3 161.333  55.904
  19  LW IDLE     LW3  6.0  5.04   2    512    3 ******* *******
  20  LW IDLE     LW3  6.0  5.04   2    512    4 161.244  55.966
  21  LW  OBS     LW6  6.0  5.04   2    512   51 161.244  55.967
  22  LW  OBS     LW6  6.0  5.04   2    512   20 161.282  55.983
  23  LW  OBS     LW6  6.0  5.04   2    512   20 161.319  56.000
  24  LW  OBS     LW6  6.0  5.04   2    512   20 161.356  56.016
  25  LW  OBS     LW6  6.0  5.04   2    512   19 161.386  55.995
  26  LW  OBS     LW6  6.0  5.04   2    512   20 161.349  55.979
  27  LW  OBS     LW6  6.0  5.04   2    512   20 161.311  55.962
  28  LW  OBS     LW6  6.0  5.04   2    512   20 161.274  55.946
  29  LW  OBS     LW6  6.0  5.04   2    512   20 161.303  55.925
  30  LW  OBS     LW6  6.0  5.04   2    512   20 161.341  55.941
  31  LW  OBS     LW6  6.0  5.04   2    512   19 161.378  55.958
  32  LW  OBS     LW6  6.0  5.04   2    512   20 161.415  55.974
  33  LW  OBS     LW6  6.0  5.04   2    512   20 161.445  55.953
  34  LW  OBS     LW6  6.0  5.04   2    512   20 161.408  55.937
  35  LW  OBS     LW6  6.0  5.04   2    512   20 161.370  55.920
  36  LW  OBS     LW6  6.0  5.04   2    512   19 161.333  55.904
  37  LW IDLE     LW6  6.0  5.04   2    512    2 161.333  55.904
  38  LW FLAT LW DARK  6.0  2.10   2    512    4 161.333  55.904
  39  LW FLAT     LW6  1.5  2.10   2    512  107 161.333  55.904
  40  LW IDLE     LW6  1.5  2.10   2    512   12 161.333  55.904
```

The SCDs numbered from 2 to 17 all have FLTRWHL equal to LW3 and MODE equal to OBS. This means that these SCDs make up the first CONFIGURATION of this observation. The SCDs numbered from 21 to 36 also have FLTRWHL equal to LW6 and MODE equal to OBS. So these SCDs must correspond to the second configuration. The remaining SCDs represent STATEs of CAM between CONFIGURATIONs (e.g SCD number 16 and 17) or CAM acquiring calibration data (e.g SCD number 38 and 39).

At the moment we are only interested in the LW6 CONFIGURATION, so there are many SCDs we need to discard. There are two ways of achieving this. (i) **sscd_clean**[1] will automatically break up an observation into its constituent CONFIGURATIONs – one SSCD per CONFIGURATION. (ii) **scd_find** can be used to search for SCDs we wish to discard and in doing so create an SSCD for the CONFIGURATION we desire.

(a) **sscd_clean** is invoked simply with...

```
CIA> cleaned_sscds = sscd_clean( spd_sscd )
Out of 41 SCDs:
7 are rejected due to mode
6 are rejected due to csh flag
8 are rejected due to qla flag
In total 33 are accepted
 1-Jun-1998 17:27:36.00  SSCD_CLEAN v.2.0
<Splitting SSCD into 3 segments - I>
```

sscd_clean will have split *spd_sscd* into three SSCDs (of course the number of SSCDs returned will vary from observation to observation, just as the number of CONFIGURATIONs do). Two of the SSCDs correspond to the LW3 and LW6 CONFIGURATION and the remaining SSCD contains calibration data.

```
CIA> print, cleaned_sscds
 CSSC143006010001_98060117273666 CSSC143006010002_98060117274484
 CSSC143006010003_98060117275395
```

We will ignore the calibration data (few observer's will have such data) and concentrate on the LW3 and LW6 CONFIGURATION. We will use **sscd_info** to find which SSCD corresponds to which CONFIGURATION.

```
CIA> sscd_info, cleaned_sscds[0], /deg
          16 SCDs in the SSCD: CSSC143006010001_98060117273666
 seq channel mode fltrwhl pfov tint gain offset size    ra       dec
   0  LW  OBS     LW3  6.0  5.04   2    512   31 161.244  55.967
   1  LW  OBS     LW3  6.0  5.04   2    512   20 161.282  55.983
   2  LW  OBS     LW3  6.0  5.04   2    512   20 161.319  56.000
   3  LW  OBS     LW3  6.0  5.04   2    512   20 161.356  56.016
etc...

CIA> sscd_info, cleaned_sscds[1], /deg
          16 SCDs in the SSCD: CSSC143006010002_98060117274484
 seq channel mode fltrwhl pfov tint gain offset size    ra       dec
   0  LW  OBS     LW6  6.0  5.04   2    512   51 161.244  55.967
   1  LW  OBS     LW6  6.0  5.04   2    512   20 161.282  55.983
```

[1]Note that **sscd_clean** does not work with beam-switch data unless the **spdtoscd** keyword */bs* is set (see Section 19.5.1).

```
   2  LW  OBS    LW6  6.0  5.04   2    512   20 161.319  56.000
   3  LW  OBS    LW6  6.0  5.04   2    512   20 161.356  56.016
etc...
```

Now we will store the name of each SSCD in appropriately named variables.

```
CIA> lw3_sscd = cleaned_sscds[0]

CIA> lw6_sscd = cleaned_sscds[1]
```

Now we can discard our original *SPD* SSCD and save our sliced SSCDs.

```
CIA> sscd_write, lw3_sscd, dir=scd_dir

CIA> sscd_write, lw6_sscd, dir=scd_dir

CIA> sscd_remove, spd_sscd, dir=scd_dir
```

(b) We begin by finding those SCDs we don't want with **scd_find**, and then deleting them. We will choose to keep the LW6 CONFIGURATION...

```
CIA> unwanted_scds = scd_find( 'fltrwhl', 'lw6', /NOTFIND )
Searching for FLTRWHL=NOT(LW6)
Found 22 occurrences

CIA> scd_del, unwanted_scds
```

We must delete also SCDs containing calibration data or data accrued when CAM is not in OP-MODE OBS.

```
CIA> unwanted_scds = scd_find( ['mode'], ['obs'], /NOTFIND )
Searching for MODE=NOT(OBS)
Found 3 occurrences

CIA> scd_del, unwanted_scds
```

To avoid confusion it is best at this stage to rename the IDL variable containing the name of our modified SSCD:

```
CIA> lw6_sscd = spd_sscd
```

The SSCD in memory is automatically modified when some of its SCDs are deleted:

```
CIA> print, sscd_get( 'nscd', lw6_sscd )
      16
```

The final step is to save the modified SSCD that we have created. It is best to keep this SSCD and the sliced *SPD* SCDs separate from the unsliced ones. Create a new directory to hold the sliced data and for convenience create the following IDL string variables to hold their names.

```
CIA> lw6_scd_dir = 'DKA200:[MDELANEY.14300601.scds.lw6]'
```

OR

```
CIA> lw6_scd_dir = '/home/mdelaney/14300601/scds/lw6'
```

```
CIA> sscd_write, lw6_sscd, dir=lw6_scd_dir
```

To slice the LW3 CONFIGURATION we would reload the *SPD* SCDs.

```
CIA> spd_sscd = sscd_read( spd_sscd, dir=scd_dir )
```

And then use **scd_find** to delete all SCDs not corresponding to the LW3 CONFIGURATION and save the remaining LW3 SSCD and SCDs in their own subdirectory. When you are completely finished slicing you can delete the original unsliced SSCD from disk.

```
CIA> sscd_remove, spd_sscd, dir=scd_dir
```

12.2.3 Slicing a CVF observation (AOT#4)

An example of automatic slicing of a CVF observation is given in this section. This example is not so thorough as that of Section 12.2.2, since in principle, slicing is similar for data products of all observation types. Note that this is a relatively simple example – for a more complex CVF observation comprising of multiple segments see Section 19.6. Here, the slicing process begins at SPD level with the CISP file.

1. Start a CIA session.
2. Assign the IDL variables *path* and *scds_path* with the directory holding the data products and the destination directory for the SCDs (example is given for both UNIX and VMS).

```
CIA> path = 'DKA200:[MDELANEY.CVF_OBS]'
CIA> scd_dir = 'DKA200:[MDELANEY.CVF_OBS.SCDS]'
```

OR

```
CIA> path = '/home/mdelaney/cvf_obs'
CIA> scd_dir = '/home/mdelaney/cvf_obs/scds'
```

3. Both these directory paths and the IDL variable *cvf_sscd* are passed to **spdtoscd**:

```
CIA> spdtoscd, 'cisp20305604.fits', cvf_sscd, dir=path, $
CIA> scd_dat=scd_dir, ack=ack
```

The *SPD* SCDs have now been created and the name of their SSCD is held in *cvf_sscd*.

```
CIA> print, cvf_sscd
CSSC203056040001_96091918462427
```

4. To get an overview of the SCDs use **sscd_info**:

```
CIA> sscd_info, cvf_sscd
          77 SCDs in the SSCD: CSSC203056040001_96091918462427
 seq channel mode fltrwhl pfov tint gain offset size    ra       dec
   0       LW  OBS LW-CVF2  3.0  2.10   1    512   99 191233.30 +673941.59
   1       LW  OBS LW-CVF2  3.0  2.10   1    512   21 191233.30 +673941.59
   2       LW  OBS LW-CVF2  3.0  2.10   1    512   20 191233.30 +673941.56
   3       LW  OBS LW-CVF2  3.0  2.10   1    512   20 191233.32 +673941.59
   4       LW  OBS LW-CVF2  3.0  2.10   1    512   21 191233.30 +673941.64
   5       LW  OBS LW-CVF2  3.0  2.10   1    512   20 191233.31 +673941.62

                 ⋮                  ⋮                  ⋮
                 ⋮                  ⋮                  ⋮

  72       LW  OBS LW-CVF2  3.0  2.10   1    512   20 191233.30 +673941.62
  72       LW  OBS LW-CVF2  3.0  2.10   1    512   20 191233.30 +673941.62
  73       LW  OBS LW-CVF2  3.0  2.10   1    512   20 191233.30 +673941.59
  74       LW  OBS LW-CVF2  3.0  2.10   1    512   21 191233.31 +673941.64
  75       LW  OBS LW-CVF2  3.0  2.10   1    512   21 191233.31 +673941.62
  76       LW IDLE LW-CVF2  3.0  2.10   1    512    4 191233.32 +673941.51
```

5. So, from the above you can see that there are 76 STATEs of CAM under OP-MODE OBS and the final STATE under OP-MODE IDLE (the list has been abbreviated for neatness). This STATE we can discard. This is easily done by putting all the SCDs in an IDL string array, *cvf_scds* and then deleting the guilty IDLE STATE.

```
CIA> cvf_scds = sscd_elem( cvf_sscd )

CIA> scd_del, cvf_scds[76]
```

6. The CAM parameter of most interest in a CVF observation is of course the wavelength of the CVF position in each STATE – in fact this is the only parameter we expect to see changing. We can use **sscd_info** to specifically list this information per SCD, i.e. per STATE:

```
CIA> sscd_info, cvf_sscd, param=['wavelength', 'mode']
19-Sep-1996 19:31:37.00  PARCHECK V.1.0 <SUCCESS - S>
          77 SCDs in the SSCD: CSSC203056040001_96091918462427
 seq wavelength mode
   0       8.78200  OBS
   1       9.00300  OBS
   2       9.11300  OBS
   3       9.22300  OBS
   4       9.33300  OBS
   5       9.44200  OBS

   ⋮         ⋮       ⋮
```

```
70        16.0500   OBS
71        16.1400   OBS
72        16.2400   OBS
73        16.3300   OBS
74        16.4200   OBS
75        16.5200   OBS
```

The sequence number of each STATE and wavelength of the CVF (in microns) is listed along with the OP-MODE.

7. The data are now sliced. We can store our results with **sscd_write**:

```
CIA> sscd_write, cvf_sscd, dir=scd_dir
```

12.3 Data slicing with x_slicer

This section[2] introduces you to the CIA's **x_slicer**. This is a widget-based program which allows you to interactively select your own slicing criteria. The advantage of this way of preparing the data is that you get to see exactly how your observation was performed (including all intermediate states) and you can create an SSCD so it contains the states that you choose. Generally, you will use **x_slicer** to create an SSCD for each configuration of an AOT and then create a PDS from the SSCD. **x_slicer** clearly displays all the states within an AOT and their associated parameters, presenting you with an overview of how the observation had been performed.

12.3.1 Starting x_slicer

In order to produce your first SCDs, all you need is to :

1. Start a CIA session
2. Invoke **x_slicer**.[3]

```
CIA> x_slicer
```

The Figure 12.1 shows the first window of the x_slicer.

The top half of the window displays information about the files that you are slicing, the bottom part allows you to choose the variables according to which you will slice your file and to ask the slicer to perform some actions.

Let's go on for our first slicing. It will be divided into a few steps:

- Load the file to be sliced

[2]Taken from Aussel H., 1996, *ISOCAM Data Preparation with X_slicer v2.1*, Section 2.

[3]If you want to force **x_slicer** to choose the current directory as the default location it searches for data, then type:

```
CIA> x_slicer, /here
```

Alternatively, VMS users may redefine *arc_dat*:

```
# define arc_dat
```

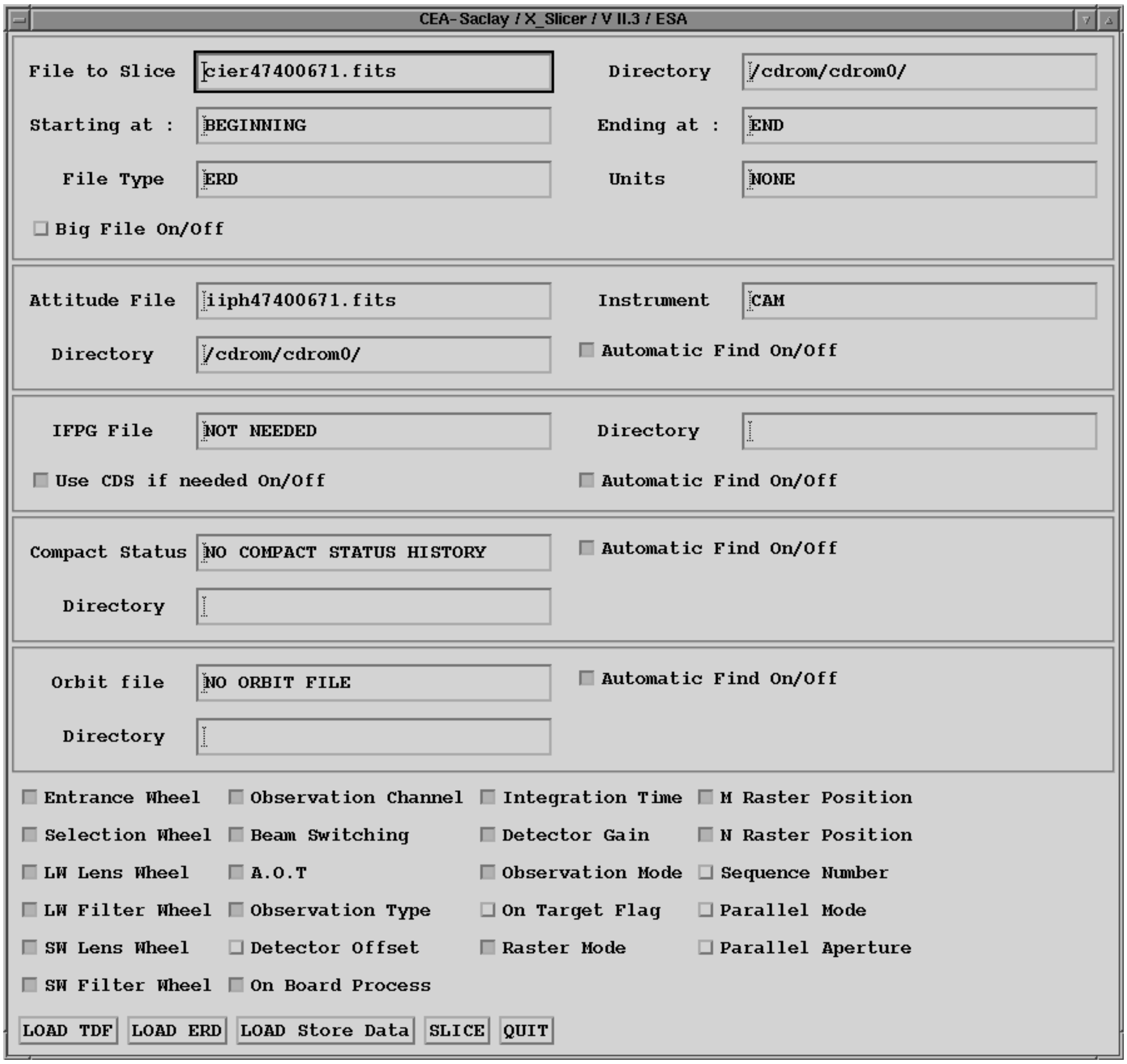

Figure 12.1: **x_slicer** window.

- Select the variables to perform the slicing
- Run the slicer
- Select the SCDs that you really want to build

12.3.2 Selecting a file

The first step to produce our SCDs is to select the file that contains the data. This is done very easily by following these few steps :

- Check that the three 'Automatic Find' buttons are 'On' (pressed). This should be the case, since it is their default configuration. These three buttons tell the **x_slicer** to look for the files it will need at some default places. If you meet problems using these buttons, take a look at Section 19.7.1.
- Check that the 'Use CDS if needed' button in the IFPG file area is 'On' (pressed). This again should be the case, since it is its default configuration. This button allows you to select which IFPG file is to be taken into account, if one is needed (see below).
- Click on the 'Load ERD' button. A pickfile menu will appear. Select your file and click the 'OK' button at the end.

The fields in the top half of the x_slicer window are now filled with information. Let's have a look at it. The first six fields contain information about the file we are going to slice, (*i.e.* its name, directory, type and so on) you can note that for an ERD file, the starting and ending fields are automatically set to 'Beginning' and 'End': we will always slice an entire ERD. This is not the case for a TDF file for example.

Going down, the next fields give the name and directory of the attitude file. This file is the Instantaneous Instrument Pointing History (IIPH) – see Chapter 9. It allows the slicer to compute the astronomical coordinates of each SCD that you will produce. If the 'Automatic Find' was not 'On', or if the slicer couldn't find it automatically, you would have been asked to give the IIPH file corresponding to the ERD with a Pickfile menu. At this point, hitting the 'Cancel' button aborts the search of the IIPH file and the field displays 'No Attitude Control'. This has no effect on the rest of the slicing, but the fields of Right Ascension, Declination and Roll will be left undefined in your SCDs and SSCD: you will be unable to re-project properly your data on the sky after having fully treated them.

A worst case arises when your data are very old: coordinates are given but no information about which instrument they describe appear in the IIPH. You then have two choices: work on your data hoping the coordinates are CAM ones, or wait until your data are reprocessed with a more recent version of the pipeline. If you choose the first case, and find a big discrepancy (*i.e.* more than 30 arcsec) between ISO coordinates and what you asked for, you can be sure that your coordinates are not CAM ones.

In all cases, remember that even if ISO is telling you that it didn't observed where you asked for, it has truly observed it...

The next fields give the name and directory of the Compact STAtus file (CSTA) – again see Chapter 9. With this file, the slicer checks if all commands sent to the camera during your observation were well followed. The same remarks about 'Automatic Find' apply here too.

The last fields of the top part of the window display the name of the orbit file. This file contains the position and speed of the satellite around the Earth. The same remarks about Automatic find apply here too.

12.3.3 Selecting slicing variables

The next step to slice a file is to select the variables according to which the slicer will slice the file. For each record in your file, the slicer will check the chosen variables. If their value has changed, it will flag the record and propose to build a new SCD starting from this record.

You can slice according to the following variables :

- Entrance Wheel: the position of the Entrance Wheel of CAM (hole, polarizers...).
- Selection Wheel: the wheel that reflects the light on the SW or LW channel. WARNING: the selection wheel on SW for example does not imply that the operating channel is SW! (Think about a LW dark measurement for example.)
- LW lens Wheel: the position of the LW lens wheel of CAM.
- LW filter Wheel: the position of the LW filter Wheel of CAM.
- SW lens Wheel: the position of the SW lens wheel of CAM.
- SW filter Wheel: the position of the SW filter wheel of CAM.
- Observation Channel: it tells you if you are observing with the LW or SW detector. WARNING: its value is independent of the selection wheel value.
- Beam Switching: beam-switch mode, flags the cycle number, the reference and the source.
- AOT: the one of the three AOT of CAM that was used.
- Observation Type: AOT, CUS (Calibration Up-link System), or other type of driven observations.
- Detector Offset: No longer useful, since there is only one offset per gain of the detector.
- On Board Process: Normal mode, accumulated mode or sampled mode of CAM detector reading.
- Integration Time: value of the integration time.
- Detector Gain: value of the detector gain (x1, x2 or x4).
- Observation mode: OBS, DARK, WAIT, IDLE, GAP ...
- On target Flag: On if the telescope is on target and the qla_flag is OK (see Section 19.7.7).
- Raster Mode: for raster, micro-scan or staring observations.
- M Raster Position: the position along the first axis of the raster.
- N Raster Position: the position along the second axis of the raster.
- Sequence Number: order number of the SCD in the observation sequence.
- Parallel Mode: CAM is parallel or prime instrument.

We suggest you use all the variables at least once during your first slicing: it will greatly help you to understand how ISOCAM operates.

After some time, you will discover that some variables are not useful to you, and that you can leave them aside, if you remember how you programmed your observations. For example, if you are working on a raster, you won't need the Beam Switching Flag. As a matter of fact, there is only one detector offset per detector gain, therefore the offset is useless. Some other variables may look useless to you, but may be indeed of some interest, especially if you are working on data from the PV phase.

For this particular first slicing, don't use the enhanced OTF in case of telemetry drops. More will be told about this in Section 19.

12.3.4 Run the slicer

The next step is to click on the 'SLICE' button. The Slicer will run on the data you have selected, according to the variables you have selected. Let's see what this means.

The slicer will read your file (the CIER.FIT you have selected), and flag each record where the value of one of the selected variables has changed. It will propose to build a new SCD each time such an event happens. Of course, you don't want to produce all these SCDs, because some of them will be made of frames taken while the camera is adjusting its configuration for the observation that you have requested. One other point is that some observations have been made with different configurations in a single AOT: for example you will find in the same file the same raster on the same sky position done in two different filters. If they are put in the same file, it does not mean that they should be assigned to the same SSCD. All these data manipulations are done on the **x_handle_slice** window that appears in front of you.

12.3.5 The x_handle_slice window.

Figure 12.2 shows the **x_handle_slice** window. It is divided in two parts, each of them is divided in two parts too. Let's go through them.

The top of the window presents the results of the slicing process. On the left, a large table gives you the values of the variables you choose for each SCDs that the **x_slicer** has found (remember that the slicer creates a new SCD each time one of the variables changes). The titles on the top have been shortened, so Table 12.1 presents the conversion.

On the right of this table are two columns of buttons, named SCD # and Merge SCD where # stands for the number of the SCD. Note that by default, all SCDs are selected, and none are merged together. There are scroll bars that allow you to align the row containing the informations about the SCD Number # and the corresponding buttons.

The bottom of the **x_handle_slice** window is made of a set of buttons on the left and a pull-down menu named 'Advanced Slicing' on the right.

The functions of the buttons are:

- Redisplay: Redisplay the **x_handle_slice** window, showing only the selected SCDs. Note that you must redisplay *before* slicing.
- Continue: When satisfied with the slicing, go into the SSCD, SCD and eventually DSD creation...
- Unselect All SCDs: self explanatory.

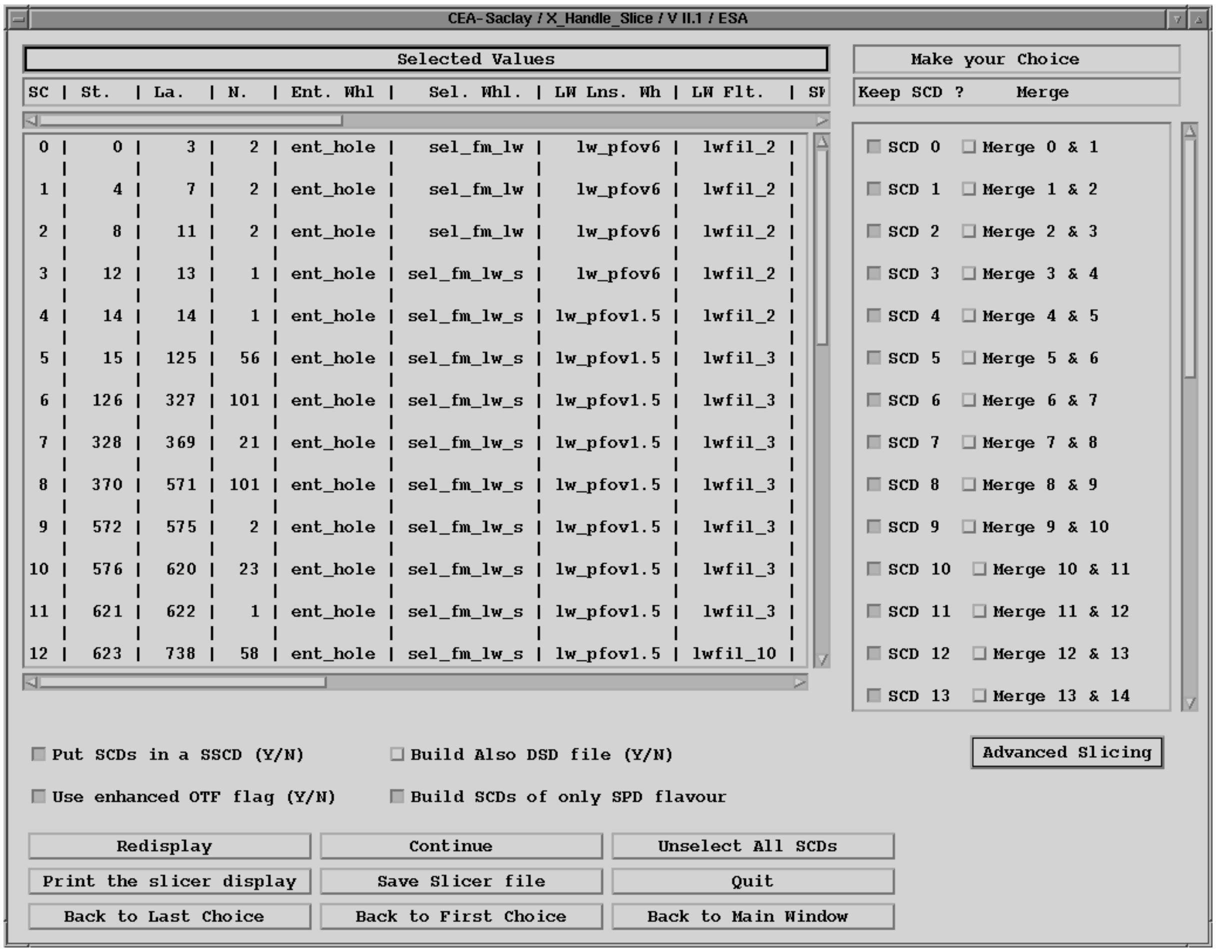

Figure 12.2: **x_handle_slice** window.

title	**description**	**displayed?**
staring SC	SCD Number	always
St.	starting readout of the SCD	always
La.	last readout of the SCD	always
N.	maximum number of images that can be built	always
Ent. Whl	entrance wheel position	if selected
Sel. Whl.	selection wheel position	if selected
LW Lns.	LW lens wheel position	if selected
LW Flt.	LW filter wheel position	if selected
SW Lns.	SW lens wheel position	if selected
SW Flt.	SW filter wheel position	if selected
Ch	channel (which detector is on)	if selected
Beam Switching	Beam Switching	if selected
A.O.T	AOT variable	if selected
Ob. T.	observation type	if selected
Off.	detector offset	if selected
Integ. Ti	integration time	if selected
G.	detector gain	if selected
Obs.	observation mode	if selected
OTF	on target flag	if selected
R.M	raster mode	if selected
M R.P	M raster position	if selected
N R.P	N raster position	if selected
S. N.	sequence number	if selected
P. M.	parallel mode flag	if selected

Table 12.1: Conversion table for the variable names displayed by **x_handle_slice**.

- Print the slicer display: will print the big table in a text file. The name of this file will be the official name of the observation, with the *.prn* extension. For example: *1380302.prn*.
- Save Slicer File: save the work done under a form re-usable by the slicer (see *Handling big datasets* in the *Advanced Slicing* section).
- Quit: Quit **x_slicer**.
- Back to last Choice: will redisplay the state of the slicing operation before you hit the button 'Redisplay'.
- Back to First Choice: will redisplay the first screen that appeared just after the slicing proposal was computed.
- Back to Main Window: will go back to the **x_slicer** window. WARNING: you will lose all the slicing work that has been done, unless it has been saved.

The *Advanced Slicing* pull-down menu contains the list of all lens, filters, gains and integration times used in your data, plus a *Number of exposures* menu.

12.3.6 Selecting SCDs and SSCDs

Let us now begin the real slicing work. Looking at the big board where the proposition of slicing has been displayed, you will soon notice that there are many SCDs that you will not need, especially in the beginning of observations and between two filters or PFOV when more than a configuration was programmed in a single AOT. You now have to select which SCDs you want to create and in which SSCD you want to put them.

Let's examine together a concrete example:

Figure 12.2 displays the **x_handle_slice** for an observation of a cluster. This observation consists of two 4x4 rasters made with the 3 arcsec PFOV lens, one with the LW2 filter, the other with the LW3 filter.

Of course, we are going to make two different SSCDs with these ERD, one for the LW2 filter, the other for the LW3 filter.

Let's begin with the LW2 raster:

- First click on the 'Unselect All SCDs' button. All the 'toggle' buttons on the 'Make your Choice' column are now unselected, and the 'Merge' buttons are insensitive.
- Select the 'lwfil_2' item in the Advanced Slicing/Filters menu. All LW2 SCDs are now selected.
- Click on the 'Redisplay' button.

You are back in the same situation as before making any choices, but the **x_handle_slice** window displays only data obtained with LW2 filter. A way to select the frames of our LW2 observation is to use the number of exposures: all the lens and filters motions take only a few frames, but (in this case), at least 30 frames have been taken per raster position. Here it goes again:

- First click on the 'Unselect All SCDs' button.

- Select the 'Nb Exposure' item in the Advanced Slicing menu. A window appears. Enter the number 30 in the field, since you want to select all SCDs with at least 30 frames. Hit the 'Enter' or 'Return' key. All the wished for SCDs have now been selected.
- Click on the 'Redisplay' button.

If you are not yet satisfied with the displayed SCDs and want to unselect more, you can use other items from the Advanced slicing menu (you just have to remember to unselect all SCDs before, because this menu makes the selection). You can even unselect the last SCDs by hand, using directly the SCDs toggle buttons (but do not try to MERGE SCDs at this point).

Once everything looks good, hit the 'Continue' button.

12.3.7 Choosing names

Once you have clicked on the 'Continue' button a new window will appear, asking you to give the name of the SSCD you want. You can choose to enter a name, for example '_obs_lw2', or to use the official name (in our example it is '180014050000' because the data comes from orbit 180, TDT sequence number 014, OSN 05 and CIA's sequence number 0000). In any case, an SCD name is made of 31 characters, for example an official name would look like

```
CSCD180014050002_96051517052200
```

and an unofficial user defined name would look like

```
CSCD_obs_lw2__02_96051517052200
```

The first four characters are always 'CSCD' (or 'cscd') for an SCD, 'CSSC' (or 'cssc') for an SSCD. The next 10 may be user defined (12 next for an SSCD) or may follow the official convention of: the sequence number (i.e. the position of the SCD within the SSCD), then an underscore, then the creation date in the order year, month, day, hour, minutes, seconds and centiseconds. User defined names that fall short of the full name length will be automatically padded out with underscores.

Keep the following in mind when selecting a name:

- *Do not forget to hit the return key after entering characters.*
- If you press the 'Official' button, you will get the official name for your data.
- If you press 'Cancel' button, you will go back to the **x_handle_slice** window *with your previous choice.*

When you are satisfied press the 'Go' button. **x_slicer** will begin processing and a message will appear when it has finished. You can then press the 'Quit' button of the **x_handle_slice** window to exit **x_slicer**, or continue to process the data remaining in your ERD.

Chapter 13

Data calibration

This chapter introduces you to the process of data calibration with CIA (see Figure 11.1). Through out this chapter calibration will be described as being performed on PDSs, which are of course derived from SCDs. It is worth mentioning at this point that calibration can also be performed directly on SSCDs and SCDs, and indeed more and more users are doing so for reasons described in Section 13.1.1. However, it is probably best for the novice user to initially work with PDSs if it is possible to do so.

Note that the material in this chapter follows closely from Chapter 12.

13.1 Creating a PDS from an SSCD

At this stage it is assumed that the data has been prepared/sliced correctly and placed in a set of *SPD* SCDs (see Figure 11.1). To progress to the calibration process you need to create a PDS from this set of *SPD* SCDs. The PDS[1] can conveniently hold all the data we need for further processing. There are four flavours of PDS to accommodate some differences in the nature of the data from different CAM AOTs and observations: the *raster* PDS, the *CVF* PDS, the *BS* PDS and the *general* PDS.

13.1.1 PDS caveats

Before we proceed there are some important points about PDSs which should be understood. The PDS must only contain data from STATEs where CAM is in OP-MODE OBS. Other STATEs, such as IDLE, will crash CIA processing routines. Also, the PDS must be perfectly filled for calibration to work – telemetry drops, missing raster points, gain change due to saturation, can all contaminate a PDS. A PDS is not suitable for some kinds of data, e.g. data from a polarization observation, or calibration methods where contiguous data is required as input. For instructions on how to calibrate an SSCD see Section 20.3. In addition, Chapter 8 contains an example of calibrating SSCDs containing data from a polarization observation.

If you have used **sscd_clean** to slice your SSCD then telemetry drops will have been patched with fake SCDs and the subsequently derived PDS should be fine. Alternatively, you can continue the data analysis process until the image level by reducing the SSCD to a SSAD. This later option is unavoidable if the SSCD is comprised of data with non-constant t_{int} or PFOV.

[1]See Section 15.5 for a description of the architecture of a PDS.

13.1.2 *raster* PDS

The *raster* PDS is designed to handle sliced data from the following observation types:

- raster (AOT#1)
- micro-scan observation (AOT#1)

To create the *raster* PDS from the sliced M×N *SPD* SCDs (where N and M are the raster dimensions) use **get_sscdraster**. To continue the example of Section 12.2, we assume that the IDL variables *lw6_sscd* and *lw6_scd_dir* contain strings with the name of an SSCD (cataloging the sliced *SPD* SCDs) and the name of the directory path where it is has been saved, respectively. To avoid confusion and save on memory, it is a good idea to delete all SSCDs/SCDs (that may be have been left over from the slicing process) using **scd_del** or **sscd_del**, reload the sliced SSCD you want to calibrate from disk and then proceed to build your PDS:

```
CIA> lw6_sscd = sscd_read('cssc143006010002_98060117274484.cub', dir=scd_dir)

CIA> lw6_raster = get_sscdraster( lw6_sscd )
CCGLWDARK_97031713382678  not exact matching for : GAIN= 1 <>  0;
CCGLWOFLT_98050815090326  not exact matching for : TINT= 15 <>  36;
```

lw6_raster is a *raster* PDS containing all your prepared data. Don't worry too much about any unusual looking messages like the two appearing above. These messages appear when there is slight mismatch between the CONFIGURATION of the best available calibration data and the data in the PDS. Now that you have created a PDS it might be nice to take a look at its architecture (Section 15.5.4 will guide you through its structure.) Since it is simply an ordinary IDL structure as opposed to a CIA data structure, we can use IDL's HELP to look at its innards:

```
CIA> help, lw6_raster, /str
** Structure <13d390>, 54 tags, length=2294560, refs=1:
   RASTERCOL       INT              4
   RASTERLINE      INT              4
   M_STEPCOL       FLOAT          96.0000
   N_STEPLINE      FLOAT          96.0000
   RA_RASTER       DOUBLE           161.34459
   DEC_RASTER      DOUBLE           55.959980
   ANGLE_RASTER    DOUBLE           321.73999
   RASTER_ROTATION DOUBLE           411.73999
   RASTER_ORIENTATION
                   STRING    = 'SPACECRAFT Y_AXIS'
   ASTR            STRUCT    -> ASTR_STRUC Array[1]
   NX_RASTER       INT             80
   NY_RASTER       INT             80
   RASTER          FLOAT     Array[80, 80]
   RASTER_UNIT     STRING    ''
   RMSRASTER       FLOAT     Array[80, 80]
   NPIXRASTER      FLOAT     Array[80, 80]
   AOT             STRING    'RASTER'
   TARGET          STRING    'HAR003  '
```

```
OBSERVER        STRING    'LMETCALF'
TDTOSN          LONG          14300601
CHANNEL         STRING    'LW'
PFOV            FLOAT          6.00000
TINT            FLOAT          5.04018
GAIN            FLOAT          2.00000
FLTRWHL         STRING    'LW6'
WAVELENGTH      FLOAT          7.75000
NSCD            INT            16
NBR_FRAME       INT           348
FROM            INT       Array[16]
TO              INT       Array[16]
TAB_FRAME       INT       Array[16]
ADU_SEC_COEFF   FLOAT     Array[16]
TABFLATCOEF     FLOAT     Array[16]
CUBE            FLOAT     Array[32, 32, 348]
CUBE_UNIT       STRING    'ADU'
MASK            BYTE      Array[32, 32, 348]
OTF             BYTE      Array[348]
DU              FLOAT     Array[348]
DV              FLOAT     Array[348]
UTK             LONG      Array[348]
BOOTTIME        LONG      Array[348]
TEMPERATURE     FLOAT     Array[10, 348]
IMAGE           FLOAT     Array[32, 32, 16]
IMAGE_UNIT      STRING    ''
RMS             FLOAT     Array[32, 32, 16]
NPIX            FLOAT     Array[32, 32, 16]
CCIM            STRUCT    -> <Anonymous> Array[1]
INFO            STRUCT    -> <Anonymous> Array[1]
DARK            FLOAT     Array[32, 32]
FLAT            FLOAT     Array[32, 32]
CALG            STRUCT    -> <Anonymous> Array[1]
HISTORY         STRING    Array[70]
SSCD_NAME       STRING    'CSSC143006010002_98060117274484'
SAD_NAME        STRING    'CSAD143006010204_98060119435439'
```

Before proceeding it is recommended to save this structure using IDL's SAVE (with */xdr* set for portability).

```
CIA> save, filename='lw6_raster.dat', lw6_raster, /xdr
```

13.1.3 *general* PDS

This PDS is for general use and currently accommodates data from the following observations:

- Staring (AOT#1)
- Tracking (AOT#1)

- Polarization (AOT#5)

Again, like other PDSs the *general* PDS is created with **get_sscdstruct** from an SSCD containing sliced *SPD* SCDs. For example,

```
CIA> staring_pds = get_sscdstruct( staring_sscd )
```

where it is assumed that *staring_sscd* is an IDL variable holding the name of the sliced SSCD.
In structure, it is the simplest of all PDSs (see Section 15.5.2 for more on the architecture of the *general* PDS).

13.1.4 *BS* PDS

The *BS* PDS is designed to handle data from a beam-switch observation (AOT#3). Once you have sliced your *SPD* SCDs you can convert them to a *BS* PDS in a similar manner to the other PDS flavours. This time you use the routine **get_sscdbs**. Note that there is one irregularity which sometimes arises in beam-switch observations: some observers have programmed their observations in reverse! Section 19.4 tells you how **get_sscdbs** can be used to deal with this problem.
Assuming that *bs_sscd* is an IDL variable containing the name of your sliced SCDs' SSCD then you can create a *BS* PDS named *bs_pds* with:

```
CIA> bs_pds = get_sscdbs( bs_sscd )
```

13.1.5 *CVF* PDS

The *CVF* PDS is designed to handle data from a CVF observation (AOT#4). In principle, the *CVF* PDS is created like the *raster* PDS of Section 13.1.2, though with the CIA routine **get_sscdcvf**. The following example follows that of Section 12.2.3.
If the sliced *SPD* SCDs are not currently in memory they can be recovered from disk with **sscd_read**:

```
CIA> cvf_sscd = sscd_read( 'CSSC203056040001_96091918462427', dir=scd_dir )
```

The next step is to create the *CVF* PDS from the *SPD* SCDs – the IDL string variable, *cvf_sscd* contains the name of their SSCD:

```
CIA> cvf_pds = get_sscdcvf( cvf_sscd, /del )
```

Note that the keyword *del* is set. CVF observations yield large quantities of data and it is not wise to duplicate it in both an SSCD and a PDS – */del* deletes the SSCD automatically upon creation of the PDS.
As for a *raster* PDS you can look at the structure of a *CVF* PDS with IDL's HELP:

```
CIA> help, cvf_pds, /str
** Structure <ad39580>, 47 tags, length=9171368, refs=1:
   CVF_NAME        STRING    'LW-CVF2      8.78200      0.110000'
   ASTR            STRUCT    -> ASTR_STRUC Array[1]
   ENTWHL          STRING    'HOLE'
   SELWHL          STRING    'LW large Mirror'
   CVF_INCR        INT              4
```

```
WAVELENGTH_START
                FLOAT     =      8.78200
WAVELENGTH_END  FLOAT          16.5200
RESPONSE        STRUCT    -> <Anonymous> Array[76]
SPEC_NAME       STRING    'CCGLWSPEC_98040514330660'
AOT             STRING    'STARING'
TARGET          STRING    'NGC1342'
OBSERVER        STRING    'ANON'
TDTOSN          LONG         5805004
CHANNEL         STRING    'LW'
PFOV            FLOAT          6.00000
TINT            FLOAT          2.10007
GAIN            FLOAT          1.00000
FLTRWHL         STRING    'LW-CVF2'
WAVELENGTH      FLOAT          12.6510
NSCD            LONG              76
NBR_FRAME       INT          1630
FROM            INT       Array[76]
TO              INT       Array[76]
TAB_FRAME       INT       Array[76]
ADU_SEC_COEFF   FLOAT     Array[76]
TABFLATCOEF     FLOAT     Array[76]
CUBE            FLOAT     Array[32, 32, 1630]
CUBE_UNIT       STRING    'ADU'
MASK            BYTE      Array[32, 32, 1630]
OTF             BYTE      Array[1630]
DU              FLOAT     Array[1630]
DV              FLOAT     Array[1630]
UTK             LONG      Array[1630]
BOOTTIME        LONG      Array[1630]
TEMPERATURE     FLOAT     Array[10, 1630]
IMAGE           FLOAT     Array[32, 32, 76]
IMAGE_UNIT      STRING    ''
RMS             FLOAT     Array[32, 32, 76]
NPIX            FLOAT     Array[32, 32, 76]
CCIM            STRUCT    -> <Anonymous> Array[1]
INFO            STRUCT    -> <Anonymous> Array[1]
DARK            FLOAT     Array[32, 32]
FLAT            FLOAT     Array[32, 32]
CALG            STRUCT    -> <Anonymous> Array[1]
HISTORY         STRING    Array[70]
SSCD_NAME       STRING    'CSSC203056040001_96091918462427'
SAD_NAME        STRING    'CSAD203056040039_96091919440421'
```

Refer to Section 15.5.3 for guide to the *CVF* PDS architecture.

Again, before proceeding it is recommended to save the newly created *CVF* PDS.

```
CIA> save, filename='cvf_pds.dat', cvf_pds, /xdr
```

13.2 Calibrating the PDS

So you have your PDS created and want to calibrate it. There is a core set of CIA calibration routines that perform corrections which are required for all CAM data regardless of AOT, e.g. dark correction, deglitching, stabilization, flat-fielding. These routines are described in the Section 13.2.1. Further sections describe routines that perform AOT dedicated processing, e.g. raster MOSAIC creation. So to do a complete calibration of your data first read Section 13.2.1 followed by one of Sections 13.2.2, 13.2.4 or 13.2.5 depending on the flavour of your PDS.

13.2.1 Core calibration

The first three core calibration routines that we will meet are **corr_dark**, **deglitch** and **stabilize**. These routines perform the following core corrections: (i) dark correction, (ii) deglitching, (iii) stabilization. All have a choice of different methods, though for simplicity we will just use the default here. Though we use a *raster* PDS in our example below, these routines accept any flavour of PDS as input.

1. Following from Section 13.1.2 we will restore our previously saved *raster* PDS and perform the first calibration steps on the IMAGEs in *lw6_raster.cube*.

```
CIA> restore, 'lw6_raster.dat', /verb

CIA> corr_dark, lw6_raster

CIA> deglitch, lw6_raster

CIA> stabilize, lw6_raster
```

2. We can look at the results of our processing so far by viewing the the corrected IMAGEs in *lw6_raster.cube* with **x3d**.

```
CIA> x3d, lw6_raster
```

   Clicking on the button *mask* indicates the pixels that have been masked as unstable. By looking at the temporal history of the pixels you can immediately see if good stabilisation and deglitching have been achieved. A vertical profile of some background pixels will tell you if a good dark correction has been applied.

3. Now can proceed with flat-fielding. Firstly, all the IMAGEs per CAM pointing /SCD must be averaged to EXPOSUREs – the routine **reduce** does just that. Then the flat-fielding is performed on each EXPOSURE with **corr_flat**.

```
CIA> reduce, lw6_raster

CIA> corr_flat, lw6_raster
```

   Now we have completed the final processing step with the core calibration routines.

13.2.2 Raster MOSAIC creation

The final step in processing raster observation data is creation of the raster MOSAIC from the corrected EXPOSUREs. The routine **raster_scan** will project all the EXPOSUREs on to the raster field-of-view and place the resultant raster MOSAIC in the PDS field *.raster*.

```
CIA> raster_scan, lw6_raster
```

If you wish you can also convert the MOSAIC pixels to milli-janskys (mJy).

```
CIA> conv_flux, bs_pds, /raster
```

By looking at the MOSAIC you can really get an idea as to how good the calibration is. We can display the MOSAIC most easily with **tviso**.

```
CIA> tviso, lw6_raster.raster
```

13.2.3 Staring analysis

Actually there is no real dedicated staring analysis. After the creation of the corrected EXPOSUREs (as in Section 13.2.1) the analysis of staring observation data is almost complete – the final formality is the conversion of the EXPOSURE pixels to mJy.

```
CIA> conv_flux, staring_pds, /image

CIA> print, staring_pds.image_unit
mJy/pix
```

To check the quality of your analysis you can use **tviso** to display individual EXPOSUREs.

```
CIA> tviso, staring_pds.image[*,*,0]
```

Or you can use **x3d** (though only if you have more than one EXPOSURE).

```
CIA> x3d, staring_pds.image
```

13.2.4 Beam-switch MOSAIC creation

The final step in calibrating the *BS* PDS is the creation of the beam-switch MOSAIC, i.e. the difference between the on source field EXPOSUREs and the reference field EXPOSUREs. The routine **reduce_bs** will perform this operation.

```
CIA> reduce_bs, bs_pds
```

For historical reasons the beam-switch MOSAIC is stored in the PDS field *.raster* (even though beam-switching has nothing to do with rasters!). Now you can convert the MOSAIC pixels to mJy.

```
CIA> conv_flux, bs_pds, /raster
```

As for the *raster* PDS you can check the results of the analysis with **tviso**.

```
CIA> tviso, bs_pds.raster
```

13.2.5 CVF analysis

Unlike the *raster* PDS or *BS* PDS there is no MOSAIC to create in a *CVF* PDS– the most interesting information is spectral rather than spatial. Since each EXPOSURE was observed at a different wavelength, spectra can be obtained directly from EXPOSURE pixels. Since most observers would like their calibrated spectra to be in janskys CIA provides **conv_flux** to convert the EXPOSUREs pixels from ADU to mJy.

```
CIA> conv_flux, cvf_pds, /image
```

To get a look at the calibrated CVF data use **cvf_display**.

```
CIA> cvf_display, cvf_pds
```

By using the mouse to point and click on selected EXPOSURE pixels you can display spectra. See Section 14.2.1 for more information on **cvf_display**.

13.3 Data calibration with x_cia

This section[2] serves as an introduction to the widget interface to CIA, **x_cia**.

13.3.1 Introduction

x_cia was designed to help visitors of 'Centre ISO français du CEA-Saclay' to become familiar with ISOCAM data analysis quite rapidly. Once ISOCAM data has been sliced/prepared, a standard quick look analysis can be performed (Section 13.3.2). ISOCAM data are mainly affected by the dark current, cosmic ray impacts, transient effects and flat-field errors. Correction of one of these effects is usually the result of several runs (Section 13.3.3). Note however, that not all CIA processing steps are available to **x_cia**.

Some advanced functionality is available but should only be used by advanced users of **x_cia** (Section 23). Guidelines to help the user to choose the most appropriate processing methods are given in Section 20. These guidelines are based on the joint experiences of all support astronomers of 'Centre ISO français du CEA-Saclay' and they are very likely to evolve as new methods are available. A general help on **x_cia** is presented in Section 23.2.

13.3.2 Quick Look analysis with x_cia

It is strongly advised to read Chapter 8 of the *ISOCAM Handbook*, in order to inform oneself about the calibration of ISOCAM images. The final calibrated images are indeed the result of many attempts. Nevertheless, it is very useful to get a rough idea of what ISOCAM has detected or to check that saturation has not occurred. In that sense, the following commands can be considered as the standard quick look analysis:

1. Start a CIA session

2. Type the following on the CIA command line:

   ```
   CIA> x_cia
   ```

[2] Taken from Claret A., 1996, *ISOCAM Data Analysis with X_CIA*, v2.2, Sections 1-3.

3. Choose AOT Type (Default is raster scan.)
4. Data / Load / SSCD
5. Data / AOT Info
6. Process / Default
7. Use all available functions of View and Tools menu to explore calibrated data.
8. Tools / Hardcopy to create hardcopies.
9. Data / Save / IDL File (If result is worth saving.)
10. CIA / Quit

13.3.3 Guidelines for using x_cia

13.3.3.1 Calling sequence

Full calling sequence of **x_cia** is,

```
x_cia, data=my_data, histo=my_histo, indark=my_dark, inflat=my_flat,
       logfile='my_logfile', noinit=noinit
```

where:

data is a PDS already in memory (also referred to as ISODATA in **x_cia**).

histo is a string array containing the history of the PDS (512 lines max).

indark is a 32×32 array containing the optional user input dark frame.

inflat is a 32×32 array containing the optional user input flat-field.

logfile is the log file name of the **x_cia** session (default name is cia_log.txt).

/noinit prevents initialisation of the log file (new information is appended to it).

Since all keywords are optional, the calling sequence can be reduced to **x_cia** only. If DATA and HISTO are used, then their modified values are returned at the end of the **x_cia** session (see below Example 5).

x_cia screen is displayed in Figure 13.1. It is important to read all error, warning and info messages displayed in the text window at the bottom of the **x_cia** screen.

13.3.3.2 Example 1: testing a calibration method other than the quick look analysis

The default calibration method consists of using the dark model, the multi-resolution median (*MM*) deglitching method, the so-called 90% Of final flux (*s90*) method to detect non-stabilised values, and a CAL-G or library flat field. Note that it may be better to use the *auto* flat field if you have a raster with sufficent background and not very extended sources. This global calibration is well adapted to get a rough idea of what ISOCAM has detected. So let's assume now that the user would like to test another calibration method, other than the default one, such as:

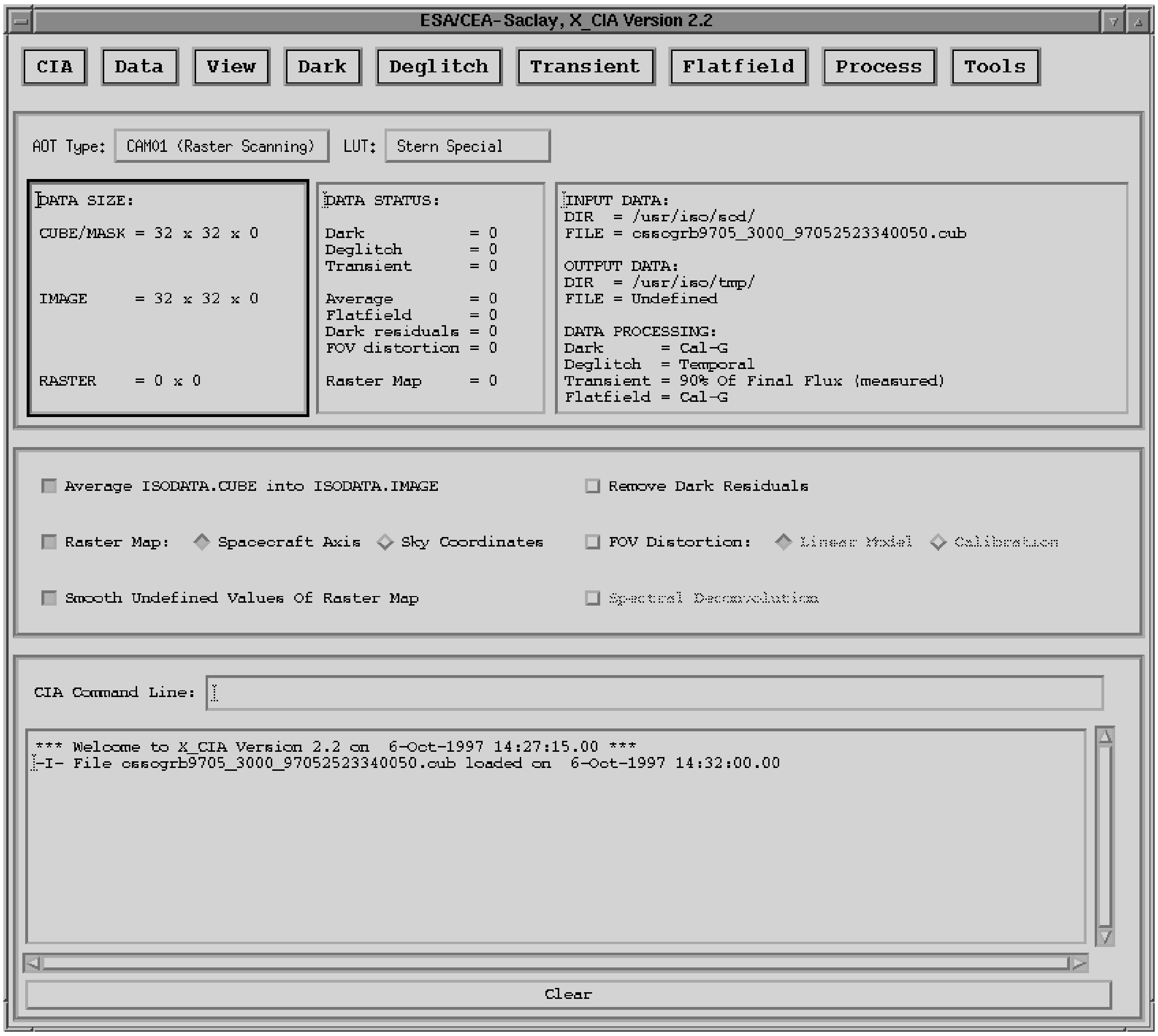

Figure 13.1: **x_cia** window.

- Subtract the user's own dark current frame (an IDL 32×32 array called *my_dark* in the following).
- Use the spatial deglitch method.
- Correct transient effects using the *SAP model fitting*.
- build flat-field by selecting a few frames of the raster (if no assumption on the extension or intensity of sources can be done).

Then the sequence of commands would be the following:

1. Start a CIA session.
2. Start **x_cia** with the following:

   ```
   CIA> x_cia, indark=my_dark
   ```

3. Choose AOT Type (default is raster scan).
4. Data / Load / SSCD
5. Data / AOT Info
6. Dark / User Input
7. Deglitch / Spatial
8. Transient / SAP Model Fitting
9. Flat-field / Manual
10. Process / Selected
11. Use all available functions of View and Tools menu to explore calibrated data.
12. Tools / Hardcopy to create hardcopies.
13. Data / Save / IDL File (If result is worth saving.)
14. CIA / Quit

13.3.3.3 Processing several data calibration methods in a single session

The PDS contains two fields in which calibrated data are stored: .CUBE (computed IMAGEs) and .IMAGE (reduced EXPOSUREs per raster position, per CVF wheel position or per filter). The PDS field .MASK (which has the same size as .CUBE) is flagged to 1 for all pixels containing a bad value (glitch, not stabilised or dead pixel) or for 'slewing frames' (during which the OTF was off). For the *raster* PDS, the additional field .RASTER is built from .IMAGE and contains the constructed raster map.

Dark correction updates the .CUBE of the PDS, and glitch and transient corrections update both .CUBE and .MASK of the PDS. Then .CUBE can be averaged into .IMAGE, excluding all bad pixel values flagged in .MASK. This averaging operation is optional but is performed automatically whenever a flat-field correction is required. Chapter 20 gives a more detailed description the various calibration methods and their impact on the PDS.

It is possible to correct data for dark current and glitches only, save the result in an IDL file, and then try different methods for transient and flat-field corrections (reloading the dark and glitch corrected data before each new try). So the user must take care of what corrections have been performed on a data set – including the order in which they have been performed – before trying any other correction. It is possible to perform the same correction more then once on a data set, though novice users are strongly discouraged from doing so. It is recommended that you carefully read Chapter 20 if you really want to try this. The *DATA STATUS* window displays some useful information about the number of times that each calibration was performed on the data set. If the data set is loaded from an IDL restore command, then all indices start at 900 instead of zero in order to indicate that the data status is unknown.

A data history has been designed in order to help the user to keep a track of all processes which have been applied to the data set. This history can be displayed on the screen at any time (using *Data / Display History*). The file *data_history.txt* is then created in the current directory.

A log file of the whole **x_cia** session can also be displayed on the screen (using *CIA / Log File*). This log file contains all commands, warning and error messages of the current **x_cia** session. The default name of the log file is *cia_log.txt*. This file is automatically overwritten at the beginning of a **x_cia** session unless the */noinit* keyword is used. The user can also choose another name by typing:

```
CIA> x_cia, logfile='my_name.txt'
```

13.3.3.4 Example 2: testing several methods for flat-field correction

Let's assume that the user wants to correct data from dark current, glitches and transient effects, and then try several methods for the flat-field correction. Assuming that a user input frame (an IDL 32×32 array called *my_flat* in the following) is used as flat-field, it is necessary to load it using the INFLAT keyword.

Hence, the sequence of commands would be the following:

1. Start a CIA session.
2. Start **x_cia** with the following:

   ```
   CIA> x_cia, inflat=my_flat
   ```

3. Choose AOT Type (default is raster scan).
4. Data / Load / SSCD
5. Data / AOT Info
6. Process / None
7. Dark / User Input
8. Deglitch / Spatial
9. Transient / SAP Model Fitting
10. Process / Selected

11. Data / Save / IDL File (my_file.xdr)
12. Data / Load / IDL File (my_file.xdr)
13. Process / None
14. Flat-field / Auto
15. Process / Selected
16. Use all available functions of View and Tools menu to explore calibrated data.
17. Data / Reload Original Data
18. Flat-field / Manual
19. Process / Selected
20. Use all available functions of View and Tools menu to explore calibrated data.
21. Data / Reload Original Data
22. Flat-field / User Input
23. Process / Selected
24. Use all available functions of View and Tools menu to explore calibrated data.
25. Tools / Hardcopy to create hardcopies.
26. ...
27. CIA / Quit

Note that once data have been corrected from dark current, glitches and transient effects, it is necessary to save them in a IDL file (step 13) in order to be able to reload this IDL file (step 14) just after. Indeed, the *Data / Reload Original Data* commands (steps 19 and 23) permit to retrieve data in the same state since the last loading (i.e. already dark, glitch and transient corrected data in that case). Then it make sense to try a new flat-field correction (steps 17, 21 and 25). Note that if *Data / Save / IDL File* (step 13) is omitted, then data would be retrieved in the same state after the *Data / Load / SSCD* command (i.e. raw data) and dark, glitches and transients effect would be visible in final images.

13.3.3.5 Example 3: data calibration for 'false beam-switching'

Let's assume that a 2×1 raster was performed, the first position corresponding to the source position ("ON source") and the second one to the background position ("OFF source"). With such an observing mode – which could be called 'false beam-switching' – it is possible to remove simultaneously the dark current and the background. In that case, data just need to be deglitched before this operation. Transient correction can first be considered as optional and will be detailed in Example 4.

Hence, the sequence of commands would be the following:

1. Start a CIA session.

2. Start **x_cia** – simply type **x_cia** on the CIA command line.
3. Choose AOT Type (default is raster scan).
4. Data / Load / SSCD
5. Data / AOT Info
6. Process / None
7. Deglitch / Spatial
8. Process / Selected
9. Data / Save / IDL File (my_file.xdr)
10. CIA / Quit

If the user answers 'Yes' to the question 'Average ISODATA.CUBE into ISODATA.IMAGE ?' after step 10, the removing of dark and background can be made in a CIA session with the following IDL commands:

```
CIA> restore, 'my_file.xdr', /verb

CIA> source = isodata.image(*,*,0)

CIA> background = isodata.image(*,*,1)

CIA> image = source - background

CIA> tviso, image
```

If the user answers 'No' to the question 'Average ISODATA.CUBE into ISODATA.IMAGE ?' after step 9, the removing of dark and background can be done in a CIA session with the following IDL commands:

```
CIA> restore, 'my_file.xdr', /verb

CIA> on_source = isodata.cube(*,*,isodata.from(0):isodata.to(0))

CIA> off_source = isodata.cube(*,*,isodata.from(1):isodata.to(1))

CIA> source = reduce_cube(on_source)

CIA> background = reduce_cube(off_source)

CIA> image = source - background

CIA> tviso, image
```

13.3.3.6 Example 4: testing several methods for transient correction

Let's assume that the user wants to try different transient corrections on the previous data (see above Example 3).

Hence, the sequence of commands would be the following:

1. Start a CIA session
2. Start **x_cia** – simply type **x_cia** on the CIA command line.
3. Choose AOT Type (default is raster scan).
4. Data / Load / IDL File (my_file.xdr)
5. Process / None
6. Transient / SAP Model Fitting
7. Process / Selected
8. Use all available functions of View and Tools menu to explore calibrated data.
9. Transient / 90% Of Final Flux (measured)
10. Process / Selected
11. Use all available functions of View and Tools menu to explore calibrated data.
12. Tools / Hardcopy to create hardcopies.
13. ...
14. CIA / Quit

13.3.3.7 Example 5: loading IDL data structure from memory

If an IDL data structure is already in memory, it is possible to load it directly into the **x_cia** session, by typing:

```
CIA>x_cia, data=my_structure
```

The modified value of *my_structure* will then be returned at the end of session.

If a string array is already in memory, the user can also use it as its own data history by typing:

```
CIA>x_cia, data=my_structure, histo=my_history
```

The string array *my_history* will then be returned at the end of **x_cia** session. If the array *my_history* is not empty at the beginning of the session, new information will be appended to it.

13.3.3.8 Returning to home institute

You may choose to return to your home institute with the results of your calibration either in XDR format (using *Data / Save / IDL File*) or in FITS format (using *Data / Save / SAD (Fits)*).

13.3.4 x_cia caveats

x_cia can only load a PDS from an IDL save file if the PDS has the variable name *isodata*. The name of the actual save file is not limited in anyway and may be independent of the saved PDS.

13.4 Calibrating a PDS the old way

This section describes the calibration of a PDS using the old **calib_raster**, **calib_bs**, **calib_cvf** and **calib_struct** routines. While these routines continue to work they are no longer maintained within CIA. Use of the core calibration routines described in Section 13.2.1 gives the user more flexibility and control over the calibration process. You are strongly advised to use the core calibration routines.

13.4.1 calib_raster

We present here an example calibration of a *raster* PDS with **calib_raster**. This example follows from Section 13.1.2.

1. First we make a copy of the uncalibrated *raster* PDS.

   ```
   CIA> original_lw6_raster = lw6_raster
   ```

2. We apply a basic calibration treatment to the *raster* PDS by calling **calib_raster** with the keyword */all* set – see the **calib_raster** in the on-line help for the details of *all*.

   ```
   CIA> calib_raster, lw6_raster, /all
   ```

   Then save the calibrated MOSAIC so we can later compare with different calibration results:

   ```
   CIA> lw6_raster_all = lw6_raster.raster
   ```

3. We can look at the results of our calibration by loading the cube of IMAGEs and the MASK from our *raster* PDS into **x3d**:

   ```
   CIA> x3d, lw6_raster.cube, lw6_raster.mask
   ```

   Clicking on the button *mask* indicates the pixels that have been masked as unstable. By looking at the temporal history of the pixels you can immediately see if good stabilisation and deglitching have been achieved. A vertical profile of some background pixels will tell you if a good dark correction has been applied.

   Another way of judging the quality of the calibration is to look at the signal-to-noise ratio of some of the EXPOSUREs:

   ```
   CIA> xsnr, lw6_raster.image[*,*,0]
   ```

   Or simply display the MOSAIC with **tviso**.

   ```
   CIA> tviso, lw6_raster_all
   ```

4. Try another calibration using deglitch method *MM*, stabilisation method *s90* and the CAL-G flat.

   ```
   CIA> lw6_raster = original_lw6_raster

   CIA> calib_raster, lw6_raster, /dark, deglitch='mm', stab='s90', $
   CIA> flat='calg', raster='proj'

   CIA> lw6_raster_mm = lw6_raster.raster
   ```

   This process of calibration and re-calibration may be repeated *ad nauseum*.

5. Finally to save your work, use IDL's SAVE with the *xdr* option for portability:

   ```
   CIA> save, filename='lw6_raster', lw6_raster, lw6_raster_mm, $
   lw6_raster_spat,/xdr
   ```

13.4.2 calib_cvf

Calibration of the *CVF* PDS is similar to that of the *raster* PDS in Section 13.4.1, the only difference being that you don't have a MOSAIC to create from the EXPOSUREs in a CVF observation.

Our calibration example here follows the slicing example of Section 13.1.5:

1. It is a good idea to begin by making a copy of the uncalibrated *CVF* PDS.

   ```
   CIA> original_cvf_pds = cvf_pds
   ```

2. The *CVF* PDS is passed to **calib_cvf** for calibration:

   ```
   CIA> calib_cvf, cvf_pds, stab='s90', deglitch='spat', flat='calg', $
   CIA> dark='calg'
   ```

   Our call to **calib_cvf** specified using the spatial deglitching technique, the *s90* method of stabilisation and the CAL-G DARK and FLAT.

3. As for a *raster* PDS(see Section 13.4.1), you can inspect the quality of the calibration of your *CVF* PDS with **x3d** and **xsnr**. However, there is no MOSAIC image to be displayed.

4. You can reassign *cvf_pds* with the uncalibrated *CVF* PDS and run **calib_cvf** on it again.

   ```
   CIA> cvf_pds = original_cvf_pds

   CIA> calib_cvf, cvf_pds, deglitch='mm', stab='s90', /flat
   ```

5. And you can finally save your work with IDL's save:

   ```
   CIA> save, filename='cvf_pds', cvf_pds, /xdr
   ```

13.4.3 calib_bs

The *BS* PDS is calibrated with **calib_bs**. Again it is called in a very similar manner to **calib_raster**. Like **calib_raster**, **calib_bs** does create a MOSAIC image – though in this case this is simply the difference between the on source field and the reference field.

- As with the other calibration routines you specify the general calibration treatments, e.g:

  ```
  CIA> calib_bs, bs_pds, deglitch='tcor', /dark, flat='calg', $
  CIA> stab='s90'
  ```

- Then to create the beamswitch MOSAIC you set the keyword *bs*.

  ```
  CIA> calib_bs, bs_pds, /bs
  ```

  Of course, as in the **calib_raster** examples, all the keywords can be set at one call of **calib_bs**.

- For compatibility reasons, the beamswitch MOSAIC is stored in the field .RASTER. To view the result of your calibration you can use x3d and **xsnr**, or simply.

  ```
  CIA> tviso, bs_pds.raster
  ```

13.4.4 calib_struct

The *general* PDS is calibrated with **calib_struct**. This is in fact the same routine that **calib-_raster** calls to do calibration on a *raster* PDS. The main difference between the routines is that **calib_raster** has the additional capability to create a raster MOSAIC. Read Section 13.4.1, keeping the following points in mind.

- As when using **calib_raster** you specify the type of calibration you want, though of course there is no keyword *raster* with **calib_struct**, e.g:

  ```
  CIA> calib_struct, staring_pds, deglitch='tcor', /dark, flat='calg', $
  CIA> stab='s90'
  ```

- Similarly, you can also use x3d and **xsnr** to look at your resulting data. However, there is no MOSAIC in a *general* PDS to view.

Chapter 14

Image analysis and display

This chapter serves as an introduction to the variety of image and cube analysis, and display routines in CIA.

They are categorised as:

- General analysis routines that perform some type of statistical or photometric analysis: **xsnr**, **xradial**, **flux_sum**, **xphot**.
- CVF image analysis routines: **cvf_display**.
- Image analysis routines that only operate on 2D images: **xdisp**.
- Cube analysis routines that operate on a cube, yielding temporal and spatial information on a pixel: **xselect_frame**, **show_frame**, **xsubcube**, **xcube**, **x3d**, **xv_temp**, **xv_raster**, **xmovie**.
- Image display routines that just display images or images from cubes: **tviso** and **xv_sscd**.
- Image overlaying routines: **isocont**, **x_isocont**.
- Finally routines to aid you to produce postscript output: **xcontour**, **ps_color**, **white**, **black**, **ps_open**, **ps_print**.

14.1 General analysis routines

These routines are **xsnr** for S/N estimation, **xradial** for determining the energy radial profile and **flux_sum** for estimating the flux from a point source.

14.1.1 Estimating S/N in a cube or image

Estimating S/N in a CAM image is a useful indicator of the strength of source detection or the quality of a calibration.

xsnr is a widget based program that is used to interactively choose noise and signal regions in an image. The image could be the MOSAIC (.RASTER in a *raster* PDS) or an EXPOSURE (.IMAGE in all PDSs). As an example we can take the fifth EXPOSURE from a *CVF* PDS.

```
CIA> xsnr, cvf_pds.image(*,*,4)
```

The **xsnr** window presents three subwindows to the user. Two contain a display of the noise region selected and of the signal region selected. Selection of these regions can be made with sliders. The third subwindow displays the full image (the first in a cube) with both noise and signal regions indicated by a white box. Clicking on the button *Calcul* with calculate the ratio of the signal and noise in each region.

14.1.2 Energy radial profile

An estimation of the energy radial profile of a source may be done with the widget-based program **xradial**.

```
CIA> radial_profile=xradial(raster_pds.raster)
```

The brightest pixel in the image is automatically selected and the user may interactively set the radius of the disk around that pixel, from which the profile is taken, with a slider. The *integrate* button toggles on/off integration of the profile.

Calculated data are displayed in a subwindow of the widget and is also the output of the function. In the preceding example *radial_profile* contains these data when **xradial** is quitted. To reproduce the profile, try:

```
CIA> plot, radial_profile
```

14.1.3 Estimating the total source flux

The routine **flux_sum** will help estimate the total flux from a point source and the background level in the image. Upon calling, it displays a plot window of integrated flux against distance from source. The user selects the interval for the background fit and enters it upon request.

An example call to **flux_sum** could be:

```
CIA> flux_sum,cvf_pds.image(*,*,2),flux,flux_rms,background,background_rms

CIA> print, flux, background
```

In the example, the input argument is the 3rd image in the .IMAGE of a PDS. The subsequent arguments are outputs and return the estimated flux, the RMS on the flux, the estimated background level and the RMS on the background.

14.1.4 Photometry measurements with xphot

xphot is a photometry tool specifically designed to work with CAM data. A detailed description of **xphot** can be found in Sauvage & Aussel 1997. Here we will give a simple walk-through example.

1. Invoke **xphot** with...

   ```
   CIA> xphot, raster_pds.raster
   ```

   As you might guess from the syntax above, we are going to perform photometry on the MOSAIC of a *raster* PDS. You can supply any kind of CAM image to **xphot** – though you may prefer it to contain a source. If you wish to do PSF fitting and the theoretical PSFs (see Section 15.4.1 for a description) have not been installed in the correct location then you can specify an alternate path with the keyword *psf_dir*:

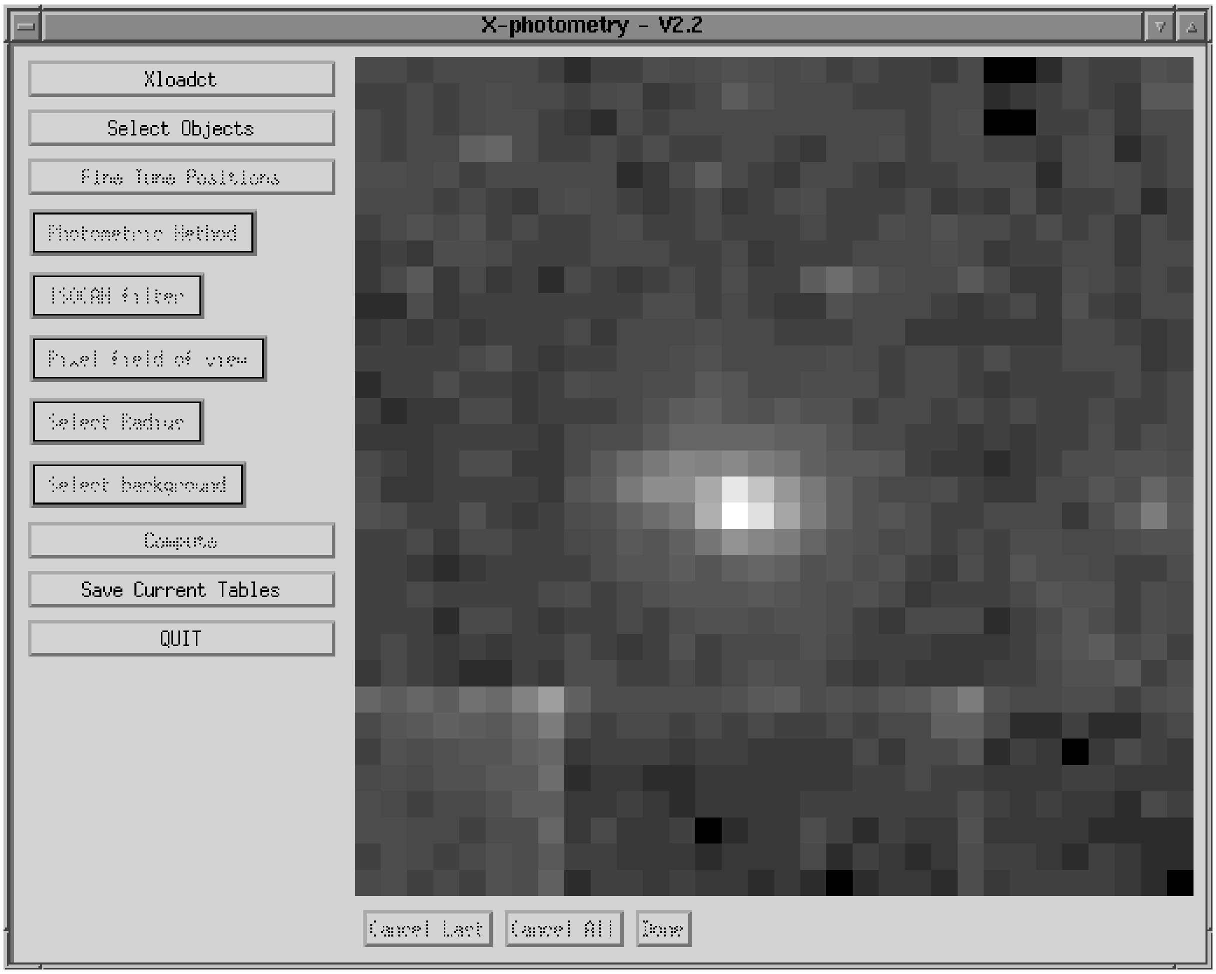

Figure 14.1: The **xphot** window.

```
CIA> xphot, raster_pds.raster, psf_dir='/home/cia/isocia/ia/PSF/'
```

2. Take a look at Figure 14.1. It contains the image we supplied when invoking **xphot**, i.e. a raster MOSAIC, and a number of buttons. The button *Select Objects* is highlighted, so the first action you must take is to select the sources that you wish to do photometry on.

   Click on *Select Objects* to begin the selection process. A pop-up window will appear with some information – read and dismiss. Now you can choose your sources by clicking on the image. In the example image displayed in Figure 14.1 there is clearly only one source. Click on *done* when you are finished making the selection.

3. Now click on *Photometric Method* to choose the photometric method to apply to the source. In our example here we will select the first option: simple aperture photometry.

4. The next step is to determine the radius over which to perform the photometry. Click on *Select Radius* and choose the simplest method, *1 for all*. A new window will appear, displaying a region of the image centered on the source. Click on this window to indicate the desired radius. Click *done* when finished.

5. The final step is to determine the background. Clicking on *Select Background* will give you a choice of methods, the simplest being *1 for all*. Again, we will choose this option for our example here. Now select a region of the displayed image for determination of the background: click on the lower-left point of the region first, followed by the upper-right point.

6. You can now click on the button *compute* to perform the photometric calculation according to the parameters that you have chosen above. The results will be printed to screen in the following order: the source/object number, its pixel coordinates, the aperture radius, the background value and the source flux. When you quit **xphot** (click on *quit*) these data will be saved to disk. To recover...

```
CIA> restore,'xphot_tables_final.save',/verb
% RESTORE: Portable (XDR) SAVE/RESTORE file.
% RESTORE: Save file written by mdelaney@bikini, Thu Oct  9 17:07:40 1997.
% RESTORE: IDL version 5.0 (sunos, sparc).
% RESTORE: Restored variable: ID.
% RESTORE: Restored variable: X.
% RESTORE: Restored variable: Y.
% RESTORE: Restored variable: RAD.
% RESTORE: Restored variable: BKG.
% RESTORE: Restored variable: FLUX.

CIA> print, flux
      106.678
```

14.2 CVF image analysis

14.2.1 cvf_display

cvf_display is a widget program which you can use to calculate a CVF spectrum. Assuming

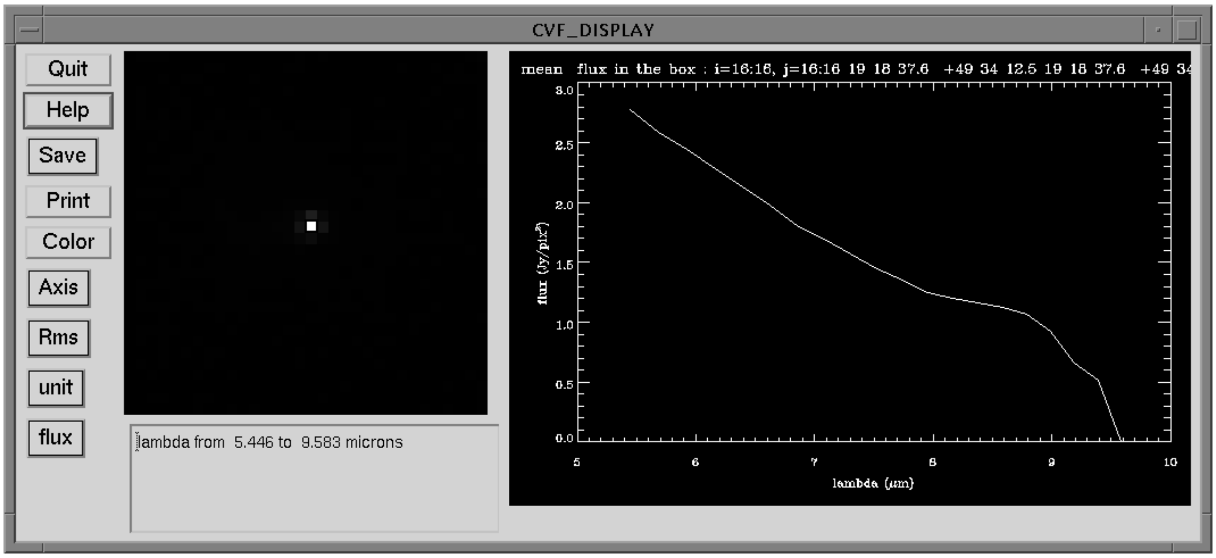

Figure 14.2: **cvf_display** window.

that you have a calibrated *CVF* PDS[1], then **cvf_display** can be simply invoked with:

```
CIA> cvf_display, cvf_pds, flux=cvf_spectrum
```

A single window will appear with an image display subwindow and a plotting subwindow – see Figure 14.2. A mean of the EXPOSUREs in the *CVF* PDS field .IMAGE is initially displayed.

1. Clicking on a pixel will display its spectrum (arbitrary flux units against wavelength (microns)) in the plotting subwindow.
2. Clicking on a point on the plot with the right mouse button will display the EXPOSURE at that wavelength.
3. A region of an EXPOSURE can be selected and the mean spectrum of its pixels plotted. Click on a pixel in the EXPOSURE with the middle mouse button and then release the button over another pixel. A mean spectrum of the subarray defined by these pixels will be displayed.
4. If you have kept the SAD created by **get_sscdcvf** when you first made the *CVF* PDS, then the current spectrum in the plot window can be placed in the SAD substructure, .CSSP. Click on *Update SAD* and when you exit **cvf_display** you will find .CSSP filled:

   ```
   CIA> cvf_cssp=sad_get('cssp', cvf_pds.sad_name)

   CIA> help, cvf_cssp, /str
   ** Structure SPECTRUM_STRUC_1, 11 tags, length=4152:
      NAME            STRING    'spec000000000000_00'
      NSPEC           LONG                   0
   ```

[1]See Section 15.5.3 for a description of the *CVF* PDS data structure and Section 13.4.2 for an introduction to calibrating a CVF observation.

```
RA              FLOAT          288.132
DEC             FLOAT          67.6615
N               LONG                76
WAVELENG        FLOAT     Array(200)
BANDWIDT        FLOAT     Array(200)
STAT            FLOAT     Array(200)
FLUX            FLOAT     Array(200)
DFLUX           FLOAT     Array(200)
MASK            BYTE      Array(4, 32)
```

5. If you have lost the SAD, maybe by saving the *CVF* PDS only and restoring without the SAD in another CIA session, then don't worry. The keyword *flux* returns the last spectrum displayed before quitting **cvf_display**. Wavelength information can be found in the *CVF* PDS fields .CVF_INCR, .WAVELENGTH_START and .WAVELENGTH_END.

6. **xloadct** can be invoked by clicking on *Color*.

7. The button *Axis* allows you to change either or both axes to a logarithmic scale.

14.2.2 xcvf

xcvf is a visualization tool for CVF data and a front end for the lower level routine **conv_cvf2isap**. **xcvf** only takes *CVF* PDSs as input, while **conv_cvf2isap** works with both *CVF* PDSs and AAR products. Since this chapter is concerned mostly with image visualization, the description **conv_cvf2isap** may be found later in Section 18.1. Both these routines output the CVF data to a file that is readable by ISAP.

The calling sequence for **xcvf** is:

```
CIA> xcvf, cvf_pds
```

xcvf also accepts the keywords:

iz Exposure number to display.

z1 Lower bound on the range of intensities to display.

z2 Upper bound on the range of intensities to display.

outfile Name of the default output file.

help If set then **xcvf** will display a short help screen.

On startup the user is greeted with a summary of mouse commands and explanation of the plot symbols and lines.

14.2.2.1 The display

The row of buttons and fields at the top of GUI perform the following operations.

QUIT Exit **xcvf**.

HELP Re-print the short summary listed on startup.

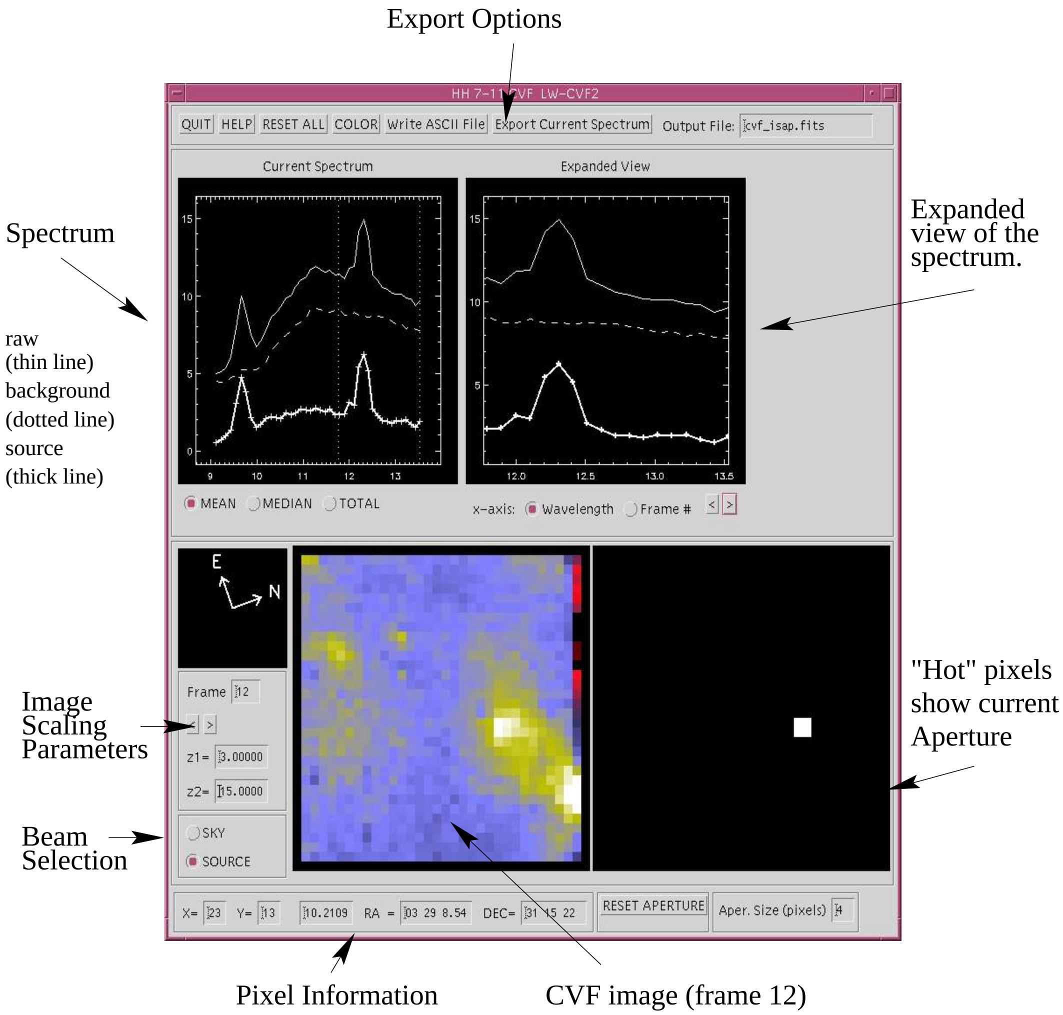

Figure 14.3: The **xcvf** window.

RESET ALL Reset all apertures, as well as the lower and upper bounds on the intensity ranges to display.

COLOR Bring up IDL's *xcolor* widget for interactive control of color tables and stretches.

Write ASCII File This will write the current spectrum as an ASCII file with column headers and a short summary of apertures and combination option used to produce the spectrum.

Export Current Spectrum Call **conv_cvf2isap** and export the current spectrum for use in ISAP. The current spectrum is the one shown by the thick solid line and the '+' symbols in the plotting window (see below).

Output File The name given by **conv_cvf2isap** to its output file. You must follow any changes to this name with the return (enter) key for the new name to take effect.

The two display units below the top row show the current spectrum in the source aperture (solid line), the sky aperture (dashed line) and the difference between the two (thick solid line marked with '+' symbols). The left panel shows the full spectrum. The right panel shows an expanded (zoomed) view. The zoomed region is marked by two vertical dotted lines in the left panel. The set of buttons immediately beneath the plotting windows allow the user to select how the spectrum is computed. Choices are average, median or the sum of all spectra currently in the aperture. The axis selection allows one to plot the spectrum versus the wavelength or the frame number. The arrows (<, and >) allow the user to reposition the two vertical lines marking the boundaries of the expanded view.

The section of the GUI beneath the plotting region displays frames from the reduced CVF cube. The left-most part of this region displays astrometry information, and allows the user to select which frame to display as well the range of intensity to use. The panel at the bottom left of the display region is used to select the beam. The left panel in the display region shows the current frame in the reduced data cube. The right panel highlights the pixels currently in the aperture. In both panels, clicking mouse buttons has the following effects:

mouse LEFT Selects the pixel and makes it part of the aperture. The right panel highlights the pixel. If the pixel is already selected, clicking on it has no effect.

mouse RIGHT Removes the pixel from the aperture.

mouse MIDDLE While held down, shows the spectrum of the current pixel in the plotting window, and the astrometry information in the bottom panel (see below). When released, the previous spectrum (if an aperture has been selected) is displayed again.

The last part of the GUI displays the following information about the current pixel selected:

1. its row and column on the array
2. its intensity in the currently displayed frame
3. the equatorial co-ordinates corresponding to the pixel's spatial location in J2000 equinox.

Additional information and buttons in the last row are:

1. the 'RESET APERTURE' button which resets the aperture and
2. the number of pixels in the current aperture.

14.2.2.2 Defining and editing apertures

First, you need to the select the beam (either sky or source). The apertures are defined by pressing the LEFT mouse button on the pixel which is to be included in the aperture. Either the frame display or the aperture display windows are clickable. To remove a pixel from an aperture, position the cursor over the pixel and press the RIGHT mouse button.

14.2.2.3 Quick Look

This is useful when you are curious about how the spectrum form a particular pixel looks but don't necessarily want to include the pixel in the aperture. Position the cursor over the pixel of interest and press and hold the MIDDLE mouse button. The spectrum will remain visible while the button is held down. The astrometry information at the bottom panel is also updated.

14.2.2.4 Some miscellaneous tips

1. All messages (if any) from **xcvf** are displayed in your IDL session window. If you become confused, check to see if there were any messages from **xcvf**.

2. When using TOTAL (sum of all spectra in the aperture) to compute the current spectrum, the user should take care to use the same size aperture for both the SKY and the SOURCE beam for the sky subtraction to be meaningful. Currently, **xcvf** warn users if a discrepancy exists and the user tries to export the current spectrum. However, the output file will be written after the warning message is displayed.

14.3 2-D image analysis.

Currently two routines exist for 2-D image analysis. These are **xdisp** and **sad_display**. You have met **sad_display** before, see Section 10.2, so here we will give only a brief example of its use with **struct2sad**.

14.3.1 xdisp

xdisp is a CIA routine to display and analyse single images. They can be in several formats: IDL 2-D array, FITS format images, the CCIM from an SAD or images in MIDAS format. It can interactively display profiles of rows or columns, display histograms, create contour plots (uses **xcontour**, see Section 14.7.1), perform simple statistical analyses and compute the Fourier transform of an image. A recently added feature is the option to interactively save a region of the displayed image as a FITS file. This FITS file can then be used as an input to other analysis packages or other CIA routines such as **isocont**.

14.3.1.1 The xdisp window

Figure 14.4 shows the **xdisp** window. All the operations described above can be performed by clicking on the following window buttons:

- *quit* – Quit the application.

- *load* – Load an image. Image formats can be fits, MIDAS, SAD or 2-D array.

- *lut* – Modify the LUT, i.e. call xloadct.

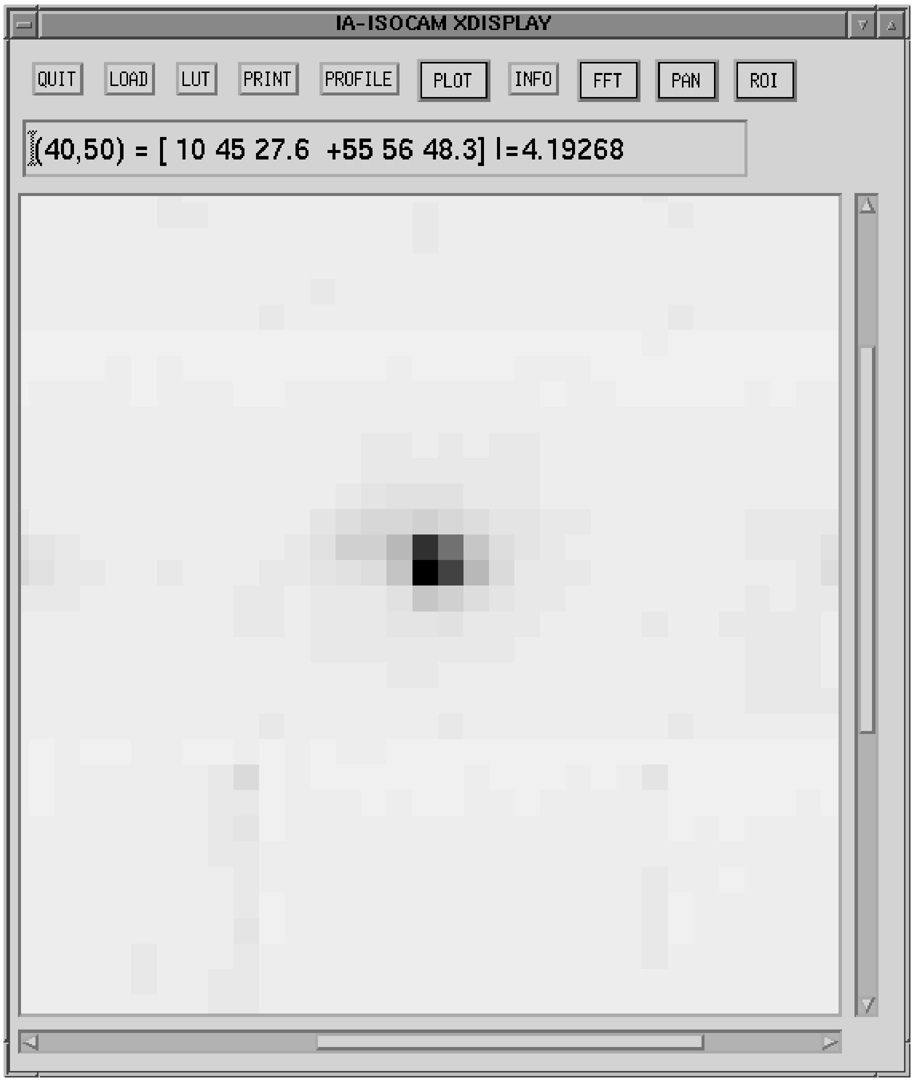

Figure 14.4: The **xdisp** window.

- *get / profile* – Examine lines or columns.
- *get / cursor* – Examine pixel values. If image format is FITS, PDS, or SAD and if the header contains astrometric information, then pixel position in the sky is given (R.A. and Dec.).
- *plot / histo* – Plot the histogram.
- *plot / contours* – Invoke **xcontour**.
- *plot / 3d visu* – Invoke interactive widget to display a surface plot.
- *plot / load ..* – copy image to an ordinary IDL window.
- *info* – Print the min, max, mean, sigma of the image
- *fft* – Computes the Fourier transform and display either the power spectrum, the phase, the real part or the imaginary part.
- *pan* – Pan the image. The zoom factor can be 2, 4, 8, 1/2, 1/4, 1/8.
- *ROI / select* – Interactively select a region of the displayed image.
- *ROI / save* – Save the selected region as as a FITS file.

14.3.1.2 xdisp examples

1. Following the examples of Section 12.2 we can examine the raster image in the raster PDS. Figure 14.4 was created with this example.

   ```
   CIA> xdisp, lw6_raster.raster, raster2hdr(lw6_raster)
   ```

   In the above example, the routine **raster2hdr** is used to create a FITS header from the PDS *lw6_raster* and this header is passed directly to **xdisp**. This is particularly useful for getting coordinates of raster MOSAIC pixels (see description of menu option *get / cursor* in Section 14.3.1.1).

   We can also use the select and save a selected region of the image to a FITS file (see description of menu option *ROI*). Since we have supplied a FITS header to **xdisp** then the FITS file will contain astrometry for the selected region. For example, click on *ROI / select* and use the mouse to drag the selection zone over the region you are interested in saving (watch the screen for information). After selection is completed click on *ROI / save* to save the selected region. You will get a screen message like:

   ```
   File ima33x33.fits written ...
   ```

   The FITS file can be used as input to **isocont** – see Section 14.6.2.

2. Using **xdisp** for viewing FITS data in CIA:

   ```
   CIA> haro3_image=readfits('haro3_iso.fits', haro3_header)

   CIA> xdisp, haro3_image, haro3_header
   ```

3. **xdisp** is designed to directly read SADs from disk or memory and display the contents of the CCIM.DATA field:

   ```
   CIA> xdisp, 'CSAD143006010205_96090620365354'
   ```

14.3.2 sad_display and struct2sad

In section Section 10.2 we used **sad_display** to create SADs from CCIM and CMOS data products and then display their corresponding EXPOSUREs and MOSAICs. We were only concerned with viewing AAR data products at that time.

However, since we can create an SSAD from a PDS using **struct2sad** we can use **sad_display** to view the results of our own calibration.

Here are some examples:

1. Any PDS can be converted to an SSAD and viewed with **sad_display**:

   ```
   CIA> sad_display, struct2sad(any_pds, /all)
   ```

   The optional keyword *all* tells **struct2sad** to make a complete as possible set of SADs. If *any_pds* is a *raster* PDS or *BS* PDS, then clicking on the button *future* (see Figure 10.1) will display the raster or beam-switch MOSAIC.

2. If you only want to see the raster or beam-switch MOSAIC then set the keyword *raster*:

   ```
   CIA> sad_display, struct2sad(raster_pds, /raster)
   ```

14.4 Cube analysis

14.4.1 Extracting images from cubes with xselect_frame

This is a simple widget-based program to interactively allow the user to select a single image from a given cube – see Figure 14.5. It is invoked as:

```
CIA> frame=xselect_frame(cvf_pds.image, frame_index)
```

frame would contain the selected image upon quitting and *frame_index* is the index of the selected image in the cube.

14.4.2 Extracting images from cubes with xsubcube

xsubcube is similar to **xselect_frame**, though it allows for a subcube of images to be extracted from a cube rather than just a single image to be extracted. It is a widget-based program with several parameters that may be interactively selected:

- The top slider specifies the *number of frames per packet*, where a *packet* refers to an image in the final output subcube. In other words it specifies the number of images in the input cube to average to produce an image in the output subcube.
- The next two sliders allow for choosing of the first and last image in the input cube to make up the output subcube.

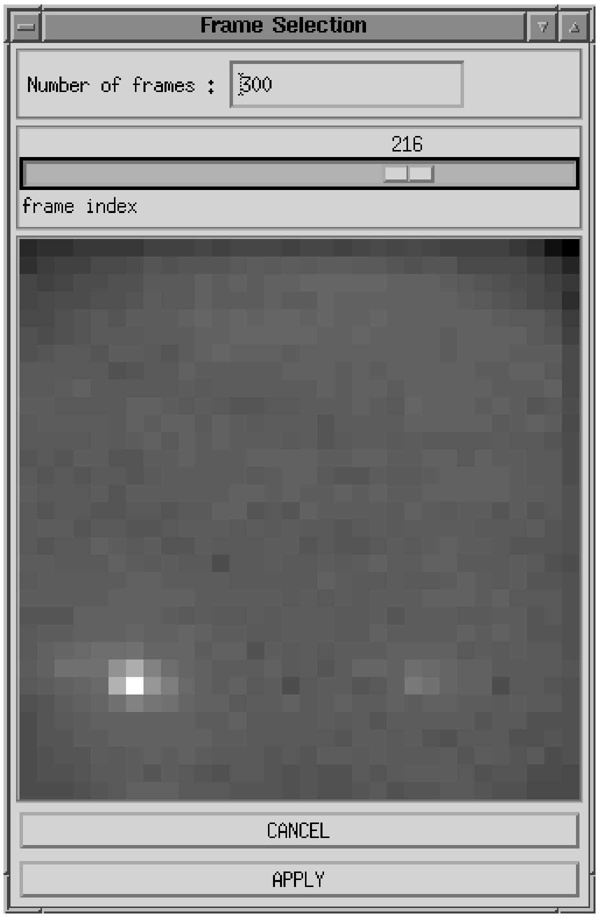

Figure 14.5: **xselect_frame** window.

- The last slider scrolls each image or the *packet* of the output cube.
- The button *Area Extraction* calls another widget which allows for the selection of a subimage from the images of the input cube.
- The buttons *reset* and *cancel* do the obvious. *apply* applies the selection and quits.

A typical application may be to select a subcube from a .CUBE or .IMAGE of a PDS:

```
CIA> image_subcube=xsubcube(cvf_pds.image)
```

14.4.3 xcube

xcube is a widget-based program for cube or image analysis and visualisation. It serves as an interface to some other widget-based routines – in such cases this is indicated and the routine cross-referenced. Using xcube for processing is deprecated. Use **x_cia** instead. Data can be read from disk or from memory. The input data (see Section 14.4.3.1) can be in different formats, defined by keywords.

xcube contains tools for:

- Input-Output: Load an image or cube (FITS, SAD, SCD). Write an image or cube (Fits, SAD, SCD). Create a Postscript file.
- 2D Display: Grey-scale, contours, shaded surface, profiles, ...
- 3D Display: Animation, average, temporal mean, analysis
- Selection: Select a LUT, a frame in a cube, a cube in an SCD, an image in an SAD,...
- 2D Processing: Extract a part of an image, radial profile, read values interactively with the cursor in the image.
- General Processing: Histogram, Median, SNR evaluation.

Its calling sequence is:

```
xcube [, scd=scd, sad=sad, type_data=type_data, data=data, fitshd=fitshd]
```

The keywords are: *scd*, *sad*, *type_data*, *data* and *fitshd*. *scd* and *sad* accept CIA data structures and *type_data* indicates the field of the structure we want to analyse[2].

14.4.3.1 xcube data I/O options

The different options for inputting and outputting data to xcube are:

- If no keyword is set, the user can load data interactively. A drop down menu will appear when the button *LOAD* is clicked on. Options will allow you to load the type of data you wish.

  Example:

  ```
  CIA> xcube
  ```

[2] See Chapter 15 for more details on data structure architecture.

- If the *scd* keyword is set, the keyword *type_data* can be equal to: 'data', 'model', 'reset', 'eoi', 'bad', 'dead', 'memory', 'glitch' or any SCD field (e.g. 'mask.glitch').

  Example:

  You can also load a single SCD and analyse the images from a single CAM state. In the example the SCD is loaded from disk, though if it resides in memory it can be directly passed to **xcube**.

  ```
  scd_example = scd_read('cscd95032164058_9503216405900')

  xcube, scd=scd_example
  ```

  To look at the mask in the SCD include the keyword *type_data* in the call:

  ```
  xcube, scd=scd_example, type_data='mask'
  ```

- If *sad* keyword is set, the keyword *type_data* can be equal to: 'ccim' for CCIM.DATA, 'data' for CCIM.DATA, 'rms' for CCIM.RMS_IMAGE, 'rms_image' for CCIM.RMS_IMAGE, 'weight' for CCIM.WEIGHT_IMAGE, or any SAD field (e.g. 'cmap.rms_image').

  Example:

  A SAD can be loaded in a similar to a SCD. Assuming that the SAD is memory, the following example does this.

  ```
  xcube, sad='CSAD950316111108_95031611120000', type_data='ccim'
  ```

- The keyword *data* can accept an IDL 2 or 3 dimensional array. The keyword *fitshd* accepts a fits header corresponding to the array passed to *data*. See Section 14.4.3.2 for examples.

- *scd*, *sad* and *data* are also output keywords. If *scd* and *sad* are set, then *scd* is the input data and *sad* is the output data. If *scd* and *data* are set, then *scd* is the input data and *data* is the output data.

14.4.3.2 Analysing cubes of images with xcube

A brief overview of *xcube*'s cube analysis functions follows.

After invoking **xcube** a control panel window appears. Click on the button *display*, a drop-down menu appears. Click on the option *cube* to reveal a menu containing the options below:

- *display / cube / analysis* – interactive analysis of cube pixels. Calls the routine **x3d**. See Section 14.4.4.

- *display / cube / movie* - Displays an animation of all the frames in your cube. Calls the routine **xmovie**. See Section 14.4.11.

- *display / cube / average* - Displays an averaged image of the frames in your cube.

- *display / cube / frames* - Invokes **show_frame** (see Section 14.5.2) to display individual frames from the input cube.

- *display / cube / temporal mean* - Plots a mean of the each frame against frame number.

14.4.4 Cube analysis with x3d

x3d is a widget-based program specifically for analysing cubes of images. **x3d** is started by calling it on the CIA command line with an IDL cube or a PDS as input. If the input is a PDS then the data in CUBE and MASK are loaded. Optionally, the CUBE and MASK can be explicitly specified as the first two arguments. An example call is:

```
CIA> x3d, cvf_pds
```

The **x3d** window (Figure 14.6) displays the current image from the cube being investigated. The current image is changed by either clicking on the buttons (*next frame* and *previous frame*), or by moving the slider bar to scroll through the cube. A plot window displays either the temporal history of the current pixel, a horizontal cut of the row of the current pixel or a vertical cut of the column of the current pixel. The current pixel is changed simply by clicking on the image and the type of plot displayed in the plot window is changed by choosing one of the buttons: *temporal cut*, *horizontal cut* or *vertical cut*.

If the **x3d** input is a PDS, or the CUBE and MASK are explicitly supplied, then the masked pixels in each image of the CUBE may be viewed. Clicking on the button *mask* will activate this feature. In the plot window, all the masked pixels will be marked with an +, and in the image display window, masked pixels will not be displayed, i.e. will show up blank in the image.

A newly added feature of **x3d** is the button *glitch* – see Section 14.4.5.

14.4.5 x3d as a calibration aid

x3d is an excellent tool for examining cubes of CAM images and looking at the temporal and spatial behaviour of a pixel in a CAM observation. It can even perform manually deglitching. This makes it a very useful calibration aid. To preform the functions described below **x3d** should be given a PDS or a CUBE and MASK as input.

- **Dark correction:** In a PDS that has not been dark corrected, a regular pattern of dark and bright lines are apparent in the IMAGEs in the CUBE. When viewing the CUBE with **x3d**, click on the button *vertical cut* to display a plot of pixel intensity against image line number. If the IMAGE has had no dark correction or is badly dark corrected a regular *saw-tooth* pattern will be easily discernible in the lines away from any source signal. If the CUBE has good dark correction then the plot should just reveal background noise.

- **Stabilisation:** CAM's CCD detectors can be slow to stabilise when the intensity of the incident light changes. This is most evident when a strong source comes in to a pixel's FOV. In **x3d** click on the button *temporal cut* and then on any bright pixel in the image window. If the pixel contains a glitch, then you may see that it takes several read-outs to recover. If the pixel contains a source then it is very likely that you will see a slow response or transient behaviour in the pixel to the source detection.

  If the PDS has been calibrated, without transient fitting, i.e. with *s90* or m90, then click on the button *mask* to look at pixels flagged in the MASK[3]. The detected unstable pixels and dead pixels (column 24 in the LW detector) will now be marked in the plot window. If calibration has been performed well then only those pixels which exhibit transient behaviour will be masked.

[3] If !mask=1 (complex) then all instances of non-zero mask are indicated

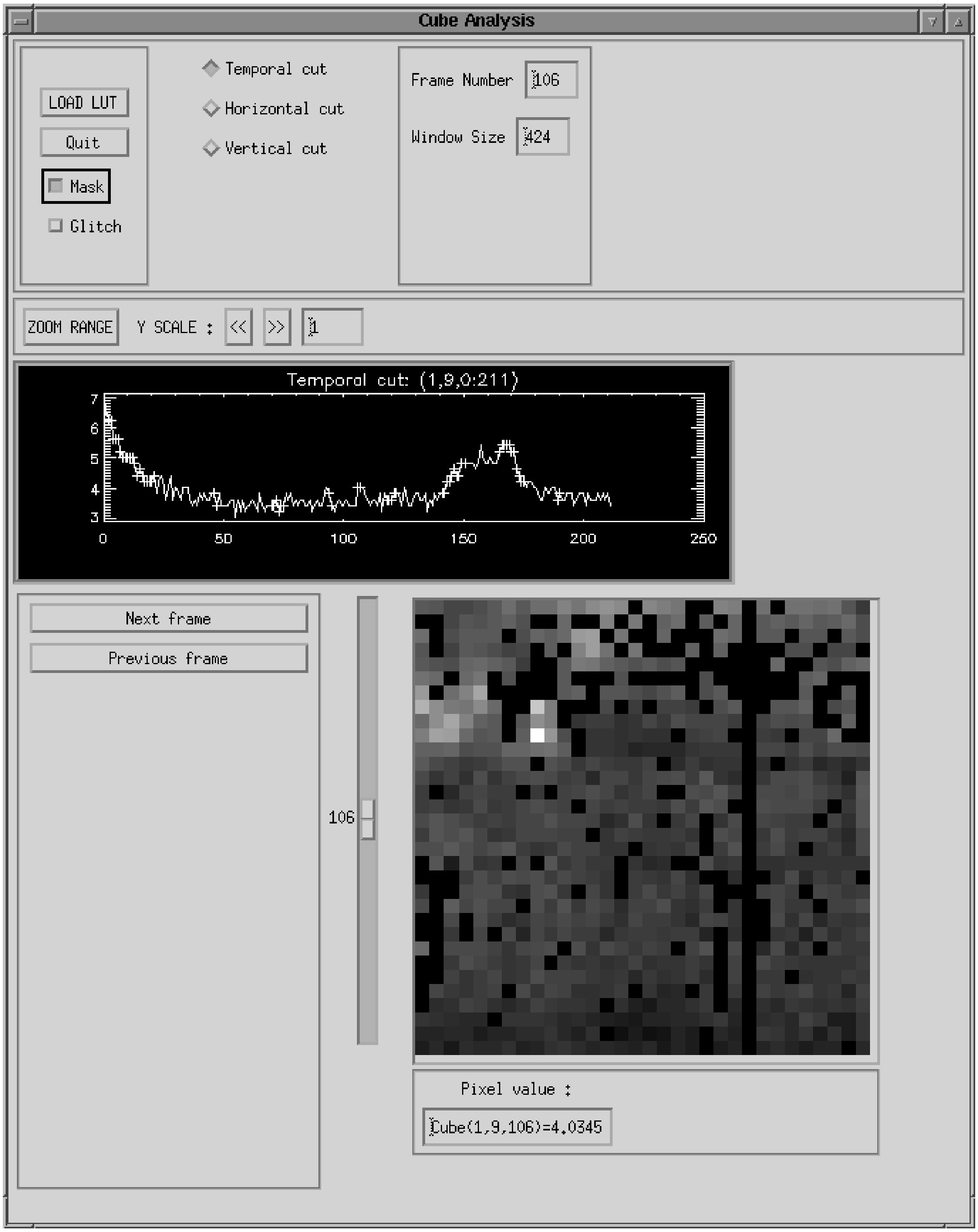

Figure 14.6: **x3d** window.

Other methods of stabilisation which involve transient fitting, i.e. *fs*, *inv* and *vision*, may flag pixels in the MASK that are to be discarded in future analysis and modify other unstable pixels in the CUBE. The flagged or masked pixels can of course still be seen by clicking on *mask* and the temporal behaviour of the modified pixels of the CUBE can be compared with that of the unmodified CUBE.

- **Glitches:** Looking at any single 2D CAM image it may be easy to confuse a glitch with a source. However, if you look at the temporal behaviour of a pixel in the IMAGEs accrued by CAM in a STATE (i.e. DATA in an SCD), or in the CUBE from a PDS, then glitches are usually easily distinguishable from sources. A CAM observation may be made of many STATEs, each STATE containing many IMAGEs of the same sky position. In principle, when CAM is pointing at a source then source signal will be present in all the IMAGEs of that particular STATE, but most glitches are of such a transient nature that their duration will be quite short, though intense, appearing in one or possibly a few IMAGEs.[4] Therefore in most cases, sources will appear in temporal space as relatively low-intensity long duration signal and glitches will appear as spiky intense events.

  Click on *temporal cut*. Scroll through the images of the CUBE until you find an IMAGE containing a pixel of high intensity. Click on the pixel and a plot of its history will appear in the plot window. The shape of the plot should reveal whether the selected pixel of the current IMAGE was hit by a cosmic ray or is a source detection. A well deglitched CUBE will have few or no spiky pixels.

  Glitches can be manually removed by clicking on the button *glitch* and then clicking on a pixel in the image with the middle mouse button. The selected pixel will be masked and the MASK will be automatically updated (note that the MASK should not be supplied to **x3d** as an expression if the update is to be saved). See Section 20.2.2 for alternative manual deglitching methods within CIA. Note that there are two other manual deglitchers, **man_mask** and **sl_viewcube**, in the *contrib* directory of CIA.

14.4.6 xv_temp

xv_temp is yet another tool for examining cubes. It does however offer different functionality from **x3d** and **xcube**.

It can be invoked simply as:

```
CIA> xv_temp, any_pds.cube, to=any_pds.to
```

Take a look at Figure 14.7. Note the following points:

- The image displayed is the mean of all the IMAGEs in *any_pds.cube*. A white box surrounds a feature in this image – we will call the pixels enclosed within this box the *region*.

- The position of the *region* can be changed by clicking anywhere on the image. Clicking the button *max* will centre the *region* on the image pixel of maximum intensity. The buttons marked $1x1$, $3x3$, etc., can be used to quickly set the size of the *region*. Alternatively, more customized sizes can be set with the sliders.

[4] All of this depends of course on such factors as the integration time, the number of frames per state, the response of detector to the source and the intensity of the glitch.

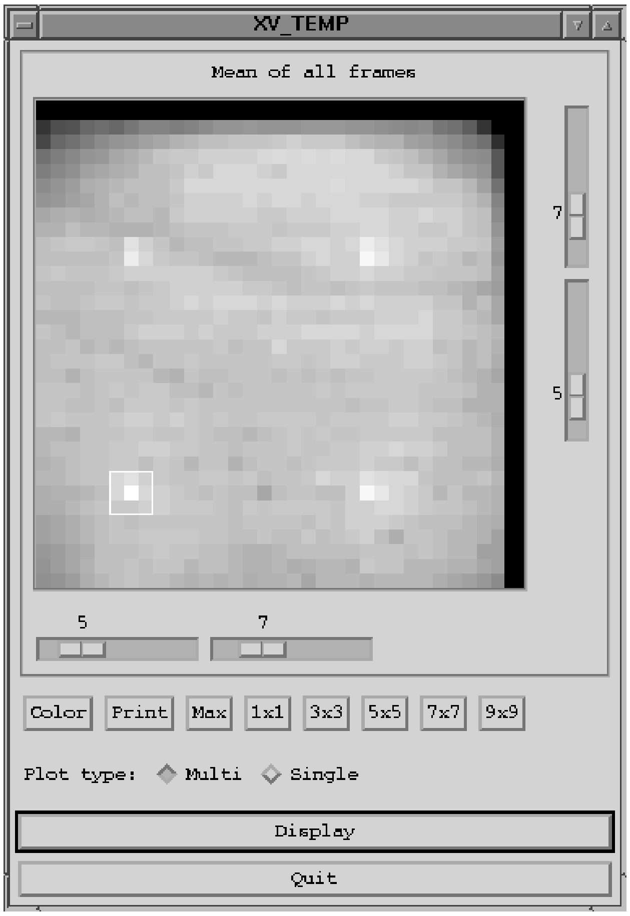

Figure 14.7: **xv_temp** window.

- A plot of the history of each pixel in the *region* will be displayed by clicking on the button *display*. There are two possible plot types: overlay of all histories (click on *single*) or individual pixel history plots (click on *multi*).
- A postscript file of the plot, *xv_temp.ps*, can be create by clicking on *print*.
- The button *color* will invoke IDL's **XLOADCT**.

14.4.7 ximage

This section[5] introduces you to CIA's **ximage**. This is a widget program for astronomical image display and was designed to mimic SAO-image. What makes this tool different from the rest is that it provides you with a variety of plotting, input and output options, and that it can show you the pixel histories in the context of the raster MOSAIC. This gives you a better idea how each pixel of the PDS cube affects the outcome of the raster MOSAIC, and permits especially to verify that apparent sources are not remainders of glitches or transients.

14.4.7.1 Getting started

The input of **ximage** can be one image with or without astrometry, in various formats:

- 2-D IDL array
- 2-D IDL array with a FITS header
- 2-D IDL array with and an IDL astrometry structure
- a *raster* PDS (the image is taken from .RASTER and the astrometry from .ASTR)
- a file containing data in one of the following formats: FITS, IDL saveset (xdr), MIDAS format (bdf)

You can start **ximage** without any arguments and then load an image from a file using the file menu.

14.4.7.2 Banner

ximage provides five menus in its banner: File, Scale, Tools, Zoom, Help. The contents of each of these menus is described below.

File
- Load: you can load an image with the various input formats described above just by clicking.
- Save: you can save the image as a PostScript, GIF, TIFF or FITS file, with or without the color bar.
- Plot image: you can plot the image with or without the color bar this image is in a new window. This window is kept after exiting **ximage**. The title of this plot is given by the sub-window labelled title in the **ximage** widget (see below).
- Quit: exit **ximage**.

Scale
- linear

[5] Taken from Chanial, P. & Gastaud, R., 2000, Ximage manual

- logarithmic
- equalization of histogram

Tools
- maximum: the maximum value and its location is echoed in the terminal and displayed in the intensity and coordinate sub-window.
- profile: not implemented.
- histogram: the histogram of the image is plotted in a new window
- contour: the contours of the image are drawn in a new window.
- 3d surface: the surface in perspective (3D) of this image is drawn in a new window

Zoom
- Reset
- Zoom 2
- Zoom 4
- Zoom 1/2
- Zoom 1/4

Help
- has not yet been implemented in the current version of **ximage**.

14.4.7.3 min, max and thumbnails

The upper panel of the **ximage** window, just below the menus, contains the following features:

- Intensity scaling:
  - Min,Max (text widget): you can type in this text window the intensity thresholds, the image will be displayed according to these values.
  - Fixed/Auto (toggle button): if Auto is selected, then an automatic intensity scaling of the image is done between the minimum and maximum value of the displayed image and NOT the whole image. If Fixed is select, the minimum and maximum values don't change when moving the image.
  - Reset (button): use the minimum and maximum values of the whole image to scale the image.
- thumbnail1: contains the full image reduced to thumbnail size, useful for big images or zoomed images.
- arrows: to move the cursor, useful for small images.
- thumbnail2: displays a zoomed image controlled by:
  1. dragging the mouse on the image: around the mouse position.
  2. dragging the mouse outside the image: around the cursor.

14.4.7.4 Mouse Mode

This panel contains a multiple choice button on the left which changes the menus and buttons on the right.

- Data mode:
  - Raster (toggle button): creates a new window, valid only when the input is a *raster* PDS (see below).
  - Statistic (toggle button): a new window appears, you can select a region and get the statistics of this region.
- Color mode:
  - multiple choice menu: selects the color table
  - Reset (button): resets the luminosity and contrast of the selected color table
  - you can also click or drag the mouse inside the main image.
    * left to right: increases the luminosity.
    * top to down: increases the contrast (γ).
- Overplot mode: in this mode, a button labelled skyview appears, valid only when the input is a raster, it overdraw on the main image the limits of each sky pointing.

14.4.7.5 Title

This panel contains a text widget, with the target name of the image and some information on the configuration (this is valid only for PDS input). The user can type his or her own title, it will be used as title in all the plots created by **ximage** (e.g. File/Plot image or Raster plot).

14.4.7.6 Intensity and coordinates

This panel contains a text widget, only readable, with the intensity and coordinates (with its astrometry).

14.4.7.7 Main Image

The clicking behaviour depends upon the mouse mode.

- Datamode:
  - dragging the mouse on the image: the zoom in thumbnail2, the intensity and the coordinates are those of the mouse position and not the cursor.
  - dragging the mouse outside the image: the zoom in thumbnail2, the intensity and the coordinates are those of the cursor.
  - left click: moves the cursor to the mouse position.
  - middle click: moves the cursor to the mouse position and echoes the intensity and the coordinates in the terminal of the new cursor position. If the input is a raster then additional information about the data cube are printed.
  - right click: moves the image so that the position of the cursor is in the middle of the displayed image.

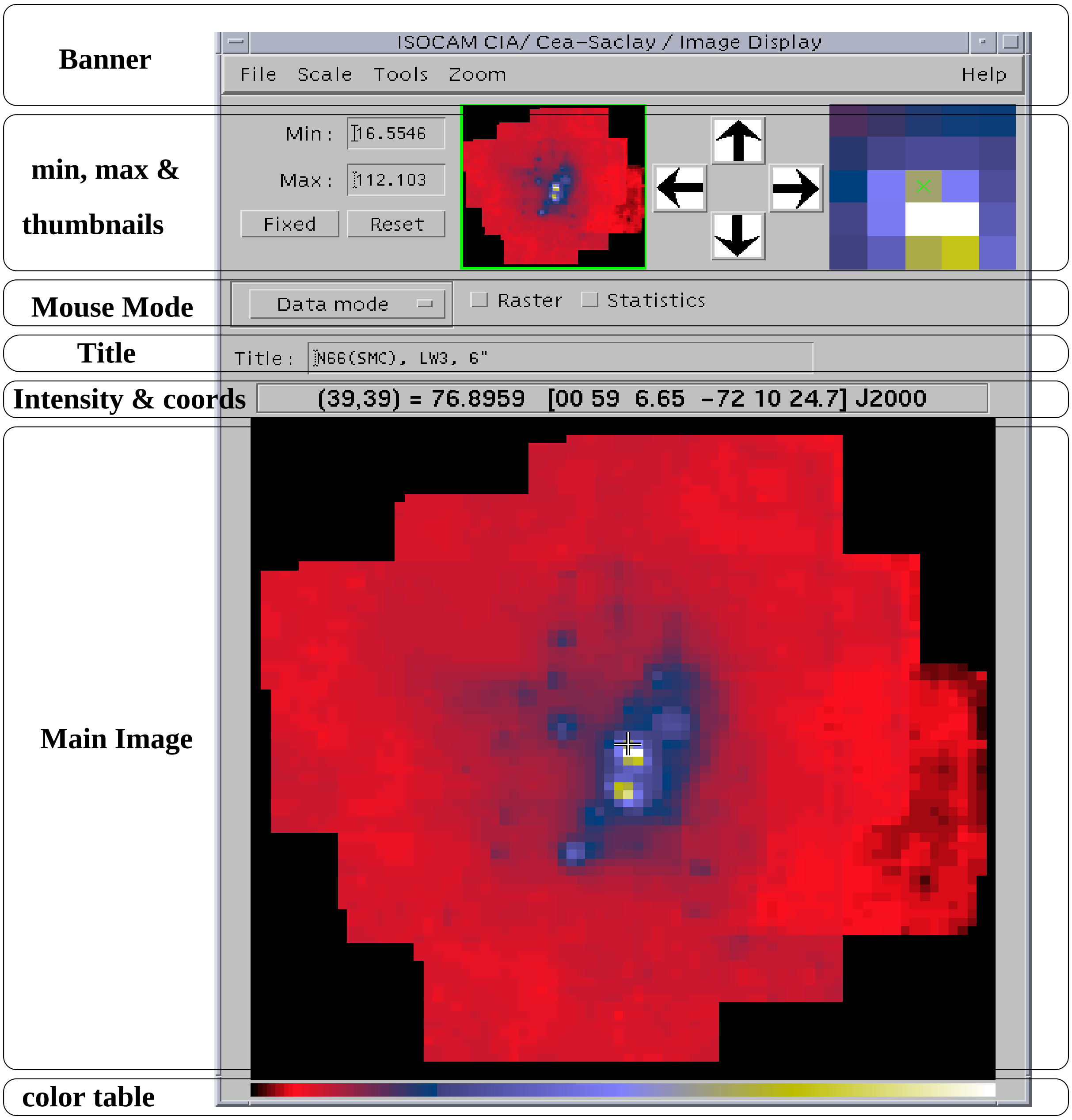

Figure 14.8: The main window of **ximage**

14.4.8 Raster visualization

This functionality only applies for a *raster* PDS (see Sections 15.5 and 15.5.4).

If Data mode is selected, a click on the button *raster* raises a "cube analysis" window. The main image is a mosaic of several sky views which overlap. Each sky view (or SCD) contains several readouts (usually from 10 to 30). This new window contains a plot of the history of all the camera pixels that have contributed to the final flux of the pixel selected by the cursor in the main image. This camera pixels will be called "useful" below. In the banner of the "cube analysis" window, there are three scrolling menus.

- File
  - New window: creates a new window which is a copy of the "cube analysis" window and stays when you quit this widget for further comparison.
  - Save: saves the "cube analysis" window as a PostScript or encapsulated PostScript file.
  - Quit: closes this "cube analysis" window and unselect the raster button.
- Plot
  - Single: in the window there is only one graph. The whole time history of each "useful" pixel camera is plotted. The part of the time history which contributes to the sky pixel is bold, scd limits are drawn around this part, and the scd number is written at the beginning of this part. All the plots are on the same graph (overplotting). This can be useful when there is a gain variation.
  - Multi: The whole time history of each "useful" pixel is drawn, each on a different plot. The part of the time history which contributes to the sky pixel is bold, scd limits are drawn around this part.
  - Compare: for each "useful" pixel camera, only the part which contributes to the sky pixel is drawn. The plots are put one after the other on the same graph, the abscissa is not the real time.
- Options
  - Bad Pixel interpolation (toggle button): pixels flagged as bad are interpolated by the neighbour valid pixels or kept as it.
  - Flat Correction (toggle button): during standard data reduction, only the fields `raster.image` and `raster.raster` of the *raster* PDS structure are flat corrected and not the whole cube (`raster.cube` field). This option corrects the cube for flat-fielding and makes easier the comparison of the flux histories.
  - SCDs Limits (toggle button): this is valid only for the plot options Multi and Single (in the compare mode, the SCD limits are always drawn).

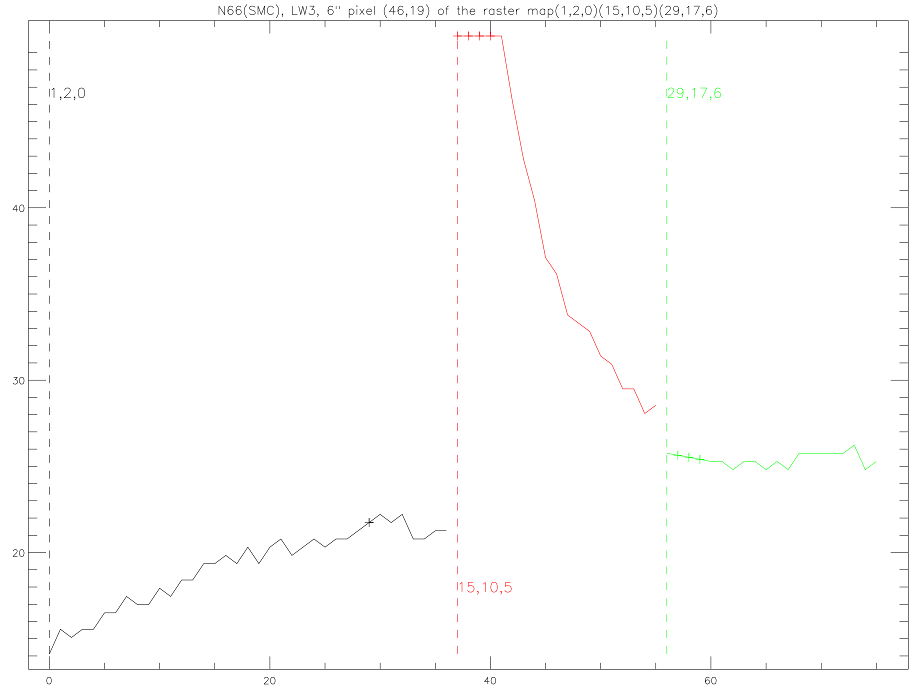

Figure 14.9: The raster window in **ximage**: a downward transient

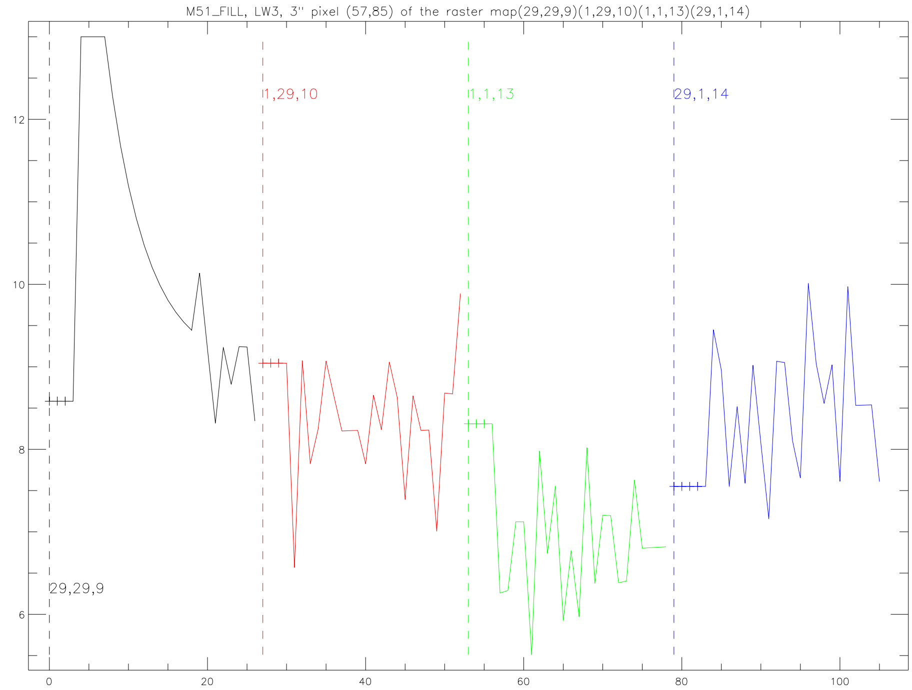

Figure 14.10: The raster window in **ximage**: the tail of glitch

14.4.9 An example

Simply starting **ximage** as,

```
CIA> ximage, raster
```

gives the following output:

```
>> click on data mode multiple choices button, choose overplot mode
>> click on the sky-view button
>> click on overplot mode multiple choices button, choose data mode
>> click on the raster button
>>  move the mouse, click the left button to select a pixel
```

This tool can be used for example to test if one bright point in an image of a *raster* PDS is real or an artifact due to memory effects (ghost) or to glitches.

Figure 14.9 shows a point of the mosaic (46,19) which is a ghost: one of the sky pointing has seen a bright source before and the downward transient correction is not perfect. Figure 14.10 shows a point of the mosaic which is also an artifact: one of the sky pointing contains the tail of long-time glitch (fader).

14.4.10 xv_raster

xv_raster, the pre-curser of **ximage**, is another cube analysis tool. What makes this tool different from the rest is that it is shows you the pixel histories in the context of the raster MOSAIC. This gives you a better idea how each pixel of the PDS cube affects the outcome of the raster MOSAIC.

It can be invoked simply as:

```
CIA> xv_raster, raster_pds, output
```

Note from above that you need to supply the whole PDS to **xv_raster**. The variable *output* will return an array of pixel history data most recently viewed.

Take a look at Figure 14.11. Note the following points:

- The image displayed is the raster MOSAIC of the input PDS (in our example call above this is *raster_pds*). A cross-hair is positioned on the pixel of maximum intensity. The cross-hair can be moved by clicking on the image or by clicking on the buttons *left*, *right*, *up*, *down*, *center*. Clicking on *max* will return it to the pixel of maximum intensity. Clicking on *Raster FOV* will place a grid, representing the FOV of each EXPOSURE, over the raster MOSAIC (as display in Figure 14.11).
- A plot of the history of the pixel under cross-hairs should also appear (this is not shown in Figure 14.11.) Since you are viewing a raster observation, each raster MOSAIC pixel can be made up of several EXPOSURE pixels, one per each SCD or STATE of the observation. For this reason a history of each EXPOSURE pixel is shown in the plot window. Clicking on *multi* or *single* will switch to individual plots or to an overplot of EXPOSURE pixel histories respectively.
- Clicking the button *mask* will mark each masked pixel in the plot. Clicking on *SCD limits* will place markers on the plots at the SCD boundaries.
- A postscript file of the plot, *xv_raster.ps*, can be create by clicking on *print*.
- The button *color* will invoke IDL's **XLOADCT**.

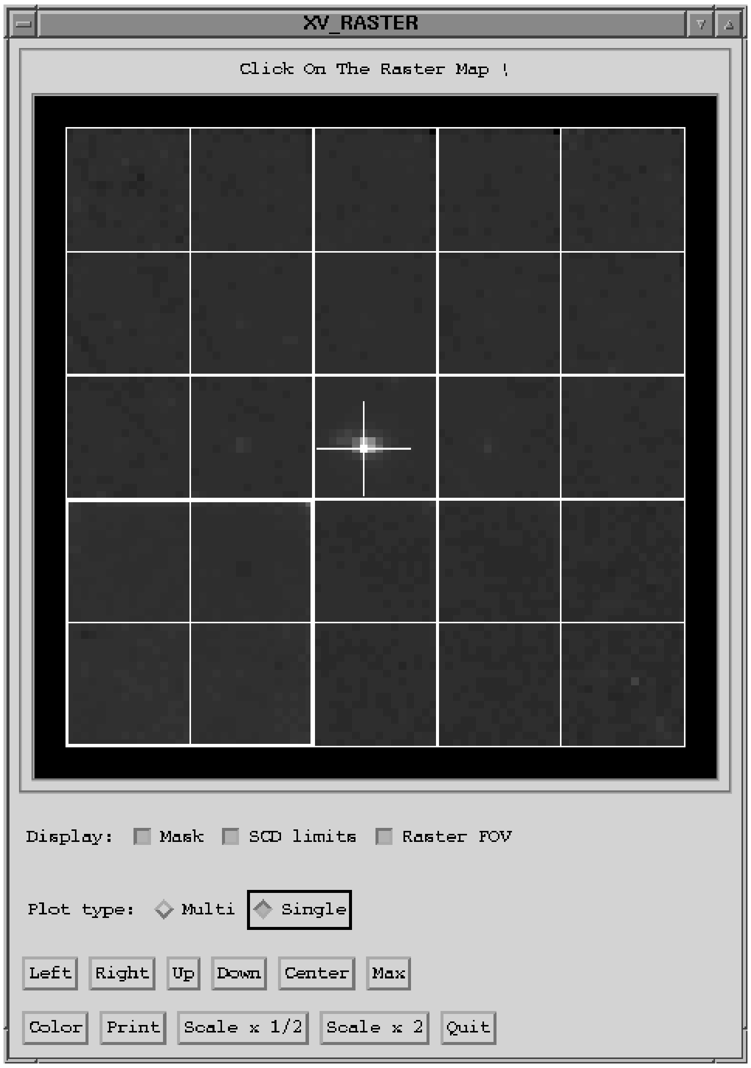

Figure 14.11: **xv_raster** window.

14.4.11 Cube animation with xmovie

An animated movie of all the images in an IDL cube can be displayed with **xmovie**. The cube can be taken directly from a CIA data structure, for example the .CUBE from a PDS:

```
CIA> xmovie, cvf_pds.cube
```

One note of warning: since **xmovie** needs a lot of memory to perform the animation, a large cube may crash it. Don't worry, your data should not be affected.

14.5 Simple image display

14.5.1 tviso

The simplest form of image display is with **tviso**. The purpose of this routine is to display a single image and intensity scale in an IDL window. All images are rebinned to 320×320 for visualisation. Its calling sequence is simple:

```
tviso, image
```

14.5.2 Cube display with show_frame

All the frames in an IDL cube can be displayed in a single window with the routine **show_frame** – see Figure 14.12. Each frame can be enlarged by clicking on it. If there are too many frames to fit in the window, you can scroll through the cube by clicking on the buttons *previous* or *next*. Clicking on *postscript* produces a postscript file of the window. The button *colour* invokes **xloadct** and the button *clone* clones the **show_frame** widget.

As an example, since the EXPOSUREs from a *CVF* PDS is essentially an IDL cube, a call to **show_frame** could be:

```
CIA> show_frame, cvf_pds.image
```

14.6 Image comparison and overlaying

Astronomers may find it useful to do counterpart comparison and overlaying of images. In general the counterpart will be optical and the Digitized Sky Survey (see Section 14.6.1) provides suitable optical images for comparison with CAM images. If you have your own source of images, in FITS format, then these will do just as well. CIA provides **isocont** (see Section 14.6.2) to help you visualise overlays of FITS images and/or CAM images.

14.6.1 Obtaining images from the Digitized Sky Survey

Your site may possess the DSS CD-ROM set. If so, ask your CIA administrator for information on accessing the CD-ROMs. Alternatively, DSS images can be retrieved from the on-line ESO/ST-ECF archive.

- To obtain a DSS image from the on-line ESO archive point your WWW browser at:

  `http://arch-http.hq.eso.org/cgi-bin/dss`

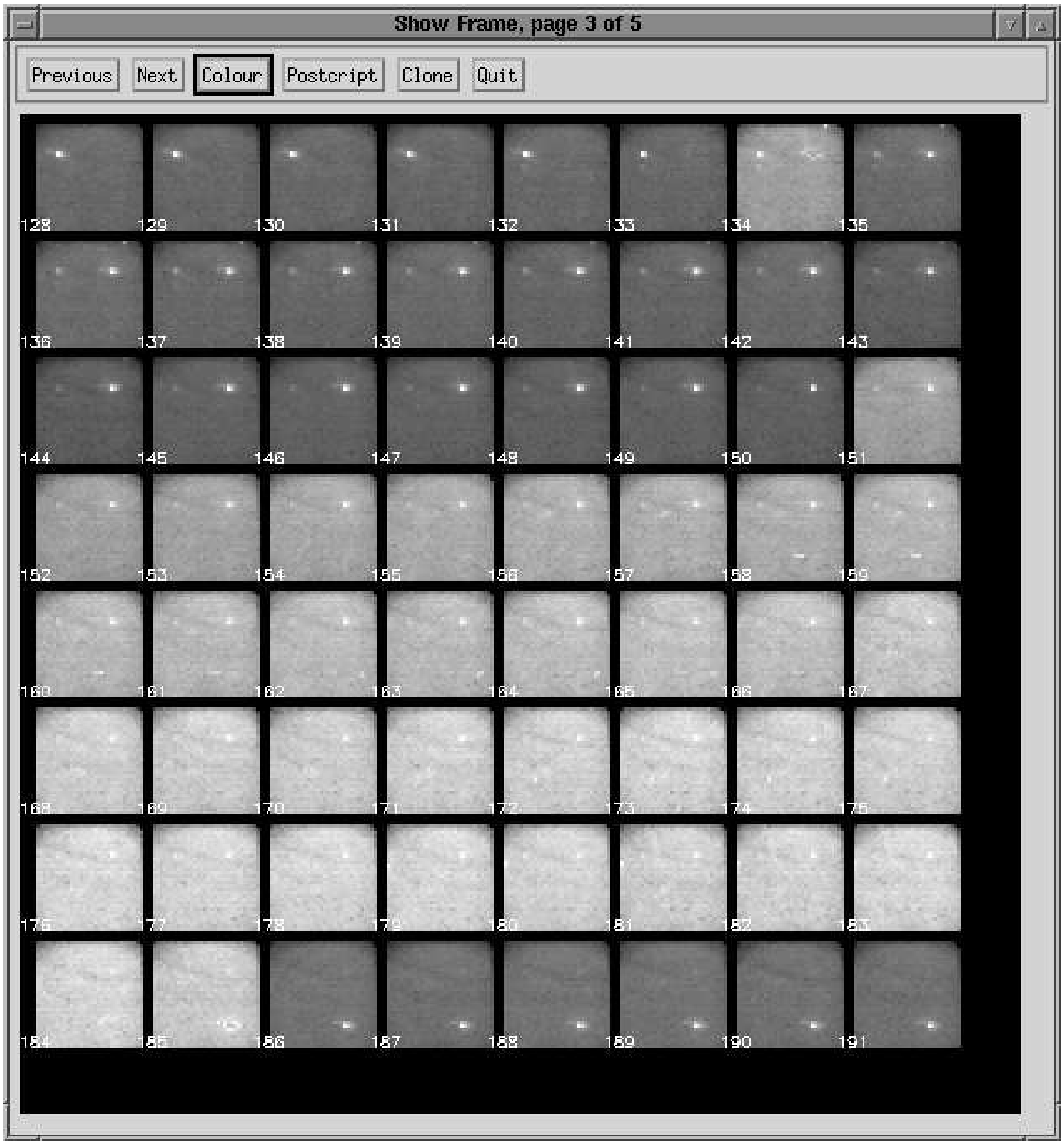

Figure 14.12: **show_frame** window.

You need to supply the astronomical name of the object which you wish to overlay, or its position in RA(J2000) and DEC(J2000), and the size of the image in arcminutes. (Be sure to request an image of size comparable to the CAM image you wish to compare it to.) Further instructions can be found on the Web page.

Note that the images from the DSS are in a particular FITS format (GSSS astrometry), and need to be converted to ISO FITS with the CIA routine **palomar_to_iso**. The first argument is the DSS FITS file and the second is the output ISO FITS file.

```
CIA> palomar_to_iso, 'dss.10.45.22+55.57.35.fits', 'haro3_iso.fits'
```

You can now use your FITS file as input to **isocont** for displaying (see Section 14.6.2) or **xdisp** for analysis (see section 14.3.1).

14.6.2 isocont

We include a only brief description of **isocont** with examples here. For a fuller guide to its use your attention is drawn to – Claret A., Charmandaris V., Gastaud R., 1997, *A Learning Guide for ISOCONT*, v1.0.

isocont accepts two arguments, either one can be a FITS image or a CAM image in a raster data structure[6]. (Because **isocont** needs astrometry for both images, the CAM image must be contained in a raster data structure and the entire structure must be passed as an argument.) Whichever image is the first argument, its astrometry is used as a reference, and the image of the second argument is rotated or aligned so positions and orientation of both images match. The first argument is plotted in grey-scale or colour, and the second as contour levels. Figure 14.13 shows an optical image in grey-scale overlayed with contours from a CAM image.

Examples of **isocont** usage are:

- Following the examples of Section 12.2 and Section 13.2, our first example here displays an optical image of Haro 3, overlayed with a CAM raster image. The keyword *mag* magnifies the image for readability.

  ```
  CIA> isocont, 'haro3.fits', lw6_raster, mag=4
  ```

  This command produced the image in Figure 14.13.

- Following from the example in Section 14.3.1.2 we can overlay the **xdisp** generated FITS file, which contains a 'zoom' of the LW6 CAM raster image of Haro 3, with contours from the optical image of Haro 3. The keyword *putinfo* adds information to the window: the intensity scale, contour level values etc.

  ```
  CIA> isocont, 'ima33x33.fits', 'haro3.fits', /putinfo
  ```

- This example is similar to the previous one, except the keywords *radec* and *scan* are set. Respectively, these display the image oriented in standard astronomical fashion (i.e. orienting the +DEC axis upwards and labelling it in *degs min secs* and orienting the +RA axis leftwards and labelling it *hrs mins secs*) and indicate the individual positions of each CAM state in the raster.

[6] The CAM raster data structure must be present in the memory of the CIA session and the FITS image must be accessible on disk.

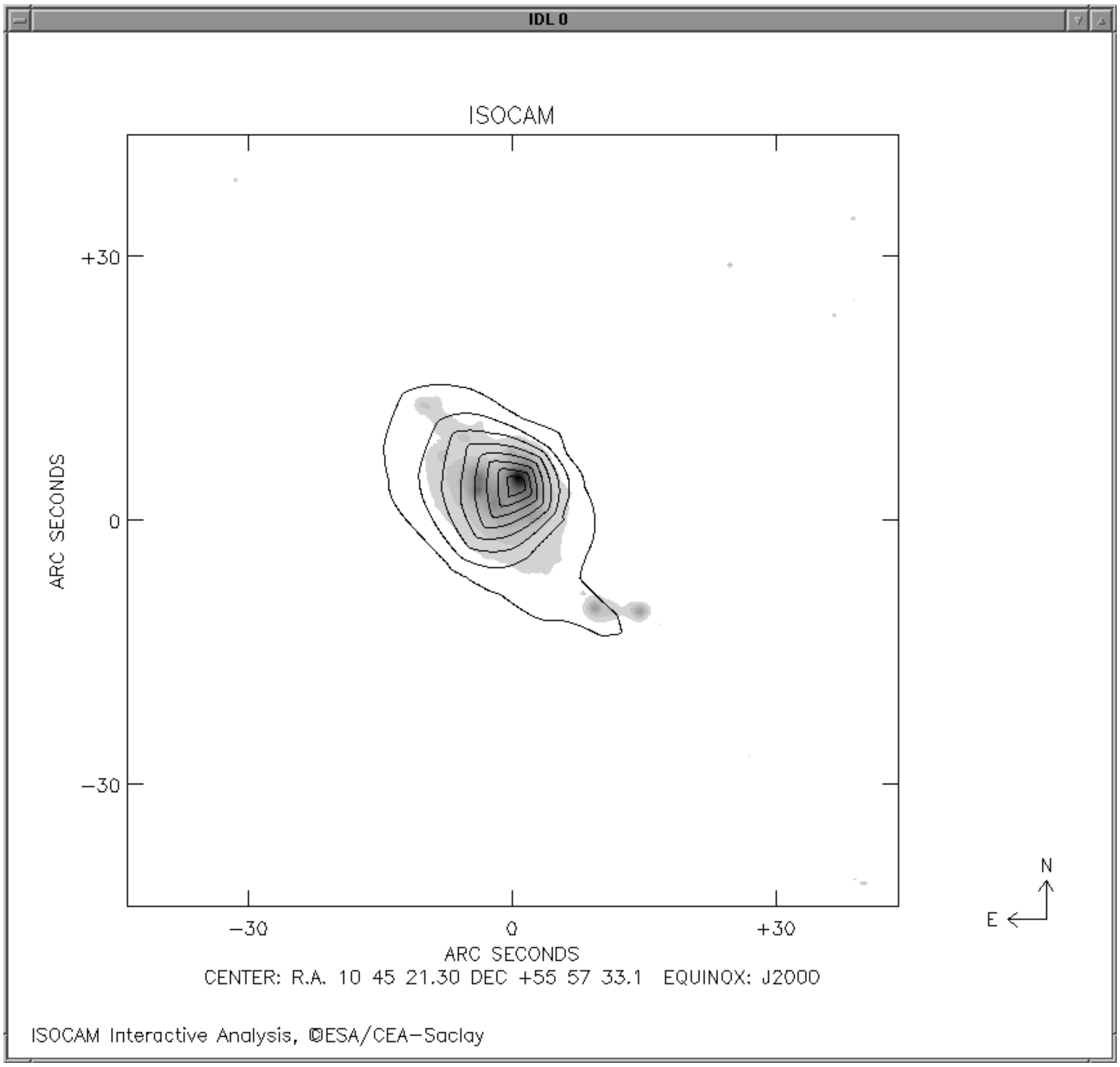

Figure 14.13: **isocont** is used to overlay an optical image with contours from a CAM image. The CAM image has been aligned and rotated to match the astrometry of the optical.

```
CIA> isocont, lw6_raster, 'haro3.fits', /radec, /scan
```

- The final example overlays image from two CAM configurations, both raster images of the same object. *radec* is set to orient them in the standard astronomical fashion. The keyword *shift* will make **isocont** attempt to match the images not only by astrometry but by correlation also. This helps to compensate for possible pointing inaccuracies.

```
CIA> isocont, lw6_raster, lw3_raster, /radec, /shift
```

14.6.3 x_isocont

x_isocont is a widget interface to **isocont** (see Section 14.6.2). It has been designed for easy experimentation of overlays – rather then specify keywords you can point and click. With **x_isocont** you can easily set contour levels, titles, produce hardcopies etc...

x_isocont is started in the same way as **isocont**, but without the keywords, e.g.:

```
CIA> isocont, lw6_raster, 'haro3.fits'
```

Looking at Figure 14.14 you can see that it is split into several panels and drop-down menus. Click on *info / help* for an online help description of the functions of the drop-down menus.

The functions of the panels are described below, with the corresponding **isocont** keyword given for each.

Input Images Displays names of files or data structures. The data structure name is taken from the field .SAD_NAME. Check buttons can be set to switch on/off display of images. The button *FOV* is equivalent to the keyword *scan*.

Astrometry Parameters Menus for setting astrometry parameters. These parameters are equivalent to keywords *epoch*, *ref*, *magnify*.

Contour Levels Set number of contour levels, minimum and maximum values, provide your own custom levels etc. Contour colours can also be set. These parameters are equivalent to keywords *nlevels*, *min_value*, *max_value*, *levels*, *black*, *c_style*, *c_color*.

Optional Overlays Allows some extra features to be added to the plot. These parameters are equivalent to keywords *grid*, *nonorth*, *putinfo*, *title*, *star*.

Register Parameters Set shift configuration. **isocont** can perform manual shifting of the images, i.e. you provide the number of pixels to shift by in x and y directions, or automatic shifting by image correlation. These parameters are equivalent to keywords *shift*, *offset*.

Display Parameters Set parameters equivalent to keywords *nosample*, *rect*, *missing*.

Window Parameters

Font Parameters

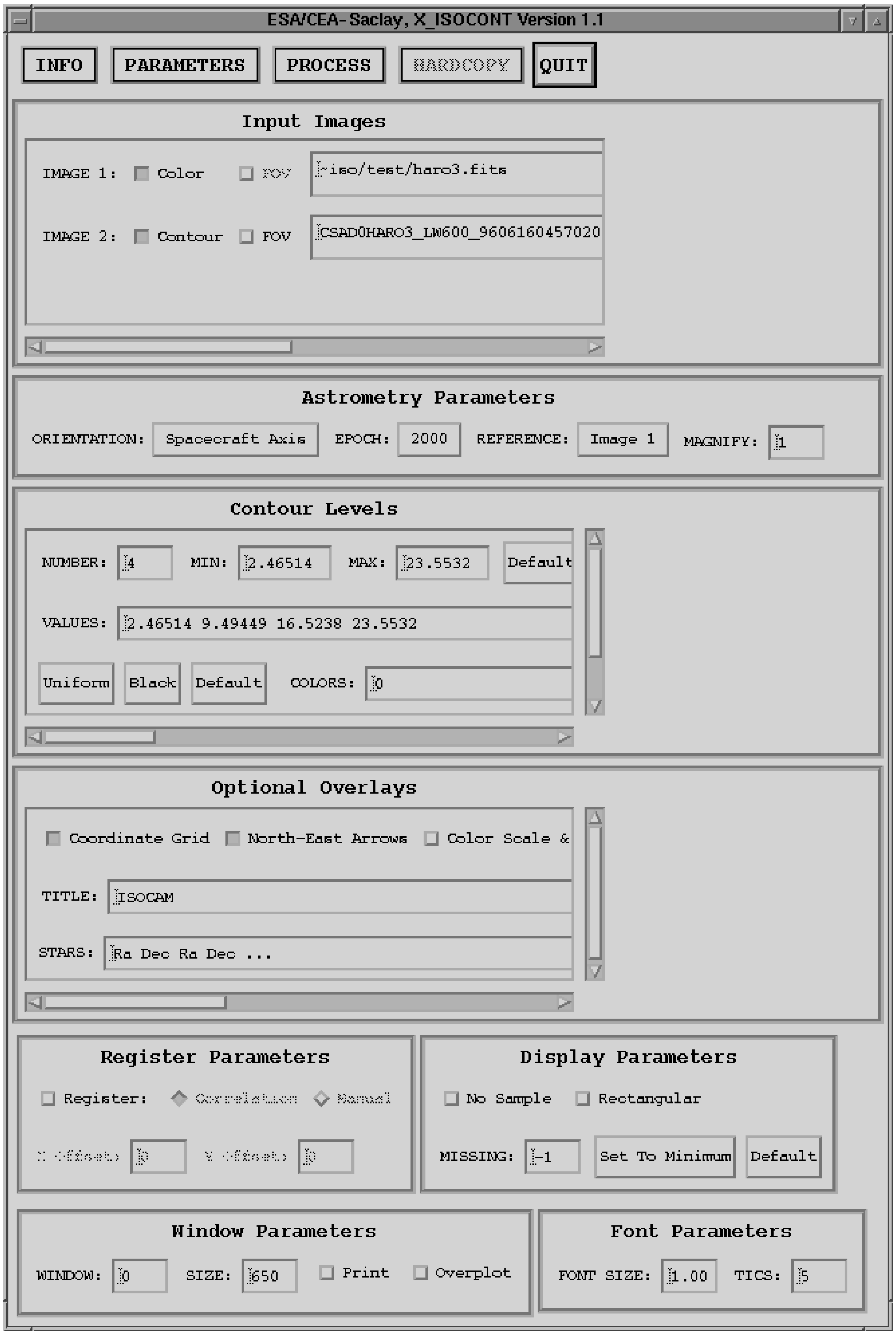

Figure 14.14: **x_isocont** window.

14.7 Creating hardcopy plots

14.7.1 Using xcontour

This is a widget-based program for producing contour plots for screen display and postscript output. You call it with a single image, specifying the window for display (keyword *win*) if you wish:

```
CIA> xcontour, raster_pds.raster, win=1
```

xcontour presents you with a widget for choosing contour levels and contour label sizes.

- Click on *apply* when you want to see your contours plotted.
- Click on *annotate* to display a widget which allows you to annotate text and graphics to your plot. This is usual for marking regions of interest in an image. *options* gives you the postscript output options available. Options for producing an output file are available in the drop-down menu under the push button *file*.
- Click on *print* to screen dump the graphics window to a postscript file.

14.7.2 Screen dumps with ps_color

ps_color produces a postscript or GIF format file from the contents of an IDL graphics window. This routine has keywords for: specifying a filename for the output file, choosing the window to dump to the file, setting the orientation of the postscript output to landscape, passing a title for the plot and placing the ESA/CEA copyright notice in the plot window margin. Note that **ps_color** needs sufficient room in the margins for placing text. If the margins are not wide enough for this you may have to adjust your plot (see the *IDL User's Manual*). Each of these keywords appear in the following example in the order in which they have just been described.

```
CIA> ps_color, filename='test.ps', win=0, /landscape, title='hhh', $
CIA> /copyright
```

Alternatively, **ps_color** can produce GIF output:

```
CIA> ps_color, filename='test.gif', win=0, /gif
```

You may find the routines **white** and **black** useful with **ps_color**. These routines change the colour table to black on white and back to white on black, respectively, making your output plot more readable (and saving printer toner).

14.8 Redirecting graphics to the postscript device

ps_open and **ps_print** are used to redirect graphics to the postscript device. The former opens an output postscript file and the latter closes the file.

Several keywords can be supplied to **ps_open**. In the example below the output file to be created contains a plot in portrait orientation, in colour and in encapsulated format. Without setting the keywords it defaults to landscape, grey-scale and normal postscript, respectively. The name of the output file is *idl.ps*.

```
CIA> ps_open, /portrait, /color, /encapsulated
```

Any plotting routine that does not specify a window can be used to direct output to *idl.ps*.

```
CIA> contour, raster_pds.raster
```

The following call to **ps_print** will rename *idl.ps* to *raster.ps* and print it to a printer named *e15* – deleting the old file in the process.

```
CIA> ps_print, 'raster', /delete, printer='e15'
```

14.8.1 Avoiding postscript problems

In this section we will address some problems which are commonly experienced when printing to IDL the postscript device.

- Because postscript device pixels are scalable, any output graphics will expand to fill the entire postscript device area. This means that a square image on your screen could end up as a rectangle in your postscript file.
- Modifying the colour table (by using **XLOADCT** for example) can mess up the IDL's internal colour table. This can give unexpected results when you output colour graphics to the postscript device.

The above problems can be avoided by following the procedure in the example below.

1. Create a new window by calling **WINDOW** with keyword *retain* set to 2.

   ```
   CIA> window, 1, retain=2
   ```

2. Display your image.

   ```
   CIA> tviso, image
   ```

   Modify the colour table (with **XLOADCT**) as you desire.

3. Save the displayed image using **TVRD**.

   ```
   CIA> image_get = tvrd(1)
   ```

4. Now open the postscript device

   ```
   CIA> set_plot, 'PS'
   ```

   Determine the graphic size.

   ```
   CIA> ss = size( image_get )
   CIA> x = ss[1]
   CIA> y = ss[2]
   ```

If x is greater then y then

```
CIA> device, xsize=n*x/y, ysize=n, yoffset=27-n, bits=8, /color
```

where n is less then 27 cm.

If x is less then y then

```
CIA> device, xsize=n, ysize=n*y/x, yoffset=27-(n*y/x), bits=8, /color
```

where n is less then 18 cm.

The **DEVICE** keywords *xsize*, *ysize* and *offset* determine the x-axis size, y-axis size and offset from $(0, 0)$ of the postscript device area. The keyword *color* is set for obvious reasons.

5. Output the graphic

```
CIA> tv, image_get
```

6. Close the postscript device.

```
CIA> device, /close
CIA> set_plot, 'x'
```

Part III

Data Management

Introduction

The purpose of this part of the *CIA User's Manual, Data Management* is to provide a guide on how data is managed by CIA.

- Chapter 15 presents an overview of the data architecture of the CIA data structures.
- Chapter 16 describes the CIA data structure user interface and handling routines.
- Chapter 17 describes CIA routines for converting FITS data products into CIA data structures and ordinary IDL data structures.
- Chapter 18 shows you how to export/import CIA data structures to/from FITS.

Chapter 15

CIA data structure high-level architecture

This chapter describes the high-level architecture of CIA data structures. It is hoped that it will help you to understand how your data is organised within CIA. Examples are included to aid description of the data structures. These examples include CIA manipulation routines which may be unfamiliar to you – in such cases refer to Chapter 16 for a description.

15.1 Introduction

The data is managed in a CIA session by purpose designed data structures. These structures may seem large and unwieldy to the novice user, but experience has shown that they are the most convenient way of managing CAM data. AOTs can be quite complex and many CAM FRAMEs may be accrued during a typical observation. In addition the many CAM parameters that exist need to be stored along with the FRAMEs, IMAGEs, EXPOSUREs and MOSAICs. The structures attempt to neatly store all these data. CIA makes the structures as transparent as possible to the user by using dedicated manipulation routines (see Section 16), or by allowing you to transform CIA data structures into ordinary IDL structures (PDS). It is hoped that this chapter will help you work with the structures, but not get too deeply involved with their architecture.

The structures are broadly grouped into those containing observation data (see Section 15.2) and those that contain calibration data (see Section 15.3).

15.2 Observation data structures

Observation data structures are designed to hold actual image data from a CAM observation and detailed information about CAM and ISO parameters during the observation. Where possible the data is presented in fields of the structure in a user-friendly format, i.e. the original telemetry coded parameters have been converted to more readable strings.

The structures containing observation data are outlined below and a reference to a more detailed section is given with each description.

Science CAM Data (SCD) – ERD/SPD Level The SCD structure has two flavours, one is used to hold data of ERD product type, the *ERD* SCD, and the other of SPD product type, the *SPD* SCD. The *ERD* SCD contains all the EOI and RESET FRAMEs from a single STATE and in addition parameters describing that STATE, e.g. coordinates, lens, filter, etc.... The *SPD* SCD differs primarily in that it holds IMAGEs which have been computed from the EOI and RESET FRAMEs. These IMAGEs are either directly taken from the CISP data product or computed by CIA from the FRAMEs in the *ERD* SCD. See also Section 15.2.2.

Set of Science CAM Data (SSCD) The SSCD is primarily designed to catalogue a set of SCDs (either *ERD* SCDs or *SPD* SCDs, but not both together) belonging to the same CONFIGURATION and variables which describe that CONFIGURATION. However, it may be used to catalogue any number of STATEs: from all the STATEs in a AOT down to a single STATE. See also Section 15.2.3.

Science Analysed Data (SAD) – AAR Level Contains two EXPOSUREs, one calibrated in detector coordinates (from the CCIM data product) and the other calibrated in celestial coordinates (from the CMAP data product). The SAD is also used to hold EXPOSUREs (from the CMOS data product). In addition to AAR data products, the SAD may be used to hold CIA calibrated EXPOSUREs and MOSAICs. See also Section 15.2.4.

Set of Science Analysed Data (SSAD) Contains a catalogue of a set of SADs. Its function is analogous to that of the SSCD. In general, best use of the SSAD is made when it catalogues SADs that belong to a single CONFIGURATION. However, it may catalogue any subset of SADs from an AOT. See also Section 15.2.5.

Diagnostic Specific Data (DSD) Contains physical parameters of the camera: temperatures, voltages, wheel positions, etc. It is directly used only by instrument experts for in-depth investigation of ISOCAM behaviour. Such data that is of interest to the normal user is also held in the SCD. Documentation of the DSD structure is beyond the scope of the *CIA User's Manual*[1].

15.2.1 Standard fields of observation data structures

There is some similarity between observation data structures in their architecture, that is to say standard fields exist in all. Generally, they contain information which is relevant to each data structure, e.g. it is necessary to know the PFOV during an observation whether you have raw FRAMES in a *ERD* SCD or calibrated IMAGEs in a SAD, but it is not necessary to keep RESET FRAMEs in the SAD. This section describes these standard fields. Later sections describe fields particular to each data structure.

1. **NAME:** The data structure name is unique to each data structure. The name is used to store the structure on disk and as a pointer to the structure in memory. If you use **x_slicer** to slice your data then you have some choice in the selection of your own naming scheme, see Section 12.3.7. Otherwise, if you are creating the data structures using IDL conversion routines of Chapter 17 then each data structure is named by either one of two conventions, as follows.

   If the TDT and OSN is defined then the following convention is followed:

   CDDDttttttooccss_yymmddhhmmssdd

   where,

[1] Further details on the DSD may be found in the *ISO Data Product Document* or the *CAM Parameter Characteristic Document*, 1991, ref ISOCAM ST110.

variable	definition
DDD	structure type abbreviated name, i.e. SCD, SAD, SSC(D), SSA(D)
tttttt	TDT number
oo	observation sequence number
cc	configuration number
ss	state number
yy	year of creation
mm	month of creation
dd	day of creation
hh	hour of creation
mm	minute of creation
ss	second of creation
dd	0.01 second of creation

Otherwise, if neither TDT or OSN is defined then the convention is:

C*DDDYYMMDDHHMMSS*_*yymmddhhmmssdd*

where,

variable	definition
DDD	structure type abbreviated name, i.e. SCD, SAD, SSC(D), SSA(D)
YY	UTC at beginning of data (year)
MM	UTC at beginning of data (month)
DD	UTC at beginning of data (day)
HH	UTC at beginning of data (hour)
MM	UTC at beginning of data (minute)
SS	UTC at beginning of data (second)
yy	year of creation
mm	month of creation
dd	day of creation
hh	hour of creation
mm	minute of creation
ss	second of creation
dd	0.01 second of creation

2. **AOCT:** AOT number. **Type:** integer.

   This field can have the following values:

value	description
0	CUS use only
1	CAM01: staring, raster, micro-scan or tracking
2	CAM02: not used (historic reasons only)
3	CAM03: beam-switch
4	CAM04: cvf
5	CAM05: polarization

3. **TARGET:** Target name as it appears on original proposal. **Type:** string. e.g. 'HARO 3'

4. **OBSERVER:** Observer's name as it appears on the original proposal. **Type:** string. e.g. 'LMETCALF'

5. **TDT:** TDT number. **Type:** integer.

6. **OSN:** OSN number. **Type:** integer.

7. **F_RASTER:** Indicates the type of CAM01. Set to 'UNDEFINED' for AOTs other then CAM01. **Type:** string.

   The table below lists the possible values of F_RASTER.

value	**description**
'RASTER'	raster observation
'STARING'	staring observation
'MICRO_SCAN'	micro-scan observation
'TRACKING'	tracking observation
'UNDEFINED'	observation is not CAM01

8. **CHANNEL:** CAM channel used in the observation. Possible values are: 'LW' or 'SW'. **Type:** string.

9. **MODE:** String indicating CAM OP-MODE. **Type:** string.

   MODE may have the following values:

value	**CAM mode description**
'IDLE'	idle
'OBS'	performing an astronomical observation
'DARK'	obtaining a dark frame
'FLAT'	obtaining an internal flat-field image
'CLEAN'	detector is being cleaned

10. **FLTRWHL:** Filter wheel used in the observation[2]. **Type:** string. e.g. 'LW6'.

11. **GAIN:** Detector electronic gain. Possible values: 1, 2, 4. **Type:** integer.

12. **TINT:** Integration time of each FRAME. **Type:** float. **Unit:** seconds.

13. **PFOV:** Pixel field of view. **Type:** float. **Unit:** arcseconds.

14. **WAVELENGTH:** Wavelength of the current filter. **Type:** float. **Unit:** microns.

[2]See the ISOCAM Observer's Manual for possible values of FLTRWHL.

15.2.2 Science CAM data (SCD)

Following the definition of an SCD at the beginning of Section 15.2 we will look at where the FRAMEs, IMAGEs and CAM parameters are stored. All of these data are stored in the fields of the SCD. Those fields unique to the SCD are listed here, the rest being listed in Section 15.2.1.

1. **N_RASTER:** The position of the STATE in a raster in the N direction (see Appendix E). The first position corresponds to N_RASTER=1. For non-raster SCDs N_RASTER is always one. **Type:** integer.

2. **M_RASTER:** The position of the STATE in a raster in the M direction (see Appendix E). The first position corresponds to M_RASTER=1. For non-raster SCDs N_RASTER is always one. **Type:** integer.

3. **ASNUMBER:** Its meaning depends on the type of on-board processing[3], as indicated by the field .CAL.OBC (see Section 15.2.2.20), which is performed. Type: integer.

   - **.CAL.OBC=0** No on-board processing. ASNUMBER will also be 0.
   - **.CAL.OBC=1** IMAGEs computed on-board and accumulated before transmission to ground. ASNUMBER indicates the number of on-board accumulations:

value	description
0	no accumulation i.e. EOI and RESET FRAMEs in telemetry.
1	two IMAGEs accumulated
2	three IMAGEs accumulated
⋮	⋮
15	16 IMAGES accumulated

   - **.CAL.OBC=2** Only sampled IMAGEs in telemetry. ASNUMBER indicates the sampling frequency. E.g. ASNUMBER=4 implies that 1 in 4 IMAGEs are transmitted to ground.

4. **CRPIX1:** RA reference pixel of IMAGE in FITS convention[4]. **Type:** float.

5. **CRPIX2:** DEC reference pixel of IMAGE in FITS convention[4]. **Type:** float.

6. **RA:** RA(J2000) of CRPIX1. **Type:** double. **Unit:** decimal degrees.

7. **DEC:** DEC(J2000) of CRPIX2. **Type:** double. **Unit:** decimal degrees.

8. **ROLL:** The spacecraft roll (see Appendix E). **Type:** double. **Unit:** decimal degrees.

9. **ENTWHL:** Entrance wheel position (see the *ISOCAM Handbook* for possible values of ENTWHL.). **Type:** string. e.g. 'HOLE'.

10. **SELWHL:** Selection wheel position (see the *ISOCAM Handbook* for possible values of ENTWHL.). **Type:** string. e.g. 'FABRY MIRROR LW'.

[3]Usually, on-board processing only takes place when TINT=0.28s or for CAM parallel data.

[4]By FITS convention (CRPIX1, CRPIX2) are the coordinates of the *centre* of the reference pixel, counting from 1. E.g. If the reference pixel is indexed in IDL as (9,9), then CRPIX1=10.5 and CRPIX2=10.5.

11. **SIZE:** Number of IMAGEs in the *SPD* SCD, and the number of FRAMEs in the *ERD* SCD. **Type:** integer.

12. **DATA:** Cube containing IMAGEs in the *SPD* SCD. Does not exist for *ERD* SCD[5]. **Type:** integer IDL cube. **Unit:** ADU.

13. **EOI:** Cube containing EOI FRAMEs. Only exists for *ERD* SCD[5]. **Type:** integer IDL cube. **Unit:** ADU.

14. **RESET:** Cube containing RESET FRAMEs – present in both *ERD* SCD and *SPD* SCD[5]. **Type:** integer IDL cube. **Unit:** ADU.

15. **MODEL:** Cube available for holding processed images. MODEL has the same dimensions as DATA or RESET and is only present in an *SPD* SCD. **Type:** float IDL cube.

16. **EOI_MODEL:** Similar to MODEL, though used for processed EOI frames. EOI_MODEL is only present in *ERD* SCD. **Type:** float IDL cube.

17. **RESET_MODEL:** Similar to MODEL, though used for processed RESET frames. RESET_MODEL is only present in *ERD* SCD. **Type:** float IDL cube.

18. **MASK:** A cube of the same dimensions as DATA, EOI or RESET. Each pixel in the MASK corresponds to a pixel in DATA (*SPD* SCD), or to a pair of pixels in EOI/RESET (*ERD* SCD), and records the status of that pixel. Masked pixels have one of two possible byte representations in the MASK. This is determined by the value of the system variable !MASK. Setting !MASK to 0 selects the simple MASK and 1 selects the complex MASK (for configuring !MASK see Section 2.3.4). In both cases the first bit of the MASK value determines if a pixel is good or bad: 0 is good and 1 is bad. The simple representation only uses this first bit so the MASK will be filled with ones or zeros. The complex representation uses the remaining 7 bits to record more information about the masked pixel. For example, if MASK[16,16,0] = 2, then pixel $[16, 16]$ is dead in the first IMAGE of DATA. If MASK[16,16,0] = 4 then the same pixel has been deglitched. In the complex representation a pixel is still only considered to be bad if the first bit of the corresponding MASK pixel is 1, so this means odd MASK values are bad and even are good regardless of what other status the pixel may have. A list of possible types or keywords of flagged pixels is given in the table below. Section 16.4 describes how the MASK may be manipulated. **Type:** integer IDL cube.

type / keyword	rank	description
bad	1	bad pixel
dead	2	dead pixel
glitch	4	deglitched pixels
memory	8	pixel affected memory effects/transients
blind	16	blind pixel (i.e. not illuminated)
qla	32	pixels flagged as bad by qla or csh
spare	64	spare
ext_source	128	extended source

[5]Do not modify this field directly. It contains a reference copy of your CAM image data.

19. **HK:** Substructure containing house-keeping parameters.

    For every IMAGE/FRAME pair there is a set of house-keeping parameters. These are listed in the table below. The index, i, refers to the HK parameters for the i-th IMAGE/FRAME pair. As an example, if you wish to obtain the UTK of the first IMAGE/FRAME pair in the SCD you could try the following:

```
CIA> print, scd_get('hk(0).utk', scd_of_interest)
```

subfield	**description**
HK(i).DMA_COUNTER	Rank of IMAGE/FRAME
HK(i).UTK	Uniform Time Key of frame
HK(i).QLA_FLAG	Quick Look Analysis flag
HK(i).NGLITCH	number of glitches detected in IMAGE
HK(i).STABLE	denotes stability of IMAGE/FRAME: 0=unstable, 1=stable
HK(i).DU	jitter offsets in spacecraft x-axis
HK(i).DV	jitter offsets in spacecraft y-axis

20. **CAL:** Structure containing original encoded parameters taken directly from the data product FITS file (see *ISO Data Product Document*).

    The following example displays the contents of CAL.

```
CIA> cal=scd_get('cal','CSCD143006010214_96090420275809')

CIA> help, cal, /str
** Structure CAL_STRUC_2, 14 tags, length=28:
   TYPE            INT              33
   COMMANDER       INT               1
   TELEMETRY       INT             100
   DEID            INT               1
   MODE            INT               1
   OBC             INT               0
   TINT            INT              36
   EWHL            INT             308
   SWHL            INT              88
   PFOV            INT             360
   FCVF            INT             275
   GAIN            INT               1
   OFFSET          INT               1
   SPARE           INT               0
```

With the exception of .CAL.OBC (see Section 15.2.2.3), these parameters are of little use to the typical user. Their values as they appear in .CAL are encoded. CIA decodes these values into a user-friendly and readable format. In some cases the parameter name itself is changed to make it more intelligible, e.g. FCVF is FLTRWHL in the CIA data structure (see Section 15.2.1).

15.2.3 Set of SCDs (SSCD)

As you now know, the SCD corresponds to a single STATE of the camera. However a meaningful observation is a collection of STATEs, such as all the STATEs of a raster observation or all the wavelength measurements of a CVF scan. The SSCD is a catalogue of SCDs, either *ERD* SCDs or *SPD* SCDs. It may be a catalogue all the SCDs in an AOT or a subset of an AOT, though it is usually wise to create SSCDs that catalogue SCDs from a CONFIGURATION only. In itself, it holds no data other than CONFIGURATION parameters and the names of the SCDs attached to the CONFIGURATION. The SSCD contains the standard fields of Section 15.2.1 and the following additional fields:

1. **NSCD:** The number of SCDs belonging to this SSCD. **Type:** integer.

2. **RASTER_COLUMNS:** Number of steps in a raster observation in the M direction[6]. **Type:** integer.

3. **RASTER_LINES:** Number of steps in a raster observation in the N direction[6]. **Type:** integer.

4. **M_STEP_SIZE:** Step size of raster in M direction[6]. **Type:** float. **Unit:** arcsecond.

5. **N_STEP_SIZE:** Step size of raster in N direction[6]. **Type:** float. **Unit:** arcsecond.

6. **RASTER_ORIENTATION:** The reference for a raster orientation[6]. **Type:** string.

   Possible values are:

value	description
'UNDEFINED'	not a raster observation or incomplete SCD
'NORTH'	raster performed w.r.t. celestial axes
'Spacecraft Y_axis'	raster performed w.r.t. spacecraft axes

7. **RASTER_ROTATION:** Angle of rotation of CAM with respect to celestial axes[6]. **Type:** double. **Unit**: decimal degrees.

8. **NUMBER_OF_REFERENCES:** The number of reference fields in a beam-switch observation (undefined for other AOTs). Possible values range from 1 to 4. **Type:** integer.

9. **NUMBER_OF_CYCLES:** The number of cycles in a beam-switch observation (undefined for other AOTs).

10. **RA_REFERENCES:** RA coordinates of the reference fields in a beam-switch observation (undefined for other AOTs). **Type:** double IDL array. **Unit:** decimal degrees.

11. **DEC_REFERENCES:** DEC coordinates of the reference fields in a beam-switch observation (undefined for other AOTs). **Type:** double IDL array. **Unit:** decimal degrees.

[6]These fields are only defined for a raster or micro-scan observation. See Appendix E for more on CAM angles *vis-à-vis* the raster observation.

12. **WAVELENGTH_START, WAVELENGTH_END, WAVE_INCREMENT:** Applies to a CVF AOT only. WAVELENGTH_START and WAVELENGTH_END are the wavelengths at the beginning and end of a CVF scan. WAVE_INCREMENT is the number of wheel steps between each position in the scan. **Type:** float (WAVELENGTH_START, WAVELENGTH_END) and byte (WAVE_INCREMENT). **Unit:** microns.

13. **RA:** RA(J2000) of the centre of the final MOSAIC that is constructed from the EXPOSUREs in the SCD. **Type:** double. **Unit:** decimal degrees.

14. **DEC:** DEC(J2000) of the centre of the final MOSAIC that is constructed from the EXPOSUREs in the SCD. **Type:** double. **Unit:** decimal degrees.

15. **ROLL:** As for SCD (see Section 15.2.2).

15.2.4 Science Analysed Data (SAD)

The SAD may contain an EXPOSURE or a MOSAIC. In CIA programming terminology the set of EXPOSUREs used to create the MOSAIC is referred to as the *origin* and the actual MOSAIC itself as the *future*. This gives rise to two flavours of SAD: *origin* SAD and *future* SAD.

Two array substructures exist in the SAD data structure: CCIM and CMAP. (Don't confuse these substructures with the data products of the same name, they have been named 'CCIM' and 'CMAP' for historical reasons.) We use these substructures for:

1. **CCIM**
   Holds an averaged EXPOSURE in detector coordinates. It may be used to hold your CIA processed data or AAR from the CCIM data product file.

2. **CMAP**
   In the *origin* SAD it holds an EXPOSURE calibrated in astronomical coordinates. In the *future* SAD it holds a MOSAIC. It may be used to hold your CIA processed data or AAR from the CMAP data product file or the CMOS data product file. Because the size of the MOSAIC may vary, IDL pointers are used to handle the CMAP data structure. The routines of Chapter 16 make this transparent to the user, so you need only think of this structure in terms of fields that you may manipulate. The CMAP substructure is of dimensions: 32×32 for holding an EXPOSURE; dynamic when used to hold a MOSAIC.

The SAD contains all the standard fields of Section 15.2.1 and in addition extra fields associated with the type of analysis which may have been performed on the data. For example, the PFOV is a standard field because it is relevant to an image regardless of the processing, but the unit of pixel intensity, TUNIT, is dependent on how the data has been processed. Below is a list of fields that are considered most useful to the CIA user.

1. **CCIM/CMAP:** Both CCIM and CMAP each contain further subfields. To access these replace *DATA* with either CCIM or CMAP.

subfield	**description of contents**	type
DATA.TUNIT	pixel unit of intensity	string
DATA.RA	RA (J2000) of CRPIX1	double
DATA.DEC	DEC (J2000) of CRPIX2	double
DATA.ROLL	roll of the DATA	double
DATA.CRPIX1	reference pixel	float
DATA.CRPIX2	reference pixel	float
DATA.DATA	actual image data	float IDL array
DATA.RMS_IMAGE	error on image	float IDL array
DATA.WEIGHT_IMAGE	exposure time (secs) per pixel	integer IDL array
DATA.MASK	as for SCD (Section 15.2.2)	byte IDL array

The subfields RA, DEC, ROLL, CRPIX1 and CRPIX2 contain parameters that follow the FITS convention. RA, DEC and ROLL are in decimal degrees. CRPIX1 and CRPIX2 are the x and y coordinates of the reference pixel respectively, and refer to the centre of the reference pixel e.g. for reference pixel $(16, 16)$, CRPIX1=16.5 and CRPIX2=16.5.

2. **N_RASTER:** As for SCD (see Section 15.2.2).
3. **M_RASTER:** As for SCD (see Section 15.2.2).
4. **ASNUMBER:** As for SCD (see Section 15.2.2).
5. **CAL:** As for SCD (see Section 15.2.2).

15.2.5 Set of SADs (SSAD)

The SSAD is a catalogue of a set of SADs, either *origin* SADs or *future* SADs. The SSAD has the same relationship to SADs that SSCDs have with SCDs. It also has similar structure to the SSCD. In addition to the standard fields of Section 15.2.1 it has the following fields:

1. **NSAD:** The number of of SADs belonging to this SSAD. **Type:** integer.
2. **ORIGIN:** If a *future* SSAD, returns the name of the *origin* SSAD which catalogues the *origin* SADs. **Type:** string.
3. **FUTURE:** If an *origin* SSAD, returns the name of the *future* SSAD which catalogues the *future* SADs. **Type:** string.
4. **RASTER_COLUMNS:** As for SSCD.
5. **RASTER_LINES:** As for SSCD.
6. **RASTER_ORIENTATION:** As for SSCD.
7. **RASTER_ROTATION:** As for SSCD.
8. **RA:** RA(J2000) of the centre of the final MOSAIC that is constructed from the EXPOSUREs in the SAD. **Type:** double. **Unit:** decimal degrees.
9. **DEC:** DEC(J2000) of the centre of the final MOSAIC that is constructed from the EXPOSUREs in the SAD. **Type:** double. **Unit:** decimal degrees.
10. **ROLL:** As for SCD (see Section 15.2.2).

15.3 Calibration Data Structure (CDS)

One generic structure is used to contain all of the different types of calibration data. This is the Calibration Data Structure. It contains a dynamic substructure or field named DATA (see Section 15.3.2) and a standard set of fields (see Section 15.3.1). DATA holds the actual calibration data: dark images, flat images, point spread images, etc. The standard fields holds information related to the nature of the CDS structure itself, e.g. the CDS name and size.

The CDS is almost a direct conversion from FITS format data to IDL data structure, the important difference being that the actual image data is scaled, so the BZERO and BSCALE keywords are discarded. So the CDS differs from the CIA data structures in that the CAM parameters are NOT presented in the CIA user friendly format, but as raw values taken from the CAL-G FITS files.

The CDS is not a structure that you need to be very familiar with. It is mainly handled by CIA routines. For example, **get_sscdraster** will automatically obtain OFLT, DFLT and DARK CDSs from the CAL-G archive for calibration of a PDS. However if you wish to view the data in a CAL-G file, you can use CIA conversion routines to make a CDS and convert the CAM parameters to a readable format (we give an example later in this section). Alternatively you may use IDL's Astronomy Library routines and follow the guidelines in the *ISOCAM Handbook*. In any case, a detailed explanation of the CAL-G FITS files can be found in the *ISO Data Product Document*. You may also need to refer to the *ISO Satellite Handbook* and *ISOCAM Handbook* for explanation of CAM parameters.

15.3.1 Standard fields of the CDS

The standard fields of the CDS are listed here.

1. **NAME:** Generally of the form,

   CCG*CCTTTT_yymmddhhmmssdd*[7]

   where,

Variable	Definition
CC	Channel, i.e. LW or SW
TTTT	CDS name of calibration data (see Table 15.3.2)
yy	year of creation
mm	month of creation
dd	day of creation
hh	hour of creation
mm	minute of creation
ss	second of creation
dd	0.01 second of creation

   Note that if the CDS is created from a CAL-G file, i.e. using **calg2cds** (see Section 17.1.5, then *creation* refers to the date of creation of the CAL-G file and NOT the date the CDS was created. Should the CDS have been created using **cds_init** (see section 16.1.1.5), then *creation* refers to the date **cds_init** was executed.

2. **SPARE:** This is an empty array. **Type:** byte 1-D IDL array.

[7] There are some exceptions to this rule, e.g. the GAIN CDS is named *CCIGAIN_yymmddhhmmssdd*.

3. **CREATION:** Contains details about CDS owner and CDS creation time. **Type:** string array.

4. **DATA:** The calibration data is held in this substructure. DATA is in fact an array of IDL structures. Each structure holding a calibration record and the number of structures in the array is equal to NUMBER. (See Section 15.3.2.)

5. **NUMBER:** Refers to the number of records in the substructure DATA. A record corresponds to a single independent calibration measurement (such as a DARK image) and associated CAM parameters.

15.3.2 CDS and CAL-G files

The different types of calibration data (i.e. CAL-G files) and the CDS name for each is given in Table 15.3.2.

An example of how the subfields of DATA would look for a CDS containing DARK images is given below. The index, i, refers to a particular record within the DATA field, where the range of i is: $0 \leq i <$ NUMBER.

subfield	**description of contents**
DATA(i).SPARE	empty array (36*1 bytes)
DATA(i).TINT	integration time of image (CAMTU)
DATA(i).EWHL	entrance wheel step number
DATA(i).SWHL	selection wheel step number
DATA(i).GAIN	electronics gain
DATA(i).OFFSET	electronics offset
DATA(i).TEMPERAT	mean temperature (K)
DATA(i).TRMS	RMS temperature (K)
DATA(i).TMIN	minimum temperature (K)
DATA(i).TMAX	maximum temperature (K)
DATA(i).VOLTAGE	mean bias voltage (mV)
DATA(i).VRMS	RMS voltage (mV)
DATA(i).VMIN	minimum voltage (mV)
DATA(i).VMAX	maximum voltage (mV)
DATA(i).BUNIT	pixel unit of intensity
DATA(i).BLANK	value corresponding to an undefined pixel
DATA(i).IMAGE(*, *, j)	dark images ($j = 0$: data, $j = 1$: error)

We can create a CDS from a CAL-G using **calg2cds** (see Chapter 17 for a description of CIA conversion routines):

For convenience, assign directory paths where the CAL-G data products may be found (see Section C.2.2) to IDL string variables. If you are working with VMS CIA then you could make the following assignment:

```
CIA> calg_dark_dir = 'DKA200:[MDELANEY.P0007780.14300601.OTHERS.CCGLWDAR]'
```

If you are using UNIX, the notation is:

```
CIA> calg_dark_dir = '/home/mdelaney/p0007780/14300601/others/CCGLWDAR'
```

CAL-G file	CDS mnemonic	description
basic **calibration libraries:**		
CSCGCROSS	CROSS	SW noise/cross talk decorrelation matrices
CHCGCONV	-	house keeping interpolation values
CCG*DEAD[a]	*DEAD	dead pixel map
CCG*DARK[a]	*DARK	dark current exposures
CCG*DFLT[a]	*DFLT	detector flat field library
CCG*OFLT[a]	*OFLT	optical flat field library
CCG*SPEC[a]	*SPEC	filter & CVF spectral data
CSWCVF	SWCVF	SW CVF description
CLWCVF1	LWCVF1	LW CVF1 description
CLWCVF2	LWCVF2	LW CVF2 description
CCG*SLP[a]	-	CVF spectral line profile
CCG*PSF[a]	*PSF	point spread function library
IFPG	IFPG	focal plane geometry
-	*GAIN	gain conversion table
CWHEELS	-	CAM wheels information table[b]
ORBIT	-	ISO orbital parameters[b]
higher level **calibration libraries:**		
-	*AMDIST	polynomial distortion coefficents
-	*XFLT	distortion flat
CCGLWDMOD	LWDMOD	parameters for the time dependent dark corrections
CCG*TRANS[a]	*TRANS	model transients
CCG*LINEAR[a]	*LINEAR	linearity correction library
CCG*FRAME[a]	*FRAME	detector astrometric calibration
CCG*GLITCH[a]	*GLITCH	glitch Model
CCG*STRAY[a]	*STRAY	non-dark local light model

[a]One CAL-G file or CDS exists for each CAM detector: replace * with *LW* or *SW* for full name.
[b]See Section 15.4 for an alternative handling of these data.

Table 15.1: The calibration data and associated CDS mnemonic used for naming purposes (see Section 15.3.1 for CDS naming convention). A missing entry denotes that a CDS does not exist for that CAL-G file – in the case of the CWHEELS and ORBIT data these are handled in an alternative manner by CIA (see Section 15.4). Refer to Section 9.3.5 for further information on CAL-G files and where they may be sourced. Also your attention is drawn to the *ISOCAM Handbook* and *ISO Data Product Document* for more detailed information on the contents of CAL-G files.

Now convert the CAL-G file, named *K.fits* in our example, to a CDS[8]:

```
CIA> dark = calg2cds( 'K', dir=calg_dark_dir )
 n_rows          5
creating CDS CCGLWDARK_96052312123000
```

Listing the CDSs in memory we see that two CDS exist:

```
CIA> print, cds_list()
CCIGAIN_9504417414600 CCGLWDARK_96052312123000
```

CCIGAIN_9504417414600 is simply a conversion table for the raw values in DATA(i).GAIN.

```
CIA> gain = cds_get( 'data', 'CCIGAIN_9504417414600' )

CIA> help, gain, /str
** Structure CCIGAIN_STRUC, 1 tags, length=12:
   GAIN            FLOAT     Array(3)

CIA> print, gain
{      1.00000      2.00000      4.00000
}
```

CCGLWDARK_96052312123000 or *dark* is the CDS we are interested in. We can extract the DATA substructure with **cds_get** and display its contents.

```
CIA> dark_data = cds_get( 'data', dark )

CIA> help, dark_data
DARK_DATA       STRUCT    = -> CCGLWDARK_STRUC Array(5)

CIA> help, dark_data, /str
** Structure CCGLWDARK_STRUC, 17 tags, length=8572:
   SPARE           BYTE      Array(36)
   TINT            INT              2
   EWHL            INT            308
   SWHL            INT            112
   GAIN            BYTE         1
   OFFSET          BYTE         1
   TEMPERAT        FLOAT     Array(10)
   TRMS            FLOAT     Array(10)
   TMIN            FLOAT     Array(10)
   TMAX            FLOAT     Array(10)
   VOLTAGE         FLOAT     Array(10)
   VRMS            FLOAT     Array(10)
   VMIN            FLOAT     Array(10)
   VMAX            FLOAT     Array(10)
   BLANK           LONG            -32768
   BUNIT           BYTE      Array(12)
   IMAGE           FLOAT     Array(32, 32, 2)
```

[8]Now that you have CDSs in memory, as an alternative to manually playing around with the data structure refer to Section 15.3.3 for a description of the CIA routine **cds_display**.

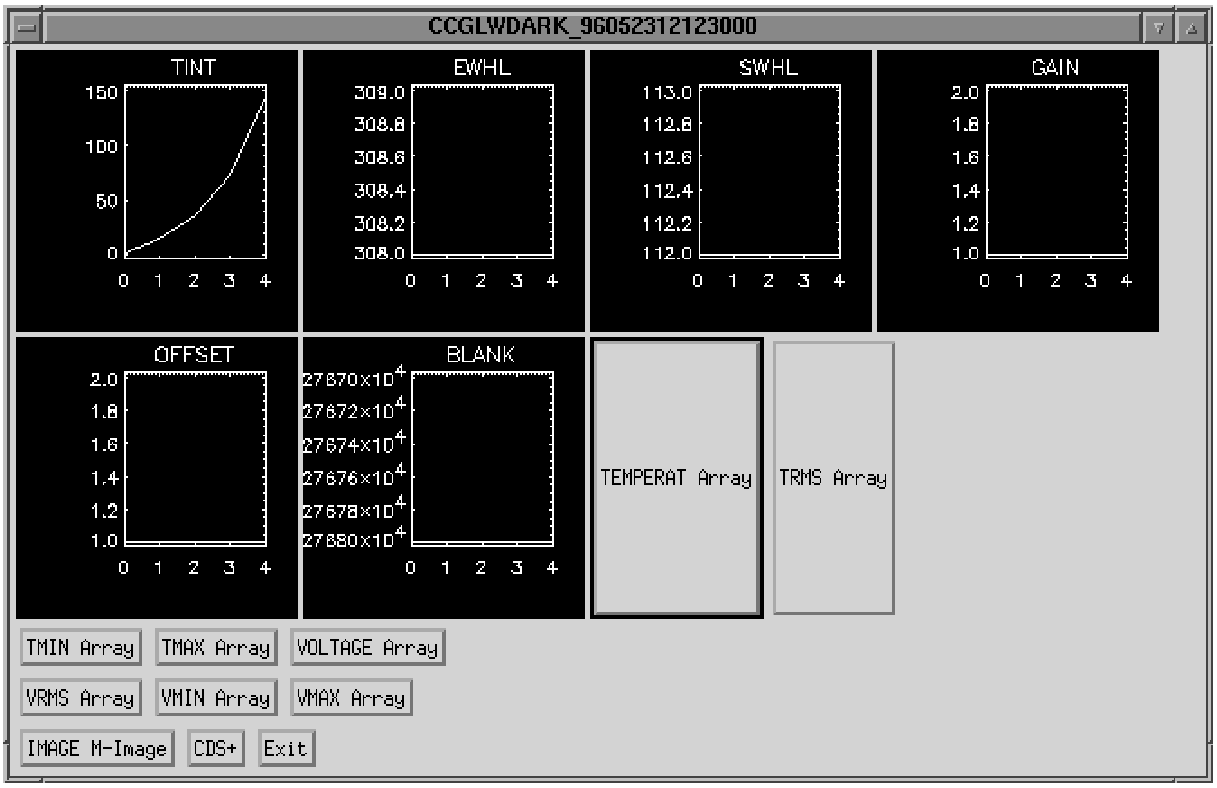

Figure 15.1: **cds_display** window.

As you can see from the above, no user-friendly strings exist in the CDS – as stated earlier calibration is handled internally by CIA so usually the innards of the CDS are not touched by a CIA user. For completeness, an example of how to convert the coded data into a readable format follows.

If we want to know the pixel unit of intensity, it has to be converted to string:

```
CIA> print, string( dark_data[0].bunit )
ADU/G/s
```

Furthermore, data such as GAIN are in telemetry coded format and also have to be converted:

```
CIA> print, convert_gain( fix( dark_data[0].gain ) )
      2
```

However, displaying the actual DARK image is easy:

```
CIA> tviso, dark_data[0].image[*,*,0]
```

For an example on how to extract PSF images from a CDS see Section 16.5.

15.3.3 cds_display

If you simply want to browse through the contents of a CDS, then **cds_display** is the tool for you. It may be called by simply typing **cds_display**, without arguments, on the CIA command line. **cds_display** will then prompt you for the name of a CDS stored on disk. Alternatively, if you have a CDS in memory it may be supplied as an argument. For example, to view the DARK CDS created in the previous section:

```
CIA> cds_display, 'CCGLWDARK_96052312123000'
```

All the parameters from the CAL-G FITS file are graphically displayed – see Figure 15.1. Clicking on the appropriate button will display plots of voltage and temperature statistics. Clicking on *IMAGE M-Image* will display a window containing the data image alongside its error image and allow you to scroll through all the images in the CDS. You can also load an additional CDS from disk by clicking on *CDS+*.

15.4 Auxiliary calibration data

Along with CDSs, there are additional calibration data contained in the CIA distribution. These are described in the following subsections.

15.4.1 Theoretical PSFs

CIA is distributed with a limited number of theoretical PSFs for use by **xphot**. A full set of 210 theoretical PSFs for all possible optical CONFIGURATIONS of CAM, along with the observed PSFs described in Section 15.4.2, can be found in the subdirectory *psf* at the official CIA ftp site. In order for CIA routines to find these PSFs they should be placed in the directory *$cia_vers/data/cds/*. The PSFs same may be shared between all CIA versions at your site by using soft links.

The theoretical PSFs are derived from the theoretical camera model – the model includes the transmission profile of a particular filter by assuming a flat spectral shape of the source. Note that no stray-light features have been considered. Though originally generated for use by **xphot** many observers have found them invaluable for advanced analyses such as jitter correction (Section 22.1) and source simulation.

The theoretical PSFs are stored as IDL save sets and with filenames of the following format:

Fixed filters e.g. `psf_lw5_6p0as.save`

psf_filter_pfov.save

CVF filters e.g. `psf_9p5mic_3p0as.save`

psf_wavelength-in-microns_pfov.save

Extracting a PSF can be easily done with the aid of **make_psf_name**. Just give this routine a PDS and it return the correct filename for the appropriate PSF library:

```
CIA> restore, make_psf_name(raster_pds), /verb
% RESTORE: Portable (XDR) SAVE/RESTORE file.
% RESTORE: Save file written by aussel@challenger, Sat Nov 29 23:24:43 1997.
% RESTORE: IDL version 5.0 (sunos, sparc).
% RESTORE: Restored variable: PSF.
```

.

If you don't have a PDS then you can supply the necessary parameters directly:

```
CIA> restore, make_psf_name({aot:'raster', wavelength:'6.75', pfov:3.0, $
CIA> fltrwhl:'lw2'}, ack=ack), /verb
```

.

Have a look at the restore variable PSF:

```
CIA> help, psf, /str
** Structure <81418>, 7 tags, length=1809884, refs=1:
   PFOV            FLOAT          3.00000
   WAVELENGTH      STRING    'LW2'
   RESOLUTION      FLOAT         0.142857
   STEP            FLOAT         0.142857
   IMA             FLOAT     Array[32, 32, 441]
   XCEN            FLOAT     Array[441]
   YCEN            FLOAT     Array[441]
```

The field .IMA holds the actual theoretical PSF image, and .XCEN and .YCEN the coordinates of the center of the PSF. The step size and resolution of the PSFs are given by .STEP and .RESOLUTION. respectively.

15.4.2 Observed PSFs

The observed PSFs were generated by replacing theoretical computed PSFs with the best fitted observed PSFs. At the date of publication of the *CIA User's Manual* observed PSFs had only been generated for the 1.5″ PFOV. The observed PSFs are stored as IDL save sets and have the following shape *psf_filter_pixel-field-of-view_simul.save*, e.g. psf_lw10_1p5as_simul.save. They have the same structure as the theoretical PSFs described in Section 15.4.1. The observed PSFs may be retrieved along with the theoretical PSFs.

15.4.3 House keeping and CAM wheels data

CAM wheels position data are distributed in CIA as simple text files. These files are used by routines such as **convert_wheel_back** to determine the wheel position from the wheel description – see Section 16.5 for an example.

To access these files directly follow the procedure below.

1. Firstly, let's have a look at the *hk_wheels_*.txt* files.

   In unix these files can be found in...

   ```
   CIA> hk_dir = '$cia_vers/tables/'
   ```

   ...and in VMS in...

   ```
   CIA> hk_dir = 'CIA_TABLES:'
   ```

   Now list the contents of *hk_dir*:

   ```
   CIA> cd, hk_dir
   ```

   ```
   CIA> dir, 'hk_wheel_*.txt'
   hk_wheel_1.txt  hk_wheel_3.txt  hk_wheel_5.txt
   hk_wheel_2.txt  hk_wheel_4.txt  hk_wheel_6.txt
   ```

Each file corresponds to a different CAM wheel.

file	**wheel number**	**description**
hk_wheel_1.txt	WHEEL 1	entrance wheel
hk_wheel_2.txt	WHEEL 2	selection wheel
hk_wheel_3.txt	WHEEL 3	LW lens wheel
hk_wheel_4.txt	WHEEL 4	LW filter wheel
hk_wheel_5.txt	WHEEL 5	SW lens wheel
hk_wheel_6.txt	WHEEL 6	SW filter WHEEL

2. Now take a look in *hk_wheel_1.txt* for entrance wheel positions:

```
CIA> $more hk_wheel_1.txt
C  WHEEL 1 Entrance wheel  R Gastaud   10-Jan-1995
CREATOR        C  R Gastaud DAPNIA CEA SACLAY
CALIBRAT       C  Interactive Analysis version 1.0
VERSION        C  Individual file version number 1.0
DATE           C  Date of file creation 10/01/95
TIME           C  Time of file creation 10:00:00
TELESCOP       C  'ISO'
INSTRUME       C  'CAM'
END
360
   0    UNKNOWN
   1    UNKNOWN
   2    UNKNOWN
   3    UNKNOWN
   4    UNKNOWN
   5    UNKNOWN
   6    UNKNOWN
   7    polarizor 2

etc...
```

So entrance wheel position 7 corresponds to 'polarizor 2'.

15.4.4 Miscellaneous auxiliary calibration data

This section describes some miscellaneous auxiliary data files that are distributed with CIA. All of this data may be found in the directory *$cia_vers/data/cds*.

CVF position data is stored in the files:

```
conv_filter_lw_table.xdr
conv_filter_sw_table.xdr
```

ORBIT data are stored in the files:

```
orbit.dat
orbit.fits
orbit.hdr
orbit_unix.xdr
orbit_vms.xdr
```

The file *orbit.fits* is the ISO ORBIT data product. It contains the orbital parameters of ISO. The data in this file is also contained in *orbit.dat* and its FITS headers are contained in *orbit.hdr*. The data is also contained in the IDL savesets *orbit_unix.xdr* and *orbit_vms.xdr*. These files are for the UNIX and VMS version of CIA respectively.

Dark model (see Section 20.2.1) auxiliary data is stored in the files:

```
cam_reset_level.xdr
activation_utk.save
mod19980320.xdr
```

Note that the dark model also uses data in the ORBIT files listed above.

xphot (see Section 14.1.4) needs the aperture correction data stored in the file:

```
xphot_aper.save
```

xradial (see Section 14.1.2) needs the radial coefficents stored in the file:

```
xcoef.inc
```

15.5 Prepared Data Structure (PDS)

The PDS was designed for convenience: to neatly hold all the data that you need for calibration (see Chapter 13 and Chapter 20) in one structure. However, as described in Section 13.1.1 there is a cost for such convenience. Because of individual differences between AOTs, four distinct PDSs currently exist: a *raster* PDS, a *CVF* PDS, a *BS* PDS and a *general* PDS. As might be expected the first three are specific to a raster, CVF and beam-switch observation respectively. The latter is a multi-purpose PDS. The routines **get_sscdraster**, **get_sscdcvf**, **get_sscdbs** and **get_sscdstruct** are used to create the appropriate PDS from an SSCD containing sliced *SPD* SCDs.

In the following sections (Sections 15.5.1, 15.5.2, 15.5.3 and 15.5.4) each of the PDSs are described. Tables listing the fields and substructures of the PDS are given and where appropriate a reference to where you may find the original data in the SSCD and SCDs (Sections 15.2.1 to 15.2.5).

Note that the PDS differs from other observation data structures, i.e. SSCDs/SCDs and SSADs/SADs, in that it is not accompanied by dedicated CIA manipulation routines. You may treat the PDS like any other IDL structure.

15.5.1 Standard fields of the PDS

The fields that are present in all flavours of PDS are listed in the table below, along with a brief description and where appropriate a reference to the SSCD or SCD field where the data originates, or otherwise, the type of the data in the field.

subfield	description	reference or type
.OBSERVER	ID of observer	Section 15.2.1.3
.TARGET	target name	Section 15.2.1.3
.TDTOSN	TDTOSN number	Section 15.2.1.5
.SSCD_NAME	input SSCD name	Section 15.2.1.1
.AOT	observation type	string – e.g. 'raster'
.SAD_NAME	output SAD name	Section 15.2.1.1
.NSCD	number of SCDs	Section 15.2.3.1
.CHANNEL	CAM channel	Section 15.2.1.8
.FLTRWHL	CAM filter wheel	Section 15.2.1.10
.WAVELENGTH	wavelength of MOSAIC	Section 15.2.1.14
.NBR_FRAME	no. of IMAGES in .CUBE	integer
.TAB_FRAME	no. of IMAGES per SCD	fltarr(.nscd)
.PFOV	pixel field of view	Section 15.2.1.13
.TINT	integration time	Section 15.2.1.12
.GAIN	gain	Section 15.2.1.11
.ADU_SEC_COEFF	normalisation factor (ADU to ADU/gain/sec)	-
.FROM(i)	IMAGEs in .INFO.SCD_NAME(i) are:	intarr(.nscd)
.TO(i)	.CUBE(*,*, .FROM(i):.TO(i))	intarr(.nscd)
.MASK	mask of .CUBE	Section 15.2.2.18
.OTF[a]	On Target Flag	bytarr(.nbr_frames)
.RMS	RMS error on EXPOSUREs	fltarr(32,32,.nscd)
.NPIX	no. of IMAGE pixels per EXPOSURE pixel	fltarr(32,32,.nscd)
.CUBE	cube of IMAGEs from SCDs	Section 15.2.2.12
.CUBE_UNIT	IMAGE units	string
.IMAGE	cube of EXPOSUREs	fltarr(32,32,.nscd)
.IMAGE_UNIT	EXPOSURE units	string
.DU	jitter offsets in spacecraft x-axis	Section 15.2.2.19
.DV	jitter offsets in spacecraft y-axis	Section 15.2.2.19
.UTK	Uniform Time Key	Section 15.2.2.19
.TABFLATCOEFF[b]	long-term transient compensation factor	intarr(.nscd)
.BOOTTIME	CAM boot-time	lonarr(.nbr_frame)
.TEMPERATURE	CAM temperature data	fltarr(10,.nbr_frame)
.DARK	DARK used for data calibration	fltarr(32,32)
.FLAT	FLAT used for data calibration	fltarr(32,32)
.HISTORY	info. on PDS creation etc...	strarr[70]

[a].OTF is not used by CIA. Information stored in .OTF is propagated into .MASK. Note that the data in .OTF is derived from the OTF and qla_flag (see Section 19.7.7). This means that .OTF=0 is bad and .OTF=1 is good.

[b].TABFLATCOEFF is filled by **rel_cal** – see Chapter 20.

15.5.2 *general* PDS

The *general* PDS is used for holding data from all observations other than a raster or CVF observation. Its structure is a basic version of the *raster* PDS and *CVF* PDS. It has the following substructures and fields:

- .CAL-G – see Section 15.5.6.

- .CCIM – see Section 15.5.8.
- .INFO – See Section 15.5.7.
- Standard PDS fields of Section 15.5.1.

15.5.3 *CVF* PDS

The *CVF* PDS is used to hold CVF data and so has specific fields for holding data that is pertinent to a CVF observation. All the fields and substructures are listed below.

- .CAL-G – see Section 15.5.6.
- Standard PDS fields of Section 15.5.1.
- *CVF* PDS specific fields: These fields are listed in the table below, along with a brief description and where appropriate a reference to the SSCD or SCD field where the data originates, or otherwise, the type of the data in the field.

subfield	description	reference type
.CVF_NAME	CVF SAD name	string
.SPEC_NAME	filter & CVF spectral data CDS name	string
.ENTWHL	entrance wheel	Section 15.2.2.9
.SELWHL	selection wheel	Section 15.2.2.10
.CVF_INCR	CVF increment step	Section 15.2.3.12
.WAVELENGTH_START	CVF start wavelength	Section 15.2.3.12
.WAVELENGTH_END	CVF end wavelength	Section 15.2.3.12
.RESPONSE[*].WAVELENG	wavelength	float
.RESPONSE[*].TRANS	transmission at WAVELENG	float
.RESPONSE[*].SENSITIV	ADU/(gain sec pixel) to mJy/pixel factor	float
.RESPONSE[*].FCVF	index wheel number	integer
.RESPONSE[*].TRANSMIS	strange transmission	fltarr(.nscd)

15.5.4 *raster* PDS

The *raster* PDS is a raster observation dedicated data structure, so as with the *CVF* PDS, it has fields specifically relating to a raster observation. These fields and additional substructures are listed here.

- .CAL-G – see Section 15.5.6.
- .CCIM – see Section 15.5.8.
- .INFO – see Section 15.5.7.
- Standard PDS fields of Section 15.5.1.
- *raster* PDS specific fields: These fields are listed in the table below, along with a brief description and where appropriate a reference to the SSCD or SCD field where the data originates, or otherwise, the type of the data in the field.

subfield	description	reference or type
.RA_RASTER	RA of MOSAIC centre	Section 15.2.3.13
.DEC_RASTER	DEC of MOSAIC centre	Section 15.2.3.14
.ANGLE_RASTER	spacecraft ROLL	Section 15.2.3.15
.RASTER_ORIENTATION	orientation reference	Section 15.2.3.6
.RASTER_ROTATION	rotation w.r.t. celestial axes	Section 15.2.3.7
.M_STEPCOL	M step size of raster	Section 15.2.3.4
.N_STEPLINE	N step size of raster	Section 15.2.3.5
.RASTERCOL	no. of pointings in M direction	Section 15.2.3.2
.RASTERLINE	no. of pointings in N direction	Section 15.2.3.3
.NX_RASTER	no. of columns in MOSAIC	integer
.NY_RASTER	no. of lines in MOSAIC	integer
.RMSRASTER	RMS error on MOSAIC	fltarr(.nx_raster,.ny_raster)
.NPIXRASTER	no. of IMAGE pixels per MOSAIC pixel	fltarr(.nx_raster,.ny_raster)
.RASTER	MOSAIC image	fltarr(.nx_raster,.ny_raster)
.RASTER_UNIT	MOSAIC units	string

15.5.5 *BS* PDS

The *BS* PDS is a beam-switch observation dedicated data structure. Its substructures and fields are listed below. You will notice that the *BS* PDS is almost identical to the *raster* PDS. This is purely for reasons of simplicity and compatibility of previously written processing routines:

- .CAL-G – see Section 15.5.6.
- .CCIM – see Section 15.5.8.
- .INFO – see Section 15.5.7.
- .ASTR – see Section 15.5.9.
- Standard PDS fields of Section 15.5.1.
- *BS* PDS specific fields: These fields are listed in the table below, along with a brief description and where appropriate a reference to the SSCD or SCD field where the data originates, or otherwise, the type of the data in the field. Again you will see similarities with the architecture of the *raster* PDS – some fields common to both have fixed values. Note that the MOSAIC in this case is the source−reference EXPOSUREs.

subfield	description	reference or type
.RA_RASTER	RA of MOSAIC centre	Section 15.2.3.13
.DEC_RASTER	DEC of MOSAIC centre	Section 15.2.3.14
.ANGLE_RASTER	spacecraft ROLL	Section 15.2.3.15
.RASTER_ORIENTATION	not used by *BS* PDS	string
.RASTER_ROTATION	rotation w.r.t. celestial axes	Section 15.2.3.7
.M_STEPCOL	not used by *BS* PDS	integer
.N_STEPLINE	not used by *BS* PDS	integer
.RASTERCOL	not used by *BS* PDS	integer
.RASTERLINE	not used by *BS* PDS	integer
.NX_RASTER	no. of cols. in MOSAIC	integer
.NY_RASTER	no. of lines in MOSAIC	integer
.RMSRASTER	RMS error on MOSAIC	fltarr(.nx_raster,.ny_raster)
.NPIXRASTER	no. of IMAGE pixels per MOSAIC pixel	fltarr(.nx_raster,.ny_raster)
.RASTER	MOSAIC image	fltarr(.nx_raster,.ny_raster)
.RASTER_UNIT	MOSAIC units	string
.BEAM	beam-switch flag	intarr(.nscd)
.SRC_IMAGE	index to source field EXPOSUREs	fltarr
.REF_IMAGE	index to ref. EXPOSUREs	fltarr
.NCYCLES	number of beam-switch cycles	integer

15.5.6 CAL-G PDS substructure

Here we describe the CAL-G data substructure which is found in all flavours of PDS. It is used to contain data taken from the CAL-G files distributed with CIA (see Section 9.3.5). It is automatically filled when the PDS is created.

subfield	type	description of contents
.CALG.DARK	fltarr(32,32)	CAL-G DARK
.CALG.DARK_NAME	string	DARK name
.CALG.FLAT[9]	fltarr(32,32)	CAL-G FLAT
.CALG.OFLAT_NAME	string	CAL-G OFLT name
.CALG.DFLAT_NAME	string	CAL-G DFLT name
.CALG.PSF	fltarr(32,32)	CAL-G PSF
.CALG.PSF_NAME	string	PSF name

15.5.7 INFO PDS substructure

This substructure preserves parameters from the individual SCDs used to create the PDS. The table lists the fields of the INFO substructure along with a reference to the SCD field.

subfield	description	reference
.INFO.SCD_NAME(i)	SCD name	Section 15.2.1.1
.INFO.RA(i)	RA	Section 15.2.2.6
.INFO.DEC(i)	DEC	Section 15.2.2.7
.INFO.ROLL(i)	ROLL	Section 15.2.2.8
.INFO.CRPIX1	CRPIX1	Section 15.2.2.4
.INFO.CRPIX2	CRPIX2	Section 15.2.2.5
.INFO.GAIN(i)	Gain	Section 15.2.1.11
.INFO.OFFSET(i)	offset	-
.INFO.TINT(i)	integration time	Section 15.2.1.12
.INFO.ENTWHL(i)	entrance wheel	Section 15.2.2.9
.INFO.SELWHL(i)	selection wheel	Section 15.2.2.10
.INFO.FLTRWHL(i)	filter wheel	Section 15.2.1.10
.INFO.PFOV(i)	pixel field of view	Section 15.2.1.13
.INFO.N_ACCU(i)	on-board accumulations	Section 15.2.2.3
.INFO.N_RASTER(i)	raster position in N direction	Section 15.2.2.1
.INFO.M_RASTER(i)	raster position in M direction	Section 15.2.2.2

i is the index to the data for an individual SCD.

15.5.8 CCIM

subfield	type	description of contents
.CCIM.IMAGE	fltarr(32,32,.nscd)	EXPOSURE in detector co-ords
.CCIM.RMS	fltarr(32,32,.nscd)	RMS error on above}
.CCIM.NPIX	fltarr(32,32,.nscd)	No. of IMAGE pixels per EXPOSURE pixel

The RMS or standard deviation is independent of the number of samples taken, e.g. reducing normal (Gaussian) distributed IMAGEs will lead to a values close to 1 in CCIM.RMS. Care has to be taken to distinguish between standard deviation (σ) and standard error; the standard error is defined by $\sigma/\sqrt{n-1}$ and therefore goes asymptotically to 0 for a large n.

15.5.9 ASTR

The .ASTR substructure can be found in the *raster* PDS, *CVF* PDS and the *BS* PDS. This substructure contains MOSAIC, e.g. .RASTER in *raster* PDS, astrometry information.

```
CIA> help, bs_pds.astr,/str
** Structure ASTR_STRUC, 8 tags, length=104:
   CD              DOUBLE    Array[2, 2]
   CDELT           DOUBLE    Array[2]
   CRPIX           FLOAT     Array[2]
   CRVAL           DOUBLE    Array[2]
   CTYPE           STRING    Array[2]
   LONGPOLE        FLOAT          180.000
   PROJP1          FLOAT         -1.00000
   PROJP2          FLOAT         -2.00000
```

The data in the .ASTR substructure follows the FITS convention – the field .CRVAL contains RA and DEC coords and the field .CD contains the CD matrix.

Chapter 16

Data structure manipulation

In Chapter 15 we looked at the high-level architecture of the CIA data structures and how to create them. As a user, you don't have to worry too much about the low-level architecture of the data structures. Manipulation such as extraction of data is done via CIA routines. In this chapter the use of these routines are described and examples are given.

16.1 CIA data structure interface routines

Below is a list of user interface routines that are used to manipulate CIA data structure. In usage, replace the stem *structure* with the name of the data structure type. The following sections provide examples and descriptions of their use. A formal and more comprehensive description of each routine, and additional routines of this type not documented here, can be found in the on-line help (see Section 2.3.2).

- *structure*_**init** initialises a structure in memory, and returns its name.
- *structure*_**extract** extracts the data from a structure and places it in a regular IDL structure.
- *structure*_**put** fills a field of this structure.
- *structure*_**get** returns a field of this structure.
- *structure*_**write** writes this structure to a file.
- *structure*_**read** reads this structure from a file.
- *structure*_**list** returns the names of all existing structures of this type.
- *structure*_**del** deletes a structure.
- *structure*_**find** returns names of structures with a field equal to a given value.
- *structure*_**info** returns information about a structure's fields.
- *structure*_**elem** returns the names of the SADs/SCDs contained in an SSAD/SSCD. Applies to SSADs/SSCDs only.

16.1.1 *structure*_**init**

This function is used to initialise a data structure in memory. That is to say, it will create an empty structure for you to fill with your own data. Due to differences in the nature of the data structures each flavour of *structure*_INIT are slightly different and so will be treated separately.

16.1.1.1 sscd_init

To initialise a new SSCD in memory we can do the following.

```
CIA> sscd1 = sscd_init( '143006010101', ack=ok )

CIA> help, sscd1, ok
SSCD1           STRING    = 'CSSC143006010101_96080512201019'
OK              INT       =        1
```

The first argument is the combined <TDT+OSN+CN+STATE> number that is used to name the returned SSCD, *sscd1* that is created by **sscd_init** (see Section 15.2.1). A keyword argument not used here is *source*, which can be used to pass an already existing SSCD to **sscd_init** so as to copy its parameters to *sscd1*.

16.1.1.2 scd_init

This function initialises an SCD in memory. An example, using all the available options and following from the example of Section 16.1.1.1, follows.

```
CIA> scd1 = scd_init( '143006010101', 10, ssid=sscd1, /ERD, deid=0, ack=ok )
CIA> help, scd1, ok
SCD1            STRING    = 'CSCD143006010101_96080512231512'
OK              INT       =        1
```

The first argument is the combined <TDT+OSN+CN+STATE> number that is used to name the returned SCD, *scd1* that is created by **scd_init** (see Section 15.2.1). The second argument specifies the number of planes in *scd1*. The keyword *ssid* specifies the name of the SSCD we have initialised in Section 16.1.1.1, and to which *scd1* now belongs. If the keyword *ERD* is set *scd1* is an SCD of ERD flavour (see Section 15.2.2 otherwise it defaults to an SCD of SPD flavour). The keyword *deid* is either 0 : LW detector, or 1 : *SW* detector, and defaults to LW detector.

16.1.1.3 ssad_init

To initialise a new SSAD in memory we can do the following.

```
CIA> ssad1 = ssad_init( '143006010101', ack=ok )
CIA> help , ssad1
SSAD1           STRING    = 'CSSA143006010101_96080512201019'
OK              INT       =        1
```

The first argument is the combined <TDT+OSN+CN+STATE> number that is used to name the returned SSAD, *ssad1* that is created by **ssad_init** (see Section 15.2.1). A keyword argument not used here is *old_ssad*, which can be used to pass an already existing SSAD to **ssad_init** so as to copy its parameters to *ssad1*.

16.1.1.4 sad_init

This function initialises an SAD in memory. An example, following from the Section 16.1.1.3 is:

```
CIA> sad1 = sad_init( '143006010101', ssid=ssad1, ack=ok )

CIA> help, sad1, ok
SAD1            STRING    = 'CSAD143006010101_96080513124826'
OK              INT       =        1
```

The first argument is the combined <TDT+OSN+CN+STATE> number that is used to name the returned SSAD, *ssad1*, that is created by **ssad_init** (see Section 15.2.1). The keyword *ssid* specifies the name of the SSAD we have initialised in Section 16.1.1.3, and to which sad1 now belongs. A keyword argument not used here is *source*, which can be used to pass an already existing SAD to **sad_init** so as to copy its parameters to *sad1*.

16.1.1.5 cds_init

This function initialises a CDS in memory. An example of its usage follows.

```
CIA> cds1 = cds_init( 'ccgswdark', ack=ack )

CIA> help, cds1, ack
 <EXPRESSION>    STRING    = 'CCDS123456789012_95012318401579'
 ACK             INT       =        1
```

The first argument is the type of the returned CDS, *cds1*. The possible types of CDSs are given in Table 15.3.2.

16.1.2 *structure*_extract

This function simply extracts data from a CIA structure and places it in a regular IDL structure. A simple example for an SCD is:

```
CIA> regular_struct = scd_extract( 'CSCD143006010110_97092611592803' )
```

Now we can use IDL's HELP to look at the guts of *regular_struct*.

```
CIA> help, regular_struct, /str

etc...
```

16.1.3 *structure*_put

Procedure to place a value in a field of a structure. This can be a useful routine when a data structure field is undefined or incorrectly defined. The fields for the relevant structure are given in Chapter 15. Note that you should be careful when using *structure*_**put**. Changing a field value unnecessarily may disrupt your data. On no account change the field .NAME – it will confuse CIA memory management. Also read Section 16.4 before attempting to manipulate the MASK with **scd_put**.

As an example, the string 'EARTH' is placed in the field .NAME of the specified SCD.

```
CIA> scd_put, 'target', 'EARTH', 'CSCD143006010105_96080110071423', ack=ack
```

16.1.4 *structure*_get

Function to return a value of a specified field of a structure. Following from the example in Section 16.1.3, the value of the field NAME is extracted.

```
CIA> help, scd_get( 'name', 'CSCD143006010105_96080110071423' ), ack
 <Expression>    STRING    = 'EARTH'
 ACK             INT       =        1
```

See Section 2.4 for restrictions of use.

16.1.5 *structure*_write

Procedure to write a specified structure to a file. The filename will be the name of the structure with *.cub* appended. When using **sscd_write** and **ssad_write** the entire set of SCDs or SADs are written to disk. In the example below a single SCD is written to disk in the directory *product_dir* (see Section 17.1.1).

```
CIA> scd_write, 'CSCD143006010105_96080110071423', dir=scd_dir, ack=ack

CIA> help, ack
 ACK             INT       =        1
```

See Section 2.4 for restrictions of use.

16.1.6 *structure*_read

Function to read a named structure from disk. A pointer to the structure is returned. Our example below follows from that of Section 16.1.5.

```
CIA> scd = scd_read( 'CSCD143006010105_96080110071423', dir=scd_dir, ack=ack )
```

16.1.7 *structure*_list

Function to return the a list of structures of type *structure* in memory. Following from Section 16.1.6 our example lists the SCDs in memory.

```
CIA> print, scd_list( num=no_scds, ack=ack )
CSCD143006010105_96080110071423

etc...

CIA> help, no_scds, ack
NO_SCDS         INT       =       42
ACK             INT       =        1
```

The keyword *num* returns the number of scds found.

16.1.8 *structure*_del

Procedure to delete a specified structure of type *structure* from memory. A simple example, which follows from Section 16.1.7 is:

```
CIA> scd_del, 'CSCD143006010105_96080110071423'
```

Using a little of IDL's power we can delete all the SSCDs in memory with the following example:

```
CIA> sscds = sscd_list()

CIA> help, sscds
SSCDS           STRING    = Array(4)

CIA> for i=0,3 do sscd_del, sscds(i)

CIA> print, sscd_list()
UNDEFINED
```

Note that when deleting SSCDs or SSADs the whole set of SCDs or SADs is erased from memory.

16.1.9 *structure*_info

Procedure to display a list of fields of a structure of type *structure* and their values. The routine varies for each structure, but in general the example below is valid for all flavours.

Assuming an SCD called *CSCD143006010001_96082815175532* in memory:

```
CIA> scd_info, 'CSCD143006010001_96082815175532'

*************************************************************
SCD = CSCD143006010001_96082815175532
Detector channel         : LW
Entrance wheel           : HOLE
Selection wheel          : FABRY MIRROR LW
Lens                     :       6.00000
Lens wheel position      :      360
Filter                   : LW2
Filter wheel position    :       95
Integration time (second):       2.10007
Integration time (camtu) :       15
Gain                     :        1
Offset                   :      512
Mode                     : IDLE

Model Min                :      -25.0000
Model Max                :       15.0000
Model Mean               :      -3.26709
Model Median             :      0.000000
```

```
Model standard deviation :       4.83574
*************************************************************
```

Additional inputs allow for reading of saved structures directly from disk and displaying of selected fields only. It is recommended that you refer to the on-line help for detailed information on each form of the routine *structure***_info**.

16.1.10 *structure***_find**

This function returns a list of structures of type *structure* which have a field containing a given value. Calling the routine is identical for all its flavours, the only difference being the fields of the *structure*. Note that arguments are restricted to the numeric or string type, i.e. you can't mix a search for a string value of one field and a numeric value of another.

```
CIA> print, scd_find( ['mode','fltrwhl'], ['OBS','Lw6'] )
Searching for MODE=OBS
Searching for FLTRWHL=LW6
CSCD143006010202_96082815180230 CSCD143006010203_96082815180271
 CSCD143006010204_96082815180303 CSCD143006010205_96082815180339
 CSCD143006010206_96082815180373 CSCD143006010207_96082815180407
 CSCD143006010208_96082815180442 CSCD143006010209_96082815180478
 CSCD143006010210_96082815180510 CSCD143006010211_96082815180542
 CSCD143006010212_96082815180578 CSCD143006010213_96082815180610
 CSCD143006010214_96082815180644 CSCD143006010215_96082815180676
 CSCD143006010216_96082815180710 CSCD143006010217_96082815180742

CIA> print, scd_find( 'gain', 1 )
Searching for GAIN=       1
CSCD143006010001_96082815175532
```

The keyword *list* allows you to supply a list of structures (restricted to the same type). In the following example **scd_find** supplies a list to a second call to **scd_find**.

```
CIA> print, scd_find( 'gain', 1, $
CIA> list = scd_find( ['mode','fltrwhl'], ['OBS','lw6'] ) )
Searching for MODE=OBS
Searching for FLTRWHL=LW6
Searching for GAIN= 1
% STRUCT_FIND: Failed to find: GAIN =        1.
UNDEFINED
```

16.1.11 *structure***_elem**

This is a function to return the list of SCDs in an SSCD or a list of SADs in an SSAD – it doesn't exist for the SCD, SAD or CDS. Calling *structure***_elem** is quite simple:

```
CIA> print, sscd_elem( 'CSSC143006010001_96082814202966', num=noscds )

CSCD143006010001_96082815175532 CSCD143006010002_96082815175578
```

```
etc...

CIA> print, noscds
          41
```

16.2 An example of SAD manipulation

In Section 10.2 we used **sad_display** to create SADs from the AAR data products. At that point you were probably only interested in a quick look at your data – now we will delve a little deeper into the data and take a look at the SADs themselves.

Most likely you have exited the session where you first used **sad_display** to look at your AAR data products. To regenerate the SADs in a CIA session, use **sad_display** as described in Section 10.2, or if the SADs are on disk use **ssad_read**.

Upon exiting **sad_display** all the SADs created will remain in memory. Try the following command:

```
CIA> print, ssad_list()
CSSA000014300601_96091713360439 CSSA000014300601_96091713360451
```

ssad_list lists all the SSADs in memory. Two such SSADs exist in our example, one catalogues the set of SADs that hold the EXPOSUREs from the CCIM and CMAP data products and the other the SADs that hold the set of MOSAICs from the CMOS data products. The former SADs are known as *origin* SADs and the latter as *future* SADs. To see the full set of *future* SADs we can use the CIA routine **ssad_elem**. Given the name of an SSAD it will list all of its SADs:

```
CIA> print, ssad_elem( 'CSSA000014300601_96091713360451' )

CSAD000014300601_96091713370548 CSAD000014300601_96091713370625
CSAD000014300601_96091713370691
```

In this case there are two *future* SADs, with a MOSAIC in each, and a third SAD, containing the contents of the AAR data product CSSP, which you can ignore. Listing the *origin* SADs catalogued in our second SSAD we see that there are many:

```
CIA> print, ssad_elem( 'CSSA000014300601_96091713360439' )

CSAD000014300601_96091713360476 CSAD000014300601_96091713360767
CSAD000014300601_96091713360860 CSAD000014300601_96091713360985
CSAD000014300601_96091713362332 CSAD000014300601_96091713362446
CSAD000014300601_96091713362951 CSAD000014300601_96091713363048
CSAD000014300601_96091713363160 CSAD000014300601_96091713363716
CSAD000014300601_96091713363810 CSAD000014300601_96091713363912
CSAD000014300601_96091713364023 CSAD000014300601_96091713364107

 etc...
```

Each of these *origin* SADs contains one EXPOSURE from the CCIM file and one from the CMAP file, and of course we can expect to have many such EXPOSUREs in a typical AOT. This explains why there are so many more *origin* SADsthan *future* SADs.

So we have seen that there are two MOSAICs, but many EXPOSUREs. Our example is taken from a raster observation where two CONFIGURATIONs of CAM were employed: one was performed with the LW3 filter and the other with the LW6 filter. Each of these CONFIGURATIONs yields data from which a MOSAIC may be created.

A SAD contains more than just image data, it also contains CAM parameters during the observation. As an example, we can see which MOSAIC corresponds to which filter by extracting information from the corresponding *future* SAD. The *future* SAD's field WAVELENGTH holds the wavelength of the filter-wheel used in the CAM CONFIGURATION from which AA has created the MOSAIC. (Likewise the same field in the *origin* SAD will hold the wavelength of the filter-wheel used in a CAM STATE.) The value of a field in a CIA data structure can be extracted with the CIA routine **sad_get**.

```
CIA> print, sad_get( 'wavelength', 'CSAD000014300601_96091713370625' )
      7.75000

CIA> print, sad_get( 'wavelength', 'CSAD000014300601_96091713370548' )
      14.5000
```

We can also extract the image from the SAD and place it in a regular IDL array. The field CMAP.DATA in the *future* SAD contains the MOSAIC from the CMOS data product. (The same field in the *origin* SAD contains an EXPOSURE from the CMAP data product.)

```
CIA> lw3_image = sad_get( 'cmap.data', 'CSAD000014300601_96091713370548' )

CIA> lw6_image = sad_get( 'cmap.data', 'CSAD000014300601_96091713370625' )

CIA> tviso, lw6_image / lw3_image
```

In the above example the MOSAICs are extracted from the SADs and their ratio is displayed. The CIA routine **tviso** is a useful variation on IDL's TVSCL, the difference being that it displays the image in a convenient size.

Though you have created SSADs in memory, they have not been saved to disk. You may wish to save them rather than having to go through the slow process of regenerating them again. **ssad_write** is dedicated to saving an SSAD and all its catalogued SADs.

```
CIA> ssad_write, 'cssa000014300601_96092310115335', dir='./sads'
```

where, *dir* is the destination directory. You may repeat the command for the other SSAD.

16.3 Saving and restoring PDSs

Since a PDS is a regular IDL data structure it can be saved with the IDL commands **SAVE** and **RESTORE**. It is recommended that you use the keyword */xdr* so as to avoid compatability problems when sharing data across platforms.

If you have completed the calibration of your data and you want to save the results, then you can remove some unnecessary data from your PDS. **prune_pds** removes the fields .MASK and .CUBE – these fields contains the vast bulk of the data in a PDS, so removing them greatly decreases the PDS size and saves disk and memory space.

```
CIA> raster_pds = prune_pds( raster_pds )

CIA> save, file='raster_pds.dat', raster_pds, /verb
% SAVE: Portable (XDR) SAVE/RESTORE file.
% SAVE: Saved variable: raster_pds.
```

raster_pds is a *raster* PDS which is originally more than 2 MB in size. **prune_pds** reduces it to about 0.5 MB.

```
CIA> $ls -la raster_pds.dat
-rw-r--r--   1 mdelaney ssamr     500568 Oct 14 13:17 raster_pds.dat
```

16.4 Manipulating the MASK

This section describes CIA MASK manipulation routines. For more details on the MASK in CIA data structures and a list of the possible MASK values see Section 15.2.2.18. Also take a look at Section 2.3.4 for how to configure the MASK handling.

- **ia_put_mask** routine to modify the MASK in a CIA data structure or IDL array. To work with ordinary IDL arrays you can use **put_mask**.
- **ia_get_mask** routine to extract the MASK from a CIA data structure.

16.4.1 Extracting the MASK from CIA data structures

Follow the procedure below to extract and examine a MASK from a CIA data structure.

1. Suppose we have a CIA structure. The first step is to extract the entire MASK from the structure.

   - If our structure was an SCD this can be done as:

     ```
     CIA> mask = scd_get( 'mask', 'CSCD143006010110_97092611592803' )
     ```

   - If it is a PDS:

     ```
     CIA> mask = pds.mask
     ```

2. Now, to look at the dead pixels use **ia_get_mask** to extract and **tviso** to display.

   ```
   CIA> dead_mask = ia_get_mask( 'dead', mask )

   CIA> tviso, dead_mask[*,*,0]
   ```

   If you look at the values in *dead_mask* you will see that they are either 0 or 1. However the values in *original_mask* can be 0,1,2,4,6,8,16,32,64,128, each value having a separate meaning (see the table in Section 15.2.2.18). *ia_get_mask* translates these values to a simple 0 or 1. In our example above, we asked for the pixels masked as dead to be translated, so all the pixels with a value of 2 in *original_mask* became 1 in *dead_mask*.

Alternatively, you can extract the dead pixels from an SCD directly:

```
CIA> dead_mask = scd_get( 'mask.dead', 'CSCD143006010110_97092611592803' )
```

16.4.2 Modifying the MASK

Following on from Section 16.4.1 we will now create and modify our own MASK.

1. Firstly, create an empty array that will hold our MASK, let's call this *our_mask*. We will make it the same size as *orginal_mask*:

   ```
   CIA> our_mask = bytarr(32,32,19)
   ```

2. Suppose that the first column of all our CAM IMAGEs is blind, we can set the first column pixels to the appropriate value with **put_mask**.

   ```
   CIA> our_mask[0,*,*] = put_mask( 'blind', our_mask[0,*,*] )
   ```

3. Now look at the first element in *our_mask*. This should be a blind pixel.

   ```
   CIA> print, our_mask[0]
     16
   ```

   Since it is a blind pixel it has a value of 16.

4. Our MASK can be added to a CIA structure with **scd_put**. The keyword *put* decides how out MASK is inserted into the structure. For example, to combine the MASK bits in our MASK with the CIA structure's existing MASK bits, i.e an OR operation:

   ```
   CIA> scd_put, 'mask', mask, 'CSCD143006010110_97092611592803', put='put'
   ```

   Actually this is the default behaviour of **scd_put**. To replace the existing MASK with the input MASK then *put* should be set to 'set'. Setting *put* to 'clear' will cause the existing MASK bits to be set to zero (i.e. cleared) where the input MASK bits are one.

Alternatively, we can create a MASK and put it directly into a structure, merging it with the structure's MASK. The structure could be a PDS or ordinary structure extracted from a CIA structure, e.g.

```
CIA> struct = scd_extract( 'CSCD143006010110_97092611592803' )
```

Now follow the following procedure.

1. Create a variable to hold our MASK.

   ```
   CIA> our_mask = bytarr(32,32,19)
   ```

2. set the pixels we wanted masked to 1. Again we use the example that the first column of all our CAM IMAGEs are blind.

   ```
   CIA> our_mask[0,*,*] = 1
   ```

3. Call **ia_put_mask** to put *our_mask* into *struct*, converting the pixels of value of 1 to the appropriate value for blind pixels.

   ```
   CIA> ia_put_mask, 'blind', struct, our_mask
   ```

16.5 CDS data extraction

This section describes ways in which you can examine the contents of a CDS (remember that a CDS is a CIA data structure which holds CAM calibration data – see Section 15.3).

Let's assume you want to find out (without using **find_best_psf** - see Section 20.11.2) what PSFs we have for an observation configured with a 3" PFOV and the LW2 filter.

1. Firstly, you have to read in the LW PSF CDS:

```
CIA> psf_cds = cds_read( 'lwpsf' )
```

2. Secondly, you have to know the LW2 filter wheel position and the LW lens wheel position for the 3P̈FOV. You can find this information either in the ISOCAM documentation (e.g. *ISOCAM User's Manual*), in the CIA files *hk_wheels_n.txt* (see Section 15.4.3) or with the function **convert_wheel_back**:

```
CIA> lw2_position = convert_wheel_back( 'LW FILTER', 'LW2' )

CIA> pfov_position = convert_wheel_back( 'PFOV LW', 3.0 )
```

3. As described in Section 15.3.2, all calibration data reside in the .DATA substructure. To extract .DATA from our PSF CDS:

```
CIA> data = cds_get( 'data', psf_cds )
```

4. Inside this substructure you will find the cube DATA(*).IMAGE, and the arrays DATA(*).PFOV and DATA(*).FCVF. Respectively, these hold a variety of PSFs with their corresponding lens wheel positions and PFOVs. Using the IDL command **WHERE** we can pick out the PSFs that we require.

```
CIA> nn = where( ( DATA.FCVF eq lw2_position ) AND $
CIA> ( DATA.PFOV eq pfov_position ) )

CIA> psf = data[nn].image[*,*,0]
```

Now the variable *psf* will hold a cube of PSF images corresponding to the CAM configuration of filter wheel LW2 and a PFOV 3:' You could preview these with **x3d**:

```
CIA> x3d, psf
```

16.6 Manipulating CIA data structure history

The processing applied within CIA to CIA data structures, also known as the history, is recorded in the CIA data structure field PROCESS. The following sections will show you how to extract and replace this field.

16.6.1 Extracting the history

The history information stored in a CIA structure can be accessed using the *structure***_get** routine. As an example, suppose we have an SCD in memory. Let's assign its name to the variable *scd* just for convenience.

```
CIA> scd = 'CSCDMKN297L6P100_97040515444142'
```

Now take a look at the history of *scd*:

```
CIA> print, scd_get( 'history',scd )
 date=05-Apr-1997 15:44:37 node=ISOW40 user=LMETCALF procedure=x_slicer V II.1
algorithm=user

 CIER.FIT;1 ISOW40$DKC100:[IA.LEO.WORK.MKN297] IIPH.FIT;1
ISOW40$DKC100:[IA.LEO.WORK.MKN297] NO COMPACT STATUS HISTORY NO ORBIT

 FILE starting record = 1850 ending record = 1963 cia otf
END date=05-Apr-1997 15:44:38 node=ISOW40  user=LMETCALF

 procedure=erd2spd V 3.1  algorithm=default CSCDMKN297L6P100_97040515443475 END
```

The output we get in the above example is in a user-friendly format. Within CIA structures the history is stored in a coded fashion in the field PROCESS. In the example above the PROCESS was automatically converted into a readable history. In the following example we take a look at the actual contents of PROCESS:

```
CIA> process = scd_get( 'process', scd )
```

Have a look at *process*.

```
CIA> print, process
{   260639077 ISOW40  LMETCALF x_slicer V II.1  user           0          31 }
{   260639079 ISOW40  LMETCALF erd2spd V 3.1
 default          0          53 }
```

As you can see from the above it is not in a very user-friendly format. To make it into a readable history we can use **process2history**:

```
CIA> history = process2history( process )
CIA> print, history

etc...
```

In fact it is this more readable history which can be found in the field .HISTORY in the PDS. For example:

```
CIA> print, raster_pds.history
date=28-Sep-1998 14:50:33 node=bermuda user=mdelaney
 procedure=darklibrary V 1.1 algorithm=Find best ccglwdark_97031713382678 END
```

```
date=28-Sep-1998 14:48:27 node=bermuda user=mdelaney
procedure=spdtoscd V 2.0 algorithm=default
(7) cisp02600506.fits <undefined> <undefined> <undefined> <undefined>
 <undefined> <undefined> (2) 2 (2) 47 END
date=28-Sep-1998 14:50:34 node=bermuda user=mdelaney
procedure=flat_library V 1.1 algorithm=Find best ccglwoflt_98050815090326 END
date=28-Sep-1998 14:50:35 node=bermuda user=mdelaney
procedure=flat_library V 1.1 algorithm=Find best ccglwdflt_98031519384439 END
date=28-Sep-1998 14:50:28 node=bermuda user=mdelaney
procedure=get_sscdraster V 4.0 algorithm=default
CSSC026005060001_98092814502500 model= 1 END
```

16.6.2 Replacing the history

You can use the routines **history2process** and *structure_***put** to convert a history into a format suitable for insertion into the field PROCESS in CIA data structure.

Following from the example in the previous section, we can recreate the contents of the variable *process* and then insert into our SCD.

```
CIA> process = history2process( history )

CIA> scd_put, 'process', process, scd
```

Chapter 17

Importing ISO data products to CIA

CIA provides dedicated FITS handling routines to create the data structures from the data products distributed on the ISO CD-ROMs or retrieved from the IDA. This chapter provides an overview of their use.

17.1 Importing FITS to CIA data structures

This section describes the CIA routines for importing ISO data products to CIA data structures. Section 17.2 describes routines for importing ISO data products, or any extended FITS files for that matter, to regular IDL structures.

17.1.1 Assigning working directories

To avoid tedious typing, it is useful to assign the path names of the directories you will use in the calls to the CIA routines to IDL string variables. The directories used in the examples in this chapter are assigned as follows:

In VMS...

```
product_dir = 'DKA200:[MDELANEY.14300601]'
scd_dir = 'DKA200:[MDELANEY.14300601.scds]'
sad_dir = 'DKA200:[MDELANEY.14300601.sads]'
tdtosn = '14300601'
```

...and in UNIX...

```
product_dir = '/home/mdelaney/14300601'
scd_dir = '/home/mdelaney/14300601/scds'
sad_dir = '/home/mdelaney/14300601/sads'
tdtosn = '14300601'
```

Also, the variable *tdtosn* is assigned the combined TDT and OSN number.

17.1.2 SADs from AAR: aa2sad

To convert AAR data products to SADs in memory or on disk we use the procedure **aa2sad**. Note that this routine is a commmand line version of the high-level widget program **sad_display** – see Section 10.2. Using the assigned variables of Section 17.1.1 an example call is:

```
CIA> aa2sad, tdtosn, ssad_origin, ssad_future, scd_dir=scd_dir, $
CIA> sad_dir=sad_dir, arc_dat=product_dir, ack=ack, /noscd
```

Two SSADs are created, one being the *origin* and the other the *future* SSAD (see Section 15.2.4). The keywork *ack* returns a logical value, 1 for a successful call and 0 for a failure. Other useful optional keywords to **aa2sad** are *nowrite* and *noscd*. Setting *nowrite* will inhibit writing of SADs to disk. Setting *noscd*, as in the example, inhibits the default creation of SCDs along with the SADs.

17.1.3 SCDs from SPD: spdtoscd

To convert SPD data products to *SPD* SCDs in memory or on disk we use the procedure **spdtoscd**. Using the assigned variables of Section 17.1.1, an example call is:

```
CIA> spdtoscd, 'cisp14300601.fits', sscd, dir=product_dir, $
CIA> scd_dat=scd_dir, ack=ack
```

The output parameter *sscd* will contain the name of the created SSCD. Only one SSCD exists for a single CISP file. The SSCD and its SCDs will be written to *scd_dir* (permission to write on this directory is required). The keyword *ack* returns a logical value, 1 for a successful call and 0 for a failure. For more advanced users, the UTK start time and end time of required SCD records can be passed as keywords to **spdtoscd** (see the on-line help or **cia_help**).

In addition the keyword *bs* may be of interest to observers with beam-switch data. Beam-switch observations have an intermediate step which occurs while slewing and data accrued in this step may appear in an SCD. Normally this SCD is considered to contain invalid data. Setting */bs* will make **spdtoscd** merge this SCD with the previous SCD and users can later manually unmask this 'invalid' data for inclusion in the finally MOSAIC image.

17.1.4 SCDs from ERD: erdtoscd

With the exception that **erdtoscd** creates *ERD* SCDs from ERD data products, everthing in the description of **spdtoscd** in Section 17.1.3 also applies here. An example call might be:

```
CIA> erdtoscd, 'cier14300601.fits', sscd, dir=product_dir, $
CIA> scd_dat=scd_dir, ack=ack
```

17.1.5 CDSs from CAL-G files: calg2cds

To convert CDS data products from files on the CD-ROM to CDSs in memory or on disk we use the function **calg2cds**[1]. Using the assigned variables of Section 17.1.1 an example call is:

[1]Remember that since the CDSs in CIA are more recent then the distributed CAL-G files (Section 9.4.3) it is unlikely that you will frequently use this routine.

```
CIA> cds = calg2cds( 'ccglwdark.fits', dir=product_dir, ack=ok )

CIA> help, cds, ok
CDS             STRING    = 'CCGLWDARK_96080515324098'
OK              INT       =        1
```

calg2cds returns the name of the formed CDS in memory. Nothing is written to disk by this function. Use the CIA procedure **cds_write** to save the CDS data structure to disk (see Section 16.1.5).

17.2 Importing FITS to regular IDL data structures

This section describes CIA routines that can be used to read any extended FITS data products. As such they can read the ISO data products into a regular IDL structure. The IDL ASTROLIB also contains several FITS reading routines which are of interest: MRDFITS and READFITS. CIA comes with a modified version of MRDFITS – the keyword option */savemem* has been added to save memory when reading large files.

17.2.1 Reading an ISO data product

The CIA routine **ia_extended_fits_read** reads an extended FITS file into a conventional IDL structure (not a PDS or SCD). For example, to read a CMOS data product (remember that ISO data products are delivered in extended FITS file) we do the following:

```
CIA> ia_extended_fits_read, 'cmos14300601.fits', header, array

CIA> help, hdr
HDR             STRING    = Array[209]

CIA> help, array
ARRAY           STRUCT    = -> <Anonymous> Array[6]
```

The output arguments *header* and *array* contain the header of the FITS file and the data from the FITS extensions. We can look at lines in the header by indexing *header*:

```
CIA> print, header[0]
SIMPLE  =                    T / file does conform to FITS standard
```

To look at the binary table:

```
CIA> help, array, /str
** Structure <1242100>, 38 tags, length=25896, refs=1:
   CHANNEL         STRING    'LW  '
   INDEX           LONG                 1
   TYPE            STRING    'FLUX    '
   BITPIX          LONG                32

etc...
```

Each ISOCAM image is stored in a vector (*ISOCAM Handbook*, Section 2.3.4). Before we can reform and display them they need to be rescaled (*ISOCAM Handbook*, Chapter 7):

```
CIA> bscale = array.bscale

CIA> bzero = array.bzero

CIA> data = float(array.array)

CIA> for n = 0, 5 do data[ *, n ] = data[ *, n ]* bscale[n] + bzero[n]
```

The scaled data must now be reformed to the appropriate image size. To find the actual image size in rows and columns we look at the fields .NAXIS1 and .NAXIS2 respectively.

```
CIA> naxis1 = array.naxis1

CIA> naxis2 = array.naxis2

CIA> print, naxis1, naxis2
          80          80          80          79          79          79
          80          80          80          80          80          80
```

To reform the images:

```
CIA> nb = naxis1 * naxis2 - 1

CIA> for n=0, 5, 3 do tviso, $
CIA>   reform( data[ 0: nb[n], n ], naxis1[n], naxis2[n] )
```

When image sizes differ, only the first *nb* entries in each row of *data* are valid. Each sky image stored in the CMOS is accompanied by a second equally sized image indicating the noise level per pixel and a third image indicating the statistical weight assigned to each pixel. So in the example above we only display every third image, i.e. the sky images.

17.2.2 Extracting a key from an ISO data product

With the procedure **by_extended_fits_key** we can read selected keys directly from a FITS file – this is more efficent then unnecessarily reading the entire file. The following extracts the data from a CISP file and displays an image.[2]

```
CIA> by_extended_fits_key, 'cisp14300601.fits', data, 'cispdata'

CIA> help, data
DATA            INT       = Array(1024, 813)

CIA> tviso, reform( data[ *, 0], 32, 32 )
```

[2]Refer to the *ISO Data Product Document* to find where the data is stored in the binary table, i.e. *cispdata*.

Chapter 18

Export of CIA data structures

Many CIA users probably also use non-IDL based analysis packages, such as IRAF[1]. The most convenient way to share data with such external packages is by using the FITS format. Due to the inability of many external packages to handle extended FITS files we need to use both non-extended and extended FITS. It is impossible to export all data from a CIA structure without using FITS extensions, so for archiving purposes FITS extensions are unavoidable. With these problems in mind two kinds of export have been developed: (i) export for external analysis using non-extended FITS and (ii) export for archiving using FITS extensions. A small exception to this rule has been made with the routine **conv_cvf2isap**.

18.1 Export to the spectral analysis package ISAP

conv_cvf2isap allows export of ISOCAM CVF data to the spectral analysis package ISAP:

```
CIA> conv_cvf2isap, cvf_pds, 'cvf_isap.fits'
```

As you may guess bf conv_cvf2isap accepts a *CVF* PDS as input. As output it writes a FITS file with extensions that can be understood by ISAP. The keyword option */pix* accepts a two element integer array specifing an ISOCAM pixel and a spectrum is extracted a this pixel position. bf conv_cvf2isap also works with ISOCAM CVF AAR products – in this case it accepts the name of a CMAP file as input.

18.2 Export to external packages

This section describes CIA routines for export of data, to non-extended FITS files, for analysis by external packages.

raster2fits writes the *raster* PDS fields .RASTER, .RMSRASTER and .NPIXRASTER to a FITS primary array.

```
CIA> raster2fits, raster_pds, name='raster.fits'
```

The keyword option */iraf* will make the FITS file compatible with the IRAF preferred CD matrix format of *CDx_y*, instead of *CD00x00y*, and only .RASTER will be written to the

[1]It appears that MIDAS has a problem reading CIA generated FITS files – see Section 2.4.

primary array. Since the *BS* PDS is compatible with the *raster* PDS, this routine also works for beam-switch (CAM03) data. To save a beam-switch MOSAIC to a FITS file:

```
CIA> raster2fits, bs_pds, name='bs.fits'
```

These FITS files can be viewed with packages like SAOIMAGE:

```
CIA> $saoimage raster.fits
```

imagette2fits writes each frame or EXPOSURE in the PDS field .IMAGE (along with the corresponding frame of .RMS and .NPIX) to the primary array of an individual FITS file. This makes it useful for exporting CVF observation (CAM04) and staring observation (CAM01) data contained in an *CVF* PDS or *general* PDS. In the example below 20 FITS files are created with a prefix taken from the keyword *name*.

```
CIA> help, cvf_pds.image
<Expression>    FLOAT     = Array[32, 32, 20]

CIA> imagette2fits, cvf_pds, name='cvf.fits'

CIA> $ls cvf*.fits
cvf1.fits   cvf13.fits  cvf17.fits  cvf20.fits  cvf6.fits
cvf10.fits  cvf14.fits  cvf18.fits  cvf3.fits   cvf7.fits
cvf11.fits  cvf15.fits  cvf19.fits  cvf4.fits   cvf8.fits
cvf12.fits  cvf16.fits  cvf2.fits   cvf5.fits   cvf9.fits
```

sad_write_fits Expert users of CIA may use SADs to hold their calibrated data. To write the SAD CMAP to a FITS file:

```
CIA> sad_write_fits, 'CSAD000014300601_98072013530646', /cmap
```

Note that the FITS files created by the routines described above can be easily loaded into IDL with the ASTROLIB routine READFITS.

18.3 Export for archiving

For archiving purposes we want to save the entire contents of a CIA data structure to an extended FITS file.

struct2fits writes an entire CIA data structure to a FITS file. The data is stored in extensions.

```
CIA> struct2fits, raster_pds, name='raster_archive.fits'
```

fits2struct recovers the output of **struct2fits**. It initializes the appropriate PDS and then fills it as best it can. It is very useful for upgrading the architecture of an obsolete PDS.

```
CIA> fits2struct, 'raster_archive.fits', hdr, raster_pds_recovered

CIA> help, raster_pds_recovered, /str
** Structure <123a0f0>, 54 tags, length=1399800, refs=1:
   RASTERCOL       INT              4
   RASTERLINE      INT              2
   M_STEPCOL       FLOAT           84.0000
   N_STEPLINE      FLOAT           84.0000

etc...
```

Optionally, setting the keyword **/orig** will cause **fits2struct** to recreate the original architecture of the PDS. This means that an archived PDS with obsolete architecture will be recovered into a later version of CIA without being upgraded.

Part IV

Advanced Use of CIA

Introduction

This part of the *CIA User's Manual* is intended as a guide to advanced use of CIA. It is hoped that it will help you to get the most out of your CAM data.

- Chapter 19 is an advanced guide to data slicing. It follows from Chapter 12.
- Chapter 20 takes a closer look at the core calibration routines and at AOT dedicated processing. It also gives guidelines on how to determine the best calibration for your data. It follows from Chapter 13.
- Chapter 21 serves as an introduction to the **SLICE** package and its long term transient correction (or LTT) and variable flat-field correction (or VFF) algorithms.
- Chapter 22 describes second order corrections for CAM data.
- Chapter 23 is an advanced guide to **x_cia** – it follows from Section 13.3.

Chapter 19

Advanced slicing

This chapter describes and gives examples of advanced slicing.

19.1 Advanced slicing options

This section describes advanced slicing keyword options that are available to the automatic slicers **spdtoscd** and **erdtoscd**.

corrected Since OLP version 6.1 improved of values of RA, DEC and ROLL, namely CRA, CDEC and CROLL, are provided in the data products (refer to the *ISOCAM Handbook* for caveats on their use). These improved values are obtained by applying a correction for errors in focal lengths, sensor alignments and sun position. Set the keyword *corrected* to access these corrected values.

new_otf Setting this keyword implements the improved OTF slicing criteria.

bs This keyword is only useful for beam-switch observation data – see Section 19.5.1.

method='nomode' A new slicing mode 'nomode' has been introduced to **erdtoscd** and **spdtoscd** which slices only according to hardware changes of ISOCAM (such as filter, integration time, etc...) and ignores changes of flags set by the commanding software (such as the observing mode).

/old_data This flag should be set when slicing CAM parallel data (CIPH) that has been produced by OLP versions prior to OLP 8.0. Data produced by OLP 8.0 do not need this flag.

19.2 Slicing 'continuous' or 'glued' ERD/SPD data products

Experts users of CIA may be interested in slicing 'continuous' ERD/SPD files, i.e. data products that were concatenated from sequential observations and therefore hold contiguously observed data. The data products are 'glued' together with the CAM Glue tool (available from the IDA).

The ERD files (CIER or CPER) are concatenated into a CCER file, the SPD files (CISP or CPSP) are concatenated into a CCSP file, the pointing files are concatenated into CCIP and CCRP, and the compact status into a CCST file. CIA 4.0 is now capable of directly slicing these files. Use one of **spdtoscd**, **erdtoscd** or **x_slicer** according to choice.

19.3 Saturation warnings during slicing

spdtoscd warns users of saturated data with messages like

```
pixel [14,17] is affected by saturation at SCD 3 with the average
value 4093.00 (value for End-of-Integration, 3 readouts)
```

This indicates that pixel [14,17] was saturated, and its photometry has to be carefully assessed. Using the complex mask option (!MASK = 1) and setting the option */sat_mask* these pixels can be masked out, too.

19.4 A beam-switch observation caveat

Some observers have programmed their beam-switch observations in reverse – this means that the reference field is programmed as the source field and *vice versâ*. This problem will become apparent when the final beam-switch MOSAIC image is viewed (as contained in the *BS* PDS field .RASTER). The MOSAIC will appear inverted and will incorrectly have the astrometry of the reference field. To counter this problem set the keyword */reverse* when calling **get_sscdbs** to re-reverse the fields in the created *BS* PDS:

```
CIA> bs_pds = get_sscdbs( bs_sscd, /reverse )
```

19.5 Advanced slicing of beam-switch data (CAM03)

This section introduces the CIA user to advanced concepts in slicing beam-switch data.

19.5.1 Concatenating intermediate SCDs in a beam-switch observation

The slicing keyword *bs* may be of interest to observers with beam-switch data. Beam-switch observations have an intermediate step which occurs while slewing and data accrued in this step will appear in an SCD. Normally this SCD is considered to contain invalid data, though in reality this data may be quite usable. Setting */bs* will make **spdtoscd** merge this SCD with the previous SCD and users can later manually unmask this 'invalid' data for inclusion in the finally MOSAIC image. In addition using the keyword */bs* has the added advantage of making it possible to use **sscd_clean** on the SCDs.

Compare the following with Chapter 6.

```
CIA> spdtoscd, 'cisp05804610.fits', sscd, dir='$cia_vers/test', /nowrite

CIA> sscd_info, sscd
          15 SCDs in the SSCD: CSSC058046100101_98092617482201
 seq channel mode fltrwhl pfov tint gain offset size    ra       dec
   0  LW IDLE      LW2  6.0 25.20   1    512    1 ******* *******
   1  LW IDLE      LW2  6.0  2.10   2    512    1 ******* *******
   2  LW  OBS      LW2  6.0  2.10   2    512    1 ******* *******
   3  LW  OBS      LW2  6.0  2.10   2    512    1 ******* *******
   4  LW  OBS     LW10  1.5  2.10   2    512   14 295.462  50.518
   5  LW  OBS     LW10  1.5  2.10   2    512   47 295.462  50.518
   6  LW  OBS     LW10  1.5  2.10   2    512   48 295.437  50.547
```

```
   7  LW  OBS    LW10  1.5  2.10   2    512   48 295.462  50.517
   8  LW  OBS    LW10  1.5  2.10   2    512   47 295.417  50.501
   9  LW  OBS    LW10  1.5  2.10   2    512   48 295.463  50.518
  10  LW  OBS    LW10  1.5  2.10   2    512   47 295.488  50.488
  11  LW  OBS    LW10  1.5  2.10   2    512   48 295.463  50.518
  12  LW  OBS    LW10  1.5  2.10   2    512   28 295.508  50.534
  13  LW IDLE    LW10  1.5  2.10   2    512   24 295.463  50.518
  14  LW IDLE     LW2  6.0 25.20   1    512    1 ******* *******

CIA> cleaned_sscd =  sscd_clean(sscd)
Out of 15 SCDs:
4 are rejected due to mode
5 are rejected due to csh flag
6 are rejected due to qla flag
In total 8 are accepted

CIA> sscd_info, cleaned_sscd, /deg
           8 SCDs in the SSCD: CSSC058046100001_98092617532800
 seq channel mode fltrwhl pfov tint gain offset size    ra       dec
   0  LW  OBS    LW10  1.5  2.10   2    512   47 295.462  50.518
   1  LW  OBS    LW10  1.5  2.10   2    512   48 295.437  50.547
   2  LW  OBS    LW10  1.5  2.10   2    512   48 295.462  50.517
   3  LW  OBS    LW10  1.5  2.10   2    512   47 295.417  50.501
   4  LW  OBS    LW10  1.5  2.10   2    512   48 295.463  50.518
   5  LW  OBS    LW10  1.5  2.10   2    512   47 295.488  50.488
   6  LW  OBS    LW10  1.5  2.10   2    512   48 295.463  50.518
   7  LW  OBS    LW10  1.5  2.10   2    512   28 295.508  50.534

CIA> bs_pds = get_sscdbs( cleaned_sscd )
```

Calibration can proceed as normal from this point.

19.6 Advanced slicing of CVF data (CAM04)

This section introduces the CIA user to advanced concepts in slicing CVF data.

19.6.1 Up and down CVF observation

Since the LW CVF is split into two segments, one complete LW CVF scan must be performed in two observations. This means that a complete up and down LW CVF scan is comprised of four observations. Consequently, **spdtoscd** will output four SSCDs from the CISP data product.

```
CIA> spdtoscd, 'cisp10402702.fits', raw_sscd, dir='OLP1:[IA.SVAL.7_0_B_3]', $
CIA> /nowrite

CIA> cleaned_sscds = sscd_clean( raw_sscd )
Out of 345 SCDs:
5 are rejected due to mode
2 are rejected due to csh flag
```

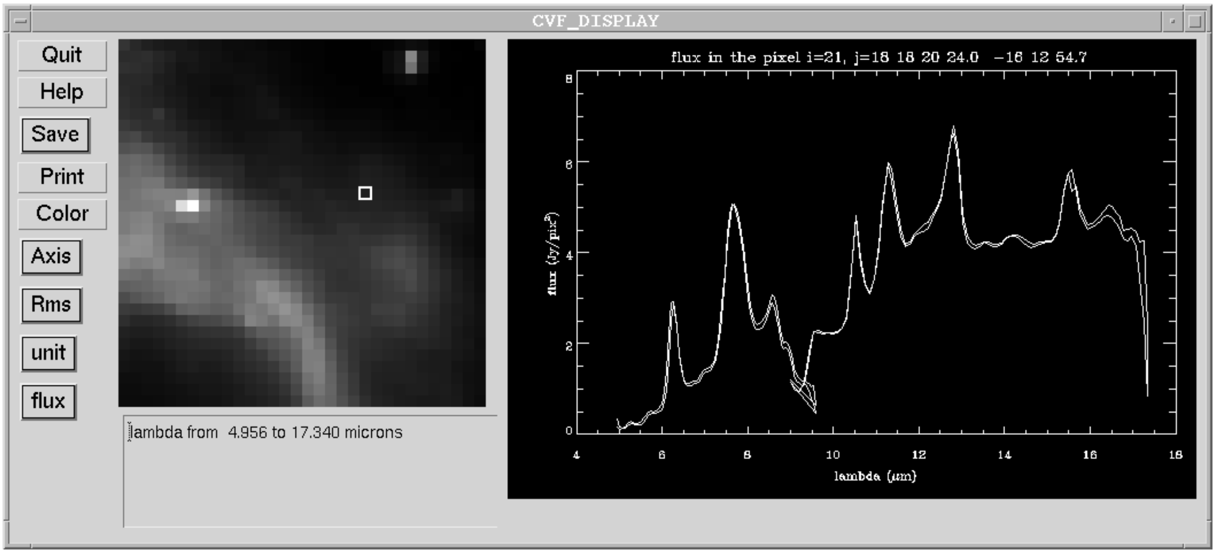

Figure 19.1: Spectrum from an up and down LW CVF scan. The overlap between LW CVF segments at 9.5 μm can clearly be seen.

```
8 are rejected due to qla flag
In total 337 are accepted
16-Jun-1998 19:19:37.00  SSCD_CLEAN v.2.0
<Splitting SSCD into 4 segments - I>

CIA> print, cleaned_sscds
CSSC104027020001_98061619193759 CSSC104027020002_98061619194660
CSSC104027020003_98061619195839 CSSC104027020004_98061619201089
```

These four SSCDs may be concatenated into one SSCD with **sscd_concatene**. Below, the last three SSCDs are concatenated to the first SSCD.

```
CIA> sscd_concatene, cleaned_sscds[0], cleaned_sscds[1]
CIA> sscd_concatene, cleaned_sscds[0], cleaned_sscds[2]
CIA> sscd_concatene, cleaned_sscds[0], cleaned_sscds[3]
```

A *CVF* PDS containing all the data from the up and down LW CVF scan can be created from this single big SSCD.

```
CIA> cvf_pds = get_sscdcvf( cleaned_sscds[0] )
```

After calibration a spectrum from an up and down CVF scan should look something like that in Figure 19.1.

Note that the same procedure can be applied to an up and down SW CVF scan. The difference is that there will only be 2 SSCDs to concatenate since there is only one SW CVF segment.

19.6.2 Mixed LW and SW CVF observation

The example in this section is from a CVF observation that is comprised of a mixture of LW and SW data. This is what the output SSCD of **spdtoscd** looks like:

```
CIA> sscd_info, sscd
        140 SCDs in the SSCD: CSSC849007010101_98092714013700
 seq channel mode fltrwhl pfov tint gain offset size    ra       dec
   0  LW IDLE LW-CVF2  1.5 25.20   1     512    1 999999.99 +999999.99
   1  LW IDLE LW-CVF2  1.5  2.10   1     512    1 999999.99 +999999.99
   2  LW IDLE LW-CVF2  1.5  2.10   2     512    4 174615.21 -285011.96
   3  LW  OBS LW-CVF2  1.5  2.10   2     512    2 174615.22 -285012.12
   4  LW  OBS LW-CVF2  1.5  2.10   2     512    1 999999.99 +999999.99
   5  LW  OBS LW-CVF1  1.5  2.10   2     512  117 174615.22 -285012.00
   6  LW  OBS LW-CVF1  1.5  2.10   2     512   15 174615.22 -285012.00
   7  LW  OBS LW-CVF1  1.5  2.10   2     512   15 174615.22 -285012.00
   8  LW  OBS LW-CVF1  1.5  2.10   2     512   16 174615.22 -285012.01
   9  LW  OBS LW-CVF1  1.5  2.10   2     512   15 174615.23 -285011.99

                    ⋮                  ⋮                  ⋮
                    ⋮                  ⋮                  ⋮

  44  LW  OBS LW-CVF1  1.5  2.10   2     512   16 174615.21 -285011.98
  45  LW IDLE LW-CVF1  1.5  2.10   2     512    8 174615.22 -285012.01
  46  LW  OBS LW-CVF1  1.5  2.10   2     512    1 999999.99 +999999.99
  47  LW  OBS LW-CVF2  1.5  2.10   2     512   37 174615.22 -285012.03
  48  LW  OBS LW-CVF2  1.5  2.10   2     512   15 174615.23 -285012.02

                    ⋮                  ⋮                  ⋮
                    ⋮                  ⋮                  ⋮

  87  LW  OBS LW-CVF2  1.5  2.10   2     512   16 174615.22 -285012.00
  88  LW  OBS LW-CVF2  1.5  2.10   2     512   16 174615.22 -285011.99
  89  LW IDLE LW-CVF2  1.5  2.10   2     512    8 174615.22 -285012.00
  90  SW IDLE SW DARK  3.0  2.10   2     512    1 999999.99 +999999.99
  91  SW CLEA SW DARK  3.0  0.28   2     512   49 174615.22 -285012.02
  92  SW IDLE SW DARK  3.0  2.10   2     512    6 174615.22 -285011.94
  93  SW  OBS SW DARK  3.0  6.02   2     512    1 999999.99 +999999.99
  94  SW  OBS SW DARK  3.0  6.02   2     512    1 999999.99 +999999.99
  95  SW  OBS  SW-CVF  1.5  6.02   2     512   41 174615.22 -285011.99
  96  SW  OBS  SW-CVF  1.5  6.02   2     512   10 174615.22 -285012.00
```

sscd_clean will discard the intermediate states and split the raw SSCD into three SSCDs, two for each LW segment and one for the SW segment.

```
CIA> cleaned_sscd = sscd_clean( sscd )
Out of 140 SCDs:
10 are rejected due to mode
14 are rejected due to csh flag
13 are rejected due to qla flag
In total 125 are accepted
27-Sep-1998 15:47:05.00  SSCD_CLEAN v.2.3
<Splitting SSCD into 3 segments - I>
```

The variable *cleaned_sscd* is a string array containing the names of three SSCDs with data from observations done with the first CVF LW segment, the second CVF LW and the CVF SW segment. The LW data from both segments can be combined into one SSCD for conversion to a *CVF* PDS:

```
CIA> sscd_concatene, cleaned_sscd[0], cleaned_sscd[1]

CIA> lw_cvf_pds = get_cvfstruct( cleaned_sscd[0] )
```

19.7 Advanced slicing with x_slicer

This section[1] completes the guide, begun in Section 12.3, to **x_slicer**.

This section addresses all possibilities that are allowed by **x_slicer**. They have been classified by the order of appearance of the windows in which they occur.

19.7.1 Files, directories and x_slicer customization

19.7.1.1 Where are the data

By default, **x_slicer** will search some directories for the data. It uses three variables to do so. When you use the 'Automatic Find' to get all the files, these variables are used. They are all defined at the beginning of the x_slicer.pro routine (see Section 19.7.1.2 to customize these settings). Here is how it goes for ERD:

- Start a pickfile (IDL program routine) in the *default_data_directory*.
- From this point, the user searches for ERD and selects it. Let us call the selected directory *user_directory*.
- **x_slicer** looks for the IIPH file in the *user_directory*.
- If needed, and if a CDS is not to be used, **x_slicer** looks for the IFPG file in the *user-_directory*.
- If the IFPG file is not in the *user_directory*, **x_slicer** assumes that it is on the ISO CD-ROM and looks for it in *user_directory/../OTHERS* (assuming UNIX notations).
- **x_slicer** looks for the CSTA file in the *user_directory*.
- **x_slicer** looks for the orbit file in the *user_directory/../OTHERS* with the name ORBIT.FIT.
- If the orbit file is not found, **x_slicer** looks for it in the *default_orbit_directory* under the name *default_orbit_file*.

Unless you are working at the ISO data center at Saclay, the *default_data_directory* is set to ./ on the UNIX system and to `arc_dat` on the VAX system. In UNIX, the search for the ERD (or STore Data, or TDF) will therefore start from the directory where **x_slicer** is invoked. On the VAX/VMS system, the user can choose **x_slicer**'s working directory by typing the following command *before* entering the CIA session:

```
$ DEFINE ARC_DAT SAPIO1$DKA200:[DELANEY.X_SLICER_DATA]
```

[1] Taken from Aussel H., 1996, *ISOCAM Data Preparation with X_slicer v2.1*, Sections 3 & 4.

19.7.1.2 Customizing the default data directory.

By editing the **x_slicer** code, you can customize the default data directory for all the people working on a node. To do this, follow these steps:

- Run a CIA session on the machine where your data are.
- Enter the following commands:

  ```
  CIA> COMMON SESSION_PARAMS
  CIA> print, NODE
  ```

  The IDL name of your machine is printed out.
- Open the x_slicer.pro routine with a text editor.
- Look for the lines *CASE node OF*.
- Add your machine name in the case statement with the proper definition of the variables *default_data_directory*, *default_orbit_file* and *default_orbit_directory*.

19.7.1.3 Bypassing the default data directory

In any case, if you wish to bypass the value of the *default_data_directory*, you can use the keyword */here* to start slicing in the directory where you are working with CIA. Just enter:

```
CIA> x_slicer, /here
```

19.7.1.4 Output Directory

Even if you are working on a VAX/VMS system, the **x_slicer** will never write to the ARC_DAT directory. If you do not give it a directory where it can write the data, he will by default try to write them to the current directory, i.e. where CIA is currently running.

19.7.2 Slicing STore Data

19.7.2.1 STore Data files (STD)

STore Data files contain data from the ground calibration. They therefore contain no astrometry information. They are IDL SAVE files. They contain the following variables:

- CAM_CUBE: the data cube of CAM frames.
- IMT_BLOCK: an array containing the block of CAM telemetry except the frames.
- IMT_RECORD: a structure to find the variables in the IMT_BLOCK.
- plus copies of values of important variables.

STD files contain only CAM telemetry.

19.7.2.2 Slicing STD files

Since STD files contain only CAM telemetry, none of the IIPH, IPFG, CSTA and ORBIT file are usable. Otherwise, these files are sliced the same way as other types of files.

19.7.2.3 Problems with STD files

Since they were mostly produced on VAX/VMS you may have problems to read them under UNIX. If this occurs, **x_slicer** will inform you of this kind of problem. What you have to do is:

- Log onto a VAX system.
- Begin an IDL session.
- Restore the STD file.
- Save it again, using the */XDR* keyword.
- FTP it to your UNIX system (do not forget the BIN command...)

Or, more simply, slice it directly on your VAX system...

Otherwise, except for pointing information, STD is the internal format that is used by **x_slicer** for the data representation. Therefore, no problem should occur.

19.7.3 Slicing ERD files

Edited Raw Data files (ERD) were preferred to Standard Processed Data files (SPD) for historical reasons – mostly because the pipeline has made the least alterations to ERD with respect to the telemetry.

To slice an ERD properly you also need:

- an IIPH file
- a CSTA file
- and sometimes, an IFPG file

19.7.3.1 IIPH files

IIPH files are Instantaneous Instrument Pointing History files: they contain the information about the RA, DEC and ROLL angle of the satellite during the observation as well as information about satellite jitter. They are calculated for the prime instrument.

IIPH files have evolved as the pipeline software (OLP) has been refined. In the distant past it was possible to accidentally receive an IIPH file from another instrument while CAM was prime – see Section E.2. Now that OLP has been scientifically validated these problems should no longer exist. A brief history of OLP...

- Up to version 2.41, OLP did not specify the instrument for which the IIPH was computed.
- Version 2.43 and later specified the instrument for which the IIPH was computed.
- Version 4.1 and later filtered the RA, DEC and ROLL values.
- Version 6.1 and later provides improved values of RA, DEC and ROLL, namely CRA, CDEC and CROLL.

Moreover, it is in the IIPH that **x_slicer** reads the crucial information about how the observation went (was it OK, or was the Target not acquired, etc...). If you have the misfortune to see on your screen a message like the one of Figure 19.2, two possible reasons are:

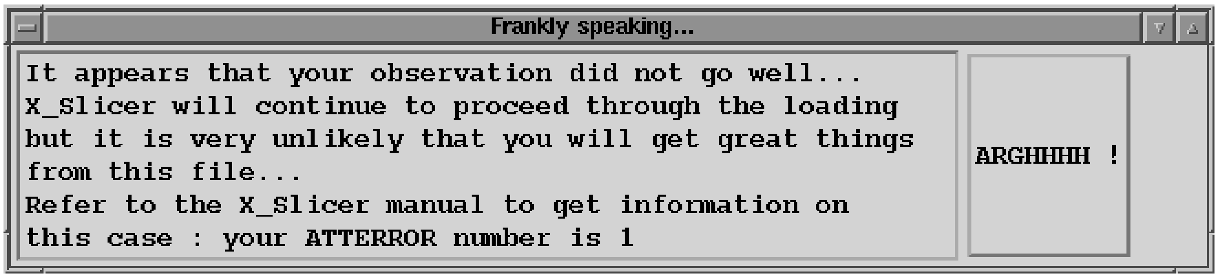

Figure 19.2: **x_slicer**'s dreaded message!

- ATTERROR = 1: The target was not acquired. Your observation is lost.
- ATTERROR = 2: Due to a break in the telemetry down-link system, the IIPH file will contain uncorrected data from the star-trackers. All your RA, DEC and Roll will be false, but your images should be OK.

For normal observations, both these problems should be spotted during the quality checking process – affected data would not be distributed by ESA.

19.7.3.2 IFPG files

The Instrument Focal Plane Geometry file contains the rotations that allow transformation of the pointing information given in an IIPH from one instrument to another – this may be useful if you wish to attempt to work on non-scientifically validated data (see Section E.2). Since the focal plane of ISO will not change during the mission, it has been written to a CDS (Calibration Data Set). The IFPG file is therefore no longer useful, use the CDS instead.

19.7.3.3 CSTA files

CSTA files are Compact STAtus files. They contain a summary of the CAM CONFIGURATION. This information can be used to check that the observation was properly done. Since the **x_slicer** displays what is in the data, you can do that job too. Therefore, you may omit it when slicing, especially if you know what should be in the data.

19.7.3.4 ORBIT files

This file is used to calculate the position and speed of ISO with respect to the Earth. The position may be useful if you are observing very nearby solar system objects because of the parallax. Though the speed is useful for the ISO spectrometers, the spectral resolution of ISOCAM's CVF mode is of the order of $\lambda/50$ and so is not affected by ISO's speed. You may therefore omit the ORBIT file.

19.7.4 Slicing TDF

TDF files are Telemetry Distribution Files. They contain CAM telemetry for the prime and parallel mode[2]. TDF files are used by instrument experts only, and can be accessed at the ISO Data Centers at VilSpa, Saclay or Orsay. Telemetry was delivered only during PV phase by ESA.

[2] If CAM is parallel, the TDF also contains the telemetry of the prime instrument.

19.7.4.1 TDF files types

Here it becomes a bit tricky...the CAM microprocessor is of the 680x0 family. Its internal coding of integers follow the IEEE convention: the most significant byte is the first. Telemetry is received at VilSpa ESA Station and processed on VAX computers. These, like all based on Intel microprocessor follow the opposite convention: the least significant byte is the first... A lot of byte swapping is therefore done. Two kinds of TDF files are created – they are distinguished by their extension.

- LTDF files are sequential files (.LTDF).
- TDFG files are indexed files (.TDFG).

All CAM slicers work on LTDF files. If you feed them with a TDFG file, it will be converted into the LTDF format first (without you being told about).This operation can last some time...

19.7.4.2 Loading TDF

TDF files are very big files, normally covering a whole orbit. It may be a challenge for your computer just to load the entire file. Therefore it is better to select a part of a telemetry file rather than the whole. Three keys may be used for this, in order to pick the interesting part of the TDF: they are UTK, ITK and FMT.

- UTK stands for Uniform Time Key: its value is the number of 1/24 s that have passed since the 1 November 1995 at 0h00. Its value is updated every image in the TDF (F2 format).
- ITK stands for Instrument Time Key: its value is the number of *CAMTU* or CAM Time Unit (roughly 0.14 s) that have passed since the 1 November 1995 at 0h00. Its value is updated every image in the TDF (F2 format)
- FMT stands for TDF FORMAT. Its value is the number of TDF formats (one every 2 seconds) sent since the 16 November 1995 18h36. It is of course updated every TDF format.

See the *ISO Data Product Document*, Appendix B, for additional information on these keys.

19.7.4.3 Slicing TDF

Taken all that has been said in the last two subsections into account, slicing TDF files is just the same as slicing ERD files. Remember that you cannot use any coordinate information like IIPH, or ORBIT. It is also impossible to check the contents of a TDF versus the commanded position.

19.7.5 Selecting slicing variables

The problem of selecting slicing variables has been widely addressed in Section 12.3, but since it is one of the most important things that the user has to do when using **x_slicer**, we wish to recall some points here. First of all, Table 19.7.5 gives the complete list of the slicing variables that can be used with **x_slicer**. In order to avoid confusion, the official names of variables in the telemetry files, in SCDs and DSD, are also given.

Let us now discuss some of these variables.

x_slicer name	**telemetry name**	**SCD names**[a]	**DSD names**
CAM wheels positions			
entrance wheel	F1_W1_POS	ENTWHL CAL.EWHL	F1_BLOCK.HK1.POS1
selection wheel	F1_W2_POS	SELWHL CAL.SWHL	F1_BLOCK.HK1.POS2
LW lens wheel	F1_W3_POS	PFOV CAL.PFOV	F1_BLOCK.HK1.POS3
LW filter wheel	F1_W4_POS	FLTRWHL CAL.FCVF	F1_BLOCK.HK1.POS4
SW lens wheel	F1_W5_POS	PFOV CAL.PFOV	F1_BLOCK.HK1.POS5
SW filter wheel	F1_W6_POS	FLTRWHL CAL.FCVF	F1_BLOCK.HK1.POS6
detector and its configuration			
observation channel	F2_IM_ORIG	CHANNEL CAL.DEID	F2_BLOCK.F2IMORIG
integration time	F2_INT_TIME	TINT CAL.TINT	F2_BLOCK.F2INTTIM
detector gain	F2_ADC_GAIN	GAIN CAL.GAIN	F2_BLOCK.F2ADCGAI
detector offset	F2_ADC_OFFS	OFFSET CAL.OFFSET	F2_BLOCK.F2ADCOFF
electronic setup			
on-board process	F2_IM_PROC	OBP_MODE CAL.OBC	F2_BLOCK.F2IMPROC
observation mode	F2_AOT_OPM	MODE CAL.MODE	F2_BLOCK.F2AOTOPM
observation related variables			
observation type	F2_AOT_OBS	COMMANDER CAL.COMMANDER	F2_BLOCK.F2AOTOBS
A.O.T	F2_AOT_AOT	AOCT	F2_BLOCK.F2AOTAOT
beam-switching	F2_OPER_FLAG	BEAM	F2_BLOCK.F2BMSWFL
raster mode	F_RAST	F_RASTER	F2_BLOCK.F_RAST
M raster position	M_RAST	M_RASTER	F2_BLOCK.M_RAST
N raster position	N_RAST	N_RASTER	F2_BLOCK.N_RAST
parallel mode	F2_PARALLEL	TELEMETRY CAL.TELEMETRY	F2_BLOCK.F2PARALL
specific variables			
On Target Flag	F2_OTFLAG	CUBE.HK.QLA_FLAG[b]	F2_BLOCK.F2OTFLAG
sequence number	F2_AOT_CNF	STN	F2_BLOCK.F2AOTCNF

[a] SCD names column sometimes contains two names. This is because you will find an explicit value of the variable in the SCD top level, and an integer value in one of its substructures (usually the .CAL one).

[b] Note that only the OTF is allowed to vary in an SCD, because it is stored in the HK (house keeping) substructure. All other variables have to be constant in an SCD. Moreover, OTF is only a part of the QLA flag.

Table 19.1: Slicing variables used in **x_slicer**.

19.7.5.1 Wheels position and Detector related variables

The meanings of these variables are straightforward. The detector offset variable is no longer useful, because there is only one offset per gain of the detector. The important point is the difference between channel and selection wheel position.

19.7.5.2 Channel versus selection wheel

One of the common mistakes is to think that choosing one of these two variables is enough to determine if the images are LW ones or SW ones. Think twice: it can happen (due to wheel motion or dark measurement for examples) that the light beam goes through the SW channel while the LW detector is on... You will then get LW images, but the slicer (and you) will believe that they are SW ones. It may result that the SCD will not be created if you mix the two kinds of images.

19.7.5.3 Electronic setup

These two variables are rather important:

- On Board Processing: This variable describes the kind of treatment that is applied to your observations by ISOCAM's processor. It can take three values: 'Normal', 'Accumulated', or 'Sampled'. If you work in 'Accumulated' mode (frames coadded), CIA will take into account this kind of observation. Remember two things: in your SCD, the field 'TINT' displays the effective integration time, i.e. the number of accumulated frames times the integration time of the camera for each frame. On the other hand, the field CAL.TINT gives you the integration time for one individual frame (in CAMTU).

- Observation Mode: This variable tells you if you are observing ('OBS'), idle ('IDLE'), waiting for a good configuration ('WAIT'), or in GAP or DARK mode. If you are looking at CUS data (see the value of the Observation Type variable) it is very likely that you will get an 'IDLE' value instead of an 'OBS' for your observations. This can be corrected in the SCDs:

```
CIA> scd1 = sscd_elem( sscd )

CIA> for i=0, n_elements( scd1 ) - 1 do scd_put, 'MODE', 'OBS', scd1[i]
```

19.7.5.4 Observation-related variables

These variables are again simple to understand. Only two remarks have to be made:

- the F_RAST variable in telemetry can only be on (1) or off (0). But in an SCD its value can be 'RASTER', 'MICRO_SCAN', 'STARING', 'TRACKING' or 'UNDEFINED' (see Section 15.2.1). The definition of a micro-scan is very loose:

  M step-size and N step-size $\leq 16''$
  M step-size or N step-size $\leq 8''$

  This means that a micro-scan refers to more than the special case of where the step-size is not an integer number of pixels.

- M_RASTER and N_RASTER can be set to zero by a break of telemetry on a raster position for one or two frames. Check these variables carefully because even if M_RASTER and N_RASTER are false, the images will be correct. The **x_slicer** will propose to break the SCD corresponding to this raster position into three parts. You will have to use the 'Merge' buttons to rebuild a whole SCD.

19.7.5.5 On target flag and sequence number

The On Target Flag (OTF) variable appears in two places while using **x_slicer**. First, you can select it as a slicing variable. This is no longer recommended, but that possibility has been left available. Second, you are asked in the **x_handle_slice** window if you wish to use the Enhanced OTF. These are two different things. It will be explained in a whole section below.

The Sequence number can sometimes be useful when more than one observation is contained in a file as it can help to discriminate between them. The Sequence number (STN) in the SCDs are recomputed according to their place in the SSCD. It is likely that you will get different values in the STN field than the one displayed by **x_slicer**.

19.7.5.6 Automatic slicers slicing variables

As you probably know by now, there are two other slicers in CIA. These are the automatic slicers: **spdtoscd** and **erdtoscd**. See Section 12.2 for examples of usage, and also Sections 17.1.3 and 17.1.4. They both use three methods of slicing (the default method, *Andy* and *gap_mode*), described by the Table 19.7.5.6. The default is *new_method*.

The main difference between these automatic slicers and **x_slicer**, is that they only create one SSCD per observation. You can use **x_slicer** to create any number of SSCDs per observation, though in general it is only desirable to have one per CONFIGURATION.

19.7.6 The x_handle_slice window

This window is really the heart of the program (from the user's point of view). Most of the important actions are chosen here.

19.7.6.1 Using the 'Advanced Slicing' Menu

This menu makes the manipulation of data very easy. There is only one point to remember: it ONLY allows to PERFORM SELECTIONS. You have therefore to press the 'Unselect All SCDs' button before using it. Then, choose the desired parameter in the menu and click on it. Use then the 'Redisplay' to see the result of your choice. With this last button, you can select the SCDs according to an 'AND' logical rule, for example *LW2 filter AND 5.0 s of integration AND PFOV of 3 arcsec AND number of frames greater than 15.*

19.7.6.2 Using choice related buttons

A choice in **x_slicer** is the list of the selected SCDs just before you hit the 'Redisplay' button. Except for the 'First Choice' that is the result of the **slicer_a_moi** routine that was first displayed to you.

variable	new_method	Andy	gap_mode
F1_W1_POS	yes	no	no
F1_W2_POS	yes	no	no
F1_W3_POS	yes	no	no
F1_W4_POS	yes	no	no
F1_W5_POS	yes	no	no
F1_W6_POS	yes	no	no
F2_OPER_FLAG	yes	yes	no
F2_AOT_OBS	yes	yes	no
F2_AOT_AOT	yes	yes	no
F2_AOT_OPM	yes	yes	no
F2_AOT_CNF	yes	yes	no
F2_IM_ORIG	yes	yes	yes
F2_INT_TIME	yes	no	no
F2_IM_PROC	yes	no	yes
F2_ADC_GAI	yes	no	no
F2_ADC_OFF	yes	no	no
GPSCRPID[a]	yes	yes	no
WMOTION[b]	no	yes	no

[a]GPSCRPID is the variable containing Instrument Time Key (ITK) for each record of the file and is therefore not really a slicing variable.

[b]WMOTION is a non zero variable if one wheel is moving.

Table 19.2: Slicing variables used by CIA's automatic slicers.

19.7.6.3 Selecting and Merging SCDs

The **x_handle_slice** window allows you to select SCDs one by one. To select or unselect the SCDs, just click on the radio button on the SCD. The **x_handle_slice** window allows you also to 'merge' SCDs. Merging two SCDs is simply to glue them together in order to have only one. two rules must be followed :

- you can ONLY MERGE SELECTED SCDs
- you can ONLY MERGE CONTINUOUS SCDs, i.e. no hole in the data.

If you do not follow these rule, it is very likely that the slicer will crash.

Merging SCDs is very interesting if you work on data that have suffered from telemetry drop-outs, for example if you have 'false' M and N_RASTER values (see above). If this is the case, something like the following is displayed:

SCD number	no. frames	M_RASTER	N_RASTER
i	3	3	2
i+1	2	0	0
i+2	7	3	2

Select the SCDs i, i+1 and i+2, then press the 'Merge i and i+1' button and the 'Merge i+1 and i+2' button. The error will be corrected.

19.7.6.4 Build only SCDs of SPD flavor

If your observation is done using the 'Normal' mode, you receive two frames per image: a Reset frame and a End Of Integration (EOI) frame. If you do not ask for it, the **x_slicer** will create SCDs of ERD flavor, *i.e.* with EOI end Reset frames. If you ask for it, it makes the slicer call the **erdtospd** routine that will translate this format into real images (it is only a subtraction LW and little more complicated for the SW).

19.7.6.5 DSD files

DSDs are Diagnostic Specific Data files. These files are interesting only for the ISOCAM experts in case of an instrument problem. Fortunately there was no ISOCAM problems during the mission, so there was no reason to access these files. CIA contains some routines (**ia_all_status**, **ia_temp_status**, ...) to perform some checks on the DSD files.

19.7.6.6 Saving slicer file

The use of this button is important if you work on big datasets (see below).

19.7.6.7 Entering names and directories

At the point of really creating your SCDs (after hitting the 'Continue' button) you will be asked to enter a name and a directory.

AGAIN DO NOT FORGET TO HIT THE RETURN KEY, otherwise **x_slicer** will not take into account what you entered...

For people used to the previous version, **x_slicer** now fills in the 12 characters with some '_'s if your name is shorter.

19.7.7 On Target Flag

Two kinds of On Target Flag (OTF) exist. The first one is the F2_OTF_FLAG that is in the telemetry. This is one of the slicing variables. This flag is set on when the pointing of the satellite is less than 10 arcsec away from the intended position. This is of course much less than the ISO real pointing precision (but not far off its initial specifications). This flag was of some use at the beginning of CIA because some routines did not separate on target frames from other ones. This is no longer the case. Moreover, some routines work on a cube without holes: transient correction for example. By using this variable, you will create such holes. For these reasons, this slicing variable should no longer be used.

The second one is the Enhanced OTF (see Section 19.7.7.2).

An important point to understand is that the OTF that appears in SCDs is not this F2_OTF_FLAG, but a combination of it plus some other variables. This point will be addressed now.

19.7.7.1 OTF in SCDs: QLA_FLAG

In your SCDs, the cube.hk.qla_flag variable plays a big role. It tells you if your frame is in good shape (this role is played by the F2_QLA telemetry variable, coded on one bit), if the pointing is correct for normal mode (F2_OTF_FLAG) and for sample or accumulated mode (F2_OTF_SUM). Its value is defined as:

$$scd.cube.hk.qla\_flag = F2\_QLA + 2 * F2\_OTF\_FLAG + 4 * F2\_OTF\_SUM$$

From this definition, a good value of the scd.cube.hk.qla_flag is 7.

19.7.7.2 Enhanced OTF

A button in the **x_handle_slice** window asks you if you wish to use the 'Enhanced OTF'. The default configuration is 'Yes'. This OTF relies on the IIPH OTF. It makes a fit on the RA and DEC coordinates of the IIPH and finds out the best positions near the source, it then flags them on. To discriminate this method from the other, the value 128 is added to the scd.cube.hk.qla_flag. Good values are then 143.

You should not use this feature if you work with real micro-scans (see M_RASTER comment) or if your data has suffered telemetry drops. This is because the fit of RA and DEC will not work: their variations are too small between two raster positions and the routine may crash.

19.7.8 Handling big datasets

Sometimes, you will face a huge amount of data in one file. By huge, we mean something like more than 5000 frames. Since **x_slicer** uses a large amount of memory for its widgets, this reduces the available memory, and introduces the risk of a crash when handling large data sets.

This problem can be avoided by use of the *big file* button option in the main **x_slicer** window (see Figure 12.1). Clicking on this button will cause **x_slicer** to work in a mode where it minimizes the amount of data that it loads into memory, e.g. when it needs to know the values of some keys in FITS file, instead of loading the entire file into memory it simply reads the required keys. The behavior of **x_slicer** in this mode is transparent to the user.

In addition to using the *big file* button, you may want also want use the *save slicer file* button (located in the **x_handle_slice** window – see Section 12.2). See Section 19.7.9 for details on its use.

19.7.9 The *save slicer file* button and slicer_to_cia

If you have a large data set, you can simply slice them with **x_slicer**, working on data with the **x_handle_slice** window. Only the last action is different. Instead of hitting the 'Continue' Button, hit the 'Save Slicer file' button. You will then be asked to enter a SSCD name. Once you have done that, hit the 'OK' button. **x_slicer** creates a file named *sscd_name.sli*. This file contains the current **x_slicer** settings, but no data.

To use this file, just quit **x_slicer** and even quit CIA in order to free all memory. Then enter CIA again and type:

```
CIA> COMMON slicer_data,toto

CIA> slicer_to_cia,'sscd_name.sli',dir_sli='directory_of_sscd_name.sli', $
CIA> sscd, dsd, /save
```

All your SCDs will be created, just as if you never exited **x_slicer**. Moreover, SSCD and DSD variables will contain your created SSCD and DSDs.

IMPORTANT: check first that the data you entered in **x_slicer** are still in the same directories, because the .sli file does not contain any data – it simply informs the **slicer_to_cia** routine where they may be found.

19.7.10 Getting your SSCDs

To work on your data after having exited **x_slicer**, there are two ways to proceed:

- use the routine **sscd_read** to read them back from the disk. This is time consuming if you have big SSCDs.
- enter the command:

  ```
  CIA> print,sscd_list()
  ```

  Pick the sscd that you are interested in.

19.7.11 Frequently Asked Question

19.7.11.1 Why use x_slicer

Because it is beautiful, uh ?

19.7.11.2 x_slicer compilation

- **x_slicer** doesn't compile...

  You should probably reread the section about **x_slicer** customization, because it does compile...

- **slicer_to_cia** doesn't compile...

  That is because a COMMON block is not defined. You probably exited IDL after having sliced a big file, and you tried to read your .sli file... Reread carefully Section 19.7.8 or type the following commands:

```
CIA> COMMON slicer_data, Oh_my_God_what_a_stupid_name_for_a_variable

CIA> .run slicer_to_cia
```

19.7.11.3 x_slicer did not do what I told it to do...

- **x_slicer** did not write the SCDs where it was told to...

  Are you sure that you hit the return key after entering the path in the directory field ? Remember that the default directory is the one from which you launched **x_slicer**.

- **x_slicer** did not create all the SCDs I asked for...

  You probably asked for SCDs of 1 frame: as they do not exist, they were not created.

19.7.11.4 x_slicer crashes...

- **x_slicer** crashes just in the beginning...

  You should probably reread the section about **x_slicer** customization, because you probably made a mistake in entering your preferred directory.

- **x_slicer** crashes while creating SCDs...

  You probably asked it to build a SCD with no frames, or with a hole in it (by misusing the merge buttons).

- **x_slicer** crashes while saving SCDs...

  Are you sure you have enough place on your disk? Or even the right to write on it?

Chapter 20

Advanced data calibration

In this chapter of the *CIA User's Manual* different calibration methods and algorithms are briefly reviewed and their implementation in CIA is described. Detailed information on calibration algorithms can be found in the technical reports listed in Appendix I. Probably the most helpful report for a novice is the *ISOCAM Handbook*. This report contains a broad treatment of calibration algorithms – it, and the technical reports, will be referenced throughout the chapter where appropriate. Since more up-to-date algorithms and routines are continually being developed it is also a good idea to refer to the on-line help or **cia_help** (see Section 2.3.2).

20.1 Before reading this chapter...

Throughout this chapter references are made to the Prepared Data Structure (PDS) (Section 15.5) and the CIA calibration routines (Chapter 13). Familiarity with both topics is assumed. You will also frequently encounter references such as .IMAGE, .CUBE, .MASK, etc.... These are general references to fields and substructures in the PDS.

20.2 Core calibration

You have already been introduced to the core calibration routines in Section 13.2.1. Here we will look a little deeper into the different correction methods that are available for each of the core calibration routines. The chosen correction method can be specified with the keyword *method*. Each method is implemented by a low-level routine which may accept keywords for tuning algorithm parameters. These keywords may be specified when calling the core calibration routine and IDL will pass the keywords to the low-level routine (i.e. using IDL's keyword inheritance functionality). To find out what tuning can be done for a particular method, look for the relevant low-level routine in the online help (Section 2.3.2).

20.2.1 Dark correction

In principle, dark correction is a simple procedure – a matter of subtracting the DARK image from the IMAGE. The hard part is choosing a good DARK. All dark correction methods are handled by **corr_dark**. Technical details of some of these methods can be found in the references given below.

In general, **corr_dark** performs the dark correction in the following manner:

1. Obtains the DARK as instructed by the user setting of the keyword *method* or *indark*.

2. Divides the IMAGEs of .CUBE with .ADU_SEC_COEFF (so as to normalize .CUBE to ADU/sec/gain.)
3. Subtracts the IMAGEs of .CUBE by the DARK.
4. Places the DARK in the field .DARK.
5. Finally, updates the field .CUBE_UNIT to ADU/sec/gain.

The different dark correction methods are detailed below.

One important point about dark correction that the user should be aware of: **corr_dark** treats all the data in .CUBE as having the same integration time. However, since it is possible to create a *general* PDS from data of mixed CONFIGURATIONs then the situation may arise where the integration time is not constant throughout the .CUBE. In these cases **corr_dark** will fail. To avoid this problem the user should perform dark correction on the SSCD before creating the PDS.

1. `method='vilspa'`

   method: DARK model correction. Computes a dark which depends on the detector temperature, time since activation and the integration time.

   called routine: dark_vilspa

   PDS side effects: The field .DARK is filled with the DARK used for correction. The IMAGEs in .CUBE are dark corrected, i.e. .CUBE is modified.

   reference: Ott S. and Roman P., 2000, In preparation.

2. `method='model'`

   method: DARK model correction. Compute the best dark for given revolution and time-since-activation of ISOCAM. This is a new method and seems to yield excellent results. It is also the default method.

   called routine: darkmodel

   PDS side effects: The field .DARK is filled with the DARK used for correction. The IMAGEs in .CUBE are dark corrected, i.e. .CUBE is modified.

   reference: *ISOCAM dark current calibration report*

3. `method='library'` or `method='calg'`

   method: CAL-G DARK correction. A DARK may be selected from the CAL-G DARK library. As you probably know by now, the most-up-to date CAL-G files are distributed in CDSs with the CIA system. When you create a PDS from a SSCD, **get_sscdstruct** automatically uses **find_best** (see Section 20.11) to place the most suitable DARK from the latest CIA DARK CDS into the PDS field .CALG.DARK (see Section 15.3.2). With this method .CALG.DARK is applied to the .CUBE.

   called routine: darklibrary

   PDS side effects: The field .DARK is filled with the DARK used for correction. The IMAGEs in .CUBE are dark corrected, i.e. .CUBE is modified.

   reference: *ISOCAM Handbook*, Chapter *Data processing methods*, Section *CALG dark*.

4. `indark=my_dark`

   method: Correcting with your own DARK. If you have your own DARK (obtained for example from CAM in OP-MODE DARK) then it can be supplied to **corr_dark** via the keyword *indark*. This DARK would then used to apply correction in the normal way.

   called routine: *N/A*

   PDS side effects: The field .DARK is filled with the DARK used for correction. The IMAGEs in .CUBE are dark corrected, i.e. .CUBE is modified.

   reference: *N/A*

20.2.2 Deglitching

The core calibration routine **deglitch** serves as an interface to all the available low-level deglitching routines. As for the other core calibration routines, you can choose a particular deglitching method by setting the keyword *method* to a particular string value. The possible values for *method* and the associated deglitching methods (and the associated routines which **deglitch** calls) are described in the following subsections. Technical details of some of these methods can be found in the references given below.

1. `method='spat'`

   method: Spatial and temporal deglitch. This method attempts to use both temporal and spatial information of the .CUBE to perform deglitching. It finds the difference between successive IMAGEs in the .CUBE – we will refer to this as the difference image. Then it applies a thresholding criterion to each pixel of the difference image, based on the median of the standard deviation of the difference image and the median of the mean of the difference image. A pixel failing this criterion corresponds to a glitch in the IMAGE.

   It is considered to be a good method when the .CUBE is not stabilized. However, it may be poor at detecting long duration or close, successive glitches.

   routine called : deglitch_spat

   PDS side effects: Glitches removed from IMAGEs in .CUBE, i.e. .CUBE is modified. Glitched pixels also flagged in .MASK.

   reference: *ISOCAM Handbook*, Chapter *Data processing methods*, Section *Spatial and temporal deglitch*.

2. `method='temp'`

   method: Temporal deglitch. By this method pixels failing a combined median and standard deviation test in temporal space are considered glitches.

   This method needs a steady background level and so is not very suitable for very unstable data.

   routine called: deglitch_sig

   PDS side effects: Glitches removed from IMAGEs in .CUBE, i.e. .CUBE is modified. Glitched pixels also flagged in .MASK.

   reference: *ISOCAM Handbook*, Chapter *Data processing methods*, Section *Temporal deglitch*.

3. `method='tcor'`

method: Deglitch *tcor*. This method attempts to attempts to avoid the problems that arise from instability by initially transforming the cube into a zero-mean cube and then deglitching. The algorithm is:

(a) Compute the zero-mean, D, of the input .CUBE, $cube\_in$:
$D = \frac{2}{\sqrt{6}} cube\_in(*, *, i) - \frac{1}{\sqrt{6}} cube\_in(*, *, i-1) + cube\_in(*, *, i+1))$

(b) Compute S_t, the $Nsigma$ clipping of D

(c) For each IMAGE of $cube\_in$:

i. Compute I, where $I = IMAGE - median(IMAGE, 5)$

ii. Compute S_p, the $Nsigma$ clipping of I

iii. A glitch has occurred if $abs(D(*, *, f)) > S\_t$ and $abs(I) > S\_p$

iv. Replace glitched pixels $cube\_out(i, j, k)$ by $cube\_out(i, j, k-1)$

Some criticisms of the *tcor* have been made – under certain conditions it will eradicate your data. As with all deglitching techniques it should be used with care.

routine called: method_tcor

PDS side effects: Glitches removed from IMAGEs in .CUBE, i.e. .CUBE is modified. Glitched pixels also flagged in .MASK.

reference: online help.

4. `method='mm'`

method: Deglitch *MM* (Multiresolution Median Transform). The Multiresolution Median Transform works on the principle that in temporal space glitches are small scale structures and source signal will always be large scale structure. Consequently, this method may fail for glitches of very long duration. In general though, it is very robust and handles non-stabilized data well. It is especially good for observations where many IMAGEs are accrued. This method is the default.

routine called: deglitch_mm

PDS side effects: Glitches removed from IMAGEs in .CUBE, i.e. .CUBE is modified. Glitched pixels also flagged in .MASK.

reference: *ISOCAM Handbook*, Chapter *Data processing methods*, Section *Deglitching using the Multiresolution Median Transform (MMT)*.

5. `method='part'`

method: Particle method. Transforms the .CUBE of a PDS into a new .CUBE where glitched pixels are replaced by the value in the previous IMAGE of .CUBE. The algorithm is:

(a) Elimination of IMAGEs with a strong transient. It has been shown to be impossible to correct these IMAGEs from glitches.

(b) Identification of all glitches. For that, a local threshold is computed by calculating the noise between successive IMAGEs after an N_sigma clipping technique.

(c) Subtract the original .CUBE from the deglitched .CUBE obtained in the first step.

(d) Particles impacts are identified. All single pixel or double pixel events are retained only if they have a minimum energy. All 3 or more pixels events are retained.

(e) Original .CUBE is deglitched again by replacing all found glitched pixels by the values of the corresponding previous frame of the cube. Histograms of number of pixel/events, total energy, and max energy in a single pixel are calculated.

6. `method='manu'`

method: Deglitch *manu*. The IMAGEs in .CUBE can be manually deglitched with this method. Upon calling, instructions for use are displayed in the CIA command window and a graphics window displaying a single IMAGE at a time from .CUBE is opened. The mouse is used to select suspected glitched pixels and those pixels are replaced with the median of their neighboring pixels.

Note that there are two other manual deglitchers, **man_mask** and **sl_viewcube**, in the *contrib* directory of CIA. In addition **x3d** (Section 14.4.4) now has manual deglitching capability.

An example call is:

```
CIA> deglitch, pds, method='manu'
```

This method is arduous, and is probably best used to get rid of persistent glitches that other deglitching methods have failed to find.

routine: deglitch_man

PDS side effects: Glitched pixels are removed from IMAGEs in .CUBE, i.e. .CUBE is modified. Glitched pixels are also flagged in .MASK.

reference: *ISOCAM Handbook*, Chapter *Data processing methods*, Section *Manual deglitch*.

20.2.3 Stabilization

There are several methods that may be used to achieve stabilization, all of which are implemented by **stabilize** and individual low-level routines. As with the other core calibration routines the keyword *method* selects the required method. All the available methods are described below, but before reading on it is worth noting that there are two kinds of stabilization methods. (i) Masking methods that simply flag unstable pixels in .MASK. (ii) Fitting methods that modify .CUBE in an attempt to compensate the data for the transient response of the CAM detector. The fitting methods may also flag pixels. Technical details of some of these methods can be found in the references given below and in the technical reports listed in Appendix I.

Before proceeding with the details of the different methods, **stabilize** has an important keyword option that the user should be aware of. This is the keyword *timeline*. It determines how the timeline is generated for the transient correction. It can be set to one of the following string values:

arrival uses the BOOTTIME (available for OLP 7.0 products) or UTK.

tint creates the timeline from the integration time. No telemetry drops are taken into account.

scd creates the timeline from the integration time. However, telemetry drops within an SCD are taken into account.

1. `method='fs'`

method: This method uses a fitting technique based on the Fouks-Schubert model. It is best used on an SSCD – see Section 20.3 for an example.

There are three different implementations of this algorithm: an IDL version, a Fortran version and a C++ version. To use the IDL or Fortran version set the **stabilize** keyword */idl* or */fortran*.

Some points to note about these different implementations:

- The IDL version does not work well with irregularly gridded data, as can occur in the pixel history if a glitched pixel is recorded by the complex MASK. Consequently it is recommended to only use the simple MASK with the IDL version. See Section 2.3.4 for configuring the MASK.
- The Fortran and C++ versions can handle irregularly gridded data. In this case both MASK settings are fine, though the complex MASK is recommended as it correctly indicates that glitched pixels should be ignored.

routine called: corr_transient_fouks

PDS side effects: Unstable pixels are flagged in .MASK and .CUBE is modified.

reference: Coulais A. and Abergel A., 2000

2. `method='s90'`

method: Pixels which have not achieved 90% of their stable values are masked. This is the default stabilization method.

routine called: corr_transient_s90

PDS side effects: Unstable pixels are flagged in .MASK and .CUBE remains unmodified.

reference: *ISOCAM Handbook*, Chapter *Data processing methods*, Section *Find the first stabilized pixel at each position without using a model.*

3. `method='m90'`

method: This method applies a model to find the first stabilized pixel at each STATE. Pixels which have not achieved 90% of their stable values are masked.

routine called: corr_transient_m90

PDS side effects: Unstable pixels are flagged in .MASK and .CUBE remains unmodified.

reference: *ISOCAM Handbook*, Chapter *Data processing methods*, Section *Find the first stabilized pixel at each position using a model.*

4. `method='inv'`

method: This method uses a fitting technique called the IAS or INVERSION Model. This method is one of the more successful fitting methods.

routine called: corr_transient_inv

PDS side effects: Unstable pixels are flagged in .MASK and .CUBE is modified.

reference: *ISOCAM Handbook*, Chapter *Data processing methods*, Section *Simplified model of the CAM LW response.*

5. `method='vision'`

method: This method uses a model called the VISION Model. This is a rather severe method and can kill off much of your data. It does produce very nice clean images though – it is good at eliminating ghosts from raster MOSAICs.

routine called: corr_transient_vision

PDS side effects: Unstable pixels are flagged in .MASK and .CUBE is modified.

reference: *ISOCAM Handbook*, Chapter *Data processing methods*, Section *Vision model*.

6. `method='med'`

method: Global transient removal using a median filter. Best for extracting faint point-like sources from a raster observation with many pointings. See the online documentation (Section 2.3.2) for detailed restrictions and caveats.

routine called: corr_transient_med

PDS side effects: Undefined pixels are flagged in .MASK and .CUBE is modified.

reference: online help.

20.2.4 Reducing IMAGEs to EXPOSUREs

All the IMAGEs accrued in each CAM STATE (or ISO pointing) can be averaged to an EXPOSURE. Although different averaging methods exist, in general the routine **reduce** reduces the IMAGEs as follow:

1. The sets of IMAGEs corresponding to individual STATEs are extracted by **reduce** from .CUBE.
2. Each set of IMAGEs, along with .MASK, is passed to **reduce_cube**.
3. **reduce_cube** averages the IMAGEs to an EXPOSURE taking .MASK into account – pixels which have been flagged because of instability, glitches, etc... are ignored. The EXPOSURE, its RMS image and the weight image are returned. Each pixel of the weight image equals the number of IMAGE pixels which have been averaged to create a corresponding pixel in the EXPOSURE.
4. The EXPOSURE, its RMS image and the weight image are placed into the PDS fields .IMAGE, .RMS and .NPIX respectively.

The default averaging method is to take a mean of the IMAGEs. Other available methods are described below.

1. `/median`

method: Takes the median of the IMAGEs to form the EXPOSURE.

routine called: la_median

PDS side effects: The PDS fields .IMAGE, .RMS and .NPIX are filled.

20.2.5 Flat-fielding

Currently several methods of flat-fielding, all of which are handled by **corr_flat**, are available. In general **corr_flat** performs flat-fielding as follows:

1. Looks at the keywords *method* and *inflat* to determine how the FLAT is to be selected.
2. Divides each EXPOSURE of .IMAGE with the selected FLAT, or, if the **corr_flat** keyword */cube* is set then the IMAGEs in .CUBE are divided by the FLAT.
3. Finally, places the FLAT in the PDS field .FLAT.

For all the different methods of flat-fielding the impact on the PDS is the same. The field .FLAT is filled with the .FLAT used for flat-fielding. Either the reduced EXPOSUREs are flat-fielded (.IMAGE is modified) as is the default, or if the keyword */cube* is set then the IMAGEs are flat-fielded (.CUBE is modified).

1. `method='library'` or `method='calg'`

   method: CAL-G or library FLAT correction. In a similar manner as for the CAL-G DARK, **get_sscdstruct** uses **find_best** (see Section 20.11) to find the most suitable DFLT and OFLT images from the CIA CDSs and places the product of these images, i.e. the FLAT, in the PDS field .CALG.FLAT. When the the keyword *method* is set to `method='library'`, **corr_flat** uses the FLAT in .CALG.FLAT.

   If the **corr_flat** keyword */cube* is set then the flat-fielding is performed on the IMAGEs, otherwise it is performed on the EXPOSUREs.

   called routine: flat_library

2. `method='oflat'`

   method: CAL-G or library OFLAT correction. This is exactly the same method as for `method='library'` except that only an optical flat-field correction is performed.

   called routine: flat_library

3. `method='oflat'`

   method: CAL-G or library DFLAT correction. This is exactly the same method as for `method='library'` except that only a detector flat-field correction is performed.

   called routine: flat_library

4. `method='auto'`

   method: Automatic creation of the FLAT. In the case of a raster observation, a very good FLAT can be determined from the actual observation data. To use this 'auto' FLAT set the keyword *method* to `method='auto'`.

   The algorithm used to create the 'auto' FLAT is as follows:

   (a) A median image is derived from the EXPOSUREs in .IMAGE, or if the keyword */cube* is set, a median image is derived from the IMAGEs in .CUBE.
   (b) The resulting median image is then normalized by dividing by its mean.
   (c) This FLAT is placed in the PDS .FLAT and flat-fielding proceeds in the usual way.

called routine: flat_auto

5. `method='manu'`

   method: A FLAT may be manually from observation data. In this case **corr_flat** calls the routine **flat_builder** to interactively aid you in obtaining the perfect manual FLAT – see Section 20.9. After exiting **flat_builder** your custom FLAT is placed in the PDS field .FLAT and flat-fielding proceeds as normal.

   called routines: flat_builder

6. `inflat=my_flat`

   method: Flat-fielding with your own FLAT. If you happen to have your own FLAT, e.g. say *my_flat*, then you can pass this to **corr_flat** by setting `inflat=my_flat`. Again, *my_flat* will be placed in .FLAT and the flat-fielding procedure will continue as usual.

   called routine: *N/A*

20.2.6 Flat-fielding and lens wheel jitter

Due to lens wheel jitter CAM images can become considerable shifted in the horizontal or instrument y axis direction. This shift is caused by a misalignment between the lens and the optical axis. The extent of this shift can be up to 3 pixels. This shift also has the effect of making flat-field correction with the library or CAL-G optical flat-field images invalid. These flats expect a well aligned lens wheel. As an alternative, data from suitable raster observations may be flat-fielded with the 'auto' method. However, this is not an option for data from raster observations with few raster points and certainly not an option for data from other AOTs. A further alternative is suggested here:

1. Create a rough 'auto' flat from the IMAGEs in a PDS. This will be used to measure the shift caused by the lens wheel:

   ```
   CIA> flatauto = flat_auto(raster_pds)
   ```

2. Use **xdisp** to measure the shift in the 'auto' flat image.

   ```
   CIA> xdisp, flatauto
   ```

   The shift will be apparent in two ways:

   - You may see dark columns at the edge of the image. The number of such columns helps to indicate the shift in the image.
   - Features in the rough 'auto' flat should correspond to features in the library flat. Any misalignment will indicate the amount of shift. You can check the library flat with:

     ```
     CIA> xdisp, raster_pds.calg.flat
     ```

   You might also want to try correlating *flatauto* with the library flat *raster_pds.calg.flat* using the IDL Astronomy User's Library routine **correl_optimize**. However, this routine does not usually work well with images of this nature.

   Whichever method you choose you need to measure the (x, y) positions of the limits of the image containing good quality data.

3. It is important to mark the boundary of the image beyond which the pixel values are invalid due to the shift. Given the (x, y) positions determined previously, **flag_edges** will generate a mask with 1 where pixels are valid and 0 elsewhere:

```
CIA> raster_pds.npix = $
CIA> flag_edges(x1, y1, x2, y2, raster_pds.nscd)*raster_pds.npix
```

   When the raster MOSAIC is created the invalid pixels will not be included.

4. Now correct the library flat with **register_flat**. This routine will shift the optical flat and multiply it by the detector flat.

```
CIA>  my_flat = register_flat(x1, y1, raster_pds)
```

5. Supply the corrected flat to **corr_flat**:

```
CIA> corr_flat, raster_pds, inflat=my_flat
```

   That completes the flat-field correction.

20.3 Calibrating an SSCD

In most cases users are only interested in calibrating a PDS and the SSCD is quickly discarded. However, there are certain situations where the calibration of an SSCD is very more useful. Mostly these cases require that contiguous data over as a long a time-scale as possible is available for input to the calibration routines. Remember that a PDS does not necessarily hold contiguous data – see Section 13.1.1 for details of the limitations of a PDS. In particular, data from polarization observations are best calibrated using an SSCD – see Chapter 8 for an example of this.

Here we have an example of how and why you would calibrate an SSCD. This example illustrates the usage of the VilSpa dark correction and the Fouks-Schubert transient correction method (see Section 20.2.3). This transient correction method works best with the contiguous data that is provided in an SSCD. We use the same dataset as in Chapter 3.

1. Create the SSCD and SCDs:

```
CIA> spdtoscd, 'cisp02600506.fits', sscd, dir='$cia_vers/test', /nowrite

CIA> sscd_info, sscd, /deg
          43 SCDs in the SSCD: CSSC026005060101_98052614571745
 seq channel mode fltrwhl pfov tint gain offset size    ra       dec
   0  LW IDLE     LW2  6.0 25.20   1     512     1 ******* *******
   1  LW IDLE     LW2  6.0  2.10   2     512     1 ******* *******
   2  LW  OBS     LW2  3.0  5.04   1     512    46 180.498 -18.849
   3  LW  OBS     LW2  3.0  5.04   1     512    24 180.490 -18.871
   4  LW  OBS     LW2  3.0  5.04   1     512    24 180.482 -18.893
   5  LW  OBS     LW2  3.0  5.04   1     512    24 180.473 -18.915
```

```
   6  LW   OBS     LW2  3.0  5.04   1     512    24 180.450 -18.908
   7  LW   OBS     LW2  3.0  5.04   1     512    23 180.458 -18.886
```

```
etc...
```

2. Perform dark correction and deglitching on the SSCD:

```
CIA> corr_dark, sscd, method='vilspa'

CIA> deglitch, sscd
```

3. Stabilize the entire SSCD. Remember that there are 4 CONFIGURATIONs in this observation. By performing transient correction on the SSCD we give the Fouks-Schubert fitting method (see Section 20.2.3) as much contiguous data as possible, hence increasing the quality of the correction.

   As recommended for the Fouks-Schubert method, we need to create an initial stabilized image. This is derived by a simple median reduction of the data in the first SCD that contains good data. A quick look at the output of **sscd_info** above tells us that this is the third SCD in our SSCD:

```
CIA> scds = sscd_elem(sscd)

CIA> data = scd_get('model', scds[2])

CIA> stab_img = reduce_cube(data, /median)
```

   You can ignore any output from **reduce_cube**. Now supply this image to **stabilize**:

```
CIA> stabilize, sscd, method='fs', stab_img=stab_img
```

4. Clean the SSCD and split into new SSCDs for each CONFIGURATION.

```
CIA> cleaned_sscd = sscd_clean(sscd)
```

5. Create a *raster* PDS from the first CONFIGURATION. Reduce, flat-field the data and build the MOSAIC image in the usual way.

```
CIA> raster_pds = get_sscdraster(cleaned_sscd[0])

CIA> reduce, raster_pds

CIA> corr_flat, raster_pds

CIA> raster_scan, raster_pds
```

   This step may be repeated for the remaining 3 CONFIGURATIONs or 3 SSCDs.

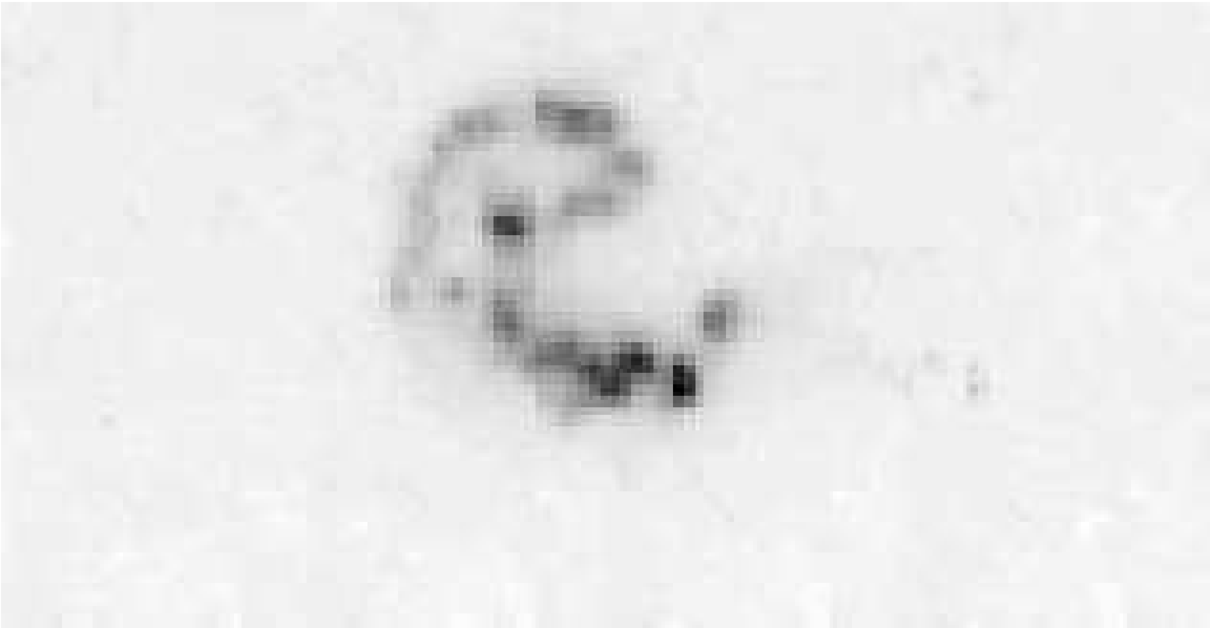

Figure 20.1: The raster MOSAIC with SSCD calibration and Fouks-Schubert transient correction. Display for comparison with Figure 3.3.

6. It might be interesting to compare the results of this calibration and those in Chapter 3 – see Figure 20.1 and Figure 3.3 respectively. Clearly in Figure 20.1 the background is cleaner and the ghost image to the left of the source is not present.

```
CIA> tviso, raster_pds.raster
```

20.4 Raster MOSAIC creation

Final step in calibrating a *raster* PDS is the creation of the raster MOSAIC from the EXPOSUREs in .IMAGE. Four methods can be used to compute the MOSAIC, all of which are handled by **raster_scan**. The different methods can be selected by setting the keyword *method*. In general all the methods involve a weighted averaging of EXPOSURE pixels, where the weight is the square root of the EXPOSURE pixel weight. That is, for each EXPOSURE pixel .IMAGE[i,j,k] the weight is taken as .NPIX[i,j,k].

In all cases the impact of raster MOSAIC creation on the PDS is the same. The field .RASTER is filled with the computed raster MOSAIC. The fields .NPIXRASTER is filled with the number of IMAGE pixels from which each pixel in .RASTER has been derived. The field .RMS is filled with the RMS of these IMAGE pixels.

Note the raster MOSAIC size is determined when the *raster* PDS is created from an SSCD with **get_sscdraster**. The size is computed to be optimum for the raster step-size and PFOV. However, the user may increase the computed size by a specified magnification factor. For example, to have a MOSAIC which is twice as large in both dimensions as the computed size:

```
CIA> raster_pds = get_sscdraster( sscd, magnify=2 )
```

1. `method='noproj'`

method: EXPOSUREs are overlayed and averaged to produce the raster MOSAIC. The position of each EXPOSURE within the raster is determined from the raster step size (.M_STEPCOL and .N_STEPLINE) and the number of raster positions (.RASTERCOL and .RASTERLINE).

This method works well when the actual coordinates of the EXPOSUREs are consistent with the intended coordinates of the raster positions. This is usually the case. However, it may fail when the raster extends over a large angle, or when the raster step-size is comparable to the PFOV.

called routine: raster_scan_noproj

2. `method='project'`

method: EXPOSUREs are corrected for FOV distortion and then projected onto the raster MOSAIC FOV according to their individual coordinates. A C-coded program, *projection* performs the actual projection.

This is probably the best method for raster MOSAIC creation. It does a good job of taking into account all the different raster types, e.g. North axis, micro-scan. It also works even if the actual coordinates of the EXPOSUREs do not match their intended positions in the raster. It is the default method.

For more on projection and coadding images with different astrometry see Section 20.14.

called routine: projette and the executable *projection*.

3. `method='idlproj'`

method: Same method as `method='project'`, the difference being that an IDL coded projection routine is used.

called routine: raster_scan_idlproj

4. `method='camera'`

method: By this method the EXPOSUREs are rebinned to 256×256 images and then coadded according to their position in the raster. The EXPOSUREs are always kept aligned to ISOCAMs axes even if the raster is a NORTH axis raster – this means that the axes of the raster MOSAIC will also be aligned to ISOCAMs axes.

This method is of most advantage when raster steps sizes are not multiples of pixel sizes – the initial rebinning of the EXPOSUREs make the raster step size a whole multiple of the pixel size.

called routine: raster_cam

20.5 Creation of the beam-switch MOSAIC

After all the calibration steps described in Section 20.2 have been performed on a *BS* PDS then the beam-switch MOSAIC maybe created. This is done with **reduce_bs**.

The process is simple:

1. Determine which of the EXPOSUREs of the *BS* PDS field .IMAGE are source pointings and which are reference pointings. (The source and reference EXPOSURES are indexed by .REF_IMAGE and .SRC_IMAGE respectively.)

2. The source and reference EXPOSURES are coadded (with **reduce_cube**) taking into account .NPIX. The coadded reference image is subtracted from the coadded source image. The result is the beam-switch MOSAIC.
3. .RASTER[1] is filled with the beam-switch MOSAIC. .NPIXRASTER is filled with the total number of EXPOSURE pixels that are used to compute each pixel in .RASTER. .RMSRASTER is filled with the RMS image of .RASTER.

20.6 CVF analysis

Dedicated CVF analysis is described in this section.

20.6.1 Sensitivity and straylight correction

Sensitivity correction, or more simply conversion from ADU to milli-janskys (mJy), can be performed by dividing the EXPOSUREs in .IMAGE by the sensitivity correction factors stored in .RESPONSE[i].SENSITIV. When a *CVF* PDS is initially created with **get_sscdcvf**, the sensitivity and straylight correction factors are taken from a CDS and placed in .RESPONSE.SENSITIV. This is similar in principle to the way the CAL-G FLATs and DARKs are handled.

conv_flux applies the correction and converts the EXPOSURE pixels to mJy.

```
CIA> conv_flux, cvf_pds, /image
```

The EXPOSURE pixels are now converted mJy. The *CVF* PDS field .IMAGE_UNIT is updated to reflect this. Note that this operation is not reversible. To re-perform sensitivity correction use **reduce** to recreate the EXPOSUREs and hence refill .IMAGE.

20.6.2 Photometry on faint point sources

For a CVF observation, the source positions moves, depending on the wavelength, slightly within each CVF segment. For changes from one segment to another (there are two LW segments), or a switch of the detector channel, the source position can jump several pixels. This behavior makes photometry on CVFs more difficult. To overcome this, **compute_spectrum** computes for each wavelength (or CVF step) for a point source the best fitting of a PSF, and uses this information to compute a shift-corrected flux spectrum.

After a the CVF observation has been fully reduced, and the approximate position of a point source has been determined, e.g. by **cvf_display**, **cvf_spectrum** can be called:

```
CIA> compute_spectrum, cvf_pds, x_pos, y_pos, $
     psf_dir="SAPI01$DKA200:[CIA.DATA.PSF]", $
     sort_wave, raw_flux, sort_flux, sort_flux_error
CIA> plot, sort_wave, sort_flux
```

`psf_dir` has to point to a local directory, containing the theoretical PSFs (see Section 15.4.1). The output of the program is:

- `sort_wave`: wavelength, sorted in ascending order.

[1] .RASTER is not a likely name for a beam-switch MOSAIC. It was chosen early in the development of the beam-switch analysis routines in order to make the *BS* PDS compatible with the *raster* PDS.

- `raw_flux`: The flux of the point source (one per executed CVF step).
- `sort_flux`: The flux of the point source (one per observed wavelength). In case a wavelength was observed twice, `sort_flux` is the mean of both observations.
- `sort_flux_error`: The error of the point source flux (one per observed wavelength). This error is computed on the fly by Monte-Carlo simulations.

20.7 Tips on CIA data calibration

If you have read the last sections then you will have realized that a great choice of calibration methods exist in CIA. Though some broad recommendations can be made on which methods to use, the only way you can be sure that you are achieving good calibration is through experimentation and experience.

A good way to know what is happening is to go through each calibration step, check your result and then proceed with the next step. However, you do have to be careful how you proceed. CIA requires that certain calibration processes follow a particular order. There are broad rules you must follow if you want to get a sensible result. Some processes can be repeated without having to start from scratch with raw data.

1. Save a copy of your PDS. Some calibration methods are irreversible (they change the data in .CUBE and .MASK) and if you don't like the result you will need to begin again with fresh data.
2. Dark correction, deglitching and stabilization are all operations on .CUBE. These processes must applied before any other and in the correct order:
   (a) Dark correction.

   Example: `corr_dark, pds, method='model'`

   Caveat: Irreversible. Dark correction can only be performed once. If you don't like the result of a dark correction, then you have to start again with the original data.

   (b) Deglitching.

   Example: `deglitch, pds, method='mm'`

   Caveat: Irreversible, but you could deglitch twice. However, there is a strong danger of over deglitching your data. Best to begin again with the original data.

   (c) Stabilization.

   Example: `stabilize, pds, method='s90'`

   Caveat: Irreversible, but you could stabilize twice with non-fitting methods such as *s90*. However, it is not recommended.
3. Reduction of .CUBE to .IMAGE.

   Example: `reduce, pds`

   Caveat: Reversible. You can reduce as many times as you like. Although processes following reduction will of course have to be repeated.
4. Flat-fielding.

Example: `corr_flat, pds, method='library'`

Caveat: If the flat-fielding is performed on EXPOSUREs (as is the default) then the flat-field correction can be reversed by re-reducing .CUBE to .IMAGE so as to create 'fresh' EXPOSUREs. If the flat-fielding was performed on IMAGEs (by setting the **corr_flat** keyword */cube*) then the correction is irreversible.

5. Raster MOSAIC creation.

   Example: `raster_scan, raster_pds, method='noproj'`

   Caveat: Reversible. You can recreate the raster MOSAIC *ad nauseum.*

6. Beam-switch MOSAIC creation.

   Example: `reduce_bs, bs_pds`

   Caveat: Reversible. You can recreate the beam-switch MOSAIC *ad nauseum.*

After any of the above steps, the results can be evaluated before proceeding with the next step. Routines to aid you are:

tviso to do general CAM image displaying (see Section 14.5.1);

x3d to look at the characteristics of the cubes .CUBE and .IMAGE (see Section 14.4.4);

xsnr to do S/N analysis of an IMAGE, EXPOSURE or MOSAIC (see Section 14.1.1).

20.7.1 PDS history

In general, operations performed on a PDS are recorded in .HISTORY. It lists routines and calibration methods applied to the PDS:

```
CIA> print, staring_pds.history
 date=26-May-1998 17:21:49 node=bikini user=mdelaney
 procedure=darklibrary V 1.0 algorithm=Find best CCGLWDARK_97031713382678 END
 date=26-May-1998 17:19:22 node=bikini user=mdelaney
 procedure=spdtoscd V 2.0 algorithm=default
 (7) cisp03001209.fits <undefined> (7) $cia_vers/test <undefined>
  <undefined> <undefined> <undefined> (2) 19 (2) 119 END
 date=26-May-1998 17:21:54 node=bikini user=mdelaney
 procedure=flat_library V 1.1 algorithm=Find best CCGLWOFLT_98041510080669 END
 date=26-May-1998 17:21:55 node=bikini user=mdelaney
 procedure=flat_library V 1.1 algorithm=Find best CCGLWDFLT_98031519384439 END
 date=26-May-1998 17:21:40 node=bikini user=mdelaney
 procedure=get_sscdstruct V 2.1 algorithm=default
 CSSC030012090001_98052617205667 <undefined> <undefined> END
 date=13-Jul-1998 10:53:06 node=bikini user=mdelaney
 procedure=corr_dark V 3.9 algorithm=cube = (cube/gain/tint)-dark model END

etc...
```

If you use CIA routines to convert your PDS to a FITS file, the text in .HISTORY is saved in the FITS header (see Chapter 18).

20.8 Dealing with dead pixels

Usually, the only dead pixels you need worry about are the four of the SW detector and column 24 of the LW detector. However, if you have very heavily glitched or unstable data, then after calibration a pixel may be masked all the way through a set of IMAGEs from a STATE. When reduced to an EXPOSURE, such pixels will be effectively dead in that EXPOSURE. Their corresponding value in .NPIX will be zero, in other words they will have a zero weight. **raster_scan** ignores all dead pixels – if EXPOSUREs do not overlap where a dead pixel occurs, then a blank spot will appear in the MOSAIC.

Note the following when you are dealing with data from the LW detector: after calibration, but before building the raster MOSAIC, **calib_struct** automatically smoothes column 24 in the .IMAGE and .CUBE. However, the weight of every pixel, i.e. the values in .NPIX, in column 24 will always remain as zero. Therefore column 24 will appear as a dead column in the MOSAIC. However, it will no longer be column 24, but column (.NX_RASTER-8).

Dead pixels appearing in the raster MOSAIC are optionally smoothed by **calib_raster**:

1. `/dead`

   method: Where dead pixels appear in the raster MOSAIC, i.e. where .NPIXRASTER is zero, a median smoothing method is applied. Only the dead pixels are smoothed into their neighbors – no other pixels are affected.

   called routine: im_smooth

   PDS side effects: Dead pixels in .RASTER are smoothed.

20.9 Making custom FLATs with flat_builder

flat_builder allows you to build a FLAT from EXPOSUREs in a raster observation. EXPOSURES can be interactively selected and a FLAT created from a median or mean. The FLAT can be tested and tailored to produce optimum results. An additional feature is that regions of a selected EXPOSURE can be excluded from the FLAT computation, so the danger of including source signal in the FLAT can be avoided.

20.9.1 Building a simple flat

Suppose we have a *raster* PDS. **flat_builder** accepts the cube of EXPOSUREs in the PDS field .IMAGE as an argument. Optionally the CAL-G FLAT (already in .CALG.FLAT) can also be supplied. This will allow you to compare your custom FLAT with the CAL-G FLAT during a **flat_builder** session. **flat_builder** can be called directly or via **corr_flat** (Section 20.2.5). If the later is chosen then upon exiting of **flat_builder** the custom FLAT will be automatically applied to the data. If the former is chosen then the custom FLAT must be used as input to **corr_flat**.

To call **flat_builder** directly:

```
CIA> built_flat=flat_builder(raster_pds.image, calg=raster_pds.calg.flat)
```

Looking at Figure 20.2 you can see the main **flat_builder** window. An additional window (not in the figure) displays all the EXPOSUREs for reference – we will call this the reference window. The following steps should guide you through using **flat_builder**:

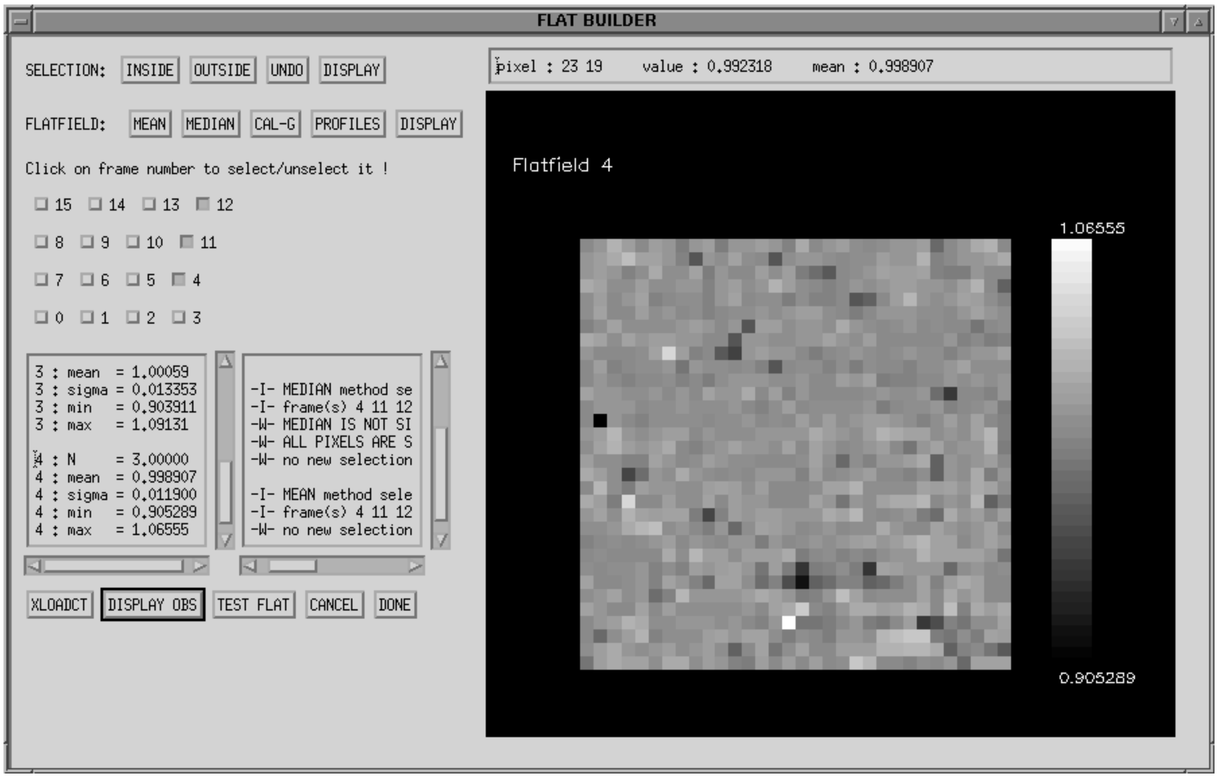

Figure 20.2: **flat_builder**'s main window.

1. You can begin by selecting EXPOSUREs which you want to make up your FLAT. Clearly it is best to pick those with no signal. Selection is made by clicking on the buttons, located in the main window under the title *click on frame number to select/unselect it.*

2. When you have finished the selection, click on *SELECTION: DISPLAY* to display all selected EXPOSUREs in the reference window.

3. You can experiment with a median of all the selected EXPOSUREs (click on *MEDIAN*) or a mean (click on *MEAN*). Each time you click the resulting FLAT is displayed in the display region of the main **flat_builder** window. Also statistical information on your FLAT and messages will appear in the text sub-windows in the main window.

4. Now click on *TEST FLAT* (bottom of main window). This applies your FLAT to the EXPOSUREs and the results are displayed in the reference window.

5. You can also examine row and column profiles of the computed FLAT by clicking on the button *PROFILES*. Along with a new plot displaying profiles, the following message will appear in the CIA command window:

   ```
   Left mouse button to toggle between rows and columns.
   Right mouse button to Exit.
   ```

   One important note: All other **flat_builder** functions are suspended while profiles are being viewed. Exit this function by clicking on the FLAT with the right mouse button.

6. You can compare your FLAT with the CAL-G FLAT. Click on *CAL-G* and then on *TEST FLAT*. The CAL-G FLAT is applied to the EXPOSUREs and again, the result is displayed in the reference window.

7. You can flip back any time to the original EXPOSUREs by clicking *DISPLAY OBS*. Also, you can invoke **XLOADCT** by clicking on *xloadct*.

When you are finished, click on *DONE*. Your FLAT will be in the array *built_flat*.

20.9.2 Advanced features

Another great feature of **flat_builder** is that you can choose regions of an EXPOSURE to be included/excluded from you FLAT. This is useful if you are short of signal-free EXPOSUREs.

1. Choose an EXPOSURE (by clicking the appropriate button under *Click on frame number to select/unselect it!*). The selected EXPOSURE is displayed in the main window.

2. With the right mouse button, click on two points on the currently selected EXPOSURE which enclose the region you desire to be excluded/included.

3. A box will appear outlining the region. Now click on *INSIDE* or *OUTSIDE*, depending on whether you want to include the selected region or exclude it.

4. Click on *SELECTION: DISPLAY* to display all the selected EXPOSUREs, including the regions of those partially selected, in the reference window.

5. You can now make the FLAT by clicking on *MEAN* – the option to use a median is not available when partial EXPOSUREs are used in the selection.

6. You test the flat by clicking on *TEST FLAT*.

7. When you are finished click on *DONE*. The most recently computed FLAT will be returned by **flat_builder**.

20.10 Background subtraction

The routine **bkg_builder** can be used to manually select IMAGEs from the cube of a PDS, or indeed any cube, for use in determining a background subtraction frame. Since this requires that you have to have IMAGES containing no source, then it is most likely that **bkg_builder** will only be of use to you if you have raster observation data.

To invoke **bkg_builder**:

```
CIA> background_frame=bkg_builder(raster_pds.cube)
```

Four windows will appear:

DATA displays all the frames, i.e. IMAGEs, in the input cube.

VIEW displays all the frames in the input cube which you have selected.

BACKGROUND displays the current background frame.

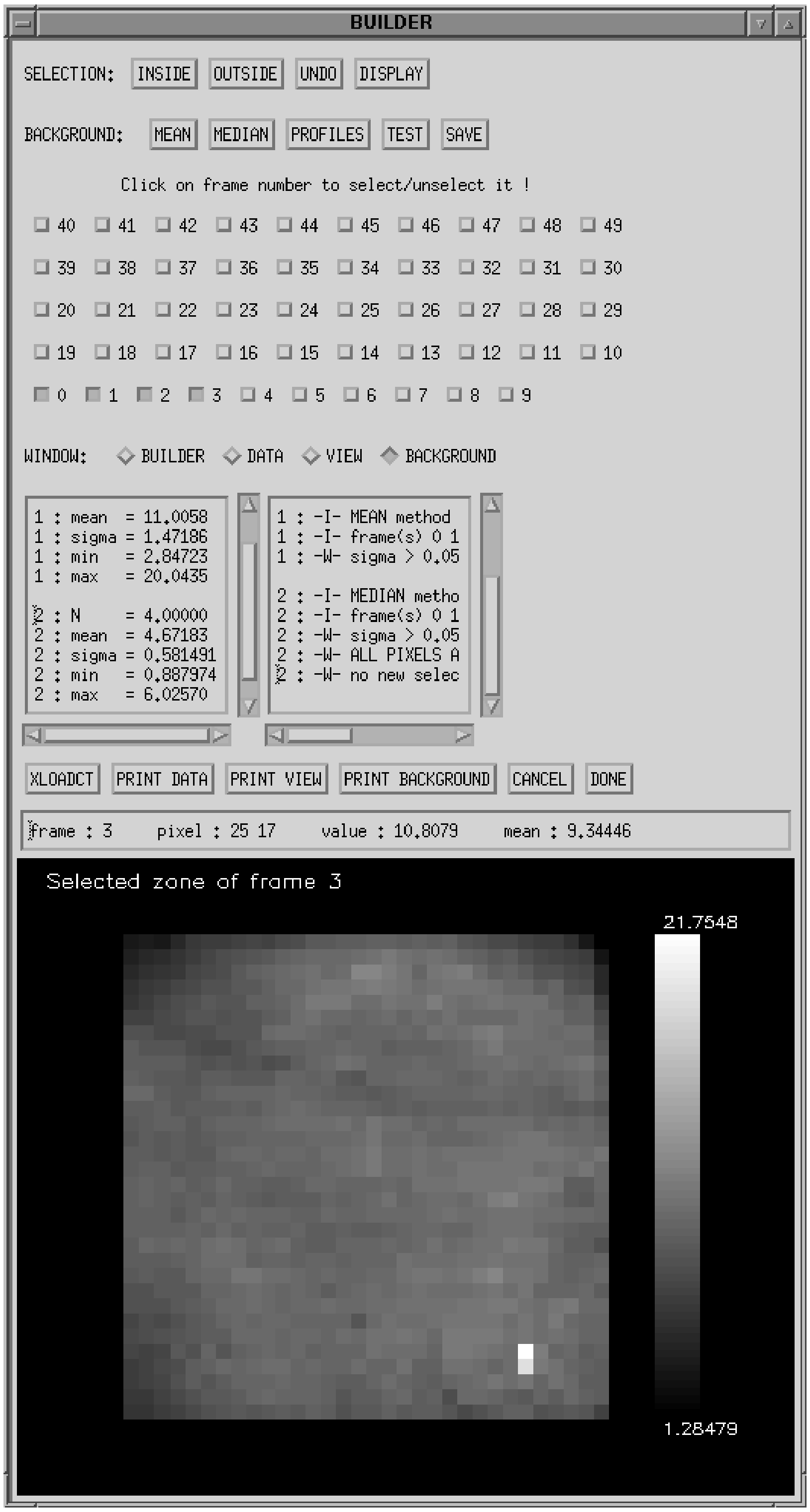

Figure 20.3: **bkg_builder**'s main window.

BUILDER is the main window and the user's interface to **bkg_builder** – see Figure 20.3.

To make a background frame with **bkg_builder** follow this basic procedure:

1. Select some frames by clicking on the panel of buttons under the legend *Click on frame number to select/unselect it!*. As you select, your choice of frames will appear in the window **VIEW** and the currently selected frame will be displayed in the bottom of the main window.
2. Now click on the buttons *mean* or *median* and the a mean or median frame of the selected frames will appear in the window **background**.
3. Some statistical information about your background frame will appear in a subwindow of the main window. When you are happy with your background frame you can click on done to exit. Now you chosen background frame will be in the variable *background_frame*.

20.11 Obtaining the best calibration record from a CDS

A CAL-G library file contains many calibration records. (Remember that the CAL-G library files are stored as CDSs in CIA – see Section 15.3.) These records are comprised of images, e.g. OFLTs, DFLTs, DARKs etc..., and associated CAM parameters, e.g. TINT, FLTRWHL etc.... Obviously, you would wish to choose the best record for your data. CIA provides the routines **find_best** and **find_best_psf** to do just this. In simple terms the algorithm employed by these routines is: they look at the CAM parameters in each calibration record and try to find the best match with the parameters of the observation data. However, some extra complexity is involved. Depending on the type of calibration data, certain parameters are given precedence and others are completely ignored.

Usually all this is handled automatically by CIA. When **get_sscdraster**, **get_sscdstruct** and **get_sscdcvf** build a PDS from an SSCD, they choose the optimal calibration record from the CDS for your observation data. These records are placed in the substructure .CALG (Section 15.5.6).

However, if you have your own CDS and an *SPD* SCD, then likewise you can use **find_best** or **find_best_psf** to extract the most suitable record for you data.

20.11.1 find_best

find_best is used for all calibration data types except PSF data (see Section 20.11.2).

Suppose that we have a an SCD and an OFLT CDS in memory:

```
CIA> spd_scd = 'CSCDLW3FOV6V4P00_96052803472412'

CIA> oflat_cds = 'CCGLWOFLT_96030223551347'

CIA> best_oflat = find_best( spd_scd, oflat_cds )
```

best_oflat is a full record from the CDS.

```
CIA> help, oflat_cds, /str

CIA> OFLT = oflat_cds.image[*,*,0]
```

Note that **find_best** finds the flat-field with the closest wavelength to that of the observation data (not with the closest wheel index).

20.11.2 find_best_psf

This routine is used in a similar way to **find_best** but of course it is just for PSF calibration data. See the on-line help or **cia_help** for more details.

20.12 Unit conversion and colour correction

This section describes how pixel units are handled in CIA and how users may convert their images into milli-janskys (mJy).

20.12.1 Propagation of pixel units within a PDS

The IMAGE pixels values in a freshly made, uncalibrated PDS, will be in units of ADUs.

```
CIA> print, pds.cube_unit
ADU
```

After dark correction the same pixels will be in units of ADU/gain/second.

```
CIA> corr_dark, pds

CIA> print, pds.cube_unit
ADU/G/s
```

After CUBE reduction these units will propagate into the EXPOSUREs.

```
CIA> reduce, pds

CIA> print, pds.image_unit
ADU/G/s
```

Likewise for MOSAIC creation.

```
CIA> raster_scan, pds

CIA> print, pds.raster_unit
ADU/G/s
```

20.12.2 Conversion to milli-janskys

At any stage in the processing you can attempt to convert pixel values to milli-janskys (mJy) with **conv_flux**. For example, after raster MOSAIC creation you can convert *.raster* pixels into mJy with:

```
CIA> conv_flux, pds, /raster

CIA> print, pds.raster_unit
mJy/pix
```

Likewise, *.cube* IMAGE pixels may be converted by setting the keyword */cube*, and *.image* EXPOSURE pixels by setting */image*.

conv_flux uses the low-level conversion routine **adu_to_mjansky**. This is a useful routine for finding the sensitivity of a particular CAM filter. For example, to convert a signal of 1 ADU observed with the LW2 filter to mJy:

```
CIA> print, adu_to_mjansky( 1, 'lw2' )
     0.429129
```

20.12.3 Color correction

The conversion to mJy as performed by **conv_flux** (Section 20.12.2 is given for the reference wavelength of each filter and assumes a source with spectral shape that follows a $F(\lambda) \sim \lambda^{-1}$ law. In reality your source may have a spectrum that deviates from this law. To correct this effect CIA provides the routine **corr_colour**. You can use this routine to calculate a correction factor that may be applied manually to the results of your CAM photometry. There are three ways to calculate this correction factor:

- If you believe your source has a blackbody spectrum then you just need to supply the blackbody temperature:

```
CIA> corr_colour, "lw10", bb=5000
filter  ok: LW10
 =====================================
 Filter                                  = LW10
 ref wavelength                          =       12.0000
 correction factor K at ref wave         =       1.27288
   by which you should divide the ISOCAM flux, to
   obtain the actual flux density of your source
```

  In this case one would correct all flux density measurements of this source by dividing by the correction factor 1.27288.

- If your source has a power law spectrum then simply supply the power law exponent:

```
CIA> corr_colour, "lw10", power=-2
```

- Finally, if you have constructed a SED (possibly from CAM measurements in different filters or from modem calculation) you can use it as input to **corr_colour**. The SED data points must be first written to a text file in a simple two column format, with wavelength (microns) in the first column and flux density (Jy) in the second. This name of this text is supplied as input to **corr_colour**:

```
CIA> $more sed.dat
     2.38014   440.23920
     2.38163   429.91971
     2.38311   421.62091
     2.38460   419.00839
```

```
      2.38610   424.57556
      2.38759   436.57462

  etc...

  CIA> corr_colour, "lw10", sed="sed.dat"
```

20.13 A note on the infamous column 24

At this stage you should be very familiar with the LW detector's dead column 24 (or by IDL convention column 23). By default most CIA processing will attempt to interpolate column 24 so that it is no longer appears in the reduced data. Not all observers will desire this. To avoid any filling of column 24 use the keyword */dead_col*. The routines that accept this keyword are:

- By default **get_sscdstruct**, **get_sscdbs**, **get_sscdraster** and **get_sscdcvf** will fill column 24 in the output PDS field .CUBE. with the average of its neighboring columns. If */dead_col* is set then column 24 is set to zero.
- **corr_dark** will ignore column 24 if */dead_col* is set, otherwise it will call **corr_col24** to fill it after performing dark correction on .CUBE.
- **corr_flat** will ignore column 24 if */dead_col* is set, otherwise it will call **corr_col24** to fill it after performing flat correction on .IMAGE (or .CUBE if the keyword */cube* is set).

Of course, at any stage in CIA processing you can fill column 24 yourself with **corr_col24**. For example to fill column 24 in .CUBE and .IMAGE

```
corr_col24, pds
```

To fill only .CUBE or only .IMAGE then set the keyword */cube* or */image* respectively.

20.14 Advanced projection

The projection is method is probably the best choice for creation of the raster MOSAIC (see Section 20.4 for other methods). There are a number of useful options available for projection and indeed the CIA projection routines have other purposes aside from the create of raster MOSAICs.

20.14.1 Distortion correction

There are a couple of options when it comes to projection and distortion correction. These can be specified as **raster_scan** keywords. Note that projection is actually performed by the lower-level routine **projette** and the C++ executable *projection*. Keywords given to **raster_scan** are passed to **projette**, so it may be a good idea to take a look at the online help for both these routines.

- Perform the projection but create the raster MOSAIC without distortion correction:

```
CIA> raster_scan, raster_pds, /nodisto
```

- Perform distortion correction, but give an alternative distortion file:

```
CIA> raster_scan, raster_pds, dist_file='lw3asr10.dis'
```

- Perform distortion correction, but use the "right" distortion file instead of the "left"

```
CIA> raster_scan, raster_pds, side=1
```

20.14.2 Weighted mean option

Normally, the weighting factor per EXPOSURE pixel is computed as a function of the good number of readouts stored in .CCIM.NPIX (see Section 15.5.8). However, in case of systematic errors (e.g. due to the flat-field uncertainty at the edge of the detector or unstabilized pixels) better results might be achieved by using the standard error instead. This error is computed as

$$\text{standard error} = \frac{\sigma}{\sqrt{n}} = \frac{.\text{CCIM.RMS}}{\sqrt{.\text{CCIM.NPIX}}} \tag{20.1}$$

The following script gives some examples how to use the different methods; **raster_pds**, a short 2×2 raster, has been dark-corrected, deglitched, transient corrected and flat-fielded with the library flat-field.

- Standard treatment (EXPOSURE pixels are weighted by CCIM.NPIX) to create the left MOSAIC of Figure 20.4:

```
CIA> raster_scan, raster_pds
```

- Projection using the weighted mean, the standard variation held in CCIM.RMS is multiplied by an estimate of the flat-field error to create the middle MOSAIC of Figure 20.4:

```
CIA> h = raster_pds.rms
CIA> corr_col24, h
CIA> raster_pds.rms = h

CIA> h = raster_pds.npix
CIA> corr_col24, h
CIA> raster_pds.npix = h

CIA> raster_pds.rms(0,*,*) = raster_pds.rms(0,*,*)*10
CIA> raster_pds.rms(31,*,*) = raster_pds.rms(31,*,*)*10
CIA> raster_pds.rms(1:30,0,*) = raster_pds.rms(1:30,0,*)*10
CIA> raster_pds.rms(1:30,31,*) = raster_pds.rms(1:30,31,*)*10
...
CIA> raster_pds.rms(2,3:28,*) = raster_pds.rms(2,3:28,*)*2
CIA> raster_pds.rms(29,3:28,*) = raster_pds.rms(29,3:28,*)*2

CIA> raster_scan, raster_pds, /weight
```

- Projection using the weighted mean, CCIM.RMS contains only an estimate of the flat-field error, no pixel variations are taken into account to create the middle MOSAIC of Figure 20.4:

```
CIA> raster_pds.rms = 1.

CIA> raster_pds.rms(0,*,*) = raster_pds.rms(0,*,*)*10
CIA> raster_pds.rms(31,*,*) = raster_pds.rms(31,*,*)*10
CIA> raster_pds.rms(1:30,0,*) = raster_pds.rms(1:30,0,*)*10
CIA> raster_pds.rms(1:30,31,*) = raster_pds.rms(1:30,31,*)*10
...
CIA> raster_pds.rms(2,3:28,*) = raster_pds.rms(2,3:28,*)*2
CIA> raster_pds.rms(29,3:28,*) = raster_pds.rms(29,3:28,*)*2

CIA> raster_scan, raster_pds, /weight
```

The resulting MOSAICs are shown below. The left mosaic demonstrates clearly the negative effects of an equal weighting of good and badly flat-fielded pixels. The middle mosaic gives optically the nicest impression. However, flux is lost as source pixels showing high flux (and therefore having a high RMS) are averaged with background pixels with low flux (and therefore a low RMS), which consequently get a higher weight. This mix-up of pixel positions is due to the imperfect astrometry (distortion correction and ISO pointing). The right mosaic avoids a flux loss, as a draw-back the low-illuminated and therefore unstabilized pixels in the upper right corner of each EXPOSURE get the same weight as stabilized pixel, and lead to artifacts in the MOSAIC.

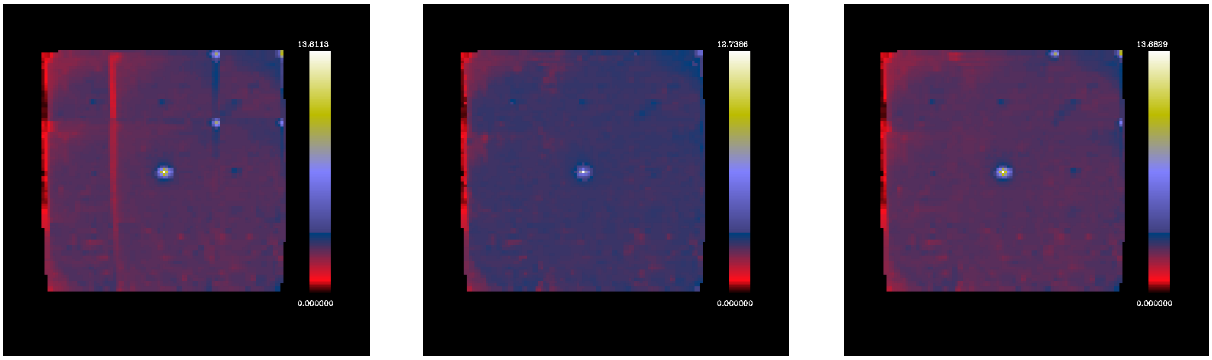

Figure 20.4: Comparison of standard projection vs. weighted mean projection. Left figure: standard projection; middle figure: weighted mean projection, using a combination of pixel variation and flat-field error; right figure: weighted mean projection, using only the flat-field error.

20.14.3 Coadding images of different astrometry

A very useful function of the projection routines are their ability to combine or coadd images that have different astrometry. This can be used to combine raster MOSAICs from different observations of the same object. To take advantage of this functionality we need to go quite low-level and use the C++ executable *projection*.

Here is an example on how to combine raster MOSAICs from different observations of the same object. Note that the same could be done for MOSAICs from different AOTs. For example, a raster MOSAIC and a beam-switch MOSAIC. Also, more than two MOSAICs can be used.

- Save both raster MOSAICs as FITS files, giving them each a name with an individual sequence number. We use **conv_flux** to make sure the data are calibrated to the same units.

```
CIA> conv_flux, raster_pds1

CIA> conv_flux, raster_pds2

CIA> raster2fits, raster_pds1, name='input1'

CIA> raster2fits, raster_pds2, name='input2'

CIA> $ls input*.fits
input1.fits  input2.fits
```

- Project the images contained in the FITS files and place the results in the file *result.fits*:

```
CIA> spawn, !cia_exec+'/projection -i2,input -o result.fits'
```

- Take a look at the result:

```
CIA> result = readfits('result.fits', hdr)

CIA> tviso, result[*, *, 0]
```

20.14.4 Back projection

The projection routines also allow the possibility to back project a raster MOSAIC to a set of EXPOSUREs. A real source will, due to the many effects present in the ISOCAM data, have a somewhat different signal in each EXPOSURE. The purpose of the data calibration is to minimize or eliminate this difference. Knowledge of how the signal varies from EXPOSURE to EXPOSURE is not contained in the raster MOSAIC, so the back-projected EXPOSUREs will each contain an averaged source signal. Figures 20.5 and 20.6 illustrate this point. Note that this averaged source signal is really an idealized signal, i.e. it assumes that the same source has the same signal in each EXPOSURE. This assumption can be useful for testing the quality of the data analysis. The less the signal in the original EXPOSUREs deviates from the idealized signal in the back-projected EXPOSUREs the better the quality of the data calibration.

The back projection is performed with the routine **back_project**. You should of course only perform back projection on a fully calibrated PDS after creation of the raster MOSAIC:

```
CIA> raster_scan, raster_pds

CIA> help, raster_pds.image
<Expression>    FLOAT     = Array[32, 32, 8]
```

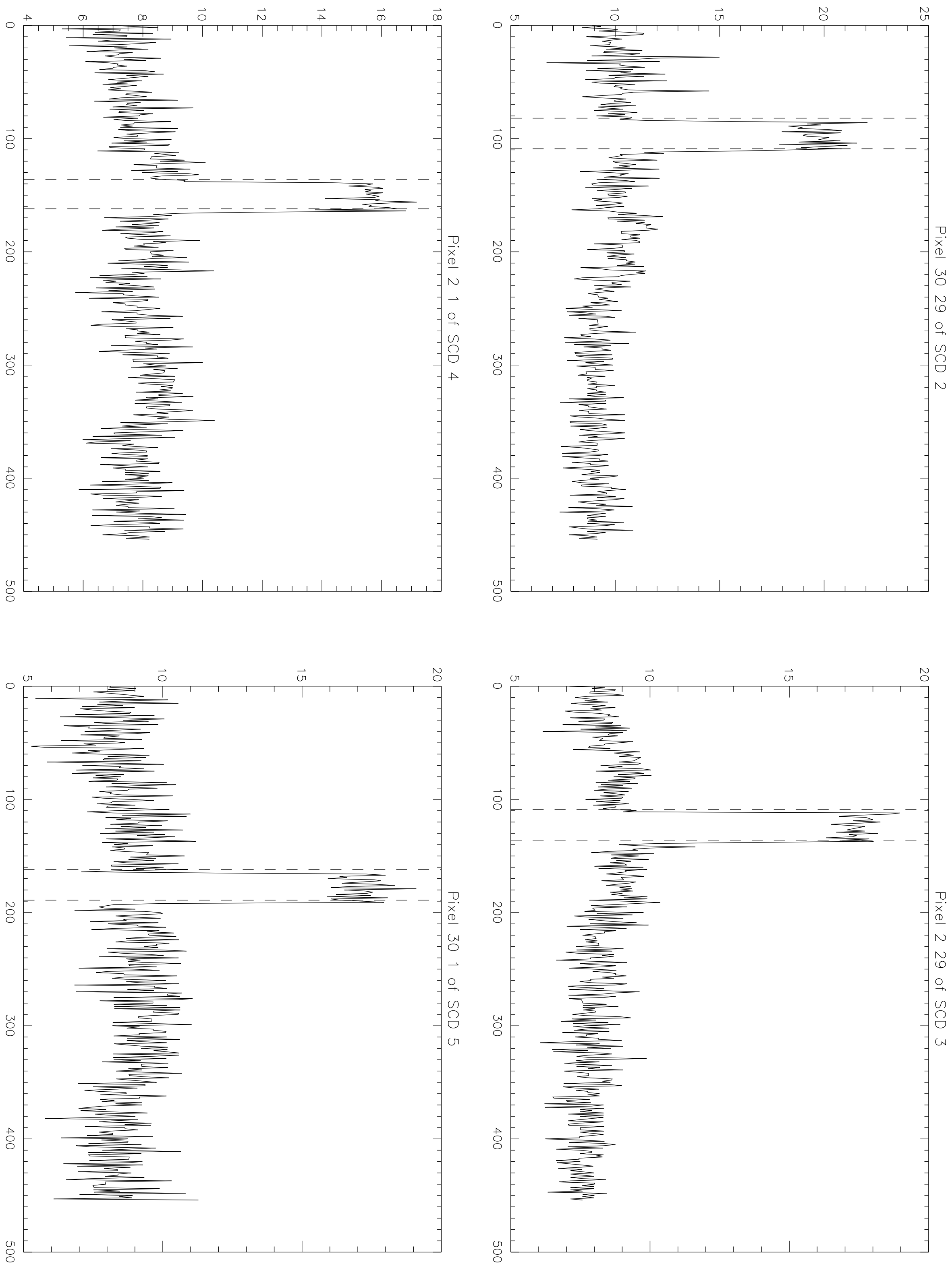

Figure 20.5: Original pixel histories of the same source. Note the source has different signal levels in different SCDs. After reduction, this will propagate into different signal levels in the corresponding EXPOSUREs.

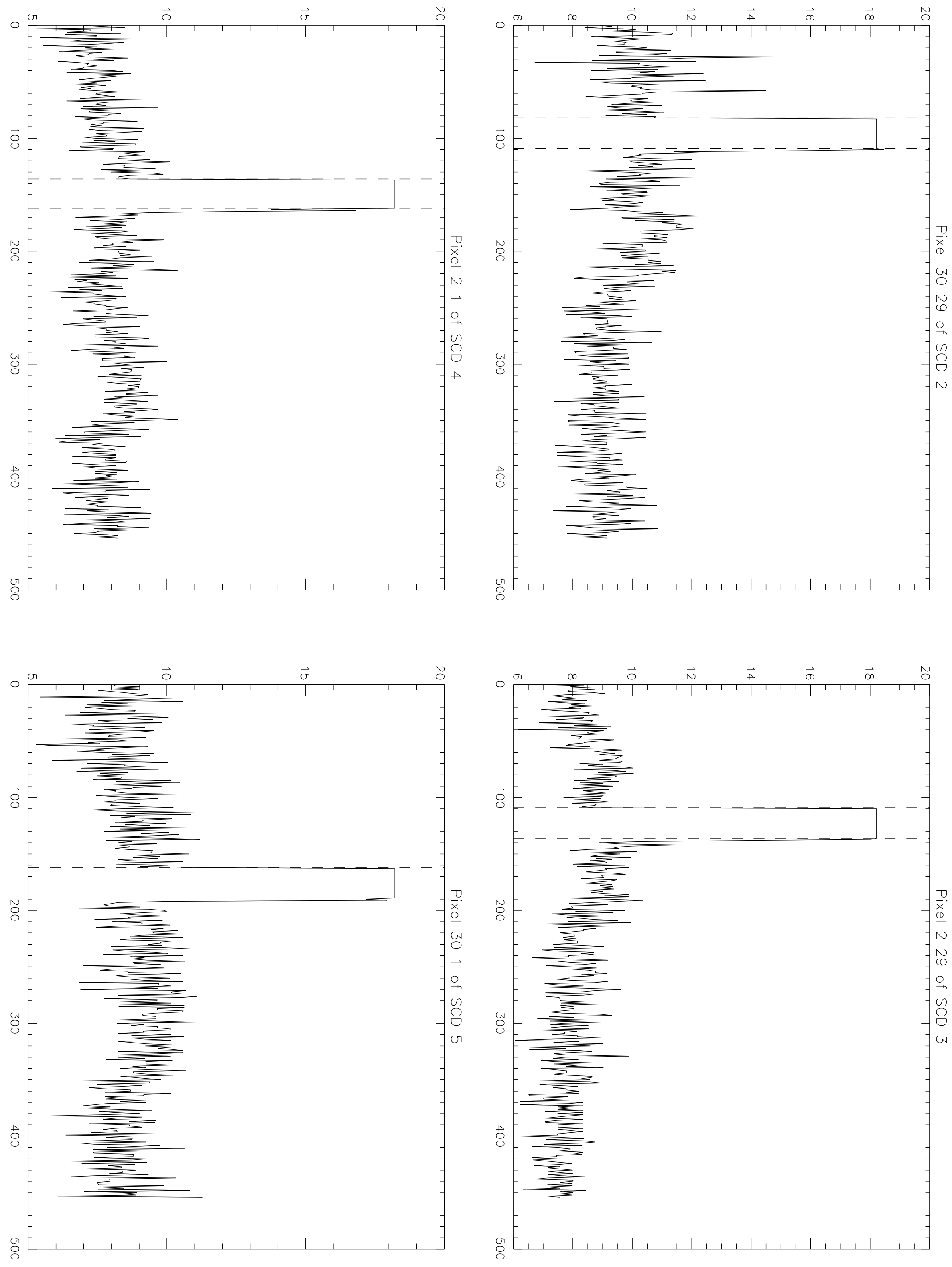

Figure 20.6: Back projected pixel histories of the same source. Note how the signal is of the same averaged or idealized level.

```
CIA> back_project, raster_pds, images

CIA> help, images
IMAGES          FLOAT     = Array[32, 32, 8]
```

You can use **stat** to check the quality of your calibration. The lower the RMS the less the averaged or idealized signal deviates from the real calibrated and corrected signal:

```
CIA> stat, images - raster_pds.image
-------------------------------------------
Image dimensions:           32          32
Number of frames:            8
Total number of pixels:         8192
-------------------------------------------
     Minimum       Maximum       Mean         Median         RMS
     -10.7506      14.4660      0.144570   0.00626230      0.902133
```

20.15 Error handling in CIA

In this section we discuss the error calculated by CIA during the calibration and data reduction stages. We use the standard definitions:

$$\text{Mean} = \bar{x} = \frac{1}{N}\sum_{j=0}^{N-1} x_j \tag{20.2}$$

$$\text{Variance} = \sigma^2 = \frac{1}{N-1}\sum_{j=0}^{N-1} (x_j - \bar{x})^2 \tag{20.3}$$

$$\text{Standard Deviation} = \sigma = \sqrt{Variance} \tag{20.4}$$

$$\text{Standard Error} = \frac{\sigma}{\sqrt{N-1}} \tag{20.5}$$

The CIA routine **reduce** (see also Section 20.2.4) not only averages the IMAGEs to EXPOSUREs, it also creates corresponding RMS images and weight images. Each pixel or element of the weight image contains the total number of IMAGE pixels that have been averaged to the EXPOSURE pixel. Each pixel in the RMS image contains the standard deviation of this sample of IMAGE pixels. To summarize using some CIA pseudo code[2]:

```
raster_pds.image[i, j, k] = $
average( raster_pds.cube[i, j, raster_pds.from[k]:raster_pds.to[k]] )

raster_pds.npix[i, j, k] = $
total( raster_pds.cube[i, j, raster_pds.from[k]:raster_pds.to[k]] )

raster_pds.rms[i, j, k] = $
stdev( raster_pds.cube[i, j, raster_pds.from[k]:raster_pds.to[k]] )
```

[2]Note however that this is for purposes of illustration only; unlike **reduce** we do not take account of unusable pixels in the MASK.

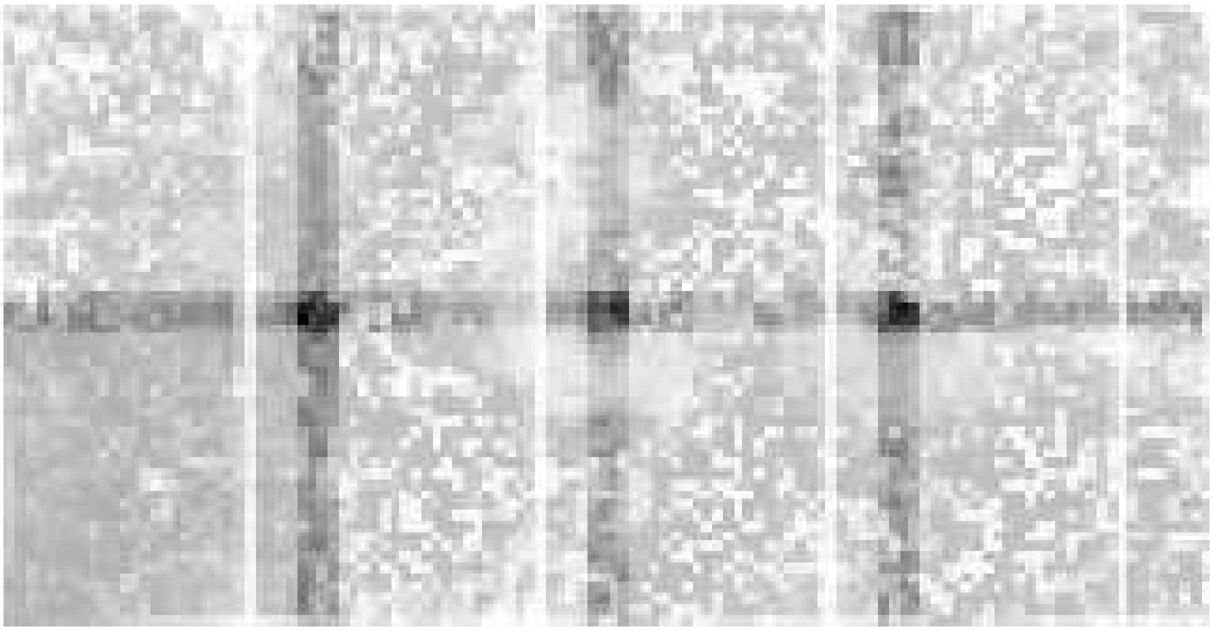

Figure 20.7: The RMS image that correspond to Figure 3.3

Similarly, the routine **raster_scan** (see also Section 20.4) not only creates the raster MOSAIC image, but also a corresponding RMS image and weight image. In this case, each RMS pixel contains the standard deviation of *all* IMAGE pixels that sample the same sky pixel as the MOSAIC pixel. Again in CIA pseudo code:

```
raster_pds.npixraster[i, j, k] = total( raster_pds.cube[i, j, *] )

raster_pds.rmsraster[i, j, k] = stdev( raster_pds.cube[i, j, *] )
```

The RMS and weight images that correspond to Figure 3.3 are given in Figures 20.8 and 20.7. These figures were generated with the commands:

```
CIA> tviso, raster_pds.npixraster

CIA> tviso, raster_pds.rmsraster
```

To obtain the standard error of the MOSAIC image (a measure of the quality of the calculation of the mean, in this case, the quality of calculated MOSAIC pixel values) we simply divide the MOSAIC image by the square root of the MOSAIC weight image.

```
CIA> raster_std_err = raster_pds.rmsraster / sqrt( raster_pds.npixraster )
```

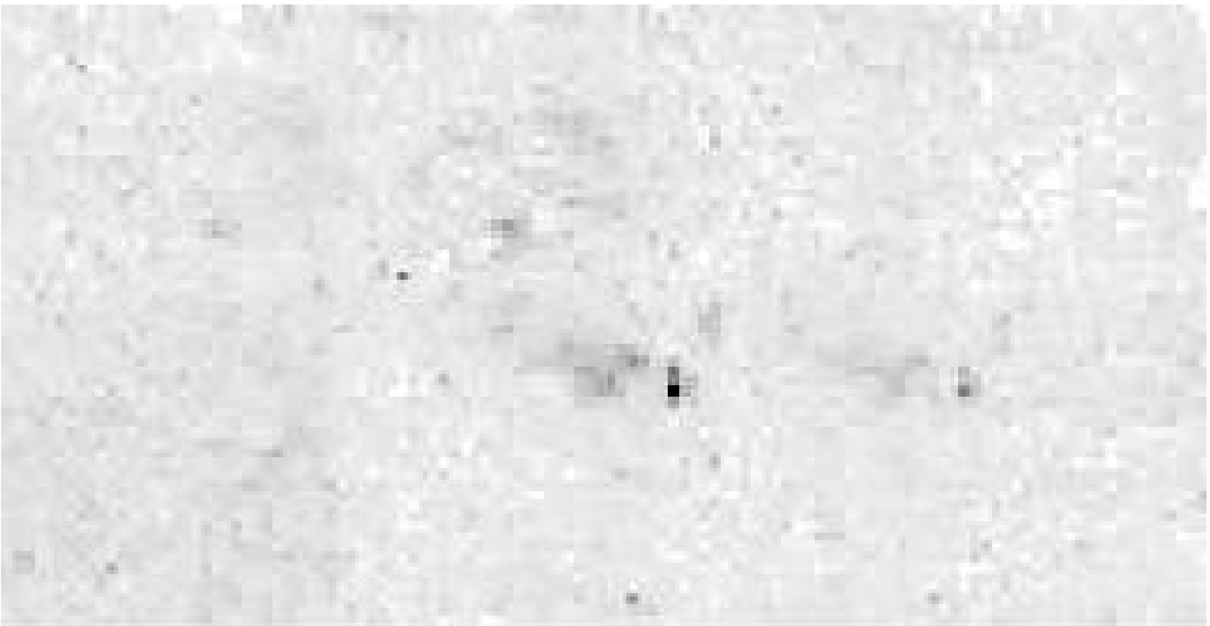

Figure 20.8: The weight image that correspond to Figure 3.3

20.16 How to save spoilt observations

The data of some observations are spoilt and therefore resists standard CIA data reduction. The most common reasons are:

1. telemetry drops
2. bad raster point IDs
3. target not acquired
4. bad QLA flag
5. bad CSH flag
6. strong saturation

CIA provides the functionality to recover some of these observations. However, only experienced CIA users should try these steps!

telemetry drops Short telemetry drops, affecting only a few readouts, will normally go unnoticed. For longer telemetry drops, were the data for one or more SCDs is missing, the following actions can be taken:

- raster observations: Calling **get_sscdraster** with the option */nocheck* will permit you to continue with the data reduction.

  ```
  CIA> raster_pds = get_sscdraster(sscd, /nocheck)
  ```

  Alternatively, **sscd_clean** will fake empty SCDs so that **get_sscdraster** is tricked into believing the raster is complete.

  ```
  CIA> cleaned_sscd = sscdclean(sscd)
  CIA> raster_pds = get_sscdraster(cleaned_sscd, /nocheck)
  ```

- CVF observations: CIA data reduction is not affected by telemetry drops.
- Beam-Switch observations: With most beam-switch observations consisting only of one on and one off-position, the observation is quite likely lost. If there are more on/off positions, use **get_sscdstruct** on the "cleaned" dataset, e.g. a dataset containing no spurious SCDs any more, and combine the results of on/off positions manually.
- Polarization observations: It will depend on the redundancy of the observation, whether the observation is lost. If there are sufficient observations with all polarizors, use **get_sscdstruct** on the "cleaned" dataset, and combine the results manually.

bad raster point IDs Some observations are affected by an invalid [0,0] raster-point ID. **sscd_clean** will remove these SCDs so you can progress with your data reduction. However, the final mosaic will contain holes. It is hoped that in a future OLP version these invalid raster points IDs can be recovered in the PMA data products.

target not acquired If the ISO satellite did not acquire the target, but had a stable pointing, then you can try to recover the data using the *no_qla_flag* option of **get_sscdstruct**.

```
CIA> struct = get_sscdstruct(sscd, no_qla_flag = 1)
```

This will declare all data as good, so care has to be taken to exclude slews by manually setting .PDS.MASK.

bad QLA flag Either as a result of a failure to uplink commands for ISOCAM, or due to a faulty programming of the command sequence, the QLA flag, normally indicating non-stabilized data, remains bad. You can try to recover the data by manually discarding bad SCDs (see Section 12.2.1) and consequently using the *no_qla_flag* option of **get_sscdstruct**.

```
CIA> spdtoscd, 'cispxxxxxxxx.fits', sscd, /nomode
CIA> sscd_info, sscd
CIA> scd_del, 'CSCD143006010105_96080110071423'
CIA> struct = get_sscdstruct(sscd, no_qla_flag = 2)
```

bad CSH flag The CSH flag will remain bad either due to a failure to uplink commands for ISOCAM (and ISOCAM consequently did not reach its commanded position), or, more likely, to an incorrectly generated CSTA file.[3] You can try to recover the data by using the *no_csh_flag* option of **get_sscdstruct**.

```
CIA> spdtoscd, 'cispxxxxxxxx.fits', sscd
CIA> cleaned_sscd = sscd_clean(sscd)
CIA> struct = get_sscdstruct(cleaned_sscd, /no_csh_flag)
```

strong saturation If **spdtoscd** warns you about saturation events (see Section 19.3) the data has to be inspected carefully to assess the impact on photometry. If it is only a short and mild saturation, affecting only one pixel, the data might still be usable. In case of strong saturations, affecting several pixels, the observation is lost.

[3] It is expected that with OLP 9 the CSTA file is produced correctly.

Chapter 21

Using SLICE within CIA

This chapter[1] explains how to use the long term transient correction (or LTT) and variable flat-field correction (or VFF) algorithms implemented in **SLICE**, which can significantly improve raster data reduction.

21.1 Preface

For the examples in Section 21.3 and 21.4, we assume that you are reducing a raster observation and that the data are currently stored in a PDS called `data`. Section 21.5 presents a worked example in more depth.

Warning: Figures 21.1, 21.2, 21.3 and 21.6 are best viewed in color. If you don't want to print the relevant pages of this manual, you can print the last pages of the original document *An Introduction to* **SLICE** *inside CIA*.

21.2 A brief description

What is **SLICE**? It stands for Simple and Light ISOCAM Calibration Environment and was developed by M. A. Miville-Deschênes (mamd@ias.fr) from the original ground-base calibration data reduction package ICE. Because it is Light, it is not as complete as CIA but is rather restricted to the raster data reduction. It contains two specific tools that are not present in CIA and that can allow significant improvements on the observation quality, when dealing with rasters: long term transient correction (or LTT) and variable flat-field correction. Rather than duplicating these tools in CIA, it was chosen to provide access to **SLICE** in CIA. As integration of two software packages into one can prove difficult, the version of **SLICE** you can access in CIA is frozen with respect to the version developed by M. A. Miville-Deschênes. Through him you can probably obtain a more recent version of **SLICE** but we cannot guarantee its compatibility with CIA.

A detailed description of the algorithms in **SLICE** can be found in M. A. Miville-Deschênes' paper (2000, A&A, in prep), while the package has it own user's manual available in the `doc` directory of the **SLICE** delivery.

[1]Taken from Sauvage M., 2000, *An Introduction to* **SLICE** *inside CIA*

21.3 Organization of data in SLICE

In CIA, data are stored in purpose designed data structures, either the SCD, SSCD, SAD, SSAD, (see Section 15.2) that you access through interface routines, or in standard IDL structures (the so-called PDS, see Section 15.5) that you manipulate directly.

In **SLICE** this is different: data are stored in IDL variables (i.e. you can manipulate them directly) but these also belong to common blocks that are accessed by **SLICE** routines. As a result, their names are defined once and for all, and you cannot use them for other purposes. This is why **SLICE** is not loaded by default when you start CIA.

Although you can do it in **SLICE**, we assume here that dark correction, de-glitching and transient correction are performed in CIA. You will generally start the **SLICE** processing after these three steps, i.e. where, in a more standard reduction session, you would have made the flat-field correction.

To start **SLICE**, the first step is to initialize its common blocks as well as place its directories in your IDL path. This is simply done with:

```
CIA> @cia_slice_init
```

This creates the following variables that will be used to store your data, **any variable with the same name is therefore erased**.

Names	Content
`c`	the data cube, i.e. all readouts of the raster
`map0`	initial version of the raster map
`map`	the raster map, as created by the current flat-field correction
`map_before`	the raster map as created by the previous flat-field correction
`map1,2,3,4,5`	equivalent to `map`, containing previous map versions
`map_mjy`	the calibrated map
`h_mjy`	header corresponding to `map_mjy`
`redun`	the redundancy map (coverage factor for sky pixels)
`error_map`	the error map associated to `map`
`obs_param`	a structure containing configuration parameters
`act`	the structure describing the actions to perform
`red_param`	a structure holding the parameters of the routines
`im_param`	a structure describing the current observation
`raster`	a structure containing the raster data (**SLICE** flavor of the PDS)
`correction`	the exact long term transient correction
`corrfit`	the fitted long term transient correction
`error`	error associated to the long term transient correction

You need now to transfer your data into these variables. This is done with the CIA routine `raster2slice`, therefore type:

```
CIA> raster2slice,data
```

The next section will describe the processing that you can do in **SLICE**. For completeness reasons, we will also show you how to transfer data back from **SLICE** into a CIA structure. This is also very simple as you just have to type:

```
CIA> new_raster = slice2raster()
```

This function requires no argument as all the required data are already in the **SLICE** commons. You will see that the content of the output raster structure is different from that of the input one. In particular, the raster field that contains the final map is generally larger. This is due to the **SLICE** convention of only building maps with North up and East left (the astronomical convention). This in general results in larger raster maps. Note however that this new raster structure is fully compatible with CIA. In particular it can be examined with `isocont`.

21.4 Processing in SLICE

Here again, **SLICE** differs from CIA. In **SLICE**, you do not manipulate directly the routines that perform the actions, rather you describe with a set of structures the actions you want to perform and then tell **SLICE** to do them. This philosophy is based on the necessity to perform the data reduction steps in the correct order (and since there are many iterative processes involved, this is very important) and on the desire to be able to "pipeline" the data processing easily.

To describe the action (i.e. data reduction steps) you will perform, you have access to two structures: the `red_param` structure contains the parameters of the data reduction routines that will be involved in the processing, and the `act` structure will contain the actions to be perform (basically the `act` structure is a structure of boolean keywords, one for each data processing step).

Once again, it is not our intention here to supersede **SLICE**'s manual. We highly recommend that before you use **SLICE**, you read its user's manual as well as M. A. Miville-Deschênes' paper to familiarize yourself with the concepts involved.

In this section we will only show an example of how to perform a perturbated flat-field correction on your raster data to illustrate the main feature of **SLICE**'s syntax (Section 21.5 provides a more detailed example). The principle of this flat-field derivation is that to first order, the flat-field is assumed to be constant over the whole raster and that temporal variations are treated as small perturbations to this flat-field. It is also assumed that for a given readout, departures from this single flat-field are dominated by high frequencies. To get them, a smoothed version of the data cube is made which is subtracted to the original data. One then derives the perturbation of the flat-field from a sliding mean on the modified cube.

The parameters for this flat-fielding method are:

`flat_smooth_window`	The window size used to smooth each readout
`nplanes`	the number of readouts used in the sliding mean
`flat_thresh`	the percentage of data discarded from the flat field computation, both in the top and bottom part of the distribution

`flat_smooth_window` is generally a small number as it is applied to 32×32 images and the window size is actually 2×`flat_smooth_window`+1, `nplanes` need not be larger than the number of exposure per raster point as most of the signal has been removed, finally `flat_thresh` larger than 50% makes no sense.

Before starting the operation, one last piece of information must be known, the TDT of your observation. Although this is in principle not necessary when using **SLICE** in CIA, modifying this would mean modifying **SLICE**, which we do not want to do. This information is quite easy to get as it resides in your raster structure, in the field `data.tdtosn`. Note however that it is an integer in the raster structure and that **SLICE** requires a string. Note also that for revolutions smaller than 100, you should add the leading 0 to that string or **SLICE** will not find your data. In our example, observation 45 taken on revolution 83, we would for instance have:

```
CIA> tdt = string(data.tdtosn, format='(i8.8)')
```

Therefore, to perform the **SLICE** processing we now have to first set the reduction parameters structure with:

```
CIA> red_param = set_red_param(tdt=tdt,flat_smooth_window=6,nplanes=30,$
CIA> flat_thresh=10)
```

Two things are worth mentioning here. First it is mandatory that the result of the `set-_red_param` function goes into the variable `red_param` as this is how **SLICE** will access the parameters. If you use another name, the **SLICE** structure will not be updated correctly. Second, you can see that the name flat-field method appears nowhere on the command lines. This is because the method is entirely determined by the set of parameters used. Thus make sure you have read the manual and understand the signification of the parameters before you start any processing.

Now that the parameters are set, you need to specify the action to perform, for flat-fielding, this is called `make_map` so, in a similar fashion, you will update the action structure:

```
CIA> act = set_act(/make_map)
```

Here again, it is absolutely mandatory to store the result of the `set_act` go into the variable called `act`.

So far, we have only updated the structures that describe the actions to perform, to actually start the processing, type:

```
CIA> slice_pipe
```

At the end of this processing, the resulting map can be found in the field `map` and is displayed by **SLICE**'s visualizing tool. If you wish to recover the flat-cube used in the flat fielding, start the slice pipeline with:

```
CIA> slice_pipe, flat_out=flat_out
```

21.5 A worked example

This section is intended to lead you through a classical use of **SLICE** on a raster observation. We once again recommend that you read M. A. Miville-Deschênes' paper before using **SLICE**.

21.5.1 The data set

The dataset we are using here is a 5×5 Y-axis raster on the irregular galaxy NGC 2366. It is performed in LW3 followed by LW2. Table 21.1 provides more details on the observing setup. These are raster with non-integer step-sizes (12.5 pixels in all direction). This is not a problem for **SLICE** which does exact projection back and forth from sky to detector. The important point is that the raster steps are less than half the array which ensures high redundancy. In principle, **SLICE** does not need a very high level of redundancy, but you will be better off with high numbers (typically when the median overlap factor is higher than 3-4)

Figure 21.1 shows the raster maps obtained with a standard CIA procedure: dark correction using the dark model, deglitching with the multiresolution method, transient correction with

Table 21.1: Observing setup for the NGC 2366 data

Filter	N_{stab}	N_{exp}	N	M	ΔN $''$	ΔM $''$	T_{int} s	PFOV $''.pix^{-1}$
LW3	28	24	5	5	75.0	75.0	2.1	6
LW2	14	12	5	5	75.0	75.0	5.04	6

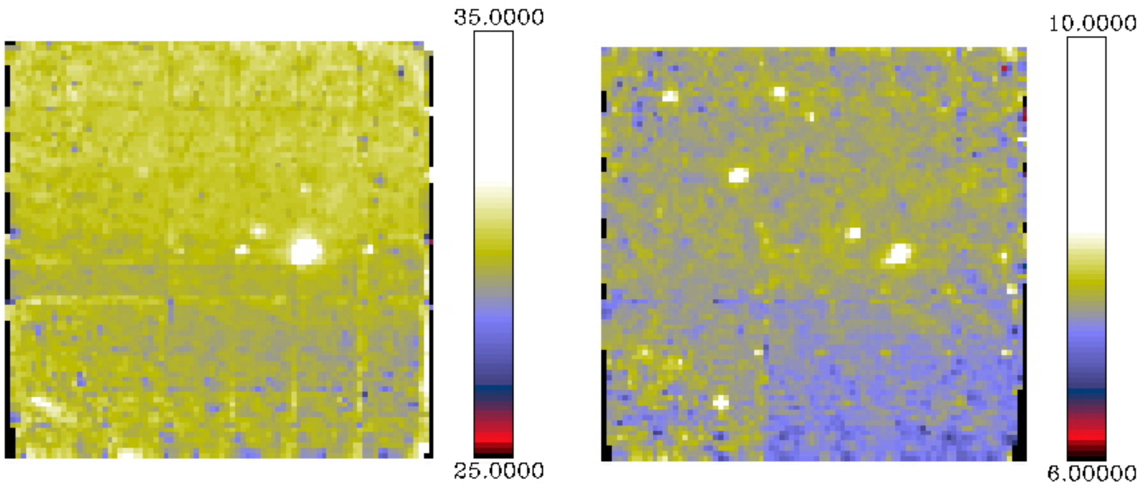

Figure 21.1: The raster maps using a standard CIA data reduction (see text for details). Left panel shows the LW3 data, while the right panel shows the LW2 data. Both data sets are affected by periodic patterns due to bad flat-field determination, as well as long term transients.

the Fouks-Schubert model, and flat-field correction using the "auto" method, i.e. the flat-field is derived from the data.

As can be seen clearly on the figure, we are suffering from periodic patterns which are due to the use of a single flat for the whole raster, and also from long term transients, both in LW3 and LW2. One can also see that due to the strong flux decrease between the LW3 and LW2 raster, we first have a decreasing, then an increasing long term transient in the LW2 raster.

At that stage, we are ready to use **SLICE**. An important point to understand is that the variable flat-field and the long-term transient are not independent effects: one must have no flat-field residuals in order to accurately determine the long-term transient, and one would need ideally a long-term transient corrected data set to pin-point variations due to the flat-field. Since determining both simultaneously is not yet possible, we are going to use an iterative process. In most cases, very few iterations are necessary. This is the outline of the process:

- We experiment with the flat-field methods to define the best choice of parameters per method.
- With that choice of parameters/method, we estimate the long-term transient component and subtract it from the cube.
- In that new cube, we make a more refined correction of the flat-field.

In general, this is enough to get a significant improvement of the data quality. However, if needed one can loop on the long-term transient, variable flat-field correction. In fact, as you will see later (see Section 21.5.3), the first step of this process is really a way to determine which flat-field method to use in the second step.

For the rest of this section, I will assume that the LW3 and LW2 data are stored in CIA raster structures (PDS) called `lw3` and `lw2`. Remember that to work with **SLICE**, it needs to be initialized (once) with the command:

```
CIA> @cia_slice_init
```

and that to transfer a raster structure to **SLICE**, you have to type

```
CIA> raster2slice,lw3
```

and respectively `lw2` for the LW2 data structure. **SLICE** cannot handle more than one structure so you will need to perform all the processing raster per raster, although for simplicity's sake, we are describing them in parallel in the following sections.

Once all the processing is done, transfer your data back from **SLICE** to CIA with:

```
CIA> new_lw3 = raster2slice()
```

if you were working on the LW3 data, modify accordingly for the LW2 data.

Important warning: It may happen that even though you have corrected for short-term transient, there is still a transient artefact at the start of the raster. In that case, it is better to completely mask the stabilization frames at the start of the raster. If this is not done, then the flat-field and long-term transient determination may fail (mostly because the sliding means cannot work on the very first and last frames by construction). This is what happens here for the LW2 case and thus we have masked the first 14 frames of the LW2 raster (see Table 21.1).

21.5.2 Choice of flat-field methods/parameters

For this first flat-field determination, we are working with data that are potentially still heavily affected by a long-term transient, we must thus use a method which is not too affected by that. **SLICE** has 4 different methods to build the flat-field. We discard the first one which is to use a single flat for all the data, as this is equivalent to the "auto" method of CIA. The three other methods are:

- **Sliding Mean flat-field**: where the flat-field is derived from the mean of the data cube taken on a sliding window of readouts. This is satisfactory if your sources are not very strong, but if this is not the case, or if you have large scale structures, you run the risk of seeing some of these structures go into the flat-field, which is not advisable. You could protect yourself from that using a large sliding window, typically around 10 time N_{exp} but this is not always possible (i.e. when you have a small number of raster positions).

- **Perturbated Single Flat-Field**: although this does not sound much better, this is actually an improved flat-field method. Here, using a median filter on the individual raster pointing, the object structures are removed before an "auto"-like flat is computed. Then, still using a sliding window over the readouts, perturbations to this single flat-field are computed. The interesting point here is that since the object structures have been removed, the size of the window can be smaller than in the previous case.

- **Variable flat with Sky Divided (or DivSky)**: this is the ideal method where an estimation of the sky is divided out of the images before the flat-field is computed. This is however the method which can be affected by the long-term transient or affect its determination, and its use should be restricted to the second flat-field determination.

You will rapidly see that in general, as far as flat-fielding quality goes, the **DivSky** method produces apparently the best results (compare, for instance, Figure 21.2 with Figure 21.3). However, it is also the method whose interaction with the long-term transient correction is the most complex, and which can lead to strong artefacts. Therefore, in this section we describe them both, but in Section 21.5.3 we will use the **Perturbated Single Flat-Field** method to determine the long-term transient correction.

Let us begin with the **Perturbated Single Flat-Field** method. One should note that this is really to allow determination of the long-term transient. Since a subtraction is done, this is not strictly speaking a flat-field correction and should not be used as such.

This method requires 4 parameters to be set with `set_red_param`: `tdt`, `flat_smooth_window`, `flat_thresh`, and `nplanes`.

- `tdt`: this is ISO-specific number which is located in the `tdtosn` field of your raster data structure. How to extract this information is explained in Section 21.4. In our example, its value is 65801627, meaning the observation number 27 of that proposal, executed on revolution 658 as the 16^{th} observation of the revolution.

- `flat_smooth_window`: this is a number of pixels used to define a smoothing window for the filter that is going to be applied to the individual 32×32 images to remove the large-scale structure before the flat-field construction. Since it is working on 32×32 images, it should be a rather small number, typically of the order or slightly larger than the radius of the PSF. Note that smaller values of this parameter usually translate in small values of the residual noise, but that is some sort of artefact since for small windows, most of the signal is removed...

Table 21.2: Our choice of parameters for the **Perturbated Single Flat-Field** method.

Filter	`tdt`	`flat_smooth_window`	`flat_thresh`	`nplanes`
LW3	65801627	4	10	60
LW2	65801627	2	10	30

- `flat_thresh`: this is a percentage that is used to discard pixels from the flat field computation. The histogram of the pixel readouts is built and we discard the bottom and top `flat_thresh` percent. This is useful when you have strong sources and structures in the field to avoid having them propagate in the flat-field.
- `nplanes`: once the filtered and histogram-selected cube of readout is built, the flat-field cube is built with a running average of `nplanes`. In that case, `nplanes` should be larger than N_{exp} but not as large as if no filtering had occurred.

Now that the definition of the parameters is clear, let's proceed. Table 21.2 lists the parameters setup for both filters. Note that both filters share the same `tdt` number since they were obtained in a single AOT. For this data set, the sources are not very important so the `flat_thresh` value is not critical. As always in **SLICE**, choosing the parameters values requires some tuning, and you'll need more than one try to arrive at a good choice of parameters. The commands line to issue to **SLICE** are thus, in the LW3 case:

```
CIA> red_param=set_red_param(tdt='65801627',flat_thresh=10,n_planes=60,$
CIA> flat_smooth_window=4)
CIA> act=set_act(/make_map)
CIA> slice_pipe
```

The results of this first flat-field determination are displayed on Figure 21.2. At that stage, the improvement from classical processing is not yet striking.

Now let's try the **DivSky** method. It is set with 5 parameters: `tdt`, `flat_thresh`, `nplanes`, `size_filter`, and `divsky`. Of these, only two are new. Let us explain them:

- `divsky`: this is just a boolean keyword, i.e. true/false, set to indicate which method is to be used (if it is not present on the command line as `/divsky`, then the **Sliding Mean Flat-Field** method is used, which is not what we want.
- `size_filter`: this is an integer whose meaning is similar to the `flat_smooth_window` of the previous method. Here also the data are filtered and removed from the readouts before computing the flat-field. However, it is the sky image that is smoothed and then divided out. Thus here its value should be of the order of the size of the structures in your image (typically 10 pixels, but we suggest to use a value which is larger than the raster step). Note that here, the spatial resolution is less important than the true object structure size.

Table 21.3 summarizes our choice of parameters for the **DivSky** method while Figure 21.3 shows the results for the two filter. For this particular method, and in the LW3 case, the command lines are:

```
CIA> red_param=set_red_param(tdt='65801627',flat_thresh=10,n_planes=60,$
CIA> size_filter=16,/divsky)
```

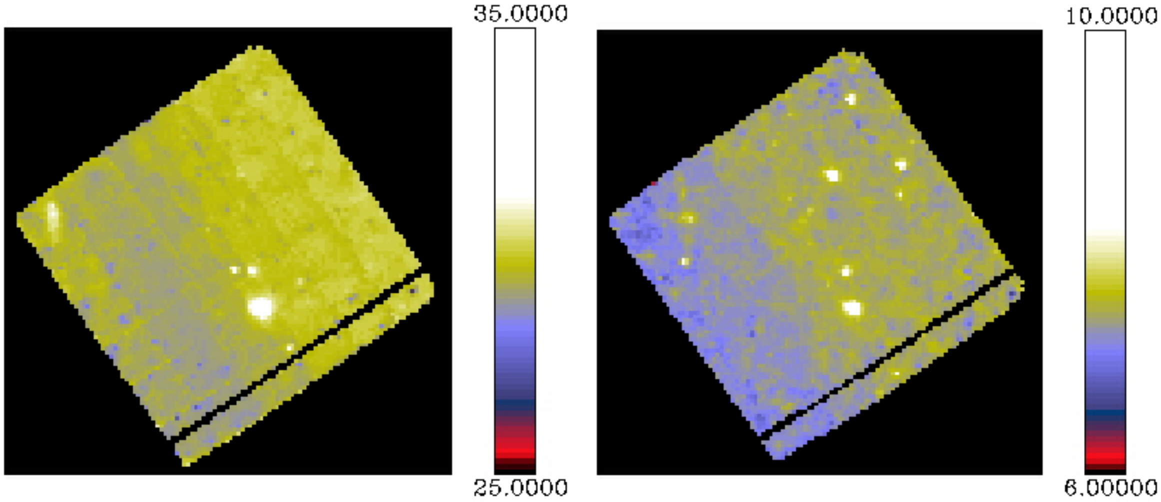

Figure 21.2: The resulting maps for the **Perturbated Single Flat-Field** determination. Note that the map orientation has changed as **SLICE** always produces maps with North up and East left. Imprints of the individual raster pointings are still visible.

Table 21.3: Our choice of parameters for the **DivSky** method.

Filter	tdt	size_filter	flat_thresh	nplanes
LW3	65801627	16	10	60
LW2	65801627	16	10	30

```
CIA> act=set_act(/make_map)
CIA> slice_pipe
```

An obvious improvement is seen here: the gradient of "emission", which is in fact the long-term transient, is much smoother now, and pointing imprints have disappeared. In your data reduction session, we suggest that you play around with both flat-field methods, before you select the one to use in the first pass. It is also important that you check that your choice of flat-field method is compatible with the long-term transient determination (see Section 21.5.3).

21.5.3 Long-term transient determination

Now that we have seen how to derive the flat-field in our data, we are going to try and determine the long-term transient which is assumed to be a global (i.e. pixel independent) additive time drift of the signal.

It is important to understand that the long-term transient determination is made by comparing the data cube to an estimation of the sky. Thus a flat-fielding method is necessary, and, among the parameters to set for this step, we will find again those which have been encountered in the previous section. It is extremely important that a good flat-fielding method is chosen, otherwise you will see very strong artefacts appearing in the long-term transient curves. These artefacts are quite characteristic: they form an oscillating signal with a number of peaks equal

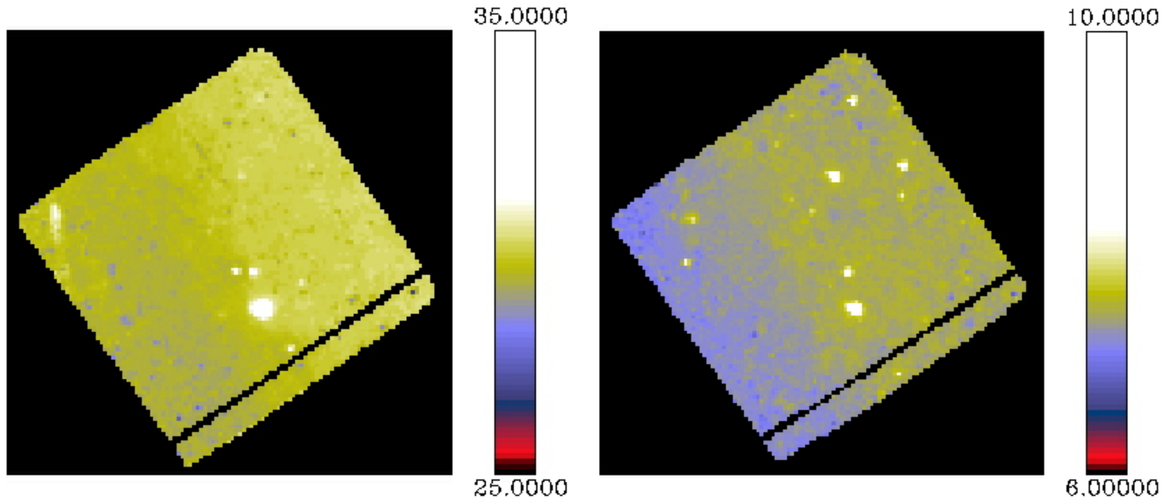

Figure 21.3: The resulting maps for the first flat-field determination with the **DivSky**. Note that the "emission gradient", produced by the long-term transient is much smoother now, and pointing imprints are mostly gone.

to half the number of raster legs. An example of such artefacts is shown in Figure 21.4. If that is your case, try and tune the flat-field parameters better or select another flat-field method (this is why we do that exercise in Section 21.5.2).

In fact, the long-term transient determination adds only one new parameter: `ltt_thresh`. This is similar to the `flat_thresh` parameter, in the sense that it allows to discard pixel readouts from the long-term transient determination. Typically, this is used to discard pixels observing strong sources where the short-term transient correction may not have been ideal. On each raster position, pixels `ltt_thresh`$\times\sigma$ above the mean level of the current sky position are discarded (note that the exact meaning is different from that of `flat_thresh`). In fact, **SLICE** displays the raster positions with the discarded pixels masked. A good way to check that your setting is correct is that the location of strong sources are masked.

In our example, for the LW3 band, and to a lesser extent for the LW2 band as well, the **DivSky** method always produces the oscillating artefacts, and thus we are going to use the **Perturbated Single Flat-Field** method, with parameters set as in Table 21.2. In this observation using `ltt_thresh = 6` is quite satisfactory. In that case, the commands to issue to **SLICE** are, for the LW3 case:

```
CIA> red_param=set_red_param(tdt=65801627,flat_smooth_window=4,nplanes=60,$
CIA> flat_thresh=10,ltt_thresh=6)
CIA> act = set_act(/ltt)
CIA> slice_pipe
```

Figure 21.5 plots the long-term transient corrections that are derived with this setup. On the graph there are two curves, a continuous one and a dashed one. The continuous on is the exact correction, as derived from the data. It will, by definition (see M. A. Miville-Deschênes' paper), end with a zero value, i.e. no long-term transient correction for the last raster position. The dashed one is a fitted correction. Most of the time, the fit and the exact solution apparently disagree. This is because the so-called "fitted-correction" is not a fit to the exact correction,

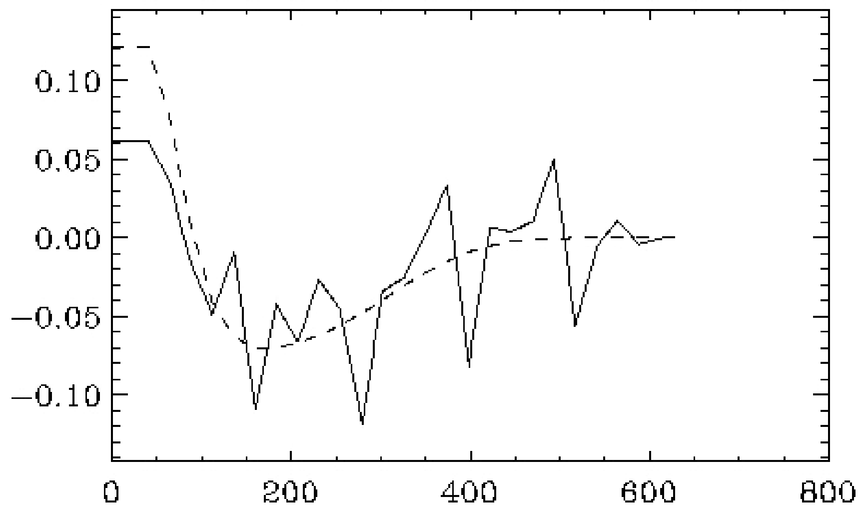

Figure 21.4: An example of artefacts obtained with a incorrect long-term transient determination: the signal oscillates and the number of complete oscillations is roughly equal to half the number of raster legs. The dashed line represent the fitted correction (see text for details). In fact, these artefacts were generated while using the **DivSky** flat-field method in the long-term transient determination for the LW3 image. The LW2 image shows similar problems.

but rather a correction derived assuming the long-term transient is the combination of two exponentials whose parameters are fitted. Therefore, when these two corrections agree in shape, you have a good sign that you have reached the "right" correction (an offset is not important as it will be taken care of by your background subtraction). On the contrary, when they disagree, it is a good indication that you should tune the parameters better.

Once you have reached a satisfactory determination, it can be subtracted from your data using the commands suggested by **SLICE**. Note that in some very complex cases, it may be interesting to subtract the fitted correction first and then iterate flat-field determination and long term transient correction with the exact determination for the remaining residuals.

21.5.4 Second flat-field determination

In a number of cases, this will be the last one, producing your final image. In principle, at this stage the main component of the long-term transient has been removed and it is safe to use the **DivSky** method. This is what we do here, using the parameter values listed in Table 21.3. The results after the long-term transient correction and the new flat-field are displayed in Figure 21.6. A comparison with Figure 21.1 shows the improvement in the image quality. Further improvement can be achieved by looking at deviant pixels, glitch impacts and so on, but for this, we refer the reader to the **SLICE** manual.

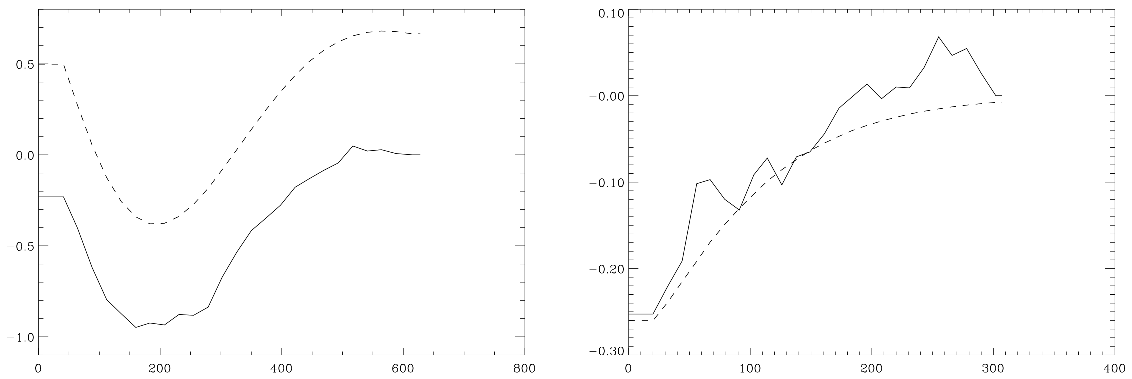

Figure 21.5: The long-term transient corrections derived by **SLICE**. The continuous curves are the exact corrections and the dashed ones the fitted corrections, assuming the long-term transient effect is a combination of two exponentionals. On the left, the LW3 case, and on the right, the LW2 case. Some oscillation appear on the LW2 exact curve, but these are not obviously related to the raster scan period.

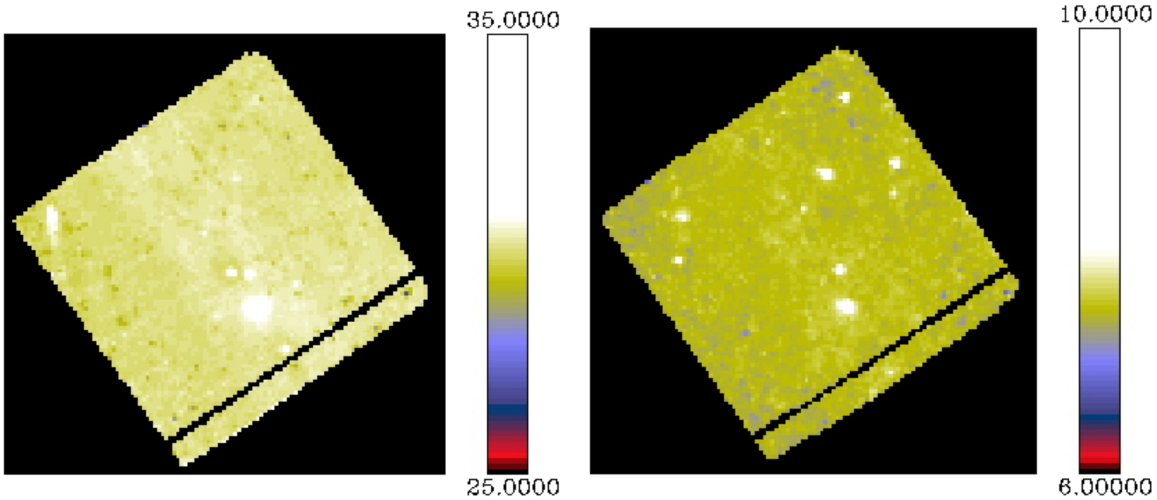

Figure 21.6: The results of the long-term transient correction and variable flat-field determination. Variable flat-field was performed using the **DivSky** method, with parameter setup as indicated in Table 21.3. LW3 is on the left, and LW2 on the right. Compare with Figure 21.1 to measure the improvement

Chapter 22

Second order corrections

This chapter describes second order calibration routines. Depending on your data such calibration maybe entirely unnecessary. All second order correction routines should be used with care.

22.1 Jitter correction

Spacecraft jitter can cause a slight displacement of a source from IMAGE to IMAGE. This displacement is limited to about $0.5''$ (Kessler et al. 1996). Figure 22.1 shows the distribution of jitter offsets for one particular observation. Jitter is most noticeable when a point source is observed. You can check for jitter in your data by using **x3d** to flick through the IMAGEs in .CUBE while watching for sub-pixel shifting of a source.

Jitter correction is performed in two steps. Firstly the jitter offsets from IMAGE to IMAGE are computed and secondly these offsets are applied to the data.

22.1.1 Computing the jitter offsets

In general, the jitter computation is performed as follows:

1. All the IMAGEs corresponding to a single ISO pointing are extracted from the PDS CUBE. These IMAGEs are averaged and the maximum pixel, $[i_{\mathrm{max\_avg}}, j_{\mathrm{max\_avg}}]$ is taken as a first estimate of the position of the center of the source.

2. A sub-cube is made from the IMAGEs, centered on $CUBE[i_{\mathrm{max\_avg}}, j_{\mathrm{max\_avg}}, k]$ and of radius *jitter*, where *jitter* is the expected maximum jitter amplitude and defaults to 1 pixel. For *each* frame of this sub-cube, the maximum pixel, $[i_{\mathrm{max}}, j_{\mathrm{max}}]_k$ is determined.

3. A new sub-cube, $SUBCUBE$ is again made from the IMAGEs, this time *each* sub-cube frame, $SUBCUBE[i, j, k]$, is centered on $CUBE[[i_{\mathrm{max}}]_k, [j_{\mathrm{max}}]_k, k]$ and of radius *bsize*.

4. A fit is applied to *each* sub-cube frame $SUBCUBE[i, j, k]$ and the center of the fitted function is taken as the observed source location, $[i_{\mathrm{source}}, j_{\mathrm{source}}]_k$

5. The jitter offsets in the x and y-axis, du and dv are taken as

   $$du_k = [i_{\mathrm{source}}]_k - median([i_{\mathrm{source}}]_k)$$

   $dv_k = [j_{\mathrm{source}}]_k - median([j_{\mathrm{source}}]_k)$

   du and dv are placed in the PDS fields DU and DV.

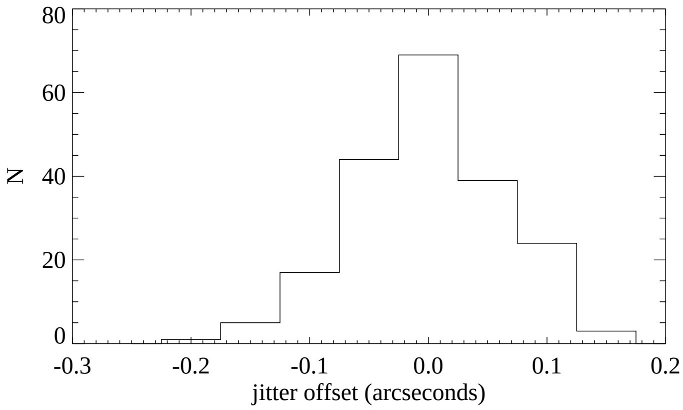

Figure 22.1: Distribution of jitter offsets as computed with the *gauss* method. Data are from a CAM calibration observation of HIC 73005 (also featured in Chapter 4). In this particular case the jitter standard deviation is observed to be about 0.065″.

The routine **corr_jitter** is the user interface to the jitter computation routines. It can operate on any flavor of PDS. It is recommended that the PDS is at least dark corrected and deglitched before attempting to compute the jitter. To perform the default jitter computation, **corr_jitter** can be run as:

```
CIA> corr_jitter, pds
```

You can view the computed jitter offsets with:

```
CIA> plot, pds.du

CIA> plot, pds.dv
```

However, jitter computation is an interactive process and a choice of methods and options are available. The available jitter computation methods are described below. See Figure 22.2 for a comparison of jitter computation methods.

1. `method='gauss'`

   method: Apply a 2D elliptical gaussian fit to each IMAGE. See the on-line help for **fit_2dgauss** for a description of the gaussian function.

   routine called: fit_2dgauss

   PDS side effects: .DU and .DV are filled with the jitter offsets du, dv for each image.

2. `method='psf'`

   method: Select the best fitting theoretical PSF from a library.

   routine called: fit_psfstruc

   PDS side effects: .DU and .DV are filled with the jitter offsets du, dv for each image.

Some useful keyword options are:

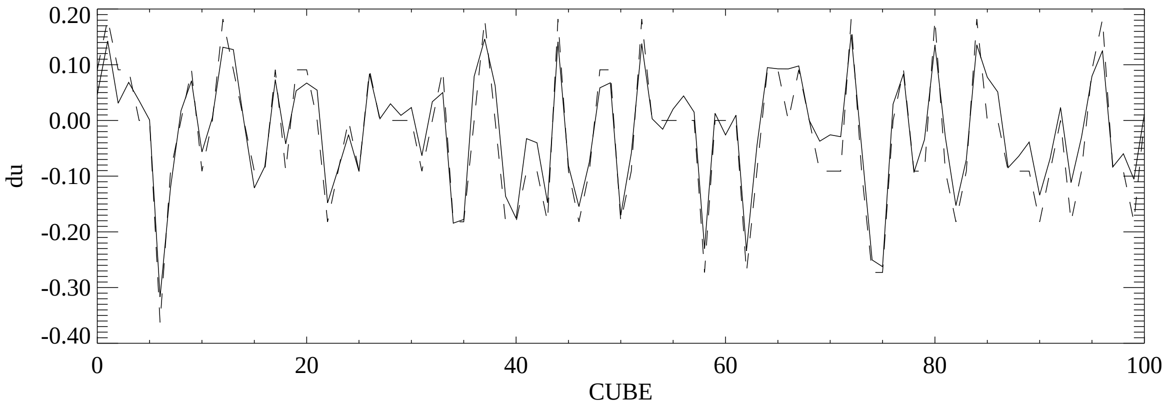

Figure 22.2: Comparison of jitter computation methods. Jitter offsets computed with the *gauss* method are represented by the solid line, and jitter offsets computed with the *psf* method are represented by the dashed line. The data used in this example are from an observation using the 1.5″ PFOV and the LW7 filter.

scd This keyword is used to select all the IMAGEs corresponding to a particular CAM STATE (ISO pointing) from the PDS CUBE. By default all STATEs are selected.

display Setting *display = 1* will display an averaged fit per CAM STATE. Setting *display = 2* will display the each fit of each IMAGE.

verb Setting *verb = 1* will output the averaged fit parameters per CAM STATE. Setting *verb = 2* will output the fit parameters per IMAGE.

method Select fitting method. The default is *method = 'gauss'* .

bsize See description of algorithm above. The default depends on the selected method.

col24 Set to remove column 24 (actually column 23 in IDL convention) from IMAGEs before fitting.

nterms Only applies to *method = 'gauss'* . Select the number of terms used in the gaussian fit.

22.1.2 Applying jitter offsets

The application of the jitter offsets du, dv is the most difficult part of jitter correction. Currently, two methods exist and neither are really effective. These methods are implemented with **reduce** and **project_cube.pro**

reduce We can attempt to correct the jitter by shifting each IMAGE by du, dv before averaging to an EXPOSURE. The limitation of this method is the shifting algorithm. So far all attempts to perform sub-pixel shifting causes some smoothing and this defeats the purpose of removing the jitter. After jitter computation with **compute_jitter** jitter correction with **reduce** can be attempted with

```
CIA> reduce, pds, /jitter

CIA> tviso, pds.image[ *, *, 0 ]
```

project_cube This routine attempts to correct jitter by applying du, dv as an astrometric correction. In the example below, the entire PDS CUBE is projected onto a MOSAIC image. In attempting to make a crude correction of the jitter a magnification factor of 3 is applied to IMAGEs before projection.

```
CIA> project_cube, pds, mosaic, /jitter, mag=3

CIA> tviso, mosaic
```

22.2 Field of view distortion

FOV distortion affects the 12″, 6″ and to a lesser extent the 3″ lens. For *raster* PDSs FOV distortion correction is integrated with the *project* raster MOSAIC creation method – see Section 20.4. However, any CAM image can be corrected for FOV distortion with **corr_field**. In order for this routine to apply the appropriate distortion correction some CAM parameters must be supplied along with the image: pfov, channel and filter wheel.

```
CIA> corr_field, image, fltrwhl='lw10', channel='lw', pfov=3.0
```

corr_field re-projects the image in order to correct the distortion. As a result of this, the output image maybe rebinned. To prevent rebinning set the keyword */norebin*.

Chapter 23

x_cia reference guide

This chapter[1] completes the description, begun in Section 13.3, of **x_cia**. It includes a description of the advanced features of **x_cia** and a useful reference guide to **x_cia**'s commands.

23.1 Advanced use of x_cia

Two advanced features of **x_cia** are described in the following sections.

23.1.1 Executing IDL commands from within x_cia

It is possible to execute directly an IDL command in the *CIA Command Line* window of the **x_cia** screen. You will be prompted in the message window if the command has been correctly executed. Note that the result of the IDL command is visible either in the IDL window from which **x_cia** was launched or in a graphics window. This was mainly designed to display or plot some interesting fields of the PDS structure without quitting **x_cia** session. It can also be useful to modify manually some fields of the PDS structure, such as the coordinates or the mask. See Section 15.5 for more details on the architecture of the PDS.

23.1.2 Buffer variables

Three variables can be used as buffers: DUMMY, TEMP and OLD. At the beginning of the session, they are initialized to zero but can contain any kind of data (including part of the PDS structure). Temporary results can be stored in them, in order for example to plot the difference between the latest result and an older one.

23.2 Help on x_cia

23.2.1 List of commands

Available functions of the main menu are:

- CIA / ISOCAM
- CIA / Log File
- CIA / Help

[1]Taken from Claret A., 1996, *ISOCAM Data Analysis with X_CIA v2.2*, Sections 4 & 6.

- CIA / Info
- CIA / Quit
- Data / Load / SSCD
- Data / Load / IDL File
- Data / Save / SAD
- Data / Save / SAD (Fits)
- Data / Save / IDL File
- Data / Change Sign
- Data / Reload Original Data
- Data / Display History
- Data / AOT Info
- View / Cube / Temporal Mean
- View / Cube / Temporal Analysis
- View / Cube / All Frames
- View / Mask / Temporal Mean
- View / Mask / Temporal Analysis
- View / Mask / All Frames
- View / Image / Averaged Frames
- View / Image / Mean Of All Frames
- View / Cal-G / Dark Current
- View / Cal-G / Flat-field
- View / Cal-G / PSF
- View / Cal-Used / Dark Current
- View / Cal-Used / Flat-field
- View / Result / Reconstructed Raster Map
- View / Result / Monochromatic CVF Frames
- View / Result / All CVF Frames
- View / Ra-Dec-Roll Info
- View / Change LUT

- Dark / None
- Dark / User Input
- Dark / Cal-G
- Dark / Model
- Dark / VilSpa
- Deglitch / None
- Deglitch / Manual
- Deglitch / Particle Impact
- Deglitch / Temporal
- Deglitch / Spatial
- Deglitch / Temporal & Spatial
- Deglitch / Multiresolution Median
- Transient / None
- Transient / 90% Of Final Flux (modelled)
- Transient / 90% Of Final Flux (measured)
- Transient / IPAC Model Fitting
- Transient / IAS Model Fitting
- Transient / SAP Model Fitting
- Transient / Remove Ghosts
- Flat-field / None
- Flat-field / User Input
- Flat-field / Cal-G
- Flatfield / Auto
- Flatfield / Manual
- Process / None
- Process / Default
- Process / Selected
- Tools / xv_temp
- Tools / xv_raster

- Tools / isocont (ISOCAM axis)
- Tools / isocont (RA/DEC axis)
- Tools / x_isocont
- Tools / hardcopy (gif)
- Tools / hardcopy (ps)

Additional corrections of secondary order are:

- Remove Dark Residuals
- Smooth Undefined Values
- FOV Distortion (linear and calibration methods)
- Spectral Deconvolution

Some other buttons concern the type of data to be calibrated (i.e. AOT). Each time a new AOT is selected, a new data set must be loaded. The last buttons concern the projection method for building raster maps, as well as spectrum output units.

23.2.2 Short description of commands

A list of all available commands – together with a short description – is given below.

- **CIA / ISOCAM:** Display a beautiful image to say 'Welcome to the ISOCAM world at CEA-Saclay'.
- **CIA / Log File:** Display the log file of all commands and all warning messages occurring during the **x_cia** session. By default, the log file is named *x_cia_log.txt* and is created in the current directory. A user named file can be created by using the command:

  ```
  CIA> x_cia, logfile='my_file.txt'
  ```

- **CIA / Help:** Display a help file.
- **CIA / Info:** Display some copyright information about the Cam Interactive Analysis (CIA) package.
- **CIA / Quit:** Quit **x_cia**.
- **Data / Load / SSCD:** Load from disk a SSCD data structure. Default input file is the current directory. An history of data is then initialized. Note that an AOT type must be chosen first (default chosen AOT is CAM01, Raster Scan). The date and time of loading file into memory are saved in the data history.
- **Data / Load / IDL File:** Same as above but data are loaded using an IDL restore command. This file must have been saved by *Data / Save / IDL File*. The restored variables are: *history* (string array containing the data history), *ihist* (next line index to be written in history), and *isodata* (PDS structure containing the data). All new processes will be added to the history of the current data. Date and time are of loading file into memory are saved in the data history.

- **Data / Save / SAD:** Save the current SAD.
- **Data / Save / SAD (Fits):** Same as above except that the output format is extended fits.
- **Data / Save / IDL File:** The saved variables are: *history* (string array containing the data history), *ihist* (next line index to be written in history), and *isodata* (IDL structure containing the data). The date and time of saving are written in the data history.
- **Data / Reload Original Data:** Before trying another processing, it may be necessary to reload the original data set (in order to avoid to deglitch it twice for example). Note that history of data is not reset to the original history.
- **Data / Display History:** Display the history of current data. Note that the history is size limited to a 512 string array.
- **Data / AOT Info:** Display useful informations such as the optical path, the detector gain, the integration time.
- **View / Cube / Temporal Mean:** Display the temporal mean of the data cube.
- **View / Cube / Temporal Analysis:** Display one by one all frames of the data cube. The temporal flux variation recorded by a given pixel (mouse selected) is also displayed.
- **View / Cube / All Frames:** Display simultaneously all frames of the data cube.
- **View / Mask / Temporal Mean:** Display the temporal mean of the data mask.
- **View / Mask / Temporal Analysis:** Display one by one all frames of the data mask. The number of times that a pixel is masked can be assessed by plotting the temporal cut of the (mouse selected) pixel.
- **View / Mask / All Frames:** Display simultaneously all frames of the data mask. This can give an idea about the temporal evolution of the "effective" area.
- **View / Image / Averaged Frames:** Display simultaneously the averaged frames corresponding to each configuration of the camera (ISODATA.IMAGE). The numbers of frames displayed along the horizontal and vertical axis correspond to the M and N parameters of the raster map.
- **View / Image / Mean Of All Frames:** Display the mean of the above averaged frames.
- **View / Cal-G / Dark Current:** Display the dark current frame extracted from the calibration database.
- **View / Cal-G / Flatfield:** Display the flat-field extracted from the calibration database.
- **View / Cal-G / PSF:** Display the point spread function extracted from the calibration database.
- **View / Cal-Used / Dark Current:** Display the dark current frame used for the data calibration process.
- **View / Cal-Used / Flatfield:** Display the flat-field used for the data calibration process.

- **View / Result / Reconstructed Raster Map:** Display the reconstructed map of the raster.
- **View / Result / Monochromatic CVF Frames:** Display one by one the averaged monochromatic frames of the CVF scan.
- **View / Result / All CVF Frames:** Display one by one all frames of the CVF scan in order to visualize the transient effects from one CVF wheel position to another.
- **View / Ra-Dec-Roll Info:** Display the Ra, Dec, Roll parameters of each averaged frames corresponding to each configuration of the camera. Ra, Dec, Angle, Rotation and Orientation of the reconstructed raster map are also displayed in the message window.
- **View / Change LUT:** Adjust the color table.
- **Dark / None:** No dark correction is performed.
- **Dark / User Input:** A user dark frame can be given as input in order to subtract the dark current. The user must then use the following command to start the session:

  ```
  CIA> x_cia, indark=my_dark
  ```

  my_dark is a 32×32 array).
- **Dark / Cal-G:** The dark frame is extracted from the calibration database.
- **Dark / Model [default]:** The dark frame is created from the dark model.
- **Deglitch / None:** No deglitch correction is performed.
- **Deglitch / Manual:** Glitches are removed interactively.
- **Deglitch / Particle Impact:** The glitch detection is based on the analysis of cosmic particle impacts.
- **Deglitch / Temporal:** All values higher than 3 times the (temporal) standard deviation of a pixel are considered as a glitch. Note that this method is not robust when data are not stabilized.
- **Deglitch / Spatial:** All values higher than 3 times the (spatial) standard deviation of a pixel are considered as a glitch.
- **Deglitch / Temporal & Spatial:** Both temporal and spatial informations are used. This method doesn't allow to detect either strong glitches which remain for many exposures, or successive glitches hitting the same pixel within a short time interval.
- **Deglitch / Multiresolution Median [default]:** Multiresolution information is taken into account to detect all significant small scale structures (not due to the noise). As no structure can be detected either in the first frame or in the last one, these frames are completely masked and lost. This method gives good results even when data are not well stabilized.
- **Transient / None:** No transient correction is performed.

- **Transient / 90% Of Final Flux (modeled):** A model based on ground-calibrations is used to determine the number of stabilized frames given the first and last flux values of pixels.

- **Transient / 90% Of Final Flux (measured) [default]:** Pixel values exceeding the last value $\pm$ 10% of the difference between the first and last values are considered as not stabilized.

- **Transient / IPAC Model Fitting:** The model developed at IPAC is used to fit the transient behavior of the detector.

- **Transient / IAS Model Fitting:** The model developed at IAS is used to fit the transient behavior of the detector.

- **Transient / SAP Model Fitting:** The model developed at SAP is used to fit the transient behavior of the detector.

- **Transient / Remove Ghosts:** An adaptive filtering is applied in order to determine if an upward or a downward transient is detected. If an upward transient is detected then the last detected flux is considered as the stabilized one. If a downward transient is detected then the stabilized flux is assumed to be equal to that of the background. This method allows to remove all ghosts in the reconstructed raster map, but it can lead to a loss of information (if a source is superimposed on a downward transient for example).

- **Flatfield / None:** No flat-field correction is performed.

- **Flatfield / User Input:** A user flat-field can be given as input in order to correct the flat-field effect. The user must then use the following command to start the session:

  ```
  CIA> x_cia, inflat=my_flat
  ```

  my_flat is a 32×32 array.

- **Flatfield / Cal-G [default]:** Flatfield is extracted from the calibration database.

- **Flatfield / Auto:** Flatfield is obtained by computing the median of all frames corresponding to the same configuration. It is normalized on the 10×10 central pixels.

- **Flatfield / Manual:** Flatfield is built interactively by selecting only a subset of all frames of the data cube and normalized on the 10×10 central pixels. This method is well adapted whenever only background is observed at the edge of large raster map. If a source is detected in each frame, it is also possible to exclude interactively some rectangular zones before computing the flat-field.

- **Process / None:** Set to 'None' all calibration fields (Dark, Deglitch, Transient and Flatfield).

- **Process / Default:** Execute the default data processing. Warnings are displayed if some corrections have already been made to the current data set. Note that these warnings concern only the current **x_cia** session. So before processing data, the user should have a look at the data history in order to check that some data corrections have not been yet processed (see above Data / Display History).

- **Process / Selected:** Execute the user selected data processing. Warnings are displayed if some corrections have already been made to the current data set. Note that these warnings concern only the current **x_cia** session. So before processing data, the user should have a look at the data history in order to check that some data corrections have not yet been processed (see above Data / Display History).
- **Tools / xv_temp:** Launch **xv_temp**. See Section 14.4.6.
- **Tools / xv_raster:** Launch **xv_raster**. See Section 14.4.10.
- **Tools / isocont (ISOCAM axis):** Launch **isocont**. See Section 14.6.2.
- **Tools / isocont (RA/DEC axis):** As above, but the MOSAIC is displayed with EAST left and NORTH up.
- **Tools / x_isocont:** Launch **x_isocont**. See Section 14.6.3.
- **Tools / hardcopy (gif):** Create a print file of the **isocont** display (see above) in GIF format.
- **Tools / hardcopy (ps):** Create a print file of the **isocont** display (see above) in postscript format.

Some additional corrections can be performed. But second order corrections should not be used unless standard processing has been performed first to assess the quality of data.

- **Remove Dark Residuals:** Dark pattern residuals can be removed using filtering in Fourier space. Note that this secondary order method updates ISODATA.IMAGE, whereas first order dark correction updates ISODATA.CUBE.
- **Smooth Undefined Values:** It may happen that ISODATA.RASTER contains undefined values (if there is no stabilized value for example). Undefined values can be replaced with smoothed values, as determined from neighboring pixels.
- **FOV Distortion:** The FOV distortion effect can be corrected: the trapezoidal shape of pixels is then taken into account in order to re-sample the image. Distortion can be assumed to be linear around the central pixel (linear method) or can be calculated from ground calibration measurements (calibration method).
- **Spectral Deconvolution:** The measured spectrum can be deconvolved of the spectral resolution of the CVF filter. Note that the spectral sensitivity is corrected by selecting the Jy/pixel units instead of ADU/s/pixel.

Appendix A

Glossary

AA Auto-Analysis. As the name implies this is an automatic analysis of the telemetry data to produce AAR.

AAR (level) Auto-Analysis Results. AAR level refers to the AA processing level that produces AAR. In the *CIA User's Manual*, generally used to refer to the data products CCIM, CMAP, CMOS.

ADU Analogue to Digital Units.

AOT Astronomical Observation Template. In the *ISOCAM Observer's Manual* an AOT is defined as: 'Representation of an observing mode in the proposal generation software (PGA).' In the *CIA User's Manual* an AOT refers to a period of observing during which only one AOT type is employed (i.e one of either AOT#1, AOT#3, AOT#4, AOT#5). In this sense AOT and observing mode are synonymous (see glossary entries for observing mode). Note the following characteristics of the AOT:

- An AOT can be identified by its <TDT+OSN> number.
- One set of data products exists for each AOT.
- An AOT corresponds to a set of CONFIGURATIONS which in turn contains several OP-MODEs comprised of one or more STATEs.

AOT#0 This AOT is reserved for CUS use.

AOT#1 An AOT type where a raster, micro-scan, staring, tracking observation may be performed.

AOT#2 Not currently used.

AOT#3 An AOT type where a beam-switch observation is performed.

AOT#4 An AOT type where a CVF observation is performed.

AOT#5 An AOT type where a polarization observation is performed. Most polarization observations were performed by the CUS as AOT#99 observations.

AOT#99 CUS observations (mainly for calibration purposes).

beam-switch (observation) Also AOT#3: an observation where CAM alternates pointings on a target object with pointings on the selected fields of 'empty' sky surrounding the object. The 'empty' sky fields are known as the reference fields. There can be up to four reference fields in such an observation.

beam-switch structure See PDS.

calibration data Calibration data refers to data such as flat fields, DARK images etc. For the user this is synonymous with CAL-G data.

calibration data structure The calibration data structure (CDS) is the CIA data structure which is used to hold calibration data.

CAL-G [1] General term referring to CAM calibration data products. Note that in the *CIA User's Manual* the terms *CAL-G* and *calibration data* are synonymous. In CIA, CAL-G files are converted to Calibration Data Structures (CDSs) and the user does not work directly with them.

calibration data Refers to flat field images, dark images etc.

CAMTU CAM Time Unit. 1 CAMTU = 0.14000498 seconds.

CDS Calibration Data Structure. A CIA data structure for holding CAL-G data products. See entry for CAL-G.

CCIM CAM Calibrated IMage. An AAR level data product containing AA computed EXPOSUREs in detector coordinates. It is also a field in an SAD used to hold data from the CCIM data product or CIA processed data.

CGLL CAM Glitch List. An AAR level data product containing a list of AA detected glitches.

CIA user-friendly format A term referring to the manner in which CAM parameters are presented in the CIA data structures.

CIDT CAM Instrument Dedicated Team. Located in VilSpa, its main tasks were to monitor and calibrate ISOCAM.

CIER CAM Instrument Edited Raw data, i.e. edited telemetry data. An ERD level data product containing EOI and RESET frames. CISP or CIER can be the input data for CIA analysis.

CISP CAM Instrument Standard Processed data. A SPD level data product containing IMAGES computed by AA from CIER data. CISP or CIER data can be the input data for CIA analysis.

CIST CAM Instrument Support Team. Located in Saclay and Orsay (France), it supported the CIDT during the mission.

CJAM CAM Jitter And Memory. Also contains stabilization information. This is an AAR level data product.

CLEAN See OP-MODE.

[1] See the *ISOCAM Handbook* for more information on CAL-G data products.

CMAP CAM MAP. A data product containing AA computed EXPOSUREs in astronomical coordinates. It is also a field in an SAD used to hold data from the CMAP data product, the CMOS data product or CIA processed data.

CMOS CAM MOSaic. An AAR level data product containing MOSAICS constructed from EXPOSUREs, contained in the CMAP data product, of the same CONFIGURATION.

CONFIGURATION A subdivision of an AOT. A CONFIGURATION is comprised of a set of chronologically ordered OP-MODEs associated with the same (mandatory) OP-MODE (OBS) where CAM is configured to observe a celestial source. (See also STATE.)

CPSL CAM Point Source List. An AAR level data product containing a catalogue of AA detected point sources.

CSSP CAM Source SPectrum. An AAR level data product containing the measured spectrum for each point source detection.

CUFF CAM User-friendly File. An AAR level data product containing a log of messages from AA processing.

CUS Calibration Uplink System. It provides a more flexible interface to command the instrument and allows more freedom in the setting of instrument and observation parameters.

CVF Circular Variable Filter.

CVF observation Also AOT#4. This is a spectral observation: CAM pointing remains constant, but the CVF wavelength changes.

CVF data structure See PDS.

DARK In the context of calibration data it refers to a dark current image. It is also an OP-MODE when CAM is performing a DARK measurement.

data preparation Also know as slicing. See entry for slicing.

data product Refers to the FITS files on your CD-ROM.

data product version number Identifies the version of a data product.

data structure A programming term referring to a record of data, organised in to a single addressable entity. The structure may contain data of mixed types (bytes, integers, strings, etc.) and may include structures. An array is a simple example of a data structure.

DFLT Detector FLaT-field.

DSD Diagnostic Specific Data. CIA data structure containing house-keeping data.

EOI End Of Integration. This refers to a CAM CCD read-out (i.e. FRAME) after integration is complete.

ERD (level) Edited Raw Data. ERD level refers to the level of pipeline processing that produces ERD. For CAM ERD is synonymous with CIER.

EXPOSURE EXPOSURE, when written in upper-case in the *CIA User's Manual*, refers to a single image, computed by averaging IMAGEs over a STATE (see also MOSAIC and FRAME).

FITS (data products) The data products on the CD-ROM are in the form of FITS files, the actual CAM image being stored in a binary extension of the FITS file.

FLAT A flat-field image. For CAM this is the product of an OFLT and a DFLT (see glossary entries for both of these terms.) Also an OP-MODE when CAM is performing an internal flat-field measurement.

FRAME FRAME, when written in upper-case in the *CIA User's Manual*, refers to a single CAM CCD read-out. There are two possible FRAMEs: EOI or RESET. (See also MOSAIC and EXPOSURE.)

future **SAD** A flavour of SAD which may contain a MOSAIC either taken from the CMOS data product or computed with CIA from EXPOSUREs. It may also contain spectra.

IDA ISO Data Archive. Archive of ISO data products at Villafranca which is accessible over the internet (`http://www.iso.vilspa.esa.es`).

IDLE See OP-MODE.

IIPH Instrument Instantaneous Pointing History. Contains instantaneous pointing information for the prime instrument during an AOT. For most users with CAM data, CAM is always prime.

IRPH Instrument Reference Pointing History. Contains reference pointing information for the prime instrument during an AOT. For most users with CAM data, CAM is always prime.

IMAGE Refers to an image computed from the EOI and RESET FRAMES. In theory, IMAGE-=EOI-RESET. For the CAM LW detector this is indeed true, but for the SW detector it is slightly more complicated and CCD lines have to be interleaved. (see MOSAIC.)

micro-scan (observation) A particular CONFIGURATION of CAM AOT#1. This is a special case of a raster observation with a small pointing step size so that EXPOSUREs greatly overlap.

MOSAIC Refers to the final image from an AOT. It is computed from the EXPOSUREs, which are in turn computed from IMAGEs, which are computed from the RESET and EOI FRAMEs. (See glossary entries for an explanation of each of these terms.)

M-direction M-direction and N-direction are the axes along which CAM performs a raster.

N-direction See M-direction.

observation data (structures) A broad term that refers to all the data products you need to make up uncalibrated CAM IMAGEs. In other words, it excludes calibration data. The observation data structure is any structure holding such data (see SCD, SSCD, SAD, SSAD).

observing mode OP-MODE when CAM is obtaining observation data. Also abbreviated as OBS.

OBS see observing mode.

official **name** Refers to the full name of a data product or CIA data structure.

OFLT Optical FLaT-field.

OLP Off Line Processing (also known as the pipeline).

OP-MODE Operational Mode. CAM can be operating in one of several modes:

> **IDLE** CAM is idle.
>
> **OBS** performing an astronomical observation.
>
> **DARK** obtaining a dark frame.
>
> **FLAT** obtaining an internal flat-field image.
>
> **CLEAN** executed to remove saturation remnants from the detector.
>
> **WAIT** CAM is waiting for a good CONFIGURATION.

An OP-MODE may be comprised of one or more STATEs. (See CONFIGURATION and AOT.)

ORBIT A CAL-G file containing information on orbital parameters for all ISO revolutions up to and at the very least including the revolution during which your AOT is performed.

origin **SAD** A flavour of SAD containing which contains an EXPOSURE one from each of the CCIM and CMAP data products, or similarly CIA processed data. Also may contain a glitch list, a point source list or jitter information.

OSN Observation Sequence Number. Identifies an AOT within a TDT. See also <TDT+OSN>.

OTF On Target Flag. OTF=1 indicates that CAM is looking at the proposed target. When OTF=0, then CAM is off target. Each IMAGE has its own OTF value.

When slicing your data, you have a choice of slicing by *our* OTF, or just OTF. The former means that you take 143 as the good value of the QLA flag and the latter means you take 7 as a good value. The criteria for deciding if CAM is on target are stricter for the latter than for the former. See also QLA flag.

PDS Prepared Data Structure. Three flavours exist: a CVF PDS, a raster PDS (also known as a raster data structure), a beam-switch PDS and a general PDS. These data structures are created from an SSCD with one of the routines **get_sscdcvf**, **get_sscdraster**, **get_sscdbs** and **get_sscdstruct** respectively. They are used to hold all the sliced (i.e. prepared) data that you need to perform normal calibration. Usually, the data in a PDS corresponds to a single CONFIGURATION.

pipeline (processing) refers to the OLP processing that creates the ERD, SPD and AA data products delivered on an ISO CD-ROM or retrieved from the IDA.

PMA Post Mission Archive. See IDA.

polarization (observation) See AOT#5.

QLA flag Quick Look Analysis flag. Originally, this was a telemetry flag, but in CIA it is redefined as a combination of the original QLA flag, the original OTF flag and the OTF-sum flag. What this all actually means is that the QLA flag has two values which may be taken to indicate a good FRAME: 7 or 143. Both of these indicate the CAM is on target, but the latter is a stricter criterion than the former. See also OTF.

raster data structure See PDS.

raster (observation) A particular CONFIGURATION of CAM AOT#1. A raster is performed when CAM makes a series of regularly spaced pointings on the sky, allowing a MOSAIC to be constructed of a region of sky greater then CAM's FOV.

Reference fields See beam-switch observation.

RESET This refers to a CAM CCD read-out (i.e. FRAME) taken before integration begins. (see MOSAIC.)

revolution (number) An orbit of ISO. The Revolution Number identifies a particular orbit.

SAD Science Analysed Data. A CIA data structure used to hold a CIA computed EXPOSURE (see *origin* SAD) or MOSAIC (see *future* SAD), or similar data from the AAR data products.

saturation Occurs when a CAM detector is exposed to a bright object and pixels reach their full well capacity. Sensitivity of affected pixels can seriously be altered for succeeding observations. See entry for CLEAN.

SCD Science CAM Data. Comes in two flavours: *ERD* SCD and *SPD* SCD. The former is a data structure used to hold EOI and RESET FRAMES from the CIER and the latter to hold either CIA computed IMAGEs or IMAGEs from the CISP.

slicing Slicing data refers to the CIA process of producing *SPD* SCDs(one per STATE) and SSCDs (one per CONFIGURATION) from the bulk AOT data contained in the CISP or CIER data products.

SPD (level) Standard Processed Data. SPD level refers to the pipeline level of processing that produces SPD. For CAM, SPD is synonymous with CISP.

SSAD Set of Science Analysed Data. A data structure cataloging SADs from an AOT or any subset of an AOT, e.g. a CONFIGURATION.

SSCD Set of Science CAM Data. A data structure cataloging SCDs from an AOT or CONFIGURATION.

staring (observation) A particular CONFIGURATION of CAM AOT#1. A staring observation is simply one pointing of CAM.

STATE The finest time division of CAM activities and a subdivision of an OP-MODE. All CAM parameters are fixed during a STATE, but parameters may change between STATEs. This depends on the observation type:

- **beam-switch (AOT#3), raster and micro-scan (AOT#1)** – CAM pointing changes with STATE
- **CVF (AOT#4)** – CVF position changes with STATE.
- **polarization (AOT#5)** – the entrance wheel changes with STATE.

TDF Telemetry Distribution File (or FORMAT). The data-stream down-linked from ISO.

TDT (sequence number) Target Dedicated Time. The contiguous time spent on the observation of a source in an ISO revolution (including integration time and overhead times.) An AOT is a subset of a TDT. The TDT sequence number identifies a TDT within a revolution.

<TDT+OSN> Combination of TDT number and OSN number. Uniquely identifies an AOT.

tracking (observation) A particular CONFIGURATION of CAM AOT#1. A tracking observation is used for observing Solar System objects. CAM pointing changes to track the target object.

Appendix B

CIA command short-list

This quick reference guide lists core CIA commands and groups them by processing level. User's may find it useful to make a copy of this chapter and keep it near to hand. A example of usage and a very brief description is given for each command. Only the more important arguments in the calling sequence are used – for an exhaustive list see the header documentation in the on-line help (Section 2.3.2). Examples of usage for almost all the commands here appear in the Quick Start Guide, or elsewhere in the *CIA User's Manual.*

B.1 Data preparation (slicing)

This section lists routines that are used for generating an *SPD* SSCD per CAM CONFIGURATION. Further routines are listed for creating the appropriately flavoured PDS from the *SPD* SSCD. For a thorough treatment of slicing go to Chapter 12.

spdtoscd creates an *SPD* SSCD (per observation) from an SPD FITS product. This is probably the simplest and fastest way to create an *SPD* SSCD.

```
CIA> spdtoscd, 'cisp02600506.fits', spd_sscd, dir='.', /nowrite
```

x_slicer is a widget program that can create an *SPD* SSCD (per CAM CONFIGURATION) from either an ERD or SPD FITS product. Very flexible and interactive approach to slicing.

```
CIA> x_slicer
```

erdtoscd creates an *ERD* SSCD (per observation) from an ERD FITS product.

```
CIA> erdtoscd, 'cier02600506.fits', erd_sscd, dir='.', /nowrite
```

frames_to_image converts an *ERD* SSCD to an *SPD* SSCD.

```
CIA> spd_sscd = frames_to_image( erd_sscd )
```

sscd_info lists all the SCDs of an SSCD.

```
CIA> sscd_info, spd_sscd
```

sscd_clean breaks an *SPD* SSCD into one *SPD* SSCD per CAM CONFIGURATION.

```
CIA> cleaned_sscd = sscd_clean( spd_sscd )
```

sscd_elem returns a list of SCDs from an SSCD. In the example the list is stored in the string array *scds*.

```
CIA> scds = sscd_elem( sscd )
```

scd_del deletes SCDs from memory. In the example, delete the first SCD named in the string array *scds*.

```
CIA> scd_del, scds[0]
```

get_sscdraster creates a *raster* PDS from an *SPD* SSCD containing raster or micro-scan observation data (CAM01).

```
CIA> raster_pds = get_sscdraster( spd_sscd, magnify=magnify )
```

get_sscdstruct creates a *general* PDS from an *SPD* SSCD containing staring observation data (CAM01).

```
CIA> staring_pds = get_sscdstruct( spd_sscd )
```

get_sscdbs creates a *BS* PDS from an *SPD* SSCD containing beam-switch observation data (CAM03).

```
CIA> bs_pds = get_sscdbs( spd_sscd )
```

get_sscdcvf creates a *CVF* PDS from an *SPD* SSCD containing CVF data (CAM04).

```
CIA> cvf_pds = get_sscdcvf( spd_sscd )
```

B.2 Data calibration

All routines listed here that have the keyword *method* have a selection of different algorithms that can be applied to the data. For an exhaustive list of methods see Chapter 20 or the on-line help. In general the variable *pds_scd* refers to an SCD, an SSCD or any flavour of PDS. The variable *pds* refers to any flavour of PDS. An appropriate variable name will be used in examples of routines which operate on a PDS of a particular flavour.

corr_dark Remove dark current from IMAGEs in .CUBE using the method *dark model*.

```
CIA> corr_dark, pds_scd, method='model'
```

deglitch Deglitch IMAGEs in .CUBE using the method *mm*.

```
CIA> deglitch, pds_scd, method='mm'
```

stabilize Stabilize IMAGEs in .CUBE using the method *s90*.

```
CIA> stabilize, pds_scd, method='s90'
```

corr_jitter Compute the jitter in the IMAGEs in .CUBE. The arrays .DU and .DV are filled with the jitter offsets.

```
CIA> corr_jitter, pds_scd
```

reduce Reduce the IMAGEs in .CUBE to EXPOSUREs in .IMAGE.

```
CIA> reduce, pds_scd
```

corr_flat Perform flat-field correction on the EXPOSUREs in .IMAGE.

```
CIA> corr_flat, pds_scd
```

raster_scan Useful only with a *raster* PDS. It creates the raster MOSAIC and places it in .RASTER. In the example here a projection method is used.

```
CIA> raster_scan, raster_pds, method='project'
```

reduce_bs Useful only with a *BS* PDS. It creates the beam-switch MOSAIC and places it in .RASTER.

```
CIA> reduce_bs, bs_pds
```

conv_flux Converts data from ADU/gain/sec to janskys.

```
CIA> conv_flux, pds
```

B.3 Data visualization

This section summarizes some of the visualization routines that appear in Chapter 14.

tviso The simplest display routine. Accepts any 2D numeric array, e.g. a raster MOSAIC.

```
CIA> tviso, raster_pds.raster
```

x3d Examine a 3D numeric array, i.e. a cube of images. To display .CUBE with the option of indicating masked pixels:

```
CIA> x3d, pds
```

isocont Displays an image and overlays with contours from another image. Screen output can be directed to a postscript file. Useful for multi-wavelength comparison. Requires that images are accompanied by astrometry. It will take FITS files and *raster* PDSs and *BS* PDSs as input. Supposing that you have a DSS FITS image, and a CAM raster of the same field-of-view, then to overlay the optical and IR data:

```
CIA> isocont, raster_pds, dss_fits
```

xv_raster Click on a raster MOSAIC pixel and get its time history.

```
CIA> xv_raster,  raster_pds
```

xv_temp Cube display. Similar to **x3d**, but more features relating to the display of the pixel time history.

```
CIA> xv_temp,  pds.cube
```

show_frame More cube display. With this routine many frames are displayed simultaneously. Display all the EXPOSUREs in .image:

```
CIA> show_frame, pds.image
```

cvf_display Interactively plot spectra from a CVF observation.

```
CIA> cvf_display, cvf_pds
```

B.4 FITS input/output routines

This section summarizes some of those routines that appear in Chapter 18.

raster2fits writes the *raster* PDS fields .RASTER, .RMSRASTER and .NPIXRASTER to a FITS primary array.

```
CIA> raster2fits, raster_pds, name='raster.fits'
```

Works also for a *BS* PDS.

```
CIA> raster2fits, bs_pds, name='bs.fits'
```

imagette2fits writes each frame or EXPOSURE in the PDS field .IMAGE (along with the corresponding frame of .RMS and .NPIX) to the primary array of an individual FITS file.

```
CIA> imagette2fits, pds, name='cvf.fits'
```

struct2fits writes an entire CIA data structure to a FITS file. The data is stored in extensions.

```
CIA> struct2fits, raster_pds, name='raster_archive.fits'
```

fits2struct recovers the output of **struct2fits**. It initializes the appropriate PDS and then fills it as best it can. It is very useful for upgrading the architecture of an obsolete PDS.

```
CIA> fits2struct, 'raster_archive.fits', hdr, raster_pds_recovered
```

B.5 Online help

The various methods of getting online help are listed here.

ciainfo is an alias for the old style **widget_olh**, where the headers of all CIA's IDL routines may be found. Note that **alias** must be compiled before attempting to invoke **ciainfo**.

cia_help invokes a searchable dedicated CIA help.

? invokes IDL's hyper-help.

Appendix C

The ISO CD-ROM

This appendix provides a guide to mounting and using the ISO CD-ROM. It is not applicable to observers who have obtain their data via IDA. It shows you where your data, the documents listed in Section 1.2 and some much less technical information (nice JPEG images of ISO and its instruments) can be found on the CD.

C.1 Mounting the CD-ROM

Mounting the CD should be a simple job[1]. The commands below should work on your system. Your system administrator can provide the *devicename* and advice if you have any problems.

C.1.1 VMS

To mount the CD-ROM on a VMS machine, try the following command.

```
> mount/noass/over=id/media=cdrom/undefined_fat=(fixed:none:512) <devicename>
```

C.1.2 UNIX

Mounting a CD-ROM on a UNIX machine really depends on the UNIX OS you are running. It is best to ask your system administrator for advice.

C.2 Overview of the CD-ROM Contents

From the top of the directory tree[2] one can find two subdirectories, */aboutiso* and */products*. Those directories below the former lead you to the ISO documentation and general ISO information, and those below the latter to the data products. Also, there are two text files in the root directory, */README* and */datalist.txt* which contain details of the relationship of the product list and the directories below *products*.

[1] As is usual with VMS, you can only access the CD-ROM from the window in which you mounted it.

[2] Refer to the *ISO Satellite Handbook* for a detailed listing of the contents of an ISO CD-ROM and a description of the directory tree structure.

C.2.1 Where to find the ISO documents

The directory */aboutiso/docs* contains two subdirectories relevant to ISOCAM observers. These are */aboutiso/docs/cam* and */aboutiso/docs/iso*. The former contains the documents specific to ISOCAM and the latter documents for all ISO observers. Each of these directories contain subdirectories named *./idum* and *./obsman*, where the IDUMs and the Observer's Manuals (see Section 1.2) can be found in both postscript format and original LaTeX. Other documents exist in parallel directories, but these are only relevant to other ISO instruments.

C.2.2 Where to find the Data Products

The file */datalist.txt* is an important reference document – making a hardcopy for future reference is advisable. In addition to some archive information, such as data product owner and lot ID number, it contains a list of all the data products (using their *official* names), along with their version number and their location on the CD-ROM (see an sample listing of *datalist.txt* below.). Chapter 9 explains what the different types of data products actually contain and what the *official* filename convention is. Here we will merely point you to your data.

```
[Owner identification]
ID :       VMARINI
Name :     Dr.   Valerie   Marini
Address :  ESA - Vilspa

           P. O. Box  50727
City :    28080  V.d. Castillo, Madrid
Country :  Spain

[Lot identification]
Serial number : 601210411878
Lot ID : 123
Seq NR : 12
CD label : I0012312

[Product sets]
Product set NR: P0007780, Proposer ID : VMARINI
        aocs14300601    0577    /products/p0007780/14300601/aocs.fit

        ccglwdark       0481    /products/p0007780/others/ccglwdar/k.fit

        ccglwdead       0443    /products/p0007780/others/ccglwdea/d.fit

etc....
```

The directory */products* will contain one subdirectory per observation, or TDT, of the form */products/pmmmmmmm*, where *pmmmmmmm* refers to the product set number. Each of these subdirectories contain further subdirectories of the form */products/pmmmmmmm/nnn*, */products/pmmmmmmm/nnnxxxyy* and */products/pmmmmmmm/others*, where *nnnxxxyy* refers to the combined TDT and OSN number (see Table C.1 for a summary of variables used to identify directories containing data products). These lower directories respectively contain data pertaining to a revolution, observation data and calibration data.

C.2.3 Where to find nice images of ISO

Images of the ISO launch and instruments in JPEG, GIF and MPEG can be found in the directory */aboutiso/images*. A standard image display package should exist on your system for viewing these images.

Variable	Description
nnn	revolution number
xxx	TDT sequence number within a revolution
yy	OSN number
nnnxxx	TDT number
nnnxxxyy	combined <TDT+OSN>
pmmmmmmm	product set number
kkkk	data product version number

Table C.1: Common variables used in CIA and the *CIA User's Manual* when identifying data products. For example, the location of the data products on an ISO CD-ROM is */products/pmmmmmmm/nnnxxxyy*.

Appendix D

Guidelines for writing CIA routines

D.1 Introduction

Users are encouraged to supply their own routines for inclusion into CIA. In such cases it is asked that the following be understood.

- Routines should not duplicate existing code or data structures. If in doubt, please ask before coding! Additional data structures will not be permitted.
- Supplying a routine to CIA carries with it an implicit agreement to its distribution to all CIA sites.
- It should be clear to all parties involved with whom responsibility for the maintenance of the routine lies.
- Routines should conform to the requirements outlined in Section D.2.

Users can also supply routines to be distributed with CIA as contributed routines (found in the directory *$cia_vers/contrib*. These are not supported by the CIA team. Bug reports should be made directly to the author.

D.2 Basic requirements

Please ensure that your routine conforms to the following requirements:

- All files must contain a header (see Section D.3 below).
- A routine should display its calling parameters if it is called without any parameters.
- High and medium level routines should use the CIA error reporting conventions (CIA is full of examples). Normally this will be routines in the 'User' category.
- Mask handling and the setting of undefined values should follow the appropriate conventions.

D.3 How to write a header

In order for automatic formatting programs to work, as well as for purposes of standardisation, these rules must be followed in all headers.

- A header begins with `;+` and ends with `;-`
- There is only one header per file, regardless of the number of routines contained therein. It may be anywhere in the file, but most programmers prefer to find it at the top.
- The keywords are:

```
; NAME:
; PURPOSE:
; CATEGORY:
; CALLING SEQUENCE:
; INPUT:
; OPTIONAL INPUT:
; KEYED INPUT:
; KEYED OUTPUT:
; OUTPUT:
; OPTIONAL OUTPUT:
; EXAMPLE:
; ALGORITHM:
; DEPENDENCIES:
; COMMON BLOCK:
; SIDE EFFECT:
; RESTRICTION:
; CALLED PROCEDURE:
; SEE ALSO:
; MODIFICATION HISTORY:
```

  Note that the space and the upper case are compulsory, but you can add extra characters, i.e. `; KEYED INPUTs:` is correct, but not `; KEYED Input:`

- The field of the keyword `; CATEGORY:` is defined in the file `category.txt`. This file can be found in the document directory of the CIA distribution. The `; CATEGORY:` field should also specify whether the routine is 'User' or 'Internal', for example:

```
; CATEGORY: III-2  User
```

- The compulsory keywords are:

```
; NAME:
; PURPOSE:
; CALLING SEQUENCE:
; INPUT:
; OUTPUT:
; EXAMPLE:
; MODIFICATION HISTORY:
```

- The order of keywords is not important, except that `; MODIFICATION HISTORY:` must be the last. It is recommended that `; NAME:` is the first.

- Format for `; INPUTS:`, `; OUTPUTS:`, `; KEYED INPUTS:`, `; KEYED OUTPUTS:`

  They can be described as you wish, but it is better to follow the following rules in order to have a nice table. Put your general comments, if needed, on the first lines, without '`--`' inside. Then at the beginning of a line: name of the variable '`--`' type of the variable : description of the variable. The separators are '`--`' and '`:`'. For the second separator you can use '`:`' or '`;`', but commas can't be used. For example:

```
; OUTPUTS:
;   imt_block -- intarr(300,n) : the IMT_block.
;   CLEAN_CUBE -- Data output cube ; CUBE_INPUT with undefined
;                 values instead of identified glitches
```

- Name:

  The name in the header can be on the same line as the keyword `; NAME:` or on a single line after. This is compulsory.

- Calling sequence:

  The calling sequence must be on the line after the keyword `; CALLING SEQUENCE:`. The calling sequence can be on a single line or on two consecutive lines. This is compulsory.

D.4 Automatic inclusion of new processing algorithms in CIA

In order for easy addition of new algorithms into CIA, the core CIA calibration routines have been designed to automatically call new low-level processing routines. In order for this automatic mechanism to work the name and calling sequence of new routines must conform to a standard. As described below, this is dependent on the type of processing the new routine will perform. In all cases additional keywords can be used by the new routine – keyword inheritance ensures that keywords are passed by the core calibration routine down to the new low-level processing routine.

raster_scan Raster MOSAIC routines should have the calling sequence

`raster_`*method*`, raster_pds`

For example, let us say that I have a new routine which implements a raster MOSAIC creation algorithm (or method) called *xxx*. Then my routine would have the calling sequence

`raster_xxx, raster_pds`

and would take a *raster* PDS as input.

corr_dark Dark correction routines should have the calling sequence

`dark`*method*`, dark, dark_error, ack=ack`

where *dark* is the output DARK image and *dark_error* is the error on *dark*.

corr_flat Flat correction routines should have the calling sequence

flat = flat_*method*(input, error=error, ack=ack)

where *flat* is the FLAT image, *input* is the user input to **corr_flat**, and *error* is the error on *flat*.

deglitch Deglitching routines should have the calling sequence

deglitch_*method*, cube, cube_out, mask_out, nsigma=nsigma

where *cube* is all the IMAGEs from one pointing of CAM, *cube_out* is the deglitched *cube*, $mask_out$ flags glitched pixels in *cube* and *nsigma* is the sigma deglitching threshold.

stabilize Transient correction routines should have the calling sequence

corr_transient_*method*, cube, mask_in, mask_out

where *cube* is all the IMAGEs from one pointing of CAM, *mask_in* is the MASK corresponding to *cube* and $mask_out$ flags unstable pixels in *cube*.

Appendix E

ISOCAM astrometry: angles and coordinates

With the roll angle of ISO constrained by the sun position, CAM does not produce images conveniently oriented to the standard astronomical convention. This appendix[1] should help you to understand how your IMAGEs, EXPOSUREs and MOSAICs are oriented and and how to go about changing their orientation. A description of the FITS convention and how it relates to CAM angles can be found in Section E.3.

E.1 Definitions

Ambiguous definitions (sometimes intentional) of angles of importance in ISOCAM observations are spreading confusion in minds and in routines. This appendix sums things up to the best of our current knowledge for the LW part of CAM (the SW is 'a priori' and simply deducible from the LW case).

Before the drawings, some points to note:

- We are using only the astronomical convention for projections. In this convention, when the celestial North is pointing **upward**, the celestial East is pointing **leftward**. In this representation δ increases upward and α increases leftward.
- ISOCAM rasters are reconstructed along axes such that the M scanning direction goes from **left to right** and the N scanning direction from **bottom to top**. Therefore the North axis is generally not pointing upward in these images. However a simple rotation is enough to restore the astronomical convention (i.e. no mirroring of the image is needed).
- The + sign after a letter is a convention to specify an oriented axis.
- The Y and Z axes are defined by the spacecraft and are indentical for all instruments (though they project differently in their various focal planes). The X axis never appears because it is pointing to the target. Note that the (XYZ) referential is a direct one in 3D space, but that, due to the astronomical convention used, the (YZ) referential is indirect in the drawings.
- Rotation of a positive angle rotates in the direct trigonometrical direction, or counter-clockwise, i.e. unlike the IDL `rot` function.

[1] Taken from Sauvage M., 1996, *Angles and ISOCAM-LW*, v3.0

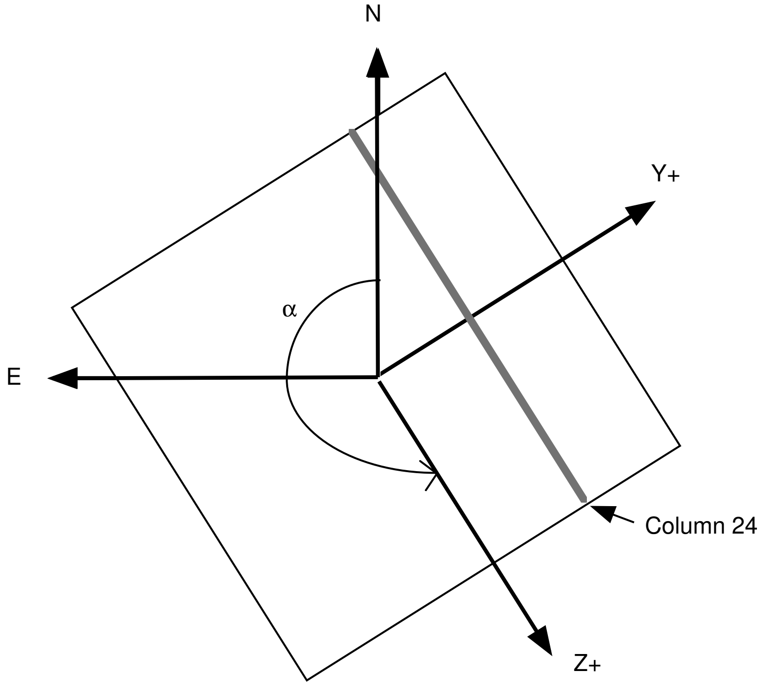

Figure E.1: Schematic of an ISOCAM-LW image. Y+ and Z+ are the satellite axes. Column 24 is indicated to help you visualize the array. α, called the roll angle, is the position angle of the Z+ axis, i.e. the angle between the celestial North and the Z+ axis, counted positively eastward.

- We assume that you have already used CIA and are quite familiar with its structure organization and basic routines.

E.1.1 Definition of the roll angle for CAM-LW

The roll angle for ISOCAM's LW detector is defined in Figure E.1. Note that while this diagram is correct, the orientation of your display can change the direction in which the axes point. In IDL, the orientation of a displayed image depends on the system variable !ORDER. Refer to Section E.2.6 for different axis orientations and the effects of !ORDER.

E.1.2 Rasters along the satellite axes

These rasters are called for simplicity's sake 'Y-axis' rasters and an example is given in Figure E.2. The sky is scanned along the satellite Y and Z axis. The raster axes are called M+ and N+ and the referential (M,N) is direct. In that referential ISO always starts its raster at the lower left corner.

The number of points in the M direction of the raster are held in the *raster* PDS field .RASTERCOL, which takes its value from the FITS keyword ATTRNPTS. The raster_pds field .M_STEPCOL (or the FITS keyword ATTRDPTS) gives the distance between adjacent columns in arcsecs. Likewise, the number of points in the N direction of the raster are held in the in the *raster* PDS field .RASTERLINE, which takes its value from the FITS keyword ATTRNLNS. The raster_pds field .N_STEPLINE (or the FITS keyword ATTRDLNS) gives the distance between adjacent lines in arcsecs.

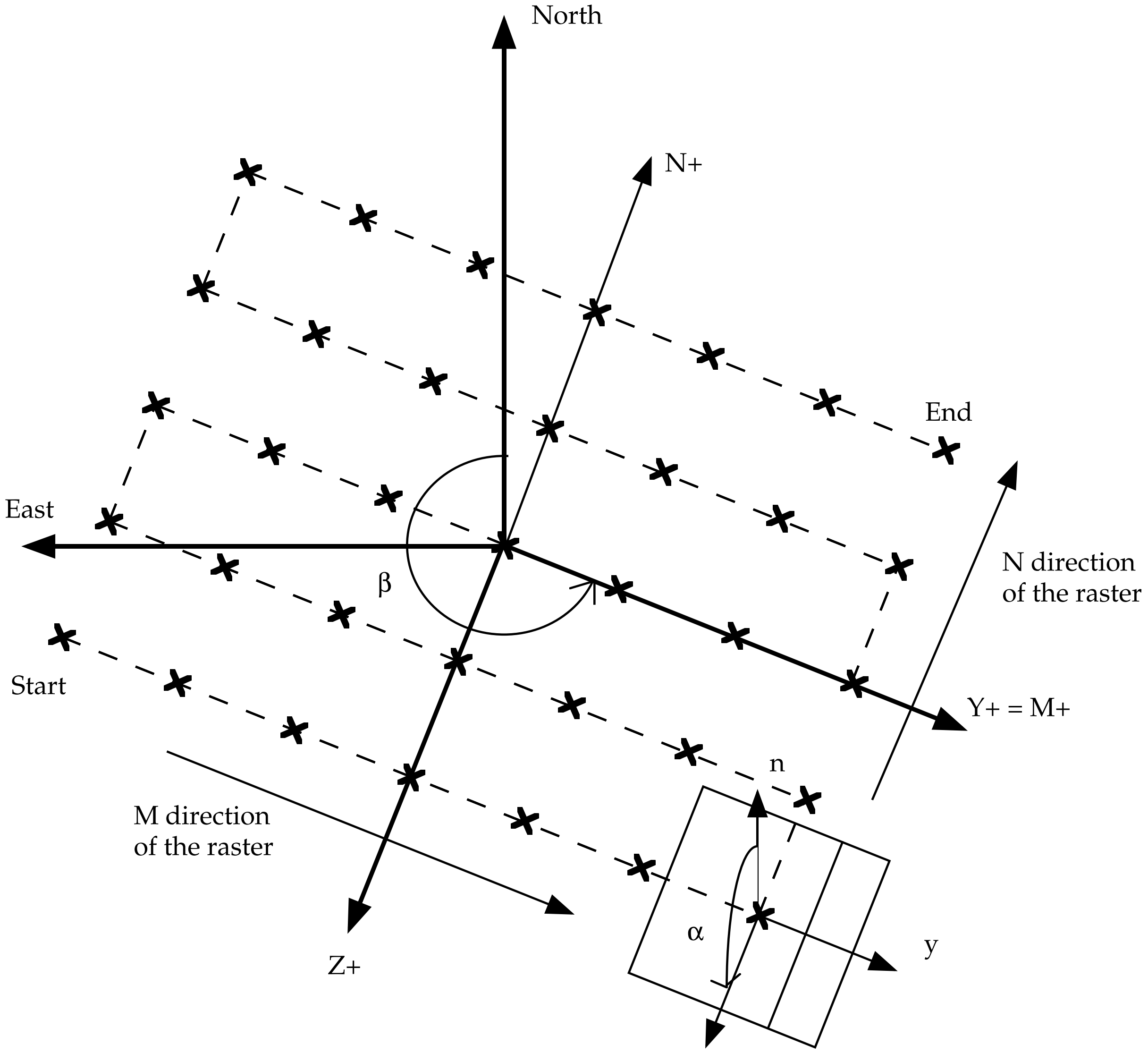

Figure E.2: Schematic of a Y-axis raster. Crosses, ×, mark successive positions of the raster. The start and end points are also indicated. The LW detector, with its 24th column marked, are shown on one position of the raster. The trajectory of ISO during the raster is shown by the broken line. This is a diagram of a M=7, N=5 raster.

By construction, The M+ axis is identical to the Y+ axis while the N+ axis is in opposition to the Z+ axis.

- α is the roll of the camera, as in Figure E.1. In the *raster* PDS α is found in the fields .ANGLE_RASTER and .INFO.ROLL. In the the SCD and SSCD, it is simply the field .ROLL.
- β is the so-called SSCD and *raster* PDS field .RASTER_ROTATION. It is the position angle of the M+ axis of the raster, i.e. the angle between the celestial North and the M+ axis, counted positively eastward.

In a Y-axis raster, RASTER_ROTATION cannot be specified by the observer at the PGA level, and a very simple relation links it with the roll angle:

$$\beta = \alpha + 90^\circ$$

or

$$\text{RASTER\_ROTATION} = \text{ROLL} + 90^\circ$$

These rasters can be reconstructed quite straightforwardly as the camera's axes are parallel to the raster axes.

E.1.3 Rasters referenced to the celestial North axis

These rasters are called in short 'North axis' rasters. An example of such a raster is shown in Figure E.3. These rasters can generate quite some confusion.

First, to program them, the observer had to specify the raster's position angle, but could only supply one comprised between 0° and 180° (in PGA this parameter was called *orientation angle*). Therefore there is a 180° uncertainty between programmation and reality (see Section E.2 to remove that uncertainty). In this section we are only concerned with 'reality', what has actually been performed.

Second, before reconstructing the raster, images have to be rotated by a certain angle. This angle is not written in the data but has to be derived from the two others that we know already, α and β. Figure E.3 show these angles, their definition follows:

- α is the roll angle of the mosaic, also called ANGLE_RASTER in CIA structures.
- β is the true RASTER_ROTATION comprised between 0° and 360°.
- γ is the angle between the M+ and Y+ axis, measured in the direct trigonometrical sense from the M+ axis to the Y+ axis.

Note that in that case, α and β cannot be deduced from one another.

Given that the rasters are reconstructed with the M+ axis as horizontal axis and the N+ axis as the vertical axis, prior to project an individual position in that image, the data have to be rotated by γ. Looking at Figure E.3, one has:

$$\gamma = \alpha - \beta + 90^\circ$$

or

$$\gamma = \text{ROLL - RASTER\_ROTATION} + 90^\circ$$

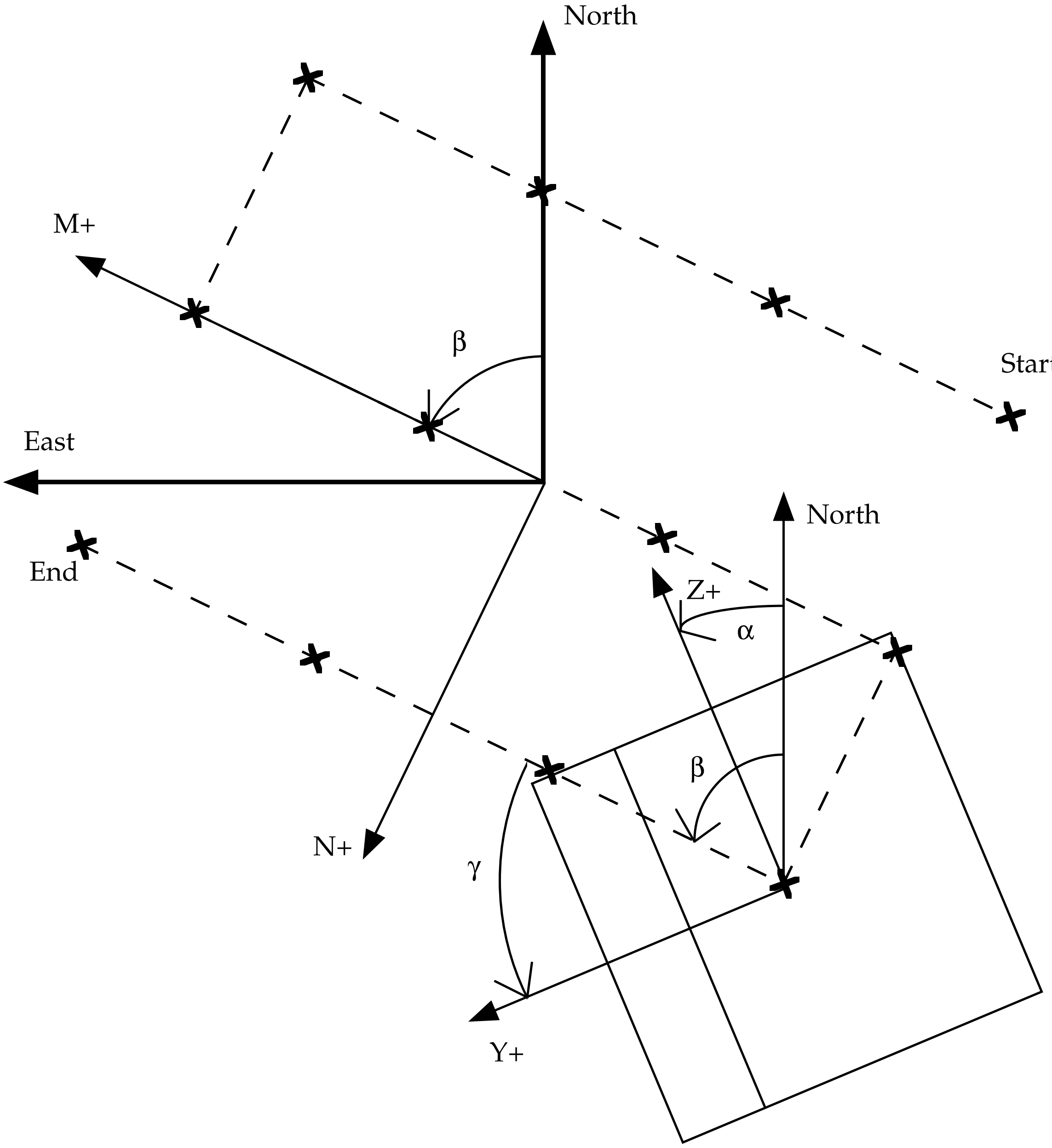

Figure E.3: Schematic of a M=4, N=3 raster oriented with reference to the North axis. Crosses,×, mark successive positions of the raster. Here the raster position angle given in PGA was approximately 65° (see text). Note however that the actual RASTER_ROTATION is in fact $\sim 245°$, as the location of start and end points shows. The camera, its 24$^{\mathrm{th}}$ column and pixel (0,0) are once again displayed for clarity purposes. Angles α, β, and γ are explained in the text.

E.2 Trouble shooting astrometry in CIA structures

You've read the previous section, you think you understand ISO's angles and you have been told that the current version of CIA complies with these definitions. Thus you confidently reconstruct your raster and…it fails! If the astrometry is seriously wrong then most likely you are using data that is not scientifically validated – data from OLP prior to version 4.0 is *not* scientifically validated. This problem and alternative problems are described in the following subsections.

If you still have not discovered what the problem is and it is not due to a known bug (Appendix 2.4), then send a report to the CIA team (see Chapter H for instructions).

E.2.1 Non-scientifically validated data

If you have such data there is a possibility that the IIPH you received with this data is not CAM's but that of another instrument. **x_slicer** may be able to correct this problem at the slicing stage by using the IFPG file. This file exists under two formats: a FITS file (obtained from the IDA or an ISO CD-ROM) or a CDS. **x_slicer** uses the CDS by default. However, it would be more advisable to obtain data reprocessed with OLP greater then version 4.0 via the ISO help desk:

```
isohelp@iso.vilspa.esa.es
```

If you want to confirm that IIPH is incorrect then simply print the header of the IIPH file:

```
CIA> image = readfits( 'iiph14300601.fit', hdr )

CIA> hprint, hdr
```

The keyword INSTRUME should be CAM, otherwise you know which instrument's IIPH you have received (usually PHT because of its serendipity mode).

E.2.2 The roll angle

The roll angle, α, has had a troubled history: it used to be the position angle of the Z+ axis but counted positively westward, or clockwise. This was changed at revolution 87. However, it is unlikely that you still have data that is old enough to be affected by this problem. If you do have such data you can attempt to correct it yourself, in which case read on, although you would be better off obtaining reprocessed data via the ISO help desk.

Note that when using the option 'project' to construct your raster MOSAIC with **raster_scan**, the roll angle of each individual STATE (SCD) is inspected. Therefore, it needs to be corrected in the SSCD, each SCDs of this SSCD, or, if you are working on a *raster* PDS, in .ANGLE_RASTER and in .INFO.ROLL.

A *raster* PDS can be fixed with:

```
CIA> rasterpds.angle_raster = 360 - rasterpds.angle_raster

CIA> rasterpds.info.roll = 360 - rasterpds.info.roll
```

To fix an SCD or SSCD, the field .ROLL may be treated similarly.

E.2.3 The raster rotation angle

If the roll angle was incorrect (described above) then it is likely that .RASTER_ROTATION will be incorrect also. For a Y-axis raster .RASTER_ROTATION is simply deduced from the ROLL by adding 90°. Thus if you have had to correct the ROLL, check the value of .RASTER_ROTATION and update it also.

If the raster is a North axis raster, there is a 50% chance that RASTER_ROTATION is wrong, except if you have created the raster structure after May 2, 1996. Indeed `get_sscdraster` computes RASTER_ROTATION for North axis rasters since then. Anyway it cannot harm to know how to do this yourself. If you have a *raster* PDS, called *raster_pds*, for which you have corrected the ROLL, then to correct .RASTER_ROTATION do the following:

```
CIA> raster_pds.raster_rotation=get_rasterorient(raster_pds)
```

E.2.4 Incorrect astrometry in beam-switch data

If after calibration the beamswitch MOSAIC (*BS* PDS field .RASTER) appears inverted and incorrectly has the astrometry of the reference field then read Section 19.4.

E.2.5 Astrometry inaccuracies

If the coordinates appear to be systematically wrong by a small amount then the problem is probably due to the repositioning accuracy of the lens and filter wheels. This can have a bigger effect than ISO's pointing accurracy. The error induced by the wheel jitter is typically less than 2 pixels, but can reach up to 3 pixels. For example, for a 6" PFOV measurement the coordinate error can be up 3*6" + 4" = 22".

E.2.6 Roll, image orientation and !ORDER

It is important to understand the effect of the IDL !ORDER system variable on the displaying of images. The difference is simply that an image will be displayed with pixel (0,0) in the upper-left corner when !ORDER=1, and in the lower-left corner when !ORDER=0. However, this simple difference can cause confusion when it comes to changing the orientation of a CAM image. Figure E.4 should help you understand the effects of !ORDER and how to correctly change the orientation of a CAM image.

E.3 Using FITS in CIA – new problems

E.3.1 FITS convention and IDL's astrolib

IDL comes with an astronomical package called the *Astrolib* which is in fact quite handy: most projection routines are already coded. However these assume that we are manipulating images that have a FITS header. As you know by now, with structures there is no need of a FITS header. Therefore to use these astrometric routines we have to create a FITS header appropriate to the images.

This is done with the idl routine `fits_header`. The syntax of the call is the following, with standard FITS keywords given with each parameter:

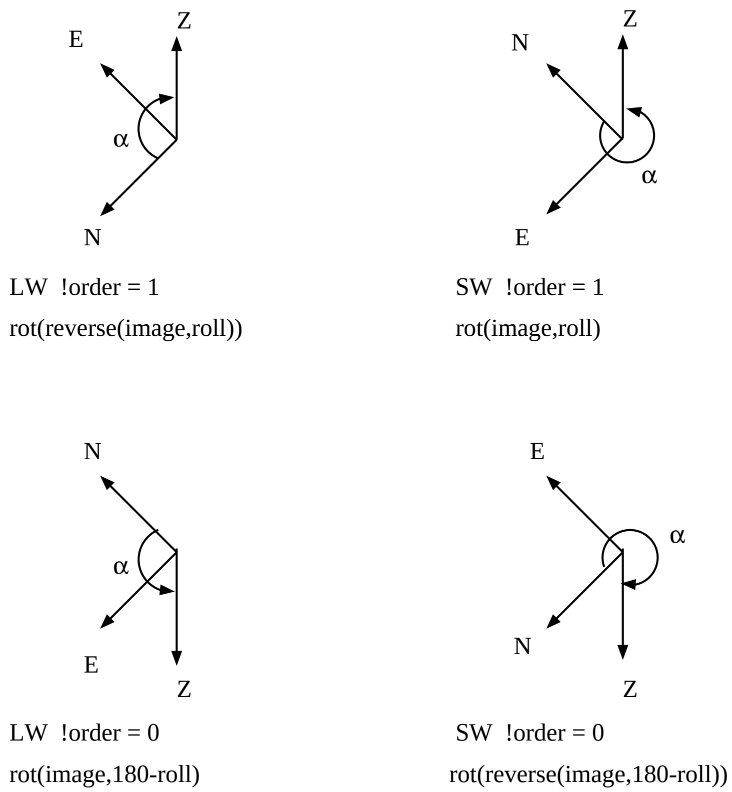

Figure E.4: The roll angle α for each detector and for each value of the IDL !ORDER system variable. The IDL command to rotate an image to the standard astronomical convention is shown for each case. Note that the ROT function rotates *clockwise*.

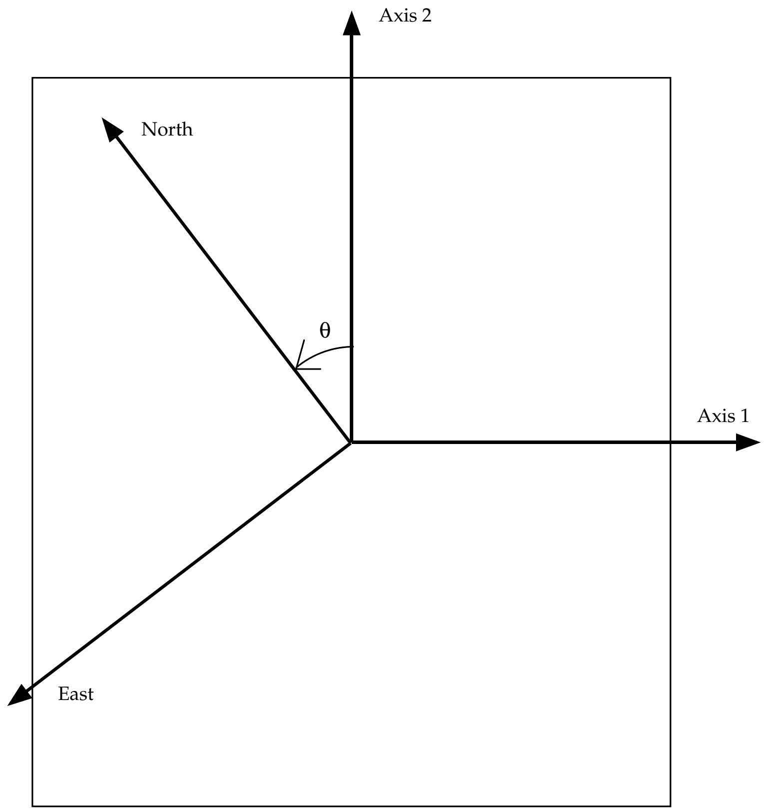

Figure E.5: Conventions for the standard astrometric keywords in a FITS header. Axis1 is the horizontal rightward axis, Axis2 is the vertical upward axis. CROTA2, θ, the rotation angle described in the header, has a definition which is **different** from that of α or β. It is the position angle of the North axis counted positively eastward from Axis2.

my_header = fits_header (

Nx,	*NAXIS1*
Ny,	*NAXIS2*
RefX,	*CRPIX1*
RefY,	*CRPIX2*
RA,	*CRVAL1*
Dec,	*CRVAL2*
Angle,	*CROTA2*
ResolX,	*CDELT1*
ResolY,	*CDELT2*
equinox = my_equinox	
cd = cd)	

Some explanations are required for those unfamiliar with FITS. To fully grasp them, please consult also Figure E.5). *NAXISn* are the number of pixels on the 2 axes. Axis1 is the horizontal rightward axis, and Axis2 is the vertical upward axis. In order to perform some astrometry, one need to know the coordinates (α, δ) of a reference pixel in the image, the plate scale and an orientation angle. *CRPIXn* are the pixel coordinates of that reference pixel (usually the central one), while *CRVALn* are its (α, δ) coordinates in degrees. *CROTA2* it the position angle of **the celestial North axis with respect to the Axis 2** counted positively eastward, i.e. it is different from the definitions of angles used in CIA where the North axis was the reference. Figure E.5 clearly shows that difference (it is the angle called θ). An incorrect value in *CROTA2* is the first error possibility when you find that `fits_header` 'does not work'. *CDELTn* are the plate scales on both axes. In the FITS header they will appear in degrees per pixel which are the units you must give to `fits_header`. Getting the units wrong is a second source of error when the routine 'does not work'.

The CDELT parameters must be given with the correct signs. These are:

LW	SW
CDELT1 > 0	CDELT1 < 0
CDELT2 < 0	CDELT2 < 0

The reason that *CDELT2* is negative is that pixel $(0, 0)$ is defined to be at the top of an ISOCAM image. This is of course an arbitrary decision but to reverse it *all* the signs must be changed.

Do not forget to specify the `equinox` keyword. IDL works by default in B1950.0 while ISO only uses J2000.0...

Optionally you can pass in the *CD* matrix if you already know it. It is defined as:

$$\begin{matrix} cos(crota2) & -sin(crota2) \\ sin(crota2) & cos(crota2) \end{matrix}$$

The easiest way to get the *CROTA2/CDELT/CD* parameters is to ask the **roll_to_crota2** routine to give you them. Its calling sequence is:

```
crota2 = roll_to_crota2(roll [,channel, pfov, cdelt, cd])
```

pfov is the pixel-field-of-view. For example,

```
CIA> crota2 = roll_to_crota2 (scd_get('roll',scd), 'lw', [3,3], cdelt, cd])
```

E.3.2 From CIA structures to FITS images

The main problem is to know which angle to supply to the `fits_header` routine. It obviously depends on the type of data.

E.3.2.1 Individual SCDs and the like

This works also for an SAD that would be the average of an SCD, for a reduced beamswitch or CVF. The only angle that is valid in that case is the ROLL, or α. The obvious way to transfer the data is to avoid unnecessary rotations during the transfer, thus the Y+ axis will be Axis1 and the Z- axis will be Axis2 (see Figures E.1 and E.5).

The angle that needs to be given to `fits_header`, CROTA2, is then the position angle of the North axis with respect to the Z- axis. Thus:

$$\theta = 360^\circ - \alpha$$

or

$$\text{CROTA2} = 360^\circ \text{ - ROLL}$$

E.3.2.2 Reconstructed raster images

In that case, the meaningful angle is .RASTER_ROTATION, β. The individual images can no longer be distinguished therefore their ROLL is irrelevant to the raster (it is only coincidentally meaningfull for Y-axis rasters). Here also the simplest way to store the image is to have the M+ axis be Axis1 and the N+ axis be Axis2. Thus, from Figures E.3 and E.5 one has:

$$\theta = 90^\circ - \beta$$

or

$$\text{CROTA2} = 90^\circ \text{ - RASTER\_ROTATION}$$

Appendix F

What is new in CIA 4.0

F.1 New and improved algorithms

- New **transient correction**: *fs* using an improved implementation of the Fouks-Schubert transient correction leads to better results and to a reduction in processing time up to a factor of 20 (Section 20.2.3).
- Improved **dark correction** *vilspa* using a second order time-dependent correction reduces the dark uncertainty by up to 30 % (Section 20.2.1).
- Access to **long-term transient and variable flat-field algorithms** routines via the **SLICE** package (Chapter 21).
- **back_project** allows the possibility to back project a raster MOSAIC to a set of EXPOSUREs to verify stability and deglitching (Section 20.14.4).
- **raster_scan** has now the option */weight* to use the **weighted mean** to create a mosaic. (Section 20.14).

F.2 New and improved functionality

- **CIA is now Y2K compliant.**
- **Handling of Solar Sytem Object observations** (Section 5).
- **Handling of polarisation observations** (Section 8).
- The new tool **ximage** provides improved functionality for astronomical image display, expecially for the analysis of data from raster observations. (Section 14.4.7).
- The new tool **xcvf** provides an interface to the spectral analysis package ISAP (Section 14.2.2).
- Inclusion of algorithms to **treat faint sources for circular variable filter (CVF) observations** (Section 20.6.2).
- **No-blocking widgets** were introduced so command-line and widgets can be used in parallel.

- **Help functionality** was improved (Section 2.3.2).
- The automatic slicer **spdtoscd** now warns about **saturated pixels**. The new option *method='nomode'* slices only according to h/w changes of ISOCAM and ignores s/w settings like the obs-mode. (Section 19.1).
- High level colour correction routine **corr_colour** is available now (Section 20.12.3).
- The **naming convention for theoretical PSFs was rationalized** (Section 15.4.1).
- **isocont** handles now **IRAF** FITS files and has the additional options *cut1, cut2 and thick.*
- **CIA's start-up files were rationalized** (Section 2.3.4).
- **Obsolete algorithms for transient correction and deglitching were removed.**

Appendix G

Upgrading old CIA structures

Structures created in previous versions of CIA will need some upgrading to be 100% compatible with CIA 3.0.

G.1 Upgrading CIA 2.0 structures

- Since the MAR98 version of CIA there are unit tags in the PDSs: .CUBE_UNIT, .IMAGE_UNIT and .RASTER_UNIT. If you want to use **conv_flux** on a *BS* PDS or *raster* PDS, then you will need to add .RASTER_UNIT to a pre-MAR98 PDS and fill it with 'ADU/gain/sec' (assuming you have not already manually changed the units). Likewise, for a *CVF* PDS and *general* PDS you will need to add .IMAGE_UNIT. This can be done with IDL's CREATE_STRUCT.

```
CIA> new_pds = create_struct( temporary( old_pds ), 'raster_unit',
CIA> 'ADU/gain/sec' )
```

  Use IDL's TEMPORARY to prevent making a duplicate copy of the structure while appending the new tag.

G.2 Upgrading CIA 1.0 structures

- There is a small bug in the .ASTR substructure of the CIA 1.0 CVF PDS: the type of the field .ASTR.CDELT is float instead of double. This problem will only manifest itself when you attempt to load a CIA 1.0 *CVF* PDS when you have a definition of the ASTR structure already in memory, eg. .ASTR in a CIA 3.0 *raster* PDS.

```
CIA> restore, 'old_cvf.xdr'
% RESTORE: Unable to restore structure, tag type disagrees with
existing definition: ASTR_STRUC.
% Temporary variables are still checked out - cleaning up...
```

  Though small, this problem is awkward to solve. Open a second CIA session in parallel with your first. In this new session load the *CVF* PDS as attempted above.

```
CIA> restore, 'old_cvf.xdr', /verb
% RESTORE: Portable (XDR) SAVE/RESTORE file.
% RESTORE: Save file written by LANDRIU@SAPI01, Thu Jun 12 17:11:05 1997.
% RESTORE: Restored variable: OLD_CVF.

CIA> help, old_cvf.astr, /str
** Structure ASTR_STRUC, 8 tags, length=96:
   CD              DOUBLE    Array[2, 2]
   CDELT           FLOAT     Array[2]
   CRPIX           FLOAT     Array[2]
   CRVAL           DOUBLE    Array[2]
   CTYPE           STRING    Array[2]
   LONGPOLE        FLOAT           180.000
   PROJP1          FLOAT          -1.00000
   PROJP2          FLOAT          -2.00000
```

This time there are no problems – no other ASTR definition exist in the new CIA session. Now export *old_cvf* as a FITS file and exit your second CIA session.

```
CIA> struct2fits, old_cvf, name='cvf.fits'

CIA> exit
I01[DELANEY]
```

Returning to your original CIA session, you can now load the FITS file into IDL memory as a *CVF* PDS. **fits2struct** will initialize a new *CVF* PDS, fill it with data from the old *CVF* PDS, and in so doing correct the type of .ASTR.CDELT.

```
CIA> fits2struct, 'cvf.fits', hdr, new_cvf

CIA> help, new_cvf.astr,/str
** Structure ASTR_STRUC, 8 tags, length=104:
   CD              DOUBLE    Array[2, 2]
   CDELT           DOUBLE    Array[2]
   CRPIX           FLOAT     Array[2]
   CRVAL           DOUBLE    Array[2]
   CTYPE           STRING    Array[2]
   LONGPOLE        FLOAT           180.000
   PROJP1          FLOAT          -1.00000
   PROJP2          FLOAT          -2.00000
```

- Due to an error in the original FITS astrometry definition which was propagated into the ASTROLIB astrometry routines[1] and the CIA routines, **roll_to_crota2** and **fits_header**, CIA 1.0 CVF.ASTR.CD was defined incorrectly and the CD matrix in FITS files created

[1] CIA is distributed with debugged versions of the ASTROLIB astrometry routines: **extast**, **putast**, **ad2xy**, **xy2ad**. The location of the CIA version of these routines is placed before the ASTROLIB location on IDL's !PATH (if CIA is installed correctly) to ensure that they are compiled first.

from CIA data structures is incorrect by a 180 degree rotation. Some programs are immune to this error, `saoimage` and `skyview` for example, because they use the FITS CDELT and CROTA2 keys to determine astrometry.

1. To update your *CVF* PDS first you need to know the roll of the spacecraft during the observation in question. To do this look in the IIPH delivered with your CVF observation data products and find the value of the key *instroll*. You can do this by simple listing the contents of the IIPH file with VMS's **type** or if you work in UNIX, you can use **more** or the CIA routine *readhdr*:

```
CIA> print, readhdr('iiph03800232.fits',key='instroll')
./ iiph03800232.fits 229.970
```

2. Now run the routine **struct_update** to fix the .ASTR.CD.

```
CIA> new_cvf = struct_update( old_cvf, roll=229.970 )
```

3. You may also want to correct FITS files you have created from CIA 1.0 *CVF* PDS. To do this you will need to re-export your files – see Section 18.

Appendix H

Reporting problems and suggestions

H.1 Problems with CIA software

If you encounter problems with CIA, or indeed if you have suggestions for improving CIA, then please submit a Software Problem Report (SPR) or a Software Change Request (SCR). Templates for both can be found in the following sections or in the */doc* subdirectory of the CIA installation as *spr.txt* and *scr.txt*.

Fill in the appropriate template and e-mail it to:

cia@iso.vilspa.esa.es

Please give adequate information to allow for the conditions which caused your problem to be reproduced. Keeping a journal (IDL's JOURNAL command) should help. Also, please keep your data and be prepared to make it available to us – it may be required to reproduce and correct the reported bug.

Please keep in mind that all efforts on CIA are manpower limited. This means that we might be unable to fix some smaller bugs or implement new functionalities, even if they are considered very desirable by all parties. CIA is delivered as source code, so if you have the necessary IDL knowledge and can fix the bug (or implement the required additional functionality) on your own, then please furnish us with the modifications. By doing so you will not only help us, but the CIA community as a whole.

H.1.1 Template for a Software Problem Report

```
ISOCAM INTERACTIVE ANALYSIS SOFTWARE PROBLEM REPORT

NUMBER:


SPR TITLE:

ORIGINATOR:


ISSUE DATE:


VERSION:
```

```
ENVIRONMENT:

PRIORITY:

PROBLEM DESCRIPTION:

ANALYST:

ANALYSIS DATE:

PROBLEM ANALYSIS:

RECOMMENDED SOLUTION:

ITEMS TO BE CHANGED:

TESTS TO BE RUN:

ESTIMATED EFFORT:

IMPLEMENTOR:

ASSIGNMENT DATE:

ITEMS CHANGED:

TESTS DONE:

ACTUAL EFFORT:

BOARD DECISION:

DATE:

STATUS:

CLOSING DATE:

FIX DELIVERY:

COMMENTS:
```

H.1.2 Template for a Software Change Request

```
ISOCAM INTERACTIVE ANALYSIS SOFTWARE CHANGE REQUEST

NUMBER:


SCR TITLE:

ORIGINATOR:
```

```
ISSUE DATE:

PRIORITY:

REASON FOR CHANGE:

CHANGE DESCRIPTION:

ANALYST:

ANALYSIS DATE:

PROBLEM ANALYSIS:

RECOMMENDED SOLUTION:

ITEMS TO BE CHANGED:

TESTS TO BE RUN:

ESTIMATED EFFORT:

IMPLEMENTOR:

ASSIGNMENT DATE:

ITEMS CHANGED:

TESTS DONE:

ACTUAL EFFORT:

BOARD DECISION:

DATE:

STATUS:

CLOSING DATE:

FIX DELIVERY:

COMMENTS:
```

H.2 Comments on this document

Comments on the *CIA User's Manual* and/or suggestions for improvements are always welcome. They may be submitted to:

cia@iso.vilspa.esa.es

Appendix I

Technical reports

Abergel A. et al., 1996, *IAS model for ISOCAM LW transient correction*

Altieri B. et al., 1998, *CAM Calibration Explanatory document*

Altieri B. et al., 1998, *ISOCAM Faint Source Report*

Aussel H., 1998, *ISOCAM LW Channel Field of View Distortion*

Aussel H., 1996, *ISOCAM Data Preparation with X_SLICER*, v2.1

Biviano A., 1998, *ISOCAM Calibration Error Budget Report*

Chanial P. and Gastaud R., 2000, *Ximage manual*

Claret A., 1996, *ISOCAM Data Analysis with X_CIA*, v2.2

Claret A. et al., 1998, *A Learning Guide for ISOCONT*, v2.0

Claret A. and Dzitko H., 1998, *ISOCAM Glitch Library*

Galais P. and Boulade O., 1998, *Report on Trend Analysis of CAM daily Calibration*

Gastaud R., 1999, *Technical Note on the error of raster in the software CIA*

Leech K. and Pollock A.M.T., 2000, *ISO Satellite and Data Handbook*, ESA document SAI/99-082/DC

Moneti A. et al., 1997, *Reference Wavelengths for ISO: CAM and PHOT filters*

Roman P. and Ott S., 1999, *Report on the behaviour of ISOCAM LW Darks*

Sauvage M., 1996, *Angles and ISOCAM-LW*, v2.0

Sauvage M. et al., 1999, **XPHOT** – *a crude photometric package for CIA*, v3.0

Sauvage M., 2000, *An Introduction to* **SLICE** *inside CIA*, V0.9

Saxton R., 1999, *ISO Data Product Document*, ESA document ISO-SSD-9111A

Siebenmorgen R. et al., 1996, *Addendum to the ISOCAM Observer's Manual*

Siebenmorgen R. et al., 1999, *ISOCAM Handbook*, ESA Document SAI/99–57/DC

Siebenmorgen R., 1999, Polarisation Observations with ISOCAM, ESA Document

Tran D. and Gastaud R., 2000, *Report on ISOCAM CVF faint source photometry*, in preparation

Bibliography

Abergel A. et al., 1998, *Transient Behaviour of LW Channel of ISOCAM*, Experimental Astronomy, accepted

Biviano A. et al., 1998, *ISOCAM Dark Current Calibration Report*, Expermental Astonomy, accepted

Biviano A. et al., 1998, *ISOCAM Flat-field Calibration Report*, Expermental Astonomy, accepted

Biviano A. et al., 1998, *ISOCAM CVF Calibration Report*, Expermental Astonomy, accepted

Blommaert J., 1998, *ISOCAM Photometry Report*, Expermental Astonomy, accepted

Boulade O. and Galais P., 1998, *The ISOCAM Detectors: An Overview*, Expermental Astonomy, accepted

Claret A. et al., 1998, *Glitch Effects in ISOCAM Detectors*, Expermental Astonomy, accepted

Coulais A. and Abergel A., 2000, *Transient correction of the LW-ISOCAM data for low contrasted illumination*, A&A, in press

Dzitko H. et al., 1998, *Cosmic Ray effects on the ISOCAM LW detector*,Expermental Astonomy, accepted

Landsman W.B., 1995, *The IDL Astronomy User's Library*, in: Astronomical Data Analysis Software and Systems IV, ed. R.A. Shaw, H.E. Payne, J.J.E. Hayes, ASP Conference Series Vol. 77, p. 437

Miville-Deschenes M.A., et al., *Optimizing ISOCAM data processing using spatial redundany*, A&A, accepted

NOST, 1983, *Definition of the Flexible Image Transport System*, NASA/Science Office of Standards and Technology, NOST standard 100-1.0, NASA Goddard Space Flight Center

Okumura K., 1998, *ISOCAM PSF Report*, Expermental Astonomy, accepted

Okumura K. et al., 1998, *Ghosts in ISOCAM Images*, Expermental Astonomy, accepted

Ott S. et al, 1997, *Design and Implementation of CIA, the ISOCAM Interactive Analysis System*, ASP Conference Series, Vol. 125

Ott S. et al., 1998, *Data Analysis with ISOCAM Interactive Analysis System – preparing for the Future*, ASP Conference Series, Vol. 145

Pantin E. and Starck J-L., 1996, *Deconvolution of Astronomical Images using the Multiresolution Maximum Entropy Method*, Astron. Astrophys. Suppl. Ser., 118, 575-585

Starck J-L., Murtagh F., Bijaoui, A., 1998, *Image Processing and Data Analysis: The Multiscale Approach*, Cambridge University Press

Tiphene, D. et al., *Modelling transient effects in the IR array of the short wavelength channel of ISOCAM, the camera onboard the ISO satellite*, Expermental Astronomy, accepted

ISOCAM

M6 SW Filters
1 turn: 360 steps

SW 8	*Brα 4*	6
Dark SW 6		13
SW 6	*L 3.5 - 4*	21
Dark SW 2		28
SW 2	*PAH 3.3*	36
Dark SW 7		43
SW 7	*H_2O 3.1*	51
Dark SW 4		58
SW 4	*2.5 - 3*	66
Dark SW 1		73
SW 1	*3 - 4*	81
Dark SW 3		88
SW 3	*LW1 4 - 5*	96
Dark SW 5		103
SW 5	*3 - 5.7*	111
CVF	*2.273* *5.122*	129 242
MA FSW Aligment		261
Hole		276
Aligment		291
Dark SW 12		298
SW 12		306
Dark SW 10		313
SW 10	*CO 4.7*	321
Dark SW 11		328
SW 11	*CO_2 4.26*	336
Dark SW 8		343
SW 9	*3.8 - 4.2*	351

M2 SELECTION
1 turn: 240 steps

ISO Mirror	0
Aligment	40
LW BB Sphere	64
LW Large Mirror	88
SW Large Mirror	112
SW BB Sphere	136
SW Small Mirror	200
LW Small Mirror	220

M4 LW Filters
1 turn: 360 steps

MA LSW Aligment		65
Hole		80
LW 2	*5 - 8.5*	95
Dark LW 2		102
LW 10	*IRAS 8-15*	110
LW 3	*12 - 18*	125
Dark LW 3		132
Dark LW 9		132
LW 9	*CO_2 14-16*	140
CVF #2	*17.34* *8.782*	150 235
LW 8	*PAH 10.7-12*	245
Dark LW 8		252
LW 7	*SiO_2 8.5-11*	260
Dark LW 7		267
LW 6	*7 - 8.5*	275
Dark LW 6		282
LW 5	*6.5 - 7*	290
Dark LW 5		297
LW 4	*5.5 - 6.5*	305
Dark LW 4		312
Dark LW 1		312
LW 1	*SW3 4-5*	320
CVF #1	*9.583* *4.956*	330 54

M5 Lens SW
1 turn: 480 steps

Pupil	24
L. Si 6	120
L. Si 3	208
L. Si 12	300
Hole	360
MA LSW Aligment	384
L. Si 1.5	432

M1 ENTRANCE
1 turn: 360 steps

Polar. 2	7
Dark	45
Polar 3.	83
Tr. M. A.	142
Park position	180
Polar. 1	225
M.A.E. Aligment	273
Hole	308

M3 Lens LW
1 turn: 480 steps

L. Ge 12	60
Hole	120
MA LLW Aligment	144
L. Ge 1.5	192
Pupil	264
L. Ge 6	360
L. Ge 3	448