IRAF notes for Observational Astronomy

Søren S. Larsen

November 29, 2016

Contents

Chapter 1

Introduction to IRAF

The handling of astronomical datasets requires specialised software. Two general-purpose packages in common use are MIDAS (the Munich Image Data Analysis System), developed by the European Southern Observatory, and IRAF (the Image Reduction and Analysis Facility), developed by the National Optical Astronomy Observatories in the US. In addition, many instruments are now so complex that dedicated *pipelines* are available for the reduction of the observations. However, once the initial pipeline processing has been done, one often returns to general-purpose packages like IRAF or MIDAS for further analysis. Both run in the UNIX environment, and have similar functionality. In this course we will use IRAF for basic reduction of CCD images.

As you might expect, IRAF is a very large package with many different functions. We will only scratch the surface in this course. You may find the user interface a bit old-fashioned, which is not surprising given that the first versions of IRAF date back to the late 1980s. However, it still remains the analysis package of choice for many astronomers across the world, and it has been updated over the years with many new tasks. For example, there is a dedicated set of tasks for the reduction and analysis of data from the Hubble Space Telescope. Similarly, the GEMINI observatory has developed its own IRAF tasks to handle data from the specific instruments available at that observatory. If you are going to work with astronomical data, it is very likely that you will encounter IRAF sooner or later.

For more information, we suggest you browse through the *Beginner's Guide to Using IRAF* by Jeanette Barnes, available via the web at: `http://iraf.net/irafdocs/beguide.pdf`. In the rest of these notes we will simply refer to this document as the Beginner's Guide. It is a general introduction to IRAF which covers the basic concepts, although parts of it are now largely obsolete (e.g. the detailed discussion of how to read tapes).

1.1 Getting started

These notes cover a few additional issues specific to running IRAF as part of the OA course. They are designed such that you can work through them while sitting in front of a computer. Throughout these notes we will use the notation

```
> command
```

to denote a command typed at the UNIX prompt, and

`ecl> command`

for commands typed at the IRAF prompt.

IRAF is installed on the workstations in the computer rooms in the Huygensgebouw. You should be able to log in on these with your student account. IRAF itself, along with lots of documentation, is available at `http://iraf.noao.edu` and `http://iraf.net`. Those are good places to look for more information.
We are ready to start.

- First, you need to boot your workstation in Linux mode.
- Then log in with your usual account and password.
- In order to display graphics correctly, it is important to start IRAF from an *xterm* window instead of the default terminal application in Ubuntu Linux. You can get an xterm window by typing the following command:

  ```
  > xterm -sb -sl 600 &
  ```

Before you can use IRAF, you need to go through the following steps:

- Make a directory called `iraf` in your home directory and go to this directory:

  ```
  > mkdir iraf
  > cd iraf
  ```

- Then call the `mkiraf` command, which generates a configuration file in the `iraf` directory:

  ```
  > mkiraf
  -- creating a new uparm directory
  Terminal types: xgterm,xterm,gterm,vt640,vt100,etc.
  Enter terminal type:
  ```

  For the terminal type, enter `xterm24` and press `Return`

Note that these steps need only be carried out once.
We are now ready to start IRAF:

- Start IRAF by typing the command in the xterm window:

  ```
  > cl
  ```

  Note that you always have to start the `cl` from the iraf directory.

1.2 The command language, CL

As noted above, the default user interface of IRAF, the *command language*, that you start with the command `cl`, is a bit old-fashioned. It is basically a command prompt similar to the one you may be familiar with from the UNIX/Linux environment, although the syntax differs somewhat (this is described in the Beginner's Guide). Remember that IRAF has been around for more than 25 years – when it was first developed, many astronomers just had a simple text-based VT100 terminal in their office without fancy graphics capabilities. So it made perfect sense to have a low-tech text-based user interface. There have been some attempts to "embed" the `cl` in a graphical interface but these haven't really been successful.

Recent versions of IRAF have a slightly more modern version of the command interpreter, `ecl` (enhanced command language). For example, the `ecl` has an arrow-key history function, which the original `cl` lacked.

1.3 Image file formats

IRAF recognises many different astronomical file formats. The Beginner's Guide describes the 'OIF' file format, in which an image is split into two parts: a header file with the extension"`.imh`" and a pixel file with the extension "`.pix`" containing the actual image data. Most astronomical images are nowadays stored in the FITS format (Flexible Image Transport System). IRAF's internal handling of the OIF is supposed to be more efficient than for FITS images, but in practice this is hardly ever noticeable. If you always specify the "`.fits`" extension explicitly when storing an image, you can be sure that it will automatically be stored in FITS format. This will make it easier to exchange data between IRAF and other software.

- IRAF has a "built-in" test image with the somewhat cryptic name `dev$pix`. You can copy this to a FITS file in your own directory using the `imcopy` task:

  ```
  ecl> imcopy dev$pix test.fits
  ```

  This will be useful in the following section.

1.4 Displaying images

You may be surprised to learn that an image display is not an integrated part of IRAF! But remember, back in those "good old days" with the VT100 terminals, graphics required special (expensive) hardware. So in IRAF the display of images remains delegated to external software, although the `cl` is designed to interact with such image display software.

The Beginner's Guide describes the IMTOOL and SAOImage programmes. These are now outdated, and a more modern alternative is the `ds9` utility. Actually, `ds9` is quite a useful tool all by itself and allows, among other things, to overlay catalogs on an image and even download images from various web services.

Start `ds9` by typing

```
> ds9 &
```

at the UNIX prompt (or `ecl> !ds9 &` within IRAF). Be careful to start only one `ds9` at a time, otherwise IRAF will get confused!

There are two ways to display an image in `ds9`:

- Within IRAF, you can use the `display` command:

  ```
  ecl> display test.fits 1
  ```

  As you can see, the `display` task takes two parameters: the name of the image to be displayed, and a number. The number indicates a buffer number in which the image should be displayed. `ds9` has 16 such buffers, and allows you to blink one image against another or show them next to each other. IRAF will convert the actual image data into an 8-bit bitmap and send this to `ds9`. Since astronomical data are generally stored with 16-bit or higher precision, this entails some loss of information. In fact, upon calling the `display` command, you will see IRAF producing a message like:

  ```
  ecl> display test.fits 1
  z1=35. z2=346.0218
  ```

  where the `z1` and `z2` numbers indicate the data values in the image that will be mapped to the lowest and highest intensity (black/white). Information outside this range will be lost. You can override this automatic scaling by specifying additional parameters when calling `display`:

  ```
  ecl> displ test.fits 2 zr- zs- z1=0 z2=2000
  ```

  In this case, the image will appear darker overall, but you can see details in the high intensity regions near the nucleus better. These will appear "burned out" with the default scaling. Note that we have now displayed the image in buffer number two, so you can use the "blink" function in `ds9` to compare the two scalings (under the "frame" menu).

- You can load the image directly into `ds9` via the `file` menu. In this case, there is no loss of information. You can use the `scale` menu in `ds9` to edit the display options. This way of displaying the image has the additional advantage that information about "World Coordinate Systems" in the image header will be read by `ds9` so that coordinates can be displayed in physical units (right ascension, declination). You can still load images into multiple buffers.

Independently of how the image is displayed, you can adjust the contrast by moving the mouse across the `ds9` display area while holding down the right mouse button. Moving the cursor up or down will decrease/increase the contrast, while moving the cursor left/right will increase/decrease the brightness. While this is useful for a quick-look inspection, it is in general better to adjust the scaling via the `scale` menu in `ds9` or via the `z1/z2` parameters in IRAF.

1.5 Tasks and packages

Similar to the UNIX command prompt, the IRAF command interpreter itself doesn't do much. Most of the functionality in IRAF is contained in various *tasks*, which are organised into *packages*. We have already encountered one such task, `display`. Like most other tasks, it has required parameters (in this case, image name and buffer number) as well as optional ones.

1.5.1 Loading packages

In order to use a task, the package in which it belongs must first be loaded. Some packages are automatically loaded when starting IRAF (this can be defined in the `login.cl` or `loginuser.cl` files in the iraf start-up directory). Others can be loaded simply by typing the name of the package, e.g.

```
ecl> images
```

will load the `images` package. Upon loading a package, the `cl` prompt will change to the two first letters of the name of the most recently loaded package (`ecl` prints the whole name). The `pyraf` prompt will remain the same. In either case, a list of the tasks contained within this package is displayed:

```
images> images
      imcoords.   imfit.      immatch.    tv.
      imfilter.   imgeom.     imutil.
images>
```

Note that the command prompt has now changed to the name of the most recently loaded package!

A package can contain other packages as well as tasks. Packages can be recognised since their names end with '.', so in the example above all the new items are packages. We can then load, for example, the `tv` package:

```
images> tv
      display     iis.        imedit      imexamine   tvmark      wcslab
tv>
```

and we find that it contains the tasks `display`, `imedit`, etc., in addition to yet another package, `iis`. Actually, the `tv` package is among those that are automatically loaded, which is why the `display` task was already accessible before we manually (re-)loaded the package.

It is not necessary to type the full name of a task or package. Any unique abbreviation can be used. Here, `im` will not be unique, but we could load the `imutil` package, for example, by just typing `imut`.

The most recently loaded package can be unloaded by typing

```
tv> bye
```

The tasks in the package will then no longer be available. Note, however, that packages are different in this sense than UNIX directories. In the example above we could have loaded the `immatch` package after loading the `tv` package without any need to say `bye` to the `tv` package. In the early days of IRAF usage, loading only a minimum of packages could help optimize the use of computer memory, but on modern systems there is no real need to unload packages once they have been loaded.

1.5.2 More about tasks

To find out more about how the `display` task works, we can use the `help` function:

```
ecl> help display
```

and IRAF responds by displaying the help text for `display`:

```
DISPLAY (Mar97)                 images.tv                DISPLAY (Mar97)



NAME
    display -- Load and display images in an image display


USAGE
    display image frame


PARAMETERS

    image
        Image to be loaded.

    frame
        Display frame to be loaded.


    bpmask = "BPM"
        Bad pixel mask.  The bad pixel  mask  is  used  to  exclude  bad
[q=quit,d=downhalf,f|sp=downfull,j|cr=downline,N=next]
```

Among other things, the top line tells us that the `display` task is found in the `tv` package, which is a sub-package of the `images` package. Of course, we knew this already, but in general this is useful information, since the `help` function works also for tasks in packages that haven't been loaded. By browsing through the help text, we find out which parameters the task accepts and what their function is.

To see all the parameters accepted by the `display` task, we can use the `lparam` command:

```
ecl> lpar display
        image = "dev$pix"       image to be displayed
        frame = 1               frame to be written into
      (bpmask = "BPM")          bad pixel mask
   (bpdisplay = "none")         bad pixel display (none|overlay|interpolate)
    (bpcolors = "red")          bad pixel colors
     (overlay = "")             overlay mask
     (ocolors = "green")        overlay colors
       (erase = yes)            erase frame
(border_erase = no)             erase unfilled area of window
(select_frame = yes)            display frame being loaded
      (repeat = no)             repeat previous display parameters
        (fill = no)             scale image to fit display window
      (zscale = yes)            display range of greylevels near median
    (contrast = 0.25)           contrast adjustment for zscale algorithm
      (zrange = yes)            display full image intensity range
       (zmask = "")             sample mask
     (nsample = 1000)           maximum number of sample pixels to use
     (xcenter = 0.5)            display window horizontal center
     (ycenter = 0.5)            display window vertical center
       (xsize = 1.)             display window horizontal size
       (ysize = 1.)             display window vertical size
        (xmag = 1.)             display window horizontal magnification
        (ymag = 1.)             display window vertical magnification
       (order = 0)              spatial interpolator order (0=replicate, 1=line
          (z1 = )               minimum greylevel to be displayed
          (z2 = )               maximum greylevel to be displayed
      (ztrans = "linear")       greylevel transformation (linear|log|none|user)
     (lutfile = "")             file containing user defined look up table
        (mode = "al")
ecl>
```

Note that all except the two first parameters appear in parentheses - this means that the first parameters are required (IRAF will ask for them if they are not specified), while the rest do not need to be specified. If they aren't specified, the default values listed by `lpar` will be used.

To change the default parameter values, we can use the `eparam` command. In the `ecl`, you get a text-based interface for editing the task parameters:

```
ecl> epar display
                                    I R A F
                        Image Reduction and Analysis Facility
PACKAGE = tv
   TASK = display
```

```
image   =              dev$pix  image to be displayed
frame   =                    1  frame to be written into
(bpmask =                  BPM) bad pixel mask
(bpdispl=                 none) bad pixel display (none|overlay|interpolate)
(bpcolor=                  red) bad pixel colors
(overlay=                    ) overlay mask
(ocolors=                green) overlay colors
(erase  =                  yes) erase frame
(border_=                   no) erase unfilled area of window
(select_=                  yes) display frame being loaded
(repeat =                   no) repeat previous display parameters
(fill   =                   no) scale image to fit display window
(zscale =                  yes) display range of greylevels near median
(contras=                 0.25) contrast adjustment for zscale algorithm
(zrange =                  yes) display full image intensity range
(zmask  =                    ) sample mask
(nsample=                 1000) maximum number of sample pixels to use
More
                                                          ESC-? for HELP
```

You can then move the cursor up and down with the arrow keys and enter other values. When you are done, exit by typing ":wq" (same syntax as the vi editor).

Like task names, parameter names can also be abbreviated on the command line - the zrange and zscale parameters correspond to the zr and zs arguments in the example in Sec. 1.4. These are examples of boolean parameters, that can take the values yes or no. The notation zr- on the command line is shorthand for zrange=no.

- Try setting the ztrans parameter in display to log. Then redisplay the image.

1.6 Interacting with the image display

Many tasks in IRAF can interact graphically with the user. As an example, let us take a look at the imexamine task which is useful for visually inspecting an image and carrying out a few basic measurements. Here we just give a few examples of things you can do with imexamine. You can read more about this task in the Beginner's Guide and in the built-in help.

If called without any parameters, imexamine will work on the image that is currently displayed in ds9. The task can also be called with an image name (or a list of image names) as parameter; in this case the (first) image will be loaded into the display. Hence, you may examine the image dev$pix like this:

```
ecl> display dev$pix 1
z1=35. z2=346.0218
ecl> imexam
```

or like this:

```
ecl> imexam dev$pix
```

Note that, in both cases, we have abbreviated the command name to `imexam`, which is still unique.

You will see that the cursor now has moved from the terminal window to `ds9`. `imexam` is now waiting for further commands (depending on how your window manager is configured, you may still have to click on the `ds9` window to make it active).

- If you move the cursor to a feature in the image and hit the "a" key, `imexamine` will measure the counts in a circular aperture centered on the feature and calculate a few statistics including the centroid, counts, ellipticity and the FWHM (full width at half maximum).
- You can get a plot of the radial profile of the source by using the "r" key.
- "s" will produce a surface plot of the region of the image near the cursor
- To get a list of all the functions, hit the "?" key. Note that the cursor now moves back to the terminal window (again, you may have to click on it). After having read the help message, type "q" to get back to the `ds9` window.
- You can play around with the different commands. When you are done, hit "q" in the `ds9` window and you will get the IRAF command prompt back.

These examples illustrate a common feature of many tasks in IRAF: there is an *image cursor* that allows you to interact with the image display and a *graphics cursor* that allows you to interact with the graphics window.

1.7 Summary

In this lesson you have learned:

- Basic steps to get get started using IRAF
- The concepts of *packages* and *tasks*
- How to load a package
- How to view and edit the parameters of a task
- How to use the *help* function in IRAF
- How to display an image and do some simple analysis

Chapter 2

Basic reduction of CCD images

The raw images delivered by a CCD camera generally contain various instrumental artefacts that need to be removed before further analysis can be done. To understand which steps are required, it is useful to recall a few basic facts about CCD data.

2.1 Removal of detector artefacts

A CCD detector is divided into a number of pixels, each of which is capable of holding a number of electrons (on order 10^5). CCD detectors used for astronomy now typically have 2048×2048 or 2048×4096 pixels, but can be mosaiced together to larger formats. When a new exposure is initiated, the whole detector is first cleared and electrons are then released by incoming photons during the exposure and accumulated in each pixel. The number of electrons N_e is, to a very high degree of accuracy, proportional to the number of incoming photons N_p:

$$N_e = \text{Q.E.} \times N_p \tag{2.1}$$

where the proportionality factor is called the *Quantum Efficiency*. For modern CCD detectors this can be as high as 90%.

When the exposure time (typically a few minutes) has elapsed, the pixels are "read out" one by one. The charge in each pixel is converted to a voltage, which in turn is amplified and converted to a 16-bit number by an analogue-to-digital (A/D) converter. The resulting "Digital Number" is again linearly related to the number of electrons accumulated during the exposure:

$$\text{D.N.} = N_e/\text{Gain} + \text{Bias} \tag{2.2}$$

Note that a *Bias* level is added during the read-out process. The use of the term "Gain" in Eq. (2.2) is unfortunately somewhat inconsistent with the standard usage e.g. in electronics, and a more proper term would be "inverse Gain". However, we are using the term Gain here in the same way it is used in many IRAF tasks, i.e. number of electrons per D.N.

It should be mentioned that electrons may also be released due to *dark current* in the CCD. The dark current is very strongly temperature sensitive, and is typically reduced to negligible

levels by cooling the detector with liquid nitrogen in professional CCD cameras. In the thermo-electrically cooled CCD cameras used at the telescopes in Nijmegen, you will still see some dark current. On a cold night, the CCDs can be cooled to a lower temperature, and you may notice that this indeed makes a significant difference. Since the dark current can vary quite strongly from one pixel to another, the best way to correct for it is to obtain a "dark exposure" with the same duration as the science exposure, but with the camera shutter closed. As long as the CCD temperature is maintained at a constant level, this can be conveniently done during the day in order to not waste observing time at night.

In reality, the sensitivity of the CCD can vary somewhat across the chip. For the purpose of the present discussion we may view this as due to variations in the quantum efficiency, although there can also be other reasons (e.g. dust particles on the CCD or on optical surfaces in front of it, or different parts of the chip being read out through different amplifiers with slightly different gains). This variation can be accounted for by observing a uniformly illuminated surface, such as the twilight sky or the inside of the telescope dome. Such an image is called a *flat-field* image. Once the Bias level has been subtracted from the science exposure, the sensitivity variations can be corrected by division with the (also bias-subtracted) flat-field image. In order to keep the Gain of the processed CCD image close to that of the raw data, the flat-field image is typically normalised to a mean value of unity.

We can then summarise the main steps involved in the initial reduction of a CCD image as follows:

- Subtract the Bias level (or dark exposure) from the science image and the flat-field exposure. *For data obtained with the CCD cameras in Nijmegen, this step may have been carried out automatically during the observation.*
- Normalise the bias-subtracted flat-field exposure to a mean value of 1
- Divide the bias-subtracted science image by the normalised flat-field.

The result of this process is then an image in which the digital numbers are directly proportional to the flux incident upon each pixel during the exposure.

2.2 Further processing

Because astronomical targets are faint, the exposure times required to obtain useful measurements are often long. Long integrations are often split into several shorter ones, for a number of reasons:

- CCDs are also sensitive to cosmic ray (CR) hits and during long exposures, a significant fraction of the pixels in a CCD image can be affected by CRs. This can prevent accurate measurements of the sources of interest. Since the CR hits are random events, they will affect different pixels if another exposure is made, and can therefore be eliminated by splitting long exposures into several shorter ones.
- Telescope tracking errors: Although best efforts are made to make the telescope track objects accurately as they move across the sky, this may occasionally fail. This might

render a long exposure useless, or at least lead to significant degradation of image quality. A shorter exposure can simply be rejected and repeated.

- It is often the case that several pixels, or columns of pixels, in a CCD detector do not provide useful data. These pixels may simply be "dead", or suffer from much higher than average dark current ("hot" pixels) so that they saturate even in a relatively short exposure. By including a small offset of the telescope pointing between multiple exposures, such bad pixels can be eliminated in the same way as CR hits.

- In long exposures the full-well capacity of a pixel may be exceeded for bright objects, leading to saturation and loss of information. Again, this can be avoided by splitting a long exposure into several shorter ones.

- Having several exposures of a field allows monitoring of the photometric stability. If the number of counts for objects in the field differ widely from one exposure to another, this may be an indication of extinction variations (for example due to passing clouds).

- For images where the spatial resolution is dominated by the pixel size ("undersampled" images), an improvement in the resolution may be achieved by combining several images shifted by sub-pixel offsets. This technique, known as *drizzling*, is commonly used for HST images.

It is then clear that one often needs to combine several exposures to a single equivalent long exposure. It may seem that the most effective way to reject outliers such as cosmic ray hits or bad pixels is to median combine the images, but this will actually lead to a reduction in S/N of about 25% compared to simple averaging of the images. Of course, for a simple average the bad pixels will not be eliminated, but this can be accomplished by using a sigma-clipping algorithm whereby pixels that deviate by more than a specified number of standard deviations from the mean are rejected.

If telescope offsets are performed, then each image will need to be shifted relative to a reference image before the combination is performed.

Flat-fields are also typically obtained by combining many short exposures. Here, the goals are (at least) two-fold:

- A flat-field image, like any other image, will be subject to Poissonian noise. The Poisson noise in the flat-field image should be small enough that it does not lead to a significant degradation of the science image. For a full-well capacity of 10^5 electrons, the Poissonian noise will be about 0.3% per pixel if the image is exposed to near the saturation limit. In practice, the response of a CCD often becomes non-linear near saturation, so it is wise to avoid getting close to saturation. A more realistic noise level may then be $\sim$ 1% for a single flat-field image. For some applications this may be comparable to the photometric accuracy aimed at, so it is desirable to beat down the Poisson noise by combining several such exposures.

- Sky flats may contain stars, even if an effort is made to point the telescope at a "blank" patch of sky. In this case, the stars represent an undesirable contamination that must

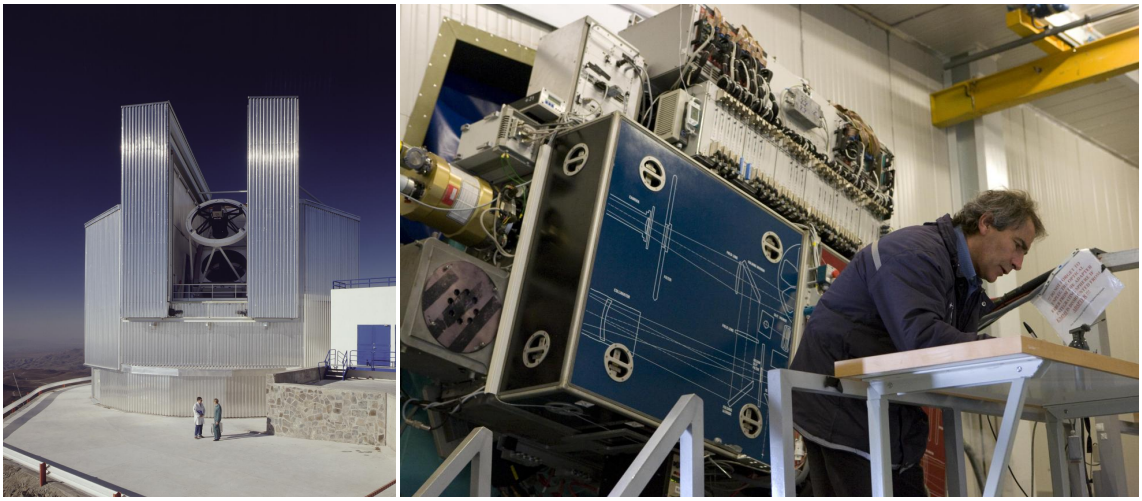

Figure 2.1: The ESO 3.5 m New Technology Telescope (left) and the EMMI instrument (right)

be eliminated. Similar to cosmic rays or bad pixels, this can be achieved by combining several exposures, pointed at different regions of the sky.

2.3 Reduction of CCD images in IRAF

The reduction steps described above can all be carried out within IRAF. In the following, we will reduce a set of CCD images taken with the *ESO Multi-Mode Instrument* (EMMI) on the *New Technology Telescope* at La Silla Observatory in Chile. The data are V-band images of the asteroid 4829 Sergestus, which belongs to a group of asteroids known as the Jupiter Trojans. These are found near the Sun-Jupiter L4 and L5 Lagrange points.

What follows is an outline of the rudimentary steps required for the reduction of CCD images, intended as an illustration of how to carry out these steps in IRAF. In practice, procedures may be somewhat different, depending on the characteristics of the particular instrument you are working with. In a subsequent exercise, we will use data from the actual telescopes in Nijmegen. As usual, more extensive documentation is available at `http://www.iraf.net`, in particular *A User's Guide to CCD Reductions with IRAF* by Phil Massey (`http://iraf.net/irafdocs/ccduser3.pdf`).

2.3.1 EMMI

EMMI is a multi-purpose instrument designed for both imaging and spectroscopy. It was installed on the NTT in 1990, and until it was decommissioned in 2008 it was one of the main workhorses of the La Silla observatory. Fig. 2.1 shows the telescope in its enclosure (left) and EMMI itself (right), mounted at one of the Nasmyth foci of the NTT.

EMMI has two "arms" for observations from 300 to 500 nm ("blue arm") and 400 nm to

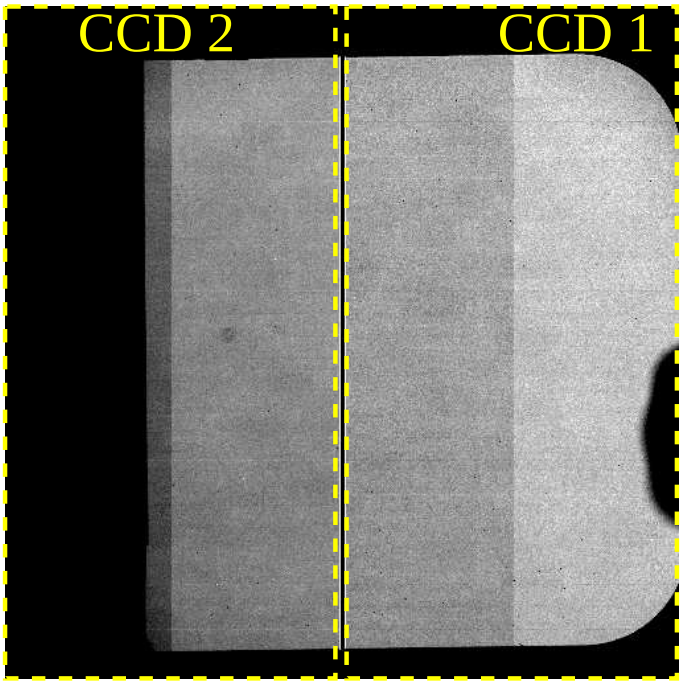

Figure 2.2: *V*-band flat-field image from EMMI

1000 nm ("red arm"). The data we will use here were taken with the red arm, which has two 2048×4096 pixel CCD detectors mounted in a mosaic, thus equivalent to a single 4096×4096 detector apart from a small gap between the two CCDs. The total field of view is $9.9' \times 9.1'$ and the pixel scale is $0.166''$ per pixel. Since such a small pixel scale is really only needed in the most excellent seeing conditions, the detectors are often *binned* by a factor of two during read-out so that each pixel in the final image really consists of 2×2 pixels on the detector. This saves time during the read-out process, and also makes the final image sizes smaller. The data used here are binned in this way.

The read-out noise is 9 e^- per pixel and the (inverse) gain is 1.25 e^- per D.N.

Fig. 2.2 shows a *V*-band flat-field image taken with EMMI. Data from the two CCD chips have been mosaiced together in a single FITS file so they can be viewed as a single image. Apart from the gap between the two detectors (the dark vertical strip in the centre of the image) you can also see that only about half of CCD #2 is illuminated, and that each chip is again divided into two halves with somewhat different mean count levels. This is because each CCD is read out through two separate amplifiers with slightly different gain and bias level.

2.3.2 Getting the data

First, make a directory for the data:

```
> mkdir OAdata
> cd OAdata
```

The data are available as a gzip'ed tar file from the course web page. It may be easier to get them with the UNIX `wget` command:

```
> wget http://www.astro.ru.nl/~slarsen/teaching/OA/data/Emmi.tar.gz
```

Then extract the data from the file:

```
> tar xvzf Emmi.tar.gz
```

You will see that three types of images are included:

- `ser_v.fits`: The science image itself
- `skyflat_v*.fits`: Twilight flat-field images
- `bias_*.fits`: Bias images. These are very short exposures taken with the shutter closed, which serve to indicate the bias levels of the CCD detectors (slightly different for each chip and A/D converter).

Use IRAF to look at the images. 4829 Ser is the object at $(x, y) = (1375, 1043)$ in the science image.

- Start IRAF (incl. `ds9`) as discussed in Chapter 1. Remember to "cd" to the `iraf` directory. You do *not* have to run `mkiraf` again.
- Within IRAF, change to the data directory you just created:

  ```
  ecl> cd ../OAdata
  ```

- Display the science image and use the `imexamine` task to measure the FWHM of some stars in the images of 4829 Ser. *What is the corresponding seeing in arcsec?* Be careful to avoid bright objects, which might be saturated! Since these are 16-bit images, the maximum possible pixel value is 65535, so if this occurs then the dynamic range has been exceeded. You can check the maximum count level under the cursor by pressing "m" in imexam. It may also be helpful to look at the radial profile of an object with the "r" key, or get a surface plot with the "s" key.

2.3.3 Preparation of calibration images

Master bias image

The first step is to prepare a master bias image, since this will need to be subtracted from both the flat-field and science images. You can see that 10 bias images are included. We will average combine these to a single master image, but first it is good to check that they are all ok (e.g. that no science or flat-field images have been mislabelled as "bias" images).

- Make a text file with a list of all the bias images. This can be done with the UNIX "ls" command:

```
ecl> !ls bias_*.fits > bias.txt
```

(note that UNIX commands can be called within IRAF by putting an "!" mark in front of them).

- Now use the `imstatistics` command to list some basic statistics for the bias images:

  ```
  ecl> imstat @bias.txt
  ```

  Many tasks in IRAF can work with *lists* of images. The `imstat` call above is an example of how this works: instead of giving a single image name, the name of a text file with a list of images can be given, preceded by an "@" character.

- Verify that all the bias images have (about) the same mean level and standard deviation.

We are now ready to combine all the individual bias images to a single master bias image. For this purpose we will use the `imcombine` task in IRAF. The task has just two required parameters: a list of images to be combined, and the output image. However, there are many optional parameters, so let's first take a look at those:

```
ecl> lpar imcombine
```

The most relevant ones are:

- `combine`: average / median / sum. The value of this parameter determines how images are combined.

- `reject`: none / minmax / ccdclip / crreject / sigclip / avsigclip / pclip. This specifies how outlying pixels are rejected. For sigclip/avsigclip/pclip, the rejection of bad pixels is based on the standard deviation calculated from the actual pixel values. For ccdclip/crreject, the standard deviation is instead calculated based on Poisson statistics, using the information about the CCD gain and read-noise.

- `scale`: none / mode / median / mean / exposure / list / keyword. `imcombine` can apply a scaling factor to each image before they are combined. The value of the `scale` parameter specifies how this is done.

- `zero`: none / mode / median / mean / list / keyword. Similarly, an offset can be applied to each image.

- `weight`: none / mode / median / mean / exposure / list / keyword. For average combination, this parameter specifies how the weighting of each image is calculated.

- `statsec`: This parameter specifies the region of the images used for the calculation of the mode/median/mean used in the scaling, zero-point offsets and weights. It is given in the usual notation for image sections: `statsec = [x1:x2,y1:y2]`. If left blank, the whole image is used.

- `lsigma / hsigma`: The limits below and above the mean where outlying pixels are rejected

- `rdnoise / gain / snoise`: CCD read noise and gain parameters, used if `reject` is set to `ccdclip` or `crreject`.

For a full discussion of these parameters and others, see the built-in help for `imcombine`.

Here we suggest to combine the bias images using `combine=average` and `reject=crreject`. Remember to set the relevant `rdnoise` and `gain` parameters (see Sect 2.3.1)! For the bias images, no image scaling or zero-point offsets are needed, so set `scale=none` and `zero=none`.

- Use the `epar` command to set the parameters for the `imcombine` task.

- Then produce the master bias by calling `imcombine`:

```
ecl> imcombine @bias.txt masterbias.fits
```

Subtract masterbias from flatfields

Now that we have a masterbias image, we can proceed to produce our flatfield images. Since the flatfield correction can be wavelength dependent, a separate flatfield is needed for each filter used for the observations.

We first need to subtract the `masterbias.fits` image from each of the flatfield images. For this we can use the `imarithmetic` task. Here we can again make use of IRAF's ability to handle lists of images, with an extra "trick":

- As for the bias images, make a text file with a list of all the raw sky flats:

```
ecl> !ls skyflat_??.fits > skyflat.txt
```

- Manually edit the `skyflat.txt` file and remove the `.fits` extension from each filename, so that the file looks as follows:

```
ecl> !cat skyflat.txt
skyflat_v1
skyflat_v2
skyflat_v3
```

(you could have skipped this last step by using a bit of extra UNIX trickery:

```
ecl> !ls skyflat_*.fits | sed s,.fits,, > skyflat.txt
```

)

- The nifty bit is now that by appending a string to the output *list* when calling `imarithmetic`, this string will be appended to each of the output `images`. We can therefore carry out the subtraction of the masterbias frame from all the skyflats in just one step:

  ```
  ecl> imarit @skyflat.txt//.fits - masterbias.fits @skyflat.txt//_b.fits
  ```

- We now have a bias-subtracted version of each skyflat with an extra "_b", e.g. `skyflat_v1_b.fits` is the bias-subtracted version of `skyflat_v1.fits`

Combine the bias subtracted flat-field images

We now need to combine the flat-field images. This can be done in the same way as for the bias images, with one exception: We need to let `imcombine` scale the individual images so that the rejection algorithm works properly. Here, the median is appropriate since we do not want the scaling to be affected by outlying pixels.

- Set the `scale` parameter to `median`
- Set the `statsec` parameter to an appropriate image section. We only want to use the illuminated part of the flatfields for scaling. Something like `statsec=[1100:1500,200:1800]` should be fine.
- Make a list of the bias-subtracted skyflats and call imcombine, i.e.

  ```
  ecl> !ls skyflat_*_b.fits > ff.lst
  ecl> imcombine @ff.lst ff.fits
  ```

  We are now left with the combined (but not yet normalised) flat-field image, `ff.fits`.

Normalise the combined flat-field

The (almost) last step before we are ready to reduce the science image is to normalise the flat-field.

- Use the `imstatistics` task to calculate the mean of the flatfield. Again, it is better to use only the illuminated fraction of each frame, e.g.

  ```
  ecl> imstat ff[500:1900,150:1800]
  ```

- Then use `imarithmetic` to divide the flatfield by the mean, e.g.

  ```
  ecl> imarith ff / 29850 masterflat.fits
  ```

Note that these examples are deliberately somewhat sloppy about specifying the `.fits` extension - IRAF will generally find the images even if the extension isn't specified (confusion may arise if there are two images where only the extension differs). However, when a new image is produced it is better to be explicit (although most installations will now have the default output format set to `.fits`).

Oh, one last thing: If you now display the normalised flatfield image you should see that the illuminated area has pixel values close to 1. The non-illuminated part will be close to, but not exactly, 0. The same will be the case for the science images, so in these non-illuminated parts the pixel values will fluctuate wildly in the final reduced images. This is quite undesirable, since one will often later use an automatic algorithm to identify sources for photometry in the data, and such an algorithm will be confused by these fluctuations. Fortunately, there is an easy way around it:

The default behaviour of `imarithmetic` is to set output pixel values to zero in case of division by zero. If we therefore set the non-illuminated regions of the flat-field images explicity to zero, these regions will also be zero in the final reduced images. This can be done with the `imreplace` task:

- Use `imreplace` to set pixel values below a certain threshold to 0. This can be controlled by the `upper` and `lower` parameters in `imreplace`. For example,

  ```
  ecl> imreplace image.fits 0.0 lower=INDEF upper=0.5
  ```

  This will leave pixels with values above 0.5 D.N. unaffected, while pixels below this value are set to 0.

Done!

2.3.4 Reduction of the science image

With all the preparatory work done, there are now just two remaining steps:

- Subtraction of the `masterbias.fits` frame from the science image
- Division by the normalised `masterflat` frame

Since there is only one science image, it's probably not really worth the effort of going through the process of generating image lists. Instead, you may just want to call `imarithmetic` manually.

- Reduce the science image by first subtracting the masterbias image, and then dividing by the masterflat image. Remember to check the `divzero` parameter in `imarithmetic`.
- Those of you who wonder if both operations can be combined into one may be delighted to learn that the answer is *yes*. The `imcalc` task in the `stsdas` package can perform multiple arithmetic operations in one step, though the syntax is slightly cryptic - see the built-in help for details!

2.3.5 Additional steps

We have now carried out the basic reduction steps and are ready to analyse the images further. However, it should be remarked that there may be variations and additions to the steps followed above. A few examples:

- *Overscan* correction: Instead of generating and subtracting a master bias image, some CCDs include an *overscan* region that can be used for bias subtraction. This is done simply by reading out more columns than are physically present on the CCD. For example, if 2100 columns are read out from an 2048×2048 pixel array, then the last 52 pixels in each row that are read out will already have been cleared by the read-out process, and the (mean) bias level can be determined from these pixels.

  Sometimes, bias and overscan corrections are treated independently. There might be variations in the bias level from pixel to pixel, which cannot be corrected using an overscan correction which only gives a mean bias level for each column. This then requires a full bias image. On the other hand, the bias level may vary systematically from one exposure to another, and in such a case the overscan region is needed to determine the mean bias level.

- Repair of bad pixels: The EMMI detectors are cosmetically quite good, with few defect pixels. However, this is not always the case. For detectors with many bad pixels, or entire bad rows or columns, it may be desirable to "fix" these in an additional step. IRAF includes a task called `fixpix`, which interpolates across image sections specified in a "bad pixel map". Clearly, such fixing should not be overdone, as bad data should ideally be excluded from analysis rather than "repaired". A better way to eliminate the effect of bad pixels is to obtained dithered exposures.

- Illumination correction: In the discussion above, it was implicitly assumed that the flat-field image is a good representation of sensitivity variations across the chip. This assumption may not always be true. In some instruments it is very difficult to obtain a uniformly illuminated image, because internal reflections in the instrument optics can introduce "ghosts".

  In particular, a common problem is "sky concentation" in which light is scattered from the edges of the field towards the centre, so that the flat-field image appears brighter near the centre. This effect can be quite deceptive, because the science images will suffer from the same effect so that the sky background will indeed appear "flat" after flat-fielding. However, stars near the centre will appear systematically too faint. The solution to this problem is to divide by an *illumination* frame, which can be constructed by placing a star (e.g. in a standard field) in various positions across the image and measure the counts from this star vs. (x, y) position after flat-field correction. The illumination frame can then be constructed by fitting a smooth two-dimensional function (e.g., a polynomial) to counts versus position.

- Dark current removal: This has been mentioned above.

- Fringe correction: Emission lines in the sky background may cause an interference pattern across the CCD image, especially at longer wavelengths. Correction for this effect is tricky because the intensity of these lines, and hence the fringe pattern, can vary strongly during the night. Assuming that the *shape* of the fringe pattern is at least constant, one can attempt to remove fringing by constructing a single fringe frame by observing an "empty" region of the sky and scaling the fringe pattern as required for each science frame. In practice, a full correction for this effect is very difficult.

2.4 Summary

In this lesson you have learned:

- Basic properties of CCD images
- Commands in IRAF for basic reduction of CCD images.
- How to handle *lists* of images