GCX User's Manual

Radu Corlan

Version 0.5.6 March 25, 2004

Contents

1 Introduction

The previous version of GCX , CX was written to control the newly designed `cpx3m` ccd camera. Once the basic camera control functions were running, it was easy to add some LX200 control functions, so that the telescope could be pointed at various objects without having to switch applications.

Having telescope control and image acquisition integrated into one program makes the following step obvious: after entering goto/get commands over several cold nights, one wants to automate the process—especially if he observes a large number of fields every night (as when doing variable star work).

The fact that the author's telescope doesn't point precisely doesn't help automation. So the ability to check/correct the pointing becomes essential. `cx` first got the ability to read star information from the GSC and overlay it on the images; that eases visual checks (one doesn't need maps anymore) but still is one step short of full automation.

Finally, when reliable field matching was implemented in GCX , it became possible to make the program fully automatic. In the current version, GCX can run through a list of observations completely unattended, and only stops if clouds roll in.

As it happens, field matching and image processing are also essential steps for CCD photometry. GCX implements *recipy files*: local catalogs of field and standard stars that are used to automatically reduce variable star frames.

1.1 Features

GCX can do the following:

Image handling

- Open 16-bit FITS image files;

  GCX uses floating-point images internally, so other FITS formats are easy to add;

- Zoom/Pan images, adjust brightness/contrast/gamma in an intuitive way, appropiate for astronomical images;

- Convert FITS files to 8-bit PNM after intensity mapping;

- Show image statistics (both global and local);

- Maintain a noise model for the image across transformations;
- Maintain bad pixel information;
- Perform ccd reductions (dark/bias/flat);
- Automatically align and stack images.

Catalogs and WCS

- Read field star information from GSC1/2 and Tycho2;
- Read object information from `edb` files;
- Read recipy files;
- Detect sources (stars) from images;
- Overlay objects on the image;
- Match image WCS to GSC or recipy field stars;
- Calculate world coordinated for image objects.

Camera Control

- Control cameras over a TCP socket using a simple protocol;

  The control proces (`cpxcntrl`) presently supports the cpx3m camera. It can be easily modified to support other cameras.
- Acquire images under script control;
- Set binning/windowing/integration times/temperature;
- Dark frames;
- All acquired frames are fully annotated in their FITS headers;
- Auto-generate descriptive names for files.

Telescope control

- Support LX200 protocol over serial;
- Point telescope under script control;
- Point telescope by object name (if edb catalogs are installed);
- Refine pointing by comparing image with GSC;

Aperture Photometry

- Do sparse field stellar photometry using circular apertures for stars, annular apertures for sky estimation;
- Aperture sizes fully programmable;
- Multiple sky estimation methods;
- Uses a complex error model thorughout, that takes into account photon shot noise, read noise, noise of the callibration frames and scintillation;
- Report noise estimates for every result;
- Take photometric targets (program and standard stars) from recipy files, or directly from the image;
- Produce a comprehensive report.

Interfacing

- Uses plain-ascii files for configuration files, reports and recipies;
- Most functions available in batch mode, so the program can be made part of a script.

1.2 Free Software

Gcx is free software, distributed under the GNU General Public License. Users can modify it to add features, reduction algoritms, support for other cameras/telescopes. It is written in C. The GUI uses the Gtk+ 1.2 toolkit. Some GNU-specific libc functions are used, but nothing fancy. It should compile and run on any system that has GNU tools, glibc and Gtk+ 1.2.

1.3 Contributing

The most important contribution you can make to GCX is to try it out, and don't give up immediately if something goes wrong. Complain to the author about it—he will try to help you.

The next important contribution one can make is to extend the hardware support of the program. When interface library are available for cameras (many manufacturers do have such libraries), it is relatively straightforward

to add support for a camera, as GCX has cleanly defined camera interface. Likewise, many mount/telescope manufacturers use the LX200 protocol, so essentially what is needed for other telescopes/mounts is tetsing and maybe a little tweaking. The program only uses a few LX200 functions, so interfacing to even a custom mount should be easy.

Third, there's the bane of free software: documentation. Any help in documenting or checking the documentation of the program is greatly appreciated, and will go a long way towards keeping GCX users happy.

And finally, the fun part: the code itself. There are many clever algorithms that can be added to the program, and which will benefit from the general infrastructure and integration provided by GCX .

1.4 About this Manual

This manual is work in progress. It starts with a tutorial introduction, so people can get a taste of what GCX is all about. The focus in that section is on operations that don't involve particular hardware (image viewing and data reduction).

What gets written next depends on feedback from users. Some aspects of the program's operation are self-explanatory, while some may be a little quircky. Ask for information, and it shall be provided.

The manual is maintained in LaTeX

1.5 Related Projects

`cpxctrl` the camera server used by GCX . Currently it supports the cpx3m camera, but should be easy to modify to control different ones;

`cpx3m` a free CCD camera design;

`avsomat` a batch variable star reduction program; more portable than GCX it shares some code, but uses a different field-matching algorithm. Avsomat and GCX use the same recipy format.

`xephem` The well known planetarium program by Elwood Downey. GCX uses the same object databse format as xephem, namely (`.edb`), as well as the same WCS annotation FITS fields. The star search algorithm is also inspired from xephem.

`libnova` A library for celestial mechanics and astronomical calculations; GCX uses some sidereal time and equatorial-to-horizontal coordinates transformation routines from libnova.

2 A Tutorial

This section is a tour of gcx's features that don't require any data files other than the ones provided with the distribution, or any special hardware. It should best be read while playing with the program.

The tutorial will go step-by-step using menu commands; this requires a fair amount of clicking. While helpful for starters and to see the basic steps, it's not a very efficient way to run GCX . Most commands have keyboard shortcuts (they are displayed in the menu next to each command). Most everything can be accomplished with a few keystrokes, or a cleverly contructed command line.

2.1 Building and Running gcx

If you're lucky (meaning that you have an i386 GNU/Linux system with compatible libc and `gtk+-1.2` is installed on your system), the precompiled binary supplied with the distribution will just work. To test, cd to the toplevel distribution directory (gcx-x.x.x) and run:

```
src/gcx
```

If all goes well, you should get an empty window with a menu. Type **ctrl-Q** or *File/Quit* to exit the program. It is recomended that the program is installed in `/usr/local/bin` for example.

If the above doesn't work (or even if it does), you have to recompile the program. Make sure `gtk+-1.2` is installed on the system (if you have Gnome, you also have gtk), then in the toplevel directory type:

```
./configure ; make
```

Configure takes some options. See the INSTALL file supplied with the distribution for more details. Also check the README file for the latest information.

If the above step completes successfully, become root and do a

```
make install
```

This will place the program in /usr/local/bin, and may also install data files in future versions.

The installation is now complete.

2.2 Starting with gcx

The `data` subdirectory of the distribution contains an example fits frame (`uori-v-001.fits.gz`), and an example recipy file for the frame (`uori.rcp`). These will be used throughout this section.

First, start the program:

```
gcx
```

If the program wasn't installed in `/usr/local/bin` or similar, you may have to type the full path to the binary; from the distribution toplevel directory type:

```
src/gcx
```

You should be presented with a empty window, with a menu at the top.

To load the example frame, type **ctrl-O** or use *File/Open Fits*; select the example fits file (`uori-v-001.fits.gz`) in the `data` directory an click *ok*. The program will load and display the frame.

Alternatively, the fits file name can be supplied on the command line. Something like:

```
gcx data/uori-v-001.fits.gz
```

will star the program and load the frame at the same time.

Two status bars are displayed at the bottom of the window. The left one shows the current display parameters: the zoom level, the *low cut* and the *high cut*. The low cut corresponds to black on the monitor, while the high cut corresponds to 100% white. The values are expressed in the same units the FITS file is.

The right-side status bar shows the various status and error messages. When loading an image, global statistics for the image are displayed. This will be referred to as the "status bar" throughout this manual.

GCX prints a lot of information on the terminal from which it was launched. This is mostly debugging information, which will probably be supressed in future versions; it can generally be ignored, but sometimes interesting information shows up.

On most errors, a beep is sounded and an error message is printed in the status bar. Sometimes though, a command may appear to do nothing. Checking the terminal will usually give a hint as to what happened.

2.3 Navigating the Image

To pan around the image, either use the scrollbars, or place the cursor over the point that you want in the center of the image and press the spacebar or the center mouse button. The image will pan only up to the point where it's edge is at the edge of the window.

You can pan back to the center of the image using **ctrl-L** or select *Image/Pan Center* from the menu.

To zoom in, place the cursor over the point you want to zoom in around, and press the = key (same key that has the '+' symbol). To zoom out, press **-**. The *Image* menu also has *Zoom In* and *Zoom Out* options.

When loading a frame, the image cuts are automatically selected for a convenient display of astronomical frames. The background is set at a somewhat dark level, and the dynamic range is set to span 2 times the standard deviation of the intensity across the frame. You can always return to these cuts by pressing **0** or selecting *Image/Auto Cuts*.

Pressing **1** – **8** will select various predefined contrast levels. **1** is the most contrasty: the image spans 4 sigmas, while **8** spans 90 sigmas. **9** will scale the image so that the full input range is represented (the cuts are set to the min/max values of the frame). Selecting *Image/Set Contrast/...* from the menu will accomplish the same effect.

To vary the brightness of the background, use **B** (*Image/Brighter*) and **D** (*Image/Darker*).

Another (sometimes more convenient) way of making contrast/brightness adjustments is to drag (move the mouse while holding the left button pressed) the pointer over the image. Dragging horisontally will change the brightness, while dragging vertically will adjust the contrast.

The key presses mentioned above are displayed in the menu alongside the respective options. **F1** or *Help/Show Bindings* will show on-line help about mouse actions.

It is important to know that all the ajustments described only apply to the display. The internal representation of the frame (and of course the disc file) is never changed in any way.

2.4 Examining the FITS Header

Select *File/Fits Header* from the menu. A new window will display the optional FITS header fields from the loaded frame.

2.5 Stars

GCX maintains a list of objects it can overlay on the display and run various processing steps on. They are called stars or sources. The stars can be extracted from the image, or loaded from catalogs or recipy files.

2.5.1 Detecting Stars

Ctrl-click on a star image. A round circle will appear around it (you cannot mark very faint or saturated stars). You don't need to click precisely on the peak - the program will search around, find a star and create an object (a *user star*) positioned at the centroid of the star image.

Click inside the circle. Information about the star will be displayed in the status bar: the start type (field star), the pixel coordinates (counting from the top-left corner), and the world coordinates if possible. Since the frame we loaded contained WCS information, but it couldn't be verified by the program, the status bar will show world coordinates, but will mark them as "uncertain" and disable all operations that depend on these objects' WCS. More on validating the WCS below.

Right clicking on a star will pop up a specific menu. As our WCS isn't validated yet, only the 'delete' option is active at this point.

Now press **S** or select *Stars/Detect Sources*. The program will search the whole frame, and mark stars. There is a limit as to how many stars will be marked. The limit can be changed by selecting *File/Edit Options*, clicking on the "+" next to *Star Detection and Search Options* and increasing the number in the *Maximum Number of Detected Stars* field.

There is also a limit on how faint the detected stars can be. Decreasing the value in the *SNR for Star Detection* field will make the program look for fainter stars. Note that a very low value of SNR will increase the run time of the detection routine considerably. Don't go below 1.0 or so.

To remove the detected stars from the display, use *Stars/Remove Detected Stars* or press **shift-S**.

Automatically detected stars and manually marked (user) stars are displayed with different symbols and deleted with separate commands, but otherwise equivalent. The program considers automatically detected stars somewhat expendable, but tries not to remove user stars unless specifically requested.

2.5.2 Catalog Stars

A second class of stars handled by GCX are catalog stars. They can be loaded from the GSC if it's installed on the system, or from recipy files.

Installing the GSC will be described later in this manual. For the moment, we will load the example recipy file from the `data` directory of the distribution.

Select *Photometry/Load Recipy* from the menu, then select the example recipy file in the `data` directory (`uori.rcp`) and click ok.

Three types of stars will show up. Diamond-shaped ones are field stars. They are used to fit and validate the WCS. Target-shaped symbols are the standard stars. Their magnitudes are used to photometrically calibrate the frame. Cross symbols are 'variable' stars - stars that we want to measure, but we don't know their magnitude in advance.

To find out more about a star, right-click on a star symbol, and select *E*dit Star from the pop-up menu. This will open a dialog and display information about the star, which can be edited. The designation, coordinates and comments fields should be obvious. Two types of magnitudes are shown: standard magnitudes are obtained from the catalog or recipy file; instrumental magnitudes are measured by the program.

A magnitude entry looks like this: `<band_name>(<system>)/<error>`. The error field is optional. The *band name* is the name of the filter ('v', 'b', etc). The *system* describes the source of the data. For instance, v(aavso) means 'v' magnitudes taken from aavso charts, while b(landolt) would be used for 'b' magnitudes of landolt standards. The program will not mix magnitudes from different systems when reducing data.

Four flags can be present: The "standard" flag shows that a star can be used as a standard in a photmetric reduction (so it's standard magnitudes come from an independent source). The "variable" flag shows that we want the star measured—it's standard magnitude will be replaced by a measured one by the photometry run; "standard" and "variable" are mutually exclusive. The "astrometric" flag shows that the star can be used for WCS calibration. Finally, the "field star" flag tells that the program can delete the star without asking the user.

2.6 World Coordinate System

Each time a frame is loaded, the program keeps track of the relation between the the positions within the frame, and the 'true' positions of the objects. This relation is called the "WCS" inside the program.

If no information is known about the position of the field, the WCS is called "invalid". This can happen if the frame doesn't have WCS information in the header. When some information is available, we say the we have an "initial WCS". The program will treat wcs information from the header as approximate. If we have an initial WCS and some field stars, we can match the positions of the field stars with stars detected from the frame. If the program finds a good-enough match, it will decide that the WCS can be reliably used, and mark the WCS as 'valid'.

Our example frame already has an initial WCS. We have field stars loaded from the recipy file (or we could have some from GSC). We will first press **S** to detect starts from the frame. Select *Wcs/Auto Pairs* (or press **P**). This will match the stars and create pairs, which are drawn with dotted lines. Next, press **W** (or *Wcs/Fit Wcs from Pairs*), and the program will fit the WCS so that the pairs overlap, and display the mean error of the fit in the status bar. If enough pairs are fitted and the error is small enough, the fit will be validated.

Pressing **M** or *Wcs/Auto Wcs* will do all the above steps in one operation (detect stars, load field stars from GSC if possible, find pairs and fit the WCS). Pressing **shift-M** or *Wcs/Quiet Auto Wcs* will do the same, but will remove the detected stars and field stars after the fit. It will do nothing if the WCS is already valid.

The fitting algorithm can be tuned by changing parameters under *WCS fitting options* in the options dialog.

Once we have a valid WCS, we have new uses for the detected and user stars. Clicking on them will print their true coordinates on the status bar. It is also possible to mark them as variable stars, so they can be measured, or as standard stars, so they can participate in the photometry solution (for example when inputting data from a paper chart).

Choose a few detected stars, right click on them and choose *Edit Star*. Now check the "variable" flag. The star will be transformed into a variable, and its symbol changed to a cross.

2.7 Photometry

Now that we have our valid WCS and we know which stars we want to measure and which standards to use, the actual photometry is easy: just press **shift-P** or *Photometry/Run*.

A quick result for the first variable stars is printed in the status bar, while a complete report is output on the terminal.

The reduction process has a number of parameters, which can be accessed through the options dialog, under *Aperture Photometry Defaults*.

For more details about the photometry run check the Aperture Photometry section below.

All the clicking in this section can be eliminated with one command. From the toplevel directory, run:

```
src/gcx data/uori-v-001.fits.gz-P datauori.rcp
```

The program will load the frame, load the recipy, fit the WCS and run the photometry. A report will be written to standard out (all debugging messages are pronted to stderr, so redirecting stdout to a file will write just the report to that file. For example,

```
src/gcx data/uori-v-001.fits.gz-P datauori.rcp>outf
```

will write the report to outf.

2.8 Configuration Options

GCX has a large number of configuration options. They can be accesses through the options dialog, which can be brought up by pressing **O** or *File/Edit Options*. Clicking on "save" in the options dialog will update the default configuration file (`.gcxrc`), located in the home directory. Note that option changes are not saved in the config file automatically.

When the program stars, it looks for the configuration file. If it cannot be read, it will initialise all parameters with defaults. Next, if the `-r` option is specified, it will read more options from the supplied file, so the parameters can be changed for a specific run.

The configuration file can be changed through the options dialog, or it can be edited directly.

2.9 Going Further

GCX has many more features and options than the ones described above. To find out about them, read below (as the next sections become available), browse the menus, or ask the author.

3 Image Files

4 CCD Reductions

5 Stars

6 Catalogs

7 The World Coordinate System

8 Aperture Photometry

8.1 Prerequisites

To measure stars, we need to know where the stars are located. We need some standards to compare them with. And we need some way of estimating the accuracy of our results.

In GCX all photometry targets are specified using their world coordinates (right ascension, declination and epoch). While it is possible to trick the program into just using stars from the image, this is not very useful. Looking up maps and clicking on stars is a tedious and error-prone process. It's also very difficult to check the results.

The program has a powerful world coordinate fitting algorithm. It is always prefferable to try to use it, than attempting to work without. See the tutorial and the WCS section for more details.

During a photometry run, the following information is used:

- The position, orientation an scale of the field — an *initial WCS* in GCX parlance. These vales only have to be known approximately; the program reads this information from the FITS header.
- The exact catalog positions of several *field stars* distributed across the field; these can be read from a *recipy file* or directly from the GSC.
- The positions and magnitudes of one or more standard stars. These are normally read from a recipy file.
- The positions of the stars we want to measure (called *variable stars*). These also come from the recipy file.
- Some parameters from which the noise can be estimated. They are: the number of electrons per intensity unit, the read noise of the camera,

the noise of the dark frames used to calibrate, the noise of the flat field used to calibrate, telescope aperture, integration time and airmass (for estimating scintillation). These parameters are read from the FITS header.

- The airmass. The program can calculate the airmass of the observation if it knows the location of the observing site, the date/time of the observation and of course the field coordinates) – which again come from the FITS header. Of course, the time of the observation should be recorded accurately in any case.

- A bad pixel map for the camera; it is used to flag situations where a bad pixel falls on a star we want to measure.

Not all parameters are equally important. Some noise parameters may be "guessed" by the program. Many parameters can be added to the fits header after the observation is made, as they are fixed for a given observer and equipment setup.

8.2 Reduction Steps

Once all the data is gathered, reduction proceeds as follows:

1. The WCS of the frame is fitted;

2. For each star we want to measure, the program calculated the pixel position inside the frame and looks around that position for a peak that looks like a star. If a peak is found within a certain distance, the position is corrected to the centroid of that star image. This step can be disabled.

3. The total flux within a circular aperture centered on the corrected position is calculated. The size of the aperture can be set by the user. If bad pixels are found within the aperture, the situation is flagged. The noise associated to this value is calculated using the supplied noise parameters.

4. A histogram is constructed from the values of pixels in an annular aperture concentric with the first. The size of the annulus can be set by the user. The sky backgound level is estimated from this area, using one of the following methods: mean, median, mean with κ-σ clipping, mean-median or synthetic mode. The noise associated to the background is also estimated.

5. The background is substracted from the star flux, and the noise contributions are added in quadrature. Scintillation noise (a multiplicative factor) is estimated and added to the noise.

6. The resulting flux and noise is converted to the magnitude scale.

7. For standard stars the error from the input data is added to the noise. From here on, we will use the term error. For standard stars without error information, a default error is added, and the situation is flagged.

8. A robust fit algorithm is used to determine the transformation between the instrumental and standard values.

   The fit algorithm also calculates the expected error of the transformation coefficients and the *mean error of unit weight*, which indicates the quality of the fit.

9. Finally, a report is generated. In many cases, finding an accurate solution is not possible just from analysing one frame. The report contains enough information so that many observations can be reduced together in a later step. However, collective reduction of many observations is beyond the scope of this program.

8.3 Noise and Error Modelling

The issue of noise modelling is essential in any photometric endeavour. Measurement values are next to meaningless if they aren't acompanied by a measure of ther uncertainty.

One can assume that the noise and error modelling only applies to deriving an error figure. This in true only in extremely simple cases. In general, the noise estimates will also affect the actual values. For instance, suppose that we use several standards to calibrate a field. From the noise estimate, we know that one of the standards has a large probable error. Then, we choose to exclude (or downweight) that value from the solution—this will change the calibration, and directly affect the result (not just it's noise estimate).

Precision and Accuracy. The precision of a measurement denotes the degree to which different measurements of the same value will yield the same result; it measures the repeatability of the measurement process. A precise measurement has a small *random error*.

The accuracy of a measurement denotes the degree to which a measurement result will represent the true value. The accuracy includes the *random error* of the measurement, as well as the *systematic error*.

Random errors are in a way the worst kind. We have to accept them and take into account, but they cannot be calculated out. We can try to use better equipment, or more telescope time and reduce them. On the other hand, since random errors are, well, random in nature (they don't correlate to anything), we can in principle reduce them to an aribitrarily low level by averaging a lerge number of measurements.

Systematic errors on the other hand can usually be eliminated (or at least reduced) by calibration. Systematic errors are not that different from random errors. They differ fundamentally in the fact the they depend on *something*. Of course, even random errors ultimately depend on something. But that something changes incontrollably, and in a time frame that is short compared to the measurement time scale.

A systematic error can turn into a random error if we have no control over the thing that the error depends on, or we don't have something to calibrate against. We could treat this error as "random" and try to average many measurements to reduce it, but we have to make sure that the something that the error depends on has had a change to vary between the measurements we average, or we won't get very far.

Noise is the "randomest" source of random errors. We have no way to calibrate out noise, but it's features are well understood and relatively easy to model. One doesn't have a good excuse not to model noise reasonably well.

We will generally talk about "noise" when estimating random errors that derive from electrical noise sorce. Once these are combine with other error sources (like for instance expected errors of he standards), we will use the term "error". Of course, there are two ways of understanding an error value. If we know what the true value should be, we can talk about and *actual error*. If we just consider what error level we can expect, we talk about an estimated, or *expected error*.

8.3.1 CCD Noise Sources

There are several noise sources in a CCD sensor. We will see that in the end they can usually be modeled with just two parameters, but we list the main noise contributors for reference.

1. *Photon shot noise* is the noise associated with the random arrival of photons at any detector. Shot noise exists because of the discrete nature of light and electrical charge. The time between photon arrivals is goverened by Poisson statistics. For a phase-insensitive detector, such as a CCD,
$$\sigma_{\mathrm{ph}} = \sqrt{S_{\mathrm{ph}}}$$
where S_{ph} is the signal expressed in electrons. Shot noise is sometimes called "Poisson noise".

2. *Output amplifier noise* originates in the output amplifier of the sensor. It consists of two components: thermal (white) noise and flicker noise. Thermal noise is independent of frequency and has a mild temperature dependence (is proportional to the square root of the absolute temperature). It fundamentally originates in the thermal movement of atoms. Flicker noise (or 1/f noise) is strongly dependent on frequency. It originates in the existance of long-lived states in the silicon crystal (most notably "traps" at the silicon-oxide interface).

   For a given readout configuration and speed, these noise sources contribute a constant level, that is also independant of the signal level, usually called the *readout noise*. The effect of read noise can be reduced by increasing the time in which the sensor is read out. There is a limit to that, as flicker noise will begin to kick in. For some cameras, one has the option of trading readout speed for a decrease in readout noise.

3. *Camera noise.* Thermal and flicker noise are also generated in the camera electronics. the noise level will be independent on the signal. While the camera designer needs to make a distiction between the various noise sources, for a given camera, noise originating in the camera and the ccd amplifier are indistinguishable.

4. *Dark current noise.* Even in the absence of light, electron-hole pairs are generated inside the sensor. The rate of generation depends exponentially on temperature (typically doubles every 6-7 degrees). The thermally generated electrons cannot be separated from photo-generated photons, and obey the same Poisson statistic, so
$$\sigma_{\mathrm{dark}} = \sqrt{S_{\mathrm{dark}}}$$
We can substract the average dark signal, but the shot noise associated with it remains. The level of the dark current noise depends on temperature and integration time.

5. *Clock noise.* Fast changing clocks on the ccd can also generate spurious charge. This charge also has a shot noise component associated. However, one cannot read the sensor without clocking it, so clock noise cannot be discerned from readout noise. The clock noise is fairly constant for a given camera and readout speed, and independent of the signal level.

Examining the above list, we see that some noise sources are independent of the signal level. They are: the output amplifier noise, camera noise and clock noise. They can be combined in a single equivalent noise source. The level of this source is called *readout noise*, and is a characteristic of the camera. It can be expressed in electrons, or in the camera output units (ADU).

The rest of the noise sources are all shot noise sources. The resulting value will be:

$$\sigma_{\text{shot}} = \sqrt{\sigma_{\text{ph}}^2 + \sigma_{\text{dark}}^2}$$

$$\sigma_{\text{shot}} = \sqrt{S_{\text{ph}} + S_{\text{dark}}} = \sqrt{S}$$

S is the total signal from the sensor expressed in electrons. So to calculate the shot noise component, we just need to know how many ADUs/electron the camera produces. This is a constant value, or one of a few constant values for cameras that have different gain settings. We will use A to denote this value.

8.3.2 Noise of a Pixel Value

We will now try to model the level of noise in a pixel value. The result of reading one pixel (excluding noise) is:

$$s = s_b + A(S_d + S_p)$$

where s_b is a fixed bias introduced by the camera electronics, S_d is the number of dark electrons, and S_p is the number of photo-generated electrons (which is the number of photons incident on the pixel multiplied by the sensor's quantum efficiency).

Now let's calculate the noise associated with this value.

$$\sigma^2 = \sigma_r^2 + A^2(S_d + S_p) = \sigma_r^2 + A(s - s_b)$$

Where σ_r is the readout noise expressed in ADU, and S is the total signal expressed in electrons. Note that we cannot calculate the noise if we don't

know the bias value. The bias can be determined by reading frames with zero exposure time (bias frames). These will contribute some read noise though. By averaging several bias frames, the noise contribution can be reduced. Another approach is to take the average across a bias frame and use that value for the noise calculation of all pixels. Except for very non-uniform sensors this approach works well. GCX supports both ways.

Note that a bias frame will only contain readout noise. By calculating the standard deviation of pixels across the difference between two bias frames is $\sqrt{2}$ times the readout noise.

8.3.3 Dark Frame Subtraction

A common situation is when one substracts a dark frame, but doesn't use bias frames. The noise associated with the dark frame is:

$$\sigma_d^2 = \sigma_r^2 + A^2 S_d$$

The resulting pixel noise after dark frame substraction will be:

$$\sigma_{\text{ds}}^2 = 2\sigma_r^2 + A^2(2S_d + S_p)$$

while the signal will be

$$s_{\text{ds}} = AS_p$$

Using just the camera noise parameters, we cannot determine the noise anymore. We have to keep track of the dark substraction and it's noise effects. We however rewrite the dark-substracted noise equation as follows:

$$\sigma_{\text{ds}}^2 = \left(\sqrt{2\sigma_r^2 + 2A^2 S_d}\right)^2 + A^2 S_p$$

If we use the notation $\sigma_r' = \sqrt{2\sigma_r^2 + 2A^2 S_d}$, we get:

$$\sigma_{\text{ds}}^2 = \sigma_r'^2 + A^2 S_p$$

This is identical in form to the simple pixel noise equation, except that the true camera readout noise is replaced by the equivalent read noise σ_r'. What's more, the bias is no longer an issue, as it doesn't appeear in the signal equation anymore. We can derive the pixel noise from the signal directly, as:

$$\sigma_{\text{ds}}^2 = \sigma_r'^2 + As_{\text{ds}}$$

The same parameters, σ_r' and A are sufficient to describe the noise in the dark-substracted frame.

8.3.4 Flat Fielding

To flat-field a frame, we divide the dark-substracted pixel value $s_{\rm ds}$ by the flat field value f. The noise of the flat field is σ_f. The resulting signal value is

$$s_{\rm ff} = \frac{1}{f} A S_p$$

If we neglect second-order noise terms, the noise of the flat-fielded, dark substracted pixel is:

$$\sigma_{\rm ff}^2 = f\sigma_r'^2 + A^2 S_p + \left(\frac{\sigma_f}{f} A S_p\right)^2$$

$$\sigma_{\rm ff}^2 = f^2\sigma_r'^2 + A f s_{\rm ff} + (\sigma_f s_{\rm ff})^2$$

The problem with this result is that f is not constant across the frame. So in general, the noise of a flat-fielded frame cannot be described by a small number of parameters. In many cases though, f doesn't vary too much across the frame. We can then use it's average value, $\widetilde{f}$ for the noise calculation. This is the approach taken by the program.

We can identify the previous noise parameters, $\sigma_r'' = \widetilde{f}\sigma_r'$ and $A' = A\widetilde{f}$. We have to introduce a new parameter, σ_f.

Without reducing generality, we can arrange for $\widetilde{f} = 1$. This means that the average values on the frames don't change with the flatfielding operation, and is a common choice.

In this case, σ_r and A aren't affected by the flatfielding operation, while the third noise parameter becomes $\sigma_f/\widetilde{f}$, which is the reciprocal of the SNR of the flat field.

GCX models the noise of each pixel in the frame by four parameters: σ_r, A, $\sigma_f/\widetilde{f}$ and $\widetilde{s_b}$. The noise function $n(s)$ of each pixel is:

$$n^2(s) = \sigma^2 = \sigma_r^2 + A|(s - \widetilde{s_b})| + \left(\frac{\sigma_f}{\widetilde{f}}\right)^2 (s - \widetilde{s_b})^2$$

σ_r comes from the `RDNOISE` field in the frame header. A is the reciprocal of the value of the `ELADU` field. $\sigma_f/\widetilde{f}$ comes from `FLNOISE`, while $\widetilde{s_b}$ comes from `DCBIAS`.

Every time frames are processed (dark and bias substracted, flatfielded, scaled etc), the noise parameters are updated.

8.3.5 Instrumental Magnitude Errors

Once we know the noise of each pixel, deriving the expected error of an instrumental magnitude is straightforward. Let N_b be the number of pixels in the sky annulus, and s_i the level of each pixel. The noise of the sky estimate is: [1]

$$\sigma_b^2 = \frac{1}{N_b} \sum_{i=1}^{N_b} n^2(s_i)$$

Now let N_s be the number of pixels in the central aperture. The noise from these pixels is:

$$\sigma_s^2 = \sum_{i=1}^{N_s} n^2(s_i)$$

The total noise after sky substraction will be:

$$\sigma_n^2 = \sigma_s^2 + N_s \sigma_b^2$$

.

The program keeps track and reports separately the photon shot noise, the sky noise, the read noise contribution and the scintillation noise.

Scintillation is an atmospheric effect, which results in a random variation of the received flux from a star. We use the following formula for scintillation noise:

$$\sigma_{\text{sc}} = 0.09F \frac{AM^{1.75}}{D^{2/3}\sqrt{2t}}$$

Where F is the total flux received from the star, AM is the airmass of the observation, D is the telescope aperture in cm, and t is the integration time. Scintillation varies widely over time, so the above is just an estimate.

Finally, we can calculate the expected error of the instrumental magintude as

$$\epsilon_i = 2.51188 \log \left(1 + \frac{\sqrt{\sigma_n^2 + \sigma_{\text{sc}}^2}}{F} \right)$$

[1] This assumes that the method used for sky estimation has a statistical efficiency close to the mean, which isn't generally the case. Perhaps this should be taken into account, at least for methods whose efficiency is well known, like the median.

8.4 Fitting the Solution

Having obtained instrumental magnitudes and error estimates for the standard stars, the program proceeds to finding a transformation between the instrumental and standard system. If color information is not present, or there aren't enough standards of different color, only a zeropoint for the transform is calculated. If the program decides there is enough information, it will determine both a zeropoint and a color coefficient. [2]

A robust fitting algorithm is implemented, which proceeds as follows:

1. First, the natural weights for each standard star are calculated as

$$W_i = \frac{1}{\epsilon_i^2 + \epsilon_s^2}$$

   where ϵ_i is the estimated error of the instrumental magnitude, and ϵ_s is the error of the standard magnitude (obtained from the recipy file).

2. The values are fitted using these weights. For the no-colors case, the fit is simply a weighted mean; When color coefficients are fitted, linear regression is used.

3. The residuals ρ_i for each star are calculated.

4. The weights are adjusted based on how much the residuals differ from their expected values. The following function is used:

$$W_i' = W_i \left(1 + \left(\frac{\rho_i}{\alpha \sqrt{\epsilon_i^2 + \epsilon_s^2}} \right)^{\beta} \right)^{-1}$$

   Th weighting function reduces the weight of values that have residuals α times larger than expected to one half. Of course values with even larger residuals are downweighted even more. The parameter β tunes the "sharpness" of the weighting function.

5. steps 2-5 above are repeated until the solution converges (or the iteration limit is reached).

6. Finally, the error for the estimated parameters is calculated. In the no-colors situation, the error for the zeropoint is calculated as follows:

[2] the framework for color coefficient fitting is in place, but the actual fitting is not implemented yet

$$\epsilon_{\rm zp}^2 = \frac{\sum \rho_i^2 W_i}{\sum W_i}$$

fwhile the *mean error of unit weight* is:

$$\mathrm{me1}^2 = \frac{\sum \rho_i^2 W_i}{N-1}$$

where N is the number of standard stars. The mean error of unit weight is 1 in the ideal case (when all the errors are estimated correctly). A significantly larger value should raise doubts about the error estimates.

8.5 Reporting

An example report is shown below.

```
(   observation
      ( object "rs-lyr" ra "19:13:01.88" dec "33:24:46.13"
        equinox 2000 jdate 2452760.45870622
        telescope "30cm SCT FLEN=2000.0" aperture 30.0
        exptime 20.00 sns_temp 238.0 filter "v"
        )
    noise (read 7.3 eladu 2.0 flat 0.0 scint 0.003)
    ap_par (r1 3 r2 9 r3 13 sky_method "median" sigmas 3.0)
    stars (
      ( name "GSC2657-0144" type std
        smags "v(aavso)=12.300 p(gsc)=11.750" imags "v=-9.958/0.010"
        residual 0.066 weight 384.712
        flags ( centered undef_err )
        noise (sky 0.001 read 0.004 photon 0.008 scint 0.003) )
      ( name "GSC2657-0268" type std
        smags "v(aavso)=13.000 p(gsc)=12.710" imags "v=-9.149/0.016"
        residual -0.043 weight 362.665
        flags ( centered undef_err )
        noise (sky 0.003 read 0.008 photon 0.012 scint 0.003) )
      ( name "GSC2657-0835" type std
        smags "v(aavso)=9.200 p(gsc)=9.010" imags "v=-12.967/0.004"
        residual -0.025 weight 397.903
        flags ( centered undef_err )
        noise (sky 0.000 read 0.000 photon 0.002 scint 0.003) )
```

```
      ( name "RS-LYR" type var
        comments "VMAG=(9.2 15.8) SPECTYPE=M5e VARTYPE=M VARID=1909+33"
        smags "v(aavso)=13.684/0.054" imags "v=-8.508/0.025"
        flags ( centered )
        noise (sky 0.005 read 0.015 photon 0.016 scint 0.003) )
    )
    transform (zerop 22.192 zp_err 0.048 me1 1.14)
)
```

The report is in the form of an *association list*, i.e. a list of name-value pairs. Some values can also be lists. Top-level list elements are:

observation a list of parameters that globally describe the observation. They are taken straight from the FITS header of the frame being reduced.

noise the noise parameters from the frame header.

ap_par photometry process parameters used. These parameters are read from the configuration file, and can be modified in the Options dialog.

stars a list of the stars that have been measured. The instrumental magnitudes, derived standard magnitudes are listed here. The error estimate for the star is broken down in components (from the sky baground estimation, from read noise, from photon shot noise and from scintillation). The total error appears in the error field of the instrumental magnitude. For standard stars, the residual and the weight used in the fit are listed. A list of flags in included, which signal occurences in the reduction process that can have an influence on results.

transform the parameters of the trasformation from instrumental to standard, as determined by the program.

9 Camera Control

10 Telescope Control

11 Observation Scripts

12 Command Line Options and Batch Processing