ULTRACAM Pipeline Software

Tom Marsh*

May 18, 2001

Abstract

An important component of ULTRACAM will be the pipeline software for reducing the data. This document defines the requirements for this software and details its design.

1 Introduction

ULTRACAM has significant advantages over photomultiplier-based photometers, but one significant disadvantage – the potentially huge amount of data it can produce. The pipeline software is the component of the instrument which needs to alleviate this disadvantage.

This document specifies requirements and details the design path I am taking. As far as this review is concerned, if you read nothing else, please look at section 5 which details areas where I may need input from the ATC and perhaps vice-versa.

2 Requirements

The two requirements of the pipeline are

1. *To provide fully-reduced data for the observer.*
2. *To provide the observer with an idea of what the data are like in near real-time.*

Requirement (1) is obvious. The very least the pipeline software must do is reduce ULTRACAM data. The ideal is that after setting-up, the software can be left to do its job automatically.

Requirement (2) is needed to allow the observer to make decisions based upon what is being obtained. Minimally this may be reducing every $N^{\rm th}$ image and plotting a light-curve. N can always be made large enough for this to be possible.

*e-mail: trm@astro.soton.ac.uk

3 Goal

Merely getting an idea of the data quality is not always enough. For instance an obvious application of ULTRACAM will be the observation of a phenomenon called "dwarf nova oscillations", which are very low amplitude periodic oscillations that appear and disappear for reasons that are not understood. These cannot even be seen in reduced light-curves – one must take Fourier transforms. These sorts of constraints lead to the goal of the pipeline software which is

- *To produce fully-reduced data in near-real time.*

(Analysis after this stage is relatively trivial.)

4 Design

4.1 Introduction

I am focussing on standard photometric data reduction rather than other potential ULTRACAM applications such as shift-and-add imaging. These are the first steps needed to reduce photometric data:

1. Prepare flat field, bias, dark and bad pixel frames.
2. Identify the target and comparison stars in each CCD.
3. Define star and sky photometric apertures for each object of interest in each CCD (see appendix A for a figure illustrating this).
4. Specify the noise characteristics of each CCD (and possibly of each separate readout region).

Having done these, the time-critical part of the software must do the following:

1. Load flat field, bias, dark and bad pixel frames.
2. Load the photometry apertures (hereafter just "apertures")
3. Define the methods e.g. of centroiding, sky estimation and flux extraction.
4. Identify the next data frame to be reduced (if any).
5. Load this frame.
6. Centroid the apertures on the objects of interest.
7. Estimate the sky background and extract fluxes of each object while applying dark, bias, flat-field and bad pixel corrections.

8. Save the results to disk.
9. Return to step 4.

4.2 Implementation

4.2.1 A monolithic program for the time-critical part

For speed, I believe that the time-critical element identified above must be implemented as a *single program.* By doing so the flat-field, bias, dark and bad pixel frames and the program itself are only loaded into memory once. The trickiest part of this program is almost certainly the centroiding since this is when one encounters problems such as the telescope drifting and/or jumping, clouds, cosmic rays and confusion between close-by objects. Still I don't think this is going to be a major obstacle; I intend to borrow from tried-and-tested methods as much as possible.

4.2.2 Support programs

Beyond the time-critical program identified above, many other support programs needed. For testing purposes I am developing routines to produce realistic simulated data including the generation of noise, cosmic rays, drifting and jumping star-fields, bad pixels etc. Image display and arithmetical manipulation routines will also be needed as will routines for defining windows, apertures and test star fields. Finally programs to display and perform elementary analysis of the reduced data are needed. See appendix A for example plots from the early phases of some of these routines.

4.2.3 Use of standard software *vs* starting from scratch

Many of the support functions I identify above are already available in large packages such as IRAF, KAPPA, FIGARO etc. Nevertheless I think that these will not always be directly usable. The reason is that ULTRACAM data will have a fundamentally multi-CCD/multi-window format and the above packages are all based upon a single main array plus bits-and-bobs of essentially header information. Multi-CCD data could be accommodated as a 3D array, but multi-windows is less easy to cope with. For instance an obvious way to accommodate ULTRACAM data within the range of these standard programs would be to read out the windows as a single frame, filling in the gaps with zeroes, however this could be hugely inefficient in that it would increase the amount of storage needed. Since I expect that ULTRACAM will be used with small, possibly well-separated, windows for most of its operation to obtain the fastest readout, I don't think that any such padding is acceptable except as a conversion option to allow access to the power of the large packages for small numbers of images. Thus in my opinion, storage on disk and within programs should handle the data alone. The down-side to this is the need to code up many fairly standard routines. This leads me on to the final part of this section.

4.2.4 C++ implementation

ULTRACAM data will be fairly complex, consisting of 3 CCD images, each with a number (the same number for each CCD) of windows. Thus for example, adding two ULTRACAM frames will require loops over the number of CCDs, the windows of each CCD and the pixels of each window of each CCD. Along with this, representations are needed for star-fields and photometric apertures. Handling of such data structures is well-suited to object-oriented languages, and for this reason I am coding the pipeline software in C++. This requires definitions of the basic data structures (objects) and operations on these objects. For instance, I have so far defined objects to represent an ULTRACAM frame, a bad pixel mask, a set of CCD windows, photometric apertures and a star field. This then allows one to pass ULTRACAM frames to functions as a single argument as in e.g.

```
Ultracam frame(``junk'');
do_something(frame);
```

where the object of type "Ultracam" called "frame" contains everything you need to know about an ULTRACAM frame. This sort of thing makes for rapid coding of the many basic routines and makes it much easier to maintain the code and cope with alterations than non-object oriented code.

Initially I am coding these to run as command line plus terminal input routines. Ultimately I intend to add GUIs to them to make them more user friendly.

5 Interfacing with ATC

In many ways this is the most important section for the review. I welcome input on anything I might have missed.

Much of the pipeline software can be developed stand-alone but there are several areas of overlap. Those that have occurred to me are as follows:

5.1 Identification of the next available frame

The pipeline needs some way to know what is the next frame available for input. One way would be to run a directory listing and find the most recent frame. However, given that there could easily be several thousand frames available, this is likely to be inefficient. It would be better if there was some standard file which contained the names of the most recent frames available to be read (the last 5 for example).

It would also help navigation over files if they were written with a standard sequence number e.g. 'r00000001', 'r00000002' etc.

5.2 Disk file data format

I am using a straightforward binary format at the moment, but can change it easily. This just needs agreeing upon. I list my current format (without headers as yet) at the end of this document. We need to set a date for defining this completely.

5.3 Window format restrictions

I don't know all the restrictions that will have to be satisfied by the possible windows. For example: will windows always occur in pairs? will they not be able to straddle the half-way point of the CCDs? etc. These do not matter as far as the pipeline software is concerned but may be needed if the pipeline software is to play a role in defining windows (next). If this is the case, I need to have the restrictions defined.

5.4 Window definition

I am not yet clear how the windows will be selected. I imagine that in practice one would want to take one or two full frame images and then use these to define the windows (see appendix A) for example). I have already written a routine to do this as I needed one for testing purposes. The question is, will this be useful/needed for the data acquisition system, and if so, how can it be sent back to it? It may be of course that something similar will have been developed for the DAS in any case. Nevertheless, there may still be value in being able to use windows set up by the pipeline.

5.5 Variation of parameters over the CCDs

Since the ULTRACAM CCDs will be read out at two places, I imagine that is possible (likely?) that fundamental parameters such as the bias, readout noise and gain may be different in different areas of the chip. I just need to know whether or not this is likely to be the case in order to cope with it.

6 Progress & Timescales for completion

I have made progress on several of the items referred to above. In particular the essential data structures are more-or-less defined and I am now developing individual programs and have developed initial implementations of window and aperture setting routines as well as arithmetical ones for example.

Task	Status	Time needed (weeks)
Data structure definitions	85%	1
Basic arithmetic routines	100%	—
Fake data generators	40%	0.5
Aperture definition	70%	0.5
Starfield definition	100%	—
Window definition	100%	—
Image display	100%	—
Reduction program	0%	3
Reduced data display & analysis	0%	1.5
Full test with fake data	0%	2
Total time	—	8.5

Table 1: Time estimates for pipeline software tasks

Table 1 is a list of tasks and their completion status. I assume that basic documentation and initial testing of each routine are part of each task.

My target date for completing this work is **mid-September**. At this stage I will be able to judge the frame rates that will be possible to cope with, currently the biggest unknown.

By this stage I will have a working system, but not an especially user-friendly one. The last two tasks will be provide a graphical user interface for the software and to write a guide to using it. I have started to investigate Tcl/Tk for the first task although I am still in the learning stage. I have kept user input to the routines to an absolute minimum to facilitate this task. I am using Perl-based routines to facilitate documentation which will be web-based in the first instance. The guide will essentially be a recipe-style series of pointers to get users started.

A A few plots

Here are a few plots to clarify some of the ideas used in this document. First, I anticipate obtaining full frame images which can be displayed as in Figs. 1 and 2. These can then be used to define the data windows as shown in Fig 3.

Note that no restrictions have been applied to the windows (apart from not allowing them to overlap or run off the area of the CCD). There will be other conditions in practice, but these are not relevant to most of the pipeline.

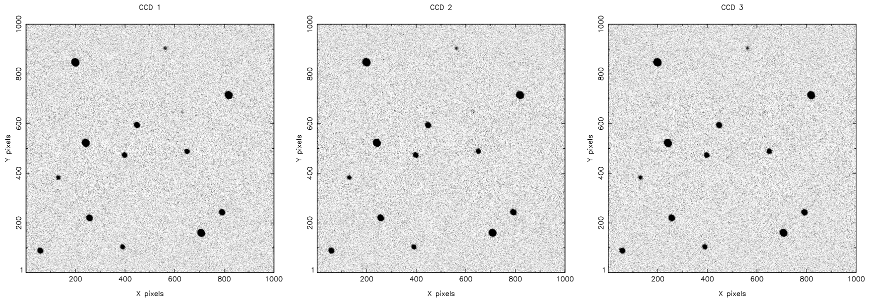

Figure 1: A partially-simulated (no bad pixels, cosmic rays etc) ULTRACAM data frame, all three CCDs

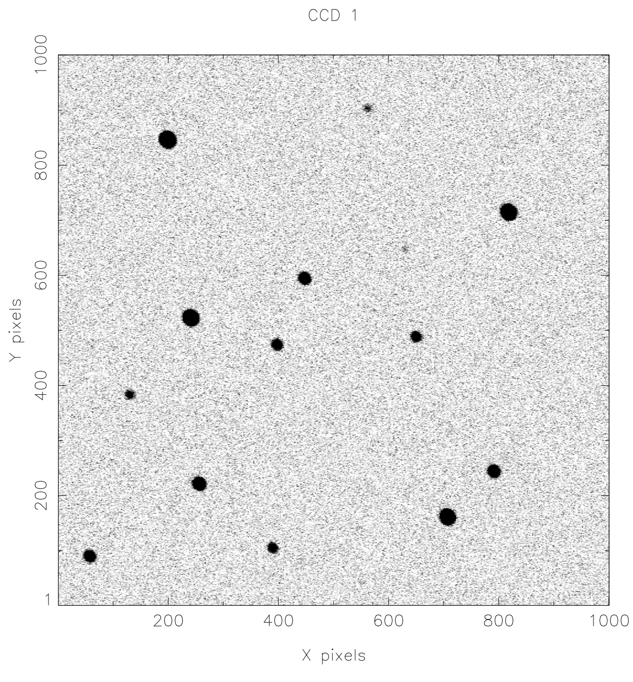

Figure 2: A partially-simulated ULTRACAM data frame, just one CCD.

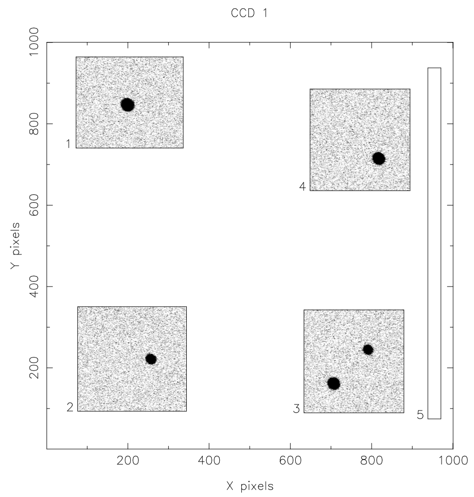

Figure 3: Same field, with 5 windows set.

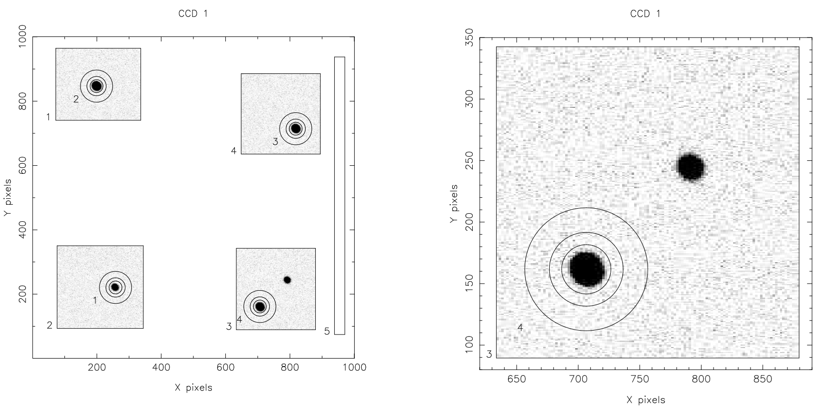

Figure 4: Same field, showing apertures (not precisely centroided). The right-hand panel zooms in on the bottom-right window.

Fig. 4 gives an idea of photometric apertures with inner circles covering the targets and two outer ones defining an annulus of sky. In the simplest photometry (which is not that far from the best one can do) the flux is merely summed over the inner circle after subtraction of a sky estimate from the annulus. This is a straightforward task. The more time-consuming part comes in centroiding the apertures in the first place. Even Tim Naylor's optimal photometry (which I intend to implement) is not that much more difficult, and thus centroiding will still be the critical task.

Parameter	Data type	Number
Number of windows/CCD (nwin)	unsigned int	1
Binning factor in X	unsigned int	1
Binning factor in Y	unsigned int	1
Total number of X pixels (unbinned) of CCDs	unsigned int	1
Total number of Y pixels (unbinned) of CCDs	unsigned int	1
X pixel of lower-left corners (unbinned) of windows	unsigned int	nwin
Y pixel of lower-left corners (unbinned) of windows	unsigned int	nwin
X dimensions (binned) of windows	unsigned int	nwin
Y dimensions (binned) of windows	unsigned int	nwin
Data type flag	int	1
Number of CCDs	unsigned int	1
The data	Depends on flag	Total # of pixels.

Table 2: Current dta file format. There are no headers as yet.

B Current disk data format

Table 2 is a list of what I currently write to disk in binary format to store ULTRACAM frames. Obviously the exact meaning of data types such as "unsigned int" are implementation dependent, but it is easy for me to change these e.g. to "unsigned short int" (which are 2-byte integers on my machine). Note that this is more general than necessary – the CCD dimensions and numbers could be fixed, but I am thinking somewhat for the future here. I expect header information to precede what is written here.