Data Flow System

Document Title:	**VISTA Infra Red Camera Exposure Time Calculator Specification**
Document Number:	**VIS-SPE-IOA-20000-0009**
Issue:	**1.1**
Date:	**2005-09-01**

Document Prepared by:	**Simon Hodgkin** (CASU)	**Signature And date:**	
Document Approved by:	**Mike Irwin** (CASU Manager)	**Signature And date:**	
Document Reviewed by:	**Will Sutherland** (VDFS Project scientist)	**Signature And date:**	
Document Released by:	**Jim Emerson** (VDFS Project leader)	**Signature And date:**	

Change Record

Issue	**Date**	**Sections Affected**	**Remarks**
0.5	2004-11-18	All	Draft document
1.0	2005-01-10	All	First Issue
1.0	2005-09-01	6, 8, Appendix C	Replaced perl code with C code and amended Section 8. Removed requirement on remote command line calls

Notification List

The following people should be notified by email that a new version of this document has been issued:

IoA	Will Sutherland
QMUL	Jim Emerson
ATC	Malcolm Stewart
RAL	Gavin Dalton

Contents

1 Introduction ... 4
1.1 Purpose ... 4
1.2 Scope ... 4
1.3 Applicable Documents ... 4
1.4 Reference Documents ... 4
1.5 Acronyms and Abbreviations ... 5
1.6 Glossary ... 5
2 Instrument Configurations ... 6
2.1 Overview ... 6
3 Observing Strategies ... 7
4 Architecture ... 7
4.1 Instrument Model ... 7
4.2 Atmospheric Transmission Model ... 8
4.3 Sky Emission Model ... 8
4.4 Source Model ... 8
4.4.1 Source Spectrum ... 8
4.4.2 Source Geometry ... 9
5 ETC Characteristic Data ... 9
5.1 ETC Database ... 9
5.1.1 Units ... 10
5.1.2 File Format ... 10
6 ETC User Interface ... 10
6.1 Input Parameters ... 10
7 Mathematical Model ... 13
7.1 Basic Signal to Noise Calculation ... 14
7.2 Calculation of object and tile exposure times, and elapsed time for a tile ... 16
8 Calculation Functions ... 17
9 Validation Sets ... 17
Appendix A – Use-Cases ... 18
Appendix B – VIRCAM ETC WWW Interface Mock-up ... 20
Appendix C – ETC C Code ... 22

1 Introduction

The Vista Infrared Camera (VIRCAM) is a wide-field near-infrared imaging instrument for the Cassegrain focus of the 4-metre Visible and Infrared Survey Telescope for Astronomy (VISTA). The pawprint of VIRCAM covers 0.59 square degrees in the Y, J, H, K_S (and possibly z_{IR}) passbands using a 4×4 array of 2k×2k non-buttable chips with 0.34 arcsecond pixels.

1.1 Purpose

This document describes an Exposure-Time Calculator (ETC) for VISTA with VIRCAM. The general requirements for an ESO ETC are described in [AD1]. The ETC programmes allow the user a large amount of control over the simulator. The main task of the ETC is to evaluate the exposure time required to reach a given signal-to-noise for a given set of source characteristics, atmospheric conditions and instrument configuration. The ETC also allows the equivalent signal-to-noise to be computed for an input exposure time.

VISTA is required to survey efficiently (i.e. to have high survey speed). The ETC therefore includes additional functionality to provide the actual elapsed time (including overheads) needed to complete a standard filled tile to the depths and signal-to-noise specified. In conjunction with the Survey Definition Tool [AD2], this enables the user to examine different observing strategies and examine/minimize the overheads.

1.2 Scope

The ETC document and software is part of the design of VIRCAM/VISTA operations, which also includes a Survey Definition Tool [AD2]. The interaction with camera templates and observation strategy is briefly discussed in Section 3.

1.3 Applicable Documents

[AD1] VLT–SPE–ESO–19000–1618, *Data Flow for VLT/VLTI Instruments: Deliverables Specification*, Issue 2.0, Date 2004–05–22

[AD2] VIS-SPE-ATC-20000-0010, *VISTA Survey Definition and Progress Tools: Functional Specification*, Issue 1.0, Date 2004-11-17

[AD3] VIS-SPE-IOA-20000-0001, *VISTA Infra Red Camera DFS User Requirements*, Issue 1.0, Date 2004-12-15

1.4 Reference Documents

[RD1] *A Theoretical Investigation of Focal Stellar Images in the Photographic Emulsion and Application to Photographic Photometry*, Moffat A.F.J., 1969, A&A, **3**, 455.

[RD2] IRTRANS4: http://www.jach.hawaii.edu/UKIRT/astronomy/utils/atmos-index.html

[RD3] Infrared Exposure Time Calculator for ISAAC: http://www.eso.org/observing/etc/doc/ut1/isaac/helpisaac.html

[RD4] *IR Camera Observation Software Design Description*, VIS-DES-ATC-06084-0001, issue 3.0, 2004-12-14.

1.5 Acronyms and Abbreviations

DICB	Data Interface Control Board
DIT	Detector Integration Time
ETC	Exposure-Time Calculator
FWHM	Full-Width at Half Maximum
IDF	Instrument Definition File
IR	Infrared
Mag	Magnitude
NDIT	Number of Detector InTegrations in an exposure
PAF	Parameter Format File
QE	Quantum Efficiency
SED	Spectral Energy Distribution
VIRCAM	VISTA InfraRed CAMera
VISTA	Visible and Infrared Telescope for Astronomy
VOTABLE	Virtual-Observatory Table format
WWW	World-Wide Web

1.6 Glossary

To aid the understanding of the concepts in logical order the glossary is not alphabetical.

Integration A simple snapshot, within the DAS, of a specified elapsed time. This elapsed time is known as the detector integration time - DIT secs.

Exposure The stored product of many individual **integration**s, that have been co-added in the DAS. Each exposure is associated with an exposure time.

Microstep (pattern) A pattern of **exposure**s at positions each shifted by a very small **movement** (<3 arcsec) from the reference position. Unlike a **jitter** the non-integral part of the shifts are specified as 0.5 of a pixel, which allows the pixels in the series to be interleaved in an effort to increase resolution. A microstep pattern can be contained within each position of a **jitter** pattern.

Jitter (pattern) A pattern of **exposures** at positions each shifted by a small **movement** (<30 arcsec) from the reference position. Unlike a **microstep** the non-integral part of the shifts is any fractional number of pixels. Each position of a jitter pattern can contain a **microstep** pattern.

Pawprint The 16 non-contiguous images of the sky produced by the VISTA IR camera, with its 16 non-contiguous chips. The name is from the similarity to the prints made by the padded paw of an animal (the analogy suits earlier 4-chip cameras better).

Tile A filled area of sky fully sampled (filling in the gaps in a pawprint) by combining multiple **pawprint**s. Because of the detector spacing the minimum number of pointed observations (with fixed offsets) required for reasonably uniform coverage is 6, which would expose each piece of sky, away from the edges of the tile, to at least 2 camera pixels.

2 Instrument Configurations

2.1 Overview

VISTA is an alt-azimuth 2-mirror telescope with a single focal station (Cassegrain), which can accommodate one of two possible cameras: an IR Camera or (if funded later) a visible camera. The telescope has a fast focal ratio (f/1 primary, f/3.25 at Cass) hence a compact structure. The telescope uses active optics, with 81 axial force actuators controlling the shape of the primary mirror (M1), and a 5-axis hexapod controlling the position of the secondary mirror (M2). The primary and secondary mirrors may be coated either with Al or with protected Ag.

The infrared camera (details in [AD3]) is a novel design with no cold stop, but instead a long cold baffle extending ~ 2.1m above the focal plane to minimize the detectors' view of warm surfaces. There is a large entrance window (95cm diameter) and 3 corrector lenses, all IR-grade fused silica. There is only one moving part (the filter wheel). The camera also contains fixed autoguiders and wavefront sensors (2 each, using CCDs operating at approximately 800nm wavelength) to control the tracking and active optics.

The filter wheel has space for 8 main filters, one of which is dark and the remaining seven are for science and include Y, J, H, K_S and any other filters that may become available (e.g. z_{IR}, K and narrowband filters).

VISTA is a survey instrument, designed to cover large areas of sky as efficiently as possible. A typical multiband survey would cover a certain area (which may or may not be contiguous), would be observed to a uniform depth in several filters and would possibly be done with some repetition to pick up variability or proper motions. Such a survey will normally be the result of the combination of many observations made over many nights and several such surveys will probably be running concurrently. The ETC and SDT together will enable the user to examine different observing strategies and examine/minimize the overheads. The main observing modes are discussed in more detail elsewhere ([AD3]).

3 Observing Strategies

VIRCAM has predefined observing templates from which the user may choose. These are described in detail in [AD3] and summarized briefly here. The component observations of a survey (filters F, tiles T, pawprints P, jitters J, microsteps M, and exposures E) can in principle be nested in various orders, subject to some restrictions, such as the innermost loop always being E. Different nestings have different overheads.

Using shorthand (based on the order of nesting of the loops for the 6 components F, T, P, J, M, E with the order of the letters indicating increasing nesting of the loop read to right) the three allowable nestings would be:

- **FTPJME —** Complete all tiles in one filter, change filters and repeat the tile sequence.
- **TFPJME —** Complete each tile in all filters before starting on the next tile in the sequence.
- **TPFJME —** Complete each pawprint of the tile in all filters before starting on the next pawprint of the tile.

[A further nesting TFJPME is defined in [RD4] but as it is not expected to be much used it is not included in the ETC, for simplicity]

It is beyond the scope of the ETC to fully calculate the total time (with overheads) required to survey arbitrary areas with these different observing strategies. However the ETC can calculate the total elapsed time (with overheads) for one tile in one filter, and will indicate the overheads for each step of the calculation. In such a situation (1 tile, 1 filter) the overheads for all of the three observing modes become identical.

Capability for calculating overheads where filter changes are involved is not currently planned to be provided within the ETC, but will be provided elsewhere, [TBD e.g. the Survey Definition Tool].

4 Architecture

VIRCAM only has an imaging mode and consequently a single instrumental model. This is combined with the source model, a sky emission model, and an atmospheric transmission model. This section describes the component models of the ETC.

The numerical information for all these components will be supplied to ESO in order to define the Instrument Definition File. Once the user has selected the desired instrument setup, the ETC builds this model from the Instrument Definition File (IDF).

4.1 Instrument Model

This model describes the sequence of instrument components that a light ray has to pass through during an observation. It consists of fixed components (e.g. primary and secondary mirrors, external baffles, cryostat window, internal baffles, 3 lenses) and a single user-selected component (filter). The ETC should be able to handle changes to these components, e.g. switching the mirror coating from Al to protected Ag. The instrument model describes the reflection and transmission of all these components. In addition, the VIRCAM detectors will see background photons emitted by these same components. The instrument model will describe the emission arising from specifically: the primary and secondary mirrors, the warm baffle, the cryostat window, the cold baffle, the three lenses, the cryostat itself and the filters. In practice, this local background will be monitored and use to calibrate the model.

4.2 Atmospheric Transmission Model

This model includes a model atmosphere, which describes the effect of atmospheric absorption and extinction as a function of source airmass and humidity. The dominant source of opacity in the near infrared is absorption by water molecules. The ETC will be provided with atmospheric transmission models that span a suitable range of airmass and humidity (e.g. [RD2]) allowing the user to select an appropriate extinction value.

4.3 Sky Emission Model

This model describes the sky brightness as a function of wavelength. The dominant source of sky emission is in the form of narrow emission lines from O and OH. The 0.8-2.5 μm sky brightness is consequently highly variable and essentially independent of lunar phase, but is weakly dependent on airmass. The ISAAC ETC [RD3] assumes a default average sky (J=16.5, H=14.4, K_S=13.0 magnitude/arcsec^2), but also simulates bright and dark cases (delta sky = ±0.5 magnitude). The VIRCAM ETC will enable the user to select the sky brightness, with suggested average site values. The sky model undergoes the same instrumental transformations as the source model, except airmass correction.

4.4 Source Model

Each simulated source is described by a spectral energy distribution and a geometry.

4.4.1 Source Spectrum

The spectral energy distribution is calculated from a list driven menu. Computed spectra are scaled to match the required magnitude or flux in the specified filter. Vega and AB magnitudes and flux densities will be supported. The following spectral energy distributions are available:

- Power Law Continuum ($F(\lambda) = k\lambda^{\alpha}$, $F(\nu) = k\nu^{\alpha}$, $\nu F(\nu) = k\nu^{\alpha}$) scaled to the object magnitude or flux, (with appropriate units e.g. W/m^2, Jansky x Δν, ergs/cm^2/s). (Ultimately, the ETC will replace the single band integrated flux with specification of a flux density [choice of F(λ), F(ν) or νF(ν)(=λF(λ))] at a user provided wavelength, and the ETC will use the selected filter bandpass to calculate the band integrated flux.)

- Blackbody (defined by the user-selected temperature in Kelvin, scaled to the object magnitude or flux)
- Emission line (a Gaussian with user-selected flux, λ_{cen} and *FWHM*)

4.4.2 Source Geometry

The ETC can simulate either point sources or extended sources.

- A point source is specified by the size-scale of the seeing-dependent point spread function. The signal-to-noise is computed over a circular aperture with user-selected diameter (a sensible default diameter would be equal to twice the seeing). The point spread function is modelled by a Moffat function [RD1], see equations in Section 7.1.
- For extended sources, all calculations are in terms of surface brightness, and magnitudes are in magnitudes per square arcsecond. Surface brightness is assumed to be uniform over the extent of the user-defined aperture.

5 ETC Characteristic Data

The VIRCAM ETC requires access to an ETC database containing the following:

1. **Calibration files:**
   - sky background emission
   - telescope emission (primary and secondary mirrors, and warm baffle)
   - camera window emission
   - cryostat emission (cold baffles, lenses, cryostat, filters)
   - atmospheric transmission (including extinction and absorption)
   - optical reflection and transmission (primary and secondary mirrors, cryostat window, lenses within cryostat)
   - filter transmission curves
   - detector characteristics, including: quantum efficiency (10%, 90% values), gain, read-noise.

2. **VIRCAM Instrument Definition File**

Additionally, The ETC will make use of the following data which are not stored beyond the duration of a simulation.

- Observation related: e.g. user-defined instrument configuration, observing characteristics. These data are stored in a temporary file and summarized in the output.
- Run-time files: these contain intermediate results from the simulation.
- Simulation results: the output is stored in a single temporary file that can be retrieved by the user.

5.1 ETC Database

The file formats and reference units are defined as follows:

5.1.1 Units

The VIRCAM ETC database will follow standard ESO practice and will use nm for wavelength and ergs/s/cm^2/nm for monochromatic flux. The user interface will allow a range of commonly used units; results will be communicated in a range of units. Transmission, reflectivity and quantum efficiency values range from 0 to 1. Sky brightness is given in magnitudes/square arcsecond. Extinction values are in magnitudes per unit airmass.

5.1.2 File Format

- Calibration files are stored as ASCII files to minimize external dependencies. Two comment lines indicate the file contents; a third gives the number of rows that follow. Sampling in wavelength will be set to 1 nm.

- The IDF is written using the ETC syntax according to [AD1].

6 ETC User Interface

The ETC will reside in Garching and the user interface will be via HTML-based web pages. The look and feel will follow existing ESO instrument ETC interfaces, with one important enhancement. VIRCAM is a survey instrument and the ETC includes some functionality to enable the user to examine different observing strategies and examine/minimize the overheads. The GUI interface will have the following layout:

- **Heading:** Links to ETC help, FAQ, useful numbers (gain, zeropoints etc)
- **Sections 1-4:** Input parameters
- **Output Page:** summary of the input parameters and the results of the simulation in form of both numbers and graphs/files that can be retrieved via the web (see below) including overheads.

The required functionality, and look-and-feel, was arrived at by considering a number of use-cases (see Appendix A – Use-Cases).

6.1 Input Parameters

This section describes the layout and instrument specific parameters of the input HTML form taking into account the instrument configuration, and to a limited extent, the possible observing strategies. From a user's point of view, the VIRCAM ETC GUI should appear very similar to the ISAAC ETC GUI (see Appendix B – VIRCAM ETC WWW Interface Mock-up). A link to the Survey Definition Tool (SDT) will be added when the ETC-SDT interface has been agreed).

The table below shows the input parameters in each section of the form (defaults are in **bold)**. The input method is either via a radio button or drop down menu where a limited number of values is allowed, or via a box in which the user types. Additional parameters which have not been included at this point, but may need to be considered are: ambient temperature, state of optical surfaces (dust) and distance from moon.

§1	Flux Distribution	Radio:**<u>PowerLaw</u>**(Menu $F(\lambda)$,$F(\nu)$,$\nu F(\nu)$,α) Blackbody (temperature, K) Line (wavelength, flux, width)Radio:
	Object Magintude	**<u>Vega: Box: 18.0</u>** AB: Box: 18.0 Flux: Box: 1.0e-16 [The flux box will be replaced by $F(\lambda)$, $F(\nu)$, $\nu F(\nu)$,and λ boxes in the next version]
	Spatial Distribution	Radio: **<u>Point Source</u>**, Extended
	Aperture	Box: **<u>2.0</u>** arcsec (diameter)
§2	Filter	Menu: Y **<u>J</u>** H K_S etc
§3	Sky Brightness	Box: **<u>15.2</u>**
	Airmass	Box: **<u>1.60</u>**
	Seeing	Box: **<u>0.80</u>**
	Atmospheric Extinction	Box: **<u>0.03</u>**
§4	Detector integration time (*DIT*) sec	Box: **<u>20</u>**
	Radio: **<u>Signal-to-noise</u>** Exposure Time Observing Strategy	Box: **<u>20</u>** Box: **<u>600</u>** secs N_{DIT} : Box: **<u>3</u>** (any integer allowed) N_{exp} : Box: **<u>1</u>** (any integer allowed) N_{micro} : Menu: **<u>1×1</u>** (no movement) or 2×2 N_{jitter} : Box: **<u>2</u>** (any integer allowed) N_{paw} : Box: **<u>6</u>** (any integer allowed) N_{tile} : Box: **<u>1</u>** (any integer allowed)

The total number of exposures (see Glossary) that make up an observation is defined as:

$$N_{tot} = N_{tile} \times N_{paw} \times N_{jitter} \times N_{micro} \times N_{\exp}$$

where, N_{tile} is the number of times a tile is observed and N_{paw} is the number of pawprints within the tile. N_{jitter} is the number of jitter positions and N_{micro} is 1 for no microstepping and 4 for 2×2 microstepping. N_{exp} is the number of exposures measured at each position. Because a single exposure is made up of N_{DIT} integrations, then the total number of integrations in an observation is:

$$N_{\text{int}} = N_{DIT} \times N_{tot}$$

Section 4 of the form allows the user to choose between signal-to-noise, exposure time or observing strategy led calculation. In all cases the detector integration time (*DIT*) should be specified (a suitable default will be automatically entered upon selection of a filter).

If the signal to noise option is chosen the user specifies the signal-to-noise they wish to reach, and the ETC will calculate the total exposure time required (adjusting this to give an integer number of integrations, N_{int}, each of duration DIT). No attempt is made to decompose this further into tiles, pawprints, jitter, microsteps or to define N_{DIT}.

The exposure time option allows the user to specify the total number of seconds of integration on source. The ETC will calculate the achieved signal-to-noise in the closest exposure time which allows for an integer number of integrations N_{int}.

Finally, if the observing strategy option is chosen, then it is up to the user to define the numbers of integrations, exposures, microsteps, jitters, pawprints and tiles. The ETC will calculate the achieved signal-to-noise and in addition will calculate the overheads in the observing sequence, that is the elapsed time as well as the exposed time. There are certain constraints on observing strategy, for example $N_{DIT} \times DIT$ needs to be greater than about 30 seconds in order for the wavefront sensor to integrate on a star.

The following should be noted:

- With the default tiling strategy, where 6 pawprints make up one tile with uniform coverage, then the number of times every sky pixel is observed is $N_{cov}=N_{paw}/3$
- The number of *integrations* (each of DIT sec) that go into the calculation of total on-chip exposure time per object per tile is $N=N_{int}/3$. With the default tiling strategy, each sky pixel will be observed twice in one tile.
- The total exposure time *per tile* (assuming uniform coverage) is $t_{exp}(tot)=t_{exp}(obj)\times 3$, i.e only about 1/3 of the tile time is spent on each particular piece of sky, the rest of the time being spent filling in the gaps between the chips.
- Non-uniform tiling options are not handled by the ETC, but can be calculated from a single pawprint.

For the Observing strategy option, the ETC will report back three times

- $t_{exp}(obj)$ the total on-chip exposure time per object
- $t_{exp}(tot)$ the total on-chip exposure time
- $t_{elapsed}(tot)$ the total elapsed time, allowing for instrument and telescope overheads (e.g. time to readout, microstep, jitter, movement to the pawprint positions making up a tile, time to write data to disk, time to start a tile)

A more detailed discussion of this is given in Section 7.2. Guidance on overheads incurred in doing another tile at the same position, with or without a filter change, is given in the help.

The output page will include warnings if users selected non-optimal or high overhead observing strategies. In addition the ETC will include extensive help pages with worked examples suggesting a number of observing strategies for common requirements.

User-selectable features will follow the ISAAC ETC and could include the standard options for plotting, e.g. Detector Illumination, S/N as a function of Exposure Time, Total Efficiency, SNR versus seeing (only for point-sources), Input spectrum in physical units. Output tabulated results include:

- summary of input parameters
- background per pixel (electrons & ADU)
- sky + instrument background + readout noise per pixel (electrons & ADU)
- DIT time factor to sky saturation
- signal in aperture (electrons and ADU)
- aperture correction (magnitudes)
- signal-to-noise
- peak signal in object (electrons & ADU)
- DIT time factor to peak saturation
- time per object for signal-to-noise
- total exposure time
- total elapsed time
- saturation/non-linearity warnings for sky and target

A draft of the html output page is shown in Figure 3.

7 Mathematical Model

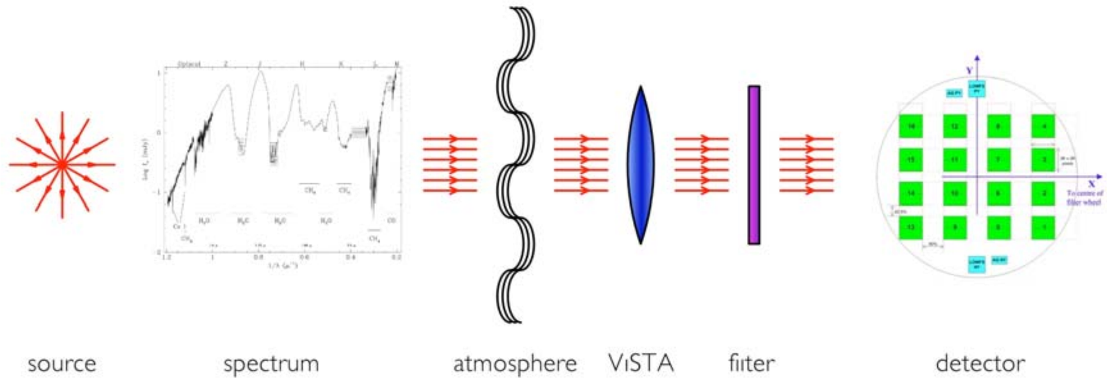


Figure 1: The propagation of the signal

7.1 Basic Signal to Noise Calculation

We start with a (template) astronomical spectrum in photons against wavelength and normalized to match the magnitude in the chosen filter. Then, the number of electrons/nm detected from the astronomical source is given by the following equation relating the input spectrum to the output spectrum as shown in Figure 1.

$$P_{\mathrm{det}}(\lambda) = t_{\mathrm{exp}}(obj) A P_{obj}(\lambda) T(\lambda,\chi) F(\lambda) R(\lambda) Q(\lambda)\,\mathrm{I}$$

where

- $P_{\mathrm{det}}(\lambda)$ is the detected spectrum in electrons/s/nm/m^2
- $t_{\mathrm{exp}}(obj)$ is the total object on-chip exposure time in seconds
- A is the unobstructed area of the main mirror in m^2
- $P_{obj}(\lambda)$ is the input spectrum in photons/s/nm/m^2
- $T(\lambda,\chi)$ is the transmission of the atmosphere at the given airmass
- $F(\lambda)$ is the throughput of the filter
- $R(\lambda)$ is the throughput of the telescope/instrument
- $Q(\lambda)$ is the Quantum Efficiency of the assumed detector (0-1)
- *I* is the *average* number of electrons liberated by a single detected photon (*I* is taken as 1 and so dropped from all following equations).

The total number of source electrons collected in bandpass λ_1 to λ_2 is then

$$N_{src} = \int_{\lambda_1}^{\lambda_2} P_{\mathrm{det}}(\lambda) d\lambda = t_{\mathrm{exp}}(obj) A \int_{\lambda_1}^{\lambda_2} P_{obj}(\lambda) T(\lambda,\chi) R(\lambda) Q(\lambda) F(\lambda) d\lambda$$

and for the background

$$N_{back} = t_{\mathrm{exp}}(obj) A \int_{\lambda_1}^{\lambda_2} P_{sky}(\lambda) T(\lambda,\chi) R(\lambda) Q(\lambda) F(\lambda) d\lambda$$
$$+ t_{\mathrm{exp}}(obj) \int_{\lambda_1}^{\lambda_2} P_{local}(\lambda) Q(\lambda) F(\lambda) d\lambda$$

where $P_{sky}(\lambda)$ is the spectrum of the sky background in photons/s/nm/m^2/arcsec^2 and $P_{local}(\lambda)$ is the spectrum of the local background radiation incident on the filter and detector. Note that, particularly in the K-band, $P_{local}(\lambda)$ will include contributions from thermal emission from the telescope structure (predominantly the primary and secondary mirrors, and the baffle), the camera itself (lenses, baffles, filters, window) and the telescope dome. See Section 5 for a list of the calibration files which will describe these components.

For a point source, we calculate the source and background photons within a user-defined aperture. For an extended source, the calculation is per square arcsecond. We assume that the point spread function can be approximately described by a Moffat function [RD1],

$$I(r) = I_O \left[1 + \left(\frac{r}{\alpha} \right)^2 \right]^{-\beta}$$

such that the number of electrons detected within a circular aperture, out to radius r (arcseconds), is:

$$N_{src}(r) = N_{src}\{1 - [1 + (r/\alpha)^2]^{1-\beta}\}$$

where

- β is the atmospheric scattering coefficient and
- α is related to the full-width half-maximum (i.e. the seeing) of the profile by the equation:

$$\alpha = \frac{FWHM}{2(2^{1/\beta} - 1)^{1/2}}$$

The number of sky background electrons enclosed in the same aperture is simply:

$$N_{back}(r) = N_{back}\pi r^2$$

From which we can calculate the signal-to-noise ratio

$$SNR = \frac{N_{src}(r)}{[N_{src}(r) + N_{back}(r) + \left(N_{dark} + (RN)^2\right)\frac{\pi r^2}{scale^2}]^{1/2}}$$

where RN is the detector read noise per pixel (e^-), N_{dark} is the number of electrons arising from dark current ($N_{dark} = t_{exp}(obj)$ × dark current per pixel/sec), and *scale* is the platescale of the detector in arcsecs/pixel. The number of detected counts is calculated from the gain g of the A/D converter.

VIRCAM has 16 detectors and inevitably there will be some spread in sensitivity across and between the arrays (expected to be <20%). For the ETC calculations we will use two values of the QE:

1. for signal-to-noise calculations we will use the 90th percentile QE (that is 90% of VIRCAM pixels have a measured QE greater than or equal to this value).
2. for saturation calculations we will use the 10th percentile QE (that is 10% of VIRCAM pixels have a measured QE greater than or equal to this value).

7.2 Calculation of object and tile exposure times, and elapsed time for a tile

Although VIRCAM allows various observing modes (e.g. FTPJME, TFPJME, TPFJME) the ETC is limited to calculations involving a single tile (which may be repeated N_{tile} times) in a single filter, for which all three modes have similar overheads. Consequently, although the elapsed time to perform an observing sequence will in practise depend on the observing mode used this is not taken in to account in the ETC. So for a given tile and filter we have $t_{\exp}(obj)$ the total on-chip exposure per object

$$t_{\exp}(obj) = DIT \times N_{tot} = DIT \times N_{DIT} \times N_{micro} \times N_{jitter} \times N_{\mathrm{cov}} \times N_{tile}$$

Where the symbols are DIT (detector integration time), N_{dit} (the number of integrations at each microstep position), N_{micro} (the number of microsteps at each jitter position), N_{jitter} (the number of jitters around each pawprint position), N_{cov} (the number of times a given sky position is covered within the tile), and N_{tile} the number of times the tile is repeated.

The total on-chip exposure time is

$$t_{\exp}(tot) = t_{\exp}(obj) \times 3.$$

The elapsed time per tile (excluding any slewing to it from the previous sky position) is:

$$\begin{aligned} t_{elapsed}(tile) = &\left\{(O_{reset} + DIT + O_{DIT}) \times NDIT + O_{save}\right\} \times N_{\exp} \times N_{micro} \times N_{jitter} \times N_{paw} \\ &+ O_{micro} \times (N_{micro} - 1) \times N_{jitter} \times N_{paw} \\ &+ O_{jitter} \times (N_{jitter} - 1) \times N_{paw} \\ &+ O_{paw} \times (N_{paw} - 1) \end{aligned}$$

and the total elapsed time is

$$t_{elapsed}(tot) = t_{elapsed}(tile) \times N_{tile}$$

where the $O_{\mathrm{subscript}}$ are the time overheads associated with each operation and $N_{paw} = N_{cov} \times 3$ for the default 6 pawprints per tile. Thus the ETC will calculate elapsed time per tile as well as actual and object exposure times per tile and in total.

A working version of the ETC is available at www.ast.cam.ac.uk/vdfs/etc.html. This code has been incorporated into Issue 1 of this document.

8 Calculation Functions

The calculation of the elapsed time, which includes overheads specific to the instrument and telescope, is required. In this document we include C code of a working VIRCAM ETC which incorporates calculation of both exposed and elapsed time and therefore takes into account observing efficiency. The following files are included and are available from the authors:

- *etc.c* test harness
- *vircam_etc_calc.c* main procedure
- *vircam_etc.h* include file
- *vircam_etc_emiss.c* calculate SED for input emission line
- *vircam_etc_plaw.c* calculate SED for input power law
- *vircam_etc_bbody.c* calculate SED for input black body
- *vircam_etc_getqecurve.c* get QE for appropriate filter
- *vircam_etc_plot.c* handle graphical output

9 Validation Sets

A validation dataset of tables and input data parameters, together with intermediate and final results will be provided with the ETC.

Appendix A – Use-Cases

To help develop the functionality of the ETC, the following use-cases were considered.

UC1: What exposure time is required to reach J=24 at 5σ (i.e. how many background limited exposures would need to be stacked) at zenith, with seeing=0.9 arcsecs assuming no jittering and no microstepping

SED	power law, f_λ, α=0
Mag	24
Geometry	point source
Aperture	2.0
Filter	J
Sky brightness	15.2
Airmass	1.0
Seeing	0.9
Extinction	0.03
DIT	20
SNR	5
Exposure Time	Calculated by ETC

UC2: What signal-to-noise do I get for a galaxy with K_S=20 mag/arcsec^2 in a 5 minute exposure with airmass=1.5 and seeing=0.8 arcsec assuming no jittering/microstepping.

SED	power law, f_λ, α=0
Mag	20
geometry	Extended
Aperture	2.0
Filter	K_S
Sky brightness	13.4
Airmass	1.5
Seeing	0.8
Extinction	0.03
DIT	10
SNR	Calculated by ETC
Exposure TIme	30

UC3: I want to image a tile with 2x2 microstepping and a 5-point jitter pattern using the default pointing and tiling mode with background limited exposures. How deep do I go and what are the overheads.

SED	power law, f_λ, α=0
Mag	20
geometry	Point source
Aperture	2.0
Filter	H
Sky brightness	14.4
Airmass	1.5
Seeing	0.8
Extinction	0.03
DIT	10s
SNR	Calculated by ETC
N_{DIT}	3
N_{EXP}	1
Microstepping	2×2
Jitter	15
Pawprints	6
Tiles	1

Appendix B – VIRCAM ETC WWW Interface Mock-up

Figure 2: Preliminary version of the ETC interface, see http://www.ast.cam.ac.uk/vdfs/etc.html

VISTA IR Camera: Exposure Time Calculator: Output

Object Setup:

```
Source type        : point source
       spectrum    : f_lambda power law
       alpha       : -2.0
       magnitude   : 18.0 (Vega)

Aperture diameter  : 2.0 arcsec
```

Instrument Setup:

```
Filter             : J
```

Sky Conditions:

```
Sky Brightness     : 15.6 mag/arcsec2
Airmass            : 1.2  sec z
Seeing             : 0.8  arcsec
Extinction         : 0.05 mag/unit airmass
```

Observation setup:

```
DIT                : 10.0 s
NDIT               : 1
Exposure loops     : 1
Microstep pattern  : 1x1
No. of jitters     : 1
No. of pointings   : 1
```

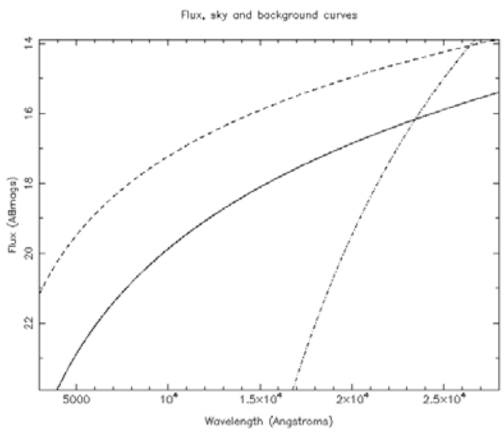
Results:

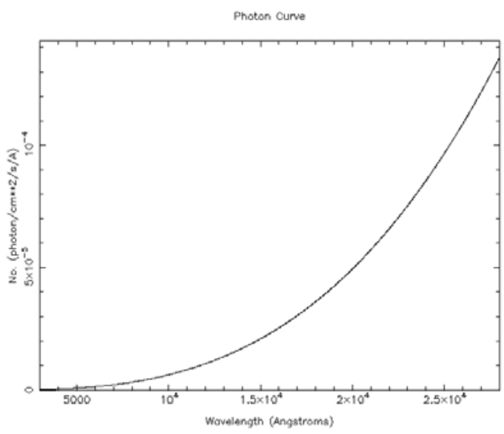
```
Sky+thermal background per pixel per DIT:  13716 (e-)    4572 (ADU)
Photon+readout noise per pixel per DIT  :    118 (e-)      39 (ADU)
DIT time factor to background saturation:   10.9

Signal in aperture per DIT              :   8980 (e-)    2993 (ADU)
Aperture correction (loss)              :   0.35 (mag)
Peak pixel signal in object per DIT     :   1083 (e-)     361 (ADU)
DIT time factor to peak saturation      :  138.5
Object signal-to-noise per DIT          :   14.5

Time per object for signal-to-noise     :   60.0 (s)
Total object signal-to-noise            :   35.5
```

Plots:





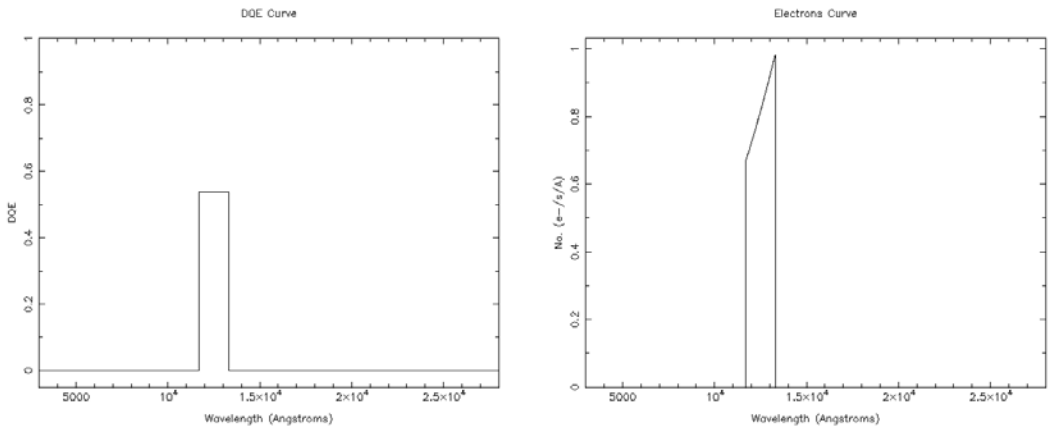


Figure 3: A preliminary version of the ETC output page. Some example diagnostic figures are shown for completeness. See http://www.ast.cam.ac.uk/vdfs/etc.html

Appendix C – ETC C Code

```
/* test harness for vircam_etc_calc. */
#define USAGE "Usage: etc [-h|-?] \n" \
        "etc -[p|b|e]\n"
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <strings.h>
#include <readline/readline.h>

#include "vircam_etc.h"

double getval(const char *prompt, const double def)
{
        char *line;
        char *prompt2;
        double y;
        int m;

        prompt2 = malloc(strlen(prompt) + 20);
        (void)sprintf(prompt2, "%s [%g]: ", prompt, def);
        line = readline(prompt2);
        m = sscanf(line, "%lf", &y);
        if ( m != 1) y = def;
        free(line);
        free(prompt2);
        return y;
}

double getival(const char *prompt, const int def)
{
        char *line;
        char *prompt2;
        int j;
        int m;

        prompt2 = malloc(strlen(prompt) + 20);
        (void)sprintf(prompt2, "%s [%d]: ", prompt, def);
        line = readline(prompt2);
        m = sscanf(line, "%d", &j);
        if ( m != 1) j = def;
        free(line);
        free(prompt2);
        return j;
```

```
}
int main(int argc, char **argv)
{
        extern char           *optarg;
        int status = 0;
        vircam_etc_input_t     input;
        vircam_etc_output_t    output;
        int                    p, n, m;
        double                 x;
        char                  *band[] = {"Z", "Y", "J", "H", "K"};
        double                 skymag[] = {18.2, 16.8, 15.6, 14.4, 13.2};
        double                 vegatoab[] = {0.5, 0.7, 0.9, 1.4, 1.9};

        (void)printf("Test VIRCAM Exposure-Time Calculator:\n");
/*      Stack input vector with defaults: */
        (void)strcpy(input.outdir, "/tmp");
        input.powerlaw=0;
        input.blackbody=0;
        input.emission=0;
        input.iflav=2;
        input.alpha= -2.0;
        input.theta=10000;
        input.refwave=15000;
        input.sigma=10.0;
        input.lineflux=1e-16;
        input.imag=1;
        input.vmag=18.0;
        input.abmag=18.0;
        input.flux=1.0e-13;    /* ergs/s/cm^2 */
        input.ispat=1;
        input.aperture=2.0;
        (void)strcpy(input.band, "J");
        input.skymag=15.6;
        input.vegatoab=0.9;
        input.airmass=1.2;
        input.seeing=0.8;
        input.extinction=0.05;
        input.dittime=10.0;
        input.idit=1;
        input.ndit=6;
        input.snr=20.0;
        input.nexp=1;
        input.nmicro=1;
        input.njitter=1;
        input.npaw=1;

        /* Process command-line options: */
        while ((p = getopt(argc, argv, "h(help)f:(flux)")) !=EOF)
        switch(p)
        {
        case 'h':
        case '?':
        default:
                (void)printf("%s", USAGE);
                exit(1);
                break;
        case 'f':
                if(*optarg == 'b') input.blackbody = 1;
                        else if(*optarg == 'p') {
                                input.powerlaw = 1;
                                } else if(*optarg == 's') {
                                        input.emission = 1;
                                        }
                break;

        }

        /* Interactively query for unset values:     */
        while(input.blackbody == 0 && input.powerlaw == 0 &&
                input.emission == 0) {
                for(n=0; n < 1 || n > 3;
                        n = getival("Input Flux Distribution:\n"
                        "1: Power Law\n"
```

```
                "2: Blackbody\n"
                "3: Single Line: ", 1));
        switch(n) {
        case 1:
                input.powerlaw = 1;
                input.iflav = getival("Power-law Type:\n"
                        "1: f(nu)\n"
                        "2: f(lambda)\n"
                        "3: nu f(nu)  ", 2);
                input.alpha = getval("alpha ", -2.0);
                break;
        case 2:
                input.blackbody = 1;
                input.theta =
                        getval("Temperature in Kelvin ", 10000.0);
                break;
        case 3:
                input.emission = 1;
                input.refwave = 10.0 *getval("Wavelength/nm ", 1500.0);
                input.lineflux = getval("Flux ergs/s/cm ", 1e-16);
                input.sigma = 10.0 * getval("Line width/mn ", 1.0);
                break;
        default:
                (void)fprintf(stderr, "bad spec ival\n");
                exit(2);
        }
}
for(n=0; n < 1 || n > 3; n = getival("Select Object Magnitude type:\n"
        "1: Vega\n"
        "2: AB\n"
        "3: Flux  ", 1));
input.imag = n;
switch (input.imag) {
case 1:
        input.vmag = getval("Vega Magnitude ", 18.0);
        break;
case 2:
        input.abmag = getval("AB Magnitude ", 18.0);
        break;
case 3:
        input.flux = getval("Flux value (ergs/s/cm**2", 1e-13);
        break;
default:
        (void)fprintf(stderr, "bad mag choice\n");
        exit(1);
        break;
}
input.aperture = getval("Aperture ", 2.0);
for (n=0; n < 0 || n > 2; n = getival("Source type:\n"
        "1: Point Source\n"
        "2: Extended Source ", 1));

(void)printf("\nInstrument Setup:\n\n");
for (n=0; n < 1 || n > 5; n = getival("Choose Filter:\n"
        "1: Z\n"
        "2: Y\n"
        "3: J\n"
        "4: H\n"
        "5: K\n", 3));
n--;
(void)strncpy(input.band, band[n], 2);
input.skymag = skymag[n];
input.vegatoab = vegatoab[n];

(void)printf("\nSky Conditions:\n\n");
input.skymag = getval("Sky Brightness (mag) ", input.skymag);
input.airmass = getval("Airmass ", 1.2);
input.seeing = getval("Seeing (arcsec) ", 0.8);
input.extinction = getval("Extinction ", 0.05);

(void)printf("\nObserving Setup:\n\n");
input.dittime = getval("Detector on-chip integration (DIT) ", 10.0);
for (n=0; n < 1 || n > 3; n = getival("Exposure Criterium:\n"
```

```
        "1: Choose Exposure time\n"
        "2: Choose S/N ratio\n"
        "3: Choose Observing Strategy  ", 1));
input.idit = n;
switch (input.idit) {
case 1:
        x = getval("Object exposure time ", 60.0);
        /* For this option convert to equivalent ndit
         * and set single everything else mode */
        input.ndit = (int)(x / input.dittime + 0.5);
        input.nexp = 1;
        input.njitter = 1;
        input.nmicro = 1;
        input.npaw = 1;
        break;
case 2:
        input.snr = getval("Target Signal/Noise ratio ", 20.0);
        /* For this option set single everything */
        input.ndit = 1;
        input.nexp = 1;
        input.njitter = 1;
        input.nmicro = 1;
        input.npaw = 1;
        break;
case 3:
        input.ndit = getival("Observing Strategy:\n"
                "Exposure coadds (ndit) ", 6);
        input.nexp = getival("Exposure Loops (nexp) ", 1);
        input.nmicro =
                getival("Microstepping pattern (1x1 or 2x2) ", 1);
        input.njitter = getival("Jitter Pattern (njitter) ", 1);
        input.npaw = getival("Number of poingings (Npaw) ", 6);
        break;
default:
        (void)fprintf(stderr, "bad strategy\n");
        exit(3);
        break;
}

status = vircam_etc_calc(&input, &output, &status);
(void)printf("ETC Calc retured status = %d\n", status);
if (status) return status;
/* Report outcome: */
(void)printf("gain = %f\n", output.gain);
(void)printf("ditbfactor = %f\n", output.ditbfactor);
(void)printf("detected = %f\n", output.detected);
(void)printf("detected_left = %f\n", output.detected_left);
(void)printf("Aperture loss = %f\n", output.aploss);
(void)printf("s2n_dit = %f\n",output.s2n_dit);
(void)printf("peak = %f\n",output.peak);
(void)printf("sky_detected_pixel = %f\n",output.sky_detected_pixel);
(void)printf("bck_detected_pixel = %f\n",output.bck_detected_pixel);
(void)printf("darkcurr = %f\n",output.darkcurr);
(void)printf("noise_detected_pixel_dit = %f\n",
                output.noise_detected_pixel_dit);
(void)printf("exptime = %f\n",output.exptime);
(void)printf("ditofactor = %f\n",output.ditofactor);
(void)printf("tile_exptime = %f\n",output.tile_exptime);
(void)printf("tile_acttime = %f\n",output.tile_acttime);
(void)printf("dittime = %f\n",input.dittime);

/* Return results in same format as CGI script: */
(void)printf("Results\n"
"Sky+thermal background per pixel per DIT: %6.0f (e-)  %6.0f (ADU)\n"
"Photon+readout noise per pixel per DIT  : %6.0f (e-)  %6.0f (ADU)\n"
 "DIT time factor to background saturation: %6.1f\n"
 "\n"
 "Signal in aperture per DIT              : %6.0f (e-)  %6.0f (ADU)\n"
 "Aperture correction (loss)              : %6.2f (mag)\n"
 "Peak pixel signal in object per DIT     : %6.0f (e-)  %6.0f (ADU)\n"
 "DIT time factor to peak saturation      : %6.1f\n"
 "Object signal-to-noise per DIT          : %7.2f\n"
 "\n"
```

```
        "Time per object for signal-to-noise     : %6.1f (s)\n"
        "Total object signal-to-noise            : %7.2f\n\n"
        "Total exposure time per tile            : %6.1f (s)\n"
        "Total elapsed time per tile             : %6.1f (s)\n"
        "Observing efficiency per tile           : %6.1f (%)\n",

        (output.sky_detected_pixel+output.bck_detected_pixel+output.darkcurr)*
               input.dittime,
        (output.sky_detected_pixel+output.bck_detected_pixel+output.darkcurr)*
               input.dittime/output.gain,
        output.noise_detected_pixel_dit, output.noise_detected_pixel_dit/
               output.gain,
        output.ditbfactor,
        output.detected_left*input.dittime,
        output.detected_left*input.dittime/output.gain,
        output.aploss,
        output.detected*output.peak*input.dittime,
        output.detected*output.peak*input.dittime/output.gain,
        output.ditofactor,
        output.s2n_dit,
        output.exptime,
        output.s2n,
        output.tile_exptime,
        output.tile_acttime,
        output.tile_exptime/output.tile_acttime*100.0);


        return status;
}
```

```
/* Master procedure to perform Exposure-Time Calculations for VIRCAM
 * 2005-05-25 Peter Bunclark Adapted from original Perl script by
 *                          Mike Irwin and Jonathan Irwin.
 */
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "vircam_etc.h"

#define NWAVE 2500            /* number of wavelength samples */

int
vircam_etc_calc(vircam_etc_input_t * inp, vircam_etc_output_t * out,
               int *status)
{
        double          dittime;

        double           wave[NWAVE];
        double           qecurve[NWAVE];
        double           qecurveb[NWAVE];

        double           f_nu[NWAVE];
        double           f_lam[NWAVE];
        double           ab_mag[NWAVE];

        double           sf_nu[NWAVE];
        double           sf_lam[NWAVE];
        double           sab_mag[NWAVE];

        double           bf_nu[NWAVE];
        double           bf_lam[NWAVE];
        double           bab_mag[NWAVE];

        double           photons[NWAVE];
        double           sphotons[NWAVE];
        double           bphotons[NWAVE];

        double           electrons[NWAVE];
        double           selectrons[NWAVE];
        double           belectrons[NWAVE];
```

```
/* Constants: */
double          beta = 2.0;   /* Moffat Function beta parameter */
double          pixel = 0.34; /* pixel size in arcsec                */
double          saturation = 1.5e5;   /* saturation in e-    */
double          ronoise = 10.0;       /* readout noise e-/pixel      */
double          darkcurr = 1.0;       /* average dark current e-/pixel */
double          gain = 3.0;   /* gain in e-/ADU              */

/* Assorted overheads: */
double          odit = 1.0;   /* reset/readout time          */
double          omicro = 4.0; /* time for microstep          */
double          ojitter = 6.0;        /* time for jitterstep         */
double          opaw = 10.0;  /* time for new pointing       */
double          oslew = 20.0; /* time for short slew         */
double          osave = 5.0;  /* disk save time              */

/* Set up wavelength range: */
double          wavestart = 3000.0;   /* start wavelength A   */
double          wavestep = 10.0;      /* wavelength step unit A      */

int             i;
double          filtwave, filtwidth, fnu;
double          alm, cumulative, aploss, radius, peak;
double          sky_ab, salpha;
double          brefwave, btheta, babmag;
double          pmin, pmax;
double          lambda;

double          detected;
double          sky_detected;
double          bck_detected;
double          emin;
double          emax;
double          teleaper;

double          sky_detected_aperture;
double          sky_detected_par;
double          sky_detected_aper;
double          bck_detected_aper;
double          rovar_aper;
double          darkvar_aper;

int             ntot;

/* If we enter with bad status, exit immediately: */
if (*status != 0)
        return *status;

dittime = inp->dittime; /* don't propagate changes to dittime */

for (i = 0; i < NWAVE; i++)
        wave[i] = (double) (i + 1) * wavestep + wavestart;

/*
 * Get combined transmission and qe factors and filter effective
 * lambda
 */
*status = vircam_etc_getqecurve(qecurve, qecurveb, wave,
                          NWAVE, &filtwave, &filtwidth, inp->band,
                             inp->airmass, status);
if (*status != 0)
        return *status;

/* Sort out magnitudes: */
switch (inp->imag) {
case 1:
        inp->abmag = inp->vmag + inp->vegatoab;
        break;
case 2:
        break;
case 3:
        fnu = (inp->flux / filtwidth) * filtwave * filtwave
              / (c * 1.0e8);
```

```
                inp->abmag = -2.5 * log10(fnu) - 48.60;
                break;
        default:
                *status = VIRCAM_ETC_NOMAGTYPE;
                return *status;
        }

        /* Compute aperture loss for Moffat */
        alm = 0.5 * inp->seeing / sqrt(pow(2.0, 1.0 / beta) - 1.0);
        cumulative = 1.0 - pow(1.0 + pow(0.5 * inp->aperture / alm, 2.0),
                                1.0 - beta);
        aploss = -2.5 * log10(cumulative);
        if (inp->ispat == 2)
                aploss = 0.0;

        /* Compute approx central flux pixel for unit total flux Moffat */
        radius = pixel / sqrt(M_PI);
        peak = 1.0 - pow(1.0 + pow(radius / alm, 2.0), 1.0 - beta);
        if (inp->ispat == 2)
                peak = pixel * pixel
                        / (M_PI * pow(0.5 * inp->aperture, 2.0));

        if (inp->blackbody) {
                inp->refwave = filtwave;
                *status = vircam_etc_bbody(inp->theta, inp->abmag,
                    inp->refwave, wave, f_nu, f_lam, ab_mag, NWAVE, status);
                if (*status)
                        return *status;
        }
        if (inp->powerlaw) {
                inp->refwave = filtwave;

                if (inp->iflav == 2)
                        inp->alpha = 2.0 - inp->alpha;       /* convert to f_nu      */
                else if (inp->iflav == 3)
                        inp->alpha -= 1.0;

                *status = vircam_etc_plaw(inp->alpha, inp->abmag, inp->refwave,
                                wave, f_nu, f_lam, ab_mag, NWAVE, status);
                if (*status != 0)
                        return *status;
        }
        if (inp->emission) {
                *status = vircam_etc_emiss(inp->lineflux, &(inp->abmag),
                                inp->refwave, inp->sigma, wave,
                                f_nu, f_lam, ab_mag, NWAVE, status);
                if (*status != 0)
                        return *status;
        }
        /* Make up a pseudo sky spectrum */
        sky_ab = inp->skymag + inp->vegatoab; /* sky AB mag/sq arcsec */
        salpha = 3.0;

        *status = vircam_etc_plaw(salpha, sky_ab, filtwave, wave, sf_nu, sf_lam,
                                sab_mag, NWAVE, status);
        if (*status != 0)
                return *status;

        /* Add in a pseudo thermal background: */
        brefwave = 25000.0;
        btheta = 300.0;
        babmag = 15.0;

        *status = vircam_etc_bbody(btheta, babmag, brefwave, wave,
                                bf_nu, bf_lam, bab_mag, NWAVE, status);
        if (*status != 0)
                return *status;

        /* convert signals to photons/cm**2/s/A              */
        pmin = 0.0;
        pmax = -1.0;

        for (i = 0; i < NWAVE; i++) {
```

```
                lambda = wave[i];
                photons[i] = f_lam[i] * lambda / (h * c * 1.0e8);
                sphotons[i] = sf_lam[i] * lambda / (h * c * 1.0e8);
                bphotons[i] = bf_lam[i] * lambda / (h * c * 1.0e8);
                pmax = photons[i] > pmax ? photons[i] : pmax;
        }

/* Now convert potential photons into captured photons (ie electrons): */
        detected = 0.0;
        sky_detected = 0.0;
        bck_detected = 0.0;
        emin = 0.0;
        emax = -1.0;
        teleaper = M_PI * pow(0.5 * TELEDIAM * 100.0, 2); /* Aperture cm**2 */

        for (i = 0; i < NWAVE; i++) {
                electrons[i] = photons[i] * qecurve[i] * teleaper;
                selectrons[i] = sphotons[i] * qecurve[i] * teleaper;
                belectrons[i] = bphotons[i] * qecurveb[i] * teleaper;
                emax = electrons[i] > emax ? electrons[i] : emax;
                detected += wavestep * electrons[i]; /* total no. detected/s */
                sky_detected += wavestep * selectrons[i]; /* e-/sq arcsec/s */
                bck_detected += wavestep * belectrons[i]; /* e-/sq arcsec/s */
        }

        /* Apply Extinction Correction to object: */
        detected *= pow(10.0, -0.4 * inp->extinction);
        if (inp->ispat == 2)
                detected *= M_PI * pow(0.5 * inp->aperture, 2);

        /* calculate output parameters: */
        out->detected_left = detected * pow(10.0, -0.4 * aploss);
        out->ditofactor = saturation / (detected * peak * dittime);
        out->sky_detected_pixel = sky_detected * pixel * pixel;
        out->bck_detected_pixel = bck_detected * pixel * pixel;
        out->noise_detected_pixel_dit = sqrt(
                                   dittime * out->sky_detected_pixel +
                                   dittime * out->bck_detected_pixel +
                              dittime * darkcurr + ronoise * ronoise);

        sky_detected_aper = sky_detected * M_PI *
                pow(0.5 * inp->aperture, 2);
        bck_detected_aper = bck_detected * M_PI *
                pow(0.5 * inp->aperture, 2);
        out->ditbfactor = saturation / ((out->sky_detected_pixel +
                        out->bck_detected_pixel + darkcurr) * dittime);
        rovar_aper = ronoise * ronoise * M_PI * pow(0.5 * inp->aperture / pixel, 2);
        darkvar_aper = darkcurr * M_PI * pow(0.5 * inp->aperture / pixel, 2);
        out->s2n_dit = out->detected_left * dittime /
                sqrt(dittime * out->detected_left +
                     dittime * sky_detected_aper +
                     dittime * bck_detected_aper +
                     dittime * darkvar_aper + rovar_aper);
        ntot = 1;
        switch (inp->idit) {
        case 1:
                out->exptime = inp->ndit * dittime;
                break;
                /* For s:n option compute equivalent ndit     */
        case 2:
                out->exptime = dittime * pow(inp->snr /
                                          out->s2n_dit, 2);
                inp->ndit = (int) (out->exptime /
                                  (dittime * inp->nexp * inp->njitter
                             * inp->nmicro * inp->nmicro * inp->npaw)) + 1;
                dittime = out->exptime /
                        (inp->ndit * inp->nexp
                         * inp->njitter * inp->nmicro * inp->nmicro
                         * inp->npaw);
                break;
        case 3:
                out->exptime = inp->ndit * dittime
                        * inp->nexp * inp->njitter
```

```
                    * inp->nmicro * inp->nmicro * inp->npaw / 3;
                /* No. of exposures     */
                ntot = inp->nexp * inp->njitter
                    * inp->nmicro * inp->nmicro * inp->npaw / 3;
                break;
        default:
                *status = VIRCAM_BAD_SETUP;
                return *status;
        }


        out->s2n = out->detected_left * out->exptime / sqrt(out->exptime
                                                * out->detected_left +
                                        out->exptime * sky_detected_aper +
                                        out->exptime * bck_detected_aper +
                                            out->exptime * darkvar_aper +
                                          inp->ndit * ntot * rovar_aper);

        /* Example of overheads: */
        out->tile_exptime = dittime * inp->ndit * inp->nexp
                * inp->nmicro * inp->nmicro * inp->njitter
                * inp->npaw;
        out->tile_acttime = (odit + (dittime + odit) * inp->ndit
                        + osave) * inp->nexp
                * inp->nmicro * inp->nmicro * inp->njitter
                * inp->npaw +
                omicro * (inp->nmicro * inp->nmicro - 1) *
                inp->njitter * inp->npaw +
                ojitter * (inp->njitter - 1) * inp->npaw +
                opaw * (inp->npaw - 1) + oslew;

        /* stuff return vector: */
        out->gain = gain;
        out->detected = detected;
        out->aploss = aploss;
        out->peak = peak;
        out->darkcurr = darkcurr;

        /* Plot out results: */
        *status = vircam_etc_plot(inp->outdir, NWAVE, wave,
                                inp->abmag,
                                ab_mag, sab_mag, bab_mag,
                                photons, qecurve, electrons, status);

        return *status;
}
```

```
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "vircam_etc.h"

int     vircam_etc_getqecurve(double qecurve[], double qecurveb[],
                double wave[],
                int npt, double *filtwave, double *filtwidth,
                char *band, double airmass, int *status)
{
        double wavelow, wavehigh, lambda, qenorm;
        double *filter, *tele, *atmos,*detector;
        int i;

        if(*status != 0 ) return *status;

        /* Compute crude approximations to them for now: */
        if (strcmp(band, "Z") == 0) {
                wavelow = 8300.0;
                wavehigh = 9250.0;
        } else if (strcmp(band, "Y")== 0) {
                wavelow = 9700.0;
                wavehigh = 10700.0;
        } else if (strcmp(band, "J")== 0) {
```

```
                wavelow = 11700.0;
                wavehigh = 13300.0;
        } else if (strcmp(band, "H") == 0) {
                wavelow = 14900.0;
                wavehigh = 17800.0;
        } else if(strcmp(band, "K") == 0) {
                wavelow = 20300.0;
                wavehigh = 23700.0;
        } else {
                *status = VIRCAM_ETC_NOBAND;
                return *status;
        }

        filter  = (double *) malloc(npt * sizeof(double));
        tele    = (double *) malloc(npt * sizeof(double));
        atmos   = (double *) malloc(npt * sizeof(double));
        detector= (double *) malloc(npt * sizeof(double));

        for (i=0; i<npt; i++) {
                lambda = wave[i];
        if(lambda > wavelow && lambda < wavehigh) {
                filter[i] = 0.8;
        }
        else {
                filter[i] = 0.0;
        }
        tele[i] = 0.85;
        atmos[i] = pow(0.7,(airmass - 1.0));  /* effect of airmass */
        detector[i] = 0.85;
        }

        /* Combine and compute effective wavelength of band: */
        *filtwave = 0.0;
        qenorm = 0.0;
        for (i=0; i<npt; i++) {
                qecurve[i] = filter[i] * detector[i] * tele[i] * atmos[i];
                qecurveb[i] = filter[i] * detector[i];
                *filtwave += qecurve[i] * wave[i];
                qenorm += qecurve[i];
        }
        *filtwave /= qenorm;
        *filtwidth = wavehigh - wavelow;      /* for flux normalisation */

        free(filter);
        free(tele);
        free(atmos);
        free(detector);

        return *status;
}
```

```
#include <stdio.h>
#include <math.h>
#include "vircam_etc.h"

int vircam_etc_bbody(double theta, double abmag, double refwave,
                double *wave, double *f_nu, double *f_lam,
                double *ab_mag, int npt, int *status)
{
        double Pconst, eta;
        double fnu, aboff, abscl;
        int i;
        double lambda, nu;

        /* return immediately if status is bad: */
        if(*status != 0 ) return *status;

        /* Plank formula constant for wavelength in Anstroms: */
        Pconst = 1.0e8 * h * c/k;
        eta = Pconst/(theta*refwave);
        eta = eta > 80.0 ? 80.0 : eta;
```

```
        fnu = 2.0 * M_PI * h * c * pow(k * theta/(c * h), 3) *
                        pow(eta, 3) / (exp(eta) - 1.0);
        aboff = -2.5 * log10(fnu) - 48.60 - abmag;
        abscl = pow(10.0, 0.4 * aboff);

        for(i=0; i<npt; i++) {
                lambda = wave[i];             /* wavelenght in A     */
                nu = 1.0e8 * c/lambda;        /* frequency in Hz     */
                eta = Pconst/(theta*lambda);
                eta = eta > 80.0 ? 80.0 : eta;        /* Avoid overflows     */
                f_nu[i] = abscl * 2.0 * M_PI * h * c *
                        pow(k * theta/(c * h), 3) * pow(eta, 3) /
                        (exp(eta) - 1.0);
                f_lam[i] = f_nu[i] * nu*nu / (1.0e8 * c);
                ab_mag[i] = -2.5 * log10(f_nu[i]) - 48.60;
        }

        return *status;
}
```

```
#include "vircam_etc.h"
#include "math.h"

int vircam_etc_plaw(double alpha, double abmag, double refwave,
                      double wave[], double f_nu[], double f_lam[],
                      double ab_mag[], int npt, int *status)
{
        double  nu;
        double  fnu;
        double  aboff;
        double  abscl;
        double  lambda;
        int     i;

        /* If bad status input, return immediately: */
        if(*status != 0) return *status;

        /* Set ABmag reference point: */
        nu = 1.0e8 * c / refwave;     /* frequency in Hz     */
        fnu = pow(nu, -alpha);
        aboff = -2.5 * log10(fnu) - 48.60 - abmag;
        abscl = pow(10.0, 0.4 * aboff);

        for(i=0; i<npt; i++) {
                lambda = wave[i];             /* wavelenght in A     */
                nu = 1.0e8 * c / lambda;      /* frequency in Hz     */

                f_nu[i] = abscl * pow(nu, -alpha);
                f_lam[i] = abscl * pow(nu, -alpha + 2.0) / (c * 1.0e8);
                ab_mag[i] = -2.5 * log10(f_nu[i]) - 48.60;
        }

        return *status;
}
```

```
#include "vircam_etc.h"
#include "math.h"

int vircam_etc_emiss(double flux, double *abmag, double refwave, double sigma,
                      double wave[], double f_nu[], double f_lam[],
                      double ab_mag[], int npt, int *status)
{
        double  nconst;
        double  lambda;
        double  arg;
        int     i;

        /* If bad status input, return immediately: */
        if(*status != 0) return *status;
```

```
        /* Normalising constant for flux:     */
        nconst = flux / sqrt(2.0 * M_PI * sigma*sigma);

        /* Set ABmag reference point:         */
        *abmag = -2.5 * log10(nconst * refwave*refwave / (c * 1.0e8)) - 48.60;

        for (i=0; i<npt; i++) {
                lambda = wave[i];      /* wavelength in A     */

                f_lam[i] = nconst * exp(-((lambda-refwave)*(lambda-refwave))
                                / (2.0 * sigma*sigma));
                f_nu[i] = f_lam[i] * lambda*lambda / (c * 1.0e8);
                if (f_nu[i] > 0.0)
                        ab_mag[i] = -2.5 * log10(f_nu[i]) - 48.60;
                        else
                        ab_mag[i] = 1000.0;
        }

        return *status;
}
```

```
/* Plot out VIRCAM ETC calculations using PGPLOT: */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include <cpgplot.h>

#include "vircam_etc.h"

/* Internal routine to set up each frame: */
int setup_gif(const char dir[], char name[], int *status)
{
        char pgdev[81];
        (void)sprintf(pgdev, "%s/%s/gif", dir, name);
        if(cpgbeg(0, pgdev, 1, 1) != 1)
                return(*status = VIRCAM_PLOT_FAILURE);
        cpgpap(PAP_SIZE, PAP_ASPECT);
        cpgscr(0, 1.0, 1.0, 1.0);
        cpgscr(1, 0.0, 0.0, 0.0);
        return *status;
}

/* Main plotting routine: */
int vircam_etc_plot(const char dir[], int nwave, double wave[],
                double abmag,
                double ab_mag[], double sab_mag[], double bab_mag[],
                double photons[], double qecurve[], double electrons[],
                int *status)
{
        float   wmin, wmax;
        float   fmin, fmax;
        float   pmin, pmax;
        float   emin, emax;
        float   grad, scale;
        int     mid;

        float  *x, *y;
        int     i;

        /* If bad status on entry, exit immediately: */
        if (*status != 0) return *status;

        x = malloc(nwave * sizeof(float));
        y = malloc(nwave * sizeof(float));

        for (i=0; i<nwave; i++) {
                x[i] = wave[i];
                y[i] = ab_mag[i];
        }
```

```
wmin = x[0];
wmax = x[nwave - 1];
fmin = (float)abmag + 5.0;
fmax = (float)abmag - 5.0;

/* Plot Flux, Sky and Background curves:     */
*status = setup_gif(dir, FLUXC_FILE, status);


cpgenv(wmin, wmax, fmin, fmax, 0, 0);
cpglab("Wavelength (Angstroms)",
       "Flux (ABmags)", "Flux, sky and background curves");
mid = nwave/2;
scale = (wmax - wmin) / (fmax - fmin);
scale = scale / (x[mid+1] - x[mid-1]);
grad = (y[mid+1] - y[mid-1]) * scale;
grad = 180.0 * atan(grad) / M_PI;
cpgptxt(x[mid], y[mid], grad, 0.0, "Flux");
cpgline(nwave, x, y);
for(i=0; i<nwave; i++) y[i] = (float)sab_mag[i];
cpgsls(2);
cpgsci(4);
grad = (y[mid+1] - y[mid-1]) * scale;
grad = 180.0 * atan(grad) / M_PI;
cpgptxt(x[mid], y[mid], grad, 0.0, "Sky");
cpgline(nwave, x, y);
for(i=0; i<nwave; i++) y[i] = (float)bab_mag[i];
cpgsls(3);
cpgsci(1);
grad = (y[mid+1] - y[mid-1]) * scale;
grad = 180.0 * atan(grad) / M_PI;
cpgptxt(x[mid], y[mid], grad, 0.0, "Background");
cpgline(nwave, x, y);
cpgsls(1);
cpgend();

/* Plot Photon Curve:  */
*status = setup_gif(dir, PHOTC_FILE, status);

pmin = 10e10; pmax = -pmin;
for(i=0; i<nwave; i++) {
       y[i] = (float)photons[i];
       pmin = pmin < photons[i] ? pmin : photons[i];
       pmax = pmax > photons[i] ? pmax : photons[i];
}
cpgenv(wmin, wmax, pmin, 1.05*pmax, 0, 0);
cpglab("Wavelength (Angstroms)", "No. (photons/cm**2/s/A)",
       "Photon Curve");
cpgline(nwave, x, y);
cpgend();

/* Plot Detected Quantum-Efficiency Curve:   */
*status = setup_gif(dir, DQEC_FILE, status);
for(i=0; i<nwave; i++)
       y[i] = (float)qecurve[i];
cpgenv(wmin, wmax, -0.05, 1.0, 0, 0);
cpglab("Wavelength (Angstroms)", "DQE", "DQE Curve");
cpgline(nwave, x, y);
cpgend();

/* Plot Electrons Curve:      */
*status = setup_gif(dir, ELEC_FILE, status);
emin = 10e10; emax = -emin;
for(i=0; i<nwave; i++) {
       y[i] = (float)electrons[i];
       emin = emin < electrons[i] ? emin : electrons[i];
       emax = emax > electrons[i] ? emax : electrons[i];
}
cpgenv(wmin, wmax, emin, 1.05*emax, 0, 0);
cpglab("Wavelength (Angstroms)", "No. (e-/s/A)",
       "Electrons Curve");
cpgline(nwave, x, y);
```

```
        cpgend();

        free(x);
        free(y);

        return *status;
}
```

```
/* include file to support VIRCAM Exposure-Time Calculator:
 * 2005-05-25 Peter Bunclark
 */
typedef struct {
        char            outdir[80];     /* directory to write output files */
        int             powerlaw;       /* boolean Power Law           */
        int             blackbody;      /* boolean Black Body          */
        int             emission;       /* boolean Emission            */
        int             iflav;          /* Power-Law type              */
        double          alpha;          /* Power-Law parameter         */
        double          theta;          /* Black-Body parameter                */
        double          refwave;        /* AB Mag ref wavelength       */
        double          sigma;          /* Emission line-width         */
        double          lineflux;       /* Line flux                   */
        int             imag;           /* Magnitude-type switch       */
        double          vmag;           /* Vega Magnitude              */
        double          abmag;          /* AB Magnitude                        */
        double          flux;           /* Flux                        */
        int             ispat;          /* source-geometry switch      */
        double          aperture;       /* aperture diameter           */
        char            band[32];       /* Filter                      */
        double          skymag;         /* sky brightness mag/sq arcsec        */
        double          vegatoab;       /* Vega->AB mag conversion     */
        double          airmass;        /* Airmass                     */
        double          seeing;         /* Seeing FWHM                 */
        double          extinction;     /* Extinction [mag]            */
        double          dittime;        /* Digitial Intigration Time   */
        int             idit;           /* setup switch                        */
        int             ndit;           /* Number of DITs/exp            */
        double          snr;            /* Required Signal/Noise       */
        int             nexp;           /* Number of exposures         */
        int             nmicro;         /* Numer of microsteps         */
        int             njitter;        /* Number of jitters           */
        int             npaw;           /* Number of "paw prints"      */
} vircam_etc_input_t;

typedef struct {
        double          gain;           /* Gain e-/ADU                 */
        double          ditbfactor;     /* DIT time factor to backg sat        */
        double          detected;       /* */
        double          detected_left;  /* */
        double          aploss;         /* Aperture loss               */
        double          s2n_dit;        /* object S/N per DIT          */
        double          s2n;            /* S/N                         */
        double          peak;           /* */
        double          sky_detected_pixel;
        double          bck_detected_pixel;
        double          darkcurr;
        double          noise_detected_pixel_dit;
        double          exptime;        /* Total exposure time         */
        double          ditofactor;     /* DIT time to peak satn factor        */
        double          tile_exptime;   /* total exp time per tile     */
        double          tile_acttime;   /* total elapsed time per time         */
} vircam_etc_output_t;

int vircam_etc_calc(vircam_etc_input_t *, vircam_etc_output_t *, int *);
int vircam_etc_getqecurve(double *, double *, double *, int, double *,
        double *, char *, double, int *);
int vircam_etc_bbody(double, double, double, double *, double *,
                double *, double *, int, int *);
int vircam_etc_plaw(double, double, double, double *, double *, double *,
                        double *, int, int *);
int vircam_etc_emiss(double, double *, double, double, double *,
```

```
                      double *, double * , double *, int, int *);
int vircam_etc_plot(const char *, int, double *,
                      double,
                      double *, double *, double *,
                      double *, double *, double *, int *);

/* Error Status values: */
#define VIRCAM_ETC_NOBAND 1
#define VIRCAM_ETC_NOMAGTYPE 2
#define VIRCAM_BAD_SETUP 3
#define VIRCAM_PLOT_FAILURE 4

#define TELEDIAM 4.0          /* Telescope aperture/m */

/* Physical Constants: */
#define k 1.38066e-16         /* Boltzmann's Constant erg K**-1 */
#define h 6.62618e-27         /* Plank's Constant erg s */
#define c 2.997925e10         /* Speed of light cm s**-1 */

/* Graph files: */
#define FLUXC_FILE "flux.gif"
#define PHOTC_FILE "photon.gif"
#define DQEC_FILE  "dqe.gif"
#define ELEC_FILE  "electron.gif"
#define PAP_SIZE 8.0
#define PAP_ASPECT 0.8
```

---

oOo