Wide-field Infrared Survey Explorer (WISE)

Tempcal Subsystem Design Document

Version 1.64

12 February 2010

Prepared by: John W. Fowler & Frank. J. Masci

Infrared Processing and Analysis Center
California Institute of Technology

WSDC D-D011

Concurred By:

Roc Cutri, WISE Science Data Center Manager

Tim Conrow, WISE Science Data Center System Architect

John Fowler, WISE Science Data Center Tempcal Cognizant Programmer

Frank Masci, WISE Science Data Center Tempcal Cognizant Engineer

Revision History

Date	Version	Author	Description
28 July 2008	1.0	J. W. Fowler & F. J. Masci	Initial Draft
8 July 2009	1.1-1.25	J. W. Fowler & F. J. Masci	Added debug aids, bug fixes, more robust pixel offset estimation, SubOff option
11 September 2008	1.26	J. W. Fowler & F. J. Masci	Installed standard WSDC error message handling
10 October 2008	1.27	J. W. Fowler & F. J. Masci	Added option to skip sky-offset processing
16 July 2009	1.3-1.61	J. W. Fowler & F. J. Masci	Added separate thresholds for frame outliers, FITS images of frame limits of transient runs, QA and debug statistics files, frame partitioning for transient analysis, latent tagging
6 August 2009	1.62	J. W. Fowler & F. J. Masci	Switched from FITS header keyword UTCS_OBS to UNIXT for time order.
17 August 2009	1.63	J.W. Fowler	Added Nframes to Badstats.tbl header
12 February 2010	1.64	J.W. Fowler	Fixed array overflow; expanded formats in QA and Badstats to handle larger numbers.

Table of Contents

1 Introduction

1.1 Subsystem Overview

This document presents the requirements, design, algorithms, and state of implementation of the Tempcal (Temporary Effects Calibration) subsystem of the WSDC data processing system. Tempcal runs offline on all or part of a single scan.

1.1.1 Requirements

Tempcal is required to compute a pixel 'sky-offset' image and to identify and flag *persistent* new "bad" pixels in a stack of *N* consecutive frames along a scan. This is the essence of dynamic pixel masking; tempcal updates FITS mask images by turning on bits in the pixel data to indicate conditions that it diagnoses. The Level 4 requirements supported by this processing are as follows.

> L4WSDC-012: Flux measurements in the WISE Source Catalog shall have a SNR of five or more for point sources with fluxes of 0.12, 0.16, 0.65 and 2.6 mJy at 3.3, 4.7, 12 and 23 micrometers, respectively, assuming 8 independent exposures and where the noise flux errors due to zodiacal foreground emission, instrumental effects, source photon statistics, and neighboring sources (*traceable to Level-1*).
>
> L4WSDC-013: The root mean square error in relative photometric accuracy in the WISE Source Catalog shall be better than 7% in each band for unsaturated point sources with SNR>100, where the noise flux errors due to zodiacal foreground emission, instrumental effects, source photon statistics, and neighboring sources. This requirement shall not apply to sources that superimposed on an identified artifact (*traceable to Level-1*).
>
> L4WSDC-024: The WSDC shall generate and maintain an archive of the calibrated, single epoch WISE images for the duration of the project for use by the Project Team. The purposes of this archive are quality assurance, transient analysis and moving object identification. Self-derived Demonstration Define duration of project.
>
> L4WSDC-037: The WSDC Pipelines subsystem shall convert raw WISE science and engineering data into calibrated images and extracted source lists from which the preliminary and final WISE data products will be derived.

L4WSDC-039: Within 3 days from receipt of a given data set at the WSDC all data shall be processed through the WSDS Scan/Frame pipeline which performs basic image calibration and source extraction from on images from individual orbits. The results of this processing step shall be Level 1 source extractions and image data, which are loaded into the WISE Level 1 extracted Source Working Database (L1WDB) and Image Archive allowing access by the WISE Science Team for external quality assessment.

L4WSDC-042: The WSDS Pipeline processing shall remove the instrumental signature from Level 0 image frames.

L4WSDC-062: The WSDC shall perform quality analysis of all WISE science data and make reports available on a regular basis.

The tempcal module is run offline on a list of preselected survey data images. Input frames will have already been precalibrated, e.g., dark subtracted, linearized and flat-fielded. The units of the pixel values are native WISE "*scaled*-slope" units as computed onboard for detector ramps. More specifically, they will be in units of *scaled* DN/SUR, where DN means Data Number and SUR means Sample Up the Ramp.

Short-term variations in the bias, dark (and possibly gain) structure over an array will not be captured by ground calibrations. The ground calibrations are designed to remove instrumental signatures that are essentially static in the long term. If the short-term systematic variations are not removed, they will persist as residuals and impact photometric accuracy. These can be corrected by computing a robust estimate of a zero-mean (or zero-median) background image from $N \sim 50$ - 100 frames within a moving block window along the WISE orbit, then subtracting this from all the frames in that window. The tempcal module will create the sky-offset image calibration product only, not apply it. It is estimated that at least 50 - 100 frames will be needed to filter out sources reliably across all bands. A window that's too big may miss the short-term instrumental variations sought for. An important assumption is that the transient bias/dark structure is approximately constant over this window span. A schematic of the concept is shown in Figure 1.

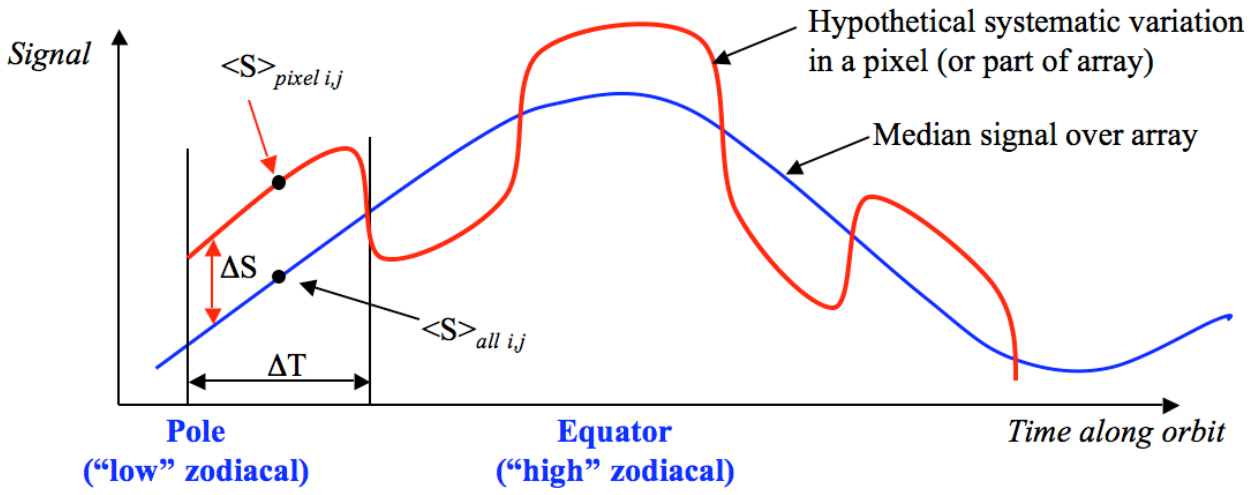


Figure 1. Sky-Offset Schematic

The labels in Figure 1 are defined as follows:

ΔT = timescale of possible systematic variation
$\langle S \rangle_{pixel\ i,j}$ = median or mean sky signal in single pixel *i*, *j* over stack of *N* frames in ΔT
$\langle S \rangle_{all\ i,j}$ = median or mean sky signal over all pixels and *N* frames in ΔT
Sky-offset correction : $\Delta S_{i,j,\Delta T} = \langle S \rangle_{pixel\ i,j} - \langle S \rangle_{all\ i,j}$

The number of frames must satisfy : $N_{min} \leq N \leq N_{\Delta T}$, where N_{min} is the minimum needed to filter out stars. If $N_{\Delta T} < N_{min}$ (fast instrumental variations), then this method can't be used. The plan is to determine optimal frame windows N_{min} and $N_{\Delta T}$ in IOC.

Note: the angled brackets can represent either a *trimmed mean*, *median* or whatever robust estimator is used to estimate the sky background seen by the pixel in the *N*-frame stack.

This same module will also perform dynamic pixel masking. The rationale behind this is that it is the only place in the infrastructure (so far) that gathers all frames pertaining to the same time interval, ΔT, along a scan. As discussed above, this time-interval may contain 50 - 100 frames, or ≲40% of a scan (North-to-South ecliptic pole). If a pixel suddenly becomes hot, it may persist in this state for the entire interval ΔT, or just part of it for a duration δ*t*. Bad pixels (where the criteria for 'bad' are defined below) occur systematically at the same location in pixel space, whereas astronomical sources do not. One can therefore envisage a method where if a pixel is detected as an outlier with respect to its neighbors in the *same* frame, and it persists in this state for a time δ*t* in subsequent frames of the stack, then it can be identified as a transient bad pixel. Pixels identified as permanently bad *a priori* (e.g., on the ground) are omitted before performing the 'dynamic' bad-pixel search.

1.1.2 Liens

None at this time.

1.2 Applicable Documents

This subsystem conforms to the specifications in the following project documents:

- WISE Science Data Center Functional Requirements Document, WSDC D-R001
- WISE Science Data System Functional Design, WSDC D-D001
- Software Management Plan, WSDC D-M003
- Instrumental Calibration Plans and Considerations: http://web.ipac.caltech.edu/staff/fmasci/home/wise/SingleOrbit_Cal.html
- Infrastructure and Instrumental Calibration Scan Pipeline: http://web.ipac.caltech.edu/staff/fmasci/home/wise/ScanPL_instrumental_cal.pdf
- Software Interface Specification (ICL01) Frame Processing Status Mask: http://web.ipac.caltech.edu/staff/fmasci/home/wise/InstruCal01.txt
- Software Specifications for the *tempcal* module: http://web.ipac.caltech.edu/staff/fmasci/home/wise/tempcal_specs.pdf

1.3 Acronyms

2MASS	Two-Micron All-Sky Survey
DN	Data Number
FITS	Flexible Image Transport System
FRD	Functional Requirements Document
IOC	In-Orbit Checkout
NAXIS1	Number of columns in an image
NAXIS2	Number of rows in an image
Nframes	Number of science images to be processed
SDS	Subsystem Design Specification
SIS	Software Interface Specification
SUR	Sample Up the Ramp
W1	WISE wavelength channel 1, 3.4 microns
W2	WISE wavelength channel 2, 4.6 microns
W3	WISE wavelength channel 3, 12 microns
W4	WISE wavelength channel 4, 22 microns
WISE	Wide-field Infrared Survey Explorer
WSDC	WISE Science Data Center
WSDS	WISE Science Data System

2 Input

2.1 Control Input

Tempcal reads control input in the form of Fortran Namelist files and command-line parameters. For control parameters included in both command line and namelist inputs, the command line inputs override.

2.1.1 Tempcal Command-Line Parameters

The command-line parameters for Tempcal are given by its tutorial display:

```
tempcal version: 1.6  A90716 - execution begun on 17-07-09 at  9:26:55

Usage: tempcal

 -f1 <inp_img_list_fname>  (Required; list of pre-calibrated frames in
                            FITS format)

 -f2 <inp_mask_list_fname> (Optional; list of bad-pixel masks in 32-bit INT
                            FITS format; only values 0 -> 2^31 are used)

 -f3 <inp_unc_list_fname>  (Optional; list of uncertainty images in FITS
                            format)

 -lt <lower_threshold>     (Optional; lower-tail threshold for in-frame
                            outlier [candidate bad-pixel and global sky
                            estimate] detection; Default = 5 sigma)

 -ut <upper_threshold>     (Optional; upper-tail threshold for in-frame
                            outlier [candidate bad-pixel and global sky
                            estimate] detection; Default = 5 sigma)

 -ng <num_grids>           (Optional; number of partitions per axis for
                            temporal outlier [candidate bad-pixel]
                            detection; Default = 3)

 -lts <lower_SO_thresh>    (Optional; lower-tail threshold for temporal
                            outlier [relative to sky] detection;
                            Default = 5 sigma)

 -uts <upper_SO_thresh>    (Optional; upper-tail threshold for temporal
                            outlier [relative to sky] detection;
                            Default = 5 sigma)

 -pn <frame_persist_num>   (Optional; minimum number of consecutive frames
                            in time-ordered sequence of stack for which an
                            'outlier pixel' must persist to be declared bad;
                            Default = number of frames in input list)

 -mp <min_pix_in_stack>    (Optional; minimum number of pixels in a stack
                            to be usable for robust estimation; Default = 5)

 -tlat <Qmax>              (Optional; maximum Q probability to tag bad-pixel
                            run as latent; default = 0.05)

 -c <chi-square_max>       (Optional; maximum reduced chi-square of sky
                            offset for reliable uncertainty; Default = 3)

 -m <inp_mask_bits>        (Optional; mask template [decimal] specifying
                            bits to flag/omit from processing; Default = 0)
```

```
-p <out_maskdy_bits>      (Required if <-f2> specified; mask template
                           [decimal] specifying bit to set in _specific_
                           input masks for dynamic bad-pixel masking)

-tf 0                     (Optional; turn off transient flagging)

-ts 0                     (Optional; turn off sky-offset computation)

-so 1                     (Optional; subtract the frame offset from
                           each pixel before computing sky offset;
                           Default = 0)

-st 0                     (Optional; do not subtract the partition offset
                           from each pixel before checking for latent
                           decay; Default = 1)

-s <out_maskso_bits>      (Required if <-f2> specified; mask template
                           [decimal] specifying bit to set in _all_ input
                           masks for unreliable/erroneous sky-offset)

-su <out_maskso_bits>     (Required if <-f2> specified; mask template
                           [decimal] specifying bit to set in _all_ input
                           masks for unreliable uncertainty in sky-offset)

-pl <out_maskso_bits>     (Required if <-f2> specified; mask template
                           [decimal] specifying bit to set in affected
                           masks for probable latent contamination)

-o1 <out_skyoff_img>      (Required if sky-offset computation is selected;
                           output sky-offset image FITS filename)

-o2 <out_skyoff_unc_img>  (Required if sky-offset computation is selected;
                           output sky-offset uncertainty image FITS filename)

-o3 <out_chi-square_img>  (Optional; output sky-offset chi-square image
                           FITS filename)

-o4 <out_Nused_img>       (Optional; output image of #pixels used in stack;
                           FITS filename)

-n  <namelist>            (Optional; namelist file name)

-qa <qa_file_name>        (Optional; switch to generate a QA file named
                           as specified)

-d                        (Optional; switch to print debug statements
                           to stdout, ancillary QA to ascii files)

-v                        (Optional; switch to increase verbosity to stdout)
```

2.1.2 Tempcal Namelist Parameters

The Tempcal module optionally reads a NAMELIST file. The name of this file must be given on the command line via the "-n" option. The name of the NAMELIST is tmpcalin. The parameters defined in the NAMELIST are as follows.

Name	Description	Dim	Type	Units	Default
ChiSqMax	Maximum reduced chi-square of sky offset (if prior uncertainties given) or ratio of dynamic range to sky-offset uncertainty (if no prior uncertainties) *not* to flag sky-offset uncertainty as potentially unreliable	1	R*4	-	3.0
ITimSlic1	Lower column number for time-slice image output (see section 4.6)	1	I*4	-	0
ItimSlic2	Upper column number for time-slice image output (see section 4.6)	1	I*4	-	0
JTimSlic1	Lower row number for time-slice image output (see section 4.6)	1	I*4	-	0
JtimSlic2	Upper row number for time-slice image output (see section 4.6)	1	I*4	-	0
Mask	Mask template [decimal] specifying bits to omit from processing	1	I*4	-	0
MinPersist	Minimum number of consecutive frames in time-ordered sequence of stack for which an 'outlier pixel' must persist to be declared bad	1	I*4	-	Nframes
MinPix	Minimum number of pixels In a stack for the stack to be usable for robust estimation	1	I*4	-	5
NamWrt	If T, namelist will be written to stdout	1	L*4	-	F

Name	Description	Dim	Type	Units	Default
Ng	Number of grid divisions per image axis for partitions in which transient behavior is analyzed	1	I*4	-	3
Qmax	Maximum Q probability (i.e., 1-P, P = cumulative probability of binomial distribution) of number of chronological pixel value drops to flag as latent	1	R*8	-	0.05
SkpST	If T, sky-offset computation and transient flagging will be skipped (to save time when only time-slice images are desired)	1	L*4	-	F
SubOff	If T, each value for a given pixel will have the corresponding frame offset subtracted before the pixel's sky offset is computed; if F, this is not done, and instead the global frame offset is subtracted from the pixel's absolute offset to obtain the pixel's sky offset	1	L*4	-	F
SubOffTran	If T, each value for a given pixel will have the corresponding partition offset subtracted before latent testing	1	L*4	-	F
ThrshHi	Positive offset in sigma units from frame offset limiting non-outlier pixel range for bad-pixel identification	1	R*4	-	5.0
ThrshHiS	Positive offset in sigma units from frame offset limiting non-outlier pixel range for sky-offset computation	1	R*4	-	5.0
ThrshLo	Negative offset in sigma units from frame offset limiting non-outlier pixel range for bad-pixel identification	1	R*4	-	5.0
ThrshLoS	Negative offset in sigma units from frame offset limiting non-outlier pixel range for sky-offset computation	1	R*4	-	5.0

2.2 ASCII Input

Tempcal reads one to three ASCII (text) file lists of FITS file names corresponding to FITS images. One is required, and its name is specified on the command line via the "-f1" flag; this is the list of names of science images. If the "-f2" flag was used, then a list of mask images is also read, and if the "-f3" flag was used, then a list of uncertainty images is read. These last two lists must be in one-to-one correspondence with the first list, i.e., the n^{th} mask image and the n^{th} uncertainty image must correspond to the the n^{th} science image. Each list is read, the number of lines is counted, and all must have the same number of lines; this number is used for memory allocation. Then the lists are rewound, each line is read, and the corresponding file is read into memory. The file name on each line must be left-justified and is case-sensitive. Path names may be included and are required if the FITS files are not in the working directory.

2.3 FITS Input

For each line in each ASCII file described in section 2.2 above, tempcal reads the FITS file whose name is given. All FITS images must have the same values for the following header parameters: NAXIS (must be 2), NAXIS1, NAXIS2, and BAND. Each frame must have its own parameter UNIXT, which is used to force time order in the frame stack. In addition, the keyword FRSETID is sought in the first input intensity image; if found, it must also be in all subsequent intensity images and is used to construct the FRMIDSEQ keyword for the output sky-offset FITS image.

3 Processing

3.1 Initialization

The tempcal module initializes itself by:

A.) reading and processing its control inputs;

B.) verifying that all required inputs were given;

C.) reading the lists of FITS files and allocating memory;

D.) reading in all specified FITS images;

E.) sorting an index array of UNIXT values;

F.) generating a table of Mmin values (see section 3.4.1) if transient analysis has been selected.

3.2 Sky-Offset Computation

The tempcal module performs sky-offset computation using all unmasked pixels unless this function was deselected ("`-ts 0`" in section 2.1.2). If no mask files were specified, then all pixels are treated as unmasked.

3.2.1 Frame Offset Computation

The first step is to compute a "standard offset" for each science image; by "standard offset", we mean a robust estimate of the representative sky background, the median signal that would be observed if there were no point-like objects or radiation hits. This will be called simply the "offset" for brevity, and this term will be applied to entire frames (spatial distribution of signal), to "partitions" of frames (spatial distribution of signal over separate portions of an entire frame), and also to pixel stacks containing all unmasked samples of a given array pixel (temporal distribution of signal seen by the pixel). We use the term "offset" to avoid the more specific terms such as "median", "trimmed average", etc., which imply specific algorithms that may not be in use. The tempcal code is modularized to facilitate installing different robust estimation algorithms for the "offset" of frames, partitions, and chronological pixel stacks; this is designed to allow different methods to be tried, and any new accepted variations will have a corresponding version of this SDS.

When initialization was performed, all frames to be processed were loaded into a three-dimensional data cube in memory occupying `4*NAXIS1*NAXIS2*Nframes` bytes for the

science images, and the same each for optional uncertainty images and optional mask images. For W1, W2, and W3, NAXIS1 = NAXIS2 = 1016, so for these science images, the amount of memory required in bytes is 4.12904e6*Nframes. For W4, NAXIS1 = NAXIS2 = 508, so for W4 science images, the amount of memory required in bytes is 1.032256e6*Nframes.

An example of a minimal processing run would be 50 W4 frames with no masks or uncertainties. The total input-image memory for this case is 51.6 MB. An example of a typical expected case would be 100 W1 frames including science, uncertainty, and mask images. The total input-image memory for this case is 1.24 GB.

A loop over all frames is executed, and in this loop, the following processing is performed on *each* frame:

- A one-dimensional stack is loaded with all pixel values that are neither masked nor NaN.
- If the number of values in the stack is less than MinPix (see section 2.1.2), an offset value of NaN is returned; otherwise processing continues.
- The stack is sorted in ascending value, and the median value is found.
- The standard deviation of the lower 50%-tile about the median is computed, σ_{50}.
- All values in the stack that are either less than median–ThrshLo*σ_{50} or greater than median+ThrshHi*σ_{50} are compressed out of the stack (see section 2.1.2, ThrshLo, ThrshHi).
- The remaining stack is re-sorted into ascending order, and the new median is found; this is the frame offset.
- The standard deviation about the offset is computed for the values in the compressed stack, σ.
- Frame outlier limits are computed and returned for storage and later use if Ng = 1 (only one partition per frame); if SubOff is T (see section 2.1.2), then these are the lower limit FramLo = -ThrshLo*σ and FramHi = ThrshHi*σ, otherwise they are FramLo = offset - ThrshLo*σ and FramHi = offset + ThrshHi*σ. Note that the purpose of this adjustment is to keep SubOff from affecting *transient* analysis; its purpose is exclusively to reduce the effects of background variation on pixel *offset* estimation.

After all *frame* offsets have been computed, all values that are not NaN are put into a stack, sorted, and the median is computed; this is the *global frame offset* that will be subtracted from each pixel's *raw* offset to obtain the pixel's *sky* offset if SubOff = F (see section 2.1.2). If SubOff = T, then each pixel will have had its corresponding frame offset subtracted *before* going into the stack used to estimate its sky offset, and no global or frame offset will need to be subtracted afterwards.

3.2.2 Pixel Offset Computation

A loop over all array pixels is performed, and inside this loop each pixel is processed as described below.

- The pixel's value in each science frame in the data cube is examined; if no uncertainties were input, then if the pixel value is not NaN and not masked, the value is put into a pixel stack; if uncertainties were input, then these same conditions must be met by the science *and* uncertainty values, and in addition the uncertainty must be greater than zero (including not NaN); if and only if these conditions are met, the science data value and the uncertainty value are both placed into separate stacks; values accepted for a stack are placed there in increasing time order (the pixels are accessed via the index array of sorted UNIXT values), and another index array is filled which specifies from which frame number in the data cube the pixel came; if `SubOff` is T (see section 2.1.2), then each science pixel's corresponding frame offset is subtracted from the science data value as this value is loaded into the stack.
- If the number of values in the stack is less than `MinPix`, values of zero are returned for the sky offset and uncertainty, and if mask inputs were given, the pixel's value in all input masks has bits set for unreliable/erroneous sky offset and uncertainty; if the number of values in the stack is greater than or equal to `MinPix`, processing continues.
- The time-ordered pixel stack is copied into a temporary buffer which is sorted into ascending order, and the median is found.
- The standard deviation of the lower 50%-tile about the median is computed, σ_{50}.
- All values in the stack that are either less than median–`ThrshLoS`*σ_{50} or greater than median+`ThrshHiS`*σ_{50} are compressed out of the stack.
- A new median is found, the median of the remaining stack; this is the pixel's "sky offset" if the frame offsets have already been subtracted off (`SubOff` = T), and otherwise it is the pixel's *raw* offset, and the global frame offset is subtracted from it to obtain the pixel's *sky* offset.
- The uncertainty of the sky offset is found as follows; if pixel uncertainties were input, then the uncertainties corresponding to pixels retained for the sky offset computation (after outlier rejection) are used to compute an inverse-variance uncertainty for the sky offset; if no pixel uncertainties were input, then the sample variance about the offset is computed for the N values in the compressed stack, divided by N-1, and the square root is used as the uncertainty; in either case, the uncertainty is then scaled by $\sqrt{\pi/2}$ to account for the additional uncertainty of the median.
- Sanity checking of the pixel sky offset is performed as follows; if pixel uncertainties were input, then chi-square is computed and the reduced chi-square value must be less than `ChiSqMax` (see section 2.1.2); if pixel uncertainties were not input, then the dynamic range of the compressed stack is divided by the uncertainty of the sky offset, and the ratio must be less than `ChiSqMax`; in either case, if the condition is not satisfied and if masking is being performed, then the bit indicating an unreliable/erroneous sky-offset uncertainty is set for the pixel in all input masks.

3.3 Transient Pixel Identification and Latent Image Tagging

Unless transient identification was deselected (no masks input or "`-tf 0`" was specified on the command line), the following processing is performed in the same loop over pixels as the above section. If frame partitioning was selected, i.e., if `Ng` > 1 (see section 2.1.2; by default, `Ng` = 3), then a modification to that description must be made; the loop in section 3.2.2 is performed *after* the partitions are set up, and what was called "frame outlier limits" in section 3.2.1, FramLo and FramHi, are computed for each partition separately before entering that loop. *This partition processing has no effect on the sky-offset computation*, and so its description was deferred to this section. By the same token, adjustments made to pixel values by `SubOff` have no effect on either transient identification or latent flagging.

3.3.1 Partition Setup And Outlier Limits

If the number of partitioned grids per axis, `Ng`, was specified greater than 1 (default 3), then arrays of pixel coordinates are set up to control loops over partitions in terms of frame pixel coordinates. These arrays are named Ig0, IgF, Jg0, and JgF. A partition is identified by its index on each axis, Ig and Jg. For example the grid corresponding to the 2nd partition in the direction of increasing column number and the 3rd partition in the increasing row direction has Ig = 2 and Jg = 3. A nested loop over this partition would run from J = Jg0(3) to JgF(3) for the outer loop and from I = Ig0(2) to IgF(2) for the inner loop. These partition limits are computed as follows.

$$\Delta C = \frac{N_{cols}}{N_g} \quad , \quad \Delta R = \frac{N_{rows}}{N_g}$$

$$Ig0(1) = 1 \, , \; Jg0(1) = 1$$

loop on $N = 1$ *to* N_g :

$$IgF(N) = round(N \times \Delta C)$$

$$JgF(N) = round(N \times \Delta R)$$

if $N > 1$ *then* $Ig0(N) = IgF(N-1) + 1$

if $N > 1$ *then* $Jg0(N) = JgF(N-1) + 1$

end loop

Since we have $N_{cols} = N_{rows}$, the partitions will never be more than one row or column off from being square, and the entire frame will be covered.

Next, the processing described in section 3.2.1 for an entire frame is repeated for each partition, and what is called the *frame offset* there becomes the *partition offset*, and what are called the *frame outlier limits* there become the *partition outlier limits*. The latter are now based on the robust estimate of spatial pixel variance *in the partition* about the *partition offset*. This compensates for background variations that can be tracked better in the partitions than in entire

frames. Note that this has no effect on the *sky-offset computation*, which remains as described in section 3.2.2; only transient and latent identification are affected by the partitioning.

3.3.2 Mmin Setup

Latent tagging is confined to runs of transient pixel behavior, as described in section 3.3.3 below. Within such runs, it is based on identifying a statistically significant excess of drops in chronologically consecutive pixel values in the stack that was set up as described in section 3.2.1 above. The decision employs the following null hypothesis: whether a pixel's sample-to-sample value increases or decreases within a transient run is equivalent to the flip of a fair coin, with heads corresponding to an increase in value and tails corresonding to a decrease. Thus the signs of the *first differences* in the stack follow the binomial distribution if the null hypothesis is true. If the null hypothesis is false, then the case of interest is when the number of *drops* is excessive, since this is expected to be typical of a latent decay signature. It is desired not to require a specific shape to the decay; although it is plausible that the shape might be that of an exponential decay, or a superposition of such decays with different time constants. The calibration of such specific models is historically difficult and unreliable. The one feature that can be expected with high confidence is an excess of negative first differences. Only after a transient run has been identified is this signature of latent decay sought, and this is done by computing the statistical significance of the number of drops given the number of first differences.

The binomial probability distribution is

$$P(n,m,p) = \frac{n!}{m!(n-m)!} p^m (1-p)^{n-m}$$

where n is the number of first differences, m is the number of drops, and p is the probability of a drop. For a "fair coin" model, $p = ½$, and we can write simply

$$P(n,m) = \frac{n!}{m!(n-m)!} \frac{1}{2^n}$$

Since it may be necessary to evaluate this probability hundreds of thousands of times, and since the possible values of n are known to run from `MinPersist/2`-1 to `NFrames`-1 (see section 2.1.2), during initialization, a table is generated that provides the values for the minimum value of m to reject the null hypothesis as a function of n. The minimum value of m is that for which

$$\sum_{k=0}^{m} P(k,n) \geq 1 - Q_{\max}$$

$$\sum_{k=0}^{m-1} P(k,n) < 1 - Q_{\max}$$

where Q_{max} is the value of the parameter `Qmax` that may be specified on the command line with the "`-tlat`" flag or via namelist (see sections 2.1.1 and 2.1.2) and which defaults to 0.05. These minimum values are stored in an array `Mmin(N)`. For example, given the default value for `Qmax`, the `Mmin(N)` values for `N` = 10 to 20 are:

N	Mmin
10	8
11	8
12	9
13	9
14	10
15	11
16	11
17	12
18	12
19	13
20	14

When a transient run of length `K` is identified as described below, the number of drops `M` is counted and tested for `M` ≥ `Mmin`(`K`-1); if this test is passed, then the transient run is also tagged as a latent, in which case the *first* sample in the transient run is *not* tagged as *either*, since it is apparently the bright source that caused the latent. There is one exception to this: if the transient run began on the very first sample in the stack, then it is more likely that the transient run was already in progress before the first sample than that the transient run coincidentally began on the first sample, and so in this case, all samples in the run are tagged as both transient and latent.

3.3.3 Transient Identification and Decay Tagging

The pixel stack that was prepared as described in section 3.2.1 above is examined in time order. If `SubOff = T`, then the frame offsets will have been subtracted from both the pixel samples and the frame/partition limits; hence `SubOff` plays no role in transient identification, which depends only on a sample's value relative to its limits. This pixel stack will be called PixStak, and the number of samples in it is NPixStak. A second pixel stack named LatStak is also prepared in the same way but with two exceptions: `SubOff` has no effect on it, and if `SubOffTran = T`, then the corresponding *partition offsets* are subtracted from each sample. LatStak is used only to define the signs of the first differences; subtracting off partition offsets generally alters the first differences because the partition offsets generally vary with time. This can affect whether a given first difference is positive or negative, the crucial question for latent identification.

Runs of *consecutive* pixel values that are all outside the outlier limits (FramLo and FramHi, as discussed in section 3.2.1, if `Ng` = 1, otherwise the partition limits discussed in section 3.3.1) are

identified. If the number of such consecutive pixels is at least as large as `MinPersist` (see section 2.1.2), and if mask inputs were given, then the corresponding mask pixels have bits set to indicate transient behavor. If the run began with the first pixel in the stack or ended at the last pixel in the stack, then it need not be as long as `MinPersist`; it need be only at least as long as (`MinPersist+1)/2`. If any part of the stack was flagged as transient, then the entire mask stack for that pixel has the bit set indicating unreliable sky offset. Note the first sample that was tagged as *transient* may be subsequently un-tagged if the run is also identified as *latent*, since presumably it is a legitimate obervation of a bright object that caused the latent behavior. This un-tagging of the first latent/transient sample will be done if and only if it is not also the first sample in the stack, since that precludes knowledge of whether it is the first sample of the bright object.

Transient identification makes use of a simple *finite state machine* to supply a memory of whether a string of outliers is being seen as the pixel stack is traversed chronologically forward. This involves a logical variable named IzOut that indicates that for a given sample, the previous one lay outside the limits, which will be denoted PixLo and PixHi below to indicate frame limits (if `Ng` = 1) or partition limits (if `Ng` > 1). The identification of a transient "run" is accomplished as illustrated in the pseudocode below, which also contains statements in *italic Arial* font whose sole purpose is to support latent flagging.

```
Initialize IzOut to False
Loop over stack chronologically forward from N = 1 to NPixStack
    If (PixStak(N) < PixLo) or (PixStak(N) > PixHi) then      {sample out of limits}
        If not IzOut then                                     {if not already in transient run}
            IzOut = True                                      {change state to "transient run underway"}
            NT1 = N                                           {mark beginning of transient run}
            NDrops = 0                                        {initialize counter for pixel value decrease}
        End if
        NT2 = N                                               {currently  in transient run, tag last sample}
        If (NT2 > NT1)  then                                  {count the number of drops}
            If (LatStak(N) < LatStak(N-1))
            then NDrops ⇐ NDrops+1
        End if
    Else                                                      {pixel sample is in limits}
        If IzOut then                                         {we just ended an out-of-limits run}
            IzOut = False                                     {reset state to "not in transient run"}
            If (NT2-NT1+1 ≥ MinPersist)
             or ((NT1 = 1) and (NT2 ≥ (MinPersist+1)/2)) then   {transient run}
                Latent = (NDrops ≥ Mmin(NT2-NT1))             {check latent}
                If Latent and (NT1 > 1)                       {if latent, don't tag first  }
                then NT1 ⇐NT1+1                               {  sample as transient/latent}
                Set transient mask bits for samples NT1 to NT2
                If Latent, set Latent mask bits for samples NT1 to NT2
            End if
        End if
    End If
End loop
If IzOut and (NPixStak-NT1+1 ≥ (MinPersist+1)/2) then     {if transient run at loop end}
    Latent = (NDrops ≥ Mmin(NT2-NT1))                     {check latent}
    If Latent and (NT1 > 1)      {if latent, don't tag first sample as transient latent}
    then NT1 ⇐NT1+1
    Set transient mask bits for samples NT1 to NT2
    If Latent, set Latent mask bits for samples NT1 to NT2
End if
```

4 Output

4.1 Sky-Offset FITS Output

The main output ("-o1" command-line parameter) is a FITS file containing the sky-offset image. An example header is below. The angle brackets indicate information whose literal content depends on actual execution or generation circumstances.

```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                  -32 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                 1016 / length of data axis 1
NAXIS2  =                 1016 / length of data axis 2
BAND    =                    1 / WISE band number (1, 2, 3 or 4)
NUMINP  =                   70 / Number of input frames used
UTCSBGN =           1260864418 / Earliest UTCS in frame stack [sec]
UTCSEND =           1261051979 / Latest UTCS in frame stack [sec]
FRMIDSEQ=         <'Num..Num'> / Range of frameIDs used
COMMENT  Generated by tempcal vsn 1.25 A80715 on 15-07-08 at 11:46:58
COMMENT  This is a sky offset image
```

Notes:

- BITPIX = -32 refers to single precision floating point.
- For bands 1, 2 and 3, the input frames will have NAXIS1 = NAXIS2 = 1016. For band 4, NAXIS1 = NAXIS2 = 508.
- The UTCSBGN, UTCSEND keywords specify the start/end observation time of frames in the input ensemble. These time tags will be available in the input frame headers [e.g., UNIXT].
- The FRMIDSEQ specifies the range of input frame IDs: a string composed of the *min* and *max* ID delimited by two dots, e.g., '31412..31505' (note: currently the frame ID is not included in the header by the ingest subsystem; it is embedded in the file name, but the WISE system engineer advises against parsing the file name to obtain frame ID on the basis that file naming may vary under certain conditions; once the ingest subsystem is upgraded to include the frame ID in the FITS headers, this keyword will be added).

4.2 Sky-Offset Uncertainty FITS Output

If the "-o2" command-line parameter was specified, a FITS file containing the sky-offset uncertainty will be generated. An example header is given below.

```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                  -32 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                 1016 / length of data axis 1
NAXIS2  =                 1016 / length of data axis 2
COMMENT  Generated by tempcal vsn 1.25 A80715 on 15-07-08 at 11:46:58
COMMENT  This is a sky offset uncertainty image
```

4.3 Mask Updates

If mask inputs and bit numbers are specified, then any of the four conditions diagnosed by tempcal result in updates to these masks. The bit numbers are specified in terms of their decimal equivalents, not bit addresses; in other words, if bit 3 is to be set (where the WISE standard for bit numbering assigns zero to the least-significant bit), then this is specified as the value 8 (2^3). If any given mask bit is desired not to be set despite its condition being diagnosed, a value of zero may be specified for the mask bit value on the command line The conditions diagnosed and command-line specification flags are:

- Transient behavior, "-p"
- Possibly unreliable sky offset, "-s"
- Possibly unreliable sky-offset uncertainty, "-su"
- Probable latent, "-pl" (note: the second character is a lower-case "L", not a numeral "one")

If a sky offset is considered possibly unreliable, its uncertainty always is also. A sky offset may be considered reliable, however, and still have a possibly unreliable uncertainty (e.g., when the outlier-rejected stack has a significant slope with respect to time). These bits are set in every mask for every input science image.

When transient/latent behavior is diagnosed, only the masks corresponding to the frames in which this behavior is believed to exist have the bits set for these conditions, although all frames are tagged as having unreliable sky offset and uncertainty. When latent behavior is identified, the first image is not tagged as either latent or transient, because it is believed to be a measurement of the true source that caused the latent. The exception to this is when the transient run began on the first sample, in which case it cannot be considered the first sample to react to a bright source.

Note that the input masks themselves are updated. Making backup copies of the masks prior to execution is advised.

4.4 Optional Chi-Square FITS Output

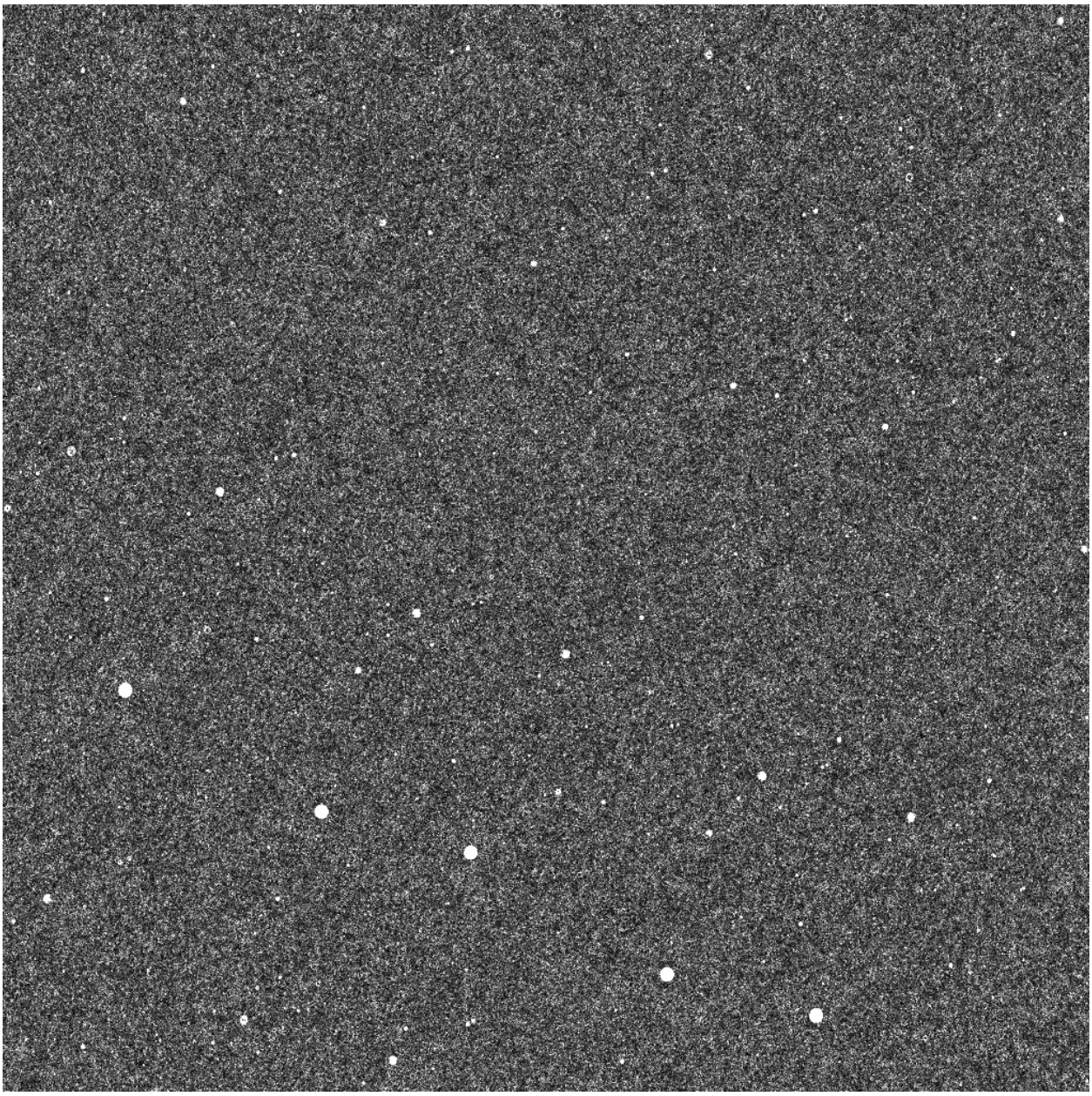
If the "-o3" command-line specification is given, and if sky-offset computation has not been deselected, and if uncertainty images have been given, then the chi-square image is generated. This is the reduced chi-square for every unmasked pixel described in section 3.2.

$$\chi^2_{ij} = \frac{1}{N_{ij}} \sum_{n=1}^{N_{ij}} \frac{\left(p_{ijn} - s_{ij}\right)^2}{\sigma^2_{ijn} - \sigma^2_{s_{ij}}}$$

where the pixel is in the hardware array at column i and row j, p_{ijn} is the pixel's sample at stack location n, and the total number of usable samples in the stack is N_{ij}. The summation shown is

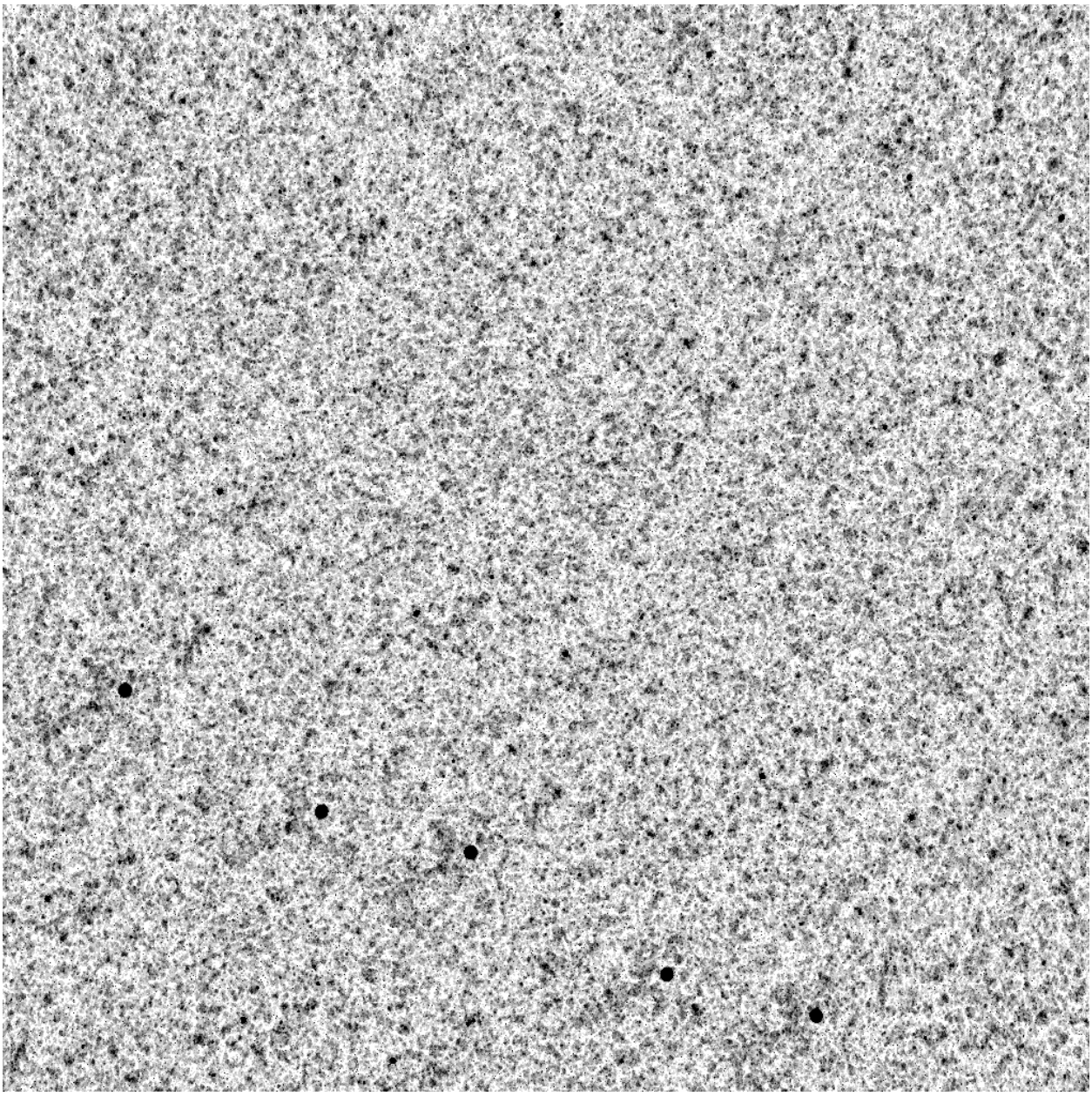
over usable samples only. The pixel's sky offset is s_{ij}, and the denominator in the summation is the uncertainty variance of p_{ijn} with the uncertainty variance of s_{ij} subtracted to compensate for the fact that the differences in the numerator are defined by observed values minus a central value computed from these very observed values.

A sample image from test data is shown below.



4.5 Optional Sample-Size FITS Output

If the "-o4" command-line specification is given, and if sky-offset computation has not been deselected, then the sample-size image is generated. This is the image of the number of pixel values remaining in the stack after outlier rejection for every unmasked pixel as described in section 3.2. A sample image from test data is shown below.



4.6 Optional Time-Slice FITS Output

If the namelist parameters `ITimSlic1` and `ITimSlic2` are specified non-zero, and/or if the namelist parameters `JTimSlic1` and `JTimSlic2` are specified non-zero (see section 2.1.2), then column-range time-slice and/or row-range time-slice images are generated, respectively. "Time-slice" imaging is the generation of FITS files in which consecutive columns correspond to consecutive times in the image stack. Rows in these images may be either hardware array rows or hardware array columns; tempcal will take a specified range of rows or columns (or both, but the results go into separate output images) and generate an image in which the horizontal axis is time and the vertical axis is either row or column. Since rows and columns are handled in analogous fashion, we will describe the column-slice case, which employs `ITimSlic1` and `ITimSlic2`.

If a range of columns is specified, then the image consists of a concatenation of data-cube slices, each slice containing one full row vs. all times, with such slices concatenated for the specified column range, up to 1016 total columns, with additional columns placed into additional output images as needed. This concatenation over column-time slices is set to come as close as possible to a square image without the number of columns exceeding the number of rows. If the number of columns ITimSlic2-ITimSlic1+1 is too large to fit all the slices into one image, then more than one image is generated, with the last column in one image being the first in the next, and the last image containing the same number of slices as the others but ending on ITimSlic2, and therefore possibly having more overlap with the previous image than a single repeated slice.

The images are contained in FITS files named automatically by tempcal as follows: column-slice and row-slice images names begin with ColSlice and RowSlice, respectively; this is followed by an underscore, a four-digit number giving the first hardware column or row in the image, another underscore, and the last hardware column or row in the image. For example, the test data set consisted of 70 time-consecutive science images, so the the time extent is 70 samples, and this is mapped into the horizontal direction. The science images consist of 1016 columns and 1016 rows, so up to 14 slices are concatenated into a single FITS image to arrive at a nearly square image product of 980 columns and 1016 rows. In such an image, the row number is either a hardware row number (for column-range slices) or hardware column number (for row-range slices), and the column number is the time-sample number plus 70 times the number of preceding slices. For example, column 500 corresponds to time sample number 10 in the 8^{th} time slice, i.e., there are seven preceding time slices, so the 10^{th} time sample in the 8^{th} time slice is at image column number $7\times70 + 10 = 500$. In general, with K frames in the stack, time-slice image column I corresponds to time sample N in slice M, where $M = [(I-1)/K]_{truncated} + 1$, and $N = \text{modulo}(I-1,K) + 1$.

For example, a test run was made in which the namelist input was:

```
$tmpcalin
  JTimSlic1 = 501, JTimSlic2 = 600,
$end
```

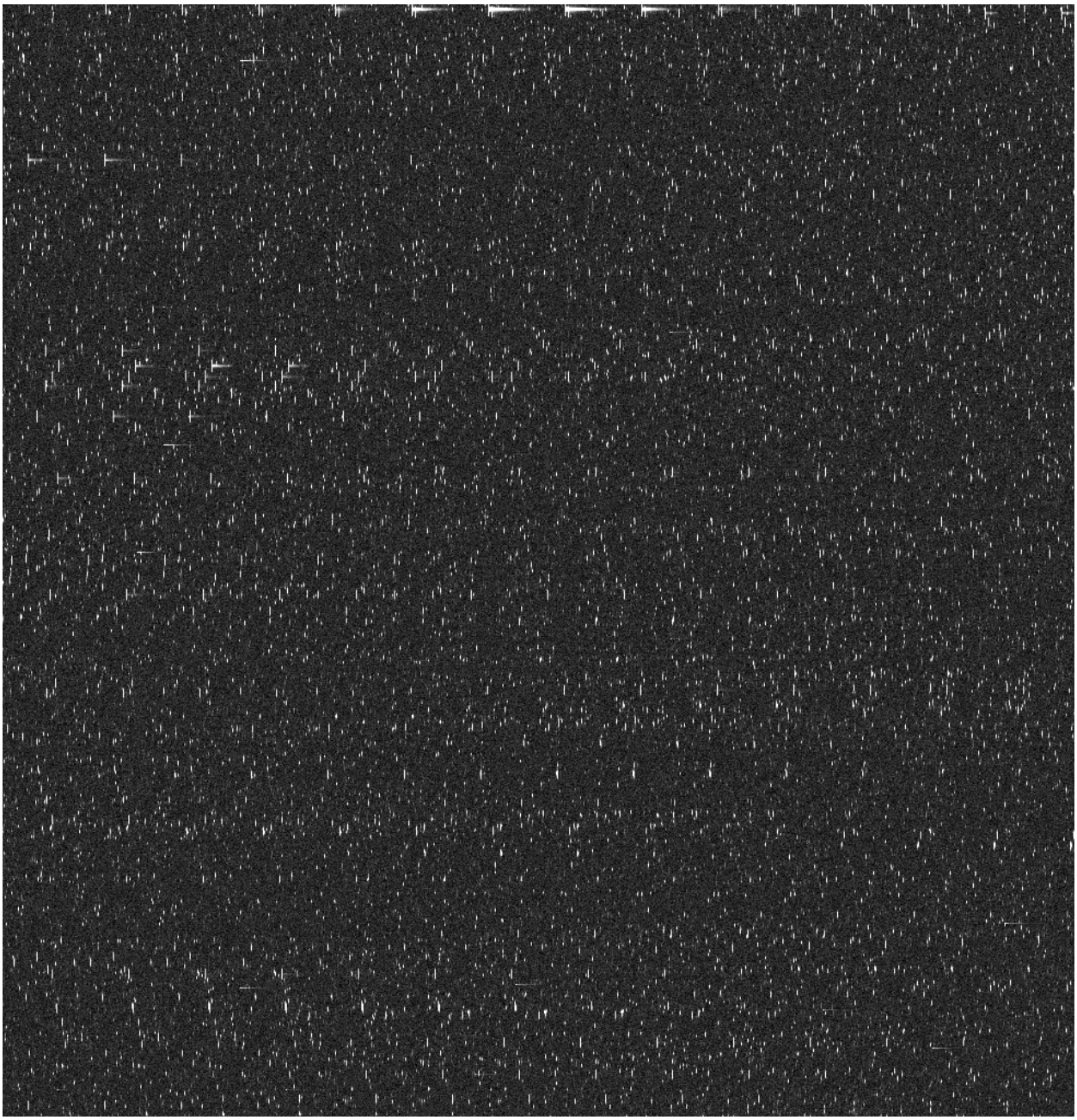
This requests 100 rows in the range 501 to 600. The first time slice contains the 70 time samples and the 1016 hardware columns at hardware row 501, so the leftmost 70 columns and 1016 rows in the time-slice image contain this slice. The next 70 columns and 1016 rows contain the time slice for hardware row 502. This tiling of time slices continues up to the time slice for hardware row 514, which is at the right side of the image. No more slices are concatenated, because this would give the image 1050 columns, more than the number of rows, and this is the arbitrary cutoff for keeping the time-slice images approximately square.

The namelist shown above produced the eight time-slice images shown below:

```
-rw-rw-r--  1 jwf wise 3985920 Jul 25 11:54 RowSlice_0501-0514.fits
-rw-rw-r--  1 jwf wise 3985920 Jul 25 11:54 RowSlice_0514-0527.fits
-rw-rw-r--  1 jwf wise 3985920 Jul 25 11:54 RowSlice_0527-0540.fits
-rw-rw-r--  1 jwf wise 3985920 Jul 25 11:54 RowSlice_0540-0553.fits
-rw-rw-r--  1 jwf wise 3985920 Jul 25 11:54 RowSlice_0553-0566.fits
-rw-rw-r--  1 jwf wise 3985920 Jul 25 11:54 RowSlice_0566-0579.fits
-rw-rw-r--  1 jwf wise 3985920 Jul 25 11:54 RowSlice_0579-0592.fits
-rw-rw-r--  1 jwf wise 3985920 Jul 25 11:54 RowSlice_0587-0600.fits
```

Note that each image contains 14 slices, and the last slice of each image is the first slice in the next. The last image ends with hardware row 600, and since it also contains 14 slices like the others, it begins with hardware row 587, and thus has more than the single-slice overlap with its predecessor. This is done to keep all the time-slice images the same size.

A sample time-slice image is shown below. This is RowSlice_0501_0514.fits. The presence of latent images is immediately obvious, as these show up as streaks in the horizontal (time) direction. The vertical extent of each point source is the image blur in the hardware-column direction. Latent images caused by bright sources generally begin with a significant vertical extent and are followed by a decaying streak. Radiation hits often appear as narrow horizontal streaks. A particularly bright source with a latent tail can be seen near the top of the image; the multiple apparitions spaced 70 columns apart are all the same source seen in adjacent rows. The 70-column width of each time slice can be visually estimated fairly easily because of repetitive patterns like this, even the less striking ones.



4.7 QA Output

If the command-line flag "`-qa`" is used to specify a QA output file name, and if transient processing has been selected, then a table file header is generated containing the information illustrated below with sample values from a test run.

```
\  statistics file generated by tempcal vsn 1.64 AA0212 on 12-02-10 at 13:41:37
\Ntrans     =         8  / Number of transient runs
\Nlat       =         1  / Number of latents tagged
\MinPersist =        15  / Minimum run length to diagnose transient
\Qmax       =     0.050  / Maximum binomial Q probability for latent
\MedTrans   =         9  / Median transient run length
\MedDrops   =         4  / Median number of pixel drops
\MedFdrop   =     0.500  / Median fraction of drops/run
\MedTransT  =         9  / MedTrans for Transient-only
\MedDropsT  =         4  / MedDrops for Transient-only
\MedFdropT  =     0.500  / MedFdrop for Transient-only
\MedTransL  =         9  / MedTrans for Latent-tagged
\MedDropsL  =         6  / MedDrops for Latent-tagged
\MedFdropL  =     0.750  / MedFdrop for Latent-tagged
```

The parameters are intended to be self-explanatory. This test case illustrates some of the numerical results than can appear confusing at first sight. `MinPersist` is 15 samples, yet the median transient run lengths are 9, which is less than 15. This is because most of the 8 transient runs began on the first sample; runs that begin on the first sample or continue through the last sample are required only to be at least `(MinPersist+1)/2` samples long. Only one run was also diagnosed as a latent (`Nlat = 1`), so the "median" values are the only values that occurred. The length was 9 samples, in which the number of drops was 6; two things appear to be incorrect: (a.) the drop fraction, `MedFdropL`, is shown as 0.750, which is not equal to `MedDropsL/MedTransL` = 6/9; (b.) for the `Qmax` shown, `Mmin`(9) = 7, but only 6 drops were observed, so this seems not to qualify as a latent. Both of these anomalies are caused by the fact that the run began on the first sample, as detailed debug output shows (see below). The run was 9 samples long, hence there were 8 first differences, of which 6 were drops; `Mmin`(8) = 6, so this qualifies as a latent, and 6/8 = 0.750, so `MedFdropL` is correct. But since the run began on the first sample, that sample is tagged in the mask and included in the run length, hence 9 is reported as the length. If an identical run had started on the second or later sample, `MedTransL` would have had the value 8.

The original detection of a "bad" run involves K samples running from N1 to N2 in the stack, K ≥ `MinPersist` (or `(MinPersist+1)/2` if the run is at the beginning or end of the stack). The K samples permit only K-1 differences, so the number of drops M can never be equal to K under any circumstances. M is always tested against `Mmin`(K-1); a latent is diagnosed if and only if M ≥ Mmin(K-1). If the run is *not* diagnosed as a latent, then the run length is N = K = N2-N1+1. If the run *is* diagnosed as a latent, then the first sample is no longer considered part of the run, *provided* that it can be established that the first sample really was the first violation of the outlier test, i.e., provided there was a sample *preceding* it that was *not* an

outlier. This cannot be established if the first sample in the run is the first sample in the stack. So when a latent *is* diagnosed, we always have M ≥ `Mmin`(K-1), but the run length N is set to K-1 if N1 > 1; this happens because if N1 > 1, then N1 is incremented, which decrements the values of N2-N1+1 = N, i.e., we label the run as starting one sample *after* the first outlier. When N1 = 1, the run is considered to be measured *from* the first sample and *including* that sample, so N1 is not incremented, and the value of N = N2-N1+1 remains equal to K. Therefore the drop fraction is M/(K-1) = M/N if N1 > 1, otherwise it is M/(K-1) ≠ M/N, since N = K in this case.

Also note that if a frame's pixel is skipped in PixStak due to masking or any other reason, the next usable pixel counts as the next chronological pixel for transient/latent purposes.

4.8 Debug Output

If the command-line flag "`-d`" is used to invoke debug output, and if transient processing has been selected, then two text files are generated, the TimeOrderedFrmList.txt file and the Badstats.tbl file, and three FITS files are generated. BadStart.fits, BadEnd.fits, and nDrop.fits. The three FITS files all have the dimensions of the WISE images being processed, and all have BITPIX = 16; they contain information concerning the single longest transient run (if any) for each pixel: the first sample number, the last sample number, and the number of drops in the run, respectively. Values of zero indicate that no transient run occurred.

The TimeOrderedFrmList.txt file contains a list of the input intensity images in time order, one per line, with an ordinal counter at the beginning of the line. This may be useful when knowledge of the chronological order is needed, since the program does not require the input lists to be in any particular order, although the list of mask files and/or uncertainty images must be in the *same* order as the list of intensity images. A sample from a test run is shown below. This test had 50 intensity images, of which the first and last five are shown.

```
    1 /wise-ops/01/wise/fmasci/NedSim_May09/tempcal/testdata1/00478a222-w3-int-1b.fits
    2 /wise-ops/01/wise/fmasci/NedSim_May09/tempcal/testdata1/00478a223-w3-int-1b.fits
    3 /wise-ops/01/wise/fmasci/NedSim_May09/tempcal/testdata1/00478a224-w3-int-1b.fits
    4 /wise-ops/01/wise/fmasci/NedSim_May09/tempcal/testdata1/00478a225-w3-int-1b.fits
    5 /wise-ops/01/wise/fmasci/NedSim_May09/tempcal/testdata1/00478a226-w3-int-1b.fits
      . . . . . .
   46 /wise-ops/01/wise/fmasci/NedSim_May09/tempcal/testdata1/00478a271-w3-int-1b.fits
   47 /wise-ops/01/wise/fmasci/NedSim_May09/tempcal/testdata1/00478a272-w3-int-1b.fits
   48 /wise-ops/01/wise/fmasci/NedSim_May09/tempcal/testdata1/00478a273-w3-int-1b.fits
   49 /wise-ops/01/wise/fmasci/NedSim_May09/tempcal/testdata1/00478a274-w3-int-1b.fits
   50 /wise-ops/01/wise/fmasci/NedSim_May09/tempcal/testdata1/00478a276-w3-int-1b.fits
```

The Badstats.tbl file contains information on the longest transient run for each pixel that had at least one such run. The header contains definitions of the column parameters. An example is shown below from the same run as that used as an example in section 4.7 above; the more detailed information here supplies the explanation for the apparent anomalies discussed there.

```
\  statistics file generated by tempcal vsn 1.64 AA0212 on 12-02-10 at 11:28:09
\Ntrans     =        8  / Number of transient runs
\Nframes    =       50  / Number of frames
\ COLUMN HEADER DESCRIPTIONS:
\ I         = Pixel column number
\ J         = Pixel row number
\ N1        = Run-begin stack sample number
\ N2        = Run-end stack sample number
\ N         = Run length
\ M         = Number of pixel-to-pixel drops in value
\ Mmin      = Minimum M to tag latent, f(N)
\ fDrop     = Ratio of pixel drops to run length, M/N
\ nR        = Number of separate runs in stack
\
|  I |  J | N1| N2| N | M |Mmin|fDrop|nR|
| int| int|int|int|int|int| int| real| i|
    3  763   1  22  22  13   14 0.619  1
    8  764   1   9   9   4    6 0.500  1
    2  770  33  49  17   9   12 0.529  1
   12  775  43  50   8   4    6 0.500  1
    3  777   9  32  24   9   16 0.375  1
   12  948   1   8   8   4    6 0.571  2
  925  948   1   9   9   3    6 0.375  1
   12  949   1   9   9   3    6 0.375  1
```

Both files show that 8 transient runs occurred. All runs with length `N` less than `MinPersist` (15) either began on sample 1 (`N1` = 1) or ended on the last sample in the stack (`N2` = 50). The last transient run was tagged as a latent; this is the only one with `M` = `Mmin` (6). Although the header says that `fDrop` is `M/N`, this pixel is an exception because the run began on the first sample, which is therefore considered part of the run of 9 samples, but with only 8 first differences to test, `fDrop` is really 6/8, as explained in section 4.7.

5 Testing and Parameter Tuning

5.1 Tempcal Testing

The tempcal module has been unit-tested with simulated data hand-edited to force the various processing paths to be traversed. It has also been tested with simulated FITS data generated by N. Wright for a mid-ecliptic-latitude region around zero degrees longitude and 30 degrees latitude.

5.2 Tempcal Parameter Tuning

The tempcal input parameters which control robust estimation of the sky offset and transient pixel behavior will require tuning prior to their use in routine pipeline processing. The chi-square image should prove useful in this activity, since effective outlier rejection should leave a set of pixel samples which is well behaved with respect to prior uncertainty and local background noise. The two most important parameters are (see section 2.1.2) `ThrshHi` and `ThrshLo`. Also

of special interest are `MinPersist`, `MinPix`, and `ChiSqMax`. Finally, the optimal number of consecutive frames has to be determined.

6 Example Command Lines

The following example reads in a list of image FITS file names (`-f1`), corresponding masks (`-f2`), and uncertainty images (`-f3`). The output sky-offset image (`-o1`) will be named skyoff.fits, the output sky-offset uncertainty image (`-o2`) will be named sig_skyoff.fits, a reduced chi-square image (`-o3`) to be named chsq.fits is requested, and a sample-size image (`-o4`) named nused.fits is also requested. The minimum number of consecutive outlier samples to define transient behavior (`-pn`) is set to 20. Samples with input mask bits 1 and 2 set will not be processed (`-m`) since the value 6 was specified, and $6 = 2^2 + 2^1$ (see section 4.3). Samples diagnosed as transient will have bit 21 set in all corresponding masks (`-p`), since 2097152 (2^{21}) was specified. Similarly, mask bits 23, 28, and 25 will be set for unreliable sky offsets (`-s`), unreliable sky-offset uncertainties (`-su`), and probable latents (`-pl`), respectively, since values of 8388608 (2^{23}), 268435456 (2^{28}), and 33554432 (2^{25}), respectively, were specified.

```
% tempcal -f1 intlist -o1 skyoff -o2 sig_skyoff -o3 chsq -o4 nused   \
          -f3 unclist -f2 masklist -pn 20 -p 2097152 -m 6 -s 8388608 \
          -su 268435456 -pl 33554432
```