Sky-Offset / Dynamic 'Bad' pixel flagging Module

F. Masci, version 1.0 (3/19/2008)

1. Background

The functions of this module shall be twofold:

i. Compute a 'sky-offset' image from a median (or trimmed average) of *N* consecutive frames along a scan, where the pixel values therein are distributed about a zero median (or zero-mean).
ii. Identify and flag *persistent* new "bad" pixels from the stack of *N* images. This is the essence of dynamic pixel masking.

Both these tasks will attempt to capture instrumental transients in the WISE detectors, and will be corrected (or propagated along) in the instrumental calibration pipeline. We discuss each in turn below. A suggested name for this module is *tempcal,* for "temporary calibration". The developer can choose a new name if this doesn't appeal. The author also thought of *transientcal*, but this sounds too long winded.

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 more-or-less static in the long-term. If the short-term systematic variations are not removed, they will persist as residuals and impact photometric accuracy (as shown on 2MASS). These can be corrected by computing a zero-mean (or zero-median) 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 only create the sky-image calibration product, not apply it.

We estimate that at least 50 - 100 frames will be needed to reliably filter out sources in any band. 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 constant over this window span. A schematic of the concept is shown in Figure 1.

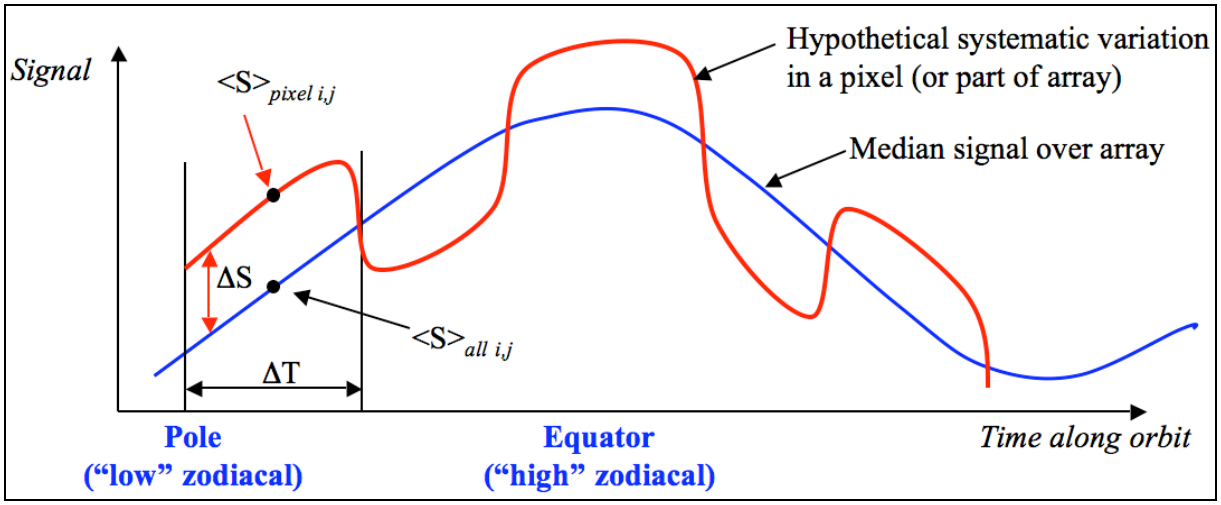

Figure 1: Sky-offset schematic

The labels in Figure 1 are defined as follows:

ΔT = timescale of possible systematic variation

$\langle \mathrm{S} \rangle_{pixel\ i,j}$ = median or mean signal in single pixel i, j over stack of N frames in ΔT

$\langle \mathrm{S} \rangle_{all\ i,j}$ = median or mean signal over all pixels and N frames in ΔT

Sky - offset correction : $\Delta S_{i,j,\Delta T} = \langle \mathrm{S} \rangle_{pixel\ i,j} - \langle \mathrm{S} \rangle_{all\ i,j}$

- Number of frames must satisfy : $N_{\min} \le N \le N_{\Delta \mathrm{T}}$, where $N_{\min}$ is minimum needed to filter out stars

- If $N_{\Delta \mathrm{T}} < N_{\min}$ (fast instrumental variations), then method can't be used

- Plan is to determine optimal frame windows $N_{\min}$ and $N_{\Delta \mathrm{T}}$ in IOC

Note: the angled brackets can represent either a *trimmed mean*, *median* or whatever estimator is used to optimally combine the pixels in the *N*-frame stack. See section 3 for more details.

The rationale behind including dynamic pixel masking in this module is that it's 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 δt.

Bad pixels (where the criteria for 'bad' are defined below) occur at the same location in pixel space, whereas astronomical sources don't. 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 δt 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. More details are given below.

A minor detail (which should not concern the developer here) is that not all frames in the mission will have an associated sky-offset calibration and dynamic bad-pixel flagging performed. This is because there may be contiguous chunks of frames that are unusable for this purpose (or for anything else for that matter), e.g., anneals and recovery thereafter, South Atlantic Anomaly passes, and other predicted events. This also includes unforeseen events and transients picked up by the *ingest*-QA subsystem, e.g., ultra-bright sources and persistence there from.

2. Command-line Synopsis and I/O

Here's a baseline (suggested) synopsis – basically the on-screen tutorial if the module is executed with no command-line arguments. Some variables are further discussed in section 3. The developer is free to reword, reorder any of the below. I'm sure there'll be more as development proceeds.

```
Program tempcal vsn 1.0

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] detection;
                             Default = 5)

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

-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)

-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)

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

-o1 <out_skyoff_img>        (Required; output sky-offset image FITS filename)

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

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

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

3. Suggested Algorithm and Details

Below is an outline of the main processing steps. I encourage the developer to take initiative on making the overall process as efficient as possible (goes without saying, right?).

3.1 Sky-offset calibration product

- Input frames will have already been pre-calibrated, e.g., dark subtracted, linearized and flat-fielded. The units of the pixel values in each are in native WISE "*scaled*-slope" units with *no offset*. More specifically, they will be in units of *scaled* DN/SUR, where DN = Data Number and SUR = Sample Up the Ramp.

- Read/store all frames and accompanying (optional) masks, uncertainty frames in memory. 150 frames with accompanying masks and uncertainty images (the maximum expected at any one time - i.e., 450 in total) won't take up more than 2 GB of memory.

- Flag or tag prior-known bad pixels according to mask bits specified by the <–m> input option. These will be omitted from processing: both when creating the sky-offset image, and when detecting new bad pixels for dynamic masking (see below).

- For a given pixel location, take all the values from the stack of frames and combine them in some *optimal* manner [e.g., to compute $<S>_{pixel\ ij}$ in Figure 1]. I suggest using a robust *non-parametric* method with optional trimming for outliers. An example is to first use the Median Absolute Deviation (MAD) from the sample median on the lower-tail values (<median) to get a robust measure of *sigma*, then using this with a given threshold to detect outliers at both ends of the distribution, then re-computing the median with the outliers thrown out. Another method involves estimating the mode and using this as the optimum measure of location. Mode estimation however is complicated! Yet another method is that used on 2MASS – the general Recursive Median-Distance Rejection (RMDR) method. Unless the developer commits to one method right off the bat and is happy with it, these methods need to be experimented with. The best estimator is that which yields the smallest variance in the least-squares sense. A Monte Carlo simulation may be needed to test the different possibilities.

- Using the same method, compute the same value for all pixels *i, j* in *all* the input frames. This is represented by $<S>_{all\ ij}$ in Figure 1. Subtract this value from the individual $<S>_{pixel\ ij}$. This will make all the values zero-mean ("mean" here is used loosely and represents the real estimator used).

- The main output (<-o1> command-line parameter) is a FITS image of the $\Delta S_{ij} = <S>_{pixel\ ij} - <S>_{all\ ij}$ values, i.e., 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  =                <Num> / length of data axis 1
NAXIS2  =                <Num> / length of data axis 2
BAND    =                <Num> / WISE band number (1, 2, 3 or 4)
NUMINP  =                <Num> / Number of input frames used
UTCSBGN =                <Num> / Earliest UTCS in frame stack [sec]
UTCSEND =                <Num> / Latest UTCS in frame stack [sec]
FRMIDSEQ=         <'Num..Num'> / Range of frameIDs used
BUNIT   =      'scaled DN/SUR' / Units of image data
COMMENT WISE sky-offset calibration product, created <YYYY-MM-DD>
COMMENT generated by tempcal, v.1.0 on <YYYY-MM-DD> at <HH:MM:SS>
```

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., UTCS_OBS]
  - 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'. The frame IDs will be present in the input frame headers [keyword unknown at time of writing].
  - The BUNIT keyword specifies the data units and will always have the string value as shown.

- Whatever method is used to optimally combine the pixel values (in a stack) into a single value, an uncertainty must also be computed. Note that estimators based on the median and associated measures of spread (the MAD) tend to yield noisier estimates relative to those optimized for a prior-assumed underlying distribution (e.g., for a normal distribution and many others, the mean is the minimum variance estimator). Also, beware that trimming (both symmetric and asymmetric versions) will also affect the uncertainty in your estimator (whether it's the mean or median). For example, a trimmed mean will not have a normal distribution even if the data were drawn from a normal population. In the end, I recommend using a robust non-parametric estimator (e.g., a median) when the sample is of reasonable size (like here). The slight increase in variance is a small price to pay, since a missed outlier (or improperly tuned trimming threshold) can wreck greater havoc on the *mean*.

- One-sigma uncertainties corresponding to the ΔS_{ij} should be stored in a separate FITS file (<-o2> command-line parameter). For these, the header can be the same as above, except that the relevant COMMENT field will need to indicate that this is a "sky-offset *uncertainty* calibration product".

- If the sky-offset was unreliable or could not be computed for any of the pixel stacks, then a bit should be set for the corresponding pixel in *all* of the input masks. This bit is specified by the <-s> command-line parameter (in decimal format). At the time of writing, the value 2^{23} (= 8388608) has been reserved for flagging pixels where the sky-offset was unreliable, or could not be computed. In production, the sky-offset will *not* be applied to such pixels.

- Note: the above formalism does not make use of prior uncertainties (propagated upstream from an error model), even though they were included in the above synopsis (<-f3> command-line parameter). If the developer thinks the overall estimation process will benefit from them (e.g., χ^2 sanity checks), then they will certainly be available for use.

3.2 Dynamic bad-pixel detection/flagging

- Dynamic pixel masking should only be performed if a list of frame processing masks were supplied on input (<-f2> command-line parameter).

- Before proceeding to identify bad pixels, it is recommended that the input frames (and accompanying mask images) be time ordered using the time-tag keyword in the headers [e.g., UTCS_OBS]. The reason for this is that a pixel may be in a 'bad state' for only part (δt) of the time interval ΔT used for the sky-offset calibration. Having a time-ordered sequence will make detection of a bad-pixel easier, e.g., by tracking its repeated occurrences in consecutive frames, rather than at random locations as represented by the unknown frame order in the input list.

- The definition of "bad" pixel here is constrained by the detection method. To keep it simple, a bad pixel shall be one whose signal behaves differently from its neighbors in the same frame, and this behavior is more-or-less consistent for some duration $\delta t \leq \Delta T$ in the frame stack. For example, the method will mainly be sensitive at detecting pixels that become highly responsive (hot), or lowly responsive (almost dead), not necessarily pixels with abnormally fluctuating values (i.e., noisy). High/low responsive pixels are defined relative to neighbors in the same frame, assuming that the majority are in their 'normal' operating state. Note that this method also has the potential to flag strong persisting latents, but these will be handled more robustly elsewhere.

- The specific steps for dynamic pixel flagging are as follows:

i. Time order the input frames and masks (see second bullet point in this section).
ii. Take the first frame in the time-ordered list and histogram the pixel values.
iii. Isolate the outliers from this histogram, e.g., using the same *robust* sigma-estimation method as used for the sky-offsets above, or some variant thereof. The lower/upper threshold command-line input parameters: <-lt> and <-ut> may be used for this purpose.
iv. Omit prior-known (static) bad pixels from this outlier list using the input masks and the bit-masking template: command-line inputs <-f2> and <-m>.
v. Repeat steps ii, iii and iv for all frames in the input list where the outliers from iii (the candidate 'bad' pixels) are stored in memory for each frame.
vi. Cycle through each outlier in each frame-outlier list and search for repeating offenders in subsequent outlier lists. If a particular outlier persists for N_b consecutive frames, where N_b is given by the input parameter <-pn>, set a bit for the respective pixel in the corresponding input masks. This bit is specified by the <-p> command-line parameter (in decimal format). At the time of writing, the value 2^{21} (= 2097152) has been reserved for the dynamic pixel flagging.