HAWC+ Data Reduction Pipeline Users Manual

SOF-US-HBK-OP10-2008

Date: March 7, 2017
Revision: -

AFRC
Armstrong Flight Research Center
Edwards, CA 93523

ARC
Ames Research Center
Moffett Field, CA 94035

German Space Agency, DLR
Deutsches Zentrum für Luft und Raumfahrt

HAWC+ Data Reduction Pipeline Users Manual

SOF-US-HBK-OP10-2008

AUTHOR:

Melanie Clarke, USRA, SOFIA DPS Development Lead	Date

APPROVAL:

William Vacca, USRA, SOFIA Associate Director for Science Data Systems	Date
Roberta Leftwich-Vann, USRA SOFIA Program Manager	Date

REVISION HISTORY

REV	DATE	DESCRIPTION
-	03/07/17	Initial Release

HAWC+ DRP User's Manual
SOF-US-HBK-OP10-2008

M. Berthoud, A. Kovács, F. Santos, G. Novak, M. Clarke

February 23, 2017

Contents

1 Introduction

The SI Pipeline Users Manual (OP10) is intended for use by both SOFIA Science Center staff during routine data processing and analysis, and also as a reference for General Investigators (GIs) and archive users to understand how the data in which they are interested was processed. This manual is intended to provide all the needed information to execute the SI Level 2 and 3 Pipeline, and assess the data quality of the resulting products. It will also provide a description of the algorithms used by the pipeline and both the final and intermediate data products.

A description of the current pipeline capabilities, testing results, known issues, and installation procedures are documented in the SI Pipeline Software Version Description Document (SVDD, SW06, DOCREF). The overall Verification and Validation (V&V) approach can be found in the Data Processing System V&V Plan (SV01-2232). Both documents can be obtained from the SOFIA document library in Windchill at location: / Software Management Development or Verification / Pipelines (DPS).

This manual applies to HAWC DRP version 1.0.1.

2 SI Observing Modes Supported

2.1 HAWC+ Instrument Information

HAWC+ is the upgraded and redesigned incarnation of the High-Resolution Airborne Wide-band Camera instrument (HAWC), built for SOFIA. Since the original design never collected data for SOFIA, the instrument may be alternately referred to as HAWC or HAWC+. HAWC+ is designed for far-infrared imaging observations in either total intensity (imaging) or polarimetry mode.

HAWC currently consists of dual TES BUG Detector arrays in a 64x40 rectangular format. A six-position filter wheel is populated with five broadband filters ranging from 40 to 250 μm and a dedicated position for diagnostics. Another wheel holds pupil masks and rotating half-wave plates (HWPs) for polarization observations. A polarizing beam splitter directs the two orthogonal linear polarizations to the two detectors (the reflected (R) array and the transmitted (T) array). Each array was designed to have two 32x40 subarrays, for four total detectors (R0, R1, T0, and T1), but T1 is not currently available for HAWC. Since polarimetry requires paired R and T pixels, it is currently only available for the R0 and T0 arrays. Total intensity observations may use the full set of 3 subarrays.

2.2 HAWC+ Observing Modes

The HAWC instrument has two instrument configurations, for imaging and polarization observations. In both types of observations, removing background flux due to the telescope and sky is a challenge that requires one of several observational strategies. The HAWC instrument may use the secondary mirror to chop rapidly between two positions (source and sky), may use discrete telescope motions to nod between different sky positions, or may use slow continuous scans of the telescope across the desired field. In chopping and nodding strategies, sky positions are subtracted from source positions to remove background levels. In scanning strategies, the continuous stream of data is used to solve for the underlying source and background structure.

The instrument has three standard observing modes for imaging: the Chop-Nod instrument mode combines traditional chopping with nodding, the Chop-Scan mode combines traditional chopping with slow scanning of the SOFIA telescope, and the Scan mode uses slow telescope scans without chopping. The Scan mode is the most commonly used for science observations. The Nod-Pol observing mode is used for all polarization observations. This mode includes chopping and nodding cycles in multiple HWP positions.

All modes that include chopping or nodding may be chopped and nodded on-chip or off-chip. Currently, only two-point chop patterns with matching nod amplitudes (nod-match-chop) are used in either Chop-Nod or Nod-Pol observations, and nodding is performed in an A-B-B-A pattern only. All HAWC modes can optionally have a small dither pattern or a larger mapping pattern, to cover regions of the sky larger than HAWC's fields of view. Scanning patterns may be either box rasters or Lissajous patterns.

3 Algorithm Description

The data reduction pipeline for HAWC has two main branches of development: the HAWC DRP provides the Chop-Nod and Nod-Pol reduction algorithms, as well as the calling structure for all steps. Scan mode reduction algorithms are provided by a standalone package called CRUSH that may be called from the DRP.

3.1 Chop-Nod and Nod-Pol Reduction Algorithms

The following sections describe the major algorithms used to reduce Chop-Nod and Nod-Pol observations. In nearly every case, Chop-Nod (total intensity) reductions use the same methods as Nod-Pol observations, but either apply the algorithm to the data from the single HWP angle available, or else, if the step is specifically for polarimetry, have no effect when

called on total intensity data. Since nearly all total intensity HAWC observations are taken with scanning mode, the following sections will focus primarily on Nod-Pol data.

See the figures below for flow charts that illustrate the data reduction process for Nod-Pol data (Figures 1 and 2) and Chop-Nod data (Figures 3 and 4).

3.1.1 Prepare

The first step in the pipeline is to prepare the raw data for processing, by rearranging and regularizing the raw input data tables, and performing some initial calculations required by subsequent steps.

The raw (Level 0) HAWC files contain all information in FITS binary table extensions located in two Header Data Unit (HDU) extensions. The raw file contains:

- Primary HDU: Contains the necessary FITS keywords in the header but no data. It contains all required keywords for SOFIA, plus all keywords required to reduce or characterize the various observing modes. Extra keywords (either from the SOFIA keyword dictionary or otherwise) have been added for human parsing.
- CONFIGURATION HDU (EXTNAME = CONFIGURATION): HDU containing MCE (detector electronics) configuration data. This HDU is omitted for products after Level 1, so it is stored only in the raw and demodulated files. Nominally, it is the first HDU but users should use EXTNAME to identify the correct HDUs. Note, the "HIERARCH" keyword option and long strings are used in this HDU. All keyword names are prefaced with "MCEn" where n=0,1,2,3. Only the header is used in this HDU.
- TIMESTREAM Data HDU (EXTNAME = TIMESTREAM): Contains a binary table with data from all detectors, with one row for each time sample. The raw detector data is stored in the column "SQ1Feedback", in FITS (data-store) indices, i.e. 41 rows and 128 columns. Columns 0-31 are for subarray R0, 32-63 for R1, 64-95 for T0 and 96-127 for T1). Additional columns contain other important data and metadata, including time stamps, instrument encoder readings, chopper signals, and astrometry data.

In order to begin processing the data, the pipeline first splits these input TIMESTREAM data arrays into separate R and T tables. It will also compute nod and chop offset values from telescope data, and may also delete, rename, or replace some input columns in order to format them as expected by later algorithms. The output data from this step has the same HDU structure as the input data, but the detector data is now stored in the "R Array" and "T Array" fields, which have 41 rows and 64 columns each.

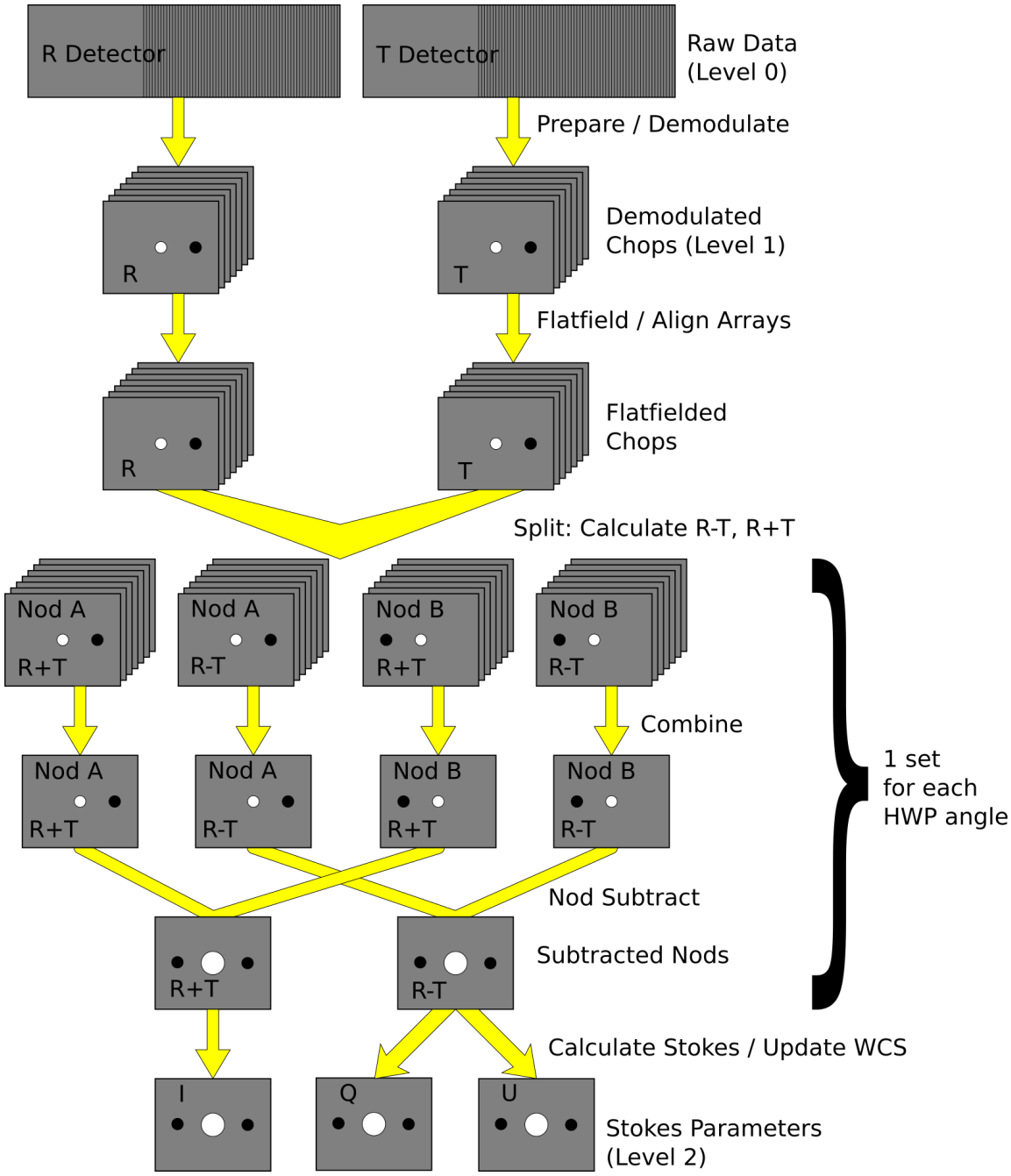

Figure 1: Nod-Pol data reduction flowchart, up through Stokes parameter calculation for a single input file

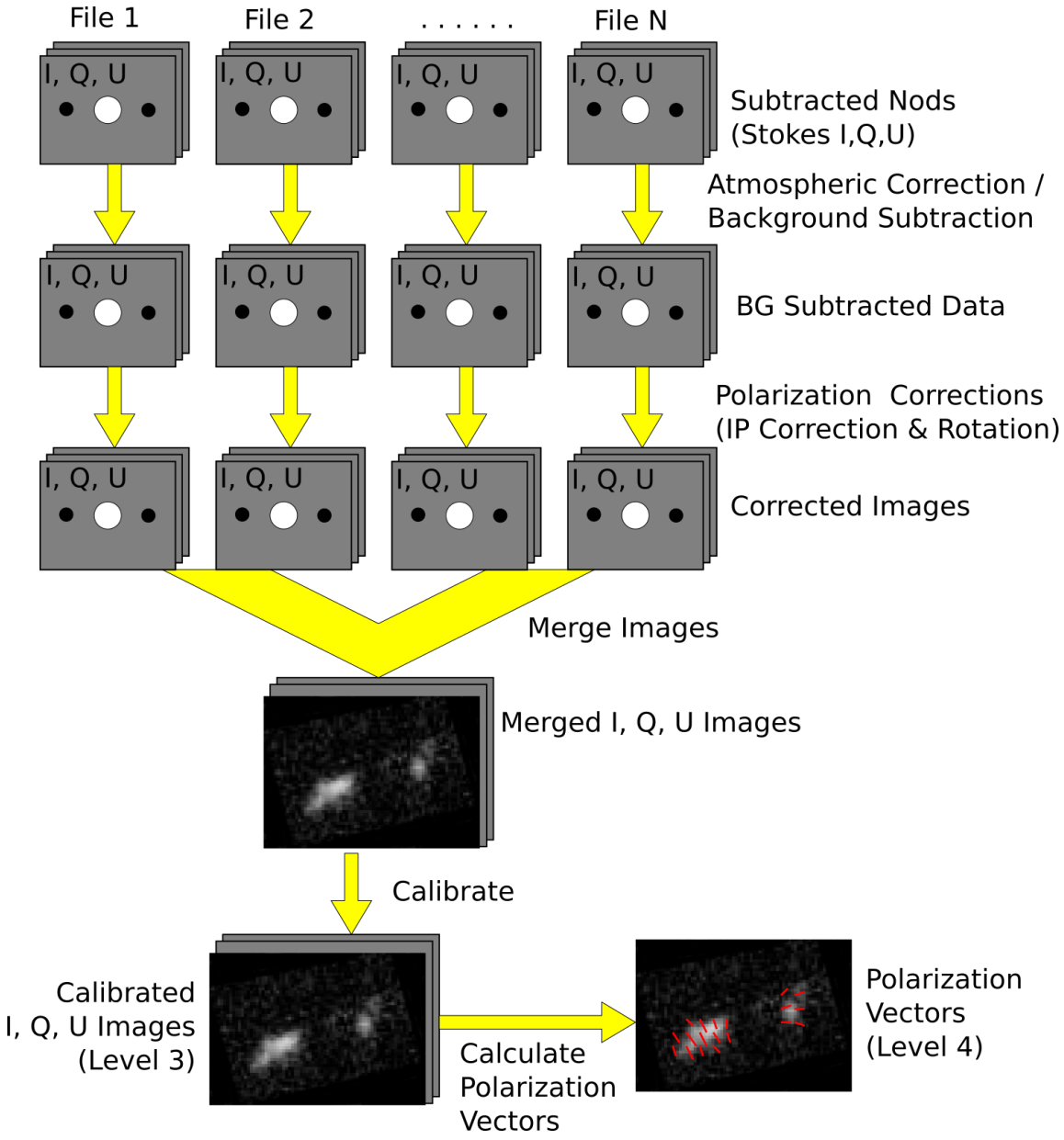

Figure 2: Nod-Pol data reduction flowchart, picking up from Stokes parameter calculation, through combining multiple input files

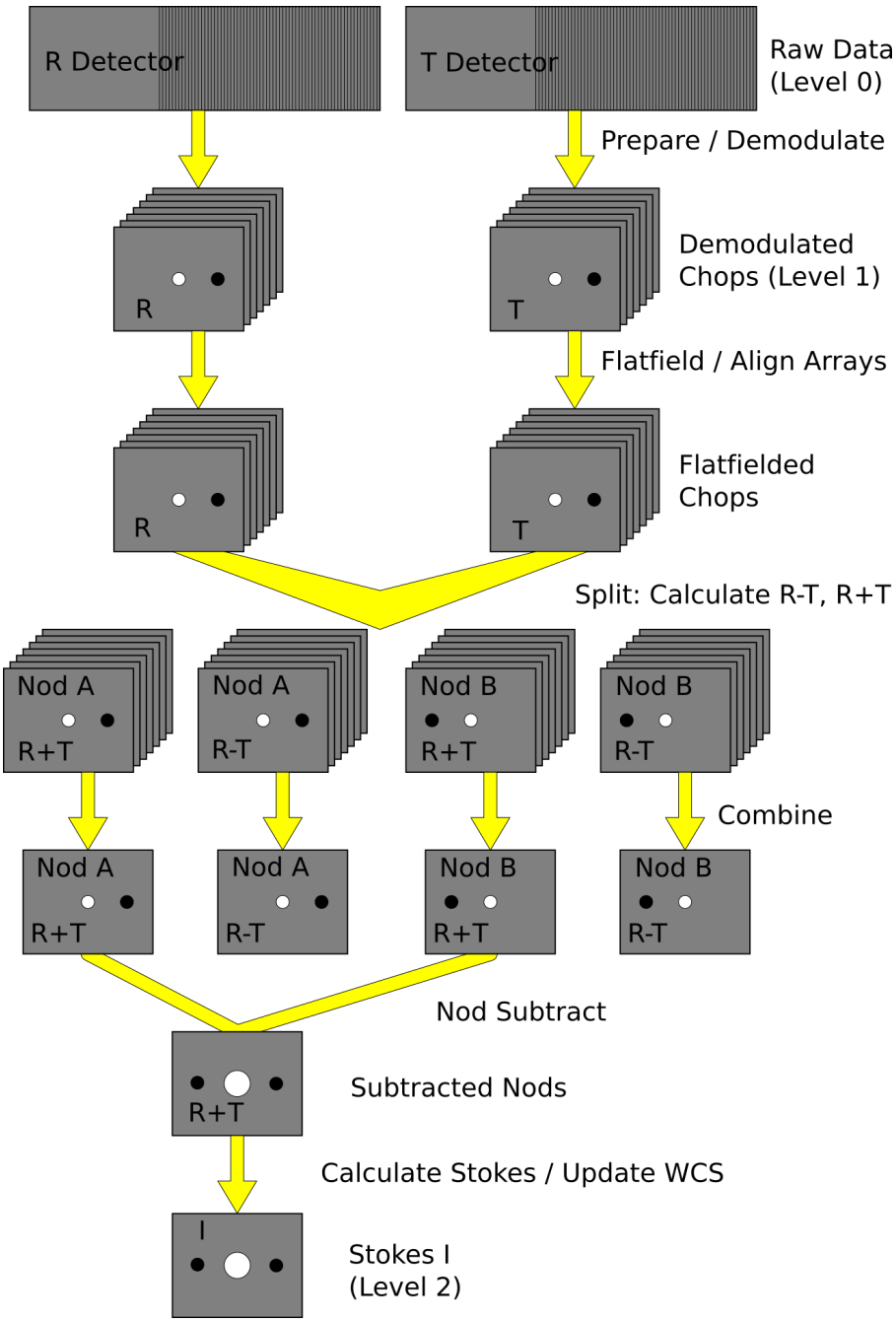

Figure 3: Chop-Nod data reduction flowchart, up through Stokes parameter calculation for a single input file

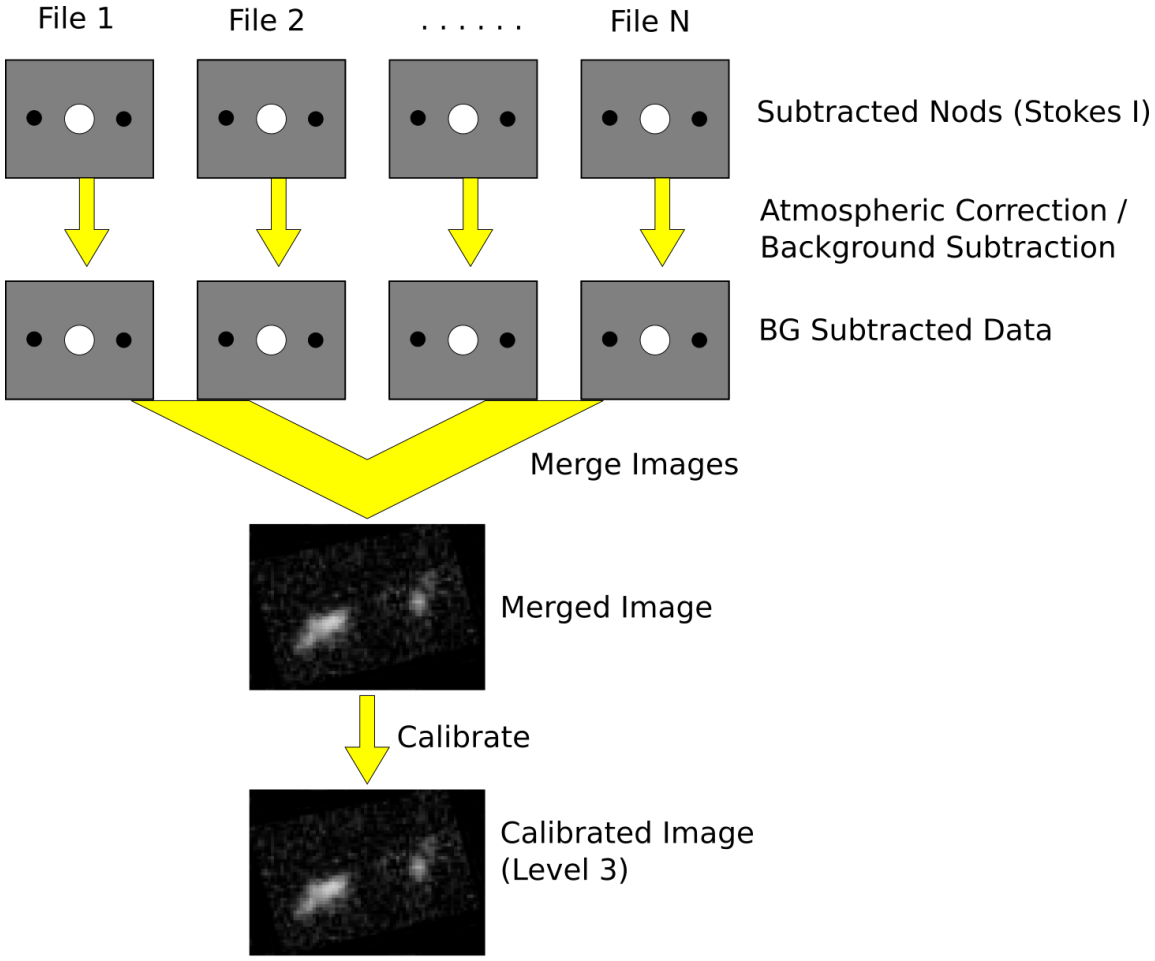

Figure 4: Chop-Nod data reduction flowchart, picking up from Stokes parameter calculation, through combining multiple input files

3.1.2 Demodulate

For both Chop-Nod and Nod-Pol instrument modes, data is taken in a two-point chop cycle. In order to combine the data from the high and low chop positions, the pipeline demodulates the raw time stream with either a square or sine wave-form. Throughout this step, data for each of the R and T arrays are handled separately.

During demodulation, a number of filtering steps are performed to identify good data. By default, the raw data is first filtered with a box high-pass filter with a time constant of one over the chop frequency. Then, any data taken during telescope movement (line-of-sight rewinds, for example, or tracking errors) is flagged for removal. In square wave demodulation, samples are then tagged as being in the high-chop state, low-chop state, or in between (not used). For each complete chop cycle within a single nod position at a single HWP angle, the pipeline computes the average of the signal in the high-chop state and subtracts it from the average of the signal in the low-chop state. Incomplete chop cycles at the end of a nod or HWP position are discarded. The sine-wave demodulation proceeds similarly, except that the data are weighted by a sine wave instead of being considered either purely high or purely low state.

During demodulation, the data is also corrected for the phase delay in the readout of each pixel, relative to the chopper signal. For square wave demodulation, the phase delay time is multiplied by the sample frequency to calculate the delay in data samples for each individual pixel. The data is then shifted by that many samples before demodulating. For sine wave demodulation, the phase delay time is multiplied with 2π times the chop frequency to get the phase shift of the demodulating wave-form in radians.

The result of the demodulation process is a chop-subtracted, time-averaged value for each nod position, HWP angle, and array. The output is stored in a new FITS table, in the extension called DEMODULATED DATA, which replaces the TIMESTREAM data extension. The CONFIGURATION extension is left unmodified.

3.1.3 Flat Correct

After demodulation, the pipeline corrects the data for pixel-to-pixel gain variations by applying a flat field correction. Flat files for each filter band are provided to the pipeline by the instrument team. They contain normalized gains for the R and T array, so that they are corrected to the same level. The flat file also contains a bad pixel mask, with zero values indicating good pixels and any other value indicating a bad pixel. Pixels marked as bad are set to NaN in the gain data. To apply the gain correction and mark bad pixels, the pipeline multiplies the R and T array data by the appropriate flat data. Since the T1 subarray is not available, all pixels in the right half of the T array are marked bad at this stage.

The output from this step contains FITS images that are propagated through the rest of the pipeline steps. The R array data is stored as an image in the primary HDU; the T array data, R bad pixel mask, and T bad pixel mask are stored as images in extensions 1 (EXTNAME="T ARRAY"), 2 (EXTNAME="R BAD PIXEL MASK"), and 3 (EXTNAME="T BAD PIXEL MASK"), respectively. The DEMODULATED DATA table is attached unmodified as extension 4. The R and T array images are 3D cubes, with dimension 64x41xN$_{frame}$, where N$_{frame}$ is the number of nod positions in the observation, times the number of HWP positions.

3.1.4 Align Arrays

In order to correctly pair R and T pixels for calculating polarization, and to spatially align all subarrays, the pipeline must reorder the pixels in the raw images. The last row is removed, R1 and T1 subarray images (columns 32-64) are rotated 180 degrees, and then all images are inverted along the y-axis. Small shifts between the R0 and T0 and R1 and T1 subarrays may also be corrected for at this stage. The spatial gap between the 0 and 1 subarrays is also recorded in the ALNGAPX and ALNGAPY FITS header keywords, but is not added to the image; it is accounted for in a later resampling of the image. Note that all corrections applied in this step are integer shifts only; no interpolation is performed. The output images are 64x40xN$_{frame}$.

3.1.5 Split Images

To prepare for combining nod positions and calculating Stokes parameters, the pipeline next splits the data into separate images for each nod position at each HWP angle, calculates the sum and difference of the R and T arrays, and merges the R and T array bad pixel masks. The algorithm uses data from the DEMODULATED DATA table to distinguish the high and low nod positions and the HWP angle. At this stage, any pixel for which there is a good pixel in R but not in T, or vice versa, is noted as a "widow pixel." In the sum image (R+T), each widow pixel's flux is multiplied by 2 to scale it to the correct total intensity. In the merged bad pixel mask, widow pixels are marked with the value 1 (R only) or 2 (T only), so that later steps may handle them appropriately.

The output from this step contains a large number of FITS extensions: one DATA image extension for each of R+T and R-T for each HWP angle and nod position, as well as a TABLE extension containing the demodulated data for each HWP angle and nod position, and a single merged BAD PIXEL MASK image. For a typical Nod-Pol observation with two nod positions and four HWP angles, there are 8 R+T images, 8 R-T images, 8 binary tables, and 1 bad pixel mask image, for 25 extensions total, including the primary HDU. The

output images, other than the bad pixel mask, are 3D cubes with dimension 64x40xN_{chop}, where N_{chop} is the number of chop cycles at the given HWP angle.

3.1.6 Combine Images

The pipeline combines all chop cycles at a given nod position and HWP angle by computing a robust mean of all the frames in the R+T and R-T images. The robust mean is computed at each pixel using Chauvenet's criterion, iteratively rejecting pixels more than 3σ from the mean value, by default. The associated standard deviation across the frames is stored as an error image in the output. The covariances between the pixels may also be calculated and stored in the output, for later use in resampling the images.

The output from this step contains the same FITS extensions as in the previous step, with all images now reduced to 2D images with dimensions 64x40. In addition, there are ERROR and COVAR image extensions for each nod position and HWP angle. Currently, covariances are not used in resampling, so they are not calculated for efficiency reasons; the COVAR images will contain NaN values only. In the example above, with two nod positions and four HWP angles, there are now 57 total extensions, including the primary HDU.

3.1.7 Subtract Beams

In this pipeline step, the sky nod positions (B beams) are subtracted from the source nod positions (A beams) at each HWP angle and for each set of R+T and R-T, and the resulting flux is divided by two for normalization. The errors previously calculated in the combine step are propagated accordingly. The output contains extensions for DATA, ERROR, and COVAR images for each set, as well as a table of demodulated data for each HWP angle, and the bad pixel mask.

3.1.8 Compute Stokes

From the R+T and R-T data for each HWP angle, the pipeline now computes images corresponding to the Stokes I, Q, and U parameters for each pixel.

Stokes I is computed by averaging the R+T signal over all HWP angles:

$$I = \frac{1}{N} \sum_{\phi=1}^{N} (R + T)(\phi),$$

where N is the number of HWP angles and $(R+T)(\phi)$ is the summed R+T flux at the HWP angle ϕ. The associated uncertainty in I is generally propagated from the previously calculated errors for R+T, but may be inflated by the median of the standard deviation of the R+T values across the HWP angles if necessary.

In the most common case of four HWP angles at 0, 45, 22.5, and 67.5 degrees, Stokes Q and U are computed as:

$$Q = \frac{1}{2}[(R-T)(0) - (R-T)(45)]$$

$$U = \frac{1}{2}[(R-T)(22.5) - (R-T)(67.5)]$$

where $(R-T)(\phi)$ is the differential R-T flux at the HWP angle ϕ. Uncertainties in Q and U are propagated from the input error values on R-T.

The output from this step contains an extension for the flux, error, and covariance of each Stokes parameter, as well as the bad pixel mask and a table of the demodulated data, with columns from each of the HWP angles merged. The STOKES I flux image is in the primary HDU. For Nod-Pol data, there will be 10 additional extensions (ERROR I, COVAR I, STOKES Q, ERROR Q, COVAR Q, STOKES U, ERROR U, COVAR U, BAD PIXEL MASK, TABLE DATA). For Chop-Nod imaging, only Stokes I is calculated, so there are 4 additional extensions (ERROR I, COVAR I, BAD PIXEL MASK, TABLE DATA).

3.1.9 Update WCS

To associate the pixels in the Stokes parameter image with sky coordinates, the pipeline uses FITS header keywords describing the telescope position to calculate the reference RA and Dec (CRVAL1/2), the pixel scale (CDELT1/2), and the rotation angle (CROTA2). It may also correct for small shifts in the pixel corresponding to the instrument boresight, depending on the filter used, by modifying the reference pixel (CRPIX1/2). These standard FITS world coordinate system (WCS) keywords are written to the header of the primary HDU.

3.1.10 Correct for Atmospheric Opacity

In order to combine images taken under differing atmospheric conditions, the pipeline corrects the flux in each individual file for the estimated atmospheric transmission during the observation, based on the altitude and zenith angle at the time when the observation was obtained.

Atmospheric transmission values in each HAWC+ filter have been computed for a range of telescope elevations and observatory altitudes (corresponding to a range of overhead precipitable water vapor values) using the ATRAN atmospheric modeling code, provided to the SOFIA program by Steve Lord. The ratio of the transmission at each altitude and zenith angle, relative to that at the reference altitude (41,000 feet) and reference zenith angle (45 degrees), has been calculated for each filter and fit with a low-order polynomial. The ratio appropriate for the altitude and zenith angle of each observation is calculated from the fit coefficients. The pipeline applies this relative opacity correction factor directly to the flux in the Stokes I, Q, and U images, and propagates it into the corresponding error images.

3.1.11 Subtract Background

After chop and nod subtraction, some residual background noise may remain in the flux images. After flat correction, some residual gain variation may remain as well. To remove these, the pipeline reads in all images in a reduction group, and then iteratively performs the following steps:

- Smooth and combine the input Stokes I images
- Compare each Stokes I image (smoothed) to the combined map to determine any background offset or scaling
- Remove offset and scaling from input (unsmoothed) Stokes I images

The final determined offsets (a) and scales (b) for each file are applied to the flux F' for each Stokes image as follows:

$$F'_I = (F_I - a)/b$$
$$F'_Q = F_Q/b$$
$$F'_U = F_U/b$$

and are propagated into the associated error images appropriately.

3.1.12 Subtract Instrumental Polarization

The instrument and the telescope itself may introduce some foreground polarization to the data which must be removed to determine the polarization from the astronomical source. The instrument team characterizes the introduced polarization in reduced Stokes ($q = Q/I$ and $u = U/I$) from the instrument and the telescope for each filter band. The combined reduced Stokes parameters are calculated as

$$q' = q_i + q_t cos(2E) + u_t sin(2E)$$

$$u' = u_i - q_t sin(2E) + u_t cos(2E)$$

where q_i and u_i are the instrumental polarization parameters, q_t and u_t are the telescope polarization parameters, and E is the average telescope elevation during the observation. The correction is then applied as

$$Q' = Q - q'I$$

$$U' = U - u'I$$

and propagated to the associated error images as

$$\sigma'_Q = \sqrt{(q'\sigma_I)^2 + \sigma_Q^2}.$$

$$\sigma'_U = \sqrt{(u'\sigma_I)^2 + \sigma_U^2}.$$

The correction is expected to be good to within $Q/I < 0.6\%$ and $U/I < 0.6\%$.

3.1.13 Rotate Polarization Coordinates

The Stokes Q and U parameters, as calculated so far, reflect polarization angles measured in detector coordinates. After the foreground polarization is removed, the parameters may then be rotated into sky coordinates. The pipeline calculates a relative rotation angle, α, that accounts for the vertical position angle of the instrument, the initial angle of the half-wave plate position, and an offset position that is different for each HAWC filter. It applies it to the Q and U images with a standard rotation matrix:

$$\begin{pmatrix} Q' \\ U' \end{pmatrix} = \begin{bmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{bmatrix} \begin{pmatrix} Q \\ U \end{pmatrix}.$$

Likewise, for the errors σ, the pipeline calculates

$$\begin{pmatrix} \sigma'^2_Q \\ \sigma'^2_U \end{pmatrix} = \begin{bmatrix} cos^2(\alpha) & -sin^2(\alpha) \\ sin^2(\alpha) & cos^2(\alpha) \end{bmatrix} \begin{pmatrix} \sigma^2_Q \\ \sigma^2_U \end{pmatrix},$$

takes the square root, and stores the result as the error images for Stokes Q and U.

3.1.14 Merge Images

All steps up until this point produce an output file for each input file taken at each telescope dither position, without changing the pixelization of the input data. To combine files taken at separate locations into a single map, the pipeline resamples the flux from each onto a common grid, defined such that North is up and East is to the left. The WCS from each

input file is used to determine the sky location of all the input pixels, then, for each pixel in the output grid, the algorithm considers all input pixels within a given radius that are not marked as bad pixels. It weights the input pixels by a Gaussian function of their distance from the grid point and, optionally, their associated errors and/or covariances. The value at the output grid pixel is the weighted average of the input pixels within the considered window.

The output from this step is a single FITS file, containing a flux and error image for each of Stokes I, Q, and U. The dimensions of the output image may vary somewhat, depending on the input parameters to the resampling algorithm, but typically the output pixel scale is similar to the input scale. An image mask is also produced, which represents how many input pixels went into each output pixel. Because of the waiting scheme, the values in this mask are not integers. A data table containing demodulated data merged from all input tables is also attached to the file.

3.1.15 Calibrate Flux

The pipeline now converts the flux units from instrumental counts to physical units of Jy/pixel. For each filter band, the instrument team determines a calibration factor in Jy/pixel/counts appropriate to data that has been opacity-corrected to the reference zenith angle and altitude. This factor is directly applied to the flux in each of the Stokes I, Q, and U and associated error images. The overall calibration is expected to be good to within about 10%.

The output of this step is the final output from the pipeline for Chop-Nod imaging data.

3.1.16 Compute Vectors

Using the Stokes I, Q, and U images, the pipeline now computes the polarization percentage (p) and angle (θ) and their associated errors (σ) in the standard way. For the polarization angle θ in degrees:

$$\theta = \frac{90}{\pi} arctan(\frac{U}{Q})$$

$$\sigma_\theta = \frac{90}{\pi(Q^2+U^2)}\sqrt{Q^2\sigma_U^2 + U^2\sigma_Q^2}.$$

The percent polarization (p) and its error are calculated from the reduced Stokes parameters q and u

$$q = Q/I$$

$$u = U/I$$

and their errors

$$\sigma_q = \sqrt{\sigma_Q^2 + (Q\sigma_I^2)}/I$$

$$\sigma_u = \sqrt{\sigma_U^2 + (U\sigma_I^2)}/I$$

as

$$p = 100\sqrt{q^2 + u^2}$$

$$\sigma_p = \frac{1}{p}\sqrt{q^2\sigma_q^2 + u^2\sigma_u^2}.$$

The debiased polarization percentage (p')is also calculated, as:

$$p' = 100\sqrt{p^2 - \sigma_p^2}.$$

Each of the θ, p, and p' maps and their error images are stored as separate extensions in the output from this step, which is the final output from the pipeline for Nod-Pol data. This file will have 15 extensions, including the primary HDU, with extension names, types, and numbers as follows:

- STOKES I: primary HDU, image, extension 0
- ERROR I: image, extension 1
- STOKES Q: image, extension 2
- ERROR Q: image, extension 3
- STOKES U: image, extension 4
- ERROR U: image, extension 5
- IMAGE MASK: image, extension 6
- PERCENT POL: image, extension 7
- DEBIASED PERCENT POL: image, extension 8
- ERROR PERCENT POL: image, extension 9
- POL ANGLE: image, extension 10
- ROTATED POL ANGLE: image, extension 11
- ERROR POL ANGLE: image, extension 12
- MERGED DATA: table, extension 13
- POL DATA: table, extension 14

The final extension contains a table representation of the polarization values for each pixel, as an alternate representation of the θ, p, and p' maps.

3.2 Scan Reduction Algorithms

This section covers the main algorithms used to reduce Scan mode data with CRUSH. It is meant to give the reader an accurate, if incomplete, overview of the principal reduction process.

3.2.1 Signal Structure

CRUSH is based on the assumption that the measured data (X_{ct}) for detector c, recorded at time t, is the superposition of various signal components and essential (not necessarily white) noise n_{ct}:

$$X_{ct} = D_{ct} + g_{(1),c}C_{(1),t} + ... + g_{(n),c}C_{(n),t} + G_c M_{ct}^{xy} S_{xy} + n_{ct}$$

We can model the measured detector timestreams via a number of appropriate parameters, such as 1/f drifts (D_{ct}), n correlated noise components ($C_{(1),t}...C_{(n),t}$) and channel responses to these (gains, $g_{(1),c}...g_{(n),c}$), and the observed source structure (S_{xy}). We can derive statistically sound estimates (such as maximum-likelihood or robust estimates) for these parameters based on the measurements themselves. As long as our model is representative of the physical processes that generate the signals, and sufficiently complete, our derived parameters should be able to reproduce the measured data with the precision of the underlying limiting noise.

Below is a summary of the principal model parameters assumed by CRUSH, in general:

- X_{ct}: The raw timestream of channel c, measured at time t.
- D_{ct}: The 1/f drift value of channel c at time t.
- $g_{(1),c}...g_{(n),c}$: Channel c gain (response) to correlated signals (for modes 1 through n).
- $C_{(1),t}...C_{(n),t}$: Correlated signals (for modes 1 through n) at time t.
- G_c: The point source gain of channel c
- M_{ct}^{xy}: Scanning pattern, mapping a sky position $\{x, y\}$ into a sample of channel c at time t.
- S_{xy}: Actual 2D source flux at position $\{x, y\}$.
- n_{ct}: Essential limiting noise in channel c at time t.

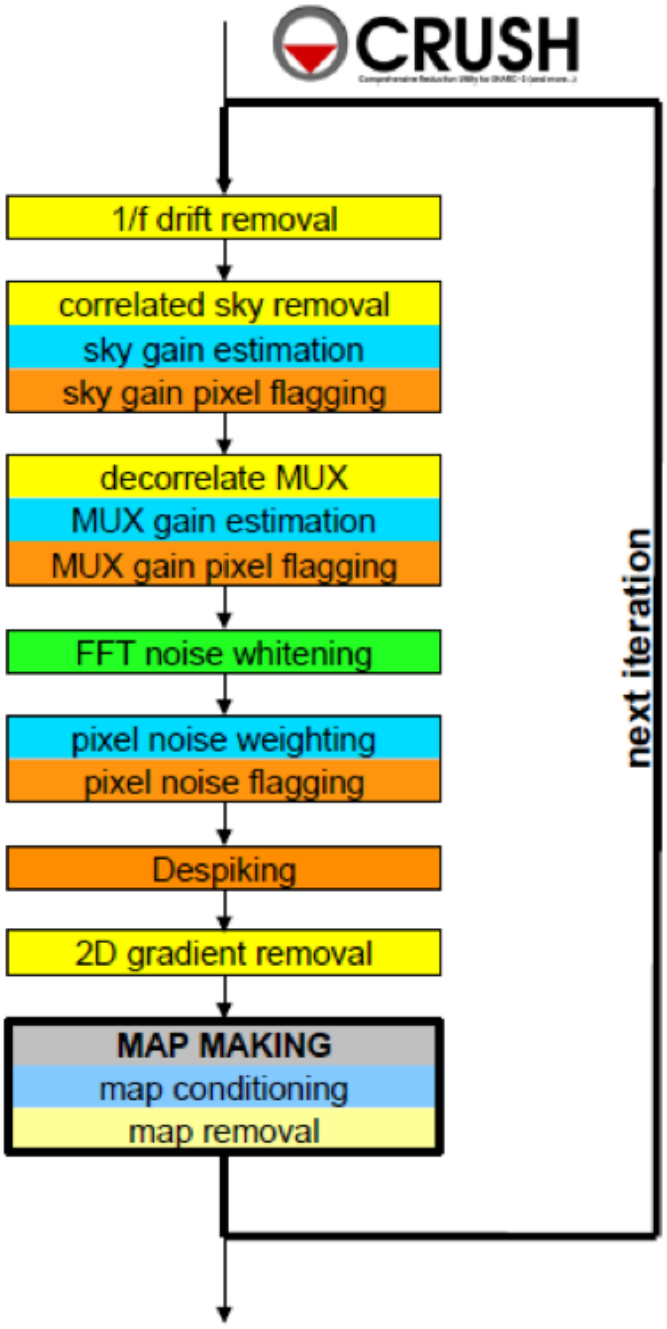

Figure 5: Scan data reduction flowchart

3.2.2 Sequential Incremental Modeling and Iterations

The approach of CRUSH is to solve for each term separately, and sequentially, rather than trying to do a brute-force matrix inversion in a single step. Such inversions are not practical for several reasons, anyway: (1) because they require a-priori knowledge of all gains and weights (covariance matrix) with great precision, (2) because they require bad data to be identified prior to inversion, (3) because degeneracies are not handled in a controlled / controllable way, (4) because linear inversions do not handle non-linearities with ease (such as solving for both gains and signals when these form a product), (5) because of the need to include spectral filtering, typically, and (6) because matrix inversions are computationally costly.

Sequential modeling works on the assumption that each term can be considered independently from one another. To a large degree this is granted as many of the signals produce more or less orthogonal imprints in the data (e.g. you cannot easily mistake correlated sky response seen by all channels with a per-channel DC offset). As such, from the point of view of each term, the other terms represent but an increased level of noise. As the terms all take turns in being estimated (usually from bright to faint) this model confusion "noise" goes away, especially with iterations.

Even if the terms are not perfectly orthogonal to one another, and have degenerate flux components, the sequential approach handles this naturally. Degenerate fluxes between a pair of terms will tend to end up in the term that is estimated first. Thus, the ordering of the estimation sequence provides a control on handling degeneracies in a simple and intuitive manner.

A practical trick for efficient implementation is to replace the raw timestream with the unmodeled residuals $X_{ct} \rightarrow R_{ct}$, and let modeling steps produce incremental updates to the model parameters. Every time a model parameter is updated, its incremental imprint is removed from the residual timestream (a process we shall refer to a synchronization).

With each iteration, the incremental changes to the parameters become more insignificant, and the residual will approach the limiting noise of the measurement.

3.2.3 DC Offset and 1/f Drift Removal

For 1/f drifts, consider only the term:

$$R_{ct} \approx \delta D_{c\tau}$$

where $\delta D_{c\tau}$ is the 1/f channel drift value for t between τ and $\tau + T$, for a 1/f time window of T samples. That is, we simply assume that the residuals are dominated by an unmodeled

1/f drift increment $\delta D_{c\tau}$. Note that detector DC offsets can be treated as a special case with $\tau = 0$, and T equal to the number of detector samples in the analysis.

We can construct a χ^2 measure, as:

$$\chi^2 = \sum_{c,t=\tau}^{t=\tau+T} w_{ct}(R_{ct} - \delta D_{ct})^2$$

where $w_{ct} = \sigma_{ct}^{-2}$ is the proper noise-weight associated with each datum. CRUSH furthermore assumes that the noise weight of every sample w_{ct} can be separated into the product of a channel weight w_c and a time weight w_t, i.e. $w_{ct} = w_c \cdot w_t$. This assumption is identical to that of separable noise ($\sigma_{ct} = \sigma_c \cdot \sigma_t$). Then, by setting the χ^2 minimizing condition $\partial\chi^2/\partial(\delta D_{ct}) = 0$, we arrive at the maximum-likelihood incremental update:

$$\delta D_{c\tau} = \frac{\sum_{t=\tau}^{\tau+T} w_t R_{ct}}{\sum_{t=\tau}^{\tau+T} w_t}$$

Note, that each sample (R_{ct}) contributes a fraction:

$$p_{ct} = w_t / \sum_{t=\tau}^{\tau+T} w_t$$

to the estimate of the single parameter $\delta D_{c\tau}$. In other words, this is how much that parameter is *dependent* on each data point. Above all, p_{ct} is a fair measure of the fractional degrees of freedom lost from each datum, due to modeling of the 1/f drifts. We will use this information later, when estimating proper noise weights.

Note, also, that we may replace the maximum-likelihood estimate for the drift parameter with any other statistically sound estimate (such as a weighted median), and it will not really change the dependence, as we are still measuring the same quantity, from the same data, as with the maximum-likelihood estimate. Therefore, the dependence calculation remains a valid and fair estimate of the degrees of freedom lost, regardless of what statistical estimator is used.

The removal of 1/f drifts must be mirrored in the correlated signals also if gain solutions are to be accurate. Finally, following the removal of drifts, CRUSH will check the timestreams for inconsistencies. For example, HAWC data is prone to discontinuous jumps in flux levels. CRUSH will search the timestream for flux jumps, and flag or fix jump-related artifacts as necessary.

3.2.4 Correlated Noise Removal and Gain Estimation

For the correlated noise (mode i), we shall consider only the term with the incremental signal parameter update:

$$R_{ct} = g_{(i),c}\delta C_{(i),t} + ...$$

Initially, we can assume $C_{(i),t}$ as well as $g_{(i),c} = 1$, if better values of the gain are not independently known at the start. Accordingly, the χ^2 becomes:

$$\chi^2 = \sum_c w_{ct}(R_{ct} - g_{(i),c}\delta C_{(i),t})^2.$$

Setting the χ^2 minimizing condition with respect to $\delta C_{(i),t}$ yields:

$$\delta C_{(i),t} = \frac{\sum_c w_c g_{(i),c} R_{ct}}{\sum_c w_c g^2_{(i),c}}.$$

The dependence of this parameter on R_{ct} is:

$$p_{ct} = w_c g^2_{(i),c} / \sum_c w_c g^2_{(i),c}$$

After we update $C_{(i)}$ (the correlated noise model for mode i) for all frames t, we can update the gain response as well in an analogous way, if desired. This time, consider the residuals due to the unmodeled gain increment:

$$R_{ct} = \delta g_{(i),c} C_{(i),t} + ...$$

and

$$\chi^2 = \sum_t w_{ct}(R_{ct} - \delta g_{(i),c} C_{(i),t})^2$$

Minimizing it with respect to $\delta g_{(i),c}$ yields:

$$\delta g_{(i),c} = \frac{\sum_t w_t C_{(i),t} R_{ct}}{\sum_t w_t C^2_{(i),t}}$$

which has a parameter dependence:

$$p_{ct} = w_t C^2_{(i),t} / \sum_t w_t C^2_{(i),t}$$

Because the signal C_t and gain g_c are a product in our model, scaling C_t by some factor X, while dividing g_c by the same factor will leave the product intact. Therefore, our

solutions for C_t and g_c are not unique. To remove this inherent degeneracy, it is practical to enforce a normalizing condition on the gains, such that the mean gain $\mu(g_c) = 1$, by construct. CRUSH uses a robust mean measure for gain normalization to produce reasonable comparisons under various pathologies, such as when most gains are zero, or when a few gains are very large compared to the others.

Once again, the maximum-likelihood estimate shown here can be replaced by other statistical measures (such as a weighted median), without changing the essence.

3.2.5 Noise Weighting

Once we model out the dominant signal components, such that the residuals are starting to approach a reasonable level of noise, we can turn our attention to determining proper noise weights. In its simplest form, we can determine the weights based on the mean observed variance of the residuals, normalized by the remaining degrees of freedom in the data:

$$w_c = \eta_c \frac{N_{(t),c} - P_c}{\sum_t w_t R_{ct}^2}$$

where $N_{(t),c}$ is the number of unflagged data points (time samples) for channel c, and P_c is the total number of parameters derived from channel c. The scalar value η_c is the overall spectral filter pass correction for channel c (see section 3.2.7), which is 1 if the data was not spectrally filtered, and 0 if the data was maximally filtered (i.e. all information is removed). Thus typical η_c values will range between 0 and 1 for rejection filters, or can be greater than 1 for enhancing filters. We determine time-dependent weights as:

$$w_t = \frac{N_{(c),t} - P_t}{\sum_c w_c R_{ct}^2}$$

Similar to the above, here $N_{(c),t}$ is the number of unflagged channel samples in frame t, while P_t is the total number of parameters derived from frame t. Once again, it is practical to enforce a normalizing condition of setting the mean time weight to unity, i.e. $\mu(w_t) = 1$. This way, the channel weights w_c have natural physical weight units, corresponding to $w_c = 1/\sigma_c^2$.

The total number of parameters derived from each channel, and frame, are simply the sum, over all model parameters m, of all the parameter dependencies p_{ct} we calculated for them. That is,

$$P_c = \sum_m \sum_t p_{(m),ct}$$

and

$$P_t = \sum_m \sum_c p_{(m),ct}$$

Getting these lost-degrees-of-freedom measures right is critical for the stability of the solutions in an iterated framework. Even slight biases in p_{ct} can grow exponentially with iterations, leading to divergent solutions, which may manifest as over-flagging or as extreme mapping artifacts.

Of course, one may estimate weights in different ways, such as based on the median absolute deviation (robust weights), or based on the deviation of differences between nearby samples (differential weights). As they all behave the same for white noise, there is really no significant difference between them. CRUSH does, optionally, offer those different (but comparable) methods of weight estimation.

3.2.6 Despiking

After deriving fair noise weights, we can try to identify outliers in the data (glitches and spikes) and flag them for further analysis. Despiking is a standard procedure that need not be discussed here in detail. CRUSH offers a few variants of the basic method, depending on whether it looks for absolute deviations, differential deviations between nearby data, or spikes at different resolutions (multires) at once.

3.2.7 Spectral Conditioning

Ideally, detectors would have featureless white noise spectra (at least after the 1/f noise is treated by the drift removal). In practice, that is rarely the case. Spectral features are bad because (a) they produce mapping features/artifacts (such as "striping"), and because (b) they introduce a covariant noise term between map points that is not easily represented by the output. It is therefore desirable to "whiten" the residual noise whenever possible, to mitigate both these effects.

Noise whitening starts with measuring the effective noise spectrum in a temporal window, significantly shorter than the integration on which it is measured. In CRUSH, the temporal window is designed to match the 1/f stability timescale T chosen for the drift removal, since the drift removal will wipe out all features on longer timescales. With the use of such a spectral window, we may derive a lower-resolution averaged power-spectrum for each channel. CRUSH then identifies the white noise level, either as the mean (RMS) scalar amplitude over a specified range of frequencies, or automatically, over an appropriate frequency range occupied by the point-source signal as a result of the scanning motion.

Then, CRUSH will look for significant outliers in each spectral bin, above a specified level (and optimally below a critical level too), and create a real-valued spectral filter profile ϕ_{cf} for each channel c and frequency bin f to correct these deviations.

There are other filters that can be applied also, such as notch filters, or a motion filter to reject responses synchronous to the dominant telescope motion. In the end, every one of these filters is represented by an appropriate scalar filter profile ϕ_{cf}, so the discussion remains unchanged.

Once a filter profile is determined, we apply the filter by first calculating a rejected signal:

$$\varrho_{ct} = F^{-1}[(1 - \phi_{cf})\hat{R}_{cf}]$$

where $\hat{R}_{cf}$ is the Fourier transform of R_{ct}, using the weighting function provided by w_t, and F^{-1} denotes the inverse Fourier Transform from the spectral domain back into the timestream. The rejected signals are removed from the residuals as:

$$R_{ct} \rightarrow R_{ct} - \varrho_{ct}$$

The overall filter pass η_c for channel c, can be calculated as:

$$\eta_c = \frac{\sum_f \phi_{cf}^2}{N_f}$$

where N_f is the number of spectral bins in the profile ϕ_{cf}. The above is simply a measure of the white-noise power fraction retained by the filter, which according to Parseval's theorem, is the same as the power fraction retained in the timestream, or the scaling of the observed noise variances as a result of filtering.

3.2.8 Map Making

The mapping algorithm of CRUSH implements a nearest-pixel method, whereby each data point is mapped entirely into the map pixel that falls nearest to the given detector channel c, at a given time t. Distributing the flux to neighboring pixels would constitute smoothing, and as such, it is better to smooth maps explicitly by a desired amount as a later processing step. Here,

$$\delta S_{xy} = \frac{\sum_{ct} M_{xy}^{ct} w_c w_t \varkappa_c G_c R_{ct}}{\sum_{ct} M_{xy}^{ct} w_c w_t \varkappa_c^2 G_c^2}$$

where M_{xy}^{ct} associates each sample $\{c, t\}$ uniquely with a map pixel $\{x, y\}$, and is effectively the transpose of the mapping function defined earlier. $\varkappa_c$ is the point-source filtering (pass) fraction of the pipeline. It can be thought of as a single scalar version of the transfer function. Its purpose is to measure how isolated point-source peaks respond to the various reduction steps, and correct for it. When done correctly, point source peaks will always stay perfectly cross-calibrated between different reductions, regardless of what reduction

steps were used in each case. More generally, a reasonable quality of cross-calibration (to within 10%) extends to compact and slightly extended sources (typically up to about half of the field-of-view (FoV) in size). While corrections for more extended structures ($\geq$ FoV) are possible to a certain degree, they come at the price of steeply increasing noise at the larger scales.

The map-making algorithm should skip over any data that is unsuitable for quality map-making (such as too-fast scanning that may smear a source). For formal treatment, we can just assume that $M_{ct}^{xy} = 0$ for any troublesome data.

Calculating the precise dependence of each map point S_{xy} on the timestream data R_{ct} is computationally costly to the extreme. Instead, CRUSH gets by with the approximation:

$$p_{ct} \approx N_{xy} \cdot \frac{w_t}{\sum_t w_t} \cdot \frac{w_c \varkappa_c^2 G_c}{\sum_c w_c \varkappa_c^2 G_c^2}$$

This approximation is good as long as most map points are covered with a representative collection of pixels, and as long as the pixel sensitivities are more or less uniformly distributed over the field of view. So far, the inexact nature of this approximation has not produced divergent behavior with any of the dozen or more instruments that CRUSH is being used with. Its inaccuracy is of no grave concern as a result.

We can also calculate the flux uncertainty in the map σ_{xy} at each point $\{x, y\}$ as:

$$\sigma_{xy}^2 = 1 / \sum_{ct} M_{xy}^{ct} w_c w_t \varkappa_c^2 G_c^2$$

Source models are first derived from each input scan separately. These may be despiked and filtered, if necessary, before added to the global increment with an appropriate noise weight (based on the observed map noise) if source weighting is desired.

Once the global increment is complete, we can add it to the prior source model $S_{xy}^{r(0)}$ and subject it to further conditioning, especially in the intermediate iterations. Conditioning operations may include smoothing, spatial filtering, redundancy flagging, noise or exposure clipping, signal-to-noise blanking, or explicit source masking. Once the model is processed into a finalized S'_{xy}, we synchronize the incremental change $\delta S'_{xy} = S'_{xy} - S_{xy}^{r(0)}$ to the residuals:

$$R_{ct} \rightarrow R_{ct} - M_{ct}^{xy}(\delta G_c S_{xy}^{r(0)} + G_c \delta S'_{xy})$$

Note, again, that $\delta S'_{xy} \neq \delta S_{xy}$. That is, the incremental change in the conditioned source model is not the same as the raw increment derived above. Also, since the source gains G_c may have changed since the last source model update, we must also re-synchronize the prior source model $S_{xy}^{(0)}$ with the incremental source gain changes δG_c (first term inside the brackets).

Typically, CRUSH operates under the assumption that the point-source gains G_c of the detectors are closely related to the observed sky-noise gains g_c derived from the correlated noise for all channels. Specifically, CRUSH treats the point-source gains as the product:

$$G_c = \varepsilon_c g_c g_s e^{-\tau}$$

where ε_c is the point-source coupling efficiency. It measures the ratio of point-source gains to sky-noise gains (or extended source gains). Generally, CRUSH will assume $\varepsilon_c = 1$, unless these values are measured and loaded during the scan validation sequence. Optionally, CRUSH can also derive ε_c from the observed response to a source structure, provided the scan pattern is sufficient to move significant source flux over all detectors. The source gains also include a correction for atmospheric attenuation, for an optical depth τ, in-band and in the line of sight. Finally, a gain term g_s for each input scan may be used as a calibration scaling/correction on a per-scan basis.

3.2.9 Point-Source Flux Corrections

We mentioned point-source corrections in the section above; here, we explain how these are calculated. First, consider drift removal. Its effect on point source fluxes is a reduction by a factor:

$$\varkappa_{D,c} \approx 1 - \frac{\tau_{pnt}}{T}$$

In terms of the 1/f drift removal time constant T and the typical point-source crossing time τ_{pnt}. Clearly, the effect of 1/f drift removal is smaller the faster one scans across the source, and becomes negligible when $\tau_{pnt} \ll T$.

The effect of correlated-noise removal, over some group of channels of mode i, is a little more complex. It is calculated as:

$$\varkappa_{(i),c} = 1 - \frac{1}{N_{(i),t}}(P_{(i),c} + \sum_k \Omega_{ck} P_{(i),k})$$

where Ω_{ck} is the overlap between channels c and k. That is, Ω_{ck} is the fraction of the point source peak measured by channel c when the source is centered on channel k. $N_{(i),t}$ is the number of correlated noise-samples that have been derived for the given mode (usually the same as the number of time samples in the analysis). The correlated model's dependence on channel c is:

$$P_{(i),c} = \sum_t p_{(i),ct}$$

Finally, the point-source filter correction due to spectral filtering is calculated based on the average point-source spectrum produced by the scanning. Gaussian source profiles with

spatial spread $\sigma_x \approx FWHM/2.35$ produce a typical temporal spread $\sigma_t \approx \sigma_x/\bar{v}$, in terms of the mean scanning speed $\bar{v}$. In frequency space, this translates to a Gaussian frequency spread of $\sigma_f = (2\pi\sigma_t)^{-1}$, and thus a point-source frequency profile of:

$$\Psi_f \approx e^{-f^2/(2\sigma_f^2)}$$

More generally, Ψ_f may be complex-valued (asymmetric beam). Accordingly, the point-source filter correction due to filtering with ϕ_f is generally:

$$\varkappa_{\phi,c} \approx \frac{\sum_f Re(\phi_f \Psi_f \phi_f)}{\sum_f Re(\Psi_f)}$$

The compound point source filtering effect from m model components is the product of the individual model corrections, i.e.:

$$\varkappa_c = \prod_m \varkappa_{(m),c}$$

This concludes the discussion of the principal reduction algorithms of CRUSH for HAWC Scan mode data. For more information, see section 3.3.

3.2.10 CRUSH output

Since the CRUSH algorithms are iterative, there are no well-defined intermediate products that may be written to disk. For Scan mode data, the pipeline takes as input a set of raw Level 0 HAWC FITS files, described in section 3.1.1, and writes as output a single FITS file containing an image of the source map, and several other extensions. The primary HDU in the output file contains the flux image (EXTNAME = SIGNAL) in units of Jy/pixel. The first extension (EXTNAME = EXPOSURE) contains an image of the nominal exposure time in seconds at each point in the map. The second extension (EXTNAME = NOISE) holds the error image corresponding to the flux map, and the third extension (EXTNAME = S/N) is the signal-to-noise ratio of the flux to the error image. The fourth and further extensions contain binary tables of data, one for each input scan.

3.3 Other Resources

For more information on the code or algorithms used in the HAWC DRP or the CRUSH pipelines, see the following documents:

DRP:

- Far-infrared polarimetry analysis: Hildebrand et. al. 2000 PASP, 112, 1215
- DRP infrastructure and image viewer: Berthoud, M. 2013 ADASS XXII, 475, 193

CRUSH:

- CRUSH paper: Kovács, A. 2008, Proc. SPIE, 7020, 45
- CRUSH thesis: Kovács, A. 2006, PhD Thesis, Caltech
- Online documentation: `http://www.submm.caltech.edu/~sharc/crush`

4 Data Products

4.1 File names

Output files from the HAWC pipeline are named according to the convention:

FILENAME = F[flight]_HA_[mode]_[aorid]_[spectel]_[type]_[fn1[-fn2]].fits

where *flight* is the SOFIA flight number, *HA* indicates the instrument (HAWC+), and *mode* is either *IMA* for imaging observations, *POL* for polarization observations, or *CAL* for diagnostic data. The *aorid* indicates the SOFIA program and observation number, *spectel* indicates the filter/band and the HWP setting. The *type* is a three-letter identifier for the pipeline product type, and *fn1* and *fn2* are the first and last raw file numbers that were combined to produce the output product. For example, a polarization vector data product with AOR-ID 81_0131_04 derived from files 5 to 6 of flight 295, taken in Band A with HWP in the A position would have the filename *F0295_HA_POL_81013104_HAWAHWPA_VEC_005-006.fits*. See the tables below for a list of all possible values for the three-letter product type.

4.2 Data format

Most HAWC data is stored in FITS files, conforming to the FITS standard (Pence et al. 2010). Each FITS file contains a primary Header Data Unit (HDU) which may contain the most appropriate image data for that particular data reduction level. Most files have additional data stored in HDU image or table extensions. All keywords describing the file are in the header of the primary HDU. Each HDU has its own header and is identified by the EXTNAME header keyword. The algorithm descriptions, above, give more information about the content of each extension.

4.3 Pipeline products

The following tables list all intermediate and final products that may be generated by the HAWC pipeline, in the order in which they are produced for each mode. The product type is stored in the primary header, under the keyword PRODTYPE. By default, for Nod-Pol mode, the *shift*, *wcs*, *calibrate*, and *polvec* products are saved. For Chop-Nod mode, the *shift*, *wcs*, *merge*, and *calibrate* products are saved. For Scan mode, only the *crush* product is produced or saved.

Table 1: Nod-Pol mode intermediate and final pipeline data products

Step	Description	PRODTYPE	PROCSTAT	Identifier	Saved
Demodulate	Chops subtracted	demod	LEVEL_1	DMD	N
Flat Correct	Flat field correction applied	flat	LEVEL_2	FLA	N
Align Arrays	R array shifted to T array	shift	LEVEL_2	SFT	Y
Split Images	Data split by nod, HWP	split	LEVEL_2	SPL	N
Combine Images	Chop cycles combined	combine	LEVEL_2	CMB	N
Subtract Beams	Nod beams subtracted	nodpolsub	LEVEL_2	NPS	N
Compute Stokes	Stokes parameters calculated	stokes	LEVEL_2	STK	N
Update WCS	WCS added to header	wcs	LEVEL_2	WCS	Y
Correct Opacity	Corrected for atmospheric opacity	opacitymodel	LEVEL_2	OPC	N
Subtract Background	Residual background removed	bgsubtract	LEVEL_2	BGS	N
Subtract IP	Instrumental polarization removed	ip	LEVEL_2	IPS	N
Rotate Coordinates	Polarization angle corrected to sky	rotate	LEVEL_2	ROT	N
Merge Images	Dithers merged to single map	merge	LEVEL_2	MRG	N
Calibrate Flux	Flux calibrated to physical units	calibrate	LEVEL_3	CAL	Y
Compute Vectors	Polarization vectors calculated	polvec	LEVEL_4	VEC	Y

Table 2: Chop-Nod mode intermediate and final pipeline data products

Step	Description	PRODTYPE	PROCSTAT	Identifier	Saved
Demodulate	Chops subtracted	demod	LEVEL_1	DMD	N
Flat Correct	Flat field correction applied	flat	LEVEL_2	FLA	N
Align Arrays	R array shifted to T array	shift	LEVEL_2	SFT	Y
Split Images	Data split by nod, HWP	split	LEVEL_2	SPL	N
Combine Images	Chop cycles combined	combine	LEVEL_2	CMB	N
Subtract Beams	Nod beams subtracted	nodpolsub	LEVEL_2	NPS	N
Compute Stokes	Stokes parameters calculated	stokes	LEVEL_2	STK	N
Update WCS	WCS added to header	wcs	LEVEL_2	WCS	Y
Correct Opacity	Corrected for atmospheric opacity	opacitymodel	LEVEL_2	OPC	N
Subtract Background	Residual background removed	bgsubtract	LEVEL_2	BGS	N
Merge Images	Dithers merged to single map	merge	LEVEL_2	MRG	Y
Calibrate Flux	Flux calibrated to physical units	calibrate	LEVEL_3	CAL	Y

Table 3: Scan mode final pipeline data product

Step	Description	PRODTYPE	PROCSTAT	Identifier	Saved
CRUSH	Source model derived iteratively with CRUSH	crush	LEVEL_3	CRH	Y

5 Grouping Level 0 Data for Processing

In order for the pipeline to successfully reduce a group of HAWC+ data together, all input data must share a common instrument configuration and observation mode, as well as target and filter band and HWP setting. These requirements translate into a set of FITS header keywords that must match in order for a set of data to be grouped together. These keyword requirements are summarized in the table below, for imaging and polarimetry data.

Table 4: Grouping Criteria for Imaging and Polarimetry Modes

Mode	Keyword	Data Type	Match Criterion
Both	OBSTYPE	string	exact
Both	OBJECT	string	exact
Both	INSTCFG	string	exact
Both	CALMODE	string	exact
Both	INSTMODE	string	exact
Both	SPECTEL1	string	exact
Both	SPECTEL2	string	exact
Both	PLANID	string	exact
Imaging only	SCNPATT	string	exact
Polarimetry only	NHWP	float	exact

6 Configuration and Execution

6.1 Installation

The HAWC pipeline is written in Python with additional modules in other languages, including C and Java. The pipeline is platform independent and has been tested on Windows, Linux, and Mac operating systems.

6.1.1 External Requirements

To run the pipeline for any mode from the DRP interface, Python (preferably 2.7 or later) is required as well as the following packages: numpy, scipy, matplotlib/pylab, astropy, logging, and configobj. Some lab and diagnostic pipeline steps may also require the pIDLy library and IDL, but the default science reductions do not. The DRP includes some C libraries that require gcc with the omp.h library to compile; gcc version 4.9 or higher is recommended.

To run the pipeline on Scan mode data with CRUSH, Java v1.7.0 or higher is also required. It can be installed, if necessary, from java.com, for example.

6.1.2 Source Code Installation

The source code for the DRP pipeline maintained by the SOFIA Data Processing Systems (DPS) team can be obtained directly from the *hawc* git repository there. This version contains all needed configuration files, auxiliary files, and Python, C, and Java code to run the pipeline on HAWC data in any observation mode. CRUSH is also available as a standalone package, and may be downloaded separately if desired, via the CRUSH website or its SourceForge page.

After obtaining the source code, the C libraries used by the DRP must be compiled. To do so, run

```
make
```

from the *hawc/pipeline/src/lib* directory.

To run the Python code, the PYTHONPATH environment variable must be set to include the DRP source folder. To use default configurations and auxiliary files, it is also recommended to set a DPS_HAWCPIPE environment variable to include the top-level *hawc* directory. For example, from bash, set:

```
export DPS_HAWCPIPE=/path/to/hawc
export PYTHONPATH=$DPS_HAWCPIPE/pipeline/src
```

For a shortcut to the command-line version of the pipeline on UNIX/Linux systems, you may also wish to set an alias as follows:

```
alias hawcpipe="python $DPS_HAWCPIPE/pipeline/src/dcs/dcspipe.py"
```

This shortcut will be used in the rest of this document when referring to the command-line interface.

6.2 Configuration

The DRP pipeline requires a valid and complete configuration file to run. Configuration files are written in plain text, in a format readable by the configobj Python library. These files are divided into sections, specified by brackets (e.g. [section]), each of which may contain keyword-value pairs or subsections (e.g. [[subsection]]). The HAWC configuration file must contain the following sections:

- General pipeline configuration, including the list of DRP modules to use
- Data configuration, including specifications for input and output file names and formats, and specifications for metadata handling
- Pipeline mode definitions for each supported instrument mode, including the FITS keywords that define the mode and the list of steps to run
- Pipeline step parameter definitions (one step for each pipeline step defined)

In the DPS environment, the pipeline is usually run with a default configuration file (*hawc/pipeline/config/pipeconf_dcs.txt*), which defines all standard reduction steps and default parameters. It may be overridden with date-specific default values, defined in (*hawc/pipeline/config/overrides/*), or with user-defined parameters. Override configuration files specified to *hawcpipe* may contain any subset of the values in the full configuration file. See Appendix B for examples of override configuration files as well as the full default file.

The CRUSH pipeline, run as a single pipeline step for Scan mode data, also has its own separate set of configuration files. These files are stored with the CRUSH distribution included with the HAWC pipeline, in *hawc/crush*. They are read from this sub-directory in the order specified below.

Upon launch, CRUSH will invoke the default configuration files (*default.cfg*) in the following order:

1. Global defaults from *crush/default.cfg*
2. Global user overrides from *∼/.crush2/default.cfg*
3. Instrument overrides from *crush/hawc+/default.cfg*
4. Instrument user overrides from *∼/.crush2/hawc+/default.cfg*

Any configuration file may invoke further (nested) configurations, which are located and loaded in the same order as above. For example, *hawc+/default.cfg* inside CRUSH invokes *sofia/default.cfg* first, which contains settings for SOFIA instruments in general, which are not HAWC+ specific.

There are also modified configurations for "bright", "faint", or "deep" sources, when one of these flags is used while invoking crush. For example:

```
crush hawc+ -faint ...
```

will invoke faint mode reduction, by parsing *faint.cfg* from the above locations, after *default.cfg* was parsed. Similarly, there can be *bright.cfg* and *deep.cfg* files specifying modified configurations for bright and deep modes. CRUSH ships with reasonably fine-tuned versions of all configurations to provide quasi-optimal results out of the box.

Users are discouraged from modifying the configuration files included in the CRUSH distribution directly. Instead, they should add relevant entries in their own user-specific configuration files under *∼/.crush2/* (for global settings) and *∼/.crush2/hawc+/* (for HAWC+ specific settings). This way, their configuration changes will survive updates to CRUSH.

A simple guide to the configuration syntax is found in *README.syntax* (inside the crush distribution and online). Common configuration options for HAWC+ are discussed in *hawc+/README* (also included in the distribution, and available online). Yet more useful options are covered by the main README (included in the distribution and online). Finally, a complete list of all available options and their descriptions is to be found in the GLOSSARY (inside the distribution and online). See Appendix B for an example of current CRUSH configuration files.

When CRUSH is called from the DRP, it invokes CRUSH command-line options via a parameter defined in its configuration file. Most parameters in the CRUSH configuration files can be overridden from DRP via this method. For example, the faint mode configuration can be invoked by adding it to the *options* parameter in the *[crush]* section of the DRP configuration file, after any other standard options:

```
[crush]
    options = 'hawc+ -drp -unit=Jy/pixel -blacklist=smooth -faint'
```

6.3 Input Data

The HAWC pipeline takes as input raw HAWC data files, which contain binary tables of instrument readouts and metadata. The FITS headers contain data acquisition and observation parameters and, combined with the pipeline configuration files and other auxiliary files on disk, comprise the information necessary to complete all steps of the data reduction process. Some critical keywords are required to be present in the raw data in order to perform a successful grouping, reduction, and ingestion into the SOFIA archive. These are defined in the DRP pipeline in a configuration file that describes the allowed values for each keyword in the configobj format (see Appendix C).

It is assumed that the input data have been successfully grouped before beginning reduction: the pipeline considers all input files in a reduction to be science files that are part of a single homogeneous reduction group, to be reduced together with the same parameters.

6.3.1 Auxiliary Files

In order to complete a standard reduction, the pipeline requires a number of files to be on disk, with locations specified in the DRP configuration file. Current default files described in the default configuration are stored along with the code, typically in the *hawc/pipeline/data* directory. See below for a table of all commonly used types of auxiliary files.

Table 5: Auxiliary files used by DRP reductions for Chop-Nod and Nod-Pol data

Auxiliary File	File Type	Pipe Step	Comments
Phase	FITS	Demodulate	Contains phase delay in seconds for each pixel
Flat	FITS	Flat Correct	Contains normalized flat field image and bad pixel mask, one per band
Response	ASCII	Opacity Correct	Contains instrumental response coefficients by altitude, ZA

The phase files used in the Demodulate step should be in FITS format, with two HDUs containing phase information for the R and T arrays, respectively. The phases are stored as images that specify the timing delay, in seconds, for each pixel.

The flat files used in the Flat Correct step are also in FITS format. They should have four image extensions: R Array Gain, T Array Gain, R Bad Pixel Mask, and T Bad Pixel Mask. The image in each extension should match the dimensions of the R and T arrays in the demodulated data (currently 64 x 41 pixels). The Gain images should contain multiplicative floating-point flat correction factors, normalized to a median of 1.0, to be applied to the demodulated flux. The Bad Pixel Mask images should be integer arrays, with value 0 (good), 1 (bad in R array), or 2 (bad in T array). Bad pixels, corresponding to those marked 1 or 2 in the mask extensions, should be set to NaN in the flat images. At a minimum, the primary FITS header for the flat file should contain the SPECTEL1 and SPECTEL2 keywords, for matching the flat band to the input demodulated files.

The instrumental response coefficients are stored in ASCII text files, with at least four white-space delimited columns as follows: filter wavelength, filter name, response reference value, and fit coefficient constant term. Any remaining columns are further polynomial terms in the response fit. The independent variable in the polynomial fit is indicated by the response filename: if it contains *airmass*, the independent variable is zenith angle (ZA); if *alt*, the independent variable is altitude in thousands of feet; if *pwv*, the independent variable is precipitable water vapor, in μm. The reference values for altitude, ZA, and PWV are listed in the headers of the text files, in comment lines preceded with #.

6.4 Automatic Mode Execution

The DPS pipeline infrastructure runs the pipeline on previously-defined reduction groups as a fully-automatic black box. To do so, it creates an input manifest (*infiles.txt*) that contains relative paths to the input files (one per line), and prepends the total number of input files to the top of the manifest. The pipeline modes and steps to run are defined in the *pipeconf_dcs.txt* configuration file included with the software installation, and the command-line interface to the pipeline is run as:

```
hawcpipe infiles.txt
```

The command-line interface will read in the specified input files, use their headers to determine the observation mode, and read the configuration file to determine the steps and parameters to run and the intermediate files to save. Output files are initially written to the input data location, then are moved to the current reduction directory. After reduction is complete, the script will generate an output manifest (*outfiles.txt*) containing the number of output files, and the relative paths to their locations.

6.5 Manual Mode Execution

This pipeline release does not include an integrated graphical user interface (GUI) for manual execution. A number of different interfaces to the pipeline algorithms, both command-line and GUI based, have been developed separately and are briefly described in Appendix A.

Manual mode execution for this pipeline entails running the *hawcpipe* script from the command-line with manually specified input files and/or configuration files. From this interface, any combination of pipeline steps and parameters may be specified. The script may be invoked directly on a file or set of files without creating an input manifest, as, for example:

```
hawcpipe 2016-12-16_HA_F360_082_POL_unk_HAWA_HWPA_RAW.fits
```

It may also start from any intermediate file previously generated by the pipeline as, for example:

```
hawcpipe *WCS*.fits
```

to reduce a set of *wcs* data products from the next defined step to the end of the reduction recipe. To override any part of the default configuration file, specify an override configuration file on the command line with the *-c* option, e.g.:

```
hawcpipe File1.fits -c manual_config.txt
```

It is also possible to redirect the output to a different directory with the *-d* option, to rename the output manifest (*-o*) or log file (*-l*), or to force the pipeline to use a different mode than the standard science reduction (*-m*). The latter may be useful for running additional diagnostic steps included in the pipeline package, but not generally run for science reductions.

6.6 Important Parameters

The following sections list some useful parameters for HAWC reductions. DRP parameters may be set directly as key/value pairs in pipeline configuration files; CRUSH parameters are generally added to the *options* parameter string in pipeline configuration files.

6.6.1 DRP parameters

Below are the most important parameters for Nod-Pol or Chop-Nod pipeline steps, as named in the configuration file, in the order they are typically run. This list is not exhaustive; see the HAWC+ DRP Developer's Manual or the code itself for more information.

- **checkhead**
  - *abort*: Set to False to allow the pipeline to continue despite incorrect header keywords. Default is True.
- **prepare**
  - *traceshift*: Set to a non-zero value to shift the data by this many samples relative to the metadata. Default is zero (no shift).
  - *multiplyminusone*: Set to True to multiply the raw data by -1, to correct for a negative Stokes I image. Default is False.
- **demod**
  - *phasefile*: Set to a FITS file for per-pixel phase shifts, or to a floating point number to apply the same phase shift to all pixels. Default is typically a file in *hawc/pipeline/data/phasefiles*.
  - *track_tol*: If non-negative, will use this number as the tracking tolerance in arcseconds. Samples with tracking deviation larger than this number will be rejected. Set to -1 to turn off. Default is 3.0 arcseconds.
  - *boxfilter*: Time constant for filter applied to data. 0 means no filter, -1 means use 1/CHPFREQ for time constant. Default is -1.

- **flat**
  - *flatfile*: Set to 'search' to detect best available file in *flatfolder*. Set to a FITS file path to override. Default is search.
  - *fitkeys*: Header keywords to match in searching for files in *flatfolder*. Default is "'SPECTEL1', 'SPECTEL2'".
  - *flatfolder*: Directory to search for matching flat files. Default is typically *hawc/pipeline/data/flats*.
- **split**
  - *rtarrays*: Set to 'RT' to use both R and T arrays, 'R' for R only, or 'T' for T only. Default is 'RT'.
- **combine**
  - *sigma*: Reject outliers more than this many sigma from the mean. Default is 3.0.
  - *covflag*: Set to True to calculate covariances. Default is False.
- **stokes**
  - *erri*: Method for inflating errors in I from standard deviation. Can be median, mean, or none. Default is median.
  - *widow*: Set to True to attempt to fix widow pixel flux from nearby pixels. Default is False.
- **wcs**
  - *offsibs_x*: Offset in pixels along X between SIBS_X and actual target position on array. Should be a comma-separated list of 5 numbers, one for each band; for example, '-0.9, 0.0, 1.1, 0.0, 1.1'. Default may vary over time.
  - *offsibs_y*: Offset in pixels along Y between SIBS_Y and actual target position on array, as for *offsibs_x*. Default may vary over time.
- **bgsubtract**
  - *bgslope*: Number of iterations to run with slope term. If zero, slope will not be fit (i.e. residual gains will not be corrected). Default is 0.
  - *bgoffset*: Number of iterations to run with offset term. If zero, offset will not be fit (i.e. residual background will not be removed). Default is 10.
- **ip**

- *qinst*: Fractional instrumental polarization in q. Should be a comma-separated list of 5 numbers, one for each band; for example, '-0.01191, 0.0, -0.01787, -0.00055, -0.01057'. Default may vary over time.
- *uinst*: Fractional instrumental polarization in u, as for *qinst*.
- *qtel*: Fractional telescope polarization in q. Should be a comma-separated list of 5 numbers, one for each band; for example, '0.0, 0.0, 0.0, 0.0, 0.0'. Default may vary over time.
- *utel*: Fractional telescope polarization in u, as for *qtel*.

- **rotate**
  - *gridangle*: Detector angle offset, in degrees. Should be a comma-separated list of 5 numbers, one for each band; for example, '-78.69, 0.0, -93.28, 48.42, 130.62'. Default may vary over time.

- **merge**
  - *cdelt*: Pixel size in arcseconds of output map, one number per band. Default is '2.55, 4.0, 4.0, 6.8, 9.1', to match detector pixel scale.
  - *fwhm*: FWHM in arcseconds of Gaussian smoothing kernel, by band. Make larger for more smoothing. Default is '2.55, 4.0, 4.0, 6.8, 9.1', for minimal smoothing.
  - *radius*: Integration radius for input pixels, by band. Set larger to consider more pixels when calculating output map. Default is '9.4,11.6,15.6,28.0,38.0'.
  - *covflag*: Set to True to use covariance image for weighting. Default is False.
  - *errflag*: Set to True to use error image for weighting. Default is True.
  - *widowstokesi*: Set to True to use widow pixels to compute the Stokes I map. Default is True.

- **calibrate**
  - *fac*: Multiplicative calibration factor in Jy/pixel/counts. Should be a comma-separated list of 5 numbers, one for each band; for example, '0.0454, 0.038, 0.0345, 0.0291, 0.0419'. Default may vary over time.

- **region**
  - *skip*: Set to a number i to plot vectors every ith pixel. Default is 1.
  - *debias*: Set to True to use debiased polarizations. Default is True.

- *mini*: Do not plot vectors from pixels with flux less than this fraction of peak flux. Default is 0.
- *minp*: Do not plot vectors with percent polarization less than this value. Default is 0.3%.
- *length*: Scale factor for polarization vectors, in pixels. Default is 10 (i.e. a 10% polarization vector is the length of one pixel).
- *sigma*: Do not plot vectors with p/σ_p less than this value. Default is 5.0.

6.6.2 CRUSH parameters

Below are some commonly used top-level parameters for CRUSH reductions of HAWC+ data with DRP. For more information, or an exhaustive list of parameters, see the CRUSH documentation online or in the *crush* distribution.

- **-bright / -faint / -deep**: a non-default brightness setting for the object (a default brightness is assumed if none of these are set). -deep is based on -faint but it also spatially filters maps to discard large-scale structures above a few beam-widths, to gain maximum sensitivity for very faint point sources.
- **-extended**: If an extended object ($\geq$ FoV) is observed, and the large scale structure is of interest. The trade-off in retaining more large-scale structures is higher noise.
- **-source.sign=X**: '+', '-' or '0' to bias for positive or negative sources, or no bias. The bias helps get rid of filter bowls surrounding bright features. It is recommended to leave it unchanged from the default ('+') unless you expect to see absorption features.
- **-ecliptic / -galactic / -supergalactic / -horizontal / -focalplane**: To produce maps in other than the default equatorial coordinates. Note that 'horizontal' actually produces maps in TARF (which for HAWC+ is the same as SIRF) disguised as horizontal maps (Azimuth / Elevation axis labels that correspond to tXEL / tEL).
- **-sourcesize=X**: Use together with **-extended** to specify a typical source diameter in arcsec.
- **-tau=X, -tau.pwv=X**: Specify an in-band zenith tau or water-vapor level for calculating extinction correction.
- **-scale=X**: Apply a calibration scaling factor ($X = F_{true}/F_{obs}$)
- **-unit=X**: Define output units. Options are Jy/pixel, Jy/beam, counts/beam, etc. Default for the DPS environment is currently Jy/pixel.

7 Data Quality Assessment

After the pipeline has been run on a set of input data, the output products should be checked to ensure that the data has been properly reduced. Data quality and quirks can vary widely in individual observations, but the following general guideline gives some strategies for approaching quality assessment for HAWC+ data.

For any mode:

- Check the instrument scientist's log for any data that is known to be of poor or questionable quality.
- Make sure that any diagnostic files are excluded from reductions. These typically have CALMODE=INT_CAL or SKY_DIP.
- Check the output to the log file (usually called *hawc[date/time].log*), written to the same directory as the output files. Look for messages marked ERROR or WARNING. The log will also list every parameter used in DRP steps, which may help disambiguate the parameters as actually-run for the pipeline.
- Check that the expected files were written to disk. There should be, at a minimum, a WCS file and a VEC file for Nod-Pol data, and a CRH file for Scan data.

For Nod-Pol mode:

- Display all WCS files together. Verify that no one file looks unreasonably noisy compared to the others, and that any visible sources appear in the same locations, according to the world coordinate system in each file's header. Particular WCS files may need to be excluded, and the last steps of the pipeline re-run.
- Check the WCS files for persistent bad pixels or detector features. If present, the flat field or bad pixel mask may need updating.
- Display the final VEC file. Verify that the mapping completed accurately, with no unexpected or unusual artifacts. The error or covariance flags may need modification, or the smoothing may need to be increased.
- Overlay the DS9 polarization vector file (**.reg*) on the VEC file. Check for unusually noisy vector maps (e.g. long vectors near the edges).
- For observations of flux standards, compare the total flux in the source, via aperture photometry, to a known model. Flux calibration should be within 20%; if it is not, the calibration factors may need to be adjusted, or some off-nominal data may need to be excluded from the reduction.

- For observations of polarimetric standards, verify that the total polarization (Q/I and U/I) is less than 0.3% in regions that should have zero total polarization. If it is not, the instrumental polarization parameters may need adjusting.

For Scan mode:

- Check the log for warnings about scans that may have been excluded from reduction or are of poor quality.
- Display the final CRH image. Check that no unusual artifacts appear (e.g. "worms" caused by bad pixels that were not properly excluded from the scans).
- Check that the map is not unusually large and does not include patches disconnected from the main image. These may be signs of poor tracking during the observation or missing metadata in the input FITS tables.
- For observations of flux standards, compare the total flux in the source, via aperture photometry, to a known model. Flux calibration should be within 20%; if it is not, the calibration factors or the opacity correction may need to be adjusted.

A Appendix: Alternate Pipeline Execution Modes

A.1 DRP Command-Line Reduction

The DRP pipeline can be run from the (UNIX/Windows) command line without the DPS wrapper as follows.

The pipeline can reduce multiple files at once and will use input file header keyword information to select the appropriate pipe mode and the pipe steps to run. Alternatively, the pipemode can be set as a command line argument. The pipeline is called as follows:

```
Usage:
python pipeline.py [-h] [-t] [--loglevel {DEBUG,INFO,WARN,ERROR,CRITICAL
    }]
                   [--logfile LOGFILE] [--pipemode PIPEMODE]
                        [config] [inputfiles [inputfiles ...]]

Positional arguments:
  config                 pipeline configuration file (default = pipeconf.
     txt)
  inputfiles             input files pathname

Optional arguments:
  -h, --help             show this help message and exit
  -t, --test             runs the selftest of the pipeline
  --loglevel {DEBUG,INFO,WARN,ERROR,CRITICAL}
                         log level (default = INFO)
  --logfile LOGFILE      logging file (default = none)
  --pipemode PIPEMODE    pipeline mode (default = none)
```

A full pipeline configuration file is required to run the pipeline.

It is also possible to run individual pipe steps directly from the command line. For example:

```
Usage:
python stepfile.py [-h] [-t] [--loglevel {DEBUG,INFO,WARN,ERROR,CRITICAL
    }]
                   [--logfile LOGFILE] [--config CONFIG]
                   [--Pipe Step specific arguments]
                   [inputfiles [inputfiles ...]]

Positional arguments:
  inputfiles                input files pathname

Optional arguments:
  -h, --help                show this help message and exit
  -t, --test               runs the selftest of this pipe step
  --loglevel={DEBUG,INFO,WARN,ERROR,CRITICAL}
```

```
                          requested log level (default = INFO)
  --logfile LOGFILE         log file pathname (default = none)
  --config=pipeconf.txt   pipeline configuration file pathname (default =
      none)
  --Pipe Step specific arguments
```

The Pipe Step specific arguments are equal to the list of parameters for the pipe step. They are the same parameters as are specified for that pipe step in the configuration file. The config file is optional, as the steps have default parameters.

The following list of commands illustrates the simplest way to fully reduce HAWC files:

```
python pipeline.py pipeconf.txt file1.raw.fits file2.raw.fits
```

This process only stores the final result, unless save steps are listed under stepslist in the configuration file.

The following commands illustrate how to reduce the data using the pipe steps directly:

```
python stepprepare.py file1.raw.fits file2.raw.fits
python stepdemod.py file1.pre.fits file2.pre.fits
python stepflat.py file1.dmd.fits file2.dmd.fits
...
```

Each program saves the files that are used as input for the next step. Each of these single step programs can have additional command-line arguments as described above.

A.2 DRP Interactive Python Reduction

You can use the Python interpreter to reduce a list of files. To do so, you need to create a pipe object, assign a config file, and run it with the file list. The necessary commands are:

```
# Import the pipeline object
from drp.pipeline import PipeLine
# Create the pipe object and set configuration
pipe = PipeLine(config = 'path/file/name/of/pipeconfig.txt')
# Run the pipeline
result = pipe(['data1.raw.fits', 'data2.raw.fits'])
# Save the result
result.save('output_filename.fits')
```

To see log messages from the pipeline you might want to set up logging:

```
# Import the logging library
import logging
# Configure logging to print messages of level info and higher
logging.basicConfig(level = logging.INFO)
```

To run individual pipeline steps from the Python interpreter, you need to load the data into a datafits object, then run the pipe step on it, then save the result. Assuming you have set the logging and the path the code would look like this:

```
# Import objects
from drp.datafits import DataFits
from drp.stepmystep import StepMyStep
# Make a datafits object and load the data into it
inputdata = DataFits(config = '/path/to/pipeconfig.txt')
inputdata.load('/path/to/data/file.fits')
# Make a pipe step object and run it on the data
mystep = StepMyStep()
outputdata = mystep(inputdata)
# Store the result
outputdata.save('/path/to/output/data/file.fits')
```

A.3 Web Data View

The HAWC Web Viewer provides online access to reduced HAWC data products. This viewer will be available to investigators on the SOFIA airplane and on the ground. See Figure 6 for a screenshot illustrating the parts of Web Viewer.

The Data Viewer is the primary interface for the SI team to view and analyze HAWC data. The screenshot above shows the elements of the interface.

Function Navigation: This panel provides access to the basic Web View functions:

- Date / AOR List: Displays a list of all current and past observations
- Data Viewer: Links to the data viewer.
- Pipeline Log: Displays the most recent messages in the pipeline log.
- Help / Manual: Links to the current version of the HAWC data reduction manual.

To open a new tab, just right click (or Alt click - Win, or Command click - Mac) on any of these links.

Data Selection: These pull-down menus allow the user to select the data to view.

- Flight: Select a flight or test date.
- AOR: Select an AOR (observation) during the selected flight.
- File: Select a raw data file from the selected AOR.
- Pipe Step: Select a data reduction step to view.

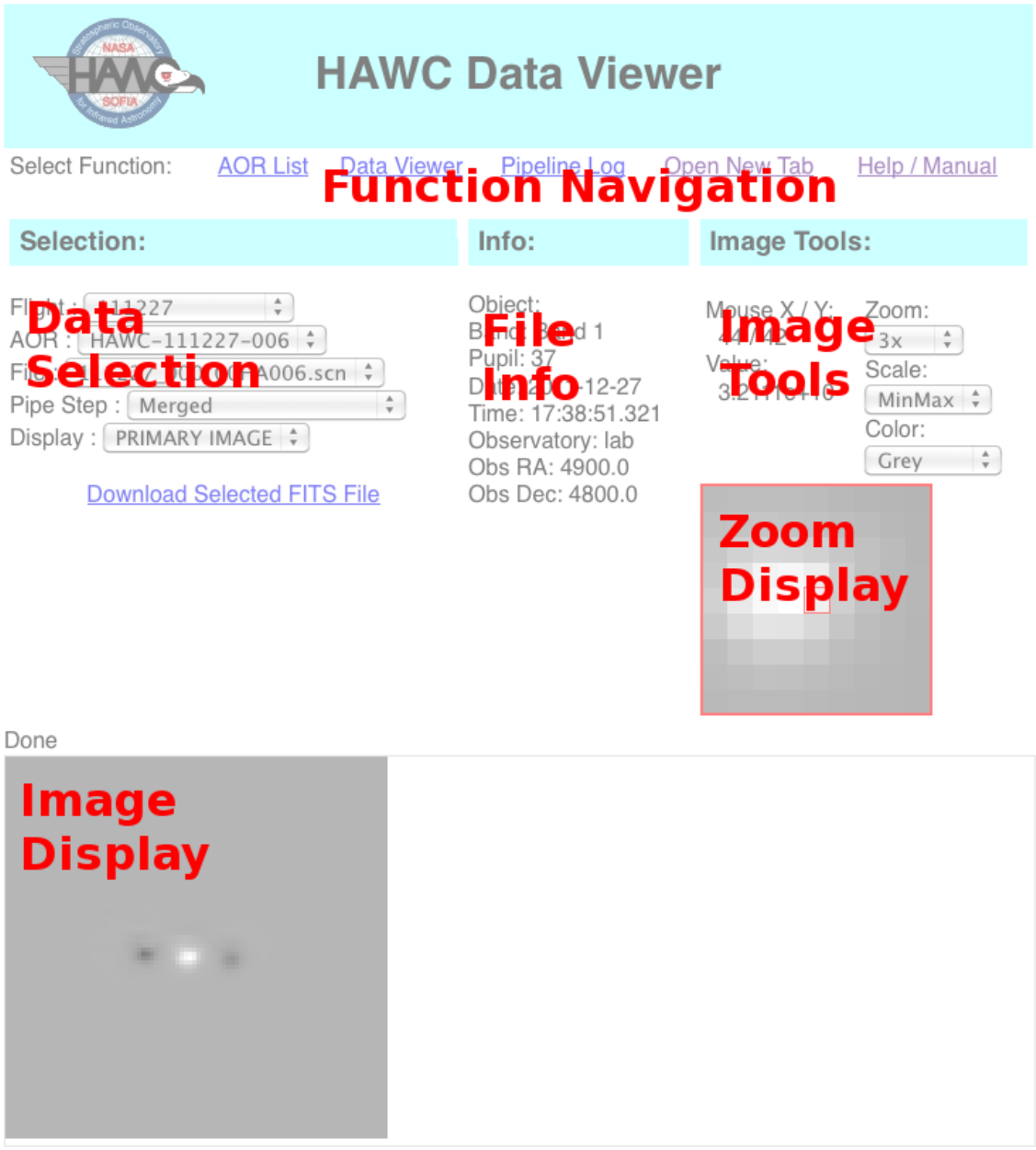

Figure 6: Web Data View Screenshot

The selections above identify an individual FITS file to view. The Download File and File Info tools both refer to that FITS file.

- Display: Which HDU of the selected FITS file to look at. Header and tabular information can also be selected here.
- Image Frame: If the viewed data contains an image cube, layers of that cube can be selected here.

File Info: This field displays information from the FITS header of the selected file.

Image Tools: This dialog provides information about the image display and allows the user to change display zoom, scale, and color. These options are only available if image data is being displayed. The Zoom Display shows a zoomed-in image of the pixels around the mouse cursor, mouse coordinates and cursor value are shown above.

Image Display: The image shows the image data. If the current image does not fit the screen, vertical and horizontal scroll bars appear.

Problems and Fixes: If there are problems with the data viewer, reload the page or reset your browser. Other errors can occur. For example, the data viewer may return an error message from Apache. Usually the cause for that is an empty Flight/AOR folder or a corrupted FITS file. Open a different date or a different AOR. Alternatively, remove offending files and folders on the web server. When the automatic data reduction is running it is possible that certain Flight/AOR folders contain incompletely reduced observations; in this case, wait until the reduction is complete and reload the page.

A.4 In-Flight Autoreduce

The automatic data reduction service reduces files independently during a SOFIA flight. Source and target folder as well as other parameters are set in the autoreduce configuration file (example: *auto_config_sky.txt*). The autoreduce also needs a regular pipeline configuration file (example: *pipeconf_sky.txt*). Several instances of the autoreduce can be run simultaneously, as long as they are not reducing the same files. **Important**: The output data folder must already exist before starting the *autoreduce.py* script.

The autoreduction code folder and configuration are in the pipeline folder under *hawcdrp/autoreduce*. Setup:

- Edit the auto_config_sky.txt file for setting the inputpath and outputpath. Make sure these folders exist.
- The autoreduce uses the pipeline config file such as pipeconf_sky.txt.
- You can also set file selection criteria such as namein and namend.

To start the program, open a terminal and go through the following steps:

```
cd /path/with/hawcdrp/autoreduce/src
python autoreduce.py path/to/auto_config_here.txt
```

Then, the pipeline should be running and you should see new log messages in the log file (usually under *hawcdrp/autoreduce/logs*).

To shutdown the pipeline, type *exit* in the pipe terminal. If that doesn't work, look up the UNIX process number by typing *ps -A*, then kill the pipeline process by typing kill NNNNNN.

A.5 CRUSH Command Line Reduction

CRUSH is a pipeline within the pipeline responsible for reducing HAWC+ scan-mode imaging data. Users are encouraged to run CRUSH from within the HAWC+ DRP, but may also choose to run it as a separate tool. The native CRUSH interface may give more flexible access to the full range of CRUSH reduction parameters, but may not produce standardized SOFIA headers and output file names. This section discusses the installation and basic use of CRUSH as a standalone tool.

A.5.1 Downloading and Installing CRUSH

Running CRUSH requires Java v1.7.0 or later.

CRUSH distribution packages (tarball, zip, and binary packages for RPM-based and Debian based Linux distros) are available from www.submm.caltech.edu/~sharc/crush/download.html.

The 'binary' .rpm and .deb packages allow one-click, system-wide installation on Linux platforms. The compressed archives (tarball and ZIP) allow both system-wide and unprivileged (user-directory) installations (Unix, Mac OS X, and Windows), and include the full source code as well.

To install from a tarball (POSIX/UNIX, incl. Mac OS X), simply unpack it in the desired location:

```
tar xzf crush-2.xx-x.tar.gz
```

and verify that it works:

```
cd crush
./crush
```

The Linux binary packages (.rpm and .deb) install system-wide, by default. All crush executables, and the related man pages, are available to all users, from any location on the machine.

To create system-wide access to the crush executables when installing from a tarball, you may wish to run install.sh (as root or with 'sudo') after unpacking crush. It will link the executables to '/usr/bin', and install the man pages. For example:

```
cd crush
sudo bash install.sh
```

You can check the success of the above optional step by typing:

```
man crush
```

If all is in order, you should see a basic description of the crush command-line syntax and options.

A.5.2 Optional Startup Environment and Java Configuration

CRUSH ships with a default Java configuration. On the most common UNIX platforms (Linux, Mac OS X, BSD, and Solaris), it will automatically attempt to set an optimal configuration. On other platforms, CRUSH will default to a fail-safe startup configuration (default java, 32-bit mode and 1GB of RAM use). To override these defaults on Windows, edit 'wrapper.bat' directly (and note, that you will have to repeat this step every time you reinstall or update CRUSH on Windows).

The preferred method for overriding defaults on POSIX systems (e.g. UNIX and Mac OS X), is by placing your settings in arbitrary files under /etc/crush2/startup or ∼/.crush2/startup. Any settings in the user's home under ∼/.crush2/startup will override the system-wide values in /etc/crush2/startup. If multiple config files exist in the same location, these will be parsed in non-specific order. These configurations are independent of the CRUSH installation, and will survive reinstall and updates to CRUSH. E.g., placing the following lines in ∼/.crush2/startup/java.conf overrides all available runtime settings:

```
JAVA="/usr/java/latest/bin/java"
DATAMODEL="64"
USEMB="4000"
JVM="server"
EXTRAOPTS="-Djava.awt.headless=true"
```

Upon startup CRUSH will find and apply these settings, so it will use '/usr/java/latest/bin/java' to run CRUSH, in 64-bit mode, with 4GB of RAM, using the HotSpot 'server' VM, and in headless mode (without display, mouse or keyboard).

Below is a guide to the variables that you can override to set your own Java runtime configuration:

- JAVA: Set to the location of the Java executable you want to use. E.g. "java" to use the default Java, or "/usr/java/latest/bin/java" to use the latest from Oracle or OpenJDK.
- DATAMODEL: Set to "32" or "64", to select 32-bit or 64-bit mode. To use 64-bit mode you will need both a 64-bit OS and a 64-bit JRE (Java Runtime Environment) installation.
- USEMB: Set to the maximum amount of RAM (in MB) available to CRUSH. E.g. "4000" for 4GB. Note, that when DATAMODEL is "32", you this value must be somewhere below 2000. Thus, "1900" is a good practical maximum value to use in 32-bit mode. Due to the volume of full-rate HAWC+ data (> 500MB/min), you will need to configure Java with sufficient RAM to accommodate entire scans.
- JVM: Usually set to "server" for Oracle or OpenJDK. If using IBM's Java, set it to "" (empty string). On ARM platforms, you probably get better performance using "jamvm" or "avian". To see what VM options are available, run 'java -help'. The VM options are listed near the top of the resulting help screen.
- EXTRAOPTS: Any other non-standard options you may want to pass to the Java VM should go here.

You can also specify environment variables, and add shell commands (bash), since these configuration files are in fact sourced as bash scripts before launching Java / CRUSH. For example you can add:

```
CRUSH_NO_UPDATE_CHECK="1"
CRUSH_NO_VM_CHECK="1"
echo "Will try to parse my own configuration now... "
if [ -f ~/mycrushconfig.sh ] ; then
   echo -n "OK"
   source ~/mycrushconfig.sh
else
   echo -n "Not found"
fi
```

The above will disable update checking (not recommended!) and VM checking (also not recommended!) and will source the contents of '∼/mycrushconfig.sh' if and when such a file exists.

A.5.3 Running CRUSH

The basic syntax to run CRUSH is:

```
<path-to-crush/>crush hawc+ [options] <scanlist> ...
```

For quick reference, simple UNIX-style man pages for all tools of the CRUSH suite (including the reduction pipeline 'crush' itself) are also available online.

Locating scan data can be done in one of two ways. The default method of locating files is by file name, which may specify either an absolute path, e.g.:

```
crush hawc+ /data/hawc+/F0004_HC_IMA_0_HAWC_HWPC_RAW_105.fits
```

or, it can be filename/path relative to 'datapath':

```
crush hawc+ F0004_HC_IMA_0_HAWC_HWPC_RAW_105.fits
```

The two are equivalent assuming that 'datapath' is set to '/data/hawc+' in the second case, e.g. in the user configuration file '~/.crush/hawc+/default.cfg', or on the command-line.

Often, the simpler way of locating input files is by a combination of flight and scan numbers. This is often shorter, and allows to specify multiple scans and ranges with more ease. Scan lookup by flight and scan number requires you to set 'datapath' to point to the data directory. E.g., by placing the line in the user configuration for HAWC+ ('~/.crush2/hawc+/default.cfg'):

```
datapath /data/hawc+
```

Now, you may simply reduce scan 105 from flight 354 as:

```
crush hawc+ -flight=354 105
```

You can also reduce multiple scans from multiple flights together. E.g.:

```
crush hawc+ -flight=354 104-105 129 -flight=356 13 16 33-35
```

The above will co-reduce 3 scans (104, 105, 129) from flight #354 with 5 scans (13, 16, 33, 34, 35) from flight #356.

A.5.4 Command-Line Options

Some common HAWC+ command-line options are listed in Section 6.6, and a complete listing is available in the CRUSH GLOSSARY file. The below are some parameters useful specifically for running CRUSH directly without the DRP.

Reduction options precede the entire list of scans. Some common reduction options to be used by HAWC+ are:

- **-datapath=<path>**: The location of the raw scan FITS data files. E.g. -outpath=/data/hawc+

- **-outpath=<path>** : The output root folder for images and other output files. E.g. -outpath=~/mydata/hawc+/crush/images
- **-name=<outputname>** : The name of the output FITS image, relative to the output path. E.g. -name=myscan.fits

Scan-specific options apply to all scans listed after the option on the command-line, but not to the scans listed before. For example:

- **-pointing=dx,dy**: Adjust the pointing by dx,dy arcsecs (in AZ / EL, i.e. tXEL / tEL).
- **-tau=X, tau.pwv=X**: Specify an inband zenith tau or water-vapor level for calculating extinction correction for each scan
- **-scale=X**: Apply a calibration scaling factor

A.5.5 CRUSH News, Feedback, and Bug Reports

If you run CRUSH as a standalone pipeline, you may wish to keep informed of its latest developments. You can get email notifications of new releases by subscribing to updates at the CRUSH SourceForge project page. Simply enter your e-mail address in the box labeled KEEP ME UPDATED, press the Follow button, and you will be notified of new releases, and updates in the future. You can stop update notifications at the same place at any time. You can file reports of bugs you might encounter in CRUSH on the SourceForge project page, under the Bugs tab. Once there, you may also subscribe to bug notification e-mails.

A.6 IDL Redux Interface

The SOFIA DPS team maintains a general-purpose interface to their pipelines, called Redux, which has been adapted to run HAWC reductions.

A.6.1 Installation

Running Redux requires IDL 8.5 or later, as well as the latest version of the IDL Astronomy User's Library, the Coyote graphics library, package, the FSpextool package, and the Redux code. FSpextool and Redux are under SOFIA DPS revision control and can be obtained directly from git repositories there. The IDL Astronomy User's Library (astrolib) is publicly available, and can be downloaded from the website at `http://idlastro.gsfc.nasa.gov/homepage.html`. The Coyote graphics library (coyote) is also publicly available and

can be downloaded from http://www.idlcoyote.com/documents/programs.php. When these packages have been installed, their locations should be added to the IDL_PATH environment variable, so that their procedures are accessible to Redux.

SOFIA may distribute the Redux and FSpextool codes as gzipped tar files. If so, unpack them, as, for example:

```
tar xvzf redux.tar.gz
tar xvzf fspextool.tar.gz
```

This will create directories called 'redux' and 'fspextool', which will contain a number of subdirectories. Each of these package directories should be added to the IDL_PATH as well.

A.6.2 Running Redux

To run Redux, start IDL, then from the prompt call the command 'redux', with no arguments. This will launch the Redux GUI.

To start an interactive reduction, select a set of HAWC files, using the file menu (*File* → *Open New Reduction*). All files selected will be reduced together as a single reduction set. Redux will decide the appropriate reduction steps from the input files, and load them into the GUI, as in the screenshot below.

Each reduction step has a number of parameters that can be edited before running the step. To examine or edit these parameters, click the *Edit Param* button next to the step name to bring up the parameter editor for that step. Within the parameter editor, all values may be edited; clicking *Done* will save the edited values and close the window. Clicking *Reset* will restore any edited values to their defaults; clicking *Cancel* will discard all changes to the parameters and close the editor window. The current set of parameters can be displayed, saved to a file, or reset all at once using the *Parameters* menu. A previously saved set of parameters can also be restored for use with the current reduction (*Parameters* → *Load Parameters*).

After all parameters for a step have been examined and set to the user's satisfaction, a processing step can be run on all loaded files either by clicking *Step*, or the *Run* button next to the step name. Each processing step must be run in order, but if a processing step is selected in the *Step to:* widget, then clicking *Step* will treat all steps up through the selected step as a single step and run them all at once. When a step has been completed, its buttons will be grayed out and inaccessible. It is possible to undo one previous step by clicking *Undo*. All remaining steps can be run at once by clicking *Reduce*. After each step, the results of the processing will be displayed in the display window. Clicking *Reset* will restore the reduction to the initial state, without resetting parameter values.

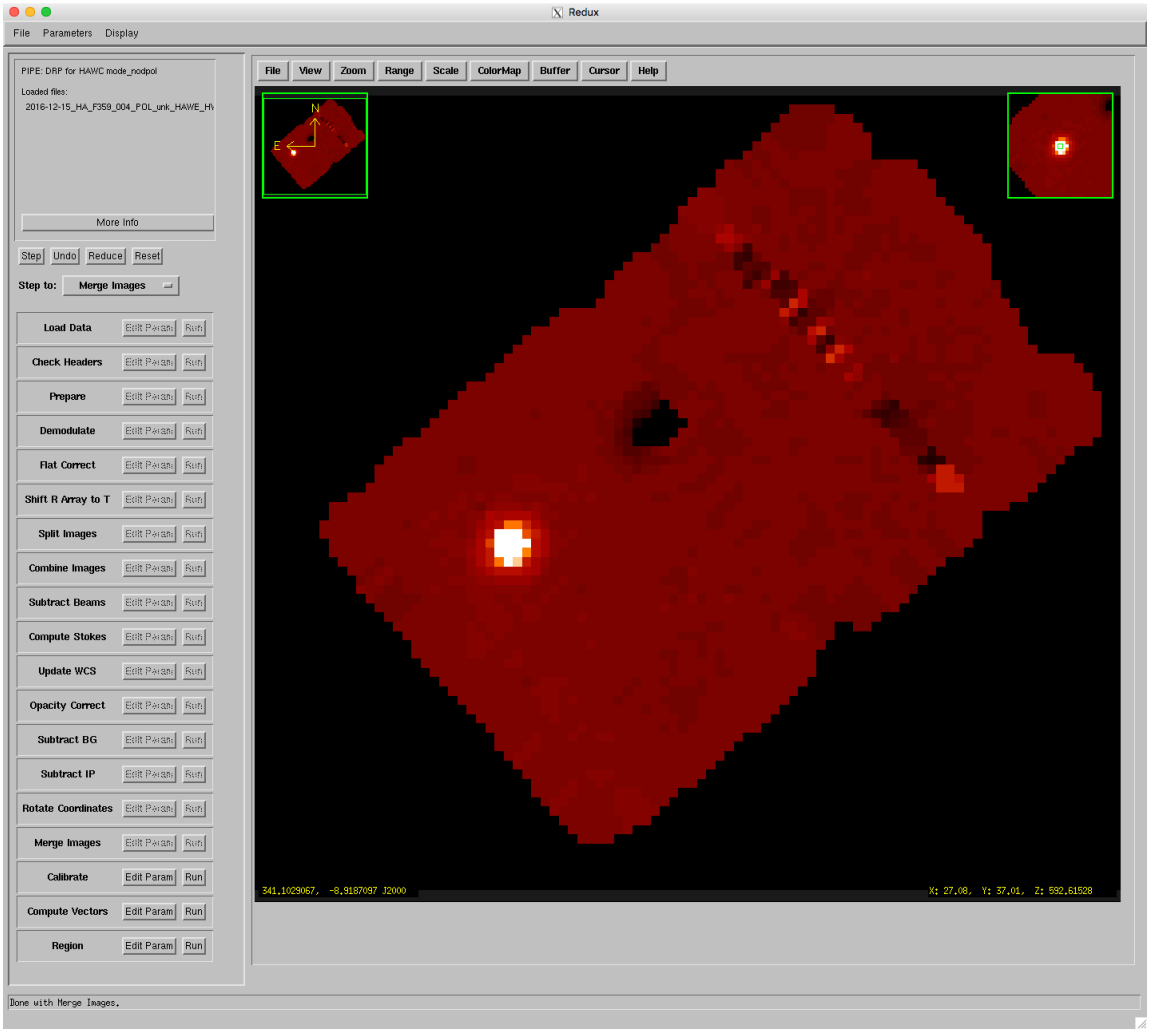

Figure 7: Redux for HAWC Screenshot

Files can be added to the reduction step (*File* → *Add Files*) or removed from the reduction set (*File* → *Remove Files*), but either action will reset the reduction for all loaded files. Selecting *Display* → *Display File Information*, or the *More Info* button, will pull up a table of information about the currently loaded files. The table rows displayed can be filtered by entering a search string into the *Filter* text box.

Redux displays images using ximgtool, a full-featured display tool distributed with FSpextool. For more information, see the ximgtool help file, available from Redux via the *Help* button just above the display.

Ximgtool has five buffers available for simultaneous display of images. If there are more than five images loaded into Redux, they can be viewed by selecting *Display* → *Quick Look* from the Redux menu. This will cycle through each data file in its current processing state, allowing interaction and analysis with each image in turn. To move between images, click the *Next File* or *Previous File* buttons, below the image. Click *Cancel* to quit the quick look display.

Note that Redux for HAWC, while useful for interactively exploring the data reduction steps, has some overhead compared to the native DRP interface, due to calling the Python functions as separate, external steps from the IDL structure.

B Appendix: Sample Configuration Files

B.1 Full DRP Configuration File

Below is a copy of the full configuration file used by the pipeline in the DPS environment (*pipeconf_dcs.txt*). The format is defined by the configobj Python module.

```
# HAWC Pipeline Configuration File - DCS Version
# v1.0.0

# General Section: configuration of the pipeline
[general]
    # list of packages to look for pipe step modules (order matters)
    steppacks = hawc, hawcV0, tesdetector, detbolo, sharp, drp, labdiag
    # list of steps for default / unknown instrument mode
    stepslist = StepLoadHAWC, StepDemod

# Data Section: information on data objects and file names
[data]
    # Regexp for part of the filename before the file step identifier
    filenamebegin = '\A((\d.+)|(F\d{3,4}_HA_[A-Za-z]+_[A-Za-z0-9]+_[A-Za-
        z0-9]+))_'
    filenameend = '_((\d+-)?\d+)\.fits(\.gz)?\Z' # HAWC+
```

```
    filenum = '(?:\A.*F\d{3,4}_((?:\d+-)?\d+)_.*\.fits(?:\.gz)?\Z)|(?:\AF
        \d{3,4}_HA.*_((?:\d+-)?\d+)\.fits(?:\.gz)?\Z)'
    dataobjects = DataFits, DataText #, DataCsv

# Pipeline Section: Configuration of the pipeline
[pipeline]
    # Number of final results to save
    finalsaveN = 5

### Pipelines Section: configuration for individual pipeline modes

# ChopNod Mode Configuration
[mode_nod]
    datakeys = 'INSTMODE = C2N (NMC)|INSTCFG = TOTAL_INTENSITY'
    # list of steps
    stepslist = load, StepCheckhead, StepFluxjump, StepPrepare, StepDemod
        , StepFlat, StepShift, save, StepSplit, StepCombine,
        StepNodPolSub, StepStokes, StepWcs, save, StepOpacityModel,
        StepBgSubtract, StepRotate, StepMerge, save, StepCalibrate, save

# Nod-Pol Mode Configuration
[mode_nodpol]
    # List of keyword=values required in file header to select this
        pipeline mode
    #   Format is: Keyword=Value|Keyword=Value|Keyword=Value
    datakeys = 'INSTMODE = C2N (NMC)|INSTCFG = POLARIZATION'
    # list of steps
    stepslist = load, StepCheckhead, StepFluxjump, StepPrepare, StepDemod
        , StepFlat, StepShift, save, StepSplit, StepCombine,
        StepNodPolSub, StepStokes, StepWcs, save, StepOpacityModel,
        StepBgSubtract, StepIP, StepRotate, StepMerge, StepCalibrate,
        save, StepPolVec, save, StepRegion

# Scan Mode Configuration
[mode_scan]
    datakeys = 'INSTMODE=OTFMAP|INSTCFG=TOTAL_INTENSITY'
    stepslist = StepCheckhead, StepCrush, save

### Pipe Step Section
#   First the definitions for the parent steps, then all HAWC
#   in alphabetical order.

# R and T alignment step
[align]
   angle = 0.0   # rotation angle of R relative to T, in degrees
       counterclockwise
   mag = 1.0, 1.0   # Magnification of R relative to T, in the x,y pixel
       direction
   disp = 0.0, 0.0   # Pixel displacement of R relative to T, in the x,y
       directions
```

```
   xpoint = 0.0, 0.0, 0.0, 0.0   # Pointing offset in x pixels of R
      relative to T, for each HWP angle
   ypoint = 0.0, 0.0, 0.0, 0.0   # Pointing offset in y pixels of R
      relative to T, for each HWP angle
   thresh = 0.5   # Mask values >= this threshold value are bad (set to
      1)

# Badpix step
[badpix]
   dead_thresh = 10.0          # Threshold variable for sd for dead pixel
   ramp_thresh = 2000000.0     # Threshold variable for sd for ramping
      pixel
   auxfile = ''                # Filename for aux file

# background subtraction step
[bgsubtract]
   cdelt = 2.55, 4.0, 4.0, 6.8, 9.1   # Pixel size in arcseconds of
      output map
   proj = TAN   # Projection of output map
   bgslope = 0   # Number of iterations of background subtraction with
      slope term
   bgoffset = 10   # Number of iterations of background subtract with
      offset (intercept) term
   fwhm = 2.5, 3.0, 5.0, 7.5, 10.0   # FWHM of gaussian smoothing kernel,
       in arcseconds
   radius = 5.0, 6.0, 10.0, 15.0, 20.0   # Integration radius for
      smoothing, in arcseconds
   chauvenet = True   # Use Chauvenet's criterion in background
      subtraction?
   fitflag = False   # Use errors in intensity when fitting?
   covflag = False   # Use covariances when performing gaussian smoothing
      ?
   errflag = True   # Use uncertainties when computing averages?
   widowstokesi = True   # Use widow pixels (flagged 1 or 2) when
      smoothing
   polthetamaps = False       # Compute preliminary polarization degree
      and angle maps

# Calibrate fluxes from data units to Jy/pixel
[calibrate]
   fac = 0.0454, 0.038, 0.0345, 0.0291, 0.0419  # Multiplicative factor
      to convert fluxes to Jy (for each band)

# Combine R-T and R+T data
[combine]
   sigma = 3.0   # Reject outliers more than this many sigma from the
      mean
   covflag = False   # Set to False to skip computation of covariances (
      faster)
```

```
# create maps of median covariance by row and by entire map and flag bad
    pixels
[covar]
   mapcut = 1.0   # cutoff for median map covariance
   rowcut = 1.0   # cutoff for median row covariance

# Check the primary FITS header for required keywords
[checkhead]
    abort = True
    headerdef = $DPS_HAWCPIPE/pipeline/config/header_req_config.txt

# Run the crush scan data reduction
[crush]
    crushpath = $DPS_HAWCPIPE/crush
    options = 'hawc+ -unit=Jy/pixel -blacklist=smooth -forget=write.png'
    verbose = True
    opacitymodel = False

# Datagroup step, used for focus script to bundle files to merge
[datagroup]
    # data reduction step to use
    redstepname = StepMerge
    # List of header keywords to decide data group membership:
    # (| separated list)
    groupkeys = 'FOCUS_ST'
    # List of group key formats to force string comparison
    # (unused if equal "", | separated list)
    groupkfmt = '%.1f'

# Demodulate the chopped data
[demod]
   chop_tol   = 0.2    # chopper tolerance in arcseconds (not used for
      sine)
   nod_tol    = 5.0    # nod tolerance in arcseconds
   hwp_tol    = 2.0    # hwp angle tolerance in degrees
   az_tol     = 5000.0 # Azimuth error tolerance in arcseconds
   el_tol     = 5000.0 # Elevation error tolerance in arcseconds
   samp_tol   = 50.0   # Sample tolerance as a percentage.  High and low
      chop states must have a difference in number of samples less than
      this value.(not used for sine)
   track_tol  = 'beam'  # Track error tolerance in arcseconds (AOIs 3 and
       4) - set negative to deactivate
   mode       = sine   # Demodulation mode.  Options are sqr and sine
   chopphase  = True   # Flag requiring chop phase correction (not used
      for sqr)
   checkhwp   = True   # Set to FALSE to avoid checking the expected
      number of HWP angles
   phasefile  = $DPS_HAWCPIPE/pipeline/data/phasefiles/
      F0005_HAWC_HWPC_138-180_phase_161011.fits # Phase file information
       file (0.0 = no phase is default)
```

```
   l0method   = RE     # Method to normalize data: REal, IMag and
      ABSolute (default = RE)
   highfilter = 0.0    # Time constant for RC hipass filter (default 0.0=
      no filter)
   boxfilter  = -1     # Time constant for box hipass filter (default
      0.0=no filter, -1 for 1/CHPFREQ)
   filtersave = False  # Flag to save filtered data (default = False)
   chopavg    = False  # Flag to save chop averaged raw data (default =
      False)

# Compute pointing drifts between HWP angles
[drift]
    ofac = 1  # Oversampling factor (integer)

# Fill in widow pixels
[fill]
   model = mean(sigma=3)  # Model specification

# Flat step configuration
[flat]
    flatfile   = search # Filename for flat file or "search" for
       searching file in flatfolder (default = search)
    fitkeys    = 'SPECTEL1', 'SPECTEL2'  # List of keys that need to
       match flat and data file (default = SPECTEL1 SPECTEL2) - only
       used if flatfile=search
    flatfolder = $DPS_HAWCPIPE/pipeline/data/flats  # Folder for
       searching flat files (default = . ) - only used if flatfile=
       search
    l0method = NO     # method to normalize data (standard is NO, options
        are NO, RE, IM and ABS)
    datalist = R array, T array      # list of input file datasets to
       flatten - Expects None or a list of image HDU or table column
       names
    addfromfile = R BAD PIXEL MASK, T BAD PIXEL MASK   # additional data
       from the flat file to add to the reduced data

[flatclean]
   computeflatbad = False # Use algorithm to compute a flat image per
      file. If False, will use flat and bad pixel masks from input
   chopcrop = 0,0 # Number of chops to crop in the beginning and end of
      the demodulated timestream
   histogram = 100,-1000,-1000,10000,10000,4 # Histogram parameters for
      robust median: nbins, lowlimR, lowlimT, highlimR, highlimT, factor
   badpix_highstd = 10 # Lower limit of ynormstd to look for very high
      std of the demodulated signal
   badpix_lowsig = 0.4 # Upper limit of ynorm to identify and eliminate
      pixels with very low signal
   badpix_highsig = 7.0 # Lower limit of ynorm to identify pixels with
      very high normalized signals
```

```
# Flux Jump step configuration
[fluxjump]
    # Filepathname specifying the jump gap map, alternatively a number
       for the
    #   gap to be used for all pixels (default = '4600')
    jumpmap = $DPS_HAWCPIPE/pipeline/data/fluxjumps/FluxjumpFS14Columns.
       fits

# Correction for instrumental polarization
[ip]
   qinst =  -0.01191, 0.0, -0.01787, -0.00055, -0.01057  # Fractional
      instrumental polarization in q
   uinst =  -0.00273, 0.0,  0.00758,  0.02179, -0.01201  # Fractional
      instrumental polarization in u
   qtel  = 0.0, 0.0, 0.0, 0.0, 0.0  # Fractional telescope polarization
      in q (each waveband)
   utel  = 0.0, 0.0, 0.0, 0.0, 0.0  # Fractional telescope polarization
      in u (each waveband)

# Merge step configuration
[merge]
   cdelt = 2.55, 4.0, 4.0, 6.8, 9.1   # Pixel size in arcseconds of
      output map
   proj = TAN   # Projection of output map
   fwhm = 2.55,4.0,4.0,6.8,9.1 # FWHM of gaussian smoothing kernel (
      arcsec)
   radius = 9.4,11.6,15.6,28.0,38.0  # Integration radius for smoothing (
      arcsec)
   covflag = False   # Use covariances when performing gaussian smoothing
      ?
   errflag = True   # Use uncertainties when computing averages?
   widowstokesi = True   # Use widow pixels (flagged 1 or 2) to compute
      Stokes I map
   conserveflux = True    # Apply flux conservation factor

# Subtract L and R nods with HWP configuration
[nodpolsub]

# Correction for atmospheric opacity step configuration
[opacity]

# Correction for model atmospheric opacity step configuration
[opacitymodel]
   respfolder = $DPS_HAWCPIPE/pipeline/data/response
   fittype = alt
   za_ref = 45.0
   alt_ref = 41000
   pwv_ref = 7.3

# Polarization vector step configuration
```

```
[polvec]
   eff = 1.0,1.0,1.0,1.0,1.0   # telescope polarization efficiency
   chi2 = 1.0   # inflate errors in q,u by sqrt of this factor
   save = False   # Write out text file of polarization data

# Prepare file for demodulation
[prepare]
    detcounts = 'SQ1Feedback' # Name of the column containing the
       detector flux values R/T arrays
    hwpcounts = 'hwpCounts'   # Name of the input fits column containing
       the HWP counts (only used if column "HWP Angle" is not present)
    hwpconv = 0.25   # Value to convert hwpcounts to HWP Angles (only
       used if column "HWP Angle" is not present)
    labmode = False  # If TRUE (processing lab data), will fill in with
       zeros a few columns and keywords that are important for the DRP
    replacenod = True  # If TRUE will replace Nod Offset by calculation
       based on RA/DEC. If False use original column (has problems)
    chpoffsofiaRS = True  # If TRUE will calculate Chop Offset based on
       SofiaChopR/S. If False the user should use colrename to specify
       which column to use
    colrename = 'AZ_Error->Azimuth Error|EL_Error->Elevation Error|AZ->
       Azimuth|EL->Elevation|SIBS_VPA->Array VPA|NOD_OFF->Nod Offset' #|
       Right ascension->RA|DEC->Dec'
    coldelete="hwpA","hwpB","FluxJumps"
    traceshift=0  # Number of samples to shift the data (default is 0 i.e
       . no shift)
    pixscalist=2.57,3.97,3.97,6.93,9.33 # List for PIXSCAL values for
       each band
    multiplyminusone = False       # Multiply raw data by (-1) to correct
        for negative Stokes I image (default is False)

# Extract ds9 region file of polarization vectors
[region]
   skip = 1   # Only plot every ith pixel
   scale = True   # Set to False to make all vectors the same length
   rotate = False   # Use rotated (B-Field) vectors
   debias = True   # Use debiased polarizations
   mini = 0.0   # Do not plot vectors with flux < this fraction of peak
      flux
   minp = 0.3   # Require percentage polarizations to be >= this value
   offset = 0, 0   # Offset in pixels in x,y (controls which pixels are
      extracted)
   length = 10.0   # Scale factor for length of polarization vectors in
      pixels
   sigma = 5.0   # p/sigmap must be >= this value
   format = ds9    # Output file format.  (ds9 or txt)

# Rotate Q and U from detector to sky frame step configuration
[rotate]
```

```
  gridangle = -78.69, 0.0, -93.28, 48.42, 130.62   #Angle of the grid in
      degrees (for each waveband)
  hwpzero_tol = 3.0   #Tolerance in the difference between commanded and
      actual initial HWP angles
  hwpzero_option = 'commanded'   #Option to use between "commanded" or "
      actual" in case the difference between the initial HWP angles is >
      hwpzero_tol

# Account for R/T misalignment and apply integer displacements (shifts)
[shift]
   angle1   = 0.0       # rotation angle of R1 relative to T1, in degrees
       counterclockwise
   angle2   = 0.0       # rotation angle of R2 relative to T2, in degrees
       counterclockwise
   mag      = 1.0, 1.0  # Magnification of R relative to T, in the x,y
      pixel direction
   disp1    = 0.0, 0.0  # Pixel displacement of R1 relative to T1, in the
       x,y directions
   disp2    = 0.0, 0.0  # Pixel displacement of R2 relative to T2, in the
       x,y directions
   gapx     = 4.0       # displacement in x pixels between T1 and T2
   gapy     = 0.0       # displacement in y pixels between T1 and T2
   gapangle = 0.0       # Rotation angle in degrees CCW between T1 and T2

# Split data by HWP angle and nod position step configuration
[split]
   nod_tol = 100.0  # Nod tolerance, as the percent difference allowed in
       number of chop cycles between 1st and 2nd left, and between left
      and right
   rtarrays = RT    # Use both R and T arrays (RT), or only R (R) or only
       T (T)

# Compute Stokes I, Q, U step configuration
[stokes]
   hwp_tol = 5.0   # HWP angles for Stokes parameters must differ by no
      more than 45+-hwp_tol degrees
   erri = median   # How to inflate errors in I.  Can be median, mean, or
       none
   widow = False     #  Set to true to fix fluxes of widow pixels
   method = 'pairs' # Method to obtain Stokes Q and U. Can be pairs or
      fit
   polthetamaps = False # Compute preliminary polarization degree and
      angle maps

# Update Parallactic angle and crval1 and crval2 for a single file
[wcs]
   add180vpa = True   # Add 180 degrees to the SIBS_VPA
   offsibs_x = -0.9, 0.0, 1.1, 0.0, 1.1 # small offset (in pixels along X
      ) between SIBS_X and actual target position
```

```
   offsibs_y = +2.1, 0.0, 4.0, 0.0, 1.1 # small offset (in pixels along Y
      ) between SIBS_Y and actual target position
   labmode = False # If labmode = True, will ignore keywords and input
      parameters and create fake astrometry

# Logsocket "step" configuration
[logsocket]
   host = 'localhost' # host
   port = 50747       # port SOfia 747
   deflogger = 'pipe.logsocket' # default logger

### Data Section

# Treatment of the FITS header: can include keyword replacement
# The keyword value and comment must be printed as they would in a FITS
    header
# If the value is another keyword, the value of that keyword will be used
# instead (This only works if the other keywords starts with an
    alphabetic
# character).
[header]
    TAUOBS = 0.0 / Estimated optical depth

# Merge Header Section: How to merge header keywords when headers from
# several files are merged. Options are:
# - FIRST (default), LAST: For all values
# - DEFAULT: For all values (-9999 for ints, UNKNOWN for strings, etc)
# - MIN, MAX, SUM: For numbers
# - AND, OR: For boolean flags
# - CONCATENATE: For strings
[headmerge]
    ALTI_END = LAST
    ASSC_AOR = CONCATENATE
    DTHINDEX = DEFAULT
    LAT_END  = LAST
    LON_END  = LAST
    FBC-STAT = LAST
    FOCUS_EN = LAST
    SIBS_X   = DEFAULT
    SIBS_Y   = DEFAULT
    UTCEND   = LAST
    WVZ_END  = LAST
    ZA_END   = LAST
    TRACERR  = OR
    TSC-STAT = LAST

# Treatment for table values when combining images
# Options are MIN, MED, AVG, FIRST, LAST, SUM
[table]
    samples           = SUM
```

```
chop offset       = WTAVG
nod offset        = WTAVG
hwp angle         = WTAVG
azimuth           = WTAVG
azimuth error     = WTAVG
elevation         = WTAVG
elevation error   = WTAVG
array vpa         = WTAVG
nod index         = WTAVG
hwp index         = WTAVG
nod offset orig   = FIRST
framecounter      = FIRST
crioframenum      = WTAVG
hwpcounts         = WTAVG
fasthwpa          = WTAVG
fasthwpb          = WTAVG
fasthwpcounts     = WTAVG
a2a               = WTAVG
a2b               = WTAVG
b2a               = WTAVG
b2b               = WTAVG
chop1             = WTAVG
chop2             = WTAVG
criottlchopout    = FIRST
sofiachops        = WTAVG
sofiachopr        = WTAVG
sofiachopsync     = WTAVG
ai23              = WTAVG
crioanalogchopout = FIRST
irigupdatediff    = FIRST
timestamp         = WTAVG
ra                = FIRST
dec               = FIRST
chop_vpa          = FIRST
lon               = FIRST
lat               = FIRST
lst               = WTAVG
los               = WTAVG
xel               = WTAVG
tabs_vpa          = FIRST
pitch             = WTAVG
roll              = WTAVG
nonsiderealra     = WTAVG
nonsiderealdec    = WTAVG
flag              = WTAVG
pwv               = FIRST
nodpositionreached = FIRST
trackerraoi3      = FIRST
trackerraoi4      = FIRST
trackerraoi5      = FIRST
```

```
r array imag      = FIRST
t array imag      = FIRST
chop offset imag  = FIRST
r array avg       = FIRST
t array avg       = FIRST
phase corr        = WTAVG
```

B.2 DRP Override Configuration File

Below is an override configuration file that demonstrates how to set override parameters to provide to the HAWC pipeline. The parameters listed here are those most likely to change from one flight series to another.

```
# HAWC Pipeline Configuration File - Overrides for FS13
# (commissioning 2 and OC4L).
# This is not a full configuration file -- it should be merged
# with pipeconf_dcs.txt before using.
#
# 2016-12-08 M. Clarke

# Calibrate fluxes from data units to Jy/pixel
[calibrate]
   fac = 0.0454, 0.038, 0.0345, 0.0291, 0.0419

# Demodulate the chopped data
[demod]
   phasefile  = $DPS_HAWCPIPE/pipeline/data/phasefiles/
      F0005_HAWC_HWPC_138-180_phase_161011.fits

# Flat step configuration
[flat]
    flatfile   = search
    fitkeys    = 'SPECTEL1'
    flatfolder = $DPS_HAWCPIPE/pipeline/data/flats

# Correction for instrumental polarization
[ip]
   qinst =  -0.01191, 0.0, -0.01787, -0.00055, -0.01057
   uinst =  -0.00273, 0.0,  0.00758,  0.02179, -0.01201
   qtel  = 0.0, 0.0, 0.0, 0.0, 0.0
   utel  = 0.0, 0.0, 0.0, 0.0, 0.0

# Rotate Q and U from detector to sky frame step configuration
[rotate]
  gridangle = -78.69, 0.0, -93.28, 48.42, 130.62

# Update Parallactic angle and crval1 and crval2 for a single file
[wcs]
```

```
offsibs_x = -0.9, 0.0, 1.1, 0.0, 1.1
offsibs_y = +2.1, 0.0, 4.0, 0.0, 1.1
```

B.3 Full CRUSH Configuration File

Below is an example of the default global configuration file for CRUSH. Other configuration files specifying values for specific instruments or modes may override values in this file. See the CRUSH documentation for more information.

```
# Set the spherical projection to use.
# The following projections are currently supported:
#
#       SIN -- Slant Orthographic
#       TAN -- Gnomonic
#       SFL -- Sanson-Flamsteed
#       ZEA -- Zenithal Equal Area
#       MER -- Mercator
#       CAR -- Plate-Carree
#       AIT -- Hammer-Aitoff
#       GLS -- Global Sinusoidal
#       STG -- Stereographic
#       ARC -- Zenithal Equidistant
#
# The default is SFL, which is widely used, and is the fastest to
    calculate...
projection SFL

# Make equatorial maps by default. Other possibilities are 'horizontal',
# 'ecliptic', 'galactic', 'supergalactic', and 'focalplane'
system equatorial

# Set the parallelisation mode. The value should be one of:
#   scans      -- Each scan is reduced in a separate thread.
#   ops        -- Each scan is reduced by parallel threads, one at a time
    .
#   hybrid     -- Optimal threading with as many scans run in parallel as
#                 possible, each reduced with some number of parallel
    threads.
parallel hybrid

# For maps aligned to focal plane coordinates, do not attempt getting
    pointing
# offsets
[system?focalplane] blacklist point

# Do not attempt pointing fir on skydips
[source.type?skydip] blacklist point
```

```
# The ordering of models in the default reduction pipeline.
ordering offsets, drifts, correlated.obs-channels, weighting.frames,
    whiten, weighting, despike, correlated.gradients, correlated.accel,
    source

# Automatically create the output path, if it does not exists
#outpath.create

# In case an outputput name was set before loading default.cfg, clear it
forget name

# Turn this option on if you want to see intermediate maps as the
    reduction
# progresses. These are (over-)written to 'intermediate.fits'.
#source.intermediates

# The default 1/f stabilty time scale. Instruments should define their
    own.
stability 15.0
[extended] stability 30.0

# Determine the velocity clipping based on stability and beam size...
vclip auto

# The telescope pointing tolerance (in beams), e.g. for positions
    switched
# photometry
pointing.tolerance 0.2

# The maximum fraction of samples which can be out-of-range before the
    channel
# is flagged for being unusable.
range.flagfraction 0.05
[source.type?skydip] range.flagfraction 0.75

# Downsample data as needed...
downsample auto

# Check for timestream gaps and fill with null frames as necessary
fillgaps

# Remove the DC offsets before entering pipeline.
level

# Signal estimators to use ('median' or 'maximum-likelihood').
estimator median
iteration.[2] estimator maximum-likelihood

# Solve for pixels gains (with specified estimator type)
gains
```

```
gains.estimator maximum-likelihood

# Whether to measure responses to signals. If the option is set,
# the responses are printed in curly brackets during reduction. The
# values represent the normalized covariance of residuals to
# the given signals. By default response calculation is disabled
# to speed up reduction. It is mainly useful for designing
# instrument pipelines, whereas it is only informational once
# the pipelines are established.
#signal-response

# Enable filtering (components need to be enabled separately).
filter

# Define how FFT filters are to be compounded
filter.ordering motion, kill, whiten

# Apply the kill filter only once before downsampling & reduction
    pipeline.
iteration.[1] forget filter.kill

# Turn on additional re-levelling of the filtered signal to be extra
    pedantic
# This will make filtering slower, without noticeable increase in
# effectiveness...
#filter.mrproper

# Set 'whiten' as an alias to 'filter.whiten' for backward compatibility
alias.whiten filter.whiten

# Define the shorthand 'motion' for 'filter.motion'
alias.motion filter.motion

# Remove the scan synchronous signals, e.g. telescope vibrations.
#filter.motion
filter.motion.range 0.01:1.0
filter.motion.s2n 6.0
filter.motion.above 0.3
#filter.motion.stability 30.0
[extended] forget filter.motion
#[filter.motion] forget whiten.below

# Shorthand 'kill' for 'filter.kill'
alias.kill filter.kill

# A frequency quenching filter can be enabled when needed, with all
    frequencies
# in the specified bands being eliminated from the timestream data...
#filter.kill
#filter.kill.bands 10.1--10.2, 12.35--12.37
```

```
# Derive pixel weights (via 'rms', 'differential' or 'robust' method).
weighting
weighting.method rms
[extended] weighting.method differential

# Specify the range of acceptable pixel noise rel. to the median pixel
    noise
# Channels outside of this admissible range will be flagged.
weighting.noiserange 0.1:10.0

# Time weighting (optionally with time-resolution in seconds or 'auto').
# Time weighting should be used with caution. It can lead to unstable
    solutions
# especially when there is bright/extended emission. Therefore it should
    be
# used with the longest possible time-scale, or not at all...
#weighting.frames
[extended] forget weighting.frames

# Set the time window (seconds) for weighting frames, or 'auto'.
weighting.frames.resolution auto

# Specify an acceptable range for time-weights. Frames with weights
    outside
# this range will be flagged.
weighting.frames.noiserange 0.3:3.0

# Solve for source!!!
source
source.type map

# Some aliases for easy selection of source types
[map] source.type map
[beammap] source.type beammap
[skydip] source.type skydip

# If beam mapping, then reduce in horizontal
[source.type?beammap] system focalplane

# Do not use initial pixel data when reducing beammaps...
[source.type?beammap] blacklist pixeldata

# Simplify pipeline for skydips...
[source.type?skydip] blacklist aclip,vclip,drifts,offsets,whiten,point
#[source.type?skydip] estimator median


# If inserting test sources into the data, use static source gains
```

```
[sources] source.fixedgains

# For chopped observations disable velocity and acceleration clipping and
# downsampling
[chopped] forget aclip

# When calculating the array perimeter (for sizing maps) how many
    sections to
# use. For very large arrays, especially with jagged geometry on its
    edges, you
# may want to use more sections than the default (100) to make sure maps
    are
# sized correctly. A negative or zero value will use all pixels (safest)
    for
# sizing maps.
perimeter auto

# Require a minimum number of good pixels for mapping as a fraction of
    the
# nominal pixel count on the array. Note, you can also set this as a
    number
# by using the option 'mappingpixels' instead...
mappingfraction 0.5

# Determine the relative coupling (i.e. relative beam efficiency)
# for each channel, based on the response to the bright (i.e. blanked)
# areas of the source map.
# EXPERIMENTAL feature! Use with caution...
#source.coupling

# Use only the points in the map that for determining coupling
    efficiencies,
# which are within the specified S/N range
source.coupling.s2n 5.0:*

# Define the acceptable dynamic range for the source coupling of channels
# When the estimated coupling falls outside of this range, the default
# value of 1.0 is assumed.
#source.coupling.range 0.3:3.0

# By default source gains are dynamically calculated from the sky-noise
    gains.
# To override this, and to use fixed gains (e.g. from RCP files),
    uncomment
# the line below (or specify it on the command line).
#source.fixedgains

# For skydip reductions, make source gains become the correlated gains
[source.type?skydip] sourcegains
```

```
# Use MEM correction on the source map?
#source.MEM
[extended] forget source.MEM
iteration.[last] forget source.MEM

# Set the 'desirability' of MEM solution (0 -- 1)
source.MEM.lambda 0.1

# Calculate coupling efficiencies suign information from RCP files
# (when defined).
rcp.gains

# Define 'array' as a shorthand for 'correlated.obs-channels'
alias.array correlated.obs-channels
# Always decorrelate observing channels.
array
array.gainrange 0.1:10.0
[extended] correlated.*.gainrange 0.01:100

# Define 'gradients' as a shorthand for 'correlated.gradients'
alias.gradients correlated.gradients
# Do not solve for gradients for extended sources
#gradients
[extended] forget gradients

# Define 'sky' as a shorthand for 'correlated.sky'
alias.sky correlated.sky

# Define 'nonlinearity' as a shorthand for 'correlated.nonlinearity'
alias.nonlinearity correlated.nonlinearity

# Define 'accel' as a shorthand for 'correlated.accel-mag'
# This definition may be overwritten by instruments...
alias.accel correlated.accel-mag

# Make 'offsets' and 'drifts' mutually exclusive
[drifts] forget offsets
[offsets] forget drifts

# Solve for pixel drifts (1/f filtering) at given timescale
drifts 30
[extended] drifts 300
drifts.method blocks
iteration.[3] drifts auto

# Set the nunber of iterations required
# To recover more extended emission, you can increase the number of
    iterations
# when using the 'extended' option. The more you iterate, the more large
    scale
```

```
# emission is recovered. However, beware that the larger scales will be
    also
# inherently noisier due to the typical 1/f-type noise interference.
rounds 6
[extended] rounds 20

# Despike with the specified method ('neighbours', 'absolute', 'gradual'
    or
# 'multires') above the critical S/N level.
despike
despike.level 100.0
despike.method neighbours
despike.flagfraction 3e-3
despike.flagcount 10
despike.framespikes 3
despike.width auto
#despike.blocks

# Default dejumping settings. Level relative to noise level, and minimum
    length
# in seconds, above which the jump will be re-levelled, below which
    flagged.
# You can also set the time-resolution (in deconds) of de-jumping. If not
     set
# all frames are dejumped individually.
dejump.level 2.0
dejump.minlength 5.0
#dejump.resolution = 0.3

# Smooth internal source generations a little to get rid of pixellization
# noise. This setting will not necessarily determine the smoothing of the
# final output map, as the setting is normally revised in the last
    iteration
# (see further below)...
smooth minimal
[extendex] smooth halfbeam

# Using lookup tables for sample -> map index can result in a significant
# increase of speed (by 30% typically). However, these tables can take up
# a lot of RAM, which may limit the reduction of large datasets.
    Therefore
# it is recommended to set a usage limit as a fraction of the maximum
# available memory. Values around 0.8 would be typical to allow for
    various
# overheads during reduction.
indexing auto
indexing.saturation 0.80

# Clip maps only to retain really bright source features, which have to
```

```
# be removed before despiking. As the despiking level is tightened, so
    the
# clipping level will drop. For the final iteration the clipping is
    omitted
# (see further below) s.t. in the end an unbiased source map is produced.
clip 30.0

# Do not clip initially when a 'source.model' is supplied.
[source.model] forget clip

# If using a source model, do not clip. (It should not be necessary,
    since
# after applying the model, one should be left with faint signals only.
#[source.model] blacklist clip

# Select the signedness of the expected sources. The masking will happen
# when the deviation from zero is larger than the 'blank' level in that
# direction. A value of 0 makes the blanking bi-directional (both
    positive
# and negative deviations will be masked if large enough...
# By default, we assume that sources are positive only, so accordingly
    set
# the blanking direction to '+'
source.sign +

# Blanking of bright sources is initially set high since
# it may hinder despiking...
blank 30.0

# If a source model is used, also adjust the blanking level to faint
    signals
[source.model] forget blank

# Now that the brightest features have been blanked, despike more tightly
    , and
# follow-up with clipping and blanking at lower S/N levels....
iteration.[2] despike.level 30.0
iteration.[2] clip 10.0
iteration.[2] blank 10.0

# Continue going for fainter fluxes in the third iteration, while
    retaining
# clipping of non-significant features.
iteration.[3] despike.level 10.0
iteration.[3] clip 4.0

# By now the bright features should be well modeled. For fainter
    structures,
# switch to using maximum-likelihood estimators
#iteration.[4] estimator maximum-likelihood
```

```
iteration.[4] despike.method absolute
iteration.[4] clip 2.0

# Once a decent enough source model is reached, disable further clipping
# and blanking, and allow unbiased modeling of the source. Extended
    emission
# will be recovered gradually with the iterations...
# WARNING! Failure to disable clipping AND blanking in time, may prevent
    the
# recovery of the extended emission. Change with this setting only if you
     know
# what you are doing...
[extended] iteration.[3] clip 2.0
[extended] iteration.[4] blacklist blank,despike

# Use a noise whitening filter on the unmodelled residuals.
iteration.[last-1] whiten
[extended] iteration.[90%] whiten

# Once solutions have sufficiently converged, allow the spectral noise
# whitening filter to clean the unmodeled residuals.
# Set the critical level above white noise beyond which to apply
    whitening
whiten.level 2.0

# By default whitening only suppresses excessive noise. It is possible to
# configure it such that the filter also brings noise up from below the
# inverted critical level -- thus resulting in a truly white spectrum. A
# white noise spectrum ensures that the map pixels/beams are uncorrelated
    .
whiten.below

# The maximum allowed power-boost to spectral channels when whitening
    noise
# below the mean...
whiten.below.max 2.0

# Set the frequency range (in Hz), in which the whitening filter is to
    measure
# the white-noise level. By default it will use the entire spectral range
# available. The value 'auto' will automatically tune the probe range for
# point sources.
whiten.proberange auto

# Weight each scan based on its measured map-noise (robust estimation)
#weighting.scans robust
[extended] forget weighting.scans

# Despiking of source (per scan) above some S/N level.
forget source.despike
```

```
# Minimum redundancy per scanmap pixel
source.redundancy 2
[source.type?beammap] forget source.redundancy

# Correct map fluxes below clipping/blanking level for the filtering
    effect
# of auxillary models when map is iterated. When the map is not iterated,
# the correction automatically takes place using a different method.
iteration.[last] source.correct

# Noise clip the final map, s.t. map pixels with noise more than 10-times
     the
# least noisy part of the map are flagged.
#noiseclip 10.0
forget noiseclip

# Clip map points that have been integrated less than the specified
    fraction of
# the best covered part.
iteration.[last] exposureclip 0.04

# Make completely sure that the last map generation is without clipping.
# The later clipping/blanking is disabled, the more faint extended
    emission
# will be filtered away...
iteration.[last] blacklist clip,blank

# Do not smooth the final map (even if intermediates were smoothed).
[extended] smooth halfbeam
iteration.[last] forget smooth

# Assuming that the source is at the end of the pipeline, there is no
    need to
# sync to time-streams in the last iteration. Instruments, or
    configurations
# in which source is moved forward in the pipeline 'ordering', should
    reset
# this...
iteration.[last] source.nosync

# Do not LSS filter the source
forget source.filter

# The filtering method (when used) -- 'convolution' or 'fft'
source.filter.type convolution

# The interpolation for the convolution filter (when used).
# Can be 'nearest', 'linear', 'quadratic' or 'cubic'
source.filter.interpolation cubic
```

```
# Additional options to beam maps...
# Process beam maps like regular maps
beammap.process

# Write individual images for every pixel
#beammap.writemaps
[source.type?beammap] blacklist exposureclip
[source.type?beammap] forget rcp

# Specify the method for determining pointing offsets (also for beammap)
# Choose between 'peak' and 'centroid'.
pointing.method centroid

# Derive pointing only if the peak S/N exceeds a critical level
pointing.significance 6.0

# Discard the underexposed parts of the map when deriving pointing
    results
# This does not affect the output image in any way
pointing.exposureclip 0.25

# Use 'point' as a shorthand for determining the pointing offsets at the
    end.
[point] final:pointing suggest

# Additional settings for skydips...
# The binning of skydips (in arcsec)
skydip.grid 900.0
[source.type?skydip] beam {?skydip.grid}
[source.type?skydip] beam.lock

# What parameters to fit: 'tau', 'offset', 'kelvin', 'Tsky'
skydip.fit tau,offset,kelvin

# Specify manually the physical sky temperature (K) to use
#skydip.Tsky 273.0

# Use uniform weighting of all sky-dip points
skydip.uniform

# The maximum number of fitting attempts for skydip data.
skydip.attempts 10

# Whether to attempt displaying the result (e.g. via 'gnuplot').
#iteration.[last] show

# For skydip show result by default
#[source.type?skydip] show
```

```
# For reducing very large datasets, i.e. what cannot be fit into memory
    in
# a single go, one has no option but to split the reduction into
    manageable
# sized chunks, and then use 'coadd' to create composite maps. Once a
# composite is made, it can be fed back into a second reduction via the
# 'source.model' key to obtain a better solution. Such manual iterating
    may be
# useful to get rid of negative bowls around the fainter areas, which are
# not bright enough in the individual chunks. To aid the reduction of
    split
# datasets, you can use the 'split' option, which disables smoothing to
    create
# raw maps suitable for coadding and external smoothing via 'imagetool'
[split] smooth.external

# Split reductions should not be clipped by exposure either...
[split] final:forget exposureclip

# Add the scan specific information at the end of the output FITS image.
    Each
# scan will contribute an extra HDU.
write.scandata

# The above will write only some very basic information about each scan.
# You can add more richness to the scan information (e.g. channel gains,
# weights, flags, noise spectra and filter profiles) by enabling the
# 'scandata.details' option
#scandata.details

# Write EPS (encapsulated postscript) images, if available
write.eps

# You can write PNG thumbnails together with FITS images...
write.png

# Choose which image plane to write ('flux', 'noise', 'weight', 'time' or
# 's2n'). Default is 'flux'.
write.png.plane s2n

# Choose the PNG size (in pixels)
write.png.size 500x500

# The PNG colorscheme ('colorful', 'grayscale', 'orange' or 'blue')
write.png.color colorful

# The PNG background Hex RGB value (e.g. 0xFFFFFF), or 'transparent'
write.png.bg transparent

# Smooth the PNG image
```

```
write.png.smooth halfbeam

# Enable bicubic spline interpolation for non-pixelized, smooth, PNG
    output
#write.png.spline

# Crop the PNG to specific bounds (arcsec)
# write.png.crop -60,-60,60,60

# Allow using gnuplot (e.g. for skydip plots). Requires a gnuplot
    installation
# to work...
gnuplot

# If gnuplot is not in youtr path, you gen specify the full path to the
# gnuplot executable instead of the above:
#gnuplot /usr/bin/gnuplot


# Options for laboratory data reduction 'lab' mode. depending on
    instrument
# support, these can bypass astrometry and telescope dtaa altogether

# Enable lab mode recution
#lab

# Set an assumed scanning speed (arcsec/s) for adjusting filter
    parameters
# If not set, CRUSH will assume 10 beams/s.
#lab.scanspeed 100

# Various modifications to configurations for 'lab' mode reductions
[lab] blacklist source
[lab] blacklist filter.motion
[lab] blacklist tau
[lab] blacklist whiten
[lab] blacklist shift
[lab] blacklist point
[lab] forget downsample
[lab] write.spectrum

# You can use the virtual option 'derive' to derive new pixeldata files.
# It will invoke the following settings conditionally, as well as any
# similar conditional settings based on brightness and/or the instrument
# configuration...
[derive] forget pixeldata,vclip,aclip
[derive] write.pixeldata
[derive] blacklist whiten
[derive] rounds 30
```

```
# If the 'source.flatfield' option is set, then load appropriate settings
# for flatfield determination...
[source.flatfield] config hawc+/flatfield.cfg

# Always sync the source model if writing timestreams, spectra, or
    covariances
[write.ascii] blacklist source.nosync
[write.spectrum] blacklist source.nosync
[write.covar] blacklist source.nosync

# Some convenient aliases:
# the keys 'altaz', 'horizontal', 'radec', 'equatorial', 'ecliptic', '
    galactic'
# and 'supergalactic' are defined. E.g.,
#
# > ./crush [...] -galactic [...]
#
# can be used to produce maps in galactic coordinates
#
alias.altaz system horizontal
alias.horizontal system horizontal
alias.equatorial system equatorial
alias.radec system equatorial
alias.ecliptic system ecliptic
alias.galactic system galactic
alias.supergalactic system supergalactic
alias.focalplane system focalplane

# 'final' is a shorthand for iteration.[last]. This can be used, for
    example
# to specify a map smoothing at the end of reduction. On the command line
# an example of this would look like:
#
# > ./crush [...] -final:smooth=beam [...]
#
# Note, that the colon (:) is used as a separator between the alias and
    the
# conditional setting on the command-lines, because spaces are not
    allowed.
#
alias.final iteration.[last]

# Some shorthand for iteration-based settings
alias.i iteration
alias.i1 iteration.[1]
alias.i2 iteration.[2]
alias.i3 iteration.[3]

# Some aliases for better backward compatibility (esp. with minicrush)
alias.center pointing
```

```
alias.time-weighting weighting.frames
alias.planetary moving
alias.reservecpus idle
#alias.extfilter source.filter
#alias.scanmap-redundancy source.redundancy
#alias.scanweighting source.weighting
#alias.scanmap-despike source.despike
#alias.relative-noise-range weighting.noiserange
#alias.rcpgains source.fixedgains

# Always reduce the Moon as 'bright'
object.[Moon] bright

# invoke the appropriate brightness configuration when one of the
    brightness
# options is set...
[bright] config bright.cfg
[faint] config faint.cfg
[deep] config deep.cfg
```

C Appendix: Required Header Keywords

The file below defines all keywords that the HAWC pipeline checks for validity before proceeding. It is normally located in the hawc distribution at *hawc/pipeline/config/-header_req_config.txt*. The path to this file should be specified in the pipeline configuration file under the '[checkhead]' section in order to perform the header check.

```
# HAWC pipeline header requirements configuration file
#
# Keywords in this list are only those required for successful
# data reduction (grouping and processing).  There may be more
# keywords required by the SOFIA DCS.
#
# Requirement value should be *, chopping, nodding, dithering,
# or scanning (as denoted by the corresponding FITS keywords).
# * indicates a keyword that is required for all data.  All
# others will only be checked if they are appropriate to the
# mode of the input data.
#
# DRange is not required to be present in the configuration --
# if missing, the keyword will be checked for presence only.  If
# drange is present, it will be checked for an enum requirement
# first; other requirements are ignored if present.  Min/max
# requirements are only used for numerical types, and are inclusive
# (i.e. the value may be >= min and <= max).
#
# 2016-08-22 Melanie Clarke: First version
```

```
[CHOPPING]
    requirement = *
    dtype = bool

[CHPAMP1]
    requirement = chopping
    dtype = float
    [[drange]]
        min = -1125
        max = 1125

[CHPANGLE]
    requirement = chopping
    dtype = float
    [[drange]]
        min = -360
        max = 360

[CHPCRSYS]
    requirement = chopping
    dtype = str
    [[drange]]
        enum = TARF, ERF, SIRF

[CHPFREQ]
    requirement = chopping
    dtype = float
    [[drange]]
        min = 0.0
        max = 20.0

[CHPONFPA]
    requirement = chopping
    dtype = bool

[DATE-OBS]
    requirement = *
    dtype = str

[DITHER]
    requirement = *
    dtype = bool

[DTHINDEX]
    requirement = dithering
    dtype = int
    [[drange]]
        min = 0
```

```
[DTHSCALE]
    requirement = dithering
    dtype = float

[DTHXOFF]
    requirement = dithering
    dtype = float

[DTHYOFF]
    requirement = dithering
    dtype = float

[EQUINOX]
    requirement = *
    dtype = float

[EXPTIME]
    requirement = *
    dtype = float
    [[drange]]
        min = 0.0

[FOCUS_EN]
    requirement = *
    dtype = float
    [[drange]]
        min = -5000.0
        max = 5000.0

[FOCUS_ST]
    requirement = *
    dtype = float
    [[drange]]
        min = -5000.0
        max = 5000.0

[HWPSTART]
    requirement = nodding
    dtype = float
    [[drange]]
        min = -360.0
        max = 360.0

[INSTCFG]
    requirement = *
    dtype = str
    [[drange]]
        enum = TOTAL_INTENSITY, POLARIZATION

[INSTMODE]
```

```
    requirement = *
    dtype = str
    [[drange]]
        enum = C2N (NMC), OTFMAP

[INSTRUME]
    requirement = *
    dtype = str
    [[drange]]
        enum = HAWC_PLUS

[MCEMAP]
    requirement = scanning
    dtype = str

[NHWP]
    requirement = nodding
    dtype = int
    [[drange]]
        min = 1

[NODDING]
    requirement = *
    dtype = bool

[NODPATT]
    requirement = nodding
    dtype = str
    [[drange]]
        enum = ABBA, A

[OBJECT]
    requirement = *
    dtype = str

[OBS_ID]
    requirement = *
    dtype = str

[PIXSCAL]
    requirement = nodding
    dtype = float
    [[drange]]
        min = 0.0

[SIBS_X]
    requirement = *
    dtype = float

[SIBS_Y]
```

```
    requirement = *
    dtype = float

[SMPLFREQ]
    requirement = *
    dtype = float
    [[drange]]
        min = 1.0

[SPECTEL1]
    requirement = *
    dtype = str
    [[drange]]
        enum = HAW_A, HAW_B, HAW_C, HAW_D, HAW_E

[SPECTEL2]
    requirement = *
    dtype = str
    [[drange]]
        enum = NONE, HAW_HWP_A, HAW_HWP_B, HAW_HWP_C, HAW_HWP_D,
            HAW_HWP_E, HAW_HWP_Open, HAW_HWP_Offset1, HAW_HWP_Offset2,
            HAW_HWP_Offset3

[SRCTYPE]
    requirement = *
    dtype = str
    [[drange]]
        enum = POINT_SOURCE, COMPACT_SOURCE, EXTENDED_SOURCE, OTHER,
            UNKNOWN

[TELDEC]
    requirement = *
    dtype = float
    [[drange]]
        min = -90.0
        max = 90.0

[TELRA]
    requirement = *
    dtype = float
    [[drange]]
        min = 0.0
        max = 24.0

[UTCSTART]
    requirement = *
    dtype = str

[WAVECENT]
    requirement = *
```

```
dtype = float
[[drange]]
    min = 0.0
```