Guest Observer Handbook for HAWC+ Data Products

May 17, 2018

Contents

1 Introduction

This guide describes the reduction algorithms used by and the data produced by the SOFIA/HAWC+ data reduction pipeline (DRP) for guest investigators. The HAWC+ observing modes, for both total intensity and polarimetric observations, are described in the SOFIA Observers Handbook, available from the Proposing and Observing page on the SOFIA Web site.

This guide applies to HAWC+ DRP version 1.3.0.

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 two standard observing modes for imaging: the Chop-Nod instrument mode combines traditional chopping with nodding; the Scan mode uses slow telescope scans without chopping. The Scan mode is the most commonly used for total intensity 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 for 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.

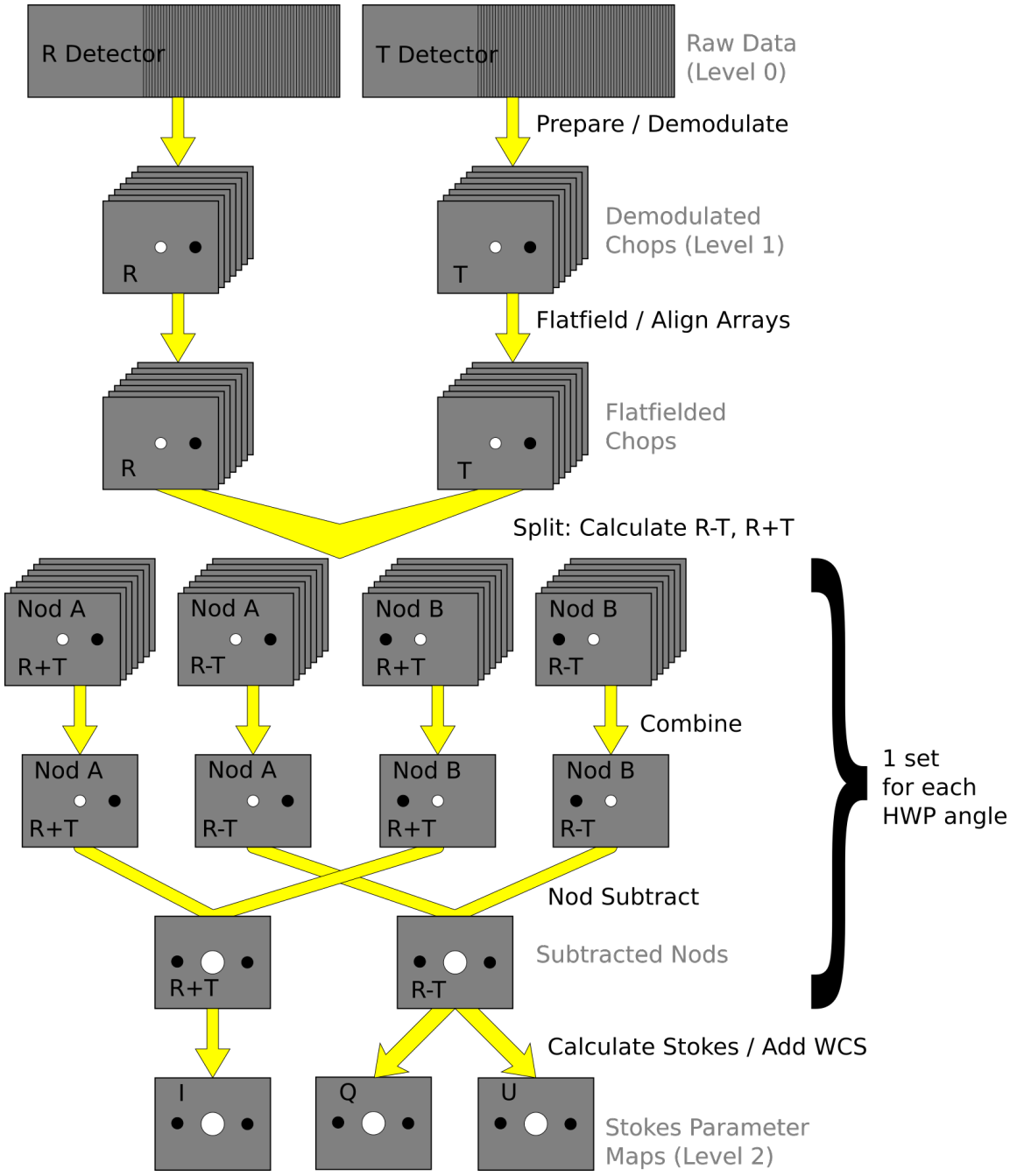

Figure 1: Nod-Pol data reduction flowchart, up through Stokes parameter calculation for a single input file.

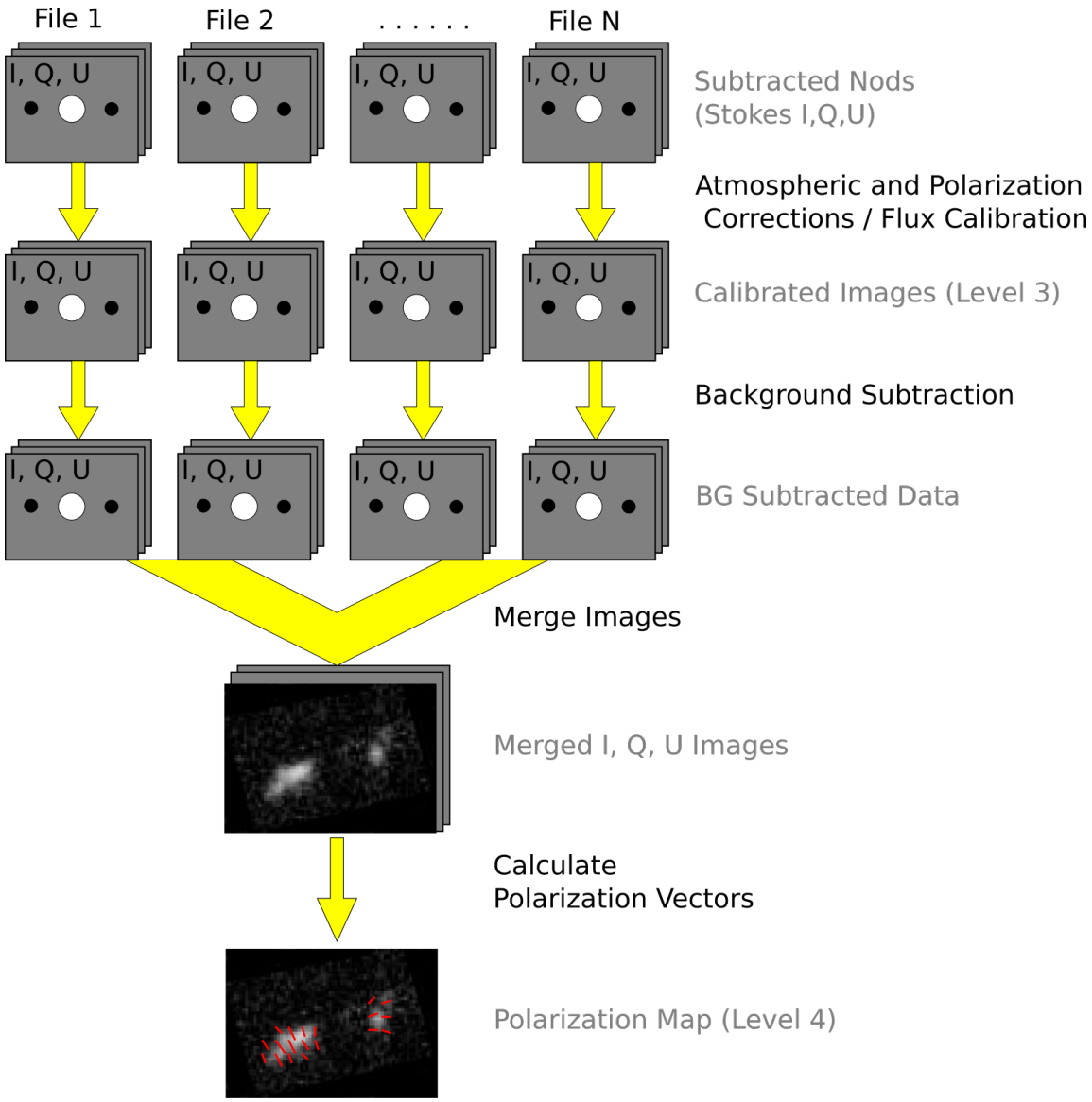

Figure 2: Nod-Pol data reduction flowchart, picking up from Stokes parameter calculation, through combining multiple input files and calculating polarization vectors.

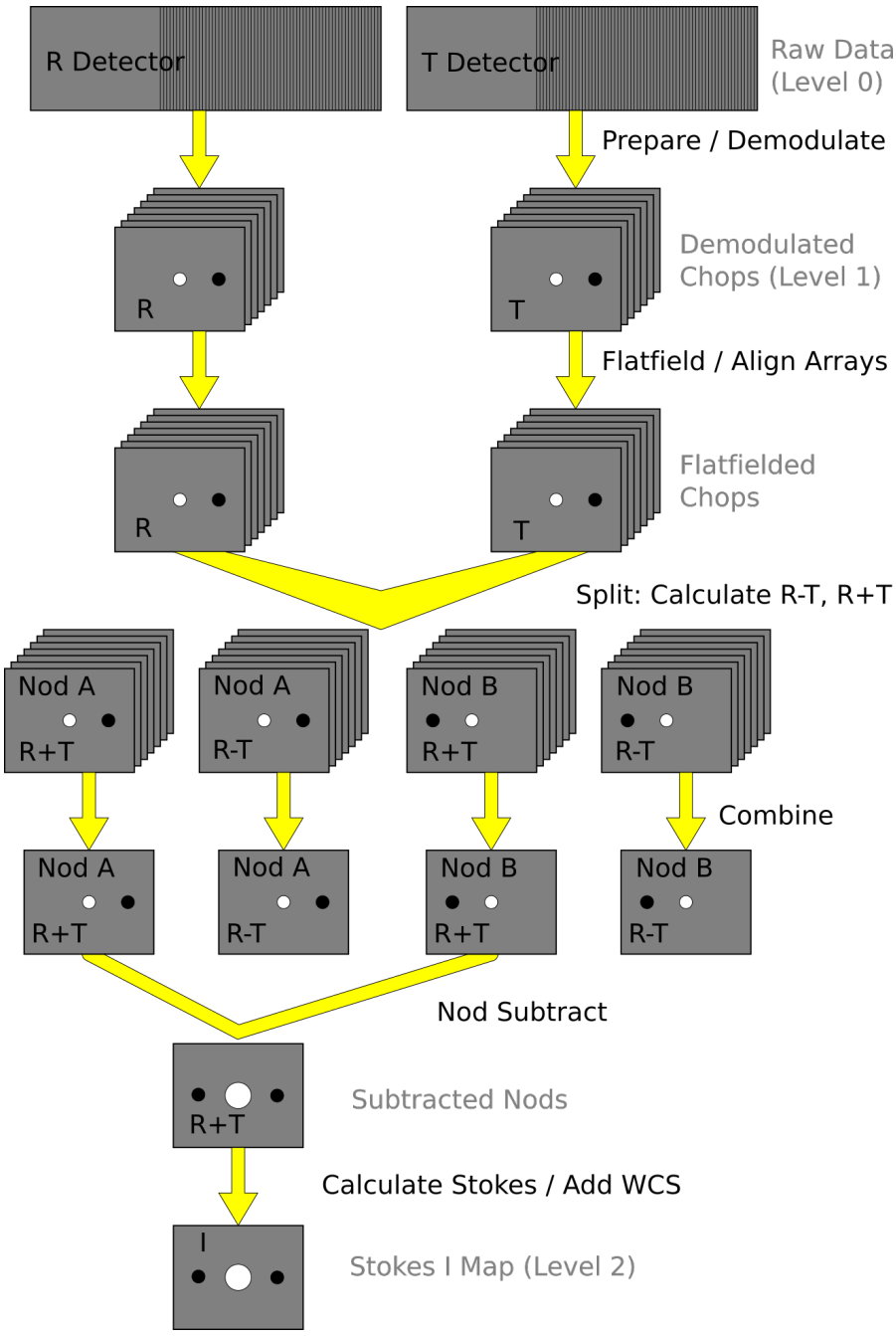

Figure 3: Chop-Nod data reduction flowchart, up through Stokes parameter calculation for a single input file.

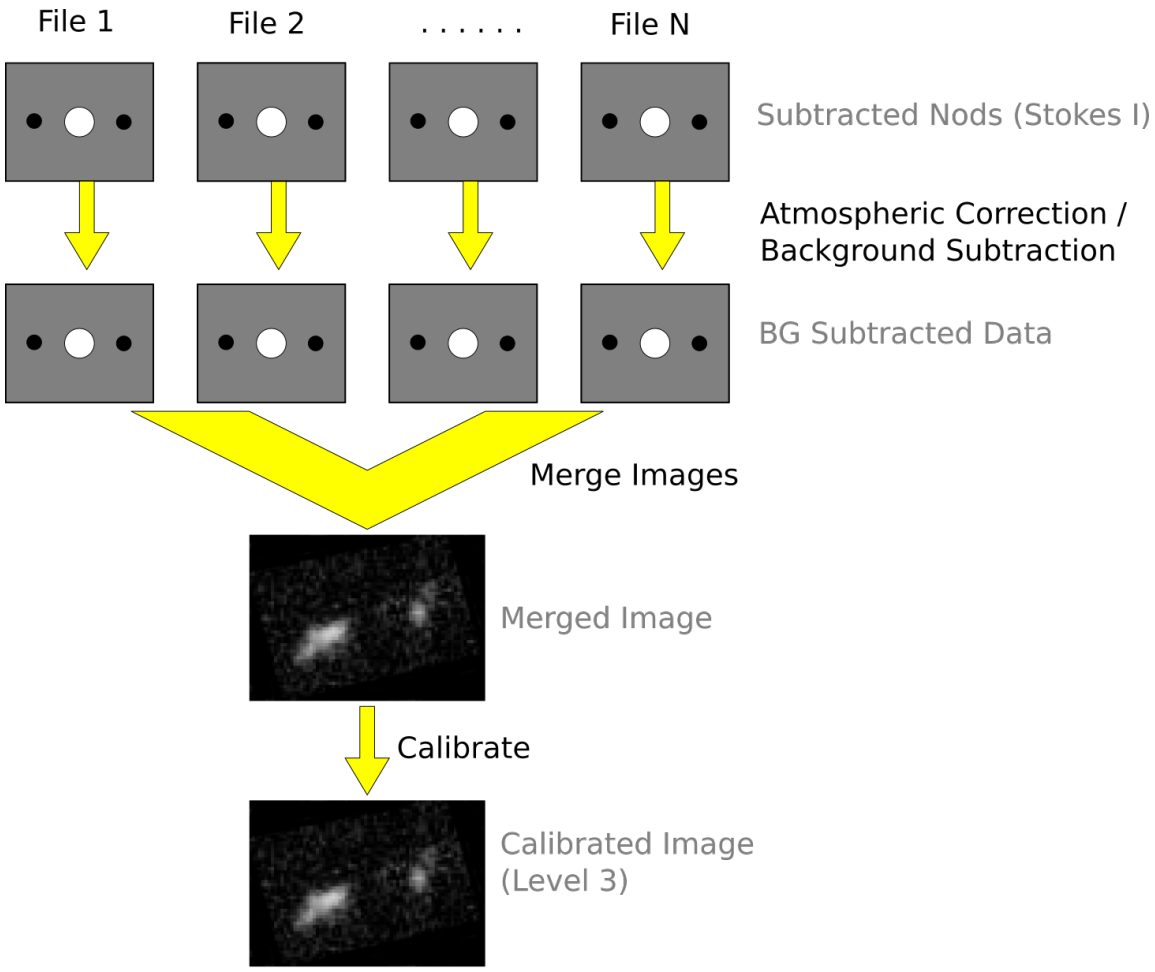

Figure 4: Chop-Nod data reduction flowchart, picking up from Stokes parameter calculation, through combining multiple input files.

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 includes the following HDUs:

- Primary HDU: Contains the necessary FITS keywords in the header but no data. It contains all required keywords for SOFIA data files, 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): Contains MCE (detector electronics) configuration data. This HDU is stored only in the raw and demodulated files; it is not stored in Level 2 or higher data products. 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 from 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.

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. The process is equivalent to identifying matched sets of chopped images and subtracting them.

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.

Alongside the chop-subtracted flux, the pipeline calculates the error on the raw data during demodulation. It does so by taking the mean of all data samples at the same chop phase, nod position, HWP angle, and detector pixel, then calculates the variance of each raw data point with respect to the appropriate mean. The square root of this value gives the standard deviation of the raw flux. The pipeline will propagate these calculated error estimates throughout the rest of the data reduction steps.

The result of the demodulation process is a chop-subtracted, time-averaged flux value and associated variance for each nod position, HWP angle, and detector pixel. 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 are generated on the fly from internal calibrator files (CALMODE=INT_CAL), taken before and after each set of science data. Flat files contain normalized gains for the R and T array, so that they are corrected to the same level. Flat files also contain associated variances and 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 flat variance values are also propagated into the data variance planes.

The output from this step contains FITS images in addition to the data tables. The R array data is stored as an image in the primary HDU; the R array variance, T array data, T array variance, R bad pixel mask, and T bad pixel mask are stored as images in extensions 1 (EXTNAME="R ARRAY VAR"), 2 (EXTNAME="T ARRAY"), 3 (EXTNAME="T ARRAY VAR"), 4 (EXTNAME="R BAD PIXEL MASK"), and 5 (EXTNAME="T BAD PIXEL MASK"), respectively. The DEMODULATED DATA table is attached unmodified as extension 6. The R and T data and variance 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: DATA and VAR image extensions for each of R+T and R-T for each HWP angle and nod position, a VAR extension for uncombined R and T arrays at 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, 32

variance images, 8 binary tables, and 1 bad pixel mask image, for 57 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 variance values are propagated through the mean, and the square root of the resulting value is stored as an error image in the output.

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, and the variance images for R+T and R-T replaced with ERROR images. For the example above, with two nod positions and four HWP angles, there are still 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 and ERROR images for each set, as well as variance images for R and T arrays, 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 propagated from the previously calculated errors

for R+T:

$$\sigma_I = \frac{1}{N}\sqrt{\sum_{\phi=1}^{N} \sigma^2_{R+T,\phi}}.$$

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:

$$\sigma_Q = \frac{1}{2}\sqrt{\sigma^2_{R-T,0} + \sigma^2_{R-T,45}}$$

$$\sigma_U = \frac{1}{2}\sqrt{\sigma^2_{R-T,22.5} + \sigma^2_{R-T,67.5}}.$$

Since Stokes I, Q, and U are derived from the same data samples, they will have non-zero covariance. For later use in error propagation, the pipeline now calculates the covariance between Q and I (σ_{QI}) and U and I (σ_{UI}) from the variance in R and T as follows:

$$\sigma_{QI} = \frac{1}{8}[\sigma^2_{R,0} - \sigma^2_{R,45} - \sigma^2_{T,0} + \sigma^2_{T,45}]$$

$$\sigma_{UI} = \frac{1}{8}[\sigma^2_{R,22.5} - \sigma^2_{R,67.5} - \sigma^2_{T,22.5} + \sigma^2_{T,67.5}]$$

The covariance between Q and U (σ_{QU}) is zero at this stage, since they are derived from data for different HWP angles.

The output from this step contains an extension for the flux and error of each Stokes parameter, as well as the covariance images, 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, STOKES Q, ERROR Q, STOKES U, ERROR U, COVAR Q I, COVAR U I, COVAR Q U, BAD PIXEL MASK, TABLE DATA). For Chop-Nod imaging, only Stokes I is calculated, so there are only 3 additional extensions (ERROR 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 right ascension and declination (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 and covariance images.

3.1.11 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 uses measurements of the sky to characterize the introduced polarization in reduced Stokes parameters ($q = Q/I$ and $u = U/I$) for each filter band at each pixel. The correction is then applied as

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

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

and propagated to the associated error and covariance images as

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

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

$$\sigma_{Q'I} = \sigma_{QI} - q'\sigma_I^2$$

$$\sigma_{U'I} = \sigma_{UI} - u'\sigma_I^2$$

$$\sigma_{Q'U'} = -u'\sigma_{QI} - q'\sigma_{UI} + qu\sigma_I^2.$$

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

3.1.12 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 the correction to the Q and U images with a standard rotation matrix, such that:

$$Q' = cos(\alpha)Q + sin(\alpha)U$$

$$U' = sin(\alpha)Q - cos(\alpha)U.$$

The errors and covariances become:

$$\sigma'_Q = \sqrt{(cos(\alpha)\sigma_Q)^2 + (sin(\alpha)\sigma_U)^2 + 2cos(\alpha)sin(\alpha)\sigma_{QU}}$$

$$\sigma'_U = \sqrt{(sin(\alpha)\sigma_Q)^2 + (cos(\alpha)\sigma_U)^2 - 2cos(\alpha)sin(\alpha)\sigma_{QU}}$$

$$\sigma_{Q'I} = cos(\alpha)\sigma_{QI} + sin(\alpha)\sigma_{UI}$$

$$\sigma_{U'I} = sin(\alpha)\sigma_{QI} - cos(\alpha)\sigma_{UI}$$

$$\sigma_{Q'U'} = cos(\alpha)sin(\alpha)(\sigma_Q^2 - \sigma_U^2) + (sin^2(\alpha) - cos^2(\alpha))\sigma_{QU}.$$

3.1.13 Calibrate Flux

The pipeline now converts the flux units from instrumental counts to physical units of Jansky per pixel (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.

The calibration factors are computed in a manner similar to that for another SOFIA instrument (FORCAST), taking into account that HAWC+ is a bolometer, not a photon-counting device. Measured photometry is compared to the theoretical fluxes of objects (standards) whose spectra are assumed to be known. The predicted fluxes in each HAWC+ passband are computed by multiplying the model spectrum by the overall response curve of the telescope and instrument system and integrating over the filter passband. For HAWC+, the standards used to date include Uranus, Neptune, Ganymede, Callisto, Ceres, and Pallas. The models of the first four objects were obtained from the Herschel project (see Mueller et al. 2016). Standard thermal models are used for Ceres and Pallas. All models are scaled to match the distances of the objects at the time of the observations. Calibration factors computed from these standards are then corrected by a color correction factor based on the mean and pivot wavelengths of each passband, such that the output flux in the calibrated data product is that of a nominal, flat spectrum source at the mean wavelength for the filter. See the FORCAST GO Handbook, available from the SOFIA webpage, for more details on the calibration process.

The calibration factor is directly applied to the flux in each of the Stokes I, Q, and U images. The overall calibration is expected to be good to within about 10%. For Chop-Nod imaging data, this factor is applied after the merge step, below.

3.1.14 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, Q, and U images
- Compare each Stokes I image (smoothed) to the combined map to determine any background offset or scaling
- Subtract the offset from the input (unsmoothed) Stokes I images; scale the input Stokes I, Q, and U images

- Compare each smoothed Stokes Q and U images to the combined map to determine any additional background offset
- Subtract the Q and U offsets from the input Q and U images

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

$$F'_I = (F_I - a_I)/b$$

$$F'_Q = (F_Q - a_Q)/b$$

$$F'_U = (F_U - a_U)/b$$

and are propagated into the associated error and covariance images appropriately.

3.1.15 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. First, 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 output grid point and, optionally, their associated errors. The value at the output grid pixel is the weighted average of the input pixels within the considered window. The output grid may subsample the input pixels: by default, there are 4 output pixels for each input pixel. For flux conservation, the output flux is multiplied by the ratio of the output pixel area to the input pixel area.

The error maps output by this algorithm are calculated from the input variances for the pixels involved in each weighted average. That is, the output fluxes from N input pixels are:

$$I' = \frac{\sum_i^N w_{i,I} I_i}{w_{tot,I}}$$

$$Q' = \frac{\sum_i^N w_{i,Q} Q_i}{w_{tot,Q}}$$

$$U' = \frac{\sum_i^N w_{i,U} U_i}{w_{tot,U}}$$

and the output errors and covariances are

$$\sigma'_I = \frac{\sqrt{\sum_i^N (w_{i,I}\sigma_{i,I})^2}}{w_{tot,I}}$$

$$\sigma'_Q = \frac{\sqrt{\sum_i^N (w_{i,Q}\sigma_{i,Q})^2}}{w_{tot,Q}}$$

$$\sigma'_U = \frac{\sqrt{\sum_i^N (w_{i,U}\sigma_{i,U})^2}}{w_{tot,U}}$$

$$\sigma'_{QI} = \frac{\sum_i^N w_{i,Q}w_{i,I}\sigma_{i,QI}}{w_{tot,Q}w_{tot,I}}$$

$$\sigma'_{UI} = \frac{\sum_i^N w_{i,U}w_{i,I}\sigma_{i,UI}}{w_{tot,U}w_{tot,I}}$$

$$\sigma'_{QU} = \frac{\sum_i^N w_{i,Q}w_{i,U}\sigma_{i,QU}}{w_{tot,Q}w_{tot,U}}$$

where w_i is the pixel weight and w_{tot} is the sum of the weights of all input pixels.

The output from this step is a single FITS file, containing a flux and error image for each of Stokes I, Q, and U, as well as the Stokes covariance images. An image mask is also produced, which represents how many input pixels went into each output pixel. Because of the weighting 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 with extension name MERGED 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:

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

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

The percent polarization (p) and its error are calculated as

$$p = 100\sqrt{\left(\frac{Q}{I}\right)^2 + \left(\frac{U}{I}\right)^2}$$

$$\sigma_p = \frac{100}{I}\sqrt{\frac{1}{(Q^2+U^2)}\Big[(Q\sigma_Q)^2+(U\sigma_U)^2+2QU\sigma_{QU}\Big]+\Big[\Big(\frac{Q}{I}\Big)^2+\Big(\frac{U}{I}\Big)^2\Big]\sigma_I^2-2\frac{Q}{I}\sigma_{QI}-2\frac{U}{I}\sigma_{UI}}.$$

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

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

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 19 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
- POL FLUX: image, extension 13
- ERROR POL FLUX: image, extension 14
- DEBIASED POL FLUX: image, extension 15
- MERGED DATA: table, extension 16
- POL DATA: table, extension 17
- FINAL POL DATA: table, extension 18

The final two extensions contain table representations of the polarization values for each pixel, as an alternate representation of the θ, p, and p' maps. The FINAL POL DATA table is a subset of the POL DATA table, with data quality cuts applied.

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.

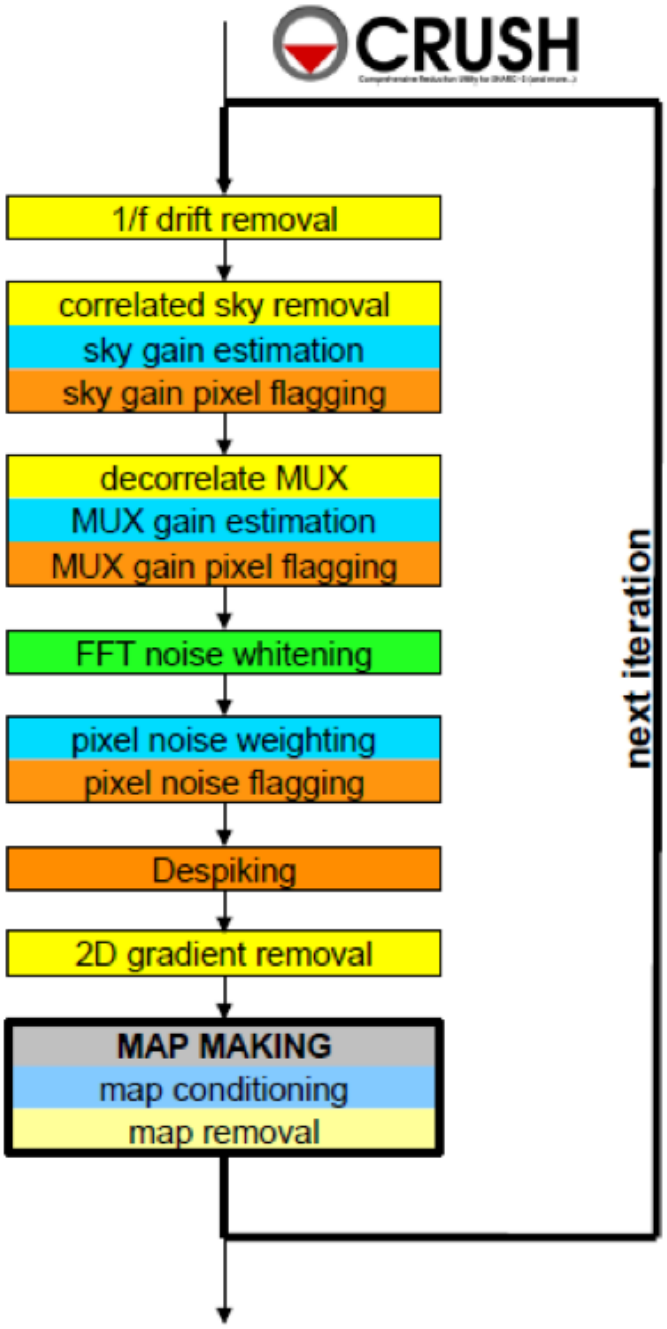

Figure 5: Scan data reduction flowchart

- S_{xy}: Actual 2D source flux at position $\{x, y\}$.
- n_{ct}: Essential limiting noise in channel c at time t.

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 as 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 the resources listed in 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.sigmyne.com/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 map 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_PMP_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 also has a minimal 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 *dmdall*, *wcs*, *calibrate*, and *polmap* products are saved. For Chop-Nod mode, the *dmdall*, *wcs*, and *calibrate* products are saved. For Scan mode, only the *crush* product is produced or saved.

For Nod-Pol data, the pipeline also generates two auxiliary products: a polarization map image in PNG format, with polarization vectors plotted over the Stokes I image, and a polarization vector file in DS9 region format, for displaying with FITS images. These products are alternate representations of the data in the FINAL POL DATA table in the polarization map (PMP) FITS file. They may be distributed to GOs separately from the FITS file products.

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

Step	Description	PRODTYPE	PROCSTAT	Identifier	Saved
Make Flat	Flat generated from Int.Cal file	obsflat	LEVEL_2	OFT	Y
Demodulate	Chops subtracted	dmdall	LEVEL_1	DMA	Y
Flat Correct	Flat field correction applied	flat	LEVEL_2	FLA	N
Align Arrays	R array shifted to T array	shift	LEVEL_2	SFT	N
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 IP	Instrumental polarization removed	ip	LEVEL_2	IPS	N
Rotate Coordinates	Polarization angle corrected to sky	rotate	LEVEL_2	ROT	N
Calibrate Flux	Flux calibrated to physical units	calibrate	LEVEL_3	CAL	Y
Subtract Background	Residual background removed	bgsubtract	LEVEL_3	BGS	N
Merge Images	Dithers merged to single map	merge	LEVEL_3	MRG	N
Compute Vectors	Polarization vectors calculated	polmap	LEVEL_4	PMP	Y

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

Step	Description	PRODTYPE	PROCSTAT	Identifier	Saved
Make Flat	Flat generated from Int.Cal file	obsflat	LEVEL_2	OFT	Y
Demodulate	Chops subtracted	dmdall	LEVEL_1	DMA	Y
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	N
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