HAWC Data Reduction Pipeline Users Manual

SCI-US-HBK-OP10-2008

Date: February 19, 2020
Revision: E

DFRC
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
SCI-US-HBK-OP10-2008

Prepared By:

Melanie Clarke, USRA, SOFIA DPS Development Lead	Date

APPROVAL:

Elizabeth Moore, USRA, SOFIA Associate Director for Software Systems	Date
William Vacca, USRA, SOFIA Deputy Associate Director for Science Ops	Date

REVISION HISTORY

REV	DATE	DESCRIPTION
-	03/07/17	Initial Release
A	04/26/17	Updates for HAWC DRP v1.1.0: updated data products table for new saved file defaults (Section 4); changed description of flat correction (Section 3.1.3); added ATRAN model to CRUSH options descriptions (Section a.5).
B	05/03/18	Updates for HAWC DRP v1.3.0
C	09/25/18	Updates for HAWC DRP v2.0.0
D	2/19/19	Updates for HAWC DRP v2.1.0 release: PIPEDEV-341, PIPEDEV-342, PIPEDEV-343, PIPEDEV-344, PIPEDEV-345, PIPEDEV-346, PIPEDEV-347
E	2/19/20	Update description of merge algorithm Update dependency descriptions for removal of C library Add new options for CRUSH step

HAWC+ DRP User's Manual

Release : SOF-US-HBK-OP10-2008 Rev. E

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

Feb 19, 2020

Contents

I Introduction **4**

II SI Observing Modes Supported **4**

1 HAWC+ Instrument Information **4**

2 HAWC+ Observing Modes **4**

III Algorithm Description **5**

3 Chop-Nod and Nod-Pol Reduction Algorithms **5**
- 3.1 Prepare . . . 5
- 3.2 Demodulate . . . 10
- 3.3 Flat Correct . . . 10
- 3.4 Align Arrays . . . 11
- 3.5 Split Images . . . 11
- 3.6 Combine Images . . . 11
- 3.7 Subtract Beams . . . 12
- 3.8 Compute Stokes . . . 12
- 3.9 Update WCS . . . 13
- 3.10 Subtract Instrumental Polarization . . . 13
- 3.11 Rotate Polarization Coordinates . . . 13
- 3.12 Correct for Atmospheric Opacity . . . 14
- 3.13 Calibrate Flux . . . 14
- 3.14 Subtract Background . . . 15
- 3.15 Merge Images . . . 15
- 3.16 Compute Vectors . . . 16

4 Scan Reduction Algorithms **17**
- 4.1 Signal Structure . . . 17
- 4.2 Sequential Incremental Modeling and Iterations . . . 19
- 4.3 DC Offset and 1/f Drift Removal . . . 19
- 4.4 Correlated Noise Removal and Gain Estimation . . . 20
- 4.5 Noise Weighting . . . 21

4.6 Despiking . . . 22
4.7 Spectral Conditioning . . . 22
4.8 Map Making . . . 23
4.9 Point-Source Flux Corrections . . . 24
4.10 CRUSH output . . . 25

5 Scan-Pol Reduction Algorithms **25**

6 Other Resources **26**

IV Data Products **26**

7 File names **26**

8 Data format **26**

9 Pipeline products **27**

V Grouping Level 0 Data for Processing **28**

VI Configuration and Execution **28**

10 Installation **29**
10.1 External Requirements . . . 29
10.2 Source Code Installation . . . 29

11 Configuration **30**

12 Input Data **31**
12.1 Auxiliary Files . . . 31

13 Redux Usage **33**
13.1 Automatic Mode Execution . . . 33
13.2 Manual Mode Execution . . . 33

14 Important Parameters **40**
14.1 DRP parameters . . . 40
14.2 CRUSH parameters . . . 44

VII Data Quality Assessment **44**

VIII Appendix: CRUSH execution **45**

15 Downloading and Installing CRUSH **46**

16 Optional Startup Environment and Java Configuration **47**

17 Running CRUSH **48**

18 Command-Line Options **48**

19 CRUSH News, Feedback, and Bug Reports **49**

IX Appendix: Sample Configuration Files **49**

20 Full DRP Configuration File **49**

21 DRP Override Configuration File **63**

22 Full CRUSH Configuration File **63**

23 HAWC+ CRUSH Configuration File **75**

X Appendix: Required Header Keywords **82**

Part I

Introduction

The SI Pipeline User's Manual (OP10) is intended for use by both SOFIA Science Center staff during routine data processing and analysis, and also as a reference for Guest Observers (GOs) 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 data reduction 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 2.4.0.

Part II

SI Observing Modes Supported

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 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. Likewise, polarization observations may be taken in either Nod-Pol or Scan-Pol mode. Nod-Pol mode includes chopping and nodding cycles in multiple HWP positions; Scan-Pol mode includes repeated scans at 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.

Part III

Algorithm Description

The data reduction pipeline for HAWC has two main branches of development: the HAWC DRP provides the chop/nod mode reduction algorithms, polarimetry calculations, and the calling structure for all pipeline steps. Scan mode reduction algorithms are provided by a standalone package, called CRUSH, that may be called from the DRP for either Scan imaging or Scan-Pol polarimetry.

3 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 (Fig. 1 and Fig. 2) and Chop-Nod data (Fig. 3 and Fig. 4).

3.1 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 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

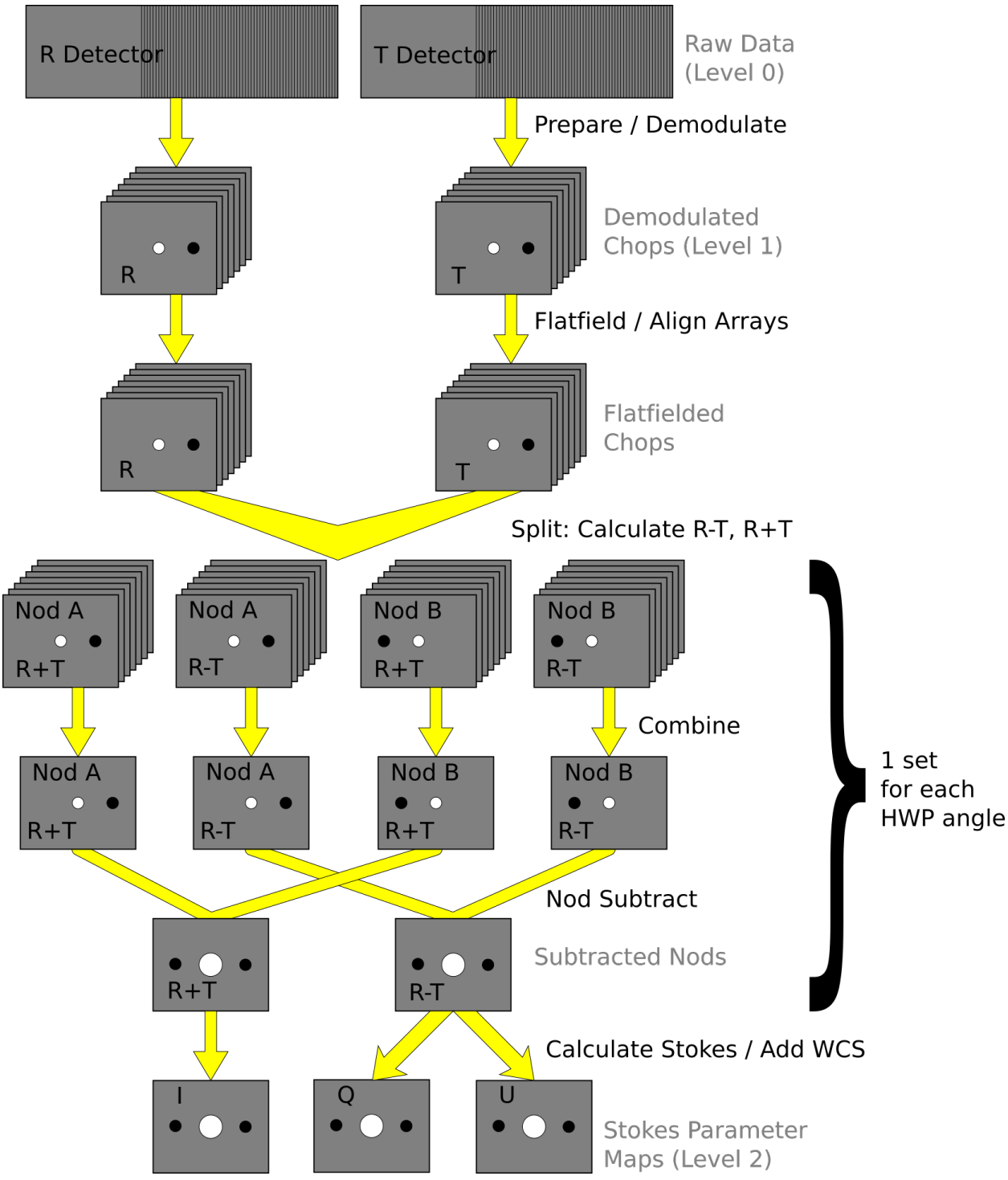

Fig. 1: Nod-Pol data reduction flowchart, up through Stokes parameter calculation for a single input file.

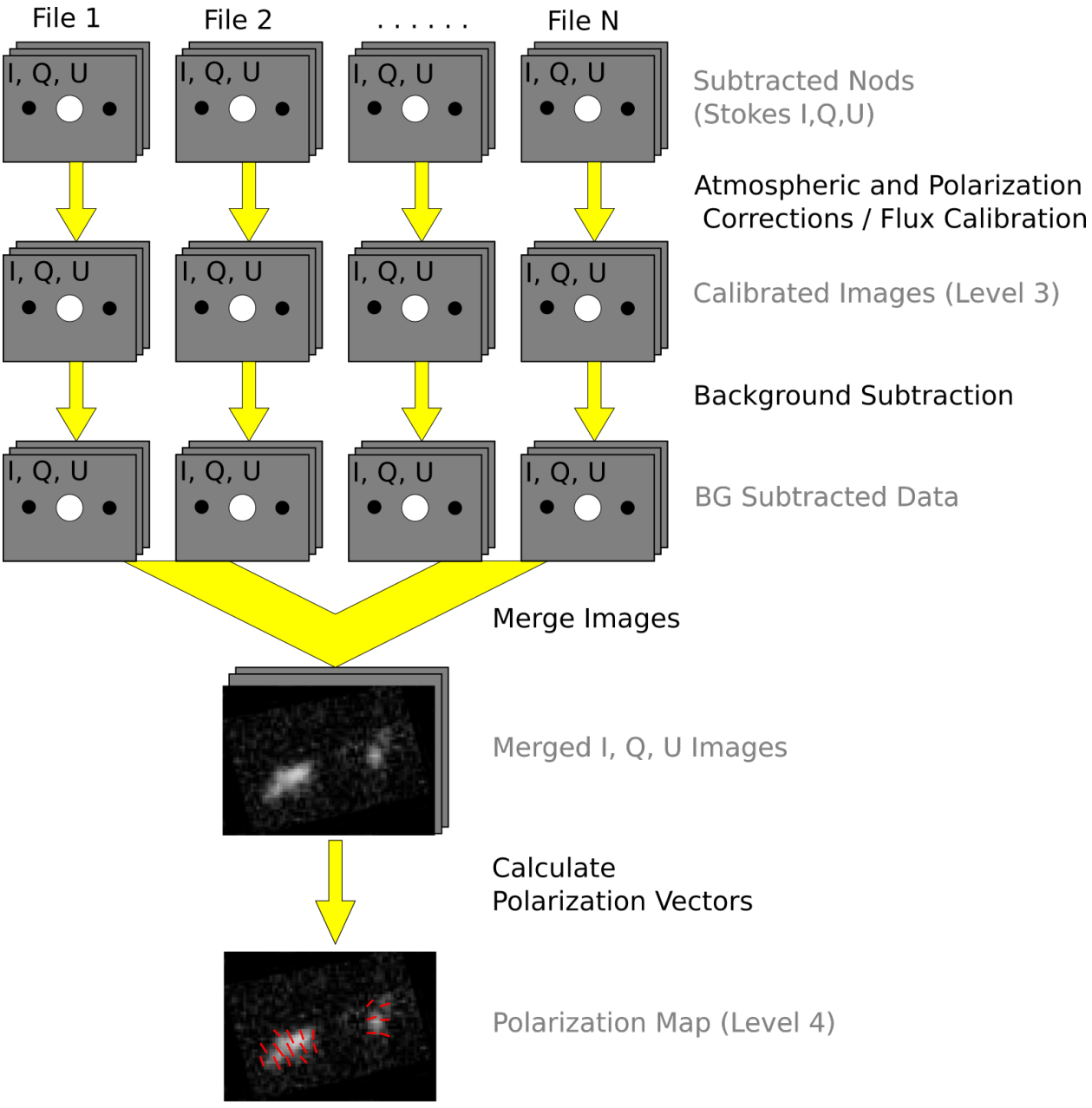

Fig. 2: Nod-Pol data reduction flowchart, picking up from Stokes parameter calculation, through combining multiple input files and calculating polarization vectors.

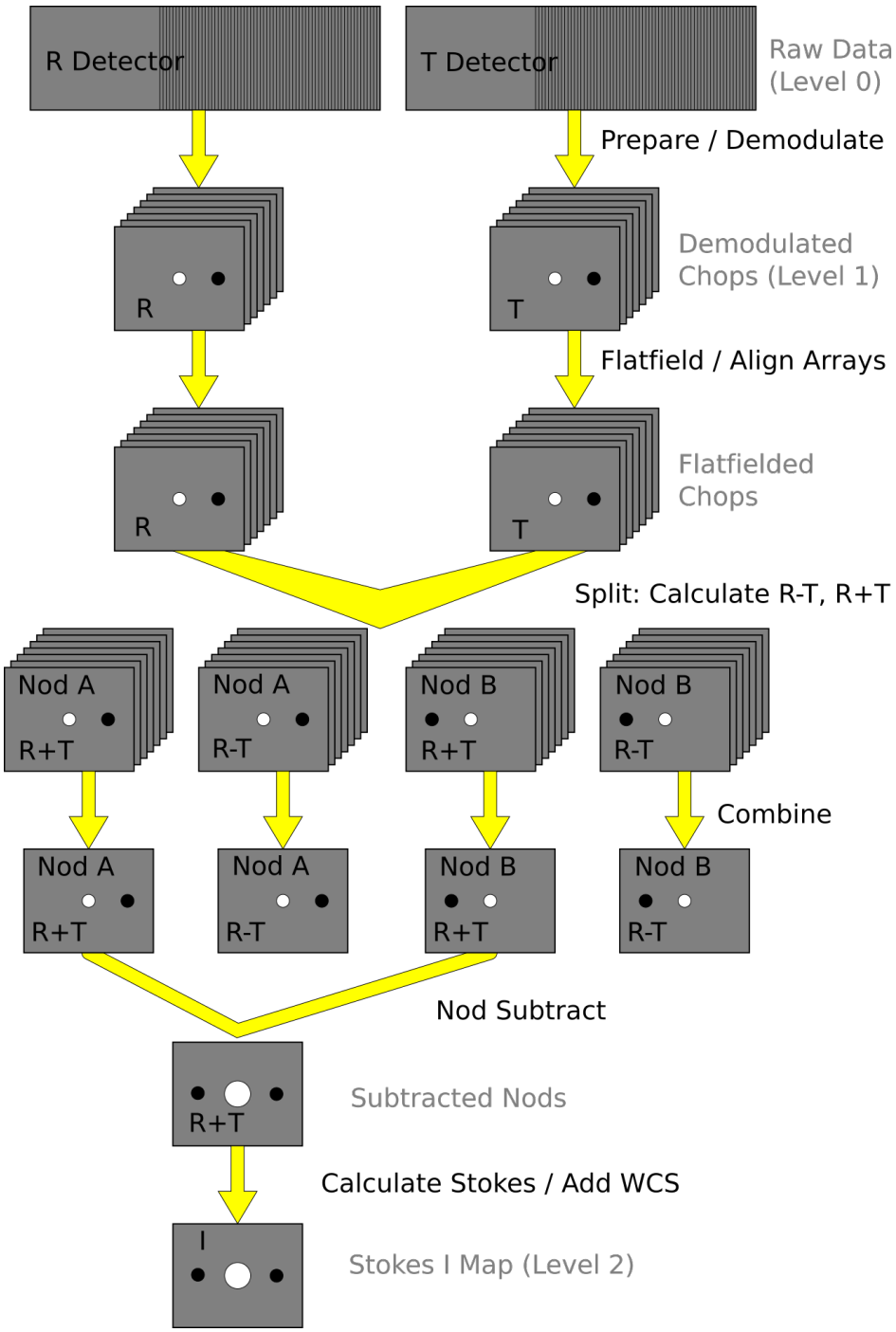

Fig. 3: Chop-Nod data reduction flowchart, up through Stokes parameter calculation for a single input file.

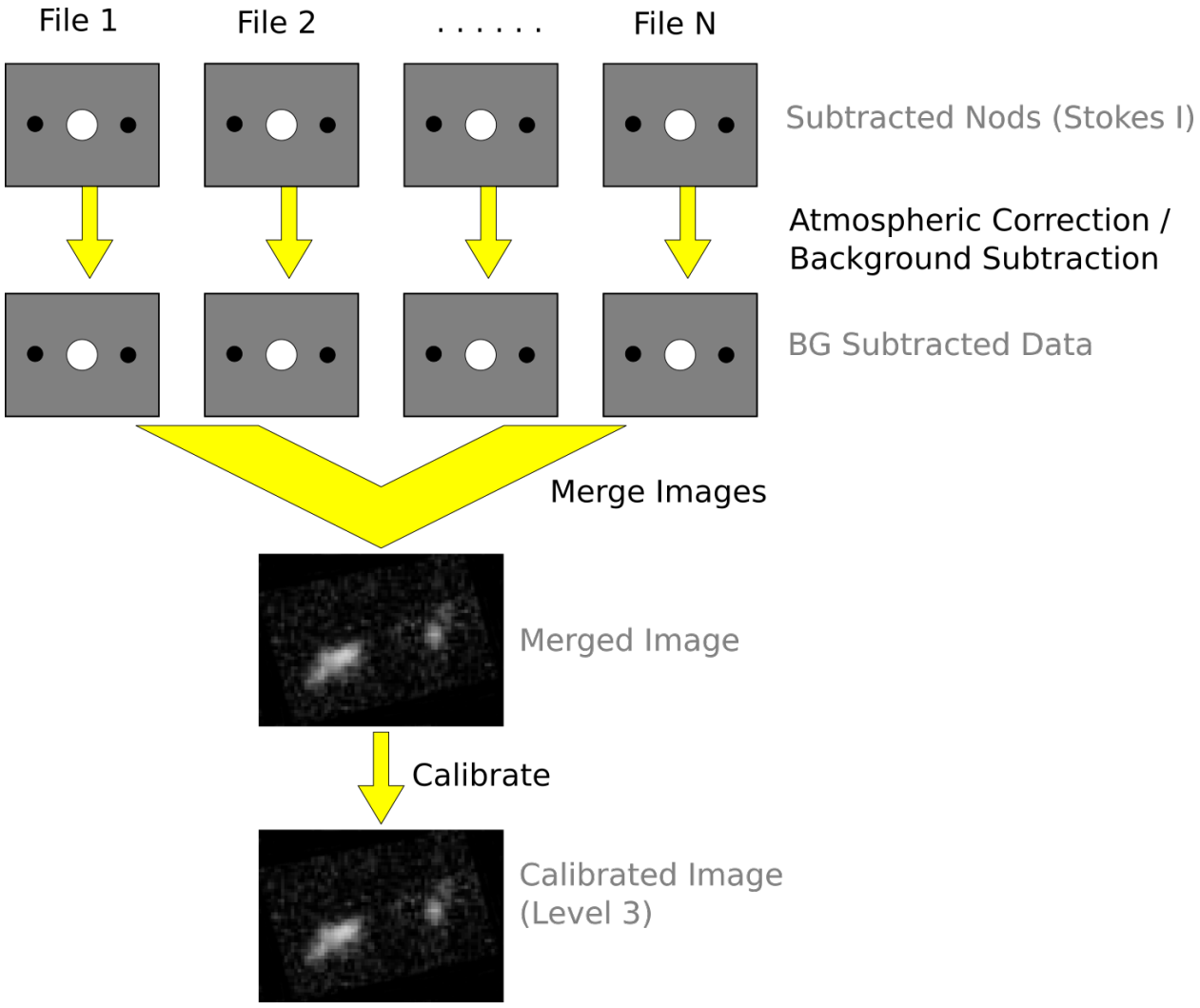

Fig. 4: Chop-Nod data reduction flowchart, picking up from Stokes parameter calculation, through combining multiple input files.

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.

3.2 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.

3.3 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.

3.4 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. The output images are 64x40xN$_{frame}$.

3.5 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.

3.6 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.

3.7 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.

3.8 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).

3.9 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.

3.10 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 $U/I < 0.6\%$.

3.11 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:

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

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

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

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

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

3.12 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.

3.13 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.

Raw calibration factors are computed as above by the pipeline, for any observation marked as a flux standard (OBSTYPE=STANDARD_FLUX), and are stored in the FITS headers of the output data product. The instrument team generally combines these factors across a flight series, to determine a robust average value for each instrument configuration and mode. The overall calibration thus determined is expected to be good to within about 10%.

For science observations, the series-average calibration factor is directly applied to the flux in each of the Stokes I, Q, and U images, and to their associated error and covariance images:

$$I' = I/f$$

$$Q' = Q/f$$

$$U' = U/f$$

$$\sigma'_Q = \sigma_Q/f$$

$$\sigma'_U = \sigma_Q/f$$

$$\sigma'_{QI} = \sigma_{QI}/f^2$$

$$\sigma'_{UI} = \sigma_{UI}/f^2$$

$$\sigma'_{QU} = \sigma_{QU}/f^2.$$

where f is the reference calibration factor. The systematic error on f is not propagated into the error planes, but it is stored in the ERRCALF FITS header keyword. The calibration factor applied is stored in the CALFCTR keyword.

Note that for Chop-Nod imaging data, this factor is applied after the merge step, below.

3.14 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 for each Stokes image as follows:

$$I' = (I - a_I)/b$$

$$Q' = (Q - a_Q)/b$$

$$U' = (U - a_U)/b$$

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

3.15 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.

As of HAWC DRP v2.4.0, the distance-weighted input pixels within the fit radius may optionally be fit by a low-order polynomial surface, rather than a weighted average. In this case, each output pixel value is the value of the local polynomial fit, evaluated at that grid location. Errors and covariances are propagated similarly.

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.

3.16 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:

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

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

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

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

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

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.

4 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.

4.1 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:

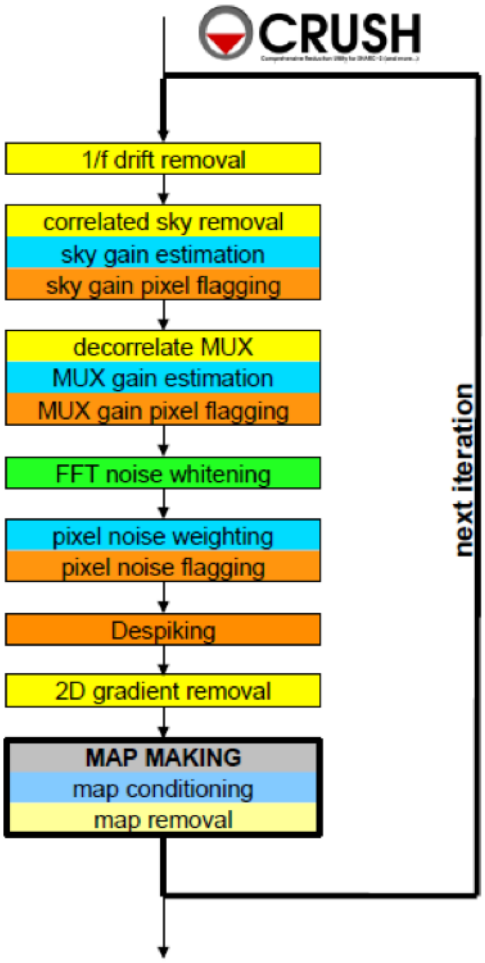

Fig. 5: Scan data reduction flowchart

- 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.

4.2 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.

4.3 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.

By default, CRUSH also clips extreme scanning velocities using, by default, a set minimum and maximum value for each instrument. The default settings still include a broad range of speeds, so images can sometimes be distorted by low or high speeds causing too little or too much exposure on single pixels. To fix this, CRUSH can optionally sigma-clip the telescope speeds to a tighter range. The default sigma multiplier is five, but the value is configurable.

4.4 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:

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

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

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

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:

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

which has a parameter dependence:

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

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.

4.5 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 [specfilt]), 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.

4.6 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.

4.7 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.

4.8 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 $S^{(0)}_{xy}$ 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.

4.9 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 [resources].

4.10 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 [prepare], 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.

5 Scan-Pol Reduction Algorithms

Scanning polarimetry reductions are a hybrid of the the Nod-Pol and Scan reduction algorithms, described above.

In scanning polarimetry mode, each input file is a scan taken at a single HWP angle. The standard HWP angles are cycled through in a set of observations. Given a set of scanning polarimetry files, the pipeline sorts them into groups, with 4 HWP angles per group, and runs the CRUSH pipeline on each set, generating separate images for the R0 and T0 subarrays. The output from CRUSH for a standard set of four HWP angles is a set of eight images, one for each of R0 and T0 at each angle.

The DRP pipeline picks up these images, and uses them to calculate Stokes parameters. The individual images are registered to each other, using their recorded WCS to determine the relative shifts between the generated images. After that, the R and T arrays are directly added and subtracted at each HWP angle, and combined as described above to generate Stokes I, Q, and U images (the *Compute Stokes* step). The output data format is the same as for the *stokes* product for the Nod-Pol pipeline.

After Stokes calculation, the following steps are also performed, in the way described above for the Nod-Pol pipeline:

- *Subtract Instrumental Polarization*
- *Rotate Polarization Coordinates*
- *Correct for Atmospheric Opacity*
- *Merge Images*
- *Compute Vectors*

Note that the CRUSH pipeline performs opacity and background level corrections, and resamples data into sky coordinates with full WCS corrections, as part of its standard processing, so these steps from the Nod-Pol pipeline are not applied.

The final output product is a polarization map, the same as is produced by the Nod-Pol pipeline.

6 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/

Part IV
Data Products

7 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.

8 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.

9 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 *demodulate*, *opacity*, *calibrate*, *merge*, and *polmap* products are saved. For Chop-Nod mode, the *demodulate*, *opacity*, *merge*, and *calibrate* products are saved. For Scan mode, the *crush* and *calibrate* products are saved. For Scan-Pol mode, the *crush*, *calibrate*, *merge*, and *polmap* products are 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	PROD-TYPE	PROC-STAT	Identi-fier	Saved
Make Flat	Flat generated from Int. Cal file	obsflat	LEVEL_2	OFT	Y
Demodulate	Chops subtracted	demodulate	LEVEL_1	DMD	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	N
Subtract IP	Instrumental polarization removed	ip	LEVEL_2	IPS	N
Rotate Coordinates	Polarization angle corrected to sky	rotate	LEVEL_2	ROT	N
Correct Opacity	Corrected for atmospheric opacity	opacity	LEVEL_2	OPC	Y
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 a single map	merge	LEVEL_3	MRG	Y
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	demodulate	LEVEL_1	DMD	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	N
Correct Opacity	Corrected for atmospheric opacity	opacity	LEVEL_2	OPC	Y
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 intermediate and final pipeline data products

Step	Description	PRODTYPE	PROCSTAT	Identifier	Saved
CRUSH	Source model derived by CRUSH	crush	LEVEL_2	CRH	Y
Calibrate Flux	Flux calibrated to physical units	calibrate	LEVEL_3	CAL	Y

Table 4: Scan-Pol mode intermediate and final pipeline data products

Step	Description	PRODTYPE	PROCSTAT	Identifier	Saved
CRUSH	Source model derived by CRUSH	crush	LEVEL_2	CRH	Y
Compute Stokes	Stokes parameters calculated	stokes	LEVEL_2	STK	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
Merge Images	HWP sets merged to single map	merge	LEVEL_3	MRG	Y
Compute Vectors	Polarization vectors calculated	polmap	LEVEL_4	PMP	Y

Part V

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 5: Grouping Criteria for Imaging and Polarimetry Modes

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

Part VI

Configuration and Execution

10 Installation

The HAWC pipeline is written in Python with an additional external package, written in Java. The pipeline is platform independent and has been tested on Windows, Linux, and Mac operating systems. Running the pipeline requires a minimum of 16GB RAM, or equivalent-sized swap file.

The pipeline is comprised of three separate software packages: HAWC DRP (Python), CRUSH (Java), and Redux (Python). The DRP and CRUSH packages provide the data processing algorithms. Redux is a data reduction interface package that provides interactive and command-line interfaces to the pipeline algorithms.

10.1 External Requirements

To run the pipeline for any mode from the Redux interface, Python 3.6 or higher is required, as well as the following packages: numpy, scipy, matplotlib, astropy, configobj, and APLpy. Some display functions for the graphical user interface (GUI) additionally require the PyQt5, pyds9, pandas, photutils, and dill packages. All required external packages are available to install via the pip or conda package managers.

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

Running the pipeline interactively also requires an installation of SAO DS9 for FITS image display. See http://ds9.si.edu/ for download and installation instructions. The *ds9* executable must be available in the PATH environment variable for the pyds9 interface to be able to find and control it.

10.2 Source Code Installation

The source code for the HAWC pipeline maintained by the SOFIA Data Processing Systems (DPS) team can be obtained directly from the *hawc*, *pyredux*, *pypecal*, *pypeutils*, and *crush* git repositories there. These repositories contains all needed configuration files, auxiliary files, and Python 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 GitHub page. See *Appendix: CRUSH execution* for more information on running CRUSH directly.

After obtaining the source code, install the four python libraries with the command:

```
python setup.py install
```

from the *hawc*, *pyredux*, *pypecal*, and *pypeutils* directories. A pre-built cross-platform distribution of the CRUSH package is packaged with the *hawc* repository; it does not need any further installation steps.

After installation, the top-level pipeline interface commands should be available in the PATH. Typing

```
redux
```

from the command line should launch the GUI interface, and

```
redux_pipe -h
```

should display a brief help message for the command line interface.

11 Configuration

The DRP pipeline requires a valid and complete configuration file to run. Configuration files are written in plain text, in the INI 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:

- 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 section for each pipeline step defined)

The pipeline is usually run with a default configuration file (*hawc/pipeline/config/pipeconf.cfg*), which defines all standard reduction steps and default parameters. It may be overridden with date-specific default values, defined in (*hawc/pipeline/config/date_overrides/*), or with user-defined parameters. Override configuration files may contain any subset of the values in the full configuration file. See *Appendix: Sample Configuration Files* for examples of override configuration files as well as the full default file.

The CRUSH pipeline, run as a single pipeline step for Scan and Scan-Pol 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/config/default.cfg*
2. Global user overrides from *~/.crush2/default.cfg*
3. Instrument overrides from *crush/config/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 available online). Common configuration options for HAWC+ are discussed in *README.hawc+* (also included in the distribution, and available online). Yet more useful options are covered by the main *README*. Finally, a complete list of all available options and their descriptions is to be found in the *GLOSSARY*. See *Appendix: Sample Configuration Files* 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 for the *crush* step:

```
options = '-faint'
```

12 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 INI format (see *Appendix: Required Header Keywords*).

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. The sole exception is that internal calibrator files (CALMODE=INT_CAL) may be loaded with their corresponding science files. They will be reduced separately first, in order to produce flat fields used in the science reduction.

12.1 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 6: Auxiliary files used by DRP reductions for Chop-Nod and Nod-Pol data

Auxiliary File	File Type	Pipe Step	Comments
Jump Map	FITS	Flux Jump	Contains jump correction values per pixel
Phase	FITS	Demodulate	Contains phase delay in seconds for each pixel
Reference Phase	FITS	Demod. Plots	Contains reference phase angles for comparison with the current observation
Sky Cal	FITS	Make Flat	Contains a master sky flat for use in generating flats from INT_CALs
Flat	FITS	Flat Correct	Contains a back-up flat field, used if INT_CAL files are not available
IP	FITS	Instrumental Polarization	Contains q and u correction factors by pixel and band

The jump map is used in a preparatory step before the pipeline begins processing to correct raw values for a residual electronic effect that results in discontinuous jumps in flux values. It is a FITS image that matches the dimensions of the raw flux values (128 x 41 pixels). Pixels for which flux jump corrections may be required have integer values greater than zero. Pixels for which there are no corrections necessary are zero-valued.

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 reference phase file used in the Demod Plots step is used for diagnostic purposes only: it specifies a baseline set of phase angle values, for use in judging the quality of internal calibrator files.

Normally, the pipeline generates the flat fields used in the Flat Correct step from internal calibrator (INT_CAL) files taken alongside the data. To do so, the Make Flats step uses a Sky Cal reference file, which has 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 (64 x 41 pixels). The Gain images should contain multiplicative floating-point flat correction factors. 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 filter to the input demodulated files.

When INT_CAL files are not available, the Flat Correct step may use a back-up flat file. This file should have the same format as the Sky Cal file, but the R Array Gain and T Array Gain values should be suitable for direct multiplication with the flux values in the Flat Correct step. There should be one back-up flat file available for each filter passband.

In addition to these files, stored with the DRP code, the pipeline requires several additional auxiliary files to perform flux calibration. These are tracked in the *pipecal* package, used to support SOFIA flux calibration for several instruments, including HAWC. The required files include response coefficient tables, used to correct for atmospheric opacity, and reference calibration factor tables, used to calibrate to physical units.

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 #.

Calibration factors are also stored in ASCII format, and list the correction factor by mode and HAWC filter band, to be applied to opacity-corrected data.

Some additional auxiliary files are used in reductions of flux standards, to assist in deriving the flux calibration factors applied to science observations. These include filter definition tables and standard flux tables, by date and source.

Table 7: Auxiliary files used for calibration (all modes)

Auxiliary File	File Type	Pipe Step	Comments
Response	ASCII	Opacity Correct	Contains instrumental response coefficients by altitude, ZA
Calibration Factor	ASCII	Calibrate	Contains reference calibration factors by filter band, mode
Filter definition	ASCII	Standard Photometry	Contains filter wavelength band and standard aperture definitions
Standard flux	ASCII	Standard Photometry	Contains reference flux values for a known source, by filter band

13 Redux Usage

Redux usage is documented in the `redux` package.

13.1 Automatic Mode Execution

The DPS pipeline infrastructure runs a 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). The command-line interface to the pipeline is run as:

```
redux_pipe infiles.txt
```

The command-line interface will read in the specified input files, use their headers to determine the observation mode, and accordingly the steps to run and any intermediate files to save. Output files are written to the current directory, from which the pipeline was called. After reduction is complete, the script will generate an output manifest (*outfiles.txt*) containing the relative paths to all output FITS files generated by the pipeline.

Optionally, in place of a manifest file, file paths to input files may be directly specified on the command line. Input files may be raw FITS files, or may be intermediate products previously produced by the pipeline. For example, this command will complete the reduction for a set of FITS files in the current directory, previously reduced through the calibration step of the pipeline:

```
redux_pipe *CAL*.fits
```

To customize batch reductions from the command line, the *redux_pipe* interface accepts a configuration file on the command line. This file may contain any subset of the full configuration file, specifying any non-default parameters for pipeline steps. An output directory for pipeline products and the terminal log level may also be set on the command line.

The full set of optional command-line parameters accepted by the *redux_pipe* interface are:

```
-h, --help            show this help message and exit
-c CONFIG, --configuration CONFIG
                      Path to Redux configuration file.
-o OUTDIR, --out OUTDIR
                      Path to output directory.
-l LOGLEVEL, --loglevel LOGLEVEL
                      Log level.
```

13.2 Manual Mode Execution

In manual mode, the pipeline may be run interactively, via a graphical user interface (GUI) provided by the Redux package. The GUI is launched by the command:

```
redux
```

entered at the terminal prompt (Fig. 6). The GUI allows output directory specification, but it may write initial or temporary files to the current directory, so it is recommended to start the interface from a location to which the user has write privileges.

From the command line, the *redux* interface accepts an optional config file (*-c*) or log level specification (*-l*), in the same way the *redux_pipe* command does. Any pipeline parameters provided to the interface in a configuration file will be used to set default values; they will still be editable from the GUI.

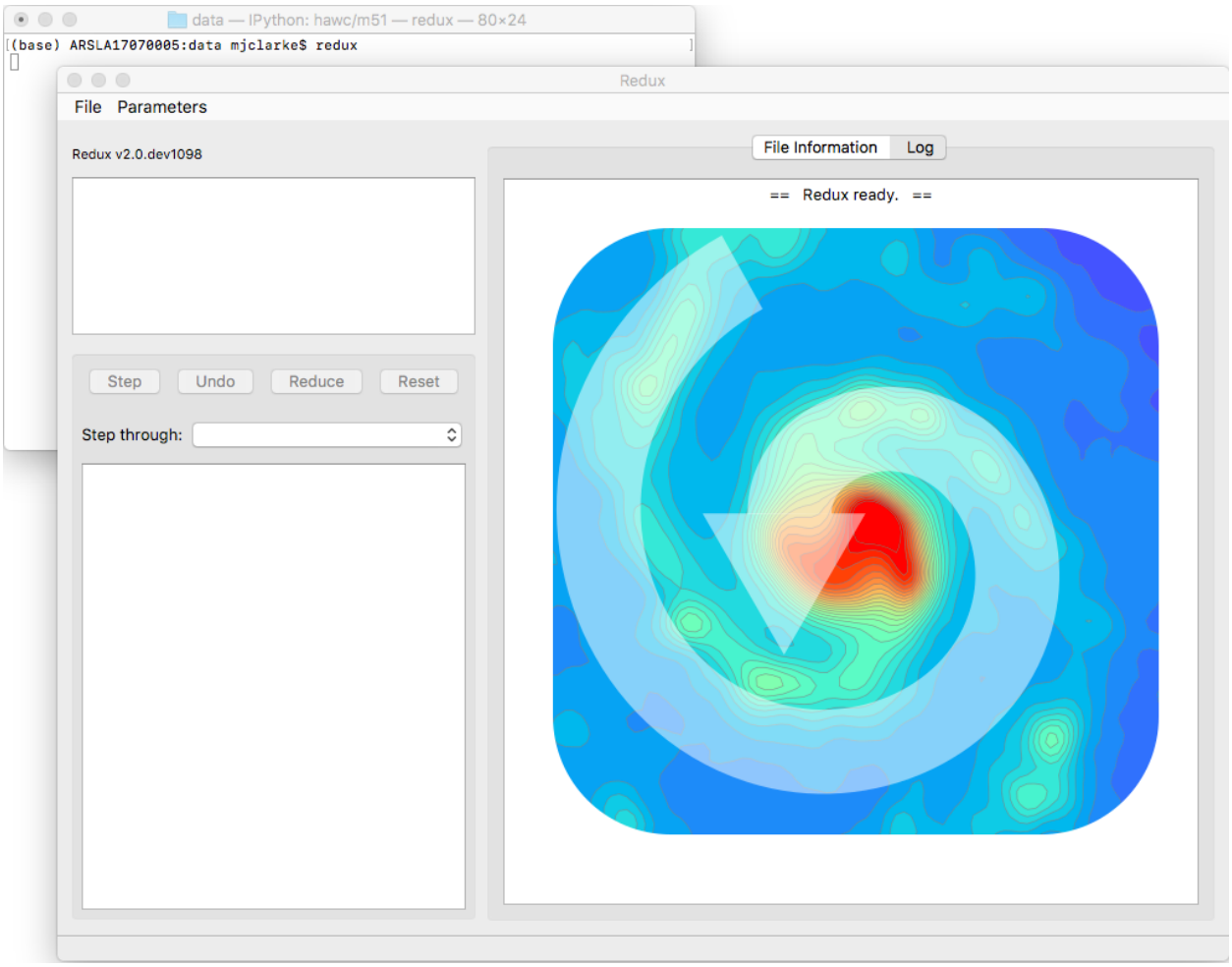

Fig. 6: Redux GUI startup.

Basic Workflow

To start an interactive reduction, select a set of input files, using the File menu (**File->Open New Reduction**). This will bring up a file dialog window (see Fig. 7). All files selected will be reduced together as a single reduction set.

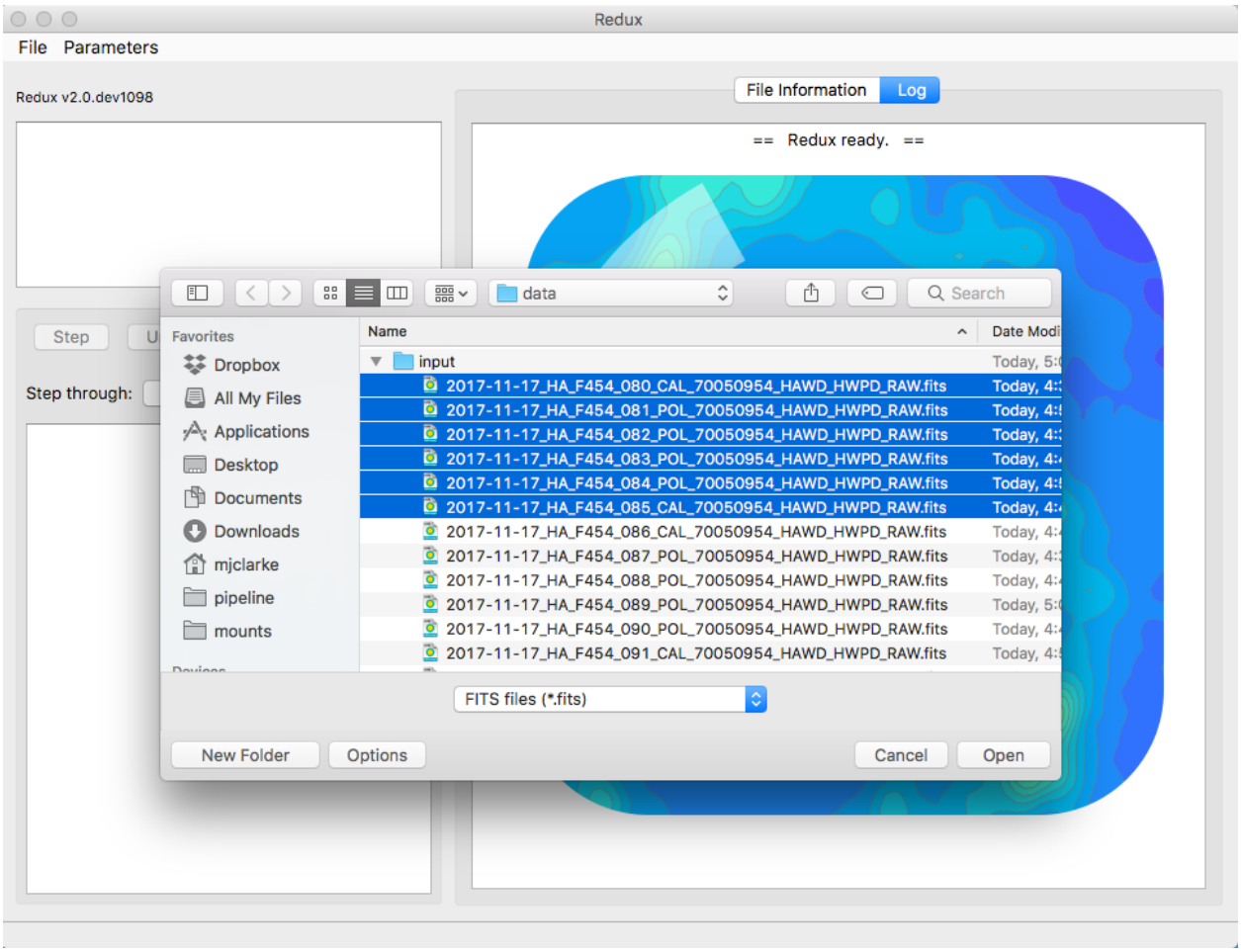

Fig. 7: Open new reduction.

Redux will decide the appropriate reduction steps from the input files, and load them into the GUI, as in Fig. 8.

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** button next to the step name to bring up the parameter editor for that step (Fig. 9). Within the parameter editor, all values may be edited. Click **OK** to save the edited values and close the window. Click **Reset** to restore any edited values to their last saved values. Click **Restore Defaults** to reset all values to their stored defaults. Click **Cancel** to 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 through:** 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

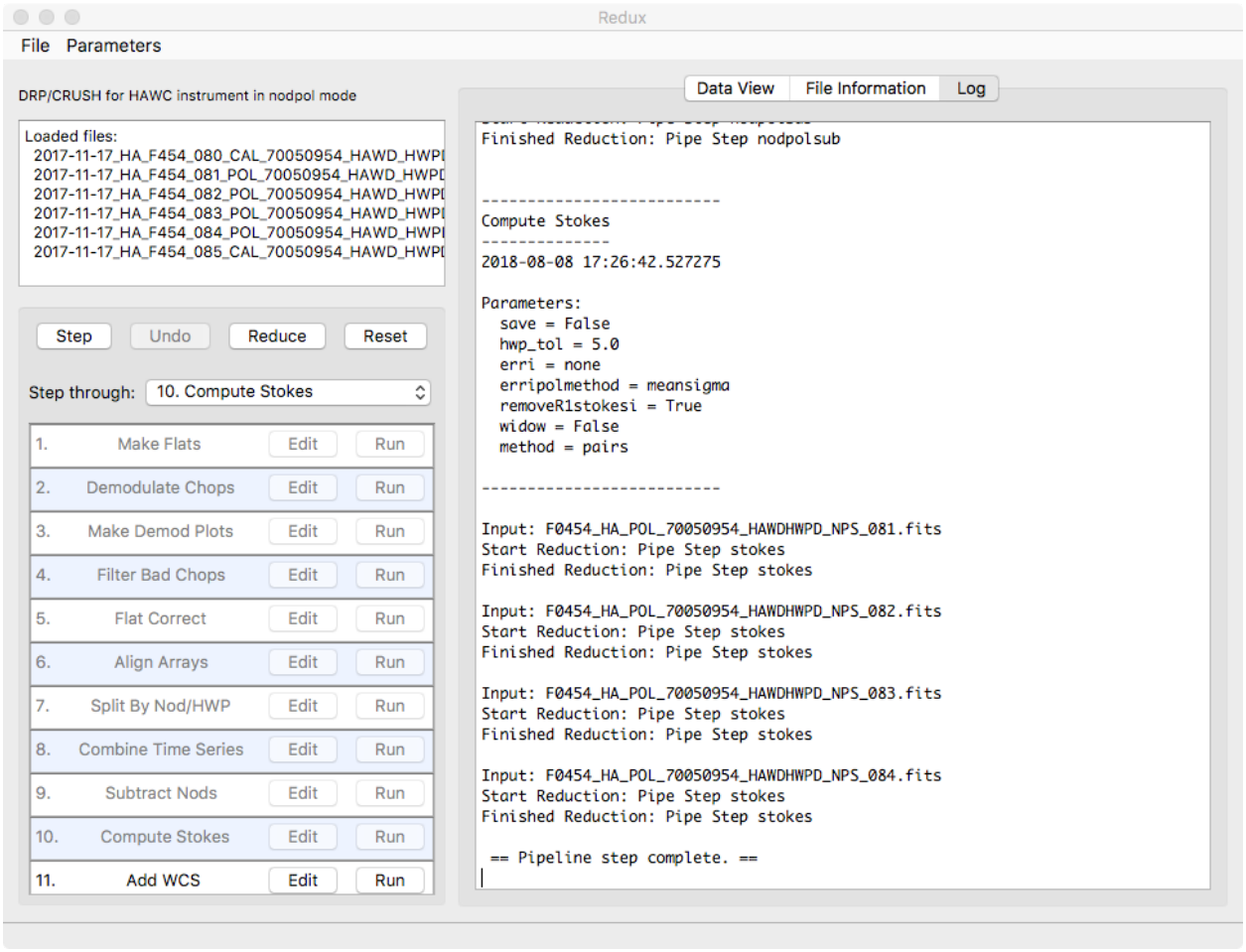

Fig. 8: Sample reduction steps. Log output from the pipeline is displayed in the **Log** tab.

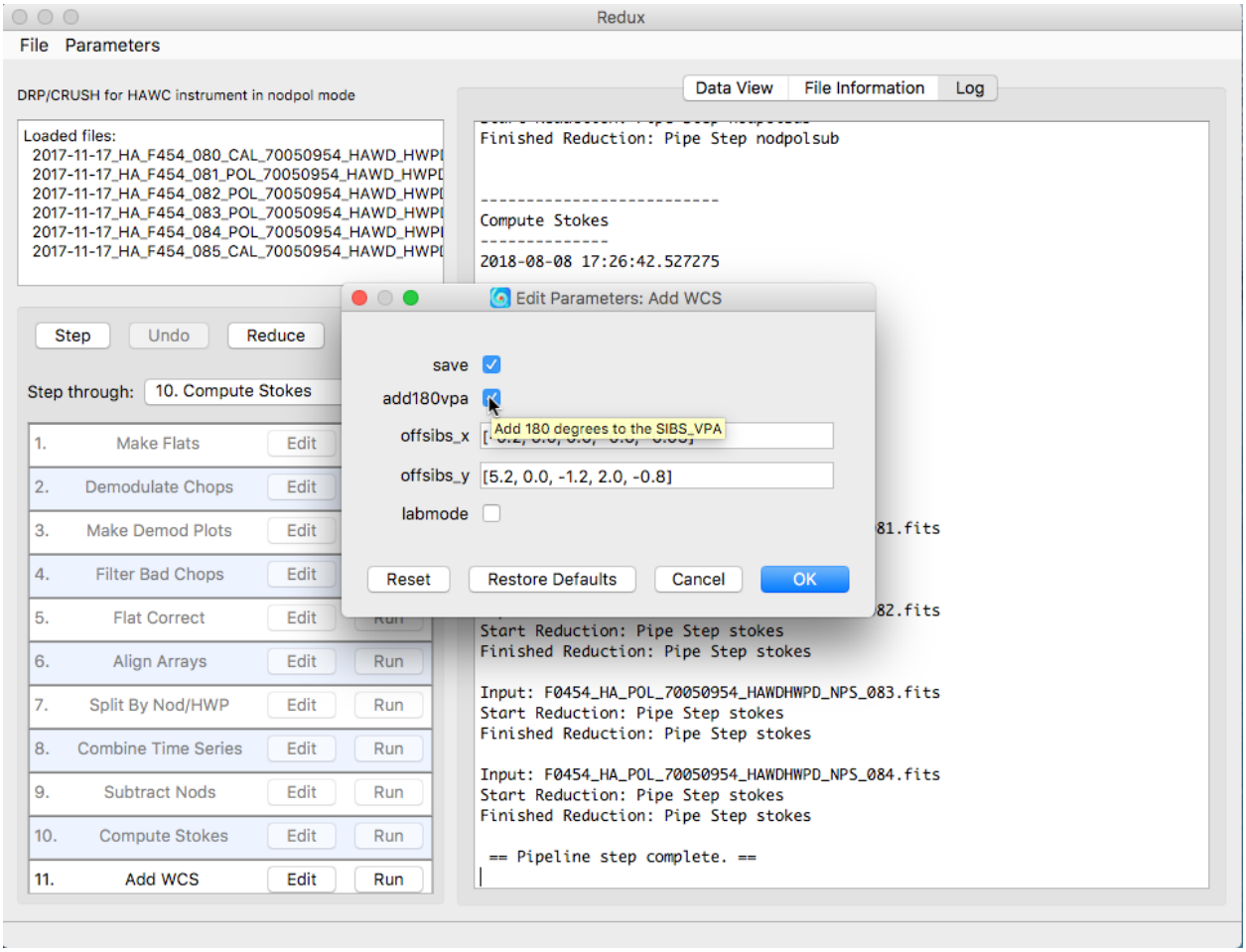

Fig. 9: Sample parameter editor for a pipeline step.

by clicking **Reduce**. After each step, the results of the processing may be displayed in a data viewer. After running a pipeline step or reduction, click **Reset** to restore the reduction to the initial state, without resetting parameter values.

Files can be added to the reduction set (**File -> Add Files**) or removed from the reduction set (**File -> Remove Files**), but either action will reset the reduction for all loaded files. Select the **File Information** tab to display a table of information about the currently loaded files (Fig. 10).

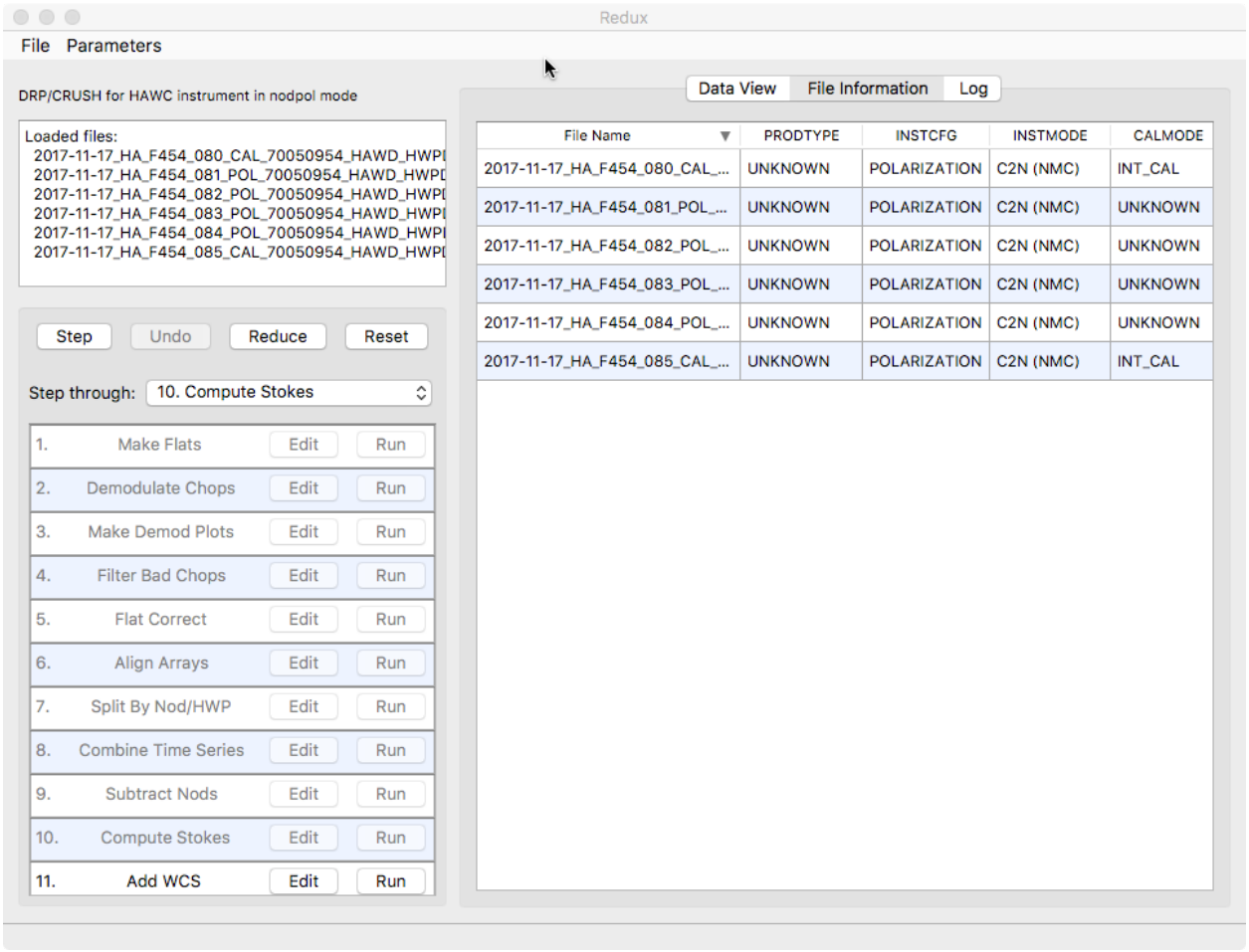

Fig. 10: File information table.

Display Features

The Redux GUI displays images for quality analysis and display (QAD) in the DS9 FITS viewer. DS9 is a standalone image display tool with an extensive feature set. See the SAO DS9 site (http://ds9.si.edu/) for more usage information.

After each pipeline step completes, Redux may load the produced images into DS9. Some display options may be customized directly in DS9; some commonly used options are accessible from the Redux interface, in the **Data View** tab (Fig. 11).

From the Redux interface, the **Display Settings** can be used to:

- Set the FITS extension to display (**First**, or edit to enter a specific extension), or specify that all extensions should be displayed in a cube or in separate frames.
- Lock individual frames together, in image or WCS coordinates.

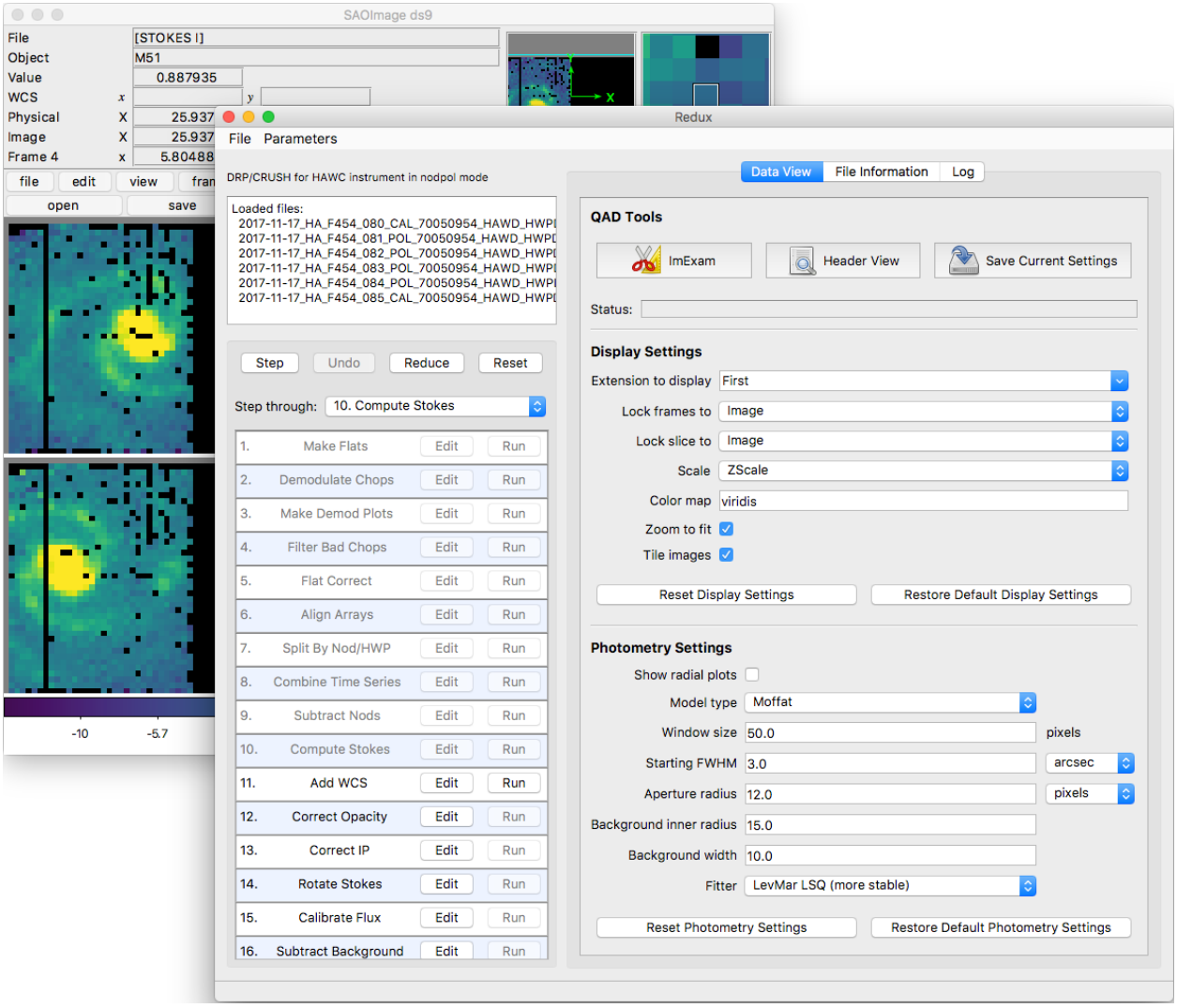

Fig. 11: Data viewer settings and tools.

- Lock cube slices for separate frames together, in image or WCS coordinates.
- Set the image scaling scheme.
- Set a default color map.
- Zoom to fit image after loading.
- Tile image frames, rather than displaying a single frame at a time.

Changing any of these options in the Data View tab will cause the currently displayed data to be reloaded, with the new options. Clicking **Reset Display Settings** will revert any edited options to the last saved values. Clicking **Restore Default Display Settings** will revert all options to their default values.

In the **QAD Tools** section of the **Data View** tab, there are several additional tools available.

Clicking the **ImExam** button (scissors icon) launches an event loop in DS9. After launching it, bring the DS9 window forward, then type 'a' over a source in the image to perform photometry at that location. Typing 'c' will clear any previous results and 'q' will quit the ImExam loop. The photometry settings (the image window considered, the model fit, the aperture sizes, etc.) may be customized in the **Photometry Settings**. After modifying these settings, they will take efffect only for new apertures (use 'c' to clear old ones first). As for the display settings, **Reset Photometry Settings** will revert to the last saved values and **Restore Default Photometry Settings** will revert to default values.

Clicking the **Header** button (magnifying glass icon) from the **QAD Tools** section opens a new window that displays headers from currently loaded FITS files in text form (Fig. 12). The extensions displayed depends on the extension setting selected (in **Extension to Display**). If a particular extension is selected, only that header will be displayed. If all extensions are selected (either for cube or multi-frame display), all extension headers will be displayed. The buttons at the bottom of the window may be used to find or filter the header text, or generate a table of header keywords. For filter or table display, a comma-separated list of keys may be entered in the text box.

Clicking the **Save Current Settings** button (disk icon) from the **QAD Tools** section saves all current display and photometry settings for the current user. This allows the user's settings to persist across new Redux reductions, and to be loaded when Redux next starts up.

14 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. All parameters listed are accessible and editable from the Redux GUI interface as well.

14.1 DRP parameters

Below are the most important parameters for Chop-Nod, Nod-Pol, and Scan-Pol 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 attempt to continue reduction despite incorrect header keywords. Default is True.
- **demodulate**
  - *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 (in seconds of delay). Default is typically a file in *hawc/pipeline/data/phasefiles*.

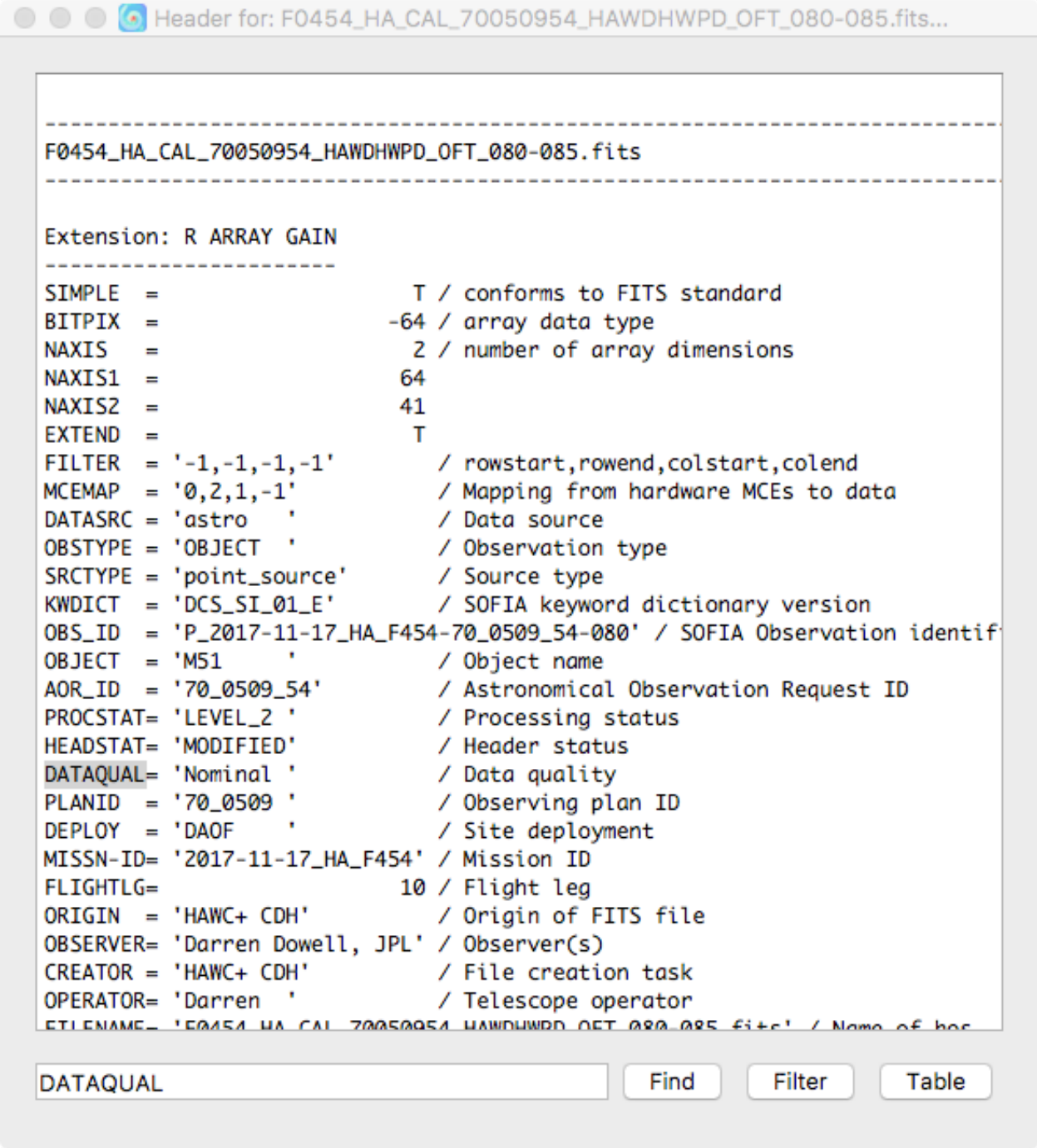

Fig. 12: QAD FITS header viewer.

  - *phaseoffset*: Set to a floating point number to apply an offset to the specified phase file. The value should be specified in degrees: it is usually determined from the median offset reported by the demodulation plots for the INT_CAL files. Default is 0.0.
  - *track_tol*: If non-negative, the pipeline step will use this number as the tracking tolerance in arcseconds. Samples with tracking deviation larger than this number will be rejected. If set to 'beam', the beam size for the filter band will be used. If set to 'centroidexp', the CentroidExpMsec data stream will be used to flag data, rather than the TrackErrAoi3/4 data stream. Set to -1 to turn off tracking rejection entirely. Default is centroidexp.
  - *data_sigma*: Sigma threshold for clipping data means; used in calculating variance. Default is 5.0.

- **flat**
  - *flatfile*: Set to a file glob to identify flats to use in processing. Default is "flats/*OFT*.fits".
  - *flatfitkeys*: Header keywords to match between data and flat fields. Default is "`SPECTEL1', 'MISSN-ID', 'FILEGPID', 'SCRIPTID'".
  - *bkupflat*: File glob specifying back-up flats in case *flatfile* does not exist. Default is "$DPS_HAWCPIPE/pipeline/data/flats/*OFT.fits".

- **split**
  - *rtarrays*: Set to 'RT' to use both R and T arrays, 'R' for R only, or 'T' for T only. Default is 'RT'.
  - *nod_tol*: Percent difference between the number of chop cycles in each nod position that will be tolerated for continuing the reduction. Set higher to reject fewer data files. Default is 50.0.

- **combine**
  - *sigma*: Reject outliers more than this many sigma from the mean. Default is 3.0.
  - *sum_sigma*: Reject additional outliers in R+T more than this many sigma from the mean. Default is 4.0.

- **stokes**
  - *erri*: Method for inflating errors in I from standard deviation across HWP angles. Can be median, mean, or none. Default is none.
  - *removeR1stokesi*: Set to False to keep the R1 array in the Stokes I image. Default is True.

- **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.

- **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*.
  - *fileip*: FITS file specifying IP corrections for each pixel and band. If set to 'uniform', the step will use the *qinst* and *uinst* values; otherwise, these values are ignored if fileip is specified.

- **rotate**
  - *gridangle*: Detector angle offset, in degrees. Should be a comma-separated list of 5 numbers, one for each band; for example, '-89.69, 0.0, -104.28, 37.42, 119.62'. 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.
  - *qubgsubtract*: Set to True to calculate and remove offsets in Q and U maps, in addition to Stokes I. Default is True; must be set to False for Chop-Nod data.
- **merge**
  - *cdelt*: Pixel size in arcseconds of output map, one number per band. Decrease to sub-sample the input pixels more. Default is '1.21, 1.95, 1.95, 3.40, 4.55', half the detector pixel scale (beam size / 4).
  - *fwhm*: FWHM in arcseconds of Gaussian smoothing kernel, by band. Make larger for more smoothing. Default is '2.57, 4.02, 4.02, 6.93, 9.43', for minimal smoothing.
  - *radius*: Integration radius for input pixels, by band. Set larger to consider more pixels when calculating output map. Default is '2.57, 4.02, 4.02, 6.93, 9.43'.
  - *errflag*: Set to True to use error image for weighting. Default is True.
  - *minnpix*: Set to a number greater than 0 to exclude pixels with fewer than this number of input pixels. May be useful to remove bad borders. Default is 0.0.
  - *fluxthreshold*: Flux threshold levels for treatment of bright pixels, one number per band. Stokes I pixels with values greater than this threshold will not be weighted by their error values. This is intended to avoid underestimating flux for bright, compact sources. If set to a negative number, bright pixels will not be treated differently. Default is '-1, -1, -1, -1, -1'.
- **region**
  - *skip*: Set to a number i to keep vectors every ith pixel. Default is 2 (as appropriate for cdelt=beamsize/4 in merge step).
  - *mini*: Do not keep vectors from pixels with Stokes I flux less than this fraction of peak flux. Default is 0.0.
  - *minisigi*: Do not keep vectors from pixels with Stokes I flux less than this many sigma. Default is 200.
  - *minp*: Do not keep vectors with percent polarization less than this value. Default is 0%.
  - *maxp*: Do not keep vectors with percent polarization greater than this value. Default is 50%.
  - *sigma*: Do not keep vectors with p/σ_p less than this value. Default is 3.0.
  - *length*: Scale factor for polarization vectors in DS9 region file, in pixels. Default is 10 (i.e. a 10% polarization vector is the length of one pixel).
  - *rotate*: Plot rotated (B field) vectors in DS9 region file. Default is True.
  - *debias*: Plot debiased vectors in DS9 region file. Default is True.
- **polmap**
  - *colormap*: Color map to use in polmap image. Default is 'rainbow'.
  - *colorvec*: Color to use for vectors in polmap image. Default is 'black'.
  - *colorcontour*: Color to use for contour lines in polmap image. Default is 'gray'.
  - *scalevec*: Scale factor for polarization vectors in polmap image. Default is 0.0003.
  - *rotate*: Plot rotated (B field) vectors in polmap image. Default is True.
  - *debias*: Plot debiased vectors in DS9 polmap image. Default is True.

14.2 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. Default for the DPS environment is -tau=atran, which specifies that an atmospheric model based on ATRAN should be used.
- **-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 Jy/pixel.
- **-sigmaclip / -sigmaclip=X**: Apply sigma clipping to the telescope velocities. If a value is not specified, the default value of 5 sigma will be used. Specifying a value will allow use of that value as the sigma multiplier.
- **-wtime=X**: Set a specific timeout, in minutes. By default, CRUSH will time out after 120 minutes.
- **-nowatch**: Allow CRUSH to continue processing indefinitely (no watchdog timer is set).

Part VII

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 across 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.
- Check the output to the log file (usually called *redux_[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 DMD, WCS, CAL, and PMP file for Nod-Pol data, and a CRH and CAL file for Scan data.

For Nod-Pol or Scan-Pol mode:

- Display all CAL 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 CAL files may need to be excluded, and the last steps of the pipeline re-run.
- Check the CAL files for persistent bad pixels or detector features. If present, the flat field or bad pixel mask may need updating.
- Display the final PMP file. Verify that the mapping completed accurately, with no unexpected or unusual artifacts. The weighting flags may need modification, or the smoothing may need to be increased.
- Overlay the DS9 polarization vector file (**.reg*) on the PMP 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 0.6% in regions that should have zero total polarization. If it is not, the instrumental polarization parameters may need adjusting.
- Check that sources appear at the expected coordinates. If they do not, the boresight offsets used by the pipeline may need to be adjusted.
- Check the FHWM and PSF shape of the source. If it is larger than expected, or not round, there may have been a problem with telescope guiding or chopping/nodding.
- If there are output products from the chi2 pipeline, review them for discrepancies between sets of dithers, or excessive noise or artifacts.

For Scan or Scan-Pol 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. holes or "worms" caused by bad pixels that were not properly excluded from the scans). Try reducing the data with the *-fixjumps* option to see if these artifacts improve.
- 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. Try reducing each scan individually to see if a bad scan may be identified and excluded.
- 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.
- Check the FHWM and PSF shape of the source. If it is larger than expected, or not round, there may have been a problem with telescope guiding or focus.
- Check that the source is at the expected coordinates. If not, the boresight offsets may need to be adjusted. Check the SIBSDX and SIBSDY keywords in the header.
- If the target is not visible in the default reduction, try reducing the data with the *faint* option.
- If the target has extended diffuse emission, it may be beneficial to try reducing the data with the *extended* option.

Part VIII

Appendix: CRUSH execution

CRUSH is a pipeline within the pipeline responsible for reducing HAWC+ scan-mode 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.

NOTE: As of CRUSH version 2.42.1, the SOFIA version of CRUSH used by the HAWC+ pipeline has diverged from the CRUSH source on sigmyne.com. Use that distribution at your own risk, and note that not all features will be cross-compatible.

15 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 the CRUSH download page.

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.

16 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.

17 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.

18 Command-Line Options

Some common HAWC+ command-line options are listed in Section [params], 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

19 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 notifications of new releases by watching the repository at the CRUSH GitHub page. You can also file reports of bugs you might encounter in CRUSH on the GitHub project page.

Part IX

Appendix: Sample Configuration Files

20 Full DRP Configuration File

Below is a copy of the full configuration file used by the pipeline in the DPS environment (*pipeconf.cfg*). It is in INI format, and is readable by the configobj Python module.

```
# HAWC Pipeline Base Configuration File
#
# This file contains all settings for reducing HAWC science and
# in-flight diagnostic data. It is intended to be used with
# additional delta configuration files.
#
# DO NOT edit this file without consulting with the HAWC data
# reduction team.

#### Basic Settings ####
#======================

# 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[\dX]{3,4}_HA_[A-Za-z]+_[A-Za-z0-9]+_
↪[A-Za-z0-9]+))_'
    filenameend = '_((\d+-)?\d+)(?:_BIN\d+)?\.fits(\.gz)?\Z' # HAWC+
    filenum = '(?:\A.*(?:(?:F\d{3,4})|(?:XXXX))_((?:\d+-)?\d+)_.*\.fits(?:\.
↪gz)?\Z)|(?:\AF[\dX]{3,4}_HA.*_((?:\d+-)?\d+)(?:_BIN\d+)?\.fits(?:\.gz)?\Z)'
    dataobjects = DataFits, DataText #, DataCsv


#### PIPE MODES ####
#==================
# configuration for individual pipeline modes. Each needs:
# - datakeys: List of keyword=values required in file header to select this␣
↪pipeline mode
```

```
#             Format is: Keyword=Value|Keyword=Value|Keyword=Value
# - stepslist: List of pipesteps to run the data through


# Lab polarimetry data -- match this first, since it depends
# on a specific CMTFILE only
[mode_labpol]
    datakeys = 'CMTFILE=Hawc_Take data at HWP positions.txt'
    # list of steps
    stepslist = StepCheckhead, StepPrepare, StepDemodulate, StepDmdPlot,␣
↪StepDmdCut, StepFlat, StepShift, StepSplit, StepCombine, StepNodPolSub,␣
↪StepStokes, StepWcs, StepPolVec, StepRegion, StepLabPolPlots
    # change stepprepare to labmode
    [[checkhead]]
        abort = False
    [[prepare]]
        labmode = True
        colrename = 'crioTTLChopOut->Chop Offset|AZ_Error->Azimuth Error|EL_
↪Error->Elevation Error|AZ->Azimuth|EL->Elevation|SIBS_VPA->Array VPA'
        chpoffsofiaRS = False
    [[demodulate]]
        track_tol = -1
    [[dmdcut]]
        mask_bits = 64
    [[flat]]
        labmode = True
    [[wcs]]
        labmode = True
        save = True
    [[region]]
        save = True
    [[header]]
        NODPATT = "'A' / Nod Pattern"
        CHPFREQ = 2.988 / Chop Frequency

# Mode for Internal Calibrator File to generate flats
[mode_intcal]
    datakeys ='CALMODE=INT_CAL'
    # list of steps
    stepslist = StepCheckhead, StepFluxjump, StepPrepare, StepDemodulate,␣
↪StepDmdPlot, StepDmdCut, StepMkflat
    # Always attempt to continue reduction
    [[checkhead]]
        abort = False
    # Stepprepare change to labmode and get Chop Offset from crioAnalogChopOut
    [[prepare]]
        labmode=True
        colrename = 'crioAnalogChopOut -> Chop Offset|AZ_Error->Azimuth␣
↪Error|EL_Error->Elevation Error|AZ->Azimuth|EL->Elevation|SIBS_VPA->Array␣
↪VPA'
        chpoffsofiars = False
    # Change demodulation options
    [[demodulate]]
        l0method = 'RE'
```

```
        phasefile = 0.0
        checkhwp = False
        track_tol = -1
    [[dmdcut]]
        mask_bits = 64
    # Change header value NODPATT = A
    [[header]]
        NODPATT  = "'A' / Nod Pattern"

# SKYDIP data
[mode_skydip]
    datakeys = 'INSTCFG=TOTAL_INTENSITY|CALMODE=SKY_DIP'
    # list of steps
    stepslist = StepCheckhead, StepCrush, StepFluxjump, StepPrepare,␣
↪StepDemodulate, StepDmdPlot, StepDmdCut, StepSkydip
    # Always attempt to continue reduction
    [[checkhead]]
        abort = False
    # Crush - no output
    [[crush]]
        noout = True
    # Stepprepare change to labmode and get Chop Offset from crioAnalogChopOut
    [[prepare]]
        labmode=True
        colrename = 'crioAnalogChopOut -> Chop Offset|AZ_Error->Azimuth␣
↪Error|EL_Error->Elevation Error|AZ->Azimuth|EL->Elevation|SIBS_VPA->Array␣
↪VPA'
            chpoffsofiaRS = False
    # Change demodulation options
    [[demodulate]]
        l0method = 'ABS'
        track_tol = -1
        track_extra = 0,0
    [[dmdcut]]
        mask_bits = 64
    [[header]]
        NODPATT  = "'A' / Nod Pattern"

# POLDIP data
[mode_poldip]
    datakeys = 'INSTCFG=POLARIZATION|CALMODE=SKY_DIP'
    stepslist = StepCheckhead, StepFluxjump, StepPrepare, StepPolDip
    # Always attempt to continue reduction
    [[checkhead]]
        abort = False
    [[prepare]]
        traceshift = 4

# AUTOFOCUS: Mode for Automatic Focusing for Scan data
[mode_autofocus]
    datakeys = 'INSTMODE=OTFMAP|INSTCFG=TOTAL_INTENSITY|CALMODE=FOCUS'
    # list of steps
    stepslist = StepCheckhead, StepCrushFocus, StepStdPhotCal, StepFocus
    [[crush]]
```

```
        frame_range = ''

# ChopNod Mode Configuration
[mode_nod_std_dmd]
    datakeys = 'INSTMODE=C2N (NMC)|INSTCFG=TOTAL_INTENSITY|OBSTYPE=STANDARD_
↪FLUX|PRODTYPE=demodulate'
    stepslist = StepDmdPlot, StepDmdCut, StepFlat, StepShift, StepSplit,
↪ StepCombine, StepNodPolSub, StepStokes, StepWcs, StepOpacity,␣
↪StepBgSubtract, StepMerge, StepStdPhotCal
[mode_nod_std]
    datakeys = 'INSTMODE = C2N (NMC)|INSTCFG = TOTAL_INTENSITY|OBSTYPE=STANDARD_
↪FLUX'
    # list of steps
    stepslist = StepCheckhead, StepFluxjump, StepPrepare, StepDemodulate,
↪ StepDmdPlot, StepDmdCut, StepFlat, StepShift, StepSplit, StepCombine,␣
↪StepNodPolSub, StepStokes, StepWcs, StepOpacity, StepBgSubtract, StepMerge,␣
↪StepStdPhotCal
    [[demodulate]]
        checkhwp = False
[mode_nod_dmd]
    datakeys = 'INSTMODE = C2N (NMC)|INSTCFG = TOTAL_INTENSITY|PRODTYPE =␣
↪demodulate'
    stepslist = StepDmdPlot, StepDmdCut, StepFlat, StepShift, StepSplit,
↪ StepCombine, StepNodPolSub, StepStokes, StepWcs, StepOpacity,␣
↪StepBgSubtract, StepMerge, StepCalibrate
[mode_nod]
    datakeys = 'INSTMODE = C2N (NMC)|INSTCFG = TOTAL_INTENSITY'
    # list of steps
    stepslist = StepCheckhead, StepFluxjump, StepPrepare, StepDemodulate,
↪ StepDmdPlot, StepDmdCut, StepFlat, StepShift, StepSplit, StepCombine,␣
↪StepNodPolSub, StepStokes, StepWcs, StepOpacity, StepBgSubtract, StepMerge,␣
↪StepCalibrate
    [[demodulate]]
        checkhwp = False

# Nod-Pol Mode Configuration
[mode_nodpol_dmd_std]
    datakeys = 'INSTMODE=C2N (NMC)|INSTCFG=POLARIZATION|OBSTYPE=STANDARD_
↪FLUX|PRODTYPE=demodulate'
    stepslist = StepDmdPlot, StepDmdCut, StepFlat, StepShift, StepSplit,
↪ StepCombine, StepNodPolSub, StepStokes, StepWcs, StepIP, StepRotate,
↪ StepOpacity, StepBgSubtract, StepMerge, StepStdPhotCal, StepPolVec,␣
↪StepRegion, StepPolMap
[mode_nodpol_std]
    datakeys = 'INSTMODE=C2N (NMC)|INSTCFG=POLARIZATION|OBSTYPE=STANDARD_FLUX'
    # list of steps
    stepslist = StepCheckhead, StepFluxjump, StepPrepare, StepDemodulate,
↪ StepDmdPlot, StepDmdCut, StepFlat, StepShift, StepSplit, StepCombine,
↪ StepNodPolSub, StepStokes, StepWcs, StepIP, StepRotate, StepOpacity,
↪ StepBgSubtract, StepMerge, StepStdPhotCal, StepPolVec, StepRegion,␣
↪StepPolMap
[mode_nodpol_dmd]
    datakeys = 'INSTMODE = C2N (NMC)|INSTCFG = POLARIZATION|PRODTYPE =␣
↪demodulate'
```

```
    stepslist = StepDmdPlot, StepDmdCut, StepFlat, StepShift, StepSplit,
↪ StepCombine, StepNodPolSub, StepStokes, StepWcs, StepIP, StepRotate,
↪ StepOpacity, StepCalibrate, StepBgSubtract, StepMerge, StepPolVec,␣
↪StepRegion, StepPolMap
[mode_nodpol]
    datakeys = 'INSTMODE = C2N (NMC)|INSTCFG = POLARIZATION'
    # list of steps
    stepslist = StepCheckhead, StepFluxjump, StepPrepare, StepDemodulate,
↪ StepDmdPlot, StepDmdCut, StepFlat, StepShift, StepSplit, StepCombine,
↪ StepNodPolSub, StepStokes, StepWcs, StepIP, StepRotate, StepOpacity,␣
↪StepCalibrate, StepBgSubtract, StepMerge, StepPolVec, StepRegion, StepPolMap

# Imaging Scan Mode Configuration
[mode_scan_std]
    datakeys = 'INSTMODE=OTFMAP|INSTCFG=TOTAL_INTENSITY|OBSTYPE=STANDARD_FLUX'
    stepslist = StepCheckhead, StepCrush, StepStdPhotCal

[mode_scanpol_std]
    datakeys = 'INSTMODE=OTFMAP|INSTCFG=POLARIZATION|OBSTYPE=STANDARD_FLUX'
    stepslist = StepCheckhead, StepCrushPol, StepScanStokes, StepIP,␣
↪StepRotate, StepStdPhotCal, StepMerge, StepPolVec, StepRegion, StepPolMap
    [[ip]]
        fileip = uniform

[mode_scan]
    datakeys = 'INSTMODE=OTFMAP|INSTCFG=TOTAL_INTENSITY'
    stepslist = StepCheckhead, StepCrush, StepCalibrate

# Scanning Polarimetry Mode Configuration
[mode_scanpol]
    datakeys = 'INSTMODE=OTFMAP|INSTCFG=POLARIZATION'
    #stepslist = StepCheckhead, StepCrushPol, StepScanStokes, StepIP,␣
↪StepRotate, StepCalibrate, StepMerge, StepPolVec, StepRegion, StepPolMap
    stepslist = StepCheckhead, StepCrushPol, StepScanStokes, StepIP,␣
↪StepRotate, StepCalibrate, StepMerge, StepPolVec, StepRegion, StepPolMap
    [[ip]]
        fileip = uniform

#### PIPE STEPS ####
#====================
#   First the definitions for the parent steps, then all HAWC steps
#   in alphabetical order. Finally diagnostic steps.

# BGSUBTRACT - background subtraction step
[bgsubtract]
    cdelt = 2.57, 4.02, 4.02, 6.93, 9.43 # Pixel size in arcseconds of output␣
↪map
    proj = TAN # Projection of output map
    sizelimit = 3000 # Upper limit on map size (either axis, in pixels)
    fwhm   = 2.57, 4.02, 4.02, 6.93, 9.43  # FWHM of gaussian smoothing␣
↪kernel, in arcseconds
    radius = 7.71, 12.06, 12.06, 20.79, 28.29  # Integration radius for␣
↪smoothing, in arcseconds
    errflag = True   # Use uncertainties when computing averages?
```

```
    widowstokesi = True   # Use widow pixels (flagged 1 or 2) when smoothing
    edge_threshold = 0.75   # Set edge pixels to NaN
    fit_order = 1 # Fit order for local regression
    bgoffset = 10 # Number of iterations of background subtract with offset␣
↪(intercept) term
    chauvenet = True   # Use Chauvenet's criterion in background subtraction?
    fitflag = False   # Use errors in intensity when fitting?
    qubgsubtract = True # Apply background offsets to individual Stokes QU␣
↪files?

# CALIBRATE - fluxes from data units to Jy/pixel
[calibrate]

# COMBINE - R-T and R+T data
[combine]
    sigma = 3.0   # Reject outliers more than this many sigma from the mean
    sum_sigma = 4.0  # Reject additional R+T outliers more than sum_sigma␣
↪from the mean
    use_error = False # Set to True to use Chauvenet output errors rather␣
↪than propagating input variances

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

# Run the crush scan data reduction
[crush]
    crushpath = $DPS_HAWCPIPE/extern/crush
    options = ''
    verbose = True
    subarray = 'R0,T0,R1'
    frame_range = ''

# Run the crush scanpol data reduction
[crushpol]
    crushpath = $DPS_HAWCPIPE/extern/crush
    options = ''
    verbose = False
    vpa_tol = 5.0
    frame_range = ''

# CRUSHFOCUS - Run crush on focus groups
[crushfocus]
    groupkeys = 'FOCUS_ST'  # header keywords to decide data group membership␣
↪(| separated)
    groupkfmt = '%.1f'   # group key formats to force string comparison (|␣
↪separated)

# DEMODULATE -  Demodulate the chopped data while keeping all samples
[demodulate]
    chop_tol = 0.2   # chopper tolerance in arcseconds
    nod_tol = 5.0   # nod tolerance in arcseconds
    hwp_tol = 2.   # hwp angle tolerance in degrees
```

```
    az_tol = 5000000.0   # Azimuth error tolerance in arcseconds
    el_tol = 5000000.0  # Elevation error tolerance in arcseconds
    track_tol = 'centroidexp'    # Track error tolerance in arcseconds (AOIs␣
↪3 and 4) - set negative to deactivate
    track_extra = 0, 0  # Extra samples removed (in seconds) before and after␣
↪samples flagged by track_tol
    chopphase = True   # Flag requiring chop phase correction
    checkhwp = True  # Set FALSE to avoid check expected number of HWP angles
    phasefile = $DPS_HAWCPIPE/data/phasefiles/masterphase_170307.fits
    phaseoffset = 0.0 # Offset to apply to phasefile, in degrees
    l0method = 'RE' # Method to normalize data: REal, IMag and ABSolute
    boxfilter = -1 # Box Highpass Filter. -1 = frequency from the header
    chopavg = True # Flag to save chop averaged raw data (default = False)
    tracksampcut = 0.5 # If fraction of all samples removed due to tracking␣
↪is larger than this number, than tracking status is BAD
    data_sigma = 5.0 # value for sigma-clipping of detector data in variance␣
↪calculation

# DMDCUT - Discard chops based from the Demodulate output
[dmdcut]
    mask_bits = 1023   # bits of 'Chop Mask' on which to discard chops
    min_samples = 1  # minimum number of samples for retaining a chop

# DMDPLOT - Plot output of Demodulate
[dmdplot]
    door_threshold = 2.0          # ratio of imaginary to real median stds␣
↪for door vignetting
    detector_i = 14               # i-location of detector pixel to plot
    detector_j = 24               # j-location of detector pixel to plot
    data_sigma = 5.0              # value for sigma-clipping of detector data
    data_iters = 3                # number of iterations for data clipping
    user_freq = 10.2              # INT_CAL: user frequency in Hz
    ref_phase_file = $DPS_HAWCPIPE/data/phasefiles/refphases_180419.fits  #␣
↪path to reference phase file in degrees (0.0 is none)
    phase_thresh = 50.0           # threshold in phase uncertainty (deg) for␣
↪blanking pixels
    save_phase = False            # Save phase images to PHS suffix
    savefolder = ''               # Folder to save plots to. '' means same as␣
↪input file

# FLAT - step configuration
[flat]
    # filename glob to find the flat files
    flatfile = flats/*OFT*.fits
    # list of keys that need to match flat and data file (only if␣
↪flatfile=search)
    flatfitkeys = 'SPECTEL1', 'MISSN-ID', 'FILEGPID', 'SCRIPTID'
    # Back up filename for auxiliary file(s). Can contain * and ? wildcards␣
↪to match
    # multiple files to be selected using fitkeys (default = bkupflatfile/*.
↪fits)
    bkupflat = $DPS_HAWCPIPE/data/flats/*OFT.fits  # Backup flat files

# FLUXJUMP - 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')
    # Default is a no-op map
    jumpmap = $DPS_HAWCPIPE/data/fluxjumps/flux_jump_dummy.fits

# FOCUS - step configuration
[focus]
    widowisgood = True       # Include widow pixels in the analysis (T) or␣
↪only good pixels (F, will assume widow pixels are bad)
    medianaverage = True     # Run a median average box through the array to␣
↪fill bad pixels (T) or not (F)
    boxaverage = 5           # Size of the median average box (if␣
↪medianaverage is True) in pixels
    autocrop = True          # Crop image automatically around the target (w/␣
↪boxsize = 1/3 of image size)
    cropimage = True         # Crop portion (box) of the image for analysis?␣
↪True or False
    xyboxcent = 87,87       # If cropimage = True, central X/Y pixel position␣
↪of the box to be cropped
    boxsizecrop = 30         # If cropimage = True, size of the box to be␣
↪cropped (in pixels)
    primaryimg = ''          # Specifies which image will be used for the␣
↪Gaussian fit. If left blank, the first image will be used.

# IP Correction for instrumental polarization step configuration
[ip]
    # IP from planets estimated using FS13.
    #qinst =  -0.0154, 0.0, -0.0151, 0.0028, -0.0129   # Fractional␣
↪instrumental polarization in q
    #uinst =  -0.0030, 0.0,  0.0090,  0.0191, -0.0111  # Fractional␣
↪instrumental polarization in u
    # Median q/u from below file
    qinst =  -0.0157, 0.0, -0.0164, 0.0009, -0.0104
    uinst =  -0.0038, 0.0,  0.0081, 0.0192, -0.0142
    # IP file from FS15 poldip
    fileip = $DPS_HAWCPIPE/data/ip/hawc_ip_FS15_poldip_v1.fits

# MERGE - step configuration
[merge]
    beamsize = 4.84, 7.80, 7.80, 13.6, 18.2 # Beam FWHM size (arcsec) to␣
↪write into BMAJ/BMIN header keywords
    cdelt  = 1.00, 1.55, 1.55, 2.75, 3.7   # Pixel size in arcseconds of␣
↪output map. celt = beamsize/4, roughly.  Matches CRUSH values.
    proj = TAN   # Projection of output map
    sizelimit = 3000 # Upper limit on map size (either axis, in pixels)
    fwhm   = 2.42, 3.9, 3.9, 6.8, 9.1  # FWHM of gaussian smoothing kernel,␣
↪in arcseconds. Default is beamsize/2.
    radius = 7.26, 11.7 , 11.7 , 20.4 , 27.3  # Integration radius for␣
↪smoothing, in arcseconds. Default is 1.5*beamsize.
    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 (due to change in␣
```

```
↪pixel size) to all output images
    edge_threshold = 0.75   # Set edge pixels to NaN
    fit_order = 1 # Fit order for local regression

# MKFLAT - Make flat file from INT_CAL files
[mkflat]
    # Path for the folder to write flat files to (default: .)
    flatoutfolder = flats
    # Header Keyword to match input files to the same observation (default:␣
↪FILEGPID)
    groupkey = "SCRIPTID"
    # Chops to exclude from the beginning of the file (default: 1)
    skip_start = 1
    # Chops to exclude from the end of the file (default: 1)
    skip_end = 1
    # Raw data threshold for dead pixels (default: 10.0)
        bad_dead = 10.0
        # Raw data threshold for ramping pixels (default: 2000000.0)
    bad_ramping = 2000000
    # Threshold for HIGH STD of DMD SIGNAL to exclude pixels (default: 10.0)
    normstd = 10.0
    # Threshold to eliminate pixels with LOW SIGNAL (default: [0.5, 0.5, 0.5])
    ynormlowlim = 0.5, 0.5, 0.5
    # Threshold to eliminate pixels with HIGH NORMALIZED SIGNAL
    # (default: [10.0, 10.0, 10.0])
    ynormhighlim = 10.0, 10.0, 10.0
    # Scale factor for T/R flatfield (default: 2.0)
    TtoR = 2.0
    # Filename for auxiliary file(s). Can contain * and ? wildcards to match
    # multiple files to be selected using fitkeys (default = skycal/*.fits)
    # (default: skycal/*SCAL.fits)
    scalfile = $DPS_HAWCPIPE/data/skycals/fs15/*.fits
    # Back up filename for auxiliary file(s). Can contain * and ? wildcards
    # to match multiple files to be selected using fitkeys
    bkupscal = $DPS_HAWCPIPE/data/skycals/fs15/*.fits
    # List of header keys that need to match auxiliary data file
    # - only used if multiple files match skycal (default = [])
    scalfitkeys = SPECTEL1

# NODPOLSUB - Subtract L and R nods with HWP
[nodpolsub]

# OPACITY - Correction for model atmospheric opacity
[opacity]

# POLDIP - Polarization skydip step
[poldip]
   hwp0 = 5.0 # Reference HWP angle
   temp0 = 0.532 # Reference temperature (ADR setpoint)
   maxrms = 0.1 # Maximum allowed reduced RMS

# POLMAP - Polarization map step
[polmap]
   mapfile = 'same'           # Alternative fits file name for background␣
```

```
↪image. 'same': matches the background and pol. files. If other background␣
↪image wants to be used then the input is the filename.
   maphdu = 'STOKES I'        # HDU name to be used in the mapfile. The HDU␣
↪used for the background image.
   scalevec = 0.0003          # Scale factor for vector sizes
   scale = True               # Set to False to make all vectors the same␣
↪length
   rotate = True              # True gives (B-Field) vectors
   debias = True              # Use debiased polarizations
   lowhighscale = 'automatic' # Low/High values for image scaling - set␣
↪automatic or [low,high]
   colorvec = 'black'         # Vector colors
   colormap = 'rainbow'
   colorcontour = 'gray'
   uselatex = False
   fillcontours = True
   ncontours = 30             # Number of contours
   title = 'info'             # Title in the polarization map
   centercrop = False         # Crop a region of the image. Default = False.␣
↪Inputs: RA, DEC, width, height in degrees.

# POLVEC - Polarization vector step configuration
[polvec]
    # telescope polarization efficiency
    eff = 0.842, 0.9, 0.939, 0.975, 0.978

# PREPARE - 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 Orig'
    # List of data columns to delete: The format ["column1","column2",...]
    coldelete = hwpA,hwpB,FluxJumps
        # Number of samples to shift the data (default is 0 i.e. no shift)
    traceshift = 0
    # List for PIXCAL values for each band - to update PIXSCAL in the header
    pixscalist  = 2.57, 4.02, 4.02, 6.93, 9.43
    # Remove data Dropouts (i.e. data with RA==Dec==0)
    removedropouts = True

# REGION - Extract ds9 region file of polarization vectors
[region]
```

```
    skip = 2       # Only plot every ith pixel. If skip =1 it will show every␣
↪pixel. If cdelt = beamsize/4, skip=2 gives Nyquist sampling.
    scale = True   # Set to False to make all vectors the same length
    rotate = True   # Use rotated (B-Field) vectors
    debias = True   # Use debiased polarizations
    length = 10.0   # Scale factor for length of polarization vectors in␣
↪pixels.
    mini = 0.0   # Do not plot vectors with flux < this fraction of peak flux
    minp = 0.0   # Require percentage polarizations to be >= this value
    offset = 0, 0   # Offset in pixels in x,y (controls which pixels are␣
↪extracted)
    sigma = 3.0   # p/sigmap must be >= this value
    minisigi = 200  # StokesI/ErrorI must be above this value
    maxp = 50  # Pol. Degree must be below this value

# ROTATE - Rotate Q and U from detector to sky frame step configuration
[rotate]
    gridangle = -89.69, 0.0, -104.28, 37.42, 119.62  #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

# SHIFT - 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 - Split data by HWP angle and nod position step configuration
[split]
    # Nod tolerance, as the percent difference allowed in number of chop␣
↪cycles
    # between 1st and 2nd left, and between left and right
    nod_tol = 50.0

# STDPHOTCAL - Run photometry on standards and calibrate to Jy
[stdphotcal]

# STOKES - 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.
    erripolmethod   = meansigma         # Options are "hwpstddev" or␣
↪"meansigma"
    removeR1stokesi = True      # Remove R1 subarray for Stokes I
    override_hwp_order = False  # If True, the first two HWP angles will be␣
↪used for Q, last two for U

# WCS - Update Parallactic angle and crval1 and crval2 for a single file
[wcs]
    add180vpa = True   # Add 180 degrees to the SIBS_VPA
    # Small Offset (in pixels along x/y) between SIBS_X/Y and actual target␣
↪position
    offsibs_x = 0.0, 0.0, 0.0, 0.0, 0.0
    offsibs_y = 0.0, 0.0, 0.0, 0.0, 0.0
    labmode = False # If labmode = True, will ignore keywords and input␣
↪parameters and create fake astrometry

#### 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]
    #INSTMODE = "'test' / instrument mode"
    #CHPFREQ  = "10.0 / Chop Frequency"
    #SKYANGL  = 0.0 / Sky Angle
    #CHOPPING = T / Chopping flag
    #CHPMODE  = "'2-POINT' / Chopping mode"
    #CHPAMP1  = 30000 / Chop Amplitude
    #CHPANGLE = 0.0 / Chop Angle
    #DETSIZE  = "'(32,41)'"
    #NHWP     = "1 / Number of HWP angles"
    #NODDING  = T / Nodding flag
    #NODANGLE = 92.8 / Nod Angle
    #NODPATT  = "'ABBA' / Nod Pattern"
    #NODTIME  = 5.0 / Nod Integration Time
    #NODSETL  = 0.05 / Nod Settle Time
    #OBSRA    = 5000 / Observation RA (now DOG units)
    #OBSDEC   = 5000 / Observation DEC (now DOG units)
    #SCANNING = T / Scanning flag
    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
    ASSC_MSN = CONCATENATE
    DTHINDEX = DEFAULT
    LAT_END  = LAST
    LON_END  = LAST
    FBC-STAT = LAST
    FOCUS_EN = LAST
    OPCCORFC = CONCATENATE
    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
    ai22              = MED
    ai23              = MED
    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
r array imag var  = FIRST
t array imag var  = FIRST
chop offset imag  = FIRST
r array avg       = FIRST
t array avg       = FIRST
phase corr        = WTAVG
nod_off           = WTAVG
centroidexpmsec   = WTAVG
# Centroid Values for FS15
centroidworkphase = WTAVG
centroidaoi       = FIRST
# SofiaHK values (temporary)
sofhkchopamp      = WTAVG
sofhkbinning      = WTAVG
sofhkaoi4col      = WTAVG
sofhkaoi4row      = WTAVG
sofhkaoi3col      = WTAVG
sofhkaoi3row      = WTAVG
sofhktrkaoi       = WTAVG
sofhkseqphase     = WTAVG
sofhkexptime      = WTAVG
sofhkaoi4err      = WTAVG
sofhkaoi3err      = WTAVG
chop mask         = FIRST
```

21 DRP Override Configuration File

Below is a sample 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 FS15 (OC5N), flight F440
# This is not a full configuration file -- it should be merged
# with pipeconf.cfg before using.
#
# 2018-02-01 D. Perera
# 2018-03-05 M. Clarke: Update SIBS offsets and IP file

# Flux Jump step configuration
[fluxjump]
    jumpmap = $DPS_HAWCPIPE/data/fluxjumps/flux_jump_FS15_v3.fits

# Correction for instrumental polarization
[ip]
    fileip = $DPS_HAWCPIPE/data/ip/hawc_ip_FS15_poldip_v1.fits

# Make flat from int_cal
[mkflat]
   scalfile = $DPS_HAWCPIPE/data/skycals/fs15/*.fits

# Update Parallactic angle and crval1 and crval2 for a single file
[wcs]
    offsibs_x = -0.2, 0., 0.0, -0.6, -0.05
    offsibs_y = 5.2, 0., -1.2, 2.0, -0.80
```

22 Full CRUSH Configuration File

Below is a copy 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

# Parse historical leap-seconds data, for accurate (within 0.5 second) UTC to
# MJD conversion, when needed...
leapseconds {?configpath}/leap-seconds.list

# 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

# 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
[pixelmap] source.type pixelmap
[skydip] source.type skydip

# For old times' sake:
[beammap] pixelmap

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

# Do not use initial pixel data when reducing pixel maps...
[source.type?pixelmap] 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 vclip,aclip

# Do not downsample chopped data...
[chopped] forget downsample

# Disable motion filtering for chopped data
[chopped] forget filter.motion

# 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 15

# 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
[extended] blank 100
```

```
# 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
iteration.[2] [extended] blank 100.0

# 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

# 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.method 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?pixelmap] 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

# Additional options to pixel maps...
# Process pixel maps like regular maps
pixelmap.process

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

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

# Restrict pointing fits to a circular area around the nominal position.
# The radius is specified in arcsec.
```

```
#pointing.radius 60.0

# 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

# Compress outputs (e.g. FITS) with gzip, if possible. (The .gz extension will
# be added as necessary).
#gzip
```

```
# 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 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 (e.g. with minicrush)
spectral.alias.resolution r
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
```

23 HAWC+ CRUSH Configuration File

Below is the HAWC+ configuration file for CRUSH. Values in this file override those in the global configuration file for HAWC reductions.

```
# ============================================================================
# CRUSH default configuration for SOFIA/HAWC+
#
# Author: Attila Kovacs <attila[AT]sigmyne.com>
# Description:
#       This configuration file is automatically loaded when crush is started
#       with hawc+ as the instrument. Users may define their own startup
#       configuration in ~/.crush2/hawc+/default.cfg which will be parsed
#       immediately after the global defaults contained here.
# See: crush/README, crush/hawc+/README
# ============================================================================
```

```
# Load SOFIA defaults
config sofia/default.cfg

# The ordering of models in the default reduction pipeline.
ordering dejump, offsets, drifts, correlated.obs-channels, correlated.sky,
 ↪ correlated.nonlinearity, correlated.polarrays, correlated.telescope-x,
 ↪ correlated.chopper-x, correlated.chopper-y, correlated.los, correlated.
 ↪pitch, correlated.roll, correlated.accel-|y|, weighting.frames, filter,␣
 ↪weighting, despike, correlated.subarrays, correlated.gradients, correlated.
 ↪bias, correlated.series, correlated.mux, correlated.rows, source

# Set keywords identifying specific runs/periods...
date.[*--2016.07.01] apr2016
date.[2016.09.01--2016.11.01] oct2016
date.[2016.11.30--2016.12.20] dec2016
date.[2017.05.01--2017.06.01] may2017
date.[2017.10.01--2017.12.01] oct2017

date.[2018.01.01--2018.07.16] oc6i
date.[2018.07.17--2018.11.01] oc6k
date.[2019.01.01--2019.03.01] oc6t
date.[2019.03.02--2019.08.01] oc7e
date.[2019.08.02--2019.10.15] oc7f

# Select specific subarrays only. E.g. if pointing to the center of R0, then
# reduce R0/T0 only...
fits.[SIBS_X?15.5] subarray T0,R0

# Reduce skydips with R0 only (least non-linear)
[source.type?skydip] subarray R0
[source.type?skydip] subarray.lock

# Transfer FITS header keys to configuration options...
fits.assign.DIAG_HZ intcalfreq

# If 'calibrated' is set, CRUSH will produce Level 3 data for single scans
# Otherwise, the default is to produce Level 2 data for single scans
#calibrated

# Specify the unit of the raw data
dataunit counts

# Specify the output units
#unit Jy/beam
#unit Jy/pixel

# Use the 'peakflux' option
[peakflux] scale 1.18

# The gain conversion to readout units
gain -1.0

# Assumes sign of source signals +, -, or 0
source.sign +
```

```
# The appropriate Jy/K conversion value (assuming 2.5m, 95% forward eff.)
K2Jy 582

# starting Oct 2016 run, assume real-time object coordinates (rtoc) are
# recorded in the FITS for all sourcess, regardless of whether they are
# sidereal or not.
rtoc

# Use data recorded between scans
#betweenscans

# The minimum length of a valid scan in seconds.
subscan.minlength 5.0

# Shift data relative to coordinates by the specified amount (seconds).
shift -0.014

# Shift chopper data to align with detectors
chopper.shift 2

# Set a tolerance (arcsec) for the chopper signal. It has to be within the
# nominal amplitude value for the frame to be used. This is useful to avoid
# smearing when reducing chopped data...
chopper.tolerance 10

# Discard slow scanning frames with entirely (instead of just flagging them).
vclip.strict

# Allow velocity clip for chopped data (mapping mode)
[chopped] recall vclip

# Apply correction for gyro drifts
[oct2017] gyrocorrect

# Set a limit to what's the largest gyro drift that can be corrected...
# (in arcsec)
gyrocorrect.max 30

# Map even if many pixels are flagged
mappingfraction 0.2

# Use the faster maximum-likelihood estimation from the start...
estimator maximum-likelihood

# Set the initial 1/f timescale..
drifts 30

# When using non-linear response corrections, make sure the drift window␣
↪covers
# the entire scan...
[correlated.nonlinearity] drifts max

# 1/f stability timescale in seconds
```

```
stability 5.0
[extended] stability 10.0

# Use shorter 'stability' timescale for short scans, such as focus scans, to
# get the crispest possible images...
obstime.[<45] stability 2.5

# Flag some MUX lines that seem to be always bad...
flag.mux 6,20,24,27,32,46-49,56,70,86
date.[*-2018.10.20] flag.mux 6,20,24,27-34,40,46-48,50,63,70,86

# Flag only MUXes that seem really dead
# flag.mux 6,20,27-32,46-48,50,70,86

# Flag only MUXes that seem to produce crazy source reponse
# flag.mux 24,33,34,40,49,63,127

# Flag rows that seem always bad
flag.row 14,15,19,52,82,83,87
date.[*--2018.10.20] flag.row 2,19,52,82,83,87,114,122,65,69,77

# Flag only rows that seem to be blind...
#flag.row 2,19

# Flag only rows that seem to have strange source response...
#flag.row 82,83,87,114,122

# Flag some more baddies
#flag R1[24,16]:R1[24,31],R1[28,16]:R1[28,31],R1[36,16]:R1[36:31]

# The overall rotation of the array from crush x,y coordinates to SI x,y.
date.[*--2017.10.01] rotation 0.9
rotation 0.1

# The relative rotations of the subarrays.
rotation.R0 0.0
rotation.R1 180.0
date.[*--2017.10.01] rotation.T0 -0.5
rotation.T0 0.5
#rotation.T1 180.5


# Subarray offsets (in pixels)
offset.R0 0.0,0.0
offset.R1 67.03,39.0
date.[*--2017.10.01] offset.T0 0.18,-0.17
#date.[2017.10.02--2018.08.01]offset.T0 0.29,-0.27
offset.T0 0.5,-0.5
#offset.T1 ???,???

# zoom constants (T vs R)
zoom.T 1.0

# Correct for flux jumps
```

```
date.[*--2017.05.01] jumpdata {?configpath}/hawc+/flux_jump_FS13_v1.fits.gz
[may2017] jumpdata {?configpath}/hawc+/flux_jump_FS14_v1.fits.gz
[oct2017] jumpdata {?configpath}/hawc+/flux_jump_FS15_v3.fits.gz

# Whether to 'fix' flux jumps
#iteration.[last-1] fixjumps

# Flag pixels outside an acceptable range of relative noise levels
weighting.noiserange 0.3:3.0

# Use neighbor-based despiking all the way...
despike.method.lock neighbours

# Define various shorthands for decorrelations
alias.pols correlated.polarrays
alias.subs correlated.subarrays
alias.biaslines correlated.bias
alias.mux correlated.mux
alias.rows correlated.rows
alias.series correlated.series
alias.accel correlated.accel-|y|
alias.los correlated.los
alias.roll correlated.roll

# The range of acceptable relative sky-noise gains.
array.signed
array.gainRange 0.3:3.0

# Decorrelate sky signal (separated from temperature signal)
[pixeldata] sky

# If the couplings are merged into the correlated gains, then do not
# decorrelate on sky separately...
[sourcegains] blacklist sky

# Decorrelate on polarization arrays
#pols

# Decorrelated on TES bias lines
biaslines
biaslines.gainrange 0.3:3.0

# Decorrelate on the series arrays (heat-sinking)
iteration.[last-1] series
series.nogains

# Decorrelate on SQUID multiplexed channels
#mux
mux.nogains
mux.gainrange 0.3:3.0
#iteration.[2] forget mux.nogains

# Decorrelate on detector rows (i.e. MUX address lines)
#rows
```

```
rows.gainrange 0.3:3.0

# For chopped data, remove the chopper-induced correlated signals...
[chopped] correlated.chopper-x
[chopped] correlated.chopper-y

# Remove correlations to second-derivative of LOS angle (a proxy for pitch
# accelerations)
#los
#los.gainRange 0.3:3.0

# Remove correlations to second-derivative of roll angle
#roll
#roll.gainRange 0.3:3.0

# Activate DRP messages over TCP/IP
#drp

# Various options for the DRP messaging service...
drp.host 127.0.0.1
drp.port 50747
drp.id hawc.pipe.step.crush
drp.fifo 100
drp.timeout 1.0
drp.timestamp

# When running in the SOFIA pipeline (assuming that TCP/IP messaging enabled
# via the 'drp' option), then disable PNG thumbnail generation to prevent any
# possinble internal Java errors due to missing/broken X11 display␣
↪connections.
#[drp] forget write.png
forget write.png
forget write.eps
forget gnuplot

# When running reductions via the DRP, smooth images slightly for better␣
↪visual
# appearance
#[drp] iteration.[last] smooth halfbeam
#iteration.[last] smooth halfbeam

# Use's Bill Vacca's ATRAN-based polynomial model for calculating opacity...
tau atran

# Use the measured PWV to calculate tau...
#tau pwv

# Calculate typical PWV values, instead of using the monitor data
#tau pwvmodel

# Set tau to 0; turn off calibration
#tau 0.0
blacklist calibrated
unit counts
```

```
# Use this model, whenever the pwv values aren't available or cannot
# be trusted...
date.[*--2016.12.01] [tau?pwv] tau pwvmodel
date.[2016.12.03--2016.12.04] [tau?pwv] tau pwvmodel

# Refer opacity relations to the PWV value (which is recorded)
tau.pwv.a 1.0
tau.pwv.b 0.0

# standard SOFIA options
#unit Jy/pixel
projection TAN

# Never smooth focus scans...
fits.[CALMODE?Focus] iteration.[last] blacklist smooth

# Reduce skydips if OBSMODE, CALMODE or DIAGMODE is set to SKYDIP
fits.[DIAGMODE?SKYDIP] skydip
fits.[OBSMODE?SkyDip] skydip

# Fit skydips on restricted elevation range only...
skydip.elrange 0:55

# For skydips, notch out the intcal signal (203.25 Hz / 68 -- and harmonics)
[source.type?skydip] notch
[intcalfreq?!-9999.0] notch.frequencies {?intcalfreq}
notch.width 0.03
notch.harmonics 35

# Produce a PNG plot for the skydip result?
#[source.type?skydip] write.png

# Downsample skydips by a fixed amount...
#[source.type?skydip] downsample 50

# Use only parts of the map for source flatfield estimation, which are within
# the specified S/N range...
source.coupling.s2n 5.0:500.0

# Set the observing band based on the SPECTEL1 header value
fits.[SPECTEL1?HAW_A] band A
fits.[SPECTEL1?HAW_B] band B
fits.[SPECTEL1?HAW_C] band C
fits.[SPECTEL1?HAW_D] band D
fits.[SPECTEL1?HAW_E] band E

# Load date-based configuration overrides...
[apr2016] config hawc+/2016-04.cfg
[oct2016] config hawc+/2016-10.cfg

# Load the appropriate configuration for each band
[band?A] config band-A.cfg
[band?B] config band-B.cfg
```

```
[band?C] config band-C.cfg
[band?D] config band-D.cfg
[band?E] config band-E.cfg

# Never segment scans if using them for determining flatfields.
[write.flatfield] blacklist segment

# If dealing with demodulated data, then load the appropriate
# settings for reducing it
fits.[PRODTYPE?demod] config demod.cfg

# Additional header keys to migrate into product headers from earliest scan...
fits.addkeys SCRIPTID, OBSMODE, CALMODE, MCEMAP, HWPSTART, NHWP, CHPONFPA,␣
↪DTHSCALE

# logging...
obslog.format date\t flight\t scanno\t band\t object\t ?skydip\t obsmins(f1)\
↪t chop.flag\t gyro.max(f1)\t ac.altkft(f1)\t tel.el(f1)\t env.pwv(f1)\t env.
↪tamb(f1)\t dfoc(f1)
```

Part X

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.cfg*. 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. This file is used
# by StepCheckhead.
#
# 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 = *
```

(continues on next page)

(continued from previous page)

```
    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
```

(continues on next page)

(continued from previous page)

```
    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
```

(continues on next page)

(continued from previous page)

```
    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

[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 = *
```

(continues on next page)

(continued from previous page)

```
    dtype = float
    [[drange]]
        min = -90.0
        max = 90.0

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

[TELVPA]
    requirement = *
    dtype = float
    [[drange]]
        min = -360.0
        max = 360.0

[UTCSTART]
    requirement = *
    dtype = str
```