Sciops Team Toolbook

April 12, 2000

This document contains tool specifications for code developed or under development by the ASCDS Sciops team. The SDS specifications are being updated during the development process and it takes time for the updates to be incorporated into the SDS toolbook.

Contents

1 aspect

Tool Number 1144

aca_calc_centr

Description: Calculate centroids and fluxes for ACA using first moment centroids or PSF-model fitting routines.

Parameters:

Inputs: ACA data

Outputs: ACA data

Processing:

This tool consist of two methods for determining centroids:

1. First moment centroids

2. PSF-model fitting centroids

1. First Moment Centroids

Read CCD-corrected star and fiducial light images (Npix x Npix arrays) from ACADATA. Use flux-weighted moment to find the center (Xfmc,Yfmsc) of distribution specified by y,z positions (ACA frame y,z row and column coordinates in pixel units) and s values (corrected number of electrons collected in a given pixel):

$$Y_{fmc} = \frac{\sum_{i=0}^{N} y_i s_i}{S} \tag{1}$$

$$Z_{fmc} = \frac{\sum_{i=0}^{N} z_i s_i}{S} \tag{2}$$

where S is a sum of all s values over good pixels in Npix x Npix array corresponding to the flux in electrons. Write first moment derived centroids and flux, to ACADATA.

2. PSF-model fitting centroids

Read CCD-corrected star and fiducial light images (Npix x Npix arrays) from ACADATA. Read the corresponding best model PSFs (eight Nsub x Nsub arrays) as derived using the 'Create_best_PSF' tool from ACACAL. Use bad pixels matrices to identify pixels which are not to be used in subsequent processing. For each star and fiducial light image use PSF-fitting routine to find the center of a distribution specified by Y,Z position (in pixel coordinates), and pixel values (B) in e-. Apply multidimensional minimization routines (e. g. Powell minimization routine) to a sum chi-squared statistics function to derive the optimal values for the following parameters: centroid coordinates (Yc,Zc), flux (Bsum) and background level (Bmin).

The inputs to the minimization routine are: - number of parameters to fit - starting guesses for parameters - the maximum and minimum value for search for each parameter

The sum chi-squared statistics function (func) used by the minimization routine is compiled as follows: - Calculate the PSF value for each pixel (i,j) in the Npix x Npix image (Mpsf(i,j)) by integrating over the model PSF Nsub x Nsub array and applying sub-pixel response.

- Subtract from this value the actual signal in the image in that pixel (B(i,j)) and calculate chi-square statistics assuming Gaussian statistics for the signal errors (Err(i,j)).

- Run over the pixels of each Npix x Npix image and calculate the sum of chi-squared statistics:

$$func = \sum_{i,j=1}^{N_{pix}} \frac{f(i,j)(Mpsf(i,j) - B(i,j))^2}{Err(i,j)^2} \tag{3}$$

where f(i,j) provides bad pixel flag (0 or 1). Use the first moment centroids and fluxes as determined using the first routine, and the ACA reported background, for position, flux and background initial guess, respectively. Provide the maximum and minimum value for search for each parameter to the minimization routine. The outputs from the minimization routine are:

- calculated centroid coordinates Yc,Zc

- calculated flux Bsum

- zero level Bmin

- fit chi-squared.

- average deviation map

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: aca_calc_centr

Spec Category: Aspect

Code Type: ASCDS

Code Source: GADS prototype

Tool Number 1149b

aca_filter_centr

Description:

Iteratively apply a smoothing filter to ACA centroids and magnitudes, and reject statistical outliers

Parameters:

aca_cent	Input/output ACACENT L1 data product
aca_cal	Input ACACAL L1 data product
T_sm_star	Smoothing time scale
T_sm_fid	Smoothing time scale
SG_order	Polynomial order for smoothing (0—2—4)
reject_sigma	Data rejection threshold (sigma)
reject_iter	Maximum number of rejection iterations
cent_alg	Use `cent_alg` centroids for smoothing

Inputs:

ACACENT - ACA image centroids
ACACAL - ACA calibration data

Outputs:

ACACENT (`ang_y_sm, ang_z_sm, status` columns)

Processing:

1. Read the tool parameters

2. Read the ACACAL data product (`acacal` parameter) and close

3. Open the ACACENT data product for reading

4. Read the primary header for the ACACENT product to define the following variables:

`img_type[MAX_IMG]`	Image type (fid, guide, etc) for each slot
`n_img[MAX_IMG]`	Number of images in file for each slot

   Note that the number of image slots (8) is called `MAX_IMG` for historical reasons. It would more consistent if it were called `MAX_SLT`.

5. For each of the active image slots, allocate `float` matrices:

   ```
   float  t_sample[MAX_IMG];
   float* ang_y[MAX_IMG];
   float* ang_z[MAX_IMG];
   float* mag[MAX_IMG];
   float* ang_y_sm[MAX_IMG];
   float* ang_z_sm[MAX_IMG];
   float* mag_sm[MAX_IMG];
   char*  rej_y[MAX_IMG];
   char*  rej_z[MAX_IMG];
   char*  rej_mag[MAX_IMG];
   ```

```
for (slt = 0; slt < MAX_IMG; slt++) {   /* step through 8 image slots (slt) */
  if (img_type[slt] == FID || img_type[slt] == GUIDE) {
    ang_y[slt] = calloc (n_img[slt], sizeof(float));
      .. and same for ang_z, ang_y_sm, etc.
  }
}
```

6. Read centroid and magnitude data

   - While there are data left, read in a row of angular centroids data from the ACACENT file.
   - If `(img_type[slt] == FID || img_type[slt] == GUIDE) && alg_typ[alg] == cent_alg` then copy the 2 columns `ang_y, ang_z`, and `counts` to the corresponding data array and increment `i_img[slt]`. `Counts` are converted to `mag` using

     `mag = acacal.mag0 - 2.5 * log10 (cnt_rate / acacal.cnt_rate_mag0);`

     where `cnt_rate = counts / acacent.t_int`.
   - If the centroid `status` flag is non-zero then set the corresponding values of `rej_y[slt]` and `rej_z[slt]` to one.

7. For each of the images, use the `time` column in ACACENT to determine the sample period `t_sample[slt]`. It should be sufficient to take the difference in `time` between any two successive images for each slot.

8. Close the ACACENT file.

9. Extract a smoothed description for each column of the fid light positions at each data point:

```
for (slt = 0; slt < MAX_IMG; i++) {
  if (img_type[slt] != FID && img_type[slt] != GUIDE)
    continue;                  /* Smooth/filter only fid and guide slots */
  t_sm = (img_type[slt] == FID) ? t_sm_fid : t_sm_star;
  n_sm = t_sm / t_sample[slt];  /* smoothing length in data points */
     /* and make sure it's well-defined and odd */

  if (SG_order == 0) {
    boxcar (ang_y[slt], ang_y_sm[slt], rej_y[slt],  n_img[slt], n_sm);
    boxcar (ang_z[slt], ang_z_sm[slt], rej_z[slt],  n_img[slt], n_sm);
    boxcar (mag[slt],   mag_sm[slt],  rej_mag[slt], n_img[slt], n_sm);
  } else {
    SG_smooth (ang_y[slt], ang_y_sm[slt], rej_y[slt], n_img[slt],
               n_sm, SG_order);
    SG_smooth (ang_z[slt], ang_z_sm[slt], rej_z[slt], n_img[slt],
               n_sm, SG_order);
    SG_smooth (mag[slt], mag_sm[slt], rej_mag[slt], n_img[slt],
               n_sm, SG_order);
  }
}
```

10. The smoothing methods are described in Numerical Recipes (2nd edition) Sec. 14.8

   (a) Savitzky-Golay smoothing filter (SG_smooth). In addition to the smoothing length parameter (n_sm), this method requires a polynomial order parameter (SG_order). If the value is different from 0, 2 or 4, a warning will be reported and the value 0 will be assumed.

   (b) Running boxcar (boxcar). This is selected by setting SG_order = 0, since the Savitzky-Golay with polynomial order of zero is a boxcar. However, the straightforward boxcar smoothing is probably faster than calling SG_smooth with SG_order = 0. Perhaps this special case could be handled within SG_smooth.

11. After smoothing, calculate the RMS of the residuals (e.g. ang_y - ang_y_sm). Flag any points that are more than reject_sigma deviations from the smoothed curve (e.g. rej_y[slt][i] = 1). If any points are rejected, recalculate the sigma and do the smoothing step again without the rejected points. Repeat this step until no additional points are rejected, up to a maximum of reject_iter iterations.

12. After each of the FID and GUIDE image slots have been smoothed, write the data back to the ACACENT data product:

   (a) Open ACACENT for read-write. Initialize to read and write the slt, alg, ang_y_sm, ang_z_sm, and status columns.

   (b) Read a row of centroid data. If
   `(img_type[slt] == FID || img_type[slt] == GUIDE) && alg_typ[alg] == cent_alg`
   then set ang_y_sm, ang_z_sm, and status appropriately using the arrays ang_y_sm[][], ang_z_sm[][], rej_y[][], and rej_z[][].

   (c) If the mag data point was rejected, then bit 5 of the status byte will be set. If the ang_y data point was rejected, then bit 6 of the status byte will be set. If the ang_z data point was rejected, then bit 7 of the status byte will be set. (Bit 7 is defined as the most significant bit).

   (d) Write the final RMS of the residuals (calculated previously) to the vector header keywords angynea<N> and angznea<N>, where <N> goes from 1 to 8.

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: aca_filter_centr

Spec Category: Aspect

Code Type: ASCDS

Code Source: New

Tool Number 1143

aca_corr_ccd

Description: Apply CCD corrections to ACA data.

Parameters:

Inputs: ACA data, Cal data

Outputs: ACA data

Processing:

1. ACA image gain factor (DN counts to electrons)

Convert each ACA image from ADC digital number (DN) counts to electrons. Each CCD quadrant has a temperature dependent gain (e^{-}/DN) in its output amplifier. ACA image pixel value are telemetered in raw ADC counts (DN). During preprocessing, the gain for each quadrant will be interpolated from ground calibration data to produce a value corresponding to the mean telemetered CCD temperature. Each pixel of each image will be multiplied by the gain of the quadrant in which it falls. It may occur that an image occupies one or more quadrants.

Multiply each image background value from ACA image telemetry by the gain appropriate for the lower-left central pixel of the image. This is called `aca_bgd_avg`.

2. Pixel screening

Mark any inactive CCD pixels, including bad or "mousebitten" pixels. Read bad pixel list from ACACAL data product. This has been pre-screened to include only bad pixels in the 64×64 pixel regions corresponding to ACA images. For each ACA image containing a bad pixel, set the appropriate bit to mark the pixel as inactive. If the ACA image is in the 6×6 mousebitten format, mark each of the corners as inactive.

3. Background subtraction

Read the set of $64{\times}64$ dark current images (e^{-}/sec) for each of the ACA images, from the ACACAL data product.

Read the `bgd_sub_method` and `star_compensation` parameters. The background subtraction method has two possible values: `uniform`, which means to ignore the dark current map and subtract a uniform background value (as in on-board processing); and `map`, in which a scaled version of the dark current calibration map is subtracted. The `star_compensation` parameter controls the method by which contaminating star or fid-light flux in the eight corner "background" pixels is removed: `none` means no compensation, as done on-board; `flux_only` means to use the image flux computed on-board to estimate the contaminating flux in background pixels; `flux_centroid` means to use both the image flux and image centroid to compensate the background value. For the latter two options, ground simulations will be used to determine empirical polynomial curves that provide the necessary compensation as a function of image centroid and/or flux. Two sets of curves will be computed, one for stars and one for fid lights.

If `star_compensation != none` then compensate the background for each image. Using the on-board image flux and centroid and the appropriate polynomial curve (for the image type and compensation method), calculate the compensation value and subtract it from the on-board background value. Store the compensated background value (`aca_comp_bgd_avg`) in the ACADATA data product. If no compensation was selected, then `aca_comp_bgd_avg` is set to the on-board value.

Subtract the background `bgd_scale[i][j]` from each image, based on `bgd_sub_method`:

`uniform` –

```
bgd_scale[i][j] = aca_comp_bgd_avg
```

`map` – For each background pixel which was used on-board to calculate the background value, extract the value of the corresponding pixel in the 64 × 64 dark current map for that image. The identity of background pixels used on-board is specified in the background pixel status byte in the ACA image telemetry. For the extracted dark current values (up to 8) calculate the mean (`dark_map_avg`). For each active image pixel, extract the corresponding dark current (`dark[i][j]`) and form a scaled background map defined by

```
bgd_scale[i][j] = dark[i][j] * aca_comp_bgd_avg / dark_map_avg
```

This equation *multiplies* the dark current map to make the measured average match the predicted average. The integration time is implicitly included in this scaling.

`map_mult` – Same as `map`. This form should be used, but `map` is retained for backward compatibility.

`map_abs` – Form an absolute background map defined by multiplying the dark current map (e-/sec) by the ACA integration time:

```
bgd_scale[i][j] = dark[i][j] * integ_time
```

`map_add` – Follow the procedure for the `map` (multiplicative method) to determine `dark_map_avg`. For each active image pixel, extract the corresponding dark current (`dark[i][j]`) and form an additively scaled background map defined by:

```
bgd_scale[i][j] = dark[i][j] * integ_time + (aca_comp_bgd_avg
                                             - dark_map_avg * integ_time)
```

This equation *adds* a constant value to the dark current map to make the measured average match the predicted average. By definition of `dark_map_avg`, the average value of `bgd_scale` over the valid background pixels is equal to `aca_com_bgd_avg`.

4. Responsivity correction

Read the 64 × 64) responsivity image for each ACA image from the ACACAL data product. The average value in this image has been normalized to unity.

For each background-subtracted ACA image, step through the active pixels and divide the pixel value by the corresponding responsivity image pixel. The final corrected images are stored in the ACADATA data product.

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: aca_corr_ccd

Spec Category: Aspect

Code Type: ASCDS

Code Source: New

Tool Number 1146

aca_corr_centr

Description: Apply corrections to star centroid positions.

Parameters:

Inputs: ACA data, ACA cal, ACA sight

Outputs: ACA sight

Processing:

1. Extract inputs from input data products

   a) The number of centroid methods used from ACASIGHT.

   b) The type of centroid methods used from ACASIGHT.

   c) The row/column centroid values from ACASIGHT.
      (1 set per image per centroid method)

   d) The row/column CTI correction values from ACACAL.
      (1 set per image)

   e) The row/column first moment correction values from ACACAL.
      (1 set per image)

   f) The Y/Z field distortion coefficients from ACACAL.
      (1 set of 20 coefficients each)

   g) The Camera Temperature from ACADATA.
      (1 value)

   h) The Y/Z color correction values from ACACAL.
      (1 set per image)

2. CTI Centroid Corrections
   Applied to the centroid values from each method, these corrections compensate for deviations in the centroid caused by the charge transfer of the star image though the CCD pixels.

   For each method,

   a) Apply the CTI correction to the image centroids.

   $$corr\_cent\_i = cent\_i + cti\_corr\_i \tag{4}$$

   $$corr\_cent\_j = cent\_j + cti\_corr\_j \tag{5}$$

   The sign of the correction factor should already take into account the node/CCD of the image so that the application is an additive offset.

3. First Moment Centroid Corrections
   Applied ONLY to the first moment centriod values only, this correction compensates for a biasing of the first moment calculation caused by the shape of the PSF.

   a) Determine which set of centroids are from the first moment method.

b) Apply the first moment correction to the image centroids.

$$corr\_cent\_i = corr\_cent\_i + fm\_corr\_i \tag{6}$$

$$corr\_cent\_j = corr\_cent\_j + fm\_corr\_j \tag{7}$$

The sign of the correction factor should be such that the application is an additive offset. NOTE: This correction is in addition to the CTI correction.

4. Convert centroid values to ACA angle coordinates
   Apply corrections for image centroid distortion (optical and/or CCD) and convert corrected centroids to ACA angles using model distortion functions. An example of a model distortion function is a 20-term third order calibration polynomial in each coordinate:

$$\begin{aligned} ang\_y &= P_y(R,C,T) && (8) \\ ang\_z &= P_z(R,C,T) && (9) \end{aligned}$$

   where R is row (corr_cent_i), C is column (corr_cent_j) and T is the camera temperature from ACADATA.

$$\begin{aligned} ang\_y \;=\; & A_{0y} + A_{1y}C + A_{2y}R + A_{3y}T + A_{4y}C^2 + A_{5y}CR + A_{6y}CT + A_{7y}R^2 \\ & + A_{8y}RT + A_{9y}T^2 + A_{10y}C^3 + A_{11y}RC^2 + A_{12y}TC^2 + A_{13y}CR^2 \\ & + A_{14y}CRT + A_{19y}T^3 + A_{16y}R^3 + A_{17y}TR^2 + A_{18y}RT^2 + A_{19y}T^3 \end{aligned} \tag{10}$$

$$\begin{aligned} ang\_z \;=\; & A_{0z} + A_{1z}C + A_{2z}R + A_{3z}T + A_{4z}C^2 + A_{5z}CR + A_{6z}CT + A_{7z}R^2 \\ & + A_{8z}RT + A_{9z}T^2 + A_{10z}C^3 + A_{11z}RC^2 + A_{12z}TC^2 + A_{13z}CR^2 \\ & + A_{14z}CRT + A_{19z}T^3 + A_{16z}R^3 + A_{17z}TR^2 + A_{18z}RT^2 + A_{19z}T^3 \end{aligned} \tag{11}$$

   The coefficients of these polynomials are determined during ground and on-orbit calibration of the Aspect Camera.

5. Color Corrections
   Applied to the centroids in ACA angle coordinates for all methods, this correction compensates for offsets caused by the ACA camera optics.

   a) Apply the Color correction to the image centroids.

$$ang\_y = ang\_y + color\_corr\_i \tag{12}$$

$$ang\_z = ang\_z + color\_corr\_j \tag{13}$$

   The sign of the correction factor should be such that the application is an additive offset.

6. Write the pixel corrected centroid positions to ACASIGHT

7. Write the ACA angle-corrected centroids to ACASIGHT

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: aca_corr_centr

Spec Category: Aspect

Code Type: ASCDS

Code Source: New

Tool Number 1145

aca_image_id

Description: Create GSPROPS and FIDPROPS data products. Identify active guide stars and fid lights using OBC image centroids and scheduled OR/ER data

Parameters:

fid_props	file	Output L1 FIDPROPS aspect data product file
gs_props	file	Output L1 FIDPROPS aspect data product file
pcad_eng	file	Input PCAD engineering file with ACA centroids
cal_align	file	Input ACA and FTS alignment file
n_samp	int	Samples of engineering data for forming averages
min_n_avg	int	Minimum image centroids to form average
marg_dist	float	Distance error for marginal identification (arcsec)
marg_angle	float	Angle error for marginal identification (degree)
bad_dist	float	Distance error for bad identification (arcsec)
bad_angle	float	Angle error for bad identification (degree)

Inputs:

PCADENG - PCAD engineering file with ACA centroids
CALALIGN - ACA and FTS alignment file from CALDB

Outputs:

FIDPROPS - L1 fiducial light properties aspect data product
GSPROPS - L1 guide star properties aspect data product

Processing:

1. Obtain the values of the 3×8 matrix `odb_guide_image` from the OCAT, valid for the observation at the current time. Store values in a variable of the same name. The column dimension (8) corresponds to the ACA image slot. Row 0 contains the image type indicator for each slot: a value of 1 →guide star, 2 →fid light, and 3 →monitor star. Any other value implies that the image slot is unused. Row 1 contains the Y angle of each image center in radians. Row 2 contains the Z angle of each image center in radians.

2. Open the PCAD engineering file containing ACA Y and Z angle telemetry, using FITS I/O structure identifier `pcadeng`.

3. Calculate the average Y and Z angle of the first `n_samp` samples of `pcadeng`, starting at time `tstart`. This is done only for active image slots which are "tracking", where `pcadeng.AOACFCT* == "TRAK"` and `pcadeng.AOIMAGE* != "NULL"`. The Y and Z angles are in `pcadeng.AOACYAN*` and `pcadeng.AOACZAN*`, in units of arcsec.

4. Identify active images as fid light or star, based on the value of `pcadeng.AOIMAGE*`. This has possible values FIDL or STAR. The number of active fid images is called `n_fid` and the number of active star images is `n_star`.

Steps 3 and 4 are represented in the following pseudo-code, which uses some Perl-like syntax:

```
for (i_samp = 0 .. n_samp-1)
  {
    PCADENG_GetRow (&pcadeng);

    for ($slot = 0 .. 7) {
      if (   pcadeng.AOACFCT${slot} == "TRAK"
          && pcadeng.AOIMAGE${slot} != "NULL")
        {
          yan[$slot] += pcadeng.AOACYAN${slot};
          zan[$slot] += pcadeng.AOACZAN${slot};
          img_type[$slot] = pcadeng.AOIMAGE${slot};
          n_avg[$slot]++;
        }
    }
  }

for (slot = 0 .. 7) {
  if (n_avg[slot] < min_n_avg)  /* need at least min_n_avg vals */
    continue;

  if (img_type[slot] == "STAR")
    {
      star_slot[n_star] = slot;
      star_y_ang[n_star] = yan[slot] / n_avg[slot];
      star_z_ang[n_star] = zan[slot] / n_avg[slot];
      n_star++;
    }
  else if (img_type[slot] == "FIDL")
    {
      fid_slot[n_fid] = slot;
      fid_y_ang[n_fid] = yan[slot] / n_avg[slot];
      fid_z_ang[n_fid] = zan[slot] / n_avg[slot];
      n_fid++;
    }
}
```

Star processing

5. If `n_star == 0`, then produce a fatal error message for the operator that no stars are found

6. If `n_star == 1`, then classify the star as "good" and produce a warning message for the operator that insufficient stars are found for an aspect solution, and go to step 13

7. For each star image `i`, calculate the distance (in arcsec) and polar angle (in degrees) to each other star image `j`, using the averaged Y and Z angles (Step 3):

   ```
   dist_obs[i][j] = sqrt((y_ang[i]-y_ang[j])^2 + (z_ang[i]-z_ang[j])^2)
   angle_obs[i][j] = atan2d( z_ang[i]-z_ang[j], y_ang[i]-y_ang[j] )
   ```

   These are the observed distances and angles for that star image. (For notational convenience, the `star_` prefix has been dropped.)

8. Similarly, calculate the distance `dist_pred` (arcsec) and polar angle `angle_pred` (degrees) for the predicted star positions (Step 1)

9. Initialize arrays `good[8], marg[8], bad[8]` to 0. For each star `i`, subtract the observed distances to all other stars `j` from the predicted values. Set `d_dist` to the absolute value of each difference. For the angle, calculate

```
d_angle = min( abs(angle_obs[i][j] - angle_pred[i][j] + 360.0),
               abs(angle_obs[i][j] - angle_pred[i][j]),
               abs(angle_obs[i][j] - angle_pred[i][j] - 360.0) )
```

   Here `min()` returns the minimum of the given argument list. The above calculation avoids possible errors for angles very near ±180. Then assign the classification for the star:

```
 if (d_dist < marg_dist && d_angle < marg_angle)    good[i]++;
 else if (d_dist < bad_dist && d_angle < bad_angle) marg[i]++;
 else bad[i]++;
```

10. If `n_star <= 4 && good[i] >= 1`, then star `i` is classified as “good”.
    If `n_star > 4 && good[i] >= 2`, then star `i` is classified as “good”.

11. If there are no “good” stars, then produce a warning that no star pairs with the predicted separation were found, and go to step 13

12. For each star that is not classified “good”, re-do step 9, comparing *only* with the “good” stars.

```
  if (bad[i] > 0) classification[i] = BAD;
  else if (marg[i] > 0) classification[i] = MARGINAL;
  else classification[i] = GOOD;
```

13. Open the GSPROPS file with WRITE access, with FITS I/O data structure `gsprops`.

14. For each of the eight ACA image slots `slot`, if the observation information from the OCAT indicates that the image slot contains a guide star, then proceed with the following step. Otherwise, continue with the next image slot. (Future release of aca_id_image will also deal with monitor stars).

15. Query the AGASC database with the `agasc_id` from the OCAT to determine the star properties. Assume the information is returned in a structure labeled `agasc`. Do

```
gsprops.agasc_id      = agasc_id          /* AGASC id number              */
gsprops.slot          = slot              /* Image slot                   */
gsprops.type          = "GUIDE"           /* Image type (guide|monitor)   */

gsprops.ra            = agasc.ra          /* Right ascension              */
gsprops.dec           = agasc.dec         /* Declination                  */
gsprops.pos_err       = agasc.pos_err     /* Position uncertainty         */
gsprops.pm_ra         = agasc.pm_ra       /* RA proper motion             */
gsprops.pm_dec        = agasc.pm_dec      /* Dec proper motion            */
gsprops.plx           = agasc.plx         /* Parallax                     */
gsprops.plx_err       = agasc.plx_err     /* PLX uncertainty              */
gsprops.mag_aca       = agasc.mag_aca     /* ACA instrumental magnitude   */
gsprops.mag_aca_err   = agasc.mag_aca_err /* MAG_ACA uncertainty          */
gsprops.star_class    = agasc.class       /* Classification of star       */
gsprops.mag           = agasc.mag         /* Magnitude                    */
gsprops.mag_err       = agasc.mag_err     /* MAG uncertainty              */
gsprops.mag_band      = agasc.mag_band    /* MAG bandpass code            */
gsprops.color1        = agasc.color1      /* Catalog or est'd B-V color   */
gsprops.color1_err    = agasc.color1_err  /* COLOR1 uncertainty           */
gsprops.var           = agasc.var         /* Variability code             */
gsprops.aspq1         = agasc.aspq1       /* Spoiler status code          */
gsprops.aspq2         = agasc.aspq2       /* Spoiler proper motion code   */
gsprops.aspq3         = agasc.aspq3       /* Spoiler position code        */

gsprops.spectral_type = "NONE"            /* Spectral type (TBR!)         */
gsprops.ra_corr       = agasc.ra          /* RA with aspect corrections   */
gsprops.dec_corr      = agasc.dec         /* Dec with aspect corrections  */
gsprops.ra_offset_obs = 0.0               /* Offset from ra_corr (obs'd)  */
gsprops.dec_offset_obs= 0.0               /* Offset from dec_corr (obs'd) */
gsprops.mag_aca_avg   = 0.0               /* ACA magnitude (avg obs'd)    */
gsprops.mag_aca_min   = 0.0               /* ACA magnitude (min obs'd)    */
gsprops.mag_aca_max   = 0.0               /* ACA magnitude (max obs'd)    */
gsprops.aspq1_obs     = 0                 /* Spoiler status code (obs'd)  */
gsprops.spoil_radius  = 0.0               /* Spoiler star radius (obs'd)  */
gsprops.spoil_angle   = 0.0               /* Spoiler star angle (obs'd)   */
gsprops.spoil_mag_aca = 0.0               /* Spoiler star ACA mag (obs'd) */
gsprops.pos_eci_eff[0]= cosd(agasc.ra)*cosd(agasc.dec)
gsprops.pos_eci_eff[1]= sind(agasc.ra)*cosd(agasc.dec)
gsprops.pos_eci_eff[2]= sind(agasc.dec)   /* Effective pos. vector (ECI)  */

if (slot has an active star, i.e. slot == star_slot[i] for some i)
  {
    if (classification[i] == GOOD)
      gsprops.id_status     = "GOOD"      /* (GOOD|MARGINAL|BAD|NO_TRACK)*/
    else if (classification[i] == MARGINAL)
      gsprops.id_status     = "MARGINAL"
    else if (classification[i] == BAD)
      gsprops.id_status     = "BAD"
    else
      gsprops.id_status     = ""
    gsprops.cel_loc_flag  = (classification[i] == GOOD)
                                           /* Use for celestial location   */
  }
else  /* slot not in star_slot[] array, which means slot is not tracking and
         guide star was not acquired for some reason */
  {
    gsprops.id_status     = "NO_TRACK"
    gsprops.cel_loc_flag  = 0
  }
```

16. Write the row of GSPROPS data to the file

Fid processing

1. If `n_fid == 0`, then produce a fatal error message for the operator that no fids are found

2. If `n_fid == 1`, then classify the fid as “good” and produce a warning message for the operator that insufficient fids are found for an aspect solution, and go to step 12

3. If `n_fid > 4`, then produce a fatal error message for the operator that too many fids are found

4. For each fid image `i`, calculate the distance (in arcsec) and polar angle (in degrees) to each other fid image `j`, using the averaged Y and Z angles (Step 3):

   ```
   dist_obs[i][j] = sqrt((y_ang[i]-y_ang[j])^2 + (z_ang[i]-z_ang[j])^2)
   angle_obs[i][j] = atan2d( z_ang[i]-z_ang[j], y_ang[i]-y_ang[j] )
   ```

   These are the observed distances and angles for that fid image.

5. Similarly, calculate the distance `dist_pred` (arcsec) and polar angle `angle_pred` (degrees) for the predicted fid positions (Step 1)

6. Initialize arrays `good[8], marg[8], bad[8]` to 0. For each fid `i`, subtract the observed distances to all other fids `j` from the predicted values. Set `d_dist` to the absolute value of each difference. For the angle, calculate

   ```
   d_angle = min( abs(angle_obs[i][j] - angle_pred[i][j] + 360.0),
                  abs(angle_obs[i][j] - angle_pred[i][j]),
                  abs(angle_obs[i][j] - angle_pred[i][j] - 360.0) )
   ```

   Here `min()` returns the minimum of the given argument list. The above calculation avoids possible errors for angles very near ±180. Then assign the classification for the fid:

   ```
    if (d_dist < marg_dist && d_angle < marg_angle)    good[i]++;
    else if (d_dist < bad_dist && d_angle < bad_angle) marg[i]++;
    else bad[i]++;
   ```

7. If `good[i] >= 1`, then fid `i` is classified as “good”.

8. If there are no “good” fids, then produce a warning that no fid pairs with the predicted separation were found, and go to step 12

9. For each fid that is not classified “good”, re-do step 6, comparing *only* with the “good” fids.

   ```
     if (bad[i] > 0) classification[i] = BAD;
     else if (marg[i] > 0) classification[i] = MARGINAL;
     else classification[i] = GOOD;
   ```

10. Query the OCAT (or appropriate database) to determine the science instrument for this observation. Call this `obs_si`, and assure that it is a string with possible values `ACIS-S, ACIS-I, HRC-S, HRC-I`.

11. Open the CALALIGN file (READ access) with FITS I/O data structure identifier `calalign`. Read the first (and only) row of the first table extension (extname = `ALIGN`).

12. Open the FIDPROPS file (WRITE access) with FITS I/O data structure identifier `fidprops`

13. Set `fidprops.lsi0_stt` to the correct choice of the `calalign` variables `as_lsi0_stt`, `ai_lsi0_stt`, `hs_lsi0_stt`, or `hi_lsi0_stt`, depending on the value of `obs_si`. Then do

```
fidprops.n_fid = n_fid
fidprops.rrc0_fc_x = calalign.rrc0_fc_x
```

14. Set `fidprops.stt0_stf` by TBD method. (*Probably from L0.5 SIM engineering*).

15. For each of the eight ACA image slots `slot`, if the observation information from the OCAT indicates that the image slot contains a fid light, then proceed with the following steps. Otherwise, continue with the next image slot.

16. Set `fid_id` to the fid light identifier (1-14) from the OCAT.

17. Find the entry in the second extension (`FIDPOS`) of the CALALIGN file which has `obs_si == calalign.si && fid_id == calalign.fid_id`, and do

```
fidprops.p_lsi          = fid_pos_lsi /* Fid position in LSI frame    */
fidprops.slot           = slot       /* Fid image slot number        */
fidprops.mag_i_cmd      = 0.0       /* Commanded instrument mag     */
fidprops.mag_i_avg      = 0.0       /* Instr. magnitude (avg obs'd) */
fidprops.mag_i_min      = 0.0       /* Instr. magnitude (min obs'd) */
fidprops.mag_i_max      = 0.0       /* Instr. magnitude (max obs'd) */
fidprops.id_string      = obs_si . "-" . calalign.fid_num_si
                                      /* Id (e.g. HRC-I-2, ACIS-5)    */
fidprops.id_num         = calalign.fid_num   /* Id fid number (1-14)  */

if (slot has an active fid, i.e. slot == fid_slot[i] for some i)
  {
    fidprops.id_status  = (classification[i] == GOOD) ? "GOOD" : "BAD"
                                      /* Id status (GOOD|BAD|NO_TRACK)*/
  }
else
  {
    fidprops.id_status  = "NO_TRACK"
  }
```

18. Write the row of FIDPROPS data to the file

File contents:

FIDPROPS Data Product Definition

Name	Type	Comment	Units	Dims
Principle Header Keywords				
lsi0stt	double	LSI origin in STT frame	mm	3
stt0stf	double	STT origin in STF frame	mm	3
rrc0fcx	double	RRC origin in FC (X value)	mm	
nfid	int	Number of active fid lights		
Binary Table Extension				
p_lsi	double	Fid position in LSI frame	mm	3
slot	int	Fid image slot number		
mag_i_cmd	float	Commanded instrument mag		
mag_i_avg	float	Instr. magnitude (avg obs'd)	mag	
mag_i_min	float	Instr. magnitude (min obs'd)	mag	
mag_i_max	float	Instr. magnitude (max obs'd)	mag	
id_string	char	Id (e.g. HRC-I-2, ACIS-5)		10
id_num	int	Id fid number (1-14)		
id_status	char	Id status (good—marginal—bad)		10

GSPROPS Data Product Definition

Name	Type	Comment	Units	Dims
Principle Header Keywords				
Binary Table Extension				
agasc_id	int	AGASC id number		
slot	int	Image slot		
type	string	Image type (guide—monitor)		15
ra	double	Right ascension	deg	
dec	double	Declination	deg	
pos_err	short	Position uncertainty	marcsec	
pm_ra	short	RA proper motion	marcsec/yr	
pm_dec	short	Dec proper motion	marcsec/yr	
plx	short	Parallax	marcsec	
plx_err	short	PLX uncertainty	marcsec	
mag_aca	float	ACA instrumental magnitude	mag	
mag_aca_err	float	MAG_ACA uncertainty	mag	
class	short	Classification of star		
mag	float	Magnitude	mag	
mag_err	short	MAG uncertainty	cmag	
mag_band	short	MAG bandpass code		
color1	float	Catalog or est'd B-V color	mag	
color1_err	short	COLOR1 uncertainty	cmag	
var	short	Variability code		
aspq1	short	Spoiler status code		
aspq2	short	Spoiler proper motion code		
aspq3	short	Spoiler position code		
id_status	string	Id status (good—marginal—bad)		10
spectral_type	string	Spectral type		15
ra_corr	double	RA with aspect corrections	deg	
dec_corr	double	Dec with aspect corrections	deg	
ra_offset_obs	float	Offset from ra_corr (obs'd)	deg	
dec_offset_obs	float	Offset from dec_corr (obs'd)	deg	
mag_aca_avg	float	ACA magnitude (avg obs'd)	mag	
mag_aca_min	float	ACA magnitude (min obs'd)	mag	
mag_aca_max	float	ACA magnitude (max obs'd)	mag	
aspq1_obs	short	Spoiler status code (obs'd)		
spoil_radius	float	Spoiler star radius (obs'd)	arcsec	
spoil_angle	float	Spoiler star angle (obs'd)	deg	
spoil_mag_aca	float	Spoiler star ACA mag (obs'd)	mag	
pos_eci	double	Effective pos. vector (ECI)		3
cel_loc_flag	Byte	Use for celestial location		

CALALIGN Data Product Definition

Name	Type	Comment	Units	Dims
Principle Header Keywords				
Binary Table Extension (ALIGN)				
aca_sc_align	double	ACA to S/C nominal alignment		3 × 3
aca_misalign	double	ACA misalignment matrix		3 × 3
fts_misalign	double	FTS misalignment matrix		3 × 3
as_lsi0_stt	double	ACIS-S LSI origin in STT frame	mm	3
ai_lsi0_stt	double	ACIS-I LSI origin in STT frame	mm	3
hs_lsi0_stt	double	HRC-S LSI origin in STT frame	mm	3
hi_lsi0_stt	double	HRC-I LSI origin in STT frame	mm	3
rrc0_fc_x	double	RRC origin in FC (X value)	mm	
Binary Table Extension (FIDPOS)				
fid_num	int	Fid number (1-14)		
fid_si	char	Fid SI (ACIS-S, HRC-I etc)		6
fid_num_si	int	Fid number wrt SI (1-6 or 1-4)		
fid_pos_lsi	double	Fid position in LSI	mm	3

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: aca_image_id

Spec Category: Aspect

Code Type: ASCDS

Code Source: New

Tool Number 1158

aca_make_psf

Description: Derive the ACA PSF's for use in centroiding. Determine location of spoiler star if necessary.

Parameters:

aca_data	file	Input L1 ACADATA data product file
aca_cal	file	Input/output L1 ACACAL data product file
det_spoiler	logical	Detect spoilers (true), else PSF lib. interpolation
aca_cent_temp	file	Temporary L1 ACACENT file
asp_sol_obs	file	OBC aspect solution file
tstart	double	
t_stack	double	Length of time for image stack (sec)
n_grid	int	Grid points for spoiler initial position
grid_edge	float	Distance from image edge for initial position grid
chisq_limit	float	Chiˆ2 (per DOF) limit for possible spoiler

Inputs:

ASPSOL - OBC ASPSOL data product file
ACADATA - ACADATA data product file
ACACENT - ACACENT data product file
ACACAL - ACACAL data product file

Outputs:

ACACAL - ACACAL data product file

Overview: This tool consist of two steps. Step 1 is done if `det_spoiler = FALSE`, otherwise step 2 is carried out.

1. Find the optimal model PSF using the PSF library
2. Detect spoiler stars and make composite model PSF

Find the optimal model PSF using the PSF library

The PSF library consist of two 9 x 9 grids of model PSFs (in 81 uniformly distributed locations on the CCD), one for star images , another for fiducial light images. The fid light PSFs are monochromatic, calibrated at one wavelength (635 nm), while for stars it may be necessary to add a third dimension in the matrix - star colors, to account for optical color effects on the PSFs. Each model PSF is sampled on a sub-CCD pixel scale and consists of Nsub x Nsub pixels (128x128 sub-CCD pixels) corresponding to ten CCD pixels. These model PSFs have been created using ray tracing or calibration methods.

Determine the approximate position of a star (or fid light) on the CCD (in CCD pixels coordinates) by using the lower left corner row and column positions of each ACA array and assuming that the star (or fid light) image is located in the center of the Npix x Npix array. Derive the best model

PSF for each star and fid light by linear interpolation between the nearest PSFs in the appropriate matrix in the PSF library. If available, use the (B-V) color to select nearest star color for the PSF.

Detect spoiler and make composite model PSF

For this step use only `t_stack` seconds of the observation data (typically 100 seconds) to determine if there is a spoiler. It is assumed that the file given by `acacent_temp` contains PSF centroids for this duration.

Apply PSF-model fitting routine from the aca_calc_centr tool to this data subset and determine the centroid for each star and fiducial light and derive the fitting statistics. Based on the fitting statistics determine if there are possible spoilers (larger then expected errors). If no evidence of a spoiler is detected for a given star the optimal model PSF for subsequent fitting is simply the interpolated library PSF . If the fitting statistics indicates larger errors than expected, and therefore possible presence of a spoiler star derive a composite model PSF for that star and the spoiler. In this case the best PSF is the composite model PSF.

Using the optimal model PSFs, and 100 s (typically) of ccd-level corrected ACA and OBC (on-board computer) aspect solution, apply the PSF-fitting algorithm using the PSF-model fitting routine on a stack of images and derive corrected centroids of a star.

The composite PSF model is derived in the following manner:

1. If `tstart == 0` or is undefined, set it to the `aspsol.time` for the first record in the file
2. Open the ASPSOL file (`asp_sol_obc`) with data I/O structure `aspsol`. Read the ASPSOL records with time between `tstart` and `tstart + t_stack`. Store the set of values `aspsol.time` in an array `aspsol_time[]`, and store the set of `aspsol.q_att[0..3]` in `aspsol_q_att[][0..3]`.
3. Open the ACACAL file (`aca_cal`) with data I/O structure `acacal`, with READWRITE access. Read the first (and only) row.
4. Open the ACACENT file (`aca_cent_temp`) with data I/O structure `acacent`
5. Open the GSPROPS file (`gs_props`) with data I/O structure `gsprops`, in order to determine the active guide stars
6. Read an entry in the GSPROPS file and check that `gsprops.type == "guide"`. If so, carry out the following steps, otherwise read the next entry until EOF

   (a) Read ACACENT file between `time = tstart` to `tstart + t_stack` and select entries which have:

   ```
      acacent.slot == gsprops.slot
   && acacent.alg_typ[acacent.alg] == PSF
   && (acacent.status == IMG_ALG_STATUS_OK || acacent.status == IMG_ALG_BAD_CENTR)
   ```

   Store the set of values `acacent.chi` in the array `acacent_chisq[]`, and `acacent.time` in the array `acacent_time[]`.

   (b) If there are less than 4 available values, issue a warning message and begin processing for the next guide star (step 6). Calculate the median of the array `acacent_chisq[]`. If this value is less than `chisq_limit`, then continue with the next guide star (step 6).

   (c) Using routines from the PSF interpolation part of aca_make_psf, calculate the interpolated PSF for the current image location (`acacal.img_loc_i` and `acacal.img_loc_j`). In this case use the reddest available spectral type from the PSF library. Label this PSF `psf2`.

(d) Initialize the binned PSF arrays (needed for fitting), to accomodate the two PSFs: the star PSF (acacal.psf_lib[slot]) which we label psf1, and the spoiler PSF psf2.

(e) Read the image pixel data for the current image slot (slot) from the appropriate ACADATA file. Read data between time = tstart to tstart + t_stack. Select only the ACADATA records for which acadata.time has a corresponding entry in the acacent_time[] array. Store these selected records in an array of acadata records called acadata_arr.

(f) Calculate the entire array of pixel offsets for star, using aspsol_q_att[] and gsprops.pos_eci_eff and ACADATA to generate offset_i[] and offset_j[] corresponding to each centroid.

   i. For each time in the acacent_time[] array, set i_aspsol_time to the index for which aspsol_time[i_aspsol_time] is nearest to time.

   ii. Set q_att = Quat (aspsol_q_att[i_aspsol_time]), where Quat is the initializer for the quaternion class. Initialize the quaternion transform matrix with q_att.SetTransform().

   iii. Calculate the predicted star pixel location with the following:

```
// Calculate the normalized direction vector for this star,
// by rotating the star vector in J2000 coordinates by the
// S/C attitude rotation matrix
d_aca = q_att.T * gsprops.pos_eci_eff; // d_aca is a column 3-vector
                                       // q_att.T is a 3x3 rotation matrix
                                       // pos_eci_eff is a column 3-vector

// Calculate y_angle and z_angle in the ACA field of view (degrees)
yag = atan2d (d_aca[1], d_aca[0]);
zag = atan2d (d_aca[2], d_aca[0]);

// Point to inverse field distortion coeffs (for notational convenience)
float* r = acacal.fd_r_star;
float* c = acacal.fd_c_star;

// Calculate average ACA temperature
T = (acadata_arr[n].temperatures[1] + acadata_arr[n].temperatures[2]
     + acadata_arr[n].temperatures[3]) / 3.0;

// Calculate row (i) and col (j) as function of y and z angle
ccd_i[n] = r[0] + r[1]*zag + r[2]*yag + r[3]*T + r[4]*zag*zag
      + r[5]*zag*yag + r[6]*zag*T + r[7]*yag*yag + r[8]*yag*T + r[9]*T*T
      + r[10]*zag*zag*zag + r[11]*zag*zag*yag + r[12]*zag*zag*T
      + r[13]*zag*yag*yag + r[14]*zag*yag*T + r[15]*zag*T*T
      + r[16]*yag*yag*yag + r[17]*yag*yag*T + r[18]*yag*T*T;

ccd_j[n] = c[0] + c[1]*zag + c[2]*yag + c[3]*T + c[4]*zag*zag
      + c[5]*zag*yag + c[6]*zag*T + c[7]*yag*yag + c[8]*yag*T + c[9]*T*T
      + c[10]*zag*zag*zag + c[11]*zag*zag*yag + c[12]*zag*zag*T
      + c[13]*zag*yag*yag + c[14]*zag*yag*T + c[15]*zag*T*T
      + c[16]*yag*yag*yag + c[17]*yag*yag*T + c[18]*yag*T*T;

// Calculate relative offset from first point
offset_i[n] = ccd_i[n] - ccd_i[0];
offset_j[n] = ccd_j[n] - ccd_j[0];
```

```
n++;
```

(g) Fit for the location of a potential spoiler star using the procedure described in M. Birkinshaw's memo "Spoiler code explanation", and prototyped by T.Aldcroft. This should use a grid of initial spoiler positions with `n_grid` points, spaced evenly between pixel coordinates `grid_edge` and `img_n_pix - 1.0 - grid_edge`.

(h) If the two-star-fit χ^2_ν (per DOF, from step 6g) is greater than the median χ^2_ν obtained in step 6b, then begin processing the next guide star in step 6. In this case the addition of a spoiler star has not improved the fit.

(i) Create the new composite PSF for this image slot, using the two PSFs and the relative offset determined in step 6g. The composite PSF is formed by starting with the original star PSF determined by PSF interpolation, and adding the spoiler star PSF at the offset found in the two-star-fit:

```
// Calculate relative contributions to PSF, based on
// best fit fluxes from image stack fit

scale1 = flux1 / (flux1 + flux2);
scale2 = flux2 / (flux1 + flux2);

// Copy original star PSF into acacal corrected PSF

for (i = 0; i < psf_n_pix; i++) {
  for (j = 0; j < psf_n_pix; j++) {
    acacal.psf_corr[slot][i][j] = psf1[i][j] * scale1;
  }
}

// Calculate offsets in PSF pixel coordinates

psf_i_offset = nint((cent_i2-cent_i1)/psf_pix_scale);
psf_j_offset = nint((cent_j2-cent_j1)/psf_pix_scale);

// Add scaled spoiler PSF, checking array bounds

for (i = 0; i < psf_n_pix; i++) {
  for (j = 0; j < psf_n_pix; j++) {
    i2 = i + psf_i_offset;
    j2 = j + psf_j_offset;
    if (i2 >= 0 && i2 < psf_n_pix && j2 >= 0 && j2 < psf_n_pix)
      {
        acacal.psf_corr[slot][i2][j2] += psf2[i][j] * scale2;
      }
  }
}
```

7. Reset the ACACAL write pointer (`acacal.wr_row = 0`) and write the updated `acacal` record back to the file.

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: asp_make_psf

Spec Category: Aspect

Code Type: ASCDS

Code Source: Prototype

Tool Number 1141

aca_read_data

Description:

The content of the ACA image telemetry stream differs depending on the image size. This tool combines elements from the L0 ACA image data with PCAD engineering data to produce a consistant data product which is used as input to the aspect determination pipeline. This dataset is designated ACADATA. A series of consistancy and data quality checks are applied during the process. The user has the option of applying a time filter in order to select only a portion of the data for processing.

Parameters:

telemfile	file		L0 ACA Image Data
engfile	file		L0 PCAD Engineering Data (ACA1)
obcfile	file		L0 PCAD OBC Data
datafile	file		Output ACADATA Data Product
tstart	double	hidden	Start of time filter
tstop	double	hidden	End of time filter
debug	int	hidden	Debug output level (0-5)

Inputs:

Stack of L0 ACA Image Data (TELIMG)
L0 PCAD Engineering Data (TELACA1)
L0 PCAD OBC Data (TEL????)

Outputs:

Stack of L1 ACA Image Data (ACADATA)

Processing:

The following is to be performed on each input L0 aspect image file. If a stack of files in input, a stack of files should be output. The input files CANNOT be combined into a single output file since they may contain images from different image slots and/or be of different dimension.

1. Open the input aspect image data file (TELIMG).
2. Determine the dimension of the image from the column definition for the imgraw column. This value will be stored in the ACADATA file as keyword IMGSIZE.
3. Create an output ACADATA file corresponding to the input file.
4. TELIMG header to ACADATA header
   Transfer the following elements from the input file header to the ACADATA file header.

TELIMG keyword	ACADATA keyword
(TBD)	

For all data records within the specified time filter, do the following:

5. TELIMG data to ACADATA header.
   Verify the consistancy of the TELIMG data elements listed below and add them to the ACADATA file header as the specified keywords.

   Images of size ¿ 4x4 require more than 1 telemetry segment to transmit the entire image, (6x6 = 2 segments, 8x8 = 4 segments). The telemetry stream repeats 3 values in each segment of an image (FID, IMGNUM, and FUNCTION). Telemetry decom retains each instance of these values in separate columns of the L0 aspect image data file. For these items, verify that they are consistant within a record as well as between records.

TELIMG columns	ACADATA keyword
integ	tint
imgnum*	imgnum
imgfid*	imgtype

6. TELIMG data to ACADATA data.
   Transfer the following data items to the ACADATA table. Perform any unit or type conversions necessary to comply with the ACADATA data product specification.

TELIMG columns	ACADATA column	
time	time	
global	global_status	
commcnt	cmd_count	
commprog	cmd_progress	
imgrow0	img_row0	
imgcol0	img_col0	
bgdavg	bgd_avg	
imgfunc*	img_func_stat	(upper 2 bits)
imgstat	img_func_stat	(lower 6 bits)
imgraw	img_raw	
if image dimension is not 4x4		
bgd_rms	bgd_rms	
tempccd	temperatures[0]	
temphous	temperatures[1]	
tempprim	temperatures[2]	
tempsec	temperatures[3]	
bgdstat	bgd_outliers	

7. Fill in from PCAD engineering data

For data with 4x4 image size, several items are not contained in the image data file. These items must be obtained from the PCAD engineering data. Select the PCAD engineering record closest in time to the image data record and transfer the following elements.

TELACA1 column		ACADATA column
aacccdpt OR aacccdrt	(degC)	temperatures[0] (degC)
aach1t	(degC)	temperatures[1] (degC)
aaotalt	(degC)	temperatures[2] (degC)
aaotasmt	(degC)	temperatures[3] (degC)

Choose the primary or redundant CCD temperature (aacccdpt and aaccdrt respectively) based on the value of the aflpmir column. If 0 (primary) choose aaccdpt, if 1 (redundant) choose aaccdrt. Verify that the value of the aflpmir column does not change during the specified time filter and write this value to the ACADATA file as the keyword AFLPMIR.

Set the background pixel status byte (bgdstat) to 255. This item is not available for 4x4 images.

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: aca_read_data

Spec Category: Aspect

Code Type: ASCDS

Code Source: New

Tool Number 2041

asp_calc_photo

Description: Calculate the aspect solution using the "photographic" method

Parameters:

tstart	double	optional	Start time (sec)
tstop	double	optional	Stop time (sec)
ra_nom	double	optional	Nominal RA (deg)
dec_nom	double	optional	Nominal dec (deg)
roll_nom	double	optional	Nominal roll (deg)
photo_interval	float		Photo interval duration (sec)
n_overlap	int		Number of overlapping intervals
kin_sol_method			Kinematic solution (aspsol|gyro_raw|gyro_corr|gyro_bias)
write_gyro_bias			Write gyro bias to GYRODATA (only for gyro_bias method)
roll_tol	float		Fit tolerance for roll (arcsec)
pitch_yaw_tol	float		Fit tolerance for pitch & yaw (arcsec)
roll_step	float		Fit step size for roll (arcsec)
pitch_yaw_step	float		Fit step size for p & y (arcsec)
gyro_data_file	file	optional	Gyro data L1 file
aca_cal_file	file	optional	ACA calibration L1 file
asp_sol_file	file	optional	Aspect solution L1 file (from OBC or Kalman)
gs_props_file	file	optional	Guide star properties L1 file
aca_cent_file	file		ACA centroids L1 file
photo_sol_file	file		Output photo method solution L1 file
debug_level	int		Debug level

Input files:

GYROCAL	-	Gyro calibration L1 file
GYRODATA	-	Gyro data L1 file
ASPSOL	-	L1 aspect solution data product
GSPROPS	-	L1 guide star properties aspect data product
ACACENT	-	L1 ACA centroids data product
FIDPROPS	-	L1 fiducial light properties aspect data product

Output file:

PHOTOSOL - Aspect solution in same format as Kalman

Overview:

The asp_calc_photo tool solves for the aspect solution using the "photographic" method. In this method, gyro data are used over a short interval (typically 100 s) in order to remove the spacecraft motion and stack star centroids relative to a common reference time. The reference time is the center of the interval. This stack of motion-corrected ACA centroids is analogous to a photograph of the star field. For each star in the photograph, the average star position is calculated, thus minimizing temporal and spatial centroiding noise. Then the constellation of averaged star positions in the photograph is compared a reference constellation to derive relative attitude of the spacecraft at the interval midpoint. For image reconstruction, the reference constellation is the set of observed stars during the first interval. For absolute celestial location, the reference is derived from the catalog RA and Dec of the expected guide stars. Finally, the attitude solution is derived by propagating the gyro-derived spacecraft motion forwards and backwards from the midpoint attitude.

Inputs:
This tool is designed to be very flexible in the inputs it uses, in order to accomodate a range of situations. This is done with the following key parameters:

- `kin_sol_method`

  In order to stack ACA centroids to create a photograph, it is necessary to know the spacecraft motion. This is derived essentially from gyro measurements, but the key complication is getting the gyro bias. There are several possible solutions, which are selected via the `kin_sol_method` parameter:

  `gyro_raw` - Integrate only raw gyro data, ignoring the bias (i.e. pretend it is zero). This may be adequate if the bias is small and slowly varying during an interval.

  `gyro_corr` - Integrate bias-corrected gyro data which has been determined either via the OBC estimate or the ground Kalman filter estimate. (Technically, it is the bias-corrected spacecraft rotation rate vector.)

  `gyro_bias` - Integrate raw gyro data and use ACA centroid deviations over the interval to determine gyro bias. A slowly-varying bias will introduce a systematic deviation which is linear with time. The gyro bias rates for all intervals are smoothed and interpolated over the entire time range (`tstart` to `tstop`). Then the same processing as for the gyro_corr method is done. If `write_gyro_bias = yes` then the column `sc_rate_corr` in GYRODATA is updated.

  `asp_sol` - Use an existing aspect solution file, either based on the OBC solution, or the ground Kalman filter solution. This may seem circular, but in fact over the normal length of a photo interval, the aspect solution is primarily dependent on the gyro rate data, with ACA data mostly influencing the gyro bias. This is the preferred kinematic solution method. If this method is chosen, and the aspect solution is based on the ground Kalman filter, then the L1 ACACAL file (`aca_cal_file`) is required in order to rotate the aspect solution from the MNC frame to the ACA frame.

  The existing prototype implements only the `asp_sol` method. The `gyro_raw` and `gyro_corr` methods are to be implemented in this release by write a temporary pseudo-ASPSOL file based on gyro data. The `gyro_bias` method will be done in a post-launch update, if necessary.

- `ra_nom, dec_nom, roll_nom`

  A nominal pointing direction is needed in order to roughly orient the aspect solution in absolute celestial coordinates. The nominal values are taken from `ra_nom, dec_nom` and `roll_nom`, unless any of them are blank, in which case the nominal reference attitude shall be taken from the aspect solution header. It is a fatal error if the nominal pointing parameters are blank for any kinematic solution method except `asp_sol`.

  The nominal pointing direction is used in one of two ways:

  - If a guide star properties file (`gs_props_file`) has been specified, then the nominal pointing roughly determines the expected position of guide stars in the ACA field of view. This must be sufficiently close to the initial observed positions allow the fitting routine to converge properly. In this case the attitude solution represents the true aspect in absolute celestial coordinates.

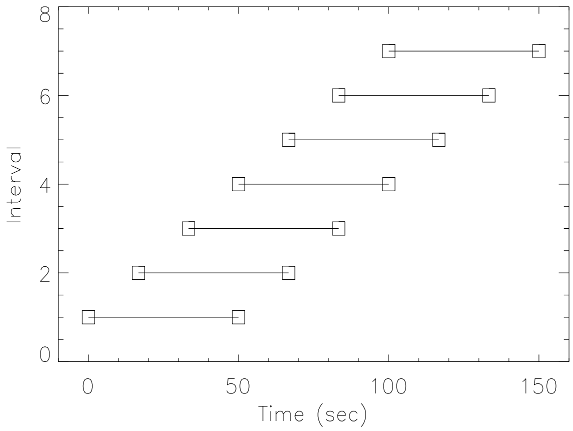

Figure 1: Example of overlapping intervals

- If no guide star properties file is available, then the nominal pointing is taken to define the spacecraft attitude at the midpoint of the first interval. This allows for X-ray image reconstruction to proceed, but with a fictitious (or approximate) celestial location.

Output:
This tool calculates the attitude history in the ACA reference frame, and produces as output a file in the KALMAN data product format. It does not process the fid light motions. Thus this tool is analogous to the Kalman filter / smoother processing, and the output must be given to `asp_solve` to generate a complete aspect solution.

Intervals:
A key concept in `asp_calc_photo` is that of the photo interval. This is not to be confused with an aspect interval. A photo interval is the interval of time over which ACA readouts of star centroids are accumulated to create a single photograph. Within an interval, gyro data are used to correct for the spacecraft motion and stack star centroids on top of one another. In order to ensure continuity of the aspect solution across intervals, the intervals are configured to overlap. This allows a weighted average of two or more attitude solutions at each point in time. The current weighting function is simply a linear function going from 1.0 at the interval midpoint to 0.0 at each edge. (This gives discontinuous first derivatives; a future release could include different, improved weighting functions). Figure 1 shows an example of the intervals chosen for input parameters `tstart=0`, `tstop=150`, `photo_interval=50`, and `n_overlap=3`.

Intervals are implementated in the prototype code as C++ objects. This is convenient because for most of the processing, each interval is independent of the others even though they may overlap. The main processing loop of `asp_calc_photo` involves stepping through the ACA centroids file (exactly once), accumulating centroids into appropriate interval objects, and then processing intervals (making photograph and attitude history within the interval). In order to accomodate these overlapping intervals without holding the entire aspect time range in memory, the concept of active intervals is introduced. An interval is activated when the current time of ACA centroids enters the time span of the interval. At that point the interval allocates array memory and begins accumulating centroids. Once the ACA centroid time has gone past the end of the interval, three object functions are called which process the interval. At that point the attitude history for that interval is available to be merged with other overlapping intervals. When all points in the interval have been merged, the interval is deactivated and it deallocates array memory.

Processing:
The processing for `asp_calc_photo` is as follows:

1. Read asp_calc_photo parameters
   (a) If any of `ra_nom, dec_nom` or `roll_nom` are blank, then the nominal reference attitude shall be taken from the aspect solution header
   (b) If `tstart` is blank then the value is taken from the ACACENT file. If `tstop` is blank then the value is taken from the ACACENT file. If both parameter values are exactly 0.0, then both are taken from the ACACENT file.
2. If `kin_sol_method` is "gyro_bias", exit with error message
3. Set nominal spacecraft RA, Dec, and Roll, and corresponding quaternion, based on rules given above in "Inputs:".
4. If `kin_sol_method` is "gyro_raw" or "gyro_corr", then create a temporary aspect solution file:
   (a) Set the following header keywords appropriately: `ra_nom`, `dec_nom`, `roll_nom`, and `dt_asp_sol`. The remaining header keywords can be left uninitialized.
   (b) The following table columns shall be set: `time` and `q_att`. The remaining columns can be left uninitialized.
   (c) The file shall use the time stamps of the GYRODATA file, and cover the time interval `tstart` to `tstop`
   (d) The initial value of the attitude quaternion (`q_att`) shall be the nominal spacecraft attitude
   (e) Subsequent values of `q_att` are derived by integrating the spacecraft rate vector in GYRODATA (`sc_rate_raw` or `sc_rate_corr`). An example of this kinematic integration is found in the aspect pipeline Kalman filter program "swkinint.F".
5. Open ACACENT data product
6. Open ASPSOL data product (`asp_sol_file`, or temporary aspect solution file if `kin_sol_method != asp_sol`).
7. Create PHOTOSOL output data product
8. If `gs_props_file` is not blank, read GSPROPS input data product and compute nominal catalog star vectors in ACA frame. These catalog vectors are used to compute absolute celestial location.
9. Calculate the number of intervals (for no overlap) as the nearest integer of `(tstop - tstart) / photo_interval`. Then evenly divide the time to determine the length of each interval. Finally, calculate the total number of intervals for the specified number of overlapping intervals `n_overlap`.

   ```
   n0 = nint ((tstop - tstart) / photo_interval);
   photo_interval = (tstop - tstart) / n0;
   n_interval = (n0 - 1) * n_overlap + 1;
   ```

10. Set `min_active_intv` and `max_active_intv` to zero, indicating no active intervals. Active intervals are the intervals which contain the current centroid (or equivalently, intervals that cover the current time).

11. Step through the ACACENT data product, reading one row at a time. Do the following steps until the last row has been processed:

    (a) Check if we have already added the last centroid. This happens if

    - GetRow returns 0, indicating the ACACENT EOF has been reached
    - Current centroid time is beyond parameter tstop (`acacent.time > tp.tstop`)

    (b) The centroid is considered bad unless it has correct type, algorithm, status, and time. Skip bad centroids, unless this is the last one, in which case we need to force processing / merging of final interval(s)

    (c) Check if a new active interval is needed for current time. If so, increment `max_active_intv` and call interval memory allocation routine

    (d) Add current centroid to each active interval

    (e) Calculate the following for any unprocessed interval that is filled, using member functions of the interval object. Once these calculations have been done, the interval is processed.

    `dq_gyro` - array of delta attitude quaternions over the interval, giving the shift from the current time to the interval midpoint

    `star_aca` - averaged set of star vectors in ACA frame, with spacecraft motion taken out. Referenced to time at interval midpoint.

    `dq_mid_aca` - delta quaternion of `star_aca` vectors in current interval relative to vectors for first interval. For the first interval, however, this is the delta quat for the `star_aca` vectors relative to the catalog star positions if the ACA were at the nominal RA, Dec, Roll

    (f) Call the MergeIntervals() function to try to determine the aspect solution (ACA attitude quaternion) at the current photosol time. This is possible if all the active intervals containing this time have been processed. If this is the case MergeIntervals calculates the weighted average of the delta quaternions in the relevant active intervals.

    (g) If MergeIntervals() succeeded, do the following:

    i. Convert the delta quaternion to an absolute quaternion (`photosol.q_att_est`)
    ii. Set the remaining fields in the PHOTOSOL data product with reasonable values
    iii. Write the PHOTOSOL record to the file

    (h) Check if any active intervals are expired (`photosol.time > intv.tstop`). If so, remove from the active list and de-allocate array memory

12. Close files and free memory

Prototype asp_calc_photo.h:

```
#include "kalman.h"

/* Tool parameters for asp_calc_photo */

typedef struct
{
  double ra_nom;                /* Nominal RA */
  double dec_nom;               /* Nominal dec */
  double roll_nom;              /* Nominal roll */
  double tstart;                /* Start time (s) */
  double tstop;                 /* Stop time (s) */
  double photo_interval;        /* Duration of interval (s) */
  int    n_overlap;             /* Number of overlapping intervals */
  int    debug_level;           /* Debug level */

  char   gyrocal_file[MAX_CHAR];
  char   gyrodata_file[MAX_CHAR];
  char   aspsol_file[MAX_CHAR];
  char   gsprops_file[MAX_CHAR];
  char   kin_sol_method[MAX_CHAR];
  char   acacent_file[MAX_CHAR];
  char   photosol_file[MAX_CHAR];

  float  roll_tol;              /* Fit tolerance for roll (arcsec) */
  float  pitch_yaw_tol;         /* Fit tolerance for pitch & yaw (arcsec) */
  float  roll_step;             /* Fit step size for roll (arcsec) */
  float  pitch_yaw_step;        /* Fit step size for p & y (arcsec) */

  int    use_ra_dec_roll;       /* Use values from parameter file? */
  int    use_tstart;            /* Use tstart from parameter file? */
  int    use_tstop;             /* Use tstop from parameter file? */

} ACP_Params;

/* Make PHOTOSOL an alias for KALMAN */

typedef Kalman Photosol;

#define PHOTOSOL_Open           KALMAN_Open
#define PHOTOSOL_Create         KALMAN_Create
#define PHOTOSOL_UpdateHeader   KALMAN_UpdateHeader
#define PHOTOSOL_GetRow         KALMAN_GetRow
#define PHOTOSOL_PutRow         KALMAN_PutRow
#define PHOTOSOL_Close          KALMAN_Close
#define PHOTOSOL_AllocMatrices1 KALMAN_AllocMatrices1
#define PHOTOSOL_AllocMatrices2 KALMAN_AllocMatrices2
#define PHOTOSOL_FreeMatrices   KALMAN_FreeMatrices

void t_powell(float p[], float **xi, int n, float atol, int *iter, float *fret,
        float (*func)(float []), float ptol[]);
```

Prototype asp_calc_photo.cc:

```
/***-*-c++-*-************************************************************
 *
 * asp_calc_photo.cc
 *
 ***********************************************************************
 *
 * purpose: Calculate CXO aspect solution using photographic method
 *
 ***********************************************************************
 *
 * by: Tom Aldcroft
 * Copyright 1999 Chandra X-ray Observatory Center, SAO
 *
 ***********************************************************************/

#define PROGRAM "asp_calc_photo"
#define VERSION "1.0"

#include <stdio.h>
#include "aspect_pipeline.h"
#include <string.h>
#include <math.h>
#include <sunmath.h>

#include "acacent.h"
#include "kalman.h"
#include "aspsol.h"
#include "pgsprops.h"

#include "interval.h"
#include "quaternion.h"
#include "asp_calc_photo.h"

extern "C"
{
  void asp_calc_photo_ReadParams (char* filename);
}

void SetAspectAlg (int& aspect_alg);
void SetTstartTstop (Acacent& acacent);
void InitIntervals (Interval*& intv, int& n_interval);
void GetCatalogStars (Quat& q_nom,
                      Matr<double>& catalog_star_aca,
                      int catalog_star_avail[]);
void Calc_photosol (Kalman& photosol, Aspsol& aspsol,
                    Quat& q_nom, Quat& q_att);

ACP_Params tp;                  // Asp_calc_photo tool parameters
AP         ap;                  // Aspect pipeline global vars

/**********************************************************************/
int main (void)
/**********************************************************************/
{
  Acacent  acacent;             // ACA centroids
  Aspsol   aspsol;              // Aspect solution (from OBC)
  Photosol photosol;            // Output photo solution
  Quat     q_nom;               // Nominal (reference) spacecraft attitude

  Matr<double> catalog_star_aca(8,3); // Guide star positions in ACA frame,
                                // derived from catalog and ra,dec,roll_nom
  int catalog_star_avail[8];    // Star avail array for catalog guide stars
```

```
Interval* intv;               // Photo intervals
int    n_interval;            // Number of intervals
int    min_active_intv;       // Lower bound of active intervals
int    max_active_intv;       // Upper bound of active intervals
int    last_centroid;         // Last centroid has been added
int    bad_centroid;          // Bad (useless) centroid flag
int    ii;                    // Interval index

int   aspect_alg;             // Centroid algorithm used for aspect sol'n
int   slot;                   // ACA image slot
int   alg;                    // Centroiding algorithm
Quat  dq_att;                 // Photo method delta attitude (ACA frame) from q_nom
char**  img_type;             // Image type ("UNKNOWN"|"GUIDE"|"MONITOR") for each slot

ANNOUNCE_TOOL_BEGIN;

/* Read aspect pipeline parameters (into struct ap)
   and asp_calc_photo parameters in struct tp (Tool Parameters) */

AP_ReadParams (ASPECT_PIPELINE_PAR, &ap);
asp_calc_photo_ReadParams (PROGRAM);

/* Set aspect algorithm based on aspect pipeline input parameter */

SetAspectAlg (aspect_alg);

/* Open ASPSOL data product */

ASPSOL_AllocMatrices1 (&aspsol);
ASPSOL_Open (&aspsol, "", tp.aspsol_file, FITS_READONLY);
aspsol.rws[RWS_READ].all_bits = ~0x0; /* read all fields */
ASPSOL_AllocMatrices2 (&aspsol);

if (ap.debug_flags & CALC_PHOTO)
  printf ("MAIN: Opened ASPSOL data product\n");

// Set nominal spacecraft RA, Dec, and Roll, and quaternion

if (!tp.use_ra_dec_roll)
  {
    tp.ra_nom = aspsol.ra_nom;
    tp.dec_nom = aspsol.dec_nom;
    tp.roll_nom = aspsol.roll_nom;
  }
q_nom.Set (tp.ra_nom, tp.dec_nom, tp.roll_nom);

/* Open ACACENT data product */

ACACENT_AllocMatrices1 (&acacent);
ACACENT_Open (&acacent, "", tp.acacent_file, FITS_READONLY);
acacent.rws[RWS_READ].all_bits = ~0x0; /* read all fields */
ACACENT_AllocMatrices2 (&acacent);

if (ap.debug_flags & CALC_PHOTO)
  printf ("MAIN: Opened ACACENT data product\n");

/* Set tstart and tstop to acacent values, unless parameter file values
   supplied or both are exactly 0.0.
   Check that specified times are valid.  */

SetTstartTstop (acacent);

/* Read PGSPROPS input data product and compute nominal catalog
```

```
   star vectors in ACA frame.  These catalog vectors are used
   to compute absolute celestial location */

GetCatalogStars (q_nom, catalog_star_aca, catalog_star_avail);

/* Create PHOTOSOL output data product */

PHOTOSOL_AllocMatrices1 (&photosol);
PHOTOSOL_Create (&photosol, "", tp.photosol_file);
photosol.rws[RWS_WRITE].all_bits = ~0x0;
PHOTOSOL_AllocMatrices2 (&photosol);

if (ap.debug_flags & CALC_PHOTO)
  printf ("MAIN: Opened PHOTOSOL data product for writing\n");

/* Calculate number of intervals and then evenly divide the time to
   determine the length of each interval */

InitIntervals (intv, n_interval);

/* Init with no active intervals.  Active intervals are the intervals
   which contain the current centroid */

min_active_intv = 0;
max_active_intv = 0;
last_centroid = 0;

while (!last_centroid)
  {
    /* Check if we have already added the last centroid.  This happens if:
       - GetRow returns 0, indicating the ACACENT EOF has been reached
       - Current centroid time is beyond parameter tstop (acacent.time > tp.tstop) */

    last_centroid = ACACENT_GetRow (&acacent) ? (acacent.time > tp.tstop) : 1;

    /* Bad centroid unless it has correct type, algorithm, status, and time */

    bad_centroid = (acacent.img_type[acacent.slot] != GuideStar  /* CHANGE ME! */
                    || acacent.alg != aspect_alg
                    || acacent.status != IMG_ALG_STATUS_OK
                    || acacent.time < tp.tstart
                    || acacent.time > tp.tstop);

    /* Skip bad centroids, unless this is the last one, in which case we need to
       force processing / merging of final interval(s) */

    if (bad_centroid && !last_centroid)
      continue;

    /* Check if we need a new active interval */

    if (!bad_centroid
        && acacent.time >= intv[max_active_intv].tstart
        && max_active_intv < n_interval)
      {
        if (!intv[max_active_intv++].AllocMemory())
          {
            fprintf (stderr, "ASP_CALC_PHOTO: Error - "
                     "Could not allocate memory for interval\n");
            exit (1);
          }
      }

    /* Add centroids to active intervals */
    if (!bad_centroid)
      {
        for (ii = min_active_intv; ii < max_active_intv; ii++)
          {
            intv[ii].AddCentroid (acacent);
          }
      }

    /* Calculate the following for any unprocessed interval that is filled:
       dq_gyro - array of delta attitude quaternions over the interval, giving
           the shift from the current time to the interval midpoint
       star_aca - averaged set of star vectors in ACA frame, with spacecraft
           motion taken out.  Referenced to time at interval midpoint.
       dq_mid_aca - delta quaternion of star_aca vectors in current interval
           relative to vectors for first interval.  For the first interval,
           this is the delta quat for the star_aca vectors relative to the
           catalog star positions if the ACA were at the nominal RA, Dec, Roll
    */

    for (ii = min_active_intv; ii < max_active_intv; ii++)
      {
        if ((last_centroid || acacent.time > intv[ii].tstop)
            && !intv[ii].processed)
          {
            intv[ii].Calc_dq_gyro (aspsol);
            intv[ii].Calc_star_aca (aspsol);
            intv[ii].Calc_dq_mid_aca (catalog_star_aca, catalog_star_avail);
          }
      }

    /* Try to determine the aspect solution (ACA attitude quaternion) at the
       current photosol time.  This is possible if all the active intervals
       containing this time have been processed.  MergeIntervals returns 1
       if it was able to calculate the weighted average of the delta quats
       in the relevant active intervals */

    while (photosol.time <= tp.tstop
           && MergeIntervals (aspsol, intv, min_active_intv,
                              max_active_intv, photosol.time, dq_att))
      {
        Calc_photosol (photosol, aspsol, q_nom, dq_att);
        PHOTOSOL_PutRow (&photosol);
        //      i_aspsol_time++;
        if (ap.debug_flags & CALC_PHOTO)
          printf ("main:: writing data to photosol at time=%f\n", photosol.time);

        photosol.time += 0.25625;  // aspsol_time[i_aspsol_time];  CHANGE ME!!

      }

    /* Check if intervals are expired (photosol.time > intv.tstop).  If so, remove
       from the active list and de-allocate array memory */

    while ((last_centroid || photosol.time > intv[min_active_intv].tstop)
           && min_active_intv < max_active_intv)
      {
        intv[min_active_intv++].FreeMemory();
      }
  }

ACACENT_FreeMatrices (&acacent);
ACACENT_Close (&acacent);
```

```
  ASPSOL_FreeMatrices (&aspsol);
  ASPSOL_Close (&aspsol);

  PHOTOSOL_FreeMatrices (&photosol);
  PHOTOSOL_Close (&photosol);

  ANNOUNCE_TOOL_END;
}

#define MAX_N_TRY 3           /* number of tries to find a record in time-ordered file */

/**********************************************************************/
int ASPSOL_GetRowAtTime (Aspsol* aspsol, double new_time)
/*
 * Get a row from ASPSOL file at a particular time
 *
 **********************************************************************/
{
  static first = 1;
  int n_try;
  int new_rd_row;

  if (first)
    {
      ASPSOL_GetRow (aspsol);
      first = 0;
    }

  n_try = MAX_N_TRY;
  while (n_try > 0)
    {
      new_rd_row = aspsol->rd_row - 1 + nint ((new_time - aspsol->time)/aspsol->dt_asp_sol);

      if (new_rd_row < 0)
        {
          aspsol->rd_row = 0;
          ASPSOL_GetRow (aspsol);
          return 0;
        }
      if (new_rd_row >= aspsol->n_rows)
        {
          aspsol->rd_row = aspsol->n_rows - 1;
          ASPSOL_GetRow (aspsol);
          return 0;
        }

      aspsol->rd_row = new_rd_row;
      ASPSOL_GetRow (aspsol);
      if (fabs(aspsol->time - new_time) <= aspsol->dt_asp_sol/2.0)
        return 1;

      n_try--;
      fprintf (stderr, "ASPSOL_GetRowAtTime: Warning - "
               "Searching for time=%e but found aspsol->time=%e\n", new_time, aspsol->time);
    }

  fprintf (stderr, "ASPSOL_GetRowAtTime: Error - "
           "Could not find aspsol record at time = %e\n", new_time);
  exit (1);
}


/*****************************************************************/
void SetAspectAlg (int& aspect_alg)
```

```
/****************************************************************/
{
  if (strcmp(ap.aspect_alg, "fm") == 0)
    aspect_alg = FM;
  else if (strcmp(ap.aspect_alg, "psf") == 0)
    aspect_alg = PSF;
  else if (strcmp(ap.aspect_alg, "gauss") == 0)
    aspect_alg = GAUSS;
  else
    {
      fprintf (stderr, "ASP_CALC_PHOTO: Error - "
               "Illegal value for aspect_alg parameter\n");
      exit (1);
    }
}


/****************************************************************/
void SetTstartTstop (Acacent& acacent)

/* Set tstart and tstop to acacent values, unless parameter file values
   supplied or both are exactly 0.0.
   Check that specified times are valid.  */
/****************************************************************/
{
  if (ap.debug_flags & CALC_PHOTO)
    printf ("MAIN: Checking tstart and tstop\n");

  if (!tp.use_tstart || (tp.tstart == 0.0 && tp.tstop == 0.0))
    {
      // Flight code should do following:
      //      tp.tstart = acacent.tstart;
    }
  else
    {
      acacent.rd_row = 0;
      ACACENT_GetRow (&acacent);
      if (tp.tstart < acacent.time)
        {
          fprintf (stderr, "ASP_CALC_PHOTO: Error - "
                   "tstart parameter is less than ACACENT first record time\n");
          exit (1);
        }
    }

  if (!tp.use_tstop || (tp.tstart == 0.0 && tp.tstop == 0.0))
    {
      // Flight code should do following:
      //      tp.tstop = acacent.tstop;
    }
  else
    {
      acacent.rd_row = acacent.n_rows - 1;
      ACACENT_GetRow (&acacent);
      if (tp.tstop > acacent.time)
        {
          fprintf (stderr, "ASP_CALC_PHOTO: Error - "
                   "tstop parameter is greater than ACACENT last record time\n");
          exit (1);
        }
      acacent.rd_row = 0;
    }

  if (ap.debug_flags & CALC_PHOTO)
```

```
    printf ("MAIN: tstart = %e and tstop = %e\n", tp.tstart, tp.tstop);

}

/***************************************************************/
void InitIntervals (Interval*& intv, int& n_interval)

/* Calculate number of intervals and then evenly divide the time to
   determine the length of each interval */
/***************************************************************/
{
  int n0;
  double photo_interval;
  int ii;

  n0 = nint ((tp.tstop - tp.tstart) / tp.photo_interval);
  photo_interval = (tp.tstop - tp.tstart) / n0;
  n_interval = (n0 - 1) * tp.n_overlap + 1;

  if (ap.debug_flags & CALC_PHOTO)
    printf ("MAIN: n0=%d n_interval=%d photo_interval=%f\n", n0, n_interval,
            photo_interval);

  /* Initialize intervals */

  intv = new Interval[n_interval+1];
  for (ii = 0; ii < n_interval; ii++)
    {
      intv[ii].Init (ii, tp.tstart + ii * photo_interval / tp.n_overlap,
                     tp.tstart + (ii + tp.n_overlap) * photo_interval / tp.n_overlap);
    }
}

/***************************************************************/
void GetCatalogStars (Quat& q_nom,
                      Matr<double>& catalog_star_aca,
                      int catalog_star_avail[])
  /*
    Read the GSPROPS file and set the catalog star_avail flags
    and the catalog star vectors in ACA frame.
    These quantities are the analog of the star_aca and
    star_avail arrays in an interval, and they are used for
    the absolute celestial location of the aspect solution

    Pgsprops is a modified version of the flight GSPROPS data
    product.  It has the "string" values taken out, since the
    prototype aspect pipeline I/O layer can't handle it.
  */
/***************************************************************/
{
  Pgsprops  pgsprops;           // Guide star properties
  int slot;
  Matr<double> star_aca(3,1);


  for (slot = 0; slot < 8; slot++)
    catalog_star_avail[slot] = 0;

  PGSPROPS_AllocMatrices1 (&pgsprops);
  PGSPROPS_Open (&pgsprops, "", tp.gsprops_file, FITS_READONLY);
  pgsprops.rws[RWS_READ].all_bits_big[0] = ~0x0; /* read all fields */
  pgsprops.rws[RWS_READ].all_bits_big[1] = ~0x0; /* read all fields */
  PGSPROPS_AllocMatrices2 (&pgsprops);
  while (PGSPROPS_GetRow (&pgsprops))
    {
      // Skip any entries which are not good guide stars

      if (pgsprops.cel_loc_flag != 1
          //  Following should be included in flight code:
          //      || strcmp (pgsprops.type, "GUIDE") != 0
          //      || strcmp (pgsprops.id_status, "GOOD") != 0
          )
        continue;

      // Set star_avail flag and star vector in the ACA frame

      slot = pgsprops.slot;
      catalog_star_avail[slot] = 1;
      star_aca = q_nom.T * pgsprops.pos_eci_eff;
      for (int i = 0; i < 3; i++)
        catalog_star_aca[slot][i] = star_aca[i][0];
    }

  if (ap.debug_flags & CALC_PHOTO)
    printf ("MAIN: Read PGSPROPS data product\n");

  PGSPROPS_Close (&pgsprops);
}

void Calc_photosol (Kalman& photosol, Aspsol& aspsol, Quat& q_nom, Quat& dq_att)
{
  Quat q_att;

  q_att = dq_att * q_nom;
  q_att.SignFix ();

  for (int i = 0; i < 4; i++)
    photosol.q_att_est[i] = q_att.q[i];

/* Set the rest...

 ta: time          : double : Time                         : s       : time        :0: 0:0:0
 ta: omega_est     : float  : Est'd S/C body rate          : deg/s   : omega       :1: 3:0:0
 ta: q_att_est     : double : Est'd S/C attitude quaternion:         : q_att       :1: 4:0:0
 ta: roll          : float  : State vector roll            : deg     : roll        :0: 0:0:0
 ta: pitch         : float  : State vector pitch           : deg     : pitch       :0: 0:0:0
 ta: yaw           : float  : State vector yaw             : deg     : yaw         :0: 0:0:0
 ta: roll_bias     : float  : State vector roll bias rate  : deg/s   : roll_bias   :0: 0:0:0
 ta: pitch_bias    : float  : State vector pitch bias rate : deg/s   : pitch_bias  :0: 0:0:0
 ta: yaw_bias      : float  : State vector yaw bias rate   : deg/s   : yaw_bias    :0: 0:0:0
 ta: covariance    : float  : Smoothed covariance matrix   :         : covariance  :2: 6:6:0
*/


}
```

Prototype interval.h:

```
/***-*-c++-*-*********************************************************/

#ifndef __INTERVAL_h
#define __INTERVAL_h

#include "acacent.h"
#include "aspsol.h"
#include "kalman.h"
#include "quaternion.h"

#define MAX_DQ_GYRO 4000

class Interval {
public:
  int    id;                    // identifier
  double tstart;                // Start time of interval
  double tstop;                 // Stop time of interval
  int    processed;             // Have centroids been processed into delta quaternions?

  double** acacent_time;        // Array of times for ACACENT records
  float**  ang_y;               // Array of ACA y-angles
  float**  ang_z;               // Array of ACA z-angles
  int      n_centroid[8];       // Number of centroids in each slot

  static Matr<double> star_aca0; // Reference star_aca, taken from first interval
  static int star_avail0[8];    // Guide star in slot of first (reference) interval

  Matr<double> star_aca;        // Averaged (over interval) star pos'n vectors for 8 slots
  Quat   dq_mid_aca;            // Delta quat from midpoint of interval to reference interval
  int    star_avail[8];         // Guide star in slot

  Quat* dq_gyro;                // Delta quats relative to attitude at interval midpoint
  int   n_dq_gyro;              // Number of dq_gyro elements
  int   i_dq_gyro;              // Current index into dq_gyro elements
  double* dq_gyro_time;         // Time for dq_gyro records


  void Init (int c_id, double c_tstart, double c_tstop);
  int  AddCentroid (Acacent& acacent);
  int  Calc_star_aca (Aspsol& aspsol);
  int  Calc_dq_gyro (Aspsol& aspsol);
  void Calc_dq_mid_aca (Matr<double>& catalog_star_aca, int catalog_star_avail[]);
  int  Get_dq_aca (Aspsol& aspsol, double time,
                   Quat& dq_aca, double& weight);
  int  AllocMemory();
  int  FreeMemory();
} ;

int MergeIntervals (Aspsol& aspsol,
                    Interval* intv,
                    int& min_active_intv,
                    int& max_active_intv,
                    double time,
                    Quat& q_att);

void Calc_photosol (Kalman& photosol, Aspsol& aspsol, Quat& q_att);

#endif
```

Prototype interval.cc:

```
/***-*-c++-*-*************************************************************
 *
 * interval.cc
 *
 ***********************************************************************
 *
 * purpose: 'Interval' class to support photo method aspect solution.
 *
 ***********************************************************************
 *
 * by: Tom Aldcroft
 * Copyright 1999 AXAF Science Center, SAO
 *
 ***********************************************************************/

#include <stdio.h>
#include "aspect_pipeline.h"
#include <string.h>
#include <math.h>
#include <sunmath.h>

#include "nr_c++.h"
#include "nrutil.h"

#include "aspsol.h"
#include "acacent.h"
#include "interval.h"
#include "matrix_utils.h"
#include "asp_calc_photo.h"

#define DELTA_TIME     0.0001 /* "Slop" allowed in time comparisons */
#define MIN_N_CENTROID 4
#define STEP_SIZE      0.5   /* Initial step size for minimization (arcsec) */

int ASPSOL_GetRowAtTime (Aspsol* aspsol, double new_time);

extern AP ap;                   // Aspect pipeline global vars
extern ACP_Params tp;                   // Asp_calc_photo tool parameters

float chi_func (float* p);

struct {
  int* star_avail0;
  int* star_avail;
  Matr<double> star_aca0;
  Matr<double> star_aca;
} chi_func_data;

/******************************************************************/
void Interval::Init (int c_id, double c_tstart, double c_tstop)
/******************************************************************/
{
  id     = c_id;
  tstart = c_tstart;
  tstop  = c_tstop;
  processed = 0;

  for (int slot = 0; slot < 8; slot++)
    n_centroid[slot] = 0;

  star_aca.init (8, 3);
  if (ap.debug_flags & CALC_PHOTO)
    printf ("Interval::Init: id=%d tstart=%f tstop=%f processed=%d\n",
            id, tstart, tstop, processed);

}

/******************************************************************/
int Interval::Calc_dq_gyro (Aspsol& aspsol)
/******************************************************************/
{
  int getrow_OK;                // Status of ASPSOL_GetRow[AtTime] call
  Quat q_att;                   // Current spacecraft attitude quaternion
  double tstart_gyro = tstart - aspsol.dt_asp_sol; // Pad the start and
  double tstop_gyro = tstop + aspsol.dt_asp_sol; // stop times

  // Get the aspect solution record at the midpoint of the interval, and put the
  // quaternion in q_mid_att

  if (!ASPSOL_GetRowAtTime (&aspsol, (tstart+tstop)/2.0))
    {
      fprintf (stderr, "Interval::calc_dq_gyro: Error - "
               "Could not find aspsol record at time=%f\n", (tstart+tstop)/2.0);
      exit (1);
    }
  Quat q_mid_att (aspsol.q_att);

  // Read the aspect solution file from the record closest to tstart - dt_asp_sol,
  // then back up one record

  i_dq_gyro = 0;

  // Read aspect solution records through time = tstop, and set the dq_gyro[] array.
  // This represents the delta quaternion, relative to the midpoint attitude,
  // based mostly on gyro information

  while ((i_dq_gyro == 0) ?
         (ASPSOL_GetRowAtTime (&aspsol, tstart_gyro) || 1) :
         ASPSOL_GetRow (&aspsol))
    {
      if (aspsol.time > tstop_gyro)
        break;

      /* Set quaternion dq_gyro from the double[4] array aspsol.qatt
         Then set delta quaternion such that
          q_att = dq_gyro * q_mid_att
          dq_gyro = q_att * q_mid_att^-1
          dq_gyro = q_att / q_mid_att             */

      q_att.Set (aspsol.q_att);
      dq_gyro[i_dq_gyro] = q_att / q_mid_att;
      dq_gyro[i_dq_gyro].SignFix(); // Make sure 4th component has positive sign
      dq_gyro_time[i_dq_gyro] = aspsol.time;

      i_dq_gyro++;
    }

  n_dq_gyro = i_dq_gyro;

  // Check to make sure we got asp. solution records for the entire
  // time interval

  if (aspsol.time < tstop - aspsol.dt_asp_sol)
    {
      fprintf (stderr, "Interval::calc_dq_gyro: Error - "
              "Unexpected EOF in ASPSOL at rd_row=%d\n", aspsol.rd_row);
      exit (1);
    }
}

/*****************************************************************/
int Interval::Calc_star_aca (Aspsol& aspsol)
/*****************************************************************/
{
  int slot;

  if (ap.debug_flags & CALC_PHOTO)
    printf ("Interval::calc_star_aca: Called for interval %d\n", id);

  if (ap.debug_flags & CALC_PHOTO)
    {
      printf ("n_centroids = [");
      for (slot = 0; slot < 8; slot++)
        {
          printf ("%d%s", n_centroid[slot], slot==7 ? "]\n" : ", ");
        }
    }

  /* Go through aspect solution file over the time span tstart to tstop,
     and assemble an array of delta quaternions relative to the
     middle attitude quaternion */

  Matr<double> d_cur_aca(3,1);  // Star direction vector at current time in ACA
  Matr<double> d_mid_aca(3,1);  // Star direction vector at midpoint time in ACA
  Matr<double> d_rot_aca(3,1);  // Star direction rotated from current to midpoint time
  Matr<double> d_rot_aca_avg(8,3); // Average rotated star direction
  d_rot_aca_avg.set (0.0);

  for (slot = 0; slot < 8; slot++)
    {
      /* Ignore slots without enough (or any) centroids */

      star_avail[slot] = (n_centroid[slot] >= MIN_N_CENTROID);
      if (!star_avail[slot])
        continue;

      i_dq_gyro = 0;    // reset pointer into dq_gyro array

      for (int i_centroid = 0; i_centroid < n_centroid[slot]; i_centroid++)
        {

          /* Set star centroid direction vector */

          d_cur_aca[0][0] = 1.0;
          d_cur_aca[1][0] = tand (ang_y[slot][i_centroid]);
          d_cur_aca[2][0] = tand (ang_z[slot][i_centroid]);

          // Find nearest value of dq_gyro in order to transform direction
          // vector so it is referenced to midpoint of interval

          while (fabs(dq_gyro_time[i_dq_gyro] - acacent_time[slot][i_centroid]) > aspsol.dt_asp_sol/2.0)
            {
              i_dq_gyro++;
              if (i_dq_gyro >= n_dq_gyro) // Shouldn’t happen if calc_dq_gyro ran OK
                {
                  fprintf (stderr, "Interval::Calc_star_aca: Error - "
                           "i_dq_gyro >= n_dq_gyro (%d)\n", n_dq_gyro);
                  exit (1);
                }
            }

          // Make a temp. copy of dq_gyro and invert it by negating the 4th element

          Quat dq = dq_gyro[i_dq_gyro];
          dq.q[3] = -dq.q[3];
          dq.SetTransformMatr();

          // Rotate the current ACA star vector to take out the rotation of Chandra
          // between current time and the interval midpoint.

          d_rot_aca = dq.T * d_cur_aca;

          // If this is the centroid

           if (i_centroid == n_centroid[slot]/2)
            d_mid_aca = d_rot_aca;

          /* Accumulate average direction vector */

          for (int i = 0; i < 3; i++)
            d_rot_aca_avg[slot][i] += d_rot_aca[i][0];

          if (ap.debug_flags & CALC_PHOTO)
            {
              printf ("Interval::calc_star_aca: time=%e time=%e rot=(%8.5f, %8.5f)"
                      " mid=(%8.5f, %8.5f)\n",
                      dq_gyro_time[i_dq_gyro], acacent_time[slot][i_centroid], d_rot_aca[1][0]*206000.,
                      d_rot_aca[2][0] * 206000.,d_mid_aca[1][0] * 206000.,
                      d_mid_aca[2][0] * 206000.);
            }
        }

      /* Finish averaging and then normalize to a unit vector */

      double sumsq = 0;
      for (int i = 0; i < 3; i++)
        {
          d_rot_aca_avg[slot][i] /= n_centroid[slot];
          sumsq += d_rot_aca_avg[slot][i] * d_rot_aca_avg[slot][i];
        }

      for (int i = 0; i < 3; i++)
        star_aca[slot][i] = d_rot_aca_avg[slot][i] / sqrt(sumsq);

      if (ap.debug_flags & CALC_PHOTO)
        {
          printf ("Interval::calc_star_aca: id=%d star_aca[%d] = (%8.5f, %8.6f)\n", id,
                  slot, star_aca[slot][1]*206000, star_aca[slot][2]*206000);
        }

    }

  i_dq_gyro = 0;                // reset pointer into dq_gyro array
  processed = 1;                // mark interval as processed

  return 1;
}

/*****************************************************************/
void Interval::Calc_dq_mid_aca (Matr<double>& catalog_star_aca,
                                int catalog_star_avail[])
/*****************************************************************/
{
```

```
int i, j;
int slot;
void powell (float p[], float **xi, int n, float ftol, int *iter,
             float *fret, float (*func)(float []), float ptol[]);

/* Set the global data needed for chi_func minimization */

chi_func_data.star_aca = star_aca;
chi_func_data.star_avail = star_avail;

/* If this is the first interval, set the reference values for the guide
   star vectors in the ACA frame (star_aca0) and  star_avail.
   The "guide_catalog" values are set by rotating the AGASC coordinates
   into the ACA frame.  */

if (id == 0)
  {
    if (strlen(tp.gsprops_file) > 0)
      {
        chi_func_data.star_aca0 = catalog_star_aca;
        chi_func_data.star_avail0 = catalog_star_avail;
      }
    else
      {
        chi_func_data.star_aca0 = star_aca;
        chi_func_data.star_avail0 = star_avail;
        dq_mid_aca.Set (0.0, 0.0, 0.0, 1.0);
        return;
      }
  }

/* Set up for call to Num. Recipes Powell minimization routine */

float *p;
float **xi;
int     n;
float   atol;
int     iter;
float   fret;
float *ptol;

/* Check that there are at least two stars in common */

int n_star_avail = 0;
for (slot = 0; slot < 8; slot++)
  {
    if (chi_func_data.star_avail[slot]
        && chi_func_data.star_avail0[slot])
      n_star_avail++;
  }
if (n_star_avail < 2)
  {
    fprintf (stderr, "Interval::Calc_dq_mid_aca: Error - "
             "Only %d star(s) in common between interval 0 and %d\n",
             n_star_avail, id);
    exit (1);
  }

n = 3;

p = vector(1,n);
p[1] = 0.0;
p[2] = 0.0;
p[3] = 0.0;
```

```
  xi = matrix(1,n,1,n);
  for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
      xi[i][j] = 0.0;
    }
  }
  xi[1][1] = 20.0;              // roll_step
  xi[2][2] = 1.0;               // pitch_yaw_step
  xi[3][3] = 1.0;               // pitch_yaw_step

  atol = 0.02;

  // Fitting tolerances.  Powell stops when step size < tolerance

  ptol = vector (0, n);
  ptol[0] = n*n;
  ptol[1] = tp.roll_tol;
  ptol[2] = tp.pitch_yaw_tol;
  ptol[3] = tp.pitch_yaw_tol;

  t_powell (p, xi, n, atol, &iter, &fret, &chi_func, ptol);

  if (ap.debug_flags & CALC_PHOTO)
    printf ("id = %d  final p = %f %f %f\n", id, p[1], p[2], p[3]);

  for (int i = 1; i <= 3; i++)
    p[i] *= RAD_PER_ARCSEC;

  dq_mid_aca.Set (p[1]/2.0, p[2]/2.0, p[3]/2.0,
               sqrt(1.0 - p[1]*p[1]/4.0 - p[2]*p[2]/4.0 - p[3]*p[3]/4.0));

  if (id == 0)
    {
      chi_func_data.star_aca0 = star_aca;
      chi_func_data.star_avail0 = star_avail;
    }


}


/******************************************************************/
int Interval::Get_dq_aca (Aspsol& aspsol, double time,
                          Quat& dq_aca, double& weight)
/******************************************************************/
{
  if (time < tstart || time > tstop)
    return 0;

  if (ap.debug_flags & CALC_PHOTO)
    printf ("Interval::Get_dq_aca: Called for interval %d\n", id);


  weight = 1.001 - fabs((tstop+tstart)/2.0 - time) / ((tstop-tstart)/2.0);

  while (dq_gyro_time[i_dq_gyro] < time - DELTA_TIME)
    {
      i_dq_gyro++;
      if (i_dq_gyro >= n_dq_gyro) // Shouldn't happen if calc_dq_gyro ran OK
        {
          fprintf (stderr, "Interval::Get_dq_aca: Error - "
                   "i_dq_gyro >= n_dq_gyro (%d)\n", n_dq_gyro);
          exit (1);
        }
    }

  dq_aca = dq_gyro[i_dq_gyro] * dq_mid_aca;
  dq_aca.SignFix();

  if (ap.debug_flags & CALC_PHOTO && tp.debug_level >= 4)
    {
      printf ("Interval::Get_dq_aca: Interval %d\n", id);
      printf ("                      dq_aca       ");
      dq_aca.Print ();
      printf ("                      dq_gyro[%d] ", i_dq_gyro);
      dq_gyro[i_dq_gyro].Print ();
      printf ("                      dq_mid_aca   ");
      dq_mid_aca.Print ();
    }
  return 1;
}

/*******************************************************************/
float chi_func (float* p)
/*******************************************************************/
{
  float p0 = p[1]/3600.;
  float p1 = p[2]/3600.;
  float p2 = p[3]/3600.;
  double cx = cosd(p0);
  double sx = sind(p0);
  double cy = cosd(p1);
  double sy = sind(p1);
  double cz = cosd(p2);
  double sz = sind(p2);
  double R10, R11, R12, R20, R21, R22;
  double v0, v1, v2;
  double vr1, vr2;
  double d1, d2;
  float  chi2 = 0.0;
  int    slot;
  const float sigma = 0.1 / 206000; // 0.1 arcsec

  R10 = -cx*sz;
  R11 = (cx*cz - sx*sy*sz);
  R12 = (cz*sx + cx*sy*sz);
  R20 = sy;
  R21 = -cy*sx;
  R22 = cx*cy;

  for (slot = 0; slot < 8; slot++) {
    if (chi_func_data.star_avail[slot] &&
        chi_func_data.star_avail0[slot])
      {
        v0 = chi_func_data.star_aca0[slot][0];
        v1 = chi_func_data.star_aca0[slot][1];
        v2 = chi_func_data.star_aca0[slot][2];
        vr1 = R10 * v0 + R11 * v1 + R12 * v2;
        vr2 = R20 * v0 + R21 * v1 + R22 * v2;
        d1 = (vr1 - chi_func_data.star_aca[slot][1]) / sigma;
        d2 = (vr2 - chi_func_data.star_aca[slot][2]) / sigma;
        chi2 += d1*d1 + d2*d2;
      }
  }

  if (ap.debug_flags & CALC_PHOTO && tp.debug_level >= 5)
    {
      printf ("chi_func: p = %f %f %f  chi2 = %f\n", p[1], p[2], p[3], chi2);
    }

  return (chi2);

  /*
     Rv[0][i] = v[0][i]*cx*cz + v[1][i]*(cz*sx*sy + cx*sz) +
       v[2][i]*(-(cx*cz*sy) + sx*sz);
     Rv[1][i] = -(v[0][i]*cx*sz) + v[2][i]*(cz*sx + cx*sy*sz) +
       v[1][i]*(cx*cz - sx*sy*sz);
     Rv[2][i] = v[2][i]*cx*cy - v[1][i]*cy*sx + v[0][i]*sy;
  */

}

/******************************************************************/
int Interval::AddCentroid (Acacent& acacent)
/******************************************************************/
{
  if (acacent.time < tstart || acacent.time > tstop)
    return 0;

  if (ap.debug_flags & CALC_PHOTO && tp.debug_level > 5)
    {
      printf ("Interval::AddCentroid: Adding centroid interval=%d "
              " slot=%d alg=%d time=%f ang_y=%f ang_z=%f\n", id, acacent.slot,
              acacent.alg, acacent.time, acacent.ang_y, acacent.ang_z);
    }

  int slot = acacent.slot;

  acacent_time[slot][n_centroid[slot]] = acacent.time;
  ang_y[slot][n_centroid[slot]] = acacent.ang_y;
  ang_z[slot][n_centroid[slot]] = acacent.ang_z;
  n_centroid[slot]++;

  return 1;
}

/******************************************************************/
int Interval::AllocMemory ()
/******************************************************************/
{
  if (ap.debug_flags & CALC_PHOTO)
    printf ("Interval::AllocMemory: Allocating memory for interval %d\n", id);

  acacent_time = (double **) AllocMatrix2 (DOUBLE, 8, nint (tstop - tstart));
  ang_y = (float **) AllocMatrix2 (FLOAT, 8, nint (tstop - tstart));
  ang_z = (float **) AllocMatrix2 (FLOAT, 8, nint (tstop - tstart));

  dq_gyro = new Quat[MAX_DQ_GYRO];
  dq_gyro[20].T.Print ("%8.4f");
  dq_gyro_time = new double[MAX_DQ_GYRO];

  return 1;
}

/******************************************************************/
int Interval::FreeMemory ()
/******************************************************************/
{
  if (ap.debug_flags & CALC_PHOTO)
    printf ("Interval::FreeMemory: Freeing memory for interval %d\n", id);
  FreeMatrix2 ((void **) acacent_time);
  FreeMatrix2 ((void **) ang_y);
  FreeMatrix2 ((void **) ang_z);

  delete dq_gyro;
  delete dq_gyro_time;

  return 1;
}


#define MAX_MERGE 5

/*********************************************************************/
int MergeIntervals (Aspsol& aspsol,
                    Interval* intv,
                    int& min_active_intv,
                    int& max_active_intv,
                    double time,
                    Quat& dq_att)
/*********************************************************************/
{
  int    ii;
  int    ii_avg;
  int    i_q;
  int    done = 0;
  int    unprocessed_intervals = 0;
  int    n_avg = 0;
  Quat   dq_aca[MAX_MERGE];
  double weight[MAX_MERGE];
  double dq[4];
  double w_sum;

      /* Step through active intervals and determine if there are any
         unprocessed intervals which contain data at the current
         time (time).  If so, return. */

  for (ii = min_active_intv; ii < max_active_intv; ii++)
    {
      if (intv[ii].tstart <= time
          && !intv[ii].processed)
        {
          unprocessed_intervals = 1;
        }
      else
        {
          /* Get dq_aca and weight from interval for current time and store
             in dq_aca[n_intv].  If no corresponding dq_aca exists
             in interval, function returns 0 */

          if (intv[ii].Get_dq_aca (aspsol, time, dq_aca[n_avg], weight[n_avg]))
            {
              if (ii > 0)
                dq_aca[n_avg] = dq_aca[n_avg] * intv[0].dq_mid_aca;
              n_avg++;
              ii_avg = ii;
            }
        }
    }

  if (unprocessed_intervals)
    return 0;

  /* No unprocessed intervals at current time, so use values of
```

```
     dq_aca[] (array of quaternions) and weight[] to calculate the
     average dq */

  /* First check that there were available data points */

  if (n_avg == 0)
    {
      fprintf (stderr, "MergeIntervals:: Error - "
               "n_avg == 0 with no unprocessed intervals\n");
      exit (1);
    }

  /* calculate weighted average of first 3 quaternion components */

  if (ap.debug_flags & CALC_PHOTO)
    printf ("Interval::MergeIntervals: Calculating average n_avg=%d\n", n_avg);

  dq[0] = dq[1] = dq[2] = 0.0;
  w_sum = 0.0;

  for (int i_avg = 0; i_avg < n_avg; i_avg++)
    {
      for (i_q = 0; i_q < 3; i_q++)
        {
          dq[i_q] += weight[i_avg] * dq_aca[i_avg].q[i_q];
        }
      w_sum += weight[i_avg];
    }

  for (i_q = 0; i_q < 3; i_q++)
    {
      dq[i_q] /= w_sum;
    }

  /* Calculate 4th quat. component and assign dq_att (delta attitude quat) */

  dq[3] = sqrt(1.0 - dq[0]*dq[0] - dq[1]*dq[1] - dq[2]*dq[2]);
  dq_att = Quat (dq);

  return 1;
}
```

Prototype read_param.c:

```
#include <stdio.h>
#include "aspect_pipeline.h"
#include "parameter.h"

#include "asp_calc_photo.h"

extern ACP_Params tp;

/*********************************************************************/
int pgetd_string (paramfile pfile, char* param_name, double* param_val)
/*********************************************************************/
{
  char param_str[MAX_CHAR];
  pgetstr (pfile, param_name, param_str, MAX_CHAR);
  return (sscanf (param_str, "%lf", param_val) == 1);
}

/*********************************************************************/
void asp_calc_photo_ReadParams (char* filename)
/*********************************************************************/
{
  paramfile pfile;

  if ((pfile = paramopen (filename, NULL, 0, "r")) == NULL)
    {
      fprintf(stderr, "AP_ReadParams: error - couldn't open %s\n",
              filename);
      exit(1);
    }

  tp.use_ra_dec_roll  = pgetd_string (pfile, "ra_nom", &tp.ra_nom);
  tp.use_ra_dec_roll &= pgetd_string (pfile, "dec_nom", &tp.dec_nom   );
  tp.use_ra_dec_roll &= pgetd_string (pfile, "roll_nom", &tp.roll_nom  );

  tp.use_tstart        = pgetd_string (pfile, "tstart", &tp.tstart    );
  tp.use_tstop         = pgetd_string (pfile, "tstop", &tp.tstop     );

  tp.photo_interval    = pgetd (pfile, "photo_interval");
  tp.n_overlap         = pgeti (pfile, "n_overlap");
  tp.debug_level       = pgeti (pfile, "debug_level");

  tp.roll_tol          = pgetf (pfile, "roll_tol");
  tp.pitch_yaw_tol     = pgetf (pfile, "pitch_yaw_tol");
  tp.roll_step         = pgetf (pfile, "roll_step");
  tp.pitch_yaw_step    = pgetf (pfile, "pitch_yaw_step");

  pgetstr (pfile, "gyro_cal_file",  tp.gyrocal_file, MAX_CHAR);
  pgetstr (pfile, "gyro_data_file", tp.gyrodata_file, MAX_CHAR);
  pgetstr (pfile, "asp_sol_file",   tp.aspsol_file, MAX_CHAR);
  pgetstr (pfile, "gs_props_file",  tp.gsprops_file, MAX_CHAR);
  pgetstr (pfile, "kin_sol_method", tp.kin_sol_method, MAX_CHAR);
  pgetstr (pfile, "aca_cent_file",  tp.acacent_file, MAX_CHAR);
  pgetstr (pfile, "photo_sol_file", tp.photosol_file, MAX_CHAR);

  paramclose(pfile);
}
```

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: asp_calc_photo

Spec Category: Aspect

Code Type: ASCDS

Code Source: Prototype

Tool Number 1249

asp_forward_kalman

Description:

Forward Kalman filter to combine aspect camera and gyro data.

Parameters:

ACASIGHT file	file	L1 ACA Image Centroids
GYRODATA file	file	L1 Gyro Data
ACACAL file	file	ACA Calibration Data
GYROCAL file	file	Gyro Calibration Data
GSPROPS file	file	Guide Star Properties
KALMAN file	file	L1 Kalman Data Product

Inputs:

ACASIGHT - ACA Image Centroids
GYRODATA - Gyro Data
ACACAL - ACA Calibration Data
GYROCAL - Gyro Calibration Data
GSPROPS - Guide Star Properties

Outputs:

KALMAN - Kalman Data Product

Processing:

Kalman filtering is applied to extract spacecraft attitude information using aspect camera star centroids and gyro inertial attitude measurements. It also estimates the gyro drift errors.

Read the ACA Sightings to obtain the image centroids and initial spacecraft attitude quaternion. Read the Gyro Data to obtain the filtered gyro angular rates and flags. Apply a forward Kalman filter to the centroid measurements and the gyro rate data up to the desired time for an estimate. Use a standard six state extended Kalman filter design, with three attitude error states and three gyro drift rate error states, as described in detail in TRW-SE11k, 11 Jan 1996.

The inputs are:

- filtered gyro angular rates (every 256 ms, from Gyro Data)
- number of stars being tracked, from Guide Star properties
- the ECI coordinates of identified stars from the catalog, from Guide Star Properties
- measured star centroids in ACA angular coordinates, from ACA Sightings
- estimated NEA for each star

- initial quaternion.

The outputs of the filter include the following estimates:

- attitude quaternion estimate
- state transition matrix
- post measurement covariance matrix
- pre measurement covariance matrix
- pre measurement state vector
- post measurement state vector

These outputs are stored in the Kalman Data table.

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: asp_forward_kalman

Spec Category: Aspect

Code Type: ASCDS

Code Source: GADS Prototype

Tool Number 1401

Name: `asp_get_calib`

Description: Make the ACACAL and GYROCAL data products

Parameters:

aca_cal	file	L1 ACACAL data product file
gyro_cal	file	L1 GYROCAL data product file
gs_props	file	L1 GSPROPS data product file
pcad_eng32	file	L0 PCAD engineering file stack (sample rate = 32)
ccd_bad_pix	file	CCD bad pixels file
ccd_char	file	CCD characteristics file
psf_star_c1	file	PSF library file for c1 stars
psf_star_c2	file	PSF library file for c2 stars
psf_star_c3	file	PSF library file for c3 stars
psf_fid	file	PSF library file for fids
field_dist	file	ACA field distortion file
dark_curr	file	Dark current image file
flat_field	file	Flat field image file
cti_cal	file	ACA CTI calibration file
aca_fts_align	file	ACA and FTS alignments file
map_n_pix	int	Size of dark/flat field arrays
tstart	double	Start of time filter (sec)
tstop	double	End of time filter (sec)

Inputs:

pcad_eng32	L0 PCAD engineering data
gs_props	L1 GSPROPS data product file
ccd_bad_pix	CCD bad pixels file
ccd_char	CCD characteristics file
psf_star_c1	PSF library file for c1 stars
psf_star_c2	PSF library file for c2 stars
psf_star_c3	PSF library file for c3 stars
psf_fid	PSF library file for fids
field_dist	ACA field distortion file
dark_curr	Dark current image file
flat_field	Flat field image file
cti_cal	ACA CTI calibration file
aca_fts_align	ACA and FTS alignments file

Outputs:

ACACAL	L1 ACA calibration data product
GYROCAL	L1 gyro calibration data product

ACACAL Data Product Definiton

Name	Type	Comment	Units	Dims
Principle Header Keywords				
nbadreg	int	Number of bad regions		
nsubpix	int	Size of sub-pixel rsp. matrix		
mapnpix	int	Size of dark/flat images		
npsf	int	Number of PSFs in library		
psfnpix	int	Size of PSF matrix		
fmcrnpix	int	Size of FM correction matrix		
psfscale	float	No. PSF pixels per CCD pixel		
imgtype	long	Image type (fid, guide, etc)		MAX_IMG
Binary Table Extension				
img_loc_i	float	Row location for PSF interp	pixel	MAX_IMG
img_loc_j	float	Col location for PSf interp	pixel	MAX_IMG
bad_region	int	Bad regions (i0 i1 j0 j1)		n_bad_region × 4
flat_row0	short	Flat field image min. row	pixel	MAX_IMG
flat_col0	short	Flat field image min. col.	pixel	MAX_IMG
dark_row0	short	Dark current image min. row	pixel	MAX_IMG
dark_col0	short	Dark current image min. col.	pixel	MAX_IMG
flat_field	float	Flat field image		MAX_IMG × map_n_pix × map_n_pix
dark_current	float	Dark current image (e-/s)	count/s	MAX_IMG × map_n_pix × map_n_pix
fd_y_star	float	Yag(Row,Col,T) polyn. (star)		FIELD_ORDER
fd_z_star	float	Zag(Row,Col,T) polyn. (star)		FIELD_ORDER
fd_y_fid	float	Yag(Row,Col,T) polyn. (fid)		FIELD_ORDER
fd_z_fid	float	Zag(Row,Col,T) polyn. (fid)		FIELD_ORDER
fd_r_star	float	Row(Yag,Zag,T) polyn. (star)		FIELD_ORDER
fd_c_star	float	Col(Yag,Zag,T) polyn. (star)		FIELD_ORDER
fd_r_fid	float	Row(Yag,Zag,T) polyn. (fid)		FIELD_ORDER
fd_c_fid	float	Col(Yag,Zag,T) polyn. (fid)		FIELD_ORDER
star_comp_coeffs	float	Star compensation coeff's		2 × COMP_ORDER
fm_corr_i	float	First moment correction (row)	pixel	MAX_IMG × fm_corr_n_pix × fm_corr_n_pix
fm_corr_j	float	First moment correction (col)	pixel	MAX_IMG × fm_corr_n_pix × fm_corr_n_pix
cti_corr_i	float	CTI correction for FM (row)	pixel	MAX_IMG
cti_corr_j	float	CTI correction for FM (col)	pixel	MAX_IMG
color_corr_i	float	Color correction (row)	deg	MAX_IMG
color_corr_j	float	Color correction (col)	deg	MAX_IMG
read_noise	float	Read noise for image (e-)	count	MAX_IMG
gain	float	CCD gain for quadrant (e-/DN)	count/DN	4
sub_pix_resp	float	Sub-pixel response		n_sub_pix × n_sub_pix
aca_align	double	ACA nominal alignment matrix		3 × 3
aca_misalign	double	ACA misalignment matrix		3 × 3
fts_misalign	double	FTS misalignment matrix		3 × 3
psf_lib	float	PSF interpolated from library		MAX_IMG × psf_n_pix × psf_n_pix
psf_corr	float	PSF corrected		MAX_IMG × psf_n_pix × psf_n_pix

GYROCAL Data Product Definiton

Name	Type	Comment	Units	Dims
Principle Header Keywords				
n_gyro	int	Number active gyro channels		
gyrosel	int	Gyro selection flag		
Binary Table Extension				
gyro_id	int	Gyro ID's (ref'd to CALDB)		N_GYRO
align	double	Gyro to S/C alignment vectors		N_GYRO × 3
sf_ma	double	Scale factor misalignment		3 × 3
gyro2sc	double	Gyro rates vector to S/C rate		3 × N_GYRO
scale_fac_pos	float	Scale factor coeff (pos)	arcsec/cts	N_GYRO
scale_fac_neg	float	Scale factor coeff (neg)	arcsec/cts	N_GYRO
bias_int	double	Bias (initial estimate)	cts/s	3
rrw_noise	double	Rate-random-walk noise		N_GYRO
arw_noise	double	Angle-random-walk noise		N_GYRO

Notes:

Processing for the GYROCAL data product is not yet included in this tool spec.

Processing:

The following processing steps assume that each of the calibration data files listed as *Inputs* have been extracted, with the correct aca_id and focal_plane. The dark current and flat field image files also require the extraction parameter image_type == DARK_CURR and image_type == RESP, respectively.

Note that the actual names of data I/O structures and internal variables are arbitrary, and are specified here only for notational convenience.

1. Initialize ACACAL data product

   (a) Create the ACACAL data product with file name set to the parameter aca_cal, and with data I/O structure acacal

   (b) Open the CCD bad pixel list FITS file (ccd_bad_pix), with a data I/O structure ccdbadpix

   (c) Set acacal.n_bad_region to the number of rows in the CCD bad pixel file (ccdbadpix.NAXIS2)

   (d) Set acacal.map_n_pix = map_n_pix (from parameter file)

   (e) Open the CCD characteristics FITS file (ccd_char), with a data I/O structure ccdchar and read the first (and only) row

   (f) Set acacal.n_sub_pix = ccdchar.n_sub_pix

   (g) Open (in turn) each of the four PSF library image files (psf_lib_star_c1, psf_lib_star_c2, psf_lib_star_c3, and psf_lib_fid) with a data I/O structure psflib. Do the following:

      i. For the primary array of the first PSF library image file, set acacal.n_psf = psflib.NAXIS1 and acacal.psf_n_pix = psflib.NAXIS3

      ii. For each file, verify that psflib.NAXIS1 == psflib.NAXIS2 == acacal.n_psf and psflib.NAXIS3 == psflib.NAXIS4 == acacal.psf_n_pix

      iii. For the first extension array of the first PSF library image file, set acacal.fm_corr_n_pix = psflib.NAXIS3

      iv. For the first extension array of each file, verify that psflib.NAXIS3 == psflib.NAXIS4 == acacal.fm_corr_n_pix

   (h) Set acacal.psf_pixscale = 10.0 (TBR, since I don't understand the values in the PSF library file headers)

   (i) Set the image type vector header keywords imgtype1 ... imgtype8. The values are derived from the OCAT, but the exact interface to this tool is TBD. Allowed keyword values are NONE, FID, GUIDE, or MONITOR.

   (j) Create dynamically allocated arrays in acacal structure

2. Set bad regions

   (a) For each row of the bad pixel file do the following:

```
acacal.bad_region[i][0] = ccdbadpix.row0;
acacal.bad_region[i][1] = ccdbadpix.row1;
acacal.bad_region[i][2] = ccdbadpix.col0;
acacal.bad_region[i][3] = ccdbadpix.col1;
i++;
```

   (b) Close the CCD bad pixel list file

3. Set field distortion coefficients

   (a) Open the field distortion FITS file (parameter `field_dist`) with a data I/O structure `fielddist`

   (b) Do the following:

   ```
   for (i = 0; i < FIELD_ORDER; i++) {
     acacal.fd_y_star[i] = fielddist.fd_y_star[i];
     acacal.fd_z_star[i] = fielddist.fd_z_star[i];
     acacal.fd_y_fid[i]  = fielddist.fd_y_fid[i];
     acacal.fd_z_fid[i]  = fielddist.fd_z_fid[i];
     acacal.fd_r_star[i] = fielddist.fd_r_star[i];
     acacal.fd_c_star[i] = fielddist.fd_c_star[i];
     acacal.fd_r_fid[i]  = fielddist.fd_r_fid[i];
     acacal.fd_c_fid[i]  = fielddist.fd_c_fid[i];
   }
   ```

   (c) Close the field distortion file

4. Set the row and column locations for PSF interpolation

   (a) Cycle through the stack of PCAD32 engineering files (parameter `pcad_eng32`) with a data I/O structure `pcad32`

   (b) Over the time range `tstart` to `tstop`, read the on board image centroids for each of the eight image slots. Determine the average Y and Z angles for each slot, and the minimum and maximum. (Check image function). Determine the average image magnitude for each slot (`mag_avg[slt]`).

   (c) Convert the angles (average, min, max) to CCD row and column pixel coordinates. For `acacal.imgtype` equal to `STAR`, `MONITOR`, or `NONE`, use the `acacal.fd_r_star` and `acacal.fd_c_star` polynomial coefficients. For `acacal.imgtype` equal to `FID`, use the `acacal.fd_r_fid`, and `acacal.fd_c_fid` coefficients.

$$
\begin{aligned}
R &= A_{0r} + A_{1r}Z + A_{2r}Y + A_{3r}T + A_{4r}Z^2 + A_{5r}ZY + A_{6r}ZT + A_{7r}Y^2 \\
&\quad + A_{8r}YT + A_{9r}T^2 + A_{10r}Z^3 + A_{11r}YZ^2 + A_{12r}TZ^2 + A_{13r}ZY^2 \\
&\quad + A_{14r}ZYT + A_{19r}T^3 + A_{16r}Y^3 + A_{17r}TY^2 + A_{18r}YT^2 + A_{19r}T^3
\end{aligned}
$$

$$
\begin{aligned}
C &= A_{0c} + A_{1c}Z + A_{2c}Y + A_{3c}T + A_{4c}Z^2 + A_{5c}ZY + A_{6c}ZT + A_{7c}Y^2 \\
&\quad + A_{8c}YT + A_{9c}T^2 + A_{10c}Z^3 + A_{11c}YZ^2 + A_{12c}TZ^2 + A_{13c}ZY^2 \\
&\quad + A_{14c}ZYT + A_{19c}T^3 + A_{16c}Y^3 + A_{17c}TY^2 + A_{18c}YT^2 + A_{19c}T^3
\end{aligned}
$$

   (d) For each of eight image slots, set `acacal.img_loc_i` to the average CCD row, and set `acacal.img_loc_j` to the average CCD column. If

5. Set the dark current and flat field (responsivity) columns

   (a) Read the 1024×1024 dark current image FITS file (parameter `dark_curr`) into an array `dark_image`

   (b) Read the 1024×1024 responsivity image FITS file (parameter `flat_field`) into an array `resp_image`

(c) For each image slot `slt = (0..7)`, do the following

```
acacal.flat_row0[slt] = acacal.img_loc_i[slt] - map_n_pix / 2;
acacal.flat_col0[slt] = acacal.img_loc_j[slt] - map_n_pix / 2;
acacal.dark_row0[slt] = acacal.img_loc_i[slt] - map_n_pix / 2;
acacal.dark_col0[slt] = acacal.img_loc_j[slt] - map_n_pix / 2;

for (ii = 0; ii < map_n_pix; ii++) {
  i_flat = acacal.flat_row0[slt] + ii;
  i_dark = acacal.dark_row0[slt] + ii;
  for (jj = 0; jj < map_n_pix; jj++) {
    j_flat = acacal.flat_col0[slt] + jj;
    j_dark = acacal.dark_col0[slt] + jj;
    acacal.flat_field[slt][ii][jj]   = resp_image[i_flat][j_flat];
    acacal.dark_current[slt][ii][jj] = dark_image[i_dark][j_dark];
  }
}
```

The values of `i_flat, j_flat, i_dark, j_dark` need to be limit checked (0 .. 1023). If a value is out of bounds, set `acacal.flat_field = 1.0` or `acacal.dark_current = 0.0`

6. Set all values of the matrix `acacal.star_comp_coeffs[][]` to 0.0

7. Set all values of the matrices `acacal.fm_corr_i[][][]` and `acacal.fm_corr_j[][][]` to 0.0. These values are updated in the `aca_make_psf` tool.

8. Calculate the CTI corrections

   (a) Open the CTI calibration coefficients FITS file (parameter `cti_cal` with a data I/O structure `cti`

   (b) For each image slot (`slt`), do the following:

      i. Determine `quadrant[slt]` based on the average row and column (rounded to the nearest integer):

Row	Column	Quadrant	Quad name
< 0	>= 0	0	A
< 0	< 0	1	B
>= 0	>= 0	2	C
>= 0	< 0	3	D

      ii. Determine the row and column shift length (`row_shift, col_shift`), which is the distance (pixels) to the nearest CCD edge (i.e. in the readout direction). These values are positive.

      iii. For each slot, find the entry in the CTI calibration file which has `cti.shift_dir == 0` and `cti.quadrant == quadrant[slt]`. Set

```
acacal.cti_corr_i = cti.delta_max * row_shift/512.0
                        * (mag_avg[slt] - cti.mag_0) / cti.mag_d
```

      iv. For each slot, find the entry in the CTI calibration file which has `cti.shift_dir == 1` and `cti.quadrant == quadrant[slt]`. Set

```
acacal.cti_corr_i = cti.delta_max * col_shift/512.0
                        * (mag_avg[slt] - cti.mag_0) / cti.mag_d
```

9. Set the color correction terms

   (a) For each image slot (`slt`) set `acacal.color_corr_i[slt] = 0.0` and `acacal.color_corr_j[slt] = 0.0`

(b) Future versions of `asp_get_calib` may use the color shift coefficients `ccdchar.color_shift`, along with image CCD positions, types, and colors (from GSPROPS) in order to calculate these corrections.

10. Set the CCD read noise and gain

    (a) For each image slot `slt`, set

    ```
    acacal.read_noise[slt] = ccdchar.read_noise[quadrant[slt]];
    ```

    (b) For each quadrant (`q = (0..3)`), set

    ```
    acacal.gain[q] = ccdchar.gain[q];
    ```

11. Set the CCD sub-pixel response

    ```
    for (i = 0; i < acacal.n_sub_pix; i++) {
      for (j = 0; j < acacal.n_sub_pix; j++) {
        acacal.sub_pix_resp[i][j] = ccdchar.sub_pix_resp[i][j];
      }
    }
    ```

12. Set the ACA and FTS alignment matrices

    (a) Open the ACA and FTS alignments FITS file (parameter `aca_fts_align`) with data I/O structure `align`, and read the first (and only) row

    (b) Copy the three aligment matrices as follows

    ```
    for (i = 0; i < 3; i++) {
      for (j = 0; j < 3; j++) {
        acacal.aca_align[i][j] = align.aca_sc_align[i][j];
        acacal.aca_misalign[i][j] = align.aca_misalign[i][j];
        acacal.fts_misalign[i][j] = align.fts_misalign[i][j];
      }
    }
    ```

13. Set each element of the 3-dimensional PSF arrays (`acacal.psf_lib` and `acacal.psf_corr`) to 0.0

Source: New
Release: 4
SDS ID: j401
SDS Name: aspect_cal_get

Tool Number 1402

Description: Make the aspect intervals data product.

Parameters:

aiprops	file		Output L1 AIPROPS data product file
ccdm_eng16	file		Input L0 CCDM engineering file stack (sample rate = 16)
pcad_eng32	file		Input L0 PCAD engineering file stack (sample rate = 32)
sim	file		Input L1 SIM file stack (sample rate = 1)
tstart	double	hidden	Start of time filter (sec)
tstop	double	hidden	End of time filter (sec)
aiprops_last	file		Previous L1 AIPROPS data product file
pcad_last	string		Previous pcad mode (if no aiprops_last)
pcad_sub_last	string		Previous pcad_sub mode (if no aiprops_last)
aspect_last	string		Previous aspect mode (if no aiprops_last)

Inputs:

CCDM16 - L0 CCDM engineering data (sample rate = 16 per major frame)
PCAD32 - L0 PCAD engineering data (sample rate = 32 per major frame)
AIPROPS_LAST - [OPTIONAL] Previous AIPROPS data product file

Outputs:

AIPROPS - Aspect interval properties L1 Data Product

Processing:

Asp_make_int uses engineering telemetry to determine the PCAD mode, PCAD submode, and the ASC-specific “aspect mode”, as a function of time. PCAD modes are Standby (SBM), normal pointing (NPM), normal maneuver (NMM), normal sun (NSM), powered flight (PFM), RCS maneuver (RMM), safe sun (SSM), derived rate safe sun (DSM) and RCS safe sun (RSM). Each PCAD mode may have submodes, such as SUN_ACQ, SUN_POINT, and ATT_HOLD within NSM. These mode parameters, coupled with additional information such as the spacecraft body rate and attitude quaternion, are used to derive the aspect mode. This is used in ASC processing to determine the time intervals over which pipeline processing is carried out. Each time any of the mode fields (PCAD mode, PCAD submode, or aspect mode) changes, a new aspect interval is created. The interval is a row in the AIPROPS (aspect interval properties) data product, and is specified by a start and stop time, the modes, a cut_criterion field, and a processing comment field. A new aspect interval can also be generated by a change in any of the key parameters which influence the aspect pipeline processing. Examples include change in the active gyros or a shift of more than 200 arcsec (TBR) in pointing.

This tool is implemented as a state machine, with a set of rules for each state that determine transitions between states. The tests needed for ”cutting” an interval (ending the current one and starting the next) are context dependent (i.e. different rules depending on the current mode). A state machine naturally handles this situation, as well as allowing for easy changes in configuration or extension.

The rules which define the asp_make_int state machine are implemented as files of C source code which define the elements of the following structure (`rule.h`):

```
#ifndef __RULE_H
#define __RULE_H

typedef struct {
    char* pcad; /* PCAD mode */
    char* pcad_sub; /* PCAD submode */
    char* aspect; /* Aspect mode for ASC processing */
    char* sim;           /* SIM mode for ASC processing */
} State;

typedef struct {
  char* description; /* Description of transition rule */
  int (*conditional)(void); /* Pointer to function to evaluate rule */
  State state; /* State for which this rule applies */
  State state_true; /* Next state if conditional is true */
  char* transition_code; /* Short code indicating transition reason */
  void (*transition_comment)(char*, Rule*);
                                /* Long comment with details on transition */
} Rule;

#endif
```

A rule generically defines the transition from one state to the next, with a cut in the aspect interval. Note that the two states can be the same. An example would be the case where the on-board star catalog is being edited while the spacecraft is stable in NPM with the kalman filter running. In this case the state would remain (NPM, “”, KALMAN), but the aspect pipeline would need to be re-initialized to accommodate the new stars.

A sample rule file is `rule12.c`, as follows. This rule is the one which detects when the spacecraft body rate within normal pointing mode has exceeded a certain threshold. This would occur if AXAF is “nudging” to a new, nearby target attitude.

```
/************************************************************************
*
* Code for Rule 12 :
*  Kalman to nudge mode
*
* Automatically generated by mk_rules_code.pl on Fri Jul 24 16:43:50 1998
*
*************************************************************************/

int rule12_conditional (void)
{
  int cond;

  cond = (SQR(pcad32.AORATE1) + SQR(pcad32.AORATE2) + SQR(pcad32.AORATE3)
          > SQR(NUDGE_RATE));
  return (cond);
}
```

```
void rule12_transition_comment (char *comment, Rule *r)
{

  sprintf (comment, "%s: \n" .
   "Spacecraft body rate in NPM is %f arcsec/sec (%f, %f, %f)",
           r->description,
   sqrt(SQR(pcad32.AORATE1) + SQR(pcad32.AORATE2)
                + SQR(pcad32.AORATE3)),
           pcad32.AORATE1, pcad32.AORATE2, pcad32.AORATE3);
}

void rule12_init (Rule *r)
{
  r->description = "Kalman to nudge mode";
  r->state.pcad  = "NPM";
  r->state.pcad_sub = "";
  r->state.aspect   = "KALMAN";
  r->state.sim      = "";

  r->conditional    = rule12_conditional;

  r->state_true.pcad  = "NPM";
  r->state_true.pcad_sub = "";
  r->state_true.aspect   = "NUDGE";
  r->state_true.sim      = "";

  r->transition_code = "KALMAN_TO_NUDGE";
  r->transition_comment = rule12_transition_comment;
}
```

This example shows the key elements of a rule, which are initialized in a function called rule$< N >$_init():

description: Description of rule

state: This is the state for which this rule applies. Only when the current state matches the rule state will the conditional function be evaluated. The state is defined by the three variables pcad, pcad_sub, and aspect. If a variable is null then it always matches (i.e. that variable is a "don't care" for this rule). If the value of a state variable in the rule is preceded by an exclamation point "!", then the value of the match for that variable is logically negated.

conditional: This is a pointer to a function which returns the value of a boolean conditional test based on the value of telemetry items. The example calculates the body rate and compares to a predefined constant. It is assumed all telemetry structures (e.g. pcad32) will be globally defined.

state_true: This is the new state if the conditional is true. If a variable in the state_true is null, then that state variable is left unchanged in the transition.

transition_code: If the conditional is true, then a transition to a new state will take place, and the aspect interval will be cut. This string is intended to be a short (¡32 characters) one-word code that subsequent processing can easily use to determine the nature of the transition or reason for the cut in the aspect interval.

transition_comment: If the conditional is true, this function provides a means to insert a meaningful comment (up to 255 characters) into the AIPROPS file. In the example it shows the spacecraft body rate at the point of the transition.

The collection of rules could be carried in a sub-directory of the source directory called "RULES", and incorporated in the main program as follows. By directly including the C source code in this way, instead of compiling them separately, the number of files that need to be correlated (e.g. header files and Makefile) is minimized.

```
#include "rule.h";

#include "RULES/rule1.c";
#include "RULES/rule2.c";
#include "RULES/rule3.c";
#include "RULES/rule4.c";
#include "RULES/rule5.c";
#include "RULES/rule6.c";
#include "RULES/rule7.c";
#include "RULES/rule8.c";
#include "RULES/rule9.c";
#include "RULES/rule10.c";
#include "RULES/rule11.c";
#include "RULES/rule12.c";

main() {
  State state; /* Current state of AXAF (regarding PCAD) */
  Rule rule[MAX_RULES]; /* Set of transition rules between states */

  i_rule = 0;
  rule1_init (&rule[i_rule++]);
  rule2_init (&rule[i_rule++]);
  rule3_init (&rule[i_rule++]);
  rule4_init (&rule[i_rule++]);
/*rule5_init (&rule[i_rule++]); */  /* Don't use this rule now */
  rule6_init (&rule[i_rule++]);
  rule7_init (&rule[i_rule++]);
  rule8_init (&rule[i_rule++]);
  rule9_init (&rule[i_rule++]);
  rule10_init (&rule[i_rule++]);
  rule11_init (&rule[i_rule++]);
  rule12_init (&rule[i_rule++]);
  n_rule = i_rule;

  ... more code follows ...
```

Once all the rules are defined, the actual processing is straightforward:

1. Read tool parameters

This includes start and stop times, input and output file names, and the initial state variables. If start and/or stop times are not specified, then no time filtering is performed.

2. Open input data products

   Currently this includes: the PCAD32 engineering file, which contains most of the useful PCAD status flags and variables; the SIM file, which contains the SIM positions and movement flags; and the CCDM16 file, which contains the commanded observation identifier. It is assumed in this specification that PCAD32 is the file with the fastest sampling rate.

   If `aiprops_last` parameter is defined and the file can be found, then read the last entry in this file to initialize the current state (`state.pcad, state.pcad_sub, state.aspect, state.sim`).

   If no `aiprops_last` parameter is defined,

3. Create new AIPROPS file

4. Read first record of PCAD32 and CCDM16 within start and stop times. Set `time = pcad32.time`

5. Initialize `aiprops` fields:

   `start_time = time`
   `aiprops.pcad_mode, pcad_submode,` and `aspect_mode` from parameter file or AIPROPS_LAST.

6. Find state transitions and cut intervals

   Loop through the rules for each entry in the PCAD32 file, changing states and writing new AIPROPS intervals when necessary. The following code snippet illustrates the basic flow, but ignores complications. For instance, there will be stacks of files for PCAD32 and CCDM16 which may start and stop independently.

```
/*  MAIN PROCESSING LOOP */
  for (each PCAD32 record between start and stop times) {
    pcad32_last = pcad32;
    time_last = time;

    PCAD32_GetRow(&pcad32);
    time = pcad32.time;

    if (needed)         /* if new record is available at current time */
      {
        ccdm16_last = ccdm16;
        CCDM16_GetRow(&ccdm16);
      }

    for (i_rule = 0; i_rule < n_rule; i_rule++)
      {
        if (state_match (&state, &rule[i_rule].state))
          {
            cond = (*r[i].conditional)();
            if (cond)
              {
                new_state (&state, &rule[i_rule], &aiprops,
                           time, time_last);
              }
          }
      }
  }


/***********************************/
int state_match (State *s0, State *s1)
/***********************************/
/*  NOTE: this code does not implement the logical negation described
    above if the rule state value (e.g. s0->pcad) begins with !
*/
{
  if (strlen(s0->pcad) > 0 && strlen(s1->pcad) > 0
      && strcmp(s0->pcad, s1->pcad) != 0)
    return 0;
  if (strlen(s0->pcad_sub) > 0 && strlen(s1->pcad_sub) > 0
      && strcmp(s0->pcad_sub, s1->pcad_sub) != 0)
    return 0;
  if (strlen(s0->aspect) > 0 && strlen(s1->aspect) > 0
      && strcmp(s0->aspect, s1->aspect) != 0)
    return 0

  return 1;
}
```

```
/**********************************/
void new_state (State *state, Rule *rule, Aiprops *aiprops,
                double time, double time_last)
/**********************************/
{
  if (strlen(rule->state.pcad) > 0)
    state.pcad = rule->state_true.pcad;
  if (strlen(rule->state.pcad_sub) > 0)
    state.pcad_sub = rule->state_true.pcad_sub;
  if (strlen(rule->state.aspect) > 0)
    state.aspect = rule->state_true.aspect;

  aiprops->stop_time = last_time;
  AIPROPS_PutRow (&aiprops);

  (*rule->transition_comment) (aiprops->trans_comm);
  aiprops->start_time = time;
  strcpy (aiprops->state_pcad,     state.pcad);
  strcpy (aiprops->state_pcad_sub, state.pcad_sub);
  strcpy (aiprops->state_aspect,   state.aspect);
}
```

7. Close down and clean up, including following code:

```
  aiprops.stop_time = time;
  AIPROPS_PutRow (&aiprops); /* "Close" final interval */
  AIPROPS_Close (&aiprops);
  PCAD32_Close (&pcad32);
  CCDM16_Close (&ccdm16);
```

Rules:

The currently available set of rules are listed below. These are automatically generated from an RDB file specifying the rules in a table list format.

File name	Description
rule1.c	Normal point mode to normal maneuver mode transition
rule2.c	Normal point mode to normal sun mode transition
rule3.c	Normal maneuver mode to standby mode
rule4.c	Normal maneuver mode to normal pointing mode
rule5.c	Standby to normal maneuver mode
rule6.c	Standby to normal sun mode
rule7.c	Standby to RCS maneuver mode
rule8.c	RCS maneuver mode to standby
rule9.c	RCS maneuver mode to powered flight mode
rule10.c	Powered flight mode to RCS maneuver mode
rule11.c	RCS maneuver mode to normal sun mode
rule12.c	Kalman to nudge mode
rule13.c	Nudge mode to kalman
rule14.c	Change in OBS ID
rule15.c	SIM moving
rule16.c	SIM stopped

The detailed specification of these rules are as follows. Note that where a parameter is missing (e.g. `SIM mode`), its value is taken to be the NULL string.

```
Description        | Normal point mode to normal maneuver mode transition
PCAD mode          | NORM_POINT
PCAD submode       |
Aspect mode        |
Conditional        | cond = (strcmp(pcad32.AOPCADMD, "NORM_MAN") == 0);
True PCAD mode     | NORM_MAN
True PCAD submode  | NONE
True aspect mode   | SLEW
Transition code    | NPM_TO_NMM
Transition comment| Normal point mode to normal maneuver mode transition

Description        | Normal point mode to normal sun mode transition
PCAD mode          | NORM_POINT
PCAD submode       |
Aspect mode        |
Conditional        | cond = (strcmp(pcad32.AOPCADMD, "NORM_SUN") == 0);
True PCAD mode     | NORM_SUN
True PCAD submode  | UNDEF
True aspect mode   | NORM_SUN
Transition code    | NPM_TO_NSM
Transition comment| Normal point mode to normal sun mode transition

Description        | Normal maneuver mode to standby mode
PCAD mode          | NORM_MAN
PCAD submode       |
Aspect mode       |
Conditional       | cond = (strcmp(pcad32.AOPCADMD, "STANDBY") == 0);
True PCAD mode    | STANDBY
True PCAD submode | NONE
True aspect mode  | STANDBY
Transition code   | NMM_TO_STANDBY
Transition comment|

Description       | Normal maneuver mode to normal pointing mode
PCAD mode         | NORM_MAN
PCAD submode      |
Aspect mode       |
Conditional       | cond = (strcmp(pcad32.AOPCADMD, "NORM_POINT") == 0);
True PCAD mode    | NORM_POINT
True PCAD submode | NONE
True aspect mode  | STANDBY
Transition code   |
Transition comment|

Description       | Standby to normal maneuver mode
PCAD mode         | STANDBY
PCAD submode      |
Aspect mode       |
Conditional       | cond = (strcmp(pcad32.AOPCADMD, "NORM_MAN") == 0);
True PCAD mode    | NORM_MAN
True PCAD submode | NONE
True aspect mode  | SLEW
Transition code   |
Transition comment|

Description       | Standby to normal sun mode
PCAD mode         | STANDBY
PCAD submode      |
Aspect mode       |
Conditional       | cond = (strcmp(pcad32.AOPCADMD, "NORM_SUN") == 0);
True PCAD mode    | NORM_SUN
True PCAD submode | UNDEF
True aspect mode  | NORM_SUN
Transition code   |
Transition comment|

Description       | Standby to RCS maneuver mode
PCAD mode         | STANDBY
PCAD submode      |
Aspect mode       |
Conditional       | cond = (strcmp(pcad32.AOPCADMD, "RCS_MAN") == 0);
True PCAD mode    | RCS_MAN
True PCAD submode | NONE
True aspect mode  | RCS_MAN
Transition code   |
Transition comment|
Description       | RCS maneuver mode to standby
PCAD mode         | RCS_MAN
PCAD submode      |
Aspect mode       |
Conditional       | cond = (strcmp(pcad32.AOPCADMD, "STANDBY") == 0);
True PCAD mode    | STANDBY
True PCAD submode | NONE
True aspect mode  | STANDBY
Transition code   |
Transition comment|

Description       | RCS maneuver mode to powered flight mode
PCAD mode         | RCS_MAN
PCAD submode      |
Aspect mode       |
Conditional       | cond = (strcmp(pcad32.AOPCADMD, "POW_FLT") == 0);
True PCAD mode    | POW_FLT
True PCAD submode | NONE
True aspect mode  | POW_FLT
Transition code   |
Transition comment|

Description       | Powered flight mode to RCS maneuver mode
PCAD mode         | POW_FLT
PCAD submode      |
Aspect mode       |
Conditional       | cond = (strcmp(pcad32.AOPCADMD, "RCS_MAN") == 0);
True PCAD mode    | RCS_MAN
True PCAD submode | NONE
True aspect mode  | RCS_MAN
Transition code   | POW_FLT_TO_RCS_MAN
Transition comment|

Description       | RCS maneuver mode to normal sun mode
PCAD mode         | RCS_MAN
PCAD submode      |
Aspect mode       |
Conditional       | cond = (strcmp(pcad32.AOPCADMD, "NORM_SUN") == 0);
True PCAD mode    | NORM_SUN
True PCAD submode | UNDEF
True aspect mode  | NORM_SUN
Transition code   | RCS_MAN_TO_NORM_SUN
Transition comment|

Description       | Kalman to nudge mode
PCAD mode         | NPM
PCAD submode      |
Aspect mode       | KALMAN
Conditional       |
  cond = (SQR(pcad32.AORATE1) + SQR(pcad32.AORATE2) + SQR(pcad32.AORATE3)
          > SQR(NUDGE_RATE));
True PCAD mode    | NPM
True PCAD submode |
True aspect mode  | NUDGE
Transition code   | KALMAN_TO_NUDGE
Transition comment|
  sprintf (comment, "%s: \n" .
           "Spacecraft body rate in NPM is %f arcsec/sec (%f, %f, %f)",
           r->description,
           sqrt(SQR(pcad32.AORATE1) + SQR(pcad32.AORATE2)
                + SQR(pcad32.AORATE3)),
           pcad32.AORATE1, pcad32.AORATE2, pcad32.AORATE3);
```

```
Description       | Nudge mode to kalman
PCAD mode         | NPM
PCAD submode      |
Aspect mode       | NUDGE
Conditional       |
  cond = (SQR(pcad32.AORATE1) + SQR(pcad32.AORATE2) + SQR(pcad32.AORATE3)
          < SQR(NUDGE_RATE * NUDGE_HYSTERESIS));
True PCAD mode    | NPM
True PCAD submode |
True aspect mode  | KALMAN
Transition code   | NUDGE_TO_KALMAN
Transition comment|
  sprintf (comment, "%s: \n" .
           "Spacecraft body rate in NPM is %f arcsec/sec (%f, %f, %f)",
           r->description,
           sqrt(SQR(pcad32.AORATE1) + SQR(pcad32.AORATE2)
                + SQR(pcad32.AORATE3)),
           pcad32.AORATE1, pcad32.AORATE2, pcad32.AORATE3);

Description       | Change in OBS ID
PCAD mode         |
PCAD submode      |
Aspect mode       |
Conditional       |
  cond = (ccdm16.COBSRQID != curr_obs_id);
True PCAD mode    |
True PCAD submode |
True aspect mode  |
Transition code   | OBS_ID_CHANGE
Transition comment|
  sprintf (comment, "%s: \n" .
           "Commanded observation ID changed from %d to %d\n",
           r->description,
           curr_obs_id, ccdm16.COBSRQID);
  curr_obs_id = ccdm16.COBSRQID;
```

```
Description       | SIM moving
PCAD mode         |
PCAD submode      |
Aspect mode       |
SIM mode          | STOPPED
Conditional       | cond = (sim.SIM_X_MOVED || sim.SIM_Z_MOVED);
True PCAD mode    |
True PCAD submode |
True aspect mode  |
True SIM mode     | MOVING
Transition code   | SIM_MOVING
Transition comment|
  sprintf (comment, "%s: \n" .
           "SIM_X_MOVED=%d  SIM_Z_MOVED=%d  at SIM_X=%f  SIM_Z=%f\n",
           r->description,
           sim.SIM_X_MOVED, sim.SIM_Z_MOVED, sim.SIM_X, sim.SIM_Z);

Description       | SIM stopped
PCAD mode         |
PCAD submode      |
Aspect mode       |
SIM mode          | SIM_MOVING
Conditional       | cond = (!sim.SIM_X_MOVED && !sim.SIM_Z_MOVED);
True PCAD mode    |
True PCAD submode |
True aspect mode  |
True SIM mode     | STOPPED
Transition code   | SIM_STOPPED
Transition comment|
  sprintf (comment, "%s: \n" .
           "SIM_X_MOVED=%d  SIM_Z_MOVED=%d  at SIM_X=%f  SIM_Z=%f\n",
           r->description,
           sim.SIM_X_MOVED, sim.SIM_Z_MOVED, sim.SIM_X, sim.SIM_Z);
```

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: asp_make_int

Spec Category: Aspect

Code Type: ASCDS

Code Source: New

Tool Number 1158

Description: Make the aspect quality data product.

Parameters:

aspqual	file	Output L1 ASPQUAL data product file
aspsol	file	Input L1 ASPSOL data product file
gyrodata	file	Input L1 GYRODATA data product file stack
pcad1eng	file	Input L0 PCAD eng file stack (rate = 32/mjf)
pcad5eng	file	Input L0 PCAD eng file stack (rate = 8/mjf)
rwspeed	file	Input RWSPEED file of bad reaction wheel speeds
tstart	double	Start of time filter (sec)
tstop	double	End of time filter (sec)
dt_asp_qual	double	Time step for output

Inputs:

ASPSOL - L1 ASPSOL data product file
GYRODATA - L1 GYRODATA data product file
PCAD1ENG - L0 PCAD eng file stack (rate = 32/mjf)
PCAD5ENG - L0 PCAD eng file stack (rate = 8/mjf)
RWSPEED - RWSPEED file of bad reaction wheel speeds

Outputs:

ASPQUAL - Aspect quality L1 Data Product

Overview:

The Aspect Quality (ASPQUAL) data product (DP) will contain a comprehensive set of aspect quality indicators which will be useful to a scientist analysing an observation. This product will be a binary table with one time-stamped row each 1.025 seconds, and is created by the asp_make_qualint tool in the aspect pipeline. Each quality parameter (column) will have associated values in the file header: a reference to the origin or derivation of the value, a nominal value, and yellow / red limits.

Data quality values which occur more frequently than 1.025 seconds (e.g. gyro-related data) will be combined in way appropriate for the data type; those which occur less frequently (e.g. ACA data or temperatures) will be repeated.

A small subset of the ASPQUAL (DP) will be the aspect parameters and their limits which will be used in determination of the good time intervals (GTI). The columns for each of these parameters will be copied to the master file for each observation which contains values that are screened to determine the GTIs. The red limit values in the ASPQUAL DP will be used as the good-time criteria for these aspect parameters.

The ASPQUAL DP is defined in Table 1. The current version is still a strawman, and does not show the full set of header keywords which give information about quality parameter limits.

Name	Type	Comment	Units	Dims
		Principle Header Keywords		
obs_id	int	Observation ID		
ra_nom	double	Nominal pointing RA	deg	
dec_nom	double	Nominal pointing dec	deg	
roll_nom	double	Nominal pointing roll	deg	
dtaspqul	double	Nominal record delta-time	s	
name1	char	Name of quality indicator 1		
descr1	char	Description of quality indictor 1		
yellow1	char	Yellow limit for quality indictor 1		
red1	char	Red limit for quality indictor 1		
name2	char	Name of quality indicator 2		
descr2	char	Description of quality indictor 2		
yellow2	char	Yellow limit for quality indictor 2		
red2	char	Red limit for quality indictor 2		
name3	char	Name of quality indicator 3		
descr3	char	Description of quality indictor 3		
yellow3	char	Yellow limit for quality indictor 3		
red3	char	Red limit for quality indictor 3		
.				
.				
.				
		Binary Table Extension		
time	double	Time	s	
img_diam_rms	float	Aspect image RMS diameter	arcsec	
img_roll_rms	float	Aspect image RMS roll err	arcsec	
target_offset	float	Target offset from nominal	arcsec	
img_fit_chisq	float	Image fit chi**2 worst case		
gyro_glitch	Byte	Gyro data glitch		
gyro_gap	Byte	Gyro data gap		
gyro_incons	Byte	Gyro data inconsistency		
aca_data_bad	Byte	ACA data unreliable		
rw_speed_bad	Byte	Reaction wheel speed bad		

Processing:

The processing for asp_make_qualint can be summarized as follows. Details are to be found in the prototype code.

1. Read tool parameters

2. Initialize input data product variables and open data product files

3. Initialize the data product array object for each data product

   A data product array is a C++ object that helps in "synchronizing" input from different data products that may be sampled asynchronously. It also maintains an array with the last `max_arr` records that were read up to the current record.

4. Create the ASPQUAL output data product and initialize header keywords

5. Initialize quality calculation routines

Each quality indicator (a column in the ASPQUAL data product) has associated with it two subroutines: an initialization routine and the actual code which calculates the quality indicator using values from the input data products. Currently the initialization includes hardcoded values of yellow, and red limits. This may change.

6. Write the header keywords which describe each quality indicator, as well as the presence of any yellow/red violations. Each quality indicator has a unique number $< N >$ associated with it, which may be determined at run-time. The YLVIOL$< N >$ and RDVIOL$< N >$ keywords are initially set to `FALSE`, but are determined during processing.

Keyword	Content
NAME$< N >$	Name (string, no spaces)
DESCR$< N >$	Description (string)
YELLOW$< N >$	Yellow limit (string)
RED$< N >$	Red limit (string)
YLVIOL$< N >$	Yellow limit violation found (string)
RDVIOL$< N >$	Red limit violation found (string)

7. Read the aspect solution (ASPSOL) up to, and including, the specified start time `tstart`.

8. Step the `time` from `tstart` to `tstop` by increments of `dt_asp_qual`, and perform following processing:

   (a) Read each input data product up to the current `time` using the `ReadUntilTime` method of the data product array

   (b) Calculate the value of each of the quality indictors. If this value is greater than or equal to the corresponding yellow limit, then set the appropriate YLVIOL$< N >$ header keyword to `TRUE`. If the calculated value is greater than or equal to the corresponding red limit, then set the appropriate RDVIOL$< N >$ header keyword to `TRUE`.

   (c) Set the ASPQUAL time column to the current `time`

   (d) Write the ASPQUAL record containing the calculated quality indicators

9. Close files and shut down. This includes updating the YLVIOL$< N >$ and RDVIOL$< N >$ header keywords.

Prototype code:

asp_make_qualint.cc

```
#include <stdio.h>
#include <stdlib.h>

#include "aspect_pipeline.h"
#include "pcad1.h"
#include "pcad5.h"
#include "asp_sol.h"
#include "aspqual.h"

#define MAX_QUAL 32

class Qual {
public:
  char name[20];                // Name of qual.ind. (no spaces)
  char descr[46];               // Text description
  int  int_lims;                // Are limits integers (1) or doubles (0)?
  union {
    int ival;                   // If integer
    double dval;                // If double
  }   red_lim, yel_lim, nominal; // Limits and nominal values
  void CalcQual() { (*calc_qual_func)(this); }
                                // Method to calculate qual. ind.
  void (*calc_qual_func)(Qual *qual);
                                // Pointer to actual function
};

#include "dp_array.h"           // Data product array object header file

// Global variables so all the quality calculations can have access

DP_Array pcad1eng_dpa;            // Data product array object for pcad1eng
Pcad1eng  *pcad1eng_arr[MAX_DP_ARRAY]; // Actual array maintained by pcad1eng_dpa
Pcad1eng   pcad1eng;                // Normal pcad1eng data product, also filled
                                //   in by pcad1eng_dpa

DP_Array asp_sol_dpa;           // Same as above, but for asp_sol
Asp_sol *asp_sol_arr[MAX_DP_ARRAY];
Asp_sol  asp_sol;

#include "img_diam_rms.cc"      // Code for img_diam_rms quality indicator
#include "img_roll_rms.cc"
#include "target_offset.cc"
#include "img_fit_chisq.cc"
#include "gyro_glitch.cc"
#include "gyro_gap.cc"
#include "gyro_incons.cc"
#include "aca_data_bad.cc"
#include "rw_speed_bad.cc"

main ()
{
  Qual     q[MAX_QUAL];         // Quality indictor objects
  Aspqual  aspqual;             // ASPQUAL output data product

  double time;                  // Current time
  double   dt_asp_qual = 1.025; // Time step (these will
  double   tstart = 4.2;        //   be read from param file)
  double   tstop = 40.0;

  int n_qual = 0;               // Number of quality indicators
  int i_qual;                   // Loop var for qual. ind.
  int read_ok = 1;              // Data product read was successful

  //
  // Read parameters from tool parameter file  (no code here yet)
```

```
//

//
// Initialize input data product variables and open data product files
//

// Initialize and open PCAD1ENG data product

PCAD1ENG_AllocMatrices1 (&pcad1eng);
PCAD1ENG_Open (&pcad1eng, "", "pcad1eng", FITS_READONLY);
pcad1eng.rws[RWS_READ].all_bits = ~0x0; /* read all data */
PCAD1ENG_AllocMatrices2 (&pcad1eng);

// Initialize and open ASP_SOL data product

ASP_SOL_AllocMatrices1 (&asp_sol);
ASP_SOL_Open (&asp_sol, "", "asp_truth", FITS_READONLY);
asp_sol.rws[RWS_READ].all_bits = ~0x0; /* read all data */
ASP_SOL_AllocMatrices2 (&asp_sol);

//
// Initialize the data product array object for each data product
//

// Initialize the PCAD1ENG history array

pcad1eng_dpa.DP_GetRow = PCAD1ENG_GetRow;  // Set pointer to GetRow function
pcad1eng_dpa.Init (sizeof(Pcad1eng), 8, &pcad1eng.time, &pcad1eng.rd_row,
              (void *) &pcad1eng, (void **) pcad1eng_arr);

// Initialize the ASP_SOL history array

asp_sol_dpa.DP_GetRow = ASP_SOL_GetRow;
asp_sol_dpa.Init (sizeof(asp_sol), 8, &asp_sol.time, &asp_sol.rd_row,
                (void *) &asp_sol, (void **) asp_sol_arr);

//
// Create the ASPQUAL output data product and initialize header keywords
//

ASPQUAL_AllocMatrices1 (&aspqual);

/*
aspqual.ra_nom      = asp_sol.ra_nom;
aspqual.dec_nom     = asp_sol.dec_nom;
aspqual.roll_nom    = asp_sol.roll_nom;
aspqual.dt_asp_qual = dt_asp_qual;
aspqual.asp_type    = asp_sol.asp_type;
*/

ASPQUAL_Create (&aspqual, "", "aspqual");
```

```
aspqual.rws[RWS_WRITE].all_bits = ~0x0;
ASPQUAL_AllocMatrices2 (&aspqual);

//
// Initialize quality calculation routines
//

img_diam_rms_Init (&q[n_qual++]);
//  img_roll_rms_Init (&q[n_qual++]);   // Commented out because they
//  target_offset_Init (&q[n_qual++]);  // don't exist yet
//  img_fit_chisq_Init (&q[n_qual++]);
gyro_glitch_Init (&q[n_qual++]);
gyro_gap_Init (&q[n_qual++]);
gyro_incons_Init (&q[n_qual++]);
//  aca_data_bad_Init (&q[n_qual++]);
//  rw_speed_bad_Init (&q[n_qual++]);

//
// Write the header keywords which give the name, description, yellow, and red
// limits for each quality indicator.
//

for (i_qual = 0; i_qual < n_qual; i_qual++)
  {
    // Create the name*, descr*, yellow*, red*, y_viol*, r_viol* headers
    // aspqual.fptr is the fits file pointer
    sprintf (keyname, "name%d", i_qual);
    sprintf (svalue,  "%s", q[i_qual].name);
    fits_update_key (aspqual.fptr, TSTRING, keyname, (DTYPE) svalue,
                     "Name of quality indicator (QI)", &status);

    sprintf (keyname, "descr%d", i_qual);
    sprintf (svalue,  "%s", q[i_qual].descr);
    fits_update_key (aspqual.fptr, TSTRING, keyname, (DTYPE) svalue,
                     "Description of QI", &status);

    sprintf (keyname, "yellow%d", i_qual);
    if (q[i_qual].int_lims)
      sprintf (svalue,  "%d", q[i_qual].yel_lim.ival);
    else
      sprintf (svalue,  "%e", q[i_qual].yel_lim.dval);
    fits_update_key (aspqual.fptr, TSTRING, keyname, (DTYPE) svalue,
                     "Yellow limit of QI value", &status);

    if (q[i_qual].int_lims)
      sprintf (svalue,  "%d", q[i_qual].red_lim.ival);
    else
      sprintf (svalue,  "%e", q[i_qual].red_lim.dval);
    sprintf (keyname, "red%d", i_qual);
    fits_update_key (aspqual.fptr, TSTRING, keyname, (DTYPE) svalue,
                     "Red limit of QI value", &status);
    }

  //
  // Read the aspect solution (ASPSOL) up to, and including, the specified
  // start time tstart
  //

  read_ok = 1;

  // Step the time from tstart to tstop by increments of
  // dt_asp_qual, and perform following processing:

  for (time = time0; time <= time1 && read_ok; )
    {
      // Read each input data product up to the current time using the
      // ReadUntilTime method of the data product array
      // ( This code needs to gracefully handle EOF as indicated by read_ok == 0)

      printf ("Time = %f\n", time);
      read_ok = asp_sol_dpa.ReadUntilTime (time);
      read_ok = pcad1eng_dpa.ReadUntilTime (time);

      // Calculate the value of each of the quality indictors

      for (i_qual = 0; i_qual < n_qual; i_qual++)
        {
          q[i_qual].CalcQual();
        }

      // Set the ASPQUAL time column to the current time

      aspqual.time = time;

      // Write the ASPQUAL record containing the calculated quality indicators

      ASPQUAL_PutRow (&aspqual);

      time = asp_sol.time + dt_asp_qual;
    }

  //
  // Close files and shut down
  //

  ASPQUAL_Close (&aspqual);
  ASP_SOL_Close (&asp_sol);
  PCAD1ENG_Close (&pcad1eng);

}
```

img_diam_rms.cc

```
//
// Calculate the RMS image diameter based on aspect solution error estimates
//
// This quantity is a Level-1 requirement, and indicates the blur of X-ray
// images due only to aspect (i.e. image diameter if the X-ray optics were
// perfect)
//

const double ARCSEC_PER_DEG    = 3600*360;
const double ACA_DEG_PER_SI_MM = 990. / 10000. / 0.024 * 5.0 / ARCSEC_PER_DEG;
        // Degrees in ACA FOV for each millimeter at SI focal plane

void img_diam_rms_CalcQual (Qual *q)
{
  // Convert dy, dz (motion of SIM) errors to angles in ACA field

  float dy_err_angle = asp_sol.dy_err * ACA_DEG_PER_SI_MM;
  float dz_err_angle = asp_sol.dz_err * ACA_DEG_PER_SI_MM;

  // Calculate image diameter RMS as sqrt of summed squares of the RA, dec,
  // dy, and dz errors.  The factor of 2 converts from radius to diameter

  aspqual.img_diam_rms =
    2.0 * sqrt (asp_sol.ra_err * asp_sol.ra_err
                + asp_sol.dec_err * asp_sol.dec_err
                + dy_err_angle * dy_err_angle
                + dz_err_angle * dz_err_angle);

  aspqual.img_diam_rms *= ARCSEC_PER_DEG; // convert to arcsec
}

void img_diam_rms_Init (Qual *q)
{
  strcpy (q->name,  "img_diam_rms");
  strcpy (q->descr, "Aspect image RMS diameter");
  q->int_lims = 0;
  q->yel_lim.dval = 0.5;
  q->red_lim.dval = 0.5;
  q->calc_qual_func = img_diam_rms_CalcQual;
}
```

img_roll_rms

```
//
// Calculate the RMS roll based on aspect solution error estimates
//

const double ARCSEC_PER_DEG    = 3600*360;

void img_roll_rms_CalcQual (Qual *q)
{
  aspqual.img_roll_rms = sqrt (asp_sol.roll_err * asp_sol.roll_err
       + asp_sol.dtheta_err * asp_sol.theta_err);

  aspqual.img_roll_rms *= ARCSEC_PER_DEG; // convert to arcsec
}

void img_roll_rms_Init (Qual *q)
{
  strcpy (q->name,  "img_roll_rms");
  strcpy (q->descr, "Aspect image RMS roll error");
  q->yel_lim.dval = 5.0;
  q->red_lim.dval = 20.0;
  q->calc_qual_func = img_roll_rms_CalcQual;
}
```

target_offset

```
//
// Calculate the Target offset on aspect solution and nominal pointing
//
// This code uses the Matr and Quat class libraries, with some overloaded
// operators.  The files matrix.h, matrix.cc, quaternion.h, and quaternion.cc
// can be found in /home/aldcroft/tools/asp_make_qualint/code.

#include "matrix.h"
#include "quaternion.h"

const double ARCSEC_PER_RAD    = 3600*180.0/3.14159265;

Quat q_target_inv; // Inverse of quaternion of nominal target
Quat q_offset; // Quaternion of offset from q_target
Matr<double> q_att_T(4,4); // 4x4 transform matrix for asp_sol.q_att

void target_offset_CalcQual (Qual *q)
{

  // q_att    = q_offset * q_target   so
  // q_offset = q_att * q_target^-1

  //  Prepare quaternion transformation matrix T(q) to rotate q'
  //            q'' = T(q) * q'
  //
  //                     -                    -
  //                    |  q4   q3   -q2   q1  |
  //                    |                      |
  //                    | -q3   q4    q1   q2  |
  //           q''  =   |                      |  q'
  //                    |  q2  -q1    q4   q3  |
  //                    |                      |
  //                    | -q1  -q2   -q3   q4  |
  //                     -                    -

  q_att_T[0][0] =  asp_sol.q_att[3];
  q_att_T[0][1] =  asp_sol.q_att[2];
  q_att_T[0][2] = -asp_sol.q_att[1];
  q_att_T[0][3] =  asp_sol.q_att[0];
  q_att_T[1][0] = -asp_sol.q_att[2];
  q_att_T[1][1] =  asp_sol.q_att[3];
  q_att_T[1][2] =  asp_sol.q_att[0];
  q_att_T[1][3] =  asp_sol.q_att[1];
  q_att_T[2][0] =  asp_sol.q_att[1];
  q_att_T[2][1] = -asp_sol.q_att[0];
  q_att_T[2][2] =  asp_sol.q_att[3];
  q_att_T[2][3] =  asp_sol.q_att[2];
  q_att_T[3][0] = -asp_sol.q_att[0];
  q_att_T[3][1] = -asp_sol.q_att[1];
  q_att_T[3][2] = -asp_sol.q_att[2];
  q_att_T[3][3] =  asp_sol.q_att[3];

  // Matrix multiply q_att_T(4,4) by the 4-vector q_target_inv.q(4),
  // and set the offset quaternion to the result

  q_offset = q_att_T * q_target_inv.q; // (check to see if Matr class does
       //  this correctly!)

  d_pitch = 2.0 * q_offset.q[0] * ARCSEC_PER_RAD; // offset in pitch (arcsec)
  d_yaw   = 2.0 * q_offset.q[1] * ARCSEC_PER_RAD; // offset in yaw (arcsec)

  aspqual.target_offset = sqrt (d_pitch*d_pitch + d_yaw*d_yaw);
}

void target_offset_Init (Qual *q)
{
  strcpy (q->name,  "target_offset");
  strcpy (q->descr, "Target offset from nominal RA, Dec");
  q->yel_lim.dval = 30.0;
  q->red_lim.dval = 30.0;
  q->calc_qual_func = target_offset_CalcQual;

  // Set the nominal target quaternion inverse based on the
  // nominal RA, Dec, Roll in aspect solution header

  q_target_inv.Set (asp_sol.ra_nom, asp_sol.dec_nom, asp_sol.roll_nom);
  q_target_inv.q[3] *= -1.0; // Invert q_target
}
```

img_fit_chisq

```
//
// Calculate the worst case image fit chi^2, for any of the 8 image slots
//
// Since ACA centroid measurements are somewhat

void img_fit_chisq_CalcQual (Qual *q)
{
  float max_chisq = -1.0;
  static float last_max_chisq = -1.0;

  // figure out how far back we can look (normally max_arr, except on startup)
  float n_lookback = min (acacent_dpa.max_arr, acacent_dpa.n_arr);

  for (i = 0; i < n_lookback; i++)
    {
      // Check to see if we are looking back past the interval
      if ((time - acacent_arr[i]->time) > dt_asp_qual)
{
  break;
}

      // Sergey: check with Mark to see how to get ALG_USED_IN_ASP_SOLUTION

      // Check if this measurement is the correct algorithm and
      // that it has status == 0 (OK) and
      // that it exceeds the current maximum of chisq
      if (acacent_arr[i]->alg == ALG_USED_IN_ASP_SOLUTION
  && acacent_arr[i]->status == 0
  && acacent_arr[i]->chisq > max_chisq )
{
  max_chisq = acacent_arr[i]->chisq;
}
    }

  if (max_chisq > 0.0)
    {
      aspqual.img_fit_chisq = max_chisq;
      last_max_chisq = max_chisq;
    }
  else
    {
      aspqual.img_fit_chisq = last_max_chisq;
    }
}

void img_fit_chisq_Init (Qual *q)
{
  strcpy (q->name,  "img_fit_chisq");
  strcpy (q->descr, "Image fit chi**2 worst case");
  q->yel_lim.dval = 5.0;
  q->red_lim.dval = 20.0;
  q->calc_qual_func = img_fit_chisq_CalcQual;
}
```

gyro_incons.cc

```
//
// Compute the gyro inconsistency field in the aspect quality data product
//
// The gyro_incons byte indicates whether there was an self-inconsistency
// amongst the four gyro channels.  Since only three axes of gyro data are
// needed to determine a spacecraft body rate, it is possible check each
// gyro channel and assure that it is consistent with the rate from the
// other three.  During gyro_process, certain bits of the status bytes
// (status0, status1, status2, status3) of the GYRODATA product are set
// to indicate a consistency error.

#include "status_flag_definitions.h"

const double dt_gyro = 0.25625; // Should come from GYRODATA, but we don’t
                                // have it in there now..

void gyro_incons_CalcQual (Qual *q)
{
  int i;
  int n_gyro_data = nint (dt_asp_qual / dt_gyro);

  aspqual.gyro_incons = 0;

  // set bits 0 - 3 of aspqual.gyro_incons to the logical-or of the
  // CONSIST_ERR bit (during the last dt_asp_qual) of gyrodata.status 0 - 3,
  // respectively.

  for (i = 0; i < n_gyro_data; i++)
    {
      if (gyrodata_arr[i]->status0 & CONSIST_ERR) // status for gyro 0
        aspqual.gyro_incons |= 1;
      if (gyrodata_arr[i]->status1 & CONSIST_ERR) // status for gyro 1
        aspqual.gyro_incons |= 2;
      if (gyrodata_arr[i]->status2 & CONSIST_ERR) // gyro 2
        aspqual.gyro_incons |= 4;
      if (gyrodata_arr[i]->status3 & CONSIST_ERR) // gyro 3
        aspqual.gyro_incons |= 8;
    }
}

void gyro_incons_Init (Qual *q)
{
  strcpy (q->name,  "gyro_incons");
  strcpy (q->descr, "Gyro rate self-inconsistency");
  q->int_lims = 1;
  q->yel_lim.ival = 1;
  q->red_lim.ival = 1;
  q->calc_qual_func = gyro_incons_CalcQual;
}
```

gyro_glitch.cc

```
//
// Compute the gyro glitch field in the aspect quality data product
//
// The gyro_glitch byte indicates whether there was a glitch in the
// gyro data   During gyro_process, certain bits of the status bytes
// (status0, status1, status2, status3) of the GYRODATA product are set
// to indicate that the data had a glitch and were rejected by the
// sigma-edit routine.

#include "status_flag_definitions.h"

const double dt_gyro = 0.25625; // Should come from GYRODATA, but we don't
                                // have it in there now..

void gyro_glitch_CalcQual (Qual *q)
{
  int i;
  int n_gyro_data = nint (dt_asp_qual / dt_gyro);

  aspqual.gyro_glitch = 0;

  // set bits 0 - 3 of aspqual.gyro_glitch to the logical-or of the
  // CONSIST_ERR bit (during the last dt_asp_qual) of gyrodata.status 0 - 3,
  // respectively.

  for (i = 0; i < n_gyro_data; i++)
    {
      if (gyrodata_arr[i]->status0 & SIGMA_REJECT) // status for gyro 0
        aspqual.gyro_glitch |= 1;
      if (gyrodata_arr[i]->status1 & SIGMA_REJECT) // status for gyro 1
        aspqual.gyro_glitch |= 2;
      if (gyrodata_arr[i]->status2 & SIGMA_REJECT) // gyro 2
        aspqual.gyro_glitch |= 4;
      if (gyrodata_arr[i]->status3 & SIGMA_REJECT) // gyro 3
        aspqual.gyro_glitch |= 8;
    }
}

void gyro_glitch_Init (Qual *q)
{
  strcpy (q->name,  "gyro_glitch");
  strcpy (q->descr, "Gyro data glitch");
  q->int_lims = 1;
  q->yel_lim.ival = 1;
  q->red_lim.ival = 1;
  q->calc_qual_func = gyro_glitch_CalcQual;
}
```

gyro_gap.cc

```
//
// Compute the gyro gap field in the aspect quality data product
//
// The gyro_gap byte indicates whether there was a gap in the
// gyro data   During gyro_process, certain bits of the status bytes
// (status0, status1, status2, status3) of the GYRODATA product are set
// to indicate that a gap in the data.

#include "status_flag_definitions.h"

const double dt_gyro = 0.25625; // Should come from GYRODATA, but we don't
                                // have it in there now..

void gyro_gap_CalcQual (Qual *q)
{
  int i;
  int n_gyro_data = nint (dt_asp_qual / dt_gyro);

  aspqual.gyro_gap = 0;

  // set bits 0 - 3 of aspqual.gyro_gap to the logical-or of the
  // CONSIST_ERR bit (during the last dt_asp_qual) of gyrodata.status 0 - 3,
  // respectively.

  for (i = 0; i < n_gyro_data; i++)
    {
      if (gyrodata_arr[i]->status0 & MISSING_DATA) // status for gyro 0
        aspqual.gyro_gap |= 1;
      if (gyrodata_arr[i]->status1 & MISSING_DATA) // status for gyro 1
        aspqual.gyro_gap |= 2;
      if (gyrodata_arr[i]->status2 & MISSING_DATA) // gyro 2
        aspqual.gyro_gap |= 4;
      if (gyrodata_arr[i]->status3 & MISSING_DATA) // gyro 3
        aspqual.gyro_gap |= 8;
    }
}

void gyro_gap_Init (Qual *q)
{
  strcpy (q->name,  "gyro_gap");
  strcpy (q->descr, "Gyro data gap");
  q->int_lims = 1;
  q->yel_lim.ival = 1;
  q->red_lim.ival = 1;
  q->calc_qual_func = gyro_gap_CalcQual;
}
```

aca_data_bad

```
//
// Determine if there are bad ACA data in this interval.  Currently this
// simply means checking if the status flag for each image centroid is 0.
//

void aca_data_bad_CalcQual (Qual *q)
{
  aspqual.aca_data_bad = 0;
  const char ONE = 1;

  // figure out how far back we can look (normally max_arr, except on startup)
  float n_lookback = min (acacent_dpa.max_arr, acacent_dpa.n_arr);

  for (i = 0; i < n_lookback; i++)
    {
      // Check to see if we are looking back past the interval
      if ((time - acacent_arr[i]->time) > dt_asp_qual)
{
  break;
}

      // Sergey: check with Mark to see how to get ALG_USED_IN_ASP_SOLUTION

      // Check if this measurement is the correct algorithm and that it is bad
      if (acacent_arr[i]->alg == ALG_USED_IN_ASP_SOLUTION
  && acacent_arr[i]->status != 0)
{
  // indicate which slot was bad
  aspqual.aca_data_bad |= (ONE << acacent_arr[i]->slot);
}
    }
}

void aca_data_bad_Init (Qual *q)
{
  strcpy (q->name,  "aca_data_bad");
  strcpy (q->descr, "ACA data unreliable");
  q->yel_lim.ival = 1;
  q->red_lim.ival = 1;
  q->calc_qual_func = aca_data_bad_CalcQual;
}
```

rw_speed_bad

```
//
// RW_SPEED_BAD quality indicator
//
// Determine if the current reaction wheel assembly (RWA) speeds fall in the
// range of known bad intervals.  In these intervals, vibrations from the RWA
// can resonantly excite the HRMA, causing X-ray image blurring.
//

void rw_speed_bad_CalcQual (Qual *q)
{
  aspqual.rw_speed_bad = 0;
  const char ONE = 1;

  if (n_rwspeed == 0)           // No bad reaction wheel speeds
    {
      return;
    }

  // Currently, the sample rate for reaction wheel speeds is 8 / major frame
  // (or 1 sample / 4.100 sec), and the sampling for quality is 1 sample / 1.025 sec.
  // In order to accomodate future changes in sample rates, use the generic mechanism
  // of looking back at all records which fall in the current ASPQUAL interval.

  // figure out how far back we can look (normally max_arr, except on startup)

  int n_lookback = min (pcad5eng_dpa.max_arr, pcad5eng_dpa.n_arr);

  for (i = 0; i < n_lookback; i++)
    {
      // Check to see if we are looking back past the interval, but make sure we
      // include at least the first record

      if (i > 0 && (time - pcad5eng_arr[i]->time) > dt_asp_qual)
        {
          break;
        }

      // Copy the telemetred wheel speeds into an array for easier processing

      rwspeed_tlm[1] = pcad5eng_arr[i].AORWSPD1;
      rwspeed_tlm[2] = pcad5eng_arr[i].AORWSPD2;
      rwspeed_tlm[3] = pcad5eng_arr[i].AORWSPD3;
      rwspeed_tlm[4] = pcad5eng_arr[i].AORWSPD4;
      rwspeed_tlm[5] = pcad5eng_arr[i].AORWSPD5;
      rwspeed_tlm[6] = pcad5eng_arr[i].AORWSPD6;

      // For each bad wheel speed range, check if the relevant RWA's are in that
      // range.  If so, set corresponding bit in aspqual.rw_speed_bad
      for (i_rwspeed = 0; i_rwspeed < n_rwspeed; i_rwspeed++)
        {
          rwa_id = rwspeed_arr[i_rwspeed].rwa_id;
          i_rwa0 = (rwa_id == 0) ? 1 : rwa_id;
          i_rwa1 = (rwa_id == 0) ? 6 : rwa_id;

          for (i_rwa = i_rwa0; i_rwa <= i_rwa1; i_rwa++)
            {
              if (rwspeed_arr[i_rwspeed].speed0 <= rwspeed_tlm[i_rwa]
                  && rwspeed_tlm[i_rwa] <= rwspeed_arr[i_rwspeed].speed1)
                {
                  aspqual.rw_speed_bad |= (ONE << i_rwa);
                }
            }
        }
    }
}

void rw_speed_bad_Init (Qual *q)
{
  strcpy (q->name,  "rw_speed_bad");
  strcpy (q->descr, "Reaction wheel speed bad");
  q->yel_lim.ival = 1;
  q->red_lim.ival = 1;
  q->calc_qual_func = rw_speed_bad_CalcQual;

  // Some initialization needs to be done, either here or in main code.
  // Following is rough pseudo-code.  The "C-style" comments are just
  // word descriptions of required code.

#define MAX_RWSPEED 500
  Rwspeed rwspeed;              // FITS I/O layer for reading rwspeed file
  Rwspeed *rwspeed_arr[MAX_RWSPEED];

  /* Open FITS file specified by parameter rwspeed */

  n_rwspeed = rwspeed.n_rows;

  /* Allocate memory for rwspeed_arr[n_rwspeed] */

  i_rwspeed = 0;

  while (RWSPEED_GetRow (&rwspeed))
    {
      /* Set rwspeed_arr[i_rwspeed] = rwspeed */

      // This could be done by copying the relevant elements, or by a direct memory
      // copy.  The latter would be robust to possible changes in column names etc

      i_rwspeed++;
    }
  if (n_rwspeed != i_rwspeed)
    {
      /* Error, mismatch between expected and actual number of file records */
    }

  /* Close rwspeed file
}
```

dp_array.h

```
#ifndef __DP_Array_h
#define __DP_Array_h

#define MAX_DP_ARRAY 64

class DP_Array {
public:
  double  time[MAX_DP_ARRAY];
  int     n_arr;
  int     (*DP_GetRow)(void *);

  void    Init(size_t c_DP_size,
               int c_max_arr,
               double *c_time_col,
               int *c_DP_rd_row,
               void *c_DP_ptr,
               void **c_DP_arr);
  int     ReadUntilTime(double time);

private:
  char   *ring_arr[MAX_DP_ARRAY];
  double  ring_time[MAX_DP_ARRAY];
  double  delta_time;

  void   *DP_ptr;
  void  **DP_arr;
  double *DP_time_col;
  int    *DP_rd_row;
  int     DP_size;

  int     i_arr;
  int     max_arr;
  char   *arr_mem;

};

#endif
```

dp_array.h

```
#include <stdio.h>
#include <stdlib.h>

#include "dp_array.h"

int DP_Array::ReadUntilTime(double next_time)
{
  while (next_time >= ring_time[i_arr] + delta_time - 1e-6)
    {
      if (! (*DP_GetRow)(DP_ptr))
        return 0;
      n_arr++;
      i_arr = (n_arr - 1) % max_arr;

      memcpy ((void *) ring_arr[i_arr], DP_ptr, DP_size);
      ring_time[i_arr] = *DP_time_col;
    }

  int i, j, k;
  for (i = n_arr; i >= 1 && i > n_arr - max_arr; i--)
    {
      j = n_arr - i;
      k = (i-1) % max_arr;
      time[j] = ring_time[k];
      DP_arr[j] = (void *) ring_arr[k];
    }

  return 1;
}

void DP_Array::Init(size_t c_DP_size,
                    int c_max_arr,
                    double *c_DP_time_col,
                    int *c_DP_rd_row,
                    void *c_DP_ptr,
                    void **c_DP_arr)
{
  DP_ptr   = c_DP_ptr;
  DP_arr   = c_DP_arr;
  DP_size  = c_DP_size;
  DP_time_col = c_DP_time_col;
  DP_rd_row = c_DP_rd_row;
  max_arr  = c_max_arr;

  if (max_arr > MAX_DP_ARRAY)
    {
      fprintf (stderr, "Error - max_arr exceeds DP_MAX_ARRAY\n");
      exit(1);
    }
  arr_mem = (char *) calloc (max_arr, DP_size);
  for (int i = 0; i < max_arr; i++)
    {
      ring_arr[i] = arr_mem + i * DP_size;
    }

  (*DP_GetRow)(DP_ptr);
  double time0 = *DP_time_col;

  (*DP_GetRow)(DP_ptr);
  double time1 = *DP_time_col;

  delta_time = time1 - time0;

  // 'rewind' file and re-read first row
  *DP_rd_row = 0;
  (*DP_GetRow)(DP_ptr);

  n_arr = 1;
  i_arr = 0;
}
```

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: asp_make_qualint

Spec Category: Aspect

Code Type: ASCDS

Code Source: Prototype

Tool Number 2040

asp_obc_solve

Description: Make the aspect solution data product using only PCAD engineering, SIM engineering, and CALDB aspect alignments

Parameters:

asp_sol	file	Output L1 ASPSOL aspect data product file
aca_fts_align	file	ACA and FTS alignments file
pcad_eng_obc	files	Stack of PCADENG L0 files with OBC data
pcad_eng_target	files	Stack of PCADENG L0 files with target attitude data
sim_eng	files	Stack of SIM L0.5 files
tstart	double	Start of time filter (sec)
tstop	double	End of time filter (sec)
t_ref	double	Reference time for ASPSOL header data
ra_nom	double	Nominal target RA (deg) (optional)
dec_nom	double	Nominal target Dec (deg) (optional)
roll_nom	double	Nominal target Roll (deg) (optional)
aspsol_frame	string	Aspect solution frame (MNC|ACA)

Inputs:

aca_fts_align - Aspect CALDB ACA and FTS alignments file
PCADENG1 - L0 PCAD engineering with OBC attitude data
PCADENG2 - L0 PCAD engineering with target attitude data
SIMENG - L0.5 SIM engineering

Outputs:

ASPSOL - L1 aspect solution data product

Processing: The asp_obc_solve tool converts on-board computer (OBC) PCAD aspect data to the format and coordinate system of the ASC aspect solution. For a discussion of the aspect solution itself, see the tool specification for asp_solve. The PCAD engineering aspect data reflect the on-board Kalman filter estimate of the AXAF attitude and gyro biases, using ACA and gyro data.

This tool will normally be used as part of the standard L1 aspect pipeline processing, producing the input data required for spoiler star detection. However, it also gives a quick way of getting an aspect solution in a form suitable for subsequent L1 processing. The loss in accuracy (due to the use of first moment centroiding, and no fid light solution) will be acceptable in certain situations, especially during OAC or anomaly resolution.

Processing takes place by the following steps:

1. If `tstart` is not defined, then set it to the first value of `time` in the first `pcad_eng_obc` file
2. If `tstop` is not defined, then set it to the last value of `time` in the last `pcad_eng_obc` file

3. If t_ref is not defined, then set it to tstart. If t_ref < tstart, set t_ref = tstart. If t_ref > tstop, set t_ref = tstop.

4. Create / open the ASPSOL file for writing (asp_sol), using FITS I/O structure aspsol

5. Get the ACA alignment matrices:

   (a) Open the ACA and FTS alignments FITS file (parameter aca_fts_align) with data I/O structure align, and read the first (and only) row

   (b) Copy the three aligment matrices as follows

   ```
   for (i = 0; i < 3; i++) {
     for (j = 0; j < 3; j++) {
       aca_align[i][j] = align.aca_sc_align[i][j];
       aca_misalign[i][j] = align.aca_misalign[i][j];
     }
   }
   ```

6. Get the SIM_X,Y,Z values from L0.5 SIM telemetry:

   (a) Read columns time, sim_x, sim_y, sim_z from the stack of SIMENG files (sim_eng), using FITS I/O identifier simeng, until simeng.time >= t_ref.

   (b) Set aspsol.sim_{x,y,z} = simeng.sim_{x,y,z}

7. If any of the parameters ra_nom, dec_nom, or roll_nom are undefined, then get values from the PCAD engineering target attitude file stack:

   (a) Read columns time, aotarqt1, aotarqt2, aotarqt3 from the stack of PCADENG2 files ((pcad_eng_target)), using FITS I/O identifier pcadeng2, until pcadeng2.time >= t_ref.

   (b) Convert target quaternion to RA, Dec, Roll

   ```
   Quat target (pcadeng2.aotarqt1, pcadeng2.aotarqt2, pcadeng2.aotarqt3,
                1.0 - sqrt( sum of squares of pcadeng2.aotarqt1,2,3 ) );
   target.SetTransformMatr ();
   target.Get (target_ra, target_dec, target_roll);
   ```

   (c) Set ASPSOL header keywords to parameter value (if defined) or PCAD engineering target attitude value:

   ```
   aspsol.ra_nom = (defined ra_nom) ? ra_nom : target_ra;
   aspsol.dec_nom = (defined dec_nom) ? dec_nom : target_dec;
   aspsol.roll_nom = (defined roll_nom) ? roll_nom : target_roll;
   ```

8. Calculate the inverse rotation matrix and quaternion transformation matrix to go from the ACA frame to the MNC (HRMA mirror nodal coordinates) frame:

   ```
   /* Set the rotation matrix which transforms from ACA to MNC frame for  */
   /* 3-vector,  e.g.  v_MNC = aca2mnc * v_ACA                            */

   aca2mnc = inverse(aca_misalign * aca_align)  /* inverse() is matrix inverse */

   /* Set the 4x4 quaternion transform matrix, which rotates a quaternion in */
   /* the ACA frame to the MNC frame e.g.  q_MNC = Tq_aca2mnc * q_ACA        */

   Quat q_aca2mnc;               /* Declare quaternion */
   ```

```
Matr<double> Tq_aca2mnc(4,4);  /* Declare 4x4 quat. transform matrix */
q_aca2mnc.Set (aca2mnc); /* Set quaternion from rotation matrix */
float* q = q_aca2mnc.q; /* For notational convenience */

/* Actually set matrix.  (Note that line below is pseudo-code) */

Tq_aca2mnc = [[  q[3] ,  q[2] ,  -q[1] ,  q[0]  ],
              [ -q[2] ,  q[3] ,   q[0] ,  q[1]  ],
              [  q[1] , -q[0] ,   q[3] ,  q[2]  ],
              [ -q[0] , -q[1] ,  -q[2] ,  q[3]  ]]
```

9. Open the PCADENG1 file (pcad_eng_obc) for reading, using FITS I/O structure pcadeng1

10. Set aspsol header variables:

```
ASPSOL
-------
ACSYS1    = (aspsol_frame == "ACA") ? "ASPSOL=ACA" : "ASPSOL=MNC"
TIMEDEL   = pcadeng1.TIMEDEL
dtaspsol  = pcadeng1.TIMEDEL
asptype   = "OBC"
biastype  = "OBC"
equinox   = 2000.0
```

11. Read each record of the PCADENG1 file between time tstart and tstop, and do the following:

    (a) Set internal variables corresponding to OBC attitude quaternion and gyro bias

    ```
    aoattqt = Quat(aoattqt1, aoattqt2, aoattqt3, aoattqt4)
    aogbias[0] = aogbias1
    aogbias[1] = aogbias2
    aogbias[2] = aogbias3
    ```

    (b) Unless the parameter aspsol_frame == "ACA", transform the attitude quaternion and gyro bias into the MNC frame:

    ```
    aoattqt = Tq_aca2mnc * aoattqt;  /* Note that Quat class doesn’t support */
    aogbias = aca2mnc * aogbias;     /*   these assignments as written */
    ```

    (c) Convert the quaternion aoattqt to an RA, Dec, and Roll and store in ra, dec, and roll columns of the ASPSOL data product:

    ```
     aoattqt.SetTransformMatr ();
     aoattqt.Get (aspsol.ra, aspsol.dec, aspsol.roll);
    ```

    (d) Set the attitude error columns of ASPSOL, using the error columns of PCADENG1:

    ```
    roll_err = sqrt(pcadeng1.aoaderr1);
    ra_err   = sqrt(pcadeng1.aoaderr2 * sqr(sin(aspsol.roll)) +  /* pitch error */
                    pcadeng1.aoaderr3 * sqr(cos(aspsol.roll)));  /* yaw error */
    dec_err  = sqrt(pcadeng1.aoaderr2 * sqr(cos(aspsol.roll)) +  /* pitch error */
                    pcadeng1.aoaderr3 * sqr(sin(aspsol.roll)));  /* yaw error */
    ```

    (e) Set the remaining aspsol variables:

```
ASPSOL
-------
time             = pcadeng1.time
q_att[*]         = aoattqt[*]
bias_rate[*]   = aogbias[*]
bias_err[0]    = sqrt(pcadeng1.aogderr1)
bias_err[1]    = sqrt(pcadeng1.aogderr2)
bias_err[2]    = sqrt(pcadeng1.aogderr3)
dy               = 0.0
dz               = 0.0
dtheta           = 0.0
dy_err           = 0.0
dz_err           = 0.0
dtheta_err       = 0.0

/* Keep track of limits for pitch and yaw */

if (pcadeng1.aoatter2 > max_pitch) max_pitch = pcadeng1.aoatter2;
if (pcadeng1.aoatter2 < min_pitch) min_pitch = pcadeng1.aoatter2;
if (pcadeng1.aoatter3 > max_yaw) max_yaw = pcadeng1.aoatter3;
if (pcadeng1.aoatter3 < min_yaw) min_yaw = pcadeng1.aoatter3;
```

(f) Write the ASPSOL record

12. Close the PCADENG1 file
13. Set `aspsol.pitchamp = (max_pitch - min_pitch) * 180.0/3.14159265` and `aspsol.yawamp = (max_yaw - min_yaw) * 180.0/3.14159265`
14. Write ASPSOL header, and close file

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: asp_solve

Spec Category: Aspect

Code Type: ASCDS

Code Source: New

Tool Number 1250

aca_smooth_kalman

Description:

Derive optimally smoothed attitude and gyro drift rate estimates.

Parameters:

KALMAN file	file	L1 Kalman Data Product
GYRODATA file	file	L1 Gyro Data
ACASOL file	file	L1 Aspect Solution

Inputs:

KALMAN - Kalman Data Product
GYRODATA - Gyro Data

Outputs:

KALMAN - Kalman Data Product
ACASOL - Aspect Solution Data Product

Processing: Process Kalman solution backwards in time by a sequential smoothing algorithm to derive optimally smoothed attitude and gyro drift rate estimates.

Run Smoothing Filter Algorithm (Rauch, Tung and Stribel) described in detail in TRW-SE11k 11 Jan 1996.

Use as inputs corrected gyro angular rates from Gyro Data, Star ECI positions from Guide Star Properties, alignment matrices from ACA Calibration, scale factors and outputs from the forward Kalman filter. The latter include:

- state transition matrix
- post measurement covariance matrix
- pre measurement covariance matrix
- pre measurement state vector
- post measurement state vector

The Smoothing filter will calculate the following:

- smoothing filter attitude quaternion estimate
- smoothed covariance matrix

- smoothed covariance matrix in reverse order (chronological time order)
- smoothed state vector
- smoothed state vector in reverse order (chronological time order)

Write Kalman ACA attitude solution and covariance matrix to Kalman Data file.

Finally, derive star field motion.

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: aca_smooth_kalman

Spec Category: Aspect

Code Type: ASCDS

Code Source: GADS Prototype

Tool Number 1179

Description: Make the aspect solution data product.

Parameters:

kalman	file		Input L1 KALMAN aspect data product file
aca_cal	file		Input L1 ACACAL aspect data product file
aca_cent	file		Input L1 ACACENT aspect data product file
fid_props	file		Input L1 FIDPROPS aspect data product file
asp_sol	file		Output L1 ASPSOL aspect data product file
tstart	double	hidden	Start of time filter (sec)
tstop	double	hidden	End of time filter (sec)
t_err	double	hidden	Length of time for calculating fid errors (sec)
n_step	int		Number of ASPSOL records per fid calculation
polint_order	int		Polynomial order for interpolation
theta_X_start	double	hidden	Initial guess at theta_X
dy_start	double	hidden	Initial guess at dy
dz_start	double	hidden	Initial guess at dz
fit_alambda	double	hidden	Value of alambda in fitting routine
t_err	double	hidden	Length of time for calculating fid errors (sec)

Inputs:

ACACAL - L1 ACA calibration data product
ACACENT - L1 centroids aspect data product
FIDPROPS - L1 fiducial light properties aspect data product
KALMAN - L1 kalman filter data product

Outputs:

ASPSOL - L1 aspect solution data product

Processing:

There are two fundamental quantities needed to assign a celestial location to a photon: the RA, Dec, and roll of the HRMA axis, and the location of the photon in HRMA nodal coordinates (HNC). With this information, one can determine the photon direction (either with a simple thin-lens model, or a more complex energy-dependent model) and therefore assign a celestial location to the photon.

It should be noted that the above differs in perspective from the original concept for the "FPSI aspect solution." In that paradigm we would provide the angular offset and roll history for the "fid light field". However, this information is not well-suited to the manner in which X-ray photons are processed.

HRMA axis

The celestial location and roll of the HRMA axis (X_N) is derived from the celestial location solution for the ACA X-axis. This is the product of kalman filtering the ACA and IRU data, as described in ASC documents SE03 and DS01. It is assumed that there is a small, non-varying misalignment between the ACA and HNC frames, which is measured during boresight calibration. The ACA aspect solution can then be easily transformed into the HNC frame.

Photon location in HNC and FC frames

The fundamental information which is provided by the fiducial lights is how to locate detected photons in the FC frame[1]. The ASC Coordinates document[2] describes the path by which a CHIP pixel location is transformed into an RA and Dec. This path begins with the transformations: CHIP → CPC → LSI → STT → STF → FC. However, the last two transformations have uncertainties: the SIM translation step size or offset may be uncertain (affecting STT → STF), and more importantly, the OBA could be flexing (affecting STF → FC). A more subtle shift occurs when the ACA mount alignment shifts, which mimics a flexure of the OBA. It is the role of the fid lights to precisely locate the STT frame in the FC system. In practice, the aspect pipeline will assume the same transformation law for STT → STF as does the X-ray photon processing, and return the appropriate STF → FC transformation as output. Nominally, the STF and FC frames are coincident.

The STF → FC transformation can be specified by a rotation followed by a translation, which is generally a function of six variables. However, the measurement of fid lights by the ACA only provide "observability" of three variables: displacement in Y, displacement in Z, and a rotation about X. These three variables are the primary output of this tool, and they are derived by the following steps. In this specification the displacement is a three vector called `stf0_fc` (with `stf0_fc[0] = 0`), and the rotation is `theta_X`.

1. Open the ACACENT data product, which has a time history of the fid light angles, smoothed on timescale T_{fid}, in the ACA frame. The columns `ang_y_sm, ang_z_sm` contain the smoothed angles for each fid light.

2. For each fid light i, estimate the noise in smoothed angles by calculating the RMS of data points with `tstart <= time < tstart + t_err`. Call the values `ang_y_err[i]` and `ang_z_err[i]`. Typically `t_err` will be ~ 100 sec.

3. Read configuration data for each fid light i from the FIDPROPS data product, as listed below. The `fid` structure array contains information about the expected properties of up to four active fid lights. The nomenclature `p_xxx` implies a 3-d position in frame `xxx`. Later on we use `d_xxx`, which is a unit-length direction vector in frame `xxx`.

   - Position in LSI coordinates: `fid[i].p_lsi` (3-vector)
   - ACA image slot number: `fid[i].slot`

4. Read SI offset and alignment data from FIDPROPS:

   - Origin of LSI frame (for current SI) in STT frame: `lsi0_stt` (3-vector)
   - Origin of STT frame in STF frame: `stt0_stf` (3-vector) (i.e. SIM-z and focus)
   - Fid light to ACA focal point distance: `fc0_2aca_fp_len`

5. Read alignment data from ACACAL

   - FTS misalignment: `fts_misalign` (3×3 matrix)
   - ACA nominal alignment (spacecraft to ACA nom.): `fc2acaN` (3×3 matrix)
   - ACA misalignment (ACA nom. to ACA): `acaN2aca` (3×3 matrix)

6. Calculate the nominal position of each fid light in STF coordinates.

   This requires doing the transformation LSI → STT → STF, using as input the nominal fid light positions in LSI, the SIM translation, and SIM focus. No misalignment is assumed at this stage.

[1] The focal coordinate (FC) frame is just the HNC frame displaced by the focal length of the telescope. FC is more natural than HNC for aspect-related transformations

[2] http://hea-www.harvard.edu/asclocal/sds/docs/jcmdocs/ps/SDS02.ps

```
fid[i].p_stf = fid[i].p_lsi + lsi0_stt + stt0_stf
```

7. Fill in aspect solution columns from Kalman filter output:

   (a) Open the KALMAN file for reading

   (b) Create / open the ASPSOL file for writing

   (c) Read `time`, `q_att_est`, `roll_bias`, `pitch_bias`, `yaw_bias`, and `covariance` for each row of the entire KALMAN file and do the following:

      i. Convert the quaternion `q_att_est` from the ACA frame to the MNC frame:

      ```
        // Initialize quaternion which represents fixed transformation from
        // ACA to MNC frame.  This should be done once at the beginning of
        // processing.  Since fc2acaN and acaN2aca are unitary, we can
        // invert them by taking transpose.  This gives the transformation matrix
        // ACA to ACA_nominal to FC (which is effectively the same as MNC).
        // The matrix is then converted to a quaternion using Set().
        Quat q_aca2mnc;
        q_aca2mnc.Set (fc2acaN.Transpose() * acaN2aca.Transpose());

        // Do transform using overloaded quaternion * operator
        q_att_mnc = q_aca2mnc * q_att_est;
      ```

      ii. Convert the quaternion `q_att_mnc` to an RA, Dec, and Roll and store in `ra`, `dec`, and `roll` columns of the ASPSOL file.

      iii. Convert the `covariance` to kalman errors using:

      ```
      roll_err = sqrt(covariance[0][0]);
      ra_err = sqrt(covariance[1][1] * sqr(sin(roll)) +  /* pitch error */
                        covariance[2][2] * sqr(cos(roll)));  /* yaw error */
      dec_err = sqrt(covariance[1][1] * sqr(cos(roll)) +  /* pitch error */
                         covariance[2][2] * sqr(sin(roll)));  /* yaw error */
      ```

      iv. Copy the following from KALMAN to ASPSOL:

      ```
      ASPSOL       KALMAN
      ------       -------
      time         = time
      quat         = q_att_mnc
      bias_rate[0] = roll_bias
      bias_rate[1] = pitch_bias
      bias_rate[2] = yaw_bias
      bias_err[0]  = sqrt(covariance[0][0])
      bias_err[1]  = sqrt(covariance[1][1])
      bias_err[2]  = sqrt(covariance[2][2])
      ```

      v. Write the ASPSOL record

   (d) Close the KALMAN file and the ASPSOL file

8. Open the ASPSOL file with read / write access. (*If a file can be created with read/write access, then do so in the previous step and simply rewind the file here.*) If the parameter `tstart` is undefined, set it `aspsol.time` for the first record in the file. If the parameter `tstop` is undefined, set it `aspsol.time` for the last record in the file.

9. Step the `time` from `tstart` to `tstop` by increments of `n_step` records in the ASPSOL file, a do the following processing:

(a) Calculate the polynomial interpolation in time of ang_y_sm and ang_z_sm for each fid light *i*, using the polint_order+1 data points nearest the current time using the Numerical Recipes routine polint. This routine returns an estimate of the uncertainty of the interpolation. Call these uncertainties ang_y_interp_err[i] and ang_z_interp_err[i] respectively.

(b) Convert the interpolated measurements of fid light y and z angles (which represent azimuth and elevation in the ACA frame) to unit-length direction vectors meas[i].d_aca:

```
meas[i].d_aca = vector(1, dtan(ang_y_interp), dtan(ang_z_interp));
meas[i].d_aca /= norm(meas[i].d_aca);
meas[i].d_aca_err = vector(1.0,
                           quad_add(ang_y_interp_err[i], ang_y_err[i]),
                           quad_add(ang_z_interp_err[i], ang_z_err[i]));
```

Here vector(x,y,z) is a 3-vector constructor, and quad_add(x,y) is a function which returns $\sqrt{x^2 + y^2}$.

(c) Do the following calculations and minimize the difference between measured and predicted fid light positions, as a function of theta_X, dy, and dz. After the first iteration, the minimization should always use the previous best-fit values. The minimization method should be the Levenberg-Marquardt method (mrqmin in Numerical Recipes), which returns a covariance matrix.

```
#define THETA_X 1
#define DY      2
#define DZ      3
#define N_FIT_PAR 3   /* number of fit parameters */

M  = acaN2aca * fc2acaN * fts_misalign; // Alignment matrix (global)

// Initialize mrqmin function arguments (once, at beginning)

float* x   = vector(1,3);  /* Num.Rec. vector initialization */
float* y   = vector(1,3);  /* Vectors and matrices should be freed */
float* sig = vector(1,3);  /* at the end */
int ndata;
float* a   = vector(1,3);
int*   ia  = ivector(1,3);
float** covar = matrix(1,3,1,3); /* Num.Rec. matrix initialzation */
float** alpha = matrix(1,3,1,3);
float  chisq;
float  alambda;

// Set up data for mrqmin (for each minimization)

i = 1;
for (i_fid = 0; i_fid < N_fid; i_fid++) {
  for (dir = 1; dir <= 2; dir++) {
    x[i] = (float) i;
    y[i] = meas[i_fid].d_aca[dir];
    sig[i] = meas[i_fid].d_aca_err[dir];
    i++;
  }
```

```
}
ndata = i-1;

if (first_minimization) {
  a[THETA_X] = theta_X_start;
  a[DY]      = dy_start;
  a[DZ]      = dz_start;

  ia[THETA_X] = 1;
  ia[DY]      = 1;
  ia[DZ]      = 1;

  ma = N_FIT_PAR;
  alambda = fit_alambda; // mrqmin fitting tolerance
}

// Do the minimization!
mrqmin (x, y, sig, ndata, a, ia, ma, covar, alpha, &chisq,
&CalcFidPos, alambda);

/*****************************************************************/
void CalcFidPos (float x, float a[], float*y, float dyda[], int na)
/*****************************************************************/
{
  static float f0 = -fc0_2aca_fp_len;
  static Matrix Rot_X = I(3,3); // Rotation matrix (init to identity)
  static Matrix dRot_X_dth = Zero(3,3); // Derivative of Rot_X (init to zero)

  if (na != N_FIT_PAR)
    {
      fprintf (stderr, "Error - unexpected number of parameters\n");
      exit 1;
    }

  i_fid = nint(x-1) / 2; /* Index into fid array */
  dir   = nint(x-1) % 2 + 1; /* Direction (1 = y, 2 = z) */

  /* set local vars for notational convenience */
  theta_X = a[THETA_X];
  dy      = a[DY];
  dz      = a[DZ];

  s_th = sin(theta_X);
  c_th = cos(theta_X);

  // Set rotation matrix and its derivative w/respect to theta_X
  Rot_X[1][1] = c_th;
  Rot_X[1][2] = s_th;
  Rot_X[2][1] = -s_th;
  Rot_X[2][2] = c_th;
```

```
          dRot_X_dth[1][1] = -s_th;
          dRot_X_dth[1][2] = c_th;
          dRot_X_dth[2][1] = -c_th;
          dRot_X_dth[2][2] = -s_th;

          stf0_fc = three_vec(0, dy, dz);     /* STF origin in FC frame */

          // First time for this fid, so do calculations for fid[i_fid].
          // If (dir == 2), then calculations were already done, so skip
          if (dir == 1) {
            fid[i_fid].p_fc = Rot_X(theta_X) * fid[i_fid].p_stf + stf0_fc;
            fid[i_fid].d_fc = fid[i_fid].p_fc;
            fid[i_fid].d_fc[0] -= fc0_2aca_fp_len;
            fid[i_fid].d_fc /= norm(fid[i_fid].d_fc);
            fid[i_fid].d_aca = M * fid[i_fid].d_fc;
            fid[i_fid].d_aca *= -1.0;   // Make sense of d_aca match rest

            f = fid[i_fid].d_aca[dir];

            // Calculate derivative of f w/respect to variational parameters
            df_dth = M * dRot_X_dth * fid[i_fid].p_stf / f0;
            df_ddy = three_vector(M[0][1], M[1][1], M[2][1]) / f0;
            df_ddz = three_vector(M[0][2], M[1][2], M[2][2]) / f0;
          }

          *y = f[dir];
          dyda[THETA_X] = df_dth[dir];
          dyda[DY]      = df_ddy[dir];
          dyda[DZ]      = df_ddz[dir];
        }
```

(d) Write dy, dz, and theta_X to the dy, dz, and dtheta columns of the ASPSOL data product.

(e) Write error estimates dy_err, dz_err and dtheta_err to the ASPSOL data product, using the minimization covariance matrix returned by mrqmin.

10. Rewind the ASPSOL data product. Calculate interpolated values of dy, dz, dtheta, dy_err, dz_err, and dtheta_err for the time stamps between tstart and tstop which were skipped over in step 9.

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: asp_solve

Spec Category: Aspect

Code Type: ASCDS

Code Source: New

Tool Number 1403

Description: Update databases **Inputs:**Aspect data products **Parameters:** **Outputs:**database updates **Source:** New **Release:** 4 **SDS ID:** j403 **SDS Name:** aspect_database_update **Processing:**

Update databases and AOSS catalog using outputs from aspect pipeline such as star and fid lights observed brightnesses, observed star RA and Dec, spoiler position vectors and brightnesses, fid lights displacements.

Tool Number 1404

Description: Correct for gyro bias drift rate

Parameters:

Not specified

Inputs:

GYRODATA - Gyro data product
PCAD1ENG - OBC estimates of gyro bias (AOGBIAS1,AOGBIAS2,AOGBIAS3)

Outputs:

GYRODATA - Aspect gyro L1 Data Product

Processing:

- Read GYRODATA values `gyrodata.time, gyrodata.sc_rate_raw`
- Find PCAD1ENG record closest in time to GYRODATA record, using current and ”next” record of PCAD1ENG as needed
- Apply gyro bias to raw spacecraft rate, write to GYRODATA element sc_rate_corr.

```
gyrodata.sc_rate_corr[0] = gyrodata.sc_rate_raw[0] - pcad1eng.aogbias1 ;
gyrodata.sc_rate_corr[1] = gyrodata.sc_rate_raw[1] - pcad1eng.aogbias2 ;
gyrodata.sc_rate_corr[2] = gyrodata.sc_rate_raw[2] - pcad1eng.aogbias3 ;
```

  NOTE: There may be unit conversions here, but I don’t know at this point.

- Write sc_rate_corr to GYRODATA product.

Source: GADS Prototype **Release:** 4 **SDS ID: SDS Name:** gyro_bias_correct

Tool Number 1138

gyro_process

Description:

Make gyro data tables

Parameters:

gyrodata	file		Name of GYRODATA file
gyrocal	file		Name of GYROCAL file
sig_ed_tim_scl	float		Sigma-edit smooth time scale
sig_ed_order	int		Sigma-edit smooth order
sig_ed_sigma	float		Sigma-edit reject sigma
sig_ed_iter	int		Sigma-edit iterations
gap_order	int		Polynomial order for gap filling
gap_max	int		Max points in gap
gap_min_contig	int		Min points on each side of gap
consist_dt	double	hidden	Data interval for consistency checking (sec)
consist_thr_warn	float	hidden	Threshold for consistency warning
consist_thr_err	float	hidden	Threshold for consistency error
debug	int		Debug output level (0-5)

Inputs:

GYRODATA - Aspect L1 gyro data product
GYROCAL - Aspect L1 gyro calibration data product

Outputs:

GYRODATA - Aspect L1 gyro data product

Processing:

1. Data status flag definitions

```
#define DATA_MISSING 0x01  /* data missing, probably in telemetry */
#define SIGMA_REJECT 0x02  /* rejected by sigma edit */
#define CONSIST_WARN 0x04  /* internal inconsistency; data acceptable */
#define CONSIST_ERR  0x08  /* internal inconsistency; fatal error */
#define INTERPOLATED 0x10  /* data value is interpolated from nearby points */
```

2. Read the the tool parameter file

3. Open GYROCAL and read the primary header and all gyro calibration data. Close GYROCAL.

4. Open GYRODATA for read-write, read the primary header of GYRODATA, and check values:
   (a) The number of active channels `n_gyro` can take the value 3 or 4. A value outside this range, or different from the value in the GYROCAL data product shall produce a fatal error.
   (b) Any disagreement between GYRODATA and GYROCAL in the values of `gyro_id` shall produce a fatal error.

5. Allocate data arrays

```
int    cts[gyrodata.n_gyro][gyrodata.n_rows];
double time[gyrodata.n_rows];
Byte   status[gyrodata.n_gyro][gyrodata.n_rows];
```

6. Read raw gyro data. The input GYRODATA product contains readouts from up to four (4) active gyro channels in units of accumulated counts. Each count represents an angular displacement about an axis perpendicular to the gyro spin axis. The typical scale factor is about 0.02 arcsec/sec.

```
while (GYRODATA_GetRow (&gyrodata)) {
  time[i]       = gyrodata.time;
  cts[0][i] = gyrodata.cts0_raw;
  cts[1][i] = gyrodata.cts1_raw;
  cts[2][i] = gyrodata.cts2_raw;
  if (gyrodata.n_gyro > 3) cts[3][i] = gyrodata.cts3_raw;
  i++;
}
```

7. Perform sigma-edit on each active gyro channel, using parameters defined in `gyro_process` parameter file. For each rejected data point, set bit 0 (LSB) of the appropriate status byte. *Should sigma_edit take a value with which to "or" a status byte array?*

8. Loop through each channel of gyro data and fill data gaps. The records in GYRODATA are spaced at regular intervals (enforced by `gyro_read_data`), but data may be bad as indicated by status flags: `DATA_MISSING` because of telemetry dropouts, or rejected by sigma-edit (`SIGMA_REJECT`). Fill these good data gaps using a polynomial interpolation routine like the following:

```
int poly_interp (int* y,                /* data to be filled */
                 double* x,     /* corresponding x coordinates */
                 Byte* status,  /* status flags */
                 int n_y,       /* size of y, x, status arrays */
                 int i_first,   /* index of first missing data point */
                 int* i_next,   /* after filling, index of next good point */
                 int min_contig, /* min contiguous good points each side of gap */
                 int max_gap,   /* maximum gap size */
                 int n_poly     /* polynomial order for interpolation */
                 )
{
  i0 = i_first - min_contig;
  if (i0 < 0) i0 = 0;
  j = 0;
  for (i = i0; i < i_first; i++) {
    y_good[j] = y[i];
    x_good[j] = x[i];
    j++;
  }

  n_contig = 0;
  while (n_contig < min_contig && i < n_y) {
    if (status[i] & (DATA_MISSING | SIGMA_REJECT)) {
      n_contig = 0;
    } else {
      y_good[j] = y[i];  /* y_good is double[] */
      x_good[j] = x[i];
      j++;
      n_contig++;
    }
    i++;
  }

  if (i - n_contig - i_first > max_gap)
    return 0;  /* FAILURE */

  i_next = i;
  n_good = j;

  /* Generic polynomial fitting routine: fits data with
   a polynomial of order n_poly, returns coefficients. */
  poly_fit (x_good, y_good, n_good, poly_coeff, n_poly);

  for (i = i_first; i < i_next - n_contig; i++) {
    if (status[i] & (DATA_MISSING | SIGMA_REJECT)) {
      y[i] = nint (poly_val (x[i], poly_coeff, n_poly));
      status[i] |= GAP_FILLED;
    }
  }

  return 1;
}
```

9. Write tool parameters to GYRODATA header

10. If any data records were modified, rewind the file and write over the modified records

11. Convert counts to raw S/C angular rate

    - Determine the differential gyro counts. This is simply the difference between the gyro count value for the current record from that of the previous record.
    - Multiply each valid differential gyro count by its scale factor, to convert from gyro counts to angular units. The scale factor depends on the particular gyro channel, but the work of extracting the correct scale factors for the present configuration is done in `asp_get_cal`. The scale factor is different for positive and negative rates, and can have a linear dependence on rate. At launch, the linear coefficient will be set to zero.

- Use the output from previous step and gyro to S/C transformation matrix, and the gyro scale-factor-misalignment matrix from GYROCAL to calculate uncorrected spacecraft angular (deg/s) rate.
- Write to output GYRODATA table.

```
sf_ma_gyro2sc = gyrocal.sf_ma * gyrocal.gyro2sc; /* matrix multiply */

for (i = 1; i < gyrodata.n_rows; i++) {
  for (chan = 0; chan < gyrodata.n_gyro; chan++) {
    delta_cts = cts[chan][i] - cts[chan][i-1];
    scale_fac = (delta_cts > 0) ?
      gyrocal.scale_fac_pos[chan][0]
      + gyrocal.scale_fac_pos[chan][1] * delta_cts
      :
      gyrocal.scale_fac_neg[chan][0]
      + gyrocal.scale_fac_neg[chan][1] * delta_cts;
    omega_gyro[chan] = scale_fac * delta_cts / delta_t;
       /* delta_t is gyro sample period, should be 0.25625 sec */
  }

  gyrodata.sc_rate_raw = sf_ma_gyro2sc * omega_gyro;  /* matrix * vector */
  /*  ** Convert to degrees ** */
  /* RWS flags set to write ONLY sc_rate_raw column */
  GYRODATA_PutRow (&gyrodata);
  if (i == 1)
    GYRODATA_PutRow (&gyrodata);  /* set 0th rate to 1st rate */

}
```

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: gyro_process

Spec Category: Aspect

Code Type: ASCDS

Code Source: GADS Prototype

Tool Number 1251

gyro_read_data

Description: Read gyro telemetry to create raw gyro data file.

Parameters:

gyrodata	file		Output L1 GYRODATA data product file
pcad_eng1	file		Input L0 PCAD engineering file stack (sample rate = 1)
pcad_eng32	file		Input L0 PCAD engineering file stack (sample rate = 32)
pcad_eng128	file		Input L0 PCAD engineering file stack (sample rate = 128)
tstart	double	hidden	Start of time filter (sec)
tstop	double	hidden	End of time filter (sec)
gyro_dt	double	hidden	Expected sample period for gyro data records (sec)
gyro_dt_unc	double	hidden	Uncertainty in gyro sample period (sec)
check_pcad32	boolean	hidden	Check flags in PCAD32 file?

Inputs:

PCADENG1 - L0 PCAD engineering data (sample rate = 1 per major frame)
PCADENG32 - L0 PCAD engineering data (sample rate = 32 per major frame)
PCADENG128 - L0 PCAD engineering data (sample rate = 128 per major frame)

Outputs:

GYRODATA - Aspect L1 Gyro Data Product

Processing:

1. Overview of input data –

   All of the telemetry items relevant to gyros are listed in Table 1, sorted alphabetically and by sampling rate. The sample rate refers to the rate per major frame (32.8 s), with 128 minor frames (0.25625 s) per major frame. For SAMPLE_RATE = 1, we are interested in all of the data: the IRU ON/OFF flags, RANGE STATUS flags, and temperatures (*TEMP). The data for SAMPLE_RATE = 4 and SAMPLE_RATE = 16 are not needed. For SAMPLE_RATE = 32, there are three flags supplied by OBC flight software. These are optionally monitored, as described below. Finally, the SAMPLE_RATE = 128 data contain the accumulated gyro count telemetry, which correspond to the CTS*RAW columns in the GYRODATA file.

2. Data status flag definitions

```
#define DATA_MISSING 0x01  /* data missing, probably in telemetry */
#define SIGMA_REJECT 0x02  /* rejected by sigma edit */
#define CONSIST_WARN 0x04  /* internal inconsistency; data acceptable */
#define CONSIST_ERR  0x08  /* internal inconsistency; fatal error */
#define INTERPOLATED 0x10  /* data value is interpolated from nearby points */
```

Table 1: IRU/GYRO telemetry in Level-0 PCAD engineering files

MSID	TECHNICAL_NAME	LENGTH	SAMPLE_RATE
AIRU1	IRU-1 ON/OFF	1	1
AIRU1BT	IRU-1 BASE TEMP	8	1
AIRU1G1T	IRU-1 GYRO #1 TEMP	8	1
AIRU1G2T	IRU-1 GYRO #2 TEMP	8	1
AIRU1R1X	IRU-1 RANGE STATUS 1X	1	1
AIRU1R1Y	IRU-1 RANGE STATUS 1Y	1	1
AIRU1R2X	IRU-1 RANGE STATUS 2X	1	1
AIRU1R2Y	IRU-1 RANGE STATUS 2Y	1	1
AIRU1VFT	IRU-1 VFC TEMP	8	1
AIRU2	IRU-2 ON/OFF	1	1
AIRU2BT	IRU-2 BASE TEMP	8	1
AIRU2G1T	IRU-2 GYRO #1 TEMP	8	1
AIRU2G2T	IRU-2 GYRO #2 TEMP	8	1
AIRU2R1X	IRU-2 RANGE STATUS 1X	1	1
AIRU2R1Y	IRU-2 RANGE STATUS 1Y	1	1
AIRU2R2X	IRU-2 RANGE STATUS 2X	1	1
AIRU2R2Y	IRU-2 RANGE STATUS 2Y	1	1
AIRU2VFT	IRU-2 VFC TEMP	8	1
AOIRURT1	F_CHANNEL_RATE(1)	1	4
AOIRURT2	F_CHANNEL_RATE(2)	1	4
AOIRURT3	F_CHANNEL_RATE(3)	1	4
AOIRURT4	F_CHANNEL_RATE(4)	1	4
AIRU1G1I	IRU-1 GYRO #1 CURRENT	8	16
AIRU1G2I	IRU-1 GYRO #2 CURRENT	8	16
AIRU2G1I	IRU-2 GYRO #1 CURRENT	8	16
AIRU2G2I	IRU-2 GYRO #2 CURRENT	8	16
AOIRUHLD	F_GYRO_HOLD_EXCEEDED	1	32
AOIRUMON	F_IRU_MON_FAULT	1	32
AOGYRSEL	F_GYRO_CONFIG (AOGYRSEL)	1	32
AOGYRCT1	SENSOR_DATA.GYRO_COUNT(1)	16	128
AOGYRCT2	SENSOR_DATA.GYRO_COUNT(2)	16	128
AOGYRCT3	SENSOR_DATA.GYRO_COUNT(3)	16	128
AOGYRCT4	SENSOR_DATA.GYRO_COUNT(4)	16	128

3. Compute IRU temperature statistics and check PCADENG1 status flags over interval

   (a) Read PCAD engineering data records from each of the files specified in the `pcad_eng1` tool parameter.

   (b) If the time filter parameters are non-zero, ignore input data which are outside the specified limits.

   (c) For each of the following items, compute the MIN, MAX, AVG, RMS. Write them as GYRODATA header keywords.

      - AIRU*BT - IRU Baseplate temperatures (DegC)
      - AIRU*VFT - IRU VFC temperatures (DegC)
      - AIRU*G*T - IRU Gyro temperatures (DegC)

   (d) Verify that the following items remain constant during the processing time interval. Report an error if the value changes, otherwise write the value to a GYRODATA header keyword.

      - AIRU* - IRU Power
      - AIRU*R*X - IRU X-axis Range Selection flags
      - AIRU*R*Y - IRU Y-axis Range Selection flags

4. Optionally check PCADENG32 status flags over time interval (*I'm not sure if we need / should do this, so I'm making it optional.*)

   (a) If tool parameter `check_pcad32 != true` then skip this processing step

   (b) Read PCAD engineering data records from each of the files specified in the `pcad_eng32` tool parameter.

   (c) If the time filter parameters are non-zero, ignore input data which are outside the specified limits.

   (d) Verify that the AOGYRSEL flag remains constant during the processing time interval. Report an error if the value changes, otherwise write the value to a GYRODATA header keyword.

   (e) Verify that the following items have the value FALSE (TBR) during the processing time interval. Report a warning if the value is TRUE (TBR). Write the logical "or" of these flags over the selected time interval to GYRODATA header keywords (AOIRUHLD and AOIRUMON).

      - AOIRUHLD – F_GYRO_HOLD_EXCEEDED (OBC flag)
      - AOIRUMON – F_IRU_MON_FAULT (OBC flag)

5. Process gyro counts data in the PCADENG128 files and generate GYRODATA file. This processing can be done one record at a time.

   (a) Read next gyro data record (AOGYRCT*) from the stack of files specified in the `pcad_eng128` tool parameter. If the time filter parameters are non-zero, process only the records within these limits.

   (b) Check record interval against the input sample period parameter. If the interval between records is less than the expected period minus the uncertainty, report a fatal error:

   ```
   pcadeng128.time - pcadeng128_time_last < gyro_dt - gyro_dt_unc
   ```

   A gap is indicated if the time interval between records is greater than the expected sample period plus the period uncertainty:

```
pcadeng128.time - pcadeng128_time_last > gyro_dt + gyro_dt_unc
```

If a gap is detected, write empty records to the output GYRODATA file for the missing times. The gyro counts (`gyrodata.cts*_raw` and `gyrodata.cts*_corr` should be set to the last valid values (*TBR, perhaps set to zero*). Set the `MISSING_DATA` bit of the status byte (all four channels) for each empty record:

```
gyrodata.status* |= MISSING_DATA;
```

The time (`gyrodata.time`) should be incremented by `gyro_dt` for each missing record. Verify that the time difference between the final missing record and the next valid record is `gyro_dt +/- gyro_dt_unc`. If not, report a fatal error.

(c) Set `gyrodata.cts*_raw` to the corresponding value (AOGYRCT*) from the PCADENG128 file.

(d) For the first `gyrodata` record, do

```
gyrodata.cts*_corr = pcadeng128.AOGYRCT*
pcadeng128_AOGYRCT*_last = pcadeng128.AOGYRCT*
```

(e) For subsequent records, set `gyrodata.cts*_corr` values, checking for and correcting gyro count rollover.

Raw gyro counts are accumulated on-board and telemetered as signed short integers. It will regularly occur that the value rolls over, from 32767 to -32768, and vice versa. The counters may roll over more than once within a file and each channel is independent of the others.

```
/* subtact current counts from last valid counts (possibly before gap) */
/* make sure arithmetic is done with ints, not short ints */
d_cts = (int) pcadeng128.AOGYRCT1 - (int) pcadeng128_AOGYRCT1_last;
if      (d_cts > 32678)  gyrodata.cts1_corr += d_cts - 0x10000;
else if (d_cts < -32768) gyrodata.cts1_corr += d_cts + 0x10000;
else                     gyrodata.cts1_corr += d_cts;
pcadeng128_AOGYRCT1_last = pcadeng128.AOGYRCT1;
```

(f) Write record to the output GYRODATA file.

- `time` - Time at end of gyro readout (s)
- `cts*_raw` - Telemetered raw gyro counts (cts)
- `cts*_corr` - Corrected gyro counts with offsets applied (cts)
- `status*` - Gyro record status byte

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: gyro_read_data

Spec Category: Aspect

Code Type: ASCDS

Code Source: New

Tool Number 1241

Description: Make the aspect stable intervals data product. **Inputs:**Aspect solution **Parameters:**Tolerances **Outputs:**Aspect intervals **Source:** New **Release:** 4 **SDS ID:** j241 **SDS Name:** aspect_stable_intervals **Processing:**

Calculate the aspect stable intervals given the aspect solution. First, create a header containing the info needed to make the sky grid for this instrument (i.e. instrument sky pixel range, pixel size, and WCS for the nominal position). The instrument ID should be in the standard header for the aspect solution, as will be the nominal RA and Dec.

Then, go through each record of the aspect solution accumulating the mean and variance of DX, DY, ROLL offsets. Based on (TBD) criteria for variance component, assign an aspect quality flag and write the record to the aspect quality file: the start time, stop time, mean and rms aspect and quality flag. Bin the entries in the table based on quality flags into a series of stable intervals with a given variance tolerance.

Tool Number XXXX

aca_proc_dark

Description: Produce Dark Current Image for the ACA CCD

Parameters:

TELCAL file	file		Name of L0 Calibration Data Product
TELENG file	file		Name of L0 Engineering Data Product
ACADRK file	file		Name of Dark Current Image file
ACAEXP file	file		Name of Pixel Exposure Map
temp_scale_method	string		(none,average,reference)
temp_reference	float	deg C	Reference temperature for scaling
max_temp_drift	float	deg C	Max. allowed temperature drift during cal run
max_aspect_wobble	float	arcsec	Max. allowed change in OBC attitude
min_asp_dist	float	arcsec	Min. distance between cal iterations
detect_cosmic	bool		Flag to apply cosmic ray detection
cosmic_outfile	file		Name of file listing cosmic ray contaminated pixels
calblocks_root	string		Root name of files for Calibration data blocks
cosmic_infile	file		File of known contaminated pixels
max_read_fraction	float	sec	Max. ratio of readout to integration times
ramp_method	string		(solve,linefit)
avg_radius	float	e-/s	Radius for averaging dark current
hot_pix_limit	float	e-/s/pix	Limit to identify hot pixels
hot_pix_merge	bool		Flag to merge hot pixel lists
hot_pix_out	file		Name of output hot pixel list

Inputs:

Level 0 calibration data product (TELCAL)
Level 0 engineering data product (TELENG)
  OBC spacecraft attitude (AOATTQT)
  CCD Temperatures
Calibration database items
  gain[node]
  dark_scale_exponent
  hot_pixel_list

Outputs:

ACA Dark Current Image (ACADRK)
ACA CCD Pixel Exposure map (ACAEXP)
Optional - Hot Pixel list
Optional - Corrupted pixel locations
Optional - Calibration block images

Processing:

1. Create calibration pixel-exposure map from ACA L0 calibration data record (TELCAL).

```
Generate an array of dimension [1024 x 1024] indicating how many times each pixel is
represented in the Calibration Data records.  Since the integration time is constant
during a calibration run, this is equivalent to the exposure time for each pixel.

byte pix_exp[1024][1024] = 0;
For each ACA calibration data record
  increment pix_exp[telcal.row][columns associated with telcal.col_neg]
```

2. Assemble ACA calibration blocks from ACA L0 calibration data (TELCAL). This includes subtracting bias offset, scaling by quadrant-based CCD gain, and normalizing values by integration time.

```
class CALBLOCK {
  float  time;                  // average time of row-readouts
  float  integ;                 // integration time
  double aca_time0;             // aca_time at first row of block
  double aca_time1;             // aca_time at last row of block
  int    row0;                  // first CCD row of block
  int    row1;                  // last CCD row of block
  Byte   col_neg;               // columns negative?
  Byte   replica;               // replica number
  Matr<float> dark_curr;        // raw dark current
  int    n_row                  // current number of rows (= row1-row0+1)
  float  ccd_temp;              // average CCD temperature
  float  dalpha;                // average delta alpha
  float  dalpha_pp;             // peak-to-peak delta alpha
  float  ddelta;                // average delta delta
  float  ddelta_pp;             // peak-to-peak delta delta
}

read through TELCAL records
  if (first record of this block)
    initialize all calblock[blocknum] values (including aca_time1, row1)
    next TELCAL record

  check conditions defining a new block
  if (abs(telcal.aca_time - calblock[blocknum].aca_time) < ACA_TIME_THRESHOLD
      && telcal.col_neg == calblock[blocknum].col_neg
      && telcal.replica == calblock[blocknum].replica
      && telcal.row + 1 == calblock[blocknum].row1)

   NOTE: The above assumes that a block is in the TELCAL file as a contiguous
        unit.  (Which is how it is defined.)  If each node clocks out a row at
        the same time and they are telemetered interleaved, this check won't work.

    update aca_time1, row1, n_row, and dark_curr[n_row][*]
    NOTE: The dark current calculation involves subtracting the offset
        (telcal.adc_bias) scaling by the appropriate gain[quadrant] from
        Cal. Database dividing by integration time (telcal.integ)
   }
   else
   {
```

```
   new block
   increment block counter
   set first record indicator
  }
 }
```

3. Apply temperature scaling to each calblock dark current matrix, if necessary. The CCD temperatures are provided in the ACA L0 Engineering data (TELENG).

   ```
   The tool parameter file should contain
      temp_scale_method = (none|average|reference) / method for scaling data
      temp_reference    = 15.0                     / reference temp. for scaling
      max_temp_drift    = 1.0                      / max allowed temp drift during cal
   ```

   If the scaling method is `none`, then no temperature scaling is done. I it is `average`, then the data are scaled to the average temperature. If it is `reference`, then data are scaled to the given reference temperature. In the latter two cases, the peak-to peak temperature change during the calibration period is calculated, and if it exceed `max_temp_drift`, then the operator is notified but processing continues

   If `temp_scale_method != none`, do the following:

   (a) For each calblock, set `ccd_temp` to the CCD temperature from TELENG nearest in time to `time - integ/2.0`

   (b) Calculate average, min, and max temperature of all calblocks.
   Notify operator if `max - min > max_temp_drift` (tool parameter).

   (c) For each calblock, multiply the dark current values by $(T_s/T)^C$, where T_s is the scale temperature (either reference or average), T is the calblock temperature, and C is the dark current scaling exponent. The scaling exponent should come from the calibration database.

   (d) Store values as header keywords in final dark current map.

      i. REFTEMP = Reference temperature (average or implicit)
      ii. MINTEMP = Min. temperature during calibration
      iii. MAXTEMP = Max. temperature during calibration
      iv. SCALEXP = Dark current scaling exponent
      v. SCALMETH = Temperature scale method

4. Check spacecraft aspect during calibration.
   OBC estimates of spacecraft attitude are provided in the L0 Engineering data (TELENG). This quaternion is telemetered once each 1.025 sec as the mnemonic `AOATTQT`

   (a) Using the first quaternion as the reference attitude, convert the remaining quaternions to $\Delta\alpha$ and $\Delta\delta$.

   (b) For each calblock, calculate the average and peak-to-peak of $\Delta\alpha$ and $\Delta\delta$ over the time spanned by that block. If either peak-to-peak value is greater than the parameter `max_aspect_wobble` (60 arcsec, TBR) then notify operator

   (c) For each calblock, find all other calblocks with overlapping CCD readout regions and with *different* `replica` number. Verify that the aspect distance to all other calblocks is greater than `min_asp_dist` (parameter). If otherwise, notify operator. The aspect distance is the angular distance (arcsec) between the $(\Delta\alpha, \Delta\delta)$ of two calblocks.

   (d) calculate the average and peak-to-peak of $\Delta\alpha$ and $\Delta\delta$ over the full calibration time period. Store these values as header keywords in the final dark current map.

      i. AVGALPHA, MINALPHA, MAXALPHA = Avg,min,max $\Delta\alpha$ during calibration

      ii. AVGDELTA, MINDELTA, MAXDELTA = Avg,min,max $\Delta\delta$ during calibration

5. Detect and remove cosmic rays from each calblock

   (a) If the `det_cosmic` boolean parameter is true, then use a TBS routine to detect cosmic rays in each calblock. This routine will likely use the full data set and some variant of median filtering. Replace each pixel value that is corrupted by a cosmic ray with IEEE NaN and subtract 1 from the pixel exposure map. If the `cosmic_outfile` parameter is non-null, then write corrupted pixel locations to that file (along with necessary information to specify in which calblock are the pixels).

   (b) If the `cosmic_infile` parameter is non-null, read the set of pixel locations (and calblocks) which are corrupted. Replace each pixel with IEE NaN and subtract 1 from the pixel exposure map.

6. Remove dark current ramp due to non-negligible CCD readout time.
   If the fractional readout time `(aca_time1 - aca_time0)/integ` is greater than `max_read_fraction` then remove signal accumulated during readout:

   (a) If `ramp_method` is `solve` then determine the solution by TBD method to the simultaneous set of row equations

   $$T_{int} \cdot S_{ij}^{obs} = T_{int} \cdot S_{ij} + \sum_{k=0}^{i-1} T_{row} \cdot S_{kj}$$

   where S_{ij}^{obs} are the observed calblock dark current values, S_{ij} are the estimated true values, T_{int} is the integration time, and T_{row} is the readout time for a single row. Note that the row index i is relative to the calblock start row (i.e. the first row read out by the ACA). T_{row} is calculated as `(aca_time1 - aca_time0)/(row1 - row0)`.

   (b) If `ramp_method` is `linefit` then subtract a ramp-surface fit in the row direction from the block:

      i. Calculate $T_i = \sum_j S_{ij}$

      ii. Fit a line to the points (as a function of i) so $T_i = M \cdot i + B$

      iii. Adjust each dark current pixel by $S_{ij} = S_{ij} - M \cdot i$

   (c) Store ramp method as RAMPMETH keyword in final dark current map

7. Create optional output files for each calblock.
   If the `calblocks_root` parameter is non-null, then write a FITS imag file for each calblock, with the name `<root>_<index>.fits`, where the index is the index value into the calblock array. Typical calibration runs should contain $\tilde{3}2$ rows/block, requiring (16 blocks * 4 quadrants * 4 replicas = 256 files)

8. Calculate median-filtered dark current map:

   (a) For each CCD pixel, find all calblocks which contain that pixel Assemble pixels and calculate their *median* value.

   (b) Calculate the average of all valid pixels within `avg_radius` of the median and store in the dark current map. (A valid pixel has not been marked with NaN)

(c) Write the median-filtered dark current map as a FITS image. Include header keywords listed in other sections as well as the following:

   i. INTEG = integration time

   ii. AVGBLKSZ, MINBLKSZ, MAXBLKSZ = calibration block size (n_row)

   iii. AVGUTC, MINUTC, MAXUTC = UTC time

(d) Write the pixel-exposure map as a FITS image, with the same header keywords as for the dark current FITS image.

9. Verify no apparent discontinuities between mean dark current levels in each pixel block. If such discontinuities are present, calculate individual scale / slope factors for each block to minimize discontinuities without changing mean dark current of CCD. Details TBS.

10. Update CCD hot pixel list:

    (a) Identify as hot pixels any pixel with a dark current greater than `hot_pix_lim` (nominal 1000 e^{-}/s/pixel)

    (b) Read the existing hot pixel list for the CCD

    (c) Notify operator of previously identified hot pixels that are no longer hot.

    (d) If `hot_pix_merge` is true, then merge the previous hot pixel list with the new pixels.

    (e) Write the new hot pixel list to the file `hot_pix_out` in format compatible with the star selection algorithm requirements

Release: 4

Group: DA

Analysis Domain: Aspect

DS Tool Class: 3

DS Tool Category: Aspect

Spec Name: aca_proc_dark

Spec Category: Calibration

Code Type: ASCDS

Code Source: New