NCCS Single-Dish Continuum Observing Algorithms

M J Gaylard + J L Jonas 1999/07/20

The three types of continuum observing are defined to be scanning, stepping and making maps from sets of scans. Observing programs that were in use previously are summarised. The algorithms for carrying out each type of observation are defined. The common subroutines needed are defined.

Contents

1 Introduction

The programs on the HP1000 making use of continuum radiometry were written by many authors. They diverged over the years to produce groups of very similar programs, each optimised slightly differently by changes to the code.

The aim here is to converge on a miminum set of generalised algorithms for scanning, stepping and making small maps. These algorithms are optimised by means of input parameters that define their mode of operation.

The types of continuum observing modes meant by scanning, stepping and mapping are first defined.

Note that there is an ambiguity in terminology, in that for the NCCS, "scan" is used as a synonym for any type of observation, including spectroscopy and pulsar timing, but "scan" also means driving the telescope in a particular way, e.g. drift scan, declination scan etc., so that a "scan" (observation) can actually be made up of a number of "scans" (drives).

For ease of reference the programs that were in use on the HP1000 are briefly described and links provided to their code.

General algorithms for each type of observing are defined.

2 Types of scans

2.1 Scan types and antenna motion

Scanning, if completely generalised, is defined by specified rates of motion in two orthogonal axes which may have no particular relation to the natural axes of the telescope, i.e. Equatorial Hour Angle and Declination.

However it may be useful to categorise scans in order of increasing complexity in relation to the natural axes of the telescope, as subtleties of algorithm design may be involved.

1. passive (drift) scan, telescope stationary
   (a) park telescope at an earlier RA than the target, Earth rotation produces the scan
2. active scan at fixed rates in the natural telescope coordinates (HA, Dec)
   (a) scan in HA at siderial rate (track RA), park in Dec
   (b) scan in HA at non-siderial rate, park in Dec
   (c) scan in HA at siderial rate (track RA), scan in Dec
   (d) park in HA, scan in Dec
   (e) scan in HA at non-siderial rate, scan in Dec
3. active scan at fixed rates in coordinate systems rotated from the natural telescope coordinates, producing variable speeds in the latter
   (a) Galactic
   (b) Ecliptic
   (c) Altazimuth
   (d) other Longitude + Latitude

The outputs of the instruments attached to the receiver are recorded during the scan, and the data are internally calibrated at some point in the scan or set of scans by firing a noise diode and recording the change in signal level.

All continuum radiometry comprises the recording of instrument outputs during one or more of the above scans.

It is of interest to see what sort of scans have been predefined for the GBT:

`AdvancedGBTprocedurewritinginGlish`, at `http://www.gb.nrao.edu/~rfisher/Glish/gbt\_procedures.html\#built\_ins`

2.2 SCANTYPES implying multiple scans

Two standard SCANTYPE definitions produce multiple scans that are treated as an uninterruptible observation, normally with a single calibration:

- STEP is defined to consist of a series of (track RA, park in Dec) scans, at the step positions defined in the next section.
- SCANPNT is defined to be two orthogonal scans centred on the specified coordinates, first being (scan in HA, park in Dec), the second (track RA, scan in DEC).

2.3 Step positions

Stepping consists of tracking in RA at positions stepped around a beam pattern at some subset of the following positions:

- second null in RA (SNE, SNW) or Dec (SNN, SNS)
- first null in RA (FNE, FNW) or Dec (FNN, FNS)
- half power points in RA (HPE, HPW) or Dec (HPN, HPS)
- beam centre (ON)
- zero-crossing point of dual beam receivers (ZC)

Suffixes A or B denote the beam in use for dual beam systems, e.g. SNEA. Suffix CAL denotes that the calibration noise diode is fired during the track, e.g. SNEACAL, producing a double length track with the noise diode on during the first half.

Stepping therefore comprises a set of a particular type of active scans carried out in a specific sequence as one, uninterruptible unit.

2.4 Mapping options

Maps are made from repeated scans. Large maps are made from scans treated as single entities that are calibrated and recorded individually.

However there remains a need for small maps comprising a set of scans treated as a single entity, calibrated once, and observed without interruption. two specialized examples are Sun and Moon maps. See "NCCS Requirements and Specifications", section 6.1.5. SCANTYPE= STEP and SCANPNT are two undersampled examples already described. Fully sampled examples could be defined to include the SCANTYPEs:

- SMALLMAP, a map of a small area such as the inner sidelobes of the beam, centred on a strong, compact radio continuum source.
- SUNMAP, a map of a small area centred on the Sun
- MOONMAP, a map of a small area centred on the Moon

There are two ways that uninterruptible small maps can be handled:

1. By generating in advance an input file comprising all the scans required, as single entities, and commanding the scheduler to treat the set as uninterruptible;

2. By defining in an input file the minimum set of parameters needed to execute the set of scans and passing these to the observing program to generate internally the required scans.

The first option is identical to making a large map, each individual scan being treated separately, except that all the scans are done without interruption.

The second option is identical to the method used to do STEP and SCANPNT observations, and is how it was done in all pre-NCCS mapping.

Clearly, the relative advantages of the two methods need careful assessment.

3 Continuum Observing Programs on the HP1000 and PC

3.1 Test programs

CALT

Essentially a cutdown version of a continuum observing program. Interactively configures and calibrates receivers by firing a noise diode and sampling the output of the selected receiver. Written by MG. Source code , in `nccs/doc/continuum/hpcontinuum/test/calt.ftn`.

3.2 Scanning programs

The first drift scan program was written by Don Cameron (sixcm).

DSCAN

Interactive multiwavelength drift scans, written by MG. Used for tests and as a training program for students. Source code , in `nccs/doc/continuum/hpcontinuum/scan/dscan.all`.

DSCN9

Automated non-interactive drift scans, derived from DSCAN, written by MG. Source code , in `nccs/doc/continuum/hpcontinuum/scan/dscn9.ftn`.

OBCIR

Multiwavelength drift scans through Circinus X-1, written by GN. Source code , in `nccs/doc/continuum/hpcontinuum/scan/obcir.ftn`.

SIXCM

Obsolete 6cm drift scan program, written by DC, GN. Source code , in `nccs/doc/continuum/hpcontinuum/scan/sixcm.ftn`.

THRCM

Obsolete 13cm drift scan program, written by DC, GN. Source code , in `nccs/doc/continuum/hpcontinuum/scan/thrcm.ftn`.

3.3 Stepping programs

The stepping programs all derive from a single program originally written by Don Bramwell.

OBCAL

Runs under SCHDL. Steps around a single beam at all frequencies. Step sequence as per TNSTP. Source code , in `nccs/doc/continuum/hpcontinuum/step/obcal.ftn`.

OBPLN

Runs under SCHDL. Observe any planet. Step sequence as per TNFPC. Source code , in `nccs/doc/continuum/hpcontinuum/step/obpln.ftn`. Routines for reading MICA planetary predictions , in `nccs/doc/continuum/hpcontinuum/step/plnet.sub`.

OBQCK

Runs under SCHDL(?) Observe any planet in highspeed "interlaced mode" at both 3.5cm and 6cm. Source code , in `nccs/doc/continuum/hpcontinuum/step/obqck.ftn`. Written by JQ. Tracks at (?):

```
ZCCAL, ON.
```

TNFPC

Source code , in `nccs/doc/continuum/hpcontinuum/step/tnfpc.ftn`. For dual beams full pointing is done only in declination. Zero-crossing point provides east-west pointing if no inflection between the beams. Tracks at:

```
FNNA, HPNA, ONA, HPSA, FNSA, ZC, FNNB, HPNB, ONB, HPSB, FNSB
```

For single beams there is no east-west pointing check. Tracks at:

```
FNN, HPN, ON, HPS, FNS
```

TNFQS

"Quick Step". Source code , in `nccs/doc/continuum/hpcontinuum/step/tnfqs.ftn`. For both dual and single beams, does no pointing. Tracks at:

```
FNNACAL, ONA, FNSA
```

TNSTP

Source code , in `nccs/doc/continuum/hpcontinuum/step/tnstp.ftn`. For dual beams, does pointing in both Dec and HA using HP points, with zero-crossing as last point, but otherwise does not use the second beam. Tracks at:

```
FNNACAL, HPNA, ONA, HPSA, FNSA, FNEA, HPEA, ONA, HPWA, FNWA, ZC
```

For single beams, does pointing in both Dec and HA using HP points. tracks at:

```
FNNCAL, HPN, ON, HPS, FNS, FNE, HPE, ON, HPW, FNW
```

3.4 Mapping Programs

The mapping programs on the HP1000 all derive from SKYMAP, developed over the aeons by Rhodes University (Pete Mountfort, Justin Jonas et al.) for mapping at 13cm wavelength.

SKYMAP

Program for mapping in equatorial coordinates at 13cm, using calls to HP-STEER. Source code , in `nccs/doc/continuum/hpcontinuum/map/skymap.ftn`.

BEAM6

Program for multiwavelength small area mapping in equatorial coordinates, using calls to HP-STEER, derived from SKYMAP by MG. Source code , in `nccs/doc/continuum/hpcontinuum/map/beam6.ftn`.

SUN7

Map the Sun in Ecliptic coordinates using calls to Pascal PC-STEER. Written by JQ? Source code , in `nccs/doc/continuum/hpcontinuum/map/sun7.ftn`. Superceded by MAPR3.

MOON

Map the Moon in apparent Equatorial coordinates using calls to Pascal PC-STEER. Written by JQ? Source code , in `nccs/doc/continuum/hpcontinuum/map/moon.ftn`. Superceded by MAPR3.

MAPR3

Map the Sun in Ecliptic coordinates, the Moon in apparent Equatorial coordinates, or other general object in Galactic or 1950, 2000 or mean (current date?) Equatorial coordinates, using calls to Pascal PC-STEER. Written by JQ. Source code , in `nccs/doc/continuum/hpcontinuum/map/mapr3.ftn`. The parameter file used by MAPR3 was called , in `nccs/doc/continuum/hpcontinuum/map/mapprm`.

The most recent mapping program is a basket-weaving program written by Sarah Buchner under Justin Jonas' tutelage. It is written in Pascal and is described in her MSc thesis (available in the library). Source code should be on NCCS1 or NCCS2 ?

4 Observing Algorithms

4.1 Generic single-dish observing algorithm

All single-dish observing algorithms follow this form:

```
get scan parameters from parser/scheduler
set up parameters for the scan(s)
check object visibility
configure the telescope, receiver and instrument
quickly test the receiver and instrument with the calibration noise diode
repeat {
        drive to start of scan
        calibrate if required
        scan and capture data until end of scan
        bin/transform/process the data if required
        scale the data if required
        plot the data on screen for visual data qualification
        }
until ( all scans done )
write captured data in required format to disk file
```

The devil, of course, lies in the details.

4.2 Receiver calibration (CALT) test program algorithm

Algorithm for a CALT replacement, using a graphical user interface. Two calibration routines are used, depending whether the receiver can be configured in absolute total power mode to give the system temperature, or only to give the receiver counts per Kelvin from firing the noise diode.

The system temperature calibration routine called by CALT has been written out as pseudocode (see later).

```
repeat
{
        repeat
        {
        // set up user interface and get configuration parameters:
        // (assumed to be a Tk/TCL front end)

        list receiver systems available -
                user selects receiver
        (or specify a frequency based on a list of standard frequencies)

        list instruments (backends) available -
                user selects instrument to connect to receiver

        list noise diode calibrations in Kelvins from SYPARM for selected
        instrument outputs -
                user can update for these tests if incorrect

        list number of samples to take (give default) -
                user can select number of samples

        list number of tests to do (give default = 1) -
                user can select number of tests
        }
        until ( full configuration defined and user selects "GO" )

        repeat
        {
        ask scheduler for control of receiver and instrument

        read from STEER:
                subreflector tilt and focus,
                UT, ST, telescope HA, RA, Dec
        configure the receiver and instrument
        connect counters to instrument outputs
        setup local oscillators (DROs, frequency synthesizers)

        if ( mode = total_power )
        then { do total_power_calibration }
        else { do relative_gain_calibration }

        // display the results on the user interface:
        display UT, ST, telescope HA, RA, Dec, subref_tilt, subref_focus
        for ( ic = 0 ; ic < Ncounters ; ic++ ) do
                {
                display "output name=", output_name[ic]
                display "nominal Tsys", Tsys_nominal
                if ( COerror = 1 ) then { display "Counter error" }
                if ( NDerror = 1 ) then { display "Noise diode not seen" }
                if ( LOerror = 1 ) then { display "no change with local osc. off" }
```

```
                if ( CperK > 0 ) then { display "Counts per Kelvin", CperK, CperK_err }
                if ( Tsys > 0 ) then { display "measured Tsys", Tsys +- Tsys_err }
                }

        write the displayed results to the log file
        }
        until ( the selected number of tests has been done )
}
until ( user selects "QUIT")
```

4.3 Drift scan algorithm

This is a first attempt at expanding the generic algorithm, for a drift scan.

```
get scan parameters from parser/scheduler
        (object, coordinates, scan_type, scan_start, scan_length,
        scan_rate, receiver, instrument, res_freq, bin, observing
        constraints)

set up scan parameters
        minscanlen = bwfn / cos(dec) + 0.1 degrees
        set scan_length = max (min_scan_len, scan_len_input)

        sampling interval = averaging time of
        samples coming in at 10 per second
        at say 10 samples / hpbw

        start_RA = POSRA (RA_of_date - scanlen/2)
        drive_RA = POSRA (start_RA - calibration_time * time_to_angle)
        endRA   = POSRA (RA_of_date + scan_len/2)
        deltRA  = RA_of_date - RA2000

check object visibility
        command_HA = PMOD (ST / deg_to_sid_secs - start_RA)

        if ( command_HA > HALIM (+90.0, dec_of_date) )
        then { object has set, so quit }
        else { object is above the horizon mask, so continue }

        if ( command_HA < HALIM (-90.0, dec_of_date) )
        then { object not risen above horizon, so quit }
        else { object is above horizon mask, so continue }

        if ( coordinates within user specification of HA, elevation, time of
        day etc ) then continue

configure the telescope, receiver and instrument
        read system parameters from syparm, using restfreq
        connect receiver L+RCP pair to instrument L+RCP pair
        configure telescope (hyperbola tilt and focus, dichroic on/off)
```

```
        configure receiver (dual/single beam, set frequency,
        Dicke mode on/off, NAD diode on/off etc.)
        configure instrument (NAD/TP etc.)

quickly test the receiver and instrument with the calibration noise diode
        (number_of_failures_allowed, fault_report)
        continue if faults clear within specified number_of_failures_allowed

carry out the scan
        drive to (drive_RA, DEC)
        wake up when on target
        if ( calibrate on this scan ) then
        { calibrate receiver (counts_per_K, error_in_counts_per_K, fault) }
        if ( RA_now > start_RA ) then { drive to start_RA, DEC }
        while ( RA_now > start_RA and RA_now < end_RA )
        do { read and save ( instrument outputs, coordinates ) }

qualify data at end of scan
        if ( binning wanted ) { bin the outputs }
        if ( scaling wanted ) { scale the outputs to Kelvins }
        plot data on screen for visual data qualification
        (counts versus angle)

write raw or binned data to disk file

quit
```

4.4 Algorithm for Orthogonal Scan Mapping

JJ: I have written the algorithm in pseudo-C, which I hope is reasonably clear. The structure is quite linear, just running from top to bottom. It assumes that the scheduler has checked the input file keywords and parameters for basic errors and inconsistencies, and has done basic hardware checks and locks. I am not sure I know where the dividing line comes between the scheduler and the observing procedure, and what the observing program gets given and is expected to return. The procedure for doing the map scan is quite trivial, what is important is the way coordinates and data are handled.

In what follows I have assumed that the observing program will have access to all of the parameters in the input file, and that it writes its own data to the FITS output file. It assumes the FITS file is created by the scheduler and that the scheduler also ensures that the primary array header is populated with housekeeping information (as seen in the output file example) from the input file and SYPARM.

Commands to Steer are assumed to be via a pseudo-structure:

```
typedef struct { char   Identifier[9];
                 double Expires;
                 short  Coordsys;
                 short  Pointing;
                 short  Dichroic;
                 char   Feedsys[25];
                 short  Secondary;
                 double Longitude[2];
```

```
                 double Latitude[2];
                 double psi, phi, theta,
                 double Parallax } SteerComdStruc;
```

and returned information from Steer is assumed to be in the form of an array of this pseudo-structure:

```
typedef struct { double X;           /* projected coordinate of beam */
                 double Y;
                 double Longitude;   /* celestial coordinate of beam */
                 double Latitude;
                 double Azimuth;     /* horizon coordinate of beam */
                 double Elevation;
                 long   Count1;      /* counts from up to 4 v/f's */
                 long   Count2;
                 long   Count3;
                 long   Count4 } SteerRetStruct;
```

The mapping algorithm is:

```
void Mapping_Scan (OBJECT, RESTFREQ, INSTRUME, MODE, COORDTYP,
                     EQUINOX, (RA et al), (DEC et al), LONGPOLE, PROJTYPE,
                     fitsfile *FitsFilePointer);

/* local variables */

struct SteerComdStruc SteerCommand;      /* Steer command structure */
struct SteerRetStruct *SteerDataPointer; /* Steer return array */
int NumberOfSamples;                     /* returned size of Steer array */
                                         /* binary table header info */
char *ttype[] = {"X","Y","xxxx","yyyy","AZ","EL","LCPcount","RCPcount"};
char *tform[] = {"1E", "1E", "1E", "1E", "1E", "1E", "1J", "1J"};
char *tunit[] = { "DEG", "DEG", "DEG", "DEG", "DEG", "DEG",
                  "COUNTS", "COUNTS" };

int status;  /* returned CFITSIO status */
             /* error returns should be trapped appropriately in code */

/* some startup stuff */

Set_up_radiometer_and_counters (RESTFREQ, INSTRUME, MODE);

SteerCommand.Identifier = OBJECT;

/* this chunk of code may be common to other observing modes too */

switch (COORDTYP)
  EQUATORIAL:
    CRVAL1 = RA; CRVAL2 = DEC;
    switch (EQUINOX)
      B1950:
```

```
        SteerCommand.Coordsys = B1950; break;
      J2000:
        SteerCommand.Coordsys = J2000; break;
      MEAN:
        SteerCommand.Coordsys = MEAN; break;
      APPARENT:
        SteerCommand.Coordsys = APPARENT; break;
      default:
        SteerCommand.Coordsys = J2000;
        GeneralPrecession (EQUINOX, 2000.0, CRVAL1, CRVAL2);
        break;
    break;
  GALACTIC:
    SteerCommand.Coordsys = GALACTIC;
    CRVAL1 = GLON; CRVAL2 = GLAT;
    break;
  ECLIPTIC:
    SteerCommand.Coordsys = ECLIPTIC;
    CRVAL1 = ELON; CRVAL2 = ELAT;
    break;
  HORIZON:
    SteerCommand.Coordsys = HORIZON;
    CRVAL1 = AZ; CRVAL2 = ALT;
    break;
  TOPOCENTRIC:
    SteerCommand.Coordsys = TOPOCENTRIC;
    CRVAL1 = HA; CRVAL2 = DEC;
    break;

  SteerCommand.Pointing = True;
  SteerCommand.Dichroic = False;   /* for now */
  SteerCommand.Feed = Some_Function_Of (RESTFREQ, INSTRUME, MODE);
  SteerCommand.Secondary = False    /* for now*/

  switch (PROJTYPE)
    NONE:
      SteerCommand.theta = 0.0; SteerCommand.phi = 0.0;
      SteerCommand.psi = 0.0;
      SteerCommand.Projection = CAR;  /* what steer currently uses */
      break;
    CAR:
      SteerCommand.theta = 0.0; SteerCommand.phi = - CRVAL2;
      SteerCommand.psi = + CRVAL1;
      SteerCommand.Projection = CAR;  /* what steer currently uses */
      break;
    SIN:
      SteerCommand.theta = +LONGPOLE; SteerCommand.phi = 90 - CRVAL2;
      SteerCommand.psi = + CRVAL1;
      SteerCommand.Projection = SIN;  /* not implemented in steer yet */
      break;
```

```
    TAN:
      SteerCommand.theta = +LONGPOLE; SteerCommand.phi = 90 - CRVAL2;
      SteerCommand.psi = + CRVAL1;
      SteerCommand.Projection = TAN;  /* not implemented in steer yet */
      break;

  SteerCommand.Parallax = 0.0;

/* drive to start of scan */

if (PROJTYPE == NONE)
    /* not a proper coordinate projection */
    SteerCommand.Longitude[0] = CRVAL1 + SCANSTRT.X;
    SteerCommand.Longitude[1] = 0.0;
    SteerCommand.Latitude[0]  = CRVAL2 + SCANSTRT.Y;
    SteerCommand.Latitude[1]  = 0.0;
  else
    /* do projection properly */
    SteerCommand.Longitude[0] = SCANSTRT.X; SteerCommand.Longitude[1] = 0.0;
    SteerCommand.Latitude[0]  = SCANSTRT.Y; SteerCommand.Latitude[1]  = 0.0;

SteerCommand.Expires = Now + 1 hour;

Slew_Antenna_and_Wait_till_tracking (SteerCommand);

/* calibration while tracking start of scan */

Do_Antenna_Temperature_Calibration_and_Write_to_FITS_file;

/* calculate scan rates for both x and y */

SteerCommand.Longitude[1] = (SCANSTOP.X - SCANSTRT.X) / SCANTIME;
SteerCommand.Latitude[1]  = (SCANSTOP.Y - SCANSTRT.Y) / SCANTIME;

/* allocate an array of appropriate size */

SteerDataPointer = malloc (sizeof(SteerRetStruct) * (SCANTIME / 0.1 + 1));

/* go off and do scan */

Scan_Antenna_and_Log_Counters (SteerCommand, SteerDataPointer,
                               &NumberOfSamples);

/* set up some FITS keywords and open binary table */

switch (SteerCommand.Coordsys)
  TOPOCENTRIC: ttype[2] = "HA"; ttype[3] = "DEC"; break;
  HORIZON: ttype[2] = "AZ"; ttype[3] = "EL"; break;
  APPARENT, MEAN, B1950, J2000: ttype[2] = "RA"; ttype[3] = "DEC"; break;
  GALACTIC: ttype[2] = "GLON"; ttype[3] = "GLAT"; break;
```

```
  ECLIPTIC: ttype[2] = "ELON"; ttype[3] = "ELAT"; break;

status = 0;
fits_create_tbl (FitsFilePointer, BINARY_TBL, 0, 8, ttype, tform, tunit,
                 "SCAN_TABLE", &status);

fits_write_key (FitsFilePointer, TSTRING, "TDISP1", "F8.3",
                "recommended display format", &status);
fits_write_key (FitsFilePointer, TSTRING, "TDISP2", "F8.3",
                "recommended display format", &status);
fits_write_key (FitsFilePointer, TSTRING, "TDISP3", "F8.3",
                "recommended display format", &status);
fits_write_key (FitsFilePointer, TSTRING, "TDISP4", "F8.3",
                "recommended display format", &status);
fits_write_key (FitsFilePointer, TSTRING, "TDISP5", "F8.3",
                "recommended display format", &status);
fits_write_key (FitsFilePointer, TSTRING, "TDISP6", "F8.3",
                "recommended display format", &status);
fits_write_key (FitsFilePointer, TSTRING, "TDISP7", "I12",
                "recommended display format", &status);
fits_write_key (FitsFilePointer, TSTRING, "TDISP8", "I12",
                "recommended display format", &status);

/* stuff data one row at a time into a binary table in the FITS file */
/* Count3 and Count4 ignored */

for (row = 1; row <= NumberOfSamples; row++, SteerDataPointer++)
  fits_write_col (FitsFilePointer, TFLOAT, 1, row, 1, 1,
                  &(SteerDataPointer->X), &status);
  fits_write_col (FitsFilePointer, TFLOAT, 2, row, 1, 1,
                  &(SteerDataPointer->Y), &status);
  fits_write_col (FitsFilePointer, TFLOAT, 3, row, 1, 1,
                  &(SteerDataPointer->Longitude), &status);
  fits_write_col (FitsFilePointer, TFLOAT, 4, row, 1, 1,
                  &(SteerDataPointer->Latitude), &status);
  fits_write_col (FitsFilePointer, TFLOAT, 5, row, 1, 1,
                  &(SteerDataPointer->Azimuth), &status);
  fits_write_col (FitsFilePointer, TFLOAT, 6, row, 1, 1,
                  &(SteerDataPointer->Elevation), &status);
  fits_write_col (FitsFilePointer, TLONG, 7, row, 1, 1,
                  &(SteerDataPointer->Count1), &status);
  fits_write_col (FitsFilePointer, TLONG, 8, row, 1, 1,
                  &(SteerDataPointer->Count2), &status);

/* leave scheduler to close FITS file */
```

5 Subroutines needed

The following is based on HP code, but there is some guesswork here as I don't really know just how SYPARM will work, what calls to it are like, or what commands can be sent to STEER and what data can be obtained from STEER.

- Input file reader
- Input parameter parser
- Syparm reader
- Telescope configurator
- Receiver configurator
- Instrument configurator
- System Temperature Calibration using noise diode and LO
- Counts per Kelvin Calibration using noise diode
- Convert input parameters to scan parameters
- Coordinate transformations
- PMOD Returns a value of angle in PMOD, -180.0 < PMOD < 180.0
- HALIM Returns east or west hour angle limits
- Telescope driver
- Instrument output and telescope coordinates reader
- Data binning
- Data scaling
- Plot data on screen annotated with key parameters
- Write housekeeping and data to file
- Write log file

5.1 System Temperature Calibration subroutine

```
// standard calibration routine for absolute total power mode:

// usually just two polarizations, but
// note that with a polarimeter there could be four counters to read

inputs: counter_to_read[Ncounters]
        noise diode to fire
        local oscillator for receiver
        ND_in_Kelvins_for_this_output[Ncounters]
        Nsamples
outputs: Tsys[Ncounters]
        Tsys_err[Ncounters]
```

```
        CperK[Ncounters]
        CperK_err[Ncounters]
        NDerror[Ncounters]
        LOerror[Ncounters]
        output[Ncounters][Nsets]Nsamples]

// three sets samples are taken
Nsets = 3

// initialise the outputs
for (ic = 0 ; ic < Ncounters ; ic++ ) do
        {
        COerror[ic] = 0 /* counter problem */
        NDerror[ic] = 0 /* noise diode problem */
        LOerror[ic] = 0 /* local oscillator problem */
        CperK[ic] = 0   /* counts per Kelvin per counter */
        CperK_err[ic] = 0 /* uncertainty in CperK per counter */
        Tsys[ic] = 0    /* System temperature per counter */
        Tsys_err[ic] = 0  /* uncertainty in Tsys per counter */
        for ( is = 0 ; is < Nsets ; is++ ) do
                {
                for ( i = 0 ; i < Nsamples ; i++ ) do
                        { output[ic][is][i] = 0 }
                }
        }

// obtain three sets of data
for ( is = 0 ; is < Nsets ; is++ ) do
        {
        if ( is = 0 ) then { turn off local oscillator,
                             turn off noise diode }
        if ( is = 1 ) then { turn on local oscillator,
                             turn on noise diode }
        if ( is = 2 ) then { turn off noise diode }

        pause to let output settle

        ask all counters in use for Nsamples of output

        check for errors reading counters

        // get mean and rms for all good counters (ic) and each sample (i)
        for ( ic = 0 ; ic <= Ncounters ; ic++ ) do
                {
                sum[ic] = 0
                sum_sqd[ic] = 0
                for ( i = 0 ; i <= Nsamples ; i++ ) do
                        {
                        sum[ic] = sum[ic] + output[ic][is][i]
                        sum_sqd[ic] = sum_sqd[ic] + output[ic][is][i]**2
```

```
                }
            mean[ic][i] = sum[ic] / Nsamples
            rms[ic][i] = sqrt ((sum_sqd[ic] - sum[ic]**2/Nsamples)
                      / (Nsamples - 1)
            }
        }

// now we have all the data, lets see what we can calculate from it
// for each good counter:
for ( ic = 0 ; ic <= Ncounters ; ic++ ) do
        {
        // check that the noise diode fired (ND > 6 x RMS noise):
        if ((mean[ic][2] - mean[ic][3]) < 3*(rms[ic][2]+rms[ic][3])) then
                { NDerror[ic] = 1 }

        // check that the local oscillator switched off:
        if ((mean[ic][3] - mean[ic][1] < 3*(rms[ic][3]+rms[ic][1])) then
                { LOerror[ic] = 1 }

        // calculate the counts per Kelvin and error:
        if ( NDerror[ic] = 0 ) then
                {
                CperK[ic] = (mean[ic][3] - mean[ic][2]) / ND_in_Kelvins[ic]

                CperK_err[ic] = sqrt (rms[ic][3]**2 + rms[ic][2]**2)
                                / ND_in_Kelvins[ic]
                }

        // calculate the system temperature and error:
        if ( NDerror[ic] = 0 && LOerror[ic] = 0) then
                {
                Tsys[ic] = (mean[ic][3] - mean[ic][1])
                           / (mean[ic][2] - mean[ic][3])
                           * ND_in_Kelvins[ic]

                Tsys_err[ic] = Tsys[ic] * sqrt ((
                                 (rms[ic][3]**2 + rms[ic][1]**2) /
                                 (mean[ic][3] - mean[ic][1])**2) +
                                ((rms[ic][2]**2 + rms[ic][3]**2) /
                                 (mean[ic][2] - mean[ic][3]**2))
                }
        }
```