MRTCAL Programmer Manual

Questions? Comments? Bug reports? Mail to: `gildas@iram.fr`

The GILDAS team welcomes an acknowledgment in publications
using GILDAS software to reduce and/or analyze data.
Please use the following reference in your publications:
`http://www.iram.fr/IRAMFR/GILDAS`

Documentation

In charge: S. Bardeau[1].
Active developers: J. Pety[1,2], A. Sievers[3].

Software

In charge: S. Bardeau[1], A. Sievers[3].
Active developers: J. Pety[1,2].

1. IRAM (Grenoble)
2. Observatoire de Paris
3. IRAM (Granada)

Related information is available in:

- MRTCAL User Manual
- CLASS Manual
- GREG: Graphical Possibilities
- SIC: Command Line Interpretor

Contents

Chapter 1

Chunks and chunksets

For the processing of the **IMBFITS** files, **MRTCAL** introduces the concept of *chunks* (see Fig 1.1). One dump of the **IMBFITS** `DATA` column is a collection of such chunks. **MRTCAL** can make no assumption on the way they are ordered (this comes from ???). The `PART` column indicates to which spectrum each chunk belongs (spectra are identified with integer numbers). Each "set of chunks belonging to the same final spectrum" is called a *chunkset* in the **MRTCAL** nomenclature. From one chunkset, **MRTCAL** will produce one spectrum[1].

The chunks in each `DATA` row can belong to several spectra (*e.g.*, 8 for FTS), and by definition to as many chunksets. **MRTCAL** maps those chunksets internally as a `chunkset_1D` Fortran structure, *i.e.*, an array of chunksets.

The `DATA` column has as many rows as the number of pixels of the receiver times the number of time dumps. This introduces 2 new dimensions which are used to map all[2], the `DATA` column into a `chunkset_3D` Fortran structure. The 2 new dimensions here are *pixels* and *time* (see Fig 1.2).

For its internal needs, **MRTCAL** has to refer to a single time dump in the 3rd dimension. It can also time-average this 3rd dimension. In those 2 use-cases, the final product is then available through a `chunkset_2D` Fortran structure.

1.1 Polarimetry

The presence of polarimetric measurements in the **IMBFITS** file is described in the `POLAR` column of the `BACKEND` table. For a non-polarimetric scan, all the chunks in the `POLAR` column have the `NONE` value. For a polarimetric scan, values can be `NONE`, `AXIS`, `REAL` or `IMAG` (note that non-polarimetric chunks `-NONE-` can be mixed with polarimetric ones).

In addition, calibration scans for polarimetric measurements must provide a `calGrid` subscan. It absent, **MRTCAL** will mark this scan failed as it can not be used to calibrate polarimetric science scans.

[1] For debugging purpose, **MRTCAL** can also produce one spectrum per chunk.

[2] **MRTCAL** can also map a fraction of the `DATA` column, process it, and then iterate on the next fraction. This can be useful on memory-limited machines.

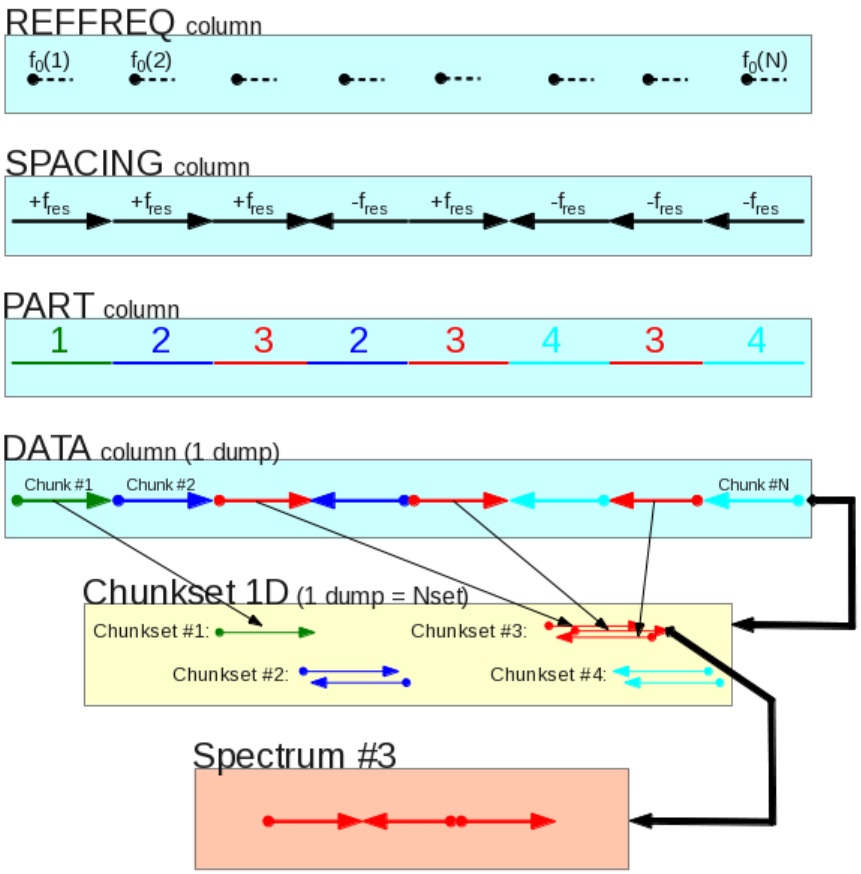

Figure 1.1: How one dump in the **IMBFITS** `DATA` column is divided into chunks, gathered by family (chunksets), and reordered to produce the spectra.

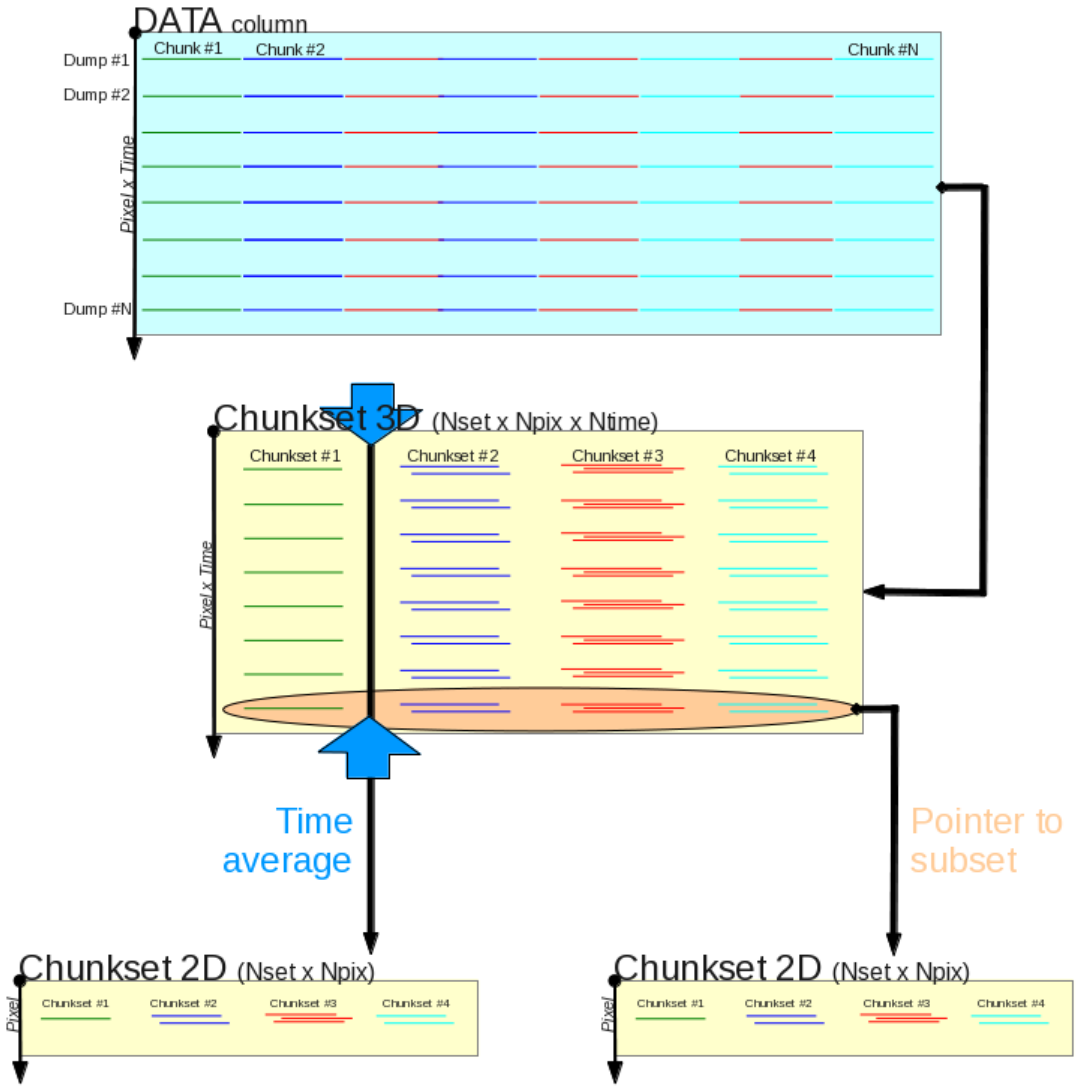

Figure 1.2: How all the dumps in the **IMBFITS** `DATA` column is divided into 3D chunksets. After this, they can be time-averaged, or can be accessed one by one.

Chapter 2

Reading the DATA column

2.1 What about a pixel dimension?

All the **MRTCAL** internal data structures and engines have been written in the order (*i.e.* from contiguous to less contiguous):

1. channels,
2. chunks,
3. pixel(s),
4. time dumps

with the assumption this is the natural and desired ordering of the data coming from the backend, which for example writes as

```
type imbfits_data_t
  integer(kind=4) :: ntime = 0  ! Number of 'time' rows read
  integer(kind=4) :: npix  = 0  !
  integer(kind=4) :: nchan = 0  ! Number of channels
  real(kind=4), pointer :: val(:,:,:) => null()  ! [nchan,npix,ntimes] The DATA block
  real(kind=4), pointer :: wei(:)     => null()  ! [nchan] The WEIGHT array (same for all cha
end type imbfits_data_t
```

in the internal data structure mapping (a piece of) the DATA column. For a given time dump and a given pixel, the dimension channels $\times$ chunks is contiguous and treated as a *spectrum* to be split and reordered when relevant (this is why the chunk dimension does not appear in the data structure).

However, HERA **IMBFITS** data does not follow this ordering. It is actually ordered as

1. channels,
2. pixel(s),
3. chunks,
4. time dumps.

Because of this, the pixel dimension could not be used. For the time being, it is kept degenerate (npix=1), and the pixel dimension is melted in the channels × chunks dimension.

For future multibeam receivers, we will work with the **IMBFITS** developers to evaluate (in particular in terms of efficiency) and if possible ensure that the data is ordered as we expect. Note that this order could for example allow for parallelized processing of the pixels.

2.2 Discarding bad time dumps

The BackendDATA table provides the data dumps collected during the subscan integration. In particular, it shows the dump `MJD`, its integration time `INTEGTIM`, its phase number through cycles (`ISWITCH`) and the data itself (channels) in the `DATA` column.

It may happen that a dump is flagged out by the control system. In this case, its `ISWITCH` value is 0. Such a dump should be discarded during the calibration process. This is achieved with the command `MSET CALIBRATION BAD YES|NO` (default `NO`, i.e. discard bad time dumps).

In practice, this is not just a matter of skipping undesired rows when dealing with the `DATA` column. The first issue is that the values in the associated columns are unreliable. This is particularly true with the `MJD` value which is not set to the actual MJD of the dump. This introduces inconsistent values in this column, resulting in an unsorted column, breaking the dichotomic search engine. In order to solve this issue, **MRTCAL** *compresses* at read time the `MJD`, `INTEGTIM`, and `ISWITCH` columns, discarding the rows where `ISWITCH` is null. The size of those columns is decreased from N_{tot} to N_{good} (with $N_{tot} = N_{good} + N_{bad}$).

MRTCAL keeps also track of the discarded dumps thanks to two new columns added. One column named `FOREPOIN` (size N_{tot}) provides a forward pointer, i.e. `FOREPOIN(i)` is the position of the i^{th} dump (as found in the original columns) in the compressed columns. If the dump is bad, the corresponding position is set to 0 (i.e. dump not available in the compressed columns). It also adds a column named `BACKPOIN` (size N_{good}) which provides a backward pointer: `BACKPOIN(j)` is the position of the j^{th} dump (as found in the compressed columns) in the original columns.

At this stage, **MRTCAL** has patched the BackendDATA columns so that the bad dumps are removed, but this is not fully transparent for later use.

2.3 Dump cycle identification

From a general point of view, the dumps have to be calibrated by cycles, each cycle repeating the same sequence of several phases. In order to do this, **MRTCAL** identifies those sequences in the `ISWITCH` column. As this column may have been compressed, the identification also takes into account the `BACKPOIN` column: a cycle is valid if the expected sequence is found in the compressed column AND if the associated back pointers are contiguous in the original columns.

2.4 Reading the `DATA` column by pieces

MRTCAL is designed to be able to process the subscans within an arbitrary limited buffer size (`MSET BOOKKEEPING SPACE Value`, default 512 MBytes). If the buffer size is large enough, the whole `DATA` column can be loaded at once in memory and its dumps can be treated in a row. If the buffer is not large enough, the `DATA` column has to be load by pieces, and each piece has to be processed separately from the other.

In practice, the calibration routine asks the reading routine to load the piece of `DATA` which contains *at least* a desired range of dumps (usually a cycle of a few dumps). *At least* means that

the reading routine ensures that the range will be present entirely in the buffer, but more dumps will also be loaded in order to fill at best the buffer. On subsequent requests, if the desired range is already present in the buffer, nothing will be reloaded, in order to save calls to `CFITSIO` and I/O accesses.

At this stage, one should consider that `CFITSIO` is asked to read a block of rows from the `DATA` column. `CFITSIO` is not able to skip some bad rows here and there, and it would be unefficient to ask for reading the desired rows one by one. So `CFITSIO` reads a piece of the `DATA` column, including good and bad dumps. But right after, **MRTCAL** sets a 3D chunkset ($N_{set} \times N_{pix} \times N_{time}$) pointing to the good dumps only. Then the calling routine has just to work with the 3D chunkset, with no need to care for bad dumps.

Chapter 3

Positions

3.1 Reference position

For each spectrum, the Class Data Format expects a reference (usually source) position. In **MRTCAL**, this reference is taken from the `LONGOBJ` and `LATOBJ` (*Source longitude and latitude in basis frame*) in the Scan HDU. The coordinate system of these values are taken from header cards `CTYPE1` and `CTYPE2` (`RA` and `DEC` for equatorial system, `GLON` and `GLAT` for galactic system). Note that the trailing characters describing a projection system (*e.g.* `-SFL`) are useless: they do not apply to any data which can be found in the Scan HDU.

3.2 Scan offsets

From the absolute reference defined at 3.1, the **IMBFITS** format can define an intermediate position, through the `SYSOFF`, `XOFFSET` and `YOFFSET` columns in the `SCAN` table

- if only the *Nasmyth* row is present, an implicit *projection* row is added at read time with (0,0) offsets,
- if the *projection* row is available, the associated offsets are added to the antenna offsets (see next section) and used into the **CLASS** Data Format,
- if something else than *projection* is available, *e.g. horizontalTrue*, a specific support is not yet available in **MRTCAL** and an error is raised. However, if the associated offsets are (0,0)[1], Mrtcal will tolerate this system of offsets. This is a temporary solution, waiting for a full support of all systems.

3.3 Antenna offsets

The antenna offsets are found in the `LONGOFF` and `LATOFF` columns in the Antenna Slow HDU. These antenna offsets are to be added to the scan offsets defined at 3.2. Note that these offsets describe the *primary dish* pointing position. In case of wobbler switching, the beam does not point to this position (see section 3.3.3 for details).

[1]This can happen *e.g.*, if the user has commanded `OFFSETS 0.0000000E+00 0.0000000E+00 /SYSTEM "trueHorizon"` in **PAKO**

3.3.1 Projection system

The antenna offsets are expressed in a system described by the card `SYSTEMOF` in the Antenna HDU. If its value is:

- `projection`, this means the offsets are radio-projected in the current coordinate system, *i.e.* the system found at 3.1; the offsets can be used "as is" in CLASS,
- something else, *e.g.* `horizontalTrue` for `calSky`, the offsets are not expressed in the correct frame and should be recomputed to the desired sky coordinate system (CLASS does not offer the possibility to express the reference in one system and the offsets in another). ZZZ this is not yet implemented, warning for now.

3.3.2 Interpolation

The antenna offsets are derived from the Antenna Slow table together with other position values. Namely the 5 following elements have to be computed:

- lambda offset from reference position,
- beta offset,
- azimuth,
- elevation,
- local sidereal time.

All these elements are available in the `ANTSLOW` table, at a typical sampling rate of 1 Hz. Since the spectra dumps can be produced at different time sampling (no assumption is made on the rate or its variations), each spectrum positions are interpolated from the `ANT` table thanks to their respective *Modified Julian Day* values

$$f = \frac{\mathrm{mjd}_S - \mathrm{mjd}_A(j)}{\mathrm{mjd}_A(j+1) - \mathrm{mjd}_A(j)} \tag{3.1}$$

where mjd_S is the spectrum MJD value, and $\mathrm{mjd}_A(j)$ is the MJD value of the j^{th} trace in the `ANTSLOW` table. j is computed thanks to a dichotomic search in the table such as

$$\mathrm{mjd}_A(j) \leq \mathrm{mjd}_S < \mathrm{mjd}_A(j+1). \tag{3.2}$$

f being the interpolation fraction between the j^{th} and $j+1^{th}$ trace, the positions are interpolated by

$$l_S = l_A(j) + f \times [l_A(j+1) - l_A(j)]\,, \tag{3.3}$$

where l_S and l_A are the spectrum and antenna lambda offsets respectively. Same formula applies for the beta, azimuth, and elevation values.

If mjd_S is found beyond the `ANTSLOW` table limits, the boundary values are applied without extrapolation. However, this is not expected to happen since such spectra should be rejected since they are out of the on-track range.

3.3.3 Wobbler switching

In case of wobbler switching, the `LONGOFF` and `LATOFF` columns describe the primary dish direction on sky. Because the secondary mirror switches with additional offsets between the ON and OFF positions, **MRTCAL** assumes that the ON phases always have (0,0) antenna offsets. For simplicity, the OFF phases offsets are also set to (0,0) (instead of $\pm\Delta/2$) but the CLASS switching section will describe properly the throw when saving the ON-OFF result, or when saving the ON and OFF separately. Remember that the wobbler throw is usually along azimuth with a given projection, while the CLASS offsets are described in equatorial or galactic system with radio projection: their addition is not straightforward.

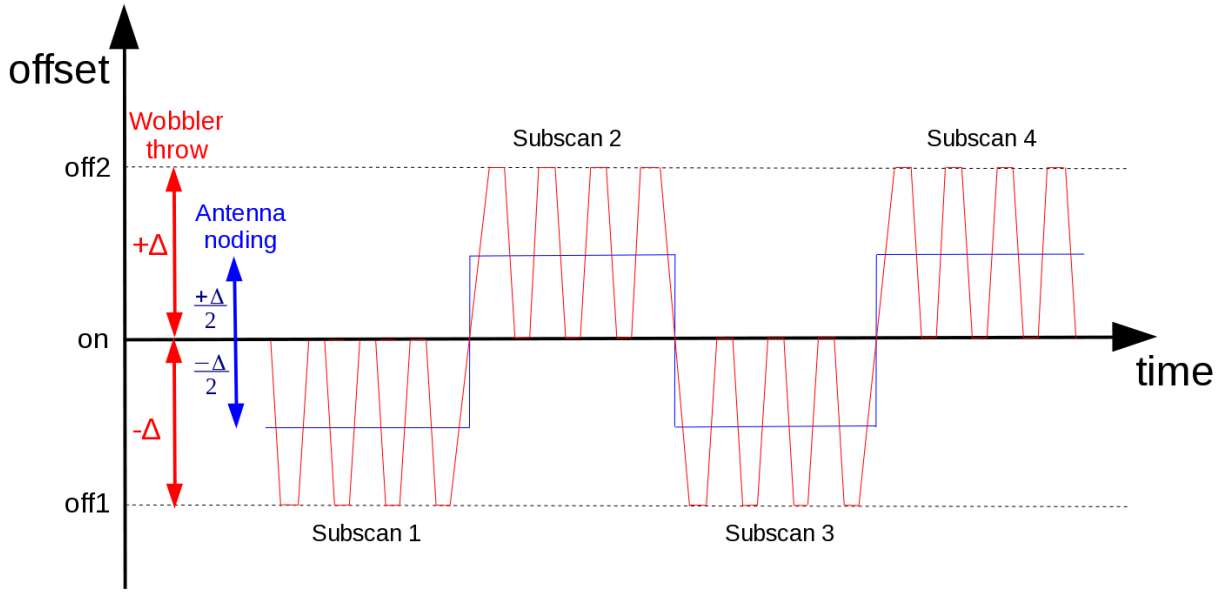

Figure 3.1: Asymmetrical wobbler switching illustration. For each subscan, the antenna (primary dish) points at $\pm\Delta/2$ away from the source, usually in the azimuthal direction (blue line). This is called *antenna noding*. During the subscan, the secondary mirror switches (red line) from the ON position (at offset 0) to one OFF position (at offset $\pm\Delta$). Δ is the *wobbler throw*. This pattern, which alternates between OFF1 and OFF2 at each subscan, is called *asymmetrical wobbler switching*, as opposed to the *symmetrical wobbler switching* (see details in Fig 3.2).

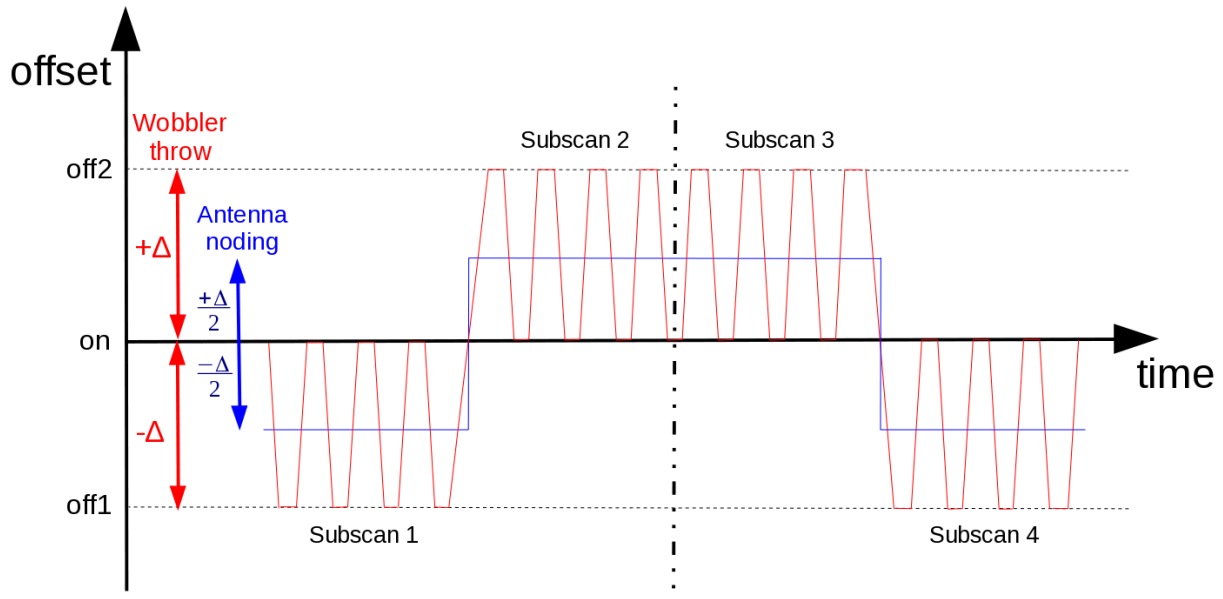

Figure 3.2: Symmetrical wobbler switching illustration. Same as Fig 3.1, except that the subscans are observed with symmetry around a given time. Thanks to this, and assuming that the atmospheric properties drift linearly through time, the atmospheric variations between the 2 OFF1 subscans and between the 2 OFF2 subscans average at the same level.

3.3.4 Beam switching and continuum drifts

Beam switching is used in the context of continuum drifts. The various positions involved are described in different parts of the FITS and CLASS headers.

In the FITS header, the relevant information from *e.g.* `iram30m-nbc-20220817s29-imb.fits` looks like:

```
IMBF-scan:
  Header:
    OBJECT   (C)  = Mars                  / Source name
    CTYPE1   (C)  = RA-SFL                / Basis system (longitude) --
    CTYPE2   (C)  = DEC-SFL               / Basis system (latitude) -- XLAT-SFL
    EQUINOX  (R8) =   2022.62422998199    / [Julian yrs] Equinox
    LONGOBJ  (R8) =   56.1815486637719    / [deg] Source longitude in basis frame
    LATOBJ   (R8) =   18.2461304985798    / [deg] Source latitude in basis frame
    SWTCHMOD (C)  = beamSwitching         / Switch mode
  Table:
    SYSOFF   (C)  = Nasmyth        projection  / WARNING! Added a dummy 'projection' value
    XOFFSET  (R4) = -1.9150140E-04   0.000000  / WARNING! Added a dummy 'projection' value
    YOFFSET  (R4) =  2.6664753E-05   0.000000  / WARNING! Added a dummy 'projection' value

IMBF-antenna (subscan #1):
  Header:
    SUBSTYPE (C)  = onTheFly              / Subscan type
    SYSTEMOF (C)  = horizontalTrue        / System for offsets
  Table:
    LONGOFF  (R8) = -2.908680211E-04 ... 2.908678739E-04 / [rad] long. offset from source in
    LATOFF   (R8) =   0.00000000     ...  0.00000000     / [rad] lat. offset from source in
```

The scan HDU is used to build the position section in the CLASS header. In this example, the reference position is taken from `LONGOBJ` and `LATOBJ` (in RA-Dec equinox 2022.6 coordinates in this example). According to the scan table, the position offset are (0,0) here (but they could be non-zero) with the radio projection (`SYSOFF` is `projection`).

The antenna HDU is used to build the drift offsets *from the position described above.* The table provides the whole drift offsets. It is stored in the CLASS `RX` array as it is not regularly spaced in the general case. Its associated system of coordinates is saved in the drift section (`ctype`, horizontal system in this example[2]). Its unit (angles here) is described in the general section (`xunit` parameter).

[2]Note that they are true (non-projected) angles on the sky, as per the `horizontalTrue` keyword. THIS IS TO BE CONFIRMED.

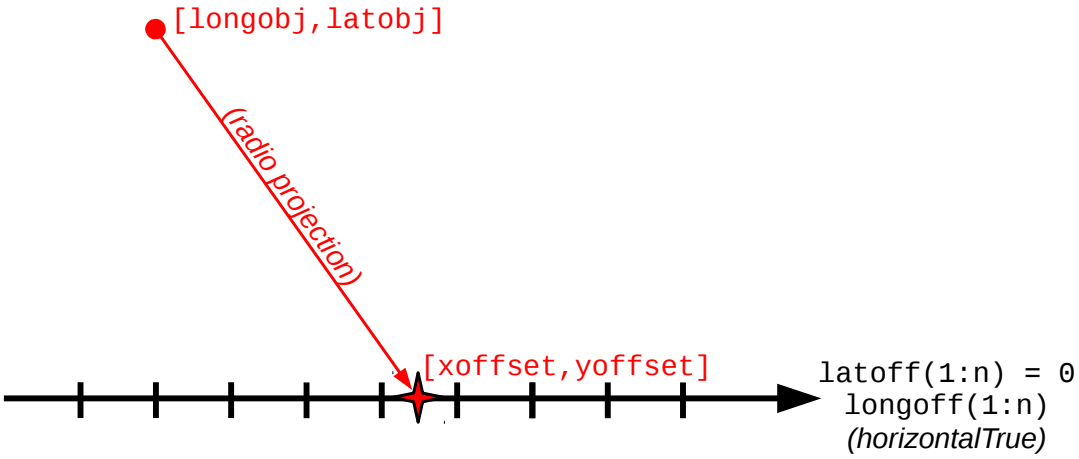

Figure 3.3: Schematic representation of a continuum drift (horizontal in this example). The red components show the scan HDU information. The black components show the antenna HDU information. Scan HDU provides the projection center, projection kind, and reference position used to fill the position section in CLASS. Antenna HDU provides the drift offsets used to fill the continuum drift section and the `RX` array in CLASS.

3.3.5 Slow and Fast traces

The **IMBFITS** files provide 2 antenna streams and associated tables: The slow (1 Hz) and the fast (128 Hz) traces. For each of those traces, some antenna position parameters are available, plus the associated time as MJD values.

As of today (**IMBFITS** version 2), the 2 traces are in the same FITS extension. Both have the same number of rows, but the fast trace has 128 values per column and per row, while the slow trace has 1 value per column and per row. However, even if they have the same number of rows, they are INDEPENDENT: For example, their MJDs are misaligned by a $\sim$2 seconds shift (2 rows). They should not be correlated! In the **IMBFITS** version 3, the 2 traces will be clearly separated in 2 FITS extensions, which will avoid any misinterpretation.

Note that the caveat exposed above explains why **MRTCAL** selects more dumps than **MIRA** in on-the-fly maps: **MIRA** looks at the antenna slow column `TRACEFLAG` to flag out MJD values in the antenna fast. Because of the $\sim$2 seconds shift between the 2 traces, more time dumps are discarded.

3.4 Offsets of multi-pixel receiver

In the **IMBFITS** files, the backend table describes how the chunks cover the bandwidth and from which receiver they come from (*e.g.* H or V polarization). In this table, the column `PIXEL` also describes from which pixel the signal comes. This is useful in the context of multi-pixel receivers. In such a case, there are several pixels which do not look at the same position on sky. In the frontend table header, the derotator mode (*sky*, *frame*, or *horizontal*) and angle describe the commanded inputs from the user. The derotator table, which is present in the **IMBFITS** files only if relevant, describes the commanded and actual derotator position through time, at typical intervals of 5 seconds. We describe below how **MRTCAL** makes use of these informations.

3.4.1 Handling the derotator

If the subscan has no derotator table, the current pixel (obviously the unique pixel) is left centered on the current position (reference + position offset) computed in the section 3.3.2. On the other hand, if a derotator table is present, **MRTCAL** will apply a per-pixel offset.

First of all, **MRTCAL** tries to guess the *actual* derotator angle *on sky*. The strategy is the following:

- angle on sky will be chosen from the column `sAct` ("sky Actual"), as opposed to `fAct` ("frame Actual") and `hAct` ("horizontal Actual"). This assumes `sAct` values are reliable even if the current derotator mode is not *sky*.
- if the column `sAct` is empty (this can happen around midnight), a warning is raised and the returned value is the commanded value as found in the frontend table header. In this case, the derotator mode must be *sky*.
- if the column `sAct` provides no value within the subscan range, a warning is raised.
- if the column `sAct` has a single row, or if the chunk MJD is beyond the table limits, the unique/nearest value is used.
- if the chunk MJD lies within the column `sAct` limits, the derotator angle is interpolated according the associated `MJD` column and to the chunk MJD computed earlier.

- if the chunk MJD is further than 5 seconds[3] from the nearest point in the column `sAct`, an additional warning is raised.

- finally, if the actual angle is more than 0.5 degrees away from the commanded value (*e.g.* derotator has reached its rotating limit), a warning is raised.

3.4.2 Computing the pixel coordinates

Given the actual position of the derotator, **MRTCAL** can compute the pixel position on sky from the current telescope position. This position has been pre-computed using the radio projection (`LONGOFF` and `LATOFF` columns in the IMB-FITS), only the per-pixel offsets are to be added. **MRTCAL** checks first if it knows the offsets for the current receiver (as found in the column `RECNAME` in the frontend table). The pixels offsets are not self-described in the **IMBFITS**: they must be hardcoded in the program, with a correct correspondance between the pixel number found in the backend table and the hardcoded offsets[4]. Once the offset of the current pixel is found, the strategy is the following.

- The current position (reference + position offset) is unprojected to absolute spherical coordinates.

- The pixel position angle on sky is computed, given the pixel position in the array and the derotator angle.

- The spherical distance from the array center is added using an inverse Haversine formula, i.e., the pixel position in absolute spherical coordinates is found by adding an arc of great-circle towards the correct direction.

- The resulting position is reprojected again as a reference + position offset in the current projection system.

The deprojection, offset in spherical coordinates, and reprojection are done by the Gildas internal engines.

3.4.3 Description of the radio projection

The unprojection-reprojection approach to add the pixel offsets is more accurate than simple offset addition to projected position, in particular far away from the equator of the coordinate system. If we look at the radio projection (its standard FITS name is Global Sinusoidal, abbreviated into GLS) of the full sky shown in Fig. 3.4, we can see that one can expect strong distortions near the pole. Note that the distortions depend on the declination of the object and NOT on the declination of the projection center[5]. In other words, the plane of projection is not tangent to the projection center, but to the equator instead. This is directly linked to the mathematical formula of the radio projection

$$x = (A - A_0) \times \cos(d), \tag{3.4}$$

$$y = d - d_0, \tag{3.5}$$

[3] 5 seconds is the derotator sampling rate under normal conditions.

[4] See the Fig.4 in the *HERA user manual* for this receiver.

[5] ... while most of the other projections offer minimal distortions near the projection center!

where (A_0, d_0) is the projection center, (A, d) the object absolute coordinates, and (x, y) its offsets in the projected map. The key point in this projection is that x is a function of d and not $d - d_0$. Hence, the deformation for wide fields of view will depend both on the distance to the projection center and the source declination! That's why the radio projection is deprecated and IAU recommends to replace it by the Sanson-Flamsteed projection (abbreviated in SFL).

The distortions near the pole may thus have a non-negligible impact on observations. Figures 3.5 to 3.7 show the effect in different conditions. The geometry of the multi-beam array shows no visible distortions, even one degree away from the source position, when the source position is located on the equator. However, the distorsions of the array geometry increase with the distance to the projection center when the source is located at high declination. Moreover, for the same source coodinates, the deformation at high source declination depends on the right ascension of the projection center (cf. Fig. 3.7), but they are independent of the projection center declination (cf. Fig. 3.6)!

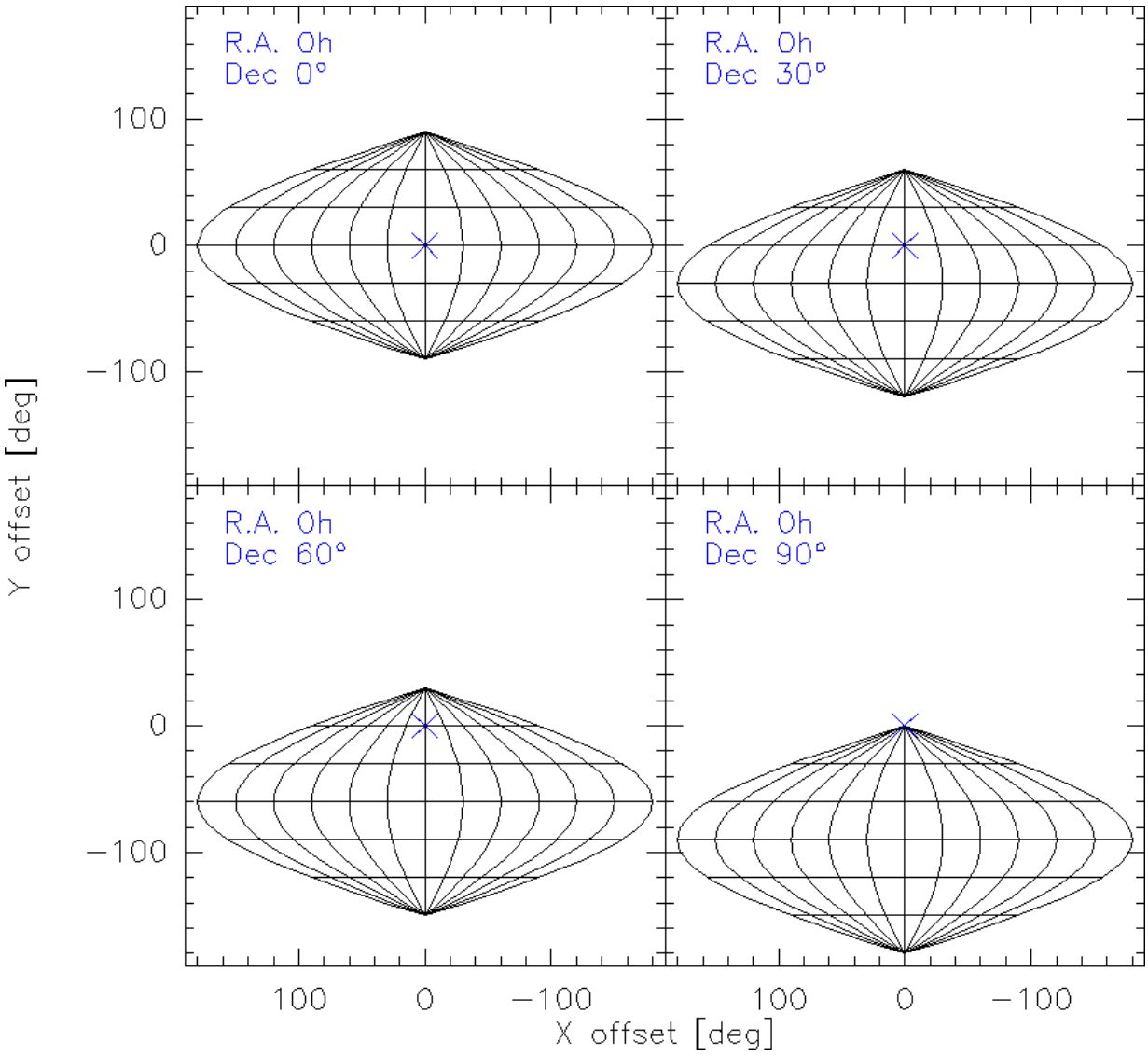

Figure 3.4: Radio projection of the full sky (right ascension from -12 to 12 hours, declination from -90 to 90 degrees), for four projection centers (marked with a blue cross) at 0 right ascension and 0, 30, 60, 90 degrees declination. The parallels (resp. the meridians) are spaced by 30 degrees (resp. 2 hours).

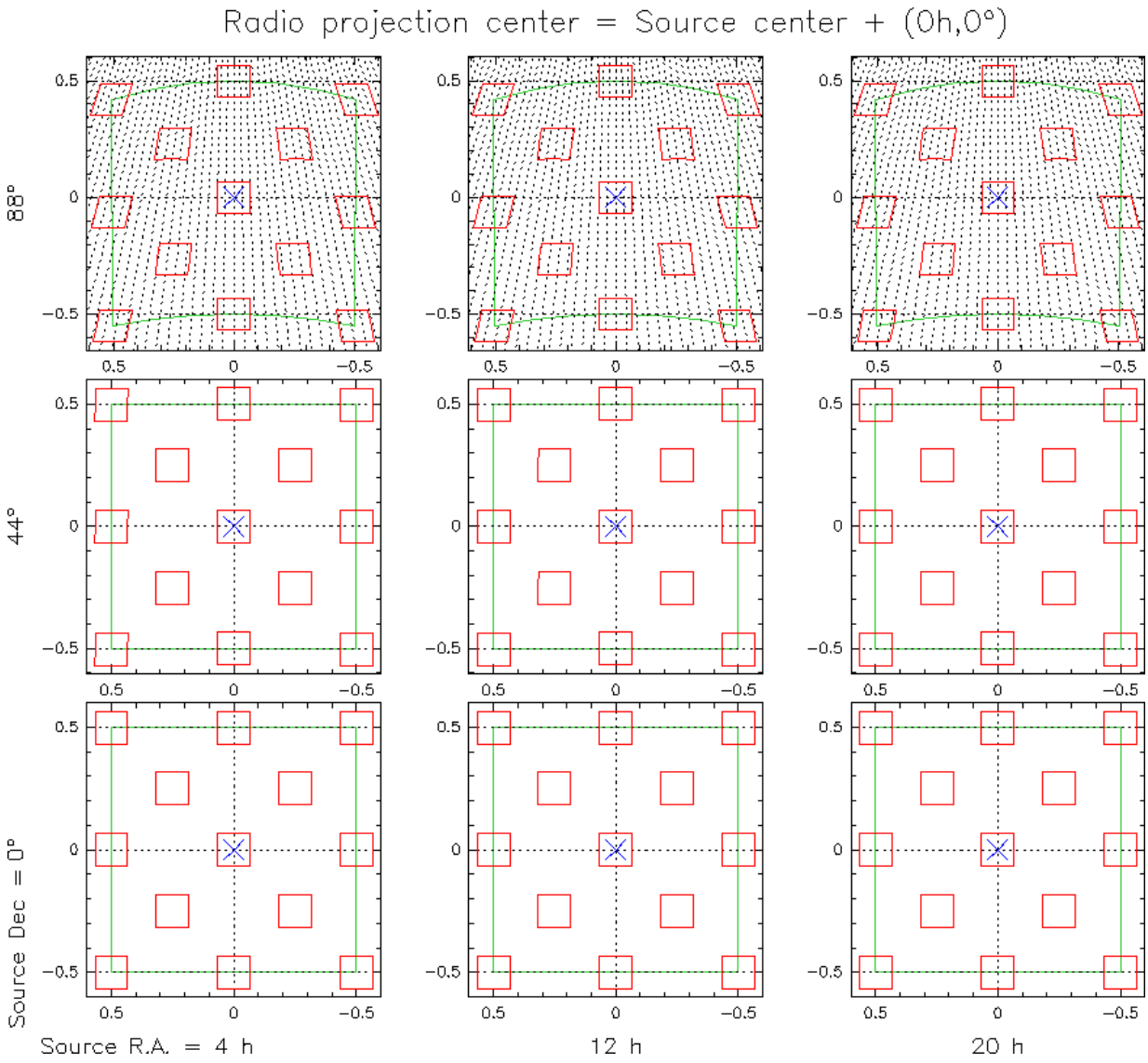

Figure 3.5: Effects of the radio projection on observations with HERA for source centers at 4h, 12h, 20h Right Ascension and 0, 44, 88 degrees declination (blue crosses). In this case, **the projection center is aligned on the source center.** The green polygon shows the shape of a 1 square degree field in sky spherical coordinates. In addition, the red parallelograms show the resulting shape of a $480'' \times 480''$ square multi-beam array as a function of the distance from the source center. Finally, the parallels (resp. meridians), shown as black dashed lines, are separated by 0.5 degree (resp. 1 degree).

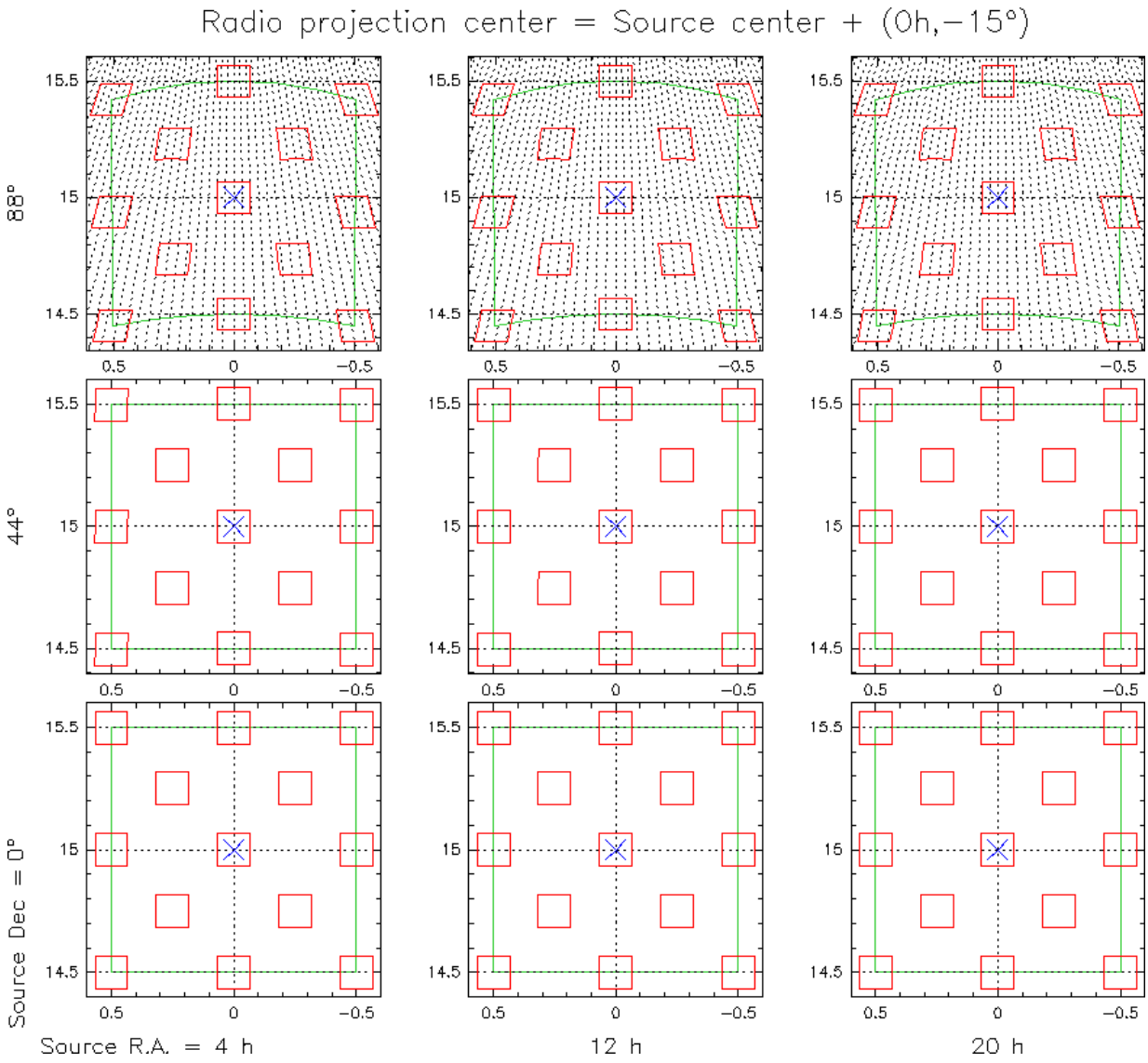

Figure 3.6: Same as Fig. 3.5, except that **the projection center is located 15 degrees south the source center.** The deformations are independent of the declination of the projection center!

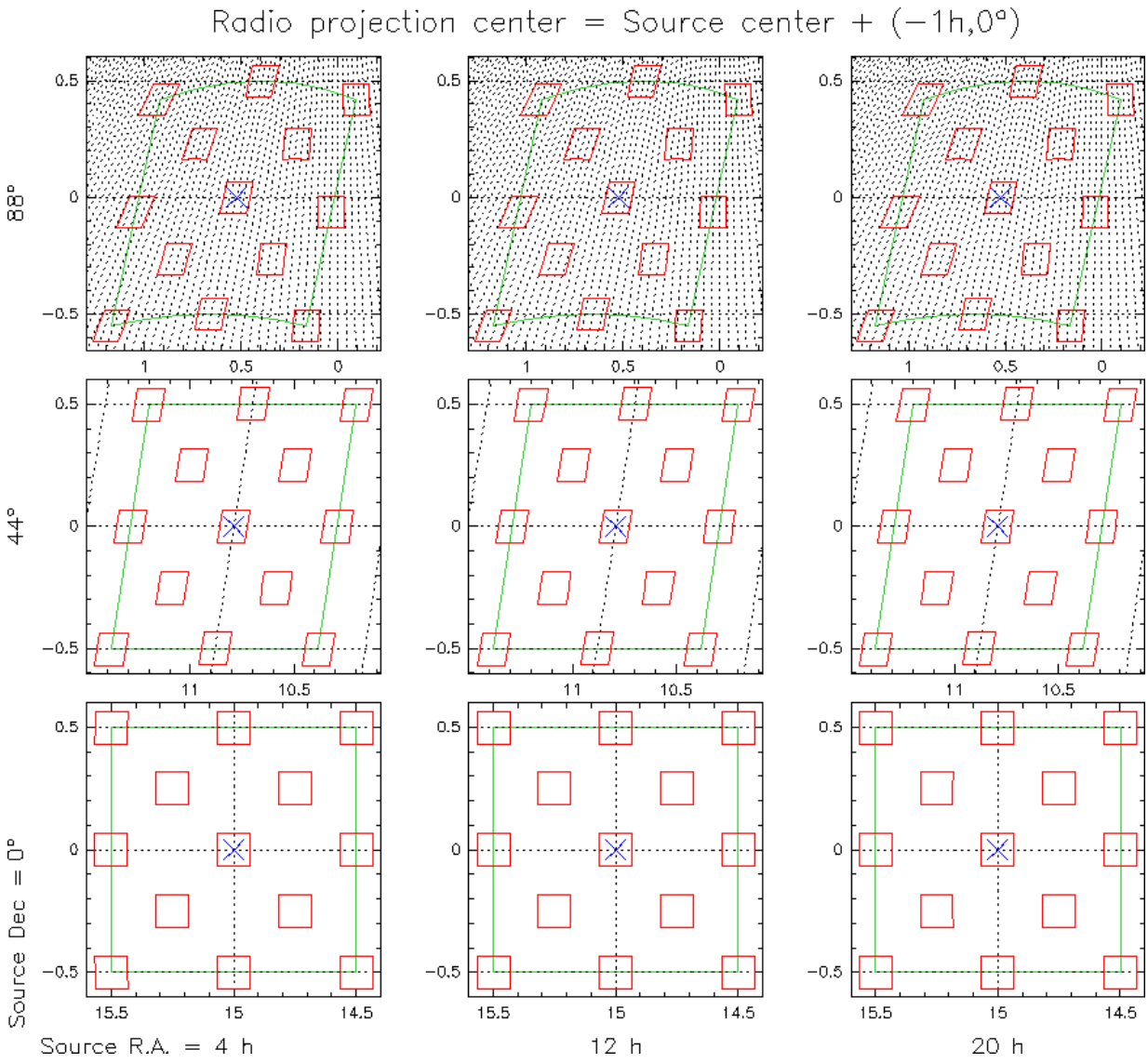

Figure 3.7: Same as Fig. 3.5, except that **the projection center is located 1 hour angle west from the source center.** The deformations depend on the right ascension of the projection center.

3.4.4 In practice: Effect on Polaris observations

For reference, we first remind the observing strategy advised in the HERA manual. Two main points will help us explain the behavior on Polaris.

- The array is rotated by a given angle that ensures Nyquist sampling and optimal sky coverage in a single observation (i.e., each portion of the sky is observed only by one of the nine pixels during a scan). This requires two contiguous subscans.
- However, the observing strategy usually set up at Pico Veleta samples regularly the *projected* sky and *not* the sky itself as naive users tries to obtained rectangular projected maps. Moreover the HERA receiver array is a perfect square in focal plane and on the sky, but *not* in the projected map!

Under observer-friendly conditions, namely *relatively* small maps near the projection center and maps *near* the equator, these rules deliver the expected result. However, if those conditions are not met (large field of view and the source is located at high declination), the sky and the projected map will both be incorrectly sampled!

Maps near Polaris (declination 87:42:04.6) where observed during the project 219-07 (PI: P.Hily-Blant). The maps extend typically from (0,0) to (-1500,2000) arcsec (projected offsets). We focus here on the scans 26 and 40 covering the projection center, and scans 98 and 115 observed at the largest distance from the projection center.

Figure 3.8 shows the scans observed near the projection center. They do not show any particular effect visible by eye. The array receiver is still square (rotated by 9.6 degrees[6]) on the projected map, and the coverage of the 2 scans is as expected. Figure 3.9 shows the scan coverages in spherical coordinates (i.e., without projection). The spherical sky is correctly sampled.

On the other hand, the scans observed far away from the projection center $(-1500'', 2000'')$ are highly affected by the projection distortions. The projected receiver array is not square anymore. It is parallelogram-shaped in the projected map. As a consequence, the rows scanned in declination are shifted. Figure 3.10 shows that the vertical scanning is not correctly sampled in the projected sky. This can be understood by looking at the scan in absolute coordinates (Fig. 3.11): The scan does not follow a South-North direction suited for the 9.6 degrees derotator angle on the sky. To first order, the angle should have been different (the exact value is not computed here)[7]. On the other hand, the scan along the right ascension is correctly sampled, but the start and end points of the rows are actually shifted in projected coordinates compared to the scan 26 near the projection center. This leads to different edge effects in right ascension according to the declination of the source center.

[6] See HERA Manual Fig.7

[7] To second order, the pattern is an arc and the derotator angle could have to be modified along the scan.

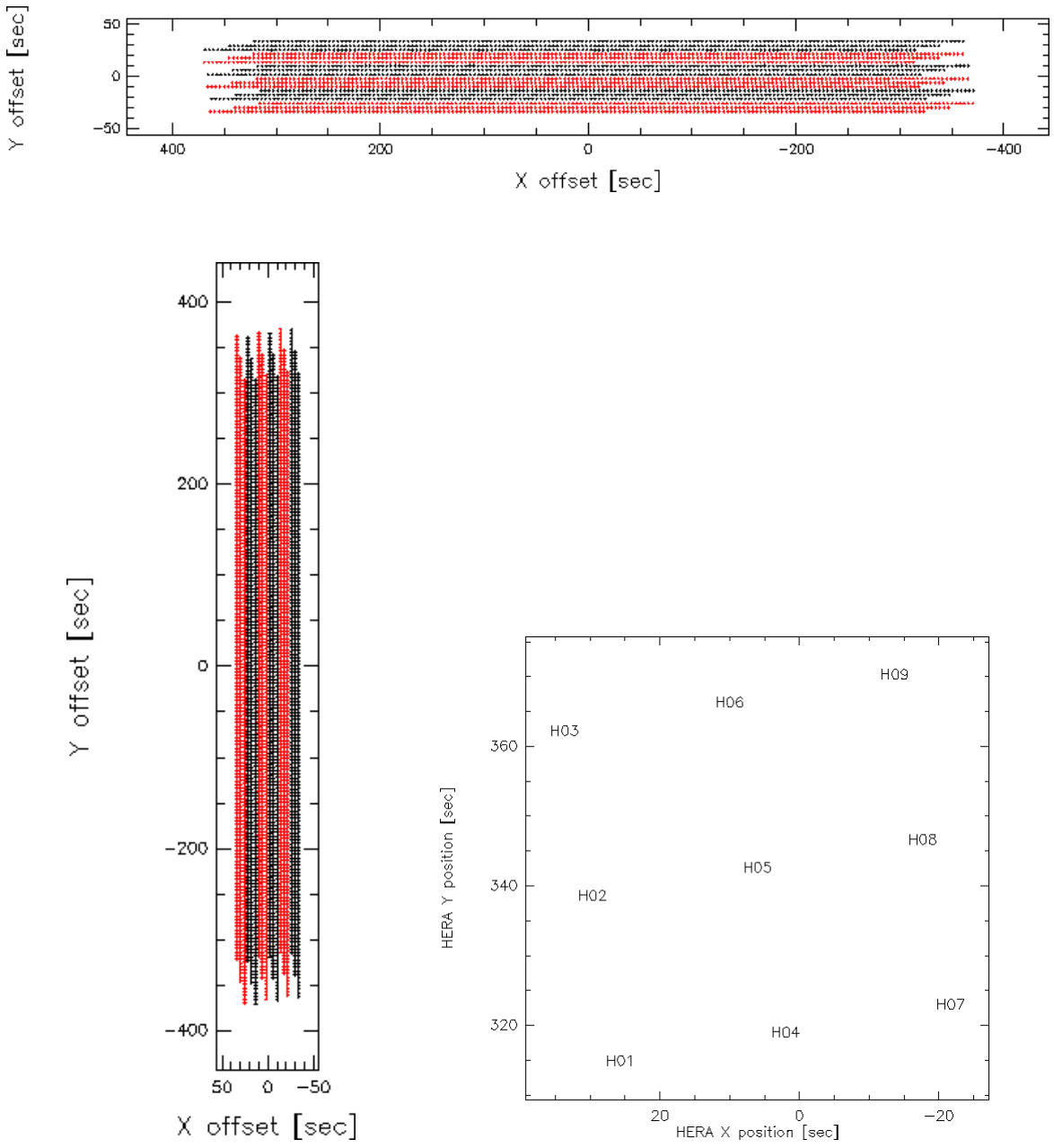

Figure 3.8: Scans 26 (top) and 40 (bottom) **relative coordinates (radio projection)** of project 219-07 observing Polaris (declination 87:42:04.6). The observing strategy is the usual one, i.e. there were 2 OTF subscans (red: first subscan, black: second subscan), with a 9.6 degrees derotator angle, and a slight shift from one subscan to another to fill the gaps. These 2 scans cover the (0,0) reference position.

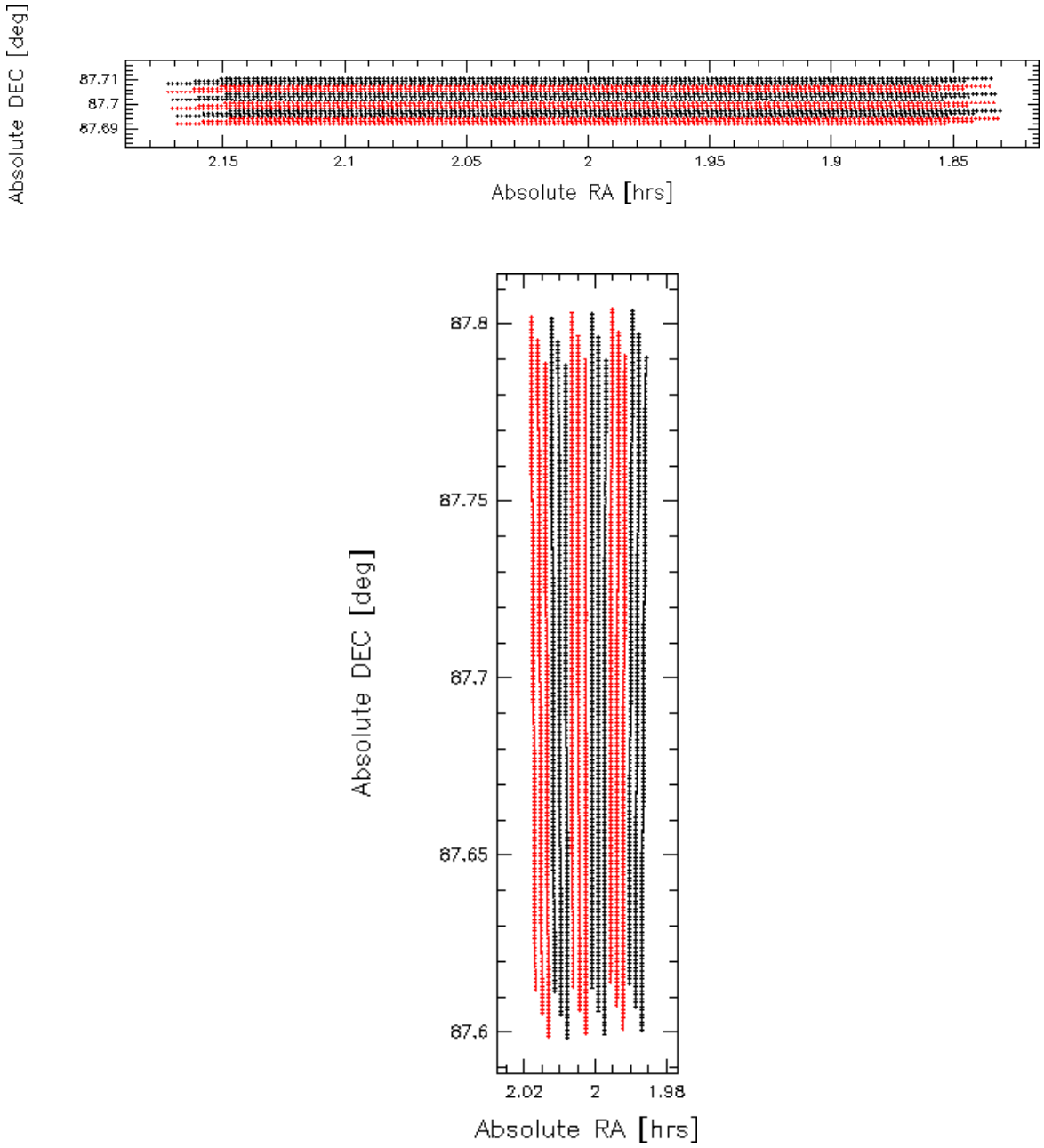

Figure 3.9: Same as Fig.3.8 (scans near the projection center) but showing **absolute coordinates**. In this case, the uniform mapping of the projected map results in uniform mapping of the sky.

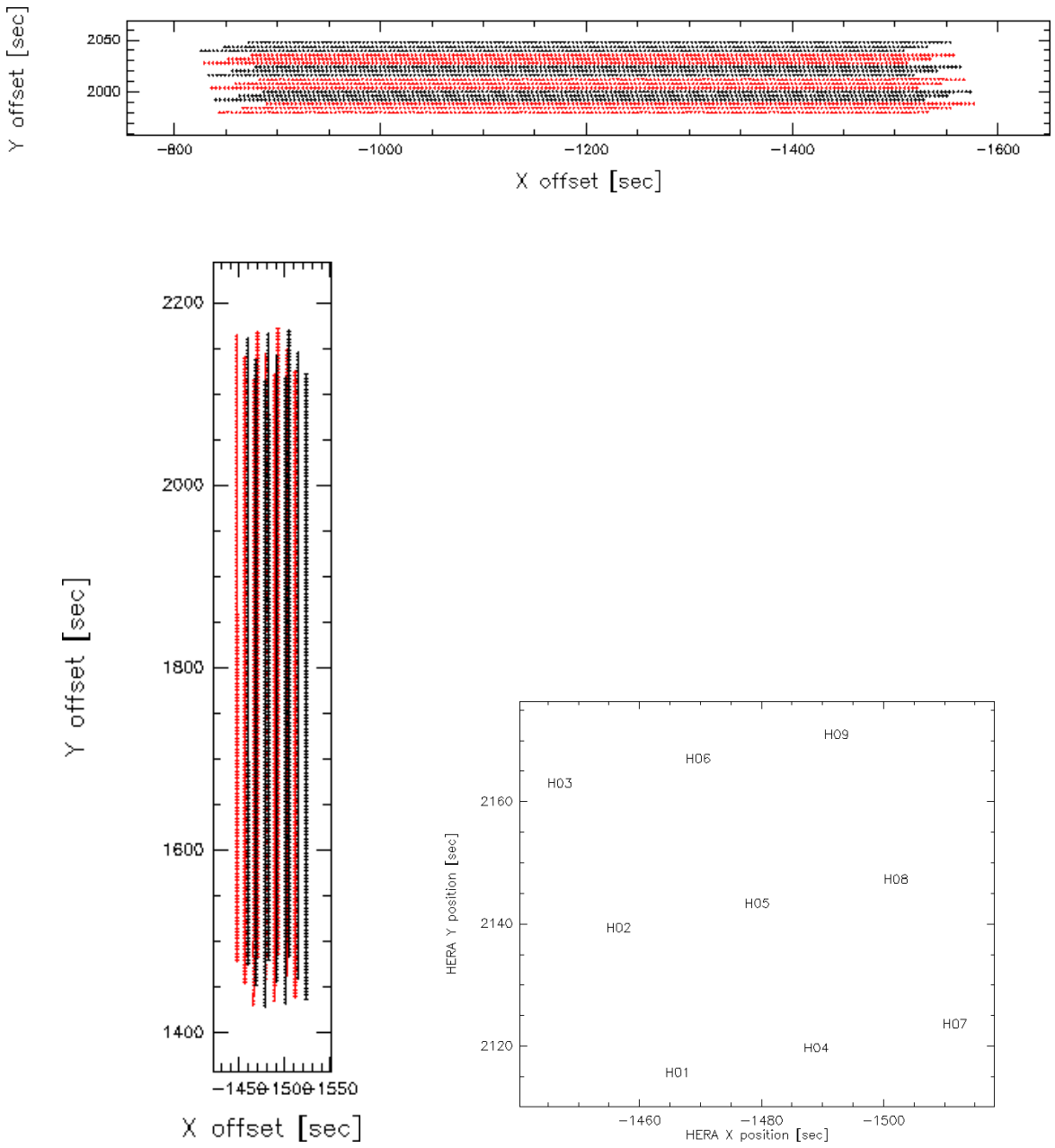

Figure 3.10: Same as Fig.3.8 but showing scans 98 (top) and 115 (bottom) **relative coordinates (radio projection)**. The same observing strategy is used. These 2 scans cover the (-1500,2000) offset position, far from the reference position. Note parallelogram shape of the receiver array in the projected map, and as a consequence the unexpected coverage of the vertical scan.

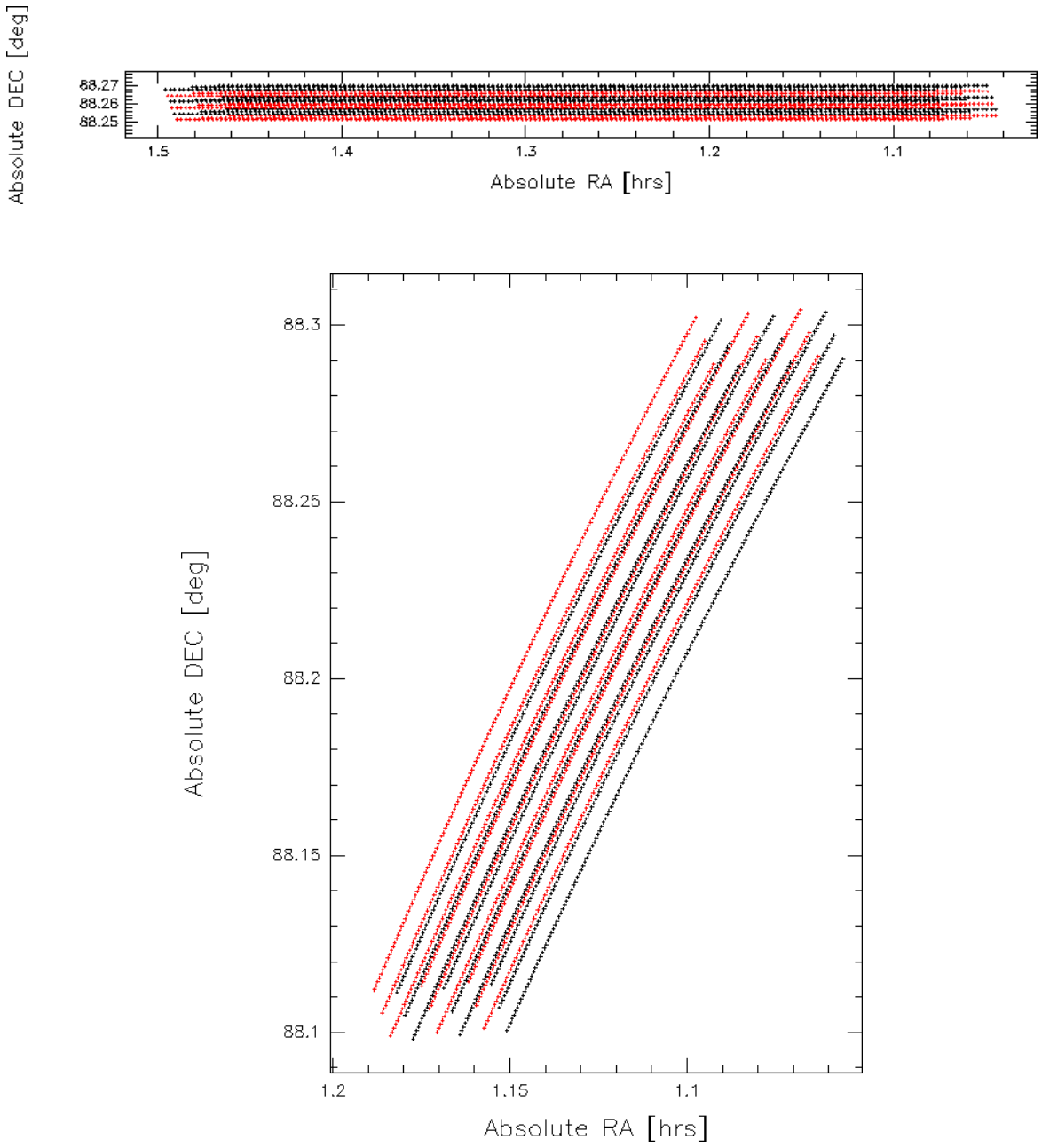

Figure 3.11: Same as Fig.3.10 (scans far from the projection center) but showing **absolute coordinates**. In this case, the attempt of uniform mapping of the projected map results in unexpected mapping of the sky. In particular, the 9.6 degrees derotator angle on the sky is obviously not suited here.

3.5 Guessing ON and OFF positions

Depending on the observing mode, the telescope is expected to observe 1 or 2 positions on the sky:

- Position switch: 2 positions (ON + OFF)
- Wobbler switch: 1 or 2 positions
- Frequency switch: 1 position

Those positions are not described directly in the IMB-FITS. Instead, the positions of each dump is available in the `LONGOFF` and `LATOFF` columns of the antenna slow table. By using equivalence classes of the X and Y offset pairs, it should straightforward to group the dumps by position.

For `projection` offset system, *i.e.* when offsets are described along a sky system of coordinates (equatorial or galactic), this approach is correct: the offsets are all strictly identical for a given position and for both coordinates. All digits are the same, there are no numerical round-off errors.

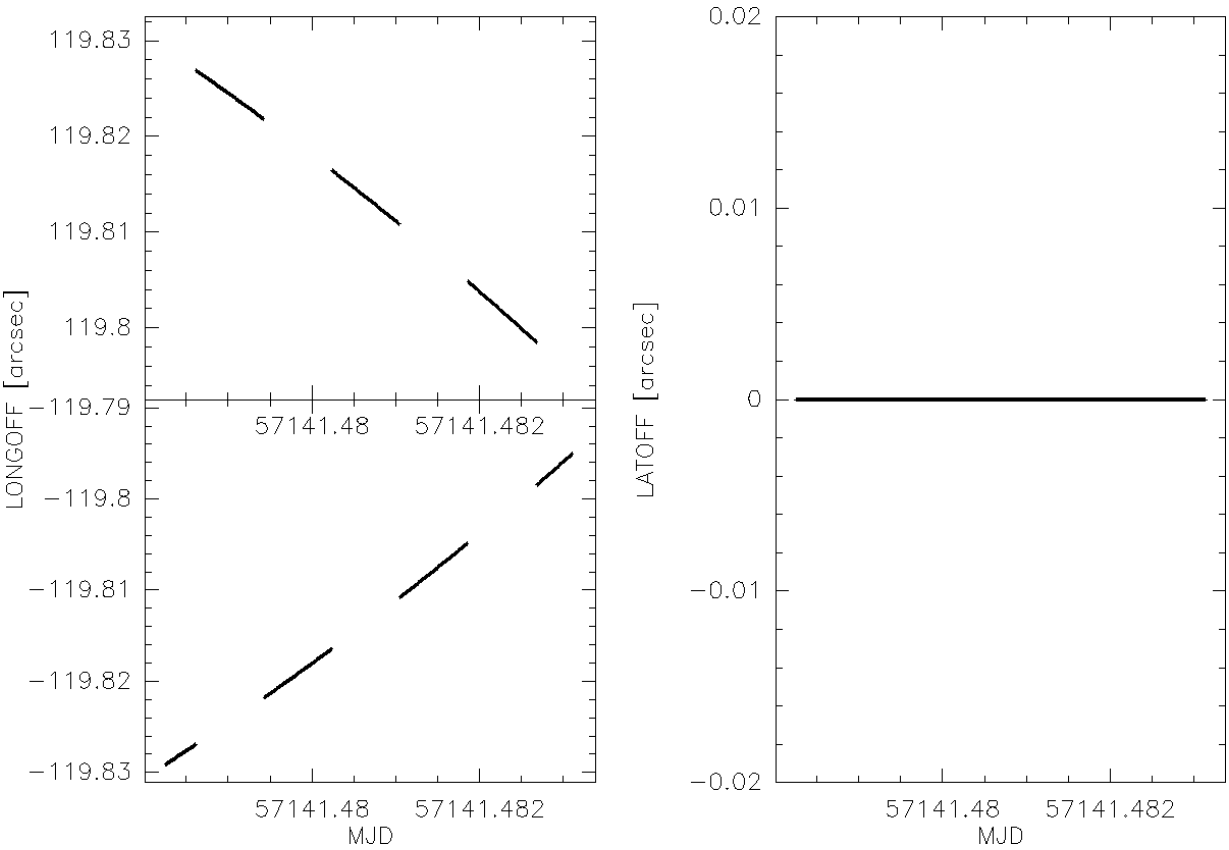

Figure 3.12: `LONGOFF` and `LATOFF` vs `MJD` for iram30m-fts-20150429s199-imb.fits (whole scan). Observing mode is tracked wobbler switch, using `horizontalTrue` offset system. During the scan, elevation changes from 82.3 to 83.6 degrees.

On the other hand, for `horizontalTrue` offset system, the offsets drift. In the `horizontalTrue` offset system, `LONGOFF` are actually offsets along azimuth, corrected by

$cos(elevation)$, and `LATOFF` are offsets along elevation. If we look at the example in Fig. 3.12, we can see that:

- `LATOFF` offsets are all null: this means that all the dumps were sharing the same elevation as the reference (source) position. The telescope is tracking the source, so that the elevation changes (this can be seen in the `CELEVATIO` column), but not the offset in this direction.
- `LONGOFF` offsets are drifting, slowly decreasing for both positions.

Because of this drift, **MRTCAL** uses a centi-arcsecond tolerance when the equivalence classes are computed. If we reuse the example in Fig. 3.12, this tolerance is still not enough and results in 6 different positions:

```
E-SUBSCAN>LIST>BUILD,  Scan has 6 positions (expected 1 or 2):
E-SUBSCAN>LIST>BUILD,     Position #1: -119.82903552  0.00000000
E-SUBSCAN>LIST>BUILD,     Position #2:  119.82643276  0.00000000
E-SUBSCAN>LIST>BUILD,     Position #3: -119.81881083  0.00000000
E-SUBSCAN>LIST>BUILD,     Position #4:  119.81592996  0.00000000
E-SUBSCAN>LIST>BUILD,     Position #5: -119.80748135  0.00000000
E-SUBSCAN>LIST>BUILD,     Position #6:  119.80419607  0.00000000
```

3.6 Azimuths and `horizontalTrue` system

As the 30m telescope has an alt-azimuth mount, the equatorial (or galactic) astronomical coordinates are converted to the azimuth-elevation system of coordinates to (or in) the mecanichal system, and the encoders return the actual telescope position in this system (*e.g.* actual position, tracking errors, etc). However, because this system is spherical, azimuths are not suited to measure distances on the sky. At the interface of the user and the telescope, the `horizontalTrue` system is defined to address this issue: `horizontalTrue` offsets in **IMBFITS** files (when duly indicated) and `trueHorizon` values in some **PAKO** options.

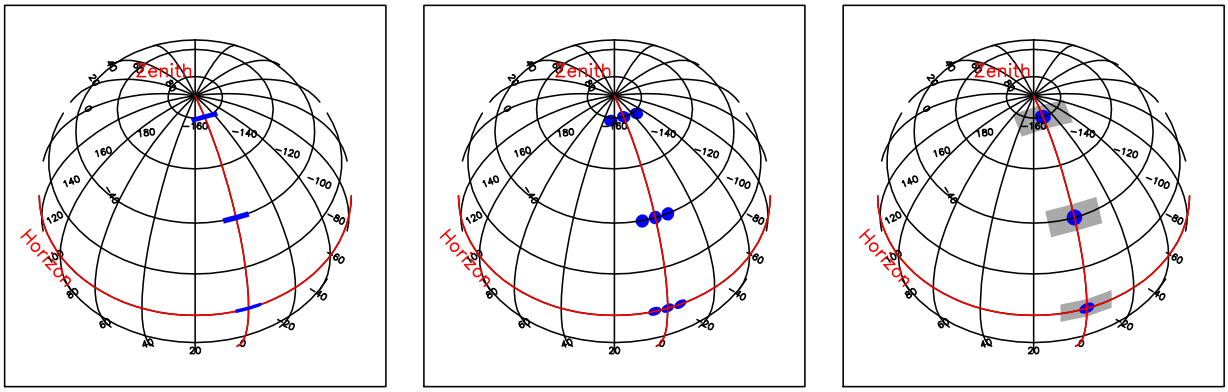

Figure 3.13: Illustration of azimuth-elevation effects on coordinates and distances. All plots show the celestial sphere with the horizon (0° elevation) and an arbitrary azimuth of observations in red. We display here the effect at 0°, 40°, and 80° elevations. **Left:** a 10° arc along azimuth is represented at the three elevations. **Center:** three sources are tracked in wobbler-switching mode at the same azimuth but different elevations, with their pair of off-positions apart. **Right:** the same map (same size, in grey) is observed at the same azimuth but different elevations, with an illustration of the telescope jittering at its center.

The Figure 3.13 illustrates the effects on the coordinates we can find in the **IMBFITS** files.

- Left: the same arc (10° `horizontalTrue` here) on the sky covers different ranges of azimuth coordinates from horizon to zenith ($10°/cos(\text{el})$).
- Center: for wobbler-switching scans, the secondary mirror wobbles along the azimuth, resulting in offsets about $4'$ apart from the source; their corresponding azimuths depend on the elevation.
- Right: the tracking errors of the main dish result as jittering around the desired position. See subsection 3.6.1 for details.

3.6.1 Tracking errors

The telescope encoders return values in the azimuth-elevation system, so the `TRACKING_AZ` values from the **IMBFITS** tables must be multiplied by $cos(\text{el})$ in order to have a reliable idea of the telescope behavior on the sky. The Figure 3.14 shows[8] three real examples for three different scans. While the tracking errors seem comparable in both directions at low elevation, the dispersion in

[8]Note that, by design, when the `TRACKING_AZ` values are converted with `GREG\CONVERT` from absolute spherical coordinates to offsets projected on the plot, the $cos(\text{el})$ factor is applied.

azimuth is completely flatenned near the pole: the same jittering of the telescope along the azimuth axis results in different jittering on sky, depending on the elevation ($cos(86^\circ) \simeq 0.07$). Question: no wind effect near zenith??

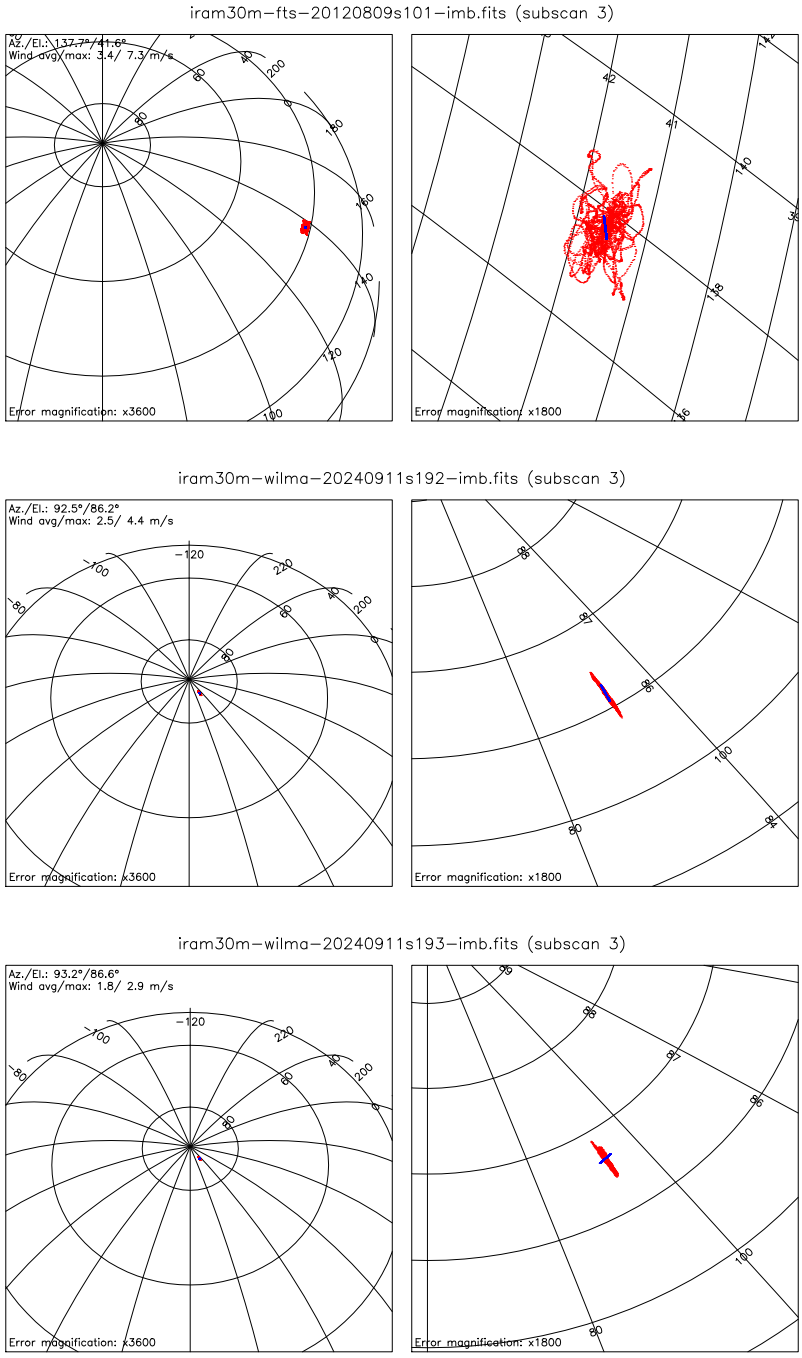

Figure 3.14: Antenna position (blue) and tracking errors (red) of three drift subscans from three different on-the-fly scans, with two zoom factors for each. The tracking errors are displayed from a unique *central* position in order to visualize the dispersion in both directions, and a magnification factor is applied for better visibility. All plots show the azimuth-elevation sphere under an arbitrary orientation and the associated grid. **Top:** On-the-fly scan from the **MRTCAL** test data, at about 40° elevation. **Center:** On-the-fly scan near the zenith (86°); subscan drifts along right-ascension, nearly equivalent to elevation scanning for the source and date-time of observation. **Bottom:** Same, but scanning along declination, nearly equivalent to azimuth scanning.

3.6.2 `CAZIMUTH` bug in WSW IMBFITS?

The `horizontalTrue` system was introduced to properly measure offsets (distances) on the sky, as the difference in azimuth coordinates is not the actual distance along azimuth. On the other hand, absolute azimuth-elevation coordinates are what they are, with no particular issue.

However, it appears that the `CAZIMUTH` (*commanded azimuth*) found in the **IMBFITS** files is incorrect with respect to this definition. Fig. 3.15 shows an example where `CAZIMUTH` is 105° and `AZIMUTH` is 154° for a 47° elevation at start of the scan, noting that $154 \times cos(47) \simeq 105$. This is even more surprising at this feature is only (and always) observed in wobbler-switch **IMBFITS** files, inconsistently with the other observing modes.

Note that **MRTCAL** uses `CAZIMUTH` and `CELEVATIO` 1) to give a typical `(az,el)` position of each IMB-FITS at indexing time, and 2) to give a `(r%head%gen%az,r%head%gen%el)` position for each CLASS spectrum it creates. CLASS relies on commanded values, not the actual from antenna-fast tables.

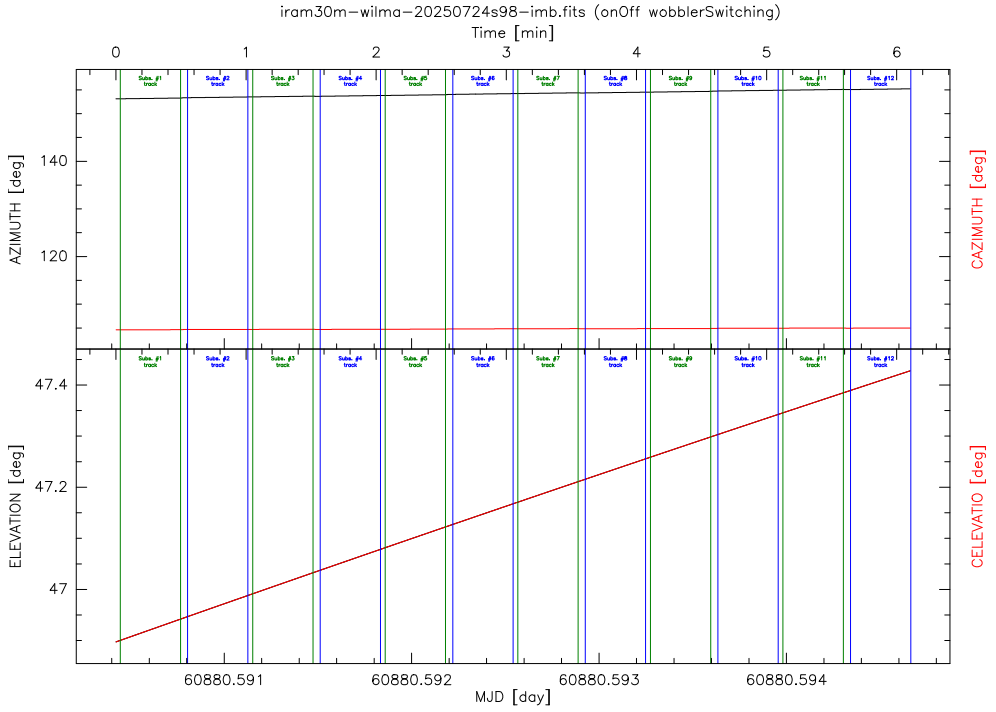

Figure 3.15: Commanded (red) and Encoder (black) azimuths and elevations as found in the antenna slow and fast traces respectivly in the named **IMBFITS** file.

Chapter 4

Interpolation of calibration products

MRTCAL computes one set of calibration products *per chunk*. These products are:

- the receiver temperature,
- the calibration temperature,
- the system temperature,
- the precipitable water vapor,
- and the zenithal opacity.

Their exact values are evaluated for the frequency at the middle of the chunk[1]. The values at other frequencies in the chunk can be evaluated using one of the two available modes:

1. in the *flat* mode, the central values are reused over all the chunk bandwidth.
2. the second mode is *interpolation*. In this case, the chunks are reordered by ascending frequency, and the values at non-central frequencies are interpolated between the current chunk and its nearest neighbour, so that it is computed at the frequency of the desired channel. This has the advantage to avoid discontinuities along the whole bandwidth and thus to avoid calibration platforming. For the two half-chunks at the boundary of the whole bandwidth, where no neighbour chunk is available, the values are extrapolated from the boundary chunk and its neighbour on the other side.

[1] Remember that the chunk bandwidth is ruled by `MSET CALIB BANDWIDTH`

Chapter 5

Building and solving pointing drifts

The figure 5.1 shows that for a given phase cycle, **MRTCAL** computes the average MJD of each of the 4 phases, and then the average of the ONs and the average of the OFFs, and uses the ON average MJD to associate the proper antenna position. The following list describes the main steps of the algorithm, which is currently implemented in the subroutine `mrtcal_solve_pointing_bsw`.

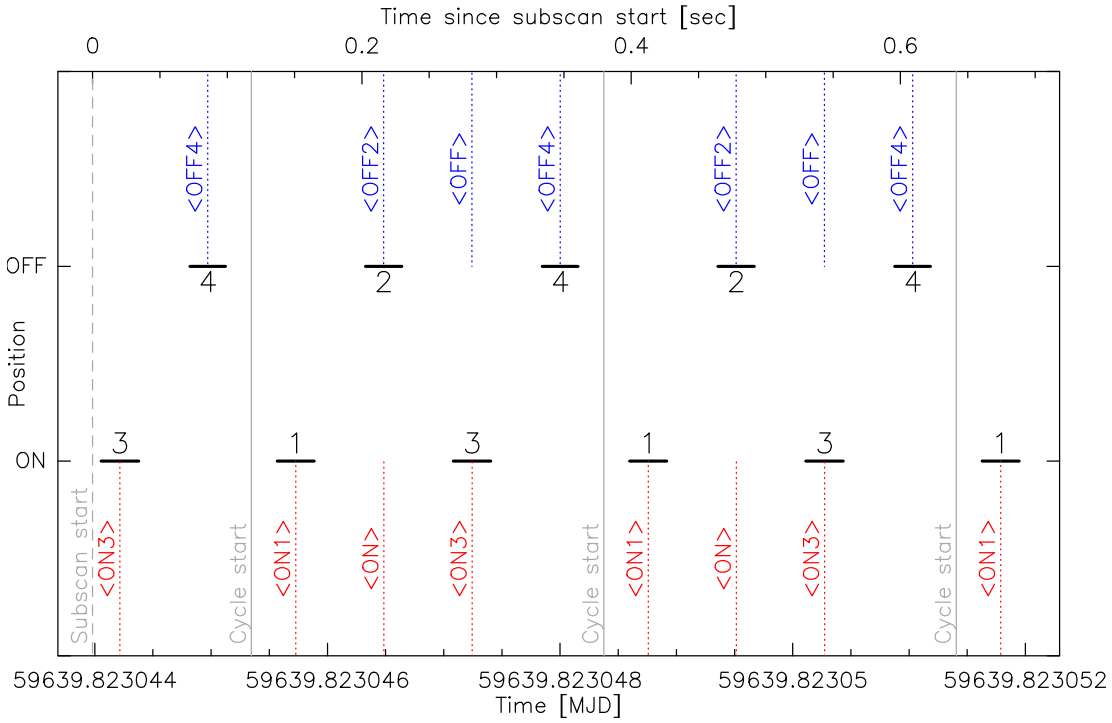

Figure 5.1: Representation of the time distribution of the first phase dumps for iram30m-nbc-20220301s205-imb.fits (subscan 1). The horizontal segments labeled from 1 to 4 represent the integration time of each phase.

1. Major loop on all the subscans:

   (a) Sub-loop on all dump cycles in a subscan:

      i. Fetching the next cycle: subroutine `mrtcal_get_next_dumpcycle` calls subroutines `mrtcal_find_next_dumpcycle` and `mrtcal_read_next_dumpcycle`. A cycle

consists typically in the dump sequence 1=ON, 2=OFF, 3=ON, 4=OFF.

ii. `mrtcal_read_next_dumpcycle` calls a cascade down to `mrtcal_read_subscan_data`

iii. This also also includes chunkset mapping: as for the other observing modes, one dump from the `IMBF-backendCONT` table is mapped as a single chunk (`chunk_t`). Since continuum backends are used, it provides only 1 intensity value (1 channel). This chunk is attributed one MJD at the center of the dump integration time, and from this MJD the chunk position is interpolated from the antslow table (`chunk%offset(1)` and `chunk%offset(2)`). See subroutine `mrtcal_chunksets_from_data_1time1pix_1chunk`.

iv. After the reading, the subroutine `mrtcal_get_next_dumpcycle`, averages the ONs ($\langle ON \rangle = 1 + 3$) and OFFs ($\langle OFF \rangle = 2 + 4$) in the subroutine `average_phases`. It uses `mrtcal_chunkset_2d_accumulate_init` and `mrtcal_chunkset_2d_accumulate_do`, ultimately calling `mrtcal_chunkset_accumulate_do` for weighted average of MJD and weighted average of offsets (weight is integration time of each phase). In return, $\langle ON \rangle$ has average MJD and offsets. Same for $\langle OFF \rangle$.

v. Computing $\langle ON \rangle - \langle OFF \rangle$: i) call `mrtcal_on_minus_off` for one dump cycle in mode `isotfmap`, ii) subroutine `mrtcal_on_minus_off_head` keeps the $\langle ON \rangle$ MJD and offsets.

(b) Loop on chunksets to create one drift per chunkset (subroutine `mrtcal_pointing_create`). Each chunkset is a collection of chunks computed at the previous steps, *i.e.* a collection of $\langle ON \rangle - \langle OFF \rangle$ with proper MJD and offsets. Gather them as a continuum drift with irregular X axis in a CLASS `type(observation)` in subroutine `mrtcal_chunkset_to_obs_con`. The operation is repeated for all chunksets *i.e.* all subscans and all backend parts (H and V, etc) (`backsci%drift%indiv`).

2. Classify the drifts (subroutine `mrtcal_pointing_classify`) as desired (`MSET SOLVE POINTING`): if there is more than 1 drift per class, resample the irregular X axis to regular and average the drifts (subroutine `mrtcal_solve_pointing_gather_regular`) also in a `type(observation)` (`backsci%drift%solved`).

3. For each class (subroutine `mrtcal_solve_and_write_obslist`):

   (a) Solve each class with a gaussian+baseline model (subroutine `mrtcal_solve_pointing_observation`) calling TELCAL `solve_pointing`. Convert TELCAL solution to CLASS pointing section (subroutine `mrtcal_fit_to_obs_poi`).

   (b) Remove the fitted baseline from each class `RY` arrays and save the original `RY` as an associated array named `POINTING` (subroutine `mrtcal_pointing_associate_array`).

   (c) Write to the output CLASS file (subroutine `mrtcal_obs_to_class`).

4. Save all the pointing results of the scan to the dedicated **MRTCAL** section in the index (subroutine `mrtcal_entry_sdrifts2poisec`).

5. Final feedback: i) ASCII table to screen or file (subroutine `mrtcal_solve_pointing_user_feedback`), and ii) XML file (subroutine `pointing_to_VO`).

Chapter 6

Memory index

6.1 Strategy

MRTCAL indexes **IMBFITS** files in the so-called *index files*. A raw index file named `index.mrt` can be produced with the command `INDEX BUILD` (see command help for more advanced uses). When one or more index files are then reopened for reading, the command `INDEX OPEN` builds a *memory index* named the "Input indeX" (IX). At this stage, IX gathers the summary of all entries in all opened index files. From this full set, the command `MFIND` can be used to make a selection of desired entries: This is the "Current indeX" (CX).

While the primary purpose of the index files is to reference **IMBFITS** files, their other advantage is to store other *products* that can be derived from the **IMBFITS** files. In particular, after the calibration, the commands `CALIBRATE` and `PIPELINE` will save new *versions* of each calibrated entry. These new versions have a modified calibration status (from *none* to *failed*, *empty*, or *done*), and optionally a calibration section added (storing the calibration results). These modified versions of the entries are themselves saved in an index file (either the original one, or a new one, depending on the user choice), AND are implicitly appended to IX. After the calibration process, it is then easy for the user to select from IX the calibrated entries (or failed, or pending, etc). The figure 6.1 shows a basic example of these steps.

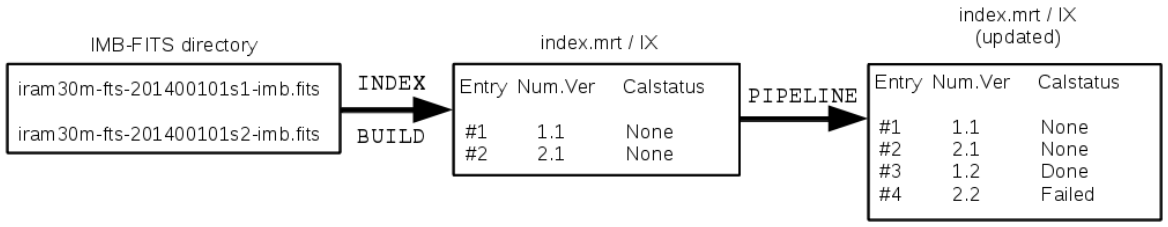

Figure 6.1: Basic example showing how the **IMBFITS** files are first indexed, and then how the calibration command (here, `PIPELINE`) adds a new version of each (tentatively) calibrated **IMBFITS**.

6.2 Contents

When the index file is loaded, its index content is duplicated in memory for all the entries. These elements are described in the Table 6.1.

Table 6.1: Memory index elements coming directly from the index file.

Parameter	Unit	Comment
`bloc`	records	Entry position in index file
`word`	words	Entry position its this record
`version`	—	Entry version
`telescope`	code	Telescope code
`projid`	—	Project id
`source`	—	Source name
`dobs`	gag_date	Observation date
`ut`	rad	Observation time
`lst`	sec	Local Sideral Time
`az`	rad	Azimuth
`el`	rad	Elevation
`frontend`	—	Receiver names
`scan`	—	Scan number
`backend`	code	Backend code
`obstype`	code	Observation type
`switchmode`	code	Switching mode
`filstatus`	code	Can read file or not?
`calstatus`	code	Calibration status
`filename`	—	**IMBFITS** file (no path)
`itime`	nanosec	Last indexing time (from 01-jan-1970)

However, these elements alone are not enough to deal with each **IMBFITS** file referenced in the index. Extra numbers are associated on-the-fly, at load time. The Table 6.2 summarizes those numbers. In details, here is the exact meaning of those numbers.

`num` The *observation number* is a unique number associated to each **IMBFITS** file. Uniqueness is ensured by a unique combination of the observing date, the scan number, and the backend. In a memory index, there can be several `version` of the same observation, meaning that the same **IMBFITS** file is indexed several times (*e.g.*, a first time when the index is built, a second time after the calibration). This number (and optionaly its version) is intended to be used as a frontend to the end-user. However, it is volatile: It can change from one session (or one INDEX OPEN) to another, depending on the index contents, and their number and order if several indexes are loaded in memory. In practice, the observation numbers are contiguous, starting from 1, and ordered by observing date, then by scan number, then by

Table 6.2: Memory-only index elements associated to each entry

Parameter	Unit	Comment
`num`	—	Observation number
`mnum`	—	Entry position in the Input indeX (IX)
`fnum`	—	Entry position in the index file
`idir`	code	Associated index file and **IMBFITS** directory
`sort`	—	Sorting array

backend code.

`mnum` The *memory number* is the entry position in the input index (IX), and thus it provides a unique identifier for each entry indexed in memory. This means that the array `ix%mnum(:)` runs contiguously from 1. Since IX is the absolute reference for all sub-indexes (*e.g.*, the current index CX), each entry of any index can easily be identified thanks to this back pointer (*e.g.*, `cx%mnum(ient)`).

`fnum` The *file number* is the entry position in its associated index file. This number can not be implicit, because several index files can be loaded in memory, and because the entries are reordered by date, scan and backend at load time.

`idir` the *directory code* identifies each index file loaded in memory, so that each entry knows easily to which index file it comes from. This code also identifies the directory the **IMBFITS** can be found in (remember the index file may not be hosted in the same directory; this is controled by the user through the command `INDEX /FILE`).

`sort` The *sorting array* is an indirection array which would order the memory index by date, by scan, and by backend. At load time, the entries are actually ordered in memory, so this array is not needed at this time. But later on, commands like `CALIBRATE` or `PIPELINE` can add new versions of the entries. These new entries are appended at the end of the memory indexes, in order to avoid breaking the cross references between indexes (in particular, the `cx%mnum(:)` back pointers to `ix`). Each application which needs to access the entries sorted by date, scan, and backend, should then use the sorting array. Typically, one has to loop on `cx%mnum(cx%sort(ient))` (with `ient` 1 to `cx%next-1`) to access the entries ordered correctly.

Chapter 7

Scan date vs observation date: Midnight issue

7.1 In MRTCAL index files

When a new scan is *prepared*[1] at the telescope, the associated **IMBFITS** file name is constructed with the current date. Later in this process (up to several minutes after), the scan is actually *started*[2], and the current date and time are used to build the `MJD-OBS` and `DATE-OBS` in the FITS Primary header. If those two steps are performed exactly before and after midnight respectively, the file name and the Primary header refer to 2 different dates, e.g.

```
iram30m-fts-20120809s323-imb.fits:
  MJD-OBS =     56149.0038078704 / MJD at observation start
  DATE-OBS= '2012-08-10T00:05:29.000'
```

For several purposes, **MRTCAL** saves the *observing date and time* of each file in the index. In order to deal correctly with the above issue, it is decided

1. to save the date as found in the file name, so that there is no difference for the user between the file name and the date exposed by **MRTCAL**;

2. the time value saved for this file refers to the above date.

The direct consequence of those 2 rules is that, in case of the midnight issue exposed above, the UT value will be larger than 24h. Note that the UT value associated to each file is saved for bookkeeping purpose, e.g. for analysis of the calibration though date and time. It is considered as a typical value for the whole scan. Each individual spectrum produced by **MRTCAL** uses its own date and time.

In details, the observing date saved in the index is used

- to check the entry uniqueness (date, scan and backend triplet should be unique),

- to sort the entries (by date, scan and backend),

- through the command `MFIND /DATE`,

[1] *scan prepared* or *scan loaded* in the NCS nomenclature.
[2] *scan started* or *subscan 1 started* in the NCS nomenclature.

- in the output of the command `MLIST`,
- in the Sic variables `MDX%DOBS` and `MHEAD%KEY%DOBS`

and the associated UT value is used

- to sort the entries,
- in the output of the command `MLIST`,
- in the Sic variables `MDX%UT` and `MHEAD%KEY%UT`

The user should be ready to encounter UT values larger than 24h in the `MLIST` output and larger than 2π in the Sic variables.

7.2 In CLASS spectrum header

In the CLASS spectrum header, the date and time of observation of the spectrum is saved in the `r%head%gen%dobs` and `r%head%gen%ut` components, encoded respectively as an integer *GAG* date in days and a fraction of day in radians. If a scan overlaps midnight, this means that some spectra produced from the corresponding IMB-FITS refer to one day, and some other ones refer to the day after. However, this also means that the later spectra refer to a day which do not correspond to the scan number. In other words:

- these spectra would NOT be found when using `FIND /OBSERVED date /SCAN number` as only the spectra before midnight have the proper date,
- these spectra would be found when using `FIND /OBSERVED date+1 /SCAN number` while they were NOT observed during the named scan of the subsequent day.

In order to avoid this issue, the solution similar to the **MRTCAL** index is choosen. All the spectra produced from one scan will use the date associated to this scan, even if this scan overlaps midnight. In this latter case, the `UT` value will extend beyond 24 hours (2π) to properly keep track of the date and time of the spectrum.

Chapter 8

Patching IMB-FITS at read time

MRTCAL is divided in 2 main libraries:

1. the libimbfits reads the IMB-FITS files and fills a memory structure from it,
2. the libmrtcal gets the libimbfits memory structure, and use it to index the files and calibrate them.

When **MRTCAL** has to deal with several versions of the IMB-FITS (e.g. V1.35 for HERA and V2 for EMIR), the variations are *hidden* in the libimfits so that the libmrtcal does not have to worry about those *details*. This means that the memory structure is unique while the disk structure differ[1]. There are also other cases when we want to fill memory structures that differ from the file on disk. All this is described hereafter.

Note that each patch is saved as a warning in the comment field of each element. Those warning are visible when using the command `MDUMP`.

8.1 Added elements when missing

The purpose of adding elements in the HDUs when they are missing is that the libmrtcal does not have to worry about this element existing or not, and and doing different things (different code) in one case or the other. The following elements are added:

- `POLEX` and `POLEY` are absent under V1.35: they are defined and default to 0,
- `SYSOFF`, `XOFFSET`, and `YOFFSET` columns in the Scan table can have 0, 1, or 2 rows. They are forced to 2 rows providing *Nasmyth* and *projection* offsets (default to 0 if they were absent),
- the `LINENAME` column in the BackEnd HDU is built from the `LINENAME` column in the FrontEnd HDU for IMB-FITS version lower than 2, or for 4MHz backend,
- `MJD_BEG` and `MJD_END` keywords are added as convenient duplicates of `DATE-OBS` and `DATE-END` keywords in all HDUs of each subscan. They offer MJD values instead of ISO strings,

[1]Note that this strategy is in use since a long time in GILDAS to ensure the backward compatibility of the various formats.

- the column TSTAMPED is added to BackendDATA tables in IMB-FITS version lower or equal to 1.35. Its defaults to 1. Remember that this value is sensitive (it introduces a shift on the time stamps) and is actually backend-dependent.
- an IFRONT column is added as a convenient backpointer from the BackEnd table to the FrontEnd table, *i.e.* each chunk knows easily to which receiver it is associated.

8.2 Modified elements

- IFCENTER, FRQOFF1, and FRQOFF2 columns from the FrontEnd table are reversed for each row where IFFLIPS is true. This factorizes the sign issues when dealing with IF2 instead ot IF1,
- The column BAND is read under the name PART for IMB-FITS version lower than 2. Same for REC1 and REC2 read under the names RECEIVER and BAND respectively,
- REFCHAN is patched to correct values (instead of 1) when dealing with continuum backend (1 channel per chunk),
- the columns MJD, INTEGTIM and ISWITCH are compressed if MSET CALIBRATION BAD is NO (this is the default). In this case all rows where ISWITCH is 0 are removed. Two new columns FOREPOIN and BACKPOIN are added as cross pointers between the original and compressed columns. See section 2.2 for details.
- the DATE-OBS and DATE-END stamps of each subscan are patched if MSET CALIBRATION MJDINTER is YES. See HELP for details.

Chapter 9

IMB-FITS version 2.13

For single pixel receiver, the IMB-FITS data format provides the following FITS extensions:

0. `Primary`: general description of the observation
1. `IMBF-scan`: scan description (source, antenna offsets, etc)
2. `IMBF-frontend`: frontend description (which receiver was connected, etc)
3. `IMBF-backend`: backend configuration i.e. how the chunks translate to sky frequency.

Then, the 3 following extensions are repeated for each subscan (example with FTS backend):

4. `IMBF-backendFTS`: provides the data dumps through time,
5. `IMBF-antenna`: provides the antenna position through time at slow and fast rates.
6. `IMBF-subreflector`: not used by **MRTCAL**.

Each of these extensions are detailed below from an example scan, showing the output of the command `MDUMP`[1]

9.1 Primary

```
SIMPLE   (L)  =                     T / file does conform to FITS standard
BITPIX   (I4) =                    32 / number of bits per data pixel
NAXIS    (I4) =                     0 / number of data axes
EXTEND   (L)  =                     T / FITS dataset may contain extensions
TELESCOP (C)  = IRAM 30m              / Telescope Name
ORIGIN   (C)  = IRAM                  / Organisation or Institution
CREATOR  (C)  = Python IRAM MBFITS Wri / Software
IMBFTSVE (R8) =   2.13000000000000    / IMBFITS version
INSTRUME (C)  = fts                   / Backend
OBJECT   (C)  = XXXXXXXX              / Source Name
LONGOBJ  (R8) =   00.0000000000000    / [deg] Source longitude in basis frame
LATOBJ   (R8) =   00.0000000000000    / [deg] Source latitude in basis frame
TIMESYS  (C)  = UTC                   / time system (TT,TAI,UTC ...)
MJD-OBS  (R8) =   57841.4547916667    / MJD at observation start
```

[1] `MDUMP` slightly modifies the header elements at read time for convenience. Some remarks are added accordingly in the output.

```
DATE-OBS (C)  = 2017-03-29T10:54:54.00 /
LST      (R8) =   83367.1998821900     / [s] Local apparent sidereal time (scan start)
PROJID   (C)  = 102-16                 / project ID
QUEUE    (C)  = 102-16                 / Observing Queue
EXPTIME  (R8) =   15.0000000000000     / Total netto integration time [s]
N_OBS    (I4) =                      3 / No of subscans defined
N_OBSP   (I4) =                      3 / No of subscans found
OBSTYPE  (C)  = calibrate              /
NUSEFEED (I4) =                      8 /
TOTANT   (I4) =                     23 / Total no. of LINES in antenna tables
TOTSUBR  (I4) =                     59 / Total no. of LINES in secondary tables
TOTBACK  (I4) =                     33 / Total no. of LINES in backenddata tables
```

9.2 IMBF-scan

```
XTENSION (C)  = BINTABLE               / binary table extension
BITPIX   (I4) =                      8 / 8-bit bytes
NAXIS    (I4) =                      2 / 2-dimensional binary table
NAXIS1   (I4) =                     23 / width of table in bytes
NAXIS2   (I4) =                      2 / number of rows in table
PCOUNT   (I4) =                      0 / size of special data area
GCOUNT   (I4) =                      1 / one data group (required keyword)
TFIELDS  (I4) =                      3 / number of fields in each row
EXTNAME  (C)  = IMBF-scan              / name of this binary table extension
TELESCOP (C)  = IRAM 30m               / Telescope Name
TELSIZE  (R8) =   30.0000000000000     / [m] Telescope Diameter
SITELONG (R8) =  -3.39875641986850     / [deg] observatory longitude (EAST)
SITELAT  (R8) =   37.0684132670517     / [deg] observatory latitude (NORTH)
SITEELEV (R8) =   2851.50000000000     / [m] observatory elevation
PROJID   (C)  = 102-16                 / Project ID
OBSID    (C)  = XX                     / Observer initials
OPERATOR (C)  = Pako                   / Operator initials
SCANNUM  (I4) =                    138 / Scan number
DATE-OBS (C)  = 2017-03-29T10:54:54.00 / scan start in TIMESYS system
DATE     (C)  = 2017-03-29T10:55:18    / Date of FITS file creation
MJD      (R8) =   57841.4547916667     / [day] Scan date/time (Modified Julian Date)
LST      (R8) =   83367.1998821900     / [s] Local apparent sidereal time (scan start)
N_OBS    (I4) =                      3 / Number of observations in this scan
EXPTIME  (I4) =                     15 / Total netto integration time [s]
TIMESYS  (C)  = UTC                    / time system (TT, TAI, UTC ...)
UT1UTC   (R8) =  0.477150000000000     / [s] UT1-UTC time translation
TAIUTC   (R8) =   35.0000000000000     / [s] TAI-UTC time translation
ETUTC    (R8) =   67.1840000000000     / [s] Ephemeris Time - UTC time translation
GPSTAI   (R8) =   0.00000000000000     / [s] GPS time - TAI translation
POLEX    (R8) =  2.714956614210000E-08 / [rad] Celestial Pole Offset X
POLEY    (R8) =  1.781205464400000E-06 / [rad] Celestial Pole Offset Y
OBJECT   (C)  = XXXXXXXX               / Source name
CTYPE1   (C)  = RA-SFL                 / Basis system (longitude) --
CTYPE2   (C)  = DEC-SFL                / Basis system (latitude) -- XLAT-SFL
RADESYS  (C)  =                        / additional system definition for ecliptic/equat
EQUINOX  (R8) =   2000.00000000000     / [Julian yrs] Equinox
CRVAL1   (R8) =   0.00000000000000     / [deg] Native frame zero in basis system (long.)
CRVAL2   (R8) =   0.00000000000000     / [deg] Native frame zero in basis system (lat.)
```

```
LONPOLE  (R8) =   0.00000000000000     / [deg] Native longitude of celestial pole: range
LATPOLE  (R8) =   0.00000000000000     / [deg] Basis latitude of native pole
LONGOBJ  (R8) =   00.0000000000000     / [deg] Source longitude in basis frame
LATOBJ   (R8) =   00.0000000000000     / [deg] Source latitude in basis frame
SWTCHMOD (C)  = totalPower             / Switch mode
NOSWITCH (I4) =                      1 / no. of switch phases in a switch cycle
PHASETIM (R8) =  0.500000000000000     / [s] Integration time per phase
WOBTHROW (R8) =   0.00000000000000     / [deg] wobbler throw
WOBDIR   (I4) =                      0 / wobbler throw direction
WOBCYCLE (R8) =   0.00000000000000     / [s] wobbler period
WOBMODE  (C)  =                        / wobbler mode (SQUARE/TRIANGULAR})
NFEBE    (I4) =                      8 / \nfebe\ number of FEBEs
PRESSURE (R8) =   728.500000000000     / [hPa] Atmospheric pressure (hPa)
TAMBIENT (R8) =  -3.00000000000000     / [deg C] Outside temperature (C)
HUMIDITY (R8) =   36.6000000000000     / [%] Relative Humidity (%)
WINDDIR  (R8) =   187.300000000000     / [deg] Wind direction (deg)
WINDVEL  (R8) =   1.50000000000000     / [m/s] Wind velocity (m/s)
WINDVELM (R8) =   2.50000000000000     / [m/s] Wind max. velocity (m/s)
DATE-WEA (C)  = 2017-03-29T10:55:15.58 / Time of weather station par.
REFRACTI (R8) =   0.00000000000000     / [none] Refraction Correction, as a function of
THOT     (R8) =   295.150000000000     / [K] Hot Load Temperature in K
DATE-HOT (C)  = 2017-03-29T10:54:44.75 / Time of Hot Load Temp. meas.
TIPTAUZ  (R8) =  0.136721000000000     / [ ] Tau meter zenith opacity
TIPTAUC  (R8) =  0.999226000000000     / [ ] Tau meter goodness of fit
DATE-TIP (C)  = 2017-03-29T10:54:50.85 / Time of Tau from Taumeter
SYSOFF   (C)  = Nasmyth        projection      / WARNING! Added a dummy 'projection' value
XOFFSET  (R4) = -1.9150140E-04   0.000000      / WARNING! Added a dummy 'projection' value
YOFFSET  (R4) =  2.6664753E-05   0.000000      / WARNING! Added a dummy 'projection' value
```

9.3 IMBF-frontend

```
XTENSION (C)  = BINTABLE               / binary table extension
BITPIX   (I4) =                      8 / 8-bit bytes
NAXIS    (I4) =                      2 / 2-dimensional binary table
NAXIS1   (I4) =                    139 / width of table in bytes
NAXIS2   (I4) =                      1 / number of rows in table
PCOUNT   (I4) =                      0 / size of special data area
GCOUNT   (I4) =                      1 / one data group (required keyword)
TFIELDS  (I4) =                     22 / number of fields in each row
EXTNAME  (C)  = IMBF-frontend          / name of this binary table extension
SCANNUM  (I4) =                    138 / Scan number
DATE-OBS (C)  =                        / observing date (Y2K format with time) in TIMESY
DEWRTMOD (C)  =                        / Dewar tracking system
DEWANG   (R8) =   0.00000000000000     / [Deg] Dewar angle
FEBEBAND (I4) =                      1 / \nbd\ number of basebands for this febe
FEBEFEED (I4) =                      8 / \nfd\ total number of feeds
NUSEFEED (I4) =                      8 / \nch\ Number of feeds in use.
VELOSYS  (R8) =   6.70000000000000     / [km/s] Source Radial Velocity in Reference Fram
SPECSYS  (C)  = LSR                    / Reference Frame
VELOCONV (C)  = optical                / Convention for doppler correction
EMIRBEAM (C)  = right                  / EMIR beam used
RECNAME  (C)  = E230                           / Receiver Name
LINENAME (C)  = L220510                        / Line Name
```

```
RESTFREQ (R8) =   220.510000000000              / [GHz] Rest frequency of line
BEAMEFF  (R4) =  0.9200000                      / Beam efficiency
ETAFSS   (R4) =  0.9200000                      / Forward efficiency
GAINIMAG (R4) =  5.0119001E-02                  / (spectral line) Gain ratio image/signal sideban
SIDEBAND (C)  = LI                              / Side Band
SBSEP    (R8) =   12500000000.0000              / [Hz] Sideband separation
WIDENAR  (C)  = NULL                            / WARNING! Column not found in table
TCOLD    (R4) =   32.82200                      / Cold Load Temperature
THOT     (R4) =   292.6630                      / Hot Load Temperature
IFEED    (I4) =              1                  / selected feed
NOFEEDS  (I4) =              1                  / No. of feeds
POLA     (C)  =                                 / Feed orientation
DOPPLER  (C)  = Doppler                         / Frequency tracking
IFCENTER (R4) =   6.250000                      / IF center Frequency
BANDWID  (R4) =   4.000000                      / [Hz] Bandwidth for this receiver
IFFLIPPS (L)  =        F                        / IF flipps frequency scale
SPECLO   (L)  =        F                        / Special LO used
TSCALE   (C)  = antenna                         / Temperature scale selected
FRQTHROW (R4) =   0.000000                      / [GHz] Frequency switching throw
FRQOFF1  (R4) =   0.000000                      / [GHz] Frequency offset 1
FRQOFF2  (R4) =   0.000000                      / [GHz] Frequency offset 2
```

9.4 IMBF-backend

```
XTENSION (C)  = BINTABLE              / binary table extension
BITPIX   (I4) =                     8 / 8-bit bytes
NAXIS    (I4) =                     2 / 2-dimensional binary table
NAXIS1   (I4) =                    81 / width of table in bytes
NAXIS2   (I4) =                    24 / number of rows in table
PCOUNT   (I4) =                     0 / size of special data area
GCOUNT   (I4) =                     1 / one data group (required keyword)
TFIELDS  (I4) =                    12 / number of fields in each row
EXTNAME  (C)  = IMBF-backend          / name of this binary table extension
SCANNUM  (I4) =                   138 / Scan number
DATE-OBS (C)  = 2017-03-29T10:54:54.00 / observing date (Y2K format with time) in T
FEBEBAND (I4) =                     1 / \nbd\ number of basebands for this febe
FEBEFEED (I4) =                     8 / \nfd\ total number of channels
NUSEFEED (I4) =                     8 / \nch\ Number of channels in use.
PART     (I4) =             2      ...                   6    / Identification
REFCHAN  (I4) =             1      ...              376833    / Refrence Channel
CHANS    (I4) =         16384      ...               16384    / Number of Channels
DROPPED  (I4) =          1843      ...                1843    / Channels dropped
USED     (I4) =         12494      ...               12494    / Channels used
RECEIVER (C)  = E2HUI              ...      E2VLO             / Receiver connected
IFRONT   (I4) =             1      ...                   1    / Backpointer to frontend table (memory
BAND     (C)  = E2HUI              ...      E2VLO             / Sub-band specification
POLAR    (C)  = NONE               ...      NONE              / Polarization
PIXEL    (I4) =             1      ...                   1    / Pixel No. (arrays)
REFFREQ  (R4) =   7200.000         ...       -8480.000        / [MHz] IF reference frequency
SPACING  (R4) = -4.8828125E-02     ...      -4.8828125E-02    / [MHz] Spacing
LINENAME (C)  = L220510            ...      L220510           / Additional Line Name
```

9.5 IMBF-backendFTS

The DATA column is omitted in this output:

```
XTENSION (C)  = BINTABLE               / binary table extension
BITPIX   (I4) =                      8 / 8-bit bytes
NAXIS    (I4) =                      2 / 2-dimensional binary table
NAXIS1   (I4) =                1572888 / width of table in bytes
NAXIS2   (I4) =                     11 / number of rows in table
PCOUNT   (I4) =                      0 / size of special data area
GCOUNT   (I4) =                      1 / one data group (required keyword)
TFIELDS  (I4) =                      5 / number of fields in each row
EXTNAME  (C)  = IMBF-backendFTS        / name of this binary table extension
SCANNUM  (I4) =                    138 / Scan number
OBSNUM   (I4) =                      1 / Observation number
BASEBAND (C)  =                        / [--] Baseband number
DATE-OBS (C)  = 2017-03-29T10:54:54.41 / observation start in TIMESYS system
MJD_BEG  (R8) =   57841.4547964699     / Duplicate of DATE-OBS (memory only)
DATE-END (C)  = 2017-03-29T10:54:58.42 / observation end in TIMESYS system
MJD_END  (R8) =   57841.4548428241     / Duplicate of DATE-END (memory only)
CHANNELS (I4) =                 393216 / \nch\ Number of channels for this baseband
NPHASES  (I4) =                      1 / no. of switch phases in a switch cycle
PHASEONE (C)  = ON                     / First phase is ON or OFF source
TSTAMPED (R8) =   1.00000000000000     / Where the time stamps apply
MJD      (R8) =   57841.4547911713     ...    57841.4548490417 / [day] MJD at integration start
INTEGTIM (R8) =  0.498954000000000     ...   0.498954000000000 / [s] Integration time
ISWITCH  (I4) =              1      ...                    1    / Integration type
FOREPOIN (I4) =              1      ...                   11    / Forward pointer to compressed columns
BACKPOIN (I4) =              1      ...                   11    / Backward pointer to uncompressed colum
```

9.6 IMBF-antenna

MRTCAL splits the antenna extension in the *antenna slow* and *antenna fast* descriptions.
Antenna slow:

```
XTENSION (C)  = BINTABLE               / binary table extension
BITPIX   (I4) =                      8 / 8-bit bytes
NAXIS    (I4) =                      2 / 2-dimensional binary table
NAXIS1   (I4) =                   6228 / width of table in bytes
NAXIS2   (I4) =                      8 / number of rows in table
PCOUNT   (I4) =                      0 / size of special data area
GCOUNT   (I4) =                      1 / one data group (required keyword)
TFIELDS  (I4) =                     19 / number of fields in each row
EXTNAME  (C)  = IMBF-antenna           / name of this binary table extension
SCANNUM  (I4) =                    138 / Scan number
OBSNUM   (I4) =                      1 / Observation number
DATE-OBS (C)  = 2017-03-29T10:54:54.41 / observation start in TIMESYS system
MJD_BEG  (R8) =   57841.4547964699     / Duplicate of DATE-OBS (memory only)
DATE-END (C)  = 2017-03-29T10:54:58.42 / observation end in TIMESYS system
MJD_END  (R8) =   57841.4548428241     / Duplicate of DATE-END (memory only)
OBSTYPE  (C)  = calibrate              / Observation type
SUBSTYPE (C)  = calSky                 / Subscan type
SUBSTIME (R8) =   5.00000000000000     / [s] Subscan time
```

```
SYSTEMOF (C)  = projection             / System for offsets
SUBSXOFF (R8) = -2.908882000000000E-03 / [rad] Subscan x offset
SUBSYOFF (R8) =   0.00000000000000     / [rad] Subscan y offset
SETYPE01 (C)  =                        / Segment type
SETIME01 (R8) =   0.00000000000000     / [s] Segment time
SEXOFF01 (R8) =   0.00000000000000     / [rad] Segment x offset
SEYOFF01 (R8) =   0.00000000000000     / [rad] Segment y offset
SEXSTA01 (R8) =   0.00000000000000     / [rad] Segment x start
SEYSTA01 (R8) =   0.00000000000000     / [rad] Segment y start
SEXEND01 (R8) =   0.00000000000000     / [rad] Segment x end
SEYEND01 (R8) =   0.00000000000000     / [rad] Segment y end
SESPES01 (R8) =   0.00000000000000     / [rad/s] Speed at start of segment
SESPEE01 (R8) =   0.00000000000000     / [rad/s] Speed at end of segment
DOPPLERC (R8) =  0.999881490337000     / Doppler correction factor
OBSVELRF (R8) =   28.8325140071000     / [km/s] Observer velocity in rest frame
MJD      (R8) =   57841.4547916667     ...    57841.4548726852 / [Julian day] MJD at integration start
LST      (R8) =   83367.2657212536     ...    83374.2848861940 / [s] Local apparent sidereal time (integ
LONGOFF  (R8) = -2.908881986513734E-03 ...  -2.908881986513734E/ [rad] long. offset from source in user
LATOFF   (R8) =   0.00000000000000     ...    0.00000000000000 / [rad] lat. offset from source in user
CAZIMUTH (R8) =   1.31577241984522     ...    1.31600523032287 / [rad] Commanded Azimuth
CELEVATI (R8) =  0.630288975389516     ...   0.630683181202881 / [rad] Commanded Elevation
TRACEFLA (I4) =              0      ...                   0    / Trace Flag
```

Antenna fast:

```
XTENSION (C)  = BINTABLE               / binary table extension
BITPIX   (I4) =                      8 / 8-bit bytes
NAXIS    (I4) =                      2 / 2-dimensional binary table
NAXIS1   (I4) =                   6228 / width of table in bytes
NAXIS2   (I4) =                      8 / number of rows in table
PCOUNT   (I4) =                      0 / size of special data area
GCOUNT   (I4) =                      1 / one data group (required keyword)
TFIELDS  (I4) =                     19 / number of fields in each row
EXTNAME  (C)  = IMBF-antenna           / name of this binary table extension
DATE-OBS (C)  = 2017-03-29T10:54:54.41 / observation start in TIMESYS system
MJD_BEG  (R8) =   57841.4547964699     / Duplicate of DATE-OBS (memory only)
DATE-END (C)  = 2017-03-29T10:54:58.42 / observation end in TIMESYS system
MJD_END  (R8) =   57841.4548428241     / Duplicate of DATE-END (memory only)
TRACERAT (I4) =                    128 / No. of fast traces per slow trace
MJDFAST  (R8) =   57841.4547569444     ...    57841.4548494466 / [Julian day] MJD at integration start
AZIMUTH  (R8) =   1.31550400027680     ...    1.31576626838095 / [rad] Encoder Azimuth
ELEVATIO (R8) =  0.631753369365499     ...   0.632199246447845 / [rad] Encoder Elevation
```

Appendix A

Obsolete chunks slicing

This section describes the chunk slicing strategy (support for `MSET CALIBRATION BANDWIDTH`) implemented in **MRTCAL** from its beginning up to GILDAS version apr26. It was abandoned because the overheads of 20 MHz chunks for large bandwidths where dominating the science scan calibration. The chunks now always match the hardware chunks width. The calibration bandwidth is only applied when calling ATM through TELCAL `chopper_t`. The calibration products (Tsys, etc) are constructed as full width chunks even if they have been computed on 20 MHz intervals. This way, science scans are completely insensitive to the calibration width: only full calibration arrays are applied, independently of the way they have been computed.

A.1 Description

[This feature was implemented in the subroutine `imbfits_resample_header_backend` which is now deleted.]

In order to be able to make calibration at a bandwidth smaller than the hardware chunks, **MRTCAL** is able to map the raw data block read with **CFITSIO** into smaller *memory* chunks. The main advantage is that this approach has very well defined boundaries: When the **IMBFITS** `BACKEND` table is read, it can be replaced on-the-fly with its sliced version. All the rest of the code does not have to know this happened. The downside is that the more memory chunks are defined, the more overheads they carry (e.g. description of their spectroscopic axis, calls to chunk-by-chunk subroutines, stitching, etc). Experience shows that slicing the 1400 MHz of FTS chunks into 14 chunks of 100 MHz introduces a penalty of 10% in the whole computation time (reading, slicing, calibrating, writing).

In practice, user can request the bandwidth he wishes. In order to slice the USED channels into comparable pieces, **MRTCAL** will choose the nearest bandwidth which gives an integer number of slices

$$N = \mathrm{nint}\left(\frac{\mathrm{USED} \times |\mathrm{SPACING}|}{W}\right) \tag{A.1}$$

where W is the requested bandwidth. N must also be corrected so that there is at least 1 and at most USED channels per subchunk

$$\mathrm{USED} \geq N \geq 1. \tag{A.2}$$

While the `CHANS` value (total number of channels) is often a multiple of 2, USED channels have no special value. In particular, it may not be a multiple of N. Their relationship can be

written as

$$\text{USED} = q \times N + r, \tag{A.3}$$

where q and r are respectively the integer quotient and the remainder of the division USED$/N$. Since $0 \leq r \leq N - 1$, USED can be divided in N subchunks, r of them having one extra channel

$$\text{USED} = [q \times (N - r)] + [(q + 1) \times r]\,. \tag{A.4}$$

This gives the best channel division and a negligible difference of subchunks bandwidth.

ZZZ Say a word about the "famous" special channel with its half weight. Slicing does not have any special problem with it.

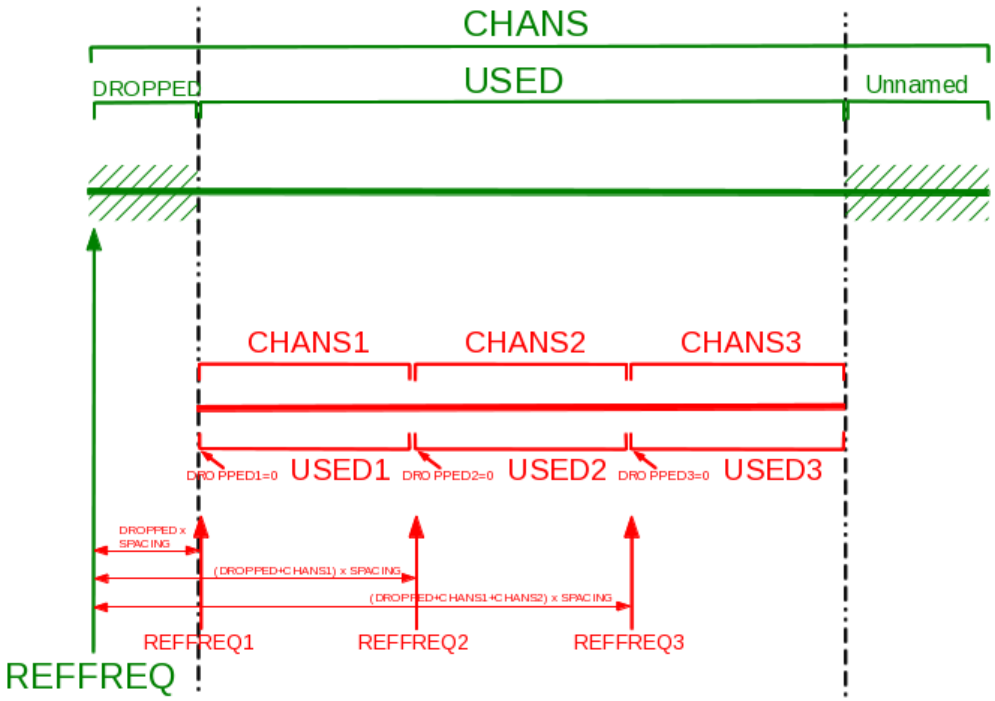

Figure A.1: Hardware chunks can be virtually divided into smaller pieces just by redefining the `BACKEND` table. In this example, the table is sliced from 1 chunk to 3 subchunks (3 times more lines in the table). The same raw data block read with **CFITSIO** will be mapped by Mrtcal with a larger number of *memory* chunks.