IRAM-COMP-007

Revision: 2.4
Nov 10,2005

Contact Author

Institut de RadioAstronomie Millimétrique

NCS 30m Antenna Mount Drive

Owners Alain Perrigouard (perrigou@iram.fr),

Keywords: 30m Drive Control

Approved by:	Date:	Signature:
A.Perrigouard	November 2005	

Change Record

REVISION	DATE	AUTHOR	SECTION/PAGE AFFECTED	REMARKS
1	May 2004	A.Perrigouard	all	
2	June 2004	A.Perrigouard	Section 3	
2.2	Oct. 2004	A.Perrigouard	Section 4 New	Data Logging
2.3	May 2005	A.Perrigouard	3.2.19 and 4.2.4	
2.4	Nov.2005	A.Perrigouard	2.2, 3.2 and 3.3	

Content

1 Hardware description....4

2 Servo kernel module....4

2.1 Servo controllers....4
2.1.1 Basic controller....4
2.1.2 Cascade controller....5
2.2 Servo implementation....8
2.2.1 Kernel module int_ant....8
2.2.2 Initialization tasks....10
2.2.3 Slow and periodic tasks....11

3 Observation module and commands....12

3.1 Introduction....12
3.1.1 Source Positions and coordinate system....12
3.1.2 Basis System....14
3.1.3 Descriptive User Coordinate System....15
3.1.4 Projection....15
3.2 Commands....17
3.2.1 sourceBody....17
3.2.2 sourcePlanet....18
3.2.3 sourceSystem....18
3.2.4 sourceOffsets....19
3.2.5 setNextSubscanTrack....19
3.2.6 setNextSubscanSlewAzimuth....19
3.2.7 setNextSubscanSlewElevation....20
3.2.8 setNextSubscanOtf....20
3.2.9 setNextSegmentLinear....20
3.2.10 setNextSegmentCircle....21
3.2.11 setNextSegmentCurve....21
3.2.12 prepareObservation....21
3.2.13 startObservation....22
3.2.14 haltObservation....22
3.2.15 setAzimuthWrap....22
3.2.16 setPointingParameters....23
3.2.17 setRefractionParameters....23
3.2.18 setTraceRate....23
3.2.19 sunAvoidance....23
3.2.20 sunCoordinates....24
3.2.21 zenithAvoidance....24
3.2.22 Initialization and test commands....24

3.3 Implementation in evItSlow....28
3.3.2 Pointing correction....30
3.4 Data logging....31
4 Some Results....33
4.1 Tracking in cascade mode....33
4.1.1 Azimuth step from 220deg to 220.2deg in cascade mode....33
4.1.2 Elevation step from 45deg to 45.2deg in cascade mode....34
4.2 On the fly scans....36
4.2.1 Source HORIZONTAL, OTF scan in HORIZONTALOFF....36
4.2.2 Source EQUATORIAL, OTF scan in HORIZONTALOFF....36
4.2.3 Source EQUATORIAL, OTF scan in EQUATORIAL....37
4.2.4 itFast timing....39

1 Hardware description

To drive the motor amplifiers and to read the encoders, a VME chassis replaces the old CAMAC interface in the 30m New Control System.
Juan Penalver wrote already an extended document: 30M Antenna Control with VME Modules.
To summarize, there is a Single Board Computer in slot 1, a Motorola MVME2041 plus a number of modules to interface the hardware:

- BC366, a VMEbus Time code processor, used for generating different interrupts and to feed a NTP server.
- IK320, a VMEbus counter card, used to read the Heidenhain ROD800 main axes encoders.
- IK340, a VMEbus counter card, used to read the Heidenhain ROD456 motor encoders.

2 Servo kernel module

2.1 Servo controllers

The implementation of the servo controllers is based on the work of Rainer Bardenheuer and his implementation made 18 years ago. Rainer's controller software was executed on a CAMAC microprocessor board and was written in assembler.
In the present implementation, the drives may be in different modes: PRESET, TRACK, INIT, STOP, etc. PRESET and INIT assume that the axes are driven in velocity. That means that the servomechanisms receive velocity requests.
TRACK and STOP mean that the positions of the axes are the controlled variables. The servomechanisms may either receive velocity requests or torque requests. We call, servo controller, the part of this new software implementation that implements the servo algorithm and establishes the velocity or the torque requests.
Two bits per axis in the "output register for elevation, azimuth and subreflector control" are used to switch from one mode of request to the other. In the following, the 2 cases are named basic and cascade.

- In basic, when the position is controlled in a closed feedback, the servo controller implemented in software is a PI controller. It is the case when the axis is in TRACK mode (variable request = TRACK) and the servomechanism is in basic (variable track = BASIC). Remark that the servomechanism is also in basic when the axis is in PRESET or INIT mode. In PRESET, The algorithm checks in a slow (loose) loop the position error, i.e. the difference between the target and the actual positions in order to reach the target at maximum speed/acceleration but without overshoot which would be generated inevitably with only a PI filter.
- In cascade, the servo loop always controls the drive position. The servo controller is a cascade of 2 PI controllers (position and velocity), the actual motor positions are feedback and there is a mechanism of friction compensation.

In the following, the 2 controllers, basic and cascade, in their CAMAC and new VME implementations, are presented. The names of the variables are the names either used in the source codes or in the manual Antenna-ServoMechanism from R.Bardenheuer of November 20, 1985.

2.1.1 Basic controller

In the manual "Antenna-servomechanism" page 4-2 the proportional gain is 3 in azimuth and 2.5 in elevation. The integration time is 5s.
However, the same coefficients are used in the CAMAC implementation (K=3) for both axes.
In the VME implementation we do the same choice, so, the same coefficients are used.

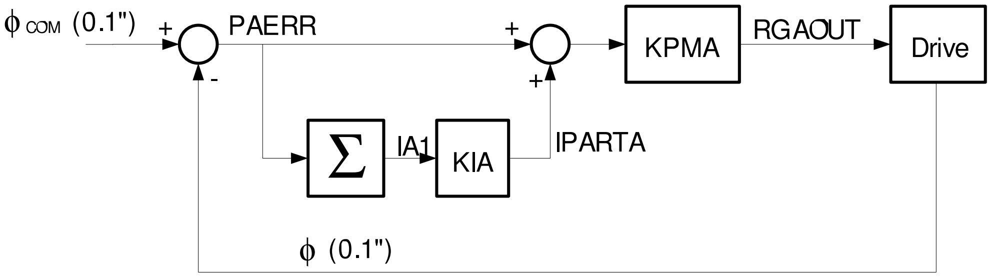


Figure 1 - Basic Controller - Camac implementation

```
Ts is the loop period. Ts = 0.006s
Σ is the sum or the integration of the input.
KPMA = 3 * 2^15 / 36000    See KPMD in AZSERVO.LIS and ELSERVO.LIS
KIA = Ts / T = 0.006 / 5   See KID in AZSERVO.LIS and ELSERVO.LIS
```

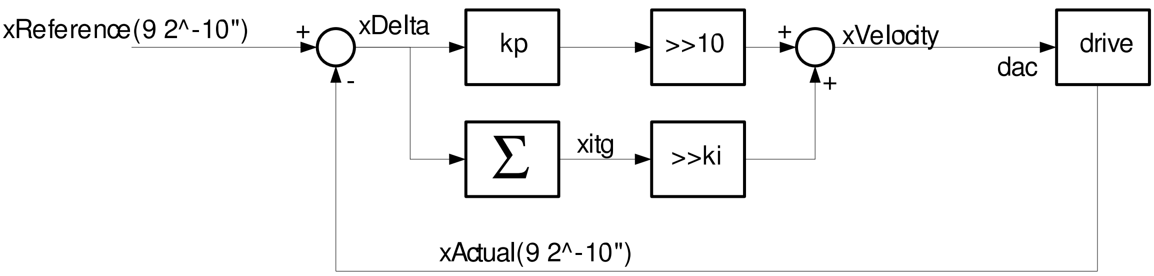


Figure 2 - Basic Controller - VME implementation

```
Ts is the loop period. Ts = 2^-7s
Σ is the Sum or the integration of the input.
Kp = 3 * 2^15 / 100 / 2^12 = 0.24
let's define kp = 0.24 * 2^10 = 246

Ki = Kp * Ts / T = 0.24 * 2^-7 / 5 = 0.000375
lets define ki such that 2^-ki ~= 0.000375  =>  ki = 11
```

2.1.2 Cascade controller

In the manual “Antenna-servomechanism” page 4-2 there is representation of the cascade controller with the following definition of the variables:

```
phiCom reference axis position in rad.
phi actual axis position in rad.
phiComDot reference axis velocity in rad/s.
phiMotDot actual axis velocity in rad/s deduced from motor encoders and
corrected of the gear ratio.
To torque in mN applied to the antenna axis.
integral is the normal time integral of the input.
K = 2 s^-1 for Az and =2.5 s^-1 for El
T = 0.24s for Az and = 0.2s for El
KM = .9 10^9 mN s/rad for Az and = 1.5 10^9 mN s/rad for El
TM = 0.18s for Az and = 0.12s for El
```

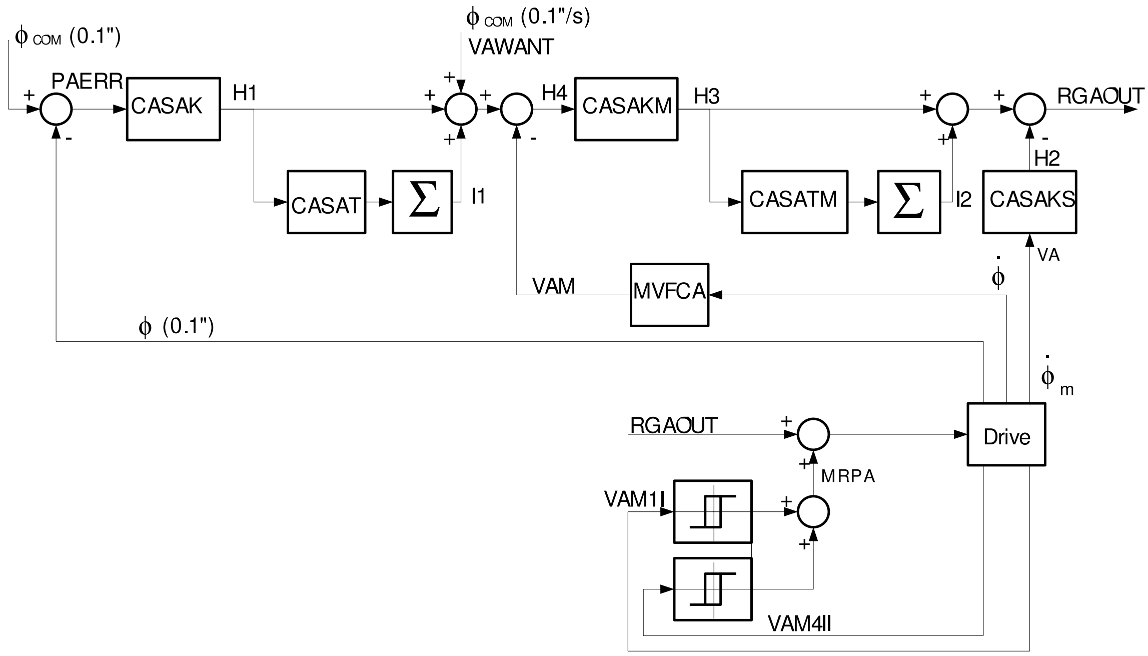


Figure 3 - Cascade controller - CAMAC implementation

```
Ts is the loop period. Ts = 0.006s
Σ is the Sum or the integration of the input.
```

Azimuth:

```
CASAK = K = 2
CASAT = Ts / T = 0.006 / 0.24s = 0.025
CASAKM = KM * RBGOFA
  RBGOFA = PI/180/36000 * 2^15/265 * 1/14165
           .1"s->rad     1/DAC gain  gear-ratio
CASATM = Ts / TM = 0.006 / 0.18 = 0.03333
MVFCA = 6.35368867
```

Elevation:

```
CASEK = K = 2.5
CASET = Ts / T = 0.006 / 0.2s = 0.03
CASEKM = KM * RBGOFE
  RBGOFE = PI/180/36000 * 2^15/265 * 1/15727
           .1"s->rad     1/DAC gain  gear-ratio
CASETM = Ts / TM = 0.006 / 0.12 = 0.05
MVFCE = 5.72264259
```

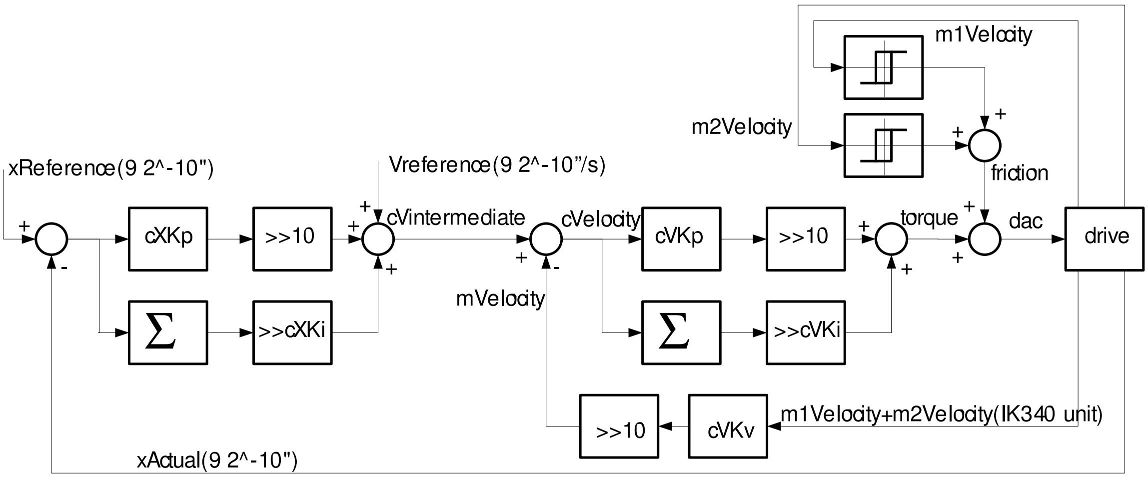


Figure 4 - Cascade controller – VME implementation

```
Ts is the loop period. Ts = 2^-7s

The axis encoder has a period of 36" and the VME module IK320 interpoles
the period by a factor 2^12.
1 unit = 9 2^-10" and 1degree is equivalent to 100 2^12 encoder units.

The motor encoder has a period of 720" (1800 periods/rev) and the VME
module IK340 applies to the period a 256-fold interpolation.
1 unit = 360 * 3600 / 1800 / 256 "
```

Azimuth:

```
xkp = K = 2
let's define cXKp * 2^10 = xkp => cXKp = 2048
cXKp * xDelta overflows for abs(xDelta) > 2^31 / 2048 equivalent to
xdelta > 2^31 / 2048 * 9 2^-10 = 9216" ~= 2.56deg OK

xki = K * Ts / T = 2 * 2^-7 / 0.24
let's define 2^-cXKi = xki  =>  cXKi = 4

vkp = .9 10^9 * PI/18000/2^12 * 2^15/265 * 1/14165 = .33477
       KM       9 2^-10"->rad  1/DAC gain  gear-ratio
let's define cVKp * 2^10 = vkp => cVKp = 343
overflow for abs(cVelocity) > 2^31 / 343 equivalent to
abs(cvelocity) > 2^31 / 343 * 9 2^-10 = 55027 "/s ~= 15.3deg/s OK

vki = vkp * Ts / TM = .33477 * 2^-7 / 0.18
let's define 2^-cVKi = vki  =>  cVKi = 6

m1Velocity = actual motor 1 position(IK340 unit) - previous motor 1
position
m2Velocity = actual motor 2 position(IK340 unit) - previous motor 2
position
vkv = 1/2 360*3600/1800/256 * 2^7 *   1/14165   *   2^10/9     = 1.4458
          IK340 unit -> "    1/Ts   gear-ratio  "->9 2^10" unit
let's define cVKv * 2^10 = vkv =>  cVKv = 1480
Overflow for abs(CVelocity) > 2^31 / 2^10 equivalent to
abs(cvelocity) > 2^31 / 2^10 * 9 2^-10 = 18432"/s ~= 5.12deg/s OK
```

Elevation:

```
xkp = K = 2.5
let's define cXKp * 2^10 = xkp => cXKp = 2560
cXKp * xDelta overflows for abs(xDelta) > 2^31 / 2560 equivalent to
xdelta > 2^31 / 2560 * 9 2^-10 = 7373" ~= 2.05deg OK
```

```
xki = K * Ts / T = 2.5 * 2^-7 / 0.2
let's define 2^-cXKi = xki  =>  cXKi = 4

vkp = 1.5 10^9 * PI/18000/2^12 * 2^15/265 * 1/15727 = .50254
       KM       9 2^-10"->rad  1/DAC gain  gear-ratio
let's define cVKp * 2^10 = vkp => cVKp = 515
overflow for abs(cVelocity) > 2^31 / 515 equivalent to
abs(cvelocity) > 2^31 / 515 * 9 2^-10 = 36649 "/s ~= 10.2deg/s OK

vki = vkp * Ts / TM = .50254 * 2^-7 / 0.12
let's define 2^-cVKi = vki  =>  cVKi = 5

dMot1 = actual motor 1 position(IK340 unit) - previous motor 1 position
vkv = 1/2 360*3600/1800/256 * 2^7 *   1/15727   *   2^10/9     = 1.3022
          IK340 unit -> "    1/Ts   gear-ratio  "->9 2^10" unit
let's define cVKv * 2^10 = vkv =>  cVKv = 1333
Overflow for abs(CVelocity) > 2^31 / 2^10 equivalent to
abs(cvelocity) > 2^31 / 1333 * 9 2^-10 = 14159"/s ~= 3.93deg/s OK
```

The implementation of the Cascade controller works perfectly in elevation with the above parameters scaled from the values found in the Rainer’s documentation and in ELSERVO.LIS.
For the azimuth implementation the coefficients of the cascade position loop are slightly changed. The scaled coefficients does not give a stable controller when an azimuth step position is applied. The new proposed coefficients are a compromise between stability and low position error for tracking on a fixed position:
cXKP=4096 and cXKi=6

2.2 Servo implementation

Linux powers the Single Board Computer MVME2041. Fast and real time operations are executed in kernel modules. In those modules no floating number operations are allowed. The servo controller algorithms performed in the fast loop of a kernel module are only coded with integer numbers. As a consequence, all the operations should be verified carefully to avoid overflow or underflow.
The implementation is triggered by 2 periodic interrupts at 128Hz and 1Hz. The period of the fast interrupt is ~7.8ms. It differs slightly from the 6ms period used in the Rainer’s implementation. This difference will not change the overall response of the servo and the power of 2 selected for the frequency of 128Hz simplifies greatly the coding.

2.2.1 Kernel module int_ant

This module provides 3 functions (itFast(), itSlow() and encoder()) to handle 2 periodic interrupts generated by the bc366VME time code processor and the ik320 VME count card completion interrupts. The 2 periodic interrupts are the so-called heartbeat at 128Hz and the 1pps generated every second. We use the functions bc366_heartbeat_action() and bc366_pps_action() from the bc366 module to assign the handlers and the functions itFast() and itSlow() to these interrupts.
The ik320 VME interrupts signals the end of the main axis encoder conversion. This interrupt, IRQ level 3, vector = 0xC1, is handled by this module by calling the function encoder().
Actually the heartbeat interrupt occurs 70ns before the 1pps interrupt, every second. As the operations executed every second should have the highest priority, and should be performed before any heartbeat handling, the 1second operations are synchronized on the hearbeat interrupt for a specific value of a counter equal to 128. This counter is incremented with the heartbeat interrupt and is cleared with the 1pps interrupt.

2.2.1.1 1pps interrupt

The function itSlow() just clears the counter count of the common area.
This counter is incremented at the beginning of itFast() and if count is equal to 128 the function onePPS() is executed right away before any other action.

2.2.1.2 Heartbeat interrupt

Beside and after the counter increment and eventually the onePPS() call, the function itFast() starts the main axes encoder position latching procedure and requests to the ik320VME module the generation of an interruption for the end of the conversion.
It reads the 2 motor encoders for the axes, azimuth and elevation. Then, it calculates the motor differential positions and deduces the motor velocities in 9 2^-10"/s units.
The function has a conditional section compiled when the variable NOIK320 is defined: In a development situation when there is neither main axis encoder nor VME module ik320, the variables az.x0 and el.x0 are extrapolated here and the function move() is also called here(instead of being called after the reception of the ik320 interrupt driven by the conversion completion).

2.2.1.3 Function onePPS()

The function onePPS() calculates for each axis, x0 and x1. x0 is the reference position for this current 1pps interrupt time and x1 is the reference position for the next 1pps interrupt time. dx is equal to the difference x1-x0 and sumDx is cleared. SumDx is used to accumulate dx every 7.8ms in encoder().

2.2.1.4 ik320 interrupt

The function encoder() reads the ik320 status register and depending on which axis is ready, reads the corresponding latched counters.
If the condition variable SIMULATION is no defined, the variable xActual is calculated from the actual values of the latch counters and is saved in the shared memory area.
If the drive is in remote, the variable xReference and vReference are updated:

```
sumDx += dx
xReference = x0 + dx
vReference = dx
```

If the variable MOVE_TEST is not defined (it's the normal situation) the function move() is called. The variable MOVE_TEST has been defined once to allow the execution of the task moveTest and the debugging of the function move() which is called by moveTest.

The function move() implements the servo code. It is called for each axis, as soon as the actual encoder position is read.
The position error, px->xDelta, is calculated and depending on the variable px->request, different algorithms may be applied. px->xDelta is the difference between px->xReference and px->xActual.
The variable px->request may have the possible values: R_PRESET, R_TRACK, R_SLEW, R_STOP or R_TRSFER.

- R_TRSFER is foreseen for a session of collecting data and for calculating the drive transfer function. It is not tested with the 30m antenna.
- R_PRESET is used to request the drive to reach a given position after a constant acceleration phase, a phase of displacement at maximum speed and then a constant deceleration phase. For a short distance, the phase at constant and maximum speed may not exist. This mode is always requested for a medium or long distance displacement to avoid tracking algorithm overshoot. When the absolute value of the axis velocity is under a certain threshold (px->xVelocityEpsil) and the absolute value of the position error is lower than px->xDeltaMin, the axis request mode is switched to R_TRACK.
- R_TRACK is used to track a position. The servo mechanism is either in basic mode or in cascade mode depending on the variable px->track equal to BASIC or CASCADE. Note that whatever the value of px->track the BASIC servo mechanism is used when the axis px->request is R_PRESET.

- R_STOP is used to keep the axis on a fixed position. When the axis px->request is set to R_STOP while it is moving, the axis is slow down and then is requested to track to the reached position by switching px->request to TRACK. Note that the DAC output will increase up to its maximum value if this mode is requested and the axis is locked for any reason (brake, amplifier switched off, etc…)
- R_SLEW is used to move the axis in basic mode and at constant speed. The axes are in this mode at start time with a requested velocity px->vReference set to 0. This mode keeps the DAC output to 0 until the antenna is free to move.

After the start of itFast(), the function move() is called around 250us after, for the 1st axis, and 300us after, for the 2nd axis, in the case that only one incremental encoder is used per axis.

2.2.1.5 Module installation

Before installing this module, universe.o and bc336.o have to be installed and the encoder VME modules should be initialized. The installation procedure starts with the 2 kernel module installation:

```
modprobe universe
modprobe bc336
```

We consider that the main axis encoder VME module ik320 has been already initialised and it is running. Another initialization of the VME module would cause a re-initialization of the incremental encoders. As a consequence, ik320Init is commented out in the procedure.

The motor encoder VME module ik340 is initialized by calling ik340Init:

```
#/control/antenna/bin/ik320Init
/control/antenna/bin/ik340Init
```

And finally:

```
insmod /control/antenna/bin/int_ant
```

2.2.2 Initialization tasks

The encoder VME module initialization tasks have to be executed before installing int_ant.o.
If ik320Init is executed, the incremental encoders should be re-initialized and the axes should be turned enough to pass their init points and to reset the counters.
ik340Init initializes the motor encoder VME module and can be executed at re-installation time.

2.2.2.1 ik320Init

The task ik320Init is based on a source code written by Juan Penalver: ik320.c.
The IK320 VMEbus Counter Card user's manual and in particular the program example are necessary to understand this program executed to initialize the VME module.

2.2.2.2 Ik340Init

The task ik340Init is based on a source code written by Juan Penalver: readenc.c.
The IK340 VMEbus Counter Card user's manual is necessary to understand this program executed to initialize the VME module.

2.2.2.3 initAntenna

Once the kernel module int_ant.o is installed and the shared memory area described with the structure struct s_antenna is registered, the task initAntenna can initialize the elements of the shared memory area.

The function antdata() returns the address of this area in the user space domain by calling mmap() . The function mmap() is one of the functions implemented in int_ant.o which is available for the device /dev/int_ant and in particular for its descriptor.
The struct s_antenna describes the content of this shared memory area. The struct s_antenna is declared in s_antenna.h.
This task sets the servo variables to their default values and switches the servo mechanisms, azimuth and elevation, to basic mode.

2.2.2.4 Config

The task config is called just after initAntenna in the installation procedure but can be called at anytime later in order to modify some configuration variables.
The values of the configuration variables are found in the file /control/antenna/config.30m. The format of each line of this file is:

```
Variable_name  value
```

For instance:

```
az.kp 246 Az proportional factor
```

If the first string does not correspond to any variable name, the line is considered as a comment line.
All strings, following on the same line the configuration value, are considered as well as comments.
The notation for the variable names is obvious. For the example, az is the element of type struct s_axes of the structure s_antenna and kp is one element of this struct s_axes.

2.2.2.5 InitObservation

This task initializes the structures needed to define the sources and the subscans for the observations. This task should be executed before starting the slow periodic task evItSlow which prepares the position interpolations.
The task calls the function shm_connect() to connect to the shared memory area identified by ‘PICO’.
The struct s_observation describes the content of this shared memory area. The struct s_observation is declared in s_observation.h.
The task sets to some default values, the elements of the structure slaInput of type struct s_slaParams.
Those elements like longitude, latitude, temperature … are needed for calling the slalib functions.
The task initializes also the roots and the chains of segments and subscans by setting all the variables firstSubscan, firstSegment, nextSubscan, nextSegment to –1.

2.2.3 Slow and periodic tasks

These slow and periodic tasks are executed to prepare the reference positions and velocities.

2.2.3.1 SlaParams

slaParams calculates, every 10s, the tables amprms[] and aoprms[] which are used in evItSlow() for the conversions between mean place and geocentric apparent place, and for the conversions between apparent to observed place.

2.2.3.2 EvItSlow

evItSlow calls first antdata() to connect to the memory area shared with the kernel module int_ant.o.
Next it opens the file descriptor fd to /dev/int_ant. This file descriptor is used later for the 1s synchronization.
Note that this connection and this open are conditioned to the non-definition of the variable PCTCP00.

If PCTCP00 is defined, antdata() executes only a connection to a shared memory area identified by 'ANTE' and defined with the same structure struct s_antenna. PCTP00 is defined only for debugging purpose.

Next, evItSlow calls shm_connect() to get the pointer of the shared memory area identified by 'PICO'.

And finally, eItSlow implements a loop which is executed every second.
If the variable PCTCP00 is not defined at compilation time, the loop is triggered by the call read(fd) which has for argument fd, the file descriptor to the device /dev/int_ant. This call synchronizes the execution of the loop to the occurrence of the 1pps interrupts.
If the variable PCTCP00 is defined there is just a sleep(1) to simulate this 1pps synchronization. The variable PCTCP00 is used here for debugging purpose.

In the loop, depending on the different observation modes, the commanded positions, az.command and el.command are calculated and finally are converted in encoder units (9 2^-10 arc-seconds) before being assigned to az.x2 and el.x2.
The different observation modes are: IDLE, HORIZON, PREPARE, READY, RUN and STOP.

For the mode PREPARE, the function prepareScan() is called. If the time to prepare is passed, the scan starting position is evaluated and when this starting position is reached, the observation mode is switched to READY.

For the mode READY, the function readyScan() is called. If a time to stop is defined and this time is passed, the observation mode is switched to STOP. Otherwise, if the time to start is passed, the observation mode is switched to RUN.

For the mode RUN, the function runScan() is called. The function calculates the running positions depending on the type of the current subscan. If a time to stop is defined and this time is passed, the observation mode is switched to STOP. If there is no more subscan or they have not yet been defined, the observation mode is switched to READY.

For the mode STOP, the function stopScan() is called. The function requests the axes to stop immediately and it clears the integration buffers of the cascade servos.

3 Observation module and commands

The observation module, evItSlow, and the observation commands follow the description and the interfaces listed in the document antennaMountDrive.h edited by H.Ungerechts, W.Brunswig, A.Sievers and A.Perrigouard. Hereafter, the astronomical background is reproduced and complemented. They are needed to understand the syntax of the commands and the implementation of the module evItSlow:

3.1 Introduction

3.1.1 Source Positions and coordinate system

Source positions and (ranges of) offsets can be specified in different projections and spherical coordinate systems, which are organized approximately in a hierarchy from "high" levels (projections and user-defined "descriptive" systems) to "low" levels, e.g., horizontal coordinates. The choice and definition of systems starts out from commonly known and standard astronomical coordinates systems, the so called basis system:

3.1.1.1 Basis System

First, there is a choice of a pre-defined "basis system".
In the command **`source`** the basis system is selected by the argument `basisSystem`, a number between 0 and 6 (**`0=GALATIC, 1=EQUATORIAL, 2=APPARENTEQUATORIAL, 3=ECLIPTIC, 4=APPARENTECLIPTIC, 5=HADEC, 6=HORIZONTAL`**) and in some cases by one equinox system (a choice between **`0=J`** and **`1=B`**) and by the equinox year.

3.1.1.2 Descriptive System

Second, there is an optional user definition of a "descriptive system". The relation between the descriptive and the selected basis system is described by the 3D rotation (3 angles) that rotates the chosen basis system to the descriptive system. These 3 angles can be specified according to one of 3 conventions, i.e., origin, polar or Euler.
The convention is selected in the command **`source`** by the argument `descriptiveSystem`, a number between 0 and 3 (**`0=NODESCRIPTIVE`**, **`1=ORIGIN, 2=POLAR, 3=EULER`**). See below for the special case 0.
The source position is defined in the descriptive system by the 2 angles lambda, the source longitude, and beta, the source latitude, with the command **`source`** either if there is no projection (`projection` = **`NOPROJECTION`**) or if the projection is of **`RADIO`** type. If an effective projection different of **`RADIO`** is defined in the command **`source`**, the source longitude and latitude are meaningless and are not considered in the coordinate transformations

3.1.1.3 Projection's native system

Third, there is the choice of a projection with an optional definition of the projection's "native system". The "native system" is chosen with its pole or its origin at the projection center, so that the formulae for the projection take a simple form. The relation between native and descriptive system is described by the 3D rotation (3 angles) that rotates the descriptive system to the native system. The kind of projection determines the mathematical relation between offsets in the projection and longitude and latitude in the native system. Moreover the choices include 2 "pseudo-projections": "none" and "radio", for which the native system is always identical with the descriptive system.
In the command **`source`** the projection is selected with the argument `projection`, a number between 0 and 11 (**`0=NOPROJECTION, 1=RADIO`**, etc… see more details in the section “projection”).

3.1.1.4 Special Cases

In often-used, but special cases, the descriptive and basis system are identical, i.e., not rotated relative to each other (argument `descriptiveSystem` set to **`0`**, **`NODESCRIPTIVE`** in the command **`source`**). Moreover, the center of the descriptive system will often be chosen to be at the source position; in this case the longitude and latitude of the source are zero in the descriptive system (`lambda`=0 and `beta`=0). Finally, if a true projection is chosen (argument `projection` > 1 in the command **`source`**), λ and β are not used and if there is no source offset in the projection system, the source position will be at the center of the projection, i.e., at the pole or origin of the native system, depending on the type of projection.

Usage of the 2 angles lambda, the source longitude, and beta, the source latitude, of the **`source`** command for the various types of descriptive systems and projection:

`projection`→ `Descriptive system`↓	NOPROJECTION	RADIO	Any other projection
NODESCRIPTIVE	λ and β in **BS** **DS** equiv to **BS**	λ and β in **BS** **DS** equiv to **BS**	λ and β not used
ORIGIN	λ and β in **DS**	λ and β in **DS**	λ and β not used
POLAR	λ and β in **DS**	λ and β in **DS**	λ and β not used
EULER	λ and β in **DS**	λ and β in **DS**	λ and β not used

λ: source longitude, β: source latitude
BS: Basis System, DS: Descriptive System
"λ and β in BS" means source position defined with λ and β in BS

3.1.1.5 Commands Overview

- `source`
  The command **`source`** sets the source position in the chosen system and selects the basis system. Optionally, it defines a descriptive system and a native system and chooses a type of projection.

- `sourceOffsets`
  The command **`sourceOffsets`** adds position offsets at different levels in the "hierarchy" of coordinate systems, or equivalently at different stages of the transformations from high-level to low-level coordinates.

  The argument `systemOffset` specifies the coordinate system in which the offsets are applied. It is a number between 0 an7 (**0**=**`PROJECTION`**, **1**=**`DESCRIPTIVE`**, **2**=**`BASIS`**, **3**=**`EQUATORIAL`**, **4**=**`HADECOFF`**, **5**=**`HORIZONTRUE`**, **6**=**`HORIZONOFF`**, **7**=**`NASMYTH`**).
  `systemOffset` can be equal to 0 (**`PROJECTION`**) only if a projection system has been defined in the `source` command (projection different of **`NOPROJECTION`**, i.e. projection >0).
  `systemOffset` can be equal to 1 (**`DESCRIPTIVE`**) only if a descriptive system has been defined in the `source` command (`descriptiveSystem` different of **`NODESCRIPTIVE`**, i.e. `descriptiveSystem` >0).

  After a command **`source`** several commands **`sourceOffsets`** can specify offsets for the different coordinate systems, however only one offset will be counted per coordinate system (the last one defined for each coordinate system).
  Furthermore some offsets are mutually exclusive: **`PROJECTION`**, **`DESCRIPTIVE`** and **`BASIS`** are mutually exclusive and **`HORIZONTRUE`** and **`HORIZONOFF`** are also mutually exclusive. For instance an offset defined in **`HORIZONOFF`** excludes any previous offset defined in **`HORIZONTRUE`** and vice-versa.
  No offsets can be specified in a "higher" level than the "highest" level specified with the **`source`** command, i.e. if `basisSystem` is equal to **`HORIZONTAL`** in the **`source`** command then `systemOffet` in the **`sourceOffsets`** commands cannot be neither **`EQUATORIAL`** nor **`HADECOFF`** and if `basisSystem` is equal to **`HADEC`** in the **`source`** command then `systemOffset` in the **`sourceOffets`** commands cannot be **`EQUATORIALOFF`**.

- Scan commands
  The commands **`setNextSubScan*`** specify the next subscan which will start automatically after the current subscan. If the next subscan is an OTF subscan, see function **`setNextSubscanOtf`**, the functions **`setNextSegment*`** specify its segments.
  During the OTF subscans the offsets defined by the commands sourceOffsets are not added and applied.

  More details about the commands are given below.

3.1.2 Basis System

The basis system is selected by a keyword (and equinox where appropriate).
This option allows the user to select one of several predefined spherical coordinate systems.

- galactic

- Mean equatorial for any equinox (normally J...) especially B1950 and B2000, J2000 and Present Time
- apparentEquatorial
- Mean ecliptic for any equinox (normally J...) especially B1950 and B2000, J2000 and Present Time
- apparentEcliptic
- apparent hourAngle and declination
- Horizontal System. In this system the position is defined by the angles azimuth, Az, and elevation, El. Several references may be considered (TBD):
  astronomical
  geodetic
  relative to encoder zero points

3.1.3 Descriptive User Coordinate System

The descriptive coordinate system is specified by a keyword for type of definition (origin, pole, euler) and 3 angles in the command **`source`**.
This allows the user to specify a new spherical coordinate system that can have any origin and rotation relative to any of the predefined "basis" systems.
An example is the use of local coordinates along the major and minor axis of M31.

Types of definition and the 3 corresponding angles:

- origin
  lambdaO longitude in basis system of the origin of the descriptive system
  betaO latitude in basis system of the origin of the descriptive system
  kappaO position angle counted from the basis meridian through (lambdaO, betaO), i.e., the origin of the descriptive system, to the zero-meridian of the descriptive system

- polar
  lambdaP longitude in basis system of the pole of the descriptive system
  betaP latitude in basis system of the pole of the descriptive system
  kappaP position angle counted from the basis meridian through (lambdaP, betaP), i.e., the pole of the descriptive system, to the zero-meridian of the descriptive system

- euler
  Successive rotations about specified Cartesian axes. The standard Euler rotation angles are:
  "phi" rotates around the Z axis
  "theta" rotates around X axis resulting from the 1st rotation
  "psi" rotates around Z axis resulting from the 2nd rotation

  Note: This convention is the same as used in 'Classical Mechanics' by Herbert Goldstein, pg. 107.
  In slalib one may use the function slaDeule with order="ZXZ" void slaDeuler (char *order, double phi, double theta, double psi, double rmat[3][3])

3.1.4 Projection

A projection is specified by a keyword for the "projection type" and 3 angles, i.e., the coordinates of the pole or the origin of the "native" system related to the projection and an additional rotation angle.
The supported projections include those foreseen in GILDAS, plus the two "pseudo-projections" of the current control system ("none" and "radio") and some more that are readily available from astronomical data services like CDS.

References for projections:

B&S: L.M. Bugayevskiy and J.P. Snyder: "Map Projections.
A Reference Manual", Taylor & Francis, London & Philadelphia 1995, 2000.
C&G: M. R. Calabretta and E.W. Greisen: "Representations of Celestial Coordinates in FITS" 2002, A&A 395, 1077.

Note: the formulae below need to be checked before implementation.

- "pseudo"-projections:
  Let x, y be the x and y-position offset in the pseudo-projection;
  l = longitude, b = latitude in the descriptive system; and let l0, b0 be the source position.

  - "none" similar to CAR: Cartesian / plate caree below
    x = l - l0, y = b - b0

  - "radio" The conventional offsets with 1/cos(b)-factor in radio astronomy;
    this is the default. It is similar to GLS: Sanson-Flansteed / global sinusoidal below, but with the source position subtracted!
    x = (l - l0)*cos(b), y = b - b0

- "zenithal" ("azimuthal" or "polar") projections onto a plane centered on the North Pole of a "native" system defined relative to the descriptive user coordinate system by the angles:

  - lambdaProjection: longitude in descriptive system of the pole of the "native" system
  - betaProjection: latitude in descriptive system of the pole of the "native" system
  - kappaProjection: position angle counted from the descriptive meridian through (lambdaProjection, betaProjection), i.e., the pole of the native system, to the zero-meridian of the native system

  (these angles are analogous to lambdaP, betaP, kappaP, defined in he section descriptive user coordinate system /polar)

  Let x, y be the x and y-position offset in the projection;
  let l = longitude, b = latitude and
  p = pi/2 -b (polar distance) in the "native" system.
  Then x = r sin(l), y = r cos(l) (y = -r cos(l) in the convention of C&G)
  where r depends on the specific projection:

  - TAN: gnomonic or tangential or standard
    r = cot(b) = tan(p)
    (projection from center of sphere; B&S page 107/108)

  - SIN: orthographic
    r = cos(b) = sin(p)
    (projection from infinity; natural for aperture synthesis; B&S page 107/108)

  - STG: stereographic
    r = 2 tan (0.5 (pi/2-b)) = 2 tan(p/2)
    (projection from opposite end of diameter of sphere; B&S page 107/108)

  - ARC: zenithalEquidistant or schmidt [or azimuthal]
    r = pi/2 - b = p
    (approximation of image of a Schmidt camera)

  - ZEA: zenithEqualArea | azimuthalEqualArea

r = 2 sin(0.5 (pi/2-b)) = 2 sin(p/2)
(B&S page 107/108)

- "cylindrical" or "global" projections:
  the origin of the projection's "native" system is defined relative to
  the descriptive user coordinate system by the angles:

  - lambdaProjection: longitude in descriptive system of the origin of the "native" system
  - betaProjection: latitude in descriptive system of the origin of the "native" system
  - kappaProjection: position angle counted from the descriptive meridian through (lambdaProjection, betaProjection), i.e., the origin of the native system, to the zero-meridian of the native system (these angles are analogous to lambdaO, betaO, kappaO defined in the section descriptive user coordinate system /origin)

  with l = longitude, b = latitude in the "native" system:

  - CAR: cartesian or plateCaree
    x = l
    y = b
    (B&S page 55)

  - MER: mercator
    x = l
    y = ln (tan (0.5(pi/2+b)))
    (B&S page 50)

  - CEA : cylindricalEqualArea or Lambert
    x = l
    y = sin(b)
    (this is a special case of CEA)
    (B&S page 52)

  - GLS: Sanson-Flansteed or globalSinusoidal
    x = l cos(b)
    y = b
    (B&S page 67)

  - AIT: Hammer-Aitoff or Hammer [or Aitoff]
    x = 2 a cos(b) sin(l/2)
    y = a sin(b)
    where a = pow(2/(1 + cos(b) cos(l/2)), 0.5)
    (B&S page 176)

3.2 Commands

3.2.1 sourceBody

```
 * ABSTRACT : Define a source with respect to a body at the origin of a      *
 *            descriptive system.                                            *
 *            sourceBody <perihelionEpoch> <ascendingNode>                   *
 *            <argumentOfPerihelion> <inclination> <perihelionDistance>      *
 *            <eccentricity>                                                 *
 *            All arguments are mandatory.                                   *
 *            perihelionEpoch         epoch of perihelion T  (TT MJD)        *
 *            ascendingNode           longitude of the ascending node (rad)  *
 *            argumentOfPerihelion    argument of perihelion  (rad)          *
```

```
*          inclination             inclination (rad)                  *
*          perihelionDistance      perihelion distance (AU)           *
*          eccentricity                                               *
```

3.2.2 sourcePlanet

```
* ABSTRACT : Define a source with respect to a planet at the origin of a    *
*            descriptive system.                                            *
*            sourcePlanet <planetNumber>                                    *
*            planetNumber    1 Mercury                                      *
*                            2 Venus                                        *
*                            4 Mars                                         *
*                            5 Sarturn                                      *
*                            6 Jupiter                                      *
*                            7 Uranus                                       *
*                            8 Neptun                                       *
*                            9 Pluto                                        *
```

3.2.3 sourceSystem

```
* ABSTRACT : Define a new source.                                           *
*            sourceSystem <sourName> <basisSystem> <equinoxSystem>          *
* <equinoxYear> <lambda> <beta> <descriptiveSystem> <alphaD> <betaD>        *
* <gammaD> <projection> <lambdaProjection> <betaProjection>                 *
* <kappaProjection>                                                         *
*            All arguments are mandatory.                                   *
*            sourceName      name of the source or NULL                     *
*            basisSystem     one number [0,6] among the basisSystemEnum     *
*                            0 GALATIC, galactic                            *
*                            1 EQUATORIAL,mean equatorial                   *
*                            2 APPARENTEQUATORIAL, apparent equatorial      *
*                            3 ECLIPTIC, mean ecliptic                      *
*                            4 APPARENTECLIPTIC, apparent ecliptic          *
*                            5 HADEC, apparent hour angle and declination   *
*                            6 HORIZONTAL                                   *
*            equinoxSystem   one number [0,1] among the equinoxSystemEnum   *
*                            0 J, IAU 1976, FK5, Fricke system              *
*                            1 B, Bessel-Newcomb, FK4 system                *
*            equinoxYear     year of the equinox                            *
*            lambda          source longitude (radian)                      *
*            beta            source latitude (radian)                       *
*            descriptiveSystem one number among the descriptiveSystemEnum   *
*                            0 NODESCRIPTIVE, no descriptive system         *
*                            1 ORIGIN, origin definition of descriptive sys.*
*                            2 POLAR, polar definition of descriptive system*
*                            3 EULER,Euler definition of descriptive system *
*            alphaD          1st angle for descriptive system (radian)      *
*            betaD           2nd angle for descriptive system (radian)      *
*            gammaD          3rd angle for descriptive system (radian)      *
*            projection      one number among the projectionEnum choice     *
*                            0 NOPROJECTION, no projection                  *
*                            1 RADIO, "radio" convention with 1/cos(beta)   *
*                            2 TAN, gnomonic                                *
*                            3 SIN, orthographic                            *
*                            4 STG, stereographic                           *
*                            5 ARC, zenithal equidistant                    *
*                            6 ZEA, zenithal equal area                     *
*                            7 CAR, cartesian / plate caree                 *
*                            8 MER, mercator                                *
*                            9 CEA, cylindrical equal area / Lambert        *
*                            10 GLS, Sanson-Flansteed                       *
*                            11 AIT, Hammer-Aitoff                          *
```

```
 *            lambdaProjection 1st angle for projection (radian)            *
 *            betaProjection   2nd angle for projection (radian)            *
 *            kappaProjection  3rd angle for projection (radian)            *

 !!! sourceName not archived
```

3.2.4 sourceOffsets

```
 * ABSTRACT : Define offsets to be applied to the next source (not the      *
 *            current or active source).                                    *
 *            sourceOffset <xOffset> <yOffset> <systemOffset>               *
 *            The next source has to be already defined.                    *
 *            The 3 arguments are mandatory.                                *
 *            xOffset      offset along x in radians                        *
 *            yOffset      offset along y in radians                        *
 *            systemOffset one number [0,7] among the enum systemOffsetEnum  *
 *                         0 PROJECTION                                     *
 *                         1 DESCRIPTIVE, descriptive coordinates           *
 *                         2 BASIS, basis system                            *
 *                         3 EQUATORIALOFF, mean equatorial J2000.0         *
 *                         4 HADECOFF, (apparent) hour angle and declination *
 *                         5 HORIZONTALTRUE, az with 1/cos(el) factor and el *
 *                         6 HORIZONTALOFF, azimuth and elevation           *
 *                         7 NASMYTH                                        *
 *            systemOffset can be PROJECTION if                             *
 *                                         source.projection != NOPROJECTION *
 *            systemOffset can be DESCRIPTIVE if                            *
 *                                 source.descriptiveSystem != NODESCRIPTIVE *
 *            PROJECTION, DESCRIPTIVE and BASIS are mutually execlusive and  *
 *            the last accepted command with one of these systemOffsets     *
 *            clears the other related offsets.                             *
 *            HORIZONTALTRUE and HORIZONTALOFF are mutually exclusive.      *
```

3.2.5 setNextSubscanTrack

```
 * ABSTRACT : Define a subscan track                                        *
 *            setNextSubscanTrack <time> <xOffset> <yOffset> <systemOffset>  *
 *            <traceFlag> <subscanId>                                       *
 *            The 6 arguments are mandatory.                                *
 *            time       duration of tracking in seconds (double)           *
 *            xOffset      offset along x in radians                        *
 *            yOffset      offset along y in radians                        *
 *            systemOffset one number [0,7] among the enum systemOffsetEnum  *
 *                         0 PROJECTION                                     *
 *                         1 DESCRIPTIVE, descriptive coordinates           *
 *                         2 BASIS, basis system                            *
 *                         3 EQUATORIALOFF, mean equatorial J2000.0         *
 *                         4 HADECOFF, (apparent) hour angle and declination *
 *                         5 HORIZONTALTRUE, az with 1/cos(el) factor and el *
 *                         6 HORIZONTALOFF, azimuth and elevation           *
 *                         7 NASMYTH                                        *
 *            traceFlag  add flag to trace, a number [0,18] among the enum   *
 *                       traceFlagsEnum                                     *
 *            subscanId  subscan identifier (0 to 32 characters)            *
 *            A passive source should be already defined. In case no passive *
 *            source exists, an active source should exists.                *
```

3.2.6 setNextSubscanSlewAzimuth

```
 * ABSTRACT : Define a subscan slew in azimuth.                             *
```

```
 *            setNextSubscanSlewAzimuth <azimuthStart> <azimuthEnd>          *
 *            <elevation> <speed> <traceFlag> <subscanId>                    *
 *            The 6 arguments are mandatory.                                 *
 *            azimuthStart azimuth start position in radians                 *
 *            azimuthEnd   azimuth end position in radians                   *
 *            elevation    elevation in radians                              *
 *            speed        speed in radians/second                           *
 *            traceFlag    add flag to trace, a number [0,18] among the enum *
 *                         traceFlagsEnum                                    *
 *            subscanId    subscan identifier (0 to 32 characters)           *
 *            A passive source should be already defined. In case no passive *
 *            source exists, an active source should exists.                 *
```

3.2.7 setNextSubscanSlewElevation

```
 * ABSTRACT : Define a subscan slew in elevation.                            *
 *            setNextSubscanSlewElevation <elevationStart> <elevationEnd>    *
 *            <azimuth> <speed> <traceFlag> <subscanId>                      *
 *            The 6 arguments are mandatory.                                 *
 *            elevationStart azimuth start position in radians               *
 *            elevationEnd   azimuth end position in radians                 *
 *            azimuth        elevation in radians                            *
 *            speed          speed in radians/second                         *
 *            traceFlag      add flag to trace, a number [0,18] among the    *
 *                           enum traceFlagsEnum                             *
 *            subscanId      subscan identifier (0 to 32 characters)         *
 *            A passive source should be already defined. In case no passive *
 *            source exists, an active source should exists.                 *
```

3.2.8 setNextSubscanOtf

```
 * ABSTRACT : Define a subscan On The Fly.                                   *
 *            setNextSubscanOtf <systemOffset> <subscanId>                   *
 *            The 2 arguments are mandatory.                                 *
 *            systemOffset one number [0,7] among the enum systemOffsetEnum  *
 *                         0 PROJECTION                                      *
 *                         1 DESCRIPTIVE, descriptive coordinates            *
 *                         2 BASIS, basis system                             *
 *                         3 EQUATORIALOFF, mean equatorial J2000.0          *
 *                         4 HADECOFF, (apparent) hour angle and declination *
 *                         5 HORIZONTALTRUE, az with 1/cos(el) factor and el *
 *                         6 HORIZONTALOFF, azimuth and elevation            *
 *                         7 NASMYTH                                         *
 *            subscanId    subscan identifier (0 to 32 characters)           *
 *            A passive source should be already defined. In case no passive *
 *            source exists, an active source should exists.                 *
 *            If systemOffset is equal to PROJECTION or DESCRIPTIVE, the     *
 *            descriptiveSystem or projection have to be already defined in  *
 *            the source.                                                    *
```

3.2.9 setNextSegmentLinear

```
 * ABSTRACT : Define a linear otf segment.                                   *
 *            setNextSegmentLinear <xStart> <yStart> <xEnd> <yEend>          *
 *            <speedStart> <speedEnd> <traceFlag> <subscanId>                *
 *            xStart       start in x (radian)                               *
 *            yStart       start in y (radian)                               *
 *            xEnd         end in x (radian)                                 *
 *            yEnd         end in y (radian)                                 *
 *            speedStart   start with this speed (radian/second)             *
 *            speedEnd     end with this speed (radian/second)               *
 *            traceFlag    add flag to trace                                 *
```

```
 *            subscanId    segment identifier (0 to 32 characters)           *
 *            Adds a segment to an otf subscan identified by subscanId as far*
 *            this subscan has been defined and still exits                  *

!!! traceFlag not implemented
```

3.2.10 setNextSegmentCircle

```
 * ABSTRACT : Define a circular otf segment.                                 *
 *            setNextSegmentLinear <xStart> <yStart> <xEnd> <yEend>          *
 *            <turnAngle> <speedStart> <speedEnd> <traceFlag> <subscanId>    *
 *            xStart       start in x (radian)                               *
 *            yStart       start in y (radian)                               *
 *            xEnd         end in x (radian)                                 *
 *            yEnd         end in y (radian)                                 *
 *            turnAngle    turn angle (radian)                               *
 *            speedStart   start with this speed (radian/second)             *
 *            speedEnd     end with this speed (radian/second)               *
 *            traceFlag    add flag to trace                                 *
 *            subscanId    segment identifier (0 to 32 characters)           *
 *            Adds a segment to an otf subscan identified by subscanId as far*
 *            this subscan has been defined and still exits                  *

!!! traceFlag not implemented
```

3.2.11 setNextSegmentCurve

```
 * ABSTRACT : Define a Bezier curve otf segment.                             *
 *            setNextSegmentLinear <xStart> <yStart> <xEnd> <yEend>          *
 *            <turnAngle> <speedStart> <speedEnd> <traceFlag> <subscanId>    *
 *            xStart       start in x (radian)                               *
 *            yStart       start in y (radian)                               *
 *            xEnd         end in x (radian)                                 *
 *            yEnd         end in y (radian)                                 *
 *            xCpStart     x of control point at start (radian)              *
 *            yCpStart     y of control point at start (radian)              *
 *            xCpEnd       x of control point at end (radian)                *
 *            yCpEnd       y of control point at end (radian)                *
 *            speedStart   start with this speed (radian/second)             *
 *            speedEnd     end with this speed (radian/second)               *
 *            traceFlag    add flag to trace                                 *
 *            subscanId    segment identifier (0 to 32 characters)           *
 *            Adds a segment to an otf subscan identified by subscanId as far*
 *            this subscan has been defined and still exits                  *

!!! traceFlag not implemented
```

3.2.12 prepareObservation

```
 * ABSTRACT : Request to prepare the next scan (next source).                *
 *            Syntax:                                                        *
 *            prepareObservation <when>                                      *
 *            The argument is mandatory.                                     *
 *            when       ISO 8601 date                                       *
 *            The command cleans all subscans and segments still connected   *
 *            to the active scan before switching to the next source.        *
 *            The command "prepare" has a sense if the next source (passive) *
 *            has already been defined. If the next source is not yet        *
 *            defined, the observation mode is set to STOP.                  *
 *            The command resets the observation start time and stop time    *
```

3.2.13 startObservation

```
 * ABSTRACT : Request to start a scan.                                      *
 *            Syntax                                                        *
 *            startObservation <when>                                       *
 *            The argument is mandatory.                                    *
 *            when       ISO 8601 date                                      *
 *            When the observation mode becomes READY and the specified date *
 *            <when> is past, then the observation starts to proceed through *
 *            the subscans. The mode is set to RUN. If there is no subscan   *
 *            the startObservation time slot is forgotten and the observation*
 *            start time is reset.                                          *
```

3.2.14 haltObservation

```
 * ABSTRACT : Request to stop a scan.                                       *
 *            Syntax                                                        *
 *            startObservation <when>                                       *
 *            The argument is mandatory.                                    *
 *            when       ISO 8601 date                                      *
 *            The command requests for the specified time to halt an        *
 *            observation and to remove the aborted scan                    *
```

3.2.14.1 haltAfterCurrentSubscan

```
 * ABSTRACT : Request to halt the current subscan at the time specified.    *
 *            Syntax                                                        *
 *            haltAfterCurrentSubscan <when>                                *
 *            The argument is mandatory.                                    *
 *            when       ISO 8601 date                                      *
 *            When the observation mode is RUN and the specified date <when> *
 *            is past, then the observation finishes the current subscan and *
 *            does not start the next subscan.                              *
```

3.2.14.2 resumeObservation

```
 * ABSTRACT : Request to resume (start) a scan.                             *
 *            Syntax                                                        *
 *            resumeObservation <when>                                      *
 *            The argument is mandatory.                                    *
 *            when       ISO 8601 date                                      *
 *            The observation mode should be READY and when the specified   *
 *            date <when> is past, then the observation starts to proceed   *
 *            through the subscans.                                         *
 *            The mode is set to RUN. If there is no subscan the resumeScan  *
 *            time slot is forgotten and the observation start time is reset.*
Same action can be achieved with the command startObservation
```

3.2.14.3 endSubscan

```
 * ABSTRACT : Request to end current subscan.                               *
```

3.2.15 setAzimuthWrap

```
 * ABSTRACT : Set the azimuth wrap                                          *
```

```
 *           setAzimuthWrap wrap                                          *
 *           wrap   one number [0, 2] among the azimuthWrapEnum           *
 *                  0 LOW,     preset into range [60deg, 420deg]          *
 *                  1 HIGH,    preset into range [100deg, 460deg]         *
 *                  2 NEAREST, move to nearest position when preset       *
```

3.2.16 setPointingParameters

```
 * ABSTRACT : Define the pointing parameters                              *
 *           setPointingParameters p1, p2, p3, p4, p5, p7, p8, p9, rxh0,  *
 *           rxve                                                         *
 *           p1           Azimuth encoder zero point error                *
 *           p2           Collimation error in azimuth                    *
 *           p3           Collimation error of axes                       *
 *           p4           Inclination 1                                   *
 *           p5           Inclination 2                                   *
 *           p7           Elevation encoder zero point error              *
 *           p8           Bending term 1                                  *
 *           p9           Bending term 2 (Bernd Harald)                   *
 *           rxho         Horizontal receiver offset in Nasmyth           *
 *           rxve         Vertical receiver offset in Nasmyth             *
 *           The 10 arguments are mandatory.                              *
```

3.2.17 setRefractionParameters

```
 * ABSTRACT : Define the refraction parameters                            *
 *           setRefractionParameters tAmbient pAtm relHumidity wavelength *
 *                                                                        *
 *           tAmbient    ambient temperature at the observer [K]          *
 *           pAtm        atmospheric pressure at the observer [mB]        *
 *           relHumidity relative humidity at the observer range 0 to 1   *
 *           wavelength  effective wavelength of the source [micron]      *
 *           The 4 arguments are mandatory.                               *
```

3.2.18 setTraceRate

```
 * ABSTRACT : Set the trace rate.                                         *
 *           Syntax                                                       *
 *           setTraceRate rate                                            *
 *           rate number of traces collected per second                   *
 *           rate is a power of 2 between 1 and 128                       *
```

setTraceRate always records the number power of 2 less or equal to the requested rate.
Right now, the rate is limited to 16

3.2.19 sunAvoidance

```
 * ABSTRACT : Enable/disable the sun avoidance and eventually sets the sun  *
 *           avoidance radius.                                            *
 *           Syntax                                                       *
 *           sunAvoidance <validation> [<radius>]                         *
 *           validation  0 (disable avoidance) 1 (enable avoidance)       *
 *           radius        avoidance radius in radian                     *
 *           the first argument (validation) is mandatory.                *
 *           the second argument is optional. If the radius is omitted, its *
 *           current value in the observation shared memory area stays    *
 *           unchanged.                                                   *
```

3.2.20 sunCoordinates

```
 * ABSTRACT : Set the apparent coordinates, RA and dec, of the sun.          *
 *            Syntax                                                         *
 *            sunCoordinates <RA> <dec>                                      *
 *            RA   Sun right ascension in radian                             *
 *            dec  Sun apparent declination in radian                        *
```

3.2.21 zenithAvoidance

```
 * ABSTRACT : Enable/disable the zenith avoidance circle (to protect our     *
 *            hardware of cloudsat emission) and eventually sets the distance*
 *            of this circle from the astro zenith.                          *
 *            Syntax                                                         *
 *            zenithAvoidance <validation> [<distance>]                      *
 *            validation  0 (disable avoidance) 1 (enable avoidance)         *
 *            distance    avoidance distance from zenith in radian           *
 *            the first argument (validation) is mandatory.                  *
 *            the second argument is optional. If the distance is omitted,   *
 *            its current value in the observation shared memory area stays  *
 *            unchanged.                                                     *
```

3.2.22 Initialization and test commands

3.2.22.1 config

```
 * ABSTRACT: Initialize the share memory with data specific to the antenna   *
 *           found in file /control/config.30m                               *
```

The config.30m looks like this:

…

```
az.vReference   0
az.cXKp         2048
az.cXKp         8192                  6Nov03
```

…

i.e. az. or el. followed by an element of the struct s_axes declared in s_anntea.h and then its configuration value. The name and its value are separated by one or more blancks. Any more string appended on the line are comments. The declaration lines can be written in any order in the file and any added prefix character at the beginning of the line (like the character #) will change the line into a comment line.

3.2.22.2 configPointing

```
 * ABSTRACT: Initialize the share memory with the pointing parameters found  *
 *           in the file /control/pointing.30m                               *
```

The pointing.30 look like this:

```
 -4.740  p1 (arc-sec) P1
 -8.840  p2 (arc-sec) P2
  2.840  p3 (arc-sec) P3
 -3.000  p4 (arc-sec) P4
-16.000  p5 (arc-sec) P5
 -6.120  p7 (arc-sec) P7
-84.500  p8 (arc-sec) P8
-24.660  p9 (arc-sec) P9
  0.0    rxho (arc-sec) RX_HOR
```

```
  0.0    rxve (arc-sec) RX_VERT
  0.002  refract3 THIRD_REFRACT
 1000    subref.rotationZero ZERO_POL
145.3    subref.rotationPhi0 (deg) PHI_POL
 26.58   subref.rotationRadius (deg) EPSILON_POL
  2.0    encoder.sinCol (arc-sec) COL_SINUS
 -0.3    encoder.cosCol (arc-sec) COL_COSINUS
```

It is a fix order/format file. Only the values in the first column may be changed. The units and names following the values are considered as comments and are not used by the command configPointing.

3.2.22.3 initAntenna

```
 * ABSTRACT : Initialize (some VME modules) and the antenna drive shared     *
 *            memory area declared with struct s_antenna                     *
```

3.2.22.4 initObservation

```
 * ABSTRACT : Initialize the shared memory segment "PICO" declared with      *
 *            stuct s_observation                                            *
```

3.2.22.5 dmpAntenna

```
 * ABSTRACT: Dump the antenna drive shared data memory area defined in       *
 *           int_ant with the stuct s_antenna                                *
```

3.2.22.6 dmpObservation

```
 * ABSTRACT : Dump the shared memory segment "PICO" defined with the struct  *
 *            s_observation                                                  *
```

3.2.22.7 horizon

```
 * ABSTRACT : Request a position in horizon coordinate system                *
 *            horizon [az] [el]                                              *
 *            az  azimuth position in degrees                                *
 *            el  elevation position in degrees                              *
 *            A command without argument will prompt for the azimuth and the *
 *            elevation but also for the velocities for the 2 axes.          *
 *            A command with only one argument means to set the az/el to the *
 *            current values.                                                *
```

Side effect: Set to 1 the flag `remote` defined in s_antennna.h

3.2.22.8 record

```
 * ABSTRACT : command to log a list a variables defined in s_antenna.h       *
 *      Usage: record [argument ...]                                         *
 *       argument ... :                                                      *
 *        reset           : Clear the list of variable names                 *
 *        add var_name    : Add a variable to the list to be recorded        *
 *                  var_name is the name of a variable defined in s_antenna.h*
 *        delete var_name : delete a variable in the list to be recorded     *
 *        list            : list the name of the variables of the list       *
 *        start           : Start recording                                  *
```

```
 *       stop          : Stop recording at 128Hz                       *
 *       save          : Display the recorded values                   *
 *       record        : Test. Record once all variables of the list.  *
 *                       Stop the periodic recording first.            *
```

3.2.22.9 scope

```
 * ABSTRACT : Command used to output any integer variable defined in        *
 *            s_antenna.h on one of the 2 spare 16 bit DACs.                *
 *            scope without argument displays the usage                     *

Usage: scope dac_nber var_name factor offset
   dac_nber  a dac channel: Either 6 or 7
   var_name  the name of a variable defined in s_antenna.h
   factor is defined in decimal and offset in hexadecimal.
The default values for factor and offset are 0.
The variable * 2^factor + offset is sent to the dac channel dac_nber of the
VMIVME4116. It's a 16bit DAC. Voltage = (variable *2^factor + offset)*10/2^15
The variable is assumed to be a signed int.
```

3.2.22.10 setEncoder

```
 * ABSTRACT : Select 1 or both incremental encoders to calculate the axis    *
 *            positions, azimuth and elevation                               *
 *            setEncoder azSetting elSetting                                 *
 *            azSetting    1|2|3|12 to select 1st, 2nd or both encoders      *
 *            elSetting    1|2|3|12 to select 1st, 2nd or both encoders      *
```

3.2.22.11 stop

```
 * ABSTRACT : Command to request to stop the antenna as quick as possible    *
```

Sequence: The axes `request` parameters are set first to R_STOP to slow down and stop eventually their rotations and then the axes are requested to clamp to the actual achieved positions. At completion the `request` parameters should be found equal to R_TRACK.
Side effect: Clear the flag `remote` defined in s_antenna.h

3.2.22.12 track

```
 * ABSTRACT : Request the tracking mode to be BASIC or CASCADE               *
 *            track [azTrackingMode] [elTrackingMode]                        *
 *            azTrackingMode 0 basic control, 1 cascade control              *
 *            elTrackingMode 0 basic control, 1 cascade control              *
 *            When one or both arguments are missing, their values are       *
 *            requested by prompting                                         *
```

3.2.22.13 unlock

```
 * ABSTRACT : without argument the command unlocks the antenna               *
 *            with one (or more argument) locks the antenna                  *
 *            Before typing unlock [arg], type the command stop              *
```

The normal sequence to free the antenna is:

```
$ stop          # force the drive (az and el) dac’s to zero.
$ unlock        # request to free the axes.
                # unlock should completes with the messages:
```

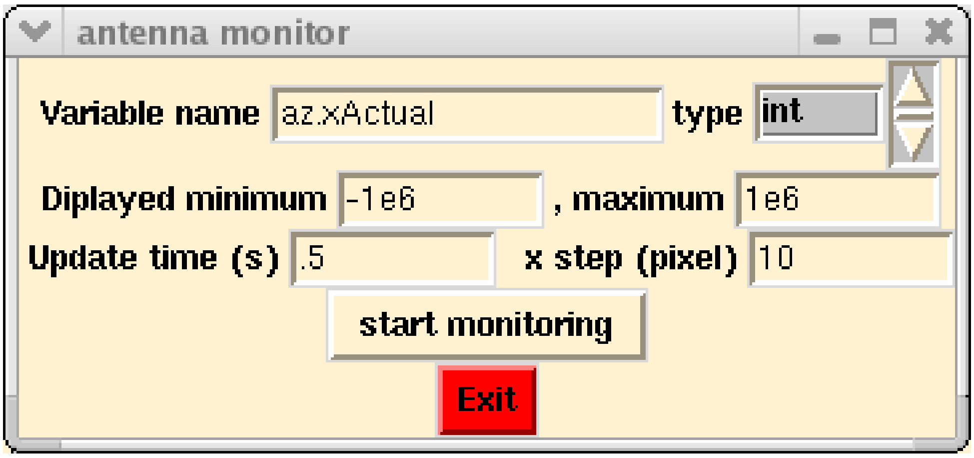
```
                # "az is now free" and "el is now free"
$ stop          # clamp the antenna axes on their current positions.
```

The normal sequence to return the antenna control to the operators:

```
$ stop          # force the drive (az and el) dac's to zero.
$ unlock NO     # request to lock the axes.
```

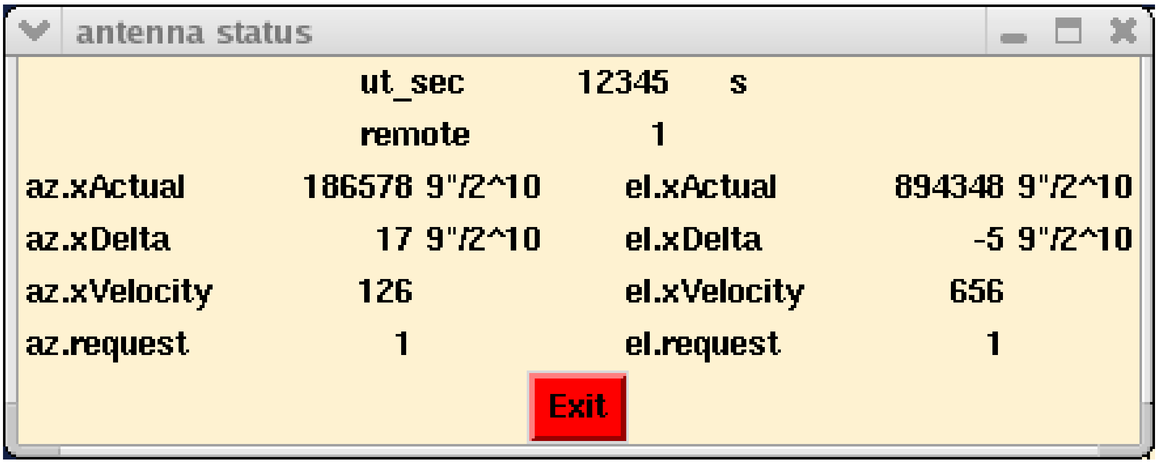
3.2.22.14 Python utilities

monitor.py



The variable name is the name of a member of the structure s_antenna declared in s_antenna.h. The variable should be of type int or float or double. Use the scrolling list box to display the type and then click on the displayed type to select it. The selected type should be in gray background.
A pop up plot widget is created upon "start monitoring" activation.
The variable plot is updated every "updated time" on the right side of the plot area and is cropped between selected "displayed minimum" and "maximum". Every "updated time" a horizontal segment of "x step (pixel)" is diplayed on the right side of the plot area while pushing the curve to the left. As a consequence, as the plot area is 300 pixel wide, the plot covers the time period of the last 300 * "update time" / "x step" seconds.
Examples of variable names: az.actual, ut_sec, el.xDelta.

status.py



This command creates a display updated every second. It is very simple to edit status.py in order to display more variables from the common area.

sun.py
This command creates a windows which shows the antenna position, az, el, in the ranges [60, 460] ,[0, 90] updated every second. The sun avoidance circle is projected in this coordinate system. Its radius is the angle set in the common area and the circle projection is updated every 100s.
The button “clean” deletes the oldest 1 minute period of the antenna trajectory displayed in this window.

3.3 Implementation in evItSlow

!!!Only linear segments for OTF subscans.

projectionToNative()
sourceOffsetProjection are not applied in case of subscan OTF, otherwise they are added.
p_source->lambda and p_source->beta are not used if projection is neither NOPROJECTION nor RADIO.

nativeToDescriptive()
sourceOffsetDescriptive are not applied in case of subscan OTF, otherwise they are added.

!!!Only RADIO is implemented

descriptiveToBasis()
ORIGIN
POLAR
EULER
sourceOffsetBasis are not applied in case of subscan OTF, otherwise they are added.

!!!Everything implemented:

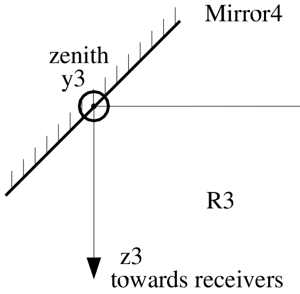
basisToHorizon()
EQUATORIAL
APPARENTEQUATORIAL
HORIZONTAL
SourceOffsets’ xOffset and yOffset with systemOffset equal to HORIZONTALTRUE or HORIZONTALOFF are applied except when HORIZONTALTRUE or HORIZONTALOFF are requested as systemOffset for the subscans OTF or TRACK. In the later case the setNextSubscan’s xOffset and yOffset are applied.
SourceOffsets’ xOffset and yOffset with systemOffset equal to NASMYTH are applied except when NASMYTH is requested as systemOffset for the subscans OTF or TRACK. In the later case the setNextSubscan’s xOffset and yOffset are applied.

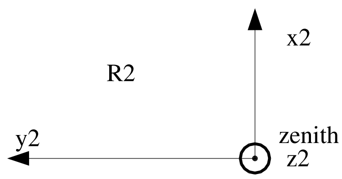
!!! GALACTIC, ECLIPTIC, APPARENTECLIPTIC, HADEC are not implemented

3.3.1.1 Nasmyth offsets

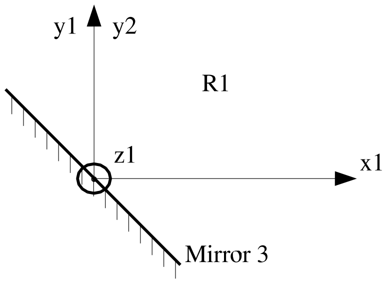
Relation between Nasmyth x and y offsets and azimuth/elevation corrections:

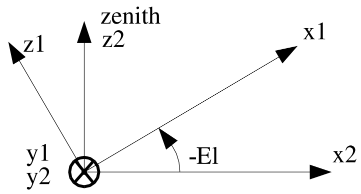
Let’s define the coordinate system (C.S.) R2: x2y2 horizontal, y2 along the elevation axis in the direction of the source beam reflected by mirror 3. y2 is towards mirror 4. z2 towards zenith.



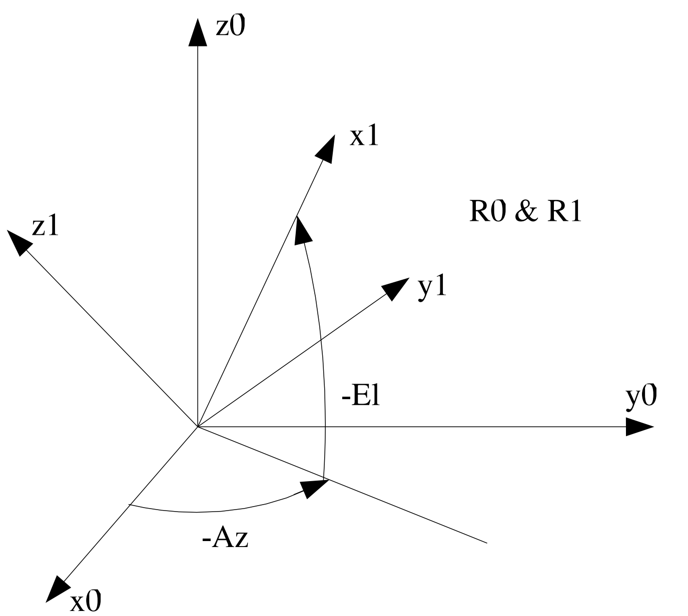


C.S. R1: x1 towards the source, angle(x0,x1)=El, y1==y2





C.S. R3: x3 y3 horizontal, x3 along the elevation axis, from mirror4 to mirror3, y3 towards zenith, z3 towards the receivers. x3 and y2 are in opposite directions.
C.S. R0: Horizon coordinate system. R1 is deduced from R0 by 2 rotations: -Az around z and -El around y, (Az positive clockwise counted around an axis towards zenith, Az=0 north, Az=Π/2 east).



Note: Vector W, W0 its representation in R0 and W1 its representation in R1. W0 = [W0x,W0y,W0z]' and W1 = [W1x,W1y,W1z]' (' notation for transpose)
W1=T/x(a)W0 with T transformation matrix corresponding to the rotation of R1/R0 of the angle a around axis x

```
T/z(a) = [ cosa sina 0]  T/x(b) = [1   0    0  ]  T/y(c) = [cosc 0 -sinc]
         [-sina cosa 0]           [0  cosb sinb]           [ 0   1   0  ]
         [  0    0   1]           [0 -sinb cosb]           [sinc 0  cosc]
```

Let's call V3 the unit verctor close to axis z3 and defined by the offsets xOf and yOf.
Its representation in R3: V33~=[xOf, yOf, 1]', valid for small offsets (approximation of sin(a) with a and cos(a) with 1).

[$\sqrt{2}$ /2, 0, $\sqrt{2}$ /2] is the unit vector perpendicular to mirror 4.

Let's call V2 the image of V3 in mirror 4. Its representation in R3:
V23=[-1, yOf, -xOf]'
Its representation in R2:
V22=[xOf, 1, yOf]'
Its representation in R1
```
V21 = T/y(-El) V22
```

[$\sqrt{2}$ /2, $\sqrt{2}$ /2, 0] is the unit vector perpendicular to the mirror3 in R1
Let's call V1 the image of V2 in mirror 3.
Its representation in R1
V11=[-V21y, -V21x, V21z]'
Lets call V0=-V1 vector towards the sky
V01=[V21y, V21x, -V21z]'
```
V01 = [1, -cosEl*xOf+sinEl*yOf, sinEl*xOf-cosEl*yOf]'
V00 = T/z(Az) T/y(El) V12

V00= [cosAz(cosEl-sinEl(sinEl*xOf-cosEl*yOf)) + sinAz(cosEl*xOf+sinEl*yOf)]
     [sinAz(-cosEl+sinEl(sinEl*xOf-cosEl*yOf)) + cosAz(cosEl*xOf+sinEl*yOf)]
     [sinEl+cosEl(sinEl*x0F-cosEl*yOf)]

VOO should be equal to
     [cos(Az+dAz)cos(El+dEl)]
     [-sin(Az+dAz)cos(El+dEl)]
     [sin(El+dEl)]
=
     [cosAz(cosEl-sinEl*dEl) - sinAz*cosEl*dAz]
     [sinAz(-cosEl+sinEl*dEl) - cosAz*cosEl*dAz]
     [sinEl+cosEl*dEl]

=> dEl = sinEl*xOf-cosEl*yOf
-cosEL*dAz = cosEL*xOf+sinEl*yOf
```
As a consequence, in order to aim to a source given with its coordinates Az and El, with nasmyth offsets xOfs and yOfs, the antenna drive angles should be offseted of
dAz = (cosEl * xOf + sinEl *yOf)/cosEl and
dEl = -sinEl * xOf + cosEl * yOf

3.3.2 Pointing correction

The easiest way to list the terms added up to build the axis corrections is to print here a snippet of the code:
```
    p_observation->pointing.azCorrection = (p2
        + p_observation->pointing.subref.commanded4 * FAC_FOCUS // subreflector
        + p_observation->pointing.subref.commanded6 * SEC_FOC   // subreflector
        + radius * (sin(phi + phi0) - sin(phi0))                // subreflector
        + (p1 + rxho) * cosEl
        + (p3 + p4 * cosAz + p5 * sinAz + rxve)* sinEl
        + p_observation->pointing.encoder.sinCol * sin2Az
        + p_observation->pointing.encoder.cosCol * cos2Az
        )/cosEl;
```
and
```
    p_observation->pointing.elCorrection = p7
        + p_observation->pointing.subref.commanded3 * FAC_FOCUS // subreflector
        + p_observation->pointing.subref.commanded7 * SEC_FOC   // subreflector
        + radius * (cos(phi + phi0) - cos(phi0))                // subreflector
        - p4 * sinAz
```

```
        + p5 * cosAz
        + (p8 + rxve) * cosEl
        + (p9 - rxho) * sinEl
        + refraction * DPI /180. /3600.;
```

with `p1, p2, p3, p4, p5, p7, p8, p9, rxho` and `rxhe` correction parameters, sums of configuration terms (see the command `configPointing` and the file `pointing.30m`) and run time terms set by the users with the command `setPointingParameters.`

```
    phi = (p_observation->pointing.subref.rotationEncoder -
           p_observation->pointing.subref.rotationZero) * SUBREF_ROT_FAC;
    phi0 = p_observation->pointing.subref.rotationPhi0;
    radius = p_observation->pointing.subref.rotationRadius;
```

`command3, command4, command6, command7` and `rotationEncoder` are values calculated in the subreflector module and transmitted through VME memory to this program.
`rotationZero, rotationPhio, rotationRadius, sinCol` and `cosCol` are parameters defined in the file `pointing.30m` (see the command `configPointing`).

DPI is equal to Π.
FAC_FOCUS, SET_FOC and `SUBREF_ROT_FAC are defined in s_observation.h:`
…

```
#define FAC_FOCUS 7.485001e-8 /* FAC_FOCUS(JBS)0.015438924 * PI /180 /3600 */
#define SEC_FOC 9.696273e-8 /* SEC_FOC(JBS)0.02 * PI /180 /3600 */
#define SUBREF_ROT_FAC 8.726646e-4  /* POL_FAC(JBS)0.05 * PI /180 */
```

`cosAz=cos(azAstro), sinAz=sin(azAstro), cos2Az=cos(2*azAstro), sin2Az=sin(2*azAstro), cosEl=cos(elAstro), sinEl=sin(elAstro)` and `cotEl=cosEl/sinEl`

Refraction calculation (`refraction` in the elevation correction term):
Code snippet:

```
    /* tKelvin: ambient temperature in degree Kelvin */
    tKelvin = p_observation->pointing.refraction.tAmbient;
    /* tCelcius: ambient temperature in degree Celcius */
    tCelcius = tKelvin - 273.15;
    exponent = (7.45 * tCelcius) / (235. + tCelcius);
    /* saturated: Saturated water vapor partial pressure in mb */
    saturated = 6.1 * pow(10, exponent);
    /* waterPartial: Actual water vapor partial pressure in mb */
    waterPartial = p_observation->pointing.refraction.relHumidity * saturated
        /100;
    /* pressure: Actual partial pressure in mb, excluding water vapor */
    pressure = p_observation->pointing.refraction.pAtm - waterPartial;
    /* waterPartial: Actual water vapor partial pressure in mm of Hg */
    waterPartial *= 0.75006;
    /* pressure: Actual partial pressure in mm of Hg, excluding water vapor */
    pressure *= 0.75006;
    refractionIndex = (0.0001034 * pressure +
                       (0.0000958 + 0.5 / tKelvin) * waterPartial) /
        tKelvin * 206264.8;
    refraction = refractionIndex * cotEl *
        (1 - p_observation->pointing.refraction.refract3 * cotEl * cotEl);
```

With `tAmbient, pAtm` and `relHumidity`, refraction parameters set to default values (273., 710., 0.) with the command `initObservation` and set to actual values by the users with the command `setRefractionParameter`.
`refraction` is exceptionally expressed in seconds of arc.

3.4 Data logging

Every second, different blocks of data are logged by calling the function writeAntTrace().

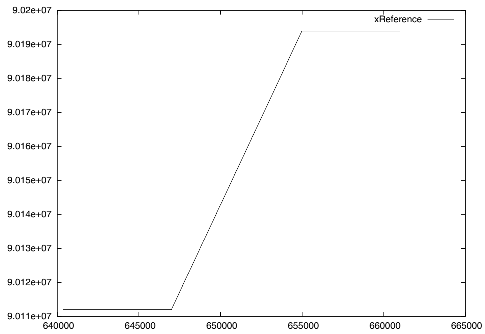
One block of type AntennaTraceSlowS holds observation parameters which need to be logged only once per second.
The other blocks are collected at a faster rate but are logged all together only every second. An argument in the function writeAntTrace gives the rate which is 128 blocks per second at maximum. Those blocks are of type AntennaTraceFastS.

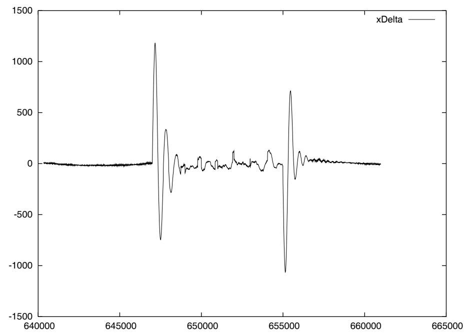
The type AntennaTraceSlowS is a structure, which has the elements xOffset and yOffset among other parameters. In case of a projection (i.e. `projection` different of **`NOPROJECTION`** in the **`source`** command and in the case of a **`setNextSubscanTrack`** or a **`setNextSubscantOtf`** command, `systemOffset` equal to **`PROJECTION`**) these offsets are the x/y positions, possibly derived from the segment definitions or directly from the subscan definition command, but in any case not affected by the transformation to the projection's native system.
In the other cases they are the longitude and latitude offsets added to the longitude and latitude angles in the systemOffset specified in the subscan commands.

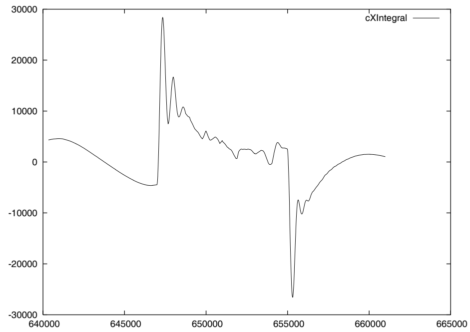
4 Some Results

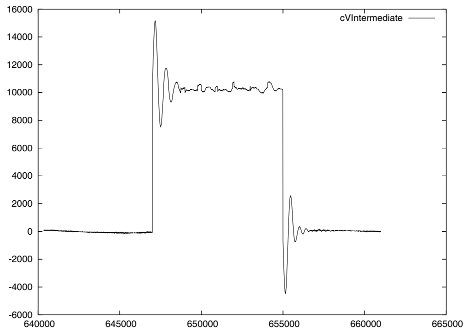
4.1 Tracking in cascade mode

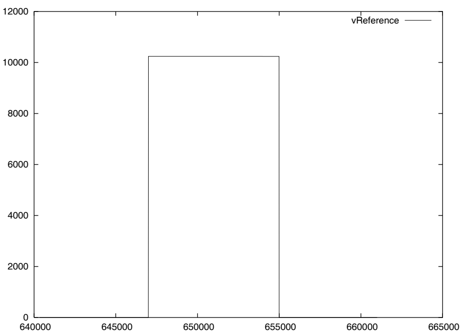
4.1.1 Azimuth step from 220deg to 220.2deg in cascade mode

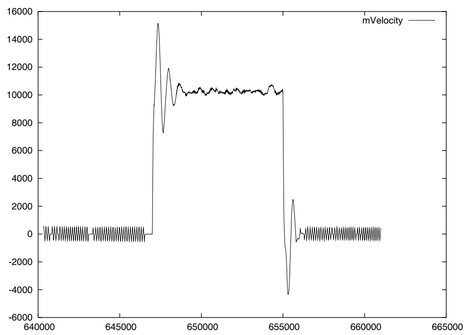


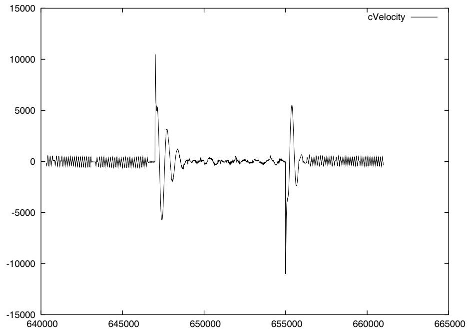


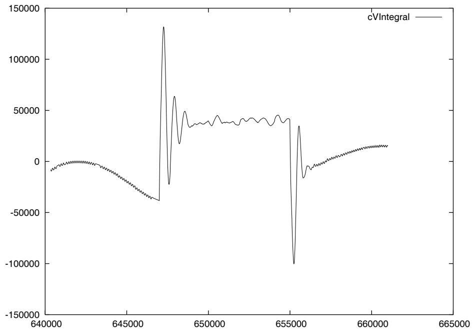


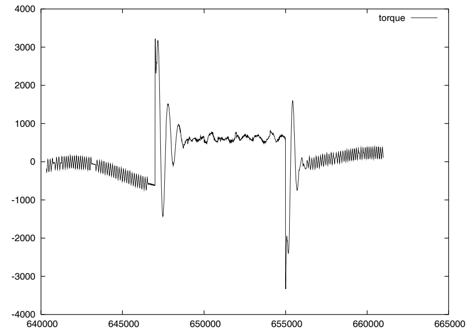


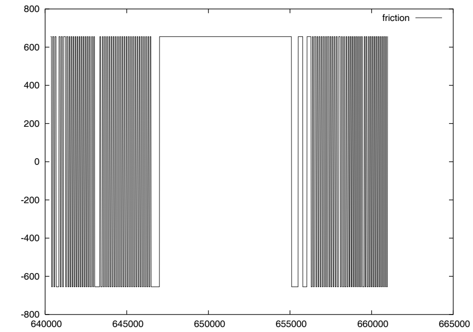


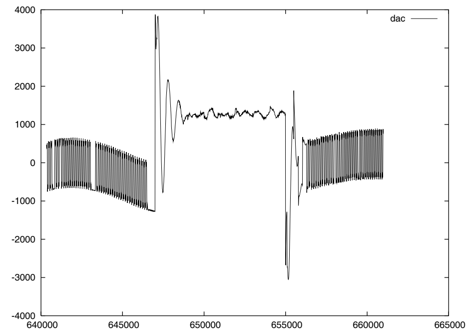




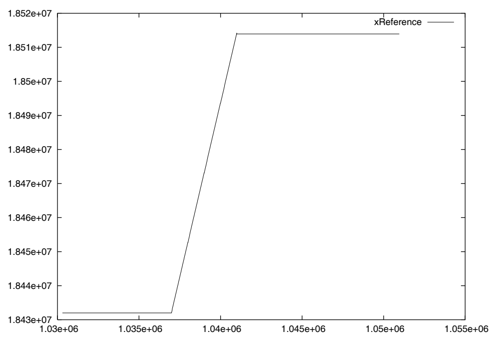


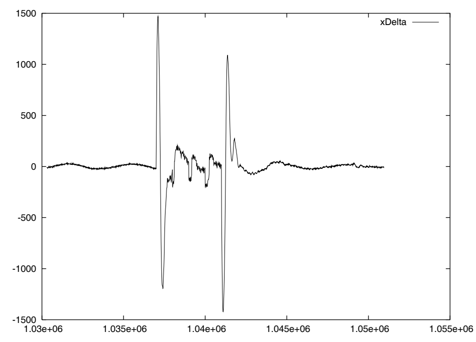


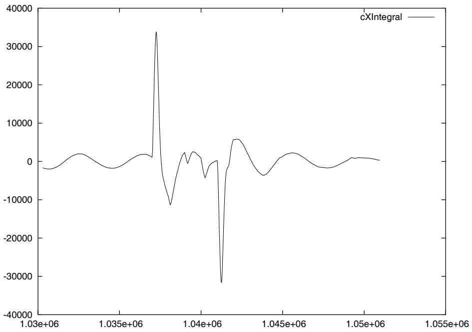


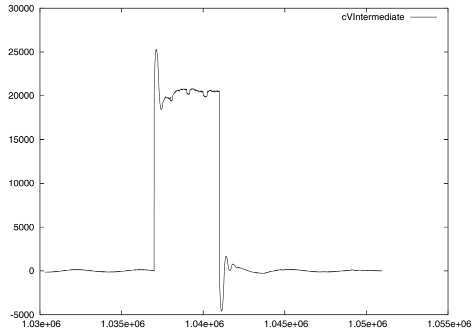


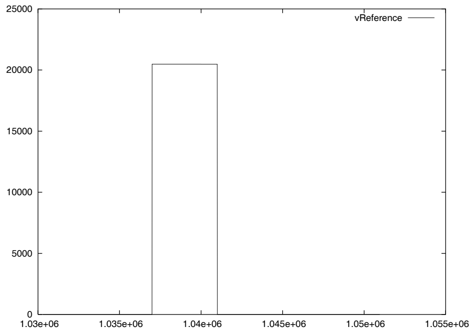
4.1.2 Elevation step from 45deg to 45.2deg in cascade mode

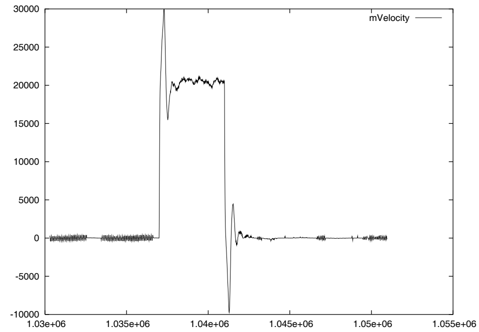


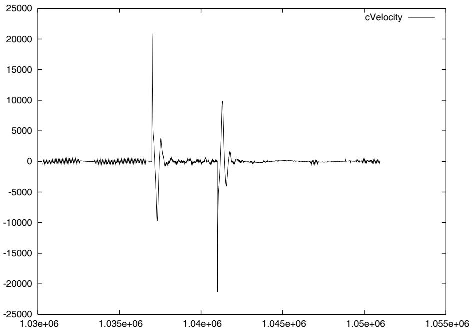


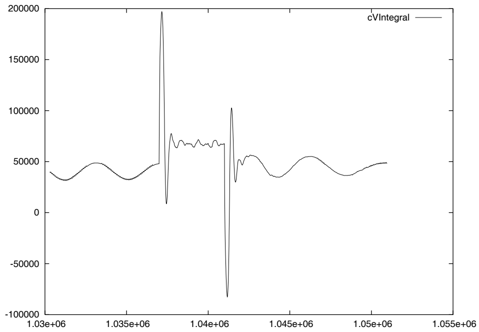

cXIntegral
40000
30000
20000
10000
0
-10000
-20000
-30000
-40000
1.03e+06
1.035e+06
1.04e+06
1.045e+06
1.05e+06
1.055e+06

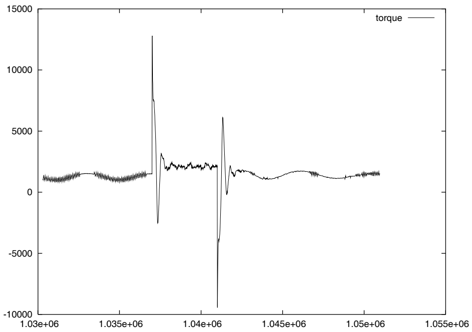

cVIntermediate
30000
25000
20000
15000
10000
5000
0
-5000
1.03e+06
1.035e+06
1.04e+06
1.045e+06
1.05e+06
1.055e+06

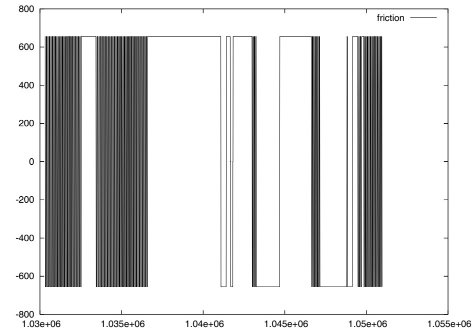

vReference
25000
20000
15000
10000
5000
0
1.03e+06
1.035e+06
1.04e+06
1.045e+06
1.05e+06
1.055e+06

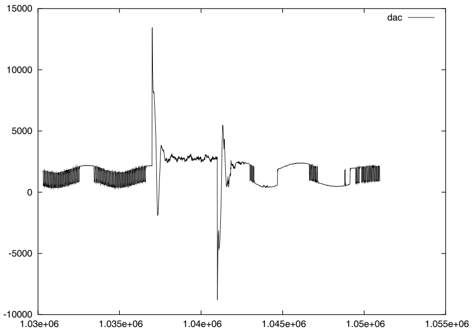

mVelocity
30000
25000
20000
15000
10000
5000
0
-5000
-10000
1.03e+06
1.035e+06
1.04e+06
1.045e+06
1.05e+06
1.055e+06


cVelocity
25000
20000
15000
10000
5000
0
-5000
-10000
-15000
-20000
-25000
1.03e+06
1.035e+06
1.04e+06
1.045e+06
1.05e+06
1.055e+06


cVIntegral
200000
150000
100000
50000
0
-50000
-100000
1.03e+06
1.035e+06
1.04e+06
1.045e+06
1.05e+06
1.055e+06


torque
15000
10000
5000
0
-5000
-10000
1.03e+06
1.035e+06
1.04e+06
1.045e+06
1.05e+06
1.055e+06

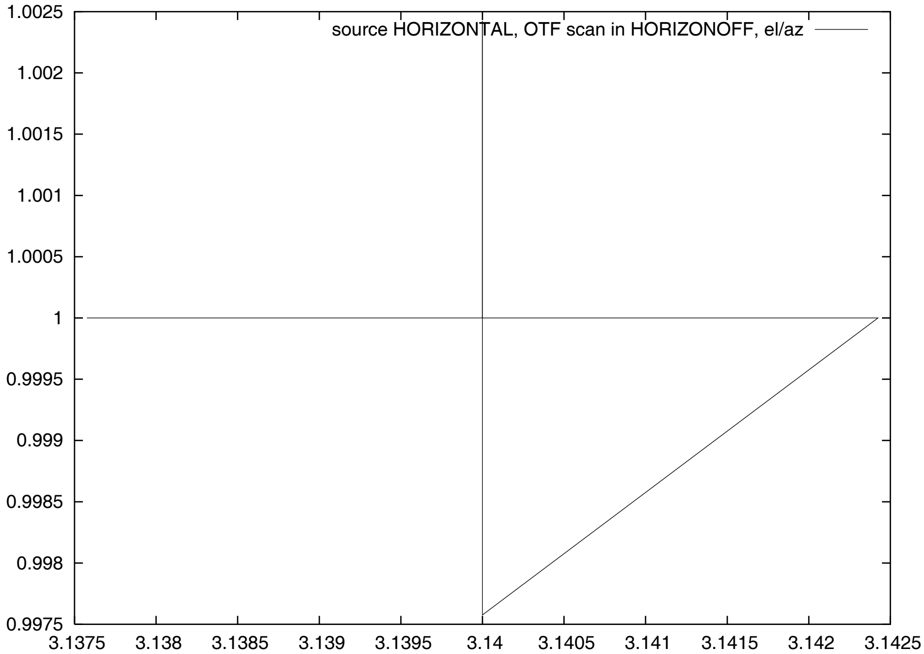

friction
800
600
400
200
0
-200
-400
-600
-800
1.03e+06
1.035e+06
1.04e+06
1.045e+06
1.05e+06
1.055e+06


dac
15000
10000
5000
0
-5000
-10000
1.03e+06
1.035e+06
1.04e+06
1.045e+06
1.05e+06
1.055e+06

4.2 On the fly scans

4.2.1 Source HORIZONTAL, OTF scan in HORIZONTALOFF

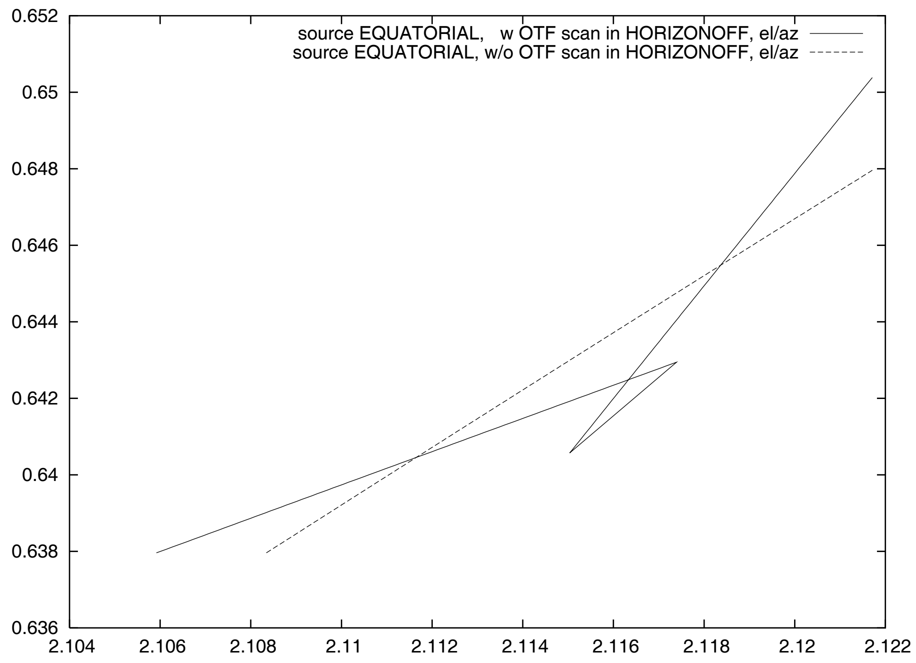
```
bin/source pi-one 6 0 2000 3.14 1 0 0 0 0  0 0 0 0
bin/setNextSubscanOtf 6 2004-04-29.1111.1
bin/setNextSegmentLinear -0.002424 0 0.002424 0 0.00004848 \
 0.00004848 1 2004-04-29.1111.1.1
bin/setNextSubscanOtf 6 2004-04-29.1111.2
bin/setNextSegmentLinear 0 -0.002424 0 0.002424 0.00004848 \
 0.00004848 1 2004-04-29.1111.2.1
```

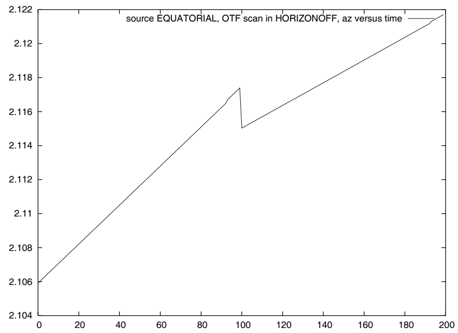


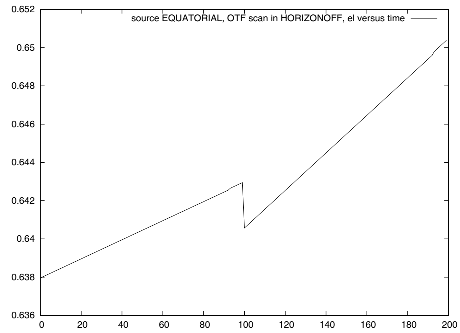
4.2.2 Source EQUATORIAL, OTF scan in HORIZONTALOFF

```
bin/source 0736+017 1 0 2000 0.49276698 0.03025334  0 0 0 0  0 0 0 0
bin/setNextSubscanOtf 6 2004-04-29.1111.1
bin/setNextSegmentLinear -0.002424 0 0.002424 0 0.00004848 \
 0.00004848 1 2004-04-29.1111.1.1
bin/setNextSubscanOtf 6 2004-04-29.1111.2
bin/setNextSegmentLinear 0 -0.002424 0 0.002424 0.00004848 \
 0.00004848 1 2004-04-29.1111.2.1

    MJD 12541.341956
    LST 6.015648
```



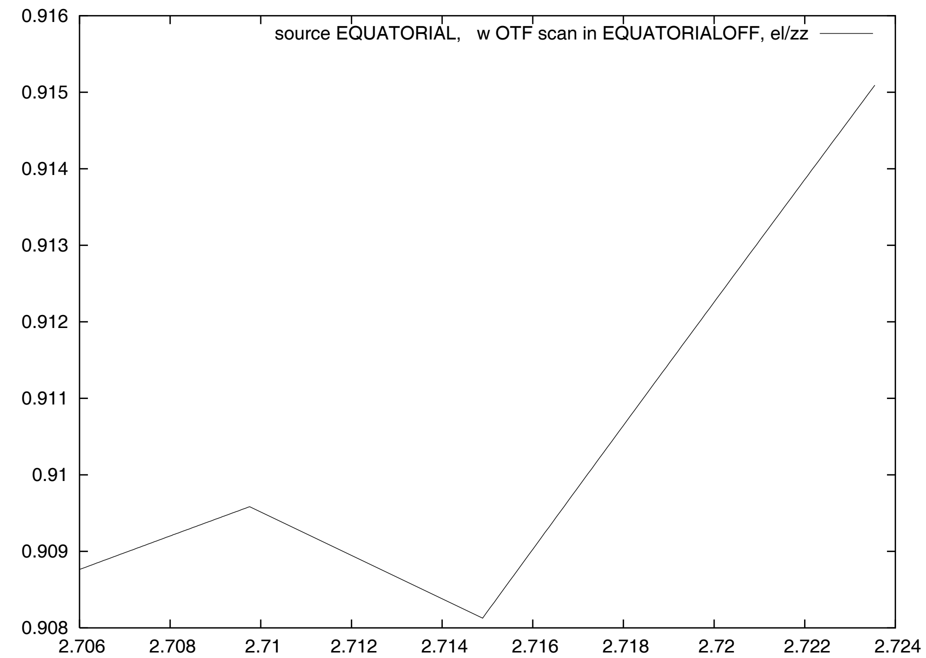


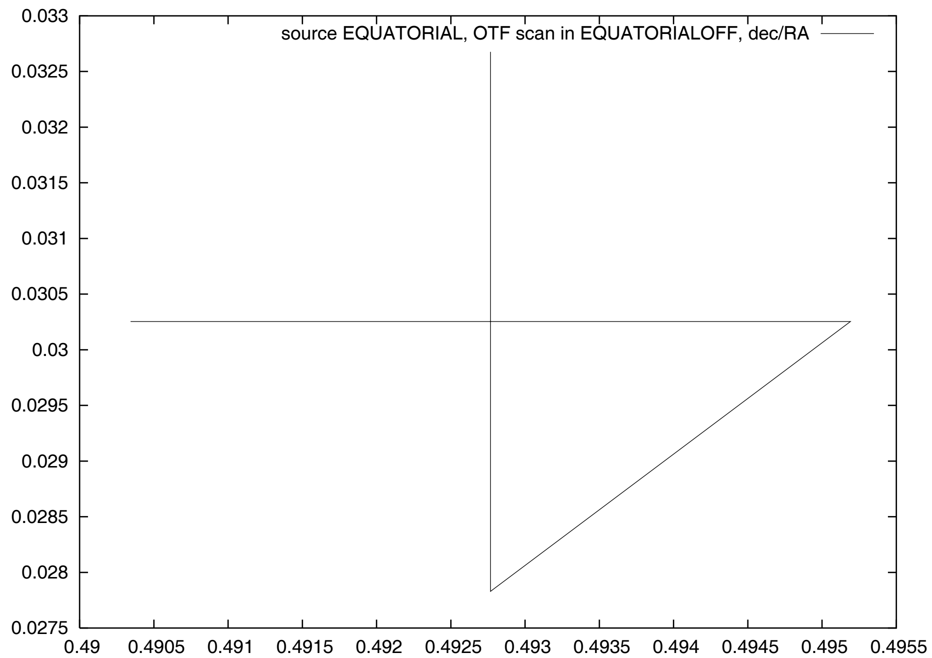


4.2.3 Source EQUATORIAL, OTF scan in EQUATORIAL

```
bin/source 0736+017 1 0 2000 0.49276698 0.03025334  0 0 0 0  0 0 0 0
bin/setNextSubscanOtf 2 2004-04-29.1111.1
bin/setNextSegmentLinear -0.002424 0 0.002424 0 0.00004848 \
 0.00004848 1 2004-04-29.1111.1.1
bin/setNextSubscanOtf 2 2004-04-29.1111.2
bin/setNextSegmentLinear 0 -0.002424 0 0.002424 0.00004848 \
 0.00004848 1 2004-04-29.1111.2.1
```

```
   MJD 12541.420972
   LST 0.230295
```


source EQUATORIAL, w OTF scan in EQUATORIALOFF, el/zz
0.916
0.915
0.914
0.913
0.912
0.911
0.91
0.909
0.908
2.706
2.708
2.71
2.712
2.714
2.716
2.718
2.72
2.722
2.724


source EQUATORIAL, OTF scan in EQUATORIALOFF, dec/RA
0.033
0.0325
0.032
0.0315
0.031
0.0305
0.03
0.0295
0.029
0.0285
0.028
0.0275
0.49
0.4905
0.491
0.4915
0.492
0.4925
0.493
0.4935
0.494
0.4945
0.495
0.4955

4.2.4 itFast timing