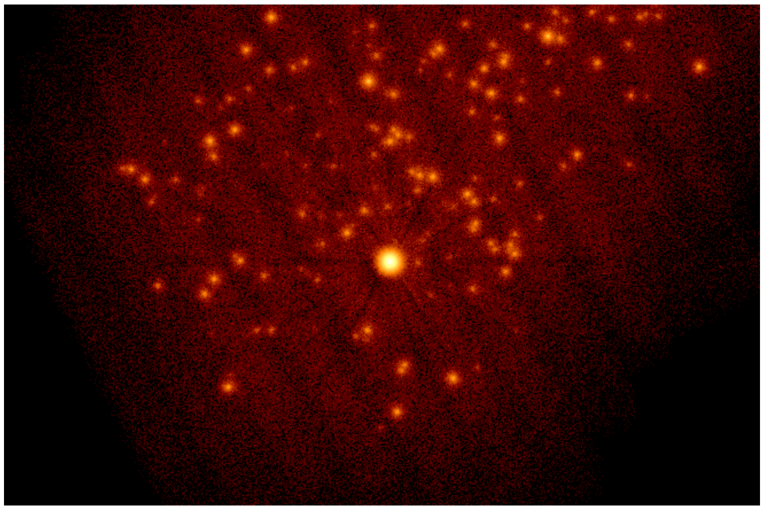
NuSim Manual



Authors
Kristin Kruse Madsen and Andreas Zoglauer
California Institute of Technology
2010

Abstract

Here go NuSim abstract.

Preface

Here be a preface.

Acknowledgement

Here go acknowledgements.

Contents

Chapter 1

Install

1.1 NuSIM quick installation instructions

1.1.1 Installation ROOT

The first prerequisite is ROOT. Here are some tips for the ROOT installation

- For the current version make sure you have installed at least version 5.22, higher is probably better, but do not install a version with an uneven minor version number, e.g. 5.25, 5.27 as those are development versions

- Make sure you don't mix 32 bit and 64 bit (i.e. if you compile NuSIM as 64 bit, you also need a 64 bit ROOT).

- Do not forget to set all ROOT variables, i.e. ROOTSYS (pointing to the directory where ROOT is installed), and add ROOT to the PATH and LD_LIBRARY_PATH (or DYLD_LIBRARY_PATH an Mac) variables if you install ROOT in a place not yet included in those paths.

1.1.2 HEAsoft

In order to read and write fits files, HEAsoft is required. A standard installation should do, but take into account the following things:

- NuSIMs configuration tool verifies that HEAsoft is installed only by checking of the HEADAS environment variable is set.

- It seems that on the latest HEADAS installations the required libcfitsio.[so/dylib] (so: Linux, dylib: Mac) does not exist but only a version one, e.g. libcfitsio_34.so. Since NuSIM should be able to run with any version of libcfitsio you might have to make a link, e.g. do: ln -s libcfitsio_3.24.[so/dylib] libcfitsio.[so/dylib] in the heasoft/<system version>/lib directory.

1.1.3 NuSIM

Next, retrieve NuSIM from its repository by using subversion:

```
svn co https://www.srl.caltech.edu/svn/nusim/trunk nusim
```

Old versions before 0.9.0 can be found in:

```
svn co https://www.srl.caltech.edu/svn/nustar/trunk/nusim nusim
```

Please ask Andrew Davis (ad@srl.caltech.edu) for a svn login and password.

The above command should have generated the directory nusim with all the source code.

Then set all your paths correctly (the paths are set in a similar way to your ROOT paths - this example is using bash on Linux):

```
export NUSIM=${some directory}/nusim
export PATH=${NUSIM}/bin:$PATH
```

For Linux (attention: in some system this variable might not yet have been defined!)

```
export LD_LIBRARY_PATH=${NUSIM}/lib:${LD_LIBRARY_PATH}
```

For Mac OS X (attention: in some system this variable might not yet have been defined!):

```
export DYLD_LIBRARY_PATH=${NUSIM}/lib:${DYLD_LIBRARY_PATH}
```

To configure NuSIM
cd $NUSIM
For Linux:
sh configure -linux -debug
For Mac OS X:
sh configure -macosx -debug
This is followed by a simple:
make
(No make install for the time being!)

If you launch NuSIM the first time, or if you expect that the configuration file format has been changed, launch NuSIM with the default configuration:

```
cd \$NUSIM
nusim -c resources/configurations/Ideal.cfg
```

Several more NuSIM configuration files can be found in the resources/configurations directory. Now you should be able to play around with NuSIM.

Some more tips:

- Calling **make man** creates doxygen documentation. Make sure you have doxygen & graphviz installed.
- If you update NuSIM make sure to completely recompile NuSIM: **make clean** followed by **make**. Otherwise you might experience unexpected crashes or weird behaviour.

1.1.4 Frequently asked questions

Example configuration

For Linux with bash the configuration files should look similar to this. Do NOT simply copy and paste this to your .bashrc-file! You have to adapt it to your own system! This is just an example, to check if you forgot something! Attention: The sequence does matter, since e.g. HEAsoft and ROOT have libraries which are named the same way. In addition, HEAsoft provides its own version of libreadline.so, which might interfere with system maintenance as super-user root.

```
PRG=/prg

# HEASOFT
if [ "$USER" != "root" ]; then
  export HEADAS=${PRG}/headas/i686-pc-linux-gnu-libc2.5
  alias heainit=". $HEADAS/headas-init.sh"
  source $HEADAS/headas-init.sh
fi

# ROOT
export ROOTSYS=${PRG}/root
export PATH=$PATH:$ROOTSYS/bin
export LD_LIBRARY_PATH=$ROOTSYS/lib:$LD_LIBRARY_PATH

# NUSIM
export NUSIM=${SOFTWARE}/Nusim
export PATH=${NUSIM}/bin:$PATH
export LD_LIBRARY_PATH=${NUSIM}/lib:${LD_LIBRARY_PATH}
```

ROOT compilation problems

Error message similar to:

```
Compiling XrdNetDNS.cc
g++ -c -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64
-D_REENTRANT -D_GNU_SOURCE -Wall -D__macos__
-Wno-deprecated -undefined dynamic_lookup -multiply_defined suppress  -O2 -DXrdDEBUG=0
     -I. -I.. XrdNetDNS.cc -o ../../obj/XrdNetDNS.o
XrdNetDNS.cc: In static member function 'static int XrdNetDNS::getHostAddr(const char*, sockaddr*, int, char**)':
XrdNetDNS.cc:73: error: 'gethostbyname_r' was not declared in this scope
XrdNetDNS.cc:82: error: 'gethostbyaddr_r' was not declared in this scope
XrdNetDNS.cc: In static member function 'static int XrdNetDNS::getPort(const char*, const char*, char**)':
XrdNetDNS.cc:393: error: 'getservbyname_r' was not declared in this scope
make[5]: *** [../../obj/XrdNetDNS.o] Error 1
make[4]: *** [Darwinall] Error 2
make[3]: *** [all] Error 2
make[2]: *** [XrdNet] Error 2
make[1]: *** [all] Error 2
make: *** [net/xrootd/src/xrootd/lib/libXrdSec.so] Error 2
```

The Xrd component of ROOT has sophisticated dependencies, but it is not required for NuSIM. Thus simply disable it during ROOT configuration with:

```
./configure -disable-xrootd
```

NuSIM configuration/compilation problems

Error message similar to:

```
(2) ROOT

Found ROOT: /home/andreas/prg/root/bin/root
Found ROOT version: 5.26/00 (minimum: 5.22, maximum: 5.28)
[: 108: ==: unexpected operator
```

You didn't use bash to run configure or you didn't start configure with: ./configure

Error message:

```
Generating dictionary... This may take a while...
rootcint: error while loading shared libraries: libCint.so: cannot open
shared object file: No such file or directory
```

Something is wrong with your ROOT installation:

- Did you install ROOT correctly?
- Does your LD_LIBRARY path contain the correct settings for ROOT?

Error message:

```
Library not found for -lcfitsio
```

Something is wrong with your HEAsoft installation:

- Your system has only a versioned version of libcfitsio, e.g. libcfitsio_34.so but no libcfitsio.so. Since the version ID changes from HEAsoft version to HEAsoft version, HEAsoft should contain a link from the versioned to the unversioned library. but it doesn't. Therefore you have to make it yourself:
  ln -s libcfitsio_3.24.[so/dylib] libcfitsio.[so/dylib] in the heasoft/<system version>/lib directory.
- If the above is not the problem, most likely HEAsoft is not or not correctly installed.

NuSIM execution problems

The NuSIM GUI program crashes with an error message like (or you see no GUI at all or have funny fonts):

```
Attaching to program: /proc/29042/exe, process 29042 done.
[Thread debugging using libthread\_db enabled]
[New Thread 0x7f9e513606f0 (LWP 29042)]
0x00007f9e49a1ffd5 in waitpid () from /lib/libc.so.6 error detected on stdin
The program is running.  Quit anyway (and detach it)? (y or n) [answered Y; input not from terminal]
Detaching from program: /proc/29042/exe, process 29042
```

Some Xft implementations (Xft is used for font smoothing) seem to have problems how ROOT is using them. You have to reconfigure ROOT with the option –disable-xft, recompile ROOT and recompile MEGAlib. As an alternative you can also comment out the line: gEnv->SetValue(”X11.UseXft”, ”true”); in the file $NUSIM/src/main/src/NGlobal.cxx

Error message similar to:

```
Error in <TUnixSystem::DynamicPathName>: MathMore[.so | .sl | .dl | .a | .dll]
does not exist in <long list of paths>
Error in <ROOT::Math::IntegratorOneDim::CreateIntegrator>:
Error loading one dimensional GSL integrator
```

Something is wrong with your ROOT installation. Either you did not compile ROOTs MathMore library (did you say disable-mathmore during configuring ROOT?), or the MathMore library couldnt be compiled because GSL (GNU scientific library) isnt installed on your system. Either way make sure GSL is installed on your system and you have configured ROOT to compile MathMore. Fixing this requires a recompilation of ROOT.

Chapter 2

The Modules

2.1 Satellite super module

The satellite super-module (internally represented by the class NSatellite) provides an interface to all other satellite modules, which are required by various other simulation and analysis modules.

2.1.1 Orbit engine

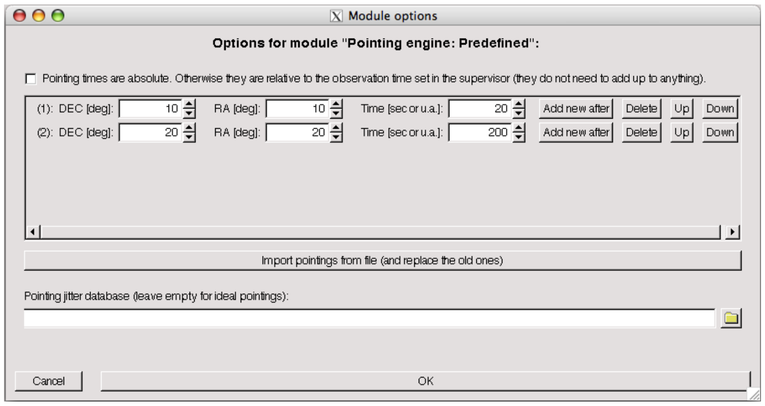
The orbit engine provides information about the current orbit position of the satellite, e.g. altitude, inclination, position in TBD coordinates.

2.1.2 Pointing engine

The pointing engine provides the pointing of the focal plane module in declination and right ascension. Figure 2.1 shows the options interface GUI for the pointing module. As illustrated the pointing module allows the user to define multiple pointings. Each pointing requires a coordinate set and an exposure time. Checking the box in the upper left turns the exposure times into absolute seconds, where as leaving it unchecked, the times become relative to the integration time given by the supervisor.

2.1.3 Orientation and alignment engine

The database and alignments module controls input databases and Figure 2.2 shows the interface options GUI. Each database entry comes in two forms: the ideal alignment of the system as defined ”pre-flight”, and the ”in-flight” alignments which will be subject to thermal perturbations. The databases are divided into 3 groups: the optics, the metrology and star tracker, and finally the rest of the spacecraft alignments. The optics, metrology and star tracker are kept separate so that they can be changed frequently without having to redo the other databases.


Module options

Options for module "Pointing engine: Predefined":

☐ Pointing times are absolute. Otherwise they are relative to the observation time set in the supervisor (they do not need to add up to anything).

(1): DEC [deg]: 10	RA [deg]: 10	Time [sec or u.a.]: 20	Add new after	Delete	Up	Down
(2): DEC [deg]: 20	RA [deg]: 20	Time [sec or u.a.]: 200	Add new after	Delete	Up	Down

Import pointings from file (and replace the old ones)

Pointing jitter database (leave empty for ideal pointings):

Cancel OK

Figure 2.1: Pointing module options GUI.

2.1.4 Time engine

At the moment the time engine provides the absolute time of the satellite. Modules such as the detector module, the metrology and star tracker module derive their own time from this absolute satellite time and transfer it to the event, star tracker and metrology data sets.

2.1.5 Geometry and detector properties

2.2 Event pipeline

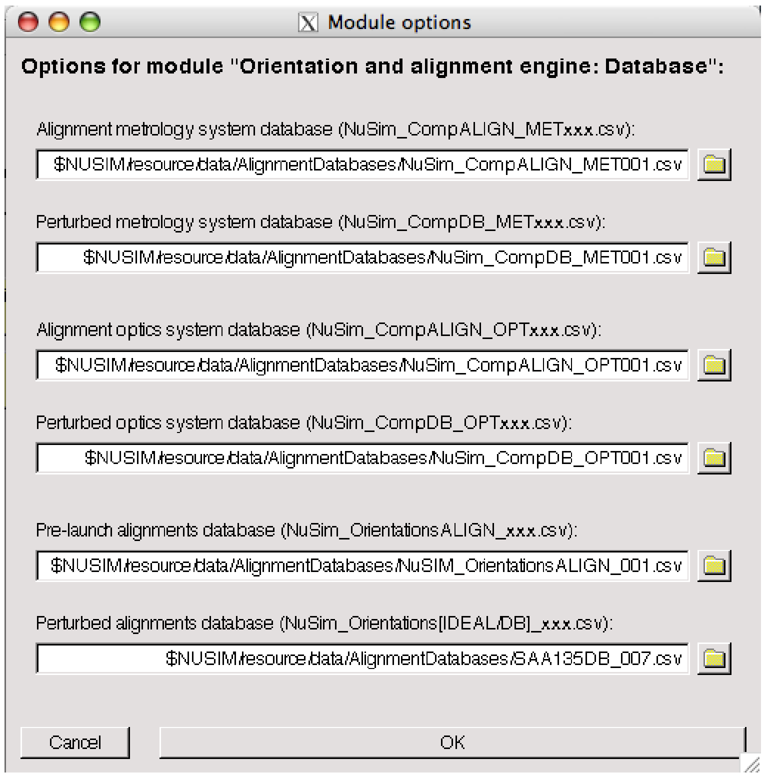
2.2.1 Source engine

The source module generates the initial spatial and spectral photon distribution for the simulator. Figure 2.3 shows the options interface GUI for the source module. Multiple sources can be generated and each source needs to have specified a beam type, spectral type and a flux. It is important to note the the flux here is in (ph/mm^2/s) and NOT (ph/cm^2/s) as is common for most of the HEASOFT software package. In addition, the flux is always the integrated flux within the given energy bounds set with the spectral options.

You need to set four parameters for a source: its name, its beam type, its spectral type, and its flux.

The beam type defines the geometry and the position of the source. The position is absolute and in degrees. Make sure that the source and telescope pointing matches. The following beam types are available:

- **Point source (far field)**: This is a point source at infinity. It requires RA and DEC coordinates in degree.
- **Disk source (far field)**: This creates a disc at infinity defined by its radius. It requires RA and DEC coordinates in degree as well as the radius of the disk in


Module options

Options for module "Orientation and alignment engine: Database":

Alignment metrology system database (NuSim_CompALIGN_METxxx.csv):
$NUSIM/resource/data/AlignmentDatabases/NuSim_CompALIGN_MET001.csv

Perturbed metrology system database (NuSim_CompDB_METxxx.csv):
$NUSIM/resource/data/AlignmentDatabases/NuSim_CompDB_MET001.csv

Alignment optics system database (NuSim_CompALIGN_OPTxxx.csv):
$NUSIM/resource/data/AlignmentDatabases/NuSim_CompALIGN_OPT001.csv

Perturbed optics system database (NuSim_CompDB_OPTxxx.csv):
$NUSIM/resource/data/AlignmentDatabases/NuSim_CompDB_OPT001.csv

Pre-launch alignments database (NuSim_OrientationsALIGN_xxx.csv):
$NUSIM/resource/data/AlignmentDatabases/NuSIM_OrientationsALIGN_001.csv

Perturbed alignments database (NuSim_Orientations[IDEAL/DB]_xxx.csv):
$NUSIM/resource/data/AlignmentDatabases/SAA135DB_007.csv

Figure 2.2: Database and alignments options GUI.

degree.

- **Point source (near field)**: This is used for mimicking a calibration source at a finite distance above the detector. It requires the position of the source (x, y, z in mm).

- **Pencil beam (near field)**: This beam is used to verify CZT detector calibrations. The pencil beam requires a start position as well as a direction (x, y, z in mm), as well as the radius of the beam.

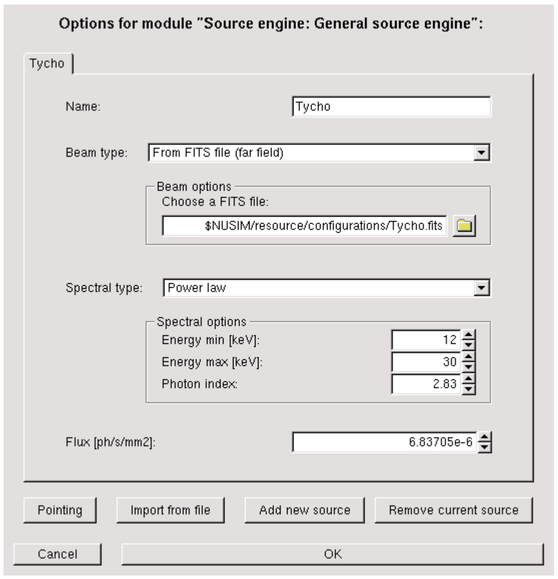
- **Read from fits file**: This creates a photon field based on a fits intensity image.

The spectral options are:

- **Mono-energetic**: Requires only the line energy in keV.

- **Linear**: Requires the upper and lower boarder of the energy range in keV.

- **Power-law**: Requires the upper and lower boarder of the energy range in keV as well as the photon index.



Options for module "Source engine: General source engine":

Tycho

Name: Tycho

Beam type: From FITS file (far field)

Beam options
Choose a FITS file:
$NUSIM/resource/configurations/Tycho.fits

Spectral type: Power law

Spectral options
Energy min [keV]: 12
Energy max [keV]: 30
Photon index: 2.83

Flux [ph/s/mm2]: 6.83705e-6

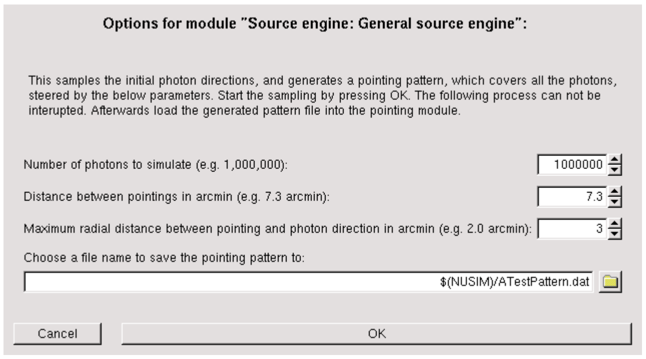
Figure 2.3: Source module options GUI.

- **Broken power-law**: Requires the upper and lower boarder of the energy range in keV, the break energy in keV, as well as the lower and upper photon index.
- **Black body**: Requires the upper and lower boarder of the energy range in keV as well as the temperature in keV.
- **File with differential flux**: Read differential spectrum from an ASCII file. The spectrum does not require any special normalization besides being 1/keV, since the total flux is as always given by the flux keyword. An example file of the format can be found at:
  $NUSIM/resource/data/SourceGenerator.examplespectrum.dat

The final required parameter is the flux in $ph/mm^2/s$. This format instead of e.g. $ph/cm^2/s/keV/sr$ had to be chosen to allow to combine each spectrum with each beam. Therefore don't forget to integrate over keV, if you use non-mono-energetic spectra which are given in keV^{-1}.

For the sources at infinite distance the photons are started randomly from a disk on a sphere surrounding the opening of the optics modules (the module is chosen randomly) to enable the correct simulation of the effective area as a function of incidence angle. The start time is randomly determined (Poisson distribution) according to the source flux. When the start position, start direction and photon energy are determined, the photon is handed over to the optics module for further simulation.

Known issuses If you have too many sources then no tab is displayed in the GUI. This is a ROOT issue within the TGTab class. If you have too many sources it can on


Options for module "Source engine: General source engine":

This samples the initial photon directions, and generates a pointing pattern, which covers all the photons, steered by the below parameters. Start the sampling by pressing OK. The following process can not be interupted. Afterwards load the generated pattern file into the pointing module.

Number of photons to simulate (e.g. 1,000,000):	1000000
Distance between pointings in arcmin (e.g. 7.3 arcmin):	7.3
Maximum radial distance between pointing and photon direction in arcmin (e.g. 2.0 arcmin):	3

Choose a file name to save the pointing pattern to:

$(NUSIM)/ATestPattern.dat

Figure 2.4: GUI for generating pointing patterns.

some systems take a long time to dispaly the GUI. This is another ROOT issue within the TGTab class.

A special feature of the source engine is that it can also generate a simple pointing pattern. To do so press the button ”Pointing” in the source engine GUI and the GUI see in Fig 2.4 should appear.

The pointing pattern is generated by first simulating a set of photons. The number of photons is given in the first entry box. It should be high enough to sample all sources, but not too high because then the simulation takes too long. 1,000,000 photons seems a reasonable compromise for most situations. For each photon the start direction in RA and DEC is stored. After the simulation, the source region is covered with a rectangular pattern of pointings. The distance between the pointings is given in the second entry field. Since the source field is not necessary rectangular many individual pointings may be not necessary. The last entry in GUI determines how close the pointing direction (i.e. the optical axis of the instrument) must be to a simulated photon in order to accept the pointing. A value of, e.g., 3 arcmin means that one photon must have been simulated within a radius of 3 arcmin of the pointing direction of the instrument. Don’t make the value too small, otherwise it’s unlikely you have simulated a photon within the disk, and do not make it larger than the half the field-of-view of the instrument. In the bottom entry box you give the name of the file to which the pointings are stored. You have to read this file in the pointing module in order to use it.

In order to test and verify the coverage of your generated pointing pattern, as well as the evenness of your image, you can make a full simulation with a flat input distribution, e.g. a disk source, which is large enough to cover all pointings.

Known issuses Currently the roll of the pointing is not aligned.

2.2.2 Optics engine

The optics engine is a full-fledged ray-trace to simulate the actual optics. It incorporates the exact geometry of the optics, and uses externally generated reflectivity files to calculate the reflection of the photons off the mirrors. Additionally it has a module to simulate deflection of the X-ray photons due to mirror imperfections, which will generate the mirror point spread function. The ray-trace also keeps track of single reflections, called "ghost rays", where the photon only reflects off one mirror while passing through.

The module has three options:

- **Scattering**: This options turns on the scattering of the photons due to figure errors in the mirrors. Scattering must be on for a science simulation. It is important to note that this is scattering in the reflection sense only, and not of any particle effects.

- **Perfect Optics**: This option focuses all rays to one spot. This option should not be used with science simulations and is primarily for debugging.

- **Ghost rays**: This option enables the ghost rays. As a default it should be on when running a science simulation.

The input of this module is a photon location, direction and energy handed to it by the source module, and the output is location, direction and energy after the photon has exited the optics.

2.2.3 Aperture engine

The aperture engine places an aperture in the photon path and rejects any photon hitting the aperture. The aperture stop is located 833.187 mm above the detector, and has an opening diameter of 58 mm.

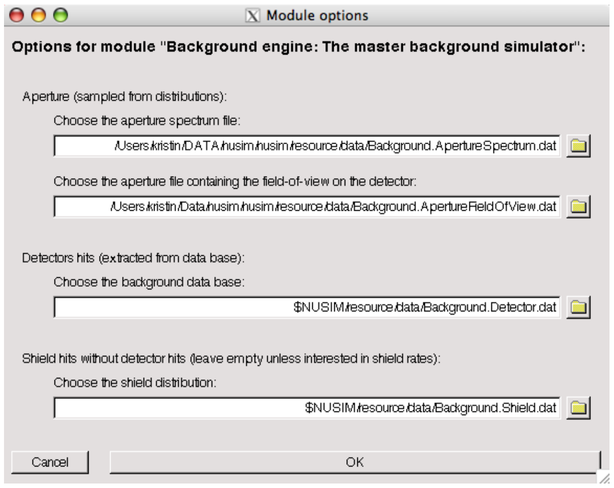
2.2.4 Background engine

The background engine draws the background from an external GEANT simulation, combining it with the expected background flux from the sky. This module has four options:

- **No Module**. It is possible to run NuSIM without having a background engine.

- **Master Background Simulator**. As the name implies this is the actual background option to use when running science simulations. Figure 2.5 shows the interface options GUI and the files can be found in **$NUSIM/resource/data/**.

- **Event Loader: Universal loader**. Allows the user to load a previously saved event list.



Options for module "Background engine: The master background simulator":

Aperture (sampled from distributions):

Choose the aperture spectrum file:

/Users/kristin/DATA/nusim/nusim/resource/data/Background.ApertureSpectrum.dat

Choose the aperture file containing the field-of-view on the detector:

/Users/kristin/Data/nusim/nusim/resource/data/Background.ApertureFieldOfView.dat

Detectors hits (extracted from data base):

Choose the background data base:

$NUSIM/resource/data/Background.Detector.dat

Shield hits without detector hits (leave empty unless interested in shield rates):

Choose the shield distribution:

$NUSIM/resource/data/Background.Shield.dat

Figure 2.5: Background interface options GUI for the "master background simulator".

2.2.5 Detector interactions engine

2.2.6 Detector effects engine

2.2.7 Trigger engine

The trigger module represents the trigger and downlink decision hardware aboard the NuSTAR satellite. It shall decide if an energy deposit in the shield represents a veto or the pattern on the detector is valid for a downlink.

2.2.8 Detector data calibrator

2.2.9 Event selector

2.2.10 Science analyzer

2.3 Metrology and Star tracker pipeline

2.3.1 Metrology engine and calibrator

The purpose of the metrology engine is to generate data at a rate equivalent to the onboard metrology system. The engine takes the known perturbed aspect of the instrument system and finds the intersection of the metrology laser with the metrology detector. It will then apply noise to the data set simulating the centroiding error of the metrology detector. This is done by applying a Gaussian error to each of the measurement axis. The 1-sigma error is reported in the database file:
resource/AlignmentDatabases/NuSim_CompDB_MET001.csv .

The engine passes the coordinates on to the Observatory Reconstructor for use in deriving the aspect reconstruction.

2.3.2 Star tracker engine and calibrator

Based on the input pointing defined by the pointing engine, the task of the Star Tracker Engine is to generate a quaternion, which defines the rotation between the Camera Head Unit, CHU, to the J2000.0 heliocentric inertial equatorial reference frame, also called VSN (Vernal, Summer, North). The origin is the intersection of the CCD plane with the optical axis of the camera. The CHU z-axis points along the boresight, and x/y axis span the CCD plane. The output of the Star Tracker is a Quaternion that defines the attitude of the CHU w.r.t. the VSN, such that

$$x_{CHU} = (Q_{CHU->VSN}) * x_{VSN}. \tag{2.1}$$

Thus if the a unit vector in the star tracker frame is transformed into the VSN frame by

$$x_{VSN} = (Q_{CHU->VSN})^T * x_{CHU}, \tag{2.2}$$

then RA=atan(y_{VSN}/x_{VSN}), DEC=acos(z_{VSN}). In the module $(Q_{CHU->VSN})^T$ is produced at a rate equivalent to the onboard Star Tracker. The module will add a Gaussian noise to the transformation to mimic solution error of the Star Camera. The 1-sigma error is reported in the database file:
resource/AlignmentDatabases/NuSim_CompDB_MET001.csv .

The transformation is passed on to the Observatory Reconstructor, which interpolates the transformation for a specific time and derives the aspect reconstruction.

2.3.3 Observatory reconstructor

The observatory reconstructor is the set of algorithms that solves the attitude and aspect problems of the NuSTAR observatory, given the metrology and star tracker data. The observatory reconstructor does not have access to the satellite module, which would contain the actual attitude and aspect of the observatory. It is responsible for recalculating that attitude and aspect, given only a certain number of inputs, which are simulated satellite data. The inputs to the observatory reconstructor are:

- The calibrated positions, in local coordinates, at which the metrology lasers impinge on their detectors. These are interpolated in time.
- The calibrated star tracker data, which is a transformation (quaternion) from local star tracker coordinates to inertial (J2000.0) coordinates. These are interpolated in time.
- The on-ground alignment data defining the ideal locations of all the instrument components. Specifically, the pointing of the optical axis in optics bench coordinates, the location and pointing direction of the metrology laser in optics bench coordinates, the location and rotation to the metrology detector from focal plane bench coordinates and to the star tracker from optics bench coordinates.

The output from the observatory reconstructor is a transformation from focal plane module coordinates to celestial coordinate R_{fbin}. For extensive details on this reconstruction, and the algorithm solution, see the memo NuSTAR Pointing Reconstruction.

2.3.4 Observatory merger

Chapter 3

NuSIM Coordinate Systems Database

3.1 Purpose

The various components of NuSIM each operate in their own coordinate systems, and so in order to simulate the entirety of the spacecraft, it is necessary to keep track of the transformations between these systems. Furthermore, for many tests it is necessary to simulate NuSIM under changing conditions, and so for each timestep a set of coordinate system transformations must be provided.

3.1.1 List of Coordinate System Transformations

Name	Transformation
Inertial-SC	Inertial Frame to Spacecraft
SC-FB	Spacecraft to Focal Bench
FB-FP0	Focal Bench to First Focal Plane Module
FB-FP1	Focal Bench to Second Focal Plane Module
FB-MD0	Focal Bench to First Metrology Detector
FB-MD1	Focal Bench to Second Metrology Detector
FB-AS0	Focal Bench to First Aperture Stop
FB-AS1	Focal Bench to Second Aperture Stop
FB-OB	Focal Bench to Optics Bench (the mast)
OB-OM0	Optics Bench to First Optics Module
OB-OM1	Optics Bench to Second Optics Module
OB-ML0	Optics Bench to First Metrology Laser
OB-ML1	Optics Bench to Second Metrology Laser
OB-ST	Optics Bench to Star Tracker

3.2 Format

In order to provide NuSIM with a set of coordinate system transformations for each time step, a comma separated values (.csv) spreadsheet is used.

3.2.1 Layout

Each time-step is a pair of sequential rows: one for the translation between the coordinate systems, presented in (x, y, z), and the other for the rotation, presented as a quaternion, with the real component last: (q1, q2, q3, q4). Thus each coordinate system transformation is a set of four adjacent columns, and on the first row of each time-step the fourth column of each coordinate system transformation is empty.

There are several rows before the time-steps begin, representing header data at the top of the database file. These will include a row in which the name of each coordinate system transformation is in the first of the four columns its values occupy.

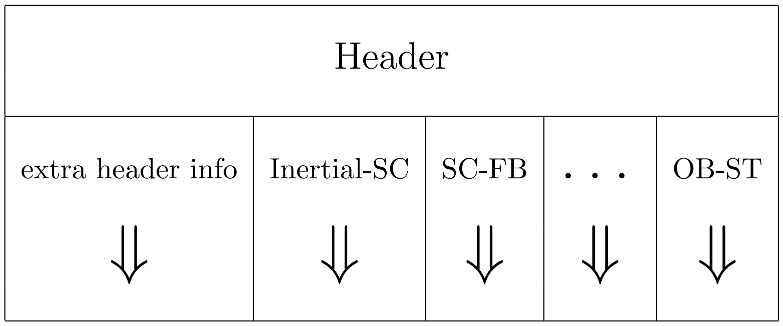
There are no rows between the time-steps, and no rows of interest after the time-steps.

3.2.2 Template

With each database format revision, a template file is produced, often in microsoft excel format. This file is simply a database with a single time-step: all the components in their ideal configuration. Historically, these file have been named along the lines of "NuSIM_OrientationsIDEAL_005.xls." Note that while Microsoft Excel's exported CSV files use carriage return ("\r") line-breaks, NuSIM requires its input databases to use newline ("\n") line-breaks.

3.2.3 Diagrams

File Layout



column layout

Section	Content			
Header	header info			
	Name			
	more header info			
Time-Step 0	x translation	y translation	z translation	
	q_1 (i coefficient)	q_2 (j coefficient)	q_3 (k coefficient)	q_4 (real)
Time-Step 1	x translation	y translation	z translation	
	q_1 (i coefficient)	q_2 (j coefficient)	q_3 (k coefficient)	q_4 (real)
Time-Step 2	x translation	y translation	z translation	
	q_1 (i coefficient)	q_2 (j coefficient)	q_3 (k coefficient)	q_4 (real)
$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$

3.3 NuSIM Database Transformer

NuSIM Database Transformer is a python library meant to simplify the process of generating multiple time-step databases with algorithmically generated coordinate system transformations. It takes as input the ideal template database (in csv format), and outputs sequential steps in accordance with an input function.

3.3.1 Requirements

NuSIM Database transformer is a Python script. That means it requires an installation of Python to run. Python comes standard on most modern Linux installations as well as OSX (although it may be in the developer tools, I don't recall), and is available free for most any operating system (even Windows) from python.org. To check to see if you have python, enter into a terminal:

```
python
```

If you find yourself in a python shell, you've got python. Exit with:

```
exit()
```

You do need to be able to program in Python to use NuSIM Database Transformer. Python is an interpreted language conforming to a number of common standards, which allows for mostly iterative programming with the ability to create functional (using functions themselves as variables) and object oriented programming. This readme talks about python objects, functions, lambdas, tuples, lists, and dictionaries. Tutorials and explanations are available at python.org

NuSIM Database transformer requires an ideal database in CSV format. It assumes that this database contains the entries:

```
'Inertial-SC','SC-FB','FB-FPM0','FB-FPM1','FB-MD0','FB-MD1',
'FB-AS0','FB-AS1','FB-OB','OB-OM0','OB-OM1','OB-ML0','OB-ML1',
'OB-ST'
```

All on the same row, and that the last two rows with numbers in them are the ideal translational coordinates and the ideal rotational quaternians, respectively. Furthermore, it expects each transformation name listed above to share a column with the first of the three adjacent translational coordinates and the first of the four adjacent quaternion components (which are stored constant last, by the way). This is the standard format as found in ideal databases 005 and 006, and the database can handle any other extra columns, rows, or comments inserted into the ideal database as long as these rules are adhered to.

3.3.2 Description

NuSIM Databse Transformer is simply a python library containing useful tools for the generation of parametrically-changing databases for NuSIM. These are:

arcsecondsToRadians(asec)

returns asec, but in radians.

eulerAnglesToQuaternion(xrot, yrot, zrot)

Converts euler angles to quaternions via the formula from http://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles according to the x-y-z convention, which is the same as the formula from "Quaternions and Rotation Sequences" by Jack B. Kuipers, page 207, for converting Aerospace angle sequences to Quaternions. This means that what we are doing here is technically rotating about Z, then Y, then X. There is a small change in convention here; because we are doing a frame rotation, and not a point rotation, we are reversing the coefficient terms (hence the (-1.0)*), and we are putting the constant term last instead of first. It returns these as a tuple with four entries.

Class Output_Writer

A simple class which will either print or write to a file depending on what you want to do. If no filename, or a filename of None is input, .write() prints out (sans newlines or space each time), but if a filename is input, then .write() writes to that filename. close does nothing if you are printing out, but it closes the file if you're writing a file.

Database (described below) writes with one of these so it is possible to run a script that writes a database and pipe it somewhere instead of writing immediately to a file.

Class Entry

Each entry in the database needs a quaternion (.quater) with 4 entries and a set of coordinates (.coords) with three entries. These can be set directly be inputting a tuple into setCoords or setQuatr, or by inputing 3 numbers into setCoords and 4 into setQuatr. The inputs to the Constructor are the same as the inputs to setCoordsAndQuatr, which is to say either all three coords followed by all four entries of the quatr, or two tupples in a row. getX, getY, getZ, and getQ1, getQ2, getQ3, getQ4 are self explanatory keep in mind that the last entry of our quaternions is the constant term, (getQ4)

Class Database

This is the REALLY USEFUL ONE. This is a class designed to facilitate the creation of database files as input for NuSIM. This class takes as constructor input a filename of an ideal file.

After construction, a Database object has the following fields:

- newline: the newline character (either \n or \r which this should use when writing a .csv
- desiredColumns: the list of all the column names to use in this database
- header: the text part of the database above the timesteps
- columns: the dictionary of all the column names as keys for their x coordinate in the .csv
- ideal: the ideal Entry objects read form the ideal database (with column names as keys)
- current: the current Entry objects read form the ideal database (with column names as keys)
- previous: the previous Entry objects read form the ideal database (with column names as keys)
- idealCoordsLine: the coordinates line (split by ”,”) which was read from the ideal database
- idealQuaterLine: the quaternions line (split by ”,”) which was read from the ideal database
- step: the current timestep (starts at 0)
- out: the Output_Writer object with which to write out the .csv

This means that you can get, for example, the current X coord of FB-OB with:

```
database.current['FB-OB'].getX()
```

Writing functions:

- open(filename): this function sets this object to write at the given filename. It does this by setting self.out to be an Output_Writer(filename), so you can fail to put in a filename and write to standard out.
- close(): call this when this Database is done writing.
- writeHeader(): writes the self.header to out. Do this first when writing a database,
- writeStep() writes the current Step, exactly as formatted in idealCoordsLine and idealQuaterLine, but with the information from current, to out.
- newStep(stepFunc): This is the really important one. This function creates a new step, sets previous step to current step, and current to new step. There are two things you can enter into newStep:
  - any function with input (this database object) that outputs the next step If you input to newStep any function that returns a dictionary with valid Entry objects keyed to every entry of desiredColumns, when given this database object as input (as in, newStep calls stepFunc(self)), newStep will happily make the new Step the one setpFunc output.
  - any dictionary of functions which output entries keyed to any subset of desiredColumns for example, if I only wanted to play with FB-OB and SC-FB, I'd:

    ```
    database.newStep({
    'FB-OB':function_that_given_input_database_outputs_entry_for_FB-OB,
    'SC-FB':function_that_given_input_database_outputs_entry_for_SC-FB
    })
    ```
- writeDatabase(steps, stepFunc, output_filename): simply starts a database at output_filename, writes the header there, and writes steps Steps to The database, iterating each time with stepFunc, and then closes the database.

3.3.3 Use

To use NuSIM Database Transformer, a python script needs to import it as a library. Then it can use the tools listed above, which should be helpful in generating varying parameter databases for NuSIM.

The most useful single method of the library is: Database.writeDatabase(steps, stepFunc, output_filename).

To use this, create a database object (remember that the Database constructor takes as input the name of an ideal database CSV file), and then call the writeDatabase method from that object. The inputs should be:

- steps: the number of steps this database should have

- stepFunc: This can be either a function, which given a database object input, returns a dictionary with Entry objects keyed to each of: 'Inertial-SC','SC- FB','FB-FPM0','FB-FPM1','FB-MD0','FB-MD1','FB- AS0','FB-AS1', 'FB-OB','OB-OM0','OB-OM1','OB- ML0','OB-ML1', and 'OB-ST', or a dictionary containing a subset of the above keys, each keying to a function that takes a database object as input and returns an Entry object. This is the function with which the next step of the database will be computed at each step.
- output_filename: the name of the file you want to write this database to. This will be a csv file in the same format as the ideal file the Database object was created with. This can be NULL if you want to just print it.

Recall that python functions can be declared anonymously (inline) in the following format:

```
lambda inputs: output
```

Using this, it is possible to write a multi-step database in as little as one line of Python (although for readability this is inadvisable.

Remember that once a Database object has written, it increments its current step and resets its current and previous step entries. Therefore, writing multiple times from one Database object produces a continuation of the database, possibly with varying stepFunc entries, or multiple files. If you want to write multiple different databases from one ideal database, it is easiest to use the copy library, and after creating a database object from an ideal database, copy.deepcopy() it each time you want a database object to write with.

To run a python script you've written using this or any other library, call from a terminal:

```
python script.py
```

3.3.4 Examples

Note that readme_nusim_database_transformer.py is a text-only copy of this section of this document written as executable python. Running it will run all these examples.

Importing

The first step to using the NuSIM Database Transformer library is to import it. Python allows for

```
import nusim_database_transformer
```

but this would mean we'd have to type

```
nusim_database_transformer.
```

before everything we used that was defined in the library. For example, we'd have:

```
db = nusim_database_transformer.Database("ideal_file.csv")
```

This is a pain. Therefore we will simply import everything in the library into the script:

```
from nusim_database_transformer import *
```

Create a Database Object

Next we have to make a database object. Here it is important to choose the input file which is of the correct format. The one we've been using recently as of the time of this writing is called "NuSIM_OrientationsIDEAL_005.csv" And so we create a database object from that ideal file:

```
db = Database("NuSIM_OrientationsIDEAL_005.csv")
```

If at some future date, we wanted database files computed from the ideal database "NuSIM_OrientationsIDEAL_006.csv", we would simply have written:

```
db = Database("NuSIM_OrientationsIDEAL_006.csv")
```

instead, and that would have been the only difference in the script.

Simple Database

Now it is time to think about what we'd like to write to our output databases.

Consider the simplest case: a database consisting of all ideal entries. This allows for a simple illustration of the writeDatabase method.

The most powerful input for newStep or writeDatabase is stepFunc: any function, which given the entire database object, can manipulate that object and return a dictionary representing the next step, computed in any possible fashion.

The simplest possible value for stepFunc here would be a function that given an input database returned exactly the ideal step every time:

```
def simpleStepFunc(database):
      return database.ideal
```

Now, remember that a single database object should not be used to write more than one database file, as it will always write a continuation of the steps it has hitherto been writing (the step number always increments), so it is convenient, rather than generating a new database object each time we want to write a database, to copy the one we have, and write from the copy.

```
import copy
db2 = copy.deepcopy(db)
```

To write a database of 100 steps to the file "100_ideal_steps_A_005.csv", we'd call:

```
db2.writeDatabase(100, simpleStepFunc, "100_ideal_steps_A_005.csv")
```

Furthermore, recall that using inline functions, this can be done in fewer lines:

```
db3 = copy.deepcopy(db)
db3.writeDatabase(
            100, lambda database: database.ideal,
            "100_ideal_steps_B_005.csv"
            )
```

In addition, remember that it is also possible to substitute for stepFunc a dictionary of Entry - returning functions for each entry you want to vary with each step. The current step is initialized to be the ideal step, and so we, in effect, simply do not wish to vary any entries. We can therefore input for stepFunc an empty dictionary:

```
db4 = copy.deepcopy(db)
db4.writeDatabase(100, {}, "100_ideal_steps_C_005.csv")
```

Constant Offset

Now consider a slightly more complicated case: We want a database of 100 entries which are all the same, but not ideal. For example, consider the case where FB-OB is shifted along x by 5 mm. Here we need simply change the current step, and then print out 100 steps which are all unchanged from the current step. The clearest (if not the absolute shortest, codewise) way to accomplish this is:

```
db5 = copy.deepcopy(db)
db5.current['FB-OB'] = Entry(
                            db5.current['FB-OB'].getX() + 5,
                            db5.current['FB-OB'].getY(),
                            db5.current['FB-OB'].getZ(),
                            db5.current['FB-OB'].getQ1(),
                            db5.current['FB-OB'].getQ2(),
                            db5.current['FB-OB'].getQ3(),
                            db5.current['FB-OB'].getQ4()
                  )
db5.writeDatabase(100, {}, "100_FB-OB_x+5_A_005.csv")
```

Translations

Next, let's move on to somewhat more practical applications.

For example, consider shifting the x translation of FB-OB by 1 mm each step. You could add 1 to the current step each time:

```
def add1mmToX(database):
    return Entry(
                database.current['FB-OB'].getX() + 1.0,
                database.current['FB-OB'].getY(),
                database.current['FB-OB'].getZ(),
                database.current['FB-OB'].getQ1(),
                database.current['FB-OB'].getQ2(),
                database.current['FB-OB'].getQ3(),
                database.current['FB-OB'].getQ4()
             )

db6 = copy.deepcopy(db)
db6.writeDatabase(
                    100,
                    {
                        'FB-OB':add1mmToX
                    },
                    "100_FB-OB_x+n_A_005.csv"
                 )
```

Rotations

Let's say you wanted to test how NuSIM reacts under various rotations about x, 0.1 arc seconds each step, between the optical bench and the focal bench, without any translation.

Recall than an Entry constructor can accept a pair of tuples, one for coordinates and one for quaternions.

```
def rotationsStepFunction(database):
    return Entry(
            database.current['FB-OB'].coords,
            eulerAnglesToQuaternion(
                                    arcsecondsToRadians(0.1*database.step),
                                    0.0,
                                    0.0
                                    )
             )

db7 = copy.deepcopy(db)
db7.writeDatabase(
                    100,
                    {
                        'FB-OB':rotationsStepFunction
                    },
                    "100_FB-OB_xrot+n_A_005.csv"
                 )
```

3.4 Thermal Mast Bending

The document "Thermal_Distortion_2_for_JPL.xlsx" details the effects on the mast under the thermal effects of sunlight. In particular, it provides x and y offsets of the focal plane intersections with the beams from the optics, as well as $rotZ$, the twisting of the mast itself. These effects are most pronounced for a 170 degree angle of the mast toward the sun. To more fully model the effects of thermal mast bending on NuSTAR, a coordinate system transformation database had to be constructed for the mast in the configurations predicted for a full orbit.

3.4.1 Modeling the Mast as an Arc

The data given for mast distortions due to heating is given as a set of data points for different positions relative to the sun, with each point composed of the displacement of the beams from the optics modules at the focal bench, as well as the rotation (twist) of the mast itself.

I operated under the following assumptions:

- The twist in the mast is small enough that its effect on the bend of the mast is not worth calculating, which is to say it can be interpreted as one of the benches at either end having been rotated about the mast.
- The mast bend in x and y dimensions can be interpreted as separate arcs, and the displacement from each is additive, as the mast is square and is likely to bend more or less independently in each direction along these small angles.
- The beams from the optics modules are meant to move exactly parallel to the mast, centered at the center of the OM coordinate system. (supported by the coordinate systems IDEAL reference documentation).
- Over these small angles, second order approximations of cosine are acceptable.

First of all, I had to translate the coordinates given for the distortion of x and y along the rotated (by ROTZ) focal plane into distortions of x and y in a plane not rotated along z with respect to the optics modules.

This is most easily done by converting to radial coordinates with the center being the mast, and subtracting $ROTZ$ from the angle.

$$r = \sqrt{x^2 + y^2}$$

$$\theta = \arctan\left(\frac{x}{y}\right)$$

$$correctedY = r \sin(\theta - ROTZ)$$

$$correctedX = r \cos(\theta - ROTZ)$$

Now that those are corrected, we need to calculate the arc of the mast.

Let m be the length of the mast itself, which is mostly invariant, so we shall consider this to be the arc length. The arc angle θ and the arc radius r can be determined as follows:

The mast is essentially an arc (of radius r) with the optics module and the exact point on the focal plane meant to receive it each on what amount to beams perpendicular to the arc extending out by a distance x. Since the distortions in x and y are positive in all cases, we can assume the center of the arc is on the opposite side of NuSTAR from the $+X$ detector. Therefore the line from the center of the arc to the optics module emitting the beam (length $r + x$) is a leg of a right triangle, with the other leg being the beam itself and the hypotenuse being the line from the center of the arc to the point where the beam hits the detector (length $r + x + \Delta x$).

Mast Bending Model Diagram

Thus:

$$\cos(\theta) = \frac{r + x}{r + x + \Delta x}$$

A general rule of arcs is:

$$m = \theta r$$

So:

$$\cos(\theta) = \frac{\frac{m}{\theta} + x}{\frac{m}{\theta} + x + \Delta x}$$

This is not solvable analytically, (or at least Mathematica couldn't), so here we shift to a second order approximation of cos, which should be accurate for these small angles:

$$1 - \frac{\theta^2}{2} \approx \frac{\frac{m}{\theta} + x}{\frac{m}{\theta} + x + \Delta x}$$

For which the solutions for θ are:

$$\frac{-m \pm \sqrt{m^2 + 8x\Delta x + 8\Delta x^2}}{2(x + \Delta x)}$$

Given that when $\Delta x = 0$, $\theta = 0$, the correct solution is:

$$\frac{-m + \sqrt{m^2 + 8x\Delta x + 8\Delta x^2}}{2(x + \Delta x)}$$

Which does indeed grow as expected as Δx grows, so it seems a reasonable solution.

Given θ, and $r = \frac{m}{\theta}$, It should be clear that the optics bench will move by $r(1 - \cos(\theta))$ along the x axis and $r\sin(\theta) - m$ along the z axis.

The optics bench itself, if we are looking at the arc with x translation, will rotate by θ about y.

The two dimension of bend (x and y) are essentially the same, and each is calculated as above, with their effects considered additive.

3.4.2 Implementation

A Coordinate System Transformation database was generated for each of the bent mast positions given by the original data using the NuSIM Database Transformer Python library. The function used to compute the transformation between the focal bench and the optical bench (the mast, "FB-OB") was as follows:

```
def compute_bent_mast_step(database):
    global thermals  # a list of (deltaX0, deltaY0, deltaX1, deltaY1,rotZ)
    x = database.ideal["OB-OM1"].getX()+database.ideal["FB-OB"].getX()
    y = database.ideal["OB-OM1"].getY()+database.ideal["FB-OB"].getY()
    rotZ = float(thermals[database.step][4])
    m=database.ideal["FB-OB"].getZ()+database.ideal["OB-OM1"].getZ()-
        database.ideal["FB-AS1"].getZ()
    deltaX = 25.4*thermals[database.step][0]
    deltaY = 25.4*thermals[database.step][1]
    rotR = math.sqrt((x**2) + (y**2))
    rotBase = math.atan(y/x)
    x = rotR*math.cos(rotBase - rotZ)
    y = rotR*math.sin(rotBase - rotZ)
    x_mast_arcangle = ((-1.0*m)+math.sqrt((m**2.0)+(8.0*x*deltaX)+
        (8.0*(deltaX**2.0))))/(2.0*(x+deltaX))
    x_mast_arcradius = m/x_mast_arcangle
    x_translation = x_mast_arcradius*(math.cos(x_mast_arcangle)-1.0)
    x_component_z_translation =
        (x_mast_arcradius*math.sin(x_mast_arcangle))-m
    y_mast_arcangle = ((-1.0*m)+math.sqrt((m**2.0)+(8.0*y*deltaY)+
        (8.0*(deltaY**2.0))))/(2.0*(y+deltaY))
    y_mast_arcradius = m/y_mast_arcangle
    y_translation = y_mast_arcradius*(math.cos(y_mast_arcangle)-1.0)
    y_component_z_translation =
        (y_mast_arcradius*math.sin(y_mast_arcangle))-m
    return Entry((
                database.ideal["FB-OB"].getX() + x_translation,
                database.ideal["FB-OB"].getY() + y_translation,
                database.ideal["FB-OB"].getZ() +
                    x_component_z_translation +
                    y_component_z_translation
               ),
                eulerAnglesToQuaternion(
                                       -y_mast_arcangle,
                                       x_mast_arcangle,
                                       rotZ
                                       )
              )
```

Chapter 4

The output file format keywords

Corresponds to NuSIM v0.9.0

Key:	SE
Parameters:	None
Description:	Indicates the start of a new event

Key:	TI
Parameters:	1: Time in seconds
Description:	The event time in seconds

Key:	ID
Parameters:	1: Number
Description:	A unique ID of the event

Key:	OG origin
Parameters:	1: Number (1: source, 2: background)
Description:	Tells if the origin of the photon is from a source or a background engine

Key:	TE
Parameters:	1: Number
Description:	The telescope ID. Either 1 or 2.

Key:	RD
Parameters:	1: Right ascension 2: Declination
Description:	The RA and DEC of the original photon empty of the photon was background.

Key:	OP original photon
Parameters:	1-3: Start position of the photon in the focal bench coordinate system in mm
Description:	4-6: Direction of the photon 7-9: Polarization of the photon 10: Energy of the photon in keV The initial parameters of the started photon (”original photon”)

Key:	IP initial photon relative to the optics module
Parameters:	see OP
Description:	Contains the parameters of the photon when it is INTIALLY rotated and translated ointo the optics module.

Key:	CP current photon
Parameters:	see OP
Description:	The last parameters of the photon (”Current Photon”): If the pipeline is saved before the detector (interactions) engine, then the current photon parameters, if the event is saved after the detector (interactions) engine then the last photon parameters.

Key:	IA interaction
Parameters:	1: d or s: detector or shield 2: Telescope ID 3: Detector ID 4-6: Ideal position within the CZT detector in the CZT detectors coordinate system mm 7: Ideal energy in keV
Description:	Interactions as determined by the detector (interactions) engine.

Key:	PH pixel hit
Parameters:	1: Telescope ID 2: Detector ID 3: x pixel hit 4: y pixel hit 5: pre-trigger sample sum (pulse height) 6: post-trigger sample sum (pulse height) 7: ideal average depth (mm) 8: noised average depth (mm) 9: ideal energy deposit (keV) 10: noised energy deposit (keV)
Description:	A pixel hit, i.e. the detector effects engine applied to the ideal interactions

Key:	SH shield hit
Parameters:	1: Telescope ID 2: Detector ID 3: ideal energy deposit (keV) 4: noised energy deposit (keV)
Description:	A shield hit, i.e. the detector effects engine applied to the ideal interactions in the shield

Key:	NH nine-pixel hit
Parameters:	1: Telescope ID 2: Detector ID 3: x central pixel 4: y central pixel hit 5: pixel 1: pre-trigger sample sum (pulse height) 6: pixel 1: post-trigger sample sum (pulse height) 7: pixel 1: trigger (bool) 29: pixel 9: pre-trigger sample sum (pulse height) 30: pixel 9: post-trigger sample sum (pulse height) 31: pixel 9: trigger (bool) 32: ideal average depth (mm) 33: noised average depth (mm) 34: ideal energy deposit (keV) 35: noised energy deposit (keV)
Description:	A pixel hit, i.e. the detector effects engine applied to the ideal interactions

Key:	PE PHE data
Parameters:	TBD.
Description:	Not yet used

Keyword:	HT
Parameters:	1: Telescope ID 2: Detector ID 3-5: Position in the focal plane module (detector) coordinate system in mm 6-8: Position resolution of the above position in mm 9: Energy in keV 10: Energy resolution in keV 11: Observatory data: time in sec 12-14: Observatory data: direction of optical axis in inertial system 15-17: Observatory data: direction of event in inertial system 18-20: Observatory data: orientation of focal plane in optic bench system: translation 21-24: Observatory data: orientation of focal plane in optic bench system: quaternion 25-27: Observatory data: orientation of optic bench in inertial system: translation 28-31: Observatory data: orientation of optic bench in inertial system: quaternion
Description:	A reconstructed hit

Keyword:	OR the keyword is optional!
Parameters:	1-7: Space craft relative to inertial 8-14: Focal plane relative to space craft 14-21: Focal plane module 1 relative to focal plane 22-28: Focal plane module 2 relative to focal plane 29-35: Metrology detector 1 relative to focal plane 36-42: Metrology detector 2 relative to focal plane 43-49: Aperture 1 relative to focal plane 50-56: Aperture 2 relative to focal plane 57-63: Optical bench relative to focal plane 64-70: Optics module 1 relative to optical bench 71-77: Optics module 2 relative to optical bench 78-84: Metrology laser 1 relative to optical bench 85-91: Metrology laser 2 relative to optical bench 92-98: Star tracker 4 relative to optical bench
Description:	All orientations at event time

Appendix A

Mast Bending report

The bending of the mast due to the sun/shade is a very important effect which needs to be implemented in NuSIM. Simulations of the mast bend are available for a handful of solar angles. These are given as deviations in the location of the optical axis on the focal plane, and assuming that the mast will arc as it bends, we decompose the transformation between the two benched due to the mast, as a rotation and translation. We then generate a database from these values and input them into NuSIM.

A.1 Mast bend model and database generation

A.1.1 Modeling the Mast as an Arc

The document "Thermal_Distortion_2_for_JPL.xlsx" details the effects on the mast under the thermal effects of sunlight. In particular, it provides x and y offsets of the optical axis impingement on the focal plane, as well as θ_z, the twisting of the mast itself around the z-axis.

A few assumptions had to be made in order to decompose the mast bend into a rotation quaternion, Q_{fbob} and a translation, T_{fbob} which together create R_{fbob}:

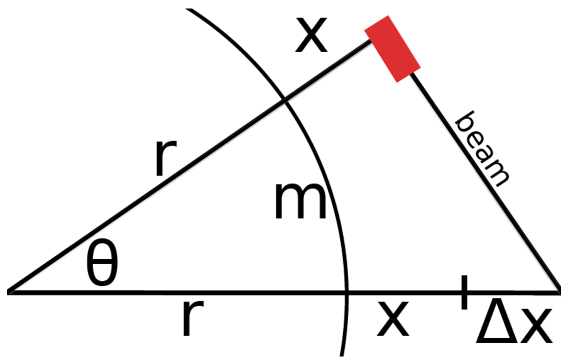


Figure A.1: Mast model.

- The twist in the mast is small enough that its effect on the bend of the mast is not worth calculating, which is to say it can be interpreted as one of the benches at either end having been rotated about the mast.
- The mast bend in x and y dimensions can be interpreted as separate arcs
- The optical axis from the optics modules are exactly parallel to the mast when there is no bend.
- Over these small angles, second order approximations of cosine are acceptable.

Figure A.1 illustrates the mast model geometry. The coordinate system in this image and for the remainder of this document is aligned such that z points up, and y points into the paper. The mast is represented by the arc labelled 'm' , which has length $m = \theta r$. The red square represents the optic and the vertices labelled 'beam' the optical axis of the optic. In the case of no bend, where $\theta = 0$, the optical axis would intersect at r+x. Because of the mast bending, the axis now intersects at r+x+Δx. In addition, the mast also rotates around its own axis by θ_z not illustrated in the figure.

First, to align the coordinate of the mast with the focal plane properly, the distortion of Δx and Δy must be modified by θ_z:

$$r = \sqrt{x^2 + y^2} \tag{A.1}$$

$$\psi = \arctan\left(\frac{\Delta x}{\Delta y}\right) \tag{A.2}$$

$$\Delta Y = r\sin(\psi - \theta_z) \tag{A.3}$$

$$\Delta X = r\cos(\psi - \theta_z) \tag{A.4}$$

$$\tag{A.5}$$

Let m be the length of the mast itself. The arc angle, θ, and the arc radius, r, can be determined as follows:

The mast is essentially an arc (of radius r) with the optics module extending out by a distance x. The angle, θ, of the arc can be found from:

$$\cos(\theta) = \frac{r + x}{r + x + \Delta x}$$

A general rule of arcs is:

$$m = \theta r$$

So:

$$\cos(\theta) = \frac{\frac{m}{\theta} + x}{\frac{m}{\theta} + x + \Delta x}$$

This is not solvable analytically, so here we shift to a second order approximation of cos, which should be accurate for these small angles:

$$1 - \frac{\theta^2}{2} \approx \frac{\frac{m}{\theta} + x}{\frac{m}{\theta} + x + \Delta x}$$

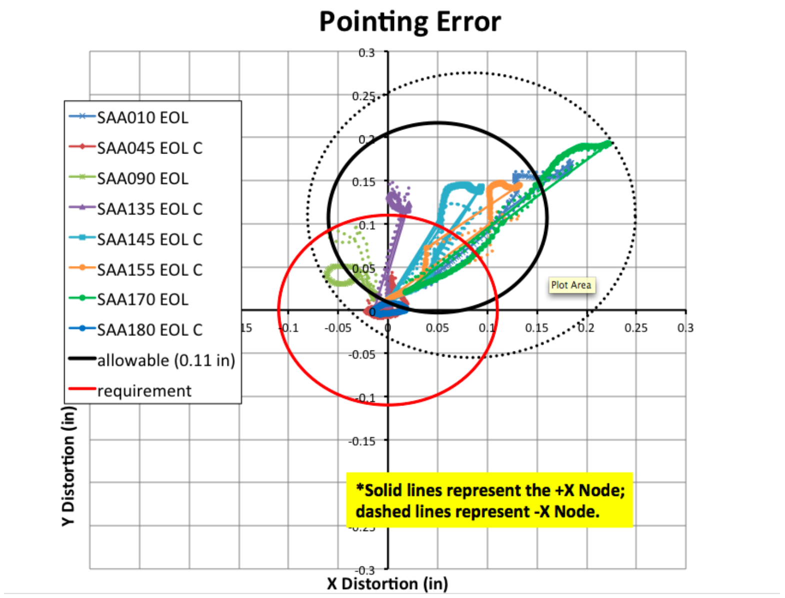


Figure A.2: Simulated Thermal scenarios.

For which the solutions for θ are:

$$\theta = \frac{-m \pm \sqrt{m^2 + 8x\Delta x + 8\Delta x^2}}{2\left(x + \Delta x\right)}$$

Given that when $\Delta x = 0$, $\theta = 0$, the correct solution is:

$$\theta = \frac{-m + \sqrt{m^2 + 8x\Delta x + 8\Delta x^2}}{2\left(x + \Delta x\right)}$$

Which does indeed grow as expected as Δx grows, so it seems a reasonable solution.

Given θ, and $r = \frac{m}{\theta}$, It should be clear that the optics bench will move by $r\left(1 - \cos\left(\theta\right)\right)$ along the x axis and $r \sin\left(\theta\right) - m$ along the z axis.

The calculated bend of the mast by a displacement Δx, is a rotation around the y-axis and vice versa.

A.1.2 Database creation

We have converted three thermal scenarios into databases: SAA90, SAA135 and SAA170. Figure A.2 shows the mast distortions of several solar angles, and SAA90 is considered a 'good case', SAA135 a "conservative case', and SAA170 a 'very bad case'. SAA90 and SAA135 have distortions within the allowed range, while SAA170 is slightly larger than the allowable distortion.

Figures A.3, A.4 and A.5 show the footprint of the optical axis on the focal plane for the three thermal scenarios. The red diamonds are the simulated thermal distortions

$(\Delta x, \Delta y)$ which are used in the formulas presented above to derive the transformation between the benches, R_{fbob}. The black stars are the result of ray-tracing the optical axis using the obtained transformation R_{fbob}. The agreement is good and tolerable considering the approximations that went into the derivation, and we consider this a good enough representation of the mast bend for use in NuSIM.

A.2 NuSIM results

The challenge for NuSIM is to reconstruct the pointing when using the mast bending databases as input. The NuSIM aspect reconstruction is insensitive to relative bend of the two benches, and it is therefore important to discover how well a pointing can be reconstructed.

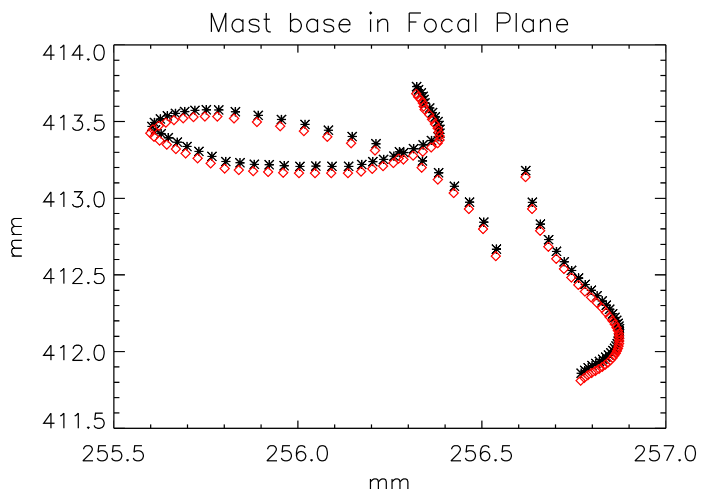
Figures A.6, A.7 and A.8 show the reconstructed Ra and Dec of the pointing. In running this test we used perfect optics without scattering included or any error terms in the sensors. The error, as can be seen, is very small, and in all cases less than the width of a pixel (12").

A.3 Comparison with external code

Even though the error is small it is important to understand its source and so we compared the reconstructed transformation, R_{fbob}, which lacks information about rotations around x- and y-axis, with the original, R_{true}. Figures A.9, A.10 and A.11 show the optical axis ray-traced with the true, R_{true}, in red diamonds and reconstructed, R_{recon}, in black dots. The error here is due to the fact that the reconstructed transformation only has a rotation around the z-axis, and has replaced the rotations around x-, y-axis with translations. The error in the pointing is thus due to the slight distortion in the footprint which can now be seen to be less than a pixel width (0.6mm).

A.4 Conclusion

The NuSIM aspect reconstruction can not perfectly reconstruct the mast pointing due to the fact that it cannot tell a rotation of the plane from a translation of the plane. However, the errors are very small and within the budget. The mast bend databases are therefore considered valid and the aspect reconstruction of the mast movement understood, acceptable and thus verified.



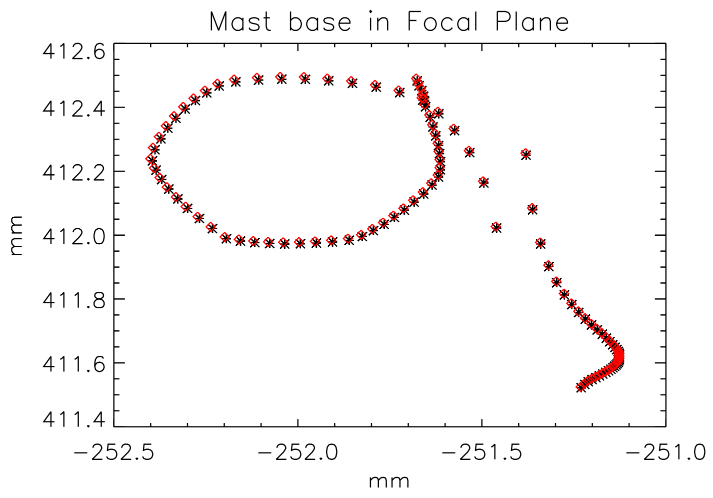
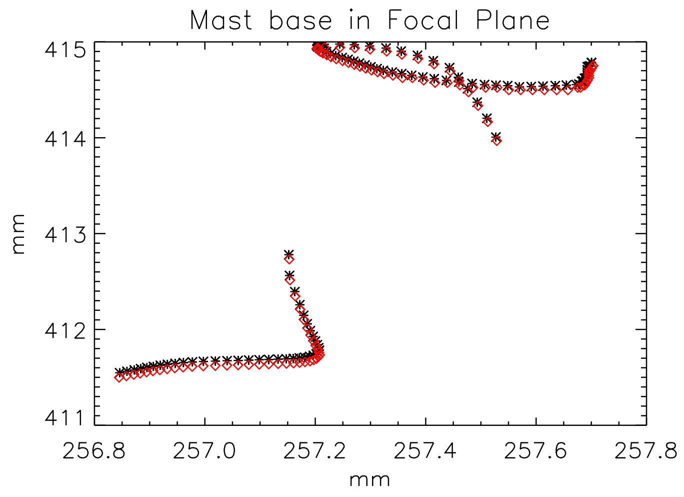


Figure A.3: SAA90 thermal mast bend distortions. Coordinates are in focal plane. Top: distortion footprint of module 1. Bottom: distortion footprint of module 2. Black stars are the ray-traced intersections of the optical axis using the transformation of the benches obtained from the thermal mast bend database which are plotted in red diamonds.



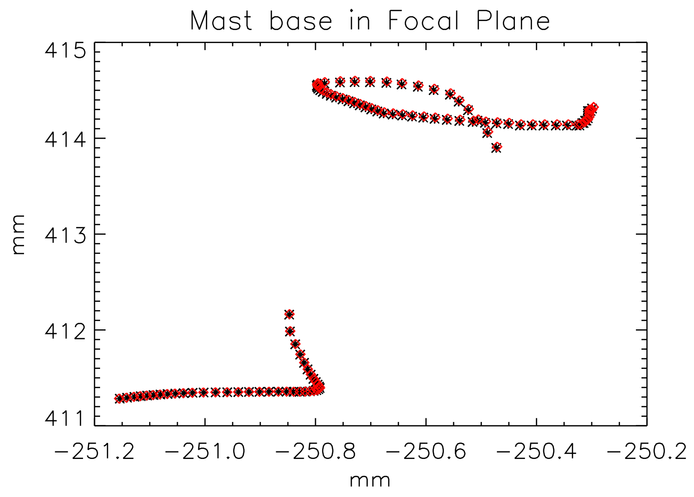
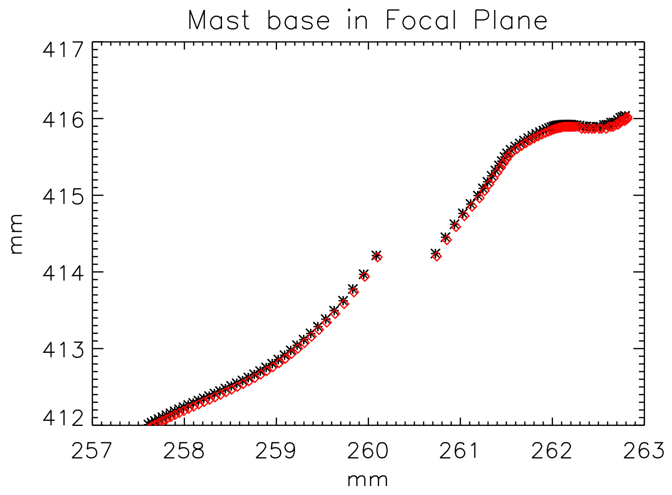


Figure A.4: SAA135 thermal mast bend distortions. Coordinates are in focal plane. Top: distortion footprint of module 1. Bottom: distortion footprint of module 2. Black stars are the ray-traced intersections of the optical axis using the transformation of the benches obtained from the thermal mast bend database which are plotted in red diamonds.



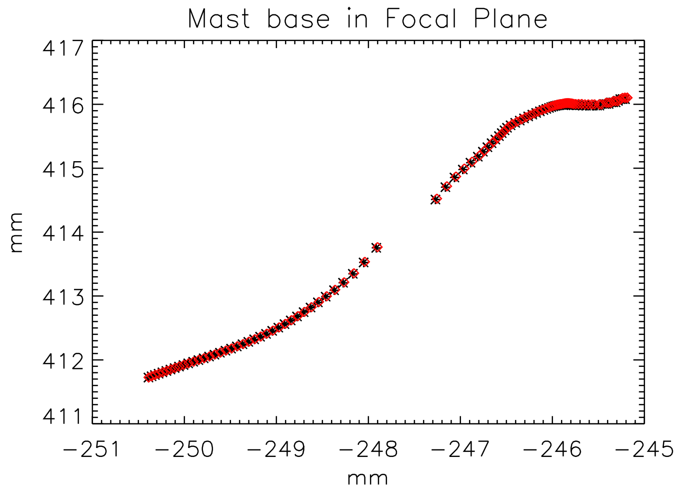


Figure A.5: SAA170 thermal mast bend distortions. Coordinates are in focal plane. Top: distortion footprint of module 1. Bottom: distortion footprint of module 2. Black stars are the ray-traced intersections of the optical axis using the transformation of the benches obtained from the thermal mast bend database which are plotted in red diamonds.

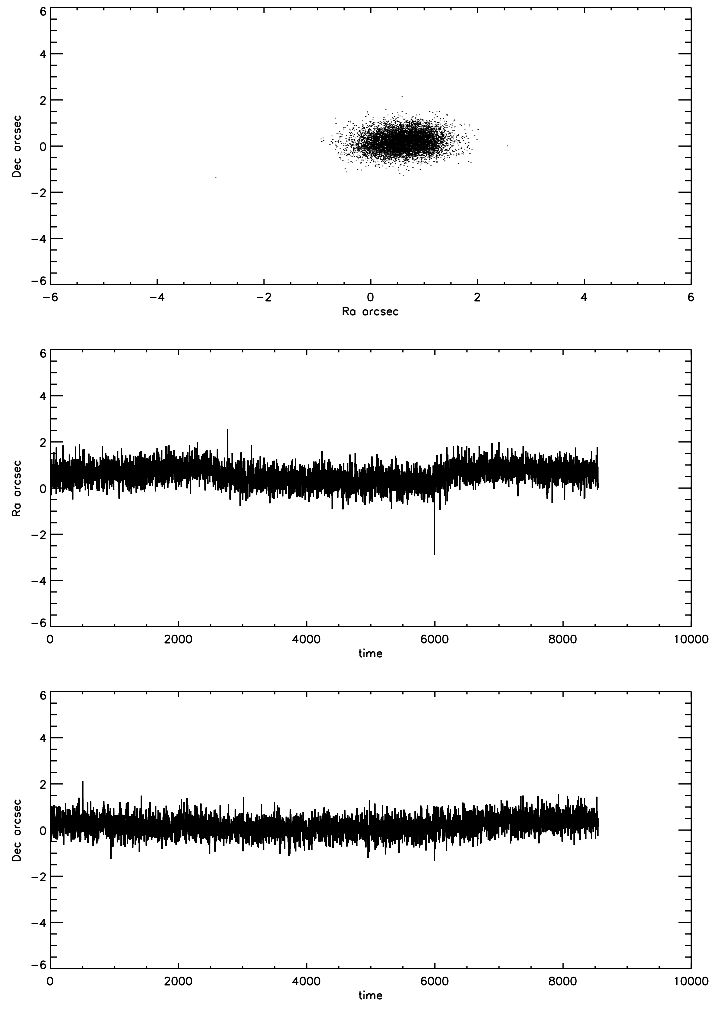


Figure A.6: Aspect reconstruction of SAA90 thermal mast bend. Top: Ra and Dec error. Source pointing is at RA,DEC=0,0. Middle: Ra error as a function of time. Bottom: Declination error as a function of time.

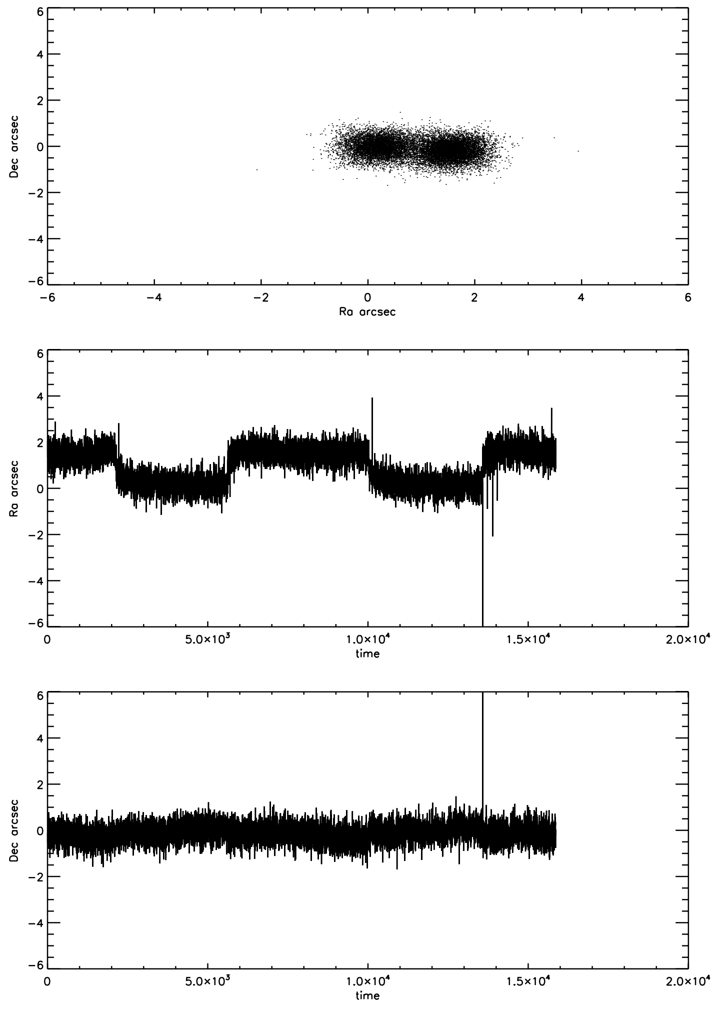


Figure A.7: Aspect reconstruction of SAA135 thermal mast bend. Top: Ra and Dec error. Source pointing is at RA,DEC=0,0. Middle: Ra error as a function of time. Bottom: Declination error as a function of time.

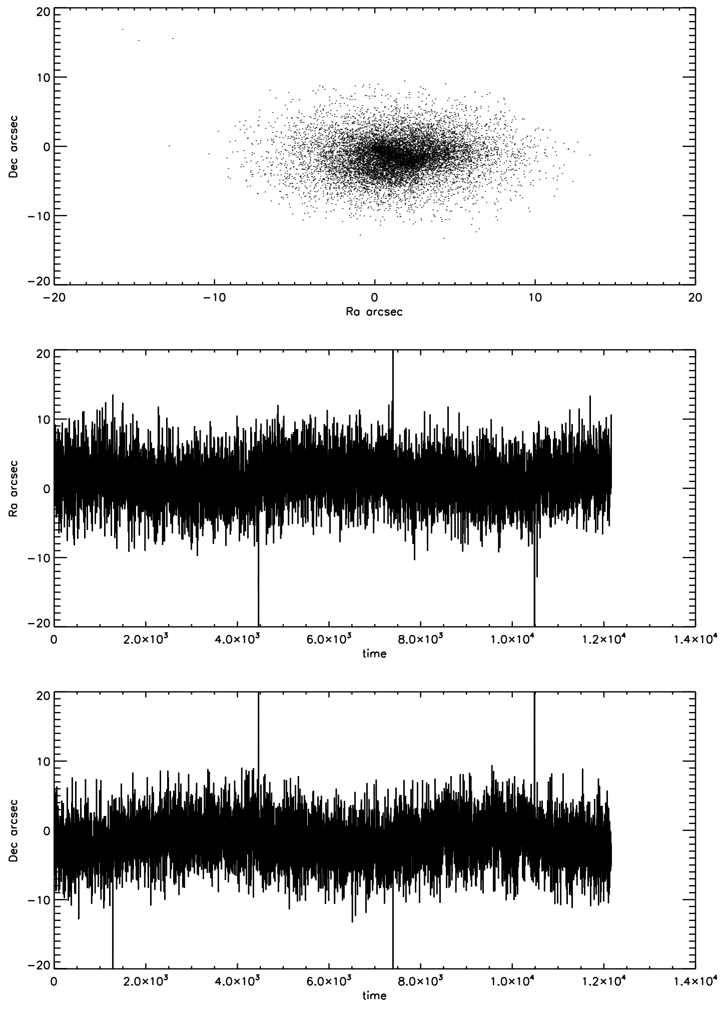
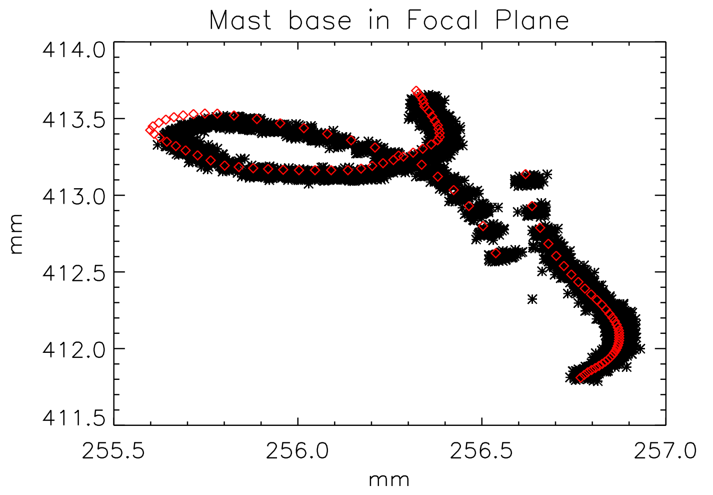


Figure A.8: Aspect reconstruction of SAA170 thermal mast bend. Top: Ra and Dec error. Source pointing is at RA,DEC=0,0. Middle: Ra error as a function of time. Bottom: Declination error as a function of time.



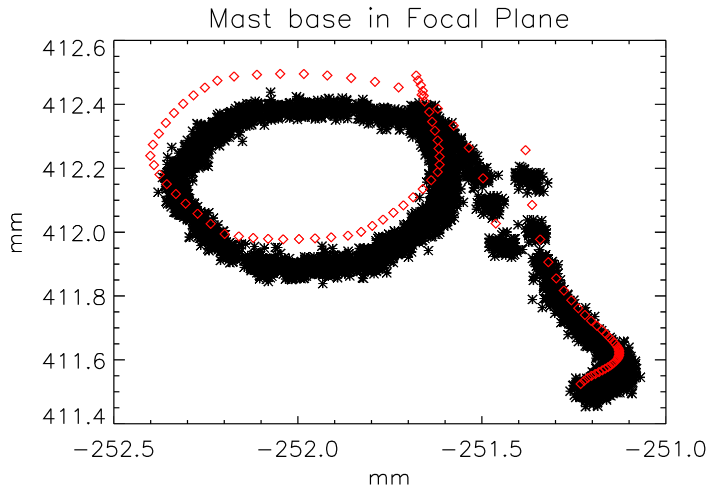
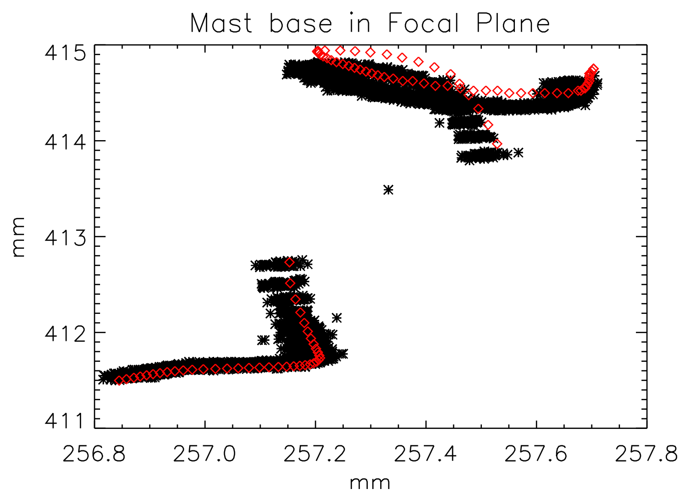


Figure A.9: SAA90 thermal mast bend distortions. Coordinates are in focal plane. Top: distortion footprint of module 1. Bottom: distortion footprint of module 2. Black stars are the ray-traced intersections of the optical axis using the reconstructed transformation of the benches obtained from the NuSIM. The original transformation is plotted in red diamonds.



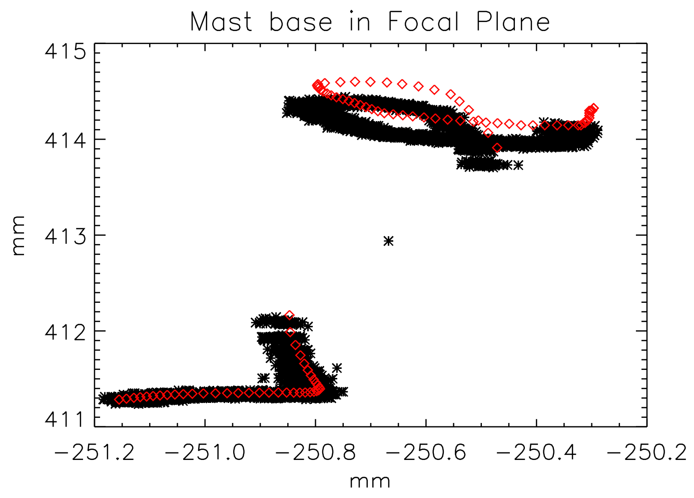
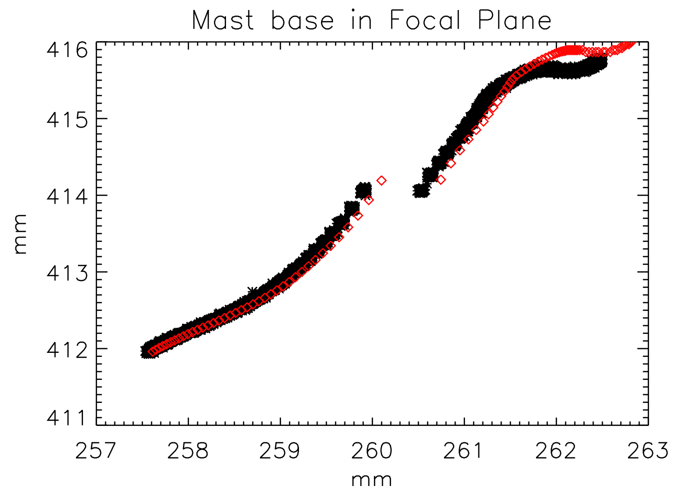


Figure A.10: SAA135 thermal mast bend distortions. Coordinates are in focal plane. Top: distortion footprint of module 1. Bottom: distortion footprint of module 2. Black stars are the ray-traced intersections of the optical axis using the reconstructed transformation of the benches obtained from the NuSIM. The original transformation is plotted in red diamonds.



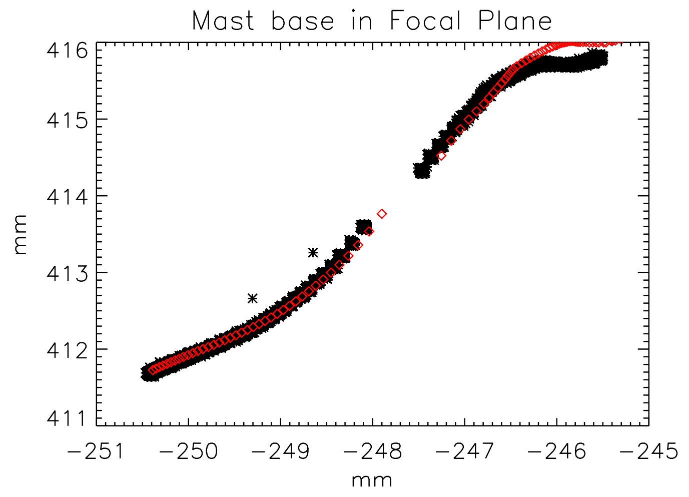


Figure A.11: SAA170 thermal mast bend distortions. Coordinates are in focal plane. Top: distortion footprint of module 1. Bottom: distortion footprint of module 2. Black stars are the ray-traced intersections of the optical axis using the reconstructed transformation of the benches obtained from the NuSIM. The original transformation is plotted in red diamonds.