Modified HIDE

Lucas Collis Olivari

September 20, 2018

1 Introduction

HIDE is a software for simulating a single-dish radio-telescope survey. It takes HEALPix maps as inputs (this will be the output of the SKY pipeline) and processes them into TOD (time-ordered data).

In the following, I describe a modified version of HIDE suited to simulate the BINGO experiment or any other experiment with multiple horns. I first explain how install and run the software and then explore the general structure of it.

2 Installing the Modified HIDE

First, you will download and install HIDE. To do so just do the following:

i) git clone https://github.com/cosmo-ethz/hide

ii) cd hide

iii) pip install -r requirements.txt

iv) python setup.py install

Then you will download and install HIDE4BINGO. To do so just do the following:

i) git clone https://lolivari@bitbucket.org/lolivari/hide4bingo.git

ii) cd hide4bingo

To modify HIDE then type in a terminal:

i) python hide4bingo_install.py

The result will be a modified HIDE, i.e., the original HIDE with some files changed or added. See below for the details.

3 Running Modified HIDE: a Quick Guide

The modified HIDE expects two input files (located in the same path as "run_hide.py"): "altitude.txt" and "azimuth.txt". These files determine the number of horns that will be simulated and assign to each one of them an altitude and an azimuth to be observed.

The spectrometer model is chosen in the "run_hide.py" file. It can either be 1 or 2 (see the details below for the description of these Fake BINGO models). The spectrometer templates should be simulated **before** running HIDE. To do so go to the "data" directory and type "python make_fake_bingo_model_1", to simulate the model 1, for instance. Note that there are some parameters that must chosen in these simulation files. Of course, you can add your own model and modify "run_python.py" accordingly.

The sky maps must be in units of brightness temperature (Kelvin) and should be in a single "fits" file in the format: n_channels vs n_pixels. The code will also expect a "fits" file with the observed frequency of the maps (a simple array with n_channels entries) so that HIDE can iterate over the sky maps array. They should be located at the "data/sky" directory.

The configuration files are located in the "config" directory. To simulate the BINGO experiment, use the file "bingo.py". It is in this file that you should modify the main parameters of the simulation. Please, do not fill in the parameters that are going to be filled in by "run_hide.py" (they are are the blank ones).

To actually **run** the software just type: python run_hide.py

The **output** will be located in a directory name "year", where year stands for the chosen year of the observation (if more than one year is simulated, there will be more than one output directory). All TOD files are in the HDF5 format, and are compressed using the common lossless gzip compression. Each HDF5 file contains several "dataset" units. Two datasets are used to store the time-frequency planes ("P/Phase0" and "P/Phase1) and an additional dataset is used for the time-axis that records the time-stamp for each pixel ("TIME"). These two time-frequency planes include the measured spectrum (P) for both voltage phases (Phase0 and Phase1). Actually, at the moment, only one phase (1) is being calculated. I think that this makes the signal to be a factor of 2 lower than expected – I am checking on this. An example of how to handle the HDF5 files is presented in the "plot_tod.py" file.

The software also output a file with the observed coordinates (it is a list of az/el and ra/dec). It is named "coord_bingo_i_date.txt", where i is the number of the horn and "date" is the date of the simulation.

4 Code Structure

This modified version of HIDE has one main goal: to simulate the BINGO experiment. Therefore, the assumptions about this experiment are the first thing to be described below. Throughout the text, to the benefit of the user/reader, a lot of unanswered questions are highlighted. These questions are crucial and must someday be addressed so that this software can provide realistic simulations of BINGO.

After describing BINGO, I describe the main modules that are needed to simulate BINGO: beam, survey strategy, astronomical signal, convolution of signal with beam, gain, baseline, RFI, and noise.

4.1 Experimental Setup: BINGO

BINGO, at the best of my knowledge and based on Battye et al (2016), involves two static dishes, each with a diameter of $\sim 40\,\text{m}$. These dishes will be significantly under-illuminated ($\sim 25\,\text{m}$) in order to suppress sibelobes, which will make the beam have a full-width half-maximum (FWHM)

Table 1: Instrumental and observing parameters for BINGO.

Parameters	
Latitude (deg)	
Longitude (deg)	
Telescope elevation (m)	
Mean horns azimuth (deg)	
Mean horns altitude (deg)	
Redshift range, $[z_{\min}, z_{\max}]$	[0.13, 0.48]
Bandwidth, $[\nu_{\min}, \nu_{\max}]$ (MHz)	[960, 1260]
Channel width, $\Delta\nu$ (MHz)	1
Number of feed horns, $n_{\rm f}$	
Sky coverage, $\Omega_{\rm sur}$ (deg^2)	
Observation time, $t_{\rm obs}$ (yr)	1
System temperature, $T_{\rm sys}$ (K)	50
Illuminated dish diameter (m)	25
Mean beamwidth, $\theta_{\rm FWHM}$ (arcmin)	40

of $\theta_{\rm FWHM} = 40$ arcmin at an observing frequency of 1 GHz (the resolution is frequency-dependent). The telescope structure and the focal plane array of ~ 50 horns will be supported by a mechanical structure. The receiver system will be a pseudo-correlation design in order to suppress the $1/f$ noise and achieve a knee frequency of $\sim$ 1 mHz over a bandwidth of 1 MHz. The output of each receiver chain will be the difference between the signal received from a main horn pointing at the dish and a reference horn which will be pointing towards one of the celestial poles (**QUESTIONS: (I) IS THIS STILL TRUE? (II) SHOULD WE ACTUALLY SIMULATE THIS DIFFERENTIATION?**). There will need to be as many reference horns as main horns, and each will have a beam size of ~ 25 deg (**QUESTIONS: (I) DOES THIS MEAN THAT ONLY ~ 25 HORNS WILL OBSERVE THE SKY FOR SCIENCE? (II) WHAT ARE THE ALTITUDE AND AZIMUTH OBSERVED BY EACH HORN?**). The overall system temperature, $T_{\rm sys}$, of this uncooled system will be $\sim$ 50 K and have an overall instantaneous bandwidth of $\Delta\nu_{\rm inst} = 300$ MHz from 960 MHz to 1260 MHz corresponding to a redshift range of $z \sim 0.13 - 0.48$.

The telescope will perform a drift-scan survey at declination $\sim X$ (**WARNING: THIS DEPENDS ON THE MEAN ALTITUDE AND AZIMUTH OBSERVED BY THE HORNS**). The receivers will be arranged so as to create an instantaneous field-of-view with a width of X deg in the direction perpendicular to the scan and up to X deg in the direction of the scan (**WARNING: AGAIN THESE NUMBERS DEPEND ON THE ALTITUDE AND AZIMUTH OBSERVED BY THE HORNS**). This will facilitate a survey of X deg $\times$ 360 $\cos(X^\circ)$ deg.

The location of the telescope will be in the Brazilian state of Paraíba, in the northeast region of the country, more precisely, at latitude X and longitude X (**QUESTION: ARE THESE VALUES FIXED?**).

The spectrometer has a phase-switch implementation. Following the default configuration of HIDE, I assume a single polarization for BINGO (**QUESTION: ARE THESE ASSUMPTIONS ABOUT THE RECEIVER OK?**).

In Table 1, I summarise the main parameters that are needed for simulating BINGO.

4.1.1 Code Implementation

HIDE, originally, assumes a single feed horn. To the best of my knowledge, however, a feed horn is just a device to increase the field-of-view of the experiment, with each one working as an independent telescope (**QUESTION: IS THIS CORRECT?**). In this case, the solution found by me has been:

i) Read horns azimuth and latitude from two user-produced files (names: "altitude.txt" and "latitude.txt");

ii) Produce several HIDE configuration files, one for each horn (saved as: "bingo_horn_i.py", where i is the number of the horn);

iii) Run HIDE several times, once for each configuration file, producing at the end TODs for each horn separately (suggested notation: "bingo_tod_horn_mode_date.h5", where mode is the number of the horn and date is the date of the TOD).

The use of HIDE, therefore, will be "hidden" from the user, who will use an simple python script (name: "run_hide.py") instead. A more elegant solution obviously exist, which is to modify HIDE so it can take into account the existence of more than one horn. The performance (time and memory) of the two solutions, however, assuming a serial run in the horn dimension, will be the same.

The inputs of HIDE will then be:

i) A standard BINGO configuration file (name: "bingo.py"), where the user will specify several parameters of the experiment, e.g., effective dish diameter, latitude, longitude, the receiver model, etc.

ii) Two files with the azimuth and latitude observed by each horn. Note that this already assumes a drift-scan strategy for the experiment.

iii) There are also several templates that must be provided at the directory "data" so the spectrometers (one for each horn) can be simulated (see below for details of how to simulate them).

Files of interest: "run_hide.py" from "hide" and "bingo.py" from "config".

4.2 Beam

A beam response pattern is loaded according to the configuration file. The current design supports parameterised Gaussian or Airy profiles and arbitrary beam patterns specified on a grid. The beam profiles can be frequency-dependent (**COMMENT: WITHOUT A PARAMETERISED BEAM, THIS ONLY MEANS THAT THE RESOLUTION IS FREQUENCY-DEPENDENT. THERE ARE NO STANDING WAVES OR OTHER SOURCES OF IMPERFECTION IN THE IDEAL SIMULATIONS**).

4.2.1 Code Implementation

In order to convolve the beam response with the simulated astronomical signal, we have to rotate the grid that defines the beam geometry on a sphere. As this step is repeated for every telescope pointing defined by the scanning strategy as well as for every simulated frequency, high efficiency for the operation is crucial.

Conventionally, spherical rotations are implemented by using the Euler matrix rotation. Applying this rotation scheme to a HEALPix map requires transforming the spherical pointing angles (θ and ϕ) into Euler coordinates (x, y, z). This involves computing the rotation matrix R, applying this matrix to the coordinate vector, transforming the results back into spherical coordinates, and finally performing the convolution. Typically, applying rotation on multiple axes requires a repetition of the above steps for each axis. This can be computationally even more demanding and numerically less stable.

HIDE has implemented all the coordinate rotations with Quaternions, a technique commonly used in 3D computer vision. Rotations over multiple axes can be concentrated into one operation with Quaternions, which make them computationally more efficient and stable. In order to efficiently find the pixels relevant for the rotation of the beam geometry HIDE also stores all the HEALPix pixel information in a KD-Tree adapted for spherical coordinates.

Files of interest: all files from "beam" and "combine_signals.py", "coord_transform.py", "qu_opt_coord_transform.py", and "load_beam_profile.py" from "plugins".

4.3 Survey Strategy

The pointing of the telescope at a given time in the survey is computed according to the desired survey strategy - BINGO will adopt a drift-scan strategy. The module converts the telescope pointing from terrestrial coordinates (azimuth, altitude or elevation) into equatorial coordinates (RA, Dec) and writes these coordinates into the coordinates final file. Different strategies can be chosen such as drift-scan surveys, or a file-based scanning schedule such that a planned survey can be exactly simulated.

Files of interest: all files from "strategy", mainly the "drift_scan.py", "sphere.py" from "utils", and "scanning_strategy.py" and "write_coord.py" from "plugins".

4.4 Astronomical Signal

The astronomical signal used for the simulation is loaded. In this step, we will use our SKY pipeline, which will calculate: the 21 cm and foregrounds (Galactic and extragalactic).

The sky maps must be in units of brightness temperature (Kelvin) and should be in a single fits file in the format: n_channels vs n_pixels. The code will also expect a fits file with the observed frequency of the maps (a simple array with n_channels entries) so that HIDE can iterate over the sky maps array.

4.4.1 Code Implementation

The routine that will read in the sky maps and the sky observed frequencies is the "hi_sky.py", located in the "astro" directory. The data (sky maps and sky observed frequencies) should be located in the "data/sky" directory of HIDE (in your working path).

In the "bingo.py", the entries "astro_signal_file_name" and "astro_signal_freq_file_name" must be filled with the appropriated file names.

Files of interest: all files from "astro", mainly "hi_sky.py", and "astro_signal.py" from "plugin".

4.5 Convolve Signals with Beam

For each telescope pointing defined by the survey strategy, the telescope beam response is convolved with the astronomical signal and appended to the TOD array.

Files of interest: see the section about the beam above.

4.6 Gain

To transform the TOD from units of Kelvin into internal units (Analog-to-digital unit, ADU) as recorded in the instrument, we multiply the TOD by a gain template. This information can come from external calibration or specifications of the instrument.

Originally, HIDE uses data obtained with the Bleien telescope. I added two fakes BINGO gain templates as options. Both assume a total bandwidth of 300 MHz and a channel width of 1 MHz.

In the first model, the gain is given by

$$G(\nu) = 1 + \epsilon(\nu) + \epsilon_{\rm osc} \sin\left(\frac{\pi\nu}{\Delta\nu_{\rm osc}}\right), \tag{1}$$

where $\epsilon(\nu)$ is a parameter that is drawn from a certain Gaussian distribution for each frequency ν and adds random errors to the gain, $\epsilon_{\rm osc}$ is the oscillatory amplitude of the gain, and $\Delta v_{\rm osc}$ is a parameter that controls the period of the oscillation of the gain. Note that in this model the mean relation between Kelvin and ADU is 1.

In the second model, the gain is given by

$$G(\nu) = A\left(1 + \epsilon(\nu) + \epsilon_{\rm osc} \sin\left(\frac{\pi\nu}{\Delta\nu_{\rm osc}}\right)\right), \tag{2}$$

where A is the gain amplitude (the Bleien data mean value), $\epsilon(\nu)$ is a parameter that is drawn from a certain Gaussian distribution for each frequency ν and adds random errors to the gain, $\epsilon_{\rm osc}$ is the oscillatory amplitude of the gain, and $\Delta v_{\rm osc}$ is a parameter that controls the period of the oscillation of the gain.

For more detail, see either "make_fake_bingo_model_1.py" or "make_fake_bingo_model_2.py" in the "data" directory.

4.6.1 Code Implementation

The fake BINGO spectrometer is implemented in the routine "fake_bingo_spectrometer.py" in the "spectrometer" directory.

Files of interest: "make_fake_bingo_model_1.py" and "make_fake_bingo_model_2.py" from "data", "fake_bingo_spectrometer.py" from "spectrometer", and "apply_gain.py" from "plugins".

4.7 Baseline

An frequency and point dependent baseline offset is added to the TOD to account for contributions to the overall intensity from the instrument, the environment, etc. (**QUESTION: I IMAGINE THIS IS SOMEHOW CORRELATED TO THE GAIN, IS THIS TRUE?**)

HIDE makes the simplified assumption that the TOD has a frequency- and elevation-dependent baseline that is otherwise independent of azimuth and time of the pointing as well as its Galactic coordinates. It also assumes that the frequency- and elevation-dependencies factorise.

Originally, HIDE uses data obtained with the Bleien telescope. I added two fake BINGO baseline templates.

In the first model, the baseline is given by:

$$B(\nu) = 1 + \epsilon_{\rm osc} \sin\left(\frac{\pi\nu}{\Delta\nu_{\rm osc}}\right), \tag{3}$$

where $\epsilon_{\rm osc}$ is the oscillatory amplitude of the baseline (the same as the gain) and $\Delta v_{\rm osc}$ is a parameter that controls the period of the oscillation of the baseline (the same as the gain). There is no random component in the baseline.

In the second model, the baseline is given by:

$$B(\nu) = A\left(1 + \epsilon_{\rm osc} \sin\left(\frac{\pi\nu}{\Delta\nu_{\rm osc}}\right)\right), \tag{4}$$

where A is the baseline amplitude (the Bleien data mean value), $\epsilon_{\rm osc}$ is the oscillatory amplitude of the baseline (the same as the gain), and $\Delta v_{\rm osc}$ is a parameter that controls the period of the oscillation of the baseline (the same as the gain).

For more detail, see either "make_fake_bingo_model_1.py" or "make_fake_bingo_model_2.py" in the "data" directory.

For now I am not changing the Bleien dependency on elevation. For model 1 I am assuming the following coefficients: $[1., -1., 1]$ and for model 2, $[1.26321397 \times 10^{10}, -1.71282810 \times 10^{10}, 2.79280833 \times 10^{10}]$.

4.7.1 Code Implementation

The fake BINGO spectrometer is implemented in the routine "fake_bingo_spectrometer.py" in the "spectrometer" directory.

Files of interest: "make_fake_bingo_model_1.py" and "make_fake_bingo_model_2.py" from "data", "fake_bingo_spectrometer.py" from "spectrometer", and "add_background.py" from "plugins".

4.8 RFI

A simulated RFI signal is added to the TOD array.

In HIDE this is done in the following way. HIDE uses a simple model for the RFI, where each RFI burst is defined by the same profile in time and frequency (either exponential or Gaussian) and an amplitude. For each frequency, the amplitudes are sampled randomly from a uniform distribution between the standard deviation of the noise and a maximum value inferred from the real data. The rate of RFI bursts per frequency is chosen such that, on average, the number of pixels affected by RFI greater than the noise-level is matching the fraction of masked pixels inferred from the real data (i.e., Bleien data). This model can be implemented efficiently by starting with a zero time-frequency plane and setting the randomly chosen positions in time to the randomly drawn amplitudes for each frequency. The final simulated RFI is then given by a two-dimensional FFT convolution of the time-frequency plane with the chosen RFI profile.

4.8.1 Code Implementation

For now, RFI is not included in the "bingo.py" default simulation and no Fake BINGO RFI has been added. Maybe in a future version of the software, I may add a Fake BINGO RFI for completeness.

Files of interest: "make_fake_bingo_model_1.py" and "make_fake_bingo_model_2.py" from "data", "fake_bingo_spectrometer.py" from "spectrometer", and "add_rfi.py", "add_rfi_phaseswitch.py" and "write_rfi.py" from "plugins".

4.9 Noise

Gaussian (thermal) noise and and $1/f$ noise are added to the data to model noise contribution from the electronics. The $1/f$ noise is produced by gain fluctuations of the amplifiers and is, therefore, correlated across all frequency channels. To simulate the noise we follow the framework presented by Harper et al. (2018), which adopts the following expression for the power spectral density (PSD) of a receiver contaminated with thermal and $1/f$ noise

$$\mathrm{PSD}(f,\omega) = \frac{T_{\mathrm{sys}}}{\delta\nu}\left[1 + C(\beta, N_\nu)\left(\frac{f_{\mathrm{k}}}{f}\right)^{\alpha}\left(\frac{1}{\omega\Delta\nu}\right)^{\frac{1-\beta}{\beta}}\right], \tag{5}$$

where is T_{sys} is the system temperature of the receiver, $\delta\nu$ is the channel bandwidth, f_{k} is known as the knee frequency, α is the spectral index of the noise, ω is the inverse spectroscopic frequency wavenumber, $\Delta\nu$ is the total receiver bandwidth, β is used to parameterise the spectral index of the PSD, and $C(\beta, N_\nu)$ is a normalization constant. The unity term inside the brackets of this equation describes the thermal noise contribution.

4.9.1 Code Implementation

The Kelvin-to-ADU response of the noise is defined in the gain and simulated in either "make_fake_bingo_model_1.py or "make_fake_bingo_model_2.py", depending on the choice of the user.

In the "plugins" directory the file "background_noise.py" was modified (now it has a function called "get_noise_lucas").

In the "utils" directory the file "signals.py" was modified (now it has three extra functions: "noise_amplitude", "thermal_noise_tod", and "color_noise_tod").

Files of interest: "make_fake_bingo_model_1.py" and "make_fake_bingo_model_2.py" from "data", "fake_bingo_spectrometer.py" from "spectrometer", "signal.py" from "utils", and "background_noise.py" from "plugins".