ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP068

IMPLEMENTATION OF AN EXTERNAL DELAY CALCULATOR FOR MeerKAT

B. Ngcebetsha[*], South African Radio Astronomy Observatory, Cape Town, South Africa

Abstract

The MeerKAT is an array of dishes designed to study the mysteries of the distant universe, it is made up of 64 dishes that operate as one telescope. The signal from the distant celestial objects encounters several distortions and is corrupted when it arrives at the receiver. Most of the distortions are due to the medium between the observatory and the object of interest. Each corruption can be quantified and corrected, the corruptions are termed ”propagation effects”. The process of correcting the propagation effects constitutes calibration and takes on various stages during and after the observation is done. The context of this paper takes a close look at signal path delay correction. This is the adjustment of the time of arrival of the signal at the correlator from all 64 antennas. This is required as the signal arrives at different times at every antenna, and the cable from each antenna is of differing length. The MeerKAT CAM system implements a delay update manager. The delay update manager calculates the delay terms, and submits the corrections to the correlator. In this paper, we describe how this solution was evolved when `katpoint` (the underlying library on which the delay corrections depend) had a change in its own dependencies. There were two major changes to `katpoint` 1) utilising `astropy` instead of `ephem`, and this meant 2) migrating telescope code from version 2 to 3. In this paper we explore the lessons learned when `katpoint` started to implement `astropy` which is implemented in `Python3` whilst the rest of the code-base was still in `Python2`. The technical benefit of this update was an improvement in the astrometry for delay calculations which will enhance the MeerKAT science and images.

INTRODUCTION

One of the objectives of radio observations by telescopes such as the MeerKAT is to make a map of the patch of sky observed and create a catalogue with a list of astronomical objects and their physical properties. The telescope is composed of pair-wise groupings of the antennas known as baselines. An illustration of a baseline is Fig. 1, where the signal arriving at both antennas is recorded ideally at the same time for further processing at the correlator. Prior to arriving at the correlator, the signal is a voltage reading, and is converted to a digital signal by the analog-to-digital converter (ADC). The main task of the correlator (dashed-box), is to multiply and time-average (correlation). An extra step in collecting the data is the tweaking of the time of arrival. This is achieved by the correlator introducing a signal

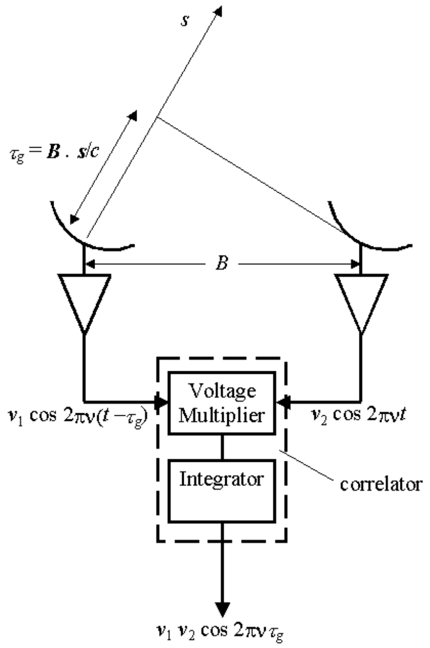

s
$\tau_g = \boldsymbol{B} \cdot \boldsymbol{s}/c$
B
Voltage Multiplier
Integrator
correlator
$v_1 \cos 2\pi\nu(t-\tau_g)$
$v_2 \cos 2\pi\nu t$
$v_1 v_2 \cos 2\pi\nu\tau_g$

Figure 1: An two-antenna interferometer, antenna1(left) detects the signal τ_g seconds later than antenna2(right). Where τ_g is called the geometric delay.

path delay. The delay is usually in the range of nanoseconds (billionth of a second), defined as the time it takes light to travel the distance $\vec{B} \cdot \vec{s}$. This is the distance between the tip of antenna2 on the right and that of antenna1 on the left in Fig. 1.

The antennas sample the signal in an arbitrary plane known as the `uvw` plane, and this is the coordinate system in which the baselines have been positioned. There exists, theoretically - a Fourier relationship between the `uvw` plane and the actual image plane, see Fig. 2. The output of the correlator is a table of complex numbers known as visibility data. The data represents snapshots at every feasible `uvw` coordinate at the location of every baseline for the duration of the observation. In order to make the data scientifically useful, astronomers make an image of the sky through the use of deconvolution algorithms built into imaging pipelines. This process produces better results with improved and accurate measurement of sky positions. The sky position measurements also rely on accurate timing of the arrival of the signal, which has travelled vast distances from outer space. The incident signal is electromagnetic in nature, the wave nature of light implies the phase at which it arrives also needs to be taken into consideration and corrected. MeerKAT generally uses an in-house *Python* library known as `katpoint` for all target-related motions in the control software. Earlier

[*] bngcebetsha@sarao.ac.za

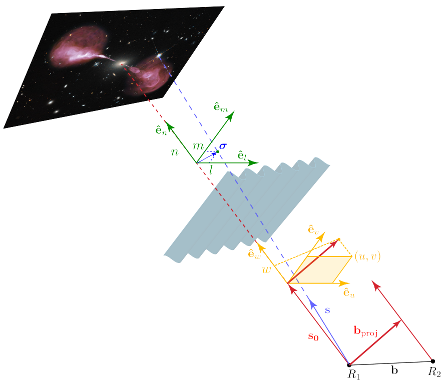


Figure 2: An illustration of the image to uvw plane relationship.

versions of the katpoint used the standard ephem library for various calculations of motions and positions of celestial objects.

To make improvements to the positional accuracy of sources in MeerKAT images there are two areas that we required to be addressed:

1. the way baseline coordinates are generated, and
2. delay calculations.

The potential improvements are both addressed by switching katpoint to utilize astropy rather than ephem as the underlying library for computations of motions and positions of celestial bodies. This poses a challenge for the MeerKAT CAM (Control And Monitoring) software team as astropy is now fully ported to Python3. Currently the CBF (Correlator Beam Former) proxy code is implemented in Python2.7, and it is not feasible to migrate it to Python3 in the required time frame (CAM codebase is yet to be fully migrated to Python3). The CBF proxy package has many other dependencies some of which are also still on Python2.7. As a solution to this issue, CAM created a new service that could run under Python3. The service would provide delay corrections on-demand. This would then require a rework of the CBF proxy to get delay correction values from the new service via KATCP (Karoo Array Telescope Control Protocol) using remote calls. With this approach the rest of CAM would stay on the old version of katpoint, however the delay corrections would be calculated with the new astropy-based version of katpoint. The delay correction model can be instantiated from a single JSON-encoded string, which is the equivalent description of the object. This may be convenient to send over KATCP. The new service is named External Delay Calculator, and will meet the requirements described above.

DELAY MODEL

The diagram on Fig. 3 illustrates the underlying classes of the External Delay Calculator and the methods with their relationships and dependencies. The two rectangles the bottom labelled DelayModel<x> indicate the lowest level. Instances of DelayModel are created to make delay calculations for each target at every antenna.

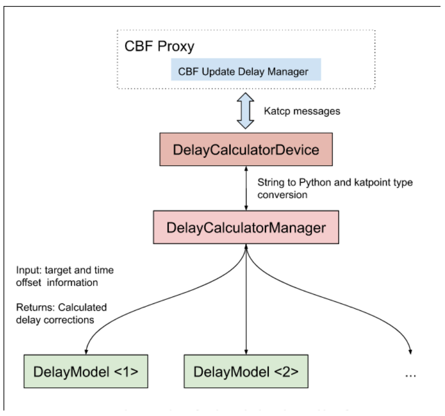

CBF Proxy
CBF Update Delay Manager
Katcp messages
DelayCalculatorDevice
String to Python and katpoint type conversion
DelayCalculatorManager
Input: target and time offset information
Returns: Calculated delay corrections
DelayModel <1>
DelayModel <2>
...

Figure 3: The underlying classes as per the Delay Calculator Design Record. All classes live in seperate modules that comprise the External Delay Calculator.

The instances of DelayModel are repositories for storing delay model parameters, allowing easy construction, inspection and saving of the delay model. The actual calculations happen in katpoint.DelayCorrection, which is more efficient as it handles multiple antenna delays simultaneously.

DELAY CALCULATOR MANAGER

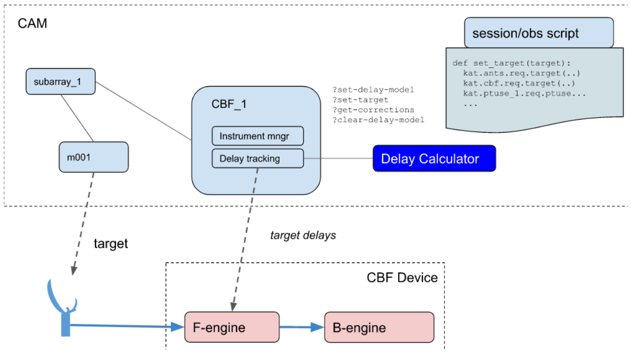
The DelayCalculatorManager will accept instructions from DelayCalculatorDevice to create DelayModel objects that are instances of katpoint.DelayModel, the desired delay calculations for the given target are performed and sent back as string to DelayCalculatorDevice. The delay manager will launch as many instances of katpoint.DelayModel as there are targets. This class inherits from katcore base classes that allow simulation and control of the external delay calculator as a device in CAM.

DELAY CALCULATOR DEVICE

The DelayCalculatorDevice class in Fig. 3 is a KATCP device server for the External Delay Calculator and interfaces with CBF proxy. It receives KATCP requests in the form of JSON strings from CBF proxy (as shown in Table 1) and decodes them into *Python* objects that can be used by DelayCalculatorManager. It then encodes the responses from DelayCalculatorManager to strings and responds to the CBF proxy.

Supported Requests

- ?set-delay-model This request creates a delay model object, using an instance of katpoint.DelayCorrection. The instance of katpoint.DelayCorrection uses delay models from an array of antennas connected to a correlator, and produce delay and phase corrections for a given target and timestamp, for all antennas at once. The delay corrections are guaranteed to be strictly positive.



```
def set_target(target):
  kat.ants.req.target(..)
  kat.cbf.req.target(..)
  kat.ptuse_1.req.ptuse...
  ...
```

Figure 4: A diagram depiction of where within CAM architecture, the External Delay Calculator lives. The part indicated by CBF1 represents CBF proxy which is the interface to the actual CBF device where the delay corrections are applied.

Each antenna is assumed to have two polarisations (H and V), resulting in two correlator inputs per antenna.

- `?set-target` The `?set-target` KATCP request will set a new target for which to calculate delay corrections.
- `?get-corrections` Calculate the delay corrections for a given target and return the results as 2 JSON encoded dictionaries that can be formatted and sent to the CBF instrument.
- `?clear-delay-model` Upon subarray deactivation, this request will remove the given delay model and its targets from memory.

CBF PROXY

At the top of Fig. 4 is represented, the CBF proxy which is the interface used by the observation script to send commands to the correlator. In the context of delay calculations, the CBF proxy will request delays for a given target for all antennas. The underlying classes will then carry out the required computation and return the result to CBF proxy. The delays are then sent to the actual CBF device (correlator) to be applied on the data streams. The MeerKAT CBF data proxy manages the data generation for an array. There is always one data proxy for each array. The underlying device is the actual correlator, called Correlator Master Controller (CMC). The role of the CBF proxy is to facillitate commands to the CMC via KATCP. The CMC responds with a message to the proxy of the outcome, or provides details in the case of unintended consequences. The CMC device exposes a KATCP interface for accepting and responding to KATCP requests.

CONCLUSION

The External Delay Calculator has been integrated into MeerKAT as of end of 2021, it has proven to give more accurate values for the delays in observations. There is currently an open issue pertaining to the rate at which the delays are fetched and applied. The team has planned to apply an appropriate solution in later sprints. The current priority is the integration of the new CBF (CBF PLUS), which will accommodate an extra 20 dishes that will be commissioned as part of MeerKAT PLUS.

Table 1: An example JSON string sent by the External Delay Calculator back to CBF proxy to be sent to CBF device. The keys are time delays and phase fringes for each antenna polarization *h* and *v* and the values in the first dictionary are the delay adjustments in seconds (s) and phase fringes in (deg) for the second dictionary.

```
{'delay_result':
    {'m008h':
        [1.968411243784553e-06, 0.0],
    'm008v':
        [1.968411243784553e-06, 0.0],
    'm001h':
        [1.0084078291945032e-07, 0.0],
    'm001v':
        [1.0084078291945032e-07, 0.0],
    'm060h':
        [1.9448476831146047e-06, 0.0],
    'm060v':
        [1.9448476831146047e-06, 0.0],
    'm051h':
        [1.9477931281983483e-06, 0.0],
    'm051v':
        [1.9477931281983483e-06, 0.0]},
 'fringe_result':
    {'m008h':
        [-3025.2007294890336, 0.0],
    'm008v':
        [-3025.2007294890336, 0.0],
    'm001h':
        [12041.62927381297, 0.0],
    'm001v':
        [12041.62927381297, 0.0],
    'm060h':
        [-2835.099113337884, 0.0],
    'm060v':
        [-2835.099113337884, 0.0],
    'm051h':
        [-2858.8618153567777, 0.0],
    'm051v':
        [-2858.8618153567777, 0.0]}
}
```