FISICA

Far Infra-red Space Interferometer Critical Assessment
Deliverable D4.1

Grant agreement no. 312818

SPA.2012.2.2-01: Key technologies enabling observations in and from space

- Collaborative project -

D4.1
PyFIInS - Python Far Infrared Instrument Simulator

WP 4 – An end-to-end simulator of FIRI

Due date of deliverable: Month 36

Actual submission date:

Start date of project: January 1st 2013 **Duration:** 36 months

Lead beneficiary for this deliverable: Science and Technology Facilities Council (STFC), UK

Lead author: John Lightfoot, UK ATC (STFC)/University of Edinburgh

Contributing authors:

Contributions also from: Roser Juanola-Parramon (NASA Goddard), Colm Bracken and Anthony Murphy (NUI Maynooth), Gibion Makiwa and David Naylor (U.Lethbridge)

Project co-funded by the European Union's Seventh Framework Programme for research, technological development and demonstration

Dissemination Level		
PU	Public	■
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Summary

This document presents a final report on the FISICA instrument simulator PyFIInS. A brief description of the simulator is given. How the software can be obtained and installed on a computer is described. How the simulator should be used is explained through an example.

History table

Version	Date	Released By	Comments
0.1	14/01/2016	John Lightfoot	First draft
1.0	15/01/2016	John Lightfoot	Completed and released version
1.0	19/01/2016	John Lightfoot	Added recommendation for computer power

Table of Contents

Key word list

Instrumentation: high angular resolution

Instrumentation: interferometers

Instrumentation: spectrographs

Instrumentation: detector arrays

Techniques: high angular resolution

Techniques: imaging spectroscopy

Definitions and acronyms

FIRI	Far Infra-Red Interferometer
FITS	Flexible Image Transport System
FTS	Fourier Transform Spectrograph
SPIRE	Spectral and Photometric Imaging Receiver

Acknowledgements

The research leading to this report has received funding from the European Union's Seventh Framework Programme for research, technology development and demonstration under Grant Agreement no. 312818 – FISICA.

Disclaimer

1. Introduction

The theory of double Fourier interferometry is straightforward, the implementation of it in a space-borne instrument is not. A simulator is required to allow modelling of various systematic errors that will inevitably corrupt the data to some degree, and which it is difficult to model theoretically. Such errors include:

- Realistic primary beams for the instrument's flux collectors. Imperfections introduce phase errors that decrease fringe visibility.
- Temperature fluctuations in the instrument that change the measured signal.
- Collector pointing errors. The signal on the detector includes the direct images from each collector and pointing errors cause these contributions to fluctuate. The interferometric component is also affected in a more subtle way.
- Baseline errors.
- Baseline movement during each FTS scan. For a tethered system we would expect the collectors to be continuously revolving around the hub to keep the tether taut and minimise fuel use.
- FTS mirror stage errors. Such errors were measured for SPIRE.
- Detector time constant.
- Cosmic ray strikes on the detector. Statistics derived from SPIRE.

A second use of the simulator is for science verification and public relations. It can be used to 'observe' objects of potential interest to illustrate - with numbers and pictures - how successful or not the proposed observation would be.

2. Requirements

The simulator must be:

- publicly available and able to run on ordinary computer platforms.
- able to model the 'strawman' instrument design.
- flexible so that variations on the strawman design can be tested.
- able to 'observe' any object. A limitation of the Matlab version was that it took too long to observe complex sources.

3. PyFIInS Description

PyFIInS is derived from a simulator written in Matlab by Roser Juanola-Parramon as part of her PhD ('A Far-Infrared Spectro-Spatial Space Interferometer'. UCL. 2014). To fulfil the requirement that the simulator be free and runnable on a wide range of machines the FISICA simulator is implemented in Python.

The simulator is implemented in 2 parts:

1. A part that generates the interferograms that would be observed of the given target. This program is called PyFIInS and takes as input parameters:
    a. An Excel file describing the instrument configuration and observation timeline. A version of this file describing the strawman instrument is released with the code. The file contains a number of spreadsheets, each associated with a particular aspect of the instrument. The instrument description is quite detailed allowing, for example, changes to the number, performance and temperature of the optical elements in the system. The spreadsheets most relevant to the ordinary user are:
        i. FTSpectrograph, specifying the Band (B1, B2, B3 or B4) of the simulation and associated FTS scan details
        ii. Telescope, describing the diameters of the flux collectors, their pointing behaviour, etc.
        iii. Interferometer, describing the *u*-*v* pattern to be swept out by the instrument through the observation.
        iv. Detector, describing the detector time constant, the 'knee' frequency of its 1/f noise, etc.
    b. A file containing the target cube (the emission at each point in an array with dimensions [nx, ny, nfreq]) in FITS format.
2. A program called PyDataProcessing, which reads in the FITS file with the interferograms produced by PyFIInS and reduces them to give the cube that is the result of the observation.

The 'strawman' instrument specified by the release Excel file is the same as that described in D1.5 and efforts have been made to match the simulator results with the sensitivity model described there.

Output from both programs comprises files written in FITS format, and html directories forming a results 'weblog' that can be examined with a browser. Example views from the weblogs of PyFIInS and PyDataProcessing are shown in the 2 figures below. The first shows one frequency plane of the target cube being 'observed' by PyFIInS. On it you can see four disks of varying size and orientation, playing the part of pre-stellar cores in a molecular cloud. The second shows the same frequency plane of the output cube. You can see the same four disks but their emission has been convolved with the 'dirty' beam associated with the incomplete *u*-*v* coverage of the observation. The buttons along the toolbar of each page give access to other parts of the result structure.

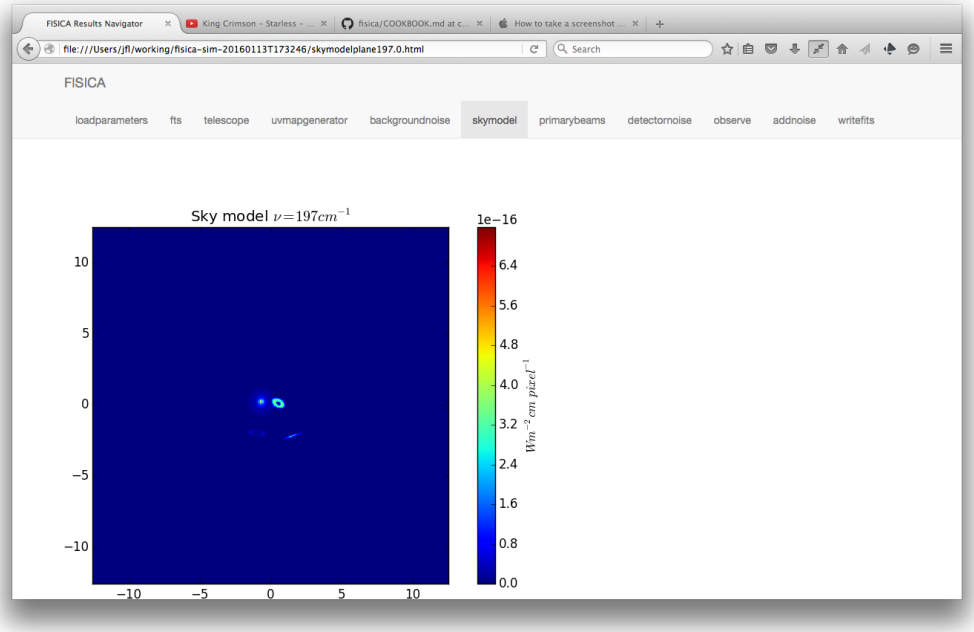

Figure 1 Weblog rendering of the 197cm-1 plane of the target cube

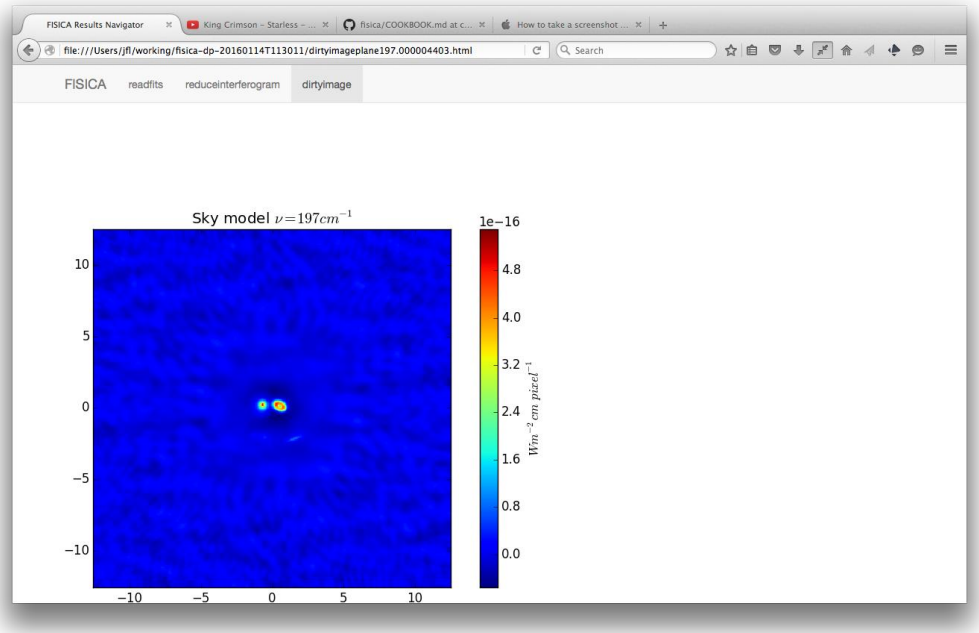

Figure 2 Weblog rendering of the 197cm-1 plane of the result cube

4. Obtaining and Installing PyFIInS

The following installation procedure applies to a machine running Linux or Mac OSX. The simulator will run on Windows machines if the required Python modules are installed first.

The simulator demanding of memory and CPU power. Simulations in Band 4 (25-50cm^{-1}) run satisfactorily on a MacBook Pro with a dual core CPU and 8GB of memory. Simulations in Band 2 (100-200cm^{-1}) do not, for these a desktop machine with roughly 60GB of memory has been found to work well. If a run does not achieve a 'user mode' CPU usage of 50% or more then you should consider using a machine with more memory.

1. The code runs under a Python 2 interpreter and requires the following Python modules:
   - numpy (fast array processing)
   - scipy (scientific functions and constants)
   - matplotlib (plotting)
   - parallelpython (parallel processing)

   One way to obtain the correct running environment is to use Anaconda:
   - Follow ipython.org/install instructions to install Anaconda.
   - Use Anaconda to install IPython, numpy, scipy and matplotlib.
   - parallelpython may not be known to Anaconda. If so it can be installed following the instructions at http:www.parallelpython.com.

   If the installation has worked then you will be able to type 'ipython', then:

   ```
   In[1]: import numpy
   In[2]: import scipy
   In[3]: import matplotlib
   In[4]: import pp
   ```

   with no errors.

2. Download the PyFIInS code.
   - The code is in a public GitHub repository at https://github.com/handofkwll/fisica. Point your browser at this webpage and click on the 'Download ZIP' button near the top right.
   - Unpack the code into a fresh directory.
   - Create a different 'working' directory to run the simulator from and copy the .xlsx files from <dir>/fisica/excel into it – that is where the program will look for them. In addition, 'gunzip' <dir>/fisica/data/ringcube-25-50.fits.gz into the working directory.

3. Start up ipython, then:

   ```
   In[1]: import sys
   In[2]: sys.path.append('<dir>/fisica-master')
   In[3]: import pyfiins
   In[4]: f=pyfiins.PyFIInS(instrument_spreadsheet='strawman.xlsx',
   sky_file='ringcube-25-50.fits',
   beam_model_dir='<dir>/fisica/beam_models/Smooth_Walled_Horn/Band 4
   GRASP')
   In[5]: f.simulate()
   ```

Once started, the program should run for roughly 4 hours eventually finishing with:

```
…
rendering observe
rendering writefits
```

If it has worked you will see a new directory in the working directory with name of form ‘fisica-sim-yyyymmddThhmmss’, containing the browseable weblog of the simulation run, and a file with name of form ‘yyyymmddThhmmss.fits’ that contains a FITS table with the simulated interferograms.

To explore the weblog, point your browser at its root directory, scroll down to the bottom of the listing and click on ‘uvmapgenerator.html’.

To reduce the interferograms start ipython as before and, instead of importing pyfiins, type:

```
In[4]: import pydataprocessing
In[5]: d=pydataprocessing.PyDataProcessing(‘<fits file>’)
In[6]: d.reduce()
```

After a further hour or so, this program will finish with:

```
...
rendering reduceinterferogram
rendering dirtyimage
```

A weblog describing the reduction results will appear in ‘fisica-dp-yyyymmddThhmmss’.

5. A PyFIInS Cookbook

This section describes how to use the FISICA simulator, and gives an example.

5.1 Process Overview

The simulation process can be broken into 3 stages:

1. Generate a datacube of the target to be observed. The datacube is stored in FITS format, whose details are described in a document CUBEFORMAT.md stored on GitHub with the code. It is best viewed on that site with a browser. The datacube can be derived from existing observational data of an object, or built from scratch using the 'cube toolkit' described below.
2. Specify the interferometer configuration during the observation in the 'instrument' Excel file. This file comprises several sheets, each describing a particular aspect of the observation. Most users will just want to set the waveband of the simulation and the *u*-*v* pattern to be used.
3. Run the simulator, giving the 'instrument' file and the target datacube as inputs. The simulator will output an html 'weblog' that can be examined with a browser, and a FITS file containing the generated interferograms.
4. Run PyDataProcessing with the 'interferograms' file as input. This will push the data through a data reduction recipe and generate a 'dirty' cube. In an ideal world this would equal the target cube convolved with the dirty beam of the interferometer, in reality noise and systematic errors will further affect the result.

5.2 Target Generation – the 'cube toolkit'

The simulator can read any target cube whose structure conforms to CUBEFORMAT.md. The 'makecube' module in the code collection has been written to make it easier to construct such cubes. Cube construction has four parts:

1. Generate a 2-d image of the target. This can be done by reading a FITS file with the observed image of a target and interpolating it to the required size and spatial resolution (*makecube.MakeImage*). Alternatively, the image can be constructed from scratch (*makecube.MakeModelThinRing*, *MakeModelThickRing* or *MakeModelComet*). The scratch images are derived from simple models but have the advantage that they are noiseless.
2. Generate the target spectrum (*makecube.MakeSpectrum*). This method constructs a grey-body spectrum of given temperature, emissivity index and peak flux. Some spectral features can be added to the grey-body, currently available are:
    a. 'forsterite'- a complex feature near 70μm.
    b. 'protostar' - a set of lines: [OI] 63μm, several water and CO lines between 66 and 325μm.
    c. 'comet' - water lines at 66 and 300μm.
3. Combine the spatial image and the spectrum to produce a cube, write the result to a FITS file (*makecube.MakeCube*). The integrated flux of the cube equals the input spectrum.
4. Combine cubes. It is possible to build complex targets by adding together cubes (*makecube.AddCubes*). No check is made that the cubes share the same dimensions, if they do not then a Python error will occur when the add is attempted. The output cube inherits its header keywords from the first FITS cube in the 'in_cubes' list.

5.3 The 'Instrument' File

As said above, the configuration and timeline of the interferometer during the simulated observation are specified via an Excel spreadsheet. The file contains several sheets, each associated with one aspect of the instrument.

The file 'fisica/excel/strawman.xlsx' contains parameters that describe the FISICA strawman instrument. It should be copied to the working directory and modified there.
Most users will only need to edit 2 parts of the strawman file. First, edit FTSpectrograph/Selection to place a 1 in the Band to be simulated, 0 for the others. Second, edit Interferometer/Select in the same way to specify which uv pattern the instrument is to use in the observation.

5.4 Example. A simulated observation of pre-stellar cores

The construction of the target cube involves the making of 4 separate 'cores', using *makecube.MakeModelThickRing* 4 times with different input parameters, then combining the 4 individual cubes with *makecube.AddCubes*. Thus:

```
In [1]: import makecube
In [2]: image1 = makecube.makeModelThickRing(rinner=0.4, router=1.0,
  tilt=20.0, rot=-90.0, xcentre=-0.7, ycentre=0.2, max_baseline=80.0,
  wn_min=100.0, wn_max=200.0)
In [3]: spectrum1 = makecube.makeSpectrum(temperature=37, beta=-1.3,
wn_min=100.0,                    wn_max=200.0, wn_step=0.5, peak_flux=10.0,
spectral_features=['forsterite'])
In [4]: makecube.makeCube(image=image1, spectrum=spectrum1,
  cubename='cube1.fits')
```

```
... repeat this process n times to produce n 'cores' in n cubes, each
... with different parameters. This will give you n FITS files: 'cube1.fits'
... to 'cuben.fits'.
In [20]: makecube.addCubes(in_cubes=['cube1.fits', 'cube2.fits',
..., 'cuben.fits'], cubename='total_cube.fits')
```

Specify the observation details by copying 'strawman.xlsx' to your working directory and modifying it as follows:

- In FTSpectrograph select Band 2 (100-200cm-1).
- In Interferometer select 'Const L spiral' as the uv pattern.

Run the simulator:

```
In [1]: import pyfiins
In [2]: f = pyfiins.PyFIInS(instrument_spreadsheet='strawman_copy.xlsx',
sky_file='total_cube.fits', beam_model_dir='/Users/jfl/test2beam/fisica-
pbmodels/beam_models/Smooth_Walled_Horn/Band 4 GRASP')
In [3]: f.simulate()
```

This will produce a weblog html tree rooted at 'fisica-sim-yyyymmddThhmmss' dated at the time of the run, and a matching FITS file with the simulated interferograms 'yyyymmddThhmmss.fits'
Reduce the interferograms:

```
In [3]: import pydataprocessing
In [4]: d=pydataprocessing.PyDataProcessing('yyyymmddThhmmss.fits',
data_quality='pure')
In [5]: d.reduce()
```

The 'data_quality' parameter in the PyDataProcessing constructor specifies whether the 'pure' or 'noisy' interferograms are to be used. The pure interferograms are the direct output from the interferometric simulation, the noisy ones have detector noise, background noise, cosmic rays and 1/f noise added.