James Webb Space Telescope (JWST)
Mid-Infrared Instrument (MIRI)
Flight Model

MIRISIM Imsim Report
Prepared by: René Gastaud, Christophe Cossou, Alain Coulais    Date: March 20, 2018

Approved by:    Date:

The presented document is Proprietary information of the MIRI European Consortium. This document shall be used and disclosed by the receiving Party and its related entities (e.g. contractors and subcontractors) only for the purposes of fulfilling the receiving Party’s responsibilities under the JWST Project and that the identified and marked technical data shall not be disclosed or retransferred to any other entity without prior written permission of the document preparer.
## DOCUMENT CHANGE DETAILS

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Page</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23/12/16</td>
<td>All</td>
<td>First Issue</td>
</tr>
<tr>
<td>2</td>
<td>05/05/17</td>
<td>All</td>
<td>Correcting (V2, V3) into (Xan, Yan)</td>
</tr>
<tr>
<td>3</td>
<td>15/05/17</td>
<td>All</td>
<td>Adding ImSim interface and distortion image</td>
</tr>
</tbody>
</table>

This document contains proprietary information as stated on the cover page.
# Table Of Contents

1 Introduction 5

2 Requirements 6

3 ImSim Interface 6
   3.1 run_imsim ................................. 7
   3.2 get_ima_v2v3_ref ......................... 7
   3.3 get_lrs_v2v3_ref ......................... 8
   3.4 get_lrs_ref_color ....................... 8
   3.5 get_ima_subarray_to_skysim_transform 9
   3.6 get_ima_subarray_to_v2v3_transform 10
   3.7 MirimPCE class ......................... 11
   3.8 jansky2photon .......................... 11

4 Inputs 12
   4.1 PCE ..................................... 12
   4.2 PSF ..................................... 12

5 Distortion CDP 13

6 Algorithm 13
   6.1 About Flux calculation .................. 15
   6.2 Point source ............................ 15
   6.3 Extended Sources ....................... 17
   6.4 Coordinate Transforms .................. 17
   6.5 How Distortion is applied .............. 17
   6.6 Architecture ........................... 17
   6.7 Dependencies ........................... 19

7 Output 19

8 Tests 19
   8.1 Input tests ............................. 19
   8.2 Intermediate steps ..................... 20
   8.3 Output tests ............................ 20
   8.4 Coordinate transforms .................. 20

A Acronyms and Abbreviations 22

B Conversion of photon/s/m2/microns into Jansky 22

C Comparison of CDP filters with manufacturer specifications 23
   C.1 Quantum ................................. 23

List of Figures

1 MIRIM optical path ........................... 5
2 Flow charts for MIRISim and ImSim ............ 5
3 PSF and PSF-OOF representation .................. 6
4 All measured PSF position of a given filter 13
5 Output of PSF functions in ImSim ............... 13
6 Coordinate Transforms ........................ 14
7 Illustration of the image distortion ........... 14
8 ImSim Algorithm ............................. 15
9 Error on PSF for a given filter ................ 16
10 PSF convolution, theory vs MIRISim .......... 17

This document contains proprietary information as stated on the cover page  iii
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>the 9 PSFs for F560W full imager</td>
<td>18</td>
</tr>
<tr>
<td>12</td>
<td>How CDPs are read</td>
<td>18</td>
</tr>
<tr>
<td>13</td>
<td>Example of output by ImSim, <strong>illumination_model</strong>, for a default simulation</td>
<td>19</td>
</tr>
<tr>
<td>14</td>
<td>Coordinate Systems</td>
<td>21</td>
</tr>
<tr>
<td>15</td>
<td>CDP filter transmission comparison for the coronographs</td>
<td>24</td>
</tr>
<tr>
<td>16</td>
<td>CDP filter transmission comparison for the wide filters</td>
<td>25</td>
</tr>
<tr>
<td>17</td>
<td>CDP and manufacturer data comparison, zoom-in for filter 2550W</td>
<td>25</td>
</tr>
<tr>
<td>18</td>
<td>Quantum efficiency</td>
<td>26</td>
</tr>
</tbody>
</table>
1 Introduction

MIRI is a mid-infrared instrument for the James Web Space Telescope (JWST) [4]. The instrument contains an imager, which can also be used as a coronagraph or low resolution spectrograph (LRS), and a medium resolution spectrograph (MRS). The MIRI imager contains a 1024x1024 pixel detector chips called Sensor Chip Assembly (SCA). The figure Figure 1 shows a schematic view of the light path, from the telescope aperture to the MIRI Imager detector.

![Figure 1: MIRIM optical path](image)

As shown in Figure 2a, ImSim takes as input the Sky generated by Skysim and the parameters of both the observation and the instrument. It produces an Illumination model (image on the detector in electron/s), taken as input by SCASim. The ImSim module is in charge of the imager specifically. It does not comprise the Low Resolution Spectrometer (LRS) and the coronographs. It simulates all the optical defects: smearing (which can depend upon position and wavelength), distortion, etc... Moreover, we also apply the PCE (Photo Conversion Efficiency, product of the filter transmission and the Quantum Efficiency), retrieved from a CDP (Calibration Data Product). All the others physical characteristics of the detector are processed by SCASim. No noise are simulated.

![Figure 2: Flow charts of MIRISim as a whole and ImSim specifically. It shows input and outputs on top of sub-modules relations.](image)
2 Requirements

ImSim must read a list of sources and produce an illumination model.

The requirements for ImSim are:

- The input are (see Section 4 and Figure 2b):  
  - a list of sources with various physical properties, SEDs, velocity maps, and position in the sky (scene.ini)  
  - instrument configuration, which filter, readout mode, ... (ima_simulation.ini)  
  - simulator properties (activate or not various types of effects, specific CDP versions) and intrinsic properties of the JWST telescope (telescope surface area)

- Apply all the necessary optical transformation mimicking the Mirim optical path

- The output is an illumination model in electron/s integrated on the selected filter. There is no wavelength dependency at this level.

In addition to that, we must pay attention to the following facts:

- Sources outside the field of view but whose PSFs intersect the field of view should be processed (see Figure 3)

- Calibration Data Products should be used as much as possible. This last requirement has an impact on the algorithm (There is no monochromatic PSF).

- PSFs are dependant on the position of the source on the detector (Figure 4)

- We can’t access the filters transmission separately. We only have a CDP for the corresponding Photo-Conversion Efficiency (PCE), product of the transmission by the quantum efficiency

- A simulation should run on a personal computer in a reasonable amount of time.

![Figure 3: Representation of 4 point sources, 2 inside the field of view, 2 others that are not, but distant from less than half of their respective PSF width. All four sources are to be accounted for when representing what the detector will see.](image)

3 ImSim Interface

Some parts of insim are called outside insim, so their interface should not change. These are:

- run_imsim

- In ima.py
  - ima.get_ima_v2v3_ref
  - ima.get_lrs_v2v3_ref
3 IMSIM INTERFACE

3.1 run_imsim

```python
def run_imsim(scene, cfgpath, filterName, pointing, rel_obsdate, simulator_config, verbose=False):
    """
    Interface function between ObsSim and ImSim. For specified parameters,
    runs an ImSim simulation and returns the resulting detector illumination
    model produced by ImSim.
    """

    Arguments
    --------
    scene: mirsim.skysim.scenes.SkyScene object
        The SkySim object that describes the scene.
    cfgpath: string
        Represents which optical path (Imager, LRS, or MRS) is the "primary
        optical path" of interest. Here only IMA is authorised, with IMA Paths:
        IMA_FULL, IMA_CIRR1065, IMA_CIRR1140,
        IMA_CIRR1550, IMA_CIRR1575, IMA BRIGHTSKY, IMA_SUB256,
        IMA_SUB128, IMA_SUB64
    filterName: string
        Imager filter to use, one of the followings: F560W, F770W, F1000W, F1130W,
    pointing: mirsim.obsim.pointing.Pointing object
        Pointing object containing representation of pointing for
        current dither position.
    rel_obsdate: float
        Relative fraction of mission lifetime when observation takes
        place (0: launch, 1: nominal end-of-mission), this is used to
        calculate the telescope efficiency.
    simulator_config: mirsim.config_parser.SimulatorConfig object
        The configuration object that describes the simulator.

    Keyword arguments
    ------------------
    verbose: boolean, optional, default: False
        Set to True to make ImSim more verbose in logging.

    Return
    ------
    illum_model: mir.mirtools.dapproduct.sim.MirIlluminationModel,
        illumination model contains one image in electron/second
    """

3.2 get_ima_v2v3_ref

```python
def get_ima_v2v3_ref(cfgpath, filterName, simulator_config=None):
    """
    Returns the reference (v2, v3) coordinate for specified imager config path,
    based on the reference col, row coordinate.

    Beware: check the compatibility with Science Instrument Aperture File (SIAF),
    and with the tool APT. Since this is hard-coded in ima.py
    if changes occur, they also need to be applied here.

    Parameters
    ----------
    cfgpath: str
        Represents which optical path (Imager or MRS) is the "primary optical path" of
        interest, relative to which the dither pattern is assumed to be executed.
    """
IMA Paths: IMA_FULL, IMA_C5R1065, IMA_C5R11140, IMA_C5R1155, IMA_C920477, IMA_BRIGHTSKY, IMA_SUB256, IMA_SUB128, IMA_SUB64

filterName: str

simulator_config: mirisim.config_parser.SimulatorConfig
The configuration object that describes the simulator.

Return
-------
(v2, v3) coordinates (arcsecond)

Example
--------
get_lrs_v2v3_ref("IMA_FULL", filterName='F770W')
(453.91033530076, 373.915302562086)

See also
--------
get_lrs_v2v3_ref and bug 63

3.3 get_lrs_v2v3_ref

```python
def get_lrs_v2v3_ref(cfgpath, simulator_config=None):
    """
    Returns the reference (v2, v3) coordinate for specified LRS config path,
    based on the reference coordinate (column, row) on detector.

    Parameters
    ----------
    cfgpath: string
        Represents which optical path is the "primary optical path" of interest. For LRSSim, this is expected to be either "LRS_SLIT" or "LRS_SLITLESS".
    simulator_config: mirisim.config_parser.SimulatorConfig
        The configuration object that describes the simulator.

    Return
    -------
    (v2, v3) reference coordinates arcseconds
    
    Example
    --------
    get_lrs_v2v3_ref("LRS_SLITLESS")
    (-378.8636122348897, -351.831636831407)
    get_lrs_v2v3_ref("LRS_SLIT")
    (-414.5212188368092, -400.58950753426112)
    
    See also
    --------
    get_ima_v2v3_ref and bug 63
    """
```

3.4 get_lrs_ref_colrow

```python
def get_lrs_ref_colrow(cfgpath, simulator_config=None):
    """
    returns the reference coordinate (column, row) on detector.

    Parameters
    ----------
```

This document contains proprietary information as stated on the cover page
**3 IMSIM INTERFACE**

**MIRISIM Imsim Report**

```python
cfgpath: str
    Represents which optical path is the "primary optical path" of interest. For LRSim, this is expected to be either "LRS_SLIT" or "LRS_SLITLESS".
simulator_config: mirsim.config.parser.SimulatorConfig object
    The configuration object that describes the simulator. It contains the version of the cdp file, e.g. 7B.08.00 as in MIRI_FM_MIRIMAGE_DISTORTION_7B.08.00.fits
```

Return
------

```python
(c0l, row) Reference coordinates in pixels
```

Example
------

```python
from mirsim.imsim.ima import get_lrs_ref_colrow
get_lrs_ref_colrow('LRS_SLIT')
(325.6, 300.2)
get_lrs_ref_colrow('LRS_SLITLESS')
(34.763)
```

See also
--------

```python
get_ima_colrow_ref and bug 272
```

3.5 `get_ima_subarray_to_skysim_transform`

```python
def get_ima_subarray_to_skysim_transform(cfgpath, filterName, v2_ref, v3_ref, v2_eff, v3_eff, 
                                          simulator_config=None):
    """
    Returns a gwcs.wcs.WCS object that represents the following imager coordinate frames and corresponding transformations:
    Frames of reference:
    (colsubref,rowsubref) [pixels] = selected subarray frame relative to reference point
    (col, row) [pixels] = imager detector pixel coordinates for selected filter
    (col770, row770) [pixels] = imager detector pixel coordinates for F770W
    (xout,yout) [mm] = MIRI imager detector focal plane (DFP)
    (xin,yin) [mm] = MIRI entrance focal plane (EFP)
    (V2, V3) [arcmin] = JWST Focal Plane (XAN, YAN)
    (ra, dec) [arcsec] = SkySim Scene coordinate frame
    """

Arguments
---------

cfgpath: str
    Represents which optical path (Imager or MRS) is the "primary optical path" of interest, relative to which the dither pattern is assumed to be executed.

filterName: str
    Imager filter to use

v2_ref : float, arcsec
    Represents the V2 coordinate (arcsec) of the reference point in (V2,V3) on which the SkySim frame zero-point is assumed to be centered. Depends on cfg and filtername

v3_ref : float, arcsec
    Represents the V3 coordinate (arcsec) of the reference point in (V2,V3) on
```

This document contains proprietary information as stated on the cover page 9
which
   the SkySim frame zero-point is assumed to be centered. Depends on cfg and
   filtername

v2_off : float
   Represents the offset in V2 (arcsec) w.r.t. v2_ref (for the current dither
   pointing).

v3_off : float
   Represents the offset in V3 (arcsec) w.r.t. v3_ref (for the current dither
   pointing).

pa : float
   Represents the position angle (degree).

Keyword arguments
-----------------------
simulator_config: mirsim.config.parser.SimulatorConfig
   The configuration object that describes the simulator.

Returns:
--------
A gwcs.wcs.WCS object that can be queried to return the transformation between
any of
the defined coordinate frames, see

Example
--------
all_transforms = get_ima_subarray_to_skysim_transform('IMA_BRIGHTSKY', 'F560W',
0.0,0.0,0.0)
print(all_transforms.available_frames)
radec_to_colrow = all_transforms.get_transform('skysim_radec','
f770W.fullarray.colrow')

***

3.6 get_ima_subarray_to_v2v3_transform

def get_ima_subarray_to_v2v3_transform(cfgpath, filterName, simulator_config=None):
   """
   Returns a gwcs.wcs.WCS object that represents the following imager coordinate
   frames and corresponding transformations:

   Frames of reference:
   (colssubref,rowsubref) [pixels] = selected subarray frame relative to
   reference point
   (colssub,rowsub) [pixels] = selected subarray frame
   (col,row) [pixels] = imager detector pixel coordinates for selected
   filter
   (col770,row770) [pixels] = imager detector pixel coordinates for F770W
   (xout,yout) [mm] = MIRI imager detector focal plane (DFP)
   (xin,yin) [mm] = MIRI entrance focal plane (EFP)
   (xan,yan) [arcmin] = JWST Focal Plane (XAN, YAN)
   (v2,v3) [arcsec] = JWST focal plane (v2, v3)
   Arguments
   """
   cfgpath: str
      Represents which optical path (Imager) is the "primary optical path" of
      interest, relative to which the dither pattern is assumed to be executed.
      IMAPaths: IMA_FULL, IMA_COR81065, IMA_COR81140,
      IMA_COR81550, IMA_COR81550, IMA_BRIGHTSKY, IMA_SUB256,
      IMA_SUB128, IMA_SUB64, LAS_SLIT, LAS_SLITLESS
   filterName: str
      Imager filter to use, one of the followings: F660W, F770W, F1000W, F1300W,
      F2300C
   simulator_config: mirsim.config.parser.SimulatorConfig

This document contains proprietary information as stated on the cover page
The configuration object that describes the simulator.
It contains the version of the cdp file, e.g., 7B.03.00
as in MIRI_FM_MIRIMAGE_DISTORTION_7B.03.00.fits

Returns
-------
A gwcs.wcs.WCS object that can be queried to return the transformation
between any of
the defined coordinate frames.

Example
--------
transforms= get_ima_subarray_to_v2v3_transform('IMA_BRIGHTSKY', 'F560W')
print(transforms.available_frames)
colrow_to_v2v3 = all_transforms.get_transform('fullarray_colrow', 'focal_v2v3')
colrow_to_v2v3(33, 45)
( -384.9986141606684, -431.3949350563111)

3.7 MirimPCE class

class MirimPCE(object):
    ""
    Add to the cdp product the possibility to interpolate at a given wavelength
    Parameters
    ""
    filterName: str
        Image filter to use
        One of the following: F560W, F770W, F1000W, F1130W, F1280W,
        F1500W, F1800W, F2100W, F2550W, F1065C, F1140C, F1550C,
        F2300C.

    subarray: str
        Subarray to use
        One of the following: FULL, MASK1065, MASK1140, MASK1550,
        MASKLYOT, BRIGHTSKY, SUEZ66, SUB128, SUB64, SLITLESSPRISM

    simulator_config: mirisim.config_parser.SimulatorConfig
        ConfigObj of a simulator.ini corresponding object
    ""

inside MirimPCE
    def getPCE(self, wavelength):
        ""
        Interpolate the pce for the given wavelength
        extrapolated values put to zero to avoid negative value for F2550W
        Parameters
        ""
        wavelength: micron

3.8 jansky2photon

def jansky2photon(flux, wave, invert=False, verbose=False):
    ""
    Convert flux from Jansky to photon/s/m\(^2\)/micron and viceversa
    Parameters
    ""
    flux: numpy.array([float])
        Flux in Jansky
    wave: numpy.array([float])
4 Inputs of Image Simulator

4.1 Photon Conversion Efficiencies

As stated in the technical note MIRI-TN-00072_PCEs_Iss5.pdf (Title: “JWST MIRI Photon Conversion Efficiencies”), the Photon Conversion Efficiency (PCE) is defined (see paragraph 2, Description of File Contents) as:

- The detector Quantum Efficiency (Curves provided by Ressler, private communication) (wavelength dependent).
- The filter transmission (wavelength dependent)
- Mirror reflectivities (0.98 per surface, based on measurements)
- A wavelength independent transmission factor of 0.8 which represents the Beginning of Life (BOL) contamination.

We define the Path Transmission Constant PTC as:

\[ PTC = (\text{Mirror reflectivity})^5 \times \text{BOL} \]
\[ PTC = 0.98^5 \times 0.8 = 0.723 \] (1)

PCE is then defined:

\[ \text{PCE}(\lambda) = PTC \times T_{\text{filter}}(\lambda) \times \text{QE}(\lambda) \] (2)

where \( T_{\text{filter}} \) is the filter transmission and \( \text{QE} \) the quantum efficiency.

The efficiencies DO NOT include a factor for the End Of Life (EOL) contamination. This factor IS included (set equal to 0.8) in the sensitivity model (RD-2).

The nominal transmission of the JWST OTE, which is assumed to be 0.88 at start of mission in RD-2, is NOT included in the PCE curves.

4.2 About Point Spread Functions

The simulator uses the PSF of the Calibration Data Products. These PSFs are measured, or in some cases modeled. The reference is [20]. For each filter we have the PSF at 9 different positions. These PSFs are not monochromatic, but integrated over the filter wavelength range for a given spectrum (grey body). Hence, no matter the spectrum, the rendering of a point source through a given filter will remain the same. The CDPs are oversampled compared to the actual sensor chip array. The pixel size depend of the filter but is given in the CDP metadata in microns. The actual size of the pitch of the SCA is 25\( \mu \)m [4]. The PSFs are oversampled, usually by a factor of 4, except for the largest wavelength filter (F2550W ; factor of 2) see the Figure 4.

To use the cdp another class has been created, MirimPSF, which reads the cdp and gives two methods to get the PSF at a given position: getCenteredPSF() and getPSF(). In both cases, we get the PSF at
the nearest position from the source location (since we have 9 PSFs for 9 different locations spread out throughout the detector. We normalize the PSFs to 1 (sum of all pixels equal 1) and rebin to the detector pixel size.

- `getCen teredPSF` prepare the PSF for extended sources. It’s meant to be used for convolution with the extended source (see left panel of figure Figure 5).

- `getPSF` prepare the PSF for a point source: There’s no need to convolve. We only shift the PSF to the source position in an oversampled array and shift it to the source position. Rebin to detector pixel size is done afterwards to ensure maximum accuracy. See right panel of figure Figure 5. It is used to simulate a point source at a given position in the detector.

![Figure 4: Left: position of the 9 PSFs. Right: PSFs resolution compared to the detector pixel size.](image)

![Figure 5: Left: getCenteredPSF output (PSF for extended source, to be convolved). Right: getPSF output (PSF shifted to point source location).](image)

5 **Distortion CDP**

This CDP contains informations to transform spacecraft axes coordinates (V2, V3) into detector coordinates (col, row). This is done in several atomic steps using intermediate coordinates systems. Figure 6 shows the complete transformation chain summarizing the process described in the Technical Note “JWST to Imager focal plane coordinate transform” [8].

6 **Algorithm**

The general algorithm is illustrated by Figure 8. For a given simulation, different pointings can be needed (e.g. for dithering). This does not appear on the workflow of ImSim simply because, for a given call
Figure 6: Coordinate Transforms

Figure 7: Illustration of the image distortion.
Figure 8: Image Simulator Algorithm. 3 separate algorithms are used depending on the type of sources (Point, Extended, Background) because the flux is different in nature, 1D for Point, 3D for Extended and Background.

The sources are split in three different categories: point sources, extended sources and constant sources.

- For point sources we substitute by the nearest PSF in position with given flux (see left panel of Figure 4)
- For extended sources we convolve by the nearest PSF.
- For Background (spatially constant source), we do not convolve by PSF, there’s no need to.

6.1 About Flux calculation

Another possibility would have been to directly sum all sources (CompositeSkyScene) and convolve the result by the PSF. To do that, a method getext(ra, dec, λ) is available in SkySim that return the flux at the exact position specified (without any PSF yet since that’s what we want to do, a point source is then a Dirac function). However, if a point source happen to be slightly off the sampling grid, the method will miss it and we’ll loose all the flux related to that point source. That’s why we split the sources into 3 different categories to compute the spreadout flux for each source consistently.

The same problem occurs when we want to integrate the SED over the filter wavelength bandwidth. While there’s no particular problems to deal with Black Body spectrum, the LinesSed class of SED poses problems by its discontinuous nature. We can have very thin intensity peaks outside the wavelength grid, thus missing part of the flux. This was partly solved by choosing a very small wavelength step (10 Å).

This is clearly not ideal. A better solution would be to directly use all parameters for each spectral line and integrate the flux following an analytical solution for the assumed line profile.

6.2 Point source

Having a special process for point sources also shall enables us to take into account intra pixel sensitivity variations. So far, there is no measurement of intra pixel sensitivity variations, but they existed in Spitzer Space Telescope Infrared Array Camera (IRAC) (see [13]).
Most of the time, the source position isn’t aligned with a pixel. For precise shift of the PSF to align it perfectly with the source position, we use the method `scipy.ndimage.interpolation.shift` that shifts the PSF in the Fourier space. Indeed, a translation in the normal space is equivalent to a phase-shift in the Fourier space. This re-gridding is flux conservative and avoid negative values.

\[
f(x - x_0, y - y_0) \Leftrightarrow F(u, v) e^{-2\pi i \left( \frac{ux_0 + vy_0}{\lambda} \right)}
\]

It’s worth noting that the PSFs are integrated on the whole range of the filter. To measure this PSF, a black body at 300K was used.

\[
\text{PSF\_filter}(x, y) = \frac{\int_{\lambda_1}^{\lambda_2} f(\lambda) \ast \text{PSF}(x, y, \lambda) \, d\lambda}{\int_{\lambda_1}^{\lambda_2} f(\lambda) \, d\lambda}
\]

where \( f \) is the source’s flux.

Since the PSF of a given filter is averaged on the whole range, and was measured for a particular spectrum, we understand from Eq. 4 that this has direct consequences on the PSF we use. No matter the shape of the SED used, as soon as the integrated flux is the same, the image will be identical.

We will miss the fact that one should normally integrate:

\[
\text{image}(x, y) = \int_{\lambda_1}^{\lambda_2} f(\lambda) \ast \text{PSF}(x, y, \lambda) \, d\lambda
\]

But in our case, we do:

\[
\text{image}(x, y) = \text{PSF\_filter}(x, y) \ast \int_{\lambda_1}^{\lambda_2} f(\lambda) \, d\lambda
\]

(since the PSF is already averaged on the filter range and thus wavelength independent).

![Figure 9](image.png)

Figure 9: We emphasize here the error we make by using one single PSF for the whole range [9; 11] \( \mu m \). First two panels show individual PSF. The third one show the differences between the two.

The rendering of a point source should be different if its SED is different. If we take for instance the F1000W filter (centered at 10\( \mu m \) but sensitive to [9; 11] \( \mu m \). A source with a peak at 9\( \mu m \) should result in a smaller image than the same point source but with an SED defined as a peak at 11 \( \mu m \). As a rule of thumb, the PSF is about \( \sim 20\% \) smaller at 9\( \mu m \) compared to 11 \( \mu m \) (see Figure 9).

The advantage to use measured PSF are:

- same than the pipeline
- have all the defects (spikes, glow)

However, in our case, we know exactly the SED for each source. Not using this information in our computation results in shape errors up to \( \pm 10\% \) error on the size (particularly critical for gravitational lensing of galaxies).

The flux is thought to be conserved in the process. But differences in shape can result in small measurement differences due to background substraction.
Figure 10: Convolution by PSFs. When integrating the source flux over the filter range, one should also vary the PSF. In the case of MIRISim, one averaged PSF is used for each filter.

The only hard coded parameter is the pixel pitch, distance between 2 pixels of the Sensor Chip Array: 25 microns. The value is found in the introduction of PASP Paper III, [4] and is not in any CDP.

For the point sources, the next step is to convert the flux from Jansky to photon/m²/s/microns, before multiplying it with the chosen PSF (for the filter and closest position in the sky (see Figure 4), PSF or out-of-field PSF if the source is positioned on the CCD or slightly outside (see Figure 3)).

\[ i = \Omega_{\text{pix}} \tau_{\text{tel}} \tau_{\text{EOL}} \int_{\Delta \lambda} P(\lambda) \tau(\lambda) \]  

(7)

### 6.3 Extended Sources

The PSF is assumed to be constant for the whole source area. If the source area is comparable or greater than the sampling step of the 9 PSFs (see Figure 4), this assumption is no longer valid. However the error should be of a few percents at most.

A better solution would be to use a variable PSF which can be obtained by warping the central PSF or split the integration into zones assigned for each zone-PSF with a better sampled PSF.

### 6.4 Coordinate Transforms

The coordinate transforms are based upon the technical report from Alistair Glass [8] and [22] (see [Section 5]).

We use as much as possible existing models in the package astropy.gwcs: the model Polynomial2D, the model AffineTransformation2D...

### 6.5 How Distortion is Applied

Distortion changes the coordinates (see above) but also the objects shape. For point source, the change of shape is taken into account in the CDP: we have 9 PSFs for 9 different locations (see Figure 4).

Figure 11 shows the 9 PSFs for filter F560W [22], each PSF has been put in a place which correspond to its real place in the imager. In the center there is the average of the 9 PSFs, and in each corner the difference between the average and the PSF of this location.

### 6.6 Architecture of the code

For historical reasons, a class has been created for each CDP (they did not exist at those time) These classes have been kept and updated to read the CDP instead of a simple fits file. They are now just a wrapper of the CDP class with some extra tools embedded in it.

For example, in the PCE class we provide a method which computes the pce at any given wavelength by interpolation. For PSF, we normalise, find the zoom, find the center of the PSF, plunge it into a bigger image (out of the field), shift the PSF, and then rebin (unzoom).

Figure 12 represent what CDP are needed by ImSim and how we access them. Changes in CDP interface in MIRI can be applied smoothly since they are located in specific functions.
Figure 11: the 9 PSFs for F560W full imager

Figure 12: How CDPs are read
6.7 Dependancies

Straightforward dependancies are summarized in Figure 2b.
We depend upon the data model of the JWST software. This data model is used in the pipeline, and is used for the CDP and the output (illumination model).

The coordinates transforms depends upon the wcs class and the gwcs of the astropy package.
Sources should have a field Center, a getext() method, and point sources should have a SED field.

7 Output

The output is an illumination_model (see Figure 13) as described in (Chapter 5, [1]).
There is only one image for a given pointing (each call of ImSim have a single pointing). This image corresponds to the spectral flux of the whole input, integrated on the specified filter. Flux is then in electron/s/pixel. Metadata for the intensity unit (illum_model.meta.intensity.units) in the output object was set accordingly. Central wavelength of the selected filter is also transmitted to SCASim.

Figure 13: Exemple of output by ImSim, illumination_model, for a default simulation.

8 Tests

8.1 Input tests

• check_footprint.py Compute and display a footprint from the PixelFlat
• show_coordinate_systems.py (see Figure 14)
• show_distortion.py
• show_filters.py Check that filters are identical to when CDP did not exist yet.
• show_pce.py
• show_PSF_distortion.py
• launch_mirim_distortion.py
• mirim_gauss2dfit.py Check the size of a PSF
• launch_mirim_gauss2dfit.py
• check_mirimPsf.py
• read_cdp_area.py
• read_reference.py
• test_background_lib.py

8.2 Intermediate steps
• script_ima.py
• script_xysub_to_xy.py
• test_jansky2photon.py
• test_imager_transform.py

8.3 Output tests
• check_flux.py
• check_ima.py
• launch_get_extended.py Test the get_extended method (method for extended source processing of ImSim)
• launch_get_points_from_scene.py Test the get_points_from_scene method (for point source processing)
• launch_mirimImager.py
• launch_mirisim_with_scene.py
• launch_one_star_background.py
• make_dummy_pointing.py
• make_scene_3stars.py
• make_scene_galaxy.py
• test_mirimImager_background.py
• test_out_of_field.py
• script_MirimImager.py
• script_simple_test.py
• test_mirimImager.py
• test_run_imsim.py

8.4 Coordinate transforms
See Figure 14 for a comparison of what we expect and the position of the MIRI imager in all intermediate coordinate systems.
Figure 14: The different coordinate systems that describe the intermediate systems throughout the optical paths are represented here. Top left: Reference figure from [[23], Fig. 3]. (V2,V3) are the spacecraft coordinates. The footprints of the instruments are represented. The other 3 panels represent the imager (LVOT mask and the 3 other corners) through 3 different coordinate systems, (V2, V3), (col, row) and (ra, dec). **Warning:** In all these figures, (V2,V3) actually refers to (Xan, Yan)
### Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEA</td>
<td>Commissariat à l’énergie atomique et aux énergies alternatives</td>
</tr>
<tr>
<td>IAS</td>
<td>Institut d’Astrophysique Spatiale</td>
</tr>
<tr>
<td>ImSim</td>
<td>MIRI Image Simulator</td>
</tr>
<tr>
<td>FITS</td>
<td>Flexible Image Transport System</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HDU</td>
<td>Header Data Unit</td>
</tr>
<tr>
<td>IDL</td>
<td>Interactive Data language</td>
</tr>
<tr>
<td>IFU</td>
<td>Integrated Field Unit</td>
</tr>
<tr>
<td>JPL</td>
<td>Jet Propulsion Laboratory</td>
</tr>
<tr>
<td>JWST</td>
<td>James Webb Space Telescope</td>
</tr>
<tr>
<td>LRS</td>
<td>Low Resolution Spectrograph</td>
</tr>
<tr>
<td>MIRI</td>
<td>Mid Infra-Red Instrument</td>
</tr>
<tr>
<td>MRS</td>
<td>Medium Resolution Spectrograph</td>
</tr>
<tr>
<td>OTE</td>
<td>Optical Telescope Element</td>
</tr>
<tr>
<td>PSF</td>
<td>Point Spread Function</td>
</tr>
<tr>
<td>PCE</td>
<td>Photo Conversion Efficiency</td>
</tr>
<tr>
<td>SCA</td>
<td>Sensor Chip Assembly</td>
</tr>
<tr>
<td>SCASim</td>
<td>Sensor Chip Assembly Simulator</td>
</tr>
<tr>
<td>STScI</td>
<td>Space Telescope Science Institute</td>
</tr>
</tbody>
</table>

For other acronyms see [http://deggial.as.arizona.edu/jad/acronyms.dat](http://deggial.as.arizona.edu/jad/acronyms.dat).

### B Conversion of photon/s/m²/microns into Jansky

The trick is to convert in two separate steps. First, we convert from photon to energy. We take for instance the case of a photon of wavelength $\lambda_0 = 10 \times 10^{-6}$ m.

By definition:

$$1 \text{Jy} = 10^{-26} \text{W.m}^{-2}.\text{Hz}^{-1}$$

$$1 \text{ photon/s/m}^2/\text{microns} = \frac{hc}{\lambda_0} \text{W/m}^2/\text{microns}$$

$$= 10^6 \frac{hc}{\lambda_0} \text{W/m}^2/\text{m}$$

Second, we need to convert flux density per wavelength into flux density per frequency. The trick is to realize that “per wavelength” is equivalent to $d\lambda$ because if we do $\int_0^\infty f_\lambda d\lambda$ it’s not “per wavelength” anymore. To calculate the conversion factor, we simply need to change variable in the integration.

We have:

$$\lambda = \frac{c}{\nu}$$

$$d\lambda = -\frac{c}{\nu^2} d\nu = -\frac{\lambda^2}{c} d\nu$$

The sign is then cancelled by inversion of integration boundaries.

$$f_\nu d\nu = f_\lambda d\lambda$$

$$f_\nu d\nu = f_\lambda \frac{\lambda^2}{c} d\nu$$

$$f_\nu = \frac{\lambda^2}{c} f_\lambda$$
The total conversion now becomes:

\[ 10^6 \frac{hc}{\lambda_0} W/m^2/m = 10^6 \frac{hc \lambda_0^2}{c} W/m^2/Hz \]
\[ = 10^6 h \lambda_0 W/m^2/Hz \]
\[ = 10^6 \times 10^{26} h \lambda_0 10^{-26} W/m^2/Hz \]

1 photon/s/m²/microns = 10^{32} h \lambda_0 Jy

with \( \lambda_0 \) given in meters.

The function jansky2photon provide the same conversion. We can conclude that this function works well.


### C Comparison of CDP filters with manufacturer specifications

Before the official calibration distribution, the IDL simulator worked with FITS files coming from different sources: manufacturer, etc...

Since CDP version 6, we can't access separately the filter transmission and quantum efficiency separately. They are combined and provided in the CDP PCE (see PASP Paper IX on sensitivity [6])

- for CDP-6 the transmission is in percentage ([0 – 100]), but previous files had transmission as a coefficient ([0 – 1]).
- the baseline is now at 0.01 (in %), and in some of the old files the transmission minima were negative.
- some shifts are visible, see for example Figure 17. One possible explanation of these shifts will be measurements taken at different temperatures.

#### C.1 About Quantum efficiency

We have now three fits files with different values for the quantum efficiency:

- bug111 detector_mirii_qe_1.fits.gz (IDL Simulator, A. Boccaletti)
- qe_measurementIM SCASIM
- jwst_miri_mirimage_qe.fits (CDP-6) As the figure shows, SCASIM has the same qe than CDP-6, but with a better range in wavelength.
Figure 15: The 4 C filters, in CDP 5 and pre-CDP 6 versions. The vertical blue lines in the plot of the transmission give the range of wavelength of the filter. Outside this domain, filter transmission can be assimilated to zero, so no computation is needed.
Figure 16: CDP and manufacturer data comparison for all wide filters. Both curves completely overlap and no apparent differences in logarithmic scale. The vertical blue lines in the plot of the transmission give the range of wavelength of the filter. Outside this domain, filter transmission can be assimilated to zero, so no computation is needed.

Figure 17: On the left, comparison of filter 2550W transmission (logarithmic scale) between data provided by manufacturer and CDP. Curves completely overlap. But the right hand side show a zoom (linear scale) on the [22; 25] μm where differences are visible.
Figure 18: Quantum efficiency
References


