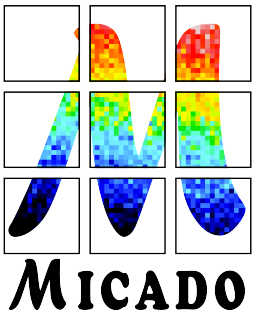


ELT – MICADO

Phase C

SimCADO v2 User Manual

ELT-TRE-MCD-56306-0058

Issue: 1.0

Date: 12. April 2021

Prepared:	K. Leschinski	2021-04-12	
	Name	Date	Signature
Approved:			
	Name	Date	Signature
Released:			
	Name	Date	Signature

Change record

Issue/Rev.	Date	Section/Parag. affected	Reason/Initiation/Documents/Remarks
0.0.1	2020-10-15	All	Layout initialised
1.0.0	2021-03-22	All	Finished initial release version

Contents

Abbreviations

DFS	Data Flow System
ELT	Extremely Large Telescope
ESO	European Southern Observatory
HCI	High Contrast Imaging
IMG	Imaging observing modi
MCAO	Multi-Conjugate Adaptive Optics
PSF	Point Spread Function
SCAO	Single Conjugate Adaptive Optics
SPEC	Spectroscopy observing modi
TBC	To Be Confirmed
TBD	To Be Decided

1 Scope

This document aims to introduce the reader to the ScopeSim instrument simulator and the MICADO instrument packages that have been developed as part of the MICADO instrument simulator work package. ScopeSim is a second generation instrument simulation environment, following on from the original SimCADO package. In this context, the title of the document "SimCADO v2 user manual" refers to the fact that the MICADO instrument simulator is now a combination of the ScopeSim software and the associated MICADO packages. The ScopeSim environment consists of a series of Python packages, with the ScopeSim instrument simulation engine at its core. All packages are available within the standard python ecosystem. The ScopeSim ecosystem maintains a very strict split between code and data. The simulation engine and all associated software packages are completely instrument agnostic. Instrument models can only be initialised by passing an instrument specific data package to the ScopeSim engine.

For MICADO two instrument data package were produced as input for the ScopeSim engine: “MICADO” and “MICADO_Sci”. The original use case for the simulator was as a means of producing raw detector frames to aid in the development of the MICADO data reduction pipeline. Hence the “MICADO“ instrument data package contains all the elements needed to simulate realistic raw detector readout images. The primary audience for the “MICADO“ instrument data package is the MICADO dataflow system work package. Many of the effect included in the “MICADO“ package will be removed by the data reduction pipeline. Simulating and subsequently removing these effects results in a substantial amount of redundant computational overhead for the second group of use cases for the software: science case feasibility studies. In order to enable rapid iteration of simulations for these feasibility studies, a second instrument data package was compiled: “MICADO_Sci“. “MICADO_Sci“ contains only the optical effects that will remain in the images after the data reduction pipeline has successfully processed the images.

This document should be seen as the initial introduction to ScopeSim and the MICADO data packages. It contains the following information:

- Introduction to ScopeSim environment
- Where to find the online documentation
- How to begin simulating MICADO observations with ScopeSim
- How to control simulations
- Building on-sky target object as input for ScopeSim
- Five use case examples for the imaging and spectroscopy modes of MICADO

It should be noted that what is contained in this document is a subset of what is available in the online documentation for ScopeSim. More examples and information are available at https://scopesim.readthedocs.io/.

2 Applicable and Reference Documents

2.1 Applicable Documents

Nr	Doc. Nr	Doc. Title	Issue	Date
AD 1	ELT-ICD-MCD-56306-0058	SimCADO User Manual	1.0	2021-04-12
AD 2	ELT-ICD-MCD-56306-0059	Instrument data packages for the MICADO simulation environment	1.0	2021-04-12
AD 3	ELT-ICD-MCD-56306-0060	ScopeSim - A modular astronomical instrument data simulation environment	1.0	2021-04-12
AD 4	ELT-TRE-MCD-56300-0014	MICADO Masks, Stops, and Filters Description	2.9	2020-12-04

2.2 Reference Documents

Nr	Doc. Nr	Doc. Title	Issue	Date
RD 1	ELT-PLA-MCD-56301-0004	Operational Concept Description	2	2019-11-09
RD 2	ELT-TRE-MCD-56300-0011	MICADO System Design and Analysis	1.5	2021-04-12
RD 3	ELT-ICD-MCD-56306-0050	SimCADO: the Data Simulator for MICADO	1.0	2018-09-27

3 SimCADO v2 = ScopeSim + MICADO

SimCADO v2 combines the ScopeSim simulation engine with dedicated MICADO packages in ScopeSim's instrument reference database

This latest iteration of SimCADO is superior to the original version in the sense that the data needed to produce the MICADO optical model is completely decoupled from the code used to simulate the observations and from the code used to describe the target source.

In a word, SimCADO has been de-spaghetti-afied.

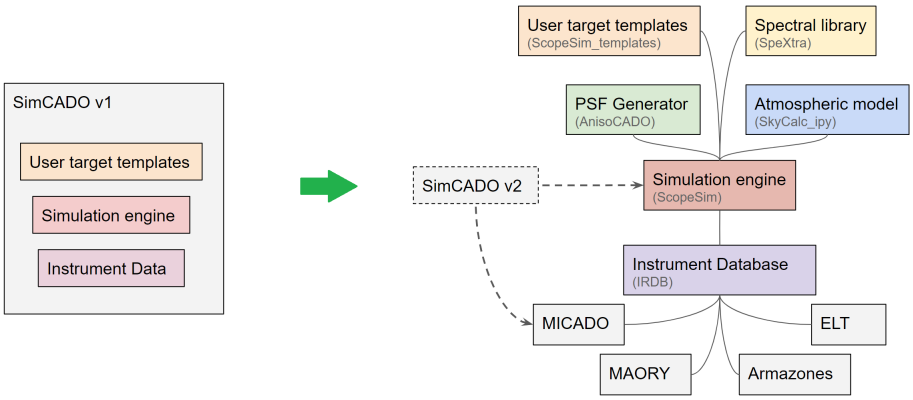


Figure 3.0.0: Left: SimCADO v1 contained everything needed to run a simulation for MICADO. However the inner algorithms were convoluted and interconnected. Right: The ScopeSim environment offers all the functionality of SimCADO v1, but with a decoupled code base. This allows each aspect of the simulation workflow to be updated and improved independently of all other systems. SimCADO v2 essentially now consists of a data package in the IRDB and utilises the in-built Effect object nativce to ScopeSim.

SimCADO v1 contained everything needed to run a simulation for MICADO. However the inner algorithms were convoluted and interconnected. The ScopeSim environment offers all the functionality of SimCADO v1, but with a decoupled code base. This allows each aspect of the simulation workflow to be updated and improved independently of all other systems. SimCADO v2 essentially now consists of a data package in the IRDB and a set of configuration files that utilise the in-built Effect objects native to ScopeSim.

A further advantage of using the ScopeSim architecture is that observations with MICADO can be compared directly with observations with other telescopes and instruments quickly and efficiently on a common platform.

4 Essentials: Documentation and Downloads

4.1 Online Documentation

Online documentation for the main packages in the ScopeSim environment can be found here:

- ScopeSim: https://scopesim.readthedocs.io/en/latest/
- ScopeSim_Templates: https://scopesim-templates.readthedocs.io/en/latest/
- IRDB: https://github.com/astronomyk/irdb

The original SimCADO package is described here:

- SimCADO: https://simcado.readthedocs.io/en/latest/

> **Note**
> In the near future we will release a wrapper for the ScopeSim engine and the MICADO instrument package.
> The doumentation for this will be added to the original SimCADO read-the-docs page

4.2 Downloading ScopeSim and the MICADO packages

The ScopeSim engine is installed using pip:

```
$ pip install scopesim
```

The casual user will also probably want to install the templates package, which contains helper functions for generating descriptions of on-sky targets like elliptical galaxies or star clusters:

```
$ pip install scopesim_templates
```

Once ScopeSim is available to the local Python (version >= 3.5) installation, the user must download **ALL** the required instrument packages from the server:

```
from scopesim.server import download_package
download_package(["locations/Armazones",
                  "telescopes/ELT",
                  "instruments/MAORY",
                  "instruments/MICADO",
                  "instruments/MICADO_Sci"])
```

> **Note**
> There are two (2) MICADO packages available: `MICADO` and `MICADO_Sci`.

For those interested in quick results, `MICADO_Sci` provides a reduced version of the MICADO package that contains all the major effects expected from the MICADO optical system. For those more concerned with accuracy, the standard `MICADO` package contains all expected optical effects. `MICADO` was originally developed for the development of the reduction pipeline, and therefore contains many effects that are beyond the scope of normal science case feasability studies.

4.3 A quick note on Primary vs Support packages

Each package contains descriptions of the optical effects associated with an optical element. For example the ELT package contains only the radiometric curves (e.g. transmission, emissivity, reflection) associated with the five ELT mirrors.

Packages are split into two categories: Primary and Support packages.

Primary packages contain detector modules that enable an on-sky target to be observed. Without a (single) primary package, ScopeSim will not be able to produce any simulated data. Examples of primary packages are: MICADO, METIS, HAWKI, NACO, etc

Support packages do not contain detector modules. They're main function is to add the additional optical effects associated with the common optical components of an optical system. `Armazones`, `ELT`, and `MAORY` are examples of such support packages.

Just like in real life, observing with only MICADO would be a difficult task. Therefore we highly encourage the user to also download the support packages needed by MICADO.

5 Basic functionality

A basic simulation of an ELT/MICADO observation using the MICADO_Sci package would look something like this:

```
# Setup everything
import scopesim_templates
from scopesim.server.database import download_package
import scopesim

scopesim.download_package(["locations/Armazones",
                    "telescopes/ELT",
                    "instruments/MICADO_Sci"])

# Make the on-sky target
src = scopesim_templates.basic.stars.cluster()

# Load the model of MICADO and the ELT
cmd = scopesim.UserCommands(use_instrument="MICADO_Sci",
                     properties=\{"!OBS.dit": 60, "!OBS.ndit": 10,
                                 "!INST.filter_name": "Ks"\})
micado = scopesim.OpticalTrain(cmd)

# Run the simulation
micado.observe(src)
micado.readout(filename="my_image.fits")
```

5.1 General Workflow

The general workflow for simulating observations can be seen in the code in the previous section. There are three (four) major components of any simulation workflow:

1. the target description,
2. the telescope/instrument model, and
3. the observation.

The zeroth (or fourth) step is the setup stage which include the import statements and the package downloads.

The first step of any simulation run is defining the on-sky target. This is probably the most time intensive step and requires the user to know what they want to observe. In the code above we make use of the ScopeSim_Templates package to create a `Source` object for an open cluster with default values (i.e. 1000 Msun at 50 kpc).:

```
src = scopesim_templates.basic.stars.cluster()
```

`Source` objects are discussed in a later chapter. Here it is sufficient to note that `Source` objects can be created from scratch if the user has both spectral and spatial data available. Otherwise the ScopeSim_Templates package contains various helper functions to aid the user in quickly generating "standard" objects.

The second step required creating an in-memory model of the optical system. This model contains a list of effects that are applied to the incoming photon description (similar to a flux cube).:

```
cmd = scopesim.UserCommands(use_instrument="MICADO_Sci",
                    properties={"!OBS.dit": 60, "!OBS.ndit": 10,
                                "!INST.filter_name": "Ks"})
micado = scopesim.OpticalTrain(cmd)
```

In this code we initialise a series of command dictionaries that control how the optical model will be built. Any default values that we wish to override can be set by passing a dictionary to the `properties` keyword. The "controlling a simulation" section goes into more detail on how to adjust these parameters. The command dictionary (`cmd`) is used to initialise an `OpticalTrain` object which acts as the in-memory model of MICADO. Here we use the `MICADO_Sci` optical model rather than the generic `MICADO` package, as the `MICADO_Sci` package has been optimised for speedy simulations.

The third step executes the simulation by passing the `Source` object through the `OpticalTrain` model.:

```
micado.observe(src)
micado.readout(filename="my_image.fits")
```

This step involves two commands: `.observe` generates an photon flux image (or images) on the focal plane(s) of the instrument. `.readout` converts the photon flux image into electrons inside the detector, and ultimately into the FITS objects that all astronomers know (and love?). The output of this step is a FITS object available either as an `astropy.fits.HDUList` object inside the python environment, or as a `.fits` file saved to the hard drive.

6 Controlling a simulation

6.1 A note on the two MICADO packages

It should be noted that there are two MICADO packages available:

1. MICADO_Sci: optimised for quick run times to enable rapid iterations of science feasibility use cases
2. MICADO: the generalised package containing all the known optical effects needed by the pipeline development team.

In general science team members are recommended to use the MICADO_Sci package as the generalised MICADO is slower and exposes many more configuration parameters to the user.

Both packages **still** require all upstream support packages, namely: Armazones, ELT, and MAORY.

6.2 Available MICADO observing modes

As mentioned in the previous sections there are two MICADO models available: MICADO_Sci and MICADO. The desired model is selected with the `use_instrument=` parameter:

```
cmd = scopesim.UserCommands(use_instrument="MICADO_Sci", ...)
```

The choice of observing modes can be set when the `UserCommand` object is initialised by passing a list of mode names to the `set_mode=` parameter. Note that multiple keywords can be passed as different modes for different optical sections can be combined. For example, using the MAORY relay optics with the MICADO wide-field imaging mode requires the following input:

```
cmd = scopesim.UserCommands(..., set_modes=["MCAO", "IMG_4mas"])
```

In contrast, to use the MICADO internal SCAO mode with the zoom imaging mode, the following input is required:

```
cmd = scopesim.UserCommands(..., set_modes=["SCAO", "IMG_1.5mas"])
```

The following table lists all available mode keywords for MICADO.

Keyword	Description
MCAO	MAORY relay optics model is included
SCAO	MICADO SCAO relay optics module is included

... continued on next page

Keyword	Description
IMG_4mas	Wide-field imaging optics included. Images have 4mas pixel scale.
IMG_1.5mas	Zoom imaging optics included. Images have 1.5mas pixel scale.
SPEC	'MICADO_Sci' package only. Produces rectified spectroscopic images.
SPEC_3000x20	'MICADO' package only. Produces full raw detector traces for the short-narrow slit (3000x20mas)
SPEC_3000x50	'MICADO' package only. Produces full raw detector traces for the short-wide slit (3000x50mas)
SPEC_15000x50	'MICADO' package only. Produces full raw detector traces for the long-wide slit (15000x50mas)

Note

While the imaging mode names are consistent between the MICADO_Sci and MICADO packages, the spectroscopy mode names differ.

6.3 Setting top level observation parameters

The `UserCommands` object is built on several hierarchical (YAML-style) nested dictionaries which contain all the information needed to build the optical model of MICADO. Checking and changing parameters in these dictionaries is done using the so-called "bang-string" notation. Bang-strings shorten the syntax for accessing nested dictionaries. E.g. `cmd["!OBS.ndit"]` is the equivalent of `cmd["OBS"]["ndit"]`

Alternatively, a dictionary of bang-string keyword-value pairs can be passed to the `UserCommands` object when initialising via the `properties` keyword:

```
scopesim.UserCommands(properties={"!OBS.dit": 60, "!OBS.ndit": 10})
```

The following table describes the top level parameter categories.

Alias	Description
OBS	Instrument specific parameters. E.g. exposure time and number (DIT, NDIT), filter name, etc
ATMO	Atmospheric specific parameters. E.g. air temperature, background brightness, location, etc
TEL	Telescope specific parameters. E.g. Dome temperature, etc
RO	Relay optics specific parameters. E.g. RO module temperature, etc
INST	Instrument specific parameters. E.g. plate scale, strehl ratio (MICADO_Sci only), etc
DET	Detector specific parameters. E.g. windowing dimensions and position (MICADO_Sci only), etc
SIM	Instrument specific parameters. E.g. spectral range, computing parameters, file locations, etc

> **Note**
> Important parameters like which filter to use and how long to observe for are generally kept in the `"!OBS"` dictionary

The whole parameter dictionary can be shown by simply printing the `UserCommand` object:

```
print(cmd)
```

Specific categories can be shown by using the category alias in the bang string format:

```
print(cmd["!OBS"])
```

A specific parameter (or sub-group thereof) can be shown by following the hierarchy in bang-string format:

```
print(cmd["!INST.psf.strehl"])
```

Parameter values can also be set in this manner. For example, to set which filter to use with MICADO, we call the `"OBS.filter_name` keyword:

```
cmd["!OBS.filter_name"] = "Ks"
```

> **Note**
> The full list of filters available in MICADO is kept in the `OpticalTrain` object. This will be discussed below.

The following list contains some commonly used parameters for the `MICADO_Sci` package:

```
MICADO_Sci
==========

OBS.filter_name: "J"          # MICADO filter name
OBS.dit: 60                   # [s] Length of exposure
OBS.ndit: 1                   # Number of exposures to stack
OBS.airmass: 1.2

ATMO.background.filter_name: "Ks"
```

```
ATMO.background.value: 13.6   # Amplitude of background flux
ATMO.background.unit: "mag"   # Unit of background flux

INST.psf.strehl: 0.8          # Desired Strehl ratio (as far as
                              #  physically possible)
INST.psf.wavelength: "Ks"     # Either wavelength [um] or common
                              #  broadband filter name
INST.aperture.x: 0            # [arcsec] (MICADO_Sci only) Slit
                              #  position relative to FOV centre
INST.aperture.y: 0
INST.aperture.width: 3        # [arcsec] (MICADO_Sci only)
                              #   The slit dimensions
INST.aperture.height: 0.05

DET.width: 1024               # [pixel] Detector dimensions
DET.height: 1024
DET.x: 0                      # [pixel] Position of window centre
                              #  relative to detector plane centre
DET.y: 0

SIM.spectral.wave_min: 0.7    # [um] Spectral borders of simulation
SIM.spectral.wave_mid: 1.6    # [um] Central wavelength for SPEC mode
SIM.spectral.wave_max: 2.5
SIM.sub_pixel.flag: False     # Turn on for astrometry observation
SIM.random.seed: 9001         # Random seed for numpy
SIM.file.use_cached_downloads: True # Turn off when checking for updates
```

The spectroscopy mode of the `MICADO_Sci` package contains a configurable slit and detector window. This allows the user to choose only the spectral range that is relevant to their science case. Restricting the spectral range like this speeds up the computation time by order and allows for fast iterations of science use cases. The pipeline-oriented `MICADO` package contains fixed slit dimensions and spectral trace layouts. Simulations run using this setup are slow and memory intensive.

6.4 The MICADO filter settings

To set the filter name, we use the `!OBS.filter_name` bang-string.

To find out which filters are included in the MICADO_Sci and MICADO packages, we must create an Optical-Train model for MICADO:

```
cmd = scopesim.UserCommands(use_instrument="MICADO_Sci")
micado = scopesim.OpticalTrain(cmd)
```

In the `MICADO_Sci` package all filters are contained in a single `filter_wheel` object:

```
micado["filter_wheel"].data["name"]
```

For the pipeline-oriented `MICADO` package, the filters are distributed among the three filter wheels: `filter_wheel_1`, `filter_wheel_2`, and `pupil_wheel`. Hence there is no central list of all filters for the `MICADO` package.

6.5 The MICADO_Sci PSF model

The `MICADO_Sci` package uses the AnisoCADO python package to generate ELT-like PSFs on the fly. The Strehl ratio can therefore be set (within reason) to whatever the user desires. The global bang-string keywords for this are `!OBS.psf.strehl` and `!OBS.psf.wavelength`:

```
cmd["!OBS.psf.strehl"] = 0.35
cmd["!OBS.psf.wavelength"] = 2.15
```

These settings will create an on-axis PSF with a 35% Strehl ratio at 2.15um. At shorter wavelengths the Strehl ratio will scale as expected. If a Strehl ratio is physically impossible (e.g. 80% at 0.9um), the largest PSF will be generated with the best physically possible Strehl ratio.

It is possible to also create PSF models for off-axis positions in SCAO mode. The offset can only be specified after the `OpticalTrain` model has been created by accessing the PSF object inside the model. The relevant MICADO `OpticalTrain` objects are `scao_const_psf` for SCAO observations and `maory_const_psf` for MCAO observations:

```
cmd = scopesim.UserCommands(use_instrument="MICADO_Sci",
                      set_modes=["SCAO", "IMG_4mas"])
cmd["!OBS.psf.strehl"] = 0.35
cmd["!OBS.psf.wavelength"] = 2.15

micado = scopesim.OpticalTrain(cmd)
micado["scao_const_psf"].meta["offset"] = [10, 0]
```

This will create an on-axis PSF with a 35% Strehl ratio at 2.15um, and then shift the 10 arcseconds off axis. The form and Strehl ratio of the PSF will be adjusted to match what is expected for the off-axis position.

Note

While the `MICADO_Sci` PSF model is configurable, the `MICADO` PSF model is not.
The `MICADO`-package uses pre-computed PSF files. These are available on the ScopeSim server: https://www.univie.ac.at/simcado/InstPkgSvr/psfs/

7 Making on-sky Sources

On-sky target source objects for ScopeSim can be created in two ways: by using the helper functions provided in the ScopeSim_Templates package, or by creating a `Source` object from scratch. The latter requires the user to have both the spatial and spectral data needed to describe the object.

7.1 ScopeSim Templates

A useful addition to the ScopeSim eco-system is the package `ScopeSim_templates`. This python package provides so-called helper functions for generating ScopeSim readable `Source` object for common astronomical objects. The `basic` subpackage contains functions for generating star clusters and grids of stars, as well as analytical and numerical representations of galaxies. The documentation for ScopeSim templates can be found on Read-The-Docs.

Here is a basic example of creating a star cluster using `ScopeSim_templates`:

```
from scopesim_templates.basic.stars import cluster

my_cluster = cluster(mass=1000, distance=50000,
                     half_light_radius=1)
```

In section 7, various helper functions are used to create the source objects observed in the presented use cases.

7.2 The ScopeSim Source object

The `Source` object is the underlying python class that contains the spatial and spectral descriptions of the on-sky target. The spatial description can be either a table of point source positions, or a 2D intensity map (image) of an extended object.

Point source tables must contain the following columns:

Column	Description
x, y	[arcsec] position of source relative to centre of FOV
ref	index of spectrum associated with the point source
(weight)	[default=1] scaling factor if multiple point sources reference the same spectrum

Extended objects intensity maps (images) must be `astropy.fits.HDUImage` objects that display a the extent of similar spectral regions. Each image may only be linked to one spectrum. An extended `Source` may however contain multiple images that reference different spectra. To illustrate the point, a spiral galaxy may be split into two intensity maps (images). The first shows the distribution of the new stellar population, while the second shows the distribution of the old stellar population. Each of these images is linked to a generic spectrum for the given population. Conceivably a third image showing the distribution of warm gas and the spectrum for warm gas could be added to the `Source` object.

Each ImageHDU must contain the following information in the header:

Header Keyword	Description
CDELT1, CDELT2	Pixel scale of the image
CUNIT1, CUNIT2	Unit of the pixel scale value.
SPEC_REF	Spectrum associated with the image

All `Source` objects require list of one or more spectra to be passed to the `Source` object when it is created. Spectra can either be `synphot.SourceSpectrum` objects or numpy arrays. If they are passed as numpy arrays, a corresponding wavelength array must also be passed.

> **Note**
>
> The ScopeSim affiliate packages `Spextra` and `Pyckles` provide access to many standard spectral libraries.
> These python packages return spectra in a variety of formats including the `synphot.SourceSpectrum` format.

The following code shows the process of creating a simple binary star system from scratch:

```
# Get two spectra
from pyckles import SpectralLibrary
pickles = SpectralLibrary("pickles", return_style="synphot")
spec = [pickles.A0V, pickles.G2V]

# Make a table
from astropy.table import Table
tbl = Table(names=["x", "y", "ref"],
            data=[[-2, 2], [0, 0], [0, 1]])

# Combine spectra and positions in the Source object
from scopesim import Source
binary = Source(table=tbl, spectra=spec)
```

Source objects can also be combined using the + operator:

```
star = Source(x=[0], y=[10], ref=[0], spectra=[pickles.M0III])
trinary = binary + star
```

The `trinary` Source object now contains two tables with the positions of the stars and three spectra. This can be verified by looking at the `<Source>.fields` and `<Source>.spectra` attributes.

We can also combine point source with extended source objects. Each object remains intact inside the new `Source` container.

8 Use case examples using the MICADO_Sci and MICADO instrument packages

8.1 Notes regarding the worked examples

> **Note**
> All examples listed here are available online as part of the ScopeSim documentation.

This section presents a series of 5 use case examples for both the science-team oriented `MICADO_Sci` package and the pipeline oriented `MICADO` package. All example are available as iPython notebooks as part of the ScopeSim online documentation. Please note that no scientifically relevant information is extracted from these simulations. The sole purpose of the attached notebooks is to illustrate how to run ScopeSim with the two MICADO packages.

- Observing a galaxy with MICADO_Sci in wide-field MCAO mode:

  This example shows how to use the science package to observe a resolved spiral galaxy using the `scopesim_templ` function. It uses a windowed detector object to avoid simulating the full MICADO detector array. The pixel scale is set to 4mas (low resolution imaging) and a MCAO field constant PSF is used. The galaxy is observed for 1 hour (60x 60s images) in the J filter.

- Astrometric observations of an open cluster in zoom SCAO mode:

  This example shows how to enable sub-pixel resolution for simulations with the `MICADO_Sci` package. Two basic examples are shown: the sub-pixel motions of a grid of 4 stars, and the proper motions of stars in an open cluster. Both MICADO pixel scales (4mas and 1.5mas) are used for the example, together with the SCAO mode. The observations are in the J filter with an exposure time of 1 hour (1x 3600s). This example also makes use of the variable size of a theoretical detector sub-frame readout (window).

- Field variations in the PSF using a star cluster and wide-field SCAO mode (advanced level):

  This example shows the effects of of field dependent variations in the shape of the PSF for off-axis stars using the pipeline-oriented `MICADO` package. A standard open cluster is observed in the Ks filter for 10 minutes (1x 600s). The use of various detector window sizes as well as the use of the full MICADO detector array are described. This example also illustrates how `Effect` objects can be exchanged for more realistic simulation outputs.

- Basic spectroscopy observation of a galaxy core with MICADO_Sci:

  This example shows how to generate rectified images of spectra for a desired wavelength range using the `SPEC` mode of the `MICADO_Sci` package. The 3 arcsec slit is placed over the centre of the galaxy and observed for 1 hour using the `Spec_IJ` filter. The final output image shows distance along the slit on the horizontal axis and wavelength on the vertical axis. ScopeSim **does not** provide functionality for extracting spectra from the images. Hence it is up to the user to extract the regions corresponding to their desired spectra from this image.

- Advanced spectroscopy observation of a binary star system with MICADO:

  This example shows how to use the pipeline-oriented `MICADO` package to generate raw detector readouts for the MICADO spectroscopy mode. The spectroscopy modes in this package are not aimed towards the casual user, but rather are meant to simulate realistic raw data frames for the data reduction pipeline.

There may however be science users who wish to have such data for their feasibility studies. The example observes a binary star system with the 3000x50 slit and the `Spec_HK` filter for 10 minutes. Each star ha a K-band brightness of 18 magnitudes.

8.2 A note on High Contrast Imaging simulations

High contrast imaging has been described as an Artform. It is almost impossible to define a "standard" HCI PSF. ScopeSim and the MICADO packages work with data models that can be standardised, or at the very least be described by a low-dimensional parameter space. As the high contrast imaging PSFs do not fit this definition, we concluded that a HCI mode was outside the scope of the `MICADO` and `MICADO_Sci` packages. Separate HCI simulation packages are being developed by the HCI work package. Readers interested in simulating HCI use cases should contact the relevant work package leaders.

Observing a galaxy with MICADO_Sci in wide-field MCAO mode

In this example we will generate a basic galaxy object with the help of the `ScopeSim_templates` package. We will then create a model of the ELT+MAORY+MICADO optical system using the `MICADO_Sci` package. The `MICADO_Sci` package has been optimised for speed so that the user can rapidly iterate on their science case.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from astropy import units as u

import scopesim as sim
import scopesim_templates as sim_tp

%matplotlib inline
%config Completer.use_jedi = False
```

Download the relevant scopesim instrument packages

We will need the `MICADO_Sci` package, as well as the support packages for `Armazones`, the `ELT`, and `MAORY`

To view all available packages, use the `sim.list_packages` command

In [3]:

```
sim.download_package(["locations/Armazones.zip",
                      "telescopes/ELT.zip",
                      "instruments/MAORY.zip",
                      "instruments/MICADO_Sci.zip",])
```

Out[3]:

```
['F:\\temp\\scopesim_fdr_notebooks\\Armazones.zip',
 'F:\\temp\\scopesim_fdr_notebooks\\ELT.zip',
 'F:\\temp\\scopesim_fdr_notebooks\\MAORY.zip',
 'F:\\temp\\scopesim_fdr_notebooks\\MICADO_Sci.zip']
```

Set up on-sky source object

For ease of use we will use the helper function from `ScopeSim_templates` to create a two-layer spiral galaxy object. The `Source` object contains two image layers (in `<Source>.fields`) that each reference a unique spectrum. The two layers correspond to the old and the new stellar populations of the galaxy.

In [2]:

```
gal = sim_tp.basic.galaxy.spiral_two_component(extent=16*u.arcsec, fluxes=(15, 15)*u.mag)
```

```
https://www.univie.ac.at/simcado/scopesim_templates/spiral_two_component.fit
s (https://www.univie.ac.at/simcado/scopesim_templates/spiral_two_component.
fits)
```

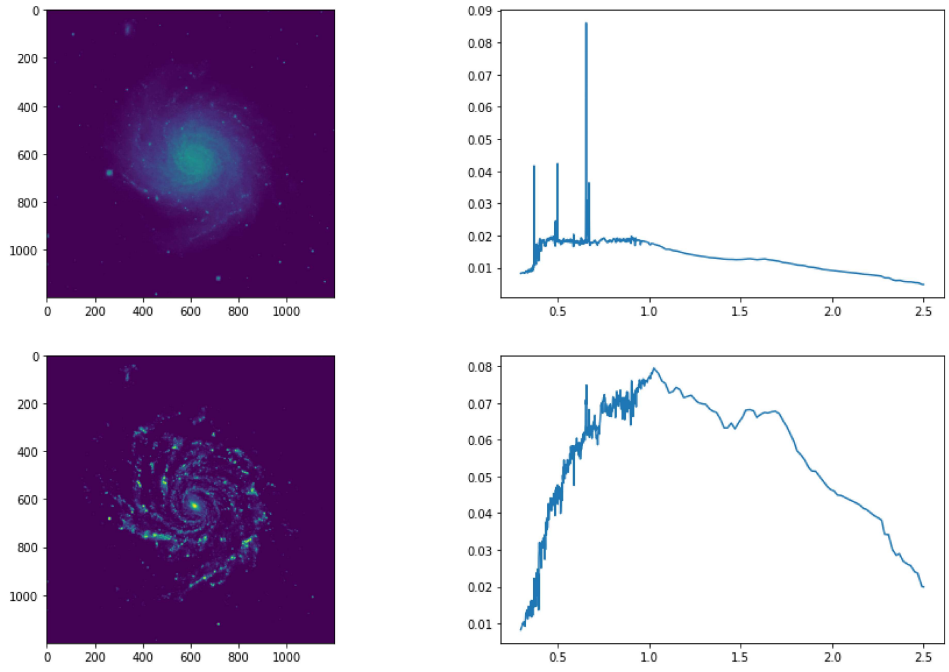
Here we can see what is contained in the `.fields` and `.spectra` lists of the `Source` object

In [3]:

```
wave = np.linspace(0.3, 2.5, 2201) * u.um
plt.figure(figsize=(15, 10))

plt.subplot(221)
plt.imshow(gal.fields[0].data)
plt.subplot(222)
plt.plot(wave, gal.spectra[0](wave))
plt.subplot(223)
plt.imshow(gal.fields[1].data)
plt.subplot(224)
plt.plot(wave, gal.spectra[1](wave))
```

Out[3]:

```
[<matplotlib.lines.Line2D at 0x201bc19e2e8>]
```



Set up the MICADO_Sci system for MCAO 4mas J-band observations

The next step is to create a model of MICADO in memory. First we need to generate a set of commands that we can manipulate. Once we have these we can create a model of the optical system.

We use the `UserCommands` class and tell it to look for the `MICADO_Sci` package. At the same time we set the observing modes to use. In this case we want `MCAO` and the 4mas per pixel wide-field optics (`IMG_4mas`)

In [5]:

```
cmd = sim.UserCommands(use_instrument="MICADO_Sci", set_modes=["SCAO", "IMG_4mas"])
```

We can update many of the observation parameters using the "!-bang" string syntax to access the values inside the hierarchical nested-dictionary structure of the `UserCommands` object.

Here we set the filter to the J-band, exposure length (DIT) to 60 seconds and number of exposures (DIT) to 1. We can pass these parameters either as a dictionary using the `.update(properties={...})` method, or individually using normal dictionary notation with the "!-bang" strings.

In [7]:

```
cmd.update(properties={"!OBS.filter_name": "J", "!OBS.ndit": 1, "!OBS.dit": 60})

cmd["!DET.width"] = 2048     # pixel
cmd["!DET.height"] = 1024
```

Now that we have the commands set, we create the MICADO optical model

In [8]:

```
micado = sim.OpticalTrain(cmd)
```

Observing and reading out the detectors is also a simple operation

In [9]:

```
micado.observe(gal)
hdus = micado.readout()
```

```
Preparing 1 FieldOfViews
```

The output of a simulation is a list of `fits.HDUList` objects. One `HDUList` for each detector plane. In MICADO there is only 1 detector plane, but the class interface still requires that this single `HDUList` be placed in another list.

We can view the output of our custom Detecotr window like we would for any regular fits image in Python

Alternatively we could save the readout file to disk with the parameter `filename=` :

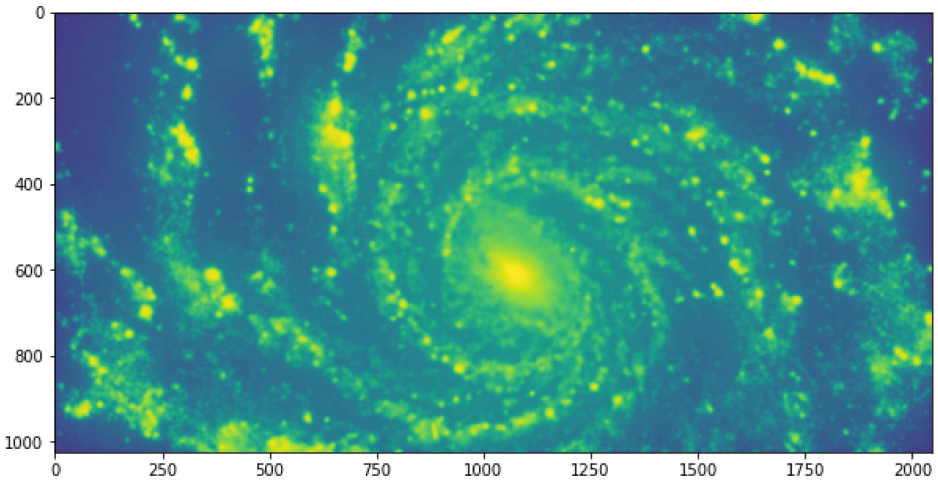
```
micado.readout(filename="my_galaxy.fits")
```

In [10]:

```
plt.figure(figsize=(10,10))
plt.imshow(hdus[0][1].data, norm=LogNorm())
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x201bd4ce278>
```



Astrometric observation simulations with MICADO_Sci and SCAO

In this eample we demonstate the sub-pixel precision of ScopeSim simulation by creating a grid of moving stars and simulating observations of two epochs. The second part then applies the same workflow to the motions of stars in a star cluster. For the sake of computational speed we use the `MICADO_Sci` package. We use the wide-field mode (4 mas / pixel) so that the sub-pixel motions of the stars are visible in the flux distribution of the PSF cores.

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from astropy import units as u

import scopesim as sim
import scopesim_templates as sim_tp

%matplotlib inline
%config Completer.use_jedi = False
```

As always, the first step is to make sure all instrument packages are present in the working directory. The assumption is that SCAO observations will use the MICADO internal SCAO system. Hence the MAORY package is not required for this optical models.

In [3]:

```
sim.server.database.download_package(["locations/Armazones.zip",
                                      "telescopes/ELT.zip",
                                      "instruments/MICADO_Sci.zip",])
```

Out[3]:

```
['F:\\temp\\scopesim_fdr_notebooks\\Armazones.zip',
 'F:\\temp\\scopesim_fdr_notebooks\\ELT.zip',
 'F:\\temp\\scopesim_fdr_notebooks\\MAORY.zip',
 'F:\\temp\\scopesim_fdr_notebooks\\MICADO_Sci.zip']
```

Set up a source with 4 stars in a grid pattern

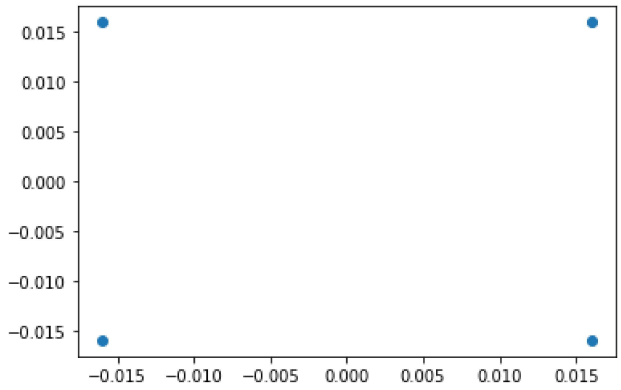
For the first part of this exercise we create a grid of stars seperated by 32 mas. The `ScopeSim_templates` pacakge has a helper function for making griods of stars in the `basic.stars` submodule:

In [4]:

```
stars = sim_tp.basic.stars.star_grid(n=4, mmin=15, mmax=15, filter_name="Ks", separation=0.

plt.scatter(stars.fields[0]["x"], stars.fields[0]["y"])
```

Out[4]:

```
<matplotlib.collections.PathCollection at 0x2a5dad4a908>
```



Set up the MICADO_Sci system for SCAO 4mas Pa-beta observations

The next step is to set up the `MICADO_Sci` optical system using the `SCAO` and wide-field optics (`IMG_4mas`) modes. As we know in advance that we want to use the Pa-Beta filter, we can set this parameter by passing a dictionary to the `properties=` keyword.

Additionally, we are only interested in a small section of the detector around where the 4 stars will be located. We can therefore set our Detector to only read out the desired window area. The required "!-bang" string keywords for this are `!DET.width` and `!DET.height` (given in units of pixels).

As we are interested in the sub-pixel movements of the stars, it is also important to set the `!SIM.sub_pixel.flag` to True

Given that we are observing at a non-standard broadband filter, we need to instruct the `OpticalTrain` to adjust the PSF strehl ratio. This is done with the `!INST.psf` dictionary.

In [56]:

```
cmd = sim.UserCommands(use_instrument="MICADO_Sci", set_modes=["SCAO", "4mas"],
                       properties={"!INST.filter_name": "Pa-beta"})
cmd["!DET.width"] = 16     # pixel
cmd["!DET.height"] = 16

cmd["!SIM.sub_pixel.flag"] = True

cmd["!INST.psf.strehl"] = 0.1
cmd["!INST.psf.wavelength"] = 1.2
```

With all these parameters set, we can finally build the optical model and observe the stars.

In [57]:

```
micado = sim.OpticalTrain(cmd)
micado.observe(stars)
hdus = micado.readout()
```

```
Preparing 1 FieldOfViews
```

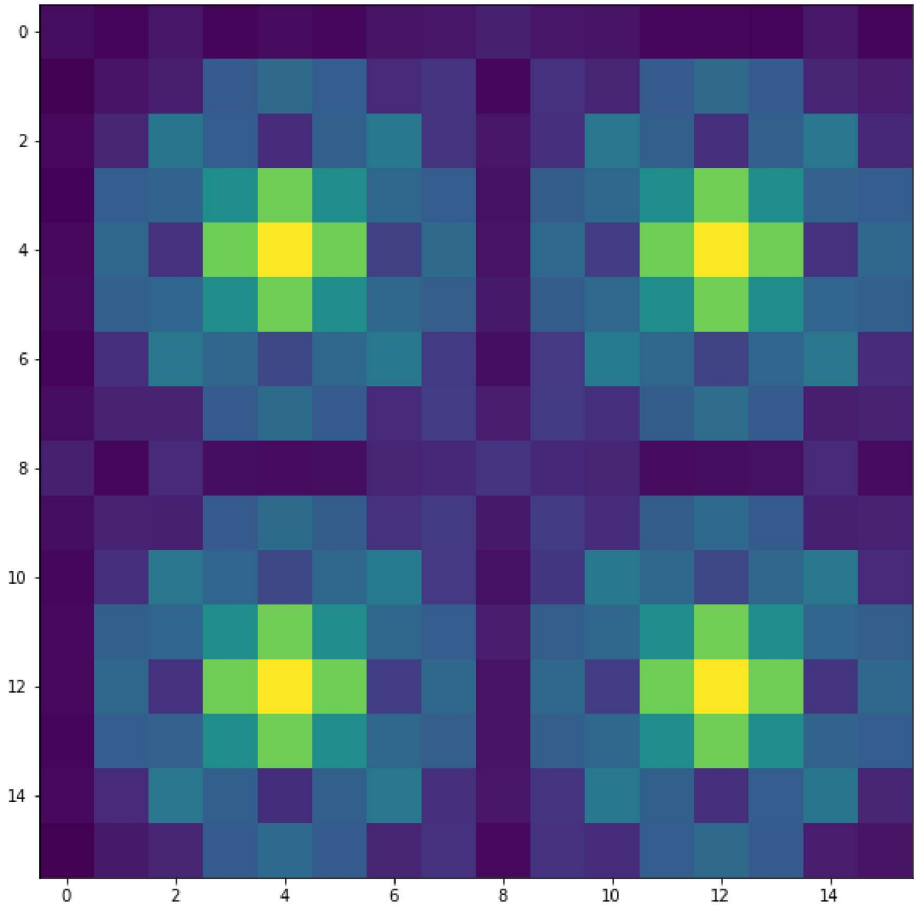
As we can see below the 4 stars appear nicely positioned in the grid. The pixelated 1st Airy ring is also visible. We chose the wide-field (4mas) mode for this example especially for this reason.

In [59]:

```
plt.figure(figsize=(10,10))
plt.imshow(hdus[0][1].data, norm=LogNorm())
```

Out[59]:

```
<matplotlib.image.AxesImage at 0x2a5dfdc5e80>
```



Observing a second Epoch

To simulate the movement of the stars due to their proper motions over time, we can shift the stars by a random fraction of a pixel in both directions using the `.shift` method of a point-source `Source` object.

In [60]:

```
dx, dy = 0.004 * np.random.random(size=(2, 4))
stars.shift(dx=dx, dy=dy)
```

As the `MICADO_Sci` optical model already exists and we do not want to change anything about the system, we can simply observe and readout the updated `Source` again.

In [65]:

```
micado.observe(stars)
hdus2 = micado.readout()
```

```
Preparing 1 FieldOfViews
```

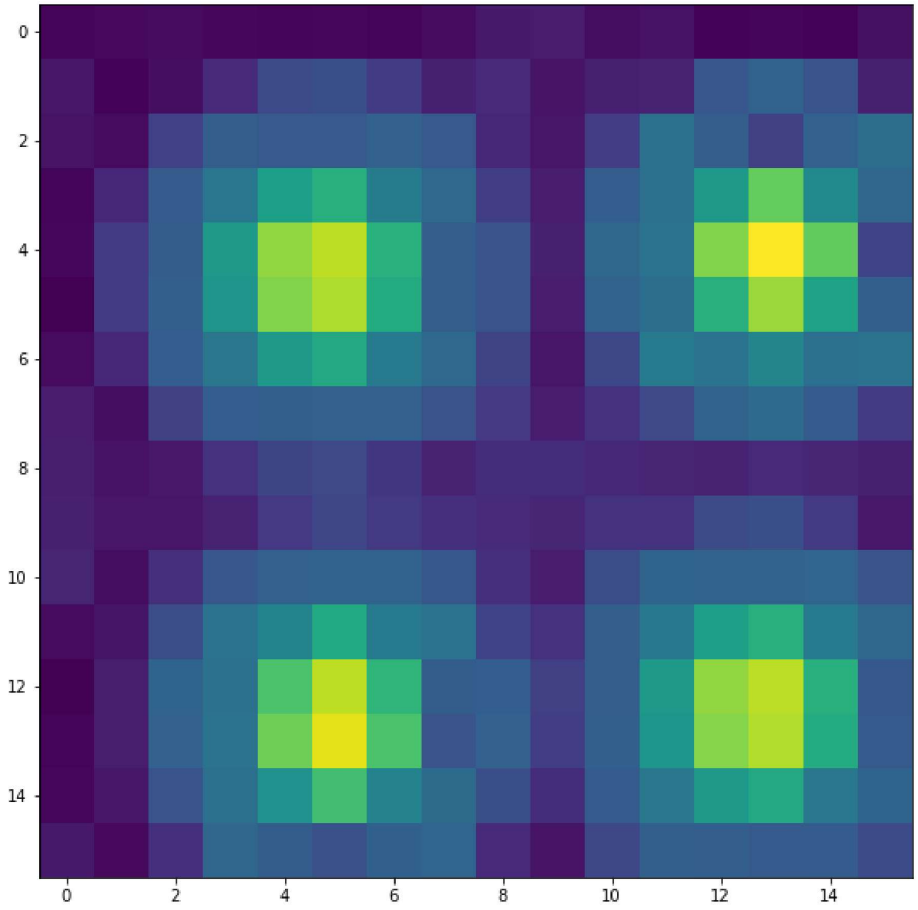
The added random fraction of a pixel motion to each star results in the centre of mass of each star being unaligned with the pixel.

In [64]:

```
plt.figure(figsize=(10,10))
plt.imshow(hdus2[0][1].data, norm=LogNorm())
```

Out[64]:

```
<matplotlib.image.AxesImage at 0x2a5deba8518>
```



Cluster movement use case

The above examples are nice for illustrating a simple use case. For more complex cases, such as the motions of stars within a star cluster, the work flow remains the same. We can use the `ScopeSim_Templates` package function to create a plausible open cluster containing 10 solar masses worth of stars, located at a distance of 8 kpc. The core radius of 0.002 pc is wildly unrealistic, but this dense core serves the purpose of illustrating the sub-pixel nature of ScopeSim simulations.

Here `t0_cluster` and `t1_cluster` are two realisation of the same cluster at different epochs.

While the `.shift` method described above would also be perfectly applicable here, we will create a copy of the original cluster object for the second Epoch. This allows us to visualise the "before and after" of the cluster simulations.

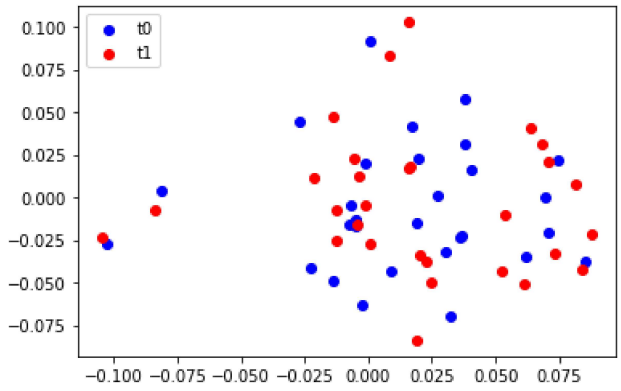
We also show how to access the table containing the spatial information inside the `Source` object's `.fields` attribute.

In [66]:

```
from copy import deepcopy
t0_cluster = sim_tp.basic.stars.cluster(mass=10, core_radius=0.002, distance=8000)
t1_cluster = deepcopy(t0_cluster)

pm_x, pm_y = np.random.normal(0, 0.02, size=(2, len(t0_cluster.fields[0])))

t1_cluster.fields[0]["x"] += pm_x
t1_cluster.fields[0]["y"] += pm_y

plt.scatter(t0_cluster.fields[0]["x"], t0_cluster.fields[0]["y"], c="b", label="t0")
plt.scatter(t1_cluster.fields[0]["x"], t1_cluster.fields[0]["y"], c="r", label="t1")
plt.legend(loc=2)
```

Out[66]:

```
<matplotlib.legend.Legend at 0x2a5dec5b668>
```



Observing the two epochs of the cluster

For an astrometric science case we are more likely to use the MICADO zoom optics (1.5mas / pixel). Therefore we recreate the optical mode with the updated `set_modes` keyword (`IMG_1.5mas`).

We also expand the size of the detector window to 256x256 pixels in order to fit our unrealisticly dense open cluster.

In [67]:

```
cmd = sim.UserCommands(use_instrument="MICADO_Sci", set_modes=["SCAO", "1.5mas"],
                       properties={"!OBS.filter_name": "J", "!OBS.dit": 3600})
cmd["!DET.width"] = 256     # pixel
cmd["!DET.height"] = 256

cmd["!SIM.sub_pixel.flag"] = True

micado = sim.OpticalTrain(cmd)
```

Observing and reading out the MICADO detector for each `Source` object is quite obvious by now

In [68]:

```
micado.observe(t0_cluster)
t0_hdus = micado.readout()

micado.observe(t1_cluster)
t1_hdus = micado.readout()
```

```
Preparing 1 FieldOfViews
Preparing 1 FieldOfViews
```

In contrast to the grid of stars being observed with the wide-field mode, the zoom-mode enables us to see much more of the PSF structure. The shift in star positions is also quite obvious.

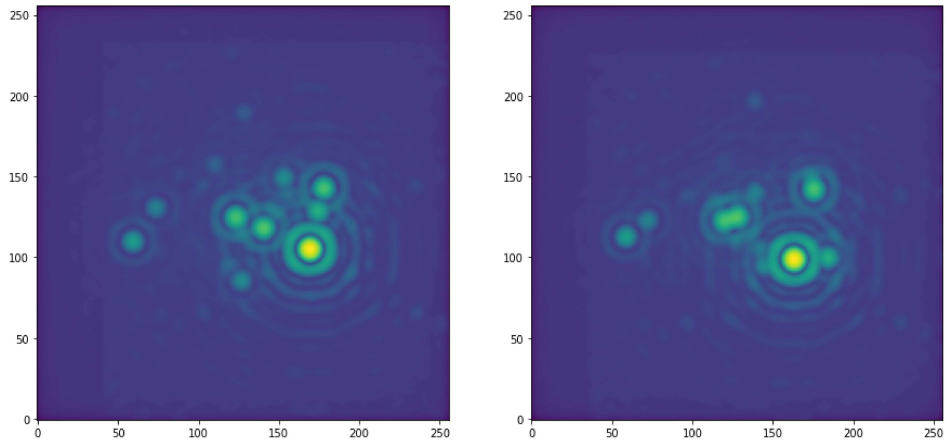
Each `fits.HDUList` could also ahve been saved to disk as a regular FITS file by passing a `filename=` keyword to the `readout` method call. Alternatively we could use the standard `astropy.fits` functionality (i.e. the `writeto()` method) to save the `HDUList` s to disk.

In [182]:

```
plt.figure(figsize=(15,8))
plt.subplot(121)
plt.imshow(t0_hdus[0][1].data, norm=LogNorm(), origin="lower")
plt.subplot(122)
plt.imshow(t1_hdus[0][1].data, norm=LogNorm(), origin="lower")
```

Out[182]:

```
<matplotlib.image.AxesImage at 0x2a42884e358>
```



Wide-field observations of PSF variations with SCAO

In this example we will show the effect of the off-axis AO correction on the shape of the PSF when observing with the SCAO mode. To make a semi-realistic observation, we will observe an open cluster created by the `ScopeSim_Templates` package. This cluster extends over the full MICADO field of view. As such it should show the full extend of the expected PSF variations over the field.

It should be noted that adding PSF field variations is a computationally expensive process. ScopeSim does it's best to reduce the level of complexity by discretising the PSF variation into different zones over the field of view. Occasionally the borders between these zones is visible

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from astropy import units as u

import scopesim as sim
import scopesim_templates as sim_tp

%matplotlib inline
%config Completer.use_jedi = False
```

The PSF field-variation are complex and computationally expensive. Hence this functionality is not inluded in the `MICADO_Sci` package. For this use case we must use the pipeline-oriented `MICADO` package. However as we will be using the SCAO mode, we can ignore the `MAORY` module.

In [2]:

```
sim.server.database.download_package(["locations/Armazones.zip",
                                      "telescopes/ELT.zip",
                                      "instruments/MICADO.zip"])
```

Out[2]:

```
['F:\\temp\\scopesim_fdr_notebooks\\Armazones.zip',
 'F:\\temp\\scopesim_fdr_notebooks\\ELT.zip',
 'F:\\temp\\scopesim_fdr_notebooks\\MICADO.zip']
```

Set up an open cluster `Source` object

The first step is to create a `Source` object. Here we take advantage of the `cluster` function from the `ScopeSim_Templates.basic.stars` submodule. The `cluster` function draws masses from an standard Kroupa IMF until the `mass` limit has been reached. It then generates randomly draw positions from a 2D gaussian distribution with a HWHM equal to the given `core_radius`. The positions are scaled by the `distance` parameter such that the final position is in units of `arcsec` from the centre of the cluster. Spectra for all the stars are taken from the Pickles (1998) catalogue and are imported from the `Pyckles` python packages.

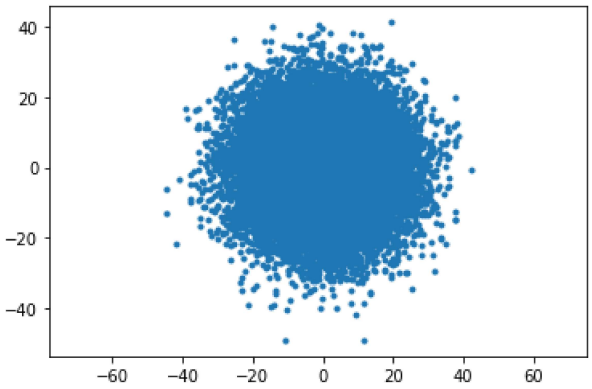
Here we choose a cluster with a mass of 10000 solar masses (~250000 stars) with a core radius of 0.5 parsec, located at a distance of 8 kpc. This essentially gives us a cluster that covers the whole MICADO filed of view in wide-field mode (4mas / pixel).

In [3]:

```
cluster = sim_tp.basic.stars.cluster(mass=10000, core_radius=0.5, distance=8000)
```

In [4]:

```
plt.plot(cluster.fields[0]["x"], cluster.fields[0]["y"], ".")
plt.axes().set_aspect('equal', 'datalim')
```



Set up the MICADO optical system for SCAO and include a field-varying PSF

The next step is the standard proceedure - set the instrument to `MICADO` and the modes to `SCAO` and `IMG_4mas`, then create an `OpticalTrain` object.

In [5]:

```
cmd = sim.UserCommands(use_instrument="MICADO", set_modes=["SCAO", "IMG_4mas"])
micado = sim.OpticalTrain(cmd)
```

By default the PSF model in the pipeline-oriented `MICADO` package is set to use a field-constant PSF cube. We need to change this to use a Field-Varying PSF model.

Note: to view all the optical effects contained within the optical model, use the `.effects` attribute in the `OpticalTrain` class.

Here we see the PSF model in the 4th last line:

In [6]:

```
micado.effects
```

Out[6]:

Table length=23

element	name	class	included
str21	str32	str31	bool
armazones	armazones_atmo_default_ter_curve	AtmosphericTERCurve	True
armazones	armazones_atmo_dispersion	AtmosphericDispersion	False
armazones	armazones_atmo_skycalc_ter_curve	SkycalcTERCurve	False
ELT	scope_surface_list	SurfaceList	True
ELT	scope_vibration	Vibration	True
ELT	eso_combined_reflection	TERCurve	False
MICADO	micado_static_surfaces	SurfaceList	True
MICADO	micado_ncpas_psf	NonCommonPathAberration	True
MICADO	filter_wheel_1	FilterWheel	True
MICADO	filter_wheel_2	FilterWheel	True
...	...	...	...
micado_detector_array	qe_curve	QuantumEfficiencyCurve	True
micado_detector_array	exposure_action	SummedExposure	True
micado_detector_array	dark_current	DarkCurrent	True
micado_detector_array	detector_linearity	LinearityCurve	True
micado_detector_array	shot_noise	ShotNoise	True
micado_detector_array	readout_noise	PoorMansHxRGReadoutNoise	True
default_ro	relay_psf	FieldConstantPSF	True
default_ro	relay_surface_list	SurfaceList	True
MICADO_IMG_LR	micado_wide_field_mirror_list	SurfaceList	True
MICADO_IMG_LR	micado_adc_3D_shift	AtmosphericDispersionCorrection	False

We can access the PSF model using the standard python dictionary notion using the name of the element. In this case it is called `relay_psf` as this refers to the internal MICADO relay optics which are connected to the SCAO system.

We do not want to use the default `relay_psf` object, so we need to tell ScopeSim not to include it

In [7]:

```
micado["relay_psf"].include = False
```

Next we want to add a `FieldVaryingPSF` object from the ScopeSim library of optical effect. Before we initialise the object though, we will need the correct PSF cube.

A small library of precomputed PSFs can be found on the ScopeSim server:
https://www.univie.ac.at/simcado/InstPkgSvr/psfs/ (https://www.univie.ac.at/simcado/InstPkgSvr/psfs/)

Once we have downloaded the PSF cube of our choice (in this case the cube corresponding to ESO median atmospheric conditions) we can pass the file path to the `FieldVaryingPSF` object.

In [8]:

```
fv_psf = sim.effects.psfs.FieldVaryingPSF(filename="AnisoCADO_SCAO_FVPSF_4mas_EsoMedian_201
```

We now need to add the new PSF to the MICADO relay optics element inside the `MICADO` optical train.

In [9]:

```
micado.optics_manager["default_ro"].add_effect(fv_psf)
```

Viewing an on-axis PSF is boring. Therefore we will move the 1024x1024 detector window off centre 10 arcseconds in both x and y directions. We will also increase the exposure time to 60 seconds.

Now we are ready to obsere and readout the MICADO detector window

In [10]:

```
micado.cmds["!OBS.dit"] = 600     # [s]
micado.cmds["!DET.x"] = 2500      # [pixel] equivalent of 10 arcseconds (10 arcsec / 0.004 a
micado.cmds["!DET.y"] = 2500

micado.observe(cluster)
hdus = micado.readout()
```

```
Preparing 2 FieldOfViews
```

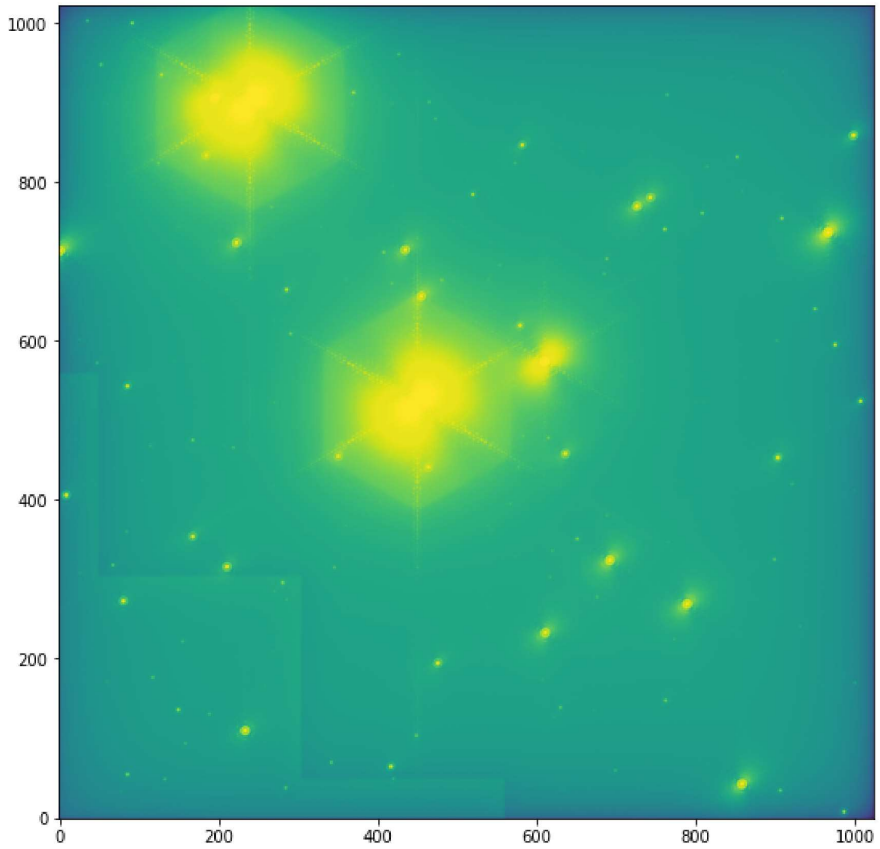
The elongation of the PSF along the radial axis is quite visible.

In [11]:

```
plt.figure(figsize=(20, 20))
plt.subplot(121)
im = hdus[0][1].data
plt.imshow(im, origin="lower", norm=LogNorm())
```

Out[11]:

```
<matplotlib.image.AxesImage at 0x2c21f68fbe0>
```



Alternatively we could re-centre the detector window, but expand the window size to 4096x4096 pixels (16x16 arcsecond) which is the equivalent of the central MICADO detector.

In [12]:

```
micado.cmds["!DET.x"] = 0
micado.cmds["!DET.y"] = 0
micado.cmds["!DET.width"] = 4096
micado.cmds["!DET.height"] = 4096

micado.observe(cluster)
hdus = micado.readout()
```

```
Preparing 2 FieldOfViews
```

It should be noted, the larger the detector size, the longer the simulation takes to run. Hence the two options.

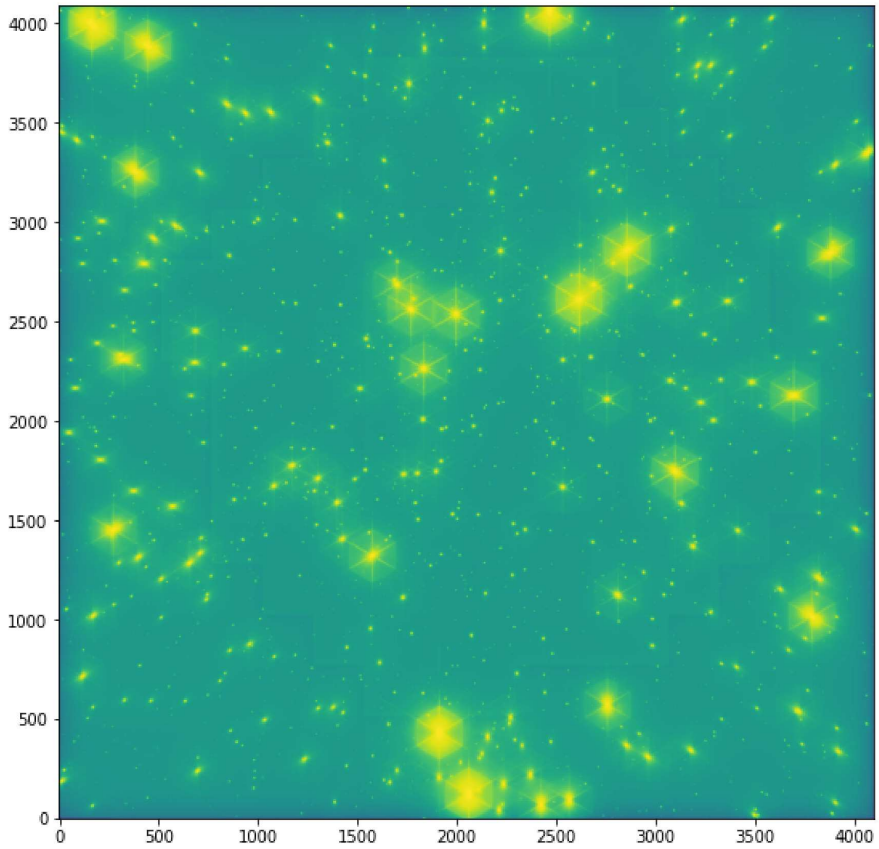
This option does however let us see the change in orientation of the PSF elongation around the on-axis position

In [13]:

```
plt.figure(figsize=(20, 20))
plt.subplot(121)
im = hdus[0][1].data
plt.imshow(im, origin="lower", norm=LogNorm())
```

Out[13]:

```
<matplotlib.image.AxesImage at 0x2c21fd10860>
```



Swap between a detector window and the full detector array

WARNING! It is ONLY recommended to run the full MICADO detector configuration is you have >8 GB of RAM

While the `MICADO_Sci` package only has a customisable detector window for simulated observations, the `MICADO` package contains a full description of the 9 detectors in the MICADO detector array.

This requires simulating a ~13000x13000 pixel field, which understandable takes a lot of computer resources and takes on the **order of 10 minutes to run.**

Ideally the user would write a script using the commands in this notebook and run that script in the background.

That said, swapping to the full detector array is a simple matter of setting the `.include` parameter to `True` and `False` for the two Detector options provided in the `MICADO` package (see `micado.effects` table above).

In [14]:

```
micado["full_detector_array"].include = True
micado["detector_window"].include = False
```

In this case, it is worth saving the final `HDUList` to disk as a multi-extension FITS file so the data are safe in case something happens to the Python session

In [15]:

```
# micado.observe(cluster)
# micado.readout(filename="micado_full_detector.fits")
```

Basic spectroscopy mode observations with MICADO_Sci

In this example we show how to use the basic spectroscopy mode provided with the `MICADO_Sci` package. The output of these spectroscopy simulations is a rectified image of the slit expanded along the wavelength axis. The x-axis is flux along the slit, the y-axis is wavelength.

Because this is a reduced version (for the sake of speed) of the spectroscopic mode, several of the expected effects, like non-parallel atmospheric background lines are not included. There is also no functionality for turning the slit relative to the parallactic angle. This functionality is available in the pipeline-oriented `MICADO` package. The main aim of `MICADO_Sci` is to enable the user to quickly iterate on their science use case.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from astropy import units as u

import scopesim as sim
import scopesim_templates as sim_tp

%matplotlib inline
%config Completer.use_jedi = False
```

As always, we must make sure we have the 3 instrument packages needed for SCAO osbservations

In [2]:

```
sim.server.database.download_package(["locations/Armazones.zip",
                                      "telescopes/ELT.zip",
                                      "instruments/MICADO_Sci.zip",])
```

Out[2]:

```
['F:\\temp\\scopesim_fdr_notebooks\\Armazones.zip',
 'F:\\temp\\scopesim_fdr_notebooks\\ELT.zip',
 'F:\\temp\\scopesim_fdr_notebooks\\MICADO_Sci.zip']
```

Set up the Source

For this basic example we will use the helper function `spiral_two_component` from the `scopesim_templates.basic.galaxy` submodule. This function creates a `Source` object with two intensity maps and two spectra. The two maps represent the distribution of the old (top) and new (bottom) stellar populations in the galaxy. The associated spectra are taken from the Brown et al (2014) galaxy spectra catalogue and are for an elliptical and a starburst galaxy.

For the purpose of illustrating how the spectroscopy mode works, this approximation will be sufficient.

Here we have selected to scale the galaxy images to 15 arcsec on a side and re-scale the spectra such that a pixel value of 1 corresponds to 15 magnitudes arcsec-2 in the AB system.

As the centre of the galaxy is not exactly in the centre of the iamge, we need to shift the `Source` 0.12 arcsec downwards so that the slit catched the galactic bulge

In [3]:

```
gal = sim_tp.basic.galaxy.spiral_two_component(extent=15*u.arcsec, fluxes=(15, 15)*u.ABmag)
gal.shift(dy=-30*0.004)
```

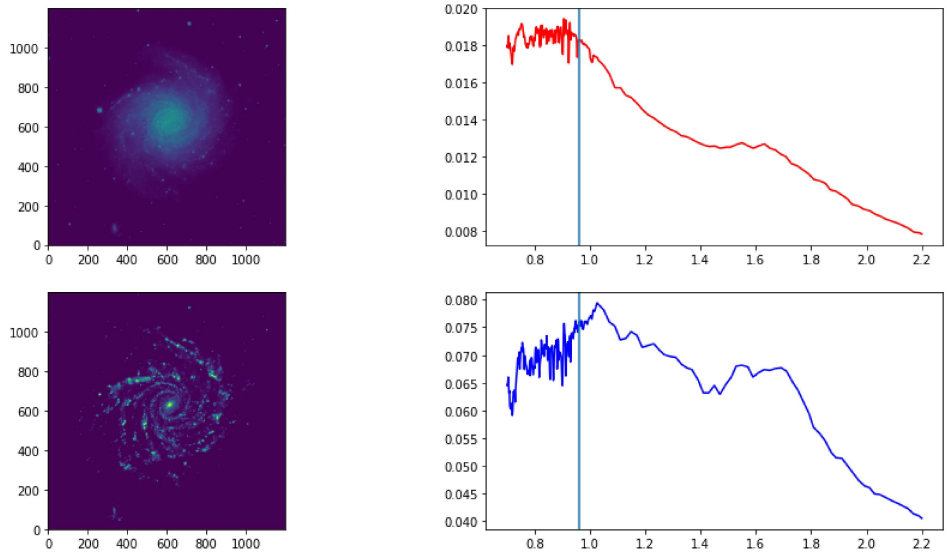
```
https://www.univie.ac.at/simcado/scopesim_templates/spiral_two_component.fit
s (https://www.univie.ac.at/simcado/scopesim_templates/spiral_two_component.
fits)
```

On the left we have the two intensity maps. On the right the spectrum associated with these maps.

The vertical line shows were we will set the wavelength that we will centre the observation around

In [38]:

```
wave = np.linspace(0.7, 2.2, 1001) * u.um

plt.figure(figsize=(15,8))
for i, c in zip([0, 1], "rb"):
    plt.subplot(2,2,2*i+1)
    plt.imshow(gal.fields[i].data, origin="lower")
    plt.subplot(2,2,2*i+2)
    plt.plot(wave, gal.spectra[i](wave), c=c)
    plt.axvline(0.96)
```



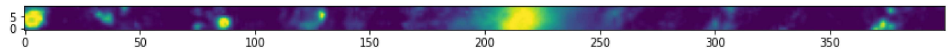
We can visualise approximately what portion of the galaxy that we want to come through the slit.

In [5]:

```
plt.figure(figsize=(15,3))
plt.imshow(gal.fields[1].data[625:635, 400:800], origin="lower")
```

Out[5]:

```
<matplotlib.image.AxesImage at 0x27af5e2a400>
```



Set up MICADO_Sci in SPEC mode

Now we follow the standard proceedure to set up the `MICADO_Sci` optical system using a `UserCommands` object. Before we initialise the optical model, we need to set a few parameters.

- `!SIM.spectral.wave_mid` : [um] sets the wavelength that the simulation will be centred on. In our case we're interested in the feature at 0.96um.
- `!DET.height` : [pixel] in this context will given the number of spectral bins. As the spectral resolution changes with wavelength, it doesn't make sense to specify a desired wavelength region. Instead the user can choose more or fewer spectral bins on the detector. For the sake of speed we will restrict this simulation to 1000 spectral bins.
- `!OBS.filter_name` : we can't forget to set the correct filter, otherwise no photons will get through
- `!OBS.dit` : [s] the exposure time

In [39]:

```
cmd = sim.UserCommands(use_instrument="MICADO_Sci", set_modes=["SCAO", "SPEC"])
cmd["!SIM.spectral.wave_mid"] = 0.96
cmd["!DET.height"] = 1000
cmd["!OBS.filter_name"] = "Spec_IJ"
cmd["!OBS.dit"] = 3600
cmd["!INST.psf.strehl"]= 0.2

micado = sim.OpticalTrain(cmd)
```

Standard process for observing and reading out the detector window

In [40]:

```
micado.observe(gal)
hdus = micado.readout()
```

```
Generated 999 headers from <SpectralTrace> "" : [0.75, 2.5]um : Ext 2 : Aper
ture 0 : ImagePlane 0
Preparing 999 FieldOfViews
```

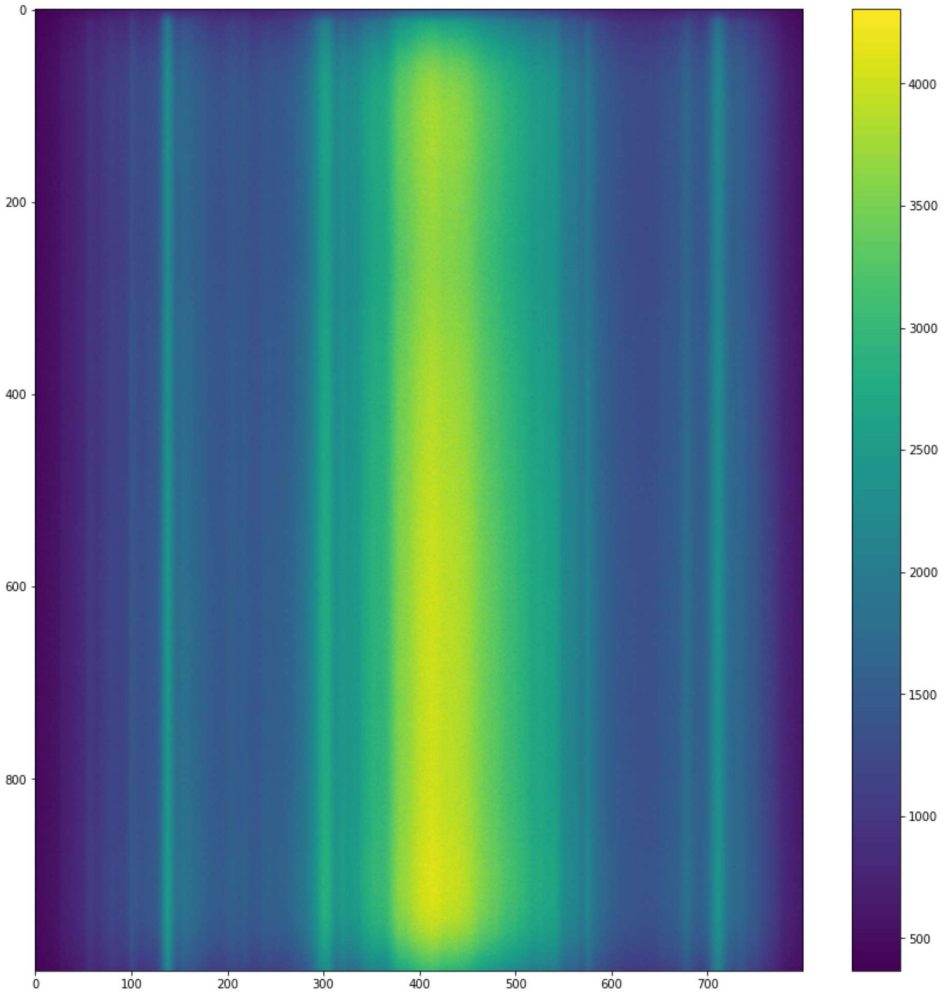
The figure below shows the detector window readout in spectroscopy mode. It's clear that the spectrum from the new population dominates. It is now up to the user to decide which regions of the spectrum need to be extracted for analysis. Below we show a very rudimentary way to extract 1D spectra from the image.

In [41]:

```
plt.figure(figsize=(15,15))
plt.imshow(hdus[0][1].data.T)
plt.colorbar()
```

Out[41]:

```
<matplotlib.colorbar.Colorbar at 0x27a835fbe80>
```



The above image doesn't tell us anything about the wavelength range we're covering, just as the raw MICADO detector readouts will not tell us anything about the observed spectrum.

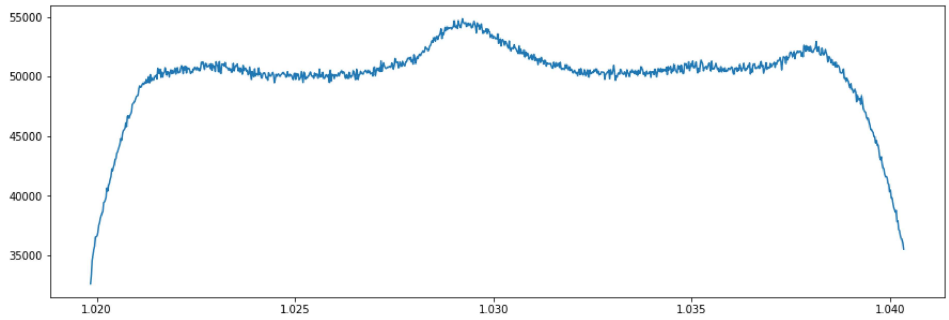
We can however extract the wavelength calibration from the MICADO `OpticalTrain` object by looking at how many spectral slices (FOV objects) were generated. This requires a bit Python Kung-Fu. Rest assured that this will be easier in later releases of ScopeSim.

In [36]:

```
waves = [fov.wavelength.value for fov in micado.fov_manager._fovs_list]
flux = hdus[0][1].data[10:, 400:410].sum(axis=1)

plt.figure(figsize=(15, 5))
plt.plot(waves[10:], flux)
```

Out[36]:

```
[<matplotlib.lines.Line2D at 0x27a939122e8>]
```



Full MICADO focal plane spectroscopic trace simulations

This example is a biggun'. And it will take **upwards of half an hour** to compute depending on your hardware. What we are doing here is using ScopeSim to project the full spectroscopic trace pattern of two point sources (a binary star system) after they have passed through the full optical system (atmosphere --> detector) onto the MICADO detector array.

This type of simulation is directed towards the team developing the data reduction pipeline. That said, there may be other users who would like to have a full readout of the MICADO detectors in this configuration. It should be noted that ScopeSim **does not** provide any kind of reduction functionality. It is therefore up to the user to reduce and analyse their own spectra derived from these simulations. A more user-friendly spectroscopy option is provided with the `MICADO_Sci` package (see the previous example notebook)

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from astropy import units as u

import scopesim as sim
import scopesim_templates as sim_tp

%matplotlib inline
%config Completer.use_jedi = False
```

The normal setup. This time using the `MICADO` package instead of the `MICADO_Sci` package.

In [2]:

```
sim.server.database.download_package(["locations/Armazones.zip",
                                      "telescopes/ELT.zip",
                                      "instruments/MICADO.zip"])
```

Out[2]:

```
['F:\\temp\\scopesim_fdr_notebooks\\Armazones.zip',
 'F:\\temp\\scopesim_fdr_notebooks\\ELT.zip',
 'F:\\temp\\scopesim_fdr_notebooks\\MICADO.zip']
```

Set up source

A simple object, the binary star system, can be created with the `stars` function in the `scopesim_templates.basic.stars` submodule. Our binary system consists of two stars with similar spectral types seperated by 200 mas. Each star is located 0.1 arcsec either side of the centre of the field of view. The stars are both 15 ABmags in the Ks filter.

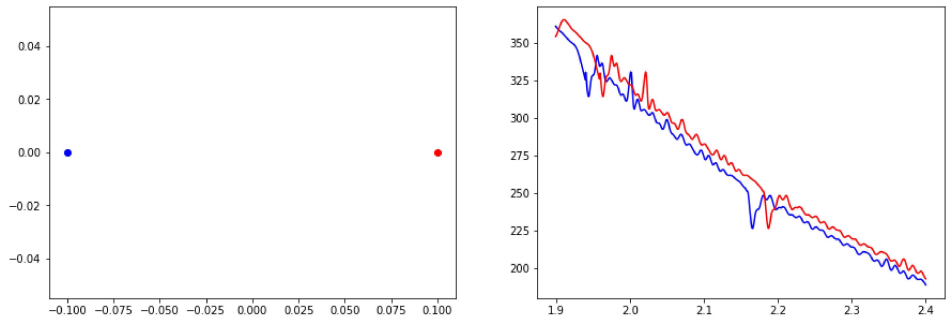
We redshift one of stars to induce a shift in the spectrum, that we will hopefully recover from the final simulated image. This is visible in the right plot below.

In [48]:

```
stars = sim_tp.basic.stars.stars(filter_name="Ks", amplitudes=[18,18]*u.ABmag,
                                 spec_types=["A0V", "A1V"], x=[-0.1, 0.1], y=[0, 0])
stars.spectra[1].z = 0.01
```

In [49]:

```
plt.figure(figsize=(15,5))
wave = np.linspace(1.9, 2.4, 1001) * u.um
for i, c in zip([0, 1], "br"):
    plt.subplot(121)
    plt.scatter(stars.fields[0]["x"][i], stars.fields[0]["y"][i], c=c)
    plt.subplot(122)
    plt.plot(wave, stars.spectra[i](wave), c=c)
```



Set up MICADO in SPEC mode, 3 arcsec slit and HK filter

There are three spectroscopy modes available in the `MICADO` package, each uses a different slit:

- `SPEC_3000x20` : Short and narrow, with a 20mas slit that is 3 arcsec long
- `SPEC_3000x50` : Short and fat, with a 50mas slit that is 3 arcsec long
- `SPEC_15000x50` : Long and fat, with a 50mas slit that is 15 arcsec long

For the sake of this example we will use the short and fat slit, or 3 arcsec by 50 mas.

The default filter in the spectroscopy modes is the `Spec_HK` filter. The other spectroscopy specific filter is names `Spec_IJ` . It is however possible to use any other filter in any of the three wheels. Different filters can be selected using the bang-string keywords: `"!OBS.filter_name_fw1"` for filter wheel 1, `"!OBS.filter_name_fw2"` for filter wheel 2, and `"!OBS.filter_name_pupil"` for the pupil wheel.

In [50]:

```
cmd = sim.UserCommands(use_instrument="MICADO", set_modes=["SCAO", "SPEC_3000x50"])
cmd["!OBS.filter_name_fw1"] = "Spec_HK"
cmd["!OBS.filter_name_fw2"] = "blank"
cmd["!OBS.filter_name_pupil"] = "blank"

cmd["!OBS.dit"] = 600     # [s]
```

Unfortunately the only way to see which filters are available is to either look in the instrument package folder in the working directory, or by initialising the `OpticalTrain` object and then looking at the contents of each filter wheel. This is done with the following command:

```
micado["filter_wheel_1"].data["names"]
```

for each of `filter_wheel_1`, `filter_wheel_2`, and `pupil_wheel`.

Once we've set which filter we want to use, we can generate the optical model of MICADO.

The purpose of this example is to show the trace pattern of MICADO's long slit spectrograph on the focal plane. We therefore need to turn off the detector window and turn on the full detector array. This is done be setting the `include` property of the relevant `Effect` objects to `True` or `False`.

To find the name of any `Effect` in the optical model, call:

```
micado.effects
```

In [51]:

```
micado = sim.OpticalTrain(cmd)

micado["full_detector_array"].include = True
micado["detector_window"].include = False
```

Finally we are set and can simulate the observation of out binary star system

WARNING: over 25000 spectral slices must be created and observed to run this simulation. Please expect this to take on the order of 30 minutes to complete (hardware dependent). The best strategy would be to create a script with the commands from this notebook and execute it in the background. Also, save as much of the simulation to disk as possible to avoid wasting execution time.

In [52]:

```
micado.observe(stars)
```

```
Generated 12624 headers from <SpectralTrace> "list of spectral order trace g
eometry on the focal plane" : [1.93, 2.46]um : Ext 2 : Aperture 0 : ImagePla
ne 0
Generated 12415 headers from <SpectralTrace> "list of spectral order trace g
eometry on the focal plane" : [1.45, 1.85]um : Ext 3 : Aperture 0 : ImagePla
ne 0
Generated 832 headers from <SpectralTrace> "list of spectral order trace geo
metry on the focal plane" : [1.16, 1.48]um : Ext 4 : Aperture 0 : ImagePlane
0
Preparing 25871 FieldOfViews
```

We can check that the `observe` command ran successfully by looking inside the `MICADO` optical model. The resulting normalised flux map in units of ph s-1 pixel-1 can be found in the first entry of the `image_planes` attribute of the optical model, in the form of a `fits.HDUList` object. If the size of the data image is ~12000x12000, then the simulation has been run successfully.

Note: this image is the "perfect" image of the source. It contains all contaminating flux (atmosphere, telescope greybody emission, etc) but has not had the poisson noise effects and detector read noise effects applied to it. It is still however useful to keep this image as a sanity check - i.e. that the expected flux levels are indeed seen at the detector plane.

We can save this intermediate image using the standard `fits.HDUList` functionality, i.e. the `writeto` method:

In [53]:

```
print(micado.image_planes[0].data.shape)
micado.image_planes[0].hdu.writeto("micado_spec_focal_plane_flux_map.fits", overwrite=True)
```

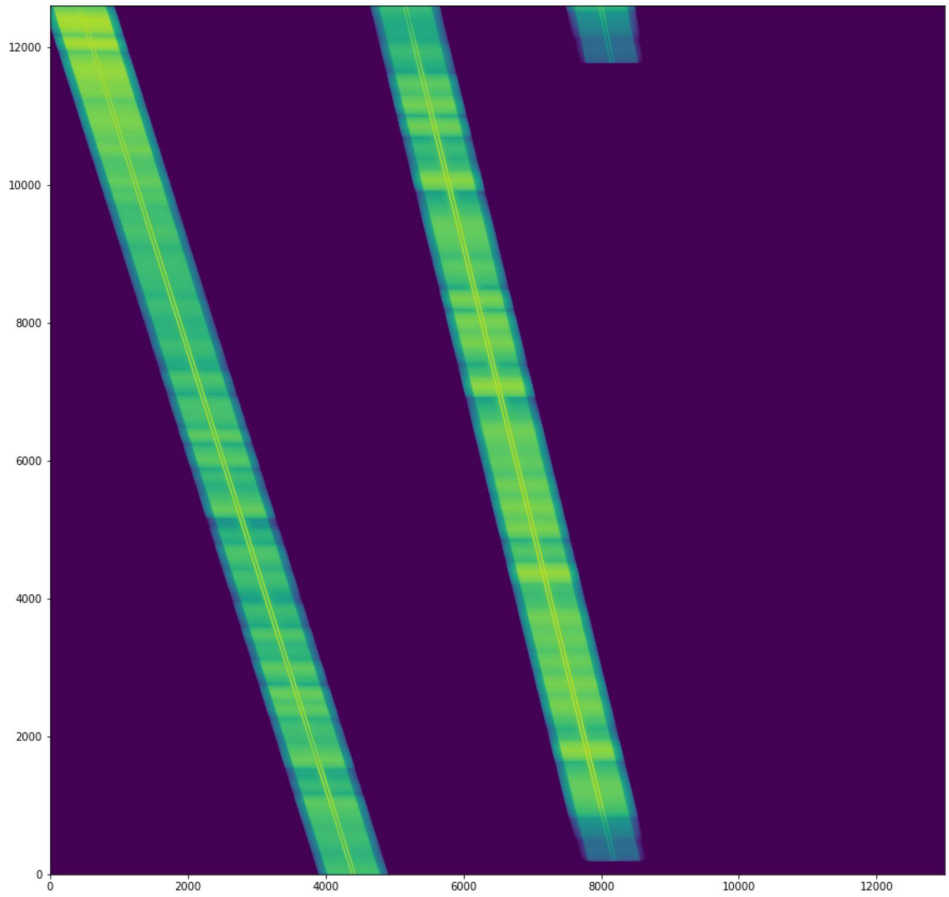
```
(12609, 12982)
```

In [54]:

```
im = micado.image_planes[0].data
im[im<1e-7] = 1e-7

plt.figure(figsize=(15, 15))
plt.imshow(im, origin="lower", norm=LogNorm())
```

Out[54]:

```
<matplotlib.image.AxesImage at 0x2dba8d79e10>
```



The above image shows the flux distribution of the three spectral traces for the `Spec_HK` filter over the full focal plane of MICADO. To convert these into individual detector read out frames, we still need to call `micado.readout`. The final output file will be a FITS file containing 10 `HDU`s. The Primary HDU (index 0)

contains meta data about the simulation. HDUs 1-9 contain the raw images from each detector in the MICADO detector array, including all known detector characterists (RON, ACN, Dark current, linearity effects, dead pixels, etc)

Don't forget to save the output image to disk.

In [55]:

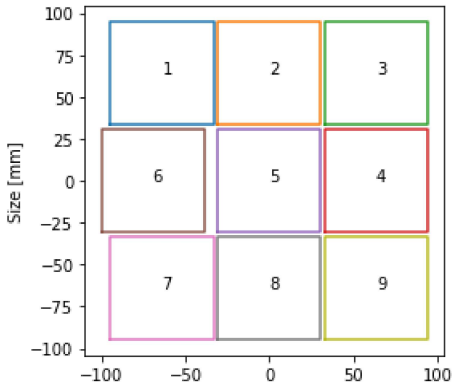
```
hdus = micado.readout(filename="micado_spec_detector_readout.fits", overwrite=True)
```

Becasue the MICADO detector array numbering scheme does not follow the matplotlib LRTB scheme, we cannot simply plot all the detector images in a standard `for` loop.

We can visualise the detector numbering scheme by calling the `plot` function of the `full_detector_array` `Effect` object:

In [56]:

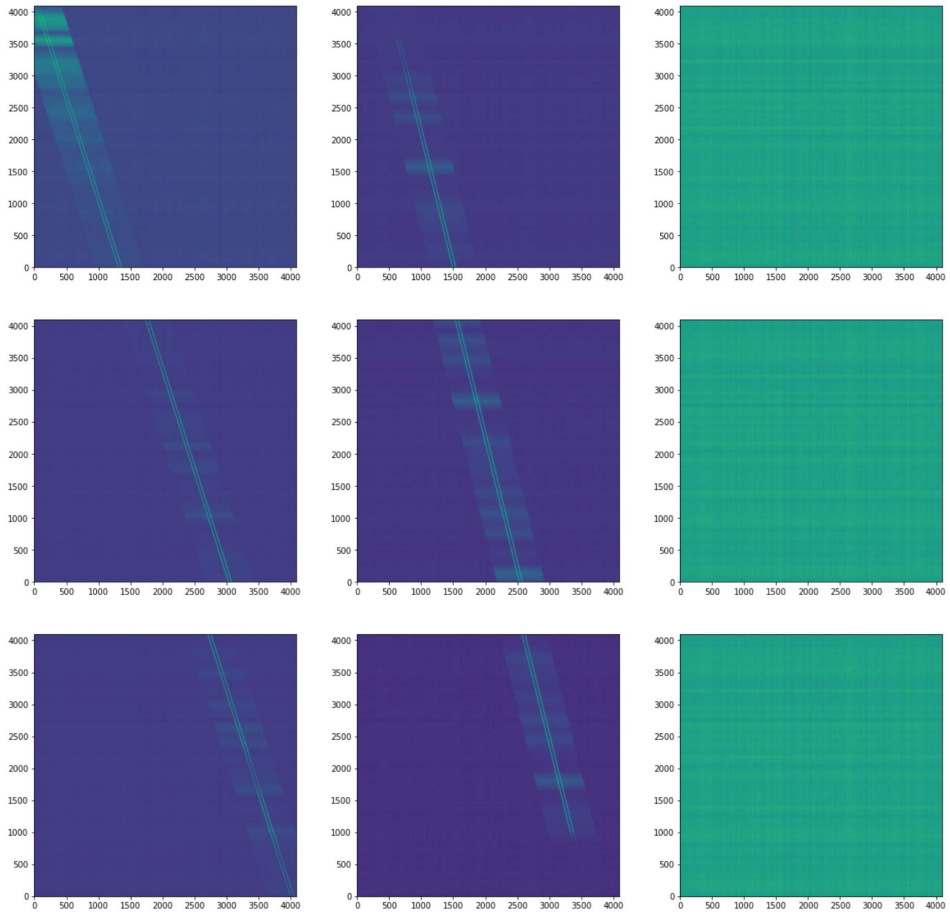
```
micado["full_detector_array"].plot()
plt.show()
```



Below we have the raw output images from the 9 MICADO detectors. Note the slight shift in the 6th detector. The right 3 detectors recieve no flux projected through the slit when the Spec_HK filter is used. The patterns visible on these detectors are due to the noise characteristics. They are identical because ScopeSim resets the numpy random seed for each operation. This reset operation could be turned off (`micado.cmds[!"SIM.random.seed"] = None`) in order to get more realistic noise proerties.

In [57]:

```
plt.figure(figsize=(20, 20))
for hdu_i, plot_i in zip([1,2,3,6,5,4,7,8,9], range(1, 10)):
    plt.subplot(3,3,plot_i)
    plt.imshow(hdus[0][hdu_i].data, origin="lower", norm=LogNorm())
```



The previous page was unintentionally left blank.