Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
INAF ISTITUTO NAZIONALE DI ASTROFISICA NATIONAL INSTITUTE FOR ASTROPHYSICS

Software dedicato sulla base degli algoritmi di ricostruzione della PSF (Point Spread Function), nell'ambito del progetto STILES – Strengthening the Italian leadership in ELT and SKA

MICADO PSFR
Pipeline Manual

2025-09-29
Vers. 1.0

WP 3302 Codice Identificativo: IR0000034; Codice Unico di Progetto C33C22000640006; presentata a seguito dell'Avviso Pubblico del 28 dicembre 2021, numero 3264 e ammesso a finanziamento nell'ambito degli "Interventi" previsti dalla "Missione 4", denominata "Istruzione e Ricerca", "Componente 2", denominata "Dalla Ricerca alla Impresa" ("M4C2"), "Linea di Investimento 3.1", denominata " Fondo per la realizzazione di un sistema integrato di infrastrutture di ricerca e innovazione ", del "Piano Nazionale di Ripresa e Resilienza" ("PNRR").
Programma Biennale degli acquisti 2023/2024 di INAF, CUI n. S97220210583202300048
CONTRATTO DI APPALTO
CUP: C33C22000640006; CIG B34864DEAC.

MICADO PSFR Pipeline Manual

v. 1.0

2025-09-29
TEIGA SRLS

1 - EDPS Workflow

1.1 - Overview

The workflow named **micado_psf_wkf** is shown in the following figures. The 5 tasks are:

1. scao_otf_elt
2. scao_otf_micado
3. scao_otf_par
4. scao_otf_perp
5. scao_final

Each task in (1. - 4.) has a main input (labelled by Raw Type) and a few secondary inputs (labelled by Static Calibrations by EDPS). The inputs of the last task (5.) are the outputs of the other tasks. There's not a significant difference among the four inputs, thus we choose the inputs in the appearance order.

Each task is associated with a single recipe (see Sect. 3 - Recipes), as shown in the following table.

Task	***Recipe***
scao_otf_elt	mcd_psfr_scao_otf_elt
scao_otf_micado	mcd_psfr_scao_otf_micado
scao_otf_par	mcd_psfr_scao_otf_par
scao_otf_perp	mcd_psfr_scao_otf_perp
scao_final	mcd_psfr_scao_final

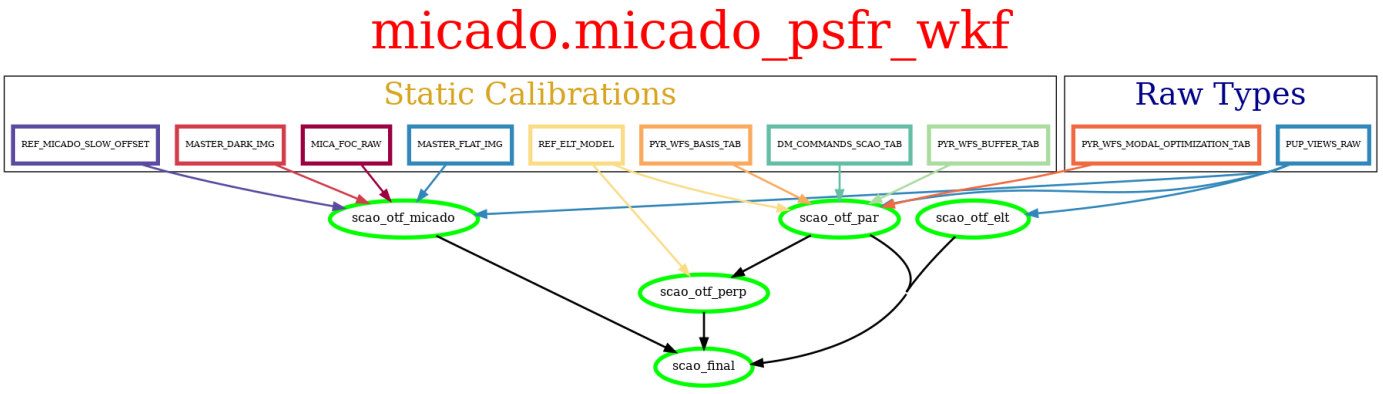


Fig. 1 - Micado Workflow obtained with the EPDS -g option. The Raw Types and Static Calibrations show the different kinds of inputs. The 5 tasks are shown in green and their connections are shown by black arrows.

micado.micado_psfr_wkf

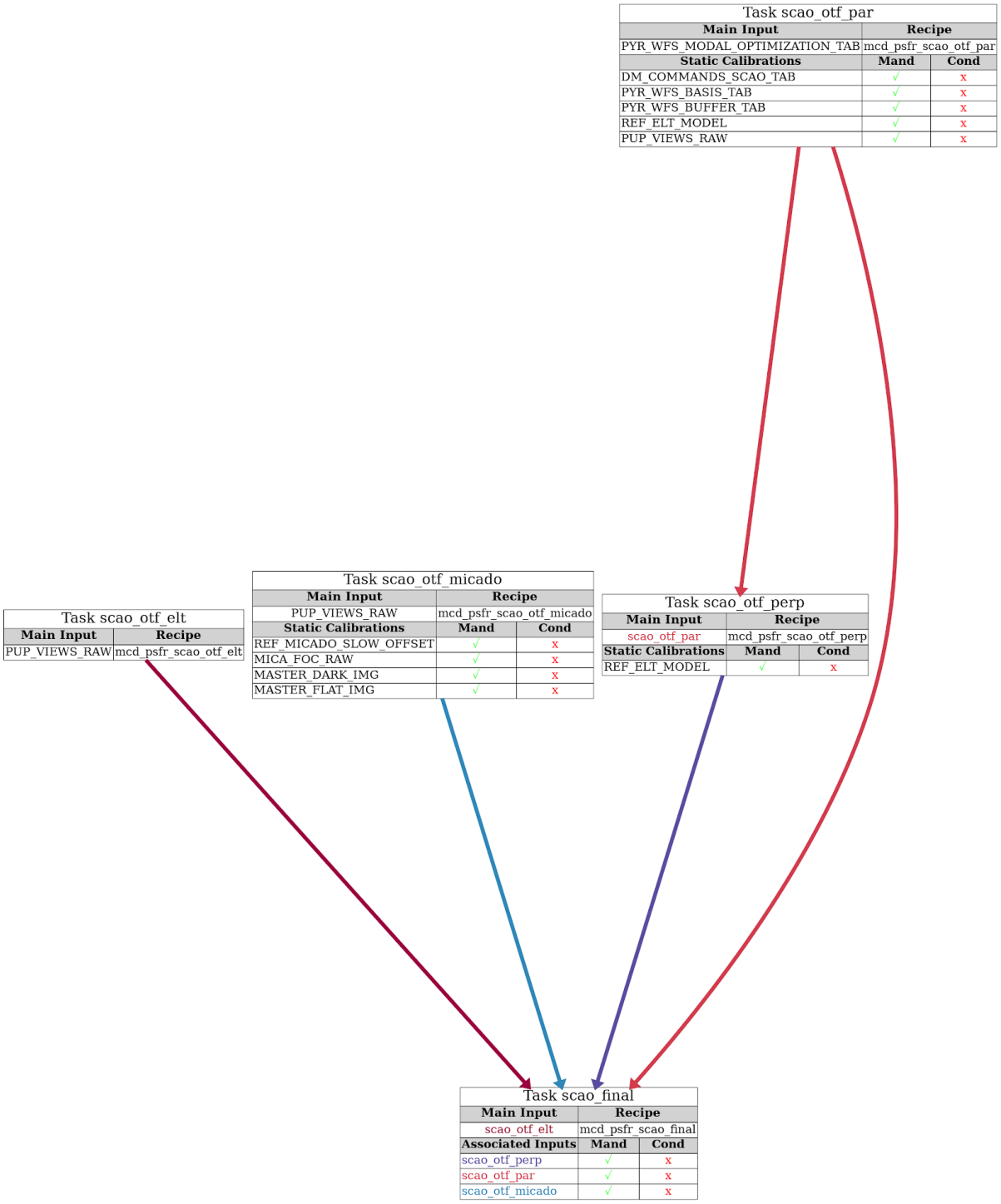


Fig. 2 - Micado Workflow obtained with the EPDS -g2 option. Each task is described by a table, in which the main input, the recipe and the Static Calibrations are shown. Each task is connected to the following one by a colored arrow.

The structure of the project must follow the standard EPDS, i.e. **<instrument>/<instrument>_wkf.py** and all the other files must be placed in the same folder. For our project, therefore we have:

```
edps/
├── workflow
    └── micado
        ├── micado_psfr_classification.py
        ├── micado_psfr_datasources.py
        ├── micado_psfr_keywords.py
        ├── micado_psfr_parameters.yaml
        ├── micado_psfr_rules.py
        ├── micado_psfr_wkf.py
```

1.2 - Workflow Tasks

In order to define a new task, we can use the object **task(t)**, where t is the name of the task, and the following methods:

- **with_main_input(d)** where d is the name of the datasource defined as the main input of the task itself (see next section).
- **with_associated_input(d)** where d is the name of the datasource defined as associated input. We can use more than one associate inputs if necessary
- **with_recipe(r)** where r is the name of the esorex recipe to be executed when the task is run.
- **build()**. This is the last method (must be placed at the end of the chain).

For example, the `scao_par_task` is defined as follows:

```
scao_par_task = (
    task("scao_otf_par")
    .with_main_input(pyr_wfs_modal_optimization_tab_datasource)
    .with_associated_input(dm_commands_scao_tab_datasource)
    .with_associated_input(pyr_wfs_basis_tab_datasource)
    .with_associated_input(pyr_wfs_buffer_tab_datasource)
    .with_associated_input(ref_elt_model_datasource)
    .with_associated_input(pupil_datasource)
    .with_recipe("mcd_psfr_scao_otf_par")
    .build()
)
```

1.3 - Workflow Datasources

In order to define a new datasource, we can use the object **data_source()** and the following methods:

- **with_classification_rule(c)**, where c is the name of the classification rule (see next section).

- **with_match_keywords(k)** where k is the keyword name (or a list of keywords) used for matching the datasources. In our case, the name of the instrument (defined by the keyword `kwd.instrume` defined in section 2.6) is used.
- **build()**. This is the last method (must be placed at the end of the chain).

For example, the `pupil_datasource` is defined as follows:

```
pupil_datasource = (
      data_source()
      .with_classification_rule(pupil_class)
      .with_match_keywords([kwd.instrume])
      .build()
)
```

1.4 - Workflow Classifications

In order to define a Classification rule, we can use the object **classification_rule(name, rule)**, where the first argument is the name of classification, and the last one is the rule to be used in the classification (see next section).
Note that the name of the classification is used in the graphs (see Fig. 1 and Fig. 2).

For example, the `pupil_class` is defined as follows:

```
pupil_class = classification_rule("PUP_VIEWS_RAW", rules.is_pupil)
```

1.5 - Workflow Rules

This section can be intended as a continuation of the previous section. For convenience, the rules are stored in a separated file.
In order to define a Rule, we can use a function (the so-called "complex" association, in the EPDS manual) that returns a boolean state: true if the rule is satisfied, false in the other case. The argument passed to that function is the file to associate.

For example, the `is_pupil` rule is defined as follows:

```
def is_pupil(f):
    return f[kwd.dpr_type] == "PUPIL" and is_image(f)
```

Remarks:

1. As we can see in the previous example, the "complex" association mode is more general than the default mode. Indeed we can combine different rules into a single function. In the example, we used the results of another rule (`is_image`) combined to the request that the DPR TYPE (that must be present in the header of the input FITS file) value must be equal to "PUPIL".
2. the rule `is_image` is defined as follows:

```
def is_image(f):
    return f[kwd.dpr_tech] == "IMAGE"
```

1.6 - Workflow Keywords

For convenience, each keyword used in the workflow can be stored in this file. When we import this module (`from . import micado_psfr_keywords as kwd`), we can refer to a single field with `kwd.<name>`.

The complete list of the keywords for micado is the following:

```
instrume = "instrume"
dpr_catg = "dpr.catg"
dpr_tech = "dpr.tech"
dpr_type = "dpr.type"
tpl_start = "tpl.start"
unique = "mjd-obs"
pro_catg = "pro.catg"
```

1.7 - Workflow and General Parameters

The parameters are stored in the so-called parameter file (in YAML format). The name of the file is `micado_psfr_parameters.yaml` and it contains a set (or more) of parameters. The name of the default parameter set must be "science_parameter".

The file can be divided in two parts: the first one lists the static parameters (i.e. used in the workflow), the last one is used for the recipe parameters.

For each task, we can list the task/recipe parameters in the `recipe_parameters` section. Each parameter name should be `prefix.name` where the prefix is the name of the recipe (not to be confused with the name of the task) and the name is the name of the parameter itself.

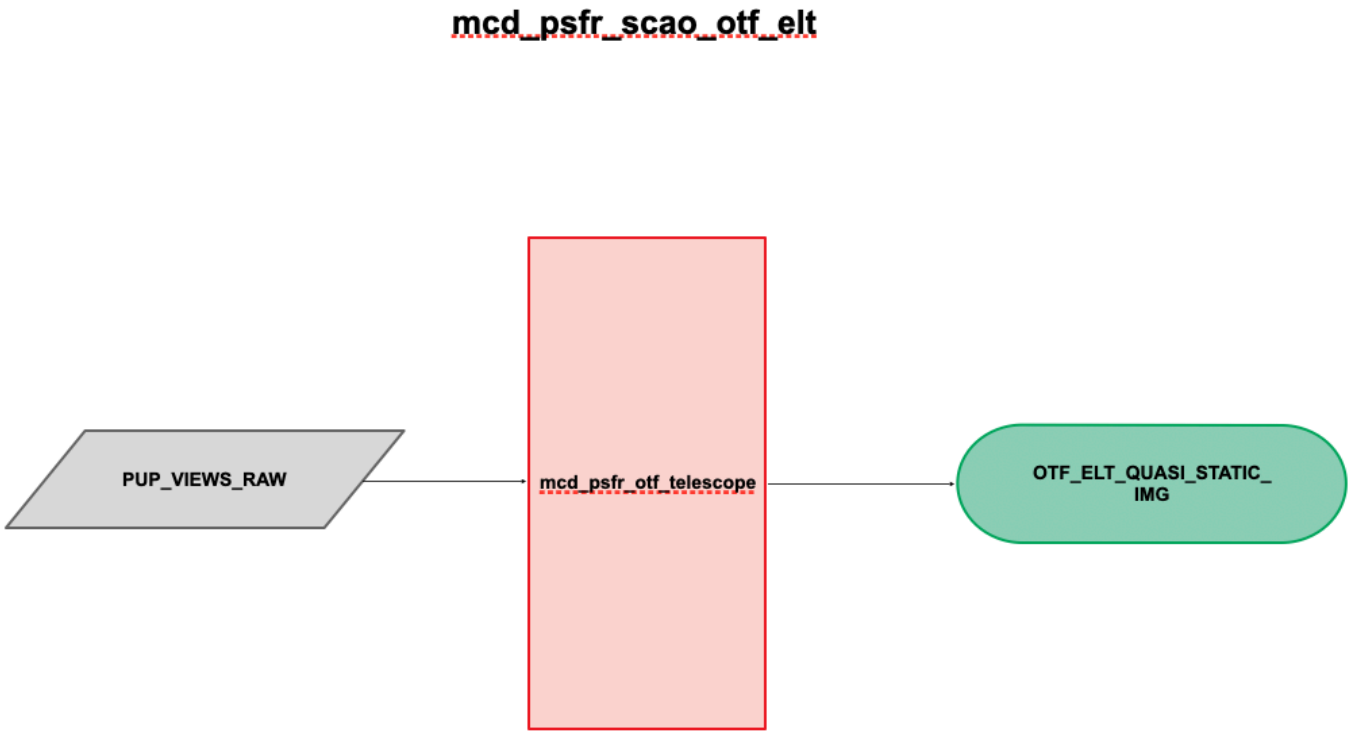
For example:

```
science_parameters:
  is_default: yes
  recipe_parameters:
    scao_otf_par:
      mcd_psfr_scao_otf_par.kappa: 50
      mcd_psfr_scao_otf_par.L0: 1e6
      mcd_psfr_scao_otf_par.r0_500nm: 0.1
    scao_otf_perp:
      mcd_psfr_scao_otf_perp.tel_diam: 38.542
      mcd_psfr_scao_otf_perp.lambda_sci: 2.2e-6
      mcd_psfr_scao_otf_perp.n_calc: 411
      mcd_psfr_scao_otf_perp.actspacing: 0.54
    scao_final:
      mcd_psfr_scao_final.pixel_scale: 4.0
      mcd_psfr_scao_final.tel_diam: 38.542
      mcd_psfr_scao_final.lambda_sci: 2.2e-6
      mcd_psfr_scao_final.psf_size: 4096
```

2. Recipes

2.1 - MCD_PSFR_SCAO_OTF_ELT

Implemented scheme



Mandatory inputs

Variable	Input from TAG[1] / header[2] / filename[3]	Comments
pupil	TAG= **PUP_VIEWS_RAW** DPR.CATG= CALIB DPR.TYPE= PUPIL DPR.TECH= IMAGE (filename ifs_pupil.fits)	Remarks 1 and 2.

[1] TAG is the classification in the SOF (set-of-frames) passed by EDPS / esorex to the recipe. Remark 3.
[2] Header contains the cards to be set (commonly DPR.TYPE) into the header of the FITS file. Remark 3.
[3] Filename is not important, since the classification made by EDPS does not search for the name of the files, but only for the contents of the FITS header. The filename, when provided, is only a reference for sake of clarity.

Outputs

Variable	Output TAG[4] / filename	Comments
otf_quasi_static_elt_img	**OTF_ELT_QUASI_STATIC_IMG** (filename as tag)	Saved by cpl_dfs_save_image. Only the image is saved, not the domain information. This is a final output (green in scheme)

Recipe algorithm

1. Read the PIXSCALE value from the header of the input FITS file
2. Read the input file and store the data image to a gridfunction data structure (pupil).
3. Call **mcd_psfr_otf_telescope** function and compute the otf_quasi_static_elt_img gridfunction.
4. Define a new header (cpl_propertylist) and set a card with the output tag OTF_ELT_QUASI_STATIC_IMG
5. Save the variable otf_quasi_static_elt_img to a FITS file (as a final product) by cpl_dfs_save_image.

Remarks

1. The suggested pupil (called pupil822.fits) causes errors in the anisoplanatic phase of the recipe mcd_psfr_scao_otf_par. On the contrary, the code runs until the end when the mentioned pupil (ifs_pupil.fits) is provided. This issue is under investigation.
2. In the header of the FITS file, the card "PIXELSCALE" must be also present. The input file (remark1) already has this card (the value is ~0.26). If another pupil is provided, it is mandatory to set the value before launching the recipe. See Input Data section for details.
3. Please refer to [1], Tables 5 and 6, pag. 56, where classification of Raw data and external data are defined. In particular, in column 1 the name of the TAGs (in the sense of SOF list, EDPS classification) is given, while in column 3 the DPR.TYPE (stored in the FITS header cards) is given.

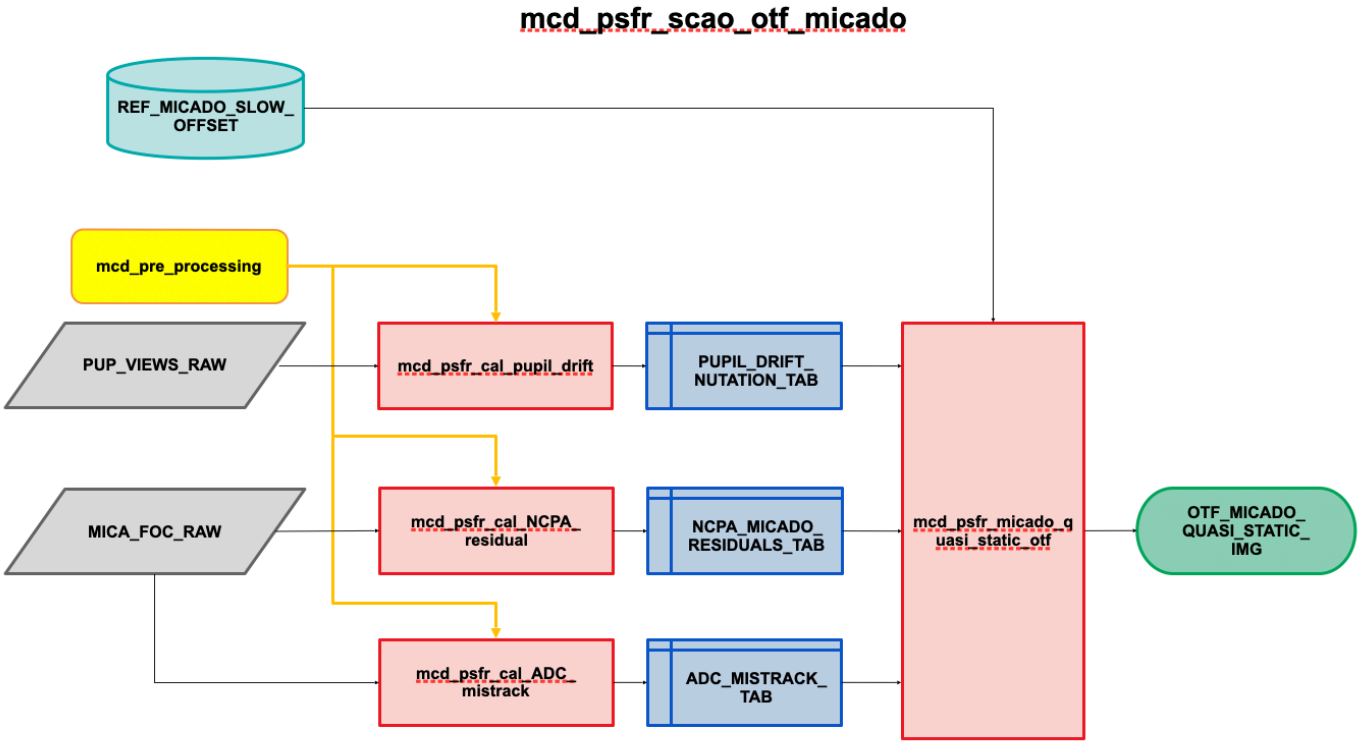
Input Data

Concerning PUP_VIEWS_RAW, please refer to documentation provided with the **mcd_psfr_scao_otf_par** recipe. The dataset and the necessary python script are described at the end of the document.

[4] Output tag is the name of PRO.CATG saved into the header of the FITS file. The same name is used for TAGging outputs in the SOF file.

2.2 - MCD_PSFR_SCAO_OTF_MICADO

Implemented scheme



Mandatory inputs

Variable	Input from TAG[5] / header[6] / filename[7]	Comments
pupil	TAG= **PUP_VIEWS_RAW** DPR.CATG= CALIB DPR.TYPE= PUPIL DPR.TECH= IMAGE (filename ifs_pupil.fits)	Remarks 1 and 2.
mica_foc	TAG= **MICA_FOC_RAW** DPR.CATG= CALIB DPR.TYPE= FOCALPLANE DPR.TECH= IMAGE (filename MICA_FOC_RAW.fits)	
master_dark	TAG= **MASTER_DARK_IMG** PRO.CATG=MASTER_DARK_IMG	mcd_pre_processing

[5] TAG is the classification in the SOF (set-of-frames) passed by EDPS / esorex to the recipe. Remark 3.
[6] Header contains the cards to be set (commonly DPR.TYPE) into the header of the FITS file. Remark 3.
[7] Filename is not important, since the classification made by EDPS does not search for the name of the files, but only for the contents of the FITS header. The filename, when provided, is only a reference for sake of clarity.

master_flat	TAG= **MASTER_FLAT_IMG** PRO.CATG=MASTER_FLAT_IMG	mcd_pre_processing
offset	TAG= **REF_MICADO_SLOW_OFFSET** DPR.CATG= CALIB DPR.TYPE= OFFSET DPR.TECH= TABLE REF_MICADO_SLOW_OFFSET.fits	

Outputs

Variable	**Output TAG[8] / filename**	**Comments**
quasi_static	**OTF_MICADO_QUASI_STATIC_IMG** (filename as tag)	Saved by cpl_dfs_save_image. Only the image is saved, not the domain information. This is a final output (green in scheme)

Recipe algorithm

1. Read the PIXSCALE value from the header of the input FITS file
2. Read the input file and store the data image to a cpl_image data structure (pupil).
3. Read all the other inputs from frameset. This part of the code is commented since no information is available for the current version of the recipe.
4. Call **mcd_psfr_cal_pupil_drift** function. (currently, this function is empty).
5. Save PUPIL_DRIFT_NUTATION_TAB as a FITS file table. (currently, this cannot be done since there is no data - This part is commented on the code).
6. Call **mcd_psfr_cal_NCPA_residual** function. (currently, this function is empty).
7. Save NCPA_MICADO_RESIDUALS_TAB as a FITS file table. (currently, this cannot be done since there is no data - This part is commented on the code).
8. Call **mcd_psfr_cal_ADC_mistrack** function. (currently, this function is empty).
9. Save ADC_MISTRACK_TAB as a FITS file table. (currently, this cannot be done since there is no data - This part is commented on the code).
10. Call **mcd_psfr_micado_quasi_static_otf** function. (currently, this function is empty).
11. Set a temporary output data, defined by an image NxN filled with 1.0 (double value). N is computed as (2*pupil_size-1) (the size of the pupil read at the beginning of the recipe). THIS STEP WILL BE REMOVED ONCE THE FUNCTION DESCRIBED IN STEP 4-7 WILL BE AVAILABLE.

[8] Output tag is the name of PRO.CATG saved into the header of the FITS file. The same name is used for TAGging outputs in the SOF file.

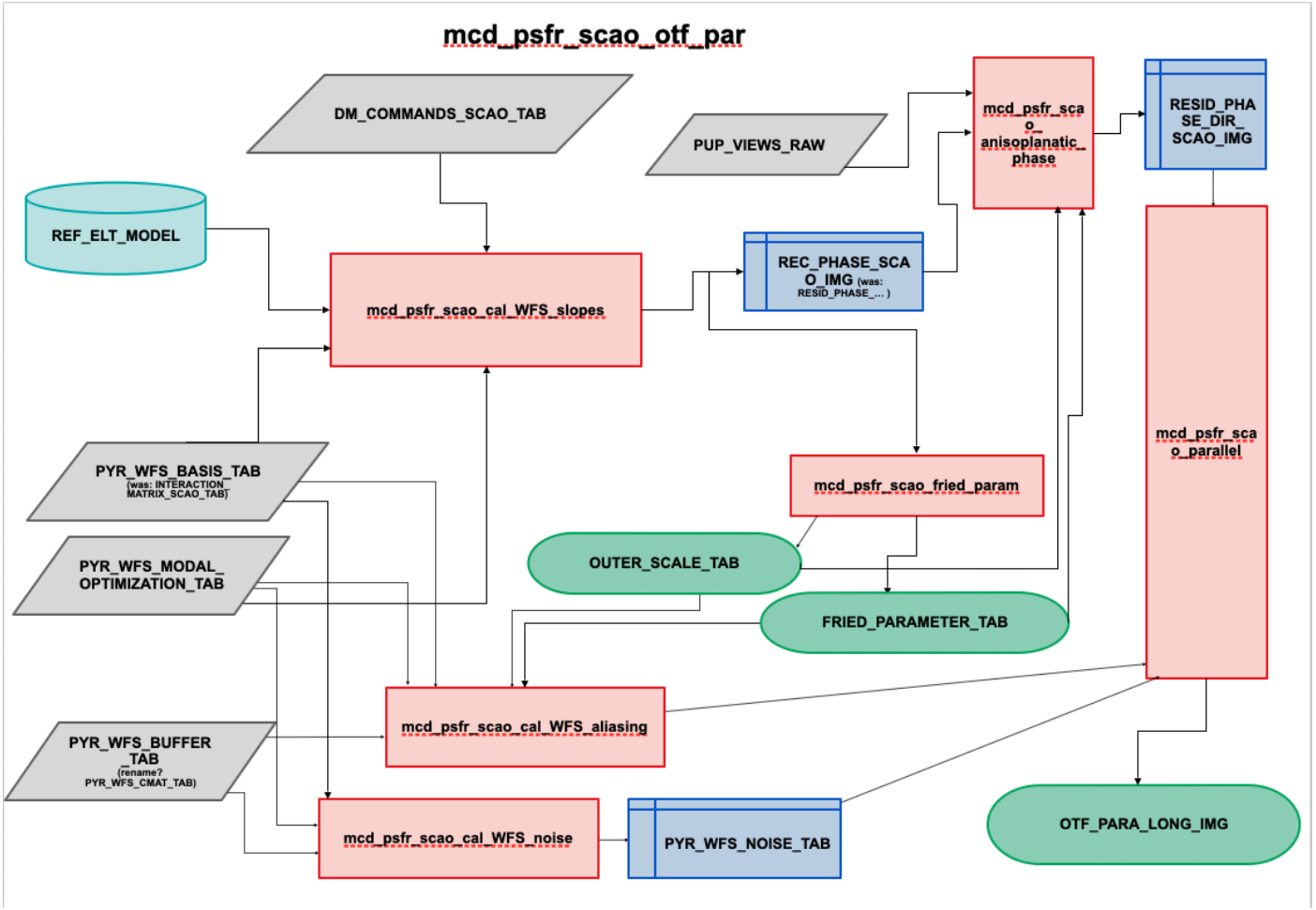
12. Define a new header (cpl_propertylist) and set a card with the output tag OTF_MICADO_QUASI_STATIC_IMG
13. Save the result to a FITS file (as a final product) by cpl_dfs_save_image.

Remarks

1. The suggested pupil (called pupil822.fits) causes errors in the anisoplanatic phase of the recipe mcd_psfr_scao_otf_par. On the contrary, the code runs until the end when the mentioned pupil (ifs_pupil.fits) is provided. This issue is under investigation.
2. In the header of the FITS file, the card "PIXELSCALE" must be also present. The input file (remark1) already has this card (the value is ~0.26). If another pupil is provided, it is mandatory to set the value before launching the recipe. For example, you can use the python code described in the mcd_psfr_scao_otf_par and copied here below for convenience.
3. Please refer to [1], Tables 5 and 6, pag. 56, where classification of Raw data and external data are defined. In particular, in column 1 the name of the TAGs (in the sense of SOF list, EDPS classification) is given, while in column 3 the DPR.TYPE (stored in the FITS header cards) is given.

2.3 - MCD_PSFR_SCAO_OTF_PAR

Implemented scheme



Mandatory inputs

Variable	Input from TAG[9] / header[10] / filename[11]	Comments
params->pupil	TAG= **PUP_VIEWS_RAW** DPR.CATG= CALIB DPR.TYPE= PUPIL DPR.TECH= IMAGE (filename ifs_pupil.fits)	Remarks 2 and 3.
params->dm_commands	TAG= **DM_COMMANDS_SCAO_TAB** DPR.CATG= CALIB DPR.TYPE= DMCOMMAND DPR.TECH= TABLE (filename volts.fits)	
params->influence_functions	TAG= **REF_ELT_MODEL** DPR.CATG= CALIB	(*) DPR.TYPE is not set in [1], we set it

[9] TAG is the classification in the SOF (set-of-frames) passed by EDPS / esorex to the recipe. Remark 4.
[10] Header contains the cards to be set (commonly DPR.TYPE) into the header of the FITS file. Remark 4.
[11] Filename is not important, since the classification made by EDPS does not search for the name of the files, but only for the contents of the FITS header. The filename, when provided, is only a reference for sake of clarity.

	DPR.TYPE= REF_ELT_MODEL (*) DPR.TECH= TABLE (filename cropped_M4IF.fits)	here for the first time.
params->basis	TAG= **PYR_WFS_BASIS_TAB** DPR.CATG= CALIB DPR.TYPE= BASISMATRIX DPR.TECH= TABLE (filename basis.tab)	The original name was INTERACTION_MATRIX_SCAO_TAB The original dpr.type was INTMATRIX.
params->modal_gains	TAG= **PYR_WFS_MODAL_OPTIMIZATION_TAB** DPR.CATG= CALIB DPR.TYPE= MODAL_OPT DPR.TECH= TABLE (filename modal_gains.fits)	
params->control_matrix	TAG= **PYR_WFS_BUFFER_TAB** DPR.CATG= CALIB DPR.TYPE= CONTMATRIX DPR.TECH= TABLE (filename cmat.fits)	

Recipe parameters

Variable	**Description**	**Comments**
params->r0_500nm	Read from EDPS parameters. Set into YAML file or pass it via `-rp scao_otf_par mcd_psfr_scao_par.r0_500nm VALUE`	default (hard-coded) = 0.144
params->L0	Read from EDPS parameters. Set into YAML file or pass it via `-rp scao_otf_par mcd_psfr_scao_par.L0 VALUE`	default (hard-coded) = 1e6
params->anisoplanatic.kappa	Read from EDPS parameters. Set into YAML file or pass it via `-rp scao_otf_par mcd_psfr_scao_par.kappa VALUE`	default (hard-coded) = 50

Outputs

Variable	Output TAG[12] / filename	Comments
parameters->dm_shape	**REC_PHASE_SCAO_IMG** (filename as tag)	FITS image. Intermediate file (blue in scheme)
otf->noise	**PYR_WFS_NOISE_TAB** (filename as tag)	FITS image. Intermediate file (blue in scheme)
otf->residual	**RESID_PHASE_DIR_SCAO_IMG** (filename as tag)	FITS image. Intermediate file (blue in scheme)
parameters->L0	**OUTER_SCALE_TAB** (filename as tag)	FITS Table (column name = "L0") This is a final output (green in scheme)
parameters->r0_500nm	**FRIED_PARAMETER_TAB** (filename as tag)	FITS Table (column name = "r0_500nm") This is a final output (green in scheme)
otf->parallel	**OTF_PARA_LONG_IMG** (filename as tag)	Saved by cpl_dfs_save_image. Only the image is saved, not the domain information. This is a final output (green in scheme)

Recipe algorithm

1. Define a new set of parameters calling mcd_psfr_parameters_new() function.
2. Read parameters passed via EDPS / ESOREX call. Here the r0_500nm value can be overwritten (if passed).
3. Read all the input files calling **mcd_psfr_parameters_from_frameset**. Here all the inputs provided via frameset are read and stored in the parameters data structure.
4. Compute the otf->telescope calling the mcd_psfr_otf_telescope function and passing the read pupil (see input table). The resulting image is also saved for debug purposes.
5. Call the **mcd_psfr_scao_cal_WFS_slopes** function.
6. Save parameters->dm_shape to RESID_PHASE_MODAL_SCAO_IMG.fits (as an intermediate fits image).
7. Call **mcd_psfr_scao_fried_param** (currently, this function is empty).

[12] Output tag is the name of the PRO.CATG card saved into the header of the FITS file. The same name is used for TAGging outputs in the SOF file.

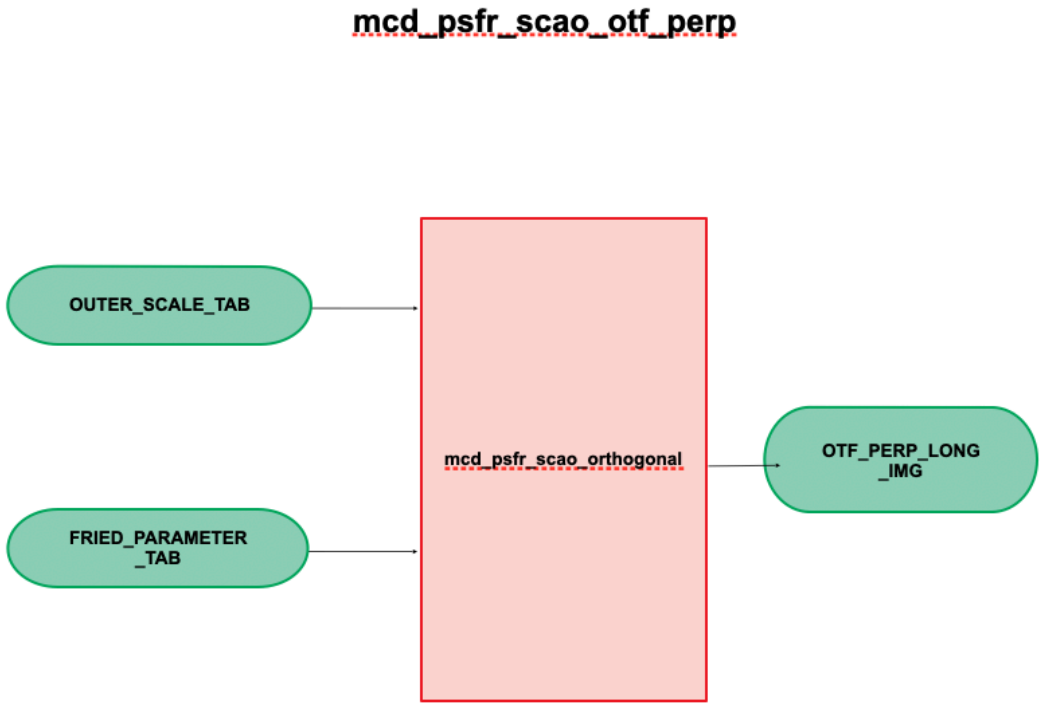
8. Save the value of parameters->L0 to OUTER_SCALE_TAB.fits (as a final fits table). Here the name of the unique column is "L0" (case sensitive!).
9. Save the value of parameters->r0_500nm to FRIED_PARAMETER_TAB.fits (as a final fits table). Here the name of the unique column is "r0_500nm" (case sensitive!).
10. Call **mcd_psfr_scao_cal_WFS_aliasing** function. (currently, this function is empty).
11. The otf->alias is not saved (but the corresponding code is present in the code and commented).
12. Call **mcd_psfr_scao_cal_WFS_noise** function.
13. Save otf->noise to PYR_WFS_NOISE_TAB.fits
14. Call **mcd_psfr_scao_anisoplanatic_phase** function.
15. Save otf->residual imagelist (i.e. a list of images, or in other words a 3D array) to REC_PHASE_SCAO_IMG.fits (as an intermediate fits image).
16. Call **mcd_psfr_scao_parallel** function.
17. Define a new header (cpl_propertylist) and set a card with the output tag OTF_PARA_LONG_IMG
18. Save the result otf->parallel to a FITS file (as a final product) by cpl_dfs_save_image.

Remarks:

1. All the inputs, both *_IMG and *_TAB, are stored in FITS files containing images and not tables. This is known, and the code is able to read the data via cpl_image_load function. Even if all inputs are images, both the TAG (written in the header of the corresponding fits file) and the EDPS workflow refer to these inputs in the "foreseen" way.
2. The suggested pupil (called pupil822.fits) causes errors in the `mcd_psfr_scao_parallel` function. On the contrary, the code runs until the end when the mentioned pupil (ifs_pupil.fits) is provided. This issue is under investigation.
3. In the header of the FITS file, the card "PIXSCALE" must be also present. The input file (remark.2) already has this card (the value is ~0.26). If another pupil is provided, it is mandatory to set the value before launching the recipe. For example, you can use the python code described here below.
4. Please refer to [1], Tables 5 and 6, pag. 56, where classification of Raw data and external data are defined. In particular, in column 1 the name of the TAGs (in the sense of SOF list, EDPS classification) is given, while in column 3 the DPR.TYPE (stored in the FITS header cards) is given.
5. Execution time is ~4.5h on Virtual Machine with UBUNTU 22.04 with 16GB RAM, 16GB Swap (hardware: MacBookPro with M4 PRO processor, 24GB of RAM). The time is reduced to ~26 min if we take into account only the first 20 frames in DM_COMMANDS_SCAO_TAB and PYR_WFS_MODAL_OPTIMIZATION_TAB input files (and setting kappa parameter = 10).

2.4 - MCD_PSFR_SCAO_OTF_PERP

Implemented scheme



Mandatory inputs

Variable	Input from TAG[13] / header[14] / filename[15]	Comments
params->r0_500nm	TAG= **FRIED_PARAMETER_TAB** PRO.CATG=FRIED_PARAMETER_TAB PRO.TECH= TABLE (filename as tag)	Read the first col and the first row of the table, get the double value and store in parameters data structure.
params->L0	TAG= **OUTER_SCALE_TAB** PRO.CATG=OUTER_SCALE_TAB PRO.TECH= TABLE (filename as tag)	Read the first col and the first row of the table, get the double value and store in parameters data structure.

[13] TAG is the classification in the SOF (set-of-frames) passed by EDPS / esorex to the recipe. Remark 6.
[14] Header is the card to be set (commonly DPR.TYPE) into the header of the FITS file. Remark 6.
[15] Filename is not important, since the classification made by EDPS does not pay attention to the name of the files, but only to the contents of the FITS header. The filename, when provided, is only a reference for sake of clarity.

Recipe parameters

Variable	Description	Comments
params->tel_diam	Read from EDPS parameters. Set into YAML file or pass it via `-rp scao_otf_perp mcd_psfr_scao_perp.tel_diam VALUE` option	default (hard-coded) value is 38.542
params->actspacing	Read from EDPS parameters. Set into YAML file or pass it via `-rp scao_otf_perp mcd_psfr_scao_perp.actspacing VALUE` option	default (hard-coded) = 0.54
params->n_calc	Read from EDPS parameters. Set into YAML file or pass it via `-rp scao_otf_perp mcd_psfr_scao_perp.n_calc VALUE`	default (hard-coded) = 313
params->lambda_sci	Read from EDPS parameters. Set into YAML file or pass it via `-rp scao_otf_perp mcd_psfr_scao_perp.lambda_sci VALUE`	default (hard-coded) = 2.2e-6

Outputs

Variable	Output TAG[16] / filename	Comments
otf->perp	**OTF_PERP_LONG_IMG** (filename as tag)	Saved by cpl_dfs_save_image. Only the image is saved, not the domain information. This is a final output (green in scheme)

Recipe algorithm

1. Define a new set of parameters calling mcd_psfr_parameters_new() function.
2. Read parameters passed via EDPS / ESOREX call.
3. Call the **mcd_psfr_scao_orthogonal** function.
4. Define a new header (cpl_propertylist) and set a card with the output tag OTF_PERP_LONG_IMG
5. Save the result otf->perp to a FITS file (as a final product) by cpl_dfs_save_image.

[16] Output tag is the name of PRO.CATG saved into the header of the FITS file. The same name is used for TAGging outputs in the SOF file.

Remarks

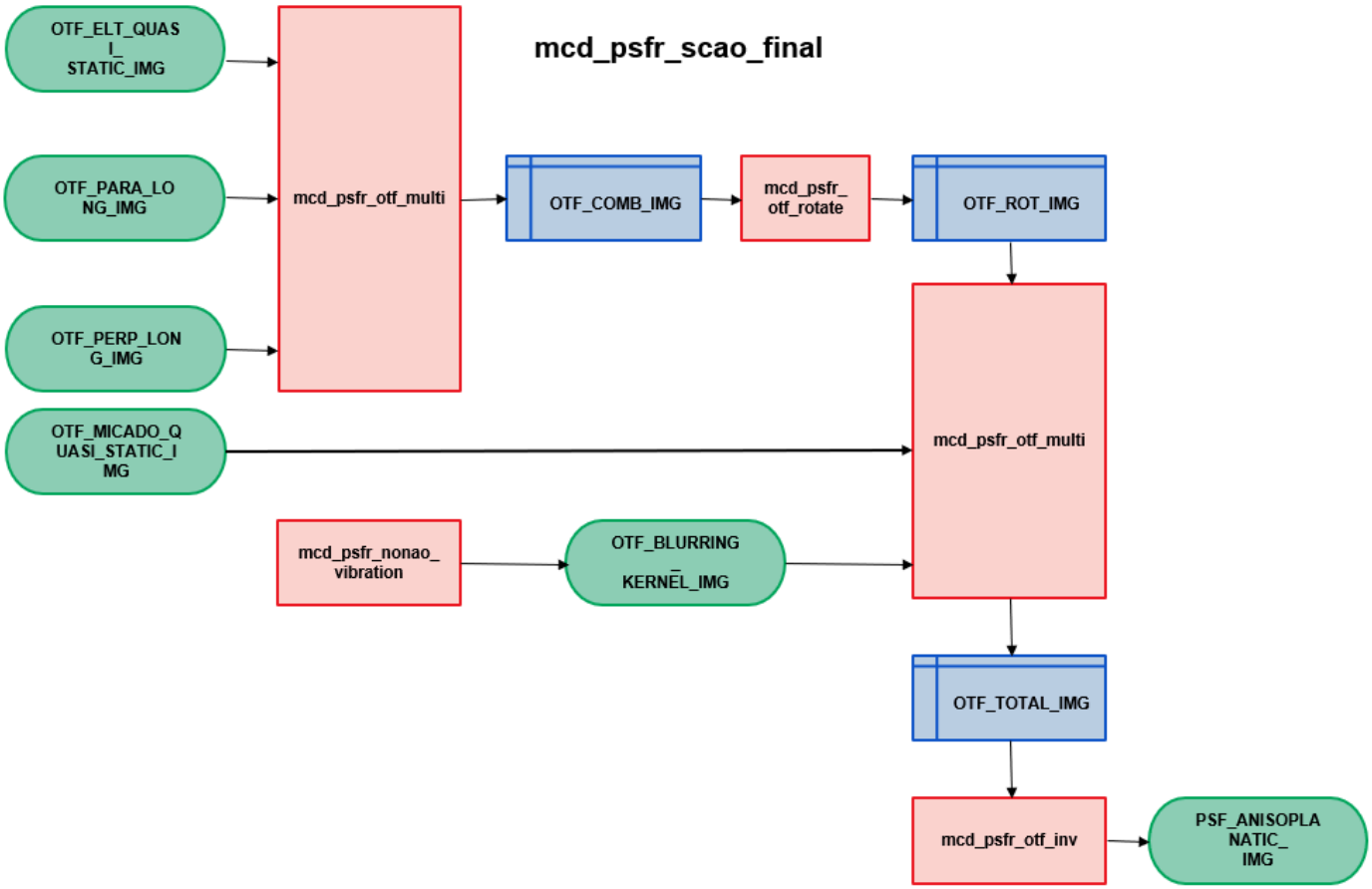
1. FRIED_PARAMETER_TAB has a unique column called "r0_500nm" (case sensitive!). At the present day, only the first row of the table contains information about the variable (double).
2. OUTER_SCALE_TAB has a unique column called "L0" (case sensitive!). At the present day, only the first row of the table contains information about the variable (double).
3. In order to reproduce Roland's data (provided for comparison) the following parameters must be set:
    a. n_calc = **411** (instead of default 313)
    b. r0_500nm = **0.1** (in the **mcd_psfr_scao_otf_par** recipe, see the documentation for more details).
4. The value of params->fft_size is computed by the following equation:
   ```
   params->fft_size = mcd_psfr_utility_next_power_of_two(2 * params->n_calc);
   ```
   See mcd_psfr_parameters_internal() function for more details.
5. The output OTF size is 2*n_calc, i.e. 822x822px, if you consider remark 3.
6. Please refer to [1], Tables 5 and 6, pag. 56, where classification of Raw data and external data are defined. In particular, in column 1 the name of the TAGs (in the sense of SOF list, EDPS classification) is given, while in column 3 the DPR.TYPE (stored in the FITS header cards) is given.

Input data

The input files of this recipe are the output results of the **mcd_psfr_scao_otf_par**. Therefore the two recipes must be executed together (in order), especially when the existing results of a previous run are not available on your system (Typically when the recipe is executed for the first time).

2.5 - MCD_PSFR_SCAO_FINAL

Implemented scheme



Mandatory inputs

Variable	**Input from TAG[17] / header[18] / filename[19]**	**Comments**
otf->alias	**OTF_PARA_LONG_IMG** (filename as tag)	
otf->perp	**OTF_PERP_LONG_IMG** (filename as tag)	
otf->telescope	**OTF_ELT_QUASI_STATIC_IMG** (filename as tag)	
otf->perp	**OTF_MICADO_QUASI_STATIC_IMG** (filename as tag)	This input overwrites the variable in the 2nd **mcd_psf_otf_multi ()** call (see item 16 of the algorithm)

[17] TAG is the classification in the SOF (set-of-frames) passed by EDPS / esorex to the recipe. Remark 4.
[18] Header contains the cards to be set (commonly DPR.TYPE) into the header of the FITS file. Remark 4.
[19] Filename is not important, since the classification made by EDPS does not search for the name of the files, but only for the contents of the FITS header. The filename, when provided, is only a reference for sake of clarity.

Recipe parameters

Variable	Description	Comments
params->pixel_scale	Read from EDPS parameters. Set into YAML file or pass it via `-rp scao_otf_par mcd_psfr_scao_par.pixel_scale VALUE`	default (hard-coded) = 1.9
params->tel_diam	Read from EDPS parameters. Set into YAML file or pass it via `-rp scao_otf_par mcd_psfr_scao_par.tel_diam VALUE`	default (hard-coded) = 38.542
params->lambda_sci	Read from EDPS parameters. Set into YAML file or pass it via `-rp scao_otf_par mcd_psfr_scao_par.lambda_sci VALUE`	default (hard-coded) = 2.2e-6
params->psf_size	Read from EDPS parameters. Set into YAML file or pass it via `-rp scao_otf_par mcd_psfr_scao_par.psf_size VALUE`	default (hard-coded) = 2048

Outputs

Variable	Output TAG[20] / filename	Comments
otf_comb_img	**OTF_COMB_IMG** (filename as tag)	FITS image. Intermediate file (blue in scheme)
otf_rot_img	**OTF_ROT_IMG** (filename as tag)	FITS image. Intermediate file (blue in scheme)
otf_total_img	**OTF_TOTAL_IMG** (filename as tag)	FITS image. Intermediate file (blue in scheme)
otf_blurring_kernel_img	**OTF_BLURRING_KERNEL_IMG** (filename as tag)	Saved by cpl_dfs_save_image. Only the image is saved, not the domain information. This is a final output (green in scheme)

[20] Output tag is the name of the PRO.CATG card saved into the header of the FITS file. The same name is used for TAGging outputs in the SOF file.

psf_anisoplanatic_img	**PSF_ANISOPLANATIC_IMG** (filename as tag)	Saved by cpl_dfs_save_image. Only the image is saved, not the domain information. This is a final output (green in scheme)

Recipe algorithm

1. Set correct group
2. Retrieve parameters from parlist
3. Ensure psf_size is PowerOfTwo (otherwise enlarge it)
4. compute the necessary pixel_size(s) and domain values
5. First TAG: MCD_PSFR_OTF_ELT_QUASI_STATIC_IMG_PROCATG (i.e. Telescope) -> call **mcd_psfr_utility_gridfunction_from_img_px**() passing cpl_image and input pixel_size_before (from previous computations)
6. Other TAGs (parallel and perp) -> call **mcd_psfr_utility_gridfunction_from_img_length**() passing cpl_image and the telescope_domain.length.
7. other data: ones matrix of a given size = telescope_domain.n_points
8. call 1st **mcd_psf_otf_multi()** passing the complete otf data structure
9. save the result with tag OTF_COMB_IMG (fits file)
10. (skipped) load angle info from input data (parameters??)
11. call **mcd_psfr_otf_rotate()** (currently, this function is empty, return input)
12. save the result with tag OTF_ROT_IMG (fits file)
13. (skipped) load REF_NONAO_VIBRATIONS info
14. call **mcd_psfr_otf_nonao_vibration()** (currently, this function is empty, return input, ones matrix of a given size = telescope_domain.n_points)
15. "telescope" component is the result of the rotate image, thus we keep the same pixel_size
16. "perp" component is read from input files, thus we use the same telescope_domain.length
17. "parallel" component is the result of previous otf_blurring_kernel, with same telescope_domain.length
18. all the other components are the same as the previous step 3. (ones matrices)
19. call 2nd **mcd_psf_otf_multi()** passing the complete otf data structure
20. save the result with tag OTF_TOTAL_IMG
21. (skipped) load info from REF_FILTER_CURVE
22. fix the final fft_size as psf_size (must be a power of two!)
23. call mcd_psfr_otf_inv()
24. save the result with tag PSF_ANISOPLANATIC_IMG (fits file)

Remarks:

1. All the inputs, both *_IMG and *_TAB, are stored in FITS files containing images and not tables. This is known, and the code is able to read the data via cpl_image_load function. Even if all inputs are images, both the TAG (written in the header of the corresponding fits file) and the EDPS workflow refer to these inputs in the "foreseen" way.

2.6 - References

[1] "MICADO Data Reduction Library Design for PSF Reconstruction Application" issue 1.1.2 (16/02/2024)

A. Setup

In order to setup a system with the necessary software, we must install the following softwares:

- CPL and required libraries
- EsoRex
- EDPS

In the following sections, we describe the detailed installation procedure. Here below the versions of the libraries (used in our developing system) are shown.

******* ESO Recipe Execution Tool, version 3.13.8 *******

Libraries used: CPL = 7.3.2, CFITSIO = 4.2.0, WCSLIB = 7.12, FFTW (normal precision) = 3.3.9, FFTW (single precision) = 3.3.9, OPENMP = 201511

A.1 - CPL and required libraries

We can follow the instructions contained in the official README.md available at this link: https://gitlab.astro-wise.org/micado/micadopipeline/-/blob/develop/README.md?ref_type=heads.
Note that the CPL version is slightly different (7.3.2 -> 7.3.1), but at the same link we can find the correct archive. Same for EsoRex.

As an alternative, we can follow the instruction details available at this link: https://www.eso.org/sci/software/cpl/download.html . Note that, depending on our system, we may need some pre-requisites such as GNU C/C++ compiler, GNU make utility, GNU tar and gzip utilities, etc. Please refer to the mentioned link for more details.

REMARK! CPL requires two versions of the FFTW library, therefore it is also necessary to install the FFTW library, in two different modes (single precision and normal precision). Thus, BEFORE installing the CPL (for example, after the installation of the CFITSIO lib), we must install:

- **FFTW - normal precision:**

```
wget https://ftp.eso.org/pub/dfs/pipelines/libraries/fftw/fftw-3.3.10.tar.gz
tar -xvf fftw-3.3.10.tar.gz
cd fftw-3.3.10
./configure --prefix=$ESOPIPELINE
make
make install
```

- **FFTW - single precision:**

```
./configure --prefix=$ESOPIPELINE --enable-float
make
make install
```

After completing the installation of the CPL library, we can optionally install also the OpenBLAS lib, following this description:

- OpenBLAS:

```
wget https://github.com/OpenMathLib/OpenBLAS/archive/refs/tags/v0.3.30.tar.gz
tar -xvf v0.3.30.tar.gz
cd OpenBLAS-0.3.30
make # add `-j4` to compile in parallel with 4 processes
make install
```

A.2 - EsoRex

We can follow the instructions contained in the official README.md available at this link: https://gitlab.astro-wise.org/micado/micadopipeline/-/blob/develop/README.md?ref_type=heads.

After installation, we can configure our system, by selecting a folder (or more) in which we will install our recipes.

In `[installation folder of esorex]/etc/esorex.rc` modify the section called `--recipe-dir`, setting the path of OUR recipes. For example:

```
# --recipe-dir
# Directory containing recipe libraries. Note that esorex will recursively
# search not only the specified directory, but all sub-directories below it
# as well. Multiple directory heads may be specified, by separating the
# starting paths with colons (:). This option may also be set using the
# environment variable ESOREX_PLUGIN_DIR.
esorex.caller.recipe-dir=/home/andrea/Developer/uves/lib/esopipes-plugins:/home/andrea/
Developer/teiga-psf-r/lib
```

A.3 - EDPS

In order to install we can refer to the main ESO page, at this link: https://www.eso.org/sci/software/edps.html or follow the instructions here below.

Install and setup EDPS

Create the environment

1. Create new virtual environment:
    - create the environment: `python -m venv .venv`, where `.venv` is the name of the environment. Basically, we can create (and rename) the environment everywhere in your system: either inside this folder, or elsewhere.
2. Activate the environment:
    - (bash): `source .venv/bin/activate`
    - (fish): `source .venv/bin/activate.fish`
    - (csh): `source .venv/bin/activate.csh`
    - (other): see documentation here.

> NOTE: If we use `conda` we can replace the steps above with the equivalent conda commands. See the appendix at the end of the tutorial available on the ESO webpage.

Install EDPS

EDPS can be installed using the following command in most Linux and MacOS versions:

```
pip install --extra-index-url
https://ftp.eso.org/pub/dfs/pipelines/repositories/stable/src edps
adari_core
```

This will install the last **stable** version of EDPS (currently, v 1.6.0).

> Remark: if we want to install another version of EDPS (e.g. 1.4.6), we can specify the version in the command line: `pip install --extra-index-url https://ftp.eso.org/pub/dfs/pipelines/repositories/stable/src edps==1.4.6 adari_core`.

Setup EDPS

Change the `application.properties` configuration file in the `$HOME/.edps` folder. In particular, we can edit the [application] section setting the path pointing to the micado folder. For example:

```
workflow_dir=/home/andrea/Developer/micado
```

> Remark: If we choose to set only one path, we will be able to see the workflows inside micado. Remember to backup the existent path before making any changes. Alternatively, we can add the new path "micado" to the existing `workflow_dir` list,

separating the paths by commas. In this way, we must be sure that only a single version for each of our workflows is present in the dir.

Micado workflows

Remember to activate the environment (follow previous section, second item). Then launch EDPS with `edps`.

Graphs

In order to produce workflow graph, we can use the following command:

```
edps -w micado.micado_psfr_wkf -g | dot -Tpng > micado_psfr.png
```

> Remark: We must install `dot` (on ubuntu: `sudo apt install graphviz`) in order to produce the graph. We can specify the PDF format by replacing "png" with "pdf" in the previous command.

Save the dataset structure

We can save the dataset structure into a .json file with the following command:

```
edps -w micado.micado_psfr_wkf -i ABSOLUTE/PATH/OF/THE/DATA -t TASK_NAME
-f > micado_psfr_TASK_NAME.json
```

> Remark: We can obtain the name of the available task with `edps -w micado.micado_psfr_wkf -lt`. We can replace the `-t TASK_NAME` with `-m all` in order to save the entire dataset.

Run a single task

In order to run a single task, we can start edps with the following command:

```
edps -w micado.micado_psfr_wkf -i ABSOLUTE/PATH/OF/THE/DATA -t TASK_NAME
```

The input data must be contained in a folder provided to the calling sequence as an **absolute** path. The output folder may be specified or not.

> Remark: All recipe products, including all logs, book-keeping and intermediate files are saved into the directory `~/EDPS_data`.

Run the entire pipeline

In order to run the entire pipeline, we can start edps with the following command:

```
edps -w micado.micado_psfr_wkf -i ABSOLUTE/PATH/OF/THE/DATA -m all
```