vison Documentation

Release 0.6+93.g5fec365

Ruyman Azzollini

Sep 28, 2018

CONTENTS

Contents:

CHAPTER

ONE

README

vison Euclid VIS Ground Calibration Pipeline

Author Ruyman Azzollini

Contact r.azzollini_at_ucl.ac.uk

issue 0.6+93.g5fec365

version 0.6+93.g5fec365

date Sep 28, 2018

This Python package "vison" is the pipeline that will be used at MSSL for ground calibration of the VIS detection chains (12 + 2 spares), including one ROE, one RPSU and three CCDs each.

CHAPTER

TWO

INSTALLATION

The package is distributed via github. The repository is hosted at:

https://github.com/ruymanengithub/vison

Detailed instructions:

2.1 Installation

2.1.1 Cloning *vison* from the repository using *git*

If you don't have *git* installed in your system, please follow this link first.

Here we will follow these instructions to clone the repository to your own computer. Follow the link for instructions in other operative systems.

Step-by-step:

- Go to https://github.com/ruymanengithub/vison.
- Click on the green "Clone or download" button.
- In the Clone with HTTPs section, click to copy the clone URL for the repository.
- Open a Terminal.
- Change the current working directory to the location where you want the cloned directory to be made.
- Type `git clone`, and then paste the URL you copied in Step 1.

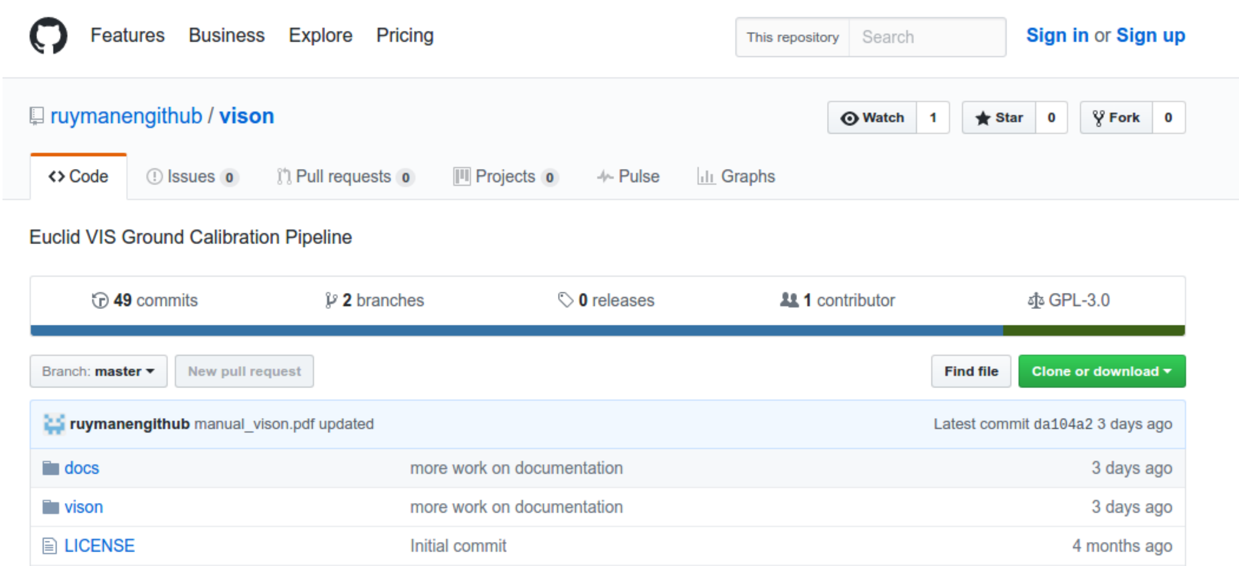
```
~$ git clone https://github.com/ruymanengithub/vison
```

- Press Enter. Your local clone will be created.

2.1.2 Installation

We recommend installing the code through a *conda* environment, with a specific list of packages, so you can be sure you have all the needed dependencies.

First, if you don't have *conda* already installed in your system already, follow the instructions in this link.



Installing conda and creating *vison* environment

Once you have succesfully installed conda, we will create an environment that will allow you to install the pipeline and meet all its dependencies (save for SAO DS9, which is only used in real-time monitoring, optionally).

Step-by-Step:

- change directory to your copy of the vison repository:

```
~$ cd vison
```

- Under the 'conda' sub-folder, you will find several text files:

```
~$ cd conda
~$ ls
env-conda_vison_linux.txt  env-pip_vison.txt
```

- Then execute the following command to create a new conda environment, *vison*.

Use the OS version that may correspond in your case (by now, only linux-64 bits version available).

:: ~$ conda create -n vison –file env-conda_[OS].txt

- When prompted, type "y" and return to install the listed packages.
- Activate the new environment

```
~$ source activate vison
```

- Update pipe

```
~$ pip install --upgrade pip
```

- Install the packages that are accessed via *pip*, within the conda environment:

```
~$ pip install -r env-pip_vison.txt
```

Installing *vison*

Finally, to install the *vison* pipeline itself, we will go back to the folder we downloaded from the github repository:

```
~$ cd ../
~$ ls
conda  docs  LICENSE  manual_vison.pdf  README.md  setup.cfg  setup_distutils.py ␣
↪setup.py  vison
```

Then do the actual installation, via:

~$ python setup.py install

Now the vison package will be accessible from anywhere in your system, whenever you start python from within the *vison* conda environment. For example:

- open a new terminal and go to your home directory

  ```
  ~$ cd
  ```

- activate the vison environment:

  ```
  ~$ source activate vison
  ```

- start the python interpreter and import vison:

  ```
  ~$ source activate vison
  ~$ python
  >>> import vison
  >>> dir(vison)
  ['Eyegore', 'FlatFielding', 'Pipe', 'Report', '__all__', '__builtins__', '__doc__
  ↪', '__file__',
  '__name__', '__package__', '__path__', '__version__', '_version', 'analysis',
  ↪'blocks', 'dark',
  'data', 'datamodel', 'eyegore', 'flat', 'image', 'inject', 'matplotlib', 'ogse',
  ↪'ogse_profiles',
  'other', 'pipe', 'plot', 'point', 'pump', 'stop', 'support']
  ```

2.2 Dependencies

Instructions to acquire a copy of the "conda" environment that provides almost all dependencies is included in the package. See *Installation* instructions for details. The only package that will not be installed by this means is SAO-DS9, which must be installed separetly, in order to be able to use some of the interactive inspection capabilities of Eyegore (data acquisition monitoring).

CHAPTER THREE

PIPELINE CORE

Pipeline master classes.

3.1 Pipeline

3.1.1 master.py

This is the main script that will orchestrate the analysis of Euclid-VIS FM Ground Calibration Campaign.

The functions of this module are:

- Take inputs as to what data is to be analyzed, and what analysis scripts are to be run on it.
- Set the variables necessary to process this batch of FM calib. data.
- Start a log of actions to keep track of what is being done.
- Provide inputs to scripts, execute the analysis scripts and report location of analysis results.

Some Guidelines for Development:

- Input data is "sacred": read-only.
- Each execution of Master must have associated a unique ANALYSIS-ID.
- All the Analysis must be divided in TASKS. TASKS can have SUB-TASKS.
- All data for each TASK must be under a single day-folder.
- All results from the execution of FMmaster must be under a single directory with subdirectories for each TASK run.
- **A subfolder of this root directory will contain the logging information:** inputs, outputs, analysis results locations.

Created on Wed Jul 27 12:16:40 2016

author Ruyman Azzollini

class `vison.pipe.master.`**`Pipe`**(*inputdict*, *dolog=True*, *drill=False*, *debug=False*, *startobsid=0*, *processes=1*)

Master Class of FM-analysis

class **`BF01`**(*inputs*, *log=None*, *drill=False*, *debug=False*)

`build_scriptdict`(*diffvalues={}*, *elvis='6.5.X'*)

Builds PTC0X script structure dictionary.

#:param exptimes: list of ints [ms], exposure times. #:param frames: list of ints, number of frames for each exposure time. #:param wavelength: int, wavelength. Default: 800 nm. :param diffvalues: dict, opt, differential values.

extract_BF()

Performs basic analysis of images:

- extracts BF matrix for each COV matrix

extract_COV()

Performs basic analysis of images:

- extracts COVARIANCE matrix for each fluence

filterexposures(*structure*, *explog*, *OBSID_lims*)

meta_analysis()

Analyzes the BF results across fluences.

set_inpdefaults(***kwargs*)

class Pipe.**BIAS01**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()

BIAS01: Basic analysis of data.

METACODE

```
f. e. ObsID:
   f.e.CCD:

       load ccdobj of ObsID, CCD

       with ccdobj, f.e.Q:
           produce a 2D poly model of bias, save coefficients
           produce average profile along rows
           produce average profile along cols
           # save 2D model and profiles in a pick file for each OBSID-CCD
           measure and save RON after subtracting large scale structure

plot RON vs. time f. each CCD and Q
plot average profiles f. each CCD and Q (color coded by time)
```

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)

Builds BIAS01 script structure dictionary.

###:param N: integer, number of frames to acquire. :param diffvalues: dict, opt, differential values. :param elvis: char, ELVIS version.

filterexposures(*structure*, *explog*, *OBSID_lims*)

meta_analysis()

METACODE

```
f. each CCD:
   stack all ObsIDs to produce Master Bias
   f. e. Q:
       measure average profile along rows
       measure average profile along cols
plot average profiles of Master Bias(s) f. each CCD,Q
(produce table(s) with summary of results, include in report)
save Master Bias(s) (3 images) to FITS CDPs
```

```
show Master Bias(s) (3 images) in report
save name of MasterBias(s) CDPs to DataDict, report
```

prep_data()
BIAS01: Preparation of data for further analysis. Calls task.prepare_images().
Applies: offset subtraction cosmetics masking

class Pipe.**CHINJ00**(*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)
Builds CHINJ00 script structure dictionary.
Parameters **diffvalues** – dict, opt, differential values.

filterexposures(*structure*, *explog*, *OBSID_lims*)

set_inpdefaults(***kwargs*)

class Pipe.**CHINJ01**(*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)
Builds CHINJ01 script structure dictionary.

#:param IDL: float, [V], value of IDL (Inject. Drain Low). #:param IDH: float, [V], Injection Drain High. #:param IG2: float, [V], Injection Gate 2. #:param IG1s: list of 2 floats, [V], [min,max] values of IG1. #:param id_delays: list of 2 floats, [us], injection drain delays. #:param toi_chinj: int, [us], TOI-charge injection. :param diffvalues: dict, opt, differential values.

filterexposures(*structure*, *explog*, *OBSID_lims*)

meta_analysis()
Find injection threshold: Min IG1 Plot and model charge injection vs. IG1 Find notch injection amount.

old_basic_analysis()
Basic analysis of data.

METACODE

```
f. e. ObsID:
    f.e.CCD:
        f.e.Q:
            extract average 2D injection pattern (and save)
            produce average profile along/across lines
            measure charge-inj. non-uniformity
            measure charge spillover into non-injection
            measure stats of injection (mean, med, std, min/max,␣
↪percentiles)

plot average inj. profiles along lines f. each CCD, Q and IG1
    save as a rationalized set of curves
plot average inj. profiles across lines f. each CCD, Q and IG1
    save as a rationalized set of  curves

Report injection stats as a table/tables
```

set_inpdefaults(***kwargs*)

class Pipe.**CHINJ02**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()

Basic analysis of data. AS IT IS, REPEATS WHAT'S DONE IN THE CHECK_DATA. CONSIDER MERGING/SKIPPING

METACODE

```
f. e. ObsID:
    f.e.CCD:
        f.e.Q:
            load average 2D injection pattern
            produce average profile along lines
            [measure charge-inj. non-uniformity]
            [produce average profile across lines]
            [measure charge spillover into non-injection]
            measure stats of injection (mean, med, std, min/max,␣
↪percentiles)

[plot average inj. profiles along lines f. each CCD, Q and IG1]
[    save as a rationalized set of curves]
[plot average inj. profiles across lines f. each CCD, Q and IG1]
[    save as a rationalized set of  curves]

save&plot charge injection vs. IDL
report injection stats as a table
```

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)

Builds CHINJ02 script structure dictionary.

#:param IDLs: list of 2 ints, [V], [min,max] values of IDL (Inject. Drain Low). #:param IDH: int, [V], Injection Drain High. #:param id_delays: list of 2 ints, [us], injection drain delays. #:param toi_chinj: int, [us], TOI-charge injection. :param diffvalues: dict, opt, differential values.

extract_data()

NEEDED? Could be merged with basic_analysis

METACODE

```
Preparation of data for further analysis:

f.e. ObsID:
    f.e.CCD:
        f.e.Q:
            subtract offset
            extract average 2D injection pattern and save
```

filterexposures(*structure*, *explog*, *OBSID_lims*)

meta_analysis()

Finds the Injection Threshold for each CCD half.

METACODE

```
f.e.CCD:
    f.e.Q:
        load injection vs. IDL cuve
        find&save injection threshold on curve

report injection threshold as a table
```

set_inpdefaults(***kwargs*)

class Pipe.**DARK01**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()
DARK01: Basic analysis of data.

METACODE

```
f. e. ObsID:
    f.e.CCD:
        f.e.Q:
            produce mask of hot pixels
            count hot pixels / columns
            produce a 2D poly model of masked-image, save coefficients
            produce average profile along rows
            produce average profile along cols
            measure and save RON after subtracting large scale structure
        save 2D model and profiles in a pick file for each OBSID-CCD

plot average profiles f. each CCD and Q (color coded by time)
```

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)
Builds DARK01 script structure dictionary.
Parameters diffvalues – dict, opt, differential values.

filterexposures(*structure*, *explog*, *OBSID_lims*)

meta_analysis()
METACODE

```
f. each CCD:
    f. e. Q:
        stack all ObsIDs to produce Master Dark
        produce mask of hot pixels / columns
        count hot pixels / columns
        measure average profile along rows
        measure average profile along cols

plot average profiles of Master Bias f. each CCD,Q
show Master Dark (images), include in report
report stats of defects, include in report
save name of MasterDark to DataDict, report
save name of Defects in Darkness Mask to DD, report
```

prep_data()
DARK01: Preparation of data for further analysis. Calls task.prepare_images().
Applies: offset subtraction [BIAS SUBTRACTION] cosmetics masking

class Pipe.**FLAT0X**(*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)
Builds FLAT0X script structure dictionary.
Parameters diffvalues – dict, opt, differential values.

do_indiv_flats()
METACODE

```
Preparation of data for further analysis and
produce flat-field for each OBSID.
```

```
f.e. ObsID:
    f.e.CCD:

        load ccdobj

        f.e.Q:

            model 2D fluence distro in image area
            produce average profile along rows
            produce average profile along cols

        save 2D model and profiles in a pick file for each OBSID-CCD
        divide by 2D model to produce indiv-flat
        save indiv-Flat to FITS(?), update add filename

plot average profiles f. each CCD and Q (color coded by time)
```

do_master_flat()

METACODE

```
Produces Master Flat-Field

f.e.CCD:
    f.e.Q:
        stack individual flat-fields by chosen estimator
save Master FF to FITS
measure PRNU and
report PRNU figures
```

do_prdef_mask()

METACODE

```
Produces mask of defects in Photo-Response
Could use master FF, or a stack of a subset of images (in order
to produce mask, needed by other tasks, quicker).

f.e.CCD:
    f.e.Q:
        produce mask of PR defects
        save mask of PR defects
        count dead pixels / columns

report PR-defects stats
```

filterexposures(*structure*, *explog*, *OBSID_lims*)

prepare_images()

FLAT0X: Preparation of data for further analysis. Calls task.prepare_images().

Applies: offset subtraction [bias structure subtraction, if available] cosmetics masking

set_inpdefaults(***kwargs*)

class Pipe.**FOCUS00**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()

This is just an assignation of values measured in check_data.

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)
Builds FOCUS00 script structure dictionary.

#:param wavelength: int, [nm], wavelength. #:param exptime: int, [ms], exposure time. :param diffvalues: dict, opt, differential values.

filterexposures (*structure*, *explog*, *OBSID_lims*)

lock_on_stars ()

meta_analysis ()

prep_data ()

class Pipe.**MOT_FF** (*inputs*, *log=None*, *drill=False*, *debug=False*)

extract_HER ()

class Pipe.**NL01** (*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)
Builds NL01 script structure dictionary.

#:param expts: list of ints [ms], exposure times. #:param exptinter: int, ms, exposure time of interleaved source-stability exposures. #:param frames: list of ints, number of frames for each exposure time. #:param wavelength: int, wavelength. Default: 0 (Neutral Density Filter) :param diffvalues: dict, opt, differential values.

do_satCTE ()
METACODE

```
select ObsIDs with fluence(exptime) >~ 0.5 FWC

f.e. ObsID:
    CCD:
        Q:
            measure CTE from amount of charge in over-scan relative to␣
↪fluence

f.e. CCD:
    Q:
        get curve of CTE vs. fluence
        measure FWC from curve in ADU

report FWCs in electrons [via gain in inputs] f.e. CCD, Q (table)
```

extract_stats ()
Performs basic analysis: extracts statistics from image regions to later build NLC.

METACODE

```
create segmentation map given grid parameters

f.e. ObsID:
    f.e.CCD:
        f.e.Q:
            f.e. "img-segment": (done elsewhere)
                measure central value
                measure variance
```

filterexposures(*structure*, *explog*, *OBSID_lims*)
Loads a list of Exposure Logs and selects exposures from test PSF0X.

The filtering takes into account an expected structure for the acquisition script.

The datapath becomes another column in DataDict. This helps dealing with tests that run overnight and for which the input data is in several date-folders.

prep_data()
Takes Raw Data and prepares it for further analysis.

METACODE

```
f.e. ObsID:
    f.e.CCD:
        f.e.Q:
            subtract offset
        opt: [sub bias frame]
        opt: [divide by FF]
        opt: [mask-out defects]
```

produce_NLCs()
METACODE

```
Obtains Best-Fit Non-Linearity Curve

f.e. CCD:
    f.e. Q:

        [opt] apply correction for source variability (interspersed␣
↪exposure
          with constant exptime)
        Build NL Curve (NLC) - use stats and exptimes
        fit poly. shape to NL curve

plot NL curves for each CCD, Q
report max. values of NL (table)
```

class Pipe.**PERSIST01**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()
Basic analysis of data.

METACODE

```
f.e.CCD:
    f.e.Q:
        use SATURATED frame to generate pixel saturation MASK
        measure stats in pix satur MASK across OBSIDs
         (pre-satur, satur, post-satur)
```

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)
Builds PERSISTENCE01 script structure dictionary.

Parameters

- **exptSATUR** – int, saturation exposure time.
- **exptLATEN** – int, latency exposure time.
- **diffvalues** – dict, opt, differential values.

check_data()

PERSIST01: Checks quality of ingested data.

METACODE

```
check common HK values are within safe / nominal margins
check voltages in HK match commanded voltages, within margins

f.e.ObsID:
    f.e.CCD:
        f.e.Q.:
            measure offsets in pre-, over-
            measure std in pre-, over-
            measure fluence in apertures around Point Sources

assess std in pre- (~RON) is within allocated margins
assess offsets in pre-, and over- are equal, within allocated  margins
assess fluence is ~expected within apertures (PS) for each frame (pre-
↪satur, satur, post-satur)


plot point source fluence vs. OBSID, all sources
[plot std vs. time]

issue any warnings to log
issue update to report
```

filterexposures(*structure*, *explog*, *OBSID_lims*)

meta_analysis()

Meta-analysis of data.

METACODE

```
f.e.CCD:
    f.e.Q:
        estimate delta-charge_0 and decay tau from time-series

report:
    persistence level (delta-charge_0) and time constant
```

prep_data()

PERSIST01: Preparation of data for further analysis. Calls task.prepare_images().

Applies: offset subtraction cosmetics masking

set_inpdefaults(***kwargs*)

class Pipe.**PTC0X**(*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)

Builds PTC0X script structure dictionary.

#:param exptimes: list of ints [ms], exposure times. #:param frames: list of ints, number of frames for each exposure time. #:param wavelength: int, wavelength. Default: 800 nm. :param diffvalues: dict, opt, differential values.

extract_PTC()

Performs basic analysis of images:

- builds PTC curves: both on non-binned and binned images

METACODE

```
create list of OBSID pairs

create segmentation map given grid parameters

f.e. OBSID pair:
    CCD:
        Q:
            subtract CCD images
            f.e. segment:
                measure central value
                measure variance
```

filterexposures (*structure*, *explog*, *OBSID_lims*)

meta_analysis ()

Analyzes the variance and fluence: gain, and gain(fluence)

METACODE

```
f.e. CCD:
    Q:
        (using stats across segments:)
        fit PTC to quadratic model
        solve for gain
        solve for alpha (pixel-correls, Guyonnet+15)
        solve for blooming limit (ADU)
            convert bloom limit to electrons, using gain

plot PTC curves with best-fit f.e. CCD, Q
report on gain estimates f. e. CCD, Q (table)
report on blooming limits (table)
```

set_inpdefaults (***kwargs*)

class Pipe.**STRAY00** (*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)

Builds STRAY00 script structure dictionary. :param diffvalues: dict, opt, differential values.

filterexposures (*structure*, *explog*, *OBSID_lims*)

set_inpdefaults (***kwargs*)

class Pipe.**TP00** (*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)

check_data ()

TP01: Checks quality of ingested data.

METACODE

```
check common HK values are within safe / nominal margins
check voltages in HK match commanded voltages, within margins

f.e.ObsID:
    f.e.CCD:
        f.e.Q.:
```

```
            measure offsets in pre-, over-
            measure std in pre-, over-
            measure mean in img-

assess std in pre- (~RON) is within allocated margins
assess offsets in pre-, and over- are equal, within allocated margins
assess offsets are within allocated margins
assess injection level is within expected margins

plot histogram of injected levels for each Q
[plot std vs. time]

issue any warnings to log
issue update to report
```

filterexposures (*structure*, *explog*, *OBSID_lims*)

set_inpdefaults (***kwargs*)

class Pipe.**TP01** (*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis ()

Basic analysis of data.

METACODE

```
f. e. ObsID [there are different TOI_TP and TP-patterns]:
    f.e.CCD:
        f.e.Q:
            load "map of relative pumping"
            find_dipoles:
                x, y, rel-amplitude, orientation

produce & report:
    map location of dipoles
    PDF of dipole amplitudes (for N and S)
    Counts of dipoles (and N vs. S)
```

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)

extract ()

Obtain maps of dipoles.

METACODE

```
f.e. id_delay (there are 2):
    f.e. CCD:
        f.e. Q:
            produce reference non-pumped injection map

f. e. ObsID:
    f.e. CCD:

        load ccdobj
        f.e.Q.:
            divide ccdobj.Q by injection map

        save dipole map and store reference
```

filterexposures (*structure*, *explog*, *OBSID_lims*)

meta_analysis ()

Meta-analysis of data:

Try to identify tau and pixel-phase location for each trap. Need to associate dipoles across TOI_TPs and TP-patterns

METACODE

```
across TOI_TP, patterns:
    build catalog of traps: x,y,I-phase, Amp
    from Amp(TOI) -> tau, Pc

Report on :
    Histogram of Taus
    Histogram of Pc (capture probability)
    Histogram of I-phases (larger phases should have more traps,
                        statistically) -> check

    Total Count of Traps
```

set_inpdefaults (***kwargs*)

class Pipe.**TP02** (*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis ()

Basic analysis of data.

METACODE

```
f. e. ObsID [there are different TOI_TP and TP-patterns]:
    f.e.CCD:
        f.e.Q:
            load raw 1D map of relative pumping (from extract_data)
            identify dipoles:
                x, rel-amplitude, orientation (E or W)

produce & report:
    map location of dipoles
    PDF of dipole amplitudes (for E and W)
    Counts of dipoles (and E vs. W)
```

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)

extract ()

Obtain Maps of Serial Dipoles.

filterexposures (*structure*, *explog*, *OBSID_lims*)

meta_analysis ()

Meta-analysis of data:

Try to identify tau and pixel-phase location for each trap. Need to associate dipoles across TOI_TPs and TP-patterns

METACODE

```
across TOI_TP, patterns:
    build catalog of traps: x,y,R-phase, amp(dwell)
    from Amp(dwell) -> tau, Pc

Report on :
```

```
Histogram of Taus
Histogram of Pc (capture probability)
Histogram of R-phases

Total Count of Traps
```

set_inpdefaults(***kwargs*)

Pipe.**catchtraceback**()

Pipe.**dotask**(*taskname*, *inputs*, *drill=False*, *debug=False*)
Generic test master function.

Pipe.**get_execution_summary**(*exectime=None*)

Pipe.**get_test**(*taskname*, *inputs={}*, *log=None*, *drill=False*, *debug=False*)

Pipe.**launchtask**(*taskname*)

Pipe.**run**(*explogf=None*, *elvis=None*)

Pipe.**wait_and_run**(*dayfolder*, *elvis='6.5.X'*)

class vison.pipe.master.**Pipe**(*inputdict*, *dolog=True*, *drill=False*, *debug=False*, *startobsid=0*, *processes=1*)
Master Class of FM-analysis

class BF01(*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)
Builds PTC0X script structure dictionary.

#:param exptimes: list of ints [ms], exposure times. #:param frames: list of ints, number of frames for each exposure time. #:param wavelength: int, wavelength. Default: 800 nm. :param diffvalues: dict, opt, differential values.

extract_BF()
Performs basic analysis of images:
- extracts BF matrix for each COV matrix

extract_COV()
Performs basic analysis of images:
- extracts COVARIANCE matrix for each fluence

filterexposures(*structure*, *explog*, *OBSID_lims*)

meta_analysis()
Analyzes the BF results across fluences.

set_inpdefaults(***kwargs*)

class Pipe.**BIAS01**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()
BIAS01: Basic analysis of data.

METACODE

```
f. e. ObsID:
   f.e.CCD:

       load ccdobj of ObsID, CCD
```

```
    with ccdobj, f.e.Q:
        produce a 2D poly model of bias, save coefficients
        produce average profile along rows
        produce average profile along cols
        # save 2D model and profiles in a pick file for each OBSID-CCD
        measure and save RON after subtracting large scale structure

plot RON vs. time f. each CCD and Q
plot average profiles f. each CCD and Q (color coded by time)
```

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)
Builds BIAS01 script structure dictionary.

###:param N: integer, number of frames to acquire. :param diffvalues: dict, opt, differential values. :param elvis: char, ELVIS version.

filterexposures (*structure*, *explog*, *OBSID_lims*)

meta_analysis ()
METACODE

```
f. each CCD:
   stack all ObsIDs to produce Master Bias
   f. e. Q:
       measure average profile along rows
       measure average profile along cols
plot average profiles of Master Bias(s) f. each CCD,Q
(produce table(s) with summary of results, include in report)
save Master Bias(s) (3 images) to FITS CDPs
show Master Bias(s) (3 images) in report
save name of MasterBias(s) CDPs to DataDict, report
```

prep_data ()
BIAS01: Preparation of data for further analysis. Calls task.prepare_images().
Applies: offset subtraction cosmetics masking

class Pipe.**CHINJ00** (*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)
Builds CHINJ00 script structure dictionary.
Parameters diffvalues – dict, opt, differential values.

filterexposures (*structure*, *explog*, *OBSID_lims*)

set_inpdefaults (***kwargs*)

class Pipe.**CHINJ01** (*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)
Builds CHINJ01 script structure dictionary.

#:param IDL: float, [V], value of IDL (Inject. Drain Low). #:param IDH: float, [V], Injection Drain High. #:param IG2: float, [V], Injection Gate 2. #:param IG1s: list of 2 floats, [V], [min,max] values of IG1. #:param id_delays: list of 2 floats, [us], injection drain delays. #:param toi_chinj: int, [us], TOI-charge injection. :param diffvalues: dict, opt, differential values.

filterexposures (*structure*, *explog*, *OBSID_lims*)

meta_analysis()

Find injection threshold: Min IG1 Plot and model charge injection vs. IG1 Find notch injection amount.

old_basic_analysis()

Basic analysis of data.

METACODE

```
f. e. ObsID:
    f.e.CCD:
        f.e.Q:
            extract average 2D injection pattern (and save)
            produce average profile along/across lines
            measure charge-inj. non-uniformity
            measure charge spillover into non-injection
            measure stats of injection (mean, med, std, min/max,␣
↪percentiles)

plot average inj. profiles along lines f. each CCD, Q and IG1
    save as a rationalized set of curves
plot average inj. profiles across lines f. each CCD, Q and IG1
    save as a rationalized set of  curves

Report injection stats as a table/tables
```

set_inpdefaults(***kwargs*)

class Pipe.**CHINJ02**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()

Basic analysis of data. AS IT IS, REPEATS WHAT'S DONE IN THE CHECK_DATA. CONSIDER MERGING/SKIPPING

METACODE

```
f. e. ObsID:
    f.e.CCD:
        f.e.Q:
            load average 2D injection pattern
            produce average profile along lines
            [measure charge-inj. non-uniformity]
            [produce average profile across lines]
            [measure charge spillover into non-injection]
            measure stats of injection (mean, med, std, min/max,␣
↪percentiles)

[plot average inj. profiles along lines f. each CCD, Q and IG1]
[    save as a rationalized set of curves]
[plot average inj. profiles across lines f. each CCD, Q and IG1]
[    save as a rationalized set of  curves]

save&plot charge injection vs. IDL
report injection stats as a table
```

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)

Builds CHINJ02 script structure dictionary.

#:param IDLs: list of 2 ints, [V], [min,max] values of IDL (Inject. Drain Low). #:param IDH: int, [V], Injection Drain High. #:param id_delays: list of 2 ints, [us], injection drain delays. #:param toi_chinj: int, [us], TOI-charge injection. :param diffvalues: dict, opt, differential values.

extract_data()

NEEDED? Could be merged with basic_analysis

METACODE

```
Preparation of data for further analysis:

f.e. ObsID:
    f.e.CCD:
        f.e.Q:
            subtract offset
            extract average 2D injection pattern and save
```

filterexposures(*structure*, *explog*, *OBSID_lims*)

meta_analysis()

Finds the Injection Threshold for each CCD half.

METACODE

```
f.e.CCD:
    f.e.Q:
        load injection vs. IDL cuve
        find&save injection threshold on curve

report injection threshold as a table
```

set_inpdefaults(***kwargs*)

class Pipe.**DARK01**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()

DARK01: Basic analysis of data.

METACODE

```
f. e. ObsID:
    f.e.CCD:
        f.e.Q:
            produce mask of hot pixels
            count hot pixels / columns
            produce a 2D poly model of masked-image, save coefficients
            produce average profile along rows
            produce average profile along cols
            measure and save RON after subtracting large scale structure
        save 2D model and profiles in a pick file for each OBSID-CCD

plot average profiles f. each CCD and Q (color coded by time)
```

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)

Builds DARK01 script structure dictionary.

Parameters diffvalues – dict, opt, differential values.

filterexposures(*structure*, *explog*, *OBSID_lims*)

meta_analysis()

METACODE

```
f. each CCD:
    f. e. Q:
        stack all ObsIDs to produce Master Dark
        produce mask of hot pixels / columns
        count hot pixels / columns
        measure average profile along rows
        measure average profile along cols

plot average profiles of Master Bias f. each CCD,Q
show Master Dark (images), include in report
report stats of defects, include in report
save name of MasterDark to DataDict, report
save name of Defects in Darkness Mask to DD, report
```

prep_data()

DARK01: Preparation of data for further analysis. Calls task.prepare_images().

Applies: offset subtraction [BIAS SUBTRACTION] cosmetics masking

class Pipe.**FLAT0X**(*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)

Builds FLAT0X script structure dictionary.

Parameters **diffvalues** – dict, opt, differential values.

do_indiv_flats()

METACODE

```
Preparation of data for further analysis and
produce flat-field for each OBSID.

f.e. ObsID:
    f.e.CCD:

        load ccdobj

        f.e.Q:

            model 2D fluence distro in image area
            produce average profile along rows
            produce average profile along cols

        save 2D model and profiles in a pick file for each OBSID-CCD
        divide by 2D model to produce indiv-flat
        save indiv-Flat to FITS(?), update add filename

plot average profiles f. each CCD and Q (color coded by time)
```

do_master_flat()

METACODE

```
Produces Master Flat-Field

f.e.CCD:
    f.e.Q:
        stack individual flat-fields by chosen estimator
save Master FF to FITS
measure PRNU and
```

```
report PRNU figures
```

do_prdef_mask()

METACODE

```
Produces mask of defects in Photo-Response
Could use master FF, or a stack of a subset of images (in order
to produce mask, needed by other tasks, quicker).

f.e.CCD:
    f.e.Q:
        produce mask of PR defects
        save mask of PR defects
        count dead pixels / columns

report PR-defects stats
```

filterexposures(*structure*, *explog*, *OBSID_lims*)

prepare_images()

FLAT0X: Preparation of data for further analysis. Calls task.prepare_images().

Applies: offset subtraction [bias structure subtraction, if available] cosmetics masking

set_inpdefaults(***kwargs*)

class Pipe.**FOCUS00**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()

This is just an assignation of values measured in check_data.

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)

Builds FOCUS00 script structure dictionary.

#:param wavelength: int, [nm], wavelength. #:param exptime: int, [ms], exposure time. :param diffvalues: dict, opt, differential values.

filterexposures(*structure*, *explog*, *OBSID_lims*)

lock_on_stars()

meta_analysis()

prep_data()

class Pipe.**MOT_FF**(*inputs*, *log=None*, *drill=False*, *debug=False*)

extract_HER()

class Pipe.**NL01**(*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)

Builds NL01 script structure dictionary.

#:param expts: list of ints [ms], exposure times. #:param exptinter: int, ms, exposure time of interleaved source-stability exposures. #:param frames: list of ints, number of frames for each exposure time. #:param wavelength: int, wavelength. Default: 0 (Neutral Density Filter) :param diffvalues: dict, opt, differential values.

do_satCTE()

METACODE

```
select ObsIDs with fluence(exptime) >~ 0.5 FWC

f.e. ObsID:
    CCD:
        Q:
            measure CTE from amount of charge in over-scan relative to␣
↪fluence

f.e. CCD:
    Q:
        get curve of CTE vs. fluence
        measure FWC from curve in ADU

report FWCs in electrons [via gain in inputs] f.e. CCD, Q (table)
```

extract_stats()

Performs basic analysis: extracts statistics from image regions to later build NLC.

METACODE

```
create segmentation map given grid parameters

f.e. ObsID:
    f.e.CCD:
        f.e.Q:
            f.e. "img-segment": (done elsewhere)
                measure central value
                measure variance
```

filterexposures(*structure*, *explog*, *OBSID_lims*)

Loads a list of Exposure Logs and selects exposures from test PSF0X.

The filtering takes into account an expected structure for the acquisition script.

The datapath becomes another column in DataDict. This helps dealing with tests that run overnight and for which the input data is in several date-folders.

prep_data()

Takes Raw Data and prepares it for further analysis.

METACODE

```
f.e. ObsID:
    f.e.CCD:
        f.e.Q:
            subtract offset
        opt: [sub bias frame]
        opt: [divide by FF]
        opt: [mask-out defects]
```

produce_NLCs()

METACODE

```
Obtains Best-Fit Non-Linearity Curve

f.e. CCD:
    f.e. Q:
```

```
        [opt] apply correction for source variability (interspersed␣
↪exposure
          with constant exptime)
        Build NL Curve (NLC) - use stats and exptimes
        fit poly. shape to NL curve

plot NL curves for each CCD, Q
report max. values of NL (table)
```

class Pipe.**PERSIST01**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()

Basic analysis of data.

METACODE

```
f.e.CCD:
    f.e.Q:
        use SATURATED frame to generate pixel saturation MASK
        measure stats in pix satur MASK across OBSIDs
         (pre-satur, satur, post-satur)
```

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)

Builds PERSISTENCE01 script structure dictionary.

Parameters

- **exptSATUR** – int, saturation exposure time.
- **exptLATEN** – int, latency exposure time.
- **diffvalues** – dict, opt, differential values.

check_data()

PERSIST01: Checks quality of ingested data.

METACODE

```
check common HK values are within safe / nominal margins
check voltages in HK match commanded voltages, within margins

f.e.ObsID:
    f.e.CCD:
        f.e.Q.:
            measure offsets in pre-, over-
            measure std in pre-, over-
            measure fluence in apertures around Point Sources

assess std in pre- (~RON) is within allocated margins
assess offsets in pre-, and over- are equal, within allocated  margins
assess fluence is ~expected within apertures (PS) for each frame (pre-
↪satur, satur, post-satur)


plot point source fluence vs. OBSID, all sources
[plot std vs. time]

issue any warnings to log
issue update to report
```

filterexposures(*structure*, *explog*, *OBSID_lims*)

meta_analysis ()
Meta-analysis of data.

METACODE

```
f.e.CCD:
    f.e.Q:
        estimate delta-charge_0 and decay tau from time-series

report:
    persistence level (delta-charge_0) and time constant
```

prep_data ()
PERSIST01: Preparation of data for further analysis. Calls task.prepare_images().
Applies: offset subtraction cosmetics masking

set_inpdefaults (***kwargs*)

class Pipe.**PTC0X** (*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)
Builds PTC0X script structure dictionary.

#:param exptimes: list of ints [ms], exposure times. #:param frames: list of ints, number of frames for each exposure time. #:param wavelength: int, wavelength. Default: 800 nm. :param diffvalues: dict, opt, differential values.

extract_PTC ()
Performs basic analysis of images:

- builds PTC curves: both on non-binned and binned images

METACODE

```
create list of OBSID pairs

create segmentation map given grid parameters

f.e. OBSID pair:
    CCD:
        Q:
            subtract CCD images
            f.e. segment:
                measure central value
                measure variance
```

filterexposures (*structure*, *explog*, *OBSID_lims*)

meta_analysis ()
Analyzes the variance and fluence: gain, and gain(fluence)

METACODE

```
f.e. CCD:
    Q:
        (using stats across segments:)
        fit PTC to quadratic model
        solve for gain
        solve for alpha (pixel-correls, Guyonnet+15)
        solve for blooming limit (ADU)
            convert bloom limit to electrons, using gain
```

```
plot PTC curves with best-fit f.e. CCD, Q
report on gain estimates f. e. CCD, Q (table)
report on blooming limits (table)
```

set_inpdefaults(***kwargs*)

class Pipe.**STRAY00**(*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)
Builds STRAY00 script structure dictionary. :param diffvalues: dict, opt, differential values.

filterexposures(*structure*, *explog*, *OBSID_lims*)

set_inpdefaults(***kwargs*)

class Pipe.**TP00**(*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)

check_data()
TP01: Checks quality of ingested data.

METACODE

```
check common HK values are within safe / nominal margins
check voltages in HK match commanded voltages, within margins

f.e.ObsID:
    f.e.CCD:
        f.e.Q.:
            measure offsets in pre-, over-
            measure std in pre-, over-
            measure mean in img-

assess std in pre- (~RON) is within allocated margins
assess offsets in pre-, and over- are equal, within allocated margins
assess offsets are within allocated margins
assess injection level is within expected margins

plot histogram of injected levels for each Q
[plot std vs. time]

issue any warnings to log
issue update to report
```

filterexposures(*structure*, *explog*, *OBSID_lims*)

set_inpdefaults(***kwargs*)

class Pipe.**TP01**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()
Basic analysis of data.

METACODE

```
f. e. ObsID [there are different TOI_TP and TP-patterns]:
    f.e.CCD:
```

```
        f.e.Q:
            load "map of relative pumping"
            find_dipoles:
                x, y, rel-amplitude, orientation

produce & report:
    map location of dipoles
    PDF of dipole amplitudes (for N and S)
    Counts of dipoles (and N vs. S)
```

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)

extract ()

Obtain maps of dipoles.

METACODE

```
f.e. id_delay (there are 2):
    f.e. CCD:
        f.e. Q:
            produce reference non-pumped injection map

f. e. ObsID:
    f.e. CCD:

        load ccdobj
        f.e.Q.:
            divide ccdobj.Q by injection map

        save dipole map and store reference
```

filterexposures (*structure*, *explog*, *OBSID_lims*)

meta_analysis ()

Meta-analysis of data:

Try to identify tau and pixel-phase location for each trap. Need to associate dipoles across TOI_TPs and TP-patterns

METACODE

```
across TOI_TP, patterns:
    build catalog of traps: x,y,I-phase, Amp
    from Amp(TOI) -> tau, Pc

Report on :
    Histogram of Taus
    Histogram of Pc (capture probability)
    Histogram of I-phases (larger phases should have more traps,
                        statistically) -> check

    Total Count of Traps
```

set_inpdefaults (***kwargs*)

class Pipe.**TP02** (*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis ()

Basic analysis of data.

METACODE

```
f. e. ObsID [there are different TOI_TP and TP-patterns]:
    f.e.CCD:
        f.e.Q:
            load raw 1D map of relative pumping (from extract_data)
            identify dipoles:
                x, rel-amplitude, orientation (E or W)

produce & report:
    map location of dipoles
    PDF of dipole amplitudes (for E and W)
    Counts of dipoles (and E vs. W)
```

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)

extract ()
Obtain Maps of Serial Dipoles.

filterexposures (*structure*, *explog*, *OBSID_lims*)

meta_analysis ()
Meta-analysis of data:
Try to identify tau and pixel-phase location for each trap. Need to associate dipoles across TOI_TPs and TP-patterns

METACODE

```
across TOI_TP, patterns:
    build catalog of traps: x,y,R-phase, amp(dwell)
    from Amp(dwell) -> tau, Pc

Report on :
   Histogram of Taus
   Histogram of Pc (capture probability)
   Histogram of R-phases

   Total Count of Traps
```

set_inpdefaults (***kwargs*)

Pipe.**catchtraceback** ()

Pipe.**dotask** (*taskname*, *inputs*, *drill=False*, *debug=False*)
Generic test master function.

Pipe.**get_execution_summary** (*exectime=None*)

Pipe.**get_test** (*taskname*, *inputs={}*, *log=None*, *drill=False*, *debug=False*)

Pipe.**launchtask** (*taskname*)

Pipe.**run** (*explogf=None*, *elvis=None*)

Pipe.**wait_and_run** (*dayfolder*, *elvis='6.5.X'*)

3.1.2 task.py

Generic Task (Test) Class.

Created on Tue Nov 14 14:20:04 2017

author Ruyman Azzollini

class vison.pipe.task.**Task** (*inputs*, *log=None*, *drill=False*, *debug=False*)

IsComplianceMatrixOK (*complidict*)

addComplianceMatrix2Log (*complidict*, *label=''*)

addComplianceMatrix2Report (*complidict*, *label=''*, *caption=''*)

addFigure2Report (*figkey*)

addFigures_ST (*dobuilddata=True*, ***kwargs*)

addFlagsToLog ()

addFlagsToReport ()

addHKPlotsMatrix ()
Adds to self.report a table-figure with HK [self.HKKeys] during test.

addHK_2_dd ()

add_data_inventory_to_report (*tDict*)

add_inputs_to_report ()

add_labels_to_explog (*explog*, *structure*)

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)

catchtraceback ()

check_HK (*HKKeys*, *reference='command'*, *limits='P'*, *tag=''*, *doReport=False*, *doLog=True*)

check_HK_ST ()

check_data (***kwargs*)
Generic check_data method

check_stat_perCCD (*arr, CCDlims, CCDs=['CCD1', 'CCD2', 'CCD3']*)

check_stat_perCCDQandCol (*arr, lims, CCDs=['CCD1', 'CCD2', 'CCD3']*)

check_stat_perCCDandCol (*arr, lims, CCDs=['CCD1', 'CCD2', 'CCD3']*)

check_stat_perCCDandQ (*arr, CCDQlims, CCDs=['CCD1', 'CCD2', 'CCD3']*)

create_mockexplog (*OBSID0=1000*)

doPlot (*figkey*, ***kwargs*)

filterexposures (*structure*, *explog*, *OBSID_lims*, *colorblind=False*, *wavedkeys=[]*, *surrogate=''*)
Loads a list of Exposure Logs and selects exposures from test 'test'.

The filtering takes into account an expected structure for the acquisition script.

The datapath becomes another column in DataDict. This helps dealing with tests that run overnight and for which the input data is in several date-folders.

prepare_images (*doExtract=True*, *doMask=False*, *doOffset=False*, *doBias=False*, *doFF=False*)

recover_progress (*DataDictFile*, *reportobjFile*)

save_progress (*DataDictFile*, *reportobjFile*)

skipMissingPlot (*key*, *ref*)

class vison.pipe.task.**Task** (*inputs*, *log=None*, *drill=False*, *debug=False*)

IsComplianceMatrixOK(*complidict*)

addComplianceMatrix2Log(*complidict*, *label=''*)

addComplianceMatrix2Report(*complidict*, *label=''*, *caption=''*)

addFigure2Report(*figkey*)

addFigures_ST(*dobuilddata=True*, ***kwargs*)

addFlagsToLog()

addFlagsToReport()

addHKPlotsMatrix()
Adds to self.report a table-figure with HK [self.HKKeys] during test.

addHK_2_dd()

add_data_inventory_to_report(*tDict*)

add_inputs_to_report()

add_labels_to_explog(*explog*, *structure*)

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)

catchtraceback()

check_HK(*HKKeys*, *reference='command'*, *limits='P'*, *tag=''*, *doReport=False*, *doLog=True*)

check_HK_ST()

check_data(***kwargs*)
Generic check_data method

check_stat_perCCD(*arr, CCDlims, CCDs=['CCD1', 'CCD2', 'CCD3']*)

check_stat_perCCDQandCol(*arr, lims, CCDs=['CCD1', 'CCD2', 'CCD3']*)

check_stat_perCCDandCol(*arr, lims, CCDs=['CCD1', 'CCD2', 'CCD3']*)

check_stat_perCCDandQ(*arr, CCDQlims, CCDs=['CCD1', 'CCD2', 'CCD3']*)

create_mockexplog(*OBSID0=1000*)

doPlot(*figkey*, ***kwargs*)

filterexposures(*structure*, *explog*, *OBSID_lims*, *colorblind=False*, *wavedkeys=[]*, *surrogate=''*)
Loads a list of Exposure Logs and selects exposures from test 'test'.

The filtering takes into account an expected structure for the acquisition script.

The datapath becomes another column in DataDict. This helps dealing with tests that run overnight and for which the input data is in several date-folders.

prepare_images(*doExtract=True*, *doMask=False*, *doOffset=False*, *doBias=False*, *doFF=False*)

recover_progress(*DataDictFile*, *reportobjFile*)

save_progress(*DataDictFile*, *reportobjFile*)

skipMissingPlot(*key*, *ref*)

CHAPTER

FOUR

DATA MODEL

Modules with classes to hold data model for inputs and outputs: exposure log, HK files, FITS files, etc.

4.1 Data Model

4.1.1 ccd.py

Data model for Euclid-VIS CCDs (ground testing at MSSL)

Created on Fri Nov 13 17:42:36 2015

author Ruyman Azzollini

class `vison.datamodel.ccd.`**`CCD`**(*infits=None, extensions=[-1], getallextensions=False, withpover=True*)

Class of CCD objects. Euclid Images as acquired by ELVIS software (Euclid LabView Imaging Software).

The class has been extended to handle multi-extension images. This is useful to also "host" calibration data-products, such as Flat-Fields.

A note on Coordinates Systems:

- 'CCD': referenced to the first pixel readout from channel H. All 4 quadrants

in a single array, their detection nodes in the 4 "corners" of the rectangle. Same system as images are displayed on DS9. In clock-wise sense, quadrants are H (bottom-left), E (top-left), F (top-right), and G (bottom-right). - 'Quadrant-canonical': Quadrant coordinates system in which the first pixel is the first pixel read out (closest pixel to the readout node), and the last is the last readout. In this system, the serial pre-scan comes before the image area, and this before the serial overscan. Parallel overscan comes after image area in the parallel direction. In this system, coordinates of pixels across quadrants, for a single readout, correspond to the same point in time. Useful when doing cross-talk analysis, for example. - 'Quadrant-relative': quadrant coordinates system with the same relative orientation as in the 'CCD' system, but referenced to the 'lower-left' pixel of the given quadrant in such system. In this system, the readout node is in a different corner for each quadrant: lower-left for H, top-left for E, top-right for F and bottom-right for G.

`add_extension`(*data*, *header=None*, *label=None*, *headerdict=None*)

Appends an extension to self (extensions are in a list).

`add_to_hist`(*action*, *extension=-1*, *vison=u'0.6+93.g5fec365'*, *params={}*)

`cooconvert`(*x*, *y*, *insys*, *outsys*, *Q='U'*)

Coordinates conversion between different systems.

`del_extension`(*extension*)

Deletes an extension from self, by index.

divide_by_flatfield(*FF*, *extension=-1*)
Divides by a Flat-field

do_Vscan_Mask(*VSTART*, *VEND*)

dummyrebin(*arr*, *new_shape*, *stat='median'*)

extract_region(*ccdobj*, *Q*, *area='img'*, *vstart=0*, *vend=2086*, *Full=False*, *canonical=True*, *extension=-1*)

get_1Dprofile(*ccdobj*, *Q*, *orient='hor'*, *area='img'*, *stacker='mean'*, *vstart=0*, *vend=2086*, *extension=-1*)

get_Q(*x*, *y*, *w*, *h*)

get_cutout(*corners*, *Quadrant*, *canonical=False*, *extension=-1*)
Returns a cutout from the CCD image, either in canonical or non-canonical orientation.

Parameters

- **corners** (*list (of int)*) – [x0,x1,y0,y1]
- **Quadrant** (*char*) – Quadrant, one of 'E', 'F', 'G', 'H'
- **canonical** (*bool*) – Canonical [True] = with readout-node at pixel index (0,0) regardless of quadrant. This is the orientation which corresponds to the data-readin order (useful for cross-talk measurements, for example). Non-Canonical [False] = with readout-node at corner matching placement of quadrant on the CCD. This is the orientation that would match the representation of the image on DS9.
- **extension** (*int*) – extension number. Default = -1 (last)

get_mask(*mask*)

get_quad(*Quadrant*, *canonical=False*, *extension=-1*)
Returns a quadrant in canonical or non-canonical orientation.

Parameters

- **Quadrant** (*char*) – Quadrant, one of 'E', 'F', 'G', 'H'
- **canonical** –

Canonical [True] = with readout-node at pixel index (0,0) regardless of quadrant. This is the orientation which corresponds to the data-reading order (useful for cross-talk measurements, for example). Non-Canonical [False] = with readout-node at corner matching placement of quadrant on the CCD. This is the orientation that would match the representation of the image on DS9.

Parameters **extension** (*int*) – extension number. Default = -1 (last)

get_region2Dmodel(*ccdobj*, *Q*, *area='img'*, *kind='spline'*, *splinemethod='cubic'*, *pdegree=2*, *doFilter=False*, *doBin=True*, *filtsize=1*, *binsize=1*, *filtertype='mean'*, *vstart=0*, *vend=2086*, *canonical=True*, *extension=-1*)

get_stats(*Quadrant, sector='img', statkeys=['mean'], trimscan=[0, 0], ignore_pover=True, extension=-1, VSTART=0, VEND=2086*)

get_tile_coos(*Quadrant*, *wpx*, *hpx*)
Returns a dictionary with a tiling [coordinates of corners of tiles] of quadrant Q, with tiles of size wpx[width] x hpx[height].

CAUTION: Returned coordinates are Q-relative.

Parameters

- **Quadrant** – str, Quadrant, one of ['E','F','G','H']

- **wpx** – int, width [along NAXIS1] of tiles, in pixels.
- **hpx** – int, height [along NAXIS2] of tiles, in pixels.

Returns tiles_dict = dict(wpx='Width of tiles, integer', hpx='Height of tiles, integer', llpix='Lower left corner of tiles, list of tuples', ccpix= 'Central pixel of tiles, list of tuples', Nsamps='Number of tiles, integer')

get_tiles (*Quadrant*, *tile_coos*, *extension=-1*)

get_tiles_stats (*Quad*, *tile_coos*, *statkey*, *extension=-1*)

getsectioncollims (*Q*)
Returns limits of [HORIZONTAL] sections: prescan, image and overscan

getsectionrowlims (*Q*)
Returns limits of [VERTICAL] sections: image [and vertical overscan]

loadfromFITS (*fitsfile, extensions=[-1], getallextensions=False*)
Loads contents of self from a FITS file.

set_extension (*data*, *header=None*, *label=None*, *headerdict=None*, *extension=-1*)
Sets extension 'extension' in self.

set_quad (*inQdata*, *Quadrant*, *canonical=False*, *extension=-1*)

sim_window (*ccdobj*, *vstart*, *vend*, *extension=-1*)

simadd_flatilim (*ccdobj*, *levels=None*, *extension=-1*)

simadd_points (*ccdobj*, *flux*, *fwhm*, *CCDID='CCD1'*, *dx=0*, *dy=0*, *extension=-1*)

simadd_poisson (*ccdobj*, *extension=-1*)

simadd_ron (*ccdobj*, *extension=-1*)

sub_bias (*superbias*, *extension=-1*)
Subtracts a superbias

sub_offset (*Quad, method='row', scan='pre', trimscan=[3, 2], ignore_pover=True, extension=-1*)

writeto (*fitsf*, *clobber=False*, *unsigned16bit=False*)

class vison.datamodel.ccd.**CCDPile** (*infitsList=[]*, *ccdobjList=[]*, *extension=-1*, *withpover=True*)
Class to hold and operate (e.g. stack) on a bunch of CCD images. Each image (a single extension picked from each) becomes an extension in the pile.

stack (*method='median'*, *dostd=False*)

class vison.datamodel.ccd.**Extension** (*data*, *header=None*, *label=None*, *headerdict=None*)
Extension Class

vison.datamodel.ccd.**cooconv_arrays_decorate** (*func*)

vison.datamodel.ccd.**test_create_from_scratch** ()

vison.datamodel.ccd.**test_load_ELVIS_fits** ()

4.1.2 ccd_aux.py

Auxiliary script to ccd.py

Created on Mon Feb 19 13:14:02 2018

author raf

class `vison.datamodel.ccd_aux.`**`Model2D`**(*img*, *corners=[]*)
Class for 2D models of images and images sections.

`bin_img`(*boxsize*, *stat='median'*)

`filter_img`(*filtsize=15*, *filtertype='median'*, *Tests=False*)

`fit2Dpol_xyz`(*xx*, *yy*, *zz*, *degree=1*)

`get_model_poly2D`(*sampling=1*, *pdegree=5*, *useBin=False*)

`get_model_splines`(*sampling=1*, *splinemethod='cubic'*, *useBin=False*)

class `vison.datamodel.ccd_aux.`**`Profile1D`**(*x*, *y*)
Class for 1D profiles of images and images sections.

`vison.datamodel.ccd_aux.`**`extract_region`**(*ccdobj*, *Q*, *area='img'*, *vstart=0*, *vend=2086*, *Full=False*, *canonical=True*, *extension=-1*)

`vison.datamodel.ccd_aux.`**`get_1Dprofile`**(*ccdobj*, *Q*, *orient='hor'*, *area='img'*, *stacker='mean'*, *vstart=0*, *vend=2086*, *extension=-1*)

`vison.datamodel.ccd_aux.`**`get_region2Dmodel`**(*ccdobj*, *Q*, *area='img'*, *kind='spline'*, *splinemethod='cubic'*, *pdegree=2*, *doFilter=False*, *doBin=True*, *filtsize=1*, *binsize=1*, *filtertype='mean'*, *vstart=0*, *vend=2086*, *canonical=True*, *extension=-1*)

`vison.datamodel.ccd_aux.`**`rebin`**(*arr*, *new_shape*, *stat='mean'*)
"Rebin 2D array arr to shape new_shape by averaging.

4.1.3 ccdsim.py

Methods to simulate data. Used by ccd.CCD class.

Created on Wed Apr 4 11:13:30 2018

author Ruyman Azzollini

`vison.datamodel.ccdsim.`**`sim_window`**(*ccdobj*, *vstart*, *vend*, *extension=-1*)

`vison.datamodel.ccdsim.`**`simadd_flatilum`**(*ccdobj*, *levels=None*, *extension=-1*)

`vison.datamodel.ccdsim.`**`simadd_points`**(*ccdobj*, *flux*, *fwhm*, *CCDID='CCD1'*, *dx=0*, *dy=0*, *extension=-1*)

`vison.datamodel.ccdsim.`**`simadd_poisson`**(*ccdobj*, *extension=-1*)

`vison.datamodel.ccdsim.`**`simadd_ron`**(*ccdobj*, *extension=-1*)

4.1.4 compliance.py

Some functions to present COMPLIANCE MATRICES.

Created on Mon Apr 9 17:32:03 2018

author raf

`vison.datamodel.compliance.`**`convert_compl_to_nesteditemlist`**(*complidict*)

`vison.datamodel.compliance.`**`gen_compliance_tex`**(*indict*, *escape=True*, *caption=''*)

`vison.datamodel.compliance.`**`removescalars_from_dict`**(*indict*)

4.1.5 core.py

DataDict Class : holds data and results across sub-tasks of a "task" (Test). This is the CORE data-structure used to do analysis and report results.

Created on Thu Sep 21 16:47:09 2017

author Ruyman Azzollini

class `vison.datamodel.core.`**`DataDict`**(*meta={}*)

`addColumn`(*array*, *name*, *indices*)

`col_has_index`(*colname*, *indexname*)

`dropColumn`(*colname*)

`initColumn`(*name*, *indices*, *dtype='float32'*, *valini=0.0*)

`loadExpLog`(*explog*)

`name_indices`()

`saveToFile`(*outfile*, *format='ascii.commented_header'*)

`vison.datamodel.core.`**`useCases`**()

#TODO:

create a DataDict object from an exposure log. # add a column indexed by ObsID, CCD and Quad # drop a column # create a column from an operation on several columns with different dimensions # save to a text / excel file # save to a pickle file

class `vison.datamodel.core.`**`vColumn`**(*array*, *name*, *indices*)

`name_indices`()

class `vison.datamodel.core.`**`vIndex`**(*name*, *vals=[]*, *N=0*)

4.1.6 EXPLOGtools.py

class `vison.datamodel.EXPLOGtools.`**`ExpLogClass`**(*elvis='6.5.X'*)

`addRow`(*row*)

`iniExplog`()

`summary`()

`writeto`(*outfile*)

`vison.datamodel.EXPLOGtools.`**`iniExplog`**(*elvis*)

`vison.datamodel.EXPLOGtools.`**`loadExpLog`**(*expfile*, *elvis='6.5.X'*)

Loads an Exposure Log from file.

`vison.datamodel.EXPLOGtools.`**`mergeExpLogs`**(*explogList*, *addpedigree=False*, *verbose=False*)

Merges explog objects in a list.

`vison.datamodel.EXPLOGtools.`**`test`**()

This Tests needs UPDATE (for data access and probably data format)

4.1.7 generator.py

Script to generate simulated data for pipeline testing purposes.

Created on Tue Aug 29 11:08:56 2017

author Ruyman Azzollini

`vison.datamodel.generator.`**`IMG_bias_gen`**(*ccdobj*, *ELdict*, *ogse=None*)

`vison.datamodel.generator.`**`IMG_chinj_gen`**(*ccdobj*, *ELdict*, *ogse=None*)

`vison.datamodel.generator.`**`IMG_flat_gen`**(*ccdobj*, *ELdict*, *ogse=None*)

`vison.datamodel.generator.`**`IMG_point_gen`**(*ccdobj*, *ELdict*, *ogse=None*)

`vison.datamodel.generator.`**`generate_Explog`**(*scrdict*, *defaults*, *elvis='6.5.X'*, *explog=None*, *OBSID0=1000*, *date=datetime.datetime(1980*, *2*, *21*, *7*, *0)*, *CHAMBER=None*)

Generates a fake ExposureLog from a test structure dictionary.

DEVELOPMENT NOTES:

To be generated: (EASY) *ObsID, *File_name, *CCD, *ROE=ROE1, *DATE, *BUNIT=ADU, SPW_clk=0?,EGSE_ver=elvis,

Temporal: SerRdDel

To be provided in defaults: (EASY) Lab_ver,Con_file,CnvStart, Flsh-Rdout_e_time,C.Inj-Rdout_e_time, FPGA_ver,Chmb_pre,R1CCD[1,2,3]T[T,B]

To be read/parsed/processed from struct: (DIFFICULT)

SerRDel?,SumWell?, IniSweep?,+etc.

`vison.datamodel.generator.`**`generate_FITS`**(*ELdict*, *funct*, *filename=''*, *elvis='6.5.X'*, *ogse=None*)

`vison.datamodel.generator.`**`generate_FITS_fromExpLog`**(*explog*, *datapath*, *elvis='6.5.X'*, *CHAMBER=None*)

`vison.datamodel.generator.`**`generate_HK`**(*explog*, *vals*, *datapath=''*, *elvis='6.5.X'*)

`vison.datamodel.generator.`**`merge_HKfiles`**(*HKfilefs*, *masterHKf*)

4.1.8 HKtools.py

House-Keeping inspection and handling tools.

History

Created on Thu Mar 10 12:11:58 2016

author Ruyman Azzollini

`vison.datamodel.HKtools.`**`check_HK_abs`**(*HKKeys*, *dd*, *limits='S'*, *elvis='6.5.X'*)

Returns report on HK parameters, in DataDict (dd), compared to absolute limits.

HK Keys which have "relative" limits, always return False.

Parameters

- **`HKKeys`** – list of HK parameters, as named in HK files (without HK_ suffix)
- **`dd`** – DataDict object
- **`limits`** – type of limits to use, either "P" (Performance) or "S" (Safe)

- **elvis** – ELVIS version to find correspondence between HK key and Exposure Log input (commanded voltage).

Returns report dictionary with pairs of HK-key : Bool. True = All values for given key are within limits. False = At least one value for given key is outside limits.

vison.datamodel.HKtools.**check_HK_vs_command**(*HKKeys*, *dd*, *limits='P'*, *elvis='6.5.X'*)
Returns report on HK parameters, in DataDict (dd), comparing inputs (commanded) vs. output (HK data).

HK Keys which do not correspond to commanded voltages always return 'True'.

Parameters

- **HKKeys** – list of HK parameters, as named in HK files (without **HK_** suffix)
- **dd** – DataDict object
- **limits** – type of limits to use, either "P" (Performance) or "S" (Safe)
- **elvis** – ELVIS version to find correspondence between HK key and Exposure Log input (commanded voltage).

Returns report dictionary with pairs of HK-key : Bool. True = All values are within limits, referred to commanded value. False = At least one value is outside limits, referred to commanded value.

vison.datamodel.HKtools.**doHKSinglePlot**(*dtobjs*, *HK*, *HKkey*, *ylabel='V'*, *HKlims=[]*, *filename=''*, *fontsize=10*)
Plots the values of a HK parameter as a function of time.

Parameters

- **dtobjs** – datetime objects time axis.
- **HK** – HK values (array)
- **HKkey** –
- **ylabel** –
- **HKlims** –
- **filename** – file-name to store plot [empty string not to save].

Returns None!!

vison.datamodel.HKtools.**filtervalues**(*values*, *key*)

vison.datamodel.HKtools.**iniHK_QFM**(*elvis='6.5.X'*, *length=0*)

vison.datamodel.HKtools.**loadHK_QFM**(*filename*, *elvis='6.5.X'*, *validate=False*)
Loads a HK file, or list of HK files.

Structure: astropy table. First column is a timestamp, and there may be a variable number of rows (readings).

Parameters

- **filename** – path to the file to be loaded, including the file itself, or list of paths to HK files.
- **elvis** – "ELVIS" version

Returns astropy table with pairs parameter:[values]

vison.datamodel.HKtools.**loadHK_QFMsingle**(*filename*, *elvis='6.5.X'*, *validate=False*)
Loads a HK file

Structure: tab separated columns, one per Keyword. First column is a timestamp, and there may be a variable number of rows (readings).

Parameters

- **filename** – path to the file to be loaded, including the file itself
- **elvis** – "ELVIS" version

Returns astropy table with pairs parameter:[values]

vison.datamodel.HKtools.**loadHK_preQM**(*filename*, *elvis='5.7.07'*)

Loads a HK file

It only assumes a structure given by a HK keyword followed by a number of of tab-separated values (number not specified). Note that the length of the values arrays is variable (depends on length of exposure and HK sampling rate).

Parameters **filename** – path to the file to be loaded, including the file itself

Returns dictionary with pairs parameter:[values]

vison.datamodel.HKtools.**mergeHK**(*HKList*)

vison.datamodel.HKtools.**parseDTstr**(*DTstr*)

vison.datamodel.HKtools.**parseHKfiles**(*HKlist*, *elvis='6.5.X'*)

Parameters

- **HKlist** – list of HK files (path+name).
- **elvis** – "ELVIS" version.

Returns [obsids],[dtobjs],[tdeltasec],[HK_keys], [data(nfiles,nstats,nHKparams)]

vison.datamodel.HKtools.**parseHKfname**(*HKfname*)

Parses name of a HK file to retrieve OBSID, date and time, and ROE number.

Parameters **HKfname** – name of HK file.

Returns obsid,dtobj=datetime.datetime(yy,MM,dd,hh,mm,ss),ROE

vison.datamodel.HKtools.**reportHK**(*HKs*, *key*, *reqstat='all'*)

Returns (mean, std, min, max) for each keyword in a list of HK dictionaries (output from loadHK).

Parameters

- **HK** – dictionary with HK data.
- **key** – HK key.

Reqstat what statistic to retrieve.

vison.datamodel.HKtools.**synthHK**(*HK*)

Synthetizes the values for each parameter in a HK dictionary into [mean,std,min,max].

Parameters **HK** – a dictionary as those output by loadHK.

Returns dictionary with pairs parameter:[mean,std,min,max]

4.1.9 inputs.py

Inputs Handling Classes and utilities.

Created on Thu Jan 11 10:34:43 2018

author Ruyman Azzollini

class `vison.datamodel.inputs.Inputs`(**args*, ***kwargs*)
Class to hold, transfer and 'document' Task Inputs.

4.1.10 QLAtools.py

Quick-Look-Analysis Tools.

History

Created on Wed Mar 16 11:31:58 2016

@author: Ruyman Azzollini

`vison.datamodel.QLAtools.dissectFITS`(*FITSfile*, *path=''*)

`vison.datamodel.QLAtools.getacrosscolscut`(*CCDobj*)

`vison.datamodel.QLAtools.getacrossrowscut`(*CCDobj*)

`vison.datamodel.QLAtools.getsectionstats`(*CCDobj*, *QUAD*, *section*, *xbuffer=(0, 0)*, *ybuffer=(0, 0)*)

`vison.datamodel.QLAtools.plotAcCOLcuts`(*dissection*, *filename=None*, *suptitle=''*)

`vison.datamodel.QLAtools.plotAcROWcuts`(*dissection*, *filename=None*, *suptitle=''*)

`vison.datamodel.QLAtools.plotQuads`(*CCDobj*, *filename=None*, *suptitle=''*)

`vison.datamodel.QLAtools.reportFITS`(*FITSfile*, *outpath=''*)

4.1.11 scriptic.py

Classes and functions to generate ELVIS commanding scripts automatically.

Created on Wed May 24 15:31:54 2017

author Ruyman Azzollini

class `vison.datamodel.scriptic.Script`(*defaults={}*, *structure={}*, *elvis='6.5.X'*)
Core Class that provides automatic test script generation and validation.

build_cargo()
Updates 'cargo' attribute. 'cargo': list of lists, each corresponding to a column in the script.

Each element in the inner lists is a register value. The first column corresponds to the column with key names.

Note: the number of frames is accumuled across columns, as ELVIS expects.

get_struct_from_cargo()

load(**args*, ***kwargs*)
alias method. Points to 'load_to_cargo'.

load_to_cargo(*scriptname*, *elvis='6.5.X'*)
Loads an script from an excel file.

Parameters

- **scriptname** – char, script to load
- **elvis** – char, ELVIS version of script to load

validate(*defaults*, *structure*, *elvis='6.5.X'*)
Not sure 'validation' will work like as implemented... TODO: validate self.validate

write(*scriptname*)
Writes self.cargo (script) to an excel file.

Parameters **scriptname** – char, name of file where to write script.

vison.datamodel.scriptic.**test0**()

vison.datamodel.scriptic.**update_structdict**(*sdict*, *commvalues*, *diffvalues*)
Updates an script structure with common values and differential values.

Parameters

- **sdict** – dict, dictionary with script structure. Takes precedence over commvalues.
- **commvalues** – dict, dictionary with common values to update sdict.
- **diffvalues** – dict, dictionaty with "differential" values to update "sdict". Takes precedence over sdict and commvalues.

CHAPTER

FIVE

ANALYSIS (SHARED)

5.1 Analysis (Shared)

5.1.1 ellipse.py

Auxiliary module with functions to generate generalized ellipse masks.

author Ruyman Azzollini

class `vison.analysis.ellipse.`**`TestEllipse`**(*methodName='runTest'*)
Unit tests for the ellipse module.

`vison.analysis.ellipse.`**`area_superellip`**(*r*, *q*, *c=0*)
Returns area of superellipse, given the semi-major axis length

`vison.analysis.ellipse.`**`dist_superellipse`**(*n*, *center*, *q=1.0*, *pos_ang=0.0*, *c=0.0*)
Form an array in which the value of each element is equal to the semi-major axis of the superellipse of specified center, axial ratio, position angle, and c parameter which passes through that element. Useful for super-elliptical aperture photometry.

Inspired on dist_ellipse.pro from AstroLib (IDL).

Note: this program doesn't take into account the change in the order of axes from IDL to Python. That means, that in 'n' and in 'center', the order of the coordinates must be reversed with respect to the case for dist_ellipse.pro, in order to get expected results. Nonetheless, the polar angle means the counter-clock wise angle with respect to the 'y' axis.

Parameters

- **`n`** – shape of array (N1,N2), it can be an integer (squared shape NxN)
- **`center`** – center of superellipse radii: (c1,c2)
- **`q`** – axis ratio r2/r1
- **`pos_ang`** – position angle of isophotes, in degrees, CCW from axis 1
- **`c`** – boxyness (c>0) /diskyness (c<0)

`vison.analysis.ellipse.`**`effective_radius`**(*area*, *q=1.0*, *c=0.0*)
Returns semi-major axis length of superellipse, given the area

5.1.2 Guyonnet15.py

Library with functions that implement the algorithms described in Guyonnet+15. "Evidence for self-interaction of charge distribution in CCDs" Guyonnet, Astier, Antilogus, Regnault and Doherty 2015

Notes:

- I renamed "x" (pixel boundary index) to "b", to avoid confusion with cartesian "x".
- In paper, X belongsto [(0,1),(1,0),(0,-1),(-1,0)]. Here b is referred to as cardinal points "N","E","S","W". It is linked to matrix index ib, running between 0 and 3.

Created on Thu Sep 22 11:38:24 2016

author Ruyman Azzollini

`vison.analysis.Guyonnet15.`**`correct_estatic`**(*img*, *aijb*)
Corrects an image from pixel-boundaries deformation due to electrostatic forces. Subtracts delta-Q.

Parameters

- **`img`** – image, 2D array
- **`aijb`** – Aijb matrix, 3D array

Returns array, img - delta-Q

`vison.analysis.Guyonnet15.`**`degrade_estatic`**(*img*, *aijb*)
Degrades an image according to matrix of pixel-boundaries deformations. Follows on Eq. 11 of G15. Adds delta-Q.

Parameters

- **`img`** – image, 2D array
- **`aijb`** – Aijb matrix, 3D array

Returns array, img + delta-Q

`vison.analysis.Guyonnet15.`**`fpred_aijb`**(*p*, *i*, *j*, *ib*)
'The smoothing model assumes that a_{ij}^x coefficients are the product of a function of distance from the source charge to the considered boundary (r_{ij}) and that it also trivially depends on the angle between the source-boundary vector and the normal to the boundary (theta_{i,j}^x)'

Eq. 18

Parameters

- **`p`** – parameters of the radial function (list of 2)
- **`i`** – pixel coordinate i
- **`j`** – pixel coordinate j
- **`ib`** – boundary index [0, 1, 2, 3]

Returns f(rij)cos(theta_ij^x)

`vison.analysis.Guyonnet15.`**`frdist`**(*i*, *j*, *ib*)
Distance from the source charge to considered boundary "b"

Parameters

- **`i`** – pixel coordinate i
- **`j`** – pixel coordinate j
- **`ib`** – boundary index [0, 1, 2, 3]

Returns distance r(ijb)

`vison.analysis.Guyonnet15.`**`ftheta_bound`**(*i*, *j*, *ib*)
"[theta_i,j^X is] the angle between the source-boundary vector and the normal to the boundary".

Parameters

- **i** – pixel coordinate i
- **j** – pixel coordinate j
- **ib** – boundary index [0, 1, 2, 3]

Returns theta_i,j^x

`vison.analysis.Guyonnet15.`**`fun_p`**(*x*, **p*)
auxiliary function to 'solve_for_psmooth'

`vison.analysis.Guyonnet15.`**`generate_GaussPSF`**(*N*, *sigma*)
Create a circular symmetric Gaussian centered on the centre of a NxN matrix/image.

`vison.analysis.Guyonnet15.`**`get_Rdisp`**(*img*, *aijb*)
Retrieves map of relative displacements of pixel boundaries, for input img and Aijb matrix.

See G15 - Eq. 6

Parameters

- **img** – image, 2D array
- **aijb** – aijb matrix, 3D array NxNx4

Returns array, relative displacements all boundaries of pixels in img

`vison.analysis.Guyonnet15.`**`get_cross_shape_rough`**(*cross*, *pitch=12.0*)

`vison.analysis.Guyonnet15.`**`get_deltaQ`**(*img*, *aijb*, *writeFits=False*)
Retrieves deltaQ map for input image and aijb matrix.

See G15 - Eq. 11

Parameters

- **img** – image, 2D array
- **aijb** – Aijb matrix, 3D array
- **writeFits** – save FITS file with resulting dQ map (optional)

Returns array, matrix with delta-Q for each pixel in img, given aijb

`vison.analysis.Guyonnet15.`**`get_kernel`**(*aijb*)
'kernel' is an array (2N-1)x(2N-1)x4. Each plane kernel[:,:,b] is a 2D array with the displacement coefficients aijb, in all directions around a pixel at (0,0).

Parameters

- **aijb** – array, matrix with displacements in 1st quadrant
- **writeFits** – save kernel to 4 FITS files

Returns kernel matrix, (2N-1)x(2N-1)x4

`vison.analysis.Guyonnet15.`**`plot_map`**(*z*, *ii*, *jj*, *title='‘*)

`vison.analysis.Guyonnet15.`**`plot_maps_ftheta`**(*f*, *ii*, *jj*, *suptitle='‘*)

`vison.analysis.Guyonnet15.`**`show_disps_CCD273`**(*aijb*, *stretch=5.0*, *peak=28571.428571428572*, *N=25*, *sigma=1.6*, *title='‘*, *figname='‘*)

vison.analysis.Guyonnet15.**solve_for_A_linalg**(*covij*, *var=1.0*, *mu=1.0*, *doplot=False*, *psmooth=None*, *returnAll=False*)

Function to retrieve the A matrix of pixel boundaries displacements, given a matrix of pixel covariances, variance, and mu.

if var==1 and mu==1, it is understood that covij is the correlation matrix.

See section 6.1 of G15.

Parameters

- **covij** – array, squared matrix with pixel covariances.
- **var** – float, variance of the flat-field.
- **mu** – float, mean value of the flat-field.
- **doplot** – if True, plot the fit of the fpred(ijb) function
- **psmooth** – coefficients of the fpred(aijb) function (Eq. 18)
- **returnAll** – bool, controls return values

Returns if returnAll == True, return (aijb, psmooth), otherwise return aijb only

vison.analysis.Guyonnet15.**solve_for_psmooth**(*covij*, *var*, *mu*, *doplot=False*)

Solving (p0,p1) parameters in Eq. 18 using covariance matrix and measured covariance matrix.

Parameters

- **covij** – array, covariance matrix
- **var** – float, variance
- **mu** – float, expected value of pixel values (“mean” of flat-field)
- **doplot** – bool, if True, plot data and best fit model

Returns best-fit parameters, and errors: 2 tuples of 2 elements each

vison.analysis.Guyonnet15.**test0**()

vison.analysis.Guyonnet15.**test_getkernel**()

vison.analysis.Guyonnet15.**test_selfconsist**()

vison.analysis.Guyonnet15.**test_solve**()

CHAPTER

SIX

CHARGE INJECTION TOOLS

6.1 Charge Injection Tools

6.1.1 InjTask.py

Created on Wed Dec 6 15:56:00 2017

author Ruyman Azzollini

class `vison.inject.InjTask.`**`InjTask`**(**args*, ***kwargs*)

`BROKEN_basic_analysis`()

Basic analysis of data.

METACODE

```
f. e. ObsID:
    f.e.CCD:
        f.e.Q:
            extract average 2D injection pattern (and save)
            produce average profile along/across lines
            measure charge-inj. non-uniformity
            measure charge spillover into non-injection
            measure stats of injection (mean, med, std, min/max, percentiles)

plot average inj. profiles along lines f. each CCD, Q and IG1
    save as a rationalized set of curves
plot average inj. profiles across lines f. each CCD, Q and IG1
    save as a rationalized set of  curves

Report injection stats as a table/tables
```

`check_data`(***kwargs*)

`check_metrics_ST`(***kwargs*)

TODO:

- offset levels (pre and over-scan), abs. and relative
- RON in pre and overscan
- mean fluence/signal in image area [script-column-dependent]
- med fluence/signal in image area [script-column-dependent]
- std in image area [script-column-dependent]

get_FluenceAndGradient_limits()

get_checkstats_ST(***kwargs*)

predict_expected_injlevels(*teststruct*)

prepare_images(*doExtract=True*, *doMask=True*, *doOffset=True*, *doBias=True*, *doFF=False*)
InjTask: Preparation of data for further analysis. Calls task.prepare_images().

Applies: offset subtraction [bias structure subtraction, if available] cosmetics masking

6.1.2 lib.py

NEEDSREVISION

Module to provide common tools for analysis of Charge Injection acquisitions.

Created on Thu Sep 14 15:32:10 2017

author Ruyman Azzollini

6.1.3 plot.py

Charge Injection Plotting Tools.

Created on Thu Sep 14 15:39:34 2017

author Ruyman Azzollini

CHAPTER

SEVEN

"FLAT" ACQ. ANALYSIS TOOLS

7.1 "Flat" Acq. Analysis Tools

7.1.1 FlatTask.py

Created on Mon Dec 4 16:00:10 2017

author Ruyman Azzollini

class `vison.flat.FlatTask.`**`FlatTask`**(**args*, ***kwargs*)

`check_data`()

`check_metrics_ST`(***kwargs*)

TODO:

- offset levels (pre and over-scan), abs. and relative
- RON in pre and overscan
- fluence in image area [script-column-dependent]
- variance in image area [script-column-dependent]

`get_checkstats_ST`(***kwargs*)

7.1.2 FlatFielding.py

Flat-fielding Utilities.

Created on Fri Apr 22 16:13:22 2016

@author: raf

class `vison.pipe.FlatFielding.`**`FlatField`**(*fitsfile='', data={}, meta={}*)

`parse_fits`()

`vison.pipe.FlatFielding.`**`fit2D`**(*xx, yy, zz, degree=1*)

`vison.pipe.FlatFielding.`**`get_ilum`**(*img, pdegree=5, filtsize=15, filtertype='median', Tests=False*)

`vison.pipe.FlatFielding.`**`get_ilum_splines`**(*img, filtsize=25, filtertype='median', Tests=False*) `vison.pipe.FlatFielding.produce_IndivFlats`(*infits*, *outfits*, *settings*, *runonTests*, *processes=6*)

`vison.pipe.FlatFielding.produce_MasterFlat`(*infits*, *outfits*, *mask=None*, *settings={}*)

Produces a Master Flat out of a number of flat-illumination exposures. Takes the outputs from produce_IndivFlats.

`vison.pipe.FlatFielding.produce_SingleFlatfield`(*infits*, *outfits*, *settings={}*, *runonTests=False*)

7.1.3 nl.py

NEEDSREVISION

Module with tools used in NL analysis.

Created on Mon Feb 5 15:51:00 2018

author Ruyman Azzollini

`vison.flat.nl.fitNL`(*fluencesNL*, *exptimes*, *display=False*)

`vison.flat.nl.get_exptime_atmiddynrange`(*flu1D*, *exp1D*, *method='spline'*, *debug=False*)

`vison.flat.nl.test_wrap_fitNL`()

`vison.flat.nl.wrap_fitNL`(*fluences*, *variances*, *exptimes*, *col_labels*, *times=array([], dtype=float64)*, *TrackFlux=True*, *subBgd=True*)

7.1.4 ptc.py

NEEDSREVISION

Module with tools used in PTC analysis.

Created on Thu Sep 14 16:29:36 2017

author Ruyman Azzollini

`vison.flat.ptc.fitPTC`(*means*, *var*)

Fits Photon Transfer Curve to obtain gain.

`vison.flat.ptc.foo_bloom`(*means*, *var*)

DUMMY function (PLACE-HOLDER) (Will) Finds blooming limit (where variance drops, if it does...).

CHAPTER

EIGHT

IMAGE

8.1 Image Analysis

8.1.1 bits.py

NEEDSREVISION

Image bits analysis tools.

Created on Thu Sep 14 15:54:14 2017

author Ruyman Azzollini

8.1.2 calibration.py

Common use CDP functions / methods.

Created on Thu Nov 2 16:54:28 2017

author Ruyman Azzollini

`vison.image.calibration.`**`load_FITS_CDPs`**(*FDict*, *dataclass*, ***kwargs*)
Dummy function to load CDPs for all 3 CCDs. Input is of type dict(CCD1='',CCD2='',CCD3='')

8.1.3 cosmetics.py

Created on Wed Aug 1 11:55:12 2018

@author: Ruyman Azzollini

`vison.image.cosmetics.`**`get_Thresholding_DefectsMask`**(*maskdata*, *thresholds*)

8.1.4 covariance.py

Tools to retrieve covariance matrices for (differences of) Flat-Field images. Used in the context of Brighter-Fatter analysis, mainly.

Created on Wed Mar 7 11:54:54 2018

author Ruyman Azzollini

`vison.image.covariance.`**`f_get_covmap`**(*sq1*, *sq2*, *N*, *debug=False*)

`vison.image.covariance.`**`get_cov_maps`**(*ccdobjList*, *Npix=4*, *doTest=False*)

8.1.5 ds9reg.py

DS9 Regions tool.

Created on Fri May 18 15:02:07 2018

author raf

`vison.image.ds9reg.`**`get_body_circles`**(*X*, *Y*, *R=None*, *radius=6.0*)

`vison.image.ds9reg.`**`get_body_ellipses`**(*X*, *Y*, *A=None*, *B=None*, *THETA=None*)

`vison.image.ds9reg.`**`save_spots_as_ds9regs`**(*data*, *regfilename=None*, *regfile=None*, *regtype='circle'*, *clobber=True*)

8.1.6 performance.py

Performance parameters of the ROE+CCDs. Compilation of CCD offsets, offset gradients, RONs... used for checks.

Created on Wed Nov 1 09:57:44 2017

author Ruyman Azzollini

`vison.image.performance.`**`get_offsets_lims`**(*offsets*, *offsets_margins*)

`vison.image.performance.`**`get_perf_rdout`**(*BLOCKID*)

8.1.7 pixbounce.py

Pixel Bounce Analysis methods.

Created on Fri Mar 9 09:50:16 2018

author Ruyman Azzollini

`vison.image.pixbounce.`**`get_pixbounce_from_overscan`**(*ccdobj*, *thresholds=None*)

Retrieves Hard Edge Respose for all Quadrants of a CCD. Uses the transition from image to overscan (along rows). Averages across rows. Input image should have high image-area fluence but not saturating. Rows can be filtered by average fluence in them via "thresholds" keyword. Do not use on images acquired with irradiated CCDs.

8.1.8 sextractor.py

Sextractor interface.

Created on Thu May 17 13:29:05 2018

author raf

class `vison.image.sextractor.`**`VSExtractor`**(*img=None*)

`load_catalog`(*catpath*)

`run_SEx`(*catroot*, *config=None*, *checks=None*, *cleanafter=False*)

`save_img_to_tmp`(*img*, *delete=True*, *close=False*)

CHAPTER NINE

MONITORING (“EYEGORE”)

Tools to monitor data acquisition on real time: plots of HK, auto-updating of visual display of Exposure Log with some interactive capabilities, and display of latest images.

9.1 Monitoring (“Eyegore”)

Fig. 9.1: You must be Igor...

9.1.1 eyegore.py

eyegore

data acquisition monitoring script for vison package.

‘- You must be Igor... - No, it’s pronounced “Eye-gore”.’

Created on Thu Feb 2 15:27:39 2017

Author Ruyman Azzollini

class `vison.eyegore.eyegore.`**`Eyegore`**(*path*, *broadcast*, *intervals=None*, *elvis='6.5.X'*, *dolite=False*, *altpath=''*, *doWarnings=False*, *dolog=False*)

`setup_MasterWG`()

`vison.eyegore.eyegore.`**`rsync_to_altlocalpath`**(*path*, *altpath*)

`vison.eyegore.eyegore.`**`rsync_to_remote`**(*path*, *broadcast*)

9.1.2 eyeCCDs.py

Eyegore: CCDs display.

Created on Fri Oct 13 16:16:08 2017

author raf

class `vison.eyegore.eyeCCDs.`**`ImageDisplay`**(*parent*, *path*, *elvis='6.5.X'*)

`gen_render`()

`setup_fig`()

9.1.3 eyeHK.py

Eyegore: House Keeping Monitoring.

Created on Fri Oct 13 14:11:41 2017

author raf

class `vison.eyegore.eyeHK.`**`HKDisplay`**(*root*, *path*, *interval*, *elvis='6.5.X'*)

`get_data`()

`search_HKfiles`()

`select_HKkeys`()

class `vison.eyegore.eyeHK.`**`HKFlags`**(*root*, *parent*, *interval=5000*, *elvis='6.5.X'*)

`MuteFlag`(*event*)

`ResetFlag`(*event*)

`UnmuteFlag`(*event*)

`bind_buttons_to_methods`(*ix*)

`changeColor`(*ix*, *color*)

`isflagraised`(*ix*)

`lowerflag`(*ix*)

`raiseflag`(*ix*)

class `vison.eyegore.eyeHK.`**`SingleHKplot`**(*root*)

`vison.eyegore.eyeHK.`**`sort_HKfiles`**(*HKfiles*)

`vison.eyegore.eyeHK.`**`validate_within_HKlim`**(*val*, *HKlim*)

violation: 0: None -1: below lower limit 1: above upper limit 2: different from limit, if limit is a single value

9.1.4 eyeObs.py

Eyegore: Exposure Log Monitoring.

Created on Fri Oct 13 16:22:36 2017

author raf

class `vison.eyegore.eyeObs.`**`ExpLogDisplay`**(*parent*, *path*, *interval*, *elvis='6.5.X'*)

`build_elementList`()

`get_data`()

`loadExplogs`()

`search_EXPLOGs`()

`sortBy`(*tree*, *col*, *descending*)
sort tree contents when a column header is clicked

`vison.eyegore.eyeObs.`**`changeNumeric`**(*data*)
if the data to be sorted is numeric change to float

`vison.eyegore.eyeObs.`**`isNumeric`**(*s*)
test if a string s is numeric

9.1.5 eyeWarnings.py

Module to handle HK-OOL Warnings

Created on Thu Apr 19 16:09:02 2018

author Ruyman Azzollini

`vison.eyegore.eyeWarnings.`**`test_URLs`**()

CHAPTER

TEN

OGSE

OGSE stands for Optical Ground Support Equipment.

10.1 OGSE Tools

10.1.1 ogse.py

Model of the calibration OGSE

Created on Fri Sep 8 12:11:55 2017

author Ruyman Azzollini

`vison.ogse.ogse.`**`get_FW_ID`**(*wavelength*, *FW={'F1': 590, 'F2': 640, 'F3': 730, 'F4': 800, 'F5': 880, 'F6': 0}*)

returns FW key corresponding to input wavelength. :param wavelength: integer, wavelength.

CHAPTER

ELEVEN

PLOTTING

General use plotting facilities.

11.1 Plotting

11.1.1 baseplotclasses.py

vison pipeline: Classes to do plots.

Created on Mon Nov 13 17:54:08 2017

author Ruyman Azzollini

class `vison.plot.baseplotclasses.`**`Beam1DHist`**`(`*data*, ***kwargs*`)`

class `vison.plot.baseplotclasses.`**`BeamPlot`**`(`*data*, ***kwargs*`)`

`populate_axes``()`

`vison.plot.baseplotclasses.`**`testBeam2ImgShow`**`()`

class `vison.plot.baseplotclasses.`**`BasicPlot`**`(`***kwargs*`)`

class `vison.plot.baseplotclasses.`**`Beam1DHist`**`(`*data*, ***kwargs*`)`

class `vison.plot.baseplotclasses.`**`BeamImgShow`**`(`*data*, ***kwargs*`)`

class `vison.plot.baseplotclasses.`**`BeamPlot`**`(`*data*, ***kwargs*`)`

`populate_axes``()`

class `vison.plot.baseplotclasses.`**`BeamPlotYvX`**`(`*data*, ***kwargs*`)`

class `vison.plot.baseplotclasses.`**`CCD2DPlot`**`(`*data*, ***kwargs*`)`

class `vison.plot.baseplotclasses.`**`ImgShow`**`(`*data*, ***kwargs*`)`

`plt_trimmer``()`

`populate_axes``()`

11.1.2 figclasses.py

Created on Mon Apr 16 16:17:13 2018

author Ruyman Azzollini

class `vison.plot.figclasses.`**`BlueScreen`**

`build_data`(**args*, ***kwargs*)

`configure`(***kwargs*)

11.1.3 trends.py

Plotting classes shared across tasks/sub-tasks and derived from plots.baseclasses. They have in common that they show trends with time of some variables / stats.

Created on Fri Jan 26 16:18:43 2018

author raf

CHAPTER

TWELVE

POINT-SOURCE ANALYSIS

12.1 Point-Source Analysis

12.1.1 basis.py

author Ruyman Azzollini

Created on Thu Apr 20 18:56:40 2017

class `vison.point.basis.`**`SpotBase`**(*data*, *log=None*, *verbose=False*)

12.1.2 display.py

Display Library for Point-Source Analysis

Created on Fri Apr 21 14:02:57 2017

requires matplotlib

author Ruyman Azzollini

`vison.point.display.`**`show_spots_allCCDs`**(*spots_bag*, *title=''*, *filename=''*, *dobar=True*)

12.1.3 gauss.py

Gaussian Model of Point-like Sources

Simple class to do Gaussian Fitting to a spot.

requires NumPy, astropy

Created on Thu Apr 20 16:42:47 2017

author Ruyman Azzollini

class `vison.point.gauss.`**`Gaussmeter`**(*data*, *log=None*, *verbose=False*, ***kwargs*)

Provides methods to measure the shape of an object using a 2D Gaussian Model.

Parameters

- **`data`** (*`np.ndarray`*) – stamp to be analysed.
- **`log`** (*`instance`*) – logger
- **`kwargs`** (*`dict`*) – additional keyword arguments

Settings dictionary contains all parameter values needed.

fit_Gauss()

12.1.4 models.py

Models (Point-Like Sources)

Library module with models for processing of point-source imaging data.

requires NumPy

author Ruyman Azzollini

Created on Wed Apr 19 11:47:00 2017

vison.point.models.**fgauss2D**(*x*, *y*, *p*)

A gaussian fitting function where p[0] = amplitude p[1] = x0 p[2] = y0 p[3] = sigmax p[4] = sigmay p[5] = floor

12.1.5 photom.py

Aperture Photometry of point-like objects

Simple class to do aperture photometry on a stamp of a point-source.

requires NumPy

Created on Thu Apr 20 14:37:46 2017

author Ruyman Azzollini

class vison.point.photom.**Photometer**(*data*, *log=None*, *verbose=False*, ***kwargs*)

Provides methods to measure the shape of an object.

Parameters

- **data** (*np.ndarray*) – stamp to be analysed.
- **log** (*instance*) – logger
- **kwargs** (*dict*) – additional keyword arguments

Settings dictionary contains all parameter values needed.

doap_photom(*centre*, *rap*, *rin=-1.0*, *rout=-1.0*, *gain=3.5*, *doErrors=True*, *subbgd=False*)

get_centroid(*rap=None*, *full=False*)

TODO: add aperture masking

measure_bgd(*rin*, *rout*)

sub_bgd(*rin*, *rout*)

12.1.6 shape.py

Quadrupole Moments Shape Measurement

Simple class to measure quadrupole moments and ellipticity of an object.

requires NumPy, PyFITS

author Sami-Matias Niemi, Ruyman Azzollini

class `vison.point.shape.`**`Shapemeter`**(*data*, *log=None*, *verbose=False*, ***kwargs*)
Provides methods to measure the shape of an object.

Parameters

- **`data`** (*`np.ndarray`*) – stamp to be analysed.
- **`log`** (*`instance`*) – logger
- **`kwargs`** (*`dict`*) – additional keyword arguments

Settings dictionary contains all parameter values needed.

`circular2DGaussian`(*x*, *y*, *sigma*)
Create a circular symmetric Gaussian centered on x, y.

Parameters

- **`x`** (*`float`*) – x coordinate of the centre
- **`y`** (*`float`*) – y coordinate of the centre
- **`sigma`** (*`float`*) – standard deviation of the Gaussian, note that sigma_x = sigma_y = sigma

Returns circular Gaussian 2D profile and x and y mesh grid

Return type dict

`ellip2DGaussian`(*x*, *y*, *sigmax*, *sigmay*)
Create a two-dimensional Gaussian centered on x, y.

Parameters

- **`x`** (*`float`*) – x coordinate of the centre
- **`y`** (*`float`*) – y coordinate of the centre
- **`sigmax`** (*`float`*) – standard deviation of the Gaussian in x-direction
- **`sigmay`** (*`float`*) – standard deviation of the Gaussian in y-direction

Returns circular Gaussian 2D profile and x and y mesh grid

Return type dict

`measureRefinedEllipticity`()
Derive a refined iterated polarisability/ellipticity measurement for a given object.

By default polarisability/ellipticity is defined in terms of the Gaussian weighted quadrupole moments. If self.shsettings['weighted'] is False then no weighting scheme is used.

The number of iterations is defined in self.shsettings['iterations'].

Returns centroids [indexing stars from 1], ellipticity (including projected e1 and e2), and R2

Return type dict

quadrupoles (*image*)
Derive quadrupole moments and ellipticity from the input image.

Parameters **img** (*ndarray*) – input image data

Returns quadrupoles, centroid, and ellipticity (also the projected components e1, e2)

Return type dict

writeFITS (*data*, *output*)
Write out a FITS file using PyFITS.

Parameters

- **data** (*ndarray*) – data to write to a FITS file
- **output** (*string*) – name of the output file

Returns None

12.1.7 spot.py

Spot Stamp Class.

Created on Thu Apr 20 15:35:08 2017

author Ruyman Azzollini

class vison.point.spot.**Spot** (*data*, *log=None*, *verbose=False*, *lowerleft=(None,)*, ***kwargs*)
Provides methods to do point-source analysis on a stamp. Aimed at basic analysis:

•Photometry

•Quadrupole Moments

•Gaussian Fit

Parameters

- **data** (*np.ndarray*) – stamp to be analysed.
- **log** (*instance*) – logger
- **kwargs** (*dict*) – additional keyword arguments

Settings dictionary contains all parameter values needed.

get_photom ()
measurements:'apflu','eapflu','bgd','ebgd'

get_shape_Gauss ()

Returns res = dict(i0,ei0,x,ex,y,ey, sigma_x,esigma_x,sigmay,esigma_y, fwhm_x,efwhm_x, fwhm_y,efwhm_y, fluence,efluence)

get_shape_Moments ()

Returns res = dict(x,y,ellip,e1,e2,a,b)

get_shape_easy (*method='G'*, *debug=False*)

measure_basic (*rap=10*, *rin=10*, *rout=-1*, *gain=3.1*, *debug=False*)
TODO: # get basic statistics, measure and subtract background # update centroid # do aperture photometry # pack-up results and return

Parameters

- **rap** – source extraction aperture radius.
- **rin** – inner radius of background annulus.
- **rout** – outer radius of background annulus (-1 to set bound by image area).
- **gain** – image gain (e-/ADU).

12.1.8 lib.py

Library module with useful data and functions for processing of point-source imaging data.

Created on Wed Apr 5 10:21:05 2017

author Ruyman Azzollini (except where indicated)

`vison.point.lib.`**`extract_spot`**(*ccdobj*, *coo*, *Quad*, *log=None*, *stampw=25*) `vison.point.lib.`**`gen_point_mask`**(*CCD*, *Quad*, *width=75*, *sources='all'*, *coodict={'CCD2': OrderedDict([('H', OrderedDict([('BRAVO', (1725.0, 1606.0)), ('CHARLIE', (1131.0, 1029.0)), ('ALPHA', (554.0, 1626.0)), ('ECHO', (1706.0, 435.0)), ('DELTA', (531.0, 446.0))])), ('E', OrderedDict([('BRAVO', (1716.0, 1700.0)), ('CHARLIE', (1126.0, 1124.0)), ('ALPHA', (542.0, 1725.0)), ('ECHO', (1695.0, 537.0)), ('DELTA', (521.0, 551.0))])), ('G', OrderedDict([('BRAVO', (1702.0, 1571.0)), ('CHARLIE', (1139.0, 1033.0)), ('ALPHA', (534.0, 1590.0)), ('ECHO', (1685.0, 394.0)), ('DELTA', (515.0, 415.0))])), ('F', OrderedDict([('BRAVO', (1745.0, 1668.0)), ('CHARLIE', (1141.0, 1144.0)), ('ALPHA', (578.0, 1686.0)), ('ECHO', (1723.0, 496.0)), ('DELTA', (553.0, 522.0))]))]), 'CCD3': {'H': {'BRAVO': (1689.4, 1668.8000000000002), 'ALPHA': (460.6, 1668.8000000000002), 'DELTA': (460.6, 417.20000000000005), 'ECHO': (1689.4, 417.20000000000005), 'CHARLIE': (1075.0, 1043.0)}, 'E': {'BRAVO': (1689.4, 3754.8), 'ALPHA': (460.6, 3754.8), 'DELTA': (460.6, 2503.2), 'ECHO': (1689.4, 2503.2), 'CHARLIE': (1075.0, 3129.0)}, 'G': {'BRAVO': (3808.4, 1668.8000000000002), 'ALPHA': (2579.6, 1668.8000000000002), 'DELTA': (2579.6, 417.20000000000005), 'ECHO': (3808.4, 417.20000000000005), 'CHARLIE': (3194.0, 1043.0)}, 'F': {'BRAVO': (3808.4, 3754.8), 'ALPHA': (2579.6, 3754.8), 'DELTA': (2579.6, 2503.2), 'ECHO': (3808.4, 2503.2), 'CHARLIE': (3194.0, 3129.0)}}, 'CCD1': {'H': {'ALPHA': (460.6, 1668.8000000000002), 'CHARLIE': (1075.0, 1043.0), 'DELTA': (460.6, 417.20000000000005), 'ECHO': (1689.4, 417.20000000000005), 'BRAVO': (1689.4, 1668.8000000000002)}, 'E': {'ALPHA': (460.6, 3754.8), 'CHARLIE': (1075.0, 3129.0), 'DELTA': (460.6, 2503.2), 'ECHO': (1689.4, 2503.2), 'BRAVO': (1689.4, 3754.8)}, 'G': {'ALPHA': (2579.6, 1668.8000000000002), 'CHARLIE': (3194.0, 1043.0), 'DELTA': (2579.6, 417.20000000000005), 'ECHO': (3808.4, 417.20000000000005), 'BRAVO': (3808.4, 1668.8000000000002)}, 'F': {'ALPHA': (2579.6, 3754.8), 'CHARLIE': (3194.0, 3129.0), 'DELTA': (2579.6, 2503.2), 'ECHO': (3808.4, 2503.2), 'BRAVO': (3808.4, 3754.8)}}, 'names': ['ALPHA', 'BRAVO', 'CHARLIE', 'DELTA', 'ECHO']}*)

CHAPTER

THIRTEEN

SCRIPTS

These are pipeline scripts, not the Test Scripts (for those keep scrolling down).

13.1 Scripts

13.1.1 HKmonitor.py

TODO find HK files in a folder parse HK files plot HK parameters vs. time assemble all plots into a pdf file

DEBUG, calls nonexistent class LaTeX

Script to produce HK reports out of HK files in a folder. Aimed at quick inspection of data from Characterization and Calibration Campaigns of Euclid-VIS.

History

Created on Tue Mar 15 10:35:43 2016

@author: Ruyman Azzollini (MSSL)

13.1.2 quickds9.py

Wrap-up of ds9 to quickly load a number of images, for inspection.

History

Created on Thu Mar 17 13:18:10 2016

@author: Ruyman Azzollini

13.1.3 run_xtalk.py

Master Script to measure and report cross-talk levels among 12 ROE channels. Takes as input a data-set composed of 3x12 CCD images, corresponding to injecting a "ladder" of signal on each of the 12 channels, using the ROE-TAB.

Created on Thu Mar 22 16:17:39 2018

author Ruyman Azzollini

`vison.scripts.run_xtalk.`**`run_xtalk`**(*incat*, *inpath=''*, *respath=''*, *metafile=''*, *doCompute=False*)

13.1.4 vis_cosmetics_masker.py

Script to create cosmetics masks in VIS Ground Calibration Campaign.

Created on Wed Aug 1 11:02:00 2018

author Ruyman Azzollini

`vison.scripts.vis_cosmetics_masker.`**`do_Mask`**(*inputs*, *masktype*, *subbgd=True*, *normbygd=False*, *validrange=None*)

`vison.scripts.vis_cosmetics_masker.`**`pre_process`**(*FITS_list*, *subOffset=False*, *validrange=None*)

`vison.scripts.vis_cosmetics_masker.`**`read_OBSID_list`**(*ff*)

`vison.scripts.vis_cosmetics_masker.`**`run_maskmaker`**(*inputs*)

13.1.5 vis_explogs_merger.py

Created on Fri Feb 9 15:20:01 2018

@author: raf

`vison.scripts.vis_explogs_merger.`**`explog_merger`**(*ELlist*, *output='EXP_LOG_merged.txt'*, *elvis='6.5.X'*)

13.1.6 vis_genDataSet.py

Development: Creating Calibration Campaign Fake Data-set

Created on Tue Sep 05 16:07:00 2017

autor Ruyman Azzollini

`vison.scripts.vis_genDataSet.`**`datasetGenerator`**(*TestsSelector*, *doGenExplog*, *doGenHK*, *doGenFITS*, *outpath*, *elvis*, *CHAMBER*, *Nrows=0*)

`vison.scripts.vis_genDataSet.`**`genExpLog`**(*toGen*, *explogf*, *equipment*, *elvis='6.5.X'*, *CHAMBER=None*)

13.1.7 vis_load_DD.py

Loading a DataDict object for inspection.

Created on Wed Aug 01 10:00:00 2018

autor Ruyman Azzollini

13.1.8 vis_mkscripts.py

Automatically Generating Calibration Campaign Data Acquisition Scripts. Aimed at ELVIS.

Created on Fri Sep 08 12:03:00 2017

autor Ruyman Azzollini

13.1.9 vis_star_finder.py

Script to find point sources in VIS Ground Calibration Campaign. Used to 'prime' the position tables of point-source objects.

Created on Tue Jun 12 16:09:31 2018

author Ruyman Azzollini

`vison.scripts.vis_star_finder.`**`write_ID_chart`**(*filename*, *Quads*, *Starnames*)

13.1.10 v_ROE_LinCalib.py

Non-Linearity Calibration of ROE (on bench).

Created on Thu Mar 15 15:32:11 2018

author Ruyman Azzollini

`vison.scripts.v_ROE_LinCalib.`**`find_adu_levels`**(*qdata*, *Nlevels*, *debug=False*)

`vison.scripts.v_ROE_LinCalib.`**`run_ROE_LinCalib`**(*inputsfile*, *incatfile*, *datapath=''*, *respath=''*, *doExtractFits=True*, *dopolyRT=False*, *debug=False*)

13.1.11 v_ROETAB_LinCalib.py

Linearity Calibration of ROE-TAB.

Created on Tue Mar 27 14:42:00 2018 Modified on Fri Sep 14 10:53:00 2018

author Ruyman Azzollini

`vison.scripts.v_ROETAB_LinCalib.`**`filter_Voltage_uni`**(*rV*, *filt_kernel*)

`vison.scripts.v_ROETAB_LinCalib.`**`find_discrete_voltages_inwaveform`**(*rV*, *levels*, *filtered=None*, *debug=False*)

`vison.scripts.v_ROETAB_LinCalib.`**`load_WF`**(*WFf*, *chkNsamp=None*, *chkSampInter=None*)

`vison.scripts.v_ROETAB_LinCalib.`**`plot_waveform`**(*WF*, *disc_voltages=[]*, *figname=''*, *chan='Unknown'*)

`vison.scripts.v_ROETAB_LinCalib.`**`run_ROETAB_LinCalib`**(*inputsfile*, *incatfile*, *datapath=''*, *respath=''*, *doBayes=False*, *debug=False*)

CHAPTER

FOURTEEN

SUPPORT CODE

14.1 Support Code

14.1.1 context.py

Common Values which are used by functions and classes throughout pipeline.

Created on Tue Jan 16 10:53:40 2018

author Ruyman Azzollini

14.1.2 ET.py

Module to issue WARNING / ALERT phone calls to designated phone numbers. Uses Twilio.

'... E.T. phone home...'

Created on Thu Sep 14 10:13:12 2017

author raf

class `vison.support.ET.`**`ET`**

Class to do phone calls.

`dial_numbers`(*url*)

Dials one or more phone numbers from a Twilio phone number.

Parameters **`url`** – char, URL with the TwiML code that Twilio uses as instructions on call. Basically, it provides a message to be voiced, as intended.

`send_sms`(*body*)

`vison.support.ET.`**`grab_numbers_and_codes`**()

Retrieves phone numbers and access codes necessary to make the phone calls.

14.1.3 excel.py

Excel Files Interfaces.

Created on Mon Mar 26 12:07:54 2018

author Ruyman Azzollini

`vison.support.excel.`**`test0`**()

Just a dummy test to show we can use openpyxl

14.1.4 files.py

IO related functions.

requires PyFITS

requires NumPy

author Sami-Matias Niemi

`vison.support.files.`**`cPickleDump`**(*data*, *output*, *protocol=2*)
Dumps data to a cPickled file.

Parameters

- **`data`** – a Python data container
- **`output`** – name of the output file

Returns None

`vison.support.files.`**`cPickleDumpDictionary`**(*dictionary*, *output*, *protocol=2*)
Dumps a dictionary of data to a cPickled file.

Parameters

- **`dictionary`** – a Python data container does not have to be a dictionary
- **`output`** – name of the output file

Returns None

`vison.support.files.`**`cPickleRead`**(*file*)
Loads data from a pickled file.

`vison.support.files.`**`convert_fig_to_eps`**(*figname*)
Converts a figure to .eps. Returns new file name.

14.1.5 flags.py

Functions and variables related to flags for vison.

Created on Wed Sep 20 17:05:00 2017

author Ruyman Azzollini

class `vison.support.flags.`**`Flags`**(*indict=None*)

14.1.6 latex.py

Just a collection of LaTeX-generating functions for use in report.py

History

Created on Mon Jan 30 2017

author Ruyman Azzollini

`vison.support.latex.`**`generate_header`**(*test*, *model*, *author*, *reference='7-XXX'*)

`vison.support.latex.`**`replace_in_template`**(*texf*, *values*)

14.1.7 logger.py

These functions can be used for logging information.

> **Warning:** logger is not multiprocessing safe.

author Sami-Matias Niemi

version 0.3

class `vison.support.logger.`**`SimpleLogger`**(*filename*, *verbose=False*)
A simple class to create a log file or print the information on screen.

`write`(*text*)
Writes text either to file or screen.

`vison.support.logger.`**`f_text_wrapper`**(*msg*)

`vison.support.logger.`**`setUpLogger`**(*log_filename*, *loggername='logger'*)
Sets up a logger.

Param log_filename: name of the file to save the log.

Param loggername: name of the logger

Returns logger instance

14.1.8 report.py

LaTEx - PDF Reporting Utilities.

History

Created on Wed Jan 25 16:58:33 2017

author Ruyman Azzollini

class `vison.support.report.`**`Container`**

`add_to_Contents`(*item*)

class `vison.support.report.`**`Content`**(*contenttype=''*)

class `vison.support.report.`**`FigsTable`**(*FigsList*, *Ncols*, *figswidth*, *caption=None*)
Class to generate table of figures

`generate_Latex`()
Generates LaTeX as list of strings

class `vison.support.report.`**`Figure`**(*figpath*, *textfraction=0.7*, *caption=None*, *label=None*)

`generate_Latex`()
Generates LaTeX as list of strings.

class `vison.support.report.`**`Section`**(*keyword*, *Title=''*, *level=0*)

`generate_Latex`()

class `vison.support.report.`**`Table`**(*tableDict*, *formats={}*, *names=[]*, *caption=None*, *col_align=None*, *longtable=False*)

PENDING:

- adjust width of table to texwidth:

esizebox{ extwidth}{!}{

... end{tabular}}

- include option to rotate table to show in landscape

generate_Latex()
Generates LaTeX as list of strings.

class vison.support.report.**Text**(*text*)

generate_Latex()

14.1.9 utils.py

General Purpose Utilities

Created on Tue Apr 10 15:18:07 2018

author Ruyman Azzollini

14.1.10 vistime.py

Accesory library: time related operations

Created on Tue Oct 10 15:08:28 2017

author Ruyman Azzollini

vison.support.vistime.**get_dtobj**(*DT*)

vison.support.vistime.**get_time_tag**()

14.1.11 vjson.py

json files handling utilities.

Created on Tue Mar 27 14:25:43 2018

author Ruyman Azzollini

vison.support.vjson.**dumps_to_json**(*pydict*)

vison.support.vjson.**load_jsonfile**(*jsonfile*, *useyaml=False*)

vison.support.vjson.**save_jsonfile**(*pydict*, *jsonfile*)

CHAPTER

FIFTEEN

UNIT TESTING

15.1 Unit Testing

15.1.1 test_ccdpile.py

Unit-testing for CCDPile class.

Created on Mon May 7 09:47:07 2018

author Ruyman Azzollini

15.1.2 test_ccd.py

Unit-testing for CCD class.

Created on Mon May 7 09:47:07 2018

author Ruyman Azzollini

CHAPTER

SIXTEEN

TEST SCRIPTS

These are the scripts that hold the description, execution, data validation and analysis of the tests that make the campaign. They are served by the infrasctructure and tools provided by the pipeline.

WARNING: Currently most of the test scripts are largely meta-code, with the exception of very basic functionality used to generate acqusition scripts and validate the acquisitions, as listed in the Exposure Log, against the description of the test. The metacode has been included in the doc-strings for ease of browsing.

16.1 Charge Injection Scripts

16.1.1 Charge Injection Scripts

CHINJ01

VIS Ground Calibration TEST: CHINJ01

Charge injection calibration (part 1) Injection vs. IG1-IG2

Created on Tue Aug 29 17:36:00 2017

author Ruyman Azzollini

class `vison.inject.CHINJ01.`**`CHINJ01`**(*inputs*, *log=None*, *drill=False*, *debug=False*)

`build_scriptdict`(*diffvalues={}*, *elvis='6.5.X'*)
Builds CHINJ01 script structure dictionary.

#:param IDL: float, [V], value of IDL (Inject. Drain Low). #:param IDH: float, [V], Injection Drain High. #:param IG2: float, [V], Injection Gate 2. #:param IG1s: list of 2 floats, [V], [min,max] values of IG1. #:param id_delays: list of 2 floats, [us], injection drain delays. #:param toi_chinj: int, [us], TOI-charge injection. :param diffvalues: dict, opt, differential values.

`filterexposures`(*structure*, *explog*, *OBSID_lims*)

`meta_analysis`()
Find injection threshold: Min IG1 Plot and model charge injection vs. IG1 Find notch injection amount.

`old_basic_analysis`()
Basic analysis of data.

METACODE

```
f. e. ObsID:
    f.e.CCD:
        f.e.Q:
            extract average 2D injection pattern (and save)
            produce average profile along/across lines
            measure charge-inj. non-uniformity
            measure charge spillover into non-injection
            measure stats of injection (mean, med, std, min/max, percentiles)

plot average inj. profiles along lines f. each CCD, Q and IG1
    save as a rationalized set of curves
plot average inj. profiles across lines f. each CCD, Q and IG1
    save as a rationalized set of  curves

Report injection stats as a table/tables
```

set_inpdefaults(***kwargs*)

CHINJ02

VIS Ground Calibration TEST: CHINJ02

Charge injection calibration (part 2) Injection vs. IDL (injection threshold)

Created on Tue Aug 29 17:36:00 2017

author Ruyman Azzollini

class vison.inject.CHINJ02.**CHINJ02**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()

Basic analysis of data. AS IT IS, REPEATS WHAT'S DONE IN THE CHECK_DATA. CONSIDER MERGING/SKIPPING

METACODE

```
f. e. ObsID:
    f.e.CCD:
        f.e.Q:
            load average 2D injection pattern
            produce average profile along lines
            [measure charge-inj. non-uniformity]
            [produce average profile across lines]
            [measure charge spillover into non-injection]
            measure stats of injection (mean, med, std, min/max, percentiles)

[plot average inj. profiles along lines f. each CCD, Q and IG1]
[    save as a rationalized set of curves]
[plot average inj. profiles across lines f. each CCD, Q and IG1]
[    save as a rationalized set of  curves]

save&plot charge injection vs. IDL
report injection stats as a table
```

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)

Builds CHINJ02 script structure dictionary.

#:param IDLs: list of 2 ints, [V], [min,max] values of IDL (Inject. Drain Low). #:param IDH: int, [V], Injection Drain High. #:param id_delays: list of 2 ints, [us], injection drain delays. #:param toi_chinj: int, [us], TOI-charge injection. :param diffvalues: dict, opt, differential values.

extract_data()

NEEDED? Could be merged with basic_analysis

METACODE

```
Preparation of data for further analysis:

f.e. ObsID:
    f.e.CCD:
        f.e.Q:
            subtract offset
            extract average 2D injection pattern and save
```

filterexposures(*structure*, *explog*, *OBSID_lims*)

meta_analysis()

Finds the Injection Threshold for each CCD half.

METACODE

```
f.e.CCD:
    f.e.Q:
        load injection vs. IDL cuve
        find&save injection threshold on curve

report injection threshold as a table
```

set_inpdefaults(***kwargs*)

16.2 Dark Scripts

16.2.1 "Dark Acquisitions" Scripts

BIAS01

TEST: BIAS01

Bias-structure/RON analysis script

Created on Tue Aug 29 16:53:40 2017

author Ruyman Azzollini

class vison.dark.BIAS01.**BIAS01**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()

BIAS01: Basic analysis of data.

METACODE

```
f. e. ObsID:
   f.e.CCD:

       load ccdobj of ObsID, CCD
```

```
        with ccdobj, f.e.Q:
            produce a 2D poly model of bias, save coefficients
            produce average profile along rows
            produce average profile along cols
            # save 2D model and profiles in a pick file for each OBSID-CCD
            measure and save RON after subtracting large scale structure

plot RON vs. time f. each CCD and Q
plot average profiles f. each CCD and Q (color coded by time)
```

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)
Builds BIAS01 script structure dictionary.

###:param N: integer, number of frames to acquire. :param diffvalues: dict, opt, differential values. :param elvis: char, ELVIS version.

filterexposures (*structure*, *explog*, *OBSID_lims*)

meta_analysis ()
METACODE

```
f. each CCD:
   stack all ObsIDs to produce Master Bias
   f. e. Q:
       measure average profile along rows
       measure average profile along cols
plot average profiles of Master Bias(s) f. each CCD,Q
(produce table(s) with summary of results, include in report)
save Master Bias(s) (3 images) to FITS CDPs
show Master Bias(s) (3 images) in report
save name of MasterBias(s) CDPs to DataDict, report
```

prep_data ()
BIAS01: Preparation of data for further analysis. Calls task.prepare_images().

Applies: offset subtraction cosmetics masking

class vison.dark.BIAS01.**Test** (*methodName='runTest'*)
Unit tests for the BIAS01 class.

test_check_data ()

Returns None

DARK01

TEST: DARK01

"Dark Current" analysis script

Created on Tue Aug 29 17:21:00 2017

author Ruyman Azzollini

class vison.dark.DARK01.**DARK01** (*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis ()
DARK01: Basic analysis of data.

METACODE

```
f. e. ObsID:
    f.e.CCD:
        f.e.Q:
            produce mask of hot pixels
            count hot pixels / columns
            produce a 2D poly model of masked-image, save coefficients
            produce average profile along rows
            produce average profile along cols
            measure and save RON after subtracting large scale structure
        save 2D model and profiles in a pick file for each OBSID-CCD

plot average profiles f. each CCD and Q (color coded by time)
```

build_scriptdict (*diffvalues={}, elvis='6.5.X'*)
Builds DARK01 script structure dictionary.

Parameters **diffvalues** – dict, opt, differential values.

filterexposures (*structure, explog, OBSID_lims*)

meta_analysis ()
METACODE

```
f. each CCD:
    f. e. Q:
        stack all ObsIDs to produce Master Dark
        produce mask of hot pixels / columns
        count hot pixels / columns
        measure average profile along rows
        measure average profile along cols

plot average profiles of Master Bias f. each CCD,Q
show Master Dark (images), include in report
report stats of defects, include in report
save name of MasterDark to DataDict, report
save name of Defects in Darkness Mask to DD, report
```

prep_data ()
DARK01: Preparation of data for further analysis. Calls task.prepare_images().

Applies: offset subtraction [BIAS SUBTRACTION] cosmetics masking

16.3 Flat-Illumination Scripts

16.3.1 Flat-Illumination Scripts

FLAT0X

VIS Ground Calibration TEST: FLAT0X

Flat-fields acquisition / analysis script

Created on Tue Aug 29 17:32:52 2017

author Ruyman Azzollini

class `vison.flat.FLAT0X.`**`FLAT0X`**(*inputs*, *log=None*, *drill=False*, *debug=False*)

`build_scriptdict`(*diffvalues={}*, *elvis='6.5.X'*)
Builds FLAT0X script structure dictionary.

Parameters **`diffvalues`** – dict, opt, differential values.

`do_indiv_flats`()
METACODE

```
Preparation of data for further analysis and
produce flat-field for each OBSID.

f.e. ObsID:
    f.e.CCD:

        load ccdobj

        f.e.Q:

            model 2D fluence distro in image area
            produce average profile along rows
            produce average profile along cols

        save 2D model and profiles in a pick file for each OBSID-CCD
        divide by 2D model to produce indiv-flat
        save indiv-Flat to FITS(?), update add filename

plot average profiles f. each CCD and Q (color coded by time)
```

`do_master_flat`()
METACODE

```
Produces Master Flat-Field

f.e.CCD:
    f.e.Q:
        stack individual flat-fields by chosen estimator
save Master FF to FITS
measure PRNU and
report PRNU figures
```

`do_prdef_mask`()
METACODE

```
Produces mask of defects in Photo-Response
Could use master FF, or a stack of a subset of images (in order
to produce mask, needed by other tasks, quicker).

f.e.CCD:
    f.e.Q:
        produce mask of PR defects
        save mask of PR defects
        count dead pixels / columns

report PR-defects stats
```

`filterexposures`(*structure*, *explog*, *OBSID_lims*)

prepare_images()
FLAT0X: Preparation of data for further analysis. Calls task.prepare_images().

Applies: offset subtraction [bias structure subtraction, if available] cosmetics masking

set_inpdefaults(***kwargs*)

NL01

VIS Ground Calibration TEST: NL01

End-To-End Non-Linearity Curve

Tasks:

- Select exposures, get file names, get metadata (commandig, HK).
- Check exposure time pattern matches test design.
- Check quality of data (rough scaling of fluences with Exposure times).
- Subtract offset level.
- Divide by Flat-field.
- **Synoptic analysis:** fluence ratios vs. extime ratios >> non-linearity curve
- extract: Non-Linearity curve for each CCD and quadrant
- produce synoptic figures
- Save results.

Created on Mon Apr 3 17:38:00 2017

author raf

class vison.flat.NL01.**NL01**(*inputs*, *log=None*, *drill=False*, *debug=False*)

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)
Builds NL01 script structure dictionary.

#:param expts: list of ints [ms], exposure times. #:param exptinter: int, ms, exposure time of interleaved source-stability exposures. #:param frames: list of ints, number of frames for each exposure time. #:param wavelength: int, wavelength. Default: 0 (Neutral Density Filter) :param diffvalues: dict, opt, differential values.

do_satCTE()
METACODE

```
select ObsIDs with fluence(exptime) >~ 0.5 FWC

f.e. ObsID:
    CCD:
        Q:
            measure CTE from amount of charge in over-scan relative to fluence

f.e. CCD:
    Q:
        get curve of CTE vs. fluence
        measure FWC from curve in ADU

report FWCs in electrons [via gain in inputs] f.e. CCD, Q (table)
```

extract_stats()

Performs basic analysis: extracts statistics from image regions to later build NLC.

METACODE

```
create segmentation map given grid parameters

f.e. ObsID:
    f.e.CCD:
        f.e.Q:
            f.e. "img-segment": (done elsewhere)
                measure central value
                measure variance
```

filterexposures(*structure*, *explog*, *OBSID_lims*)

Loads a list of Exposure Logs and selects exposures from test PSF0X.

The filtering takes into account an expected structure for the acquisition script.

The datapath becomes another column in DataDict. This helps dealing with tests that run overnight and for which the input data is in several date-folders.

prep_data()

Takes Raw Data and prepares it for further analysis.

METACODE

```
f.e. ObsID:
    f.e.CCD:
        f.e.Q:
            subtract offset
        opt: [sub bias frame]
        opt: [divide by FF]
        opt: [mask-out defects]
```

produce_NLCs()

METACODE

```
Obtains Best-Fit Non-Linearity Curve

f.e. CCD:
    f.e. Q:

        [opt] apply correction for source variability (interspersed exposure
          with constant exptime)
        Build NL Curve (NLC) - use stats and exptimes
        fit poly. shape to NL curve

plot NL curves for each CCD, Q
report max. values of NL (table)
```

PTC0X

VIS Ground Calibration TEST: PTC_0X

Photon-Transfer-Curve Analysis PTC01 - nominal temperature and wavelength PTC02 - alternative temperatures / wavelengths

Tasks:

- Select exposures, get file names, get metadata (commandig, HK).
- Check exposure time pattern matches test design.
- Check quality of data (rough scaling of fluences with Exposure times).
- Subtract pairs of exposures with equal fluence
- **Synoptic analysis:** variance vs. fluence variance(binned difference-frames) vs. fluence
- extract: RON, gain, gain(fluence)
- produce synoptic figures
- Save results.

Created on Mon Apr 3 17:00:24 2017

author raf

class `vison.flat.PTC0X.`**`PTC0X`**(*inputs*, *log=None*, *drill=False*, *debug=False*)

`build_scriptdict`(*diffvalues={}*, *elvis='6.5.X'*)

Builds PTC0X script structure dictionary.

#:param exptimes: list of ints [ms], exposure times. #:param frames: list of ints, number of frames for each exposure time. #:param wavelength: int, wavelength. Default: 800 nm. :param diffvalues: dict, opt, differential values.

`extract_PTC`()

Performs basic analysis of images:

- builds PTC curves: both on non-binned and binned images

METACODE

```
create list of OBSID pairs

create segmentation map given grid parameters

f.e. OBSID pair:
    CCD:
        Q:
            subtract CCD images
            f.e. segment:
                measure central value
                measure variance
```

`filterexposures`(*structure*, *explog*, *OBSID_lims*)

`meta_analysis`()

Analyzes the variance and fluence: gain, and gain(fluence)

METACODE

```
f.e. CCD:
    Q:
        (using stats across segments:)
        fit PTC to quadratic model
        solve for gain
        solve for alpha (pixel-correls, Guyonnet+15)
        solve for blooming limit (ADU)
            convert bloom limit to electrons, using gain
```

```
plot PTC curves with best-fit f.e. CCD, Q
report on gain estimates f. e. CCD, Q (table)
report on blooming limits (table)
```

set_inpdefaults(***kwargs*)

16.4 Point-Source Scripts

16.4.1 Point-Source Scripts

FOCUS00

TEST: FOCUS00

Focus analysis script

Tasks:

- Select exposures, get file names, get metadata (commandig, HK).
- Check quality of data (integrated fluxes are roughly constant, matching expected level).
- Subtract offset level.
- Divide by Flat-field.
- **Crop stamps of the sources on each CCD/Quadrant.**
  - save snapshot figures of sources.
- **for each source (5 x Nquadrants):**
  - measure shape using Gaussian Fit
- Find position of mirror that minimizes PSF sizes
- **Produce synoptic figures:** source size and ellipticity across combined FOV (of 3 CCDs)
- Save results.

Created on Mon Apr 03 16:21:00 2017

author Ruyman Azzollini

class vison.point.FOCUS00.**FOCUS00**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()
This is just an assignation of values measured in check_data.

build_scriptdict(*diffvalues={}*, *elvis='6.5.X'*)
Builds FOCUS00 script structure dictionary.

#:param wavelength: int, [nm], wavelength. #:param exptime: int, [ms], exposure time. :param diffvalues: dict, opt, differential values.

filterexposures(*structure*, *explog*, *OBSID_lims*)

lock_on_stars()

meta_analysis()

prep_data()

PSF0X

TEST: PSF0X

PSF vs. Fluence, and Wavelength PSF01 - nominal temperature PSF02 - alternative temperatures

Tasks:

- Select exposures, get file names, get metadata (commandig, HK).
- Check exposure time pattern matches test design.
- Check quality of data (rough scaling of fluences with Exposure times).
- Subtract offset level.
- Divide by Flat-field.
- **Crop stamps of the sources on each CCD/Quadrant.**
    - save snapshot figures of sources.
- **for each source:**
    - measure shape using weighted moments
    - measure shape using Gaussian Fit
    - Bayesian Forward Modelling the optomechanic+detector PSF
- Produce synoptic figures.
- Save results.

Created on Thu Dec 29 15:01:07 2016

author Ruyman Azzollini

16.5 Trap-Pumping Scripts

16.5.1 Trap-Pumping Scripts

TP01

VIS Ground Calibration TEST: TP01

Trap-Pumping calibration (vertical)

Created on Tue Aug 29 17:37:00 2017

author Ruyman Azzollini

class vison.pump.TP01.**TP01**(*inputs*, *log=None*, *drill=False*, *debug=False*)

basic_analysis()

Basic analysis of data.

METACODE

```
f. e. ObsID [there are different TOI_TP and TP-patterns]:
    f.e.CCD:
        f.e.Q:
            load "map of relative pumping"
            find_dipoles:
                x, y, rel-amplitude, orientation

produce & report:
    map location of dipoles
    PDF of dipole amplitudes (for N and S)
    Counts of dipoles (and N vs. S)
```

build_scriptdict (*diffvalues={}*, *elvis='6.5.X'*)

extract ()

Obtain maps of dipoles.

METACODE

```
f.e. id_delay (there are 2):
    f.e. CCD:
        f.e. Q:
            produce reference non-pumped injection map

f. e. ObsID:
    f.e. CCD:

        load ccdobj
        f.e.Q.:
            divide ccdobj.Q by injection map

        save dipole map and store reference
```

filterexposures (*structure*, *explog*, *OBSID_lims*)

meta_analysis ()

Meta-analysis of data:

> Try to identify tau and pixel-phase location for each trap. Need to associate dipoles across TOI_TPs and TP-patterns

METACODE

```
across TOI_TP, patterns:
    build catalog of traps: x,y,I-phase, Amp
    from Amp(TOI) -> tau, Pc

Report on :
    Histogram of Taus
    Histogram of Pc (capture probability)
    Histogram of I-phases (larger phases should have more traps,
                        statistically) -> check

    Total Count of Traps
```

set_inpdefaults (***kwargs*)

TP02

VIS Ground Calibration TEST: TP02

Trap-Pumping calibration (serial)

Created on Tue Aug 29 17:38:00 2017

author Ruyman Azzollini

class `vison.pump.TP02.`**`TP02`**(*inputs*, *log=None*, *drill=False*, *debug=False*)

`basic_analysis`()

Basic analysis of data.

METACODE

```
f. e. ObsID [there are different TOI_TP and TP-patterns]:
    f.e.CCD:
        f.e.Q:
            load raw 1D map of relative pumping (from extract_data)
            identify dipoles:
                x, rel-amplitude, orientation (E or W)

produce & report:
    map location of dipoles
    PDF of dipole amplitudes (for E and W)
    Counts of dipoles (and E vs. W)
```

`build_scriptdict`(*diffvalues={}*, *elvis='6.5.X'*)

`extract`()

Obtain Maps of Serial Dipoles.

`filterexposures`(*structure*, *explog*, *OBSID_lims*)

`meta_analysis`()

Meta-analysis of data:

Try to identify tau and pixel-phase location for each trap. Need to associate dipoles across TOI_TPs and TP-patterns

METACODE

```
across TOI_TP, patterns:
    build catalog of traps: x,y,R-phase, amp(dwell)
    from Amp(dwell) -> tau, Pc

Report on :
   Histogram of Taus
   Histogram of Pc (capture probability)
   Histogram of R-phases

   Total Count of Traps
```

`set_inpdefaults`(***kwargs*)

16.6 Other Test Scripts

16.6.1 Other Scripts

MOT_FF

VIS Ground Calibration TEST: MOT_FF

Brighter-Fatter Analysis Using data from test PTC01 (via BF01)

Hard Edge Response in serial / parallel Bit Correlations (ADC health)

Created on Tue Jul 31 18:04:00 2018

author raf

class `vison.other.MOT_FF.`**`MOT_FF`**(*inputs*, *log=None*, *drill=False*, *debug=False*)

`extract_HER`()

PERSIST01

VIS Ground Calibration TEST: PERSIST01

CCD Persistence test

Created on Tue Aug 29 17:39:00 2017

author Ruyman Azzollini

class `vison.other.PERSIST01.`**`PERSIST01`**(*inputs*, *log=None*, *drill=False*, *debug=False*)

`basic_analysis`()

Basic analysis of data.

METACODE

```
f.e.CCD:
    f.e.Q:
        use SATURATED frame to generate pixel saturation MASK
        measure stats in pix satur MASK across OBSIDs
         (pre-satur, satur, post-satur)
```

`build_scriptdict`(*diffvalues={}*, *elvis='6.5.X'*)

Builds PERSISTENCE01 script structure dictionary.

Parameters

- **`exptSATUR`** – int, saturation exposure time.
- **`exptLATEN`** – int, latency exposure time.
- **`diffvalues`** – dict, opt, differential values.

`check_data`()

PERSIST01: Checks quality of ingested data.

METACODE

```
check common HK values are within safe / nominal margins
check voltages in HK match commanded voltages, within margins

f.e.ObsID:
    f.e.CCD:
        f.e.Q.:
            measure offsets in pre-, over-
            measure std in pre-, over-
            measure fluence in apertures around Point Sources

assess std in pre- (~RON) is within allocated margins
assess offsets in pre-, and over- are equal, within allocated  margins
assess fluence is ~expected within apertures (PS) for each frame (pre-satur,␣
↪satur, post-satur)


plot point source fluence vs. OBSID, all sources
[plot std vs. time]

issue any warnings to log
issue update to report
```

filterexposures (*structure*, *explog*, *OBSID_lims*)

meta_analysis ()

Meta-analysis of data.

METACODE

```
f.e.CCD:
    f.e.Q:
        estimate delta-charge_0 and decay tau from time-series

report:
    persistence level (delta-charge_0) and time constant
```

prep_data ()

PERSIST01: Preparation of data for further analysis. Calls task.prepare_images().

Applies: offset subtraction cosmetics masking

set_inpdefaults (***kwargs*)

CHAPTER

SEVENTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W