APERO - A PipelinE to Reduce Observations

Contents

Latest version

Back to top

- master (long term stable) V0.5.000
- developer (tested) V0.6.001
- working (untested) V0.6.001

Pre-Installation

Back to top

1. clone this repository

```
git clone https://github.com/njcuk9999/apero-drs
```

This may take some time (in future most of the data required will be a separate download), and we still have many (now redundant) files from the spirou_py3 repository

Upgrading from the spirou_py3

This is not recommended. A clean install is recommended.

Note if you must do this you need to redirect github:

```
cd ./spirou-drs
git remote set-url origin https://github.com/njcuk9999/apero-drs
mv ../spirou-drs ../apero-drs
```

2. change to the repository directory

```
cd ./apero-drs
```

3. checkout the correct branch

- For Master version:

```
git checkout master
```

- For Developer version:

```
git checkout developer
```

- For Working version:

```
git checkout working
```

Installation

Back to top

(Currently only for `developer` and `working` versions)

Make sure pre-installation is done first!

1. run the installation script

```
python setup/install.py
```

2. Follow the step-by-step guide:

A. `User config path` : This is the path where your configuration will be saved. If it doesn't exist you will be prompted to create it. (This will be referred to as `DRS_UCONFIG` from now on (default is `/home/user/apero/`)

B. `Settings for {INSTRUMENT}` : Install {INSTRUMENT}. If yes it will install the instrument if not then it will not install the instrument. Currently only SPIRou is supported.

C. `Set up paths individually` . If [Y]es it will allow you to set each path separately (i.e. for raw, tmp, reduced, calibDB etc). If no you will just set one path and all folders (raw, tmp, reduced, calibDB etc)) will be created under this directory.

D. `Setting the directories` (either one directory or each of the sub-directories required - i.e. raw, tmp, reduced, calibDB etc)

E. `Clean install?` **WARNING**: If you type [Y]es you will be prompted (later) to reset the directories this means any previous data in these directories will be removed. Note you can always say later to individual cases.

F. This process will then repeat for all instruments. **NOTE: Currently only SPIRou is supported**

G. In the `Copying files` step if you asked for a clean install if directories are not empty you will be prompted to reset them one-by-one (**NOTE: THIS WILL REMOVE ALL DATA FROM THIS SPECIFIC DIRECTORY)**

3. Installation is then complete

4. Running apero

To run apero you need to do **one** of the following

NOTE: these three are equivalent only do **one**

i) alias to apero to your startup script (RECOMMENDED)

For example

`alias apero "source {DRS_UCONFIG}/config/apero.{SYSTEM}.setup"` (tcsh/csh) `alias apero=""source {DRS_UCONFIG}/config/apero.{SYSTEM}.setup"` (bash) to `~/.bashrc` or `~/.bash_profile` or `~/.tcshrc` or `~/.profile`

and type `apero` every time you open a new terminal

ii) source environmental variables directly

`source {DRS_UCONFIG}/config/apero.{SYSTEM}.setup` and type this command every time you open a new terminal

where:

- `{DRS_UCONFIG}` is the config path set up in step **2A**
- `{SYSTEM}` is either `bash` or `sh` depending on your shell

iii) add the contents of `{DRS_UCONFIG}/config/apero.{SYSTEM}.setup` to your startup script

i.e. one of the following `~/.bashrc`, `~/.bash_profile`, `~/.tcshrc`, `~/.profile` (apero will be ready to use in every new terminal).

For example adding to `~/.bashrc`:

```
# setup paths
export PATH="/scratch/apero_dev/apero/tools/bin":$PATH
export PATH="/scratch/apero_dev/bin":$PATH
export PATH="/scratch/apero_dev/":$PATH

# setup up python path
export PYTHONPATH="/scratch/apero_dev/apero/tools/bin":$PYTHONPATH
export PYTHONPATH="/scratch/apero_dev/bin":$PYTHONPATH
export PYTHONPATH="/scratch/apero_dev/":$PYTHONPATH

# setup drs config path
export DRS_UCONFIG="/home/user/apero/config/"

# force numpy  to only use 1 core max
export OPENBLAS_NUM_THREADS=1
export MKL_NUM_THREADS=1

# run the validation script for SPIROU
python /scratch/apero_dev/apero/tools/bin/validate.py SPIROU
```

TODO

Back to top

- finish `obj_spec_spirou` and `obj_pol_spirou` [Do not use them now]
- output files like CFHT (e.fits, p.fits, v.fits etc)
- data separate download from DRS
- setup instrument
- move `object_query_list.fits` to `calibDB`
- write documentation and paper
- go through all summary plots and decide which plots, write figure captions, improve plots, write quality control description, decide which header keys to print
- add `plot== 3` (all debug plots shown) and `plot==4` (all debug plots saved) modes
- display func for all functions
- add raw (via run) to file explorer
- Windows compatibility
- add doc strings to all functions, descriptions to all constants, review all constant min/max /dtypes
- add more debug printouts
- deal with all python warnings
- add EA mask generation from templates
- add EA template matching

Currently known issues

Back to top

- wave solution sometimes using HC wave solution sometimes FP wave solution - WHY?
- telluric correction is slightly worse than before (due to wavelength solution?)
- CCF still showing problems with noise (maybe same problem as telluric correction?)
- BERV file gets locked (Ctrl+C to unlock) - WHY?
- index.fits not found - during parallel writes to index.fits - locking system is flawed - is this

fixed?
- file explorer is broken (needs updating)
-

Using APERO

Back to top

You can use apero to individually run recipes or process a set of files

Using apero individually

Recipes (the scripts to run) are stored in the `./apero/recipes/{instrument}` path once installed these recipes are copied to the `./bin/` directory and can be used from the command line or in python or in ipython (recommended).

i.e. from the shell

```
cal_badpix_spirou.py --flatfiles file1.fits --darkfiles file2.fits
```

i.e. using python

```
python cal_badpix_spirou.py --flatfiles file1.fits --darkfiles file2.fits
```

i.e. in ipython

```
run apero/recipes/spirou/cal_badpix_spirou.py --flatfiles file1.fits --darkfiles f
```

i.e. in a python script

```
import cal_badpix_spirou
```

```
ll = cal_badpix_spirou.main('night_name', flatfiles='file1.fits', darkfiles='file2
```

NOTE: there is a --help option available for every recipe

Using `processing.py`

`processing.py` can be used in a few different ways but always requires the following

1. The instrument (`SPIROU`)
2. The run file to execute

i.e.

```
apero/tools/bin/processing.py SPIROU limited_run.ini
```

The processing run files (`{RUN_FILE}`)

These are located in the `{DRS_DATA_RUNS}` (default= `/data/runs/`) directory. They can be used in two ways

1) Process automatically

By default it processes every night and every file that can be found in the `{DRS_DATA_RAW}` (default= `/data/raw/`) directory. One can turn on specific nights to process in several ways (a) setting the `NIGHT_NAME` in the selected `{RUN_FILE}` (b) adding a night to the `BNIGHTNAMES` (blacklist = reject) or `WNIGHTNAMES` (whitelist = keep) (c) adding an extra argument to `processing.py` (`--nightname` , `--bnightnames` , `--wnightnames`)

One can also just process a single file by adding an extra argument to `processing.py` (`--filename`)

One can also tell the recipe to only process specific targets (when the recipes can accept targets -- i.e. extraction, telluric fitting, CCF) by changing the `TELLURIC_TARGETS` key for tellurics or the `SCIENCE_TARGETS` key for science objects in the `{RUN_FILE}`

For processing automatically `id00000` should be set to one of the sequence names (see below) -- note that the id number should always be unique.

Note one can turn on/off recipes in a sequence by setting `RUN_{RECIPE}` to False, or skip files that already exist by setting `SKIP_{RECIPE}` to False - these will only be affected if `{RECIPE}` is in the sequence.

Note that all found filenames will be processed in parallel if `CORES` set to a number greater than 1.

2) Process a set of specific instructions

One can also define a specific set of instructions (similar to just running recipes individually), but batch them and parallize them (if `CORES` set to a number greater than 1). They will be processed in the order given by the id number -- note that the id number should always be unique.

i.e.

```
id00001 = cal_ccf_spirou.py 2018-08-05 2295651o_pp_e2dsff_tcorr_AB.fits --mask mas
id00002 = cal_ccf_spirou.py 2018-08-05 2295652o_pp_e2dsff_tcorr_AB.fits --mask mas
id00004 = cal_ccf_spirou.py 2018-08-05 2295653o_pp_e2dsff_tcorr_AB.fits --mask mas
id00005 = cal_ccf_spirou.py 2018-08-05 2295654o_pp_e2dsff_tcorr_AB.fits --mask mas
```

The available sequences

Sequences are defined in the `apero/core/instruments/{INSTRUMENT}/recipe_definitions.py` script. These enable the user to quickly process sets of data in specific orders based on their needs.

Note these can be combined in any order the user wants but some assume others have been completed first (in terms of files needed). i.e.

```
id00001 = master_run
id00002 = calib_run
id00003 = tellu_run
id00004 = science_run
```

Currently defined sequences are:

1. `full_run`

```
cal_preprocessing
cal_dark_master
cal_badpix [master night]
cal_loc [DARK_FLAT; master night]
cal_loc [FLAT_DARK; master night]
cal_shape_master
cal_badpix [every night]
cal_loc [DARK_FLAT; every night]
cal_loc [FLAT_DARK; every night]
cal_shape [every night]
cal_ff [every night]
cal_thermal [every night]
cal_wave [HCONE_HCONE + FP_FP; every night]
cal_extract [OBJ_DARK + OBJ_FP; every night; ALL OBJECTS]
obj_mk_tellu_db
obj_fit_tellu_db
cal_ccf [OBJ_DARK + OBJ_FP; fiber=AB; every night]
```

2. `limited_run`

Similar to `full_run` but uses the `{TELLURIC_TARGETS}` and `{SCIENCE_TARGETS}` to filter the objects processed

```
cal_preprocessing
cal_dark_master
cal_badpix [master night]
cal_loc [DARK_FLAT; master night]
cal_loc [FLAT_DARK; master night]
cal_shape_master
cal_badpix [every night]
cal_loc [DARK_FLAT; every night]
cal_loc [FLAT_DARK; every night]
cal_shape [every night]
cal_ff [every night]
cal_thermal [every night]
cal_wave [HCONE_HCONE; every night]
cal_wave [HCONE_HCONE + FP_FP; every night]
cal_extract [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
cal_extract [OBJ_DARK + OBJ_FP; every night; SCIENCE_TARGETS]
```

```
obj_mk_tellu [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
obj_fit_tellu [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
obj_mk_template [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
obj_mk_tellu [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
obj_fit_tellu [OBJ_DARK + OBJ_FP; every night; SCIENCE_TARGETS]
obj_mk_template [OBJ_DARK + OBJ_FP; every night; SCIENCE_TARGETS]
obj_fit_tellu [OBJ_DARK + OBJ_FP; every night; SCIENCE_TARGETS]
cal_ccf [OBJ_DARK + OBJ_FP; fiber=AB; every night; SCIENCE_TARGETS]
```

3. `master_run`

Only run the master recipes

```
cal_preprocessing
cal_dark_master
cal_badpix [master night]
cal_loc [DARK_FLAT; master night]
cal_loc [FLAT_DARK; master night]
cal_shape_master
```

4. `calib_run`

Only run the nightly calibration sequences and make a complete calibration database. (assumes that the master run is done i.e. `master_run`)

```
cal_badpix [every night]
cal_loc [DARK_FLAT; every night]
cal_loc [FLAT_DARK; every night]
cal_shape [every night]
cal_ff [every night]
cal_thermal [every night]
cal_wave [HCONE_HCONE; every night]
cal_wave [HCONE_HCONE + FP_FP; every night]
```

5. `tellu_run`

Only run the steps required to process `{TELLURIC_TARGETS}` and make the telluric database. (assumes that calibrations have been done i.e. `calib_run`)

```
cal_extract [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
obj_mk_tellu [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
obj_fit_tellu [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
obj_mk_template [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
obj_mk_tellu [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
```

6. `science_run`

Only run the steps required to process `{SCIENCE_TARGETS}` (assumes that calibrations and tellurics have been done i.e. `calib_run` and `tellu_run`)

```
cal_extract [OBJ_DARK + OBJ_FP; every night; SCIENCE_TARGETS]
obj_fit_tellu [OBJ_DARK + OBJ_FP; every night; SCIENCE_TARGETS]
obj_mk_template [OBJ_DARK + OBJ_FP; every night; SCIENCE_TARGETS]
obj_fit_tellu [OBJ_DARK + OBJ_FP; every night; SCIENCE_TARGETS]
cal_ccf [OBJ_DARK + OBJ_FP; fiber=AB; every night; SCIENCE_TARGETS]
```

APERO run order

Back to top

As mentioned above this depends on what sequence you wish to use but as an overview the steps are as follows

1) Choose a master night

(i.e. 2018-09-25)

If using the processing script one must set this in that file else when choosing arguments one must use them from the master night choosen.

Note one has to run `cal_badpix` and `cal_loc` calibrations for the master night in order to run the shape master recipe.

2. Run all the preprocessing

Note one must preprocess ALL nights for the master to work) - it will only combine darks(for the master dark) and fps (for the master shape) from preprocessed data.

3. Run the master sequence i.e.

```
cal_preprocessing
cal_dark_master
cal_badpix [master night]
cal_loc [DARK_FLAT; master night]
cal_loc [FLAT_DARK; master night]
cal_shape_master
```

4. Run the night sequences

These must be in this order but could be night-by-night or all of one then all of the other) - the order here only matters when a file is missed/corrupt or does not pass quality control. If all badpix are run first then the loc will have the best chance at having bad pixel correction from a night close to it. If one runs night by night then the next step will only have access to calibrations from nights already processed.

Note again one should extract all telluric stars and run the telluric sequence (to create a telluric database) BEFORE correcting any science extraction.

The calibration sequence is as follows:

```
cal_badpix [every night]
cal_loc [DARK_FLAT; every night]
cal_loc [FLAT_DARK; every night]
cal_shape [every night]
cal_ff [every night]
cal_thermal [every night]
cal_wave [HCONE_HCONE + FP_FP; every night]
```

The telluric star sequence is as follows:

```
cal_extract [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
```

```
obj_mk_tellu [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
obj_fit_tellu [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
obj_mk_template [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
obj_mk_tellu [OBJ_DARK + OBJ_FP; every night; TELLURIC_TARGETS]
```

The science star sequence is as follows:

```
cal_extract [OBJ_DARK + OBJ_FP; every night; SCIENCE_TARGETS]
obj_fit_tellu [OBJ_DARK + OBJ_FP; every night; SCIENCE_TARGETS]
obj_mk_template [OBJ_DARK + OBJ_FP; every night; SCIENCE_TARGETS]
obj_fit_tellu [OBJ_DARK + OBJ_FP; every night; SCIENCE_TARGETS]
cal_ccf [OBJ_DARK + OBJ_FP; fiber=AB; every night; SCIENCE_TARGETS]
```

APERO outputs

Back to top

Files

RAW FILES

- `4096 x 4096`
- PP files use `DPRTYPE` to identify files

DPRTYPE	SBCCAS_P	SBCREF_P	SBCALI_P	OBSTYPE	T
DARK_DARK_INT	pos_pk	pos_pk	P4	DARK	C
DARK_DARK_TEL	pos_pk	pos_pk	P5	DARK	C
OBJ_DARK	pos_pk	pos_pk	?	OBJECT	T
DARK_DARK_SKY	pos_pk	pos_pk	?	OBJECT	S

DPRTYPE	SBCCAS_P	SBCREF_P	SBCALI_P	OBSTYPE	
OBJ_FP	pos_pk	pos_fp	?	OBJECT	T
FP_FP	pos_fp	pos_fp	?	ALIGN	C
FLAT_DARK	pos_wl	pos_pk	?	FLAT	C
DARK_FLAT	pos_pk	pos_wl	?	FLAT	C
FLAT_FLAT	pos_wl	pos_wl	?	FLAT	C
HCONE_DARK	pos_hc1	pos_pk	?	COMPARISON	C
DARK_HCONE	pos_pk	pos_hc1	?	COMPARISON	C
HCONE_HCONE	pos_hc1	pos_hc1	?	COMPARISON	C

Recipes

Preprocessing Recipe

Cleans file of detector effects.

Run:

```
cal_preprocessing_spirou.py [DIRECTORY] [RAW_FILES]
```

Optional Arguments:

```
    --skip, --debug, --listing, --listingall, --version, --info,
    --program, --idebug, --breakpoints, --quiet, --help
```

Output Dir:

```
DRS_DATA_WORKING   \\ default: "tmp" directory
```

Output files:

```
{ODOMETER_CODE}_pp.fits  \\ preprocessed files (4096x4096)
```

Plots:

None

Dark Master Recipe

Collects all dark files and creates a master dark image to use for correction.

Run:

```
cal_dark_master_spirou.py
```

Optional Arguments:

```
    --filetype, --database, --plot,
    --debug, --listing, --listingall, --version, --info,
    --program, --idebug, --breakpoints, --quiet, --help
```

Output Dir:

```
DRS_DATA_REDUC   \\ default: "reduced" directory
```

Calibration database entry:

```
DARKM {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
```

Output files:

```
{ODOMETER_CODE}_pp_dark_master.fits  \\ dark master file (4096x4096) + FITS-TABLE
```

Plots:

None

Notes:

Does not require a master night choice - finds darks from all preprocessed nights.

Bad Pixel Correction Recipe

Creates a bad pixel mask for identifying and deal with bad pixels.

Run:

```
cal_badpix_spirou.py [DIRECTORY] -flatfiles [FLAT_FLAT] -darkfiles [DARK_DARK_TEL]
cal_badpix_spirou.py [DIRECTORY] -flatfiles [FLAT_FLAT] -darkfiles [DARK_DARK_INT]
```

Optional Arguments:

```
    --database, --combine, --flipimage, --fluxunits, --plot, --resize,
    --debug, --listing, --listingall, --version, --info,
    --program, --idebug, --breakpoints, --quiet, --help
```

Output Dir:

```
DRS_DATA_REDUC   \\ default: "reduced" directory
```

Calibration database entry:

```
BADPIX {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
```

```
BKGRDMAP {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
```

Output files:

```
{ODOMETER_CODE}_pp_badpixel.fits  \\ bad pixel map file (3100x4088)
{ODOMETER_CODE}_pp_bmap.fits      \\ background mask file (3100x4088)
DEBUG_{ODOMETER_CODE}_pp_background.fits \\ debug background file (7x3100x4088)
```

Plots:

```
BADPIX_MAP
```

Localisation Recipe

Finds the orders on the image.

Run:

```
cal_loc_spirou.py [DIRECTORY] [FLAT_DARK]
cal_loc_spirou.py [DIRECTORY] [DARK_FLAT]
```

Optional Arguments:

```
    --database, --badpixfile, --badcorr, --backsub, --combine,
    --darkfile, --darkcorr,  --flipimage, --fluxunits, --plot, --resize,
    --debug, --listing, --listingall, --version, --info,
    --program, --idebug, --breakpoints, --quiet, --help
```

Output Dir:

```
DRS_DATA_REDUC   \\ default: "reduced" directory
```

Calibration database entry:

```
ORDER_PROFILE_{FIBER} {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
LOC_{FIBER} {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
```

Output files:

```
{ODOMETER_CODE}_pp_order_profile_C.fits  \\ order profile file (3100x4088)
{ODOMETER_CODE}_pp_loco_C.fits           \\ localisation centers map file (49x4088
{ODOMETER_CODE}_pp_fwhm-order_C.fits     \\ localisation widths map file (49x4088)
{ODOMETER_CODE}_pp_with-order_C.fits     \\ localisation superposition file (3100x
DEBUG_{ODOMETER_CODE}_pp_background.fits \\ debug background file (7x3100x4088)
```

Plots:

```
LOC_MINMAX_CENTS, LOC_MIN_CENTS_THRES, LOC_FINDING_ORDERS, LOC_IM_SAT_THRES,
LOC_ORD_VS_RMS, LOC_CHECK_COEFFS
```

Shape Master Recipe

Creates a master FP image from all FPs processed. Uses this to work out the required shifts due to the FP master image, slicer pupil geometry and the bending of the orders (found in localisation).

Run:

```
cal_shape_master_spirou.py [DIRECTORY] -hcfiles [HCONE_HCONE] -fpfiles [FP_FP]
```

Optional Arguments:

```
    --database, --badpixfile, --badcorr, --backsub, --combine,
    --darkfile, --darkcorr,  --flipimage, --fluxunits, --locofile,
    --plot, --resize,
    --debug, --listing, --listingall, --version, --info,
    --program, --idebug, --breakpoints, --quiet, --help
```

Output Dir:

```
DRS_DATA_REDUC   \\ default: "reduced" directory
```

Calibration database entry:

```
SHAPEX {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
SHAPEY {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
FPMASTER {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
```

Output files:

```
{ODOMETER_CODE}_pp_shapex.fits            \\ dx shape map (3100x4088)
{ODOMETER_CODE}_pp_shapey.fits            \\ dy shape map (3100x4088)
{ODOMETER_CODE}_pp_fpmaster.fits          \\ fp master file (3100x4088) + FITS-TAB
DEBUG_{ODOMETER_CODE}_shape_out_bdx.fits  \\ dx map before dy map (3100x4088)
DEBUG_{ODOMETER_CODE}_shape_in_fp.fits    \\ input fp before shape corr (3100x4088
DEBUG_{ODOMETER_CODE}_shape_out_fp.fits   \\ input fp after shape corr (3100x4088)
DEBUG_{ODOMETER_CODE}_shape_in_hc.fits    \\ input hc before shape corr (3100x4088
DEBUG_{ODOMETER_CODE}_shape_out_hc.fits   \\ input hc after shape corr (3100x4088)
DEBUG_{ODOMETER_CODE}_pp_background.fits \\ debug background file (7x3100x4088)
```

Plots:

```
SHAPE_DX, SHAPE_ANGLE_OFFSET_ALL, SHAPE_ANGLE_OFFSET, SHAPE_LINEAR_TPARAMS
```

Shape (per night) Recipe

Takes the shape master outputs (shapex, shapey and fpmaster) and applies these transformations to shift the image to the master fp frame, unbend images and shift to correct for slicer pupil geometry.

Run:

```
cal_shape_spirou.py [DIRECTORY] [FP_FP]
```

Optional Arguments:

```
    --database, --badpixfile, --badcorr, --backsub, --combine,
    --darkfile, --darkcorr,  --flipimage, --fluxunits, --fpmaster,
    --plot, --resize, --shapex, --shapey,
    --debug, --listing, --listingall, --version, --info,
    --program, --idebug, --breakpoints, --quiet, --help
```

Output Dir:

```
DRS_DATA_REDUC   \\ default: "reduced" directory
```

Calibration database entry:

```
SHAPEL {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
```

Output files:

```
{ODOMETER_CODE}_pp_shapel.fits            \\ local shape map (3100x4088)
DEBUG_{ODOMETER_CODE}_shape_in_fp.fits    \\ input fp before shape corr (3100x4088
DEBUG_{ODOMETER_CODE}_shape_out_fp.fits   \\ input fp after shape corr (3100x4088)
DEBUG_{ODOMETER_CODE}_pp_background.fits \\ debug background file (7x3100x4088)
```

Plots:

```
SHAPE_DX, SHAPE_ANGLE_OFFSET_ALL, SHAPE_ANGLE_OFFSET, SHAPE_LINEAR_TPARAMS
```

Flat/Blaze Correction Recipe

Extracts out flat images in order to measure the blaze and produced blaze correction and flat correction images.

Run:

```
cal_flat_spirou.py [DIRECTORY] [FLAT_FLAT]
```

Optional Arguments:

```
    --database, --badpixfile, --badcorr, --backsub, --combine,
    --darkfile, --darkcorr,  --fiber, --flipimage, --fluxunits,
    --locofile, --orderpfile, --plot, --resize,
    --shapex, --shapey, --shapel,
    --debug, --listing, --listingall, --version, --info,
    --program, --idebug, --breakpoints, --quiet, --help
```

Output Dir:

```
DRS_DATA_REDUC   \\ default: "reduced" directory
```

Calibration database entry:

```
FLAT_{FIBER} {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
BLAZE_{FIBER} {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
```

Output files:

```
{ODOMETER_CODE}_pp_blaze_{FIBER}.fits          \\ blaze correction file (49x4088)
{ODOMETER_CODE}_pp_flat_{FIBER}.fits           \\ blaze correction file (49x4088)
DEBUG_{ODOMETER_CODE}_pp_e2dsll_{FIBER}.fits   \\ debug pre extract file (7x3100x4
DEBUG_{ODOMETER_CODE}_pp_background.fits       \\ debug background file (7x3100x40
```

Plots:

```
FLAT_ORDER_FIT_EDGES1, FLAT_ORDER_FIT_EDGES2, FLAT_BLAZE_ORDER1,
FLAT_BLAZE_ORDER2
```

Thermal Correction Recipe

Extracts dark frames in order to provide correction for the thermal background after extraction of science / calibration frames.

Run:

```
cal_thermal_spirou.py [DIRECTORY] [DARK_DARK_INT]
cal_thermal_spirou.py [DIRECTORY] [DARK_DARK_TEL]
```

Optional Arguments:

```
    --database, --badpixfile, --badcorr, --backsub, --combine,
    --darkfile, --darkcorr,  --fiber, --flipimage, --fluxunits,
    --locofile, --orderpfile, --plot, --resize,
    --shapex, --shapey, --shapel, --wavefile,
    --debug, --listing, --listingall, --version, --info,
    --program, --idebug, --breakpoints, --quiet, --help
```

Output Dir:

```
DRS_DATA_REDUC   \\ default: "reduced" directory
```

Calibration database entry:

```
THERMALT_{FIBER} {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
THERMALI_{FIBER} {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
```

Output files:

```
{ODOMETER_CODE}_pp_e2ds_{FIBER}.fits              \\ extracted + flat field file (
{ODOMETER_CODE}_pp_e2dsff_{FIBER}.fits            \\ extracted + flat field file (
{ODOMETER_CODE}_pp_s1d_w_{FIBER}.fits             \\ s1d constant in pixel space (
{ODOMETER_CODE}_pp_s1d_v_{FIBER}.fits             \\ s1d constant in velocity spac
DEBUG_{ODOMETER_CODE}_pp_e2dsll_{FIBER}.fits      \\ debug pre extract file (7x310
DEBUG_{ODOMETER_CODE}_pp_background.fits       \\ debug background file (7x3100x40
{ODOMETER_CODE}_pp_thermal_e2ds_int_{FIBER}.fits  \\ extracted thermal for dark_da
{ODOMETER_CODE}_pp_thermal_e2ds_tel_{FIBER}.fits  \\ extracted thermal for dark_da
```

Plots:

None

Wavelength solution Recipe

Creates a wavelength solution and measures drifts (via CCF) of the FP relative to the FP master

Run:

```
cal_wave_spirou.py [DIRECTORY] -hcfiles [HCONE_HCONE]
cal_wave_spirou.py [DIRECTORY] -hcfiles [HCONE_HCONE] -fpfiles [FP_FP]
```

Optional Arguments:

```
    --database, --badpixfile, --badcorr, --backsub, --blazefile,
    --combine, --darkfile, --darkcorr,  --fiber, --flipimage,
    --fluxunits,  --locofile, --orderpfile, --plot, --resize,
    --shapex, --shapey, --shapel, --wavefile, -hcmode, -fpmode
    --debug, --listing, --listingall, --version, --info,
    --program, --idebug, --breakpoints, --quiet, --help
```

Output Dir:

```
DRS_DATA_REDUC   \\ default: "reduced" directory
```

Calibration database entry:

```
WAVE_{FIBER} {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
```

Output files:

```
{ODOMETER_CODE}_pp_e2ds_{FIBER}.fits              \\ extracted + flat field file (
{ODOMETER_CODE}_pp_e2dsff_{FIBER}.fits            \\ extracted + flat field file (
{ODOMETER_CODE}_pp_s1d_w_{FIBER}.fits             \\ s1d constant in pixel space (
{ODOMETER_CODE}_pp_s1d_v_{FIBER}.fits             \\ s1d constant in velocity spac
DEBUG_{ODOMETER_CODE}_pp_e2dsll_{FIBER}.fits      \\ debug pre extract file (7x310
DEBUG_{ODOMETER_CODE}_pp_background.fits       \\ debug background file (7x3100x40
{ODOMETER_CODE}_pp_e2dsff_AB_linelist_AB.dat      \\ wave stats hc line list
{ODOMETER_CODE}_pp_e2dsff_AB_wave_hc_AB.fits      \\ wave hc only wave solution (4
{ODOMETER_CODE}_pp_e2dsff_AB_waveres_AB.fits      \\ wave res table (multi extensi
{ODOMETER_CODE}_pp_e2dsff_AB_wave_fp_AB.fit       \\ wave hc + fp wave solution (4
{ODOMETER_CODE}_pp_e2dsff_AB_hc_lines_AB.tbl      \\ hv lines list
```

Plots:

```
WAVE_HC_GUESS, WAVE_HC_BRIGHTEST_LINES, WAVE_HC_TFIT_GRID, WAVE_HC_RESMAP,
WAVE_LITTROW_CHECK1, WAVE_LITTROW_EXTRAP1, WAVE_LITTROW_CHECK2,
WAVE_LITTROW_EXTRAP2, WAVE_FP_FINAL_ORDER, WAVE_FP_LWID_OFFSET,
WAVE_FP_WAVE_RES, WAVE_FP_M_X_RES, WAVE_FP_IPT_CWID_1MHC,
WAVE_FP_IPT_CWID_LLHC, WAVE_FP_LL_DIFF, WAVE_FP_MULTI_ORDER,
WAVE_FP_SINGLE_ORDER, CCF_RV_FIT, CCF_RV_FIT_LOOP
```

Extraction Recipe

Extracts any preprocessed image using all the calibrations required.

Run:

```
cal_extract_spirou.py [DIRECTORY] [PP_FILE]
```

Optional Arguments:

```
      --badpixfile, --badcorr, --backsub, --blazefile,
      --combine, --objname, --dprtype, --darkfile, --darkcorr,
      --fiber, --flipimage, --fluxunits, --flatfile,
      --locofile, --orderpfile, --plot, --resize,
      --shapex, --shapey, --shapel, --thermal, --wavefile,
      --debug, --listing, --listingall, --version, --info,
      --program, --idebug, --breakpoints, --quiet, --help
```

Output Dir:

```
DRS_DATA_REDUC   \\ default: "reduced" directory
```

Output files:

```
{ODOMETER_CODE}_pp_e2ds_{FIBER}.fits              \\ extracted + flat field file (
{ODOMETER_CODE}_pp_e2dsff_{FIBER}.fits            \\ extracted + flat field file (
{ODOMETER_CODE}_pp_s1d_w_{FIBER}.fits             \\ s1d constant in pixel space (
{ODOMETER_CODE}_pp_s1d_v_{FIBER}.fits             \\ s1d constant in velocity spac
DEBUG_{ODOMETER_CODE}_pp_e2dsll_{FIBER}.fits      \\ debug pre extract file (7x310
DEBUG_{ODOMETER_CODE}_pp_background.fits       \\ debug background file (7x3100x40
```

Plots:

```
FLAT_ORDER_FIT_EDGES1, FLAT_ORDER_FIT_EDGES2, FLAT_BLAZE_ORDER1,
FLAT_BLAZE_ORDER2, THERMAL_BACKGROUND, EXTRACT_SPECTRAL_ORDER1,
EXTRACT_SPECTRAL_ORDER2, EXTRACT_S1D, EXTRACT_S1D_WEIGHT
```

Make Telluric Recipe

Takes a hot star and calculates telluric transmission

Run:

```
obj_mk_tellu_spirou.py [DIRECTORY] [E2DS & OBJ_DARK]
obj_mk_tellu_spirou.py [DIRECTORY] [E2DSFF & OBJ_DARK]
obj_mk_tellu_spirou.py [DIRECTORY] [E2DS & OBJ_FP]
obj_mk_tellu_spirou.py [DIRECTORY] [E2DSFF & OBJ_FP]
```

***Optional Arguments*:**

```
    --database, --blazefile, --plot, --wavefile
    --debug, --listing, --listingall, --version, --info,
    --program, --idebug, --breakpoints, --quiet, --help
```

***Output Dir*:**

```
DRS_DATA_REDUC   \\ default: "reduced" directory
```

***Telluric database entry*:**

```
TELLU_CONV_{FIBER} {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
TELLU_TRANS_{FIBER} {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
```

***Output files*:**

```
{ODOMETER_CODE}_pp_tellu_trans_{FIBER}.fits    \\ telluric transmission file (49x4
{WAVEFILE}_tellu_conv_{FIBER}.npy              \\ tapas convolved with wave file (
```

***Plots*:**

```
MKTELLU_WAVE_FLUX1, MKTELLU_WAVE_FLUX2
```

Fit Telluric Recipe

Using the telluric tramission calculates principle components (PCA) to correct input images of atmospheric absorption.

Run:

```
obj_fit_tellu_spirou.py [DIRECTORY] [E2DS & OBJ_DARK]
obj_fit_tellu_spirou.py [DIRECTORY] [E2DSFF & OBJ_DARK]
obj_fit_tellu_spirou.py [DIRECTORY] [E2DS & OBJ_FP]
obj_fit_tellu_spirou.py [DIRECTORY] [E2DSFF & OBJ_FP]
```

Optional Arguments:

```
    --database, --blazefile, --plot, --wavefile
    --debug, --listing, --listingall, --version, --info,
    --program, --idebug, --breakpoints, --quiet, --help
```

Output Dir:

```
DRS_DATA_REDUC   \\ default: "reduced" directory
```

Telluric database entry:

```
TELLU_CONV_{FIBER} {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
TELLU_TRANS_{FIBER} {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
```

Output files:

```
{ODOMETER_CODE}_pp_e2dsff_tcorr_{FIBER}.fits    \\ telluric corrected e2dsff file
{ODOMETER_CODE}_pp_s1d_w_tcorr_{FIBER}.fits     \\ telluric corrected s1d constant
{ODOMETER_CODE}_pp_s1d_v_tcorr_{FIBER}.fits     \\ telluric corrected s1d constant
{ODOMETER_CODE}_pp_e2dsff_recon_{FIBER}.fits    \\ reconstructed transmission e2ds
{ODOMETER_CODE}_pp_s1d_w_recon_{FIBER}.fits     \\ reconstructed transmission s1d
{ODOMETER_CODE}_pp_s1d_v_recon_{FIBER}.fits     \\ reconstructed transmission s1d
```

Plots:

```
EXTRACT_S1D, EXTRACT_S1D_WEIGHT, FTELLU_PCA_COMP1, FTELLU_PCA_COMP2,
FTELLU_RECON_SPLINE1, FTELLU_RECON_SPLINE2, FTELLU_WAVE_SHIFT1,
FTELLU_WAVE_SHIFT2, FTELLU_RECON_ABSO1, FTELLU_RECON_ABSO2
```

Make Template Recipe

Uses all telluric corrected images of a certain object name to create and BERV and wavelength corrected template in order to server as a better model SED for telluric correction.

`obj_mk_tellu_spirou.py` and `obj_fit_tellu_spirou.py` need to be rerun after template generation.

Run:

```
obj_mk_template_spirou.py [OBJNAME]
```

Optional Arguments:

```
    --filetype, -fiber,
    --database, --blazefile, --plot, --wavefile
    --debug, --listing, --listingall, --version, --info,
    --program, --idebug, --breakpoints, --quiet, --help
```

Output Dir:

```
DRS_DATA_REDUC   \\ default: "reduced" directory
```

Telluric database entry:

```
TELLU_CONV_{FIBER} {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
TELLU_TRANS_{FIBER} {NIGHT_NAME} {FILENAME} {HUMAN DATE} {UNIX DATE}
```

Output files:

```
Template_{OBJNAME}_{filetype}_{FIBER}.fits  \\ Template for object (3100x4088) + F
```

```
Template_s1d_{OBJNAME}_sc1d_w_{FIBER}.fits  \\ Template s1d constant in pixel spac
Template_s1d_{OBJNAME}_sc1d_v_{FIBER}.fits  \\ Template s1d constant in velocity s
BigCube0_{OBJNAME}_{filetype}_{FIBER}.fits  \\ Cube of obs making template earth r
BigCube_{OBJNAME}_{filetype}_{FIBER}.fits   \\ Cube of obs making template star re
```

Plots:

```
EXTRACT_S1D
```

CCF Recipe

Cross correlates the input image against a mask and measures a radial velocity per order, and combines to give an over all radial velocity measurement. Also (where possible) takes into account the FP drift measured by a CCF in the wave solution (when wave solution used a FP)

Run:

```
cal_ccf_spirou.py [DIRECTORY] [E2DS & OBJ_FP]
cal_ccf_spirou.py [DIRECTORY] [E2DSFF & OBJ_FP]
cal_ccf_spirou.py [DIRECTORY] [E2DS_CORR & OBJ_FP]
cal_ccf_spirou.py [DIRECTORY] [E2DSFF_CORR & OBJ_FP]
cal_ccf_spirou.py [DIRECTORY] [E2DS & OBJ_DARK]
cal_ccf_spirou.py [DIRECTORY] [E2DSFF & OBJ_DARK]
cal_ccf_spirou.py [DIRECTORY] [E2DS_CORR & OBJ_DARK]
cal_ccf_spirou.py [DIRECTORY] [E2DSFF_CORR & OBJ_DARK]
```

Optional Arguments:

```
    --mask, --rv, --width, --step
    --database, --blazefile, --plot
    --debug, --listing, --listingall, --version, --info,
    --program, --idebug, --breakpoints, --quiet, --help
```

Output Dir:

```
DRS_DATA_REDUC   \\ default: "reduced" directory
```

Output files:

```
{ODOMETER_CODE}_pp_{INTYPE}_{FIBER}_ccf_{mask}_{FIBER}.fits  \\ CCF for science ch
{ODOMETER_CODE}_pp_{INTYPE}_{FIBER}_ccf_fp_{FIBER}.fits      \\ CCF for reference
```

Plots:

```
CCF_RV_FIT, CCF_RV_FIT_LOOP
```

Polarimetry Recipe

Produces all polarimetry outputs.

Run:

```
pol_spirou.py [DIRECTORY] [E2DSFF]
pol_spirou.py [DIRECTORY] [E2DSFF_CORR]
```

Optional Arguments:

```
    --blazefile, --plot, --wavefile,
    --debug, --listing, --listingall, --version, --info,
    --program, --idebug, --breakpoints, --quiet, --help
```

Output Dir:

```
DRS_DATA_REDUC   \\ default: "reduced" directory
```

Output files:

```
{ODOMETER_CODE}_pp_{INTYPE}_pol.fits                // polar file
{ODOMETER_CODE}_pp_{INTYPE}_StokesI.fits            // stokes file
{ODOMETER_CODE}_pp_{INTYPE}_null1_pol.fits          // null 1 file
{ODOMETER_CODE}_pp_{INTYPE}_null2_pol.fits          // null 2 file
{ODOMETER_CODE}_pp_{INTYPE}_lsd_pol.fits            // lsd file
{ODOMETER_CODE}_pp_{INTYPE}_s1d_w_pol.fits          // s1d polar file constant in p
{ODOMETER_CODE}_pp_{INTYPE}_s1d_v_pol.fits          // s1d polar file constant in v
{ODOMETER_CODE}_pp_{INTYPE}_s1d_w_null1.fits        // s1d null 1 file constant in
{ODOMETER_CODE}_pp_{INTYPE}_s1d_v_null1.fits        // s1d null 1 file constant in
{ODOMETER_CODE}_pp_{INTYPE}_s1d_w_null2.fits        // s1d null 2 file constant in
{ODOMETER_CODE}_pp_{INTYPE}_s1d_v_null2.fits        // s1d null 2 file constant in
{ODOMETER_CODE}_pp_{INTYPE}_s1d_w_stokesi.fits      // s1d stokes file constant in
{ODOMETER_CODE}_pp_{INTYPE}_s1d_v_stokesi.fits      // s1d stokes file constant in
```

Plots:

```
POLAR_CONTINUUM, POLAR_RESULTS, POLAR_STOKES_I, POLAR_LSD,
EXTRACT_S1D, EXTRACT_S1D_WEIGHT
```

Back to top