Python software for turbulence measurements with RINGSS.

Authors:*A. Tokovinin*
Version: 2.1
Date: 2023-03-13
File: prj/smass/doc/ringss-python2.tex

1 Overview

This document contains a short description of the python software package to measure optical turbulence profiles with the RINGSS instrument. The concept of RINGSS and the processing algorithms are covered in separate documents. Originally, it was ported from the IDL code and used to process individual image cubes (movies) saved in FITS files. In this second version, the image cubes are acquired and processed online by the robotic operation software, the intermediate results (statistical moments and other parameters) are stored in the .stm text file, and its online processing creates the final data product (the .prof text file).

The present document covers only the data processing and turbulence profile restoration. Image acquisition and robotic operation, performed by other modules, is not covered. While the data-processing software is generic (applicable to different systems), the robotic operation is strongly related to specific hardware. As an exception, the package contains the image acquisition software for one particular camera, ZWO ASI290MM. It can be used for manual operation with this detector.

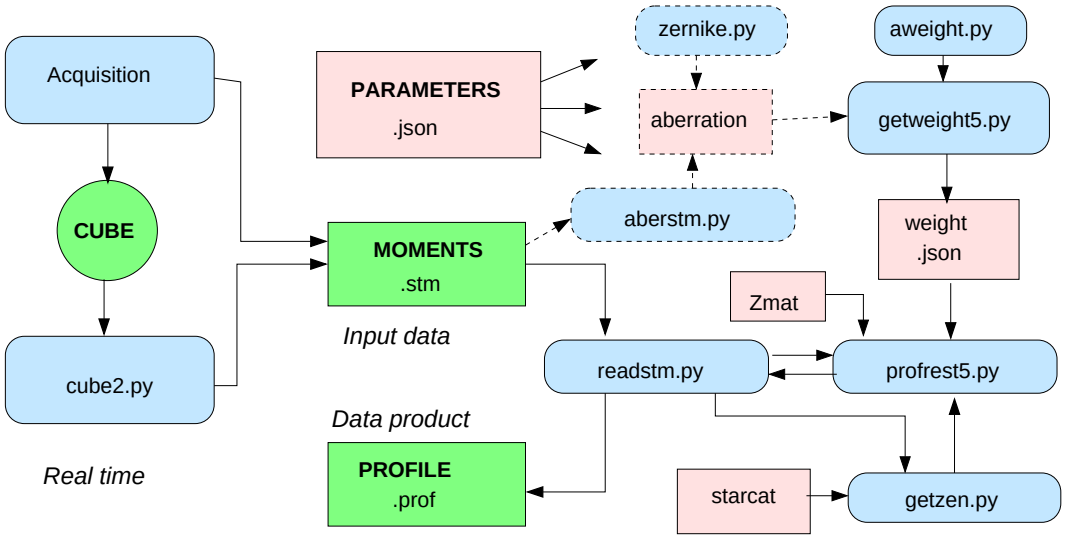

Figure 1: Scheme of the data processing. Pink boxes are json files containing dictionaries, blue boxes are python modules, and green boxes are text files. The arrows show the information flow.

The python code is ported from the version 5 of the IDL processing code, described in a separate document. The code is developed for python 3.8. The following python packages are used: numpy,

scipy (optimize), astropy (io, time, units, coordinates), json, codecs, sys. The vendor's drivers are used for image acquisition with ASI290MM. Unlike the IDL code, no simulation tools are provided (they are still not ported to python).

Figure 1 illustrates the data flow. The acquisition software takes sequences of consecutive short-exposure (1 ms) images, the image cubes (typically 2000 frames of 64x64 pixels each). The cubes are processed immediately after the acquisition, and the result is appended as a new line in the .stm text file. This file, together with the system parameters, constitutes the input data for processing. The processing result is a text file with turbulence profiles (.prof). The system parameters and other intermediate data are dictionaries stored in the json (.json) format, depicted by pink boxes in the figure. The code is organized so that each module is given only the required data and ignores other data, thus separating tasks and eliminaing mutual dependencies as much as possible. The common instrument parameters are used by almost all modules.

2 Parameters and other dictionaries

The parameter file is an ASCII JSON file that defines the parameter dictionary describing a particular instrument. It contains three sections: telescope, site, and profile. For robotic operation, other sections are edded describing the camera parameters for full-frame images and cubes. An example of the parameter file is given below. Unfortunately, the JSON standard does not allow comments in the files.

```
{"telescope":{"D":0.127,"eps":0.5,"pixel":1.78,"pdist":460,ringradpix:11.0,"ron":1.1},
 "site":{"name":"LaSerena","lon":-71.2425,"lat":-29.91744},
 "profrest":
 {"mmax":20,"zgrid":[0,250,500,1000,2000,4000,8000,16000],
  "wav":[500,525,550,575,600,625,650,675,700,725,750],
  "sp":[0.65,0.774,0.755,0.73,0.704,0.673,0.617,0.568,0.495,0.427,0.3],
  "weightfile":"weights.json",
  "aber":"aber-tololo.json",
  "zmat":"zmat.json",
  "starcat":"starcat.json"}}
```

The *telescope* section specifies the aperture diameter (m), the central obscuration, the pixel size (arcsec), the virtual propagation distance H_0 (m), the ring radius (pixels), and the detector readout noise (electrons). The H_0 parameter depends on the ring-image radius (hence on the telescope focus) and determines the relation between statistical moments and turbulence – the weighting functions (WFs).

The *site* section specifies the geographical coordinates needed for calculation of the zenith distance.

The *profrest* section contains information required for data processing. The $m_{\max} = 20$ is the maximum frequency of the angular coefficients in the moments file. The `zgrid` defines the number and heights of layers in the reconstructed turbulence profile. The spectral response of the system (product of detector efficiency and filter transmission) is defined by a list of wavelengths (in nm, preferably on a regular grid) and the response factors (with arbitrary normalization); it is equivalent to the `qedata.txt` response file used in the IDL version. The last parameters in this section are the

names of the auxiliary files (in json format) used in the data processing, namely the WF functions, the aberration coefficients (optional), the so-called Z matrix (a fixed file), and the star catalog.

3 Content of the input (.stm) and output (.prof) files

The .stm (for 'statistical moments') is a text file. The acquisition software that creates this file is not covered in this document. Each line begins by a 1-character prefix that defines its content. This convention is inherited from the MASS-DIMM software. However, we adopt the comma-separated (CSV) format, instead of space-separated stm files in MASS-DIMM.

One turbulence measurement of RINGSS consists in taking and processing 10 data cubes, of 2000 frames each. The online processing of individual cubes is recorded in lines with prefix 'm'. The beginning of a long m-line is shown below:

```
m, 2022-05-12-01:36:37,5056,7.4100E+00,5.3726E+04,1.5589E-01,1.1028E+01...
```

The first fields are tag (m), date-time, and the HR number of the star. The following 12 numbers are the image parameters (impar): background, flux (in ADU), rms flux fluctuations, ring mean radius, ring width (pixels), ring X- and Y-centers (relative to the frame center, in pixels), rms of the X- and Y-centers, coma amplitude and angle, and contrast (not used).

The following four elements are the noise coefficients used to compute the noise variances of the signals (for de-biasing the turbulence variances). Then follow the 21 variances of the angular coefficients for angular frequencies from $m = 0$ to $m = 20$ and the 21 covariances with a time lag of 1 frame. The following two numbers are the estimates of noise of the sector radii and the mean rms of the differential sector radii fluctuations used to compute the alternative seeing estimate `see2`. The last 14 numbers are the mean values of the 8 sector radii and the 3 first mean cosine and sine angular coefficients, needed for calculation of the aberrations.

After acquisition of 10 data cubes, the statistical moments are averaged internally and recorded in the .stm file in a line with prefix 'M'. Its format is identical to the format of the m-line, but it contains an additional number 10 in the 4th position (the number of averaged cubes).

Each cycle of the measurement begins by a record of an O-line like:

```
O, 2022-05-12-01:36:32,5056,2000,1,200,10,64,64
```

The O-line contains the prefix, the date-time label, the star HR number, number of frames par cube, exposure time (1 ms), camera gain (200), the number of averaged cubes (10), and the numbers of pixels in each frame (64 and 64).

The processing results are stored in the profiles (.prof) file. Each line of this file matches the M-line of the .stm file. The line of a .prof file looks like

```
2022-05-12T01:37:23,5056,25.47,5.3480E+04,  0.579,0.483,0.348,17.33,4.682,2.666,
0.036,0.084,0.77,0.00,0.00,0.06,0.52,0.02,0.58,0.09
```

The line contains the date-time label (with a letter T in the middle to match the isot standard), HR number of the star, zenith distance, and the flux (in ADU). Then, after a space, follow the global parameters: see2, seeing, free-atmosphere seeing (in arcseconds at 500 nm at zenith), effective wind speed (m/s), atmospheric time constant (ms), and isoplanatic angle (arcsec). After another space, the

line contains the total scintillation, rms error of the profile fit, and the 8 numbers of the turbulence profile ($C_n^2 dh$ integrals in units of 10^{-13} $m^{1/3}$ at heights of 0, 0.25, 0.5, 1, ... 16 km above ground). The number of reconstructed layers and their heights can be changed by modifying the parameters file and re-running the profile reconstruction.

4 Program modules

4.1 cube2.py

The main processing procedure used by the data acquisition module is `Moments(data_cube)` which works directly on the image cube (a 3D array numpy). It returns the dictionary `data` with sections `impar` (flux, background, ring radius, ring width, its position and its variance, measure of the coma), `noisepar` (a list of 4 numbers used for the noise calculation), and `moments` (statistical moments of the signals extracted from each ring image in the cube). This information is saved in an m-line of the .stm file together with the acquisition parameters (time, star HR number, etc., see § 3).

4.2 readstm.py

The new code `readstm.py` replaces the old python cube-processing (now done online). It takes the information from the O- and M-lines of the .stm file. The results are formatted as a `data` dictionary and passed to the profile-restoration tool `profrest5.py`. Parameters of the star needed in the process (zenith distance and color) are retrieved from the catalog and computed by calls to `getstarpar` and `getzen` in the module `getzen.py`. The results are written to the text file with the name of the original .stm file and the extension `.prof`, one line per each M-line of the input file.

The weighting functions (WF) dictionary contains the nominal ring radius in pixels that corresponds to the telescope parameters and the selected propagation distance H_0. If the actual ring radius r differs from its nominal value by more than 0.05 (in the relative sense), the M-line is not accepted for profile restoration and, instead, a warning line is written into the .prof file. It begins with # and contains the date-time label. The 0.05 threshold is hard-coded in `readstm.py`. Note that for a given instrument the product rH_0 is approximately constant, so it is easy to determine the correct H_0 from the measured ring radius r.

To process one .stm file, one needs to compute the WFs (see below) and then issue the command

```
> python readstm.py <stm-file> <par-file>
```

4.3 getzen.py

To interpret the RINGSS results, we need to know the zenith distance at the moment of observation and the star $B - V$ color. Stars are identified by their names (HR numbers). The module `getstar` in `getzen.py` searches the star catalog. The catalog is derived from the Bright Stars and saved in `starcat.json` by `readstarcat.py`; the name of this file is given by the parameter dictionary, which also defines the site coordinates. Once the star is found in the catalog, the module `getstarpar` computes the Julian date, zenith angle and azimuth and, together with the star HR number, V magnitude, and $B - V$, returns this information in the `starpar` dictionary.

4.4 getweight5.py

This module computes the weighting functions (WFs) needed for the profile restoration. Its input, the parameter dictionary (see § 2), contains all necessary information on the instrument (telescope diameter, conjugation height, pixel scale) but knows nothing about the star. The weights are computed on a logarithmic grid of 16 distances z from 250 m to 32 km with a step of $\sqrt{2}$ and an addition of the zero distance. The weights are computed and saved for stars with $B-V$ colors of 0 and 1 (their spectra are approximated by black bodies of temperatures 10213 and 3938 K, respectively), and the weights for other colors are obtained by linear interpolation. The spectral response of the instrument is defined in the parameter dictionary by the lists of wavelengths and response coefficients. The distance grid z, weight for $B-V=0$, and its color coefficient (slope) are saved in the weights dictionary for angular frequencies from 1 to $m_{\max}$ ($m_{\max}$ is defined in the parameter file and matches the number of moments returned by `cube2.py`). The weight array is a matrix of [17,mmax+1] size. In place of $m=0$, the weight for the sector motion is stored. The weight-calculation engine `aweight.py` is called by this module to do the job. Typically, it takes ∼30 s on a laptop to compute the weights. The WFs depend on the propagation distance, which in turn is related to the ring radius. The module uses only the ring radius and computes the corresponding propagation distance using the instrument parameters.

Apart from the weights, this module also computes six so-called U-coefficients needed for measurement of the wind speed. These coefficients, like the weights, are computed for $B-V=(0,1)$ and linearly approximated for other colors. The weight dictionary also contains the effective wavelengths for $B-V$ of 0 and 1 and the nominal ring radius in pixels.

The module `aweight.py` used for the weight calculation is ported from the IDL code `aweight3.pro` and gives the same results, within the calculus errors. Inputs: z – vector of distances [m], mmax – maximum angular frequency, d – telescope diameter [m], eps – central obscuration ratio, pdist – conjugation distance below ground [m], wav – array of wavelengths [m], sp – corresponding array of spectral response, drho – width of the ring mask in FWHM units (default is 1.5), pixel – angular pixel size [arcsec]. Optionally, a list of Zernike numbers [zn] and their amplitudes [zrad] (in radians at the nominal wavelength given in the parameters) can be given to account for the aberrations. Outputs: the matrix of weights of the size $N_m \times N_z$, in $\mathrm{m}^{-1/3}$, expressing the scintillation power for $m=1...m_{\max}$ and the sector-motion weight for $m=0$ for a turbulence integral of $J=1\,\mathrm{m}^{1/3}$. Also it returns the matrix of U-functions used for the wind estimation, same size as WF, in $\mathrm{m}^{7/3}$.

4.5 aberstm.py and aberration.py

The aberration handling is implemented in `aberration.py` that computes the relation between 7 Zernike terms (from focus to trefoil, numbers 4 to 11) and the mean coefficients, and saves relevant information in the aberration file with a name specified by the parameter file, in json format. It uses the modules `getimage.py` (calculate a ring image) and `zernike.py` (Zernike toolbox).

The .stm files contain 14 average coefficients (8 sector radii, 3 first cosine and 3 sine angular terms) to evaluate optical aberrations of the instrument. The relation between this 14-element vector A and the 7-element vector of Zernike amplitudes Z_{rad} is

$$Z_{\mathrm{rad}} = R_a\,(S \cdot A), \tag{1}$$

where S is the 7x14 selector matrix which forms linear combinations of average coefficients (it is system-independent) and R_a is a 7-element vector of aberration response, converting these combinations into Zernike amplitudes for a given telescope. The procedure `aberresp` takes the parameters as input and returns the S and R_a in the `aberresp` dictionary. For a given instrument, it must be run from the command line once. The procedure `getzampl` uses this dictionary and the actual mean coefficients and returns the $Z_{\rm rad}$ vector.

The `aberstm.py` reads the .stm file, computes the Zernike amplitudes from the mean coefficients using `getzampl`, and adds them to the aberration dictionary, saved in the same file. If the aberration dictionary file is not present in the parameter file, the WFs for an ideal system are computed by `getweight5.py`, otherwise the aberrations are accounted for in the WFs. So, for a given system, the aberration analysis must be run, and the WFs must be recalculated if the aberration amplitudes are substantial (more than ∼1 radian).

To prepare the data processing, the following commands are typically used:

```
> python aberration.py <par-file>
> python readstm.py <par-file> <stm-file>
> python getweight5 <par-file>
```

The first two commands define the aberrations (they can be omitted if the optics is perfect), the last line computes the WFs. There is no need to recompute the WFs for each night if the system parameters do not change.

4.6 profrest5.py

The turbulence profile and the associated atmospheric parameters are estimated by `profrest5.py` from the statistical moments and the WFs. This code is ported from IDL `profrest5.pro`. Its main module `Restore` takes the dictionaries of parameters, data, weight, and zmat as inputs. The output is the `profile` dictionary. The calling routine `readstm` creates the data dictionary from the M-lines of the .stm file, calls `Restore`, and writes the resulting profiles to the .prof file.

The turbulence profile is represented by 8 thin layers located at fixed heights z_0 of (0, 0.25, 0.5, 1,... 16) km above the site. Turbulence integrals in these layers J (in 10^{-13} m$^{1/3}$ units) are the measurement result; they express the turbulence strength at zenith (like in MASS). The z_0 grid is defined in the parameter file and can be easily modified. The fixed-height layers are actually seen at distances of $z_0/\cos(\gamma)$ when observed at the zenith angle γ. The pre-computed weights are interpolated linearly on this internal 'stretched' grid and for the actual star color (parameters γ and $B-V$ are taken from `starpar`).

The variances and covariances at angular frequencies from 0 to $m_{\rm max}$ are retrieved from the `moments` section of the data dictionary. The estimated noise bias is computed using `noisepar` and the flux (converted from ADU to electrons according to the camera gain, hard-coded at present). The noise-subtracted variance is corrected for the finite-exposure time and for the effect of saturation. The latter correction uses the so-called Z-matrix (fixed), which is available as JSON file (referenced in the parameters). The corrected variances S' at frequencies $m = 1, ...15$ (note the reduced maximum frequency) and the truncated matrix of weights A are inputs to the restoration. The turbulence profile vector x must minimize the quadratic difference between measurements and their model, i.e.

$$\sum_m (Ax - S')^2_m \rightarrow \min \tag{2}$$

subject to the non-negativity condition $x \geq 0$. The solution is found by the non-negative least squares (nnls) method. However, the signals $S'(m)$ vary by orders of magnitude, and the direct solution of (2) will try to accommodate the strongest (small m) terms at the detriment of the weaker large-m signals. To remedy this, the lines of A and the values of S' are weighted (multiplied) by $1/S$ (uncorrected variance, always non-negative). This modification effectively minimizes the sum of relative errors $\sum_m (Ax/S' - 1)^2_m$. The same approach is implemented in the IDL code.

When the turbulence profile (vector x) is found, the corresponding full seeing and the free-atmosphere seeing (0.5 km and above) are computed. The profile-weighted response to the differential sector motion is then used to compute the alternative estimate of the total seeing, `see2`. Finally, the effective wind speed and the atmospheric time constant are calculated. The results are returned in the `profile` dictionary.

5 Additional modules

5.1 corcent.py

This code correlated the full-frame image with a synthetic ring image to find the ring center. It is used during star acquisition and initial centering. Input parameters: image (2D array), ring radius (pixels) and width. The code returns ring center in X and Y (in pixels relative to the frame center) and the dimensionless amplitude of the correlation maximum. The amplitude is typically well above 10 for images containing the ring and around one for empty images.

5.2 ASI290mm.py

This module is used for image acquisition from the ZWO ASI290MM CMOS camera during robotic operation. However, it can be used independently, in the manual mode. In such case, it opens its own GUI. The camera drivers provided by the vendor are essential, while `astropy` is used for the FITS input/output of images. The GUI uses the `wx` widget toolbox, and the `cv2` package is used for the image display.

In the manual GUI, the camera can work in 3 hard-coded formats of square 1024, 256, and 64 pixels with a visual refresh rate of ∼10 Hz. The exposure time in each mode is also hard-coded, but the gain and star number are adjustable. To switch betwen the modes, one must first stop the video, select another mode, and run it again. The button CUBE takes one data cube and, if the checkbox Process-cube is selected, it computes the coefficients and coma parameters. During cube acquisition, the video is paused. The cubes are saved in the subdirectory `data`.

In the robotic operation, the camera server uses procedures `snapshot` and `cube` contained in this module.