PUAKO note #01:
How to install and use PUAKO

O. Beltramo-Martin*

January 6, 2021

Contents

---

1 Rationale

PUAKO stands for *Psf Utilities for Adaptive optics systems at Keck Observatory* and aims to offer to the Keck community a PSF reconstruction facility class. PUAKO results from strong and continuous efforts that have been led by W.M. Keck Observatory on PSF reconstruction (PSF-R) in close collaboration with several institutes and partners. Two operational modes are possible with PUAKO:

- **PSF reconstruction:** this modes allows to handle the *.sav* telemetry data coming from the Adaptive Optics (AO) system so as to reconstruct the PSF and save a *.fits* file. This latter contains the reconstructed PSF as well as some additional information in the file header, such as retrieved integrated atmospheric parameters (seeing, outer scale), system status (loop frequency, gain, ...), reconstructed PSF metrics (Strehl-ratio, FWHM,..) and wave-front error breakdown. Hopefully, such a PSF will be handled by Keck users to improve their image post-processing. The processing done on the telemetry and the PSF-R framework are detailed in the PUAKO notes #02 and #03 respectively.

*olivier.beltramo-martin@lam.fr

- **PRIME:** this mode, which refers to the technique introduced in [1], is a best-fitting facility that goes upon the PSF-R layer and combines the PSF-R framework and best-fitting strategies to enhance the reconstruction process. This only suits observed fields that contain point-like sources to perform the model-fitting step. For such situations, PRIME permits to (i) enhance the reconstruction step by adjusting few (~ less than 10) additional degrees of freedom, which have hopefully a physical meaning (seeing, optical gains, low-order static Zernike modes for instance) (ii) track these degrees of freedom variability regarding observing conditions to improve the PSF-R and possibly the real-time operations. This operating mode is more likely to be deployed by Keck AO scientists. An automatized pipeline could be envisioned but would necessitate an image processing and PSF detection facility, which is currently only available for engineering NIRC2 data. More information can be found on the PUAKO note #04.

2 How to install

PUAKO is versioned using a free license of gitHub but with private access rights. Consequently, you need to sign up to gitHub by clicking here and the repository can be found there. Note it's a private repository, so you need to request access to clone the code. So far, PUAKO complies with Unix platforms only, so to install PUAKO, you must open a terminal and choose the folder in which you want to have PUAKO code using:

```
[omartin@lam PUAKO]$ git clone https://puako:alphaCenBb2019@github.com/puako/PUAKO.git
```

where *alphaCenBb2019* is the password needed to clone the PUAKO repository on your local machine. Note that you can modify the code and commit your changes with your own gitHub account, this password is only needed for downloading purpose. When cloning the PUAKO repository, you should the following message:

```
Cloning into 'PUAKO'...
remote: Enumerating objects: 464, done.
remote: Counting objects: 100% (464/464), done.
remote: Compressing objects: 100% (263/263), done.
remote: Total 464 (delta 228), reused 411 (delta 175), pack-reused 0
Receiving objects: 100% (464/464), 6.58 MiB | 1.73 MiB/s, done.
Resolving deltas: 100% (228/228), done.
```

Then, you want to make sure on which developing branch you are set up by doing

```
[omartin@lam PUAKO]$ git status
On branch master
Your branch is up to date with 'origin/master'.
nothing to commit, working tree clean
```

which says that you are currently synchronized with the master branch, that is the one to be released. If you want to help developing the code, you must do it using another branch until modifications are validated by the PUAKO developers group (S. Ragland and I so far). To do so, you can switch to the *origin/dev* branch by doing

```
[omartin@lam PUAKO]$ git checkout dev
Branch 'dev' set up to track remote branch 'dev' from 'origin'.
Switched to a new branch 'dev'
```

or create your own branch with the command `git branch "new branch"`. Before any new development, you must ensure that your current version is up-to-date using the `git pull` command that is going to download

Table 1: Summary of git cheats.

Command	**Description**
git clone path	Clone given repository path from git server
git checkout master	Checkout your chosen branch (or change branch), in this case the master branch.
git branch branchname	Create a new branch of the current git repository named ”branchname”. If the branch already exists, this will change your current branch to ”branchname”.
git pull	Get any changes to your current branch (pull changes from the git server).
git add filename	Add files to the git. The add command is only required when the file is new to the repository, not for changes to the file.
git commit –m ”commit message”	Commit any changes to the git repository. Commit must be used for any changes to files and after add to upload new files to the server.
git status	Check the status of your working copy against the current git repository. Will list any changes staged/unstaged for upload to the repository.
git push	Uploads you changes staged by commit to the git repository.
git fetch –all git reset –hard origin/branchname	If you want to return your working copy to the current status of the repository. WARNING: this will overwrite all your local changes.
git rm filename	Remove file from repository. Will only be removed from the git repo after commit and push.

all changes from the git server. Then, you can modify the git repository (not the files on git server) by doing `git commit -m "commit message"` followed by `git push` to update the git server files. In Tab. 1, I report several useful git commands to keep in mind when using gitHub.

3 How to use

3.1 Instantiate PUAKO

PUAKO is a Matlab code that runs on newer version than Matlab R2014 on UNIX platforms. In the directory where you have cloned the PUAKO source, open a matlab instance

```
[omartin@lam PUAKO]$ matlab &
```

and in the command window, you can type `version` to verify the Matlab version, which is 9.4.0.813654 (R2018a) on my laptop. Then, you need to indicate to Matlab in which workspace you are working in. There are two ways to do so: either you use the `pathtool` command to select the folder to be imported, or you specify it manually using the command window from

```
path_workspace = '/home/psfr/PUAKO/PUAKO/'; addpath(genpath(path_workspace));
```

Starting from now, you must be able to access the different functions and classes that PUAKO inherits from and instantiate PUAKO as a top-level Matlab class

```
p = puako('path_imag',path_imag,'path_trs',path_trs,'path_calibration',path_calib);
```

where p is the PUAKO object that contains properties and methods. You can initialize it by providing data folders paths using the command window like

```
date = '20130801';
path_root = '/net/vm-psfr/psfrData/NIRC2/';
path_calib = '/net/vm-psfr/psfrData/CALIBRATION/';
path_imag = [path_root,date,'/IMAG/'];
path_trs = [path_root,date,'/TRS/'];
```

PUAKO will automatically verify (using functions readCalibrationFolder.m, readIMAGFolder.m and readTRS-Folder.m in PUAKO/_01_readFolder/_folderChecking) that (i) these folders exist, otherwise it will display an error dialog window shouting at you that you must provide an existing folder path (ii) verify these folders contains files. If there are no .fits file in path_imag, PUAKO will send a warning as it is not mandatory to have an image to diagnose the AO system. On the contrary, it will open a error dialog window if there is no .sav files in path_trs.
When calling the PUAKO object, you should see a message appearing the command window (Hello there, I'm PUAKO and I shall be your guide in this cruel AO world !) and if you have provided data folders paths, PUAKO will request your permission to:

- Create a MASSDIMM folder inside the path_calib folder to store the .dat files from the MASSDIMM at MaunaKea, and create Dark, Sky and Div folders inside the path_imag folder.
- Sort the calibration images acquired for dark, sky and diversity measurements (function sortDarkand-SkyandDivFiles.m in PUAKO/_01_readFolder/_folderChecking) using the keywords in the .fits file header. When the field `shutter` is 'CLOSE', the camera image corresponds undoubtedly to a dark. For detecting a sky calibration, it is more tricky from the header fields. So far, we verify that the quadratic sum of the fields values 'AOFMX' and 'AOFMY' is greater than a threshold, named 'threshSteeringMirror' that is initialized to 20" but can be tuned by the user when instantiating PUAKO. In other words, we track the offset of the steering mirror. About the diversity measurements, we look at the difference between the WFS stage z-position and the position it must be according to a telescope elevation-based model. The current WFS stage position is given by the field 'OBWF' while the model is obtained from 'AOFC-NGFO' and 'AOFCLGFO' for NGS and LGS operations respectively. If the difference is larger than 3 mm, PUAKO will consider that we have taken in/out of focus images that it will place into the Sky folder.

If you want to give full permission to PUAKO, you can instantiating it by setting the optional input 'yesToAll' to 'true'. Also, PUAKO will check the presence of the sub-folders 'TELESCOPE' and 'AOSYSTEM' inside the calibration folder and that are supposed to contain calibration measurements of the telescope and AO that are mandatory to perform PSF-R. If they are not there, PUAKO will warn you that you won't be able to use the PSF-R facility. Then, PUAKO will read data folders and instantiates its data_folder_path and data_folder_id properties (functions getTRSDataID.m, getIMAGDataID.m in PUAKO/_01_readFolder/_dataChecking)

```
>> p.data_folder_id

ans =
```

```
  struct with fields:

    imag: [1x59 struct]
    dark: [1x18 struct]
     sky: []
     div: [1x27 struct]
     trs: [1x107 struct]
>> p.data_folder_id.trs(1)

ans =

  struct with fields:

      name: 'n0001'
      type: 'sav'
    aomode: 'NGS'
      size: 17.8980
       hdr: []
      path: '/run/media/omartin/HDD_OBM/KECK_DATA/20130801/TRS/n0001_fullNGS_trs.sav'
>> p.data_folder_id.imag(1)

ans =

  struct with fields:

          name: 'n0004'
   origin_name: 'n0004'
          type: 'fits'
          size: 4.2163
           hdr: {196x3 cell}
          path: '/run/media/omartin/HDD_OBM/KECK_DATA/20130801/IMAG/n0004.fits'
```

where each field is a structure containing several useful information on the .fits (imag) and .sav (trs) files, as the object name, type, size in Mbytes and path. The .fits files id structures will also contain the file header in the field hdr. Note that we may have a different number of .fits and .sav files. This structure is really useful to get an overview of all the data you have accessed to in the folder you have instantiated PUAKO with.
Furthermore, you can change the data folder paths at any time by changing them manually in the command window

```
>> p.path_trs = '/run/media/omartin/HDD_OBM/KECK_DATA/20170314/TRS/'
Warning: 111 TRS files are not associated with a fits header !
> In getTRSDataID (line 85)
  In puako/set.path_trs (line 109)
```

PUAKO will automatically get through the verification steps described above. If you change the path_trs but not the path_imag, PUAKO will warn you that .sav files are not associated to any .fits file.

3.2 Process telemetry data

On of the PUAKO's properties is the *telemetry* class that is another Matlab class dedicated to load and process AO telemetry data. You can instantiate the telemetry object as follows

```
>> p.grabData('objname',{'n0004'});
Selection of 1 .fits file
Selection of 1 .sav file
I'm grabbing data, this may take some time
    .../run/media/omartin/HDD_OBM/KECK_DATA/20130801/TRS/n0004_fullNGS_trs.sav
creating A: structure ARRAY: 1 1
creating UID: int32 SCALAR
creating CID: uint32 SCALAR
creating NREC: int32 SCALAR
creating RX: single ARRAY: 214016   1
creating CENT_ORIGIN: single ARRAY: 608 1
creating DM_ORIGIN: single ARRAY: 349 1
creating DTT_OFFSET: single ARRAY: 2 1
creating TSTAMP_NUM: uint64 ARRAY: 1 1
creating TSTAMP_STR_START: string SCALAR
creating CENT_G: structure ARRAY: 1 1
creating DM_SERVO: single ARRAY: 7 1
creating DT_SERVO: single ARRAY: 7 1
```

which will update the p.trs field from the .fits file corresponding to the object 'n0004'. The trs field contains details about the system, atmosphere and any calibration useful at this stage:

```
>> p.trs

ans =

  telemetry with properties:

          obj_name: 'n0004'
              date: '20130801'
          path_trs:
              '/run/media/omartin/HDD_OBM/KECK_DATA/20130801/TRS/n0004_fullNGS_trs.sav'
         path_imag: '/run/media/omartin/HDD_OBM/KECK_DATA/20130801/IMAG/n0004.fits'
    path_calibration: '/run/media/omartin/HDD_OBM/KECK_DATA/CALIBRATION/'
       path_massdimm: '/run/media/omartin/HDD_OBM/KECK_DATA/CALIBRATION//MASSDIMM/'
             fitsHdr: {196x3 cell}
              aoMode: 'NGS'
             sysTime: 12.0006
                 src: [1x1 struct]
                 ngs: [1x1 struct]
                 lgs: [1x1 struct]
                 tel: [1x1 struct]
                 atm: [1x1 struct]
                 wfs: [1x1 struct]
             tipTilt: [1x1 struct]
                  dm: [1x1 struct]
```

```
           cam: [1x1 struct]
           rec: [1x1 struct]
           mat: [1x1 struct]
        holoop: [1x1 struct]
        ttloop: [1x1 struct]
           res: [1x1 struct]
           sky: [1x1 psfStats]
```

Warning: the trs field of PUAKO is a *structure* and each of its field are structures as well. They are not sub-classes as invoked in OOMAO [2]. For instance p.trs.tel gathers all information about the telescope. We have summarized the trs class properties in Tab. 2. Furthermore, you can specify several files to be loaded by providing the 'obj_name' inputs in the p.grabData call as a cell of char, like {'n0004','n0005',...}. PUAKO will instantiate its trs field as a multiple level property, i.e. the k^{th} data set is accessed to from p.trs{k}. By default, PUAKO will keep all the telemetry in memory, which means that if you ask it to load multiple files, there is a risk to overload your machine's memory. Fortunately, if the number of data you want to keep in memory is too much memory demanding, PUAKO will display an error dialog box to warn you that it can not go with the full loading if it needs more than 95%. Above 70%, PUAKO will display a warning dialog box and request your authorization to load effectively all the data and give you a single warning message in the prompt if it needs less than 70%. At any time, be careful with the memory usage if you want to process multiple files for a cross comparison. In pipeline mode, PUAKO only stores what is needed and erases the telemetry data when passing from a case to another.

When calling the p.grabData method, PUAKO (i) restores the calibrated data (pupil, shape, DM influence functions,...), the .sav file using the _restore_idl facility and the .fits file (ii) models the AO loop transfer functions from the .sav file fields and get the closest MASS-DIMM measurements of your observation (you need a network connection to load the .dat file if it's not in the MASSDIMM folder already) (iii) estimates the noise covariance matrix, the seeing and the outer scale from the telemetry (iv) processes the image (dark subtraction, flat compensation, bad pixels interpolation and background subtraction) and extracts the PSF over 150 pixels (by default, can be changed in function *initializeStructures.m*) (v) fits a Gaussian and a Moffat model over the extracted and clean PSF. The user can access results of the fit from p.trs.sky that is another Matlab class *psfStats* containing the processed image and PSF figures of merit, such as the Strehl-ratio (SR) or the Full Width at Half Maximum (FWHM). Seethe PUAKO note #5 for more information about this class and have a look on the PUAKO note #2 to get more insights on the image processing and the noise/seeing estimation.

Eventually, PUAKO will update the p.trs.res structure that contains a 'noise', 'seeing' and 'zernike' field as follows

```
>> p.trs.res.seeing

ans =

  struct with fields:

        r0: 0.2804
        L0: 17.3754
    seeing: 0.2250
       dr0: 0.1134
       dL0: 12.9309
```

```
    dseeing: 0.1102
         w0: 0.3590
        dw0: 0.1452

>> p.trs.res.noise

ans =

  struct with fields:

     Cn_ho: [441x441 double]
    std_ho: 34.7521
     Cn_tt: [2x2 double]
    std_tt: 19.7180
    method: 'autocorrelation'

>> p.trs.res.zernike

ans =

  struct with fields:

       jindex: [47x1 double]
    std_model: [47x1 double]
     std_meas: [47x1 double]
    std_noise: [47x1 double]
```

where r_0 is the Fried's parameter in meters and at 500 nm, L_0 the outer scale in meter and $w_0 = 0.976 \times \lambda/r_0$ as the seeing value in arcsec at 500 nm that does not account for the outer scale, while the 'seeing' field does [5]. Fields with the letter 'd' in front correspond to $3-\sigma$ precision of the respective quantities. The 'noise' structure permits to obtain the noise covariance matrices in the DM actuators space and in meter^2 as well as the standard-deviation in nm. By default, the noise variance is estimated using the auto-correlation method [3], but the user can specify to use the rejection transfer function [4]. The 'zernike' field refers to the Zernike coefficients standard-deviation in nm given from the reconstructed wavefront (p.trs.rec.res) decomposition onto the Zernike modes, the fitted model, the noise decomposition and the corresponding Noll's j-index. This field is initialized during the seeing estimation and note that different strategies exist to identify the couple (r_0,L_0) from the Zernike decomposition, see the PUAKO note #02 for more details. The user can display the final decomposition by typing `p.trs.displayZernikeFitting` in the command window. On top of that, the user can also display the AO loop transfer function using the `p.trs.displayAoTransferFunction` command, that will pop up the same figure than the one presented in Fig. 1. The high-order and tip-tilt loops lags are modeled with respect to the loop frequency rate as reported in the function *estimateLoopDelay.m*

3.3 PSF reconstruction

To perform the PSF-R, the user has two choices (i) instantiate manually the PUAKO object's 'psfr' field by typing `p.psfr = psfReconstruction(p.trs)` that will run the PSF-R process from the 'trs' field (ii) type `p.getRecPSF('objname',{'n0004'})` to run the pipeline mode. In the latest configuration, PUAKO will load and process all the files you have requested to be treated (if the 'objname' input is not set up, it will

Sources: p.trs.src, ngs, lgs	
x	Position in x in the image in arcsec.
y	Position in y in the image in arcsec.
height	Science target height in meter.
nSrc	Number of image.
nObj	Number of stars within the image (useful for binary)
Telescope, p.trs.tel	
Dhex	Maximal distance inside the hexagonal pupil in m.
Dcircle	Diameter of the circular pupil with same area in m.
Dnoslave	Diameter within the actuators are not salved in m.
Ddm	n_{act}×pitch in m.
cobs	Central obstruction ratio in%
pupil	2D pupil map.
resolution	Pupil resolution in pixels.
pixelscale	Pupil pixel scale in m/pix.
static_map	2D map of static aberrations in nm.
airmass	Telescope airmass.
zenith_angle	Telescope zenith angle in degrees.
elevation	Telescope elevation in degrees.
azimuth	Telescope azimuth in degrees.
pupilAngle	Telescope pupil rotation angle with respect to the detector pupil in degree.
static_fitting	System static fitting 2D map in nm.
Atmosphere, p.trs.atm	
wavelength	Wavelength in meters.
r0	Fried's parameter in meters at wavelength.
L0	Outer scale in meters.
nLayer	Number of atmospheric turbulence layers.
weights	Layer's fractional weights
heights	Layer's altitude in meter.
Cn2	Discretized atmospheric profile in $m^{-5/3}$
windSpeed	Wind velocity profile in m/s.
windDirection	Wind direction profile in radian.
WFS, p.trs.wfs	
slopes	WFS slopes matrix in pixels.
nSl	Total number of slopes.
nSl_c	Number of controlled slopes.
nExp	Number of the HO WFS exposure.
theta	WFS pupil rotation angle with respect to the science detector pupil in degrees.
validSubaperture	2D map of sub-apertures within the pupil.
zstage_defocus	WFS stage defocus in mm.
intensity	Mean WFS pixels intensity map over the full observation.
DM, p.trs.dm	
com	DM actuators command matrix in meters.
volt2meter	Volt to meter conversion factor.
nActuators	Number of DM actuators.
nCom	Number of DM commands.
pitch	DM actuators pitch.
validActuators	2D map of sub-actuators within the pupil.
pupilMask	Pupil mask at the DM actuators resolution.
Wavefront reconstruction, p.trs.rec	
res	Reconstructed wavefront in meter
focus	Reconstructed residual focus in meter.

System matrices, p.trs.mat	
R	Command matrix (HO WFS slopes to DM command in meter).
Rtt	Tip-tilt reconstructor.
dmIF	DM influence matrix at twice the DM actuators resolution.
dmIF_inv	Inverse of dmIF.
Hdm	dmIF.dmIF_inv.
dmIF_hr	DM influence matrix at the pupil resolution.
dmIF_inv_hr	Inverse of dmIF_hr.
Hz	Zernike modes filtering matrix.
nZernMode	Noll's j-index of Zernike modes added to the static map.
Tip Tilt, p.trs.tipTilt	
slopes	Residual Tip-tilt in arcsec.
com	Reconstructed tip-tilt in meters.
tilt2meter	Tip-tilt reconstruction factor.
nExp	Number of tip-tilt WFS exposures.
Camera, p.trs.cam	
name	Camera name.
rawFrame	Raw camera frame.
image	Processed image in ADU.
itime	Integration time in seconds.
coadds	Number of co-adds.
resolution	Processed image resolution in pixels.
pixelScale	Camera pixel scale in arcsec/pix.
fov	Processed image field of view in pixels.
wavelength	Camera filter central wavelength in meter.
samp	wavelength/Ddm/2/pixelScale
badPixelMap	Bad pixels map.
background	Background calibration map.
dark	Dark calibration map.
flat	Flat calibration map.
field_static_map	Calibrated field static map cube in nm.
x_stat	X-axis fiber position in pixels.
y_stat	Y-axis fiber position in pixels.
diff_field_stat	Differential field static aberrations.
x_ncpa	PSF x-position in the detector during NCPA calibration in pixel.
y_ncpa	PSF y-position in the detector during NCPA calibration in pixel.
pupilTracking	Detector pupil configuration.
Loops configuration, trs.holoop, trs.ttloop	
gain	Loop gain.
freq	Loop frequency in Hz.
lat	Loop latency in seconds.
tf.freq	Temporal frequency vector in Hz.
tf.wfs	WFS temporal transfer function.
tf.lag	Loop lag temporal transfer function.
tf.num	Numerator of the servo transfer function.
tf.den	Denominator of the servo transfer function.
tf.servo	Servo transfer function.
tf.ol	Open-loop transfer function.
tf.ctf	Closed-loop transfer function.
tf.rtf	Rejection transfer function.
tf.ntf	WFS noise transfer function.
tf.pn	WFS noise propagation factor.

Table 2: Summary of the *telemetry* class properties.

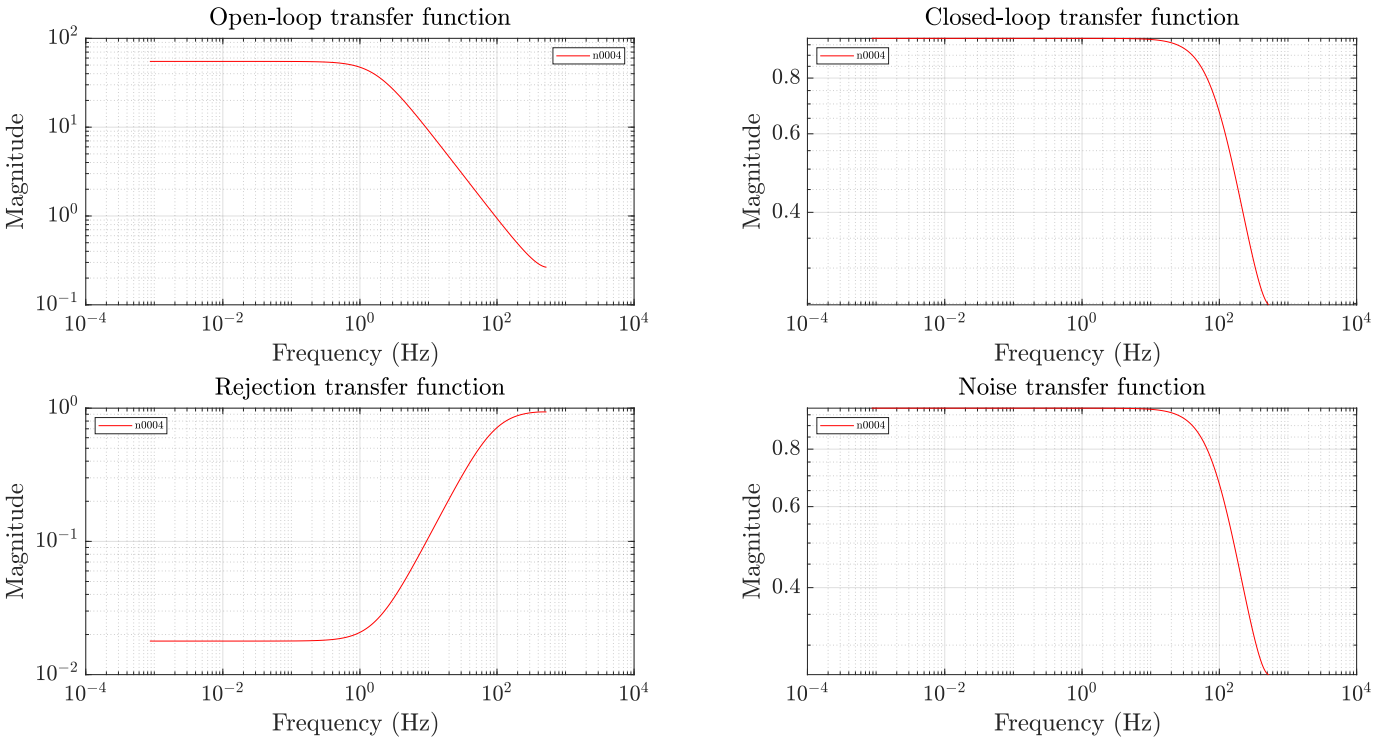

Figure 1: Illustration of the AO temporal transfer functions that PUAKO models from AO system parameters contained in the .sav files fields and calibration of loop lags.

process all the .sav files in the TRS folder) and perform the reconstruction, which should take a couple of seconds. PUAKO instantiates its 'psfr' properties that is a structure as presented as follows

```
>> p.psfr

ans =

  psfReconstruction with properties:

          trs: [1x1 telemetry]
          res: [1x1 struct]
          otf: [1x1 struct]
          cov: [1x1 struct]
           sf: [1x1 struct]
          psf: [1x1 psfStats]
        flags: [1x1 struct]
         rec_: [154x154 double]
   addStatZer: {[]}
  zStat_coefs: 0
  zStat_modes: []
```

The user has access to the different covariance matrices ('psfr.cov'), structure functions ('psfr.sf') and optical transfer functions ('psfr.otf') that serve the reconstruction process. Similarly to the 'p.trs.sky' property, the 'p.psfr.psf' is also a *psfStat* class which groups the image and PSF metrics. Moreover, the re-

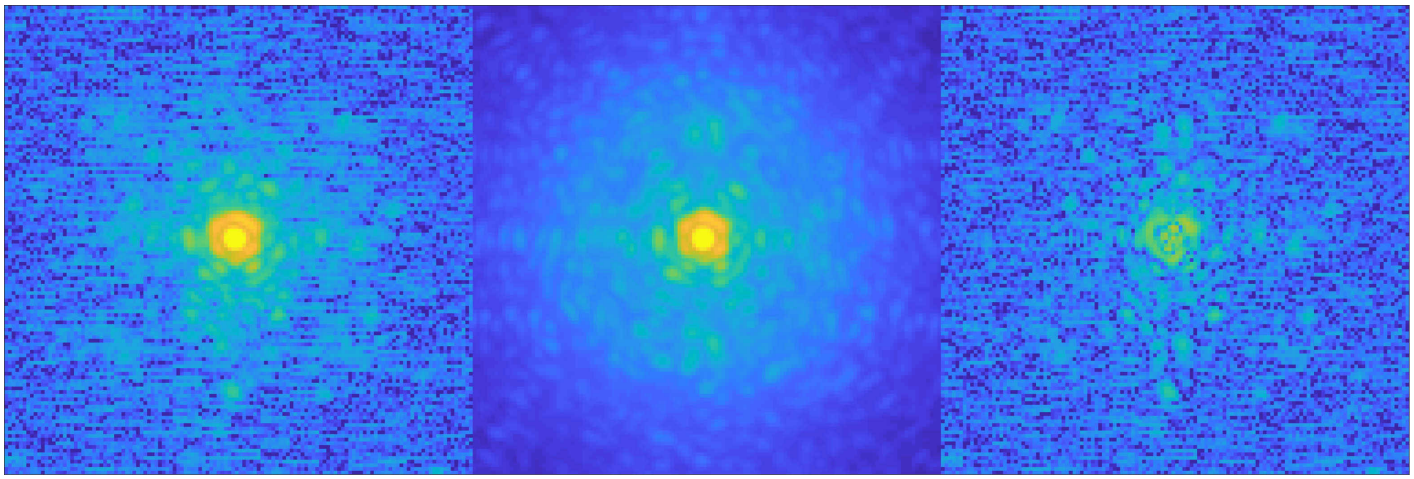

Figure 2: Illustration of a visual inspection of the PSF-R process. **Left:** sky image. **Middle** Reconstructed PSF after fitting. **Right:** residual map.

constructed PSF is adjusted (scaling, 2D position and background if 'fitBg' set to true when instantiating the 'p.psfr' property) over the sky image and the results can be found in 'p.psfr.psf.im_fit'. The command `p.vizualizeResults('resultsof','psfr')` is particularly useful to have a glance at the reconstruction results and will pop up several windows with comparison on the PSF profiles and 2D maps as presented in Fig. 2. Note that the reconstructed PSF (accessible from 'p.psfr.rec_') is encoded in a matrix larger than the final adjusted image to avoid any aliasing effect when translating the PSF (multiplication of the OTF with a phasor) to finely tune the PSF peak position.

Finally, the user can save the PSF-R results by specifying a folder (that must exist already) to save a .fits file containing the reconstructed image by defining path_save as a variable pointing to the saving folder and typing the following command `p.getRecPSF('objname',{'n0004'},'path_save',path_save)`. If you do not provide any path_save entry, PUAKO will request your authorization to go on with the reconstruction and make sure that you do not forget to save your results undeliberately. In the header of the saved .fits file, the user will find useful information about the system status during the closed-loop operation as well as results of the AO telemetry and image processing. We report in Tab. 3 all the fields that are set up when saving the .fits file and what they are corresponding to. If the user wants to add a new field *newfield*, he/she must modify the function *defineHeaderFromResults.m* and define explicitly *hdr.newfield* from whatever information stored within the PUAKO object. The exact reconstruction process process is explained in the PUAKO note #03.

3.4 Hybrid PSF reconstruction

PUAKO offers the possibility to perform hybrid PSF-R using the PRIME algorithm. Similarly as previously, the user must instantiate the 'psfp' field of the PUAKO object by either from the 'p.psfr' property directly using `p.psfp = prime(p.psfr)`, or in pipeline mode using the command `p.getPrimePSF()` to perform the hybrid PSF-R over all data that exist in the TRS folder. Identically to the PSF-R step, the user can visualize the results by typing `p.vizualizeResults('resultsof','prime')` and store the hybrid reconstructed PSF by specifying the path_save in the call. Note that the field 'p.psfp.psf.image' is directly adjusted in flux and position to match the sky observation. PUAKO will automatically name the saved .fits file as *camName_objname_prime.fits* in order to not erase the non-hybrid (or forward) reconstructed PSF file.

3.5 Results analysis

When all the files have been treated, the user can analyze the results statistically by using the Matlab class *statisticalAnalysis*, that is simply called by typing `a = statisticalAnalysis(path_save)` in the command window. This class will automatically read all the saved .fits file existing in the saving folder and fill in its properties that correspond to the field presented in Tab. 3. Note that if you add a new field in the *defineHeaderFromResults.m* function, you must modify the *statisticalAnalysis.m* function as well to access to this new field.
Then, we can plot a field value with respect to any other's one by using the 'displayPlot' method. To draw the reconstructed SR as function of the one assessed on the image, you only need to type the command `a.displayPlot('SKYSR','PRISR','Color','r','xyline',true)` that will then display the Fig. 3

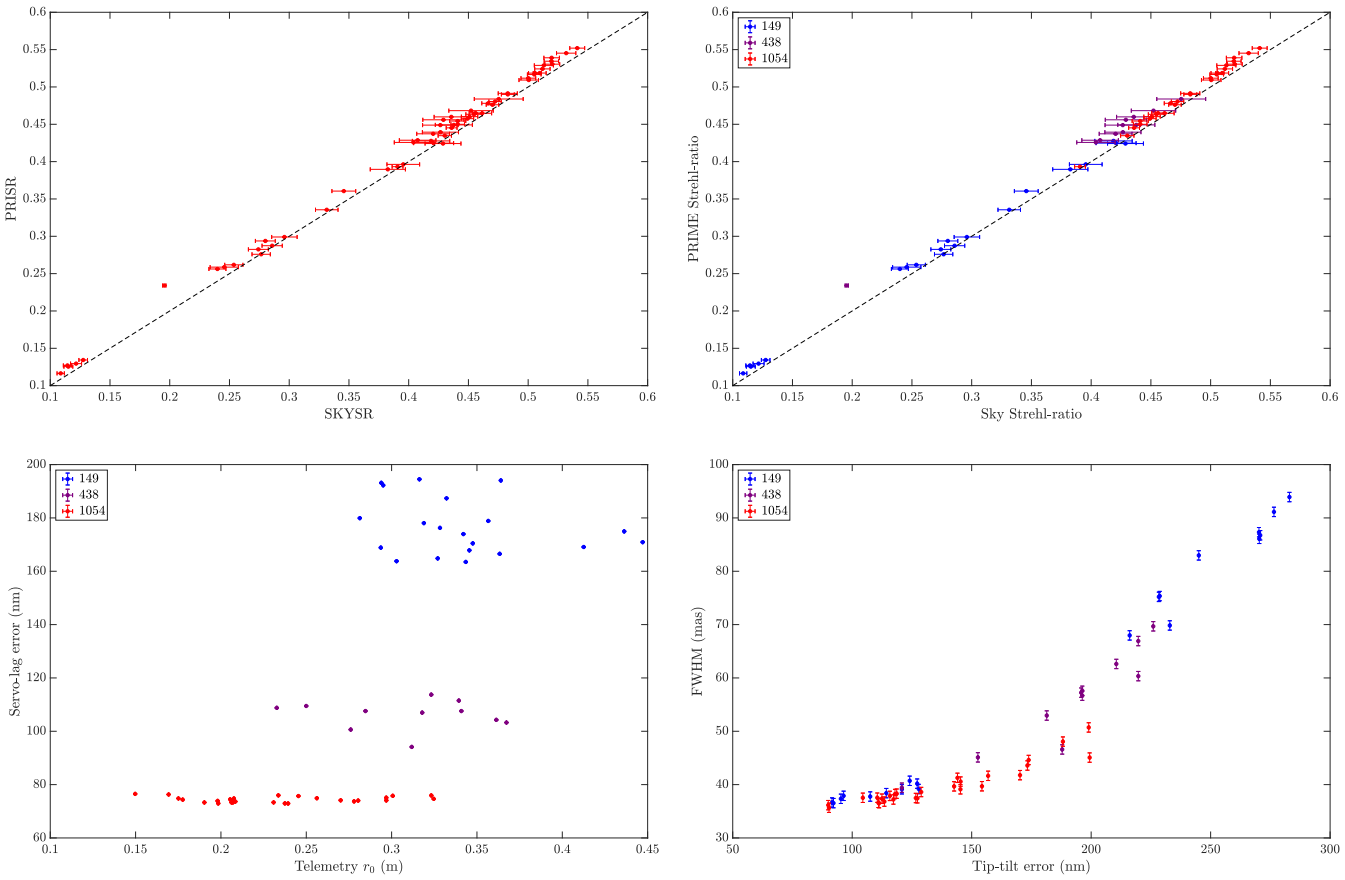

Figure 3: **Top-left:** SR obtained with PRIME as function of the sky SR for the data acquired on the 1st August 2013. **Top-right:** Using the command `a.displayPlot('SKYSR','PRISR','Color','r','xyline',true)`. **Right:** Adding `'sortby','SYSFREQ','legendy','PRIME Strehl-ratio','legendx','Sky Strehl-ratio'` in the call to sort results by loop frequency. **Bottom-left:** Servo-lag error in nm with respect to the r_0. **Bottom-right:** PSF x-axis FWHM as function of the tip-tilt residual error.

If the 'displayPlot' is not convenient for your purpose, you always have access to all the parameters described in Tab. 3 from properties of the *statisticalAnalysis* object (type `a.RECSR` to get the reconstructed Strehl-ratio for instance) as a vector.

System parameters	
SYSGAIN	HO loop gain.
SYSFREQ	HO loop sampling frequency in Hz.
SYSLAT	HO loop latency in seconds.
SYSGAINT	TT loop gain.
SYSFREQT	TT loop sampling frequency in Hz.
SYSLATT	TT loop latency in seconds.
SYSAIRM	Telescope airmass.
SYSEL	Telescope elevation in degree.
SYSAZ	Telescope azimuth in degree.
SYSHLGS	LGS height.
SYSEXP	Number of HO WFS exposure.
MASS-DIMM	
MASSR0	DIMM seeing at 500 nm and a zenith
MASSCN2	MASS C_n^2 profile in $m^{-5/3}$
MASSALT	Layers heights
Telemetry processing results	
TRSR0	Telemetry-based r_0 in meters.
TRSDR0	3-σ precision on TRSR0.
TRSL0	Telemetry-based L_0 in meters.
TRSDL0	3-σ precision on TRSDL0.
TRSSEEK	Telemetry-based Kolomogorov seeing in meters.
TRSSEEK	3-σ precision on TRSSEEK.
TRSSEEV	Telemetry-based seeing accounting for L_0 in meters.
TRSDSEEV	3-σ precision on TRSSEE.
TRSMNO	HO noise in nm.
TRSMNOTT	TT noise in nm.
TRSTIP	Residual tip in nm.
TRSTILT	Residual tilt in nm.
TRSHO	Residual wave-front in nm.
TRSNPH	HO WFS flux in ADU/m^2/s.
TRSNPHTT	TT WFS flux in ADU/m^2/s.
Image processing results	
SKYSR	Image Strehl-ratio.
SKYDSR	3-σ precision on SKYSR.
SKYFWX	X-axis image FWHM.
SKYFWY	Y-axis image FWHM.
SKYDFW	3-σ precision on SKYFWX.
SKYGSR	Gaussian-fit SR.
SKYGDSR	3-σ precision on SKYGSR.
SKYGFWX	Gaussian-fit x-axis FWHM.
SKYGDFWX	3-σ precision on SKYGFWX.
SKYGFWY	Gaussian-fit y-axis FWHM.
SKYGDFWY	3-σ precision on SKYGFWY.
SKYGX	Gaussian-fit x position.
SKYGY	Gaussian-fit y-position.
SKYGDX	3-σ precision on SKYGX.
SKYGDY	3-σ precision on SKYGY.
SKYGF	Gaussian-fit flux.
SKYGDF	3-σ precision on SKYGF.
SKYGFVU	Fit mean residual value.
SKYMSR	Moffat-fit SR
SKYMDSR	3-σ precision on SKYMSR.
SKYMFWX	Moffat fit x-axis FWHM.
SKYMDFWX	3-σ precision on SKYMFWX.
SKYMFWY	Moffat fit y-axis FWHM.
SKYMDFWY	3-σ precision on SKYMFWY.
SKYMX	Moffat-fit x position.
SKYMY	Moffat-fit x position.
SKYMDX	3-σ precision on SKYMX.
SKYMDY	3-σ precision on SKYMY.
SKYMF	Moffat-fit flux.
SKYMDF	3-σ precision on SKYMF.
SKYMFVU	Fit mean residual value.
PSF-R results	
RECSR	Reconstructed SR.
RECFWX	Reconstructed x-axis FWHM in mas.
RECFWY	Reconstructed y-axis FWHM in mas.
RECDFWX	3-σ precision on RECFWX in mas.
RECMAR	Maréchal SR from the PSF-R AO error breakdown.
RECPAR	Parenti SR from the PSF-R AO error breakdown.
RECWFE	Total reconstructed wave-front error in nm.
RECNCPA	NCPA residual in nm.
RECFIT	Fitting error in nm.
RECLAG	Servo-lag error in nm.
RECNOI	Noise error in nm.
RECALIAS	Aliasing error in nm.
RECANISO	Anisoplanatism error in nm.
RECTT	Residual tip-tilt in nm.
RECNOITT	Tip-tilt noise in nm.
RECX	Fitted x-position in mas.
RECY	Fitted y-position in mas.
RECDX	3-σ precision on RECDX in mas.
RECDY	3-σ precision on RECDY in mas.
RECF	Fitted flux on the reconstructed PSF.
RECDF	3-σ precision on RECF.
RECFVU	Mean residual on the PSF.
PRIME results	
PRISR	PRIME SR
PRIFWX	PRIME x-axis FWHM in mas.
PRIFWY	PRIME y-axis FWHM in mas.
PRIDFWX	3-σ precision on PRIFWX.
PRIMAR	Maréchal SR from the PRIME AO error breakdown.
PRIPAR	Parenti SR from the PRIME AO error breakdown.
PRIWFE	Total PRIME wave-front error in nm.
PRINCPA	Residual NCPA in nm.
PRIFIT	Fitting error in nm using the PRIME r_0.
PRILAG	PRIME servo lag error in nm using PRIGAOR.
PRINOI	PRIME noise error in nm using PRIGAOR.
PRIALIAS	PRIME aliasing error in nm using PRIGAL.
PRIANISO	PRIME anisoplanatism error using PRICN2.
PRITT	PRIME tip-tilt error in nm using PRIGTT.
PRINOITT	PRIME tip-tilt noise in nm using PRIGTT.
PRIX	PRIME image x-position in mas.
PRIY	PRIME image y-position in mas.
PRIDX	3-σ precision on PRIX.
PRIDY	3-σ precision on PRIY.
PRIF	Retrieved flux.
PRIDF	3-σ precision on PRIF.
PRIFVU	Mean residual on the PSF
PRIGAO	Retrieved HO gain.
PRIDGAO	3-σ precision on PRIGAO.
PRIGAOR	Retrived HO model gain.
PRIDGAOR	3-σ precision on PRIGAOR.
PRIGTT	Retrieved tip-tilt gain.
PRIDGTT	3-σ precision on PRIGTT.
PRIGAL	Retrieved aliasing gain.
PRIDGAL	3-σ precision on PRIGAL.
PRIR0	Image-retrieved r_0 in meters and at 500 nm.
PRIDR0	3-σ precision on PRIR0.
PRICN2	Line-of-sight retrieved $C_n^2(h)$ in $m^{-5/3}$ and at 500 nm.
PRIDCN2	3-σ precision on PRICN2.
PRICOEF	Retrieved static Zernike coefficient in nm.
PRIDCOEF	3-σ precision on PRICOEF.

Table 3: Header fields of saved .fits files containing the reconstructed PSF.

References

[1] O. Beltramo-Martin, C. M. Correia, S. Ragland, L. Jolissaint, B. Neichel, T. Fusco, and P. L. Wizinowich. PRIME: PSF Reconstruction and Identification for Multiple-source characterization Enhancement - appli-

cation to Keck NIRC2 imager. *M.N.R.A.S* , 487(4):5450–5462, Aug 2019.

[2] C. Correia, R. Conan, and O. Beltramo-Martin et al. OOMAO – Object-Oriented Matlab Adaptive Optics. In *Sixth International Conference on Adaptive Optics for Extremely Large Telescopes*, page 70, November 2019.

[3] E. Gendron and P. Léna. Astronomical adaptive optics. 1: Modal control optimization. *Astron. & Astrophys.* , 291:337–347, November 1994.

[4] L. Jolissaint, S. Ragland, and P. Wizinowich. Adaptive Optics Point Spread Function Reconstruction at W. M. Keck Observatory in Laser Natural Guide Star Modes : Final Developments. In *Adaptive Optics for Extremely Large Telescopes IV (AO4ELT4)*, page E93, October 2015.

[5] P. Martinez, J. Kolb, M. Sarazin, and A. Tokovinin. On the Difference between Seeing and Image Quality: When the Turbulence Outer Scale Enters the Game. *The Messenger*, 141:5–8, September 2010.