

---

# **SHESHA Documentation**

***Release 602***

**COMPASS team**

October 28, 2015



<b>1</b>	<b>Contents:</b>	<b>3</b>
1.1	Atmos . . . . .	3
1.2	Target . . . . .	4
1.3	Sensors . . . . .	5
1.4	RTC . . . . .	10
1.5	Param . . . . .	18
1.6	Dms . . . . .	35
<b>2</b>	<b>Indices and tables</b>	<b>39</b>
	<b>Python Module Index</b>	<b>41</b>
	<b>Index</b>	<b>43</b>



Please, read the README file to install the Shesha module:



## CONTENTS:

## 1.1 Atmos

### 1.1.1 atmos module

**class** `atmos.Atmos`

Bases: `object`

**add\_screen** ()

Add a screen to the atmos object.

**Parameters** `size`: (float) : dimension of the screen (size x size)

`amplitude`: (float) : frac

`altitude`: (float) : altitude of the screen in meters

`windspeed`: (float) : windspeed of the screen [m/s]

`winddir`: (float) : wind direction (°)

`deltax`: (float) : extrude deltax pixels in the x-direction at each iteration

`deltay`: (float) : extrude deltax pixels in the y-direction at each iteration

`device`: (int) : device number

**del\_screen** ()

Delete a screen from the atmos object

**Parameters** `alt` – (float) : altitude of the screen to delete

**disp** ()

Display the screen phase at a given altitude

**Parameters** `alt` – (float) : altitude of the screen to display

**get\_screen** ()

Return a numpy array containing the turbulence at a given altitude

**Parameters** `alt` – (float) :altitude of the screen to get

**list\_alt** ()

Display the list of the screens altitude

**move\_atmos** ()

Move the turbulence in the atmos screen following previous loaded paramters such as windspeed and wind direction

`atmos.atmos_init()`

Create and initialise an atmos object

**Parameters** `c: (naga_context) : context`

`tel: (Param_tel) : telescope settings`

`geom: (Param_geom) : geometry settings`

`loop: (Param_loop) : loop settings`

`wfss: (list of Param_wfs) : (optional) wfs settings`

`target: (Param_target) : (optional) target_settings`

## 1.2 Target

### 1.2.1 target module

**class** `target.Target`

Bases: `object`

**Lambda**

observation wavelength for each target

**add\_layer()**

Add a phase screen dm or atmos as layers of turbulence

**Parameters** `n: (int) : index of the target l_type: (str) : “atmos” or “dm”`

`alt: (float) : altitude`

`xoff: (float) : x-offset`

`yoff: (float) : y-offset`

**apod**

boolean for apodizer

**atmos\_trace()**

Raytracing of the target through the atmosphere

**Parameters** `int: (nTarget) : index of the target`

`atm: (atmos) : atmos to get through`

**dmtrace()**

**get\_amplipup()**

Return the complex amplitude in the pupil plane of the target.

**Parameters** `nTarget – (int) : index of the target`

**get\_image()**

Return the image from the target (or long exposure image according to the requested type) :parameters:

`nTarget: (int) : index of the target`

`type_im: (str) : type of the image to get (“se” or “le”)`

`puponly: (int) : if 1, image computed from phase on the pupil only`



**get\_phase()**  
Return the phase's screen of the target

**Parameters** **nTarget** – (int) : index of the target

**get\_phasetele()**  
Return the telemetry phase of the target

**Parameters** **nTarget** – (int) : index of the target

**Return data** (np.ndarray(ndim=2,np.float32)) : phase screen

**get\_strehl()**  
Return the target's strehl

**Parameters** **nTarget** – (int) : index of the target

**init\_strehlmeter()**  
Initialise target's strehl

**Parameters** **nTarget** – (int) : index of the target

**mag**  
magnitude for each target

**ntargets**  
number of targets

**xpos**  
x positions on sky (in arcsec) for each target

**ypos**  
y positions on sky (in arcsec) for each target

**target.target\_init()**  
Create a cython target from parametres structures

**Parameters** **ctxt**: (naga\_context) :

atm: (Param\_atmos) : atmos settings

geom: (Param\_geom) : geom settings

wfs: (Param\_wfs) : wfs settings

dm: (Param\_dm) : dm settings

## 1.3 Sensors

### 1.3.1 sensors module

**class** `sensors.Sensors`

Bases: `object`

Constructor: `Sensors(nsensors,type_data,npup,nxsub,nvalid,nphase,pdiam,npix,nrebin,nfft,nftota,nphot,lgs,odevice,comm_size,ra`

**Parameters** **nsensors**: (int) :

**type\_data**: list of strings) :

**npup**: (np.ndarray[ndim=1,dtype=np.int64\_t]) :

**nxsub**: (np.ndarray[ndim=1,dtype=np.int64\_t]) :

nvalid: (np.ndarray[ndim=1,dtype=np.int64\_t]) :  
nphase: (np.ndarray[ndim=1,dtype=np.int64\_t]) :  
pdiam: (np.ndarray[ndim=1,dtype=np.float32\_t]) :  
npix: (np.ndarray[ndim=1,dtype=np.int64\_t]) :  
nrebin: (np.ndarray[ndim=1,dtype=np.int64\_t]) :  
nfft: (np.ndarray[ndim=1,dtype=np.int64\_t]) :  
ntota: (np.ndarray[ndim=1,dtype=np.int64\_t]) :  
nphot: (np.ndarray[ndim=1,dtype=np.float32\_t]) :  
lgs: (np.ndarray[ndim=1,dtype=np.int32\_t]) :  
odevice: (int) :  
comm\_size: (int) : MPI communicator size  
rank: (int) : process rank

**get\_amplifoc()**

Return the 'amplifoc' array of a given wfs

**Parameters** **n** – (int) : number of the wfs to get the 'amplifoc' from

**get\_bincube()**

Return the 'bincube' array of a given wfs

**Parameters** **n** – (int) : number of the wfs to get the 'bincube' from

**get\_bining()**

Return the 'binimg' array of a given wfs

**Parameters** **n** – (int) : number of the wfs to get the 'binimg' from

**get\_camclipup()**

Return the 'camclipup' array of a given wfs

**Parameters** **n** – (int) : number of the wfs to get the 'camclipup' from

**get\_imgtele()**

Return the 'image\_telemetry' array of a given wfs

**Parameters** **n** – (int) : number of the wfs to get the 'image\_telemetry' from

**get\_offsets()**

Return the 'offset' array of a given wfs

**Parameters** **n** – (int) : number of the wfs to get the 'offset' from

**get\_phase()**

Return the phase array of a given wfs

**Parameters** **n** – (int) : number of the wfs to get the phase from

**get\_pyrimg()**

Return the image of a pyr wfs

**Parameters** **n** – (int) : number of the wfs to get the image from

**get\_rank()**

Return the rank of one of the sensors wfs

**Parameters** **n** – (int) : index of the wfs to get the rank for

**get\_slopes()**

Return the ‘slopes’ array of a given wfs

**Parameters** *n* – (int) : number of the wfs to get the ‘slopes’ from

**sensors\_addlayer()**

Call function add\_layer from the sutra\_source of a sutra\_wfs of the Sensors

**Parameters** *i*: (int) :

type\_dm: (string) :

alt: (float) :

xoff: (float) :

yoff: (float) :

**sensors\_comping()**

TODO doc

**Parameters** *n* – (in) : index of the wfs

**sensors\_initarr()**

Call the function wfs\_initarrays from a sutra\_wfs of the Sensors

**Parameters**

*n*: (int) : index of the wfs

wfs: (Param\_wfs) :

geom: (Param\_geom) :

**sensors\_initgs()**

Call the function sensors\_initgs

**Parameters** *xpos*: (np.ndarray[ndim=1,dtype=np.float32\_t]) :

*ypos*: (np.ndarray[ndim=1,dtype=np.float32\_t]) :

*Lambda*: (np.ndarray[ndim=1,dtype=np.float32\_t]) :

*mag*: (np.ndarray[ndim=1,dtype=np.float32\_t]) :

*zerop*: (float) :

*size*: (np.ndarray[ndim=1,dtype=np.int64\_t]) :

*noise*: (np.ndarray[ndim=1,dtype=np.float32\_t]) :

*seed*: (np.ndarray[ndim=1,dtype=np.int64\_t]) :

**sensors\_trace()**

Does the raytracing for the wfs phase screen in sutra\_wfs

**Parameters** *n*: (int) :

**type\_trace: (str)** [“all”][raytracing across atmos and dms seen] “dm” : raytracing across dms seen only “atmos” : raytracing across atmos only

*atmos*: (Atmos) :(optional) Atmos object

*dms*: (Dms) : (optional) Dms object

*rst*: (int) : (optional) reset before raytracing if rst = 1

**slopes\_geom()**

Compute the geometric slopes in a sutra\_wfs object

**Parameters** nsensor: (int) : wfs number

**param t** (int) : method (0 or 1)

`sensors.bin2d()`

Returns the input 2D array “array”, binned with the binning factor “binfact”. The input array X and/or Y dimensions needs not to be a multiple of “binfact”; The final/edge pixels are in effect replicated if needed. This routine prepares the parameters and calls the C routine \_bin2d. The input array can be of type long, float or double. Last modified: Dec 15, 2003. Author: F.Rigaut SEE ALSO: \_bin2d

**Parameters** data\_in: (np.ndarray) : data to binned

binfact: (int) : binning factor

`sensors.fft_goodsize()`

find best size for a fft from size s

**Parameters** s: (long) size

`sensors.init_wfs_geom()`

Compute the geometry of WFSs: valid subaps, positions of the subaps, flux per subap, etc...

**Parameters** wfs: (Param\_wfs) : wfs settings

wfs0: (Param\_wfs) : reference wfs settings

n: (int) : index of the wfs (diplay information purpose only)

atmos: (Param\_atmos) : atmos settings

tel: (Param\_tel) : telescope settings

geom: (Param\_geom) : geom settings

target: (Param\_target) : target settings

loop: (Param\_loop) : loop settings

init: (int) : (optional)

verbose: (int) : (optional) display informations if 0

`sensors.make_lgs_prof1d()`

same as prep\_lgs\_prof but cpu only. original routine from rico

**Parameters** p\_tel: (Param\_tel) : telescope settings

prof: (np.ndarray[`dtype=np.float32`]) : Na profile intensity, in arbitrary units

h: (np.ndarray[`dtype=np.float32`]) : altitude, in meters. h MUST be an array with EQUALLY spaced elements.

beam: (float) : size in arcsec of the laser beam

center: (string) : either “image” or “fourier” depending on where the centre should be.

`sensors.noise_cov()`

Compute the diagonal of the noise covariance matrix for a SH WFS (arc-sec<sup>2</sup>) Photon noise:  $(\pi^2/2) \cdot (1/N_{\text{photons}}) \cdot (d/r_0)^2 / (2 \cdot \pi \cdot d/\lambda)^2$  Electronic noise:  $(\pi^2/3) \cdot (wfs.noise^2/N^2_{\text{photons}}) \cdot wfs.npix^2 \cdot (wfs.npix \cdot wfs.pixsize \cdot d/\lambda)^2 / (2 \cdot \pi \cdot d/\lambda)^2$

**Parameters** nw: wfs number p\_wfs: (Param\_wfs) : wfs settings p\_atmos: (Param\_atmos) : atmos settings p\_tel: (Param\_tel) : telescope settings

**Returns** cov : (np.ndarray(`ndim=1,dtype=np.float64`)) : noise covariance diagonal

`sensors.prep_lgs_prof()`

The function returns an image array(double,n,n) of a laser beacon elongated by perspective effect. It is obtained by convolution of a gaussian of width “lgsWidth” arcseconds, with the line of the sodium profile “prof”. The altitude of the profile is the array “h”.

**parameters** nsensors: (int) : wfs index

p\_tel: (Param\_tel) : telescope settings

prof: (np.ndarray[dtype=np.float32]) : Na profile intensity, in arbitrary units

h: (np.ndarray[dtype=np.float32]) : altitude, in meters. h MUST be an array with EQUALLY spaced elements.

beam: (float) : size in arcsec of the laser beam

center: (string) : either “image” or “fourier” depending on where the centre should be.

Computation of LGS spot from the sodium profile: Everything is done here in 1D, because the Na profile is the result of the convolution of a function  $P(x,y) = \text{profile}(x) \cdot \text{dirac}(y)$  by a gaussian function, for which variables  $x$  and  $y$  can be split :  $\exp(-(x^2+y^2)/2.s^2) = \exp(-x^2/2.s^2) * \exp(-y^2/2.s^2)$  The convolution is (symbol  $\$$  denotes integral)  $C(X,Y) = \$\exp(-x^2/2.s^2) * \exp(-y^2/2.s^2) * \text{profile}(x-X) * \text{dirac}(y-Y) dx dy$  First one performs the integration along  $y$   $C(X,Y) = \exp(-Y^2/2.s^2) \$ \exp(-x^2/2.s^2) * \text{profile}(x-X) dx$  which shows that the profile can be computed by - convolving the 1-D profile - multiplying it in the 2nd dimension by a gaussian function

If one has to undersample the initial profile, then some structures may be “lost”. In this case, it’s better to try to “save” those structures by re-sampling the integral of the profile, and then derivating it afterwards. Now, if the initial profile is a coarse one, and that one has to oversample it, then a simple re-sampling of the profile is adequate.

`sensors.type_present()`

Check the present types in a list

**Parameters** liste: (list of str) : list of types

pyr: (int) : set to 1 if the list contains “pyr” (0 else)

roof: (int): set to 1 if the list contains “roof” (0 else)

sh: (int) : set to 1 if the list contains “sh” (0 else)

geo: (int) : set to 1 if the list contains “geo” (0 else)

return 1 if the wfs type is present (0 else)

`sensors.wfs_init()`

Create and initialise a Sensors object

**Parameters** wfs: (list of Param\_wfs) : wfs settings

p\_atmos: (Param\_atmos) : atmos settings

p\_tel: (Param\_tel) : telescope settings

p\_geom: (Param\_geom) : geom settings

p\_target: (Param\_target) : target settings

p\_loop: (Param\_loop) : loop settings

comm\_size: (int) : communicator size

rank: (int) : process rank

dm: (list of Param\_dm) : (optional) dms settings

`sensors.wheremax()`

return the index of the maximum value of the list

**Parameters** `liste` – (list of values) : values to get the index of the maximum from

## 1.4 RTC

### 1.4.1 rtc module

**class** `rtc.Rtc`

Bases: `object`

**add\_Controller()**

Add a controller in the `sutra_controller` vector of the RTC on the GPU

**Parameters** `nactu`: (int) : number of actuators

`delay`: (float) : loop delay

`type_control`: (str) : controller's type

`dms`: (Dms) : `sutra_dms` object (GPU)

`type_dmseen`: (char\*\*) : dms indices controled by the controller

`alt`: (np.ndarray[ndim=1,dtype=np.float32\_t]) : altitudes of the dms seen

`ndm`: (int) : number of dms controled

`Nphi`: (long) : number of pixels in the pupil (used in geo controler case only)

**add\_centroider()**

Add a centroider in the `sutra_centroiders` vector of the RTC on the GPU

**Parameters** `sensor`: (Sensors) : `sutra_sensors` object (GPU)

`nwfs` : (long) : number of wfs

`nvalid`: (long) : number of valid subaps

`type_centro`: (str) : centroider's type

`offset`: (float) :

`scale`: (float) :

**applycontrol()**

Compute the DMs shapes from the commands computed in a `sutra_controller_object`. From the command vector, it computes the voltage command (adding pertrubation voltages, taking delay into account) and then apply it to the dms

**Parameters** `ncontro`: (int) : controller index

**buildcmat()**

Compute the command matrix in a `sutra_controller_ls` object

**Parameters** `ncontro`: (int) : controller index

`nfilt`: (int) : number of modes to filter

`filt_tt`: (int) : (optional) flag to filter TT

**buildcmatmv()**

Compute the command matrix in a `sutra_controller_mv` object

**Parameters** ncontro: (int) : controller index  
cond: (float) : conditioning factor for the Cmm inversion

**docentroids ()**  
Compute the centroids with sutra\_controller #ncontrol object

**Parameters** ncontrol: (optional) controller's index

**docentroids\_geom ()**  
Compute the geometric centroids with sutra\_controller #ncontrol object

**Parameters** ncontrol: (optional) controller's index

**docontrol ()**  
Compute the command to apply on the DMs on a sutra\_controller object

**Parameters** ncontro: (int) : controller index

**docontrol\_geo ()**  
Compute the command to apply on the DMs on a sutra\_controller\_geo object

**Parameters** ncontro: (int) : controller index

**doimat ()**  
Compute the interaction matrix

**Parameters** ncontro: (int) : controller index  
g\_dms: (Dms) : Dms object

**doimat\_geom ()**  
Compute the interaction matrix by using a geometric centroiding method

**Parameters** ncontro: (int) : controller index  
g\_dms: (Dms) : Dms object  
geom: (int) : type of geometric method (0 or 1)

**getCenbuff ()**  
Return the centroids buffer from a sutra\_controller\_ls object. This buffer contains centroids from iteration i-delay to current iteration :parameters:  
ncontro: (int) : controller index

**Returns** data : (np.ndarray[ndim=2,dtype=np.float32\_t]) : centroids buffer

**getCentroids ()**  
Return the centroids computed by the sutra\_rtc object

**Parameters** ncontrol: (int) : controller's index

**Returns** data : (np.ndarray[ndim=1,dtype=np.float32\_t]) : centroids (arcsec)

**getCmmEigenvals ()**  
Return the eigen values of the Cmm decomposition in a sutra\_controller\_mv object :parameters:  
ncontro: (int) : controller index

**Returns** eigenvals : (np.ndarray[ndim=1,dtype=np.float32\_t]) : eigenvalues

**getCom ()**  
Return the command vector from a sutra\_controller object

**Parameters** ncontro: (int) : controller index

**Returns** data : (np.ndarray[ndim=1,dtype=np.float32\_t]) : command vector

**getEigenvals()**

Return the eigen values of the imat decomposition in a sutra\_controller object

**Parameters** ncontro: (int) : controller index

**Returns** eigenvals : (np.ndarray[ndim=1,dtype=np.float32\_t]) : eigenvalues

**getErr()**

Return the command increment (cmat\*slopes) from a sutra\_controller\_ls object

**Parameters** ncontro: (int) : controller index

**Returns** data : (np.ndarray[ndim=1,dtype=np.float32\_t]) : command increment

**getU()**

Return the eigen modes matrix of the imat decomposition from a sutra\_controller\_ls object

**Parameters** ncontro: (int) : controller index

**Returns** U : (np.ndarray[ndim=2,dtype=np.float32\_t]) : eigen modes matrix

**getVoltage()**

Return the voltage vector that will be effectively applied to the DMs

**Parameters** ncontro: (int) : controller index

**Returns** data : (np.ndarray[ndim=1,dtype=np.float32\_t]) : voltage vector

**get\_cmat()**

Return the command matrix from a sutra\_controller object

**Parameters** ncontro: (int) : controller index

**Returns** cmat : (np.ndarray[ndim=2,dtype=np.float32\_t]) : command matrix

**get\_cmm()**

Return the Cmm matrix from a sutra\_controller\_mv object

**Parameters** ncontro: (int) : controller index

**Returns** cmm : (np.ndarray[ndim=2,dtype=np.float32\_t]) : Cmm matrix

**get\_cphim()**

Return the Cphim matrix from a sutra\_controller\_mv object

**:parameters;** ncontro: (int) : controller index

**Returns** cphim : (np.ndarray[ndim=2,dtype=np.float32\_t]) : Cphim matrix

**get\_imat()**

Return the interaction matrix of a sutra\_controller object

**Parameters** ncontro: (int) : controller index

**Returns** imat : (np.ndarray[ndim=2,dtype=np.float32\_t]) : interaction matrix

**get\_mgain()**

Return modal gains from sutra\_controller

**Parameters** ncontro: (int) : controller index

**Returns** mgain : (np.ndarray[ndim=1,dtype=np.float32\_t]) : modal gains



**getcentroids()**

Return the centroids computed by the sutra\_rtc object If ncontrol <= d\_control.size, return rtc.d\_centroids  
Else, compute centroids from wfs[nwfs] with centroider[ncontrol]

**Parameters** ncontrol: (int) : controller's index

g\_wfs: (Sensors) : (optional) sutra\_sensors object

nwfs: (int) : (optional) number of wfs

**Returns** data : (np.ndarray[ndim=1,dtype=np.float32\_t]) : centroids (arcsec)

**getolmeas()**

Return the reconstructed open-loop measurement from a sutra\_controller\_mv object

**Parameters** ncontro: (int) : controller index

**Returns** data : (np.ndarray[ndim=1,dtype=np.float32\_t]) : reconstructed open-loop

**imat\_svd()**

Compute the singular value decomposition of the interaction matrix

**Parameters** ncontro – controller index

**init\_modalOpti()**

Initialize the modal optimization controller : compute the slopes-to-modes matrix and the transfer functions

**Parameters** ncontro: (int) : controller index

nmodes: (int) : number of modes

nrec: (int) : number of recorded open slopes measurements

M2V: (np.ndarray[ndim=2,dtype=np.float32\_t]) : modes-to-volt matrix

gmin: (float) : minimum gain for modal optimization

gmax: (float) : maximum gain for modal optimization

ngain: (int) : Number of tested gains

Fs: (float) : sampling frequency

**init\_proj()**

Initialize the projection matrix for sutra\_controller\_geo object. The projection matrix is  $(IFt \cdot IF)^{*(-1)} \cdot IFt$  where IF is the DMs influence functions matrix

**Parameters** ncontro: (int) : controller index

dms: (Dms) : Dms object

indx\_dm: (np.ndarray[ndim=1,dtype=np.int32\_t]) : indices of where(pup) on DM screen

unitpervolt: (np.ndarray[ndim=1,dtype=np.float32\_t]) : unitpervolt DM parameter

indx\_pup: (np.ndarray[ndim=1,dtype=np.int32\_t]) : indices of where(pup) on ipupil screen

**loadOpenLoop()**

Load an array of recoded open-loop measurements for modal optimization

**Parameters** ncontro: (int) : controller index

ol\_slopes: (np.ndarray[ndim=2, dtype=np.float32\_t]) : open-loop slopes

**loadnoisemat()**

Load the noise vector on a sutra\_controller\_mv object

**Parameters** ncontro: (int) : controller index

N: (np.ndarray[ndim=1,dtype=np.float32\_t]) : noise vector

**modalControlOptimization()**

Compute the command matrix with modal control optimization

**Parameter** ncontro: controller index

**rmcontrol()**

Remove a controller

**sensors\_compslopes()**

Compute the slopes in a sutra\_wfs object. This function is equivalent to docentroids() but the centroids are stored in the sutra\_wfs object instead of the sutra\_rtc object

**Parameters** ncentro: (int) : centroider index

**sensors\_initbcube()**

Initialize npix in the sutra\_centroider\_corr object (useless ?)

**Parameters** ncentro: (int) : centroider's index

**sensors\_initcorr()**

Initialize sutra\_centroider\_corr object

**Parameters** ncentro: (int) : centroider's index

w: (np.ndarray[ndim=1,dtype=np.float32\_t]) : weight

corr\_norm: (np.ndarray[ndim=2,dtype=np.float32\_t]) :

sizeX: (int) :

sizeY: (int) :

interpMat: ([ndim=2,dtype=np.float32\_t]) :

**sensors\_initweights()**

Load the weight array in sutra\_centroider\_wcog object

**Parameters** ncentro: (int) : centroider's index

w: (np.ndarray[ndim=2, dtype=np.float32\_t]) : weight

**setCom()**

Set the command vector of a sutra\_controller object to comvec

**Parameters** ncontro: (int) : controller index

**setEigenvals()**

Set the eigen values of the imat decomposition in a sutra\_controller\_ls object

**Parameters** ncontro: (int) : controller index

eigenvals: (np.ndarray[ndim=1,dtype=np.float32\_t]) : eigen values

**setU()**

Set the eigen modes matrix of the imat decomposition in a sutra\_controller\_ls object

**Parameters** ncontro: (int) : controller index

U: (np.ndarray[ndim=2,dtype=np.float32\_t]) : eigen modes matrix

**set\_cmat()**

Set the command matrix on a sutra\_controller object

**Parameters** ncontro: (int) : controller index  
data: (np.ndarray[ndim=2,dtype=np.float32\_t]) : command matrix to use

**set\_cmm()**  
Set the Cmm matrix on a sutra\_controller\_mv object

**Parameters** ncontro: (int) : controller index  
data: (np.ndarray[ndim=2,dtype=np.float32\_t]) : Cmm matrix

**set\_decayFactor()**  
Set the decay factor on a sutra\_controller\_generic object

**Parameters** ncontro: (int) : controller index  
decay: (np.ndarray[ndim=1,dtype=np.float32\_t]) : ask to Rico

**set\_gain()**  
Set the loop gain in sutra\_controller object

**Parameters** ncontro: (int) : controller index  
gain: (float) : loop gain

**set\_imat()**  
Set the interaction matrix on a sutra\_controller object

**Parameters** ncontro: (int) : controller index  
data: (np.ndarray[ndim=2,dtype=np.float32\_t]) : interaction matrix to use

**set\_matE()**  
Set the matrix E on a sutra\_controller\_generic object

**Parameters** ncontro: (int) : controller index  
matE: (np.ndarray[ndim=2,dtype=np.float32\_t]) : ask to Rico

**set\_mgain()**  
Set modal gains in sutra\_controller object

**Parameters** ncontro: (int) : controller index  
mgain: (np.ndarray[ndim=1,dtype=np.float32\_t]) : modal gains

**setnmax()**  
set the number of brightest pixels to consider for bpcog centroider

**Parameters** ncentro: (int) : centroider's index  
nmax: (int) : number of brightest pixels

**setthresh()**  
set threshold for the centroider #ncentro

**Parameters** ncentro: (int) : centroider's index  
thresh: (float) : threshold

**rtc.cmat\_init()**  
Compute the command matrix on the GPU

**Parameters** ncontro: (int) : g\_rtc: (Rtc) : p\_rtc: (Param\_rtc) : rtc settings p\_wfs: (list of Param\_wfs) : wfs settings p\_tel: (Param\_tel) : telescope settings clean: (int) : (optional) clean datafiles (imat, U, eigenv) simul\_name: (str) : (optional) simulation's name, use for data files' path

`rtc.compute_KL2V()`

Compute the Karhunen-Loeve to Volt matrix (transfer matrix between the KL space and volt space for a pzt dm)

**Parameters** p\_dms: (list of Param\_dm) : dms settings

controller: (Param\_controller) : controller settings

`rtc.correct_dm()`

Correct the geometry of the DMs using the imat (filter unseen actuators)

**Parameters** p\_dms: (list of Param\_dm) : dms settings

g\_dms: (Dms) : Dms object

p\_control: (Param\_controller) : controller settings

p\_geom: (Param\_geom) : geom settings

imat: (np.ndarray) : interaction matrix

simul\_name: (str) : simulation's name, use for data files' path

`rtc.create_interp_mat()`

TODO doc

**Parameters** dimx: (int) :

dimy: (int) :

`rtc.create_nact_geom()`

Compute the DM coupling matrix :param:

p\_dms : (list of Param\_dm) : dms parameters ndm : (int) : dm number

**Returns** Nact : (np.array(dtype=np.float64)) : the DM coupling matrix

`rtc.create_piston_filter()`

`rtc.doTomoMatrices()`

Compute Cmm and Cphim matrices for the MV controller on GPU

**Parameters** g\_wfs: (Sensors) :

p\_wfs: (list of Param\_wfs) : wfs settings

g\_dms: (Dms) :

p\_dms: (list of Param\_dms) : dms settings

p\_geom: (Param\_geom) : geom settings

p\_atmos: (Param\_atmos) : atmos settings

g\_atmos: (Atmos) :

p\_tel: (Param\_tel) : telescope settings

`rtc.get_r0()`

Compute r0 at lambda2 from r0 value at lambda1

**Parameters** r0\_at\_lambda1: (float) : r0 value at lambda1

lambda1: (float) : lambda1

lambda2: (float) : lambda2

`rtc.imat_geom()`

Compute the interaction matrix with a geometric method

**Parameters** g\_wfs: (Sensors) : Sensors object

p\_wfs: (list of Param\_wfs) : wfs settings

p\_control: (Param\_controller) : controller settings

g\_dms: (Dms) : Dms object

p\_dms: (list of Param\_dm) : dms settings

meth: (int) : (optional) method type (0 or 1)

`rtc.imat_init()`

Initialize and compute the interaction matrix on the GPU

**Parameters** ncontro: (int) : controller's index

g\_rtc: (Rtc) : Rtc object

p\_rtc: (Param\_rtc) : rtc settings

g\_dms: (Dms) : Dms object

g\_wfs: (Sensors) : Sensors object

p\_wfs: (list of Param\_wfs) : wfs settings

p\_tel: (Param\_tel) : telescope settings

clean: (int) : (optional) : clean datafiles (imat, U, eigenv)

simul\_name: (str) : (optional) simulation's name, use for data files' path

`rtc.manual_imat()`

Compute the interaction matrix 'manually', ie without sutra\_rtc doimat method

**Parameters** g\_rtc: (Rtc) : Rtc object

g\_wfs: (Sensors) : Sensors object

g\_dms: (Dms) : Dms object

p\_dms: (list of Param\_dm) : dm settings

`rtc.openLoopSlp()`

Return a set of recorded open-loop slopes, usefull for modal control optimization

**Parameters** g\_atm: (Atmos) : Atmos object

g\_rtc: (Rtc) : Rtc object

nrec: (int) : number of samples to record

ncontro: (int) : controller's index

g\_wfs: (Sensors) : Sensors object

p\_wfs: (list of Param\_wfs) : wfs settings

p\_tar: (Param\_target) : target settings

g\_tar: (Target) : Target object

`rtc.rtc_init()`

Initialize all the sutra\_rtc objects : centroiders and controllers

**Parameters** g\_wfs: (Sensors) : Sensors object

p\_wfs: (list of Param\_wfs) : wfs settings

`g_dms`: (Dms) : Dms object  
`p_dms`: (list of Param\_dms) : dms settings  
`p_geom`: (Param\_geom) : geom settings  
`p_atmos`: (Param\_atmos) : atmos settings  
`g_atmos`: (Atmos) : Atmos object  
`p_tel`: (Param\_tel) : telescope settings  
`p_loop`: (Param\_loop) : loop settings  
`p_tar`: (Param\_target) : (optional) target settings  
`clean`: (int) : (optional) clean datafiles (imat, U, eigenv)  
`brama`: (int) : (optional) not implemented yet  
`doimat`: (int) : (optional) force imat computation  
`simul_name`: (str) : (optional) simulation's name, use for path to save data (imat, U...)

**Returns** `Rtc` : (Rtc) : Rtc object

`rtc.selectDMforLayers()`

For each atmos layer, select the DM which have to handle it in the Cphim computation for MV controller  
:parameters:

`ncontro` : (int) : controller number  
`p_atmos` : (Param\_atmos) : atmos parameters  
`p_rtc` : (Param\_rtc) : rtc parameters  
`p_dms` : (list of Param\_dm) : dms parameters

**Returns** `indlayersDM` : (np.array(dtype=np.int32)) : for each atmos layer, the Dm number corresponding to it

## 1.5 Param

### 1.5.1 param module

`class param.Param_atmos`

Bases: object

**L0**

L0 per layers in meters.

**alt**

altitudes of each layer.

**deltax**

x translation speed (in pix / iteration) for each layer.

**deltay**

y translation speed (in pix / iteration) for each layer.

**dim\_screens**

linear size of phase screens.

**frac**

fraction of r0 for each layer.

**nsscreens**  
number of turbulent layers.

**pupixsize**  
pupil pixel size (in meters).

**r0**  
global r0.

**seeds**

**set\_L0 ()**  
Set the L0 per layers  
**Parameters** **l** – (lit of float) : L0 for each layers

**set\_alt ()**  
Set the altitudes of each layer  
**Parameters** **l** – (lit of float) : altitudes

**set\_deltax ()**  
Set the translation speed on axis x for each layer  
**Parameters** **l** – (lit of float) : translation speed

**set\_deltay ()**  
Set the translation speed on axis y for each layer  
**Parameters** **l** – (lit of float) : translation speed

**set\_dim\_screens ()**  
Set the size of the phase screens  
**Parameters** **l** – (lit of float) : phase screens sizes

**set\_frac ()**  
Set the fraction of r0 for each layers  
**Parameters** **l** – (lit of float) : fraction of r0

**set\_nsscreens ()**  
Set the number of turbulent layers  
**Parameters** **n** – (long) number of screens.

**set\_pupixsize ()**  
Set the pupil pixel size  
**Parameters** **xsize** – (float) : pupil pixel size

**set\_r0 ()**  
Set the global r0  
**Parameters** **r** – (float) : global r0

**set\_seeds ()**  
Set the seed for each layer  
**Parameters** **l** – (lit of float) : seed

**set\_winddir ()**  
Set the wind direction for each layer  
**Parameters** **l** – (lit of float) : wind directions

**set\_windspeed()**

Set the the wind speed for each layer

**Parameters** **l** – (lit of float) : wind speeds

**winddir**

wind directions of each layer.

**windspeed**

wind speeds of each layer.

**class** `param.Param_centroider`

Bases: `object`

**interpmat**

optional reference function(s) used for corr centroiding

**nmax**

number of brightest pixels

**nwfs**

index of wfs in `y_wfs` structure on which we want to do centroiding

**set\_nmax()**

Set the number of brightest pixels to use for bpcog

**Parameters** **n**: (int) : number of brightest pixels

**set\_nwfs()**

Set the index of wfs

**Parameters** **n** – (int) : index of wfs

**set\_sizex()**

Set sizex parameters for corr centroider (`interp_mat` size)

**Parameters** **s**: (long) : x size

**set\_sizey()**

Set sizey parameters for corr centroider (`interp_mat` size)

**Parameters** **s**: (long) : y size

**set\_thresh()**

Set the threshold for tcog

**Parameters** **t**: (float) : threshold

**set\_type()**

Set the centroider type :param **t**: (str) : centroider type

**set\_type\_fct()**

Set the type of ref function

**Parameters** **f** – (str) : type of ref function

**set\_weights()**

Set the weights to use with wcog or corr

**Parameters** **w**: (`np.ndarray`[`ndim=3`,`dtype=np.float32_t`]) : weights

**set\_width()**

Set the width of the Gaussian

**Parameters** **w** – (float) : width of the gaussian



**size<sub>x</sub>**  
 x-size for inter mat (corr)

**size<sub>y</sub>**  
 x-size for inter mat (corr)

**thresh**  
 Threshold

**type\_centro**  
 type of centroiding cog, tcog, bpcog, wcog, corr

**type\_fct**  
 type of ref function gauss, file, model

**weights**  
 optional reference function(s) used for centroiding

**width**  
 width of the Gaussian

**class param.Param\_controller**  
 Bases: object

**TTcond**  
 tiptilt condition number for cmat filtering with mv controller

**cmat**  
 full control matrix

**cured\_ndivs**  
 subdivision levels in cured

**delay**  
 loop delay [frames]

**gain**  
 loop gain

**gmax**  
 Maximum gain for modal optimization

**gmin**  
 Minimum gain for modal optimization

**imat**  
 full interaction matrix

**maxcond**  
 max condition number

**modopti**  
 Flag for modal optimization

**nactu**  
 number of controled actuator per dm

**ndm**  
 index of dms in controller

**ngain**  
 Number of tested gains

**nk1**

**nmodes**

Number of modes for M2V matrix (modal optimization)

**nrec**

Number of sample of open loop slopes for modal optimization computation

**nvalid**

number of valid subaps per wfs

**nwfs**

index of wfss in controller

**set\_TTcond()**

Set the tiptilt condition number for cmat filtering with mv controller

:param : (float) : tiptilt condition number

**set\_cmat()**

Set the full control matrix

**Parameters** **cmat** – (np.ndarray[ndim=2,dtype=np.float32\_t]) : full control matrix

**set\_cured\_ndivs()**

Set the subdivision levels in cured

**Parameters** **c** – (long) : subdivision levels in cured

**set\_delay()**

Set the loop delay expressed in frames

**Parameters** **d**: (float) :delay [frames]

**set\_gain()**

Set the loop gain

**Parameters** **g**: (float) : loop gain

**set\_gmax()**

Set the maximum gain for modal optimization

**Parameters** **g** – (float) : maximum gain for modal optimization

**set\_gmin()**

Set the minimum gain for modal optimization

**Parameters** **g** – (float) : minimum gain for modal optimization

**set\_imat()**

Set the full interaction matrix

**Parameters** **imat** – (np.ndarray[ndim=2,dtype=np.float32\_t]) : full interaction matrix

**set\_maxcond()**

Set the max condition number

:param : (float) : max condition number

**set\_modopti()**

Set the flag for modal optimization

**Parameters** **m** – (int) : flag for modal optimization

**set\_nactu()**

Set the number of controlled actuator

**Parameters** **l** – (list of int) : number of controlled actuator per dm

```

set_ndm ()
    Set the indices of dms

    Parameters l – (list of int) : indices of dms

set_ngain ()
    Set the number of tested gains

    Parameters n – (int) : number of tested gains

set_nkl ()
    Set the number of KL modes used for computation of covmat in case of minimum variance controller

    Parameters n – (long) : number of KL modes

set_nmodes ()
    Set the number of modes for M2V matrix (modal optimization)

    Parameters n – (int) : number of modes

set_nrec ()
    Set the number of sample of open loop slopes for modal optimization computation

    Parameters n – (int) : number of sample

set_nvalid ()
    Set the number of valid subaps

    Parameters l – (list of int) : number of valid subaps per wfs

set_nwfs ()
    Set the indices of wfs

    Parameters l – (list of int) : indices of wfs

set_type ()

type_control
    type of controller

class param.Param_dm
    Bases: object

    _com
        current command

    _i1

    _influ
        influence functions

    _influkernel

    _influpos

    _influsize
        total number of actuators

    _influstart

    _j1

    _klbas
        np.ndarray to a kl struct

    _n1
        position of leftmost pixel in largest support

```

**\_n2**  
position of rightmost pixel in largest support

**\_ninflu**  
Influence functions

**\_ntotact**  
total number of actuators

**\_pitch**  
inter-actuator space in pixels

**\_pupil**  
pupil mask for this dm

**\_puppixoffset**

**\_xpos**  
x positions of influ functions

**\_ypos**  
y positions of influ functions

**alt**  
conjugaison altitude (im m)

**coupling**  
actuators coupling (<0.3)

**hyst**  
actuators hysteresis (<1.)

**margin**

**nact**  
number of actuators in the diameter

**nk1**  
number of kl modes

**pupoffset**  
2.

**push4imat**  
nominal voltage for imat

**set\_alt()**  
set the conjugaison altitude

**Parameters a** – (float) : conjugaison altitude (im m)

**set\_coupling()**  
set the actuators coupling

**Parameters c** – (float) : actuators coupling (<0.3)

**set\_i1()**  
TODO doc

**Parameters i1** – (np.ndarray[ndim=1,dtype=np.int32\_t]) :

**set\_influ()**  
Set the influence function

**Parameters influ** – (np.ndarray[ndim=3,dtype=np.float32\_t]) : influence function

```

set_j1 ()
    TODO doc

    Parameters j1 – (np.ndarray[ndim=1,dtype=np.int32_t]) :

set_nact ()
    set the number of actuator

    Parameters n – (long) : number of actuators in the diameter

set_ntotact ()
    set the total number of actuators

    Parameters n – (long) : total number of actuators

set_push4imat ()
    set the nominal voltage for imat

    Parameters p – (float) : nominal voltage for imat

set_thresh ()
    set the threshold on response for selection

    Parameters t – (float) : threshold on response for selection (<1)

set_type ()
    set the dm type

    Parameters t – (str) : type of dm

set_unitpervolt ()
    set the Influence function sensitivity

    Parameters u – (float) : Influence function sensitivity in unit/volt

set_xpos ()
    Set the x positions of influ functions

    Parameters xpos – (np.ndarray[ndim=1,dtype=np.float32_t]) : x positions of influ functions

set_ypos ()
    Set the y positions of influ functions

    Parameters ypos – (np.ndarray[ndim=1,dtype=np.float32_t]) : y positions of influ functions

thresh
    threshold on response for selection (<1)

type_dm
    type of dm

unitpervolt
    Influence function sensitivity in unit/volt. Optional [0.01] Stackarray: mic/volt, Tip-tilt: arcsec/volt.

class param.Param_geom
    Bases: object
    _mpupil
    _n
    _n1
    _n2
    _p1

```

**\_p2**

**cent**  
central point of the simulation.

**geom\_init()**  
Initialize the system geometry

**Parameters** **tel**: (Param\_tel) : telescope settings

pupdiam: (long) : linear size of total pupil

apod: (int) : apodizer

**get\_ipupil()**  
return the full pupil support

**get\_mpupil()**  
return the padded pupil

**get\_n()**  
Return the linear size of the medium pupil

**get\_n1()**  
Return the min(x,y) for valid points for the total pupil

**get\_n2()**  
Return the max(x,y) for valid points for the total pupil

**get\_p1()**  
Return the min(x,y) for valid points for the medium pupil

**get\_p2()**  
Return the max(x,y) for valid points for the medium pupil

**get\_spupil()**  
return the small pupil

**pupdiam**  
linear size of total pupil (in pixels).

**set\_cent()**  
Set the central point of the simulation

**Parameters** **c** – (float) : central point of the simulation.

**set\_pupdiam()**  
Set the linear size of total pupil

**Parameters** **p** – (long) : linear size of total pupil (in pixels).

**set\_ssize()**  
Set linear size of full image

**Parameters** **s** – (long) : linear size of full image (in pixels).

**set\_zenithangle()**  
Set observations zenith angle

**Parameters** **z** – (float) : observations zenith angle (in deg).

**ssize**  
linear size of full image (in pixels).

**zenithangle**  
observations zenith angle (in deg).

```

class param.Param_loop
    Bases: object

    ittime
        iteration time (in sec)

    niter
        number of iterations

    set_ittime ()
        Set iteration time

        Parameters t: (float) :iteration time

    set_niter ()
        Set the number of iteration

        Parameters n: (long) : number of iteration

class param.Param_rtc
    Bases: object

    centroiders

    controllers

    nwfs

    set_centroiders ()
        Set the centroiders

        Parameters l – (list of Param_centroider) : centroiders settings

    set_controllers ()
        Set the controller

        Parameters l – (list of Param_controller) : controllers settings

    set_nwfs ()
        Set the number of wfs

        Parameters n – (int) number of wfs

class param.Param_target
    Bases: object

    Lambda
        observation wavelength for each target

    apod
        boolean for apodizer

    dms_seen
        index of dms seen by the target

    mag
        magnitude for each target

    ntargets
        number of targets

    set_Lambda ()
        Set the observation wavelength

        Parameters l – (list of float) : observation wavelength for each target

```

**set\_apod()**

Tells if the apodizer is used

The apodizer is used if a is not 0 :param a: (int) boolean for apodizer

**set\_mag()**

set the magnitude

**Parameters** **1** – (list of float) : magnitude for each target

**set\_nTargets()**

Set the number of targets

**Parameters** **n** – (int) : number of targets

**set\_xpos()**

Set the x positions on sky (in arcsec)

**Parameters** **1** – (list of float) : x positions on sky for each target

**set\_ypos()**

Set the y positions on sky (in arcsec)

**Parameters** **1** – (list of float) : y positions on sky for each target

**xpos**

x positions on sky (in arcsec) for each target

**ypos**

y positions on sky (in arcsec) for each target

**zerop**

target flux for magnitude 0

**class** param.**Param\_tel**

Bases: object

**cobs**

central obstruction ratio.

**diam**

telescope diameter (in meters).

**nbrmissing**

number of missing segments for EELT pupil (max is 20).

**pupangle**

rotation angle of pupil.

**referr**

std of reflectivity errors for EELT segments (fraction).

**set\_cobs()**

set the central obstruction ratio

**Parameters** **c** – (float) : central obstruction ratio

**set\_diam()**

set the telescope diameter

**Parameters** **d** – (float) : telescope diameter (in meters)

**set\_nbrmissing()**

set the number of missing segments for EELT pupil

**Parameters** **nb** – (long) : number of missing segments for EELT pupil (max is 20)



**set\_pupangle()**

set the rotation angle of pupil

**Parameters** **p** – (float) : rotation angle of pupil

**set\_referr()**

set the std of reflectivity errors for EELT segments

**Parameters** **ref** – (float) : std of reflectivity errors for EELT segments (fraction)

**set\_spiders\_type()**

set the secondary supports type

**Parameters** **spider** – (str) : secondary supports type

**set\_t\_spiders()**

set the secondary supports ratio

**Parameters** **spider** – (float) : secondary supports ratio

**set\_type\_ap()**

set the EELT aperture type

**Parameters** **t** – (str) : EELT aperture type

**spiders\_type**

secondary supports type: “four” or “six”.

**t\_spiders**

secondary supports ratio.

**type\_ap**

EELT aperture type: “Nominal”, “BP1”, “BP3”, “BP5” (for back-up plan with 1, 3, or 5 missing annulus).

**class param.Param\_wfs**

Bases: object

**Lambda**

observation wavelength (in  $\mu\text{m}$ ) for a subap.

**\_Nfft**

array size for fft for a subap (in pixel)

**\_Ntot**

total size of hr image for a subap (in pixel)

**\_altna**

corresponding altitude

**\_azimuth**

angles of rotation for each spot

**\_beam**

1d beam function

**\_binmap**

(int\*) array of pixels transform from full FoV hr images to binned images

**\_fluxPerSub**

fraction of nphotons per subap

**\_ftbeam**

1d beam function fft

**\_halfxy**  
(float\*) phase offset for 1/2 pixel shift in (x,y)

**\_hrmap**  
(int\*) array of pixels transform from minimal FoV image to (in case type is sh or geo)

**\_istart**  
(int\*) x start indexes for cutting phase screens

**\_isvalid**  
(int\*) array of 0/1 for valid subaps

**\_jstart**  
(int\*) y start indexes for cutting phase screens

**\_lgskern**  
lgs kernels for each subap

**\_nphotons**  
number of photons per subap

**\_nrebin**  
rebin factor from hr to binned image for a subap

**\_nvalid**  
number of valid subaps

**\_pdiam**  
pupil diam for a subap (in pixel)

**\_phasemap**  
(int\*) array of pixels transform from phase screen into subaps phase screens

**\_prof1d**  
hr profile

**\_profcum**  
hr profile cumulated

**\_profna**  
sodium profile

**\_pyr\_cx**

**\_pyr\_cy**

**\_pyr\_offsets**

**\_qpixsize**  
quantum pixel size for the simulation

**\_subapd**  
subap diameter (m)

**\_submask**  
(float\*) fieldstop for each subap

**\_validsubsx**  
(int\*) indices of valid subaps along axis x

**\_validsubsy**  
(int\*) indices of valid subaps along axis y

**atmos\_seen**  
1 if the WFS sees the atmosphere layers

**beamsize**  
laser beam fwhm on-sky (in arcsec).

**dms\_seen**  
index of dms seen by the WFS

**fracsub**  
minimal illumination fraction for valid subaps.

**fssize**  
size of field stop in arcsec.

**fstop**  
size of field stop in arcsec.

**gsalt**  
altitude of guide star (in m) 0 if ngs.

**gsmag**  
magnitude of guide star.

**kernel**

**laserpower**  
laser power in W.

**lgsreturnperwatt**  
return per watt factor (high season : 10 ph/cm2/s/W).

**lltx**  
x position (in meters) of llt.

**llty**  
y position (in meters) of llt.

**noise**  
desired noise :  $< 0$  = no noise /  $0$  = photon only /  $> 0$  photon + ron.

**npix**  
number of pixels per subap.

**nxsub**  
linear number of subaps.

**openloop**  
1 if in “open-loop” mode (i.e. does not see dm).

**optthroughput**  
wfs global throughput.

**pixsize**  
pixel size (in arcsec) for a subap.

**proftype**  
type of sodium profile “gauss”, “exp”, etc ...

**pyr\_ampl**  
pyramid wfs modulation amplitude radius [arcsec].

**pyr\_loc**  
Location of modulation, before/after the field stop. valid value are “before” or “after” (default “after”).

**pyr\_npts**  
total number of point along modulation circle [unitless].

**pyrtype**

Type of pyramid, either 0 for “Pyramid” or 1 for “RoofPrism”.

**set\_Lambda()**

Set the observation wavelength

**Parameters** **L** – (float) : observation wavelength (in  $\mu\text{m}$ ) for a subap

**set\_altna()**

Set the corresponding altitude

**Parameters** **a** – (np.ndarray[ndim=1,dtype=np.float32]) : corresponding altitude

**set\_atmos\_seen()**

Tells if the wfs sees the atmosphere layers

**Parameters** **i** – (int) : 1 if the WFS sees the atmosphere layers

**set\_beamsize()**

Set the laser beam fwhm on-sky

**Parameters** **b** – (float) : laser beam fwhm on-sky (in arcsec)

**set\_dms\_seen()**

Set the index of dms seen by the WFS

**Parameters** **dms\_seen** – (np.ndarray[ndim=1,dtype=np.int32\_t]) : index of dms seen by the WFS

**set\_fracsub()**

Set the minimal illumination fraction for valid subaps

**Parameters** **f** – (float) : minimal illumination fraction for valid subaps

**set\_fssize()**

Set the size of field stop

**Parameters** **f** – (float) : size of field stop in arcsec

**set\_fstop()**

Set the size of field stop

**Parameters** **f** – (str) : size of field stop in arcsec

**set\_gsalt()**

Set the altitude of guide star

**Parameters** **g** – (float) : altitude of guide star (in m) 0 if ngs

**set\_gsmag()**

Set the magnitude of guide star

**Parameters** **g** – (float) : magnitude of guide star

**set\_kernel()**

Set the attribute kernel

**Parameters** **k** – (float) :

**set\_laserpower()**

Set the laser power

**Parameters** **l** – (float) : laser power in W

**set\_lgsreturnperwatt()**

Set the return per watt factor

**Parameters** **lpw** – (float) : return per watt factor (high season : 10 ph/cm2/s/W)

**set\_lltx()**  
Set the x position of llt

**Parameters** **l** – (float) : x position (in meters) of llt

**set\_llty()**  
Set the y position of llt

**Parameters** **l** – (float) : y position (in meters) of llt

**set\_noise()**  
Set the desired noise

**Parameters** **n** – (float) : desired noise :  $< 0$  = no noise /  $0$  = photon only /  $> 0$  photon + ron

**set\_npix()**  
Set the number of pixels per subap

**Parameters** **n** – (long) : number of pixels per subap

**set\_nxsub()**  
Set the linear number of subaps

**Parameters** **n** – (long) : linear number of subaps

**set\_openloop()**  
Set the loop state (open or closed)

**Parameters** **o** – (long) : 1 if in “open-loop” mode (i.e. does not see dm)

**set\_optthroughput()**  
Set the wfs global throughput

**Parameters** **o** – (float) : wfs global throughput

**set\_pixsize()**  
Set the pixel size

**Parameters** **p** – (float) : pixel size (in arcsec) for a subap

**set\_profna()**  
Set the sodium profile

**Parameters** **p** – (np.ndarray[ndim=1,dtype=np.float32]) : sodium profile

**set\_proftype()**  
Set the type of sodium profile

**Parameters** **p** – (str) : type of sodium profile “gauss”, “exp”, etc ...

**set\_pyr\_ampl()**  
Set the pyramid wfs modulation amplitude radius

**Parameters** **p** – (float) : pyramid wfs modulation amplitude radius (in arsec)

**set\_pyr\_loc()**  
Set the location of modulation

**Parameters** **p** – (str) : location of modulation, before/after the field stop. valid value are “before” or “after” (default “after”)

**set\_pyr\_npts()**  
Set the total number of point along modulation circle

**Parameters** **p** – (long) : total number of point along modulation circle

**set\_pyrtype()**

Set the type of pyramid,

**Parameters** **p** – (str) : type of pyramid, either 0 for “Pyramid” or 1 for “RoofPrism”

**set\_type()**

Set the type of wfs

**Parameters** **t** – (str) : type of wfs (“sh” or “pyr”)

**set\_xpos()**

Set the guide star x position on sky

**Parameters** **x** – (float) : guide star x position on sky (in arcsec)

**set\_ypos()**

Set the guide star y position on sky

**Parameters** **y** – (float) : guide star y position on sky (in arcsec)

**set\_zerop()**

Set the detector zero point

**Parameters** **z** – (float) : detector zero point

**type\_wfs**

type of wfs : “sh” or “pyr”.

**xpos**

guide star x position on sky (in arcsec).

**ypos**

guide star x position on sky (in arcsec).

**zerop**

detector zero point.

**param.indices()**

DOCUMENT indices(dim) Return a dimxdimx2 array. First plane is the X indices of the pixels in the dimxdim array. Second plane contains the Y indices. Inspired by the Python scipy routine of the same name. New (June 12 2002): dim can either be : - a single number N (e.g. 128) in which case the returned array are

square (NxN)

- a Yorick array size, e.g. [#dimension,N1,N2], in which case the returned array are N1xN2

- a vector [N1,N2], same result as previous case

F.Rigaut 2002/04/03 SEE ALSO: span

**Parameters** **dim1**: (int) : first dimension **dim2**: (int) : (optional) second dimension

**param.make\_apodizer()**

TODO doc

**Parameters** (int) : im:

(int) : pupd:

(str) : filename:

(float) : angle:

**param.makegaussian(size, fwhm, xc, yc)**

Returns a centered gaussian of specified size and fwhm. norm returns normalized 2d gaussian

**Parameters** size: (int) :

  fwhm: (float) :

  xc: (int) : (optional) center position on x axis

  yc: (int) : (optional) center position on y axis

  norm: (int) : (optional) normalization

`param.rotate()`

Rotates an image of an angle “ang” (in DEGREES).

The center of rotation is cx,cy. A zoom factor can be applied.

(cx,cy) can be omitted :one will assume one rotates around the center of the image. If zoom is not specified, the default value of 1.0 is taken.

**Parameters** im: (np.ndarray[ndim=3,dtype=np.float32\_t]) : array to rotate

  ang: (float) : rotation angle (in degrees)

  cx: (int) : (optional) rotation center on x axis (default: image center)

  cy: (int) : (optional) rotation center on x axis (default: image center)

  zoom: (float) : (optional) zoom factor (default =1.0)

`param.rotate3d()`

Rotates an image of an angle “ang” (in DEGREES).

The center of rotation is cx,cy. A zoom factor can be applied.

(cx,cy) can be omitted :one will assume one rotates around the center of the image. If zoom is not specified, the default value of 1.0 is taken.

modif dg : allow to rotate a cube of images with one angle per image

**Parameters** im: (np.ndarray[ndim=3,dtype=np.float32\_t]) : array to rotate

  ang: (np.ndarray[ndim=1,dtype=np.float32\_t]) : rotation angle (in degrees)

  cx: (int) : (optional) rotation center on x axis (default: image center)

  cy: (int) : (optional) rotation center on x axis (default: image center)

  zoom: (float) : (optional) zoom factor (default =1.0)

## 1.6 Dms

### 1.6.1 dms module

**class** `dms.Dms`

Bases: `object`

**add\_dm()**

**comp\_oneactu()**

Compute the shape of the dm when pushing the nactu actuator

**Parameters** type\_dm: (str) : dm type

  alt: (float) : dm conjugaison altitude

  nactu: (int) : actuator number pushed

ampli: (float) : amplitude

**computeKLbasis()**

**Compute a Karhunen-Loeve basis for the dm:**

- compute the phase covariance matrix on the actuators using Kolmogorov
- compute the geometric covariance matrix
- double diagonalisation to obtain KL basis

**Parameters** type\_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

xpos: (np.ndarray[ndim=1,dtype=np.float32\_t]) : x-position of actuators

ypos: (np.ndarray[ndim=1,dtype=np.float32\_t]) : y-position of actuators

indx\_pup: (np.ndarray[ndim=1,dtype=np.int32\_t]) : indices of where(pup)

dim: (long) : number of where(pup)

norm: (float) : normalization factor

ampli: (float) : amplitude

**getComm()**

Return the voltage command of the sutra\_dm

**Parameters** type\_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

**Returns** data : (np.ndarray(dims=1,dtype=np.float32)) : voltage vector

**getInflu()**

Return the influence functions of the DM

**Parameters** type\_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

**Returns** data : (np.ndarray(dims=3,dtype=np.float32)) : influence functions

**get\_KLbasis()**

Return the klbasis computed by computeKLbasis

**Parameters** type\_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

**Returns** KLbasis : (np.ndarray(dims=2,dtype=np.float32)) : the KL basis

**get\_dm()**

Return the shape of the dm

**Parameters** type\_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

**Returns** data : (np.ndarray(dims=2,dtype=np.float32)) : DM shape

**load\_kl()**

Load all the arrays computed during the initialization for a kl DM in a sutra\_dms object



**Parameters** alt: (float) : dm conjugaison altitude

rabas: (np.ndarray[ndim=1,dtype=np.float32\_t]) : TODO

azbas: (np.ndarray[ndim=1,dtype=np.float32\_t]) :

ord: (np.ndarray[ndim=1,dtype=np.int32\_t]) :

cr: (np.ndarray[ndim=1,dtype=np.float32\_t]) :

cp: (np.ndarray[ndim=1,dtype=np.float32\_t]) :

**load\_pzt()**

Load all the arrays computed during the initialization for a pzt DM in a sutra\_dms object

**Parameters** alt: (float) : dm conjugaison altitude

influ: (np.ndarray[ndim=3,dtype=np.float32\_t]) : influence functions

influpos: (np.ndarray[ndim=1,dtype=np.int32\_t]) : positions of the IF

**npoints:** (np.ndarray[ndim=1,dtype=np.int32\_t]) [for each pixel on the DM screen,] the number of IF which impact on this pixel

istart: (np.ndarray[ndim=1,dtype=np.int32\_t]) :

xoff: (np.ndarray[ndim=1,dtype=np.int32\_t]) : x-offset

yoff: (np.ndarray[ndim=1,dtype=np.int32\_t]) :y-offset

kern: (np.ndarray[ndim=1,dtype=np.float32\_t]) : convoltuon kernel

**load\_tt()**

Load all the arrays computed during the initialization for a tt DM in a sutra\_dms object

**Parameters** alt: (float) : dm conjugaison altitude

influ: (np.ndarray[ndim=3,dtype=np.float32\_t]) : influence functions

**oneactu()**

Push on on the nactu actuator of the DM with ampli amplitude and compute the corresponding shape

**Parameters** type\_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

nactu: (int) : actuator number

ampli: (float): amplitude

**remove\_dm()**

Remove a dm from a Dms object

**Parameters** type\_dm: (str) : dm type to remove

alt: (float) : dm conjugaison altitude to remove

**resetdm()**

Reset the shape of the DM to 0

**Parameters** type\_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

**set\_comm()**

Set the voltage command on a sutra\_dm

type\_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

comm: (np.ndarray[indim=1,dtype=np.float32\_t]) : voltage vector

**shape\_dm()**

Compute the shape of the DM in a sutra\_dm object

type\_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

dms.**comp\_dmgeom()**

Compute the geometry of a DM : positions of actuators and influence functions

**Parameters** dm: (Param\_dm) : dm settings

geom: (Param\_geom) : geom settings

dms.**compute\_klbasis()**

**Compute a Karhunen-Loeve basis for the dm:**

- compute the phase covariance matrix on the actuators using Kolmogorov
- compute the geometric covariance matrix
- double diagonalisation to obtain KL basis

**Parameters** g\_dm: (Dms) : Dms object

p\_dm: (Param\_dm) : dm settings

p\_geom: (Param\_geom) : geom settings

p\_atmos: (Param\_atmos) : atmos settings

p\_tel: (Param\_tel) : telescope settings

dms.**dm\_init()**

Create and initialize a Dms object on the gpu

**Parameters** p\_dms: (list of Param\_dms) : dms settings

p\_wfs: (Param\_wfs) : wfs settings

p\_geom: (Param\_geom) : geom settings

p\_tel: (Param\_tel) : telescope settings

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



**a**

atmos, 3

**d**

dms, 35

**p**

param, 18

**r**

rtc, 10

**s**

sensors, 5

**t**

target, 4



## Symbols

\_Nfft (param.Param\_wfs attribute), 29  
 \_Ntot (param.Param\_wfs attribute), 29  
 \_altna (param.Param\_wfs attribute), 29  
 \_azimuth (param.Param\_wfs attribute), 29  
 \_beam (param.Param\_wfs attribute), 29  
 \_binmap (param.Param\_wfs attribute), 29  
 \_com (param.Param\_dm attribute), 23  
 \_fluxPerSub (param.Param\_wfs attribute), 29  
 \_ftbeam (param.Param\_wfs attribute), 29  
 \_halfxy (param.Param\_wfs attribute), 29  
 \_hrmap (param.Param\_wfs attribute), 30  
 \_i1 (param.Param\_dm attribute), 23  
 \_influ (param.Param\_dm attribute), 23  
 \_influxkernel (param.Param\_dm attribute), 23  
 \_influpos (param.Param\_dm attribute), 23  
 \_influsize (param.Param\_dm attribute), 23  
 \_influstart (param.Param\_dm attribute), 23  
 \_istart (param.Param\_wfs attribute), 30  
 \_isvalid (param.Param\_wfs attribute), 30  
 \_j1 (param.Param\_dm attribute), 23  
 \_jstart (param.Param\_wfs attribute), 30  
 \_klbas (param.Param\_dm attribute), 23  
 \_lgs kern (param.Param\_wfs attribute), 30  
 \_mpupil (param.Param\_geom attribute), 25  
 \_n (param.Param\_geom attribute), 25  
 \_n1 (param.Param\_dm attribute), 23  
 \_n1 (param.Param\_geom attribute), 25  
 \_n2 (param.Param\_dm attribute), 23  
 \_n2 (param.Param\_geom attribute), 25  
 \_ninflu (param.Param\_dm attribute), 24  
 \_nphotons (param.Param\_wfs attribute), 30  
 \_nrebin (param.Param\_wfs attribute), 30  
 \_ntotact (param.Param\_dm attribute), 24  
 \_nvalid (param.Param\_wfs attribute), 30  
 \_p1 (param.Param\_geom attribute), 25  
 \_p2 (param.Param\_geom attribute), 25  
 \_pdiam (param.Param\_wfs attribute), 30  
 \_phasemap (param.Param\_wfs attribute), 30  
 \_pitch (param.Param\_dm attribute), 24  
 \_prof1d (param.Param\_wfs attribute), 30  
 \_profcum (param.Param\_wfs attribute), 30

\_profna (param.Param\_wfs attribute), 30  
 \_pupil (param.Param\_dm attribute), 24  
 \_puppixoffset (param.Param\_dm attribute), 24  
 \_pyr\_cx (param.Param\_wfs attribute), 30  
 \_pyr\_cy (param.Param\_wfs attribute), 30  
 \_pyr\_offsets (param.Param\_wfs attribute), 30  
 \_qpixsize (param.Param\_wfs attribute), 30  
 \_subapd (param.Param\_wfs attribute), 30  
 \_submask (param.Param\_wfs attribute), 30  
 \_validsubsx (param.Param\_wfs attribute), 30  
 \_validsubsy (param.Param\_wfs attribute), 30  
 \_xpos (param.Param\_dm attribute), 24  
 \_ypos (param.Param\_dm attribute), 24

## A

add\_centroider() (rtc.Rtc method), 10  
 add\_Controller() (rtc.Rtc method), 10  
 add\_dm() (dms.Dms method), 35  
 add\_layer() (target.Target method), 4  
 add\_screen() (atmos.Atmos method), 3  
 alt (param.Param\_atmos attribute), 18  
 alt (param.Param\_dm attribute), 24  
 apod (param.Param\_target attribute), 27  
 apod (target.Target attribute), 4  
 applycontrol() (rtc.Rtc method), 10  
 Atmos (class in atmos), 3  
 atmos (module), 3  
 atmos\_init() (in module atmos), 3  
 atmos\_seen (param.Param\_wfs attribute), 30  
 atmos\_trace() (target.Target method), 4

## B

beamsize (param.Param\_wfs attribute), 30  
 bin2d() (in module sensors), 8  
 buildcmat() (rtc.Rtc method), 10  
 buildcmatmv() (rtc.Rtc method), 10

## C

cent (param.Param\_geom attribute), 26  
 centroiders (param.Param\_rtc attribute), 27  
 cmat (param.Param\_controller attribute), 21  
 cmat\_init() (in module rtc), 15

cobs (param.Param\_tel attribute), 28  
comp\_dmgeom() (in module dms), 38  
comp\_oneactu() (dms.Dms method), 35  
compute\_KL2V() (in module rtc), 15  
compute\_klbasis() (in module dms), 38  
computeKLbasis() (dms.Dms method), 36  
controllers (param.Param\_rtc attribute), 27  
correct\_dm() (in module rtc), 16  
coupling (param.Param\_dm attribute), 24  
create\_interp\_mat() (in module rtc), 16  
create\_nact\_geom() (in module rtc), 16  
create\_piston\_filter() (in module rtc), 16  
cured\_ndivs (param.Param\_controller attribute), 21

## D

del\_screen() (atmos.Atmos method), 3  
delay (param.Param\_controller attribute), 21  
deltax (param.Param\_atmos attribute), 18  
deltay (param.Param\_atmos attribute), 18  
diam (param.Param\_tel attribute), 28  
dim\_screens (param.Param\_atmos attribute), 18  
disp() (atmos.Atmos method), 3  
dm\_init() (in module dms), 38  
Dms (class in dms), 35  
dms (module), 35  
dms\_seen (param.Param\_target attribute), 27  
dms\_seen (param.Param\_wfs attribute), 31  
dmtrace() (target.Target method), 4  
docentroids() (rtc.Rtc method), 11  
docentroids\_geom() (rtc.Rtc method), 11  
docontrol() (rtc.Rtc method), 11  
docontrol\_geo() (rtc.Rtc method), 11  
doimat() (rtc.Rtc method), 11  
doimat\_geom() (rtc.Rtc method), 11  
doTomoMatrices() (in module rtc), 16

## F

fft\_goodsize() (in module sensors), 8  
frac (param.Param\_atmos attribute), 18  
fracsub (param.Param\_wfs attribute), 31  
fssize (param.Param\_wfs attribute), 31  
fstop (param.Param\_wfs attribute), 31

## G

gain (param.Param\_controller attribute), 21  
geom\_init() (param.Param\_geom method), 26  
get\_amplifoc() (sensors.Sensors method), 6  
get\_amplipup() (target.Target method), 4  
get\_bincube() (sensors.Sensors method), 6  
get\_bining() (sensors.Sensors method), 6  
get\_camplipup() (sensors.Sensors method), 6  
get\_cmat() (rtc.Rtc method), 12  
get\_cmm() (rtc.Rtc method), 12  
get\_cphim() (rtc.Rtc method), 12

get\_dm() (dms.Dms method), 36  
get\_image() (target.Target method), 4  
get\_imat() (rtc.Rtc method), 12  
get\_imgtele() (sensors.Sensors method), 6  
get\_ipupil() (param.Param\_geom method), 26  
get\_KLbasis() (dms.Dms method), 36  
get\_mgain() (rtc.Rtc method), 12  
get\_mpupil() (param.Param\_geom method), 26  
get\_n() (param.Param\_geom method), 26  
get\_n1() (param.Param\_geom method), 26  
get\_n2() (param.Param\_geom method), 26  
get\_offsets() (sensors.Sensors method), 6  
get\_p1() (param.Param\_geom method), 26  
get\_p2() (param.Param\_geom method), 26  
get\_phase() (sensors.Sensors method), 6  
get\_phase() (target.Target method), 4  
get\_phasetele() (target.Target method), 5  
get\_pyrimg() (sensors.Sensors method), 6  
get\_r0() (in module rtc), 16  
get\_rank() (sensors.Sensors method), 6  
get\_screen() (atmos.Atmos method), 3  
get\_slopes() (sensors.Sensors method), 6  
get\_spupil() (param.Param\_geom method), 26  
get\_strehl() (target.Target method), 5  
getCenbuff() (rtc.Rtc method), 11  
getCentroids() (rtc.Rtc method), 11  
getcentroids() (rtc.Rtc method), 12  
getCmmEigenvals() (rtc.Rtc method), 11  
getCom() (rtc.Rtc method), 11  
getComm() (dms.Dms method), 36  
getEigenvals() (rtc.Rtc method), 12  
getErr() (rtc.Rtc method), 12  
getInflu() (dms.Dms method), 36  
getolmeas() (rtc.Rtc method), 13  
getU() (rtc.Rtc method), 12  
getVoltage() (rtc.Rtc method), 12  
gmax (param.Param\_controller attribute), 21  
gmin (param.Param\_controller attribute), 21  
gsalt (param.Param\_wfs attribute), 31  
gsmag (param.Param\_wfs attribute), 31

## H

hyst (param.Param\_dm attribute), 24

## I

imat (param.Param\_controller attribute), 21  
imat\_geom() (in module rtc), 16  
imat\_init() (in module rtc), 17  
imat\_svd() (rtc.Rtc method), 13  
indices() (in module param), 34  
init\_modalOpti() (rtc.Rtc method), 13  
init\_proj() (rtc.Rtc method), 13  
init\_strehlmeter() (target.Target method), 5  
init\_wfs\_geom() (in module sensors), 8



interpmat (param.Param\_centroider attribute), 20  
 ittime (param.Param\_loop attribute), 27

## K

kernel (param.Param\_wfs attribute), 31

## L

L0 (param.Param\_atmos attribute), 18  
 Lambda (param.Param\_target attribute), 27  
 Lambda (param.Param\_wfs attribute), 29  
 Lambda (target.Target attribute), 4  
 laserpower (param.Param\_wfs attribute), 31  
 lgsreturnperwatt (param.Param\_wfs attribute), 31  
 list\_alt() (atmos.Atmos method), 3  
 lltx (param.Param\_wfs attribute), 31  
 llty (param.Param\_wfs attribute), 31  
 load\_kl() (dms.Dms method), 36  
 load\_pzt() (dms.Dms method), 37  
 load\_tt() (dms.Dms method), 37  
 loadnoisemat() (rtc.Rtc method), 13  
 loadOpenLoop() (rtc.Rtc method), 13

## M

mag (param.Param\_target attribute), 27  
 mag (target.Target attribute), 5  
 make\_apodizer() (in module param), 34  
 make\_lgs\_prof1d() (in module sensors), 8  
 makegaussian() (in module param), 34  
 manual\_imat() (in module rtc), 17  
 margin (param.Param\_dm attribute), 24  
 maxcond (param.Param\_controller attribute), 21  
 modalControlOptimization() (rtc.Rtc method), 14  
 modopti (param.Param\_controller attribute), 21  
 move\_atmos() (atmos.Atmos method), 3

## N

nact (param.Param\_dm attribute), 24  
 nactu (param.Param\_controller attribute), 21  
 nbrmissing (param.Param\_tel attribute), 28  
 ndm (param.Param\_controller attribute), 21  
 ngain (param.Param\_controller attribute), 21  
 niter (param.Param\_loop attribute), 27  
 nkl (param.Param\_controller attribute), 21  
 nkl (param.Param\_dm attribute), 24  
 nmax (param.Param\_centroider attribute), 20  
 nmodes (param.Param\_controller attribute), 21  
 noise (param.Param\_wfs attribute), 31  
 noise\_cov() (in module sensors), 8  
 npix (param.Param\_wfs attribute), 31  
 nrec (param.Param\_controller attribute), 22  
 nscreens (param.Param\_atmos attribute), 18  
 ntargets (param.Param\_target attribute), 27  
 ntargets (target.Target attribute), 5

nvalid (param.Param\_controller attribute), 22  
 nwfs (param.Param\_centroider attribute), 20  
 nwfs (param.Param\_controller attribute), 22  
 nwfs (param.Param\_rtc attribute), 27  
 nxsub (param.Param\_wfs attribute), 31

## O

oneactu() (dms.Dms method), 37  
 openloop (param.Param\_wfs attribute), 31  
 openLoopSlp() (in module rtc), 17  
 optthroughput (param.Param\_wfs attribute), 31

## P

param (module), 18  
 Param\_atmos (class in param), 18  
 Param\_centroider (class in param), 20  
 Param\_controller (class in param), 21  
 Param\_dm (class in param), 23  
 Param\_geom (class in param), 25  
 Param\_loop (class in param), 26  
 Param\_rtc (class in param), 27  
 Param\_target (class in param), 27  
 Param\_tel (class in param), 28  
 Param\_wfs (class in param), 29  
 pixsize (param.Param\_wfs attribute), 31  
 prep\_lgs\_prof() (in module sensors), 8  
 proftype (param.Param\_wfs attribute), 31  
 pupangle (param.Param\_tel attribute), 28  
 pupdiam (param.Param\_geom attribute), 26  
 pupixsize (param.Param\_atmos attribute), 19  
 pupoffset (param.Param\_dm attribute), 24  
 push4imat (param.Param\_dm attribute), 24  
 pyr\_ampl (param.Param\_wfs attribute), 31  
 pyr\_loc (param.Param\_wfs attribute), 31  
 pyr\_npts (param.Param\_wfs attribute), 31  
 pyrtype (param.Param\_wfs attribute), 31

## R

r0 (param.Param\_atmos attribute), 19  
 referr (param.Param\_tel attribute), 28  
 remove\_dm() (dms.Dms method), 37  
 resetdm() (dms.Dms method), 37  
 rmcontrol() (rtc.Rtc method), 14  
 rotate() (in module param), 35  
 rotate3d() (in module param), 35  
 Rtc (class in rtc), 10  
 rtc (module), 10  
 rtc\_init() (in module rtc), 17

## S

seeds (param.Param\_atmos attribute), 19  
 selectDMforLayers() (in module rtc), 18  
 Sensors (class in sensors), 5

sensors (module), 5  
sensors\_addlayer() (sensors.Sensors method), 7  
sensors\_compimg() (sensors.Sensors method), 7  
sensors\_compslopes() (rtc.Rtc method), 14  
sensors\_initarr() (sensors.Sensors method), 7  
sensors\_initbcube() (rtc.Rtc method), 14  
sensors\_initcorr() (rtc.Rtc method), 14  
sensors\_initgs() (sensors.Sensors method), 7  
sensors\_initweights() (rtc.Rtc method), 14  
sensors\_trace() (sensors.Sensors method), 7  
set\_alt() (param.Param\_atmos method), 19  
set\_alt() (param.Param\_dm method), 24  
set\_altna() (param.Param\_wfs method), 32  
set\_apod() (param.Param\_target method), 27  
set\_atmos\_seen() (param.Param\_wfs method), 32  
set\_beamsize() (param.Param\_wfs method), 32  
set\_cent() (param.Param\_geom method), 26  
set\_centroiders() (param.Param\_rtc method), 27  
set\_cmat() (param.Param\_controller method), 22  
set\_cmat() (rtc.Rtc method), 14  
set\_cmm() (rtc.Rtc method), 15  
set\_cobs() (param.Param\_tel method), 28  
set\_comm() (dms.Dms method), 37  
set\_controllers() (param.Param\_rtc method), 27  
set\_coupling() (param.Param\_dm method), 24  
set\_cured\_ndivs() (param.Param\_controller method), 22  
set\_decayFactor() (rtc.Rtc method), 15  
set\_delay() (param.Param\_controller method), 22  
set\_deltax() (param.Param\_atmos method), 19  
set\_deltay() (param.Param\_atmos method), 19  
set\_diam() (param.Param\_tel method), 28  
set\_dim\_screens() (param.Param\_atmos method), 19  
set\_dms\_seen() (param.Param\_wfs method), 32  
set\_frac() (param.Param\_atmos method), 19  
set\_fracsub() (param.Param\_wfs method), 32  
set\_fssize() (param.Param\_wfs method), 32  
set\_fstop() (param.Param\_wfs method), 32  
set\_gain() (param.Param\_controller method), 22  
set\_gain() (rtc.Rtc method), 15  
set\_gmax() (param.Param\_controller method), 22  
set\_gmin() (param.Param\_controller method), 22  
set\_gsalt() (param.Param\_wfs method), 32  
set\_gsmag() (param.Param\_wfs method), 32  
set\_i1() (param.Param\_dm method), 24  
set\_imat() (param.Param\_controller method), 22  
set\_imat() (rtc.Rtc method), 15  
set\_influ() (param.Param\_dm method), 24  
set\_ittime() (param.Param\_loop method), 27  
set\_j1() (param.Param\_dm method), 24  
set\_kernel() (param.Param\_wfs method), 32  
set\_L0() (param.Param\_atmos method), 19  
set\_Lambda() (param.Param\_target method), 27  
set\_Lambda() (param.Param\_wfs method), 32  
set\_laserpower() (param.Param\_wfs method), 32  
set\_lgsreturnperwatt() (param.Param\_wfs method), 32  
set\_lltx() (param.Param\_wfs method), 33  
set\_llty() (param.Param\_wfs method), 33  
set\_mag() (param.Param\_target method), 28  
set\_matE() (rtc.Rtc method), 15  
set\_maxcond() (param.Param\_controller method), 22  
set\_mgain() (rtc.Rtc method), 15  
set\_modopti() (param.Param\_controller method), 22  
set\_nact() (param.Param\_dm method), 25  
set\_nactu() (param.Param\_controller method), 22  
set\_nbrmissing() (param.Param\_tel method), 28  
set\_ndm() (param.Param\_controller method), 22  
set\_ngain() (param.Param\_controller method), 23  
set\_niter() (param.Param\_loop method), 27  
set\_nkl() (param.Param\_controller method), 23  
set\_nmax() (param.Param\_centroider method), 20  
set\_nmodes() (param.Param\_controller method), 23  
set\_noise() (param.Param\_wfs method), 33  
set\_npix() (param.Param\_wfs method), 33  
set\_nrec() (param.Param\_controller method), 23  
set\_nscreens() (param.Param\_atmos method), 19  
set\_nTargets() (param.Param\_target method), 28  
set\_ntotact() (param.Param\_dm method), 25  
set\_nvalid() (param.Param\_controller method), 23  
set\_nwfs() (param.Param\_centroider method), 20  
set\_nwfs() (param.Param\_controller method), 23  
set\_nwfs() (param.Param\_rtc method), 27  
set\_nxsub() (param.Param\_wfs method), 33  
set\_openloop() (param.Param\_wfs method), 33  
set\_optthroughput() (param.Param\_wfs method), 33  
set\_pixsize() (param.Param\_wfs method), 33  
set\_profna() (param.Param\_wfs method), 33  
set\_proftype() (param.Param\_wfs method), 33  
set\_pupangle() (param.Param\_tel method), 28  
set\_pupdiam() (param.Param\_geom method), 26  
set\_pupixsize() (param.Param\_atmos method), 19  
set\_push4imat() (param.Param\_dm method), 25  
set\_pyr\_ampl() (param.Param\_wfs method), 33  
set\_pyr\_loc() (param.Param\_wfs method), 33  
set\_pyr\_npts() (param.Param\_wfs method), 33  
set\_pyrtype() (param.Param\_wfs method), 33  
set\_r0() (param.Param\_atmos method), 19  
set\_referr() (param.Param\_tel method), 29  
set\_seeds() (param.Param\_atmos method), 19  
set\_sizeX() (param.Param\_centroider method), 20  
set\_sizeY() (param.Param\_centroider method), 20  
set\_spiders\_type() (param.Param\_tel method), 29  
set\_ssize() (param.Param\_geom method), 26  
set\_t\_spiders() (param.Param\_tel method), 29  
set\_thresh() (param.Param\_centroider method), 20  
set\_thresh() (param.Param\_dm method), 25  
set\_TTcond() (param.Param\_controller method), 22  
set\_type() (param.Param\_centroider method), 20  
set\_type() (param.Param\_controller method), 23

set\_type() (param.Param\_dm method), 25  
 set\_type() (param.Param\_wfs method), 34  
 set\_type\_ap() (param.Param\_tel method), 29  
 set\_type\_fct() (param.Param\_centroider method), 20  
 set\_unitpervolt() (param.Param\_dm method), 25  
 set\_weights() (param.Param\_centroider method), 20  
 set\_width() (param.Param\_centroider method), 20  
 set\_winddir() (param.Param\_atmos method), 19  
 set\_windspeed() (param.Param\_atmos method), 19  
 set\_xpos() (param.Param\_dm method), 25  
 set\_xpos() (param.Param\_target method), 28  
 set\_xpos() (param.Param\_wfs method), 34  
 set\_ypos() (param.Param\_dm method), 25  
 set\_ypos() (param.Param\_target method), 28  
 set\_ypos() (param.Param\_wfs method), 34  
 set\_zenithangle() (param.Param\_geom method), 26  
 set\_zerop() (param.Param\_wfs method), 34  
 setCom() (rtc.Rtc method), 14  
 setEigenvals() (rtc.Rtc method), 14  
 setnmax() (rtc.Rtc method), 15  
 setthresh() (rtc.Rtc method), 15  
 setU() (rtc.Rtc method), 14  
 shape\_dm() (dms.Dms method), 38  
 sizex (param.Param\_centroider attribute), 20  
 sizey (param.Param\_centroider attribute), 21  
 slopes\_geom() (sensors.Sensors method), 7  
 spiders\_type (param.Param\_tel attribute), 29  
 ssize (param.Param\_geom attribute), 26

## T

t\_spiders (param.Param\_tel attribute), 29  
 Target (class in target), 4  
 target (module), 4  
 target\_init() (in module target), 5  
 thresh (param.Param\_centroider attribute), 21  
 thresh (param.Param\_dm attribute), 25  
 TTcond (param.Param\_controller attribute), 21  
 type\_ap (param.Param\_tel attribute), 29  
 type\_centro (param.Param\_centroider attribute), 21  
 type\_control (param.Param\_controller attribute), 23  
 type\_dm (param.Param\_dm attribute), 25  
 type\_fct (param.Param\_centroider attribute), 21  
 type\_present() (in module sensors), 9  
 type\_wfs (param.Param\_wfs attribute), 34

## U

unitpervolt (param.Param\_dm attribute), 25

## W

weights (param.Param\_centroider attribute), 21  
 wfs\_init() (in module sensors), 9  
 wheremax() (in module sensors), 9  
 width (param.Param\_centroider attribute), 21  
 winddir (param.Param\_atmos attribute), 20

windspeed (param.Param\_atmos attribute), 20

## X

xpos (param.Param\_target attribute), 28  
 xpos (param.Param\_wfs attribute), 34  
 xpos (target.Target attribute), 5

## Y

ypos (param.Param\_target attribute), 28  
 ypos (param.Param\_wfs attribute), 34  
 ypos (target.Target attribute), 5

## Z

zenithangle (param.Param\_geom attribute), 26  
 zerop (param.Param\_target attribute), 28  
 zerop (param.Param\_wfs attribute), 34