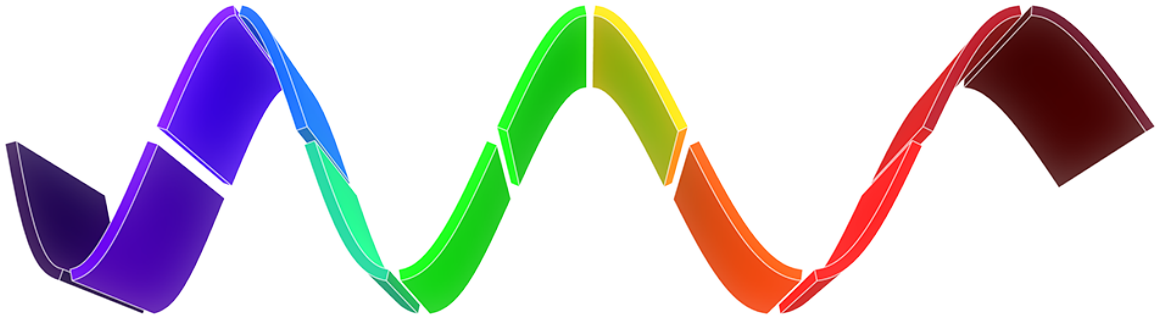




Colour - HDRI Documentation

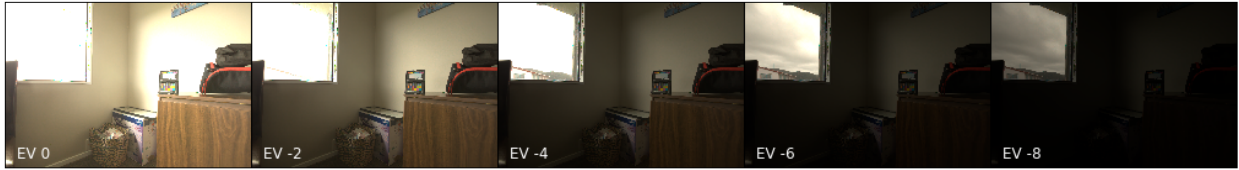
Release 0.1.4

Colour Developers

Feb 19, 2018

CONTENTS

1	Features	3
2	Installation	5
2.1	Primary Dependencies	5
2.2	Optional Features Dependencies	5
2.3	Pypi	5
3	Usage	7
3.1	API	7
3.1.1	Colour - HDRI Manual	7
3.1.1.1	Reference	7
3.1.1.2	Bibliography	50
3.2	Examples	50
4	Contributing	51
5	Bibliography	53
6	About	55
	Bibliography	57
	Index	59



A [Python](#) package implementing various HDRI / Radiance image processing algorithms.

It is open source and freely available under the [New BSD License](#) terms.

FEATURES

The following features are available:

- HDRI / Radiance Image Generation
- Debevec (1997) Camera Response Function Computation
- Grossberg (2003) Histogram Based Image Sampling
- Variance Minimization Light Probe Sampling
- Global Tonemapping Operators
- Adobe DNG SDK Colour Processing
- Absolute Luminance Calibration
- Raw Processing Helpers

INSTALLATION

Because of their size, the resources dependencies needed to run the various examples and unit tests are not provided within the Pypi package. They are separately available as [Git Submodules](#) when cloning the [repository](#).

2.1 Primary Dependencies

Colour - HDRI requires various dependencies in order to run:

- [Python 2.7](#) or [Python 3.5](#)
- [NumPy](#)
- [OpenImageIO](#)

2.2 Optional Features Dependencies

- [colour-demosaiicing](#)
- [Adobe DNG Converter](#)
- [dcraw](#)
- [ExifTool](#)
- [rawpy](#)

2.3 Pypi

Once the dependencies satisfied, **Colour - HDRI** can be installed from the [Python Package Index](#) by issuing this command in a shell:

```
pip install colour-hdri
```

The optional features dependencies are installed as follows:

```
pip install 'colour-hdri[optional]'
```

The figures plotting dependencies are installed as follows:

```
pip install 'colour-hdri[plotting]'
```

The tests suite dependencies are installed as follows:

```
pip install 'colour-hdri[tests]'
```

The documentation building dependencies are installed as follows:

```
pip install 'colour-hdri[docs]'
```

3.1 API

The main reference for [Colour - HDRI](#) is the manual:

3.1.1 Colour - HDRI Manual

3.1.1.1 Reference

Colour - HDRI

Camera Calibration

- *Absolute Luminance - Lagarde (2016)*
- *Debevec (1997)*

Absolute Luminance - Lagarde (2016)

colour_hdri

<code>absolute_luminance_calibration_Lagarde2016(...)</code>	Performs absolute <i>Luminance</i> calibration of given <i>RGB</i> panoramic image using <i>Lagarde (2016)</i> method.
<code>upper_hemisphere_illuminance_weights_Lagarde2016(...)</code>	Computes upper hemisphere illuminance weights for use with applications unable to perform the computation directly, i.e.

`colour_hdri.absolute_luminance_calibration_Lagarde2016`

```
colour_hdri.absolute_luminance_calibration_Lagarde2016(RGB, measured_illuminance,
colourspace=RGB_Colourspace(sRGB[[[
0.64, 0.33][[ 0.3, 0.6][[ 0.15,
0.06]]][[ 0.3127, 0.329][[ 0.2126,
0.4124, 0.3576, 0.1805][[ 0.0193, 0.1192,
0.7152, 0.0722][[ 0.0193, 0.1192,
0.9505]]][[ 3.2406, -1.5372, -
0.4986][[ -0.9689, 1.8758, 0.0415
][[ 0.0557, -0.204, 1.057]]],
<function oetf_sRGB>, <function
oetf_reverse_sRGB>, False, False))
```

Performs absolute *Luminance* calibration of given *RGB* panoramic image using *Lagarde (2016)* method.

Parameters

- **RGB** (array_like) – *RGB* panoramic image to calibrate.
- **measured_illuminance** (numeric) – Measured illuminance E_v .
- **colourspace** (*colour.RGB_Colourspace*, optional) – *RGB* colourspace used for internal *Luminance* computation.

Returns Absolute *Luminance* calibrated *RGB* panoramic image.

Return type ndarray

Examples

```
>>> RGB = np.ones((4, 8, 3))
>>> absolute_luminance_calibration_Lagarde2016(
...     RGB, 500)
array([[[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...]],

[[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...]],

[[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...]]])
```

```
[ 233.9912506..., 233.9912506..., 233.9912506...],
[ 233.9912506..., 233.9912506..., 233.9912506...]],

[[ 233.9912506..., 233.9912506..., 233.9912506...],
 [ 233.9912506..., 233.9912506..., 233.9912506...],
 [ 233.9912506..., 233.9912506..., 233.9912506...],
 [ 233.9912506..., 233.9912506..., 233.9912506...],
 [ 233.9912506..., 233.9912506..., 233.9912506...],
 [ 233.9912506..., 233.9912506..., 233.9912506...],
 [ 233.9912506..., 233.9912506..., 233.9912506...],
 [ 233.9912506..., 233.9912506..., 233.9912506...]]])
```

colour_hdri.upper_hemisphere_illuminance_weights_Lagarde2016

colour_hdri.upper_hemisphere_illuminance_weights_Lagarde2016(*height*, *width*)

Computes upper hemisphere illuminance weights for use with applications unable to perform the computation directly, i.e. *Adobe Photoshop*.

Parameters

- **height** (*int*) – Output array height.
- **width** (*int*) – Output array width.

Returns Upper hemisphere illuminance weights.

Return type ndarray

References

- [\[LLJ16\]](#)

Examples

```
>>> upper_hemisphere_illuminance_weights_Lagarde2016(
...     16, 1)
array([[ 0.00000000e+00,
         4.01432970e-05,
         7.33454540e-05,
         9.38655150e-05,
         9.81553760e-05,
         8.54732810e-05,
         5.80120790e-05,
         2.05200610e-05,
         0.00000000e+00,
         0.00000000e+00,
         0.00000000e+00,
         0.00000000e+00,
         0.00000000e+00,
         0.00000000e+00,
         0.00000000e+00,
         0.00000000e+00])
```

Debevec (1997)

colour_hdri

<code>g_solve(Z, B[, l_s, w, n])</code>	Given a set of pixel values observed for several pixels in several images with different exposure times, this function returns the imaging system's response function g as well as the log film irradiance values lE for the observed pixels.
<code>camera_response_functions_Debevec1997(...[, ...])</code>	Returns the camera response functions for given image stack using <i>Debevec (1997)</i> method.

colour_hdri.g_solve

`colour_hdri.g_solve(Z, B, l_s=30, w=<function weighting_function_Debevec1997>, n=256)`

Given a set of pixel values observed for several pixels in several images with different exposure times, this function returns the imaging system's response function g as well as the log film irradiance values lE for the observed pixels.

Parameters

- **Z** (array_like) – Set of pixel values observed for several pixels in several images.
- **B** (array_like) – Log Δt , or log shutter speed for images.
- **l_s** (numeric, optional) – λ smoothing term.
- **w** (callable, optional) – Weighting function w .
- **n** (int, optional) – n constant.

Returns Camera response functions $g(z)$ and log film irradiance values lE .

Return type tuple

References

- [DM97]

colour_hdri.camera_response_functions_Debevec1997

`colour_hdri.camera_response_functions_Debevec1997(image_stack, s=<function samples_Grossberg2003>, samples=1000, l_s=30, w=<function weighting_function_Debevec1997>, n=256, normalise=True)`

Returns the camera response functions for given image stack using *Debevec (1997)* method.

Image channels are sampled with s sampling function and the output samples are passed to `colour_hdri.g_solve()`.

Parameters

- **image_stack** (colour_hdri.ImageStack) – Stack of single channel or multi-channel floating point images.

- **s** (*callable*, optional) – Sampling function s .
- **samples** (*int*, optional) – Samples count per images.
- **l_s** (*numeric*, optional) – λ smoothing term.
- **w** (*callable*, optional) – Weighting function w .
- **n** (*int*, optional) – n constant.
- **normalise** (*bool*, optional) – Enables the camera response functions normalisation. Uncertain camera response functions values resulting from w function are set to zero.

Returns Camera response functions $g(z)$.

Return type ndarray

References

- [\[DM97\]](#)

HDRI / Radiance Image Generation

- [Generation](#)
- [Weighting Functions](#)

Generation

colour_hdri

<code>image_stack_to_radiance_image(image_stack[, ...])</code>	Generates a HDRI / radiance image from given image stack.
--	---

colour_hdri.image_stack_to_radiance_image

```
colour_hdri.image_stack_to_radiance_image(image_stack,
                                           weighting_function=<function
                                           weighting_function_Debevec1997>,
                                           weighting_
                                           ing_average=False, camera_response_functions=None)
```

Generates a HDRI / radiance image from given image stack.

Parameters

- **image_stack** (`colour_hdri.ImageStack`) – Stack of single channel or multi-channel floating point images. The stack is assumed to be representing linear values except if `camera_response_functions` argument is provided.
- **weighting_function** (*callable*, optional) – Weighting function w .
- **weighting_average** (*bool*, optional) – Enables weighting function w computation on channels average instead of on a per channel basis.
- **camera_response_functions** (*array_like*, optional) – Camera response functions

$g(z)$ of the imaging system / camera if the stack is representing non linear values.

Returns Radiance image.

Return type ndarray

Warning: If the image stack contains images with negative or equal to zero values, unpredictable results may occur and NaNs might be generated. It is thus recommended to encode the images in a wider RGB colourspace or clamp negative values.

References

- [\[BADC11a\]](#)

Weighting Functions

colour_hdri

<code>normal_distribution_function(a[, mu, sigma])</code>	Returns given array weighted by a normal distribution function.
<code>hat_function(a)</code>	Returns given array weighted by a hat function.
<code>weighting_function_Debevec1997(a[, ...])</code>	Returns given array weighted by <i>Debevec (1997)</i> function.

colour_hdri.normal_distribution_function

`colour_hdri.normal_distribution_function(a, mu=0.5, sigma=0.15)`

Returns given array weighted by a normal distribution function.

Parameters

- **a** (array_like) – Array to apply the weighting function onto.
- **mu** (numeric, optional) – Mean or expectation.
- **sigma** (numeric, optional) – Standard deviation.

Returns Weighted array.

Return type ndarray

Examples

```
>>> normal_distribution_function(np.linspace(0, 1, 10))
array([ 0.00386592,  0.03470859,  0.18002174,  0.53940751,  0.93371212,
        0.93371212,  0.53940751,  0.18002174,  0.03470859,  0.00386592])
```


colour_hdri.hat_function

colour_hdri.hat_function(*a*)

Returns given array weighted by a hat function.

Parameters *a* (array_like) – Array to apply the weighting function onto.

Returns Weighted array.

Return type ndarray

Examples

```
>>> hat_function(np.linspace(0, 1, 10))
array([ 0.          ,  0.95099207,  0.99913557,  0.99999812,  1.          ,
        1.          ,  0.99999812,  0.99913557,  0.95099207,  0.          ])
```

colour_hdri.weighting_function_Debevec1997

colour_hdri.weighting_function_Debevec1997(*a*, *domain_l*=0.01, *domain_h*=0.99)

Returns given array weighted by *Debevec (1997)* function.

Parameters

- *a* (array_like) – Array to apply the weighting function onto.
- *domain_l* (numeric, optional) – Domain lowest possible value, values less than *domain_l* will be set to zero.
- *domain_h* (numeric, optional) – Domain highest possible value, values greater than *domain_h* will be set to zero.

Returns Weighted array.

Return type ndarray

References

- [\[DM97\]](#)

Examples

```
>>> weighting_function_Debevec1997(np.linspace(0, 1, 10))
array([ 0.          ,  0.23273657,  0.48849105,  0.74424552,  1.          ,
        1.          ,  0.74424552,  0.48849105,  0.23273657,  0.          ])
```

Colour Models

- [Adobe DNG SDK](#)

- *RGB Models*

Adobe DNG SDK

colour_hdri

<code>xy_to_camera_neutral(xy, ...)</code>	Converts given <i>xy</i> white balance chromaticity coordinates to <i>Camera Neutral</i> coordinates.
<code>camera_neutral_to_xy(camera_neutral, ...[, ...])</code>	Converts given <i>Camera Neutral</i> coordinates to <i>xy</i> white balance chromaticity coordinates.
<code>XYZ_to_camera_space_matrix(xy, ...)</code>	Returns the <i>CIE XYZ</i> to <i>Camera Space</i> matrix for given <i>xy</i> white balance chromaticity coordinates.
<code>camera_space_to_XYZ_matrix(xy, ...[, ...])</code>	Returns the <i>Camera Space</i> to <i>CIE XYZ</i> matrix for given <i>xy</i> white balance chromaticity coordinates.

colour_hdri.xy_to_camera_neutral

`colour_hdri.xy_to_camera_neutral(xy, CCT_calibration_illuminant_1, CCT_calibration_illuminant_2, M_color_matrix_1, M_color_matrix_2, M_camera_calibration_1, M_camera_calibration_2, analog_balance)`

Converts given *xy* white balance chromaticity coordinates to *Camera Neutral* coordinates.

Parameters

- **xy** (array_like) – *xy* white balance chromaticity coordinates.
- **CCT_calibration_illuminant_1** (numeric) – Correlated colour temperature of *CalibrationIlluminant1*.
- **CCT_calibration_illuminant_2** (numeric) – Correlated colour temperature of *CalibrationIlluminant2*.
- **M_color_matrix_1** (array_like) – *ColorMatrix1* tag matrix.
- **M_color_matrix_2** (array_like) – *ColorMatrix2* tag matrix.
- **M_camera_calibration_1** (array_like) – *CameraCalibration1* tag matrix.
- **M_camera_calibration_2** (array_like) – *CameraCalibration2* tag matrix.
- **analog_balance** (array_like) – *AnalogBalance* tag vector.

Returns *Camera Neutral* coordinates.

Return type ndarray

References

- [\[AdobeSystems12d\]](#)
- [\[AdobeSystems12b\]](#)
- [\[AdobeSystems15c\]](#)
- [\[McG12\]](#)

Examples

```
>>> M_color_matrix_1 = np.array(
...     [[0.5309, -0.0229, -0.0336],
...       [-0.6241, 1.3265, 0.3337],
...       [-0.0817, 0.1215, 0.6664]])
>>> M_color_matrix_2 = np.array(
...     [[0.4716, 0.0603, -0.0830],
...       [-0.7798, 1.5474, 0.2480],
...       [-0.1496, 0.1937, 0.6651]])
>>> M_camera_calibration_1 = np.identity(3)
>>> M_camera_calibration_2 = np.identity(3)
>>> analog_balance = np.ones(3)
>>> xy_to_camera_neutral(
...     np.array([0.32816244, 0.34698169]),
...     2850,
...     6500,
...     M_color_matrix_1,
...     M_color_matrix_2,
...     M_camera_calibration_1,
...     M_camera_calibration_2,
...     analog_balance)
array([ 0.4130699..., 1..., 0.646465...])
```

colour_hdri.camera_neutral_to_xy

`colour_hdri.camera_neutral_to_xy(camera_neutral, CCT_calibration_illuminant_1, CCT_calibration_illuminant_2, M_color_matrix_1, M_color_matrix_2, M_camera_calibration_1, M_camera_calibration_2, analog_balance, epsilon=2.2204460492503131e-16)`

Converts given *Camera Neutral* coordinates to *xy* white balance chromaticity coordinates.

Parameters

- **camera_neutral** (array_like) – *Camera Neutral* coordinates.
- **CCT_calibration_illuminant_1** (numeric) – Correlated colour temperature of *CalibrationIlluminant1*.
- **CCT_calibration_illuminant_2** (numeric) – Correlated colour temperature of *CalibrationIlluminant2*.
- **M_color_matrix_1** (array_like) – *ColorMatrix1* tag matrix.
- **M_color_matrix_2** (array_like) – *ColorMatrix2* tag matrix.
- **M_camera_calibration_1** (array_like) – *CameraCalibration1* tag matrix.
- **M_camera_calibration_2** (array_like) – *CameraCalibration2* tag matrix.
- **analog_balance** (array_like) – *AnalogBalance* tag vector.
- **epsilon** (numeric, optional) – Threshold value for computation convergence.

Returns *xy* white balance chromaticity coordinates.

Return type ndarray

Raises `RuntimeError` – If the given *Camera Neutral* coordinates did not converge to *xy* white balance chromaticity coordinates.

References

- [\[AdobeSystems12c\]](#)
- [\[AdobeSystems12b\]](#)
- [\[AdobeSystems15c\]](#)
- [\[McG12\]](#)

Examples

```
>>> M_color_matrix_1 = np.array(
...     [[0.5309, -0.0229, -0.0336],
...      [-0.6241, 1.3265, 0.3337],
...      [-0.0817, 0.1215, 0.6664]])
>>> M_color_matrix_2 = np.array(
...     [[0.4716, 0.0603, -0.0830],
...      [-0.7798, 1.5474, 0.2480],
...      [-0.1496, 0.1937, 0.6651]])
>>> M_camera_calibration_1 = np.identity(3)
>>> M_camera_calibration_2 = np.identity(3)
>>> analog_balance = np.ones(3)
>>> camera_neutral_to_xy(
...     np.array([0.413070, 1.000000, 0.646465]),
...     2850,
...     6500,
...     M_color_matrix_1,
...     M_color_matrix_2,
...     M_camera_calibration_1,
...     M_camera_calibration_2,
...     analog_balance)
array([ 0.3281624...,  0.3469816...])
```

colour_hdri.XYZ_to_camera_space_matrix

`colour_hdri.XYZ_to_camera_space_matrix(xy, CCT_calibration_illuminant_1, CCT_calibration_illuminant_2, M_color_matrix_1, M_color_matrix_2, M_camera_calibration_1, M_camera_calibration_2, analog_balance)`

Returns the *CIE XYZ* to *Camera Space* matrix for given *xy* white balance chromaticity coordinates.

Parameters

- **xy** (array_like) – *xy* white balance chromaticity coordinates.
- **CCT_calibration_illuminant_1** (numeric) – Correlated colour temperature of *CalibrationIlluminant1*.
- **CCT_calibration_illuminant_2** (numeric) – Correlated colour temperature of *CalibrationIlluminant2*.
- **M_color_matrix_1** (array_like) – *ColorMatrix1* tag matrix.

- **M_color_matrix_2** (array_like) – *ColorMatrix2* tag matrix.
- **M_camera_calibration_1** (array_like) – *CameraCalibration1* tag matrix.
- **M_camera_calibration_2** (array_like) – *CameraCalibration2* tag matrix.
- **analog_balance** (array_like) – *AnalogBalance* tag vector.

Returns *CIE XYZ* to *Camera Space* matrix.

Return type ndarray

Notes

- The reference illuminant is D50 as defined per `colour_hdri.models.dataset.dng.ADOBE_DNG_XYZ_ILLUMINANT` attribute.

References

- [\[AdobeSystems12b\]](#)
- [\[AdobeSystems15c\]](#)
- [\[McG12\]](#)

Examples

```
>>> M_color_matrix_1 = np.array(
...     [[0.5309, -0.0229, -0.0336],
...      [-0.6241, 1.3265, 0.3337],
...      [-0.0817, 0.1215, 0.6664]])
>>> M_color_matrix_2 = np.array(
...     [[0.4716, 0.0603, -0.0830],
...      [-0.7798, 1.5474, 0.2480],
...      [-0.1496, 0.1937, 0.6651]])
>>> M_camera_calibration_1 = np.identity(3)
>>> M_camera_calibration_2 = np.identity(3)
>>> analog_balance = np.ones(3)
>>> XYZ_to_camera_space_matrix(
...     np.array([0.34510414, 0.35162252]),
...     2850,
...     6500,
...     M_color_matrix_1,
...     M_color_matrix_2,
...     M_camera_calibration_1,
...     M_camera_calibration_2,
...     analog_balance)
array([[ 0.4854908...,  0.0408106..., -0.0714282...],
       [-0.7433278...,  1.4956549...,  0.2680749...],
       [-0.1336946...,  0.1767874...,  0.6654045...]])
```

colour_hdri.camera_space_to_XYZ_matrix

```
colour_hdri.camera_space_to_XYZ_matrix(xy, CCT_calibration_illuminant_1,
                                       CCT_calibration_illuminant_2, M_color_matrix_1,
                                       M_color_matrix_2, M_camera_calibration_1,
                                       M_camera_calibration_2, analog_balance,
                                       M_forward_matrix_1, M_forward_matrix_2, chromatic_adaptation_transform=u'Bradford')
```

Returns the *Camera Space* to *CIE XYZ* matrix for given *xy* white balance chromaticity coordinates.

Parameters

- **xy** (array_like) – *xy* white balance chromaticity coordinates.
- **CCT_calibration_illuminant_1** (numeric) – Correlated colour temperature of *CalibrationIlluminant1*.
- **CCT_calibration_illuminant_2** (numeric) – Correlated colour temperature of *CalibrationIlluminant2*.
- **M_color_matrix_1** (array_like) – *ColorMatrix1* tag matrix.
- **M_color_matrix_2** (array_like) – *ColorMatrix2* tag matrix.
- **M_camera_calibration_1** (array_like) – *CameraCalibration1* tag matrix.
- **M_camera_calibration_2** (array_like) – *CameraCalibration2* tag matrix.
- **analog_balance** (array_like) – *AnalogBalance* tag vector.
- **M_forward_matrix_1** (array_like) – *ForwardMatrix1* tag matrix.
- **M_forward_matrix_2** (array_like) – *ForwardMatrix2* tag matrix.
- **chromatic_adaptation_transform** (unicode, optional) – {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}, Chromatic adaptation transform.

Returns *Camera Space* to *CIE XYZ* matrix.

Return type ndarray

Notes

- The reference illuminant is D50 as defined per `colour_hdri.models.dataset.dng.ADOBE_DNG_XYZ_ILLUMINANT` attribute.

References

- [\[AdobeSystems12b\]](#)
- [\[AdobeSystems12a\]](#)
- [\[AdobeSystems15c\]](#)
- [\[McG12\]](#)

Examples

```
>>> M_color_matrix_1 = np.array(
...     [[0.5309, -0.0229, -0.0336],
...       [-0.6241, 1.3265, 0.3337],
...       [-0.0817, 0.1215, 0.6664]])
>>> M_color_matrix_2 = np.array(
...     [[0.4716, 0.0603, -0.0830],
...       [-0.7798, 1.5474, 0.2480],
...       [-0.1496, 0.1937, 0.6651]])
>>> M_camera_calibration_1 = np.identity(3)
>>> M_camera_calibration_2 = np.identity(3)
>>> analog_balance = np.ones(3)
>>> M_forward_matrix_1 = np.array(
...     [[0.8924, -0.1041, 0.1760],
...       [0.4351, 0.6621, -0.0972],
...       [0.0505, -0.1562, 0.9308]])
>>> M_forward_matrix_2 = np.array(
...     [[0.8924, -0.1041, 0.1760],
...       [0.4351, 0.6621, -0.0972],
...       [0.0505, -0.1562, 0.9308]])
>>> camera_space_to_XYZ_matrix(
...     np.array([0.32816244, 0.34698169]),
...     2850,
...     6500,
...     M_color_matrix_1,
...     M_color_matrix_2,
...     M_camera_calibration_1,
...     M_camera_calibration_2,
...     analog_balance,
...     M_forward_matrix_1,
...     M_forward_matrix_2)
array([[ 2.1604087..., -0.1041...,  0.2722498...],
       [ 1.0533324...,  0.6621..., -0.1503561...],
       [ 0.1222553..., -0.1562...,  1.4398304...]])
```

RGB Models

colour_hdri

<code>camera_space_to_RGB(</code> RGB, ...)	Converts given <i>RGB</i> array from <i>camera space</i> to given <i>RGB</i> colourspace.
<code>camera_space_to_sRGB(</code> RGB, <code>M_XYZ_to_camera_space)</code>	Converts given <i>RGB</i> array from <i>camera space</i> to <i>sRGB</i> colourspace.

colour_hdri.camera_space_to_RGB

colour_hdri.**camera_space_to_RGB**(*RGB*, *M_XYZ_to_camera_space*, *RGB_to_XYZ_matrix*)
Converts given *RGB* array from *camera space* to given *RGB* colourspace.

Parameters

- **RGB** (array_like) – Camera space *RGB* colourspace array.

- **XYZ_to_camera_matrix** (array_like) – Matrix converting from *CIE XYZ* tristimulus values to *camera space*.
- **RGB_to_XYZ_matrix** (array_like) – Matrix converting from *RGB* colourspace to *CIE XYZ* tristimulus values.

Returns *RGB* colourspace array.

Return type ndarray

Examples

```
>>> RGB = np.array([0.80660, 0.81638, 0.65885])
>>> M_XYZ_to_camera_space = np.array([
...     [0.47160000, 0.06030000, -0.08300000],
...     [-0.77980000, 1.54740000, 0.24800000],
...     [-0.14960000, 0.19370000, 0.66510000]])
>>> RGB_to_XYZ_matrix = np.array([
...     [0.41238656, 0.35759149, 0.18045049],
...     [0.21263682, 0.71518298, 0.07218020],
...     [0.01933062, 0.11919716, 0.95037259]])
>>> camera_space_to_RGB(
...     RGB,
...     M_XYZ_to_camera_space,
...     RGB_to_XYZ_matrix)
array([ 0.7564180..., 0.8683192..., 0.6044589...])
```

colour_hdri.camera_space_to_sRGB

colour_hdri.camera_space_to_sRGB(*RGB*, *M_XYZ_to_camera_space*)
 Converts given *RGB* array from *camera space* to *sRGB* colourspace.

Parameters

- **RGB** (array_like) – Camera space *RGB* colourspace array.
- **M_XYZ_to_camera_space** (array_like) – Matrix converting from *CIE XYZ* tristimulus values to *camera space*.

Returns *sRGB* colourspace array.

Return type ndarray

Examples

```
>>> RGB = np.array([0.80660, 0.81638, 0.65885])
>>> M_XYZ_to_camera_space = np.array([
...     [0.47160000, 0.06030000, -0.08300000],
...     [-0.77980000, 1.54740000, 0.24800000],
...     [-0.14960000, 0.19370000, 0.66510000]])
>>> camera_space_to_sRGB(RGB, M_XYZ_to_camera_space)
array([ 0.7564350..., 0.8683155..., 0.6044706...])
```


Plotting

- [HDRI / Radiance Image](#)
- [Tonemapping Operators](#)

HDRI / Radiance Image

colour_hdri.plotting

<code>radiance_image_strip_plot(image[, count, ...])</code>	Plots given HDRI / radiance image as strip of images of varying exposure.
---	---

colour_hdri.plotting.radiance_image_strip_plot

colour_hdri.plotting.**radiance_image_strip_plot**(image, count=5, ev_steps=-2, encoding_cctf=<function oetf_sRGB>, **kwargs)

Plots given HDRI / radiance image as strip of images of varying exposure.

Parameters

- **image** (array_like) – HDRI / radiance image to plot.
- **count** (int, optional) – Strip images count.
- **ev_steps** (numeric, optional) – Exposure variation for each image of the strip.
- **encoding_cctf** (callable, optional) – Encoding colour component transfer function / opto-electronic transfer function used for plotting.

Other Parameters ****kwargs** (dict, optional) – {colour.plotting.display()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Tonemapping Operators

colour_hdri.plotting

<code>tonemapping_operator_image_plot(image, ...)</code>	Plots given tonemapped image with superimposed luminance mapping function.
--	--

colour_hdri.plotting.tonemapping_operator_image_plot

colour_hdri.plotting.**tonemapping_operator_image_plot**(image, luminance_function, log_scale=False, encoding_cctf=<function oetf_sRGB>, **kwargs)

Plots given tonemapped image with superimposed luminance mapping function.

Parameters

- **image** (array_like) – Tonemapped image to plot.
- **luminance_function** (callable) – Luminance mapping function.
- **log_scale** (bool, optional) – Use a log scale for plotting the luminance mapping function.
- **encoding_cctf** (callable, optional) – Encoding colour component transfer function / opto-electronic transfer function used for plotting.

Other Parameters ****kwargs** (dict, optional) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Definition success.

Return type bool

Image Processing

- *Adobe DNG SDK*
 - *Raw Files*
 - *DNG Files*

Adobe DNG SDK

Raw Files

colour_hdri

<code>convert_raw_files_to_dng_files(raw_files, ...)</code>	Converts given raw files to <i>dng</i> files using given output directory.
<code>RAW_CONVERTER</code>	Command line raw conversion application, usually Dave Coffin's <i>dcraw</i> .
<code>RAW_CONVERSION_ARGUMENTS</code>	Arguments for the command line raw conversion application for non demosaiced linear <i>tiff</i> file format output.
<code>RAW_D_CONVERSION_ARGUMENTS</code>	Arguments for the command line raw conversion application for demosaiced linear <i>tiff</i> file format output.

colour_hdri.convert_raw_files_to_dng_files

`colour_hdri.convert_raw_files_to_dng_files(raw_files, output_directory)`

Converts given raw files to *dng* files using given output directory.

Parameters

- **raw_files** (array_like) – Raw files to convert to *dng* files.
- **output_directory** (unicode) – Output directory.

Returns *dng* files.

Return type `list`

`colour_hdri.RAW_CONVERTER`

`colour_hdri.RAW_CONVERTER = u'dcraw'`

Command line raw conversion application, usually Dave Coffin's *dcraw*.

`RAW_CONVERTER` : unicode

`colour_hdri.RAW_CONVERSION_ARGUMENTS`

`colour_hdri.RAW_CONVERSION_ARGUMENTS = u'-t 0 -D -W -4 -T "{0}"'`

Arguments for the command line raw conversion application for non demosaiced linear *tiff* file format output.

`RAW_CONVERSION_ARGUMENTS` : unicode

`colour_hdri.RAW_D_CONVERSION_ARGUMENTS`

`colour_hdri.RAW_D_CONVERSION_ARGUMENTS = u'-t 0 -H 1 -r 1 1 1 1 -4 -q 3 -o 0 -T "{0}"'`

Arguments for the command line raw conversion application for demosaiced linear *tiff* file format output.

`RAW_D_CONVERSION_ARGUMENTS` : unicode

DNG Files

`colour_hdri`

<code>convert_dng_files_to_intermediate_files(...)</code>	Converts given <i>dng</i> files to intermediate <i>tiff</i> files using given output directory.
<code>DNG_CONVERTER</code>	Command line <i>DNG</i> conversion application, usually <i>Adobe DNG Converter</i> .
<code>DNG_CONVERSION_ARGUMENTS</code>	Arguments for the command line <i>dng</i> conversion application.
<code>DNG_EXIF_TAGS_BINDING</code>	Exif tags binding for a <i>dng</i> file.
<code>read_dng_files_exif_tags(dng_files[, ...])</code>	Reads given <i>dng</i> files exif tags using given binding.

`colour_hdri.convert_dng_files_to_intermediate_files`

`colour_hdri.convert_dng_files_to_intermediate_files(dng_files, output_directory, demosaicing=False)`

Converts given *dng* files to intermediate *tiff* files using given output directory.

Parameters

- `dng_files` (`array_like`) – *dng* files to convert to intermediate *tiff* files.
- `output_directory` (`str`) – Output directory.

- **demosaicing** (`bool`) – Perform demosaicing on conversion.

Returns Intermediate *tiff* files.

Return type `list`

`colour_hdri.DNG_CONVERTER`

`colour_hdri.DNG_CONVERTER = u'/Applications/Adobe DNG Converter.app/Contents/MacOS/Adobe DNG Converter'`
Command line *DNG* conversion application, usually *Adobe DNG Converter*.

`DNG_CONVERTER` : unicode

`colour_hdri.DNG_CONVERSION_ARGUMENTS`

`colour_hdri.DNG_CONVERSION_ARGUMENTS = u'-l -d "{0}" "{1}"'`
Arguments for the command line *dng* conversion application.

`DNG_CONVERSION_ARGUMENTS` : unicode

`colour_hdri.DNG_EXIF_TAGS_BINDING`

`colour_hdri.DNG_EXIF_TAGS_BINDING = CaseInsensitiveMapping({u'EXIF': CaseInsensitiveMapping({u'Active Area':`
Exif tags binding for a *dng* file.

`DNG_EXIF_TAGS_BINDING` : `CaseInsensitiveMapping`

colour_hdri.read_dng_files_exif_tags

```
colour_hdri.read_dng_files_exif_tags(dng_files, exif_tags_binding=CaseInsensitiveMapping({u'EXIF':
    CaseInsensitiveMapping({u'Active Area': (<function
    <lambda>>, None), u'Make': (<function
    parse_exif_string>, None), u'Baseline Exposure': (<function
    <lambda>>, None), u'Profile Calibration Sig':
    (<function parse_exif_string>, None), u'ISO': (<function
    parse_exif_numeric>, None), u'White Level': (<function
    <lambda>>, None), u'Camera Calibration 1': (<function
    <lambda>>, u'1 0 0 0 1 0 0 0 1'), u'F Number': (<function
    parse_exif_numeric>, None), u'Camera Calibration 2':
    (<function <lambda>>, u'1 0 0 0 1 0 0 0 1'), u'Black
    Level': (<function <lambda>>, None), u'Samples Per Pixel':
    (<function <lambda>>, None), u'Lens Model': (<function
    parse_exif_string>, None), u'DNG Lens Info': (<function
    parse_exif_string>, None), u'Forward Matrix 2': (<function
    <lambda>>, u'1 0 0 0 1 0 0 0 1'), u'Forward Matrix 1':
    (<function <lambda>>, u'1 0 0 0 1 0 0 0 1'), u'CFA
    Pattern 2': (<function <lambda>>, None), u'Black Level
    Repeat Dim': (<function <lambda>>, None), u'Baseline
    Noise': (<function <lambda>>, None), u'CFA Plane Color':
    (<function <lambda>>, None), u'Focal Length': (<function
    parse_exif_numeric>, None), u'As Shot Neutral': (<function
    <lambda>>, u'1 1 1'), u'Camera Calibration Sig':
    (<function parse_exif_string>, None), u'Camera Serial Num-
    ber': (<function parse_exif_string>, None), u'Orientation':
    (<function <lambda>>, None), u'Exposure Time': (<function
    parse_exif_numeric>, None), u'Calibration Illuminant
    1': (<function <lambda>>, 17), u'Calibration Illuminant
    2': (<function <lambda>>, 21), u'Reduction Matrix 1':
    (<function <lambda>>, u'1 0 0 0 1 0 0 0 1'), u'Reduction
    Matrix 2': (<function <lambda>>, u'1 0 0 0 1 0 0 0 1'),
    u'Camera Model Name': (<function parse_exif_string>,
    None), u'Color Matrix 2': (<function <lambda>>, u'1 0 0 0
    1 0 0 0 1'), u'Analog Balance': (<function <lambda>>, u'1
    1 1'), u'Color Matrix 1': (<function <lambda>>, u'1 0 0 0
    1 0 0 0 1'))}))
```

Reads given *dng* files exif tags using given binding.

Parameters

- **dng_files** (array_like) – *dng* files to read the exif tags from.
- **exif_tags_binding** (dict_like) – Exif tags binding.

Returns *dng* files exif tags.

Return type `list`

Highlights Recovery

- *Clipped Highlights Recovery*

Clipped Highlights Recovery

colour_hdri

<code>highlights_recovery_blend(</code> RGB, multipliers)	Performs highlights recovery using <i>Coffin (1997)</i> method from <i>dcraw</i> .
<code>highlights_recovery_LCHab</code> (RGB[, threshold, ...])	Performs highlights recovery in <i>CIE L*C*Hab</i> colourspace.

colour_hdri.highlights_recovery_blend

`colour_hdri.highlights_recovery_blend`(RGB, multipliers, threshold=0.99)
Performs highlights recovery using *Coffin (1997)* method from *dcraw*.

Parameters

- **RGB** (array_like) – RGB colourspace array.
- **multipliers** (array_like) – Normalised camera white level or white balance multipliers.
- **threshold** (numeric, optional) – Threshold for highlights selection.

Returns Highlights recovered RGB colourspace array.

Return type ndarray

References

- [\[Cof15\]](#)

colour_hdri.highlights_recovery_LCHab

`colour_hdri.highlights_recovery_LCHab`(RGB, threshold=None, RGB_colourspace=RGB_Colourspace(sRGB[[
0.64, 0.33][, 0.3, 0.6][, 0.15, 0.06]][, 0.3127, 0.329
], D65[[, 0.4124, 0.3576, 0.1805][, 0.2126, 0.7152,
0.0722][, 0.0193, 0.1192, 0.9505]][[, 3.2406, -
1.5372, -0.4986][, -0.9689, 1.8758, 0.0415][, 0.0557,
-0.204, 1.057]]], <function oetf_sRGB>, <function
oetf_reverse_sRGB>, False, False))

Performs highlights recovery in *CIE L*C*Hab* colourspace.

Parameters

- **RGB** (array_like) – RGB colourspace array.
- **threshold** (numeric, optional) – Threshold for highlights selection, automatically computed if not given.

- **RGB_colourspace** (RGB_Colourspace, optional) – Working *RGB* colourspace to perform the *CIE L*C*Hab* to and from.

Returns Highlights recovered *RGB* colourspace array.

Return type ndarray

Image Sampling

- *Viriyothai (2009)*
- *Grossberg (2013)*

Viriyothai (2009)

colour_hdri

light_probe_sampling_variance_minimization_Viriyothai2009(*Sample given*) light probe to find lights using *Viriyothai (2009)* variance minimization light probe sampling algorithm.

colour_hdri.light_probe_sampling_variance_minimization_Viriyothai2009

```
colour_hdri.light_probe_sampling_variance_minimization_Viriyothai2009(light_probe,
lights_count=16,
colourspace=RGB_Colourspace(sRGB[[[
0.64, 0.33 ]], [ 0.3,
0.6 ]], [ 0.15, 0.06
]]], [ 0.3127,
0.329 ], D65[[[
0.4124, 0.3576,
0.1805 ]], [ 0.2126,
0.7152, 0.0722 ]],
[ 0.0193, 0.1192,
0.9505 ] ] [[,
3.2406, -1.5372,
-0.4986 ]], [ -0.9689,
1.8758, 0.0415 ]],
[ 0.0557, -0.204,
1.057 ] ], <function
oetf_sRGB>,
<function
oetf_reverse_sRGB>,
False, False))
```

Sample given light probe to find lights using *Viriyothai (2009)* variance minimization light probe sampling algorithm.

Parameters

- **light_probe** (array_like) – Array to sample for lights.

- **lights_count** (*int*) – Amount of lights to generate.
- **colourspace** (*colour.RGB_Colourspace*, optional) – *RGB* colourspace used for internal *Luminance* computation.

Returns list of `colour_hdri.sampling.variance_minimization.Light_Specification` lights.

Return type *list*

References

- [\[VD09\]](#)

Grossberg (2013)

`colour_hdri`

<code>samples_Grossberg2003(image_stack[, samples, n])</code>	Returns the samples for given image stack intensity histograms using <i>Grossberg (2003)</i> method.
---	--

`colour_hdri.samples_Grossberg2003`

`colour_hdri.samples_Grossberg2003(image_stack, samples=1000, n=256)`

Returns the samples for given image stack intensity histograms using *Grossberg (2003)* method.

Parameters

- **image_stack** (*array_like*) – Stack of single channel or multi-channel floating point images.
- **samples** (*int*, optional) – Samples count.
- **n** (*int*, optional) – Histograms bins count.

Returns Intensity histograms samples.

Return type *ndarray*

References

- [\[BB14\]](#)
- [\[GN03\]](#)

Tonemapping Operators

- *Global*
 - *Simple*
 - *Normalisation*

- *Gamma*
- *Logarithmic*
- *Logarithmic Mapping*
- *Exponential*
- *Exponentiation Mapping*
- *Schlick (1994)*
- *Tumblin, Hodgins and Guenter (1999)*
- *Reinhard and Devlin (2004)*
- *Habbe (2010) - Filmic*

Global

Simple

colour_hdri

tonemapping_operator_simple(<i>RGB</i>)	Performs given <i>RGB</i> array tonemapping using the simple method: $\frac{RGB}{RGB + 1}$.
---	--

colour_hdri.tonemapping_operator_simple

colour_hdri.tonemapping_operator_simple(*RGB*)

Performs given *RGB* array tonemapping using the simple method: $\frac{RGB}{RGB + 1}$.

Parameters *RGB* (array_like) – *RGB* array to perform tonemapping onto.

Returns Tonemapped *RGB* array.

Return type ndarray

References

- [\[Wikb\]](#)

Examples

```
>>> tonemapping_operator_simple(np.array(
...     [[0.48046875, 0.35156256, 0.23632812],
...     [1.39843753, 0.55468757, 0.39062594]],
...     [[4.40625388, 2.15625895, 1.34375372],
...     [6.59375023, 3.43751395, 2.21875829]]))
array([[ 0.3245382...,  0.2601156...,  0.1911532...],
       [ 0.5830618...,  0.3567839...,  0.2808993...]])
```

```
[ [ 0.8150290..., 0.6831692..., 0.5733340...],
  [ 0.8683127..., 0.7746486..., 0.6893211...]]])
```

Normalisation

colour_hdri

<code>tonemapping_operator_normalisation(RGB[, ...])</code>	Performs given <i>RGB</i> array tonemapping using the normalisation method.
---	---

colour_hdri.tonemapping_operator_normalisation

```
colour_hdri.tonemapping_operator_normalisation(RGB, colourspace=RGB Colourspace(sRGB[[
    0.64, 0.33][, 0.3, 0.6][, 0.15, 0.06]][,
    0.3127, 0.329], D65[[, 0.4124, 0.3576,
    0.1805][, 0.2126, 0.7152, 0.0722][, 0.0193,
    0.1192, 0.9505]]][, 3.2406, -1.5372, -0.4986
    ][, -0.9689, 1.8758, 0.0415][, 0.0557, -0.204,
    1.057]], <function oetf_sRGB>, <function
    oetf_reverse_sRGB>, False, False))
```

Performs given *RGB* array tonemapping using the normalisation method.

Parameters

- **RGB** (array_like) – *RGB* array to perform tonemapping onto.
- **colourspace** (*colour.RGB_Colourspace*, optional) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type ndarray

References

- [\[BADC11b\]](#)

Examples

```
>>> tonemapping_operator_normalisation(np.array(
...     [[0.48046875, 0.35156256, 0.23632812],
...     [1.39843753, 0.55468757, 0.39062594]],
...     [[4.40625388, 2.15625895, 1.34375372],
...     [6.59375023, 3.43751395, 2.21875829]]]))
array([[ [ 0.1194997..., 0.0874388..., 0.0587783...],
        [ 0.3478122..., 0.1379590..., 0.0971544...]],

       [[ 1.0959009..., 0.5362936..., 0.3342115...],
        [ 1.6399638..., 0.8549608..., 0.5518382...]])
```

Gamma

colour_hdri

<code>tonemapping_operator_gamma(RGB[, gamma, EV])</code>	Performs given <i>RGB</i> array tonemapping using the gamma and exposure correction method.
---	---

colour_hdri.tonemapping_operator_gamma

colour_hdri.**tonemapping_operator_gamma**(*RGB*, *gamma*=1, *EV*=0)

Performs given *RGB* array tonemapping using the gamma and exposure correction method.

Parameters

- **RGB** (array_like) – *RGB* array to perform tonemapping onto.
- **gamma** (numeric, optional) – γ correction value.
- **EV** (numeric, optional) – Exposure adjustment value.

Returns Tonemapped *RGB* array.

Return type ndarray

References

- [\[BADC11b\]](#)

Examples

```
>>> tonemapping_operator_gamma(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...         [1.39843753, 0.55468757, 0.39062594]],
...     [[4.40625388, 2.15625895, 1.34375372],
...         [6.59375023, 3.43751395, 2.21875829]]]),
...     1.0, -3.0)
array([[[[ 0.0600585...,  0.0439453...,  0.0295410...],
          [ 0.1748046...,  0.0693359...,  0.0488282...]],
        [[ 0.5507817...,  0.2695323...,  0.1679692...],
          [ 0.8242187...,  0.4296892...,  0.2773447...]]]])
```

Logarithmic

colour_hdri

<code>tonemapping_operator_logarithmic(RGB[, q, ...])</code>	Performs given <i>RGB</i> array tonemapping using the logarithmic method.
<code>tonemapping_operator_exponential(RGB[, q, ...])</code>	Performs given <i>RGB</i> array tonemapping using the exponential method.

Continued on next page

Table 3.17 – continued from previous page

<code>tonemapping_operator_logarithmic_mapping(</code> <i>RGB</i> <code>)</code>	Performs given <i>RGB</i> array tonemapping using the logarithmic mapping method.
<code>tonemapping_operator_exponentiation_mapping(</code> <i>RGB</i> <code>)</code>	Performs given <i>RGB</i> array tonemapping using the exponentiation mapping method.
<code>tonemapping_operator_Schlick1994(</code> <i>RGB</i> <code>[, p, ...])</code>	Performs given <i>RGB</i> array tonemapping using <i>Schlick (1994)</i> method.
<code>tonemapping_operator_Tumblin1999(</code> <i>RGB</i> <code>[, ...])</code>	Performs given <i>RGB</i> array tonemapping using <i>Tumblin, Hodgins and Guenter (1999)</i> method.
<code>tonemapping_operator_Reinhard2004(</code> <i>RGB</i> <code>[, f, ...])</code>	Performs given <i>RGB</i> array tonemapping using <i>Reinhard and Devlin (2004)</i> method.
<code>tonemapping_operator_filmic(</code> <i>RGB</i> <code>[, ...])</code>	Performs given <i>RGB</i> array tonemapping using <i>Habbe (2010)</i> method.

colour_hdri.tonemapping_operator_logarithmic

`colour_hdri.tonemapping_operator_logarithmic(`*RGB*`, q=1, k=1, colourspace=`*RGB_Colourspace*`(sRGB[[`
`0.64, 0.33]``[, 0.3, 0.6 ]``[, 0.15, 0.06]``][, 0.3127,`
`0.329]``, D65[[``[, 0.4124, 0.3576, 0.1805]``[,`
`0.2126, 0.7152, 0.0722]``[, 0.0193, 0.1192,`
`0.9505]``][, 3.2406, -1.5372, -0.4986 ]``[,`
`-0.9689, 1.8758, 0.0415]``[, 0.0557, -0.204,`
`1.057]``], <function oetf_sRGB>, <function`
`oetf_reverse_sRGB>, False, False))`

Performs given *RGB* array tonemapping using the logarithmic method.

Parameters

- **RGB** (*array_like*) – *RGB* array to perform tonemapping onto.
- **q** (*numeric, optional*) – *q*.
- **k** (*numeric, optional*) – *k*.
- **colourspace** (*colour.RGB_Colourspace, optional*) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type *ndarray*

References

- [\[BADC11b\]](#)

Examples

```
>>> tonemapping_operator_logarithmic(np.array(
...     [[0.48046875, 0.35156256, 0.23632812],
...     [1.39843753, 0.55468757, 0.39062594]],
...     [[4.40625388, 2.15625895, 1.34375372],
...     [6.59375023, 3.43751395, 2.21875829]]),
...     1.0, 25)
```

```
array([[[ 0.0884587...,  0.0647259...,  0.0435102...],
        [ 0.2278222...,  0.0903652...,  0.0636376...]],

       [[ 0.4717487...,  0.2308565...,  0.1438669...],
        [ 0.5727396...,  0.2985858...,  0.1927235...]]])
```

colour_hdri.tonemapping_operator_exponential

`colour_hdri.tonemapping_operator_exponential(`*RGB*`,` *q*`=1,` *k*`=1,` *colourspace*`=`*RGB_Colourspace*`(`*sRGB*`[[`
 0.64, 0.33`][`*0.3, 0.6*`][`*0.15, 0.06*`][`*0.3127,*
 0.329`],` *D65*`[[`*0.4124, 0.3576, 0.1805*`][`
 0.2126, 0.7152, 0.0722`][`*0.0193, 0.1192,*
 0.9505`][`*3.2406, -1.5372, -0.4986*`][`
 -0.9689, 1.8758, 0.0415`][`*0.0557, -0.204,*
 1.057`]],` *<function oetf_sRGB>*`,` *<function*
 oetf_reverse_sRGB>`,` *False, False*`))`

Performs given *RGB* array tonemapping using the exponential method.

Parameters

- **RGB** (*array_like*) – *RGB* array to perform tonemapping onto.
- **q** (*numeric, optional*) – *q*.
- **k** (*numeric, optional*) – *k*.
- **colourspace** (*colour.RGB_Colourspace, optional*) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type *ndarray*

References

- [\[BADC11b\]](#)

Examples

```
>>> tonemapping_operator_exponential(np.array(
...     [[0.48046875, 0.35156256, 0.23632812],
...     [1.39843753, 0.55468757, 0.39062594]],
...     [[4.40625388, 2.15625895, 1.34375372],
...     [6.59375023, 3.43751395, 2.21875829]]]),
...     1.0, 25)
array([[[ 0.0148082...,  0.0108353...,  0.0072837...],
        [ 0.0428669...,  0.0170031...,  0.0119740...]],

       [[ 0.1312736...,  0.0642404...,  0.0400338...],
        [ 0.1921684...,  0.1001830...,  0.0646635...]]])
```

colour_hdri.tonemapping_operator_logarithmic_mapping

```
colour_hdri.tonemapping_operator_logarithmic_mapping(
    RGB, p=1, q=1,
    colourspace=RGB_Colourspace(sRGB[[[
        0.64, 0.33], [0.3, 0.6], [0.15, 0.06]
    ]], [0.3127, 0.329], D65[[[0.4124,
        0.3576, 0.1805], [0.2126, 0.7152,
        0.0722], [0.0193, 0.1192, 0.9505]
    ]], [[3.2406, -1.5372, -0.4986], [
        -0.9689, 1.8758, 0.0415], [0.0557,
        -0.204, 1.057]]], <function oetf_sRGB>,
    <function oetf_reverse_sRGB>, False,
    False))
```

Performs given *RGB* array tonemapping using the logarithmic mapping method.

Parameters

- **RGB** (array_like) – *RGB* array to perform tonemapping onto.
- **p** (numeric, optional) – *p*.
- **q** (numeric, optional) – *q*.
- **colourspace** (*colour.RGB_Colourspace*, optional) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type ndarray

References

- [\[Sch94\]](#)

Examples

```
>>> tonemapping_operator_logarithmic_mapping(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...         [1.39843753, 0.55468757, 0.39062594]],
...         [[4.40625388, 2.15625895, 1.34375372],
...         [6.59375023, 3.43751395, 2.21875829]]]))
array([[ [ 0.2532899...,  0.1853341...,  0.1245857...],
        [ 0.6523387...,  0.2587489...,  0.1822179...]],

       [ [ 1.3507897...,  0.6610269...,  0.4119437...],
        [ 1.6399638...,  0.8549608...,  0.5518382...]])
```

colour_hdri.tonemapping_operator_exponentiation_mapping

```
colour_hdri.tonemapping_operator_exponentiation_mapping(RGB, p=1, q=1,
    colourspace=RGB_Colourspace(sRGB[[[
        0.64, 0.33][[ 0.3, 0.6][[ 0.15, 0.06
        ]][[ 0.3127, 0.329], D65[[[
        0.4124, 0.3576, 0.1805][[ 0.2126,
        0.7152, 0.0722][[ 0.0193, 0.1192,
        0.9505]]][[ 3.2406, -1.5372, -
        0.4986][[ -0.9689, 1.8758, 0.0415
        ][[ 0.0557, -0.204, 1.057]]],
        <function oetf_sRGB>, <function
        oetf_reverse_sRGB>, False, False))
```

Performs given *RGB* array tonemapping using the exponentiation mapping method.

Parameters

- **RGB** (*array_like*) – *RGB* array to perform tonemapping onto.
- **p** (*numeric*, optional) – *p*.
- **q** (*numeric*, optional) – *q*.
- **colourspace** (*colour.RGB_Colourspace*, optional) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type ndarray

References

- [\[Sch94\]](#)

Examples

```
>>> tonemapping_operator_exponentiation_mapping(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...         [1.39843753, 0.55468757, 0.39062594]],
...     [[4.40625388, 2.15625895, 1.34375372],
...         [6.59375023, 3.43751395, 2.21875829]]]))
array([[ [ 0.1194997...,  0.0874388...,  0.0587783...],
        [ 0.3478122...,  0.1379590...,  0.0971544...]],

       [ [ 1.0959009...,  0.5362936...,  0.3342115...],
        [ 1.6399638...,  0.8549608...,  0.5518382...]])
```

colour_hdri.tonemapping_operator_Schlick1994

`colour_hdri.tonemapping_operator_Schlick1994(RGB, p=1, colourspace=RGB_Colourspace(sRGB[[[0.64, 0.33], [0.3, 0.6], [0.15, 0.06]]], 0.3127, 0.329], D65[[[0.4124, 0.3576, 0.1805], [0.2126, 0.7152, 0.0722], [0.0193, 0.1192, 0.9505]]], [3.2406, -1.5372, -0.4986], [-0.9689, 1.8758, 0.0415], [0.0557, -0.204, 1.057]]], <function oetf_sRGB>, <function oetf_reverse_sRGB>, False, False)`

Performs given *RGB* array tonemapping using *Schlick (1994)* method.

Parameters

- **RGB** (array_like) – *RGB* array to perform tonemapping onto.
- **p** (numeric, optional) – *p*.
- **colourspace** (*colour.RGB_Colourspace*, optional) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type ndarray

References

- [\[BADC11b\]](#)
- [\[Sch94\]](#)

Examples

```
>>> tonemapping_operator_Schlick1994(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...         [1.39843753, 0.55468757, 0.39062594]],
...     [[4.40625388, 2.15625895, 1.34375372],
...         [6.59375023, 3.43751395, 2.21875829]]]))
array([[[[ 0.1194997...,  0.0874388...,  0.0587783...],
          [ 0.3478122...,  0.1379590...,  0.0971544...],

          [ 1.0959009...,  0.5362936...,  0.3342115...],
          [ 1.6399638...,  0.8549608...,  0.5518382...]]]])
```


colour_hdri.tonemapping_operator_Tumblin1999

```
colour_hdri.tonemapping_operator_Tumblin1999(
    RGB, L_da=20, C_max=100, L_max=100,
    colourspace=RGB_Colourspace(sRGB[[[ 0.64, 0.33
    ], [ 0.3, 0.6 ], [ 0.15, 0.06 ]], [ 0.3127, 0.329
    ], D65[[[ 0.4124, 0.3576, 0.1805 ], [ 0.2126,
    0.7152, 0.0722 ], [ 0.0193, 0.1192, 0.9505 ]], [
    3.2406, -1.5372, -0.4986 ], [ -0.9689, 1.8758,
    0.0415 ], [ 0.0557, -0.204, 1.057 ]], <function
    oetf_sRGB>, <function oetf_reverse_sRGB>, False,
    False))
```

Performs given *RGB* array tonemapping using *Tumblin, Hodgins and Guenter (1999)* method.

Parameters

- **RGB** (array_like) – *RGB* array to perform tonemapping onto.
- **L_da** (numeric, optional) – L_{da} display adaptation luminance, a mid-range display value.
- **C_max** (numeric, optional) – C_{max} maximum contrast available from the display.
- **L_max** (numeric, optional) – L_{max} maximum display luminance.
- **colourspace** (*colour.RGB_Colourspace*, optional) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type ndarray

References

- [\[THG99\]](#)

Examples

```
>>> tonemapping_operator_Tumblin1999(np.array(
...     [[0.48046875, 0.35156256, 0.23632812],
...     [1.39843753, 0.55468757, 0.39062594]],
...     [[4.40625388, 2.15625895, 1.34375372],
...     [6.59375023, 3.43751395, 2.21875829]]))
array([[ 0.0400492...,  0.0293043...,  0.0196990...],
       [ 0.1019768...,  0.0404489...,  0.0284852...]],

       [[ 0.2490212...,  0.1218618...,  0.0759427...],
       [ 0.3408366...,  0.1776880...,  0.1146895...]])
```

colour_hdri.tonemapping_operator_Reinhard2004

```
colour_hdri.tonemapping_operator_Reinhard2004(
    RGB, f=0, m=0.3, a=0, c=0,
    colourspace=RGB_Colourspace(sRGB[[[ 0.64,
    0.33 ]], [ 0.3, 0.6 ]], [ 0.15, 0.06 ]], [ 0.3127,
    0.329 ], D65[[[ 0.4124, 0.3576, 0.1805 ]],
    0.2126, 0.7152, 0.0722 ]], [ 0.0193, 0.1192,
    0.9505 ]], [[ 3.2406, -1.5372, -0.4986 ]],
    -0.9689, 1.8758, 0.0415 ]], [ 0.0557, -0.204,
    1.057 ]], <function oetf_sRGB>, <function
    oetf_reverse_sRGB>, False, False))
```

Performs given *RGB* array tonemapping using *Reinhard and Devlin (2004)* method.

Parameters

- **RGB** (array_like) – *RGB* array to perform tonemapping onto.
- **f** (numeric, optional) – *f*.
- **m** (numeric, optional) – *m*.
- **a** (numeric, optional) – *a*.
- **c** (numeric, optional) – *c*.
- **colourspace** (*colour.RGB_Colourspace*, optional) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type ndarray

References

- [\[RD05\]](#)

Examples

```
>>> tonemapping_operator_Reinhard2004(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...         [1.39843753, 0.55468757, 0.39062594]],
...         [[4.40625388, 2.15625895, 1.34375372],
...         [6.59375023, 3.43751395, 2.21875829]]]),
...     -10)
array([[[ 0.0216792...,  0.0159556...,  0.0107821...],
        [ 0.0605894...,  0.0249445...,  0.0176972...]],

       [[ 0.1688972...,  0.0904532...,  0.0583584...],
        [ 0.2331935...,  0.1368456...,  0.0928316...]])
```

colour_hdri.tonemapping_operator_filmic

`colour_hdri.tonemapping_operator_filmic(`*RGB*`,` *shoulder_strength*`=0.22,` *linear_strength*`=0.3,` *linear_angle*`=0.1,` *toe_strength*`=0.2,` *toe_numerator*`=0.01,` *toe_denominator*`=0.3,` *exposure_bias*`=2,` *linear_whitepoint*`=11.2)`

Performs given *RGB* array tonemapping using *Habbe (2010)* method.

Parameters

- **RGB** (*array_like*) – *RGB* array to perform tonemapping onto.
- **shoulder_strength** (numeric, optional) – Shoulder strength.
- **linear_strength** (numeric, optional) – Linear strength.
- **linear_angle** (numeric, optional) – Linear angle.
- **toe_strength** (numeric, optional) – Toe strength.
- **toe_numerator** (numeric, optional) – Toe numerator.
- **toe_denominator** (numeric, optional) – Toe denominator.
- **exposure_bias** (numeric, optional) – Exposure bias.
- **linear_whitepoint** (numeric, optional) – Linear whitepoint.

Returns Tonemapped *RGB* array.

Return type ndarray

References

- [\[Hab10a\]](#)
- [\[Hab10b\]](#)

Examples

```
>>> tonemapping_operator_filmic(np.array(
...     [[0.48046875, 0.35156256, 0.23632812],
...      [1.39843753, 0.55468757, 0.39062594]],
...     [[4.40625388, 2.15625895, 1.34375372],
...      [6.59375023, 3.43751395, 2.21875829]]))
array([[ 0.4507954...,  0.3619673...,  0.2617269...],
       [ 0.7567191...,  0.4933310...,  0.3911730...]],

       [[ 0.9725554...,  0.8557374...,  0.7465713...],
       [ 1.0158782...,  0.9382937...,  0.8615161...]])
```

Logarithmic Mapping

`colour_hdri`

<code>tonemapping_operator_logarithmic_mapping(RGB)</code>	Performs given <i>RGB</i> array tonemapping using the logarithmic mapping method.
--	---

Exponential

`colour_hdri`

<code>tonemapping_operator_exponential(RGB[, q, ...])</code>	Performs given <i>RGB</i> array tonemapping using the exponential method.
--	---

Exponentiation Mapping

`colour_hdri`

<code>tonemapping_operator_exponentiation_mapping(RGB)</code>	Performs given <i>RGB</i> array tonemapping using the exponentiation mapping method.
<code>tonemapping_operator_Schlick1994(RGB[, p, ...])</code>	Performs given <i>RGB</i> array tonemapping using <i>Schlick (1994)</i> method.
<code>tonemapping_operator_Tumblin1999(RGB[, ...])</code>	Performs given <i>RGB</i> array tonemapping using <i>Tumblin, Hodgins and Guenter (1999)</i> method.
<code>tonemapping_operator_Reinhard2004(RGB[, f, ...])</code>	Performs given <i>RGB</i> array tonemapping using <i>Reinhard and Devlin (2004)</i> method.
<code>tonemapping_operator_filmic(RGB[, ...])</code>	Performs given <i>RGB</i> array tonemapping using <i>Habbe (2010)</i> method.

Schlick (1994)

`colour_hdri`

<code>tonemapping_operator_Schlick1994(RGB[, p, ...])</code>	Performs given <i>RGB</i> array tonemapping using <i>Schlick (1994)</i> method.
--	---

Tumblin, Hodgins and Guenter (1999)

`colour_hdri`

<code>tonemapping_operator_Tumblin1999(RGB[, ...])</code>	Performs given <i>RGB</i> array tonemapping using <i>Tumblin, Hodgins and Guenter (1999)</i> method.
---	--

Reinhard and Devlin (2004)

`colour_hdri`

<code>tonemapping_operator_Reinhard2004(</code> <code>RGB[, f, ...]</code> <code>)</code>	Performs given <i>RGB</i> array tonemapping using <i>Reinhard and Devlin (2004)</i> method.
---	---

Habbe (2010) - Filmic

`colour_hdri`

<code>tonemapping_operator_filmic(</code> <code>RGB[, ...]</code> <code>)</code>	Performs given <i>RGB</i> array tonemapping using <i>Habbe (2010)</i> method.
--	---

Utilities

- *Common*
- *EXIF Data Manipulation*
- *Image Exposure Value Computation*
- *Image Data & Metadata Utilities*

Common

`colour_hdri`

<code>vivification()</code>	Implements supports for vivification of the underlying dict like
<code>vivified_to_dict(vivified)</code>	Converts given vivified data-structure to dictionary.
<code>path_exists(path)</code>	Returns if given path exists.
<code>filter_files(directory, extensions)</code>	Filters given directory for files matching given extensions.

`colour_hdri.vivification`

`colour_hdri.vivification()`

Implements supports for vivification of the underlying dict like data-structure, magical!

Returns

Return type defaultdict

Examples

```
>>> vivified = vivification()
>>> vivified['my']['attribute'] = 1
>>> vivified['my']
defaultdict(<function vivification at 0x...>, {u'attribute': 1})
```

```
>>> vivified['my']['attribute']
1
```

`colour_hdri.vivified_to_dict`

`colour_hdri.vivified_to_dict(vivified)`

Converts given vivified data-structure to dictionary.

Parameters `vivified` (defaultdict) – Vivified data-structure.

Returns

Return type `dict`

Examples

```
>>> vivified = vivification()
>>> vivified['my']['attribute'] = 1
>>> vivified_to_dict(vivified)
{u'my': {u'attribute': 1}}
```

`colour_hdri.path_exists`

`colour_hdri.path_exists(path)`

Returns if given path exists.

Parameters `path` (unicode) – Path to check the existence.

Returns

Return type `bool`

Examples

```
>>> path_exists(__file__)
True
>>> path_exists('')
False
```

`colour_hdri.filter_files`

`colour_hdri.filter_files(directory, extensions)`

Filters given directory for files matching given extensions.

Parameters

- **directory** (unicode) – Directory to filter.
- **extensions** (tuple or list) – Extensions to filter on.

Returns Filtered files.

Return type `list`

EXIF Data Manipulation

colour_hdri

EXIF_EXECUTABLE	Command line exif manipulation application, usually Phil Harvey's <i>ExifTool</i> .
ExifTag	Hunt colour appearance model induction factors.
parse_exif_string(exif_tag)	Parses given exif tag assuming it is a string and return its value.
parse_exif_numeric(exif_tag[, dtype])	Parses given exif tag assuming it is a numeric type and return its value.
parse_exif_fraction(exif_tag[, dtype])	Parses given exif tag assuming it is a fraction and return its value.
parse_exif_array(exif_tag[, dtype, shape])	Parses given exif tag assuming it is an array and return its value.
parse_exif_data(data)	Parses given exif data output from <i>exiftool</i> .
read_exif_tags(image)	Returns given image exif image tags.
copy_exif_tags(source, target)	Copies given source image file exif tag to given image target.
update_exif_tags(images)	Updates given images siblings images pairs exif tags.
delete_exif_tags(image)	Deletes all given image exif tags.
read_exif_tag(image, tag)	Returns given image exif tag value.
write_exif_tag(image, tag, value)	Sets given image exif tag value.

colour_hdri.EXIF_EXECUTABLE

colour_hdri.EXIF_EXECUTABLE = u'exiftool'

Command line exif manipulation application, usually Phil Harvey's *ExifTool*.

EXIF_EXECUTABLE : unicode

colour_hdri.ExifTag

class colour_hdri.ExifTag

Hunt colour appearance model induction factors.

Parameters

- **group** (unicode, optional) – Exif tag group name.
- **name** (unicode, optional) – Exif tag name.
- **value** (object, optional) – Exif tag value.
- **identifier** (numeric, optional) – Exif tag identifier.

Returns a new instance of the `colour_hdri.ExifTag` class.

__init__()

`x.__init__(...)` initializes x; see `help(type(x))` for signature

Methods

<code>count(...)</code>	
<code>index((value, [start, ...])</code>	Raises <code>ValueError</code> if the value is not present.

`colour_hdri.parse_exif_string`

`colour_hdri.parse_exif_string(exif_tag)`
 Parses given exif tag assuming it is a string and return its value.

Parameters `exif_tag` (`ExifTag`) – Exif tag to parse.

Returns Parsed exif tag value.

Return type unicode

`colour_hdri.parse_exif_numeric`

`colour_hdri.parse_exif_numeric(exif_tag, dtype=<type 'numpy.float64'>)`
 Parses given exif tag assuming it is a numeric type and return its value.

Parameters

- `exif_tag` (`ExifTag`) – Exif tag to parse.
- `dtype` (`object`, optional) – Return value data type.

Returns Parsed exif tag value.

Return type numeric

`colour_hdri.parse_exif_fraction`

`colour_hdri.parse_exif_fraction(exif_tag, dtype=<type 'numpy.float64'>)`
 Parses given exif tag assuming it is a fraction and return its value.

Parameters

- `exif_tag` (`ExifTag`) – Exif tag to parse.
- `dtype` (`object`, optional) – Return value data type.

Returns Parsed exif tag value.

Return type numeric

`colour_hdri.parse_exif_array`

`colour_hdri.parse_exif_array(exif_tag, dtype=<type 'numpy.float64'>, shape=None)`
 Parses given exif tag assuming it is an array and return its value.

Parameters

- `exif_tag` (`ExifTag`) – Exif tag to parse.
- `dtype` (`object`, optional) – Return value data type.
- `shape` (`array_like`, optional) – Shape of

Returns Parsed exif tag value.

Return type ndarray

`colour_hdri.parse_exif_data`

`colour_hdri.parse_exif_data(data)`

Parses given exif data output from *exiftool*.

Parameters `data` (unicode) – Exif data.

Returns Parsed exif data.

Return type list

`colour_hdri.read_exif_tags`

`colour_hdri.read_exif_tags(image)`

Returns given image exif image tags.

Parameters `image` (unicode) – Image file.

Returns Exif tags.

Return type defaultdict

`colour_hdri.copy_exif_tags`

`colour_hdri.copy_exif_tags(source, target)`

Copies given source image file exif tag to given image target.

Parameters

- **source** (unicode) – Source image file.
- **target** (unicode) – Target image file.

Returns Definition success.

Return type bool

`colour_hdri.update_exif_tags`

`colour_hdri.update_exif_tags(images)`

Updates given images siblings images pairs exif tags.

Parameters `images` (list) – Image files to update.

Returns Definition success.

Return type bool

`colour_hdri.delete_exif_tags`

`colour_hdri.delete_exif_tags(image)`

Deletes all given image exif tags.

Parameters `image` (unicode) – Image file.

Returns Definition success.

Return type `bool`

`colour_hdri.read_exif_tag`

`colour_hdri.read_exif_tag(image, tag)`
Returns given image exif tag value.

Parameters

- **image** (unicode) – Image file.
- **tag** (unicode) – Tag.

Returns Tag value.

Return type unicode

`colour_hdri.write_exif_tag`

`colour_hdri.write_exif_tag(image, tag, value)`
Sets given image exif tag value.

Parameters

- **image** (unicode) – Image file.
- **tag** (unicode) – Tag.
- **value** (unicode) – Value.

Returns Definition success.

Return type `bool`

Image Exposure Value Computation

`colour_hdri`

<code>exposure_value(f_number, exposure_time, iso)</code>	Computes the exposure value from given image <i>FNumber</i> , <i>Exposure Time</i> and <i>ISO</i> values.
<code>adjust_exposure(a, EV)</code>	Adjusts given array exposure using given <i>EV</i> exposure value.
<code>average_luminance(f_number, exposure_time, iso)</code>	Computes the average luminance from given image <i>FNumber</i> , <i>Exposure Time</i> and <i>ISO</i> values.

`colour_hdri.exposure_value`

`colour_hdri.exposure_value(f_number, exposure_time, iso)`
Computes the exposure value from given image *FNumber*, *Exposure Time* and *ISO* values.

Parameters

- **f_number** (array_like) – Image *FNumber*.
- **exposure_time** (array_like) – Image *Exposure Time*.

- **iso** (array_like) – Image *ISO*.

Returns Image exposure value.

Return type ndarray

Examples

```
>>> exposure_value(8, 1, 100)
6.0
```

colour_hdri.adjust_exposure

colour_hdri.adjust_exposure(*a*, *EV*)

Adjusts given array exposure using given *EV* exposure value.

Parameters

- **a** (array_like) – Array to adjust the exposure.
- **EV** (numeric) – Exposure adjustment value.

Returns Exposure adjusted array.

Return type ndarray

Examples

```
>>> adjust_exposure(np.array([0.25, 0.5, 0.75, 1]), 1)
array([ 0.5,  1. ,  1.5,  2. ])
```

colour_hdri.average_luminance

colour_hdri.average_luminance(*f_number*, *exposure_time*, *iso*, *k*=12.5)

Computes the average luminance from given image *FNumber*, *Exposure Time* and *ISO* values.

Parameters

- **f_number** (array_like) – Image *FNumber*.
- **exposure_time** (array_like) – Image *Exposure Time*.
- **iso** (array_like) – Image *ISO*.
- **k** (numeric, optional) – Reflected light calibration constant *K*.

Returns Image average luminance.

Return type ndarray

References

- [\[Wika\]](#)

Examples

```
>>> average_luminance(8, 1, 100)
0.125
```

Image Data & Metadata Utilities

colour_hdri

Metadata	Defines the base object for storing exif metadata relevant to HDRI / radiance image generation.
Image([path, data, metadata])	Defines the base object for storing an image along its path, pixel data and metadata needed for HDRI / radiance images generation.
ImageStack()	Defines a convenient stack storing a sequence of images for HDRI / radiance images generation.

colour_hdri.Metadata

class colour_hdri.Metadata

Defines the base object for storing exif metadata relevant to HDRI / radiance image generation.

Parameters

- **f_number** (array_like) – Image *FNumber*.
- **exposure_time** (array_like) – Image *Exposure Time*.
- **iso** (array_like) – Image *ISO*.
- **black_level** (array_like) – Image *Black Level*.
- **white_level** (array_like) – Image *White Level*.
- **white_balance_multipliers** (array_like) – Image white balance multipliers, usually the *As Shot Neutral* matrix.

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

colour_hdri.Image

class colour_hdri.Image(path=None, data=None, metadata=None)

Defines the base object for storing an image along its path, pixel data and metadata needed for HDRI / radiance images generation.

Parameters

- **path** (unicode, optional) – Image path.
- **data** (array_like, optional) – Image pixel data array.
- **metadata** (Metadata, optional) – Image exif metadata.

path

data
metadata
read_data()
read_metadata()
__init__(*path=None, data=None, metadata=None*)

Methods

<code>__init__([path, data, metadata])</code>	
<code>read_data([decoding_cctf])</code>	Reads image pixel data at <code>Image.path</code> attribute.
<code>read_metadata()</code>	Reads image relevant exif metadata at <code>Image.path</code> attribute.

`colour_hdri.ImageStack`

class `colour_hdri.ImageStack`

Defines a convenient stack storing a sequence of images for HDRI / radiance images generation.

ImageStack()
__init__()
__getitem__()
__setitem__()
__delitem__()
__len__()
__getattr__()
__setattr__()
sort()
insert()
from_files()
__init__()

Methods

<code>__init__()</code>	
<code>append(value)</code>	<code>S.append(object)</code> – append object to the end of the sequence
<code>count(...)</code>	
<code>extend(values)</code>	<code>S.extend(iterable)</code> – extend sequence by appending elements from the iterable
<code>from_files(image_files[, decoding_cctf])</code>	Returns a <code>colour_hdri.ImageStack</code> instance with given image files.
<code>index(...)</code>	Raises <code>ValueError</code> if the value is not present.

Continued on next page

Table 3.31 – continued from previous page

<code>insert(index, value)</code>	Reimplements the <code>MutableSequence.insert()</code> method.
<code>pop(...)</code>	Raise <code>IndexError</code> if list is empty or index is out of range.
<code>remove(value)</code>	<code>S.remove(value)</code> – remove first occurrence of value.
<code>reverse()</code>	<code>S.reverse()</code> – reverse <i>IN PLACE</i>
<code>sort([key])</code>	Sorts the underlying data structure.

Indices and tables

- [genindex](#)
- [search](#)

3.1.1.2 Bibliography

Indirect References

Some extra references used in the codebase but not directly part of the public api:

- [\[AdobeSystems15a\]](#)
- [\[AdobeSystems15b\]](#)

3.2 Examples

Various usage examples are available from the [examples directory](#).

CONTRIBUTING

If you would like to contribute to [Colour - HDRI](#), please refer to the following [Contributing](#) guide for [Colour](#).

BIBLIOGRAPHY

The bibliography is available in the repository in [BibTeX](#) format.

ABOUT

Colour - HDRI by Colour Developers

Copyright © 2015-2018 – Colour Developers – colour-science@googlegroups.com

This software is released under terms of New BSD License: <http://opensource.org/licenses/BSD-3-Clause>
<http://github.com/colour-science/colour-hdri>

BIBLIOGRAPHY

- [BADC11a] Francesco Banterle, Alessandro Artusi, Kurt Debattista, and Alan Chalmers. *2.1.1 Generating HDR Content by Combining Multiple Exposures*. A K Peters/CRC Press, 2011. ISBN 978-1568817194.
- [BADC11b] Francesco Banterle, Alessandro Artusi, Kurt Debattista, and Alan Chalmers. 3.2.1 Simple Mapping Methods. In *Advanced High Dynamic Range Imaging*, pages 38–41. A K Peters/CRC Press, 2011.
- [BB14] Francesco Banterle and Luca Benedetti. PICCANTE: An Open and Portable Library for HDR Imaging. 2014.
- [Cof15] Dave Coffin. Dcraw. 2015. URL: <https://www.cybercom.net/~\protect\T1\textdollar\T1\textbackslash{}\sim\protect\T1\textdollar\dcraw/>.
- [DM97] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*, number August, 369–378. New York, New York, USA, 1997. ACM Press. URL: <http://portal.acm.org/citation.cfm?doid=258734.258884>, doi:10.1145/258734.258884.
- [GN03] M.D. Grossberg and S.K. Nayar. Determining the camera response from images: What is knowable? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(11):1455–1467, nov 2003. URL: <http://ieeexplore.ieee.org/document/1240119/>, doi:10.1109/TPAMI.2003.1240119.
- [Hab10a] John Hable. Filmic Tonemapping Operators. 2010. URL: <http://filmicgames.com/archives/75>.
- [Hab10b] John Hable. Uncharted 2: HDR Lighting. 2010. URL: <http://www.slideshare.net/ozlael/hable-john-uncharted2-hdr-lighting>.
- [LLJ16] Sebastien Lagarde, Sebastien Lachambre, and Cyril Jover. An Artist-Friendly Workflow for Panoramic HDRI. 2016. URL: http://blog.selfshadow.com/publications/s2016-shading-course/unity/s2016_pbs_unity_hdri_notes.pdf.
- [McG12] Sandy McGuffog. Hue Twists in DNG Camera Profiles. 2012. URL: <http://dcptool.sourceforge.net/HueTwists.html>.
- [RD05] Erik Reinhard and Kate Devlin. Dynamic Range Reduction Inspired by Photoreceptor Physiology. *IEEE Transactions on Visualization and Computer Graphics*, 11(01):13–24, jan 2005. URL: <http://ieeexplore.ieee.org/document/1359728/>, doi:10.1109/TVCG.2005.9.
- [Sch94] Christophe Schlick. Quantization Techniques for Visualization of High Dynamic Range Pictures. *Proceedings of the Fifth Eurographics Workshop on Rendering*, pages 7–18, 1994.
- [THG99] Jack Tumblin, Jessica K. Hodgins, and Brian K. Guenter. Two methods for display of high contrast images. *ACM Transactions on Graphics*, 18(1):56–94, jan 1999. URL: <http://portal.acm.org/citation.cfm?doid=300776.300783>, doi:10.1145/300776.300783.

- [VD09] Kuntée Viriyothai and Paul Debevec. Variance minimization light probe sampling. In *SIGGRAPH '09: Posters on - SIGGRAPH '09*, number Egsr, 1–1. New York, New York, USA, 2009. ACM Press. URL: <http://dl.acm.org/citation.cfm?id=1599393>, doi:10.1145/1599301.1599393.
- [Wika] Wikipedia. EV as a measure of luminance and illuminance. URL: https://en.wikipedia.org/wiki/Exposure_value#EV_as_a_measure_of_luminance_and_illuminance.
- [Wikb] Wikipedia. Tonemapping - Purpose and methods. URL: http://en.wikipedia.org/wiki/Tone_mapping#Purpose_and_methods.
- [AdobeSystems12a] Adobe Systems. Camera to XYZ (D50) Transform. In *Digital Negative (DNG) Specification*, pages 81. 2012.
- [AdobeSystems12b] Adobe Systems. Digital Negative (DNG) Specification. 2012.
- [AdobeSystems12c] Adobe Systems. Translating Camera Neutral Coordinates to White Balance xy Coordinates. In *Digital Negative (DNG) Specification*, pages 80–81. 2012.
- [AdobeSystems12d] Adobe Systems. Translating White Balance xy Coordinates to Camera Neutral Coordinates. In *Digital Negative (DNG) Specification*, pages 80. 2012.
- [AdobeSystems15a] Adobe Systems. Adobe DNG SDK 1.4 - dng_sdk_1_4/dng_sdk/source/dng_camera_profile.cpp - dng_camera_profile::IlluminantToTemperature. 2015. URL: http://download.adobe.com/pub/adobe/dng/dng_sdk_1_4.zip.
- [AdobeSystems15b] Adobe Systems. Adobe DNG SDK 1.4 - dng_sdk_1_4/dng_sdk/source/dng_tag_values.h - LightSource tag. 2015. URL: http://download.adobe.com/pub/adobe/dng/dng_sdk_1_4.zip.
- [AdobeSystems15c] Adobe Systems. Adobe DNG SDK 1.4. 2015. URL: http://download.adobe.com/pub/adobe/dng/dng_sdk_1_4.zip.

Symbols

[__delitem__\(\)](#) (colour_hdri.ImageStack method), 49
[__getattr__\(\)](#) (colour_hdri.ImageStack method), 49
[__getitem__\(\)](#) (colour_hdri.ImageStack method), 49
[__init__\(\)](#) (colour_hdri.ExifTag method), 43
[__init__\(\)](#) (colour_hdri.Image method), 49
[__init__\(\)](#) (colour_hdri.ImageStack method), 49
[__init__\(\)](#) (colour_hdri.Metadata method), 48
[__len__\(\)](#) (colour_hdri.ImageStack method), 49
[__setattr__\(\)](#) (colour_hdri.ImageStack method), 49
[__setitem__\(\)](#) (colour_hdri.ImageStack method), 49

A

[absolute_luminance_calibration_Lagarde2016\(\)](#) (in module colour_hdri), 8
[adjust_exposure\(\)](#) (in module colour_hdri), 47
[average_luminance\(\)](#) (in module colour_hdri), 47

C

[camera_neutral_to_xy\(\)](#) (in module colour_hdri), 15
[camera_response_functions_Debevec1997\(\)](#) (in module colour_hdri), 10
[camera_space_to_RGB\(\)](#) (in module colour_hdri), 19
[camera_space_to_sRGB\(\)](#) (in module colour_hdri), 20
[camera_space_to_XYZ_matrix\(\)](#) (in module colour_hdri), 18
[convert_dng_files_to_intermediate_files\(\)](#) (in module colour_hdri), 23
[convert_raw_files_to_dng_files\(\)](#) (in module colour_hdri), 22
[copy_exif_tags\(\)](#) (in module colour_hdri), 45

D

[data](#) (colour_hdri.Image attribute), 48
[delete_exif_tags\(\)](#) (in module colour_hdri), 45
[DNG_CONVERSION_ARGUMENTS](#) (in module colour_hdri), 24
[DNG_CONVERTER](#) (in module colour_hdri), 24
[DNG_EXIF_TAGS_BINDING](#) (in module colour_hdri), 24

E

[EXIF_EXECUTABLE](#) (in module colour_hdri), 43
[ExifTag](#) (class in colour_hdri), 43
[exposure_value\(\)](#) (in module colour_hdri), 46

F

[filter_files\(\)](#) (in module colour_hdri), 42
[from_files\(\)](#) (colour_hdri.ImageStack method), 49

G

[g_solve\(\)](#) (in module colour_hdri), 10

H

[hat_function\(\)](#) (in module colour_hdri), 13
[highlights_recovery_blend\(\)](#) (in module colour_hdri), 26
[highlights_recovery_LCHab\(\)](#) (in module colour_hdri), 26

I

[Image](#) (class in colour_hdri), 48
[image_stack_to_radiance_image\(\)](#) (in module colour_hdri), 11
[ImageStack](#) (class in colour_hdri), 49
[ImageStack\(\)](#) (colour_hdri.ImageStack method), 49
[insert\(\)](#) (colour_hdri.ImageStack method), 49

L

[light_probe_sampling_variance_minimization_Viriyothai2009\(\)](#) (in module colour_hdri), 27

M

[Metadata](#) (class in colour_hdri), 48
[metadata](#) (colour_hdri.Image attribute), 49

N

[normal_distribution_function\(\)](#) (in module colour_hdri), 12

P

[parse_exif_array\(\)](#) (in module colour_hdri), 44

[parse_exif_data\(\)](#) (in module `colour_hdri`), [45](#)
[parse_exif_fraction\(\)](#) (in module `colour_hdri`), [44](#)
[parse_exif_numeric\(\)](#) (in module `colour_hdri`), [44](#)
[parse_exif_string\(\)](#) (in module `colour_hdri`), [44](#)
[path](#) (`colour_hdri.Image` attribute), [48](#)
[path_exists\(\)](#) (in module `colour_hdri`), [42](#)

R

[radiance_image_strip_plot\(\)](#) (in module `colour_hdri.plotting`), [21](#)
[RAW_CONVERSION_ARGUMENTS](#) (in module `colour_hdri`), [23](#)
[RAW_CONVERTER](#) (in module `colour_hdri`), [23](#)
[RAW_D_CONVERSION_ARGUMENTS](#) (in module `colour_hdri`), [23](#)
[read_data\(\)](#) (`colour_hdri.Image` method), [49](#)
[read_dng_files_exif_tags\(\)](#) (in module `colour_hdri`), [25](#)
[read_exif_tag\(\)](#) (in module `colour_hdri`), [46](#)
[read_exif_tags\(\)](#) (in module `colour_hdri`), [45](#)
[read_metadata\(\)](#) (`colour_hdri.Image` method), [49](#)

S

[samples_Grossberg2003\(\)](#) (in module `colour_hdri`), [28](#)
[sort\(\)](#) (`colour_hdri.ImageStack` method), [49](#)

T

[tonemapping_operator_exponential\(\)](#) (in module `colour_hdri`), [33](#)
[tonemapping_operator_exponentiation_mapping\(\)](#) (in module `colour_hdri`), [35](#)
[tonemapping_operator_filmic\(\)](#) (in module `colour_hdri`), [39](#)
[tonemapping_operator_gamma\(\)](#) (in module `colour_hdri`), [31](#)
[tonemapping_operator_image_plot\(\)](#) (in module `colour_hdri.plotting`), [21](#)
[tonemapping_operator_logarithmic\(\)](#) (in module `colour_hdri`), [32](#)
[tonemapping_operator_logarithmic_mapping\(\)](#) (in module `colour_hdri`), [34](#)
[tonemapping_operator_normalisation\(\)](#) (in module `colour_hdri`), [30](#)
[tonemapping_operator_Reinhard2004\(\)](#) (in module `colour_hdri`), [38](#)
[tonemapping_operator_Schlick1994\(\)](#) (in module `colour_hdri`), [36](#)
[tonemapping_operator_simple\(\)](#) (in module `colour_hdri`), [29](#)
[tonemapping_operator_Tumblin1999\(\)](#) (in module `colour_hdri`), [37](#)

U

[update_exif_tags\(\)](#) (in module `colour_hdri`), [45](#)
[upper_hemisphere_illuminance_weights_Lagarde2016\(\)](#) (in module `colour_hdri`), [9](#)

V

[vivification\(\)](#) (in module `colour_hdri`), [41](#)
[vivified_to_dict\(\)](#) (in module `colour_hdri`), [42](#)

W

[weighting_function_Debevec1997\(\)](#) (in module `colour_hdri`), [13](#)
[write_exif_tag\(\)](#) (in module `colour_hdri`), [46](#)

X

[xy_to_camera_neutral\(\)](#) (in module `colour_hdri`), [14](#)
[XYZ_to_camera_space_matrix\(\)](#) (in module `colour_hdri`), [16](#)