# Colour - HDRI Documentation

*Release 0.1.7*

**Colour Developers**

**Mar 31, 2020**

# CONTENTS

A Python package implementing various HDRI / Radiance image processing algorithms.

It is open source and freely available under the New BSD License terms.

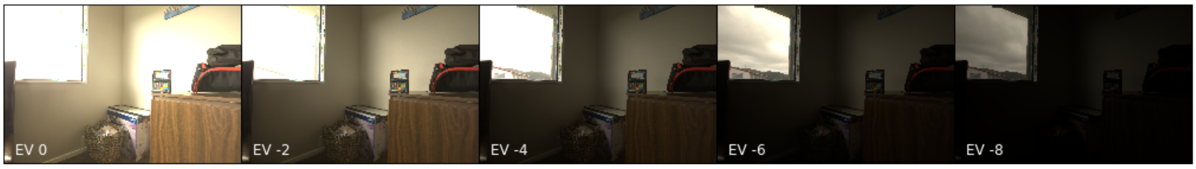


**Table of Contents**

# CHAPTER ONE

## 1.1 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
- Digital Still Camera (DSC) Exposure Model
- Raw Processing Helpers

CHAPTER

TWO

# 1.2 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 1.2.1 Primary Dependencies

**Colour - HDRI** requires various dependencies in order to run:

- Python 2.7 or Python 3.7
- Colour Science

## 2.2 1.2.2 Optional Features Dependencies

- colour-demosaicing
- Adobe DNG Converter
- dcraw
- ExifTool
- rawpy

## 2.3 1.2.3 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]'
```

The overall development dependencies are installed as follows:

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

CHAPTER

THREE

# 1.3 USAGE

## 3.1 1.3.1 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`

| | |
|---|---|
| `absolute_luminance_calibration_Lagarde2016(...)` | Performs absolute *Luminance* calibration of given *RGB* panoramic image using *Lagarde (2016)* method. |
| `upper_hemisphere_illuminance_weights_Lagarde2016(...)` | 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*], *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 eotf_inverse_sRGB>*, *<function eotf_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...],
```

(continues on next page)

(continued from previous page)

```
         [ 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...        ],
       [ 4.0143297...],
       [ 7.3345454...],
       [ 9.3865515...],
       [ 9.8155376...],
       [ 8.5473281...],
       [ 5.8012079...],
       [ 2.0520061...],
       [ 0...        ],
       [ 0...        ],
       [ 0...        ],
       [ 0...        ],
       [ 0...        ],
       [ 0...        ],
       [ 0...        ],
       [ 0...        ]])
```

## Debevec (1997)

colour_hdri

| | |
|---|---|
| g_solve(Z, B[, l_s, w, n]) | 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. |
| camera_response_functions_Debevec1997(...[, ...]) | Returns the camera response functions for given image stack using *Debevec (1997)* 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 $\Delta t$, or log shutter speed for images.
- **l_s** (numeric, optional) – $\lambda$ 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) – $\lambda$ 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]

## Exposure Computation



### Common

colour_hdri

| | |
|---|---|
| average_luminance(N, t, S[, k]) | Computes the average luminance $L$ in $cd \cdot m^{-2}$ from given relative aperture *F-Number* $N$, *Exposure Time* $t$, *ISO* arithmetic speed $S$ and *reflected light calibration constant* $k$. |
| average_illuminance(N, t, S[, c]) | Computes the average illuminance $E$ in $Lux$ from given relative aperture *F-Number* $N$, *Exposure Time* $t$, *ISO* arithmetic speed $S$ and *incident light calibration constant* $c$. |
| luminance_to_exposure_value(L, S[, k]) | Computes the exposure value $EV$ from given scene luminance $L$ in $cd \cdot m^{-2}$, *ISO* arithmetic speed $S$ and *reflected light calibration constant* $k$. |
| illuminance_to_exposure_value(E, S[, c]) | Computes the exposure value $EV$ from given scene illuminance $E$ in $Lux$, *ISO* arithmetic speed $S$ and *incident light calibration constant* $c$. |
| adjust_exposure(a, EV) | Adjusts given array exposure using given $EV$ exposure value. |

#### colour_hdri.average_luminance

colour_hdri.**average_luminance**(*N, t, S, k=12.5*)

Computes the average luminance $L$ in $cd \cdot m^{-2}$ from given relative aperture *F-Number* $N$, *Exposure Time* $t$, *ISO* arithmetic speed $S$ and *reflected light calibration constant* $k$.

**Parameters**

- **N** (array_like) – Relative aperture *F-Number* $N$.
- **t** (array_like) – *Exposure Time* $t$.
- **S** (array_like) – *ISO* arithmetic speed $S$.

- **k** (numeric, optional) – *Reflected light calibration constant* $k$. *ISO 2720:1974* recommends a range for $k$ of 10.6 to 13.4 with luminance in $cd \cdot m^{-2}$. Two values for $k$ are in common use: 12.5 (Canon, Nikon, and Sekonic) and 14 (Minolta, Kenko, and Pentax).

**Returns** Average luminance $L$ in $cd \cdot m^{-2}$.

**Return type** ndarray

### References

[Wika]

### Examples

```
>>> average_luminance(8, 1, 100)
8.0
```

## colour_hdri.average_illuminance

colour_hdri.**average_illuminance**(*N, t, S, c=250*)

Computes the average illuminance $E$ in $Lux$ from given relative aperture *F-Number* $N$, *Exposure Time* $t$, *ISO* arithmetic speed $S$ and *incident light calibration constant* $c$.

**Parameters**

- **N** (array_like) – Relative aperture *F-Number* $N$.
- **t** (array_like) – *Exposure Time* $t$.
- **S** (array_like) – *ISO* arithmetic speed $S$.
- **c** (numeric, optional) – *Incident light calibration constant* $c$. With a flat receptor, *ISO 2720:1974* recommends a range for $c$. of 240 to 400 with illuminance in $Lux$; a value of 250 is commonly used. With a hemispherical receptor, *ISO 2720:1974* recommends a range for $c$ of 320 to 540 with illuminance in $Lux$; in practice, values typically are between 320 (Minolta) and 340 (Sekonic).

**Returns** Average illuminance $E$ in $Lux$.

**Return type** ndarray

### References

[Wika]

### Examples

```
>>> average_illuminance(8, 1, 100)
160.0
```

## colour_hdri.luminance_to_exposure_value

colour_hdri.**luminance_to_exposure_value**(*L*, *S*, *k=12.5*)

Computes the exposure value $EV$ from given scene luminance $L$ in $cd \cdot m^{-2}$, *ISO* arithmetic speed $S$ and *reflected light calibration constant* $k$.

**Parameters**

- **L** (array_like) – Scene luminance $L$ in $cd \cdot m^{-2}$.
- **S** (array_like) – *ISO* arithmetic speed $S$.
- **k** (numeric, optional) – *Reflected light calibration constant* $k$. *ISO 2720:1974* recommends a range for $k$ of 10.6 to 13.4 with luminance in $cd \cdot m^{-2}$. Two values for $k$ are in common use: 12.5 (Canon, Nikon, and Sekonic) and 14 (Minolta, Kenko, and Pentax).

**Returns** Exposure value $EV$.

**Return type** ndarray

### Notes

- The exposure value $EV$ indicates a combination of camera settings rather than the focal plane exposure, i.e. luminous exposure, photometric exposure, $H$. The focal plane exposure is time-integrated illuminance.

### References

[Wika]

### Examples

```
>>> luminance_to_exposure_value(0.125, 100)
0.0
```

## colour_hdri.illuminance_to_exposure_value

colour_hdri.**illuminance_to_exposure_value**(*E*, *S*, *c=250*)

Computes the exposure value $EV$ from given scene illuminance $E$ in $Lux$, *ISO* arithmetic speed $S$ and *incident light calibration constant* $c$.

**Parameters**

- **E** (array_like) – Scene illuminance $E$ in $Lux$.
- **S** (array_like) – *ISO* arithmetic speed $S$.
- **c** (numeric, optional) – *Incident light calibration constant* $c$. With a flat receptor, *ISO 2720:1974* recommends a range for $c$. of 240 to 400 with illuminance in $Lux$; a value of 250 is commonly used. With a hemispherical receptor, *ISO 2720:1974* recommends a range for $c$ of 320 to 540 with illuminance in $Lux$; in practice, values typically are between 320 (Minolta) and 340 (Sekonic).

**Returns** Exposure value $EV$.

**Return type** ndarray

#### Notes

- The exposure value $EV$ indicates a combination of camera settings rather than the focal plane exposure, i.e. luminous exposure, photometric exposure, $H$. The focal plane exposure is time-integrated illuminance.

#### References

[Wika]

#### Examples

```
>>> illuminance_to_exposure_value(2.5, 100)
0.0
```

### 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. ])
```

## Digital Still Camera Exposure

`colour_hdri`

| | |
|---|---|
| `focal_plane_exposure`(L, A, t, F, i, H_f[, ...]) | Computes the focal plane exposure $H$ in lux-seconds ($lx.s$). |
| `arithmetic_mean_focal_plane_exposure`(L_a, A, t) | Computes the arithmetic mean focal plane exposure $H_a$ for a camera focused on infinity, $H_f << H$, $T = 9/10$, $\theta = 10°$ and $f_v = 98/100$. |
| `saturation_based_speed_focal_plane_exposure`(L, ...) | Computes the Saturation-Based Speed (SBS) focal plane exposure $H_{SBS}$ in lux-seconds ($lx.s$). |
| `exposure_index_values`(H_a) | Computes the exposure index values $I_{EI}$ from given focal plane exposure $H_a$. |
| `exposure_value_100`(N, t, S) | Computes the exposure value $EV100$ from given relative aperture *F-Number* $N$, *Exposure Time* $t$ and *ISO* arithmetic speed $S$. |

Continued on next page

Table 4 – continued from previous page

| | |
|---|---|
| photometric_exposure_scale_factor_Lagarde2014(EV100) | Converts the exposure value $EV100$ to photometric exposure scale factor using *Lagarde and de Rousiers (2014)* formulation derived from the *ISO 12232:2006 Saturation Based Sensitivity* (SBS) recommendation. |

### colour_hdri.focal_plane_exposure

`colour_hdri.`**`focal_plane_exposure`**(*L, A, t, F, i, H_f, T=0.9, f_v=0.98, theta=10*)

Computes the focal plane exposure $H$ in lux-seconds ($lx.s$).

**Parameters**

- **L** (array_like) – Scene luminance $L$, expressed in $cd/m^2$.
- **A** (array_like) – Lens *F-Number* $A$.
- **t** (array_like) – *Exposure Time* $t$, expressed in seconds.
- **F** (array_like) – Lens focal length $F$, expressed in meters.
- **i** (array_like) – Image distance $i$, expressed in meters.
- **H_f** (array_like) – Focal plane flare exposure $H_f$, expressed in lux-seconds ($lx.s$).
- **T** (array_like, optional) – Transmission factor of the lens $T$.
- **f_v** (array_like, optional) – Vignetting factor $f_v$.
- **theta** (array_like, optional) – Angle of image point off axis $\theta$.

**Returns** Focal plane exposure $H$ in lux-seconds ($lx.s$).

**Return type** ndarray

#### Notes

- Focal plane exposure is also named luminous exposure or photometric exposure and is time-integrated illuminance.
- Object distance $o$, focal length $F$, and image distance $i$ are related by the thin-lens equation: $\frac{1}{f}=\frac{1}{o}+\frac{1}{i}$
- This method ignores the *ISO* arithmetic speed $S$ and is not concerned with determining an appropriate minimum or maximum exposure level.

#### References

[ISO06]

### Examples

```
>>> focal_plane_exposure(4000, 8, 1 / 250, 50 / 1000, 50 / 1000, 0.0015)
...
0.1643937...
```

## colour_hdri.arithmetic_mean_focal_plane_exposure

colour_hdri.**arithmetic_mean_focal_plane_exposure**($L\_a$, $A$, $t$)

Computes the arithmetic mean focal plane exposure $H_a$ for a camera focused on infinity, $H_f << H$, $T = 9/10$, $\theta = 10°$ and $f_v = 98/100$.

**Parameters**

- **L_a** (array_like) – Arithmetic scene luminance $L_a$, expressed in $cd/m^2$.
- **A** (array_like) – Lens *F-Number* $A$.
- **t** (array_like) – *Exposure Time* $t$, expressed in seconds.

**Returns** Focal plane exposure $H_a$.

**Return type** ndarray

### Notes

- Focal plane exposure is also named luminous exposure or photometric exposure and is time-integrated illuminance.
- Object distance $o$, focal length $F$, and image distance $i$ are related by the thin-lens equation: $\frac{1}{f} = \frac{1}{o} + \frac{1}{i}$
- This method ignores the *ISO* arithmetic speed $S$ and is not concerned with determining an appropriate minimum or maximum exposure level.

### References

[ISO06]

### Examples

```
>>> arithmetic_mean_focal_plane_exposure(4000, 8, 1 / 250)
...
0.1628937...
```

## colour_hdri.saturation_based_speed_focal_plane_exposure

colour_hdri.**saturation_based_speed_focal_plane_exposure**(*L, A, t, S, F=0.05, i=0.050505050505050504, H_f=0, T=0.9, f_v=0.98, theta=10*)

Computes the Saturation-Based Speed (SBS) focal plane exposure $H_{SBS}$ in lux-seconds ($lx.s$).

The model implemented by this definition is appropriate to simulate a physical camera in an offline or realtime renderer.

**Parameters**

- **L** (array_like) – Scene luminance $L$, expressed in $cd/m^2$.
- **A** (array_like) – Lens *F-Number* $A$.
- **t** (array_like) – *Exposure Time* $t$, expressed in seconds.
- **S** (array_like) – *ISO* arithmetic speed $S$.
- **F** (array_like) – Lens focal length $F$, expressed in meters.
- **i** (array_like) – Image distance $i$, expressed in meters.
- **H_f** (array_like) – Focal plane flare exposure $H_f$, expressed in lux-seconds $(lx.s)$.
- **T** (array_like, optional) – Transmission factor of the lens $T$.
- **f_v** (array_like, optional) – Vignetting factor $f_v$.
- **theta** (array_like, optional) – Angle of image point off axis $\theta$.

**Returns** Saturation-Based Speed focal plane exposure $H_{SBS}$ in lux-seconds $(lx.s)$.

**Return type** ndarray

## Notes

- Focal plane exposure is also named luminous exposure or photometric exposure and is time-integrated illuminance.
- Object distance $o$, focal length $F$, and image distance $i$ are related by the thin-lens equation: $\frac{1}{f}=\frac{1}{o}+\frac{1}{i}$
- The image distance default value is that of an object located at 5m and imaged with a 50mm lens.
- The saturation based speed, $S_{sat}$, of an electronic still picture camera is defined as: $S_{sat}=\frac{78}{H_{sat}}$ where $H_{sat}$ is the minimum focal plane exposure, expressed in lux-seconds $(lx.s)$, that produces the maximum valid (not clipped or bloomed) camera output signal. This provides $1/2$ "stop" of headroom (41% additional headroom) for specular highlights above the signal level that would be obtained from a theoretical 100% reflectance object in the scene, so that a theoretical 141% reflectance object in the scene would produce a focal plane exposure of $H_{sat}$.
- The focal plane exposure $H_{SBS}$ computed by this definition is almost equal to that given by scene luminance $L$ scaled with the output of `colour_hdri.photometric_exposure_scale_factor_Lagarde2014()` definition.

## References

[ISO06]

### Examples

```
>>> saturation_based_speed_focal_plane_exposure(
...     4000, 8, 1 / 250, 400, 50 / 1000, 50 / 1000, 0.0015)
0.8430446...
```

## colour_hdri.exposure_index_values

colour_hdri.**exposure_index_values**(*H_a*)

Computes the exposure index values $I_{EI}$ from given focal plane exposure $H_a$.

**Parameters** **H_a** (array_like) – Focal plane exposure $H_a$.

**Returns** Exposure index values $I_{EI}$.

**Return type** ndarray

### References

[ISO06]

### Examples

```
>>> exposure_index_values(0.1628937086212269)
61.3897251...
```

## colour_hdri.exposure_value_100

colour_hdri.**exposure_value_100**(*N*, *t*, *S*)

Computes the exposure value $EV100$ from given relative aperture *F-Number* $N$, *Exposure Time* $t$ and *ISO* arithmetic speed $S$.

**Parameters**

- **N** (array_like) – Relative aperture *F-Number* $N$.
- **t** (array_like) – *Exposure Time* $t$.
- **S** (array_like) – *ISO* arithmetic speed $S$.

**Returns** Exposure value $EV100$.

**Return type** ndarray

### References

[ISO06], [LdR14]

### Notes

- The underlying implementation uses the `colour_hdri.luminance_to_exposure_value()` and `colour_hdri.average_luminance()` definitions with same fixed value for the *reflected light calibration constant* $k$ which cancels its scaling effect and produces a value equal to $log_2(\frac{N^2}{t}) - log_2(\frac{S}{100})$ as given in [LdR14].

### Examples

```
>>> exposure_value_100(8, 1 / 250, 400)
11.9657842...
```

## colour_hdri.photometric_exposure_scale_factor_Lagarde2014

`colour_hdri.`**`photometric_exposure_scale_factor_Lagarde2014`**(*EV100, T=0.9, f_v=0.98, theta=10*)

Converts the exposure value $EV100$ to photometric exposure scale factor using *Lagarde and de Rousiers (2014)* formulation derived from the *ISO 12232:2006 Saturation Based Sensitivity* (SBS) recommendation.

The model implemented by this definition is appropriate to simulate a physical camera in an offline or realtime renderer.

**Parameters**

- **T** (`array_like, optional`) – Exposure value $EV100$.
- **T** – Transmission factor of the lens $T$.
- **f_v** (`array_like, optional`) – Vignetting factor $f_v$.
- **theta** (`array_like, optional`) – Angle of image point off axis $\theta$.

**Returns** Photometric exposure in lux-seconds ($lx.s$).

**Return type** ndarray

### Notes

- The saturation based speed, $S_{sat}$, of an electronic still picture camera is defined as: $S_{sat} = \frac{78}{H_{sat}}$ where $H_{sat}$ is the minimum focal plane exposure, expressed in lux-seconds ($lx.s$), that produces the maximum valid (not clipped or bloomed) camera output signal. This provides $1/2$ "stop" of headroom (41% additional headroom) for specular highlights above the signal level that would be obtained from a theoretical 100% reflectance object in the scene, so that a theoretical 141% reflectance object in the scene would produce a focal plane exposure of $H_{sat}$.
- Scene luminance $L$ scaled with the photometric exposure value computed by this definition is almost equal to that given by the `colour_hdri.saturation_based_speed_focal_plane_exposure()` definition.

#### References

[ISO06], [LdR14]

#### Examples

```
>>> EV100 = exposure_value_100(8, 1 / 250, 400)
>>> H = photometric_exposure_scale_factor_Lagarde2014(EV100)
>>> print(H)
0.0002088...
>>> H * 4000
0.8353523...
```

## HDRI / Radiance Image Generation



### Generation

colour_hdri

| | |
|---|---|
| image_stack_to_radiance_image(image_stack[, ...]) | 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_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

| | |
|---|---|
| normal_distribution_function(a[, mu, sigma]) | Returns given array weighted by a normal distribution function. |
| hat_function(a) | Returns given array weighted by a hat function. |
| weighting_function_Debevec1997(a[, ...]) | Returns given array weighted by *Debevec (1997)* 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

`colour_hdri`

| | |
|---|---|
| `xy_to_camera_neutral`(xy, ...) | Converts given *xy* white balance chromaticity coordinates to *Camera Neutral* coordinates. |
| `camera_neutral_to_xy`(camera_neutral, ... [, ...]) | Converts given *Camera Neutral* coordinates to *xy* white balance chromaticity coordinates. |
| `XYZ_to_camera_space_matrix`(xy, ...) | Returns the *CIE XYZ* to *Camera Space* matrix for given *xy* white balance chromaticity coordinates. |
| `camera_space_to_XYZ_matrix`(xy, ... [, ...]) | Returns the *Camera Space* to *CIE XYZ* matrix for given *xy* 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=2.220446049250313e-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,
```

(continues on next page)

(continued from previous page)

```
...     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.datasets.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]),
```

(continues on next page)

(continued from previous page)

```
...     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='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.datasets.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`

| | |
|---|---|
| `camera_space_to_RGB`(RGB, ...) | Converts given *RGB* array from *camera space* to given *RGB* colourspace. |
| `camera_space_to_sRGB`(RGB, M_XYZ_to_camera_space) | Converts given *RGB* array from *camera space* to *sRGB* 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.
- **M_XYZ_to_camera_space** (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

`colour_hdri.plotting`

| | |
|---|---|
| `plot_radiance_image_strip`(image[, count, ...]) | Plots given HDRI / radiance image as strip of images of varying exposure. |

##### colour_hdri.plotting.plot_radiance_image_strip

`colour_hdri.plotting.`**`plot_radiance_image_strip`**(*image*, *count=5*, *ev_steps=-2*, *cctf_encoding=<function eotf_inverse_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.
- **cctf_encoding** (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 and axes.

**Return type** tuple

### Tonemapping Operators

colour_hdri.plotting

| | |
|---|---|
| plot_tonemapping_operator_image(image, ...) | Plots given tonemapped image with superimposed luminance mapping function. |

#### colour_hdri.plotting.plot_tonemapping_operator_image

colour_hdri.plotting.**plot_tonemapping_operator_image**(*image*, *luminance_function*, *log_scale=False*, *cctf_encoding=<function eotf_inverse_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.
- **cctf_encoding** (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** Current figure and axes.

**Return type** tuple

### Image Processing



### Adobe DNG SDK

#### Raw Files

colour_hdri

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

Continued on next page

Table 11 – continued from previous page

| | |
|---|---|
| RAW_D_CONVERSION_ARGUMENTS | Arguments for the command line raw conversion application for demosaiced linear *tiff* 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 = 'dcraw'**
Command line raw conversion application, usually Dave Coffin's *dcraw*.

RAW_CONVERTER : unicode

### colour_hdri.RAW_CONVERSION_ARGUMENTS

colour_hdri.**RAW_CONVERSION_ARGUMENTS = '-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 = '-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

| | |
|---|---|
| convert_dng_files_to_intermediate_files(…) | Converts given *dng* files to intermediate *tiff* files using given output directory. |
| DNG_CONVERTER | |
| DNG_CONVERSION_ARGUMENTS | Arguments for the command line *dng* conversion application. |
| DNG_EXIF_TAGS_BINDING | Exif tags binding for a *dng* file. |
| read_dng_files_exif_tags(dng_files[, …]) | Reads given *dng* 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 = None`

### colour_hdri.DNG_CONVERSION_ARGUMENTS

`colour_hdri.DNG_CONVERSION_ARGUMENTS = '-cr7.1 -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({'EXIF': CaseInsensitiveMapping({'Make': (<fun`

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

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

colour_hdri

| | |
|---|---|
| highlights_recovery_blend(RGB, multipliers) | Performs highlights recovery using *Coffin (1997)* method from *dcraw*. |
| highlights_recovery_LCHab(RGB[, threshold, ...]) | Performs highlights recovery in *CIE L*C*Hab* 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 eotf_inverse_sRGB>*, *<function eotf_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 eotf_inverse_sRGB>, <function eotf_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`

| | |
|---|---|
| `samples_Grossberg2003`(image_stack[, samples, n]) | Returns the samples for given image stack intensity histograms using *Grossberg (2003)* 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

`colour_hdri`

| | |
|---|---|
| `tonemapping_operator_simple`(RGB) | Performs given *RGB* 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

| | |
|---|---|
| tonemapping_operator_normalisation(RGB[, ...]) | Performs given *RGB* 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 eotf_inverse_sRGB>, <function eotf_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

| | |
|---|---|
| tonemapping_operator_gamma(RGB[, gamma, EV]) | Performs given *RGB* 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) – $\gamma$ 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

| | |
|---|---|
| tonemapping_operator_logarithmic(RGB[, q, ...]) | Performs given *RGB* array tonemapping using the logarithmic method. |
| tonemapping_operator_exponential(RGB[, q, ...]) | Performs given *RGB* array tonemapping using the exponential method. |
| tonemapping_operator_logarithmic_mapping(RGB) | Performs given *RGB* array tonemapping using the logarithmic mapping method. |
| tonemapping_operator_exponentiation_mapping(RGB) | Performs given *RGB* array tonemapping using the exponentiation mapping method. |
| tonemapping_operator_Schlick1994(RGB[, p, ...]) | Performs given *RGB* array tonemapping using *Schlick (1994)* method. |

Continued on next page

Table 19 – continued from previous page

| | |
|---|---|
| tonemapping_operator_Tumblin1999(RGB[, ...]) | Performs given *RGB* array tonemapping using *Tumblin, Hodgins and Guenter (1999)* method. |
| tonemapping_operator_Reinhard2004(RGB[, f, ...]) | Performs given *RGB* array tonemapping using *Reinhard and Devlin (2004)* method. |
| tonemapping_operator_filmic(RGB[, ...]) | Performs given *RGB* array tonemapping using *Habble (2010)* 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 eotf_inverse_sRGB>, <function eotf_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 eotf_inverse_sRGB>, <function eotf_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 eotf_inverse_sRGB>, <function eotf_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 eotf_inverse_sRGB>*, *<function eotf_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 eotf_inverse_sRGB>, <function eotf_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 eotf_inverse_sRGB>, <function eotf_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 eotf_inverse_sRGB>, <function eotf_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 *Habble (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`

| | |
|---|---|
| `tonemapping_operator_logarithmic_mapping`(RGB) | Performs given *RGB* array tonemapping using the logarithmic mapping method. |

## Exponential

`colour_hdri`

| | |
|---|---|
| `tonemapping_operator_exponential`(RGB[, q, ...]) | Performs given *RGB* array tonemapping using the exponential method. |

## Exponentiation Mapping

`colour_hdri`

| | |
|---|---|
| `tonemapping_operator_exponentiation_mapping`(RGB) | Performs given *RGB* array tonemapping using the exponentiation mapping method. |
| `tonemapping_operator_Schlick1994`(RGB[, p, ...]) | Performs given *RGB* array tonemapping using *Schlick (1994)* method. |
| `tonemapping_operator_Tumblin1999`(RGB[, ...]) | Performs given *RGB* array tonemapping using *Tumblin, Hodgins and Guenter (1999)* method. |
| `tonemapping_operator_Reinhard2004`(RGB[, f, ...]) | Performs given *RGB* array tonemapping using *Reinhard and Devlin (2004)* method. |
| `tonemapping_operator_filmic`(RGB[, ...]) | Performs given *RGB* array tonemapping using *Habble (2010)* method. |

### Schlick (1994)

`colour_hdri`

| | |
|---|---|
| tonemapping_operator_Schlick1994(RGB[, p, ...]) | Performs given *RGB* array tonemapping using *Schlick (1994)* method. |

### Tumblin, Hodgins and Guenter (1999)

`colour_hdri`

| | |
|---|---|
| tonemapping_operator_Tumblin1999(RGB[, ...]) | Performs given *RGB* array tonemapping using *Tumblin, Hodgins and Guenter (1999)* method. |

### Reinhard and Devlin (2004)

`colour_hdri`

| | |
|---|---|
| tonemapping_operator_Reinhard2004(RGB[, f, ...]) | Performs given *RGB* array tonemapping using *Reinhard and Devlin (2004)* method. |

### Habble (2010) - Filmic

`colour_hdri`

| | |
|---|---|
| tonemapping_operator_filmic(RGB[, ...]) | Performs given *RGB* array tonemapping using *Habble (2010)* method. |

### Utilities



### Common

`colour_hdri`

| | |
|---|---|
| vivification() | Implements supports for vivification of the underlying dict like data-structure, magical! |
| vivified_to_dict(vivified) | Converts given vivified data-structure to dictionary. |
| path_exists(path) | Returns if given path exists. |
| filter_files(directory, extensions) | 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 *ExifTool*. |
| 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 *exiftool*. |
| 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 = '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__**()

Initialize self. See help(type(self)) for accurate signature.

### Methods

| | |
|---|---|
| __init__ | Initialize self. |
| count(value) | |
| index(value, [start, [stop]]) | Raises ValueError if the value is not present. |

### Attributes

| | |
|---|---|
| group | Alias for field number 0 |
| identifier | Alias for field number 3 |
| name | Alias for field number 1 |
| value | Alias for field number 2 |

## 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=<class '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=<class '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=<class '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 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**

Bases: colour_hdri.utilities.image.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.

Create new instance: Metadata(f_number, exposure_time, iso, black_level, white_level, white_balance_multipliers)

### colour_hdri.Image

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

Bases: `object`

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**()

**property data**

Property for **self._data** private attribute.

**Returns** self._data.

**Return type** unicode

**property metadata**

Property for **self._metadata** private attribute.

**Returns** self._metadata.

**Return type** unicode

**property path**

Property for **self._path** private attribute.

**Returns** self._path.

**Return type** unicode

**read_data**(*cctf_decoding=None*)

Reads image pixel data at `Image.path` attribute.

**Parameters** **cctf_decoding** (`object`, `optional`) – Decoding colour component transfer function (Decoding CCTF) or electro-optical transfer function (EOTF / EOCF).

**Returns** Image pixel data.

**Return type** ndarray

**read_metadata**()

Reads image relevant exif metadata at `Image.path` attribute.

**Returns** Image relevant exif metadata.

**Return type** *Metadata*

### colour_hdri.ImageStack

**class** colour_hdri.**ImageStack**

Bases: collections.abc.MutableSequence

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**()

**static from_files**(*image_files*, *cctf_decoding=None*)

Returns a colour_hdri.ImageStack instance with given image files.

**Parameters**

- **image_files** (array_like) – Image files.
- **cctf_decoding** (object, optional) – Decoding colour component transfer function (Decoding CCTF) or electro-optical transfer function (EOTF / EOCF).

**Returns**

**Return type** *ImageStack*

**insert**(*index*, *value*)

Reimplements the MutableSequence.insert() method.

**Parameters**

- **index** (int) – Item index.
- **value** (object) – Item value.

**sort**(*key=None*)

Sorts the underlying data structure.

**Parameters key** (callable) – Function of one argument that is used to extract a comparison key from each 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 1.3.2 Examples

Various usage examples are available from the examples directory.

CHAPTER

# FOUR

## 1.4 CONTRIBUTING

If you would like to contribute to Colour - HDRI, please refer to the following Contributing guide for Colour.

# CHAPTER FIVE

## 1.5 BIBLIOGRAPHY

The bibliography is available in the repository in BibTeX format.

# CHAPTER SIX

## 1.6 CODE OF CONDUCT

The *Code of Conduct*, adapted from the Contributor Covenant 1.4, is available on the Code of Conduct page.

CHAPTER

SEVEN

# 1.7 ABOUT

**Colour - HDRI** by Colour Developers
Copyright © 2015-2020 – Colour Developers – colour-developers@colour-science.org
This software is released under terms of New BSD License:
https://opensource.org/licenses/BSD-3-Clause
https://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\textdollardcoffin/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, November 2003. URL: http://ieeexplore.ieee.org/document/1240119/, doi:10.1109/TPAMI.2003.1240119.

[Hab10a] John Habble. Filmic Tonemapping Operators. 2010. URL: http://filmicgames.com/archives/75.

[Hab10b] John Habble. Uncharted 2: HDR Lighting. 2010. URL: http://www.slideshare.net/ozlael/hable-john-uncharted2-hdr-lighting.

[ISO06] ISO. INTERNATIONAL STANDARD ISO12232-2006 - Photography - Digital still cameras - Determination of exposure index, ISO speed ratings, standard output sensitivity, and recommended exposure index. 2006.

[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.

[LdR14] Sébastian Lagarde and Charles de Rousiers. Moving Frostbite to Physically Based Rendering 3.0. *Siggraph 2014*, pages 119, 2014.

[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, January 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, January 1999. URL: http://portal.acm.org/citation.cfm?doid=300776.300783, doi:10.1145/300776.300783.

[VD09] Kuntee 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.

# INDEX

## Symbols



## A



## C



## D



## E



## F



## G



## H



## I



## L