pyunraw Documentation

Release 1.2

Vincent Vande Vyvre

Mar 05, 2019

Contents

pyunraw is a Python 3 implementation of dcraw, a C application to decode raw digital photos.

It is provided in two modules, a library in CPython including dcraw and an api in Python.

pyunraw is distributed under the GPL version 3 license.

Contents:

CHAPTER 1

API documentation

1.1 pyunraw

The module pyunraw contains three attributes version and one class PyUnraw.

Attributes

- *version*
- *__version__*
- *__dcraw_version__*

Documentation

version
A tuple containing the three components of the version number: major, minor, micro.

__version__
The version of the module as a string (major.minor.micro).

__dcraw_version__
A tuple containing the two components of the version number: major, minor.

class pyunraw.**PyUnraw**

Instance Attributes

- *filename*
- *out_filename*
- *data*
- *is_raw*
- *profile*
- *has_preview*

Instance Methods

- *__init__(filename="")*
- *apply_new_profile(profile)*
- *clean_artifacts(passes)*
- *find_white_balance(x=0, y=0, w=0, h=0)*
- *fix_dead_pixels(fname="")*
- *get_file_properties()*
- *get_preview(dest="")*
- *get_unraw_parameters()*
- *ignore_white_balance()*
- *quick_unraw(dest="")*
- *read_profile(fname)*
- *reset_profile()*
- *rotate_fuji(rotate)*
- *save_profile(fname, pickle_=False)*
- *set_black_layer(filename=None)*
- *set_brightness(factor)*
- *set_color_space(space)*
- *set_darkness(level)*
- *set_gamma_curve(power, toe_sloop=0)*
- *set_highlights(level)*
- *set_ICC_profile(profile, output="")*
- *set_interpolation_method(method)*
- *set_noise_threshold(threshold)*
- *set_orientation(orientation)*
- *set_output_format(use_tiff=True)*
- *set_saturation(level)*
- *set_source_file(path)*
- *set_white_balance(red, green, blue, green1=0)*
- *stretch_red_blue_layers(red, blue)*
- *unraw(index=0, dest="")*
- *unraw_dark(filename, dest="")*
- *validate(filename)*
- *write_sixteen_bits(apply_, linear=False)*

Documentation

filename
The complete path of the raw file.

out_filename
The complete path of the decoded image file.

data
A dictionnary which contains the properties of the raw image.

is_raw
True if the file can be decoded with dcraw, otherwise False.

profile
A dictionnary which contains the user's decoding parameters.

has_preview
True if the raw file contains at least one preview.

Methods

__init__ (*filename=""*)
Initialise the class PyUnraw with the raw file name.

Argument:

- *filename* The path/name.ext of the raw file, optionnal

Raises:

- *ValueError* if the file doesn't exists
- *TypeError* if the file is clearly identified as a non-raw format, (jpeg, tiff, ppm, ...). See: `validate(filename)`

apply_new_profile (*profile*)
Apply a decoding parameters profile.

Argument:

- *profile* A dictionnary with the decoding custom parameters

clean_artifacts (*passes*)
Set the number of passes for the artifacts cleaning.

Argument:

- *passes* The number of passes

find_white_balance (*x=0*, *y=0*, *w=0*, *h=0*)
Use a rectangular area to compute the white balance. The parameters calculated will be applyed automatically. Use PyUnraw.find_white_balance() to cancel this method.

Arguments:

- *x* The left side of the area
- *y* The top side of the area
- *w* The width of the area
- *h* The height of the area

fix_dead_pixels (*fname=""*)
Fix the dead pixels given with the file fname.

Argument:

- *fname* Absolute path of the file

get_file_properties()
Return the image properties as a dictionnary, same as the attribute PyUnraw.data.

get_preview(*dest=""*)
Extract a preview embeded into the raw file.

The attribute PyUnraw.has_preview define if there's at least one preview in the raw file. After writing the preview on disk, the attribute PyUnraw.preview define the file name or None if the process failed.

If there's more than one preview into the file, dcraw extract the first one found. If you want to choose the preview consider py3exiv2.

Argument:

- *dest* Absolute path of destination file. If not provided, the name of the preview is build with the name of the file source and the suffix *_thumb* i.e. DSC0786.NEF –> DSC0786_thumb.jpg

get_unraw_parameters()
Return all the parameters (default and custom) used for the demosaication as python dictionnary.

ignore_white_balance()
Use a fixed white level, ignoring the camera parameters.

quick_unraw(*dest=""*)
Decode a half-size middle-quality of the raw image.

This only usefull to see a preview of the image when the (old) camera doesn't include a thumbnail into the file.

If some custom parameters are already fixed, the profile is cleared and will be restaured after the demosaication.

New in version 1.2

Argument:

- *dest* Destination file name whitout extension

Return the path of the file created

read_profile(*fname*)
Read a profile from a file.

The file must be pickled or write as a json file. The new profile is applied immediately.

Argument:

- *fname* Absolute path of the file

reset_profile()
Reset all parameters to the default values.

The profile dictionnary will be emptyed.

rotate_fuji(*rotate*)
Rotate the decoded image to 45°.

For Fuji Super CCD cameras where the image are tilted 45 degrees.

Argument:

- *rotate* Boolean: True, apply the rotation (default) False, keep the image tilted 45 degrees

save_profile(*fname*, *pickle_=False*)
Save the profile on disk.

Arguments:

- *fname* Absolute path of the file to save the profile

- *pickle_* Boolean: If False (default) the profile will be saved as a json file, if True it will be pickled

set_black_layer(*filename=None*)
Set the name of the dark frame.

If the file is already a .pgm file, the file is checked for validity. If the file is yet a raw file it will be converted into a valid .pgm

Argument:

- *filename* Absolute path of the file

Raises:

- *PgmSizeError* if the size of the frame is not the same size of the source file
- *PgmEncodingError* if the .pgm is not encoded in 16 bits

set_brightness(*factor*)
Divide the white level by this number, 1.0 by default.

Argument:

- *factor* The value to divide the white level

set_color_space(*space*)
Define the output color space.

This cancel a previously ICC profile defined. Use PyUnraw.set_color_space(1) to reset to default value (sRGB D65).

Argument:

- ***space* Index of the color space:**
  - 0 : Raw color (unique to each camera)
  - 1 : sRGB D65 (default)
  - 2 : Adobe RGB (1998) D65
  - 3 : Wide Gamut RGB D65
  - 4 : Kodak ProPhoto RGB D65
  - 5 : XYZ
  - 6 : ACES

set_darkness(*level*)
Set the level of darkness.

Argument:

- *level* The level of darkness

set_gamma_curve(*power*, *toe_sloop=0*)
Set the gamma curve values.

Use PyUnraw.set_gamma_curve(0) to reset to the default values BT.709 (2.222 4.5).

Arguments:

- *power* The value of the power
- *toe_sloop* The value of the sloop

set_highlights(*level*)
Set the highlights level.

Use PyUnraw.set_highlights(0) to reset the parameter.

Argument:

- ***level* The level of highlights, the value must be in range(10):**
  - 0 : Clip all highlights to solid white (default)
  - 1 : Leave highlights unclipped in various shades of pink
  - 2 : Blend clipped and unclipped values together for a gradual fade to white
  - 3~9 : Reconstruct highlights. Low numbers favor whites; high numbers favor colors. Try 5 as a compromise

set_ICC_profile(*profile*, *output=""*)
Apply an ICC profile to the image.

The ICC profile can be embeded into the camera, that can be checked with PyUnraw.data["embeded_icc"].

Arguments:

- ***profile* The ICC profile:**
  - False : Don't apply an ICC profile
  - True : Apply the embeded ICC profile
  - Absolute path of "camera.icm"
- *output* "path/camera.icm": File name to write the embeded ICC profile

set_interpolation_method(*method*)
Set the interpolation method.

Argument:

- ***method* The index of the method:**
  - 0 : Use high-speed, low-quality bilinear interpolation
  - 1 : Use Variable Number of Gradients (VNG) interpolation
  - 2 : Use Patterned Pixel Grouping (PPG) interpolation
  - 3 : Use Adaptive Homogeneity-Directed (AHD) interpolation (default)
  - 4 : Interpolate RGB as four colors
  - 5 : Show the raw data as a grayscale image with no interpolation
  - 6 : Same as #5, but with the original unscaled pixel values
  - 7 : Same as #6, but masked pixels are not cropped
  - 8 : Output a half-size color image. Twice as fast as #0

set_noise_threshold(*threshold*)
Fix the noise threshold limit.

The best value should be somewhere between 100 and 1000. Use PyUnraw.set_noise_threshold(0) to reset the parameter.

Argument:

- *threshold* The value of the threshold limit

set_orientation(*orientation*)
Set the orientation of the decoded image.

Argument:

- ***orientation*** **Integer which define the orientation. The integer must be in (-1, 0, 3, 5, 6, 90, 180, 270):**
  - -1 : apply the orientation found into the metadata
  - 0 : no rotation (Default)
  - 3 or 180 : rotate 180°
  - 5 or 270 : rotate 270°
  - 6 or 90 : rotate 90°

set_output_format(*use_tiff=True*)
Use the tiff format with metadata instead of ppm, pgm or pam.

Argument:

- *use_tiff* Boolean: Use tiff format if True (default), otherwise, depending of the source file, the format pgm, ppm or pam will be choosed

set_saturation(*level*)
Set the level of saturation.

Argument:

- *level* The level of saturation

set_source_file(*path*)
Set a new RAW source file.

This doesn't reset the unrawing parameters previously fixed. This method is intended to unraw a list of raw files with the same parameters.

Argument:

- *filename* Absolute path of RAW source file

set_white_balance(*red*, *green*, *blue*, *green1=0*)
Set the value of the white balance.

To reset the white balance to the default values use 0 for each value.

Arguments:

- *red* The red value
- *green* The green value
- *blue* The blue value
- *green1* The second green value, if 0 the value of green will be used

stretch_red_blue_layers(*red*, *blue=0*)
Fix the chromatic aberration. Stretch the raw red and blue layers by the given factors, typically 0.999 to 1.001. Use PyUnraw.stretch_red_blue_layers(0) to reset to default values.

Arguments:

- *red* The red factor
- *blue* The blue factor

unraw(*index=0*, *dest=""*)
Run the demosaication process.

Arguments:

- *index* The index of the image or "all" if there's more than one image into the file
- *dest* The absolute file name for the image decoded. If a file with the same name already exists, it will be overwritten

Raises:

- *IndexError* if index >= raw.data["image_count"]

unraw_dark(*filename*, *dest=""*)
Decode a raw image as a dark.

Arguments:

- *filename* Absolute path of raw file
- *dest* Absolute path of destination, if not provided the pgm file will be named wit the raw file name

Returns the path of the pgm file or None if failed

validate(*filename*)
Returns True if the file is a valid RAW image supported by dcraw.

Argument:

- *path* Absolute path of the file

write_sixteen_bits(*apply_*, *linear=False*)
Write sixteen bits per sample.

Arguments:

- *apply_* Boolean: apply 16 bits if True
- *linear* Boolean: apply 16 bits linear if True

CHAPTER 2

Tutorial

This tutorial is meant to give you a quick overview of what *pyunraw* allows you to do. You can just read it through or follow it interactively, in which case you will need to have *pyunraw* installed. It doesn't cover all the possibilities offered by *pyunraw*, only a basic subset of them. For complete reference, see the *API documentation*.

First we import *pyunraw*:

```
>>> import pyunraw
```

We can read the version of the module:

```
>>> pyunraw.version
(1, 0, 0)
```

With *pyunraw.__version__* we will get the version as string and, with *pyunraw.__dcraw_version__*, this is the version of *dcraw* which is returned as tuple.

We then instanciate *PyUnraw* with an image and read his properties:

```
>>> raw = pyunraw.PyUnraw("_DSC2849.NEF")
>>> for key, value in raw.data.items():
...     print(" %-20s%s" %(key, str(value)))
...
 camera              Nikon
 model               D5000
 time                1507263878
 ISO                 250
 image_count         1
 owner
 shutter             30.0
 aperture            9.0
 focal               62.0
 embeded_icc         0
 white_balance       (2.00850820541, 0.92519360780, 1.0760494470596313, 0.0)
 multipliers         (421.0, 256.0, 496.0, 256.0)
 image_size          (4310, 2868)
```

(continues on next page)

(continued from previous page)

```
output_size          (4310, 2868)
color_count          3
bits_per_sample      12
preview              1
date_time            2017-10-06 06:24:38
color_space          1
icc_profil           None
white_adjust         None
template_wb          None
color_matrice        True
```

The key *image_count* is the number of raw image found into the file, in this case: 1.

We can check if the file is a valid raw file:

```
>>> raw.is_raw
True
```

Before we adjust some demosaicing parameters, we need to see the image decoded with the parameters provided by the camera:

```
>>> raw.unraw(0, "_DSC2849_1")
Scaling with darkness 0, saturation 3840, and
multipliers 2.170906 1.000000 1.163053 1.000000
AHD interpolation...
Converting to sRGB colorspace...
>>> raw.out_filename
'_DSC2849_1.tiff'
```

The attribute *raw.out_filename* returns the name of the file created.

The default format is *Tiff*, if we prefer let *dcraw* to decide the type of the file, we have to use:

```
>>> raw.set_output_format(False)
```

After seeing the image produced, we can, now, change one or more parameters and re-run the demosaicing:

```
>>> raw.set_gamma_curve(2.4, 12.9)
>>> raw.set_interpolation_method(1)
>>> raw.unraw(0, "_DSC2849_2")
```

If we don't change the output file name, the last one will be overwritten.

Profile

The custom parameters are permanent, if we want to make a new demosaication without a parameter already fixed we have to cancel it:

```
>>> raw.set_gamma_curve(0)    # reset to default (BT.709)
>>> raw.set_interpolation_method(3)
>>> raw.unraw(0, "_DSC2849_3")
```

Now, if we want to proceed with an new raw image, we have two choices. Create a new instance of *PyUnraw* or just change the source raw file name:

```
>>> raw = pyunraw.PyUnraw("_DSC2850.NEF")
>>> # or
>>> raw.set_source_file("_DSC2850.NEF")
```

In the first case, all customs parameters are reset to the default values, in the other case, the new image can be decoded immediately with the same parameters as the last one.

pyunraw keep a demosaicing profile into a dictionnary which contains the custom parameters. We can get this profile:

```
>>> raw.profile
{'gamma': [2.4, 12.9], 'interpolation': 1}
```

The profile can be saved on disk with *raw.save_profile("profile_1.jsn")* or loaded from disk with *raw.read_profile("profile_1.jsn")*. The profile loaded will be applied immediatelly.

We can also provide a dictionnary with *raw.apply_new_profile(profile)* or reset all parameters to the default values with *raw.reset_profile()*.

After demosaication, we can get all parameters, default and custom, used for this image:

```
>>> p = raw.get_unraw_parameters()
>>> for k, v in p.items():
...     print(" %24s : %s" %(k, str(v)))
...
         White balance : (0.0, 0.0, 0.0, 0.0)
           Auto bright : 1
           Gamma curve : (0.4166666567325592, 12.899999618530273)
  Chromatic aberration : (1.0, 1.0)
           Color space : 1
           ICC embeded : 0
             File .icm : None
       Noise threshold : 0.0
            Highlights : 0
               Quality : 1
         Document mode : 0
             Half size : 0
        Four color RGB : 0
       Clean artifacts : 0
              Darkness : -1
            Saturation : -1
            Brightness : 1.0
               16 bits : 8
        16 bits linear : 0
            Embeded WB : 0
```

Workflow

pyunraw is NOT thread safe. That's means we can decode one image into a thread but we can't run several process in the same time. We have to wait the end of a thread before run the next one.

If we need to sort raw files and non-raw files, we have the method `validade(filename)`:

```
>>> # we can instanciate PyUnraw without file name
>>> raw = pyunraw.PyUnraw()
>>> for f in list_of_files:
...     if raw.validate(f):
...         # use f as raw file
```

Differences with dcraw

The default output format is Tiff, see: *set_output_format(use_tiff=True)*

The images taken in portrait orientation are not rotated, see: *set_orientation(orientation)*.

Change the access and modification times with the datetime of the image is not implemented, Python os.stats() and os.utime() can do that.

CHAPTER 3

Developers

If you are a developer and use *pyunraw* in your project, you will find here useful information.

3.1 Getting the code

pyunraw's source code is versioned with bazaar, and the main development focus (sometimes referred to as *trunk*), is hosted on Launchpad.

To get a working copy of the latest revision of the development branch, just issue the following command in a terminal:

```
bzr branch lp:pyunraw
```

3.2 Dependencies

To build *pyunraw*

- *gcc*
- *python-all-dev* (3.2)
- *liblcms2-dev* (>= 2.2-1)
- *libjpeg-dev* (>= 8c)

To use *pyunraw*

- *python 3*
- *liblcms2*
- *libjpeg*

3.3 Building and installing

Open a terminal into the folder and enter:

```
$ python3 setup.py build
```

Then you will find into */build/lib...* a file named *_pyunraw.cpython-xym...* where xy is your version of python.

In your *python3/dist-packages* folder create a subfolder *pyunraw* and copy into this folder the files *__init__.py, pyunraw.py* and the file *_pyunraw.cpython-xym...*

Now, run your python3 interactive interpreter and import pyunraw to see if there's no error.

3.4 Install from PyPI

pyunraw is available from PyPI

Use, with admin rights if necessary, the command:

```
$ pip3 install pyunraw
```

3.5 Documentation

The present documentation is generated using Sphinx from reStructuredText sources found in the doc/ directory. Invoke `make html` to (re)build the HTML documentation.

The index of the documentation will then be found under doc/_build/html/index.html.

3.6 Contributing

pyunraw is Free Software, meaning that you are encouraged to use it, modify it to suit your needs, contribute back improvements, and redistribute it.

If you want to fork it on GitHub, you will find usefull information on how to import the branch from Launchpad to GitHub here.

Bugs are tracked on Launchpad. When reporting a bug, don't forget to include the following information in the report:

- version of pyunraw
- a minimal script that reliably reproduces the issue
- a sample image file with which the bug can reliably be reproduced (or a link to)

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index