pillow_affine

Release 0.1

Philip Meier

Mar 24, 2020

CONTENTS

`pillow_affine` provides affine transformation utilities for `Pillow`. While `Pillow` includes functionality for affine transformations

```
from PIL import Image

image = Image.open(...)
image.transform(image.size, Image.AFFINE, data=None)
```

the `data` parameter is not well documented. Even if you are familiar with affine transformations, it is inconvenient to use. `pillow_affine` can help with that by providing an intuitive and convenient interface:

```
from PIL import Image
from pillow_affine import transforms

image = Image.open(...)
transform = transforms.Rotate(30.0)

transform_params = transform.extract_transform_params(image.size)
image.transform(*transform_params)
```

`pillow_affine` requires Python 3.6 or later. The code lives on GitHub and is licensed under the 3-Clause BSD License.

CHAPTER

ONE

GETTING STARTED

1.1 Installation

`pillow_affine` is a proper Python package and listed on PyPI. To install the latest stable version run

```
pip install pillow_affine
```

To install the latest unreleased version from source run

```
git clone https://github.com/pmeier/pillow_affine
cd pillow_affine
pip install .
```

1.1.1 Installation for developers

If you want to contribute to `pyimagetest` please install from source with the `[dev]` extra in order to install all required development tools.

```
git clone https://github.com/pmeier/pyimagetest
cd pyimagetest
pip install .[dev]
```

1.2 Coordinate system

Warning: `pillow_affine` uses a different, presumably more intuitive, coordinate system than `Pillow`:

Property	`Pillow`	`pillow_affine`
origin, i.e. *(0.0, 0.0)*	top-left	bottom-left
horizontal positive direction	rightwards	rightwards
vertical positive direction	downwards	upwards

1.3 Usage examples

Every affine transformation is build from 4 `ElementaryTransform` s:

- *Shear*
- *Rotate*
- *Scale*
- *Translate*

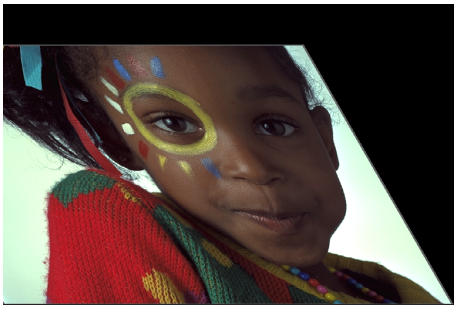
To create a more complex transformation these `ElementaryTransform` s can be chained together with a *ComposedTransform*

The following examples showcase the functionality of `pillow_affine` based on the following image:

Note: The above image can be downloaded here and is cleared for unrestricted usage.

1.3.1 Shear

```
from pillow_affine import Shear

transform1 = Shear(30.0)
transform2 = Shear(30.0, clockwise=True)
transform3 = Shear(30.0, center=(0.0, 0.0))
```

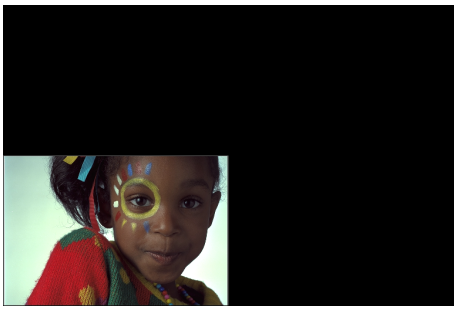


1.3.2 Rotate

```
from pillow_affine import Rotate

transform1 = Rotate(30.0)
transform2 = Rotate(30.0, clockwise=True)
transform3 = Rotate(30.0, center=(0.0, 0.0))
```

1.3.3 Scale

```
from pillow_affine import Scale

transform1 = Scale(2.0)
transform2 = Scale((0.3, 1.0))
transform3 = Scale(0.5, center=(0.0, 0.0))
```

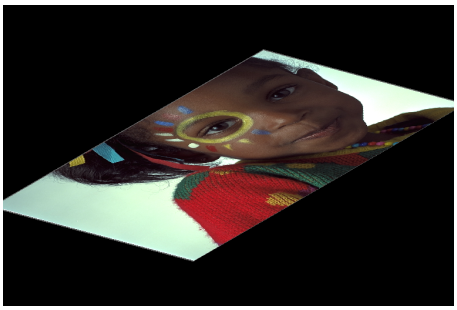


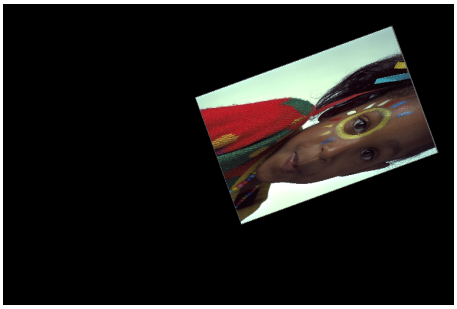
1.3.4 Translate

```
from pillow_affine import Translate

transform1 = Translate((100.0, 50.0))
transform2 = Translate((100.0, 50.0), inverse=True)
```

1.3.5 ComposedTransform

```
from pillow_affine import Shear, Rotate, Scale, Translate, ComposedTransform

transform1 = ComposedTransform(
    Shear(45.0),
    Rotate(30.0),
    Scale(0.7),
)
transform2 = ComposedTransform(
    Scale((0.3, 0.7)),
    Rotate(70.0, clockwise=True),
    Translate((50.0, 20.0))
)
```





1.3.6 expand

```
from pillow_affine import Shear

transform = Shear(30.0)
transform_params1 = transform.extract_transform_params(size)
transform_params2 = transform.extract_transform_params(size, expand=True)
```

CHAPTER

TWO

REFERENCE

2.1 pillow_affine package

2.1.1 Submodules

2.1.2 pillow_affine.matrix module

`pillow_affine.matrix.`**`shearing_matrix`**(*angle*, *clockwise=False*)

Creates an affine horizontal shearing matrix in the form

$$\mathbf{S} = \begin{pmatrix} 1 & -\sin\varphi & 0 \\ 0 & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **`angle`** (`float`) – Angle φ in degrees.
- **`clockwise`** (`bool`) – If `True`, the shearing will be performed clockwise. Defaults to `False`.

Return type `Tuple[float, float, float, float, float, float]`

Returns Parameters a, b, c, d, e, f.

`pillow_affine.matrix.`**`rotation_matrix`**(*angle*, *clockwise=False*)

Creates an affine rotation matrix in the form

$$\mathbf{R} = \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **`angle`** (`float`) – Angle φ in degrees.
- **`clockwise`** (`bool`) – If `True`, the rotation will be performed clockwise. Defaults to `False`.

Return type `Tuple[float, float, float, float, float, float]`

Returns Parameters a, b, c, d, e, f.

`pillow_affine.matrix.`**`scaling_matrix`**(*factor*)

Creates an affine scaling matrix in the form

$$\mathbf{C} = \begin{pmatrix} c_{\text{horz}} & & 0 \\ 0 & c_{\text{vert}} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$$

Parameters **factor** (Union[float, Tuple[float, float]]) – Horizontal and vertical scaling factors (c_{horz}, c_{vert}). If scalar, the same factor is used for both directions.

Return type Tuple[float, float, float, float, float, float]

Returns Parameters a, b, c, d, e, f.

pillow_affine.matrix.**translation_matrix**(*translation*, *inverse=False*)

Creates an affine scaling matrix in the form

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & t_{\text{horz}} \\ 0 & 1 & t_{\text{vert}} \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **translation** (Tuple[float, float]) – Horizontal and vertical translation. (t_{horz}, t_{vert})
- **inverse** (bool) – If True, the translation will be performed in the opposite direction. Defaults to False.

Return type Tuple[float, float, float, float, float, float]

Returns Parameters a, b, c, d, e, f.

2.1.3 pillow_affine.transforms module

class pillow_affine.transforms.**AffineTransform**

Bases: abc.ABC

ABC for all affine transformations.

extract_transform_params(*size*, *expand=False*)

Extracts the transformation parameters that need to be passed to Image.transform() for the affine transformation. An simple call might look like:

```
from PIL import Image
from pillow_affine import transforms


image = Image.open(...)
transform = transforms.Rotate(30.0)

transform_params = transform.extract_transform_params(image.size)
transformed_image = image.transform(*transform_params)
```

Parameters

- **size** (Tuple[int, int]) – Image size (width, height).
- **expand** (bool) – If True, expands the canvas to hold the complete transformed motif. Defaults to False.

Note: If you use the expand flag the motif is centered on the canvas and thus any final translation is removed.

Return type Tuple[Tuple[int, int], int, Tuple[float, float, float, float, float, float]]

Returns

size, method, and data parameters for Image.transform()

.

class pillow_affine.transforms.**Shear**(*angle*, *clockwise=False*, *center=None*)
Bases: pillow_affine.transforms.ElementaryTransform

Affine horizontal shearing transformation.

Parameters

- **angle** (float) – Shearing angle in degrees.
- **clockwise** (bool) – If True, the shearing will be performed clockwise. Defaults to False.
- **center** (Optional[Tuple[float, float]]) – Optional center of the shearing. Defaults to the center of the image.

class pillow_affine.transforms.**Rotate**(*angle*, *clockwise=False*, *center=None*)
Bases: pillow_affine.transforms.ElementaryTransform

Affine rotation transformation.

Parameters

- **angle** (float) – Rotation angle in degrees.
- **clockwise** (bool) – If True, the rotation will be performed clockwise. Defaults to False.
- **center** (Optional[Tuple[float, float]]) – Optional center of the rotation. Defaults to the center of the image.

class pillow_affine.transforms.**Scale**(*factor*, *center=None*)
Bases: pillow_affine.transforms.ElementaryTransform

Affine scaling transformation

Parameters

- **factor** (Union[float, Tuple[float, float]]) – Horizontal and vertical scaling factors. If scalar, the same factor is used for both directions.
- **center** (Optional[Tuple[float, float]]) – Optional center of the scaling. Defaults to the center of the image.

class pillow_affine.transforms.**Translate**(*translation*, *inverse=False*)
Bases: pillow_affine.transforms.ElementaryTransform

Affine translation transformation

Parameters

- **translation** (Tuple[float, float]) – Horizontal and vertical translation.
- **inverse** (bool) – If True, the translation will be performed in the opposite direction. Defaults to False.

class pillow_affine.transforms.**ComposedTransform**(**transforms*)

Bases: *pillow_affine.transforms.AffineTransform*

Composed affine transformation by chaining multiple *AffineTransform* s together. An simple example might look like:

```
from PIL import Image
from pillow_affine import transforms

image = Image.open(...)
transform = transforms.ComposedTransform(
    transforms.Rotate(30.0), transforms.Translate((50.0, 100.0))
)

transform_params = transform.extract_transform_params(image.size)
transformed_image = image.transform(*transform_params)
```

Parameters **transforms** (*AffineTransform*) – Individual *AffineTransform* s.

2.1.4 pillow_affine.utils module

pillow_affine.utils.**Coordinate**

alias of typing.Tuple

pillow_affine.utils.**Matrix**

alias of typing.Tuple

pillow_affine.utils.**matmul**(*matrix1*, *matrix2*)

Matrix product

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_2 & b_2 & c_2 \\ d_2 & e_2 & f_2 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **matrix1** (Tuple[float, float, float, float, float, float]) – Parameters a_1, b_1, c_1, d_1, e_1, f_1.
- **matrix2** (Tuple[float, float, float, float, float, float]) – Parameters a_2, b_2, c_2, d_2, e_2, f_2.

Return type Tuple[float, float, float, float, float, float]

Returns Parameters a, b, c, d, e, f.

pillow_affine.utils.**left_matmuls**(**matrices*)

Matrix product of N matrices from the left

$$\begin{pmatrix} a_N & b_N & c_N \\ d_N & e_N & f_N \\ 0 & 0 & 1 \end{pmatrix} \cdot \quad \dots \quad \cdot \begin{pmatrix} a_n & b_n & c_n \\ d_n & e_n & f_n \\ 0 & 0 & 1 \end{pmatrix} \quad \dots \quad \cdot \begin{pmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$$

Parameters ***matrices** – Parameters a_n, b_n, c_n, d_n, e_n, f_n of each matrix.

Return type Tuple[float, float, float, float, float, float]

Returns Parameters a, b, c, d, e, f.

pillow_affine.utils.**matinv**(*matrix*)
Matrix inverse

$$\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} a' & b' & c' \\ d' & e' & f' \\ 0 & 0 & 1 \end{pmatrix}$$

Parameters **matrix** (Tuple[float, float, float, float, float, float]) – Parameters a, b, c, d, e, f.

Return type Tuple[float, float, float, float, float, float]

Returns Parameters a', b', c', d', e', f'.

pillow_affine.utils.**deg2rad**(*angle_in_deg*)
Converts an angle from degrees to radians

Parameters **angle_in_deg** (float) – Angle in degrees.

Return type float

Returns Angle in radians.

pillow_affine.utils.**transform_coordinate**(*coordinate*, *matrix*)
Transforms a coordinate based on an affine matrix

$$\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}$$

Parameters

- **coordinate** (Tuple[float, float]) – Coordinate (x, y).
- **matrix** (Tuple[float, float, float, float, float, float]) – Affine parameters a, b, c, d, e, f.

Return type Tuple[float, float]

Returns Transformed coordinate (x', y').

2.1.5 Module contents

PYTHON MODULE INDEX

p

INDEX