jrosebr1 / **imutils**

A series of convenience functions to make basic image processing operations such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV and Python.

133 commits | **1** branch | **0** releases | **13** contributors | MIT

Branch: **master** | New pull request | Create new file | Upload files | Find File | Clone or download

jrosebr1 Merge pull request #121 from AdamSpannbauer/master ... Latest commit 4430083 on Feb 24

bin	add an option to display a live preview of what remains in the image ...	3 years ago
demo_images	fix paths.py issue	8 months ago
demos	add text demo	4 months ago
docs/images	Committing rest of files	4 years ago
imutils	add text convenience functions	4 months ago
.gitignore	Improved code quality based on PEP-8	4 years ago
LICENSE.txt	Updated licesnse year to 2015-5016	3 years ago
MANIFEST	Updated version number in setup.py	2 years ago
README.md	Update README.md	2 years ago
setup.cfg	Initial commit of the imutils package	5 years ago
setup.py	Preparing for v0.5.2 release	7 months ago

README.md

imutils

A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV and ***both*** Python 2.7 and Python 3.

For more information, along with a detailed code review check out the following posts on the PyImageSearch.com blog:

- http://www.pyimagesearch.com/2015/02/02/just-open-sourced-personal-imutils-package-series-opencv-convenience-functions/
- http://www.pyimagesearch.com/2015/03/02/convert-url-to-image-with-python-and-opencv/
- http://www.pyimagesearch.com/2015/04/06/zero-parameter-automatic-canny-edge-detection-with-python-and-opencv/
- http://www.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/
- http://www.pyimagesearch.com/2015/08/10/checking-your-opencv-version-using-python/

Installation

Provided you already have NumPy, SciPy, Matplotlib, and OpenCV already installed, the `imutils` package is completely `pip` -installable:

```
$ pip install imutils
```

Finding function OpenCV functions by name

OpenCV can be a big, hard to navigate library, especially if you are just getting started learning computer vision and image processing. The `find_function` method allows you to quickly search function names across modules (and optionally sub-modules) to find the function you are looking for.

Example:

Let's find all function names that contain the text `contour`:

```
import imutils
imutils.find_function("contour")
```

Output:

```
1. contourArea
2. drawContours
3. findContours
4. isContourConvex
```

The `contourArea` function could therefore be accessed via: `cv2.contourArea`

Translation

Translation is the shifting of an image in either the *x* or *y* direction. To translate an image in OpenCV you would need to supply the *(x, y)*-shift, denoted as (t_x, t_y) to construct the translation matrix *M*:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

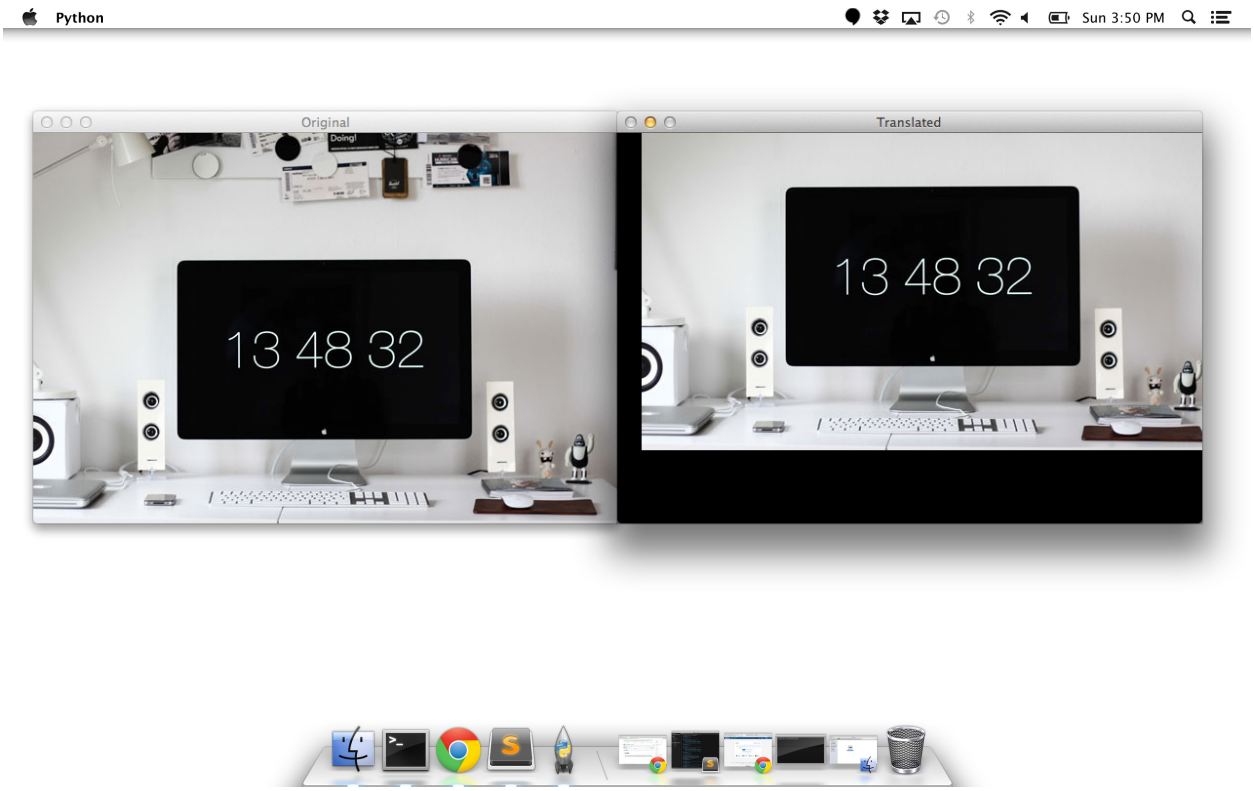
And from there, you would need to apply the `cv2.warpAffine` function.

Instead of manually constructing the translation matrix *M* and calling `cv2.warpAffine`, you can simply make a call to the `translate` function of `imutils`.

Example:

```
# translate the image x=25 pixels to the right and y=75 pixels up
translated = imutils.translate(workspace, 25, -75)
```

Output:



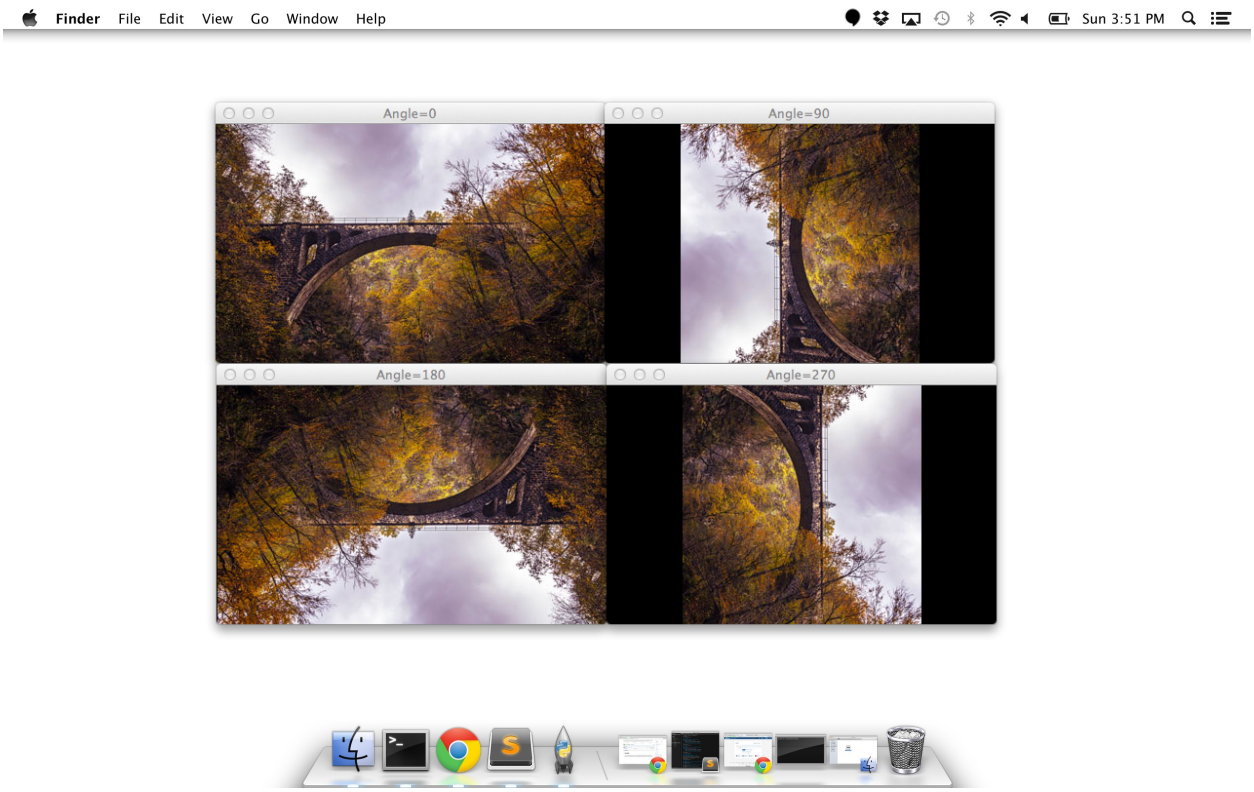
Rotation

Rotating an image in OpenCV is accomplished by making a call to `cv2.getRotationMatrix2D` and `cv2.warpAffine`. Further care has to be taken to supply the *(x, y)*-coordinate of the point the image is to be rotated about. These calculation calls can quickly add up and make your code bulky and less readable. The `rotate` function in `imutils` helps resolve this problem.

Example:

```
# loop over the angles to rotate the image
for angle in xrange(0, 360, 90):
        # rotate the image and display it
        rotated = imutils.rotate(bridge, angle=angle)
        cv2.imshow("Angle=%d" % (angle), rotated)
```

Output:



Resizing

Resizing an image in OpenCV is accomplished by calling the `cv2.resize` function. However, special care needs to be taken to ensure that the aspect ratio is maintained. This `resize` function of `imutils` maintains the aspect ratio and provides the keyword arguments `width` and `height` so the image can be resized to the intended width/height while (1) maintaining aspect ratio and (2) ensuring the dimensions of the image do not have to be explicitly computed by the developer.

Another optional keyword argument, `inter`, can be used to specify interpolation method as well.

Example:

```
# loop over varying widths to resize the image to
for width in (400, 300, 200, 100):
        # resize the image and display it
        resized = imutils.resize(workspace, width=width)
        cv2.imshow("Width=%dpx" % (width), resized)
```

Output:

Skeletonization

Skeletonization is the process of constructing the "topological skeleton" of an object in an image, where the object is presumed to be white on a black background. OpenCV does not provide a function to explicitly construct the skeleton, but does provide the morphological and binary functions to do so.

For convenience, the `skeletonize` function of `imutils` can be used to construct the topological skeleton of the image.

The first argument, `size` is the size of the structuring element kernel. An optional argument, `structuring`, can be used to control the structuring element -- it defaults to `cv2.MORPH_RECT`, but can be any valid structuring element.

Example:

```
# skeletonize the image
gray = cv2.cvtColor(logo, cv2.COLOR_BGR2GRAY)
skeleton = imutils.skeletonize(gray, size=(3, 3))
cv2.imshow("Skeleton", skeleton)
```

Output:

Displaying with Matplotlib

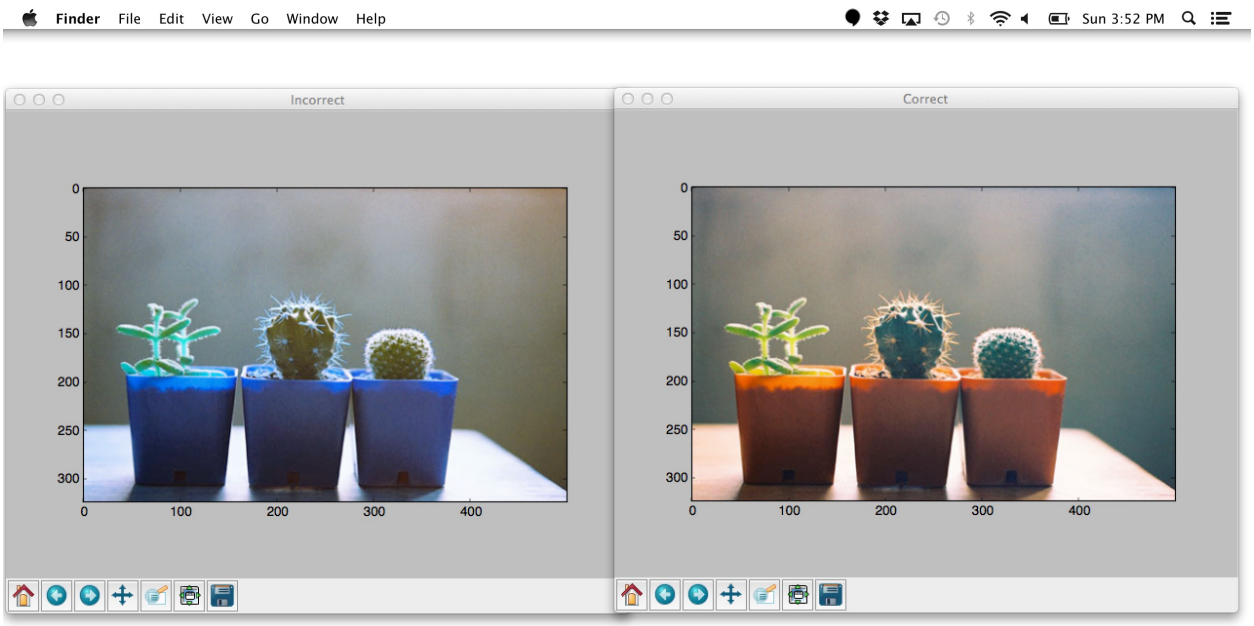
In the Python bindings of OpenCV, images are represented as NumPy arrays in BGR order. This works fine when using the `cv2.imshow` function. However, if you intend on using Matplotlib, the `plt.imshow` function assumes the image is in RGB order. A simple call to `cv2.cvtColor` will resolve this problem, or you can use the `opencv2matplotlib` convenience function.

Example:

```
# INCORRECT: show the image without converting color spaces
plt.figure("Incorrect")
plt.imshow(cactus)

# CORRECT: convert color spaces before using plt.imshow
plt.figure("Correct")
plt.imshow(imutils.opencv2matplotlib(cactus))
plt.show()
```

Output:



URL to Image

This the `url_to_image` function accepts a single parameter: the `url` of the image we want to download and convert to a NumPy array in OpenCV format. This function performs the download in-memory. The `url_to_image` function has been detailed here on the PyImageSearch blog.

Example:

```
url = "http://pyimagesearch.com/static/pyimagesearch_logo_github.png"
logo = imutils.url_to_image(url)
cv2.imshow("URL to Image", logo)
cv2.waitKey(0)
```

Output:

Checking OpenCV Versions

OpenCV 3 has finally been released! But with the major release becomes backward compatibility issues (such as with the `cv2.findContours` and `cv2.normalize` functions). If you want your OpenCV 3 code to be backwards compatible with OpenCV 2.4.X, you'll need to take special care to check which version of OpenCV is currently being used and then take appropriate action. The `is_cv2()` and `is_cv3()` are simple functions that can be used to automatically determine the OpenCV version of the current environment.

Example:

```
print("Your OpenCV version: {}".format(cv2.__version__))
print("Are you using OpenCV 2.X? {}".format(imutils.is_cv2()))
print("Are you using OpenCV 3.X? {}".format(imutils.is_cv3()))
```

Output:

```
Your OpenCV version: 3.0.0
Are you using OpenCV 2.X? False
Are you using OpenCV 3.X? True
```

Automatic Canny Edge Detection

The Canny edge detector requires two parameters when performing hysteresis. However, tuning these two parameters to obtain an optimal edge map is non-trivial, especially when working with a dataset of images. Instead, we can use the `auto_canny` function which uses the median of the grayscale pixel intensities to derive the upper and lower thresholds. You can read more about the `auto_canny` function here.

Example:

```
gray = cv2.cvtColor(logo, cv2.COLOR_BGR2GRAY)
```

```
edgeMap = imutils.auto_canny(gray)
cv2.imshow("Original", logo)
cv2.imshow("Automatic Edge Map", edgeMap)
```

Output:

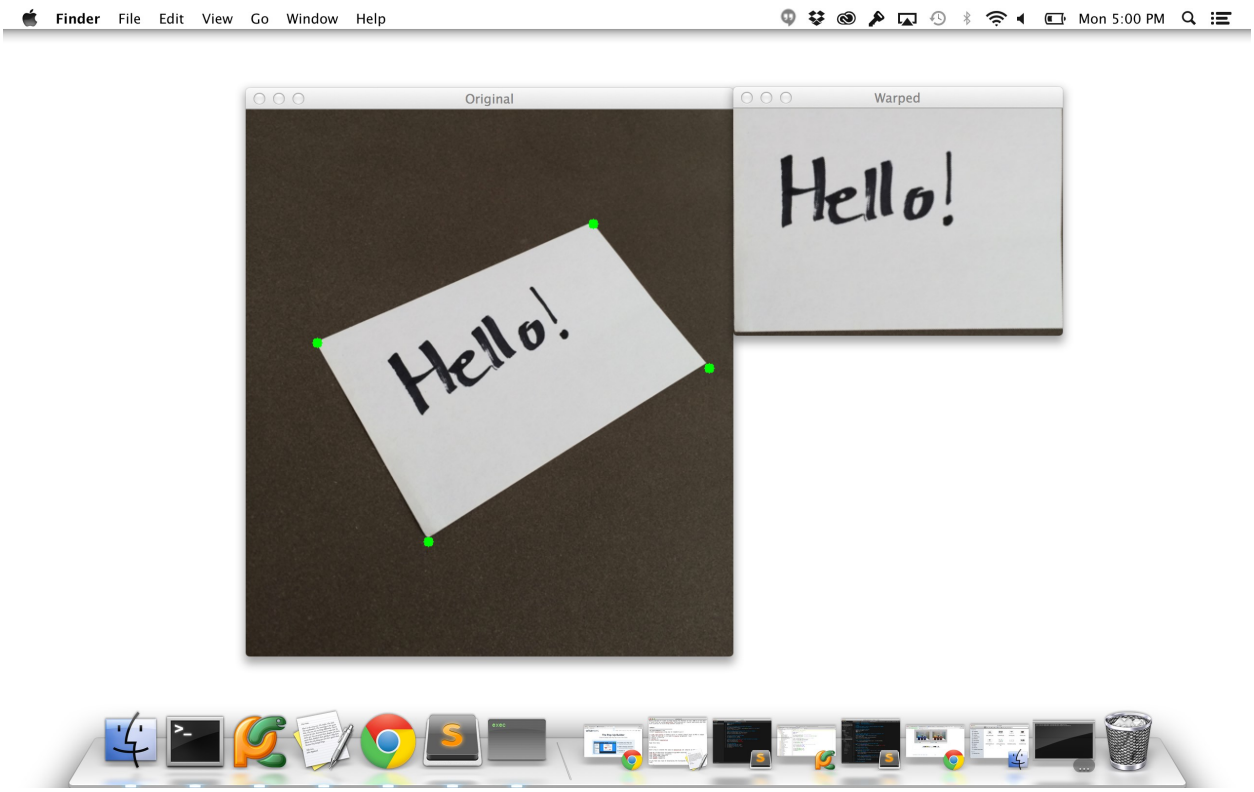
4-point Perspective Transform

A common task in computer vision and image processing is to perform a 4-point perspective transform of a ROI in an image and obtain a top-down, "birds eye view" of the ROI. The `perspective` module takes care of this for you. A real-world example of applying a 4-point perspective transform can be bound in this blog on on building a kick-ass mobile document scanner.

Example

See the contents of `demos/perspective_transform.py`

Output:



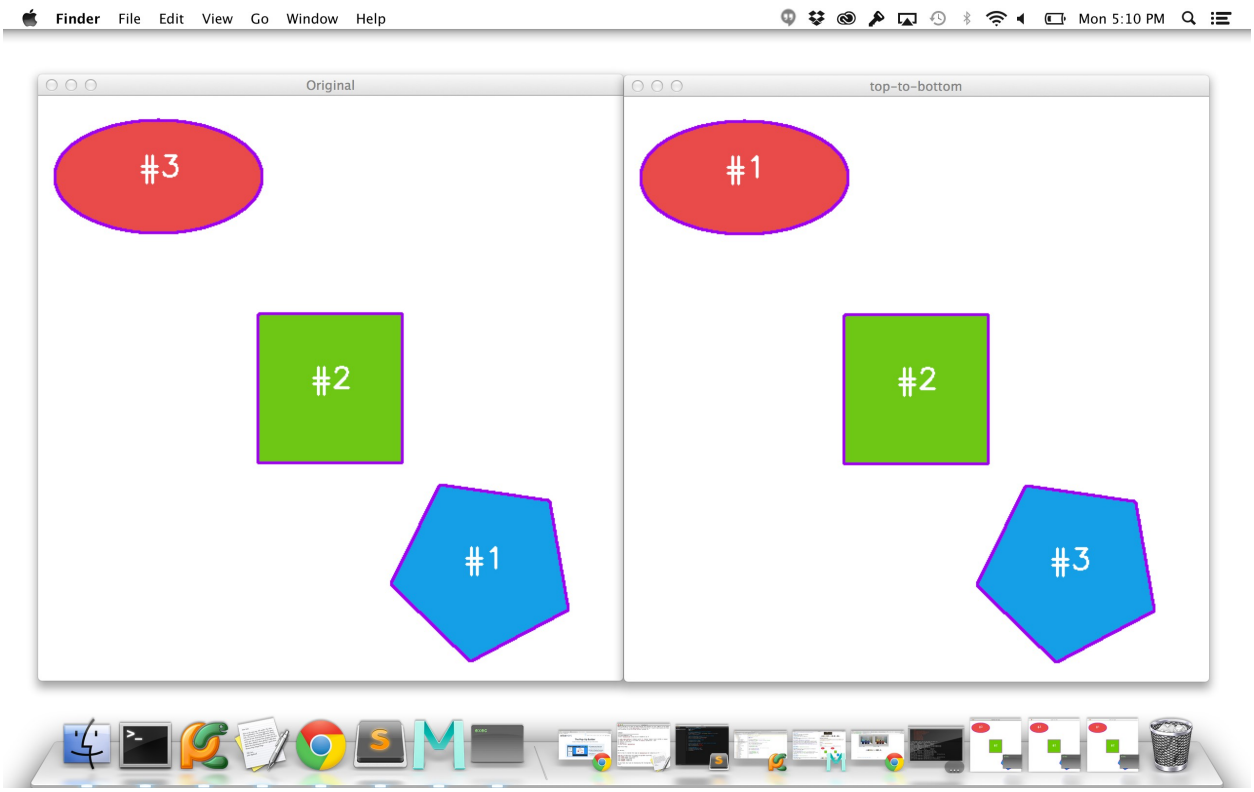
Sorting Contours

The contours returned from `cv2.findContours` are unsorted. By using the `contours` module the the `sort_contours` function we can sort a list of contours from left-to-right, right-to-left, top-to-bottom, and bottom-to-top, respectively.

Example:

See the contents of `demos/sorting_contours.py`

Output:



(Recursively) Listing Paths to Images

The `paths` sub-module of `imutils` includes a function to recursively find images based on a root directory.

Example:

Assuming we are in the `demos` directory, let's list the contents of the `../demo_images`:

```
from imutils import paths
for imagePath in paths.list_images("../demo_images"):
        print imagePath
```

Output:

```
../demo_images/bridge.jpg
../demo_images/cactus.jpg
../demo_images/notecard.png
../demo_images/pyimagesearch_logo.jpg
../demo_images/shapes.png
../demo_images/workspace.jpg
```