(https://gurus.pyimagesearch.com/)

PyImageSearch Gurus Course

(HTTPS://GURUS.PYIMAGESEARCH.COM) ›

1.4.2: Rotation

Topic Progress: (https://gurus.pyimagesearch.com/topic/translation/) (https://gurus.pyimagesearch.com/topic/rotation/) (https://gurus.pyimagesearch.com/topic/resizing/) (https://gurus.pyimagesearch.com/topic/flipping/) (https://gurus.pyimagesearch.com/topic/cropping/) (https://gurus.pyimagesearch.com/topic/image-arithmetic/) (https://gurus.pyimagesearch.com/topic/bitwise-operations/) (https://gurus.pyimagesearch.com/topic/masking/) (https://gurus.pyimagesearch.com/topic/splitting-and-merging-channels/)

Feedback

← Back to Lesson (https://gurus.pyimagesearch.com/lessons/basic-image-processing/)

In the last section (https://gurus.pyimagesearch.com/topic/translation/) we reviewed how to translate (i.e. shift) an image up, down, left, and right (or any combination). Now we are going to move on to our next image processing topic — *rotation*.

Rotation is exactly what it sounds like: rotating an image by some angle θ. We'll use θ to represent by how many degrees we are rotating the image. And later, I'll provide another convenience method, `rotate`, to make performing rotations on images easier.

Objectives:

By the end of this topic you should understand rotation and know how to rotate an image using OpenCV.

Rotation

Similar to translation (https://gurus.pyimagesearch.com/topic/translation/), and perhaps unsurprisingly, rotation by an angle, θ can be defined by constructing a matrix *M* in the form:

$$M = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

Given an *(x, y)*-Cartesian plane, this matrix can be used to rotate a vector θ degrees (counter-clockwise) about the origin. In this case, the origin is normally the *center* of the image; however, in practice we can define any arbitrary *(x, y)* coordinate as our rotation center.

From the original image *I*, the rotated image *R* is then obtained by simple matrix multiplication: $R = IM$

However, OpenCV also provides the ability to (1) scale (i.e. resize) an image and (2) provide an arbitrary rotation center to perform the rotation about.

Our modified rotation matrix *M* is thus:

$$M = \begin{bmatrix} \alpha & \beta & (1-\alpha) \times c_x - \beta \times c_y \\ -\beta & \alpha & \beta \times c_x + (1-\alpha) \times c_y \end{bmatrix}$$

Where:

$\alpha = scale * cos\theta$ and $\beta = scale * sin\theta$ and c_x and c_y are the respective *(x, y)*-coordinates that the rotation is performed about.

If the mathematics is starting to get a bit overwhelming, no worries — we're about to jump into some code that will make these concepts a lot more clear.

Open up a new file, name it `rotate.py` , and let's get started:

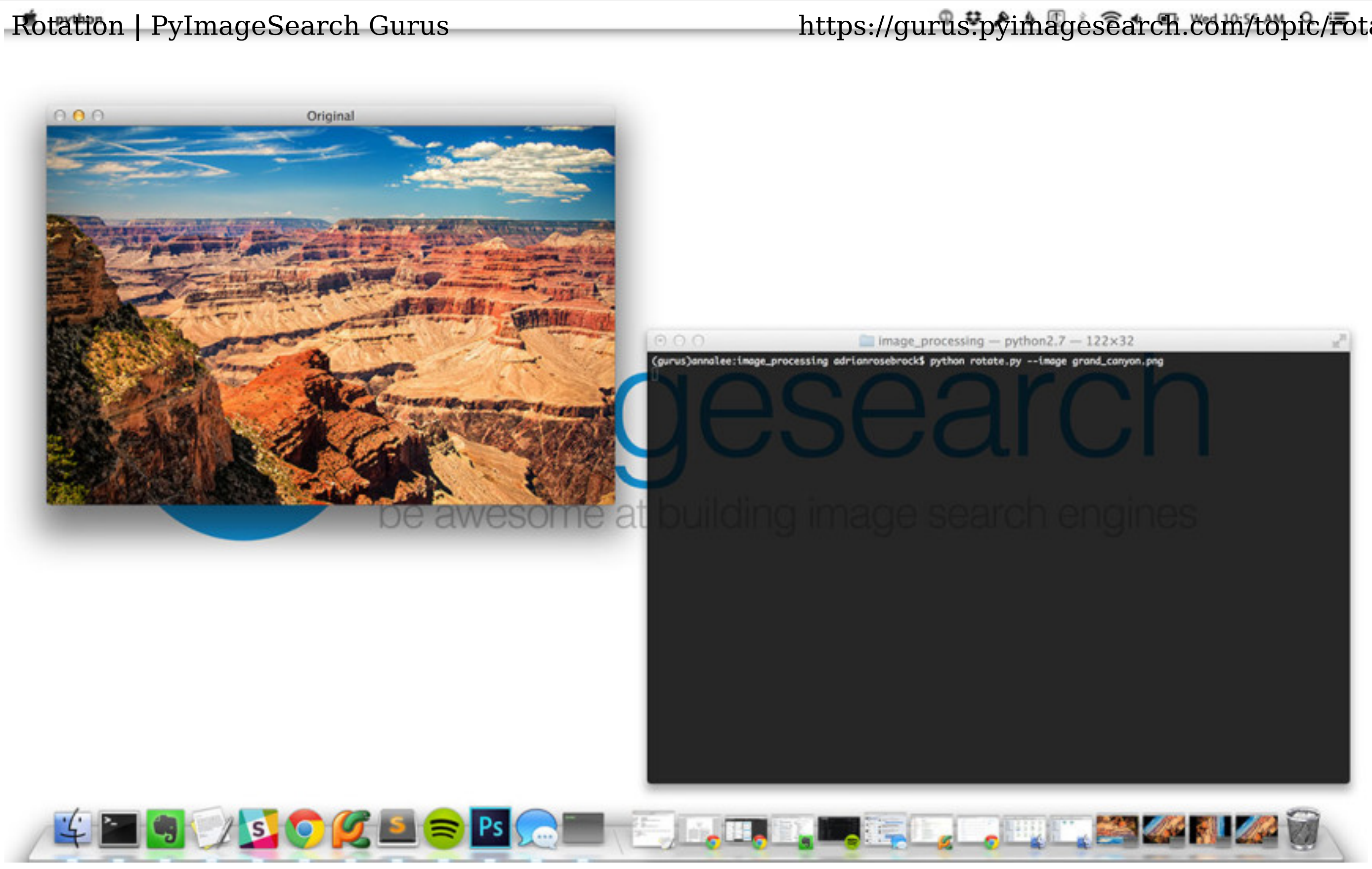
```
rotate.py                                                              Python
```

```
 1 # import the necessary packages
 2 import numpy as np
 3 import argparse
 4 import imutils
 5 import cv2
 6
 7 # construct the argument parser and parse the arguments
 8 ap = argparse.ArgumentParser()
 9 ap.add_argument("-i", "--image", required=True, help="Path to the image")
10 args = vars(ap.parse_args())
11
12 # load the image and show it
13 image = cv2.imread(args["image"])
14 cv2.imshow("Original", image)
15
16 # grab the dimensions of the image and calculate the center of the image
17 (h, w) = image.shape[:2]
18 (cX, cY) = (w / 2, h / 2)
19
20 # rotate our image by 45 degrees
21 M = cv2.getRotationMatrix2D((cX, cY), 45, 1.0)
22 rotated = cv2.warpAffine(image, M, (w, h))
23 cv2.imshow("Rotated by 45 Degrees", rotated)
24
25 # rotate our image by -90 degrees
26 M = cv2.getRotationMatrix2D((cX, cY), -90, 1.0)
27 rotated = cv2.warpAffine(image, M, (w, h))
28 cv2.imshow("Rotated by -90 Degrees", rotated)
```

Lines 1-5 again import the packages we need. You should especially take note of the `imutils` package — once again we will be defining a convenience method to make our lives easier.

Lines 8-10 construct our argument parser. We only need one argument, `--image`, which is the path to the image we are going to use for our rotation example. We then load our image off disk and display it.

For a point of reference, here is our original image prior to any rotation being performed:



```
(gurus)annalee:image_processing adrianrosebrock$ python rotate.py --image grand_canyon.png
```

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/rotation_original.jpg)

FIGURE 1: OUR ORIGINAL IMAGE PRIOR TO PERFORMING ANY ROTATION OPERATIONS.

Feedback

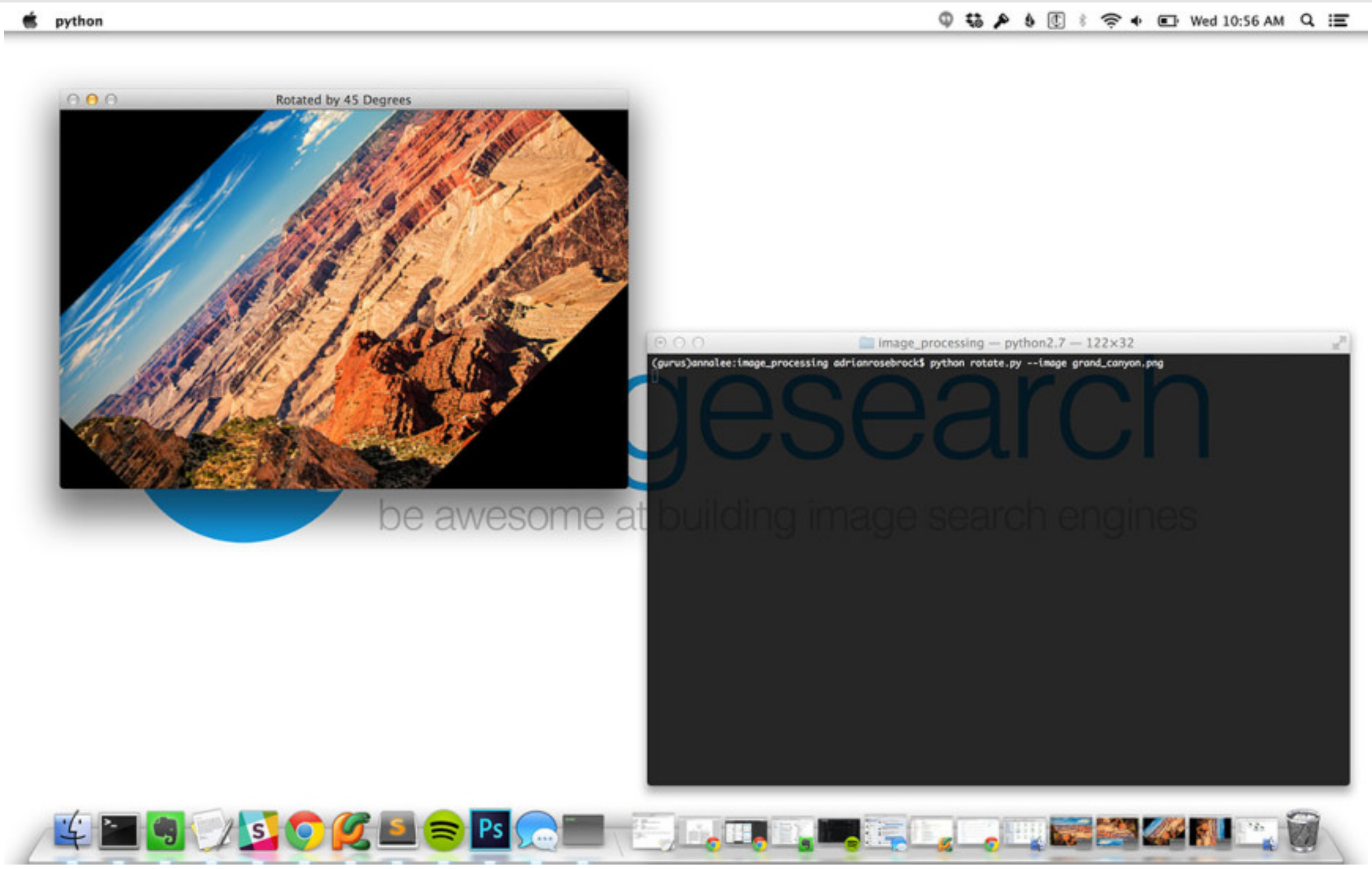
When we rotate an image, we need to specify which point we want to rotate about. In most cases, you will want to rotate around the *center* of an image; however, OpenCV allows you to specify any arbitrary point you want to rotate around (as detailed above). Let's just go ahead and rotate about the center of the image. **Lines 17 and 18** grab the width and height of the image, then proceed to divide each component by 2 to determine the center of the image.

Just as we defined a matrix to translate an image, we also define a matrix to rotate the image. Instead of manually constructing the matrix using NumPy (which can be a bit tedious), we'll just make a call to the `cv2.getRotationMatrix2D` method on **Line 21**.

The `cv2.getRotationMatrix2D` function takes three arguments. The first argument is the point in which we want to rotate the image about (in this case, the `center` of the image). **We then specify θ, the number of (counter-clockwise) degrees we are going to rotate the image by.** In this case, we are going to rotate the image 45 degrees. The last argument is the scale of the image. We haven't discussed resizing an image yet, but here you can specify a floating point value, where *1.0* means the same, original dimensions of the image are used. However, if you specified a value of *2.0*, the image would be doubled in size. Similarly, a value of *0.5* halve the size of the image.

Once we have our rotation matrix *M* from the `cv2.getRotationMatrix2D` function, we can apply the rotation to our image using the `cv2.warpAffine` method on **Line 22**. The first argument to this function is the image we want to rotate. We then specify our rotation matrix *M* along with the output dimensions (width and height) of our image. **Line 23** then shows our image rotated by 45 degrees.

When can see the output of our 45 degree rotation below:


python
Wed 10:56 AM
Rotated by 45 Degrees
image_processing — python2.7 — 122×32
be awesome at building image search engines

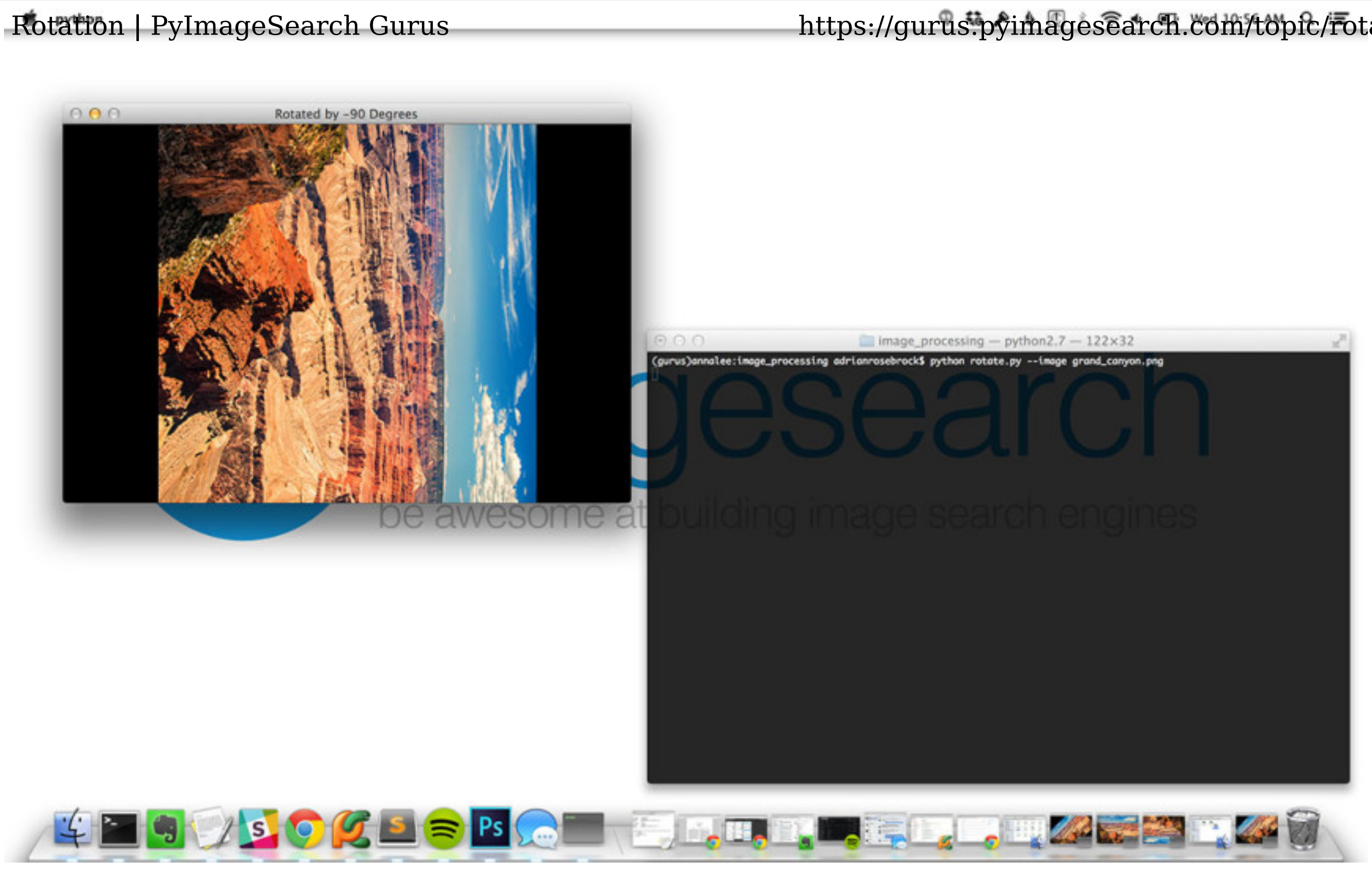
```
(gurus)annalee:image_processing adrianrosebrock$ python rotate.py --image grand_canyon.png
```

[(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/rotation_45_degrees.jpg)](https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/rotation_45_degrees.jpg)

FIGURE 2: ROTATING AN IMAGE 45 DEGREES COUNTER-CLOCKWISE.

As you can see, our image has been rotated. But also take a second to note that OpenCV *does not* automatically allocate space for our *entire* rotated image to fit into the frame. In fact, this is the intended behavior! If you want the entire image to fit into view after the rotation you'll need to modify the width and height, denoted as `(w, h)` in the `cv2.warpAffine` function.

On **Lines 26-28** we perform another rotation. The code is identical to that **Lines 21-23**, only this time we are rotating by -90 degrees rather than 45:



```
(gurus)annalee:image_processing adrianrosebrock$ python rotate.py --image grand_canyon.png
```

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/rotation_90_degrees.jpg)

FIGURE 2: BY SPECIFYING A NEGATIVE ANGLE VALUE WE CAN ROTATE OUR IMAGE 90 *CLOCKWISE* RATHER THAN *COUNTER-CLOCKWISE*.

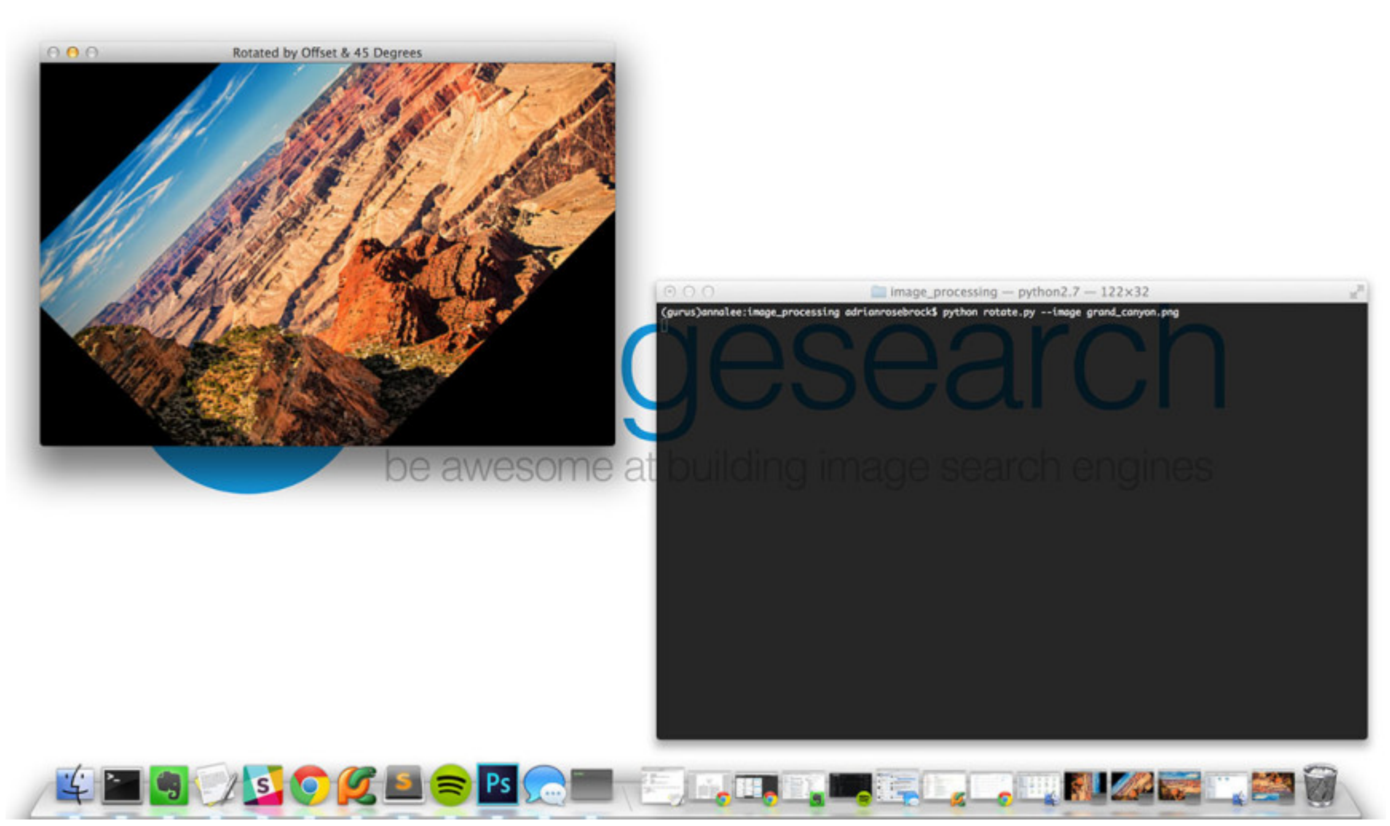
Up until this point we have only rotated an image about the *center* of the image. But what if we wanted to rotate the image about some arbitrary point?

Let’s go ahead and see how this can be accomplished:

```python
# rotate.py                                                    Python
# rotate our image around an arbitrary point rather than the center
M = cv2.getRotationMatrix2D((cX - 50, cY - 50), 45, 1.0)
rotated = cv2.warpAffine(image, M, (w, h))
cv2.imshow("Rotated by Offset & 45 Degrees", rotated)
```

By now this code should look fairly standard for performing a rotation. However, take a note of the first argument of the `cv2.getRotationMatrix2D` function — here we are indicating that we want to rotate the image about a point that is 50 pixels *to the left* and 50 pixels *above* the center of the image.

When we apply this rotation, our output image looks like this:


Rotated by Offset & 45 Degrees
image_processing — python2.7 — 122×32

```
(gurus)annalee:image_processing adrianrosebrock$ python rotate.py --image grand_canyon.png
```

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/rotation_offset_45_degrees.jpg)

FIGURE 3: ROTATING OUR IMAGE ABOUT A POINT THAT IS *NOT* THE CENTER OF THE IMAGE.

Clearly we can see that the center of the rotation is no longer the center of the image — it's the *(x, y)*-coordinate that is both 50 pixels to the left and 50 pixels above the calculated center of the image.

However, just like translating an image, making calls to both `cv2.getRotationMatrix2D` and `cv2.warpAffine` can become quite tedious — not to mention it also makes our code substantially more verbose.

Let's reduce the amount of code we have to write and define our own custom `rotate` method in the `imutils` package:

imutils.py Python

```
13 def rotate(image, angle, center=None, scale=1.0):
14     # grab the dimensions of the image
15     (h, w) = image.shape[:2]
16 
17     # if the center is None, initialize it as the center of
18     # the image
19     if center is None:
20         center = (w / 2, h / 2)
21 
22     # perform the rotation
23     M = cv2.getRotationMatrix2D(center, angle, scale)
24     rotated = cv2.warpAffine(image, M, (w, h))
25 
26     # return the rotated image
27     return rotated
```

Our new `rotate` method takes four arguments. The first is our image. The second is the angle θ in which we want to rotate the image. We provide two optional keyword arguments, `center` and `scale`. The `center` parameter is the *(x, y)*-coordinate in which we wish to rotate our image about. If a value of `None` is provided, the method automatically determines the center of the image on **Lines 19 and 20**. Finally, the `scale` parameter is used to handle if the size of the image should be changed during the rotation. The `scale` parameter has a default value of *1.0*, implying that no resizing should be done.

The actual rotation of the image takes place on **Lines 23 and 24**, were we construct our rotation matrix *M* and apply it to the image. Finally, our image is returned on **Line 27**.

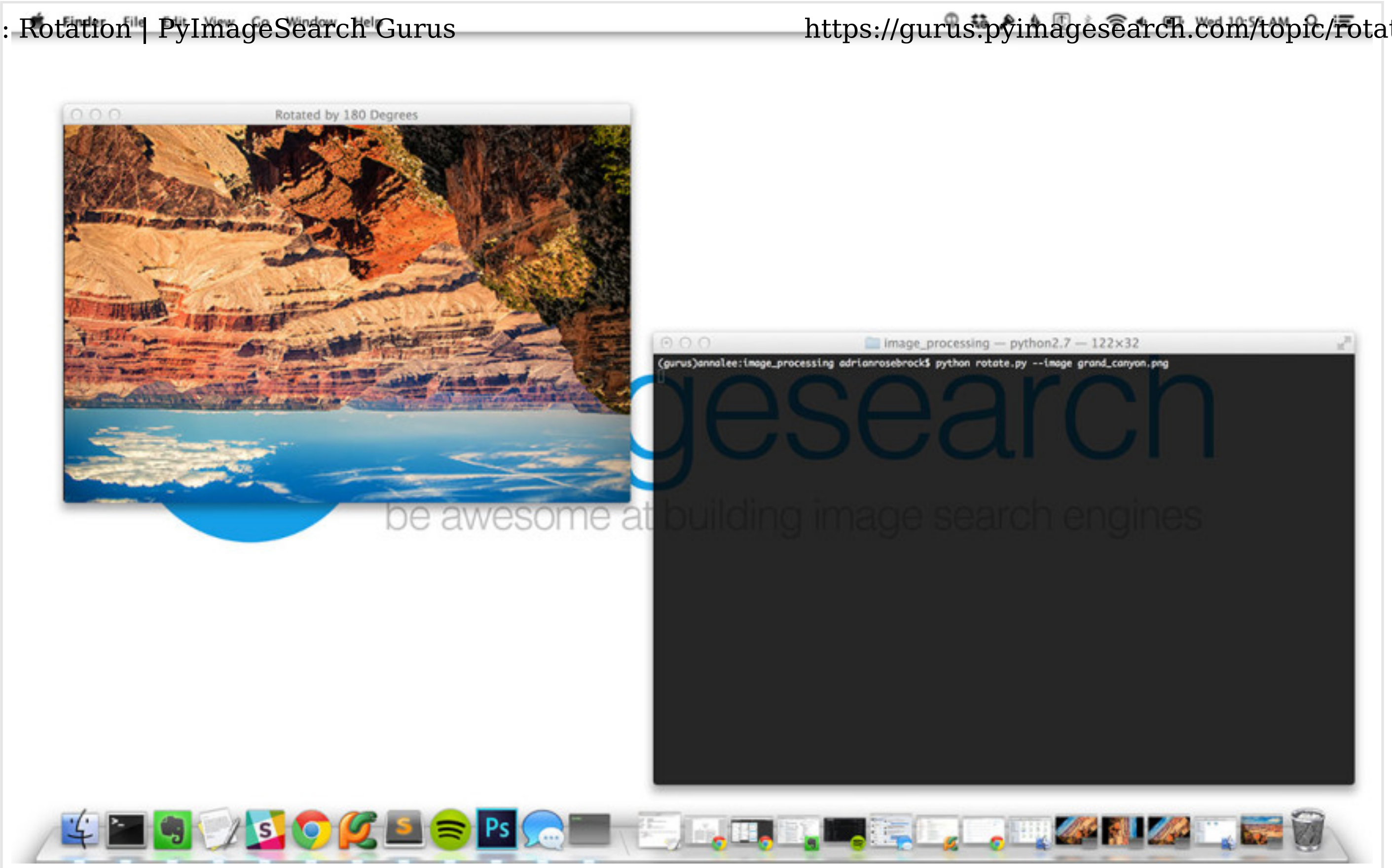
Now that we have defined our `rotate` method, let's apply it:

rotate.py Python

```python
# finally, let's use our helper function in imutils to rotate the image by
# 180 degrees (flipping it upside down)
rotated = imutils.rotate(image, 180)
cv2.imshow("Rotated by 180 Degrees", rotated)
cv2.waitKey(0)
```

Here we are rotating our image by 180 degrees — but we were able to make our code substantially less verbose by using the `rotate` method of `imutils`.

The figure below displays the output of our rotation:


Rotated by 180 Degrees
image_processing — python2.7 — 122×32
gesearch
be awesome at building image search engines

```
(gurus)annalee:image_processing adrianrosebrock$ python rotate.py --image grand_canyon.png
```

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/rotation_180_degrees.jpg)

FIGURE 4: USING THE ROTATE CONVENIENCE FUNCTION OF THE IMUTILS PACKAGE TO ROTATE OUR IMAGE 180 DEGREES COUNTER-CLOCKWISE.

Feedback

And that's all there is to it!

Summary

In this topic you learned how a rotation matrix is used to rotate an image about an arbitrary point in the Cartesian space. You then learned how OpenCV utilizes this matrix to perform rotations.

From there we viewed a few examples of rotating an image, followed by defining our own convenience function to make rotating images easier.

In the next topic we'll explore *resizing*, and how different interpolation methods can give you dramatically different results when resizing your images.

Downloads:

Download the Code (https://gurus.pyimagesearch.com/protected/code/computer_vision_basics/rotation.zip)

Quizzes		Status
1	Rotation Quiz (https://gurus.pyimagesearch.com/quizzes/rotation-quiz/)	

← Previous Topic (https://gurus.pyimagesearch.com/topic/translation/) Next Topic → (https://gurus.pyimagesearch.com/topic/resizing/)

Course Progress

Ready to continue the course?

Click the button below to **continue your journey to computer vision guru**.

I'm ready, let's go! (/pyimagesearch-gurus-course/)

Feedback

Resources & Links

- PyImageSearch Gurus Community (https://community.pyimagesearch.com/)
- PyImageSearch Virtual Machine (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/)
- Setting up your own Python + OpenCV environment (https://gurus.pyimagesearch.com/setting-up-your-python-opencv-development-environment/)
- Course Syllabus & Content Release Schedule (https://gurus.pyimagesearch.com/course-syllabus-content-release-schedule/)
- Member Perks & Discounts (https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/)
- Your Achievements (https://gurus.pyimagesearch.com/achievements/)
- Official OpenCV documentation (http://docs.opencv.org/index.html)

Your Account

- Account Info (https://gurus.pyimagesearch.com/account/)
- Support (https://gurus.pyimagesearch.com/contact/)
- Logout (https://gurus.pyimagesearch.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fgurus.pyimagesearch.com%2F&_wpnonce=5736b21cae)

Search

Feedback