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

PyImageSearch Gurus Course

(HTTPS://GURUS.PYIMAGESEARCH.COM) ›

1.4.1: Translation

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/)

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

Feedback

Module 1.4: Basic image processing is one of the largest (and perhaps most important) modules in all of the PyImageSearch Gurus courses. While the techniques you'll learn inside this course are quite basic, they form the cornerstones of building more advanced computer vision and image processing algorithms.

For example, when we build custom image classifiers we'll need to examine images at various *scales* (i.e. sizes) — and in order to obtain these various scales of an image we'll need to understand how to *resize* an image.

After we have detected an object of interest using our custom image classifier, how we will go about extracting the object? Using *cropping* of course!

Later in this course we'll explore how to extract features to represent and quantify the contents of an image. Feature extraction plays a vital role in image classification, image search engines, and many other sub-fields of computer vision. But there may be times where we *only* want to extract and quantify *part* of an image rather than *all* of it — and when this happens, we'll need to understand *bitwise operations* and *masking*.

Again, the techniques you learn inside this module are not super advanced or challenging topics to master. But they are extremely important to understand when we move on to more advanced topics. Furthermore, these image processing topics are the cornerstones on which more advanced algorithms are built. It is more than likely that you'll use one or more of these techniques inside your own computer vision applications.

Today, we'll review the first of these important image processing techniques: *translation*.

Objectives:

To understand how to *translate* an image using OpenCV.

Translation

Translation is the shifting of an image along the *x* and *y* axis. Using translation, we can shift an image up, down, left, or right, along with any combination of the above.

Mathematically, we define a *translation matrix* M that we can use to translate an image:

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

This concept is better explained through some code:

```
translation.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 # NOTE: Translating (shifting) an image is given by a NumPy matrix in

17 # the form:
18 #   [[1, 0, shiftX], [0, 1, shiftY]]
19 # You simply need to specify how many pixels you want to shift the image
20 # in the X and Y direction -- let's translate the image 25 pixels to the
21 # right and 50 pixels down
22 M = np.float32([[1, 0, 25], [0, 1, 50]])
23 shifted = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))
24 cv2.imshow("Shifted Down and Right", shifted)
25
26 # now, let's shift the image 50 pixels to the left and 90 pixels up, we
27 # accomplish this using negative values
28 M = np.float32([[1, 0, -50], [0, 1, -90]])
29 shifted = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))
30 cv2.imshow("Shifted Up and Left", shifted)
```

On **Lines 1-5** we simply import the packages we will make use of. At this point, using `numpy`, `argparse`, and `cv2` should feel commonplace. However, I am introducing a new package here: `imutils`. This isn't a package included in NumPy or OpenCV. Rather, it's a library that I personally wrote (http://www.pyimagesearch.com/2015/02/02/just-open-sourced-personal-imutils-package-series-opencv-convenience-functions/) that contains a handful of "convenience" methods to more easily perform common tasks like translation, rotation, and resizing (and with less code).

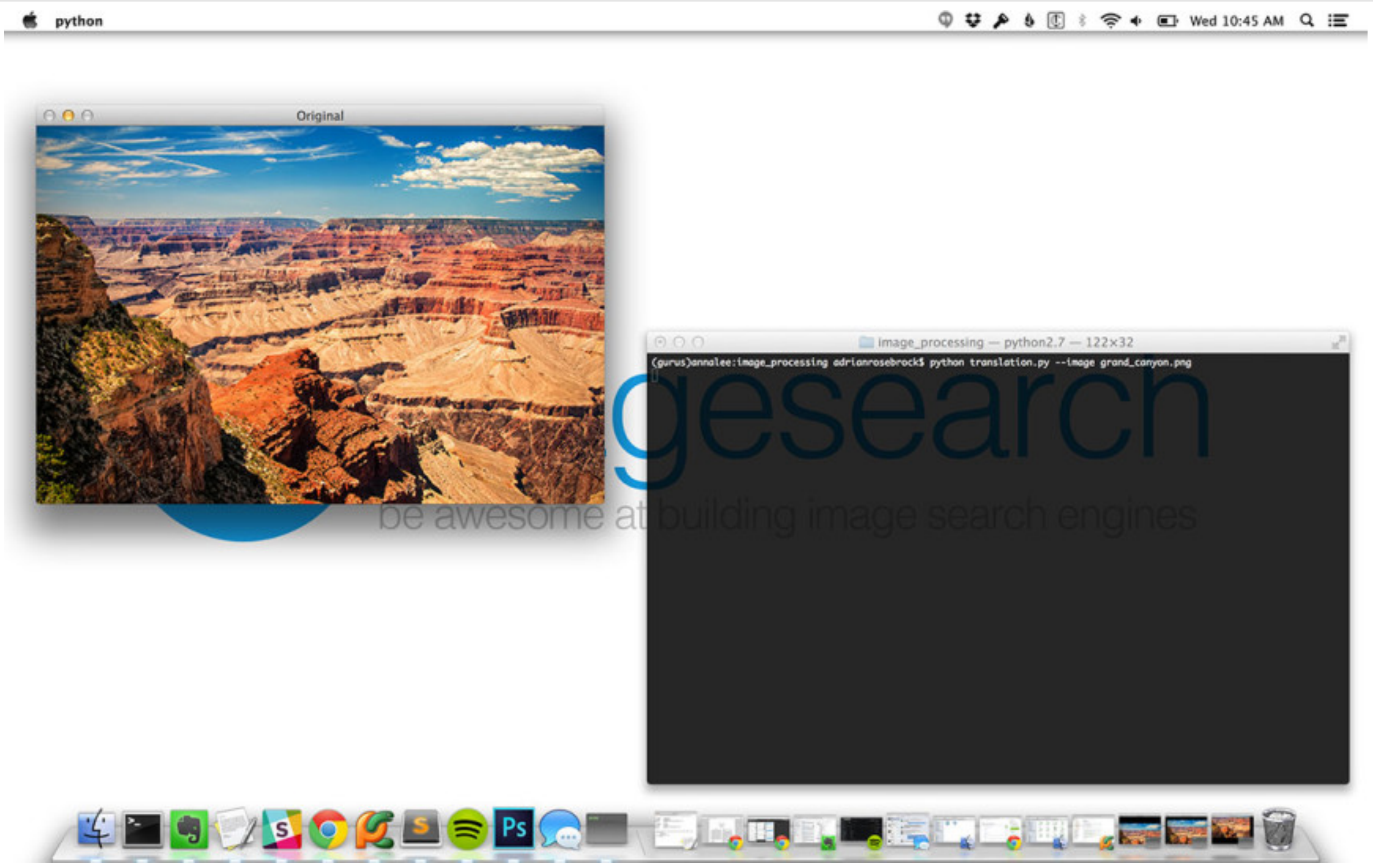
If you are using the PyImageSearch Gurus virtual machine (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/) or if you followed the steps to creating your own custom development environment (https://gurus.pyimagesearch.com/setting-up-your-python-opencv-development-environment/) then the `imutils` package is already installed for you.

Otherwise, you can grab the source off GitHub (https://github.com/jrosebr1/imutils) or simply use pip to install it:

Installing imutils Shell

```
1 $ pip install imutils
```

Anyway, after we have the necessary packages imported, we construct our argument parser and load our image on **Lines 8-13**. Below we can see our original image:


python
Wed 10:45 AM
Original
image_processing — python2.7 — 122×32
gesearch
be awesome at building image search engines

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

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

FIGURE 1: OUR ORIGINAL IMAGE LOADED FROM DISK.

The first actual translation takes place on **Lines 22-24**, where we start by defining our translation matrix *M*. This matrix tells us how many pixels to the left or right our image will shifted, and then how many pixels up or down the image will be shifted.

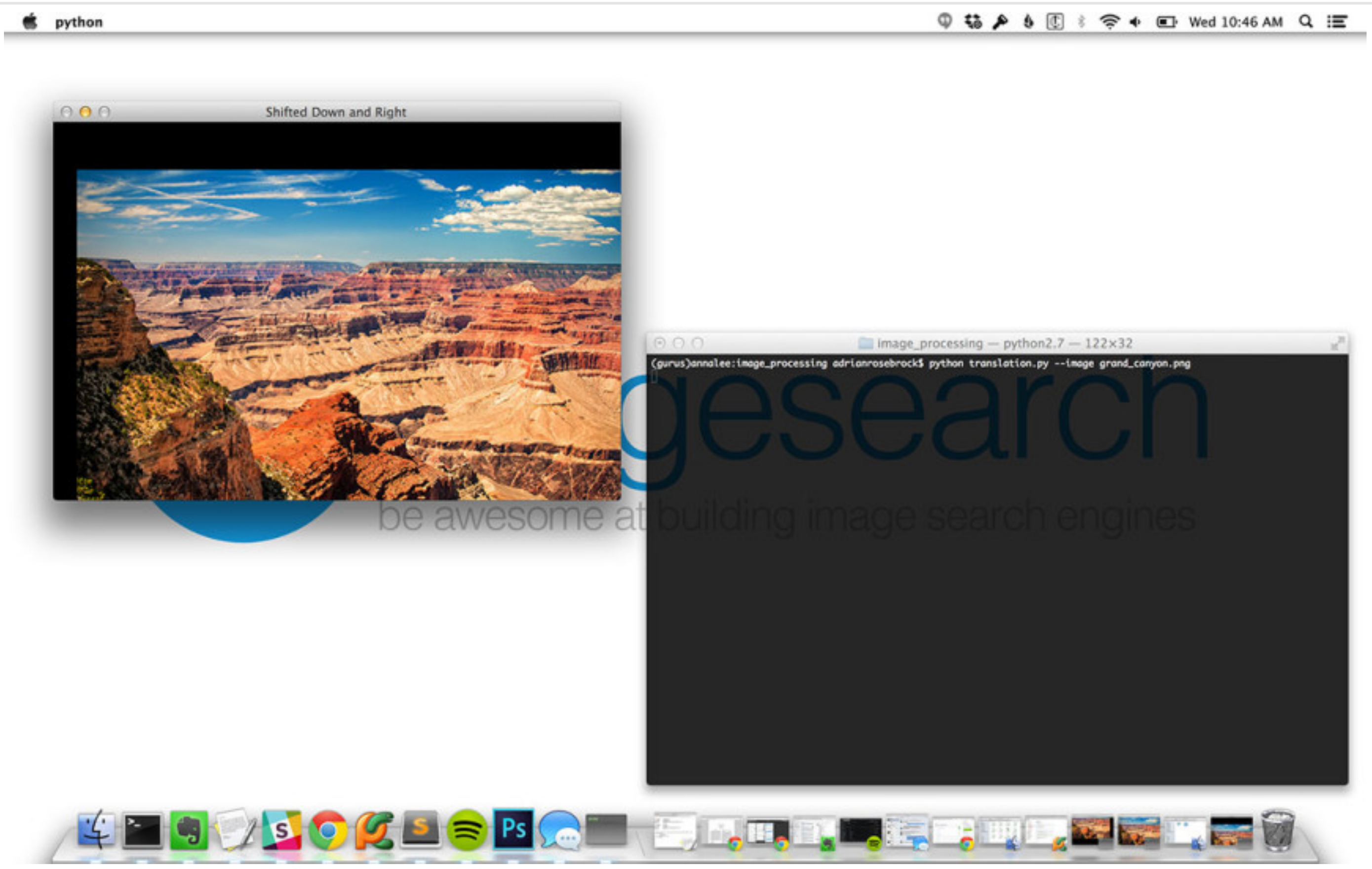
Our translation matrix *M* is defined as a floating point array — this is important because OpenCV expects this matrix to be of floating point type. The first row of the matrix is $[1, 0, t_x]$, where t_x is the number of pixels we will shift the image left or right. Negative values of t_x will shift the image to the left and positive values will shift the image to the right.

Then, we define the second row of the matrix as $[0, 1, t_y]$, where t_y is the number of pixels we will shift the image up or down. Negative values of t_y will shift the image up and positive values will shift the image down.

Using this notation, on **Line 22** we can see that $t_x = 25$ and $t_y = 50$, indicating that we are shifting the image 25 pixels to the right and 50 pixels down.

Now that we have our translation matrix defined, the actual translation takes place on **Line 23** using the `cv2.warpAffine` function. The first argument is the image we wish to shift and the second argument is our translation matrix *M*. Finally, we manually supply the dimensions (width and height) of our image as the third argument.

Line 24 displays the results of the translation which we can see below:



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

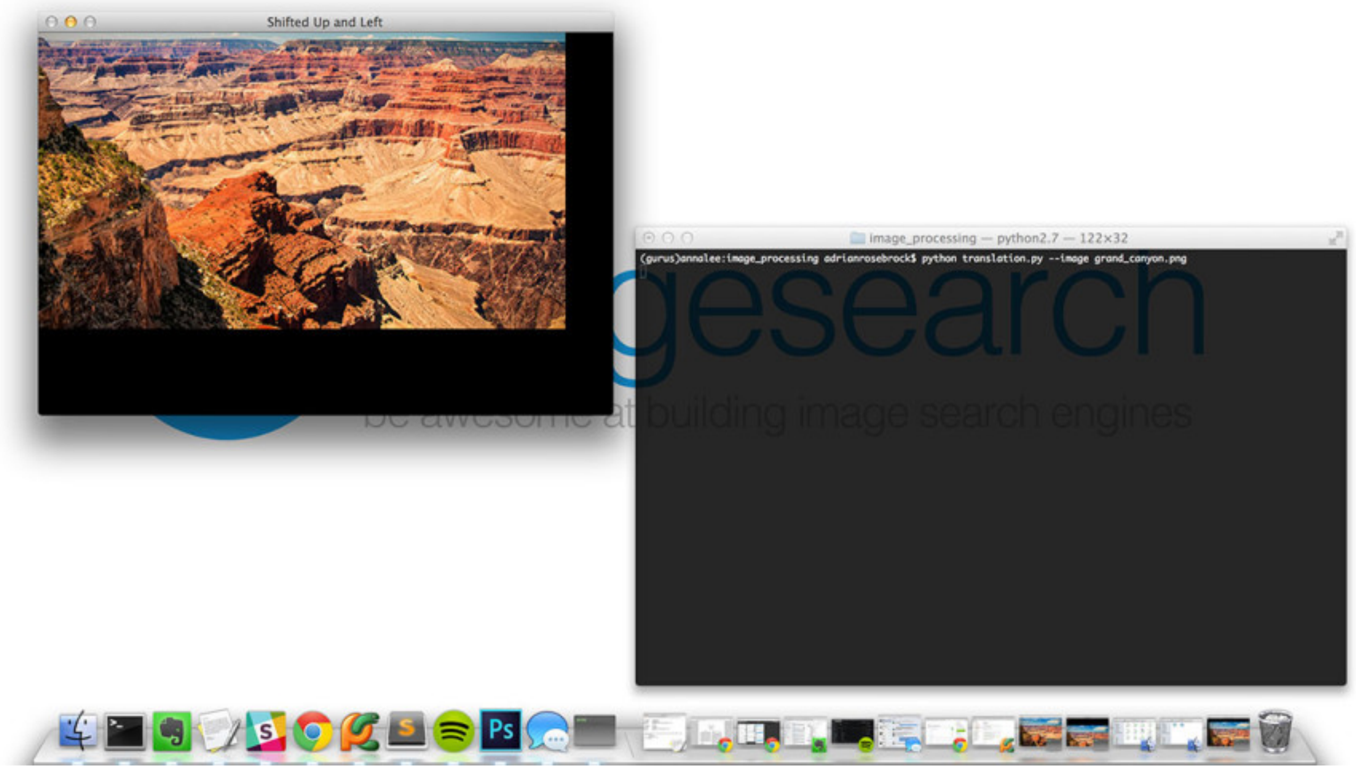
[(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/translation_shifted_down_and_right.jpg)](https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/translation_shifted_down_and_right.jpg)

FIGURE 2: TRANSLATING OUR IMAGES 25 PIXELS TO THE RIGHT AND 50 PIXELS DOWN.

Notice how the image has clearly be "shifted" down and to the right.

Moving on to **Lines 28-30**, we perform another translation. Here, we set $t_x = -50$ and $t_y = -90$, implying that we are shifting the image 50 pixels to the left and 90 pixels up. The image is shifted *left* and *up* rather than *right* and *down* because we are providing a negative values for both t_x and t_y.

The figure below shows the output of supplying negative values for both t_x and t_y:



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

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

FIGURE 3: BY SUPPLYING NEGATIVE VALUES, WE CAN SHIFT OUR IMAGE UP AND TO THE LEFT.

Again, notice how our image is "shifted" to the left 50 pixels and up 90 pixels.

However, manually constructing this translation matrix and calling the `cv2.warpAffine` method takes a fair amount of code — and it's not pretty code either!

This is where the imutils (https://github.com/jrosebr1/imutils) package comes in. Instead of having to define our matrix *M* and make a call to `cv2.warpAffine` each time we want to translate an image, let's define a `translate` convenience function that takes care of this for us:

imutils.py Python

```python
# import the necessary packages
import numpy as np
import cv2

def translate(image, x, y):
    # define the translation matrix and perform the translation
    M = np.float32([[1, 0, x], [0, 1, y]])
    shifted = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))

    # return the translated image
    return shifted
```

Our `translate` method takes three parameters: the `image` we are going to translate, the number of pixels that we are going to shift along the x-axis, and the number of pixels we are going to shift along the y-axis.

This method then defines our translation matrix *M* on **Line 7** and then applies the actual shift on **Line 8**. Finally, we return the shifted image on **Line 11**.

And again, this `translate` function is already part of the imutils (https://github.com/jrosebr1/imutils) package — there is no need for you to define this function yourself!

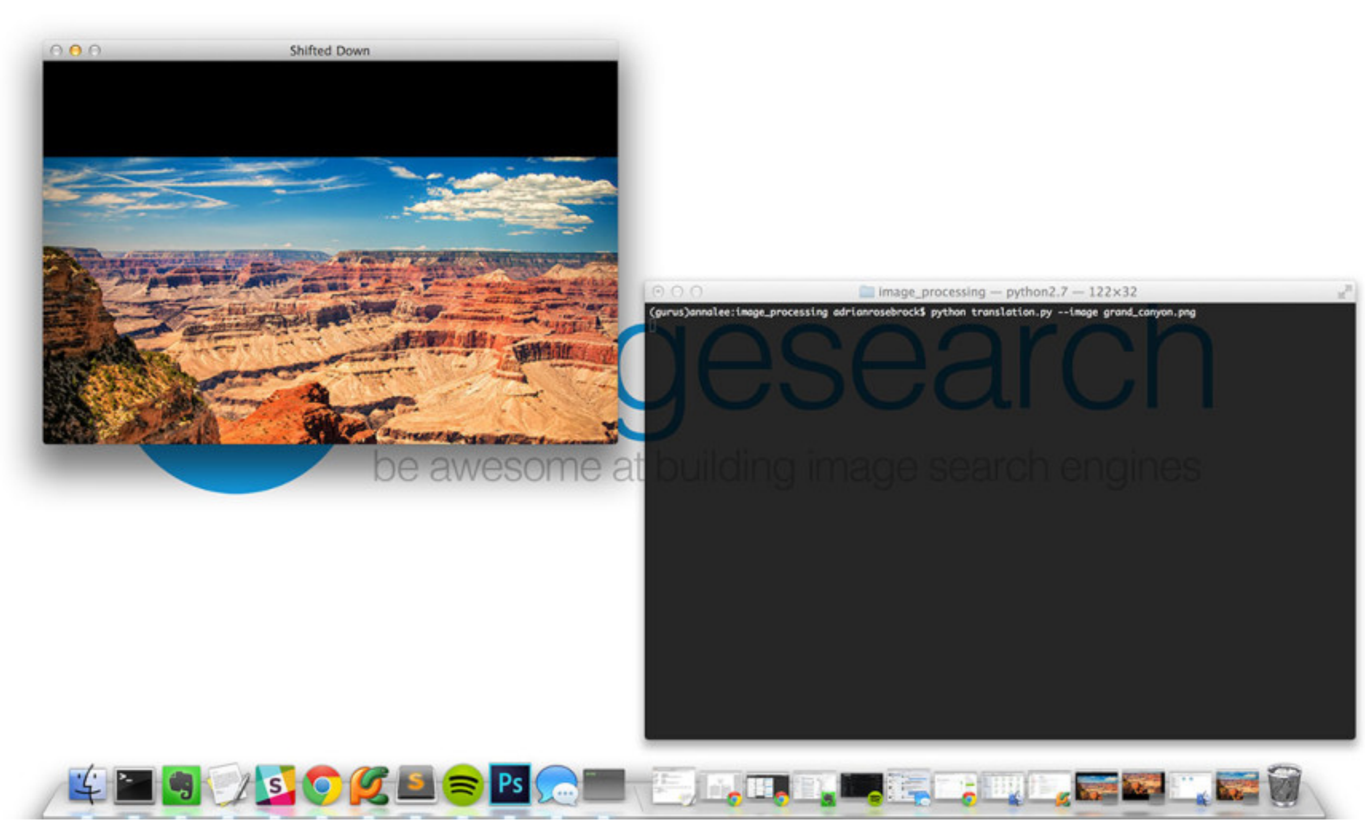
Anyway, let's apply our translate method and compare to the methods discussed above:

translation.py Python

```python
# finally, let's use our helper function in imutils to shift the image down
# 100 pixels
shifted = imutils.translate(image, 0, 100)
cv2.imshow("Shifted Down", shifted)
cv2.waitKey(0)
```

Using our convenience `translate` method, we are able to shift the image 100 pixels down using a single line of code. Furthermore, this `translate` method is much easier to use — less code is required and based on the function name, we conveniently know what image processing task is being performed.

As you can see from the below image, the output is as expected — Our output image is translated shifted 100 pixels down:


Shifted Down
image_processing — python2.7 — 122×32
gesearch
be awesome at building image search engines

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

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

FIGURE 4: SHIFTING OUR IMAGE 100 PIXELS DOWN USING THE TRANSLATE IMUTILS CONVENIENCE FUNCTION.

Feedback

Summary

In this section we explored how to shift an image up, down, left, and right. We were also introduced to the imutils (https://github.com/jrosebr1/imutils) package which contains a handful of convenient functions that make our lives easier when performing basic image processing operations.

Next up, we'll explore how to rotate an image.

Downloads:

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

	Quizzes	Status
1	Translation Quiz (https://gurus.pyimagesearch.com/quizzes/translation/)	

Next Topic → (https://gurus.pyimagesearch.com/topic/rotation/)

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/)

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)

Feedback

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