

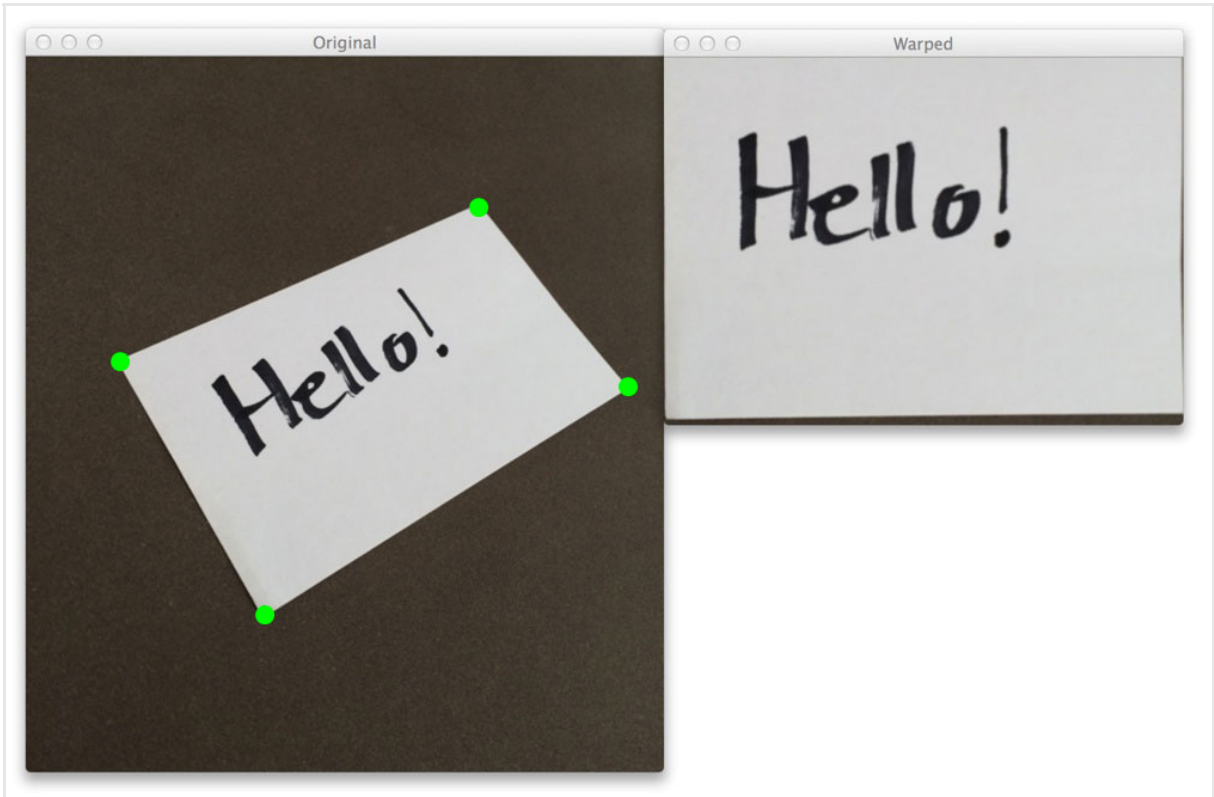


# 4 Point OpenCV getPerspective Transform Example

by **Adrian Rosebrock** on August 25, 2014 in **Image Processing, Tutorials**

5

1



4:18am. Alarm blaring. Still dark outside. The bed is warm. And the floor will feel so cold on my bare feet.

But I got out of bed. I braved the morning, and I took the ice cold floor on my feet like a champ.

Why?

Because I'm excited.

Excited to share something very special with you today...

You see, over the past few weeks I've gotten some really great emails from fellow PyImageSearch readers. These emails were short, sweet, and to the point. They were simple "thank you's" for posting actual, honest-to-goodness Python and OpenCV code that you could take and use to solve your own computer vision and image processing problems.

And upon reflection last night, I realized that I'm not doing a good enough job sharing the libraries, packages, and code that I have developed for myself for everyday use — so that's exactly what I'm going to do today.

In this blog post I'm going to show you the functions in my `transform.py` module. I use these functions whenever I need to do a 4 point `cv2.getPerspectiveTransform` using OpenCV.

And I think you'll find the code in here quite interesting...and you'll even be able to utilize it in your own projects.

So read on. And checkout my 4 point OpenCV `cv2.getPerspectiveTransform` example.

Looking for the source code to this post?  
[Jump right to the downloads section.](#)

**OpenCV and Python versions:**  
This example will run on **Python 2.7/Python 3.4+** and **OpenCV 2.4.X/OpenCV 3.0+**.

# 4 Point OpenCV getPerspectiveTransform Example

You may remember back to my posts on building a real-life Pokedex, specifically, [my post on OpenCV and Perspective Warping](#).

In that post I mentioned how you could use a perspective transform to obtain a top-down, “birds eye view” of an image — provided that you could find reference points, of course.

This post will continue the discussion on the top-down, “birds eye view” of an image. But this time I’m going to share with you *personal code* that I use *every single time* I need to do a 4 point perspective transform.

So let’s not waste any more time. Open up a new file, name it `transform.py`, and let’s get started.

OpenCV getPerspectiveTransform ExamplePython

```
1 # import the necessary packages
2 import numpy as np
3 import cv2
4
5 def order_points(pts):
6     # initialzie a list of coordinates that will be ordered
7     # such that the first entry in the list is the top-left,
8     # the second entry is the top-right, the third is the
9     # bottom-right, and the fourth is the bottom-left
10    rect = np.zeros((4, 2), dtype = "float32")
11
12    # the top-left point will have the smallest sum, whereas
13    # the bottom-right point will have the largest sum
14    s = pts.sum(axis = 1)
15    rect[0] = pts[np.argmin(s)]
16    rect[2] = pts[np.argmax(s)]
17
18    # now, compute the difference between the points, the
19    # top-right point will have the smallest difference,
20    # whereas the bottom-left will have the largest difference
21    diff = np.diff(pts, axis = 1)
22    rect[1] = pts[np.argmin(diff)]
23    rect[3] = pts[np.argmax(diff)]
24
25    # return the ordered coordinates
26    return rect
```

We'll start off by importing the packages we'll need: NumPy for numerical processing and `cv2` for our OpenCV bindings.

Next up, let’s define the `order_points` function on **Line 5**. This function takes a single argument, `pts`, which is a list of four points specifying the (x, y) coordinates of each point of the rectangle.

It is absolutely crucial that we have a *consistent ordering* of the points in the rectangle. The actual ordering itself can be arbitrary, *as long as it is consistent* throughout the implementation.

Personally, I like to specify my points in top-left, top-right, bottom-right, and bottom-left order.

We'll start by allocating memory for the four ordered points on **Line 10**.

Then, we'll find the top-left point, which will have the smallest  $x + y$  sum, and the bottom-right point, which will have the largest  $x + y$  sum. This is handled on **Lines 14-16**.

Of course, now we'll have to find the top-right and bottom-left points. Here we'll take the difference (i.e.  $x - y$ ) between the points using the `np.diff` function on **Line 21**.

The coordinates associated with the smallest difference will be the top-right points, whereas the coordinates with the largest difference will be the bottom-left points (**Lines 22 and 23**).

Finally, we return our ordered functions to the calling function on **Line 26**.

**Again, I can’t stress again how important it is to maintain a consistent ordering of points.**

And you’ll see exactly why in this next function:

OpenCV getPerspectiveTransform ExamplePython

```
28 def four_point_transform(image, pts):
29     # obtain a consistent order of the points and unpack them
30     # individually
31     rect = order_points(pts)
32     (tl, tr, br, bl) = rect
33
34     # compute the width of the new image, which will be the
35     # maximum distance between bottom-right and bottom-left
36     # x-coordiates or the top-right and top-left x-coordinates
37     widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
38     widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
39     maxWidth = max(int(widthA), int(widthB))
40
41     # compute the height of the new image, which will be the
42     # maximum distance between the top-right and bottom-right
```

Free 21-day crash course on computer vision & image search engines

```
43 # y-coordinates or the top-left and bottom-left y-coordinates
44 heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
45 heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
46 maxHeight = max(int(heightA), int(heightB))
47
48 # now that we have the dimensions of the new image, construct
49 # the set of destination points to obtain a "birds eye view",
50 # (i.e. top-down view) of the image, again specifying points
51 # in the top-left, top-right, bottom-right, and bottom-left
52 # order
53 dst = np.array([
54     [0, 0],
55     [maxWidth - 1, 0],
56     [maxWidth - 1, maxHeight - 1],
57     [0, maxHeight - 1]], dtype = "float32")
58
59 # compute the perspective transform matrix and then apply it
60 M = cv2.getPerspectiveTransform(rect, dst)
61 warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))
62
63 # return the warped image
64 return warped
```

We start off by defining the `four_point_transform` function on **Line 28**, which requires two arguments: `image` and `pts` .

The `image` variable is the image we want to apply the perspective transform to. And the `pts` list is the list of four points that contain the ROI of the image we want to transform.

We make a call to our `order_points` function on **Line 31**, which places our `pts` variable in a consistent order. We then unpack these coordinates on **Line 32** for convenience.

Now we need to determine the dimensions of our new warped image.

We determine the width of the new image on **Lines 37-39**, where the width is the largest distance between the bottom-right and bottom-left x-coordinates or the top-right and top-left x-coordinates.

In a similar fashion, we determine the height of the new image on **Lines 44-46**, where the height is the maximum distance between the top-right and bottom-right y-coordinates or the top-left and bottom-left y-coordinates.

*Note: Big thanks to Tom Lowell who emailed in and made sure I fixed the width and height calculation!*

**So here’s the part where you really need to pay attention.**

Remember how I said that we are trying to obtain a top-down, “birds eye view” of the ROI in the original image? And remember how I said that a consistent ordering of the four points representing the ROI is crucial?

On **Lines 53-57** you can see why. Here, we define 4 points representing our “top-down” view of the image. The first entry in the list is `(0, 0)` indicating the top-left corner. The second entry is `(maxWidth - 1, 0)` which corresponds to the top-right corner. Then we have `(maxWidth - 1, maxHeight - 1)` which is the bottom-right corner. Finally, we have `(0, maxHeight - 1)` which is the bottom-left corner.

The takeaway here is that these points are defined in a consistent ordering representation — and will allow us to obtain the top-down view of the image.

To actually obtain the top-down, “birds eye view” of the image we’ll utilize the `cv2.getPerspectiveTransform` function on **Line 60**. This function requires two arguments, `rect` , which is the list of 4 ROI points in the original image, and `dst` , which is our list of transformed points. The `cv2.getPerspectiveTransform` function returns `M` , which is the actual transformation matrix.

We apply the transformation matrix on **Line 61** using the `cv2.warpPerspective` function. We pass in the `image` , our transform matrix `M` , along with the width and height of our output image.

The output of `cv2.warpPerspective` is our `warped` image, which is our top-down view.

We return this top-down view on **Line 64** to the calling function.

Now that we have code to perform the transformation, we need some code to drive it and actually apply it to images.

Open up a new file, call `transform_example.py` , and let’s finish this up:

OpenCV getPerspectiveTransform Example	Python
<pre>1 # import the necessary packages 2 from pyimagesearch.transform import four_point_transform 3 import numpy as np 4 import argparse 5 import cv2 6 7 # construct the argument parse and parse the arguments 8 ap = argparse.ArgumentParser() 9 ap.add_argument("-i", "--image", help = "path to the image file")</pre>	<div>Free 21-day crash course on computer vision &amp; image search engines</div>

```
10 ap.add_argument("-c", "--coords",
11     help = "comma seperated list of source points")
12 args = vars(ap.parse_args())
13
14 # load the image and grab the source coordinates (i.e. the list of
15 # of (x, y) points)
16 # NOTE: using the 'eval' function is bad form, but for this example
17 # let's just roll with it -- in future posts I'll show you how to
18 # automatically determine the coordinates without pre-supplying them
19 image = cv2.imread(args["image"])
20 pts = np.array(eval(args["coords"]), dtype = "float32")
21
22 # apply the four point tranform to obtain a "birds eye view" of
23 # the image
24 warped = four_point_transform(image, pts)
25
26 # show the original and warped images
27 cv2.imshow("Original", image)
28 cv2.imshow("Warped", warped)
29 cv2.waitKey(0)
```

The first thing we'll do is import our `four_point_transform` function on **Line 2**. I decided put it in the `pyimagesearch` sub-module for organizational purposes.

We'll then use NumPy for the array functionality, `argparse` for parsing command line arguments, and `cv2` for OpenCV bindings.

We parse our command line arguments on **Lines 8-12**. We'll use two switches, `--image`, which is the image that we want to apply the transform to, and `--coords`, which is the list of 4 points representing the region of the image we want to obtain a top-down, "birds eye view" of.

We then load the image on **Line 19** and convert the points to a NumPy array on **Line 20**.

Now before you get all upset at me for using the `eval` function, please remember, this is just an example. I don't condone performing a perspective transform this way.

And, as you'll see in next week's post, *I'll show you how to **automatically** determine the four points needed for the perspective transform* — no manual work on your part!

Next, we can apply our perspective transform on **Line 24**.

Finally, let's display the original image and the warped, top-down view of the image on **Lines 27-29**.

## Obtaining a Top-Down View of the Image

Alright, let's see this code in action.

Open up a shell and execute the following command:

OpenCV getPerspectiveTransform Example	Shell
1 \$ python transform_example.py --image images/example_01.png --coords "[(73, 239), (356, 117), (475, 265), (187, 443)]"	

You should see a top-down view of the notecard, similar to below:

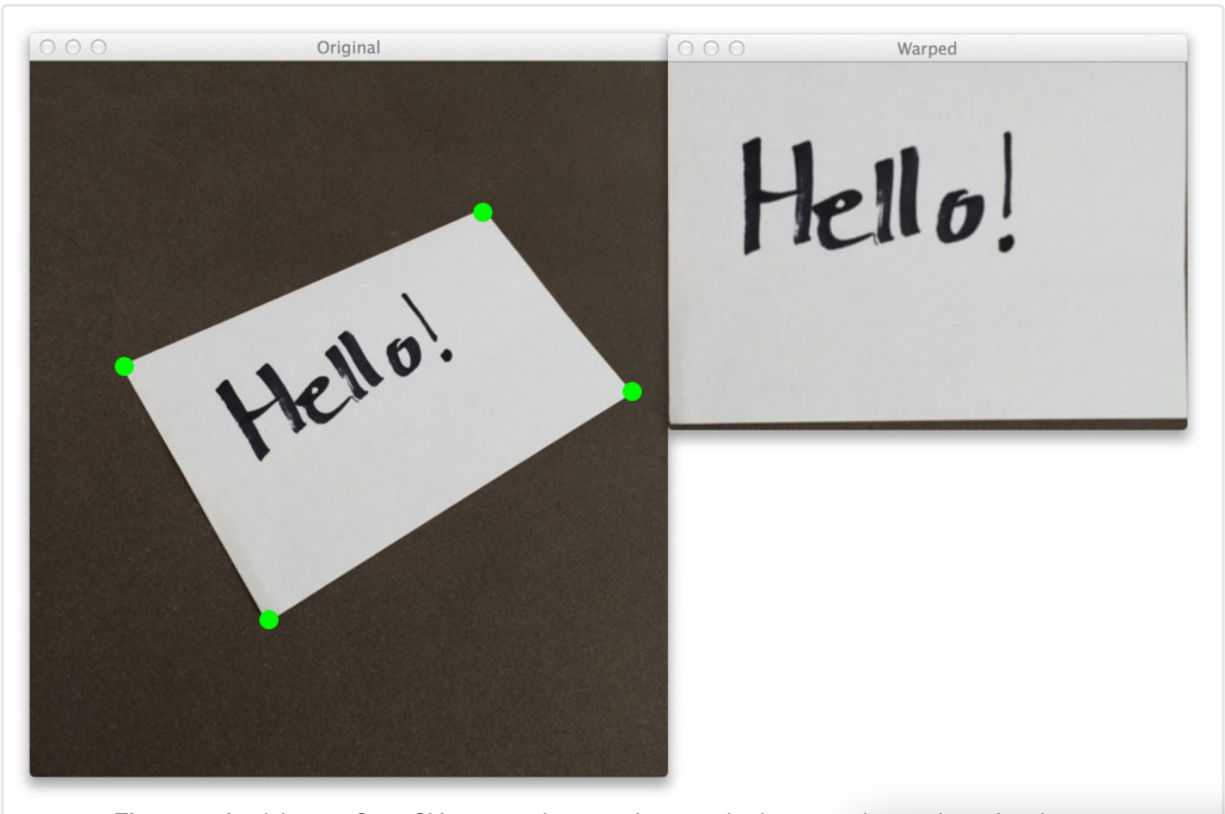


Figure 1: Applying an OpenCV perspective transform to obtain a "top-down" view of the image

Free 21-day crash course on computer vision & image search engines



Let’s try another image:

OpenCV getPerspectiveTransform Example	Shell
1 \$ python transform_example.py --image images/example_02.png --coords "[ (101, 185), (393, 151), (479, 323), (187, 441) ]"	



Figure 2: Applying an OpenCV perspective transform to warp the image and obtain a top-down view.

And a third for good measure:

OpenCV getPerspectiveTransform Example	Shell
1 \$ python transform_example.py --image images/example_03.png --coords "[ (63, 242), (291, 110), (361, 252), (78, 386) ]"	



Figure 3: Yet another OpenCV getPerspectiveTranform example to obtain a birds eye view of the image.

As you can see, we have successfully obtained a top-down, “birds eye view” of the notecard!

In some cases the notecard looks a little warped — this is because the angle the photo was taken at is quite severe. The closer we come to the 90-degree angle of “looking down” on the notecard, the better the results will be.

## Summary

In this blog post I provided an OpenCV `cv2.getPerspectiveTransform` example using Python.

I even shared code from my *personal library* on how to do it!

But the fun doesn’t stop here.

You know those iPhone and Android “scanner” apps that let you snap a photo of a document and then have it “scanned” into your phone?

That’s right — I’ll show you how to use the 4 point OpenCV getPerspectiveTransform example code to build one of those document scanner apps!

I’m definitely excited about it, I hope you are too.

Free 21-day crash course on computer vision & image search engines

Anyway, be sure to signup for the PyImageSearch Newsletter to hear when the post goes live!

## Downloads:



If you would like to download the code and images used in this post, please enter your email address in the form below. Not only will you get a .zip of the code, I'll also send you a **FREE 11-page Resource Guide** on Computer Vision and Image Search Engines, including **exclusive techniques** that I don't post on this blog! Sound good? If so, enter your email address and I'll send you the code immediately!

Email address:

DOWNLOAD THE CODE!

### Resource Guide (it's totally free).



Enter your email address below to get my **free 11-page Image Search Engine Resource Guide PDF**. Uncover **exclusive techniques** that I don't publish on this blog and start building image search engines of your own!

DOWNLOAD THE GUIDE!

🔖 opencv, perspective, transformation, warp

< Skin Detection: A Step-by-Step Example using Python and OpenCV

How to Build a Kick-Ass Mobile Document Scanner in Just 5 Minutes >

## 89 Responses to 4 Point OpenCV getPerspective Transform Example



**Vivek** February 8, 2015 at 4:49 am #

REPLY ↩

Hello Adrian,

This was really a wonderful post it gave me a very insightful knowledge of how to apply the perspective transform. I just have a very small question about the part where you were finding the maxHeight and maxWidth. For maxHeight (just considering heightA) you wrote

```
np.sqrt(((tr[1] - br[1]) ** 2) + ((tr[1] - br[1]) ** 2))
```

but i think that the height should be

```
np.absolute(tr[1] - br[1])
```

because you know this gives us the difference in the Y coordinate but the equation that you wrote gives us

1.4142 \* difference of the y coordinates. Why is so?



**Adrian Rosebrock** February 8, 2015 at 6:49 am #

REPLY ↩

Hi Vivek. The equation is utilizing the sum of squared differences (Euclidean distance), whereas the equation you proposed is just the absolute value of the differences (Manhattan distance). Try converting to the code to use the np.absolute function and let me know how the results look.



**Vertex** February 15, 2015 at 5:02 pm #

REPLY ↩

Hi Adrian,

Free 21-day crash course on computer vision & image search engines

is it possible, that you mixed up top and bottom in the comments of the function `order_points()` ? When I did an example `rect[0]` was BL, `rect[1]` was BR, `rect[2]` was TR and `rect[3]` was TL.



**Adrian Rosebrock** February 16, 2015 at 7:00 am #

REPLY ↩

Hi Vertex. Hm, I don't think so. The `dst` array assumes the ordering that I mentioned above and it's important to maintain that order. If the order was not maintained then the results from applying the perspective transform would not be correct.



**Vertex** February 16, 2015 at 1:04 pm #

REPLY ↩

Hi Adrian, thanks for your answer, I have to say I am newbie and I tried the following to get a better understanding:

```
import numpy as np

rect = np.zeros((4, 2), dtype = "float32")

# TL,BR,TR,BR
a = [[3,6],[3,3],[6,6],[6,3]]
rect[0] = np.argmin(np.sum(a,axis=1))
rect[2] = np.argmax(np.sum(a,axis=1))
rect[1] = np.argmin(np.diff(a,axis=1))
rect[3] = np.argmax(np.diff(a,axis=1))
print(rect)

[[ 1.  1.]
 [ 3.  3.]
 [ 2.  2.]
 [ 0.  0.]
```

I guess I got a faulty reasoning.



**Adrian Rosebrock** February 16, 2015 at 6:16 pm #

REPLY ↩

Ah, I see the problem. You are taking the `argmin/argmax`, but not grabbing the point associated with it. Try this, for example:

```
rect[0] = a[np.argmin(np.sum(a,axis=1))]
```

The `argmin/argmax` functions give you the index, which you can then apply to the original array.



**Nithin SK** June 4, 2015 at 9:49 am #

REPLY ↩

Hi Adrian! I'm a newbie.  
Spent lots of time mulling over this.  
Lines 53-57  
`dst = np.array([`  
`[0, 0],`  
`[maxWidth - 1, 0],`  
`[maxWidth - 1, maxHeight - 1],`  
`[0, maxHeight - 1]), dtype = "float32")`  
  
Isn't  
`[0,0]` – Bottom Left  
`[maxWidth - 1, 0]` – Bottom Right  
`[maxWidth - 1, maxHeight - 1]` – Top Right  
`[0, maxHeight - 1]` – Top Left

So it's bl,br,tr,tl? I'm a bit confused. Could you please explain?



**Adrian Rosebrock** June 4, 2015 at 10:52 am #

REPLY ↩

Hey Nithin, it's actually:

```
[0, 0] – top left
[maxWidth - 1, 0] – top right
```

Free 21-day crash course on computer vision & image search engines

[maxWidth – 1, maxHeight – 1] – bottom right

[0, maxHeight – 1] – bottom-left

Python arrays are zero-indexed, so we start counting from zero. Furthermore, the top-left corner of the image is located at point (0,0). For more information on the basics of image processing, including the coordinate system, I would definitely look at [Practical Python and OpenCV](#). I have an entire chapter dedicated to learning the coordinate system and image basics.



**Bogdan** April 12, 2016 at 4:21 am #

might be super simple, but I still don't get it why do you extract 1 from maxWidth and maxHeight for the tr, br and bl ?



**Adrian Rosebrock** April 13, 2016 at 7:01 pm #

In order to apply a perspective transform, the coordinates must be a *consistent* order. In this case, we supply them in top-left, top-right, bottom-right, and bottom-left order.



**Ken Wood** March 9, 2015 at 3:53 pm #

REPLY ↩

Great sample. My question is regarding the transformation matrix. Could it be used to tranform only a small region from the original image to a new image instead of warping the entire image? Say you used the Hello! example above but you wanted to only relocate the exclamation mark from the original image to a new image to end up with exactly the same output you have except without the “Hello” part, just the exclamation mark. I guess the question is whether you can use the TM directly without using the warping function.

Thanks!



**Adrian Rosebrock** March 9, 2015 at 3:56 pm #

REPLY ↩

Hi Ken, the transformation matrix M is simply a matrix. On its own, it cannot do anything. It's not until you plug it into the transformation function that the image gets warped.

As for only warping part of an image, you could certainly only transform the exclamation point. However, this would require you to find the four point bounding box around the exclamation point and then apply the transform — which is exactly what we do in the blog post. And that point you're better off transforming the entire index card and cropping out the exclamation point from there.



**Ken Wood** March 14, 2015 at 3:10 pm #

REPLY ↩

Hey Adrian,

I agree completely with what you say...I apologise, it was a poor example...what I was wondering about was how the mapping worked.

Fwiw, given the transformation matrix M you can calculate the mapped location of any source image pixel (x,y) (or a range of pixels) using:

dest(x) = [M11x + M12y + M13]/[M31x + M32y + M33]

dest(y) = [M21x + M22y + M23]/[M31x + M32y + M33]

Why bother?

I used this method to map a laser pointer from a keystoned camera image of a large screen image back onto the original image...allowing me to “draw” on the large screen.

Thanks!



**Adrian Rosebrock** March 14, 2015 at 4:18 pm #

REPLY ↩

Wow, that's really awesome Ken! Do you have an example video of your application in action? I would love to see it.



**wiem** April 17, 2015 at 1:04 pm #

REPLY ↩

Hi Adrian,

I am newbie in opencv.

is it possible to measuring angles in getPerspective Transform

can u give the function?

Free 21-day crash course on computer vision & image search engines



Thanks in advance



**Adrian Rosebrock** April 17, 2015 at 1:19 pm #

REPLY ↩

Hi Wiem, I'm not sure I understand what you're asking? If you want to get the angle of rotation for a bounding box, you might want to look into the cv2.minAreaRect function. I cover it a handful of times on the PyImageSearch blog, but you'll want to lookup the actual documentation on how to get the angle.



**wiem** April 18, 2015 at 11:26 am #

REPLY ↩

Hi Adrian, thanks for your answer  
I looked at "hello image" (original vs wraped) there is an angle of rotation.  
i want to know how to get the angle.  
sorry for my english  
  
Hope to hear from you  
Regards.



**Adrian Rosebrock** April 18, 2015 at 1:31 pm #

REPLY ↩

Hey, Wiem — please see my previous comment. To get the angle of rotation of the bounding box just use the cv2.minAreaRect function, which you can read more about [here](#). Notice how the angle of rotation is returned for the bounding box.



**wiem** April 18, 2015 at 1:45 pm #

REPLY ↩

Ups yeah.  
  
thanks for fast response.  
Thank you very much  
regards



**Aamir** April 27, 2015 at 10:04 am #

REPLY ↩

Is there equivalent function for order\_points(Rect ) in opencv for C++?  
  
P.S. Thanks for your tutorials.  
  
Thanks.



**Adrian Rosebrock** May 1, 2015 at 7:06 pm #

REPLY ↩

Hey Aamir, if there is a C++ version, I do not know of one. The order\_points function I created was entirely specific to Python and is not part of the core of OpenCV.



**palom** May 3, 2015 at 2:49 pm #

REPLY ↩

The code shows this error:  
  
"TypeError: eval() arg 1 must be a string or code object"  
  
thanks




**Adrian Rosebrock** May 4, 2015 at 6:19 am #

REPLY ↩


Hi Palom, make sure you wrap the coordinates in quotes: python transform\_example.py --image images/example\_01.png --coords "[(73, 239), (356, 117), (475, 265), (187, 443)]"

Free 21-day crash course on computer vision & image search engines

**Kunal Patil** December 4, 2015 at 5:21 pm <#>

REPLY ↩


That(wrapping in quotes) is already done in the code and the problem persists!

**Adrian Rosebrock** December 5, 2015 at 6:23 am <#>

REPLY ↩

Try removing the argument parsing code and then hardcoding the points, like this:


```
pts = np.array([(73, 239), (356, 117), (475, 265), (187, 443)], dtype = "float32")
```

**Singh** June 9, 2015 at 3:55 pm <#>

REPLY ↩


Hey, i am trying to implement the same thing in java using openCV but I cant seem to find the workaround of the numpy function can you help me out please.....My aim is to implement document scanner in java(ref :<http://www.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/>).....

Thanking you in anticipation

**Adrian Rosebrock** June 9, 2015 at 4:09 pm <#>

REPLY ↩


Hey Singh, I honestly don't do much work in Java, so I'm probably not the best person to answer that question. But you could probably use something like [jblas](#) or [colt](#).

**Tarik** August 11, 2015 at 1:18 pm <#>

REPLY ↩

Hey Adrian,

I believe your order\_points algorithm is not ideal, there are certain perspective in which it will fail, giving non contiguous points, even for a rectangular subject. A better approach is to find the top 2 points and 2 bottom points on the y axis, then sort these pairs on the x axis.

**Tarik** August 11, 2015 at 8:30 pm <#>

REPLY ↩

Actually my solution can also fail.


If the points in the input are contiguous, the best would be to choose a begin point meeting a choosing arbitrary ordering constraint, whilst conserving their original order.

Otherwise, a correct solution involve tracing a polygon without intersection, e.g. using the gift wrapping algorithm — simplified for a quadrilateral.

**Adrian Rosebrock** August 12, 2015 at 6:20 am <#>


REPLY ↩

Thanks for the tip Tarik!

**dan** September 5, 2015 at 11:29 am <#>

REPLY ↩


i am receiving an error message no module named pyimageseach.transform any idea what i have missed?

**Adrian Rosebrock** September 5, 2015 at 1:27 pm <#>

REPLY ↩

Please download the source code using the form at the bottom of this post that includes the pyimagesearch module.


Free 21-day crash course on computer vision & image search engines



**dan** September 6, 2015 at 1:55 pm #

REPLY ↩


Thanks for the quick reply.Okay i downloaded the file on my old laptop running windows 7 what do i do with it. Sorry for the stupid question I am a newbie and I am also old so i have two strikes against me,but if you learn from mistakes I should be a genius in no time!



**Adrian Rosebrock** September 7, 2015 at 8:24 am #

REPLY ↩

Please see the “Obtaining a Top-Down View of the Image” section of this post. In that section I provide the example commands you need to run the Python script.



**Ari** October 28, 2015 at 2:39 am #

REPLY ↩

Hi Adrian,


Thanks so much for the code. I have tried your code and running well on my MacBook OSX Yosemite, Python 2.7.6 and openCV 3.0.

I am just wondering that if we can improve it by automatically detect the four points. So, the variable input will be only the image. ☐ Should it be possible? What will be the algorithm to auto-detect the points?

☐

Thanks!


Ari



**Adrian Rosebrock** November 3, 2015 at 10:41 am #

REPLY ↩

Yep! We can absolutely automatically detect the points! Please see my post on [building a document scanner](#).




**HongQQ** November 11, 2015 at 3:42 pm #

REPLY ↩

Thanks for your kind sharing information about python & openCV.

Wonderful!




**nami** December 10, 2015 at 7:50 am #

REPLY ↩

hi adrian,

thanks for sharing..


what is mean index 0 and 1 in equation  $widthA = \sqrt{(br[0] - bl[0])^2 + (br[1] - bl[1])^2}$  ?



**Adrian Rosebrock** December 10, 2015 at 2:25 pm #

REPLY ↩

The 0 index is the *x*-coordinate and the 1 index is the *y*-coordinate.




**Ben** January 9, 2016 at 7:23 pm #

REPLY ↩

Hi!

Thank you for the tutorial. Have you got any tutorial how to transform perspective for the whole image? So user chooses 4 reference points and then the whole image is transformed (in result image there are some black fragments added). I know that I should calculate appropriate points for input image, but I have no idea how to do it. Can you help?

Regards



**Adrian Rosebrock** January 10, 2016 at 7:47 am #

REPLY ↩

Given your set of four input points, all you need to do is build your transformatic

Free 21-day crash course on computer vision & image search engines

this tutorial.



**Jan** January 22, 2016 at 6:55 pm #

REPLY ↩

Excellent tutoroal! Would it be possible to do the opposite? I mean, given a top-view of an image produce a distorsioned one?



**Adrian Rosebrock** January 23, 2016 at 2:10 pm #

REPLY ↩

Sure, the same principles apply — just modify the transformation matrix prior to applying it.



**Matheus Torquato** February 16, 2016 at 9:36 pm #

REPLY ↩

Hi, Adrian

I'm passing by just to say this post is really helpful and thank you very much for it.  
I'm starting studying Computer Vision and your blog it is really helping my development.  
Well done and keep going. =)

Cheers,  
Matheus Torquato



**Adrian Rosebrock** February 17, 2016 at 12:37 pm #

REPLY ↩

Thanks Matheus! ☐



**Matt** February 19, 2016 at 5:16 pm #

REPLY ↩

You have a slight typo in your description of line 21. It should say (i.e.  $y - x$ ).



**Damien Mac Namara** March 23, 2016 at 12:40 pm #

REPLY ↩

Hi Adrian,

Can the getPerspectiveTransform method return the angle that the image makes with the Y-axis?



**Adrian Rosebrock** March 24, 2016 at 5:15 pm #

REPLY ↩

Can you be more specific what you mean by “the angle the image makes with the y-axis”? I'm not sure I understand your question.



**Bozhidar Stanchev** April 6, 2016 at 10:01 am #

REPLY ↩

Do you happen to have the c++ version of this?



**Adrian Rosebrock** April 7, 2016 at 12:44 pm #

REPLY ↩

Sorry, I only have Python versions, I don't do much C++ coding anymore.



**Ayush** June 10, 2016 at 8:14 am #

REPLY ↩

I'm a complete newbie to OpenCV but shouldn't warping perspective off a Mat using the output of minAreaRect be a one-line command? I mean, you clearly have extracted some of these things out as 'utils' and a nice importable github repo for that, too, for which, we all thank you but don't you think that if it were so “regularly used by devs for their image proces  
\*really really\* honest, my “duckduckgo-ing” about warping off a rect perspective led me to thi

Free 21-day crash course on computer vision & image search engines



\*knew\* the code presented obviously works but I didn't immediately start using it \*ONLY AND ONLY\* because I \*believed\* there would be something to the effect of

warpedMat = cv2.warpPerspective(imageToWarp, areaToWarp)

Ultimately, on asking my colleague on how to do it, she suggested goin' ahead with your written utils only! ☐



**Christine B** June 17, 2016 at 3:25 pm #

REPLY ↩

Adrian,  
Thank you so much for your awesome tutorials!! I've been learning how to use the raspberry pi the past few weeks in order to make an automated testing system, and your tutorials have been so thorough and helpful. Thank you so so much for making these!!



**Adrian Rosebrock** June 18, 2016 at 8:14 am #

REPLY ↩

Thanks Christine, I'm happy i could help ☐



**Gabe** July 4, 2016 at 5:01 am #

REPLY ↩

I tried this code and it's pretty cool but it can't handle images like this:  
<http://vari-print.co.uk/wp-content/uploads/2013/05/business-cards-1.jpg>  
I tried to cut out the business card but I couldn't. I got some strange results. Why and how can I fix it?



**Adrian Rosebrock** July 5, 2016 at 1:52 pm #

REPLY ↩

In order to apply a perspective transform (and get reasonable results), you need to be able to detect all four corners of the object. If you cannot, your results will look very odd indeed. Please see [this post for more details](#) on applying perspective transforms to rectangular objects.



**Chris** July 20, 2016 at 5:30 am #

REPLY ↩

Hi Adrian. sorry for my English. ☐  
I'm newbie in opencv. thank you so much for awesome tut 😊  
i want crop this image <https://goo.gl/photos/kAmDRokUeLcqpycX7> with original on left and i want crop image on right. may u help me?  
thanks in advance



**Adrian Rosebrock** July 20, 2016 at 2:34 pm #

REPLY ↩

If you are trying to manually crop the region, I would use something like the GUI code I detail [in this blog post](#).



**Chris** July 21, 2016 at 2:31 am #

thanks your tut. it very excited!  
in this image, i want to get the road, and outside will be black, white or other color, because i'm researching about raspberry pi, and i want it process  
least as possible. do you have any idea?



**Karthikey** July 20, 2016 at 8:28 am #

REPLY ↩

Hi Adrian,  
  
I dont understand how (x-y) will be minimum for the top right corner.... Consider this square:  
tl= 0,0  
tr= 1,0  
br= 1,1  
bl =0,1

Free 21-day crash course on computer vision & image search engines

(x-y) is minimum for bl, ie. 0-1 = -1, nd not tr... Am i going wrong somewhere??



**Adrian Rosebrock** July 20, 2016 at 2:33 pm #

REPLY ↩

The origin (x, y)-coordinates start at the top-left corner and increase going down and to the right. For what it's worth, I recommend using [this updated version of the coordinate ordering function](#).



**Rima Borah** August 3, 2016 at 1:01 am #

REPLY ↩

I gave the input image of a irregular polygon formed after applying convex hull function to a set of points, supplying the four end pintps of the polygon in the order you mentioned. However the output I get is a blank screen. No polygon in it. Can you please tell how to give irregular polygons as input to the above code.



**Adrian Rosebrock** August 4, 2016 at 10:20 am #

REPLY ↩

It's hard to say without seeing an example, but I imagine your issue is that you did not supply the coordinates in top-left, top-right, bottom-left, and bottom-right order prior to applying the transformation.



**Abhishek Mishra** October 3, 2016 at 3:09 pm #

REPLY ↩

Adrian, great post. I was trying to build this for a visiting card. My problem is that the card could be aligned in any arbitrary angle with respect to the desk as well as with respect to the camera.

When I use this logic, there are alignments at which the resultant image is like a rotated and shrieked one. In your case, the image is rotated towards the right to make it look correct. However, if the original card was rotated clockwise 90 degrees, then the logic of top right, top left does not work correctly.

I tried using the "width will always be greater than height" approach but that too fails at times.

Any suggestions?



**Adrian Rosebrock** October 4, 2016 at 6:56 am #

REPLY ↩

This sounds like it may be a problem with the coordinate ordering prior to applying the perspective transform. I would suggest instead suggest using the [updated implementation](#).



**sandesh chand** October 4, 2016 at 10:19 am #

REPLY ↩

Hello Adrain,  
I am quit new in opencv. i am working in Imageprocessing progam in python 2.7.Actually i am facing Problem during cropping of Image. I am working in differrent Images having some black side background. The Picture is taken by camera maualally so the Image is also different in size. I want to crop the Image and separte it from balck Background. could you suggest how can i make a program that can detect the Image first and crop automatically.  
Thanks



**Adrian Rosebrock** October 6, 2016 at 7:02 am #

REPLY ↩

Hey Sandesh — do you have any examples of images that you're working with?



**Manuel** October 16, 2016 at 1:54 pm #

REPLY ↩

Hi Adrian,  
I have a doubt when thinking about the generalization of this example regarding the destiny points. This example is specifically aimed to quadrangular objects, right? I mean, you get the destiny image points because you simple say "how wide it will be a box let's get the max height max width and that's it".

Free 21-day crash course on computer vision & image search engines

But wouldn't be so easy if the original object would have had a different shape, right?

Thanks.



**Adrian Rosebrock** October 17, 2016 at 4:05 pm #

REPLY ↩

Yes, this example presumes that the object you are trying to transform is rectangular. I'm not sure I understand what you mean by the "generalization" of this technique?



**Siladittya Manna** December 8, 2016 at 3:17 pm #

REPLY ↩

I want to know if there is a way that the program can automatically detect the corners??



**Adrian Rosebrock** December 10, 2016 at 7:23 am #

REPLY ↩

Absolutely. Please see [the followup blog post to this one](#).



**rene godbout** December 14, 2016 at 9:16 pm #

REPLY ↩

When the rectangular dimensions of the source target are known, the result is much better if you input x,y values for the destination image that conform to the x/y ratio of the source target. The estimation of the output size described here will only be perfect if the target is perpendicular to the viewpoint. This is why the distortion increases in proportion to the angle(s) away from perpendicular. Otherwise, you have to know the focal length of the camera, focus distance, etc. (much more complicated calculations...) to estimate the "real" proportions of the target's dimensions (or x/y ratio).



**rene godbout** December 14, 2016 at 10:16 pm #

REPLY ↩

As an example, the page size in your samples looks like "legal" 8.5 x 14 paper. Once this is established, if you replace the maxHeight calculation with "maxHeight = (maxWidth \* 8) / 14", the output image(s) are much better looking as far as the x/y ratio is concerned (no apparent distortion on the last sample). Of course, one must know the target's x/y ratio...



**Adrian Rosebrock** December 18, 2016 at 9:09 am #

REPLY ↩

Good point, thanks for sharing Rene. If the aspect ratio is know then the output image can be much better. There are methods that can attempt to (automatically) determine the aspect ratio but they are outside the scope of this post. I'll try to do a tutorial on them in the future.



**Mazine** January 28, 2017 at 10:09 am #

REPLY ↩

Hi, this is really nice ! What bugs me is the way to find those four corners since the picture does not have the right perspective



**Adrian Rosebrock** January 29, 2017 at 2:45 pm #

REPLY ↩

If you want to automatically find the four corners, [take a look at this blog post](#).



**Patrick** February 5, 2017 at 5:04 pm #

REPLY ↩


Hi Andrian, thanks for the tutorial. I cannot help noticing that you mentioned the difference of the coordinates is (x-y), but np.diff([x, y]) actually returns (y-x).




**kamell** February 6, 2017 at 6:34 am #

REPLY ↩

Free 21-day crash course on computer vision & image search engines



Traceback (most recent call last):  
File “transformexple.py”, line 2, in  
from pyimagesearch.transform import four\_point\_transform  
ImportError: No module named pyimagesearch.transform  
how can install it ????




Adrian Rosebrock

February 7, 2017 at 9:12 am #


REPLY

Make sure you use the “Downloads” section to download the source code associated with this blog post. It includes the “pyimagesearch” directory structure for the project.



Ahmed March 2, 2017 at 4:14 pm #

It is really nice way to get the bird’s eye view, but when I tried to use it in my algorithm i failed to get the bird’s eye  
I want to get a top view of a lane and a car ?




Adrian Rosebrock

March 4, 2017 at 9:44 am #

REPLY


This method is an example of applying a top-down transform using pre-defined coordinates. To automatically determine those coordinates you’ll have to write a script that detects the top of a car.



Christian March 8, 2017 at 12:20 pm #

Hi Adrian,

Thanks a lot for the post, this is great for the app I’m trying to build. The thing is that I’m translating all your code to Java and I don’t know if everything is correctly translated because the image I get after the code is rotated 90° and flipped... I’m investigating what could be happening but maybe you think of something that could be happening. Thanks again for the post.




Adrian Rosebrock

March 8, 2017 at 12:56 pm #


REPLY

Hi Christian — thanks for the comment. However, I haven’t used the OpenCV + Java bindings in a long time, so I’m not sure what the exact issue is.



Christian March 9, 2017 at 3:57 am #

Well, thanks anyway. I’ll giving it a second shot today ☐




Arvind Mohan

March 22, 2017 at 3:10 am #

REPLY

Hi Adrian,

Would affine transform make any sense in this context?




Adrian Rosebrock

March 22, 2017 at 6:34 am #

REPLY

[This page](#) provides a nice discussion on affine vs. perspective transforms. An affine transform is a special case of a perspective transform. In this case perspective transforms are more appropriate — we can make them even better if we can estimate the aspect ratio of the object we are trying to transform. See the link for more details.




Muhammed March 27, 2017 at 2:43 pm #

There is another idea to order four points by  
comparing centroid with the four points

Free 21-day crash course on computer vision & image search engines



there is one limitation when oriantation object =45 degree



Michael

April 29, 2017 at 10:02 am


#

REPLY ↩

Hi Adrian,

Thanks for sharing the example, very inspiring!

Perspective transform works great here as the object being warped is 2D. What if the object is 3D, like a cylinder, <http://cdn.free-power-point-templates.com/articles/wp-content/uploads/2012/07/3d-cilinder-wrap-text-ppt.jpg>?



Bruno Andrade

June 16, 2017 at 2:18 pm

#

REPLY ↩

Hi Adrian,

I followed your tutorial in one of my projects but the object height decreases.  
i used your tuturiel 'Building a Pokedex in Python' part4 and 5 in order to have the corners points.

<http://imgur.com/a/vt4Er>

Can you tell me what im doing wrong.  
Thanks in advance.

Trackbacks/Pingbacks

[Recognizing digits with OpenCV and Python - PyImageSearch](#) - February 13, 2017  
[...] with a rectangular shape — the largest rectangular region should correspond to the LCD. A perspective transform will give me a nice extraction of the [...]

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Resource Guide (it's totally free).

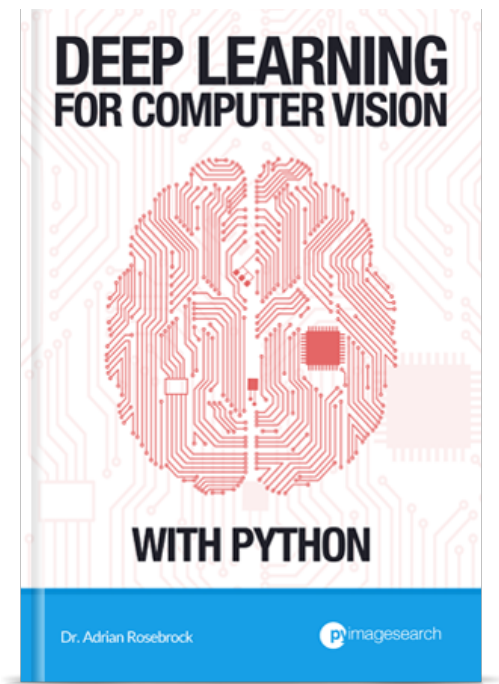


Click the button below to get my **free 11-page Image Search Engine Resource Guide PDF**. Uncover **exclusive techniques** that I don't publish on this blog and start building image search engines of your own.

Download for Free!

Deep Learning for Computer Vision with Python Book

Free 21-day crash course on computer vision & image search engines



You're interested in deep learning and computer vision, *but you don't know how to get started*. Let me help. [My new book will teach you all you need to know about deep learning.](#)

CLICK HERE TO PRE-ORDER MY NEW BOOK

You can detect faces in images & video.



Are you interested in **detecting faces in images & video**? But **tired of Googling for tutorials** that *never work*? Then let me help! I guarantee that my new book will turn you into a **face detection ninja** by the end of this weekend. [Click here to give it a shot yourself.](#)

CLICK HERE TO MASTER FACE DETECTION

PyImageSearch Gurus: NOW ENROLLING!

The PyImageSearch Gurus course is *now enrolling!* Inside the course you'll learn how to perform:


- Automatic License Plate Recognition (ANPR)
- Deep Learning
- Face Recognition
- *and much more!*

Click the button below to learn more about the course, take a tour, and get 10 (FREE) sample lessons.

TAKE A TOUR & GET 10 (FREE) LESSONS

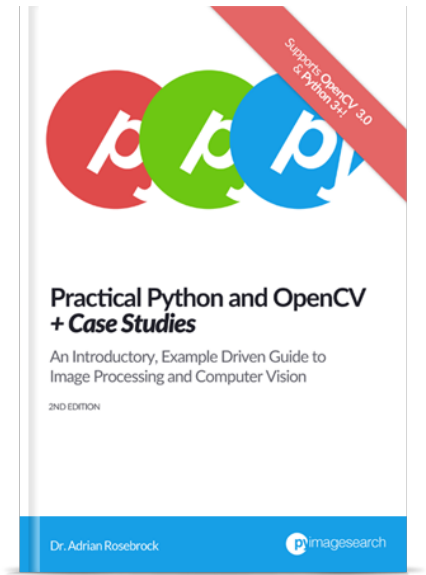
Hello! I'm Adrian Rosebrock.

Free 21-day crash course on computer vision & image search engines



I'm an entrepreneur and Ph.D who has launched two successful image search engines, [ID My Pill](#) and [Chic Engine](#). I'm here to share my tips, tricks, and hacks I've learned along the way.


Learn computer vision in a single weekend.



Want to learn computer vision & OpenCV? I can teach you in a **single weekend**. I know. It sounds crazy, but it's no joke. My new book is your **guaranteed, quick-start guide** to becoming an OpenCV Ninja. So why not give it a try? [Click here to become a computer vision ninja](#).

CLICK HERE TO BECOME AN OPENCV NINJA

Subscribe via RSS




Never miss a post! Subscribe to the PyImageSearch RSS Feed and keep up to date with my image search engine tutorials, tips, and tricks

POPULAR

<b>Install OpenCV and Python on your Raspberry Pi 2 and B+</b> FEBRUARY 23, 2015
<b>Home surveillance and motion detection with the Raspberry Pi, Python, OpenCV, and Dropbox</b> JUNE 1, 2015
<b>Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3</b> APRIL 18, 2016
<b>How to install OpenCV 3 on Raspbian Jessie</b> OCTOBER 26, 2015
<b>Basic motion detection and tracking with Python and OpenCV</b> MAY 25, 2015
<b>Accessing the Raspberry Pi Camera with OpenCV and Python</b> MARCH 30, 2015
<b>Install OpenCV 3.0 and Python 2.7+ on Ubuntu</b> JUNE 22, 2015

Search



Find me on [Twitter](#), [Facebook](#), [Google+](#), and [LinkedIn](#).  
© 2017 PyImageSearch. All Rights Reserved.

Free 21-day crash course on computer vision & image search engines