



Building a Pokedex in Python: Indexing our Sprites using Shape Descriptors (Step 3 of 6)

by **Adrian Rosebrock** on April 7, 2014 in **Building a Pokedex**, **Examples of Image Search Engines**, **Tutorials**



Using shape descriptors to quantify an object is a lot like playing Who's that Pokemon as a kid.

So, how is our Pokedex going to “know” what Pokemon is in an image? How are we going to describe each Pokemon? Are we going to characterize the color of the Pokemon? The texture? Or the shape?

Well, do you remember playing Who's that Pokemon as a kid?

You were able to identify the Pokemon based only on its outline and silhouette.

We are going to apply the same principles in this post and quantify the outline of Pokemon using shape descriptors.

Looking for the source code to this post?

[Jump right to the downloads section.](#)

You might already be familiar with some shape descriptors, such as Hu moments. Today I am going to introduce you to a more powerful shape descriptor — Zernike moments, based on Zernike polynomials that are orthogonal to the unit disk.

Sound complicated?

Trust me, it's really not. With just a few lines of code I'll show you how to compute Zernike moments with ease.

OpenCV and Python versions:

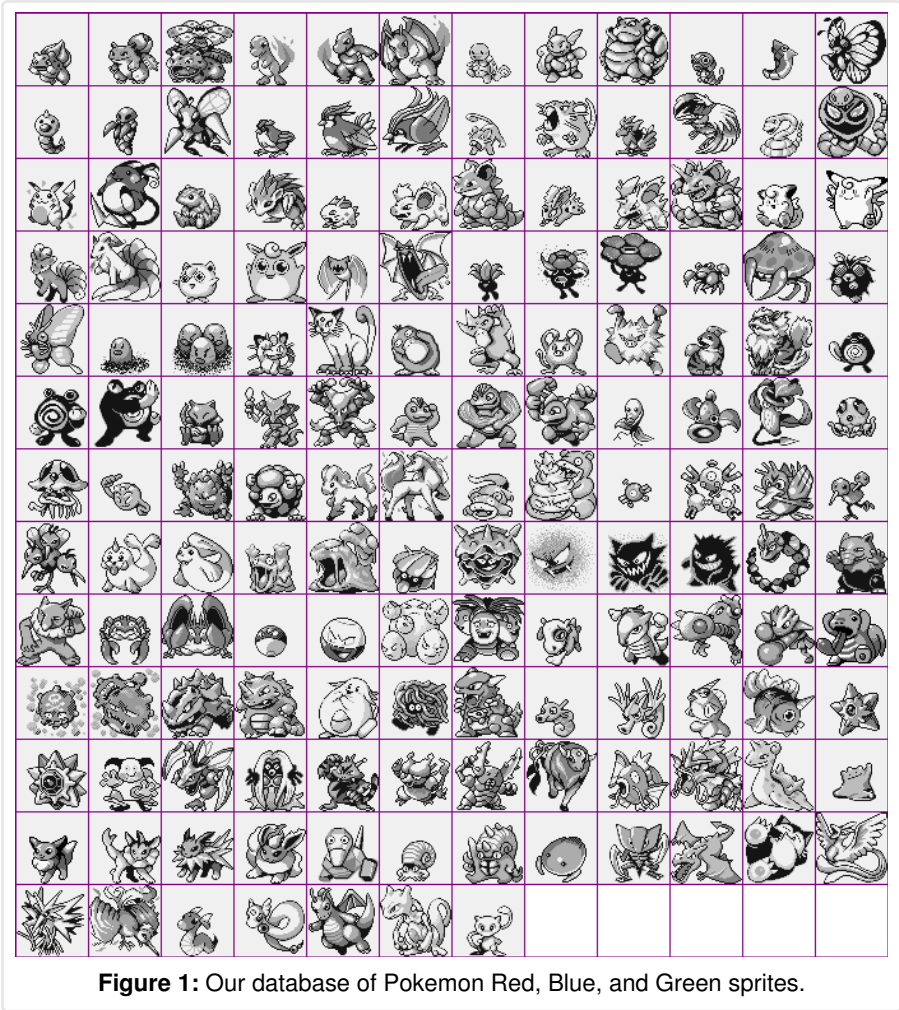
This example will run on **Python 2.7** and **OpenCV 2.4.X**.

Previous Posts

This post is part of an on-going series of blog posts on how to build a real-life Pokedex using Python, OpenCV, and computer vision and image processing techniques. If this is the first post in the series that you are reading, go ahead and read through it (there is a lot of awesome content in here on how to utilize shape descriptors), but then go back to the previous posts for some added context.

- **Step 1:** [Building a Pokedex in Python: Getting Started \(Step 1 of 6\)](#)
- **Step 2:** [Building a Pokedex in Python: Scraping the Pokemon Sprites \(Step 2 of 6\)](#)

Descriptors



At this point, we already have [our database of Pokemon sprite images](#). We gathered, scraped, and downloaded our sprites, but now we need to quantify them in terms of their outline (i.e. their shape).

Remember playing “Who’s that Pokemon?” as a kid? That’s essentially what our shape descriptors will be doing for us.

For those who didn’t watch Pokemon (or maybe need their memory jogged), the image at the top of this post is a screenshot from the Pokemon TV show. Before going to commercial break, a screen such as this one would pop up with the outline of the Pokemon. The goal was to guess the name of the Pokemon based on the outline alone.

This is essentially what our Pokedex will be doing — playing Who’s that Pokemon, but in an automated fashion. And with computer vision and image processing techniques.

Zernike Moments

Before diving into a lot of code, let’s first have a quick review of Zernike moments.

Image moments are used to describe objects in an image. Using image moments you can calculate values such as the area of the object, the centroid (the center of the object, in terms of x, y coordinates), and information regarding how the object is rotated. Normally, we calculate image moments based on the contour or outline of an image, but this is not a requirement.

OpenCV provides the [HuMoments](#) function which can be used to characterize the structure and shape of an object. However, a more powerful shape descriptors can be found in the [mahotas](#) package — [zernike_moments](#). Similar to Hu moments, Zernike moments are used to describe the shape of an object; however, since the Zernike polynomials are orthogonal to each other, there is no redundancy of information between the moments.

One caveat to look out for when utilizing Zernike moments for shape description is the scaling and translation of the object in the image. Depending on where the image is translated in the image, your Zernike moments will be drastically different. Similarly, depending on how large or small (i.e. how your object is scaled) in the image, your Zernike moments will not be identical. However, the magnitudes of the Zernike moments are independent of the rotation of the object, which is an extremely nice property when working with shape descriptors.

In order to avoid descriptors with different values based on the translation and scaling of the image, we normally first perform segmentation. That is, we segment the foreground (the object in the image we are interested in) from the background (the “noise”, or the part of the image we do not want to describe). Once we have the segmentation, we can form a tight bounding box around the object and crop it out, obtaining translation invariance.

Finally, we can resize the object to a constant $N \times M$ pixels, obtaining scale invariance.

From there, it is straightforward to apply Zernike moments to characterize the shape of th

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

riance prior to applying Zernike moments.

The Zernike Descriptor

Alright, enough overview. Let’s get our hands dirty and write some code.

Zernike moments in Python, OpenCV, and mahotas	Python
<pre>1 # import the necessary packages 2 import mahotas 3 4 class ZernikeMoments: 5 def __init__(self, radius): 6 # store the size of the radius that will be 7 # used when computing moments 8 self.radius = radius 9 10 def describe(self, image): 11 # return the Zernike moments for the image 12 return mahotas.features.zernike_moments(image, self.radius)</pre>	

As you may know from the [Hobbits and Histograms](#) post, I tend to like to define my image descriptors as classes rather than functions. The reason for this is that you rarely ever extract features from a single image alone. Instead, you extract features from a dataset of images. And you are likely utilizing the exact same parameters for the descriptors from image to image.

For example, it wouldn’t make sense to extract a grayscale histogram with 32 bins from image #1 and then a grayscale histogram with 16 bins from image #2, if your intent is to compare them. Instead, you utilize identical parameters to ensure you have a *consistent representation* across your entire dataset.

That said, let’s take this code apart:

- **Line 2:** Here we are importing the mahotas package which contains many useful image processing functions. This package also contains the implementation of our Zernike moments.
- **Line 4:** Let’s define a class for our descriptor. We’ll call it ZernikeMoments.
- **Lines 5-8:** We need a constructor for our ZernikeMoments class. It will take only a single parameter — the radius of the polynomial in pixels. The larger the radius, the more pixels will be included in the computation. This is an important parameter and you’ll likely have to tune it and play around with it to obtain adequately performing results if you use Zernike moments outside this series of blog posts.
- **Lines 10-12:** Here we define the describe method, which quantifies our image. This method requires an image to be described, and then calls the mahotas implementation of zernike_moments to compute the moments with the specified radius, supplied in **Line 5**.

Overall, this isn’t much code. It’s mostly just a wrapper around the mahotas implementation of zernike_moments. But as I said, I like to define my descriptors as classes rather than functions to ensure the consistent use of parameters.

Next up, we’ll index our dataset by quantifying each and every Pokemon sprite in terms of shape.

Indexing Our Pokemon Sprites

Now that we have our shape descriptor defined, we need to apply it to every Pokemon sprite in our database. This is a fairly straightforward process so I’ll let the code do most of the explaining. Let’s open up our favorite editor, create a file named index.py, and get to work:

Indexing Pokemon Sprites in Python, OpenCV, and mahotas	Python
<pre>1 # import the necessary packages 2 from pyimagesearch.zernikemoments import ZernikeMoments 3 import numpy as np 4 import argparse 5 import cPickle 6 import glob 7 import cv2 8 9 # construct the argument parser and parse the arguments 10 ap = argparse.ArgumentParser() 11 ap.add_argument("-s", "--sprites", required = True, 12 help = "Path where the sprites will be stored") 13 ap.add_argument("-i", "--index", required = True, 14 help = "Path to where the index file will be stored") 15 args = vars(ap.parse_args()) 16 17 # initialize our descriptor (Zernike Moments with a radius 18 # of 21 used to characterize the shape of our pokemon) and 19 # our index dictionary 20 desc = ZernikeMoments(21) 21 index = {}</pre>	

Lines 2-8 handle importing the packages we will need. I put our ZernikeMoments class in the pyimagesearch sub-module for organizational sake. We will make use of numpy when constructing multi-dimensional arrays, argparse for parsing command line arguments, cPickle for writing our index to file, glob for grabbing the paths to our sprite images, and cv2 for our OpenCV functions.

Then, **Lines 10-15** parse our command line arguments. The --sprites switch is the path to

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

Line 20 handles initializing our ZernikeMoments descriptor. We will be using a radius of 21 pixels. I determined the value of 21 pixels after a few experimentations and determining which radius obtained the best performing results.

Finally, we initialize our index on Line 21. Our index is a built-in Python dictionary, where the key is the filename of the Pokemon sprite and the value is the calculated Zernike moments. All filenames are unique in this case so a dictionary is a good choice due to its simplicity.

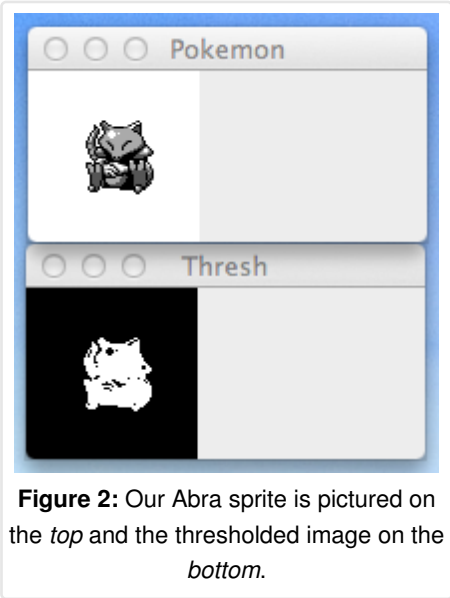
Time to quantify our Pokemon sprites:

Indexing Pokemon Sprites in Python, OpenCV, and mahotas	Python
<pre>23 # loop over the sprite images 24 for spritePath in glob.glob(args["sprites"] + "/*.png"): 25 # parse out the pokemon name, then load the image and 26 # convert it to grayscale 27 pokemon = spritePath[spritePath.rfind("/") + 1:].replace(".png", "") 28 image = cv2.imread(spritePath) 29 image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) 30 31 # pad the image with extra white pixels to ensure the 32 # edges of the pokemon are not up against the borders 33 # of the image 34 image = cv2.copyMakeBorder(image, 15, 15, 15, 15, 35 cv2.BORDER_CONSTANT, value = 255) 36 37 # invert the image and threshold it 38 thresh = cv2.bitwise_not(image) 39 thresh[thresh > 0] = 255</pre>	

Now we are ready to extract Zernike moments from our dataset. Let’s take this code apart and make sure we understand what is going on:

- **Line 24:** We use glob to grab the paths to our all Pokemon sprite images. All our sprites have a file extension of .png. If you’ve never used glob before, it’s an extremely easy way to grab the paths to a set of images with common filenames or extensions. Now that we have the paths to the images, we loop over them one-by-one.
- Line 27: The first thing we need to do is extract the name of the Pokemon from the filename. This will serve as our unique key into the index dictionary.
- **Line 28 and 29:** This code is pretty self-explanatory. We load the current image off of disk and convert it to grayscale.
- **Line 34 and 35:** Personally, I find the name of the copyMakeBorder function to be quite confusing. The name itself doesn’t really describe what it does. Essentially, the copyMakeBorder “pads” the image along the north, south, east, and west directions of the image. The first parameter we pass in is the Pokemon sprite. Then, we pad this image in all directions by 15 white (255) pixels. This step isn’t necessarily required, but it gives you a better sense of the thresholding on **Line 38**.
- **Line 38 and 39:** As I’ve mentioned, we need the outline (or mask) of the Pokemon image prior to applying Zernike moments. In order to find the outline, we need to apply segmentation, discarding the background (white) pixels of the image and focusing only on the Pokemon itself. This is actually quite simply — all we need to do is flip the values of the pixels (black pixels are turned to white, and white pixels to black). Then, any pixel with a value greater than zero (black) is set to 255 (white).

Take a look at our thresholded image below:



This process has given us the mask of our Pokemon. Now we need the outermost contours of the mask — the actual outline of the Pokemon.

Indexing Pokemon Sprites in Python, OpenCV, and mahotas	Python
<pre>41 # initialize the outline image, find the outermost 42 # contours (the outline) of the pokemone, then draw 43 # it 44 outline = np.zeros(image.shape, dtype = "uint8") 45 (cnts, _) = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, 46 cv2.CHAIN_APPROX_SIMPLE) 47 cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[0] 48 cv2.drawContours(outline, [cnts], -1, 255, -1)</pre>	

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

1

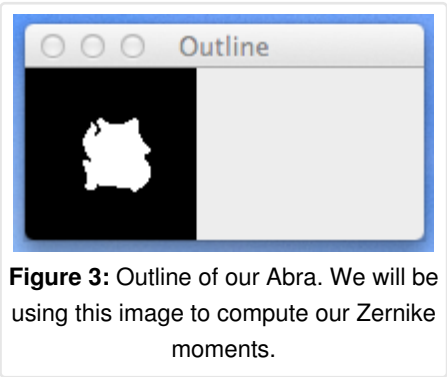
3

line on **Line 44** and fill it with zeros with the same

Then, we make a call to `cv2.findContours` on Line 45. The first argument we pass in is our thresholded image, followed by a flag `cv2.RETR_EXTERNAL` telling OpenCV to find only the outermost contours. Finally, we tell OpenCV to compress and approximate the contours to save memory using the `cv2.CHAIN_APPROX_SIMPLE` flag.

As I mentioned, we are only interested in the largest contour, which corresponds to the outline of the Pokemon. So, on **Line 47** we sort the contours based on their area, in descending order. We keep only the largest contour and discard the others.

Finally, we draw the contour on our outline image using the `cv2.drawContours` function. The outline is drawn as a filled in mask with white pixels:



We will be using this outline image to compute our Zernike moments.

Computing Zernike moments for the outline is actually quite easy:

Indexing Pokemon Sprites in Python, OpenCV, and mahotas	Python
<pre>50 # compute Zernike moments to characterize the shape 51 # of pokemon outline, then update the index 52 moments = desc.describe(outline) 53 index[pokemon] = moments</pre>	

On **Line 52** we make a call to our `describe` method in the `ZernikeMoments` class. All we need to do is pass in the outline of the image and the `describe` method takes care of the rest. In return, we are given the Zernike moments used to characterize and quantify the shape of the Pokemon.

So how are we quantifying and representing the shape of the Pokemon?

Let's investigate:

Indexing Pokemon Sprites in Python, OpenCV, and mahotas	Python
<pre>1 >>> moments.shape 2 (25,)</pre>	

Here we can see that our feature vector is of 25-dimensionality (meaning that there are 25 values in our list). These 25 values represent the contour of the Pokemon.

We can view the values of the Zernike moments feature vector like this:

Indexing Pokemon Sprites in Python, OpenCV, and mahotas	Python
<pre>1 >>> moments 2 [0.31830989 0.00137926 0.24653755 0.03015183 0.00321483 0.03953142 3 0.10837637 0.00404093 0.09652134 0.005004 0.01573373 0.0197918 4 0.04699774 0.03764576 0.04850296 0.03677655 0.00160505 0.02787968 5 0.02815242 0.05123364 0.04502072 0.03710325 0.05971383 0.00891869 6 0.02457978]</pre>	

So there you have it! The Pokemon outline is now quantified using only 25 floating point values! Using these 25 numbers we will be able to disambiguate between all of the original 151 Pokemon.

Finally on **Line 53**, we update our index with the name of the Pokemon as the key and our computed features as our value.

The last thing we need to do is dump our index to file so we can use when we perform a search:

Indexing Pokemon Sprites in Python, OpenCV, and mahotas	Python
<pre>55 # write the index to file 56 f = open(args["index"], "w") 57 f.write(cPickle.dumps(index)) 58 f.close()</pre>	

To execute our script to index all our Pokemon sprites, issue the following command:

Indexing Pokemon Sprites in Python, OpenCV, and mahotas	
<pre>1 \$ python index.py --sprites sprites --index index.cpickle</pre>	

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

1

Later in this series of blog posts, I'll show you how to automatically extract a Pokemon from a Game Boy screen and then compare it to our index.

Summary

In this blog post we explored Zernike moments and how they can be used to describe and quantify the shape of an object.

In this case, we used Zernike moments to quantify the outline of the original 151 Pokemon. The easiest way to think of this is playing “Who’s that Pokemon?” as a kid. You are given the outline of the Pokemon and then you have to guess what the Pokemon is, using only the outline alone. We are doing the same thing — only we are doing it automatically.

This process of describing and quantifying a set of images is called “indexing”.

Now that we have our Pokemon quantified, I’ll show you how to search and identify Pokemon later in this series of posts.

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:

Your 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!

Your email address

DOWNLOAD THE GUIDE!

🔖 descriptors, hu moments, mahotas, opencv, pokedex, pokemon, shape, zernike moments

< Building a Pokedex in Python: Scraping the Pokemon Sprites (Step 2 of 6)

Building a Pokedex in Python: Finding the Game Boy Screen (Step 4 of 6) >

18 Responses to *Building a Pokedex in Python: Indexing our Sprites using Shape Descriptors (Step 3 of 6)*



Hamid August 31, 2015 at 3:14 am #

REPLY ↩

Dear Adrian,

I need to appreciate your kind sharing and wise approaches here. Actually, I am doing my PhD and on the way about one of my mini projects need to reconrtcut the images from Zernike moments extracted from Mahotas toolbox that you introduced. There is one code I found on the web but relate to matlab and am not very sure of that. I wonder if you could kindly give me an advice . Thank you very much.

Best regards.

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

1

3

REPLY ↩

Congrats on doing your PhD Hamid, that's very exciting! As for Zernike Moments, I would suggest looking at the [source code directly of the Mahotas implementation](#). You'll probably need to modify the code to do the reconstruction. I would also suggest sending a message on GitHub to Luis, the developer and maintainer of Mahtoas — he is an awesome guy and knows a lot about CV.



Rishit Bansal February 5, 2016 at 1:27 pm #

REPLY ↩

So even if the image of the Pokemon to be determined is laterally inverted, will the zernike moments search still find it as the normal image in the index?



Adrian Rosebrock February 6, 2016 at 9:57 am #

REPLY ↩

Correct, Zernike moments are invariant under rotation.



siyer December 8, 2016 at 2:52 am #

REPLY ↩

Hi Adrian

While calculating the zernlike moments how to determine or end up using the the right radius value for the specific set of images ?

Could you elaborate on this item of your post pls

Lines 5-8: We need a constructor for our ZernikeMoments class. It will take only a single parameter — the radius of the polynomial in pixels. The larger the radius, the more pixels will be included in the computation. This is an important parameter and you'll likely have to tune it and play around with it to obtain adequately performing results if you use Zernike moments outside this series of blog posts.



Adrian Rosebrock December 10, 2016 at 7:27 am #

REPLY ↩

The easiest way to do this is to compute the cv2.minEnclosingCircle of the contour. This will give you a radius that encapsulates the *entire* object. Or, if you have *a priori* knowledge about the problem you can hardcode it. I discuss this more [inside the PyImageSearch Gurus course](#).



Aven Kidur February 8, 2017 at 5:02 pm #

REPLY ↩

I just wanted to say your posts are so well written. Even though I learned something I didn't know, I did not feel lost or confused for a moment. Instead of aversion masquerading as boredom, it is **exciting** and **fun** to read the next line.

You may be naturally gifted, but I suspect you've put great energy and care into crafting these lessons.

Thank you



Adrian Rosebrock February 10, 2017 at 2:06 pm #

REPLY ↩

Thank you for the kind words Aven, I really appreciate that ☐ Comments like these are a real pleasure to read and make my day.



Oscar February 21, 2017 at 11:23 am #

REPLY ↩

Very nice tutorial Adrian!

One question, can you index multiple images for 1 pokemon?

If so, I can index the images of every generation (gold/silver,black/white,...) and my pokedex will become smarter. This way it might also be able to process a random image of a Pokemon and still be accurate.

Or am I wrong?

Thanks in advance!

Oscar

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

1

3

REPLY ↩

Hi Oscar — you can certainly generate as many indexes as you want.



Babak Abad May 19, 2017 at 5:55 pm #

REPLY ↩

Thanks a a lot. I was searching for Zernik moments. I found no thing except one web site providing MATLAB codes which was unclear to me. You explain all things practically, specially where results of codes are provided. I appreciate you dear Adrian.



Adrian Rosebrock May 21, 2017 at 5:18 am #

REPLY ↩

Thank you, I'm really happy to hear that I could help ☐



kaia June 8, 2017 at 4:36 pm #

REPLY ↩

Dear Adrian,

I am getting the following problem when I run the code for this lesson:

```
$ python index.py --sprites sprites --index index.cpickle
Traceback (most recent call last):
File "index.py", line 48, in
(cnts, _) = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
ValueError: too many values to unpack
```

Please suggest solution.

Thanks



kaia June 8, 2017 at 4:51 pm #

REPLY ↩

Dear Adrian,

Changing the 45th line to the following did it for me.

```
_cnts,_ = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Could you explain what the content of cnts might be? Why is the sorted function required if cv2.RETR_EXTERNAL flag is used? For our case, the image contains only one pokemon sprite. Should it give more than one external contour?

Thanks for the tutorial.



Adrian Rosebrock June 9, 2017 at 1:38 pm #

REPLY ↩

This blog post assumed you are using OpenCV 2.4; however, you are using OpenCV 3, where the cv2.findContours return signature changed. You can read more about this change [here](#). As for sorting the contours, sometimes there is “noise” in our input image. Sorting just ensures we grab the largest region.

Trackbacks/Pingbacks

- [Building a Pokedex in Python: Finding the Game Boy Screen \(Step 4 of 6\) - PyImageSearch](#) - April 23, 2014
[...] Step 3: Building a Pokedex in Python: Indexing our Sprites using Shape Descriptors (Step 3 of 6) [...]
- [Python and OpenCV Example: Warp Perspective and Transform](#) - May 5, 2014
[...] Step 3: Building a Pokedex in Python: Indexing our Sprites using Shape Descriptors (Step 3 of 6) [...]
- [Comparing Shape Descriptors for Similarity using Python and OpenCV](#) - May 19, 2014
[...] the web and built up a database of Pokemon. We've indexed our database of Pokemon sprites using Zernike moments. We've analyzed query images and found our Game Boy screen using edge detection and contour [...]

Leave a Reply

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

1

3

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

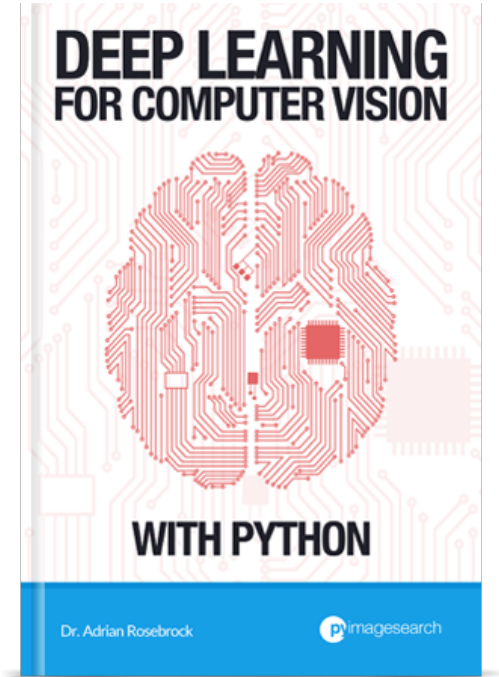
Resource Guide (it’s totally free).



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



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!](#)

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

PylImageSearch Gurus: NOW ENROLLING!


The PylImageSearch 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.

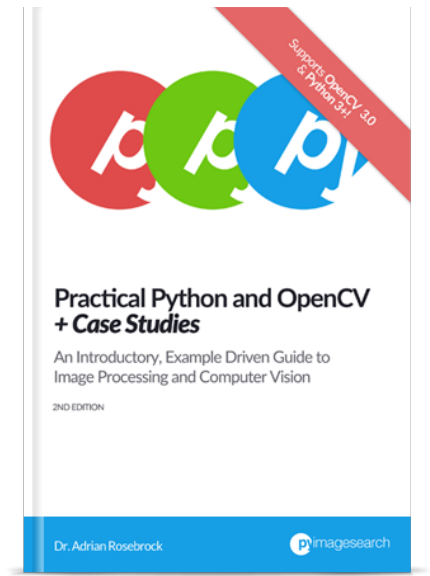
TAKE A TOUR & GET 10 (FREE) LESSONS

Hello! I'm Adrian Rosebrock.



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 PylImageSearch RSS Feed and keep up to date with my image search engine tutorials, tips, and tricks

POPULAR

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

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

Search

Search...

Q

Find me on **Twitter**, **Facebook**, **Google+**, and **LinkedIn**.
© 2017 PyImageSearch. All Rights Reserved.

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