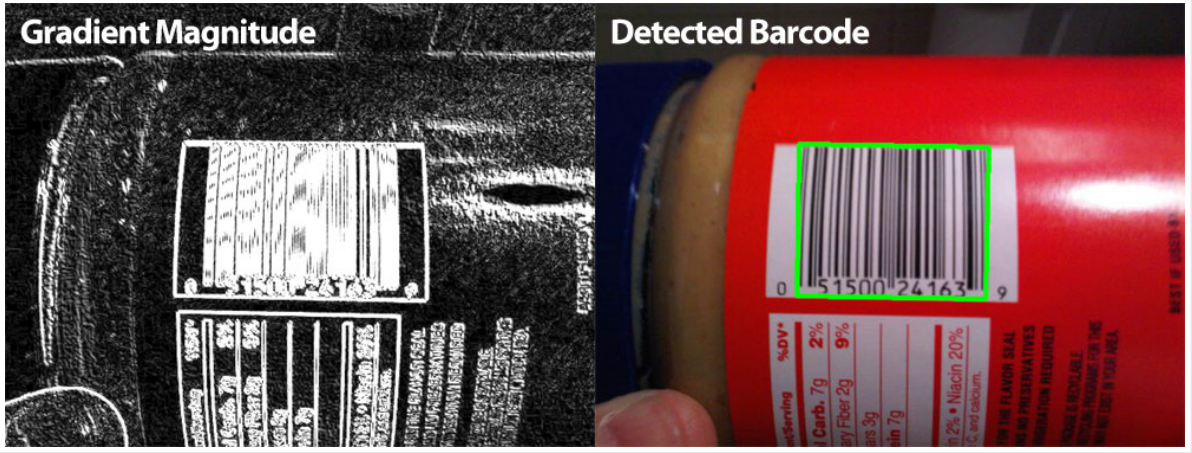
pyimagesearch
be awesome at building image search engines

Detecting Barcodes in Images with Python and OpenCV

by **Adrian Rosebrock** on November 24, 2014 in **Image Processing**, **Tutorials**



UPDATE: The introduction to this post may seen a little "out there". For some context, I had just finished watching the *South Park Black Friday episodes* prior to writing this post so I definitely had some inspiration regarding zombie shoppers, Black Friday chaos, and *Game of Thrones*.

Black Friday is coming.

Hordes of angry shoppers. Stampedes of middle-aged midwestern women, their toothless gums bloodthirsty for 75% off the latest season of *Game of Thrones* at the local Wal-Mart.

They'll lineup outside the Wal-Mat doors on Thanksgiving at midnight. They'll rally, beating at the locked doors with their hands and heads until their bodies are

raw and bloody, like zombies from *28 Days Later*. But instead of human flesh, they crave petty consumer sustenance. Their war cries of discounts and sales will reach the heavens. And their thunderous footsteps will cause earthquakes across the Great Plains.

Of course, the media won't help — they will sensationalize every last little piece. From frostbitten families camping out all night in the blistering cold, to the little old lady trampled by raging bargain hunters as the doors open, akin to the Gallimimus stampede in *Jurassic Park*. All of this simply because she wanted to purchase the latest Halo game for Timmy, her little 9 year old grandson, who's parents passed away this time last year. At a Wal-Mart. During Black Friday.

And I have to ask, is all this chaos and bedlam worth it?

Hell. No.

Any shopping I do this Black Friday will be from (safely) behind my laptop screen, likely nursing a hangover from the night before with a cup of coffee and a handful of Tylenol.

But if you decide you are going to venture out into the real-world and brave the bargain hunters, ***you'll want to download the source code to this blog post first…***

Imagine how silly you would feel, standing in line, waiting to checkout, only to scan the barcode on the latest season of *Game of Thrones* only to find out that Target has it for $5 cheaper?

In the rest of this blog post I'll show you how to detect barcodes in images using nothing but Python and OpenCV.

> **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** and **OpenCV 2.4.X**.

Detecting Barcodes in Images using Python and OpenCV

The goal of this blog post is to demonstrate a basic implementation of barcode detection using computer vision and image processing techniques. My implementation of the algorithm is originally based loosely on this StackOverflow question. I have gone through the code and provided some updates and improvements to the original algorithm.

It's important to note that this algorithm will not work for *all* barcodes, but it should give you the basic intuition as to what types of techniques you should be applying.

For this example, we will be detecting the barcode in the following image:

Figure 1: Example image containing a barcode that we want to detect.

Let's go ahead and start writing some code. Open up a new file, name it `detect_barcode.py` , and let's get coding:

Detecting Barcodes in Images with Python and OpenCV — Python

```
# import the necessary packages
import numpy as np
import argparse
```

```
4 import cv2
5
6 # construct the argument parse and parse the arguments
7 ap = argparse.ArgumentParser()
8 ap.add_argument("-i", "--image", required = True, help = "path to the image file")
9 args = vars(ap.parse_args())
```

The first thing we'll do is import the packages we'll need. We'll utilize NumPy for numeric processing, `argparse` for parsing command line arguments, and `cv2` for our OpenCV bindings.

Then we'll setup our command line arguments. We need just a single switch here, `--image` , which is the path to our image that contains a barcode that we want to detect.

Now, time for some actual image processing:

Detecting Barcodes in Images with Python and OpenCV — Python

```
11 # load the image and convert it to grayscale
12 image = cv2.imread(args["image"])
13 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
14
15 # compute the Scharr gradient magnitude representation of the images
16 # in both the x and y direction
17 gradX = cv2.Sobel(gray, ddepth = cv2.cv.CV_32F, dx = 1, dy = 0, ksize = -1)
18 gradY = cv2.Sobel(gray, ddepth = cv2.cv.CV_32F, dx = 0, dy = 1, ksize = -1)
19
20 # subtract the y-gradient from the x-gradient
21 gradient = cv2.subtract(gradX, gradY)
22 gradient = cv2.convertScaleAbs(gradient)
```

On **Lines 12 and 13** we load our `image` off disk and convert it to grayscale.

Then, we use the Scharr operator (specified using `ksize = -1`) to construct the gradient magnitude representation of the grayscale image in the horizontal and vertical directions on **Lines 17 and 18**.

From there, we subtract the *y*-gradient of the Scharr operator from the *x*-gradient of the Scharr operator on **Lines 21 and 22**. By performing this subtraction we are left with regions of the image that have high horizontal gradients and low vertical gradients.

Our `gradient` representation of our original image above looks like:

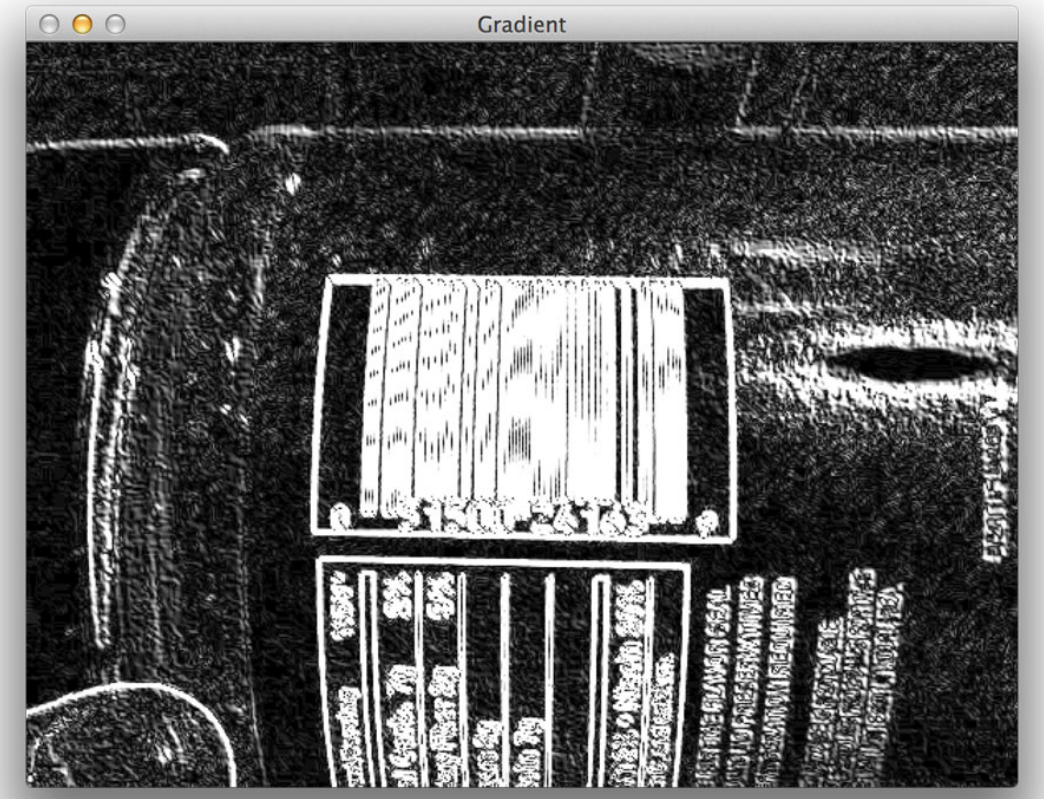


Figure 2: The gradient representation of our barcode image.

Notice how the barcoded region of the image has been detected by our gradient operations. The next steps will be to filter out the noise in the image and focus solely on the barcode region.

Detecting Barcodes in Images with Python and OpenCV — Python

```
24 # blur and threshold the image
25 blurred = cv2.blur(gradient, (9, 9))
26 (_, thresh) = cv2.threshold(blurred, 225, 255, cv2.THRESH_BINARY)
```

The first thing we'll do is apply an average blur on **Line 25** to the gradient image using a *9 x 9* kernel. This will help smooth out high frequency noise in the gradient representation of the image.

We'll then threshold the blurred image on **Line 26**. Any pixel in the gradient image that is not greater than 225 is set to 0 (black). Otherwise, the pixel is set to 255 (white).

The output of the blurring and thresholding looks like this:

Figure 3: Thresholding the gradient image to obtain a rough approximation to the rectangular barcode region.

However, as you can see in the threshold image above, *there are gaps between the vertical bars of the barcode.* In order to close these gaps and make it easier for our algorithm to detect the "blob"-like region of the barcode, we'll need to perform some basic morphological operations:

```
Detecting Barcodes in Images with Python and OpenCV                    Python
```

```
28 # construct a closing kernel and apply it to the thresholded image
29 kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (21, 7))
30 closed = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)
```

We'll start by constructing a rectangular kernel using the `cv2.getStructuringElement` on **Line 29**. This kernel has a width that is larger than the height, thus allowing us to close the gaps between vertical stripes of the barcode.

We then perform our morphological operation on **Line 30** by applying our kernel to our thresholded image, thus attempting to close the the gaps between the bars.

You can now see that the gaps are substantially more closed, as compared to the thresholded image above:

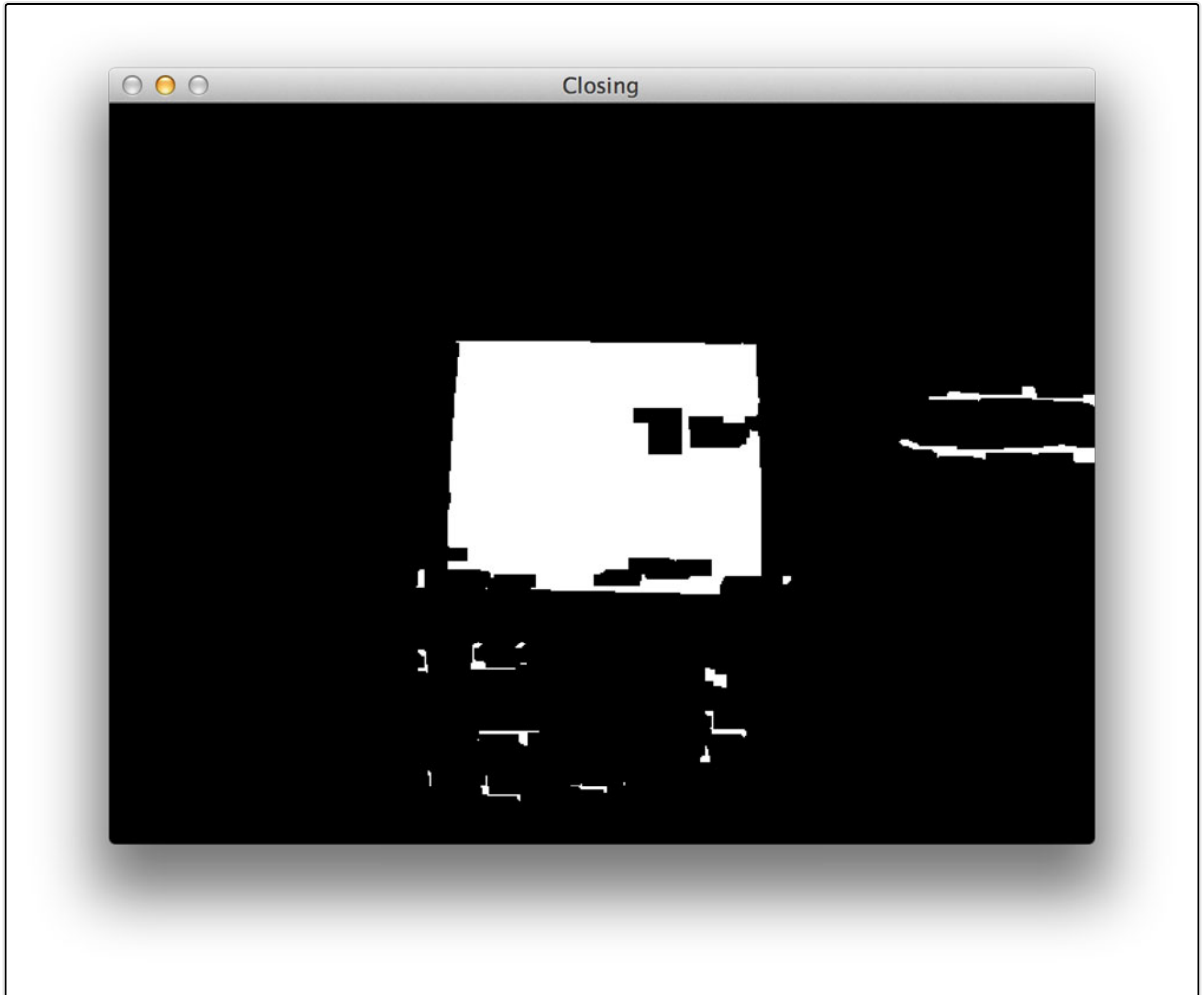


Figure 4: Applying closing morphological operations to close the gap between barcode stripes.

Of course, now we have small blobs in the image that are not part of the actual barcode, but may interfere with our contour detection.

Let's go ahead and try to remove these small blobs:

Detecting Barcodes in Images with Python and OpenCV — Python

```
32 # perform a series of erosions and dilations
33 closed = cv2.erode(closed, None, iterations = 4)
34 closed = cv2.dilate(closed, None, iterations = 4)
```

All we are doing here is performing 4 iterations of erosions, followed by 4 iterations of dilations. An erosion will "erode" the white pixels in the image, thus removing the small blobs, whereas a dilation will "dilate" the remaining white pixels and grow the white regions back out.

Provided that the small blobs were removed during the erosion, they will not reappear during the dilation.

After our series of erosions and dilations you can see that the small blobs have been successfully removed and we are left with the barcode region:

Figure 5: Removing small, irrelevant blobs by applying a series of erosions and dilations.

Finally, let's find the contours of the barcoded region of the image:

Detecting Barcodes in Images with Python and OpenCV — Python

```
36 # find the contours in the thresholded image, then sort the contours
37 # by their area, keeping only the largest one
38 (cnts, _) = cv2.findContours(closed.copy(), cv2.RETR_EXTERNAL,
39     cv2.CHAIN_APPROX_SIMPLE)
40 c = sorted(cnts, key = cv2.contourArea, reverse = True)[0]
41
42 # compute the rotated bounding box of the largest contour
43 rect = cv2.minAreaRect(c)
```

```
44 box = np.int0(cv2.cv.BoxPoints(rect))
45
46 # draw a bounding box arounded the detected barcode and display the
47 # image
48 cv2.drawContours(image, [box], -1, (0, 255, 0), 3)
49 cv2.imshow("Image", image)
50 cv2.waitKey(0)
```

Luckily, this is the easy part. On **Lines 38-40** we simply find the largest contour in the image, which if we have done our image processing steps correctly, should correspond to the barcoded region.

We then determine the minimum bounding box for the largest contour on **Lines 43 and 44** and finally display the detected barcode on **Lines 48-50**.

As you can see in the following image, we have successfully detected the barcode:

Figure 6: Successfully detecting the barcode in our example image.

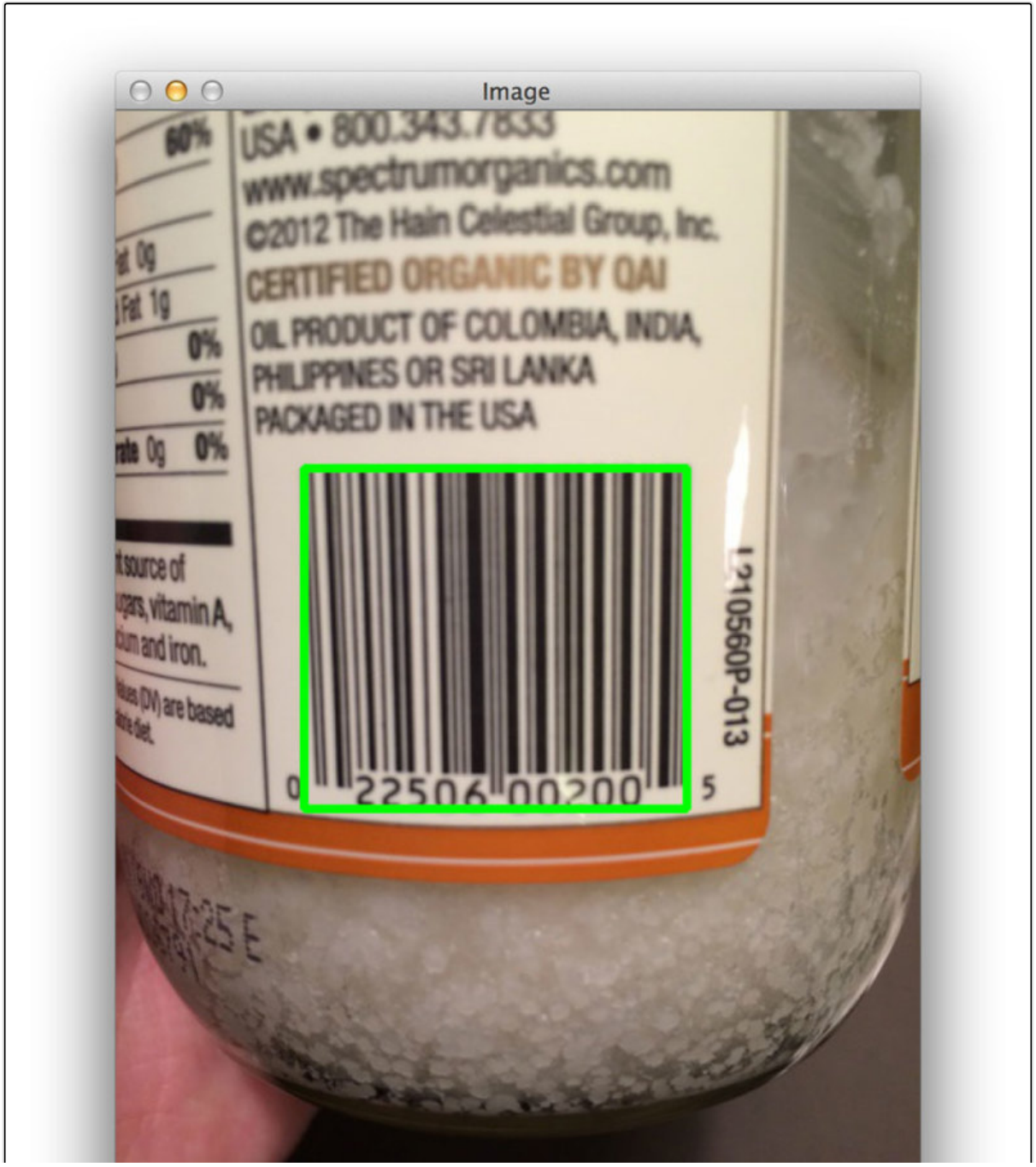
In the next section we'll try a few more images.

Successful Barcode Detections

To follow along with these results, use the form at the bottom of this post to download the source code and accompanying images for this blog post.

Once you have the code and images, open up a terminal and execute the following command:

```
$ python detect_barcode.py --image images/barcode_02.jpg
```



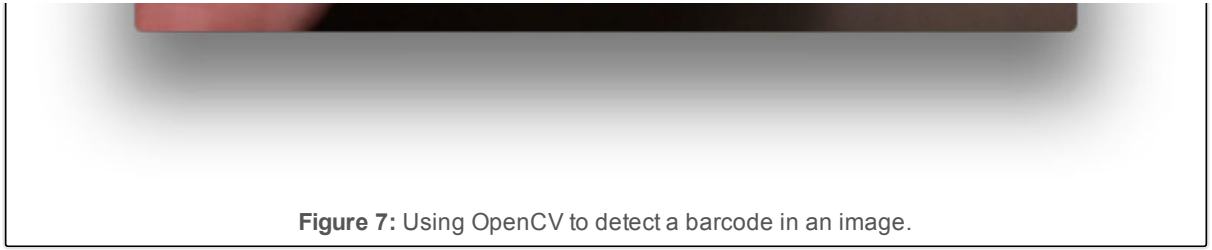
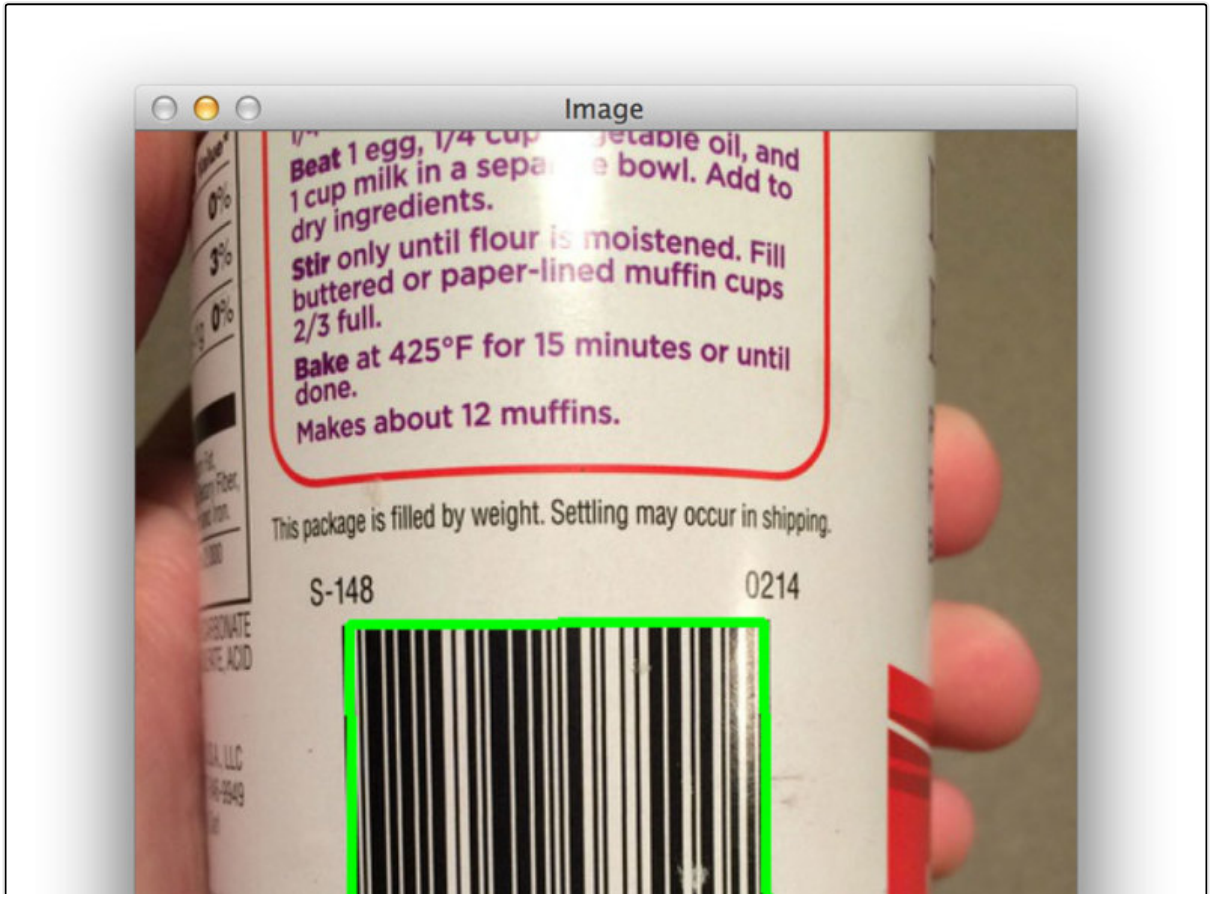


Figure 7: Using OpenCV to detect a barcode in an image.

No problem detecting the barcode on that jar of coconut oil!

Let's try another image:

```
Detecting Barcodes in Images with Python and OpenCV                    Shell
$ python detect_barcode.py --image images/barcode_03.jpg
```



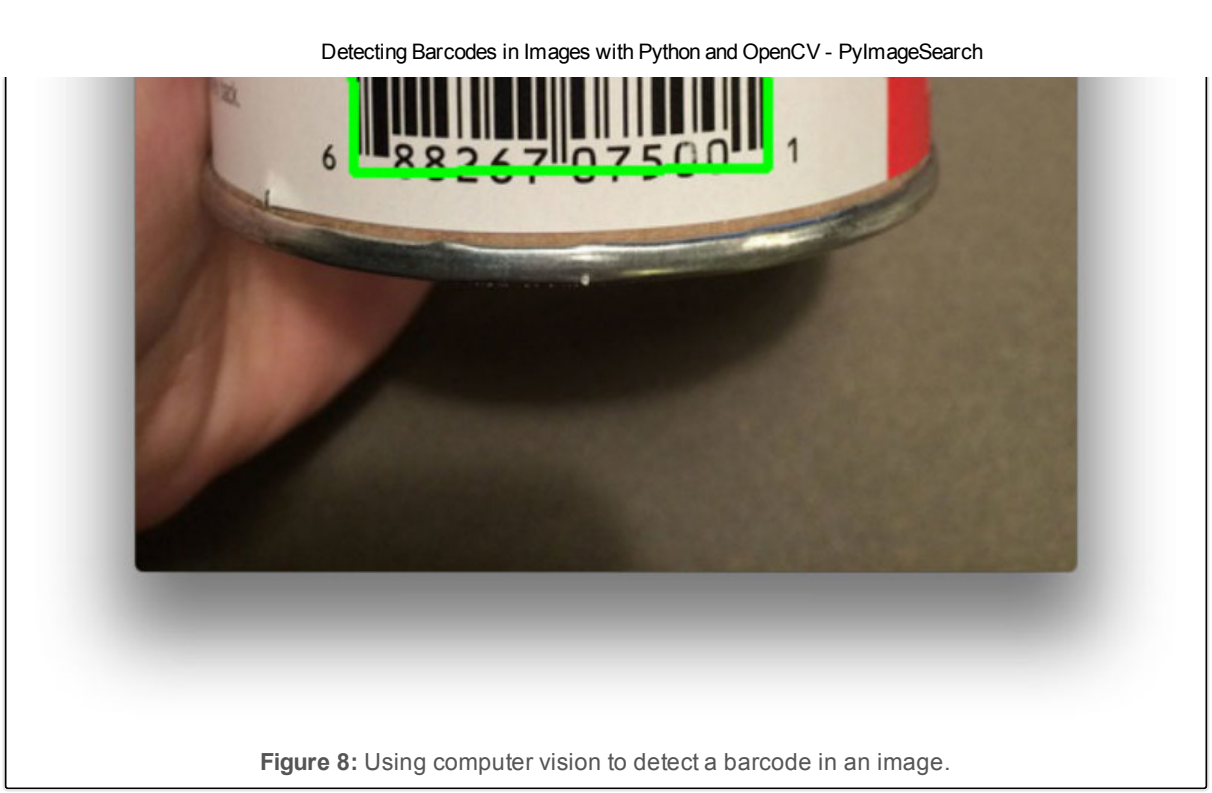
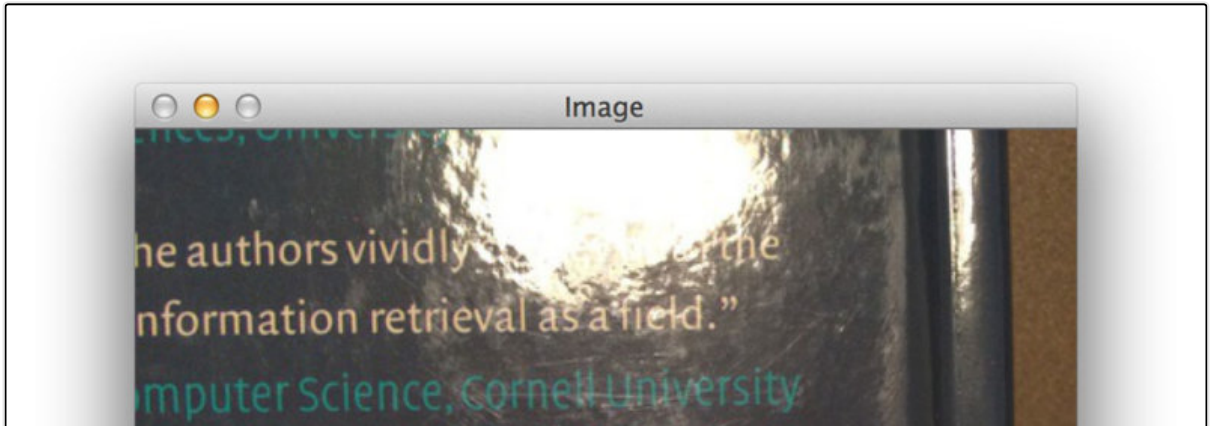


Figure 8: Using computer vision to detect a barcode in an image.

We were able to find the barcode in that image too!

But enough of the food products, what about the barcode on a book:

```
Detecting Barcodes in Images with Python and OpenCV                    Shell
1 $ python detect_barcode.py --image images/barcode_04.jpg
```



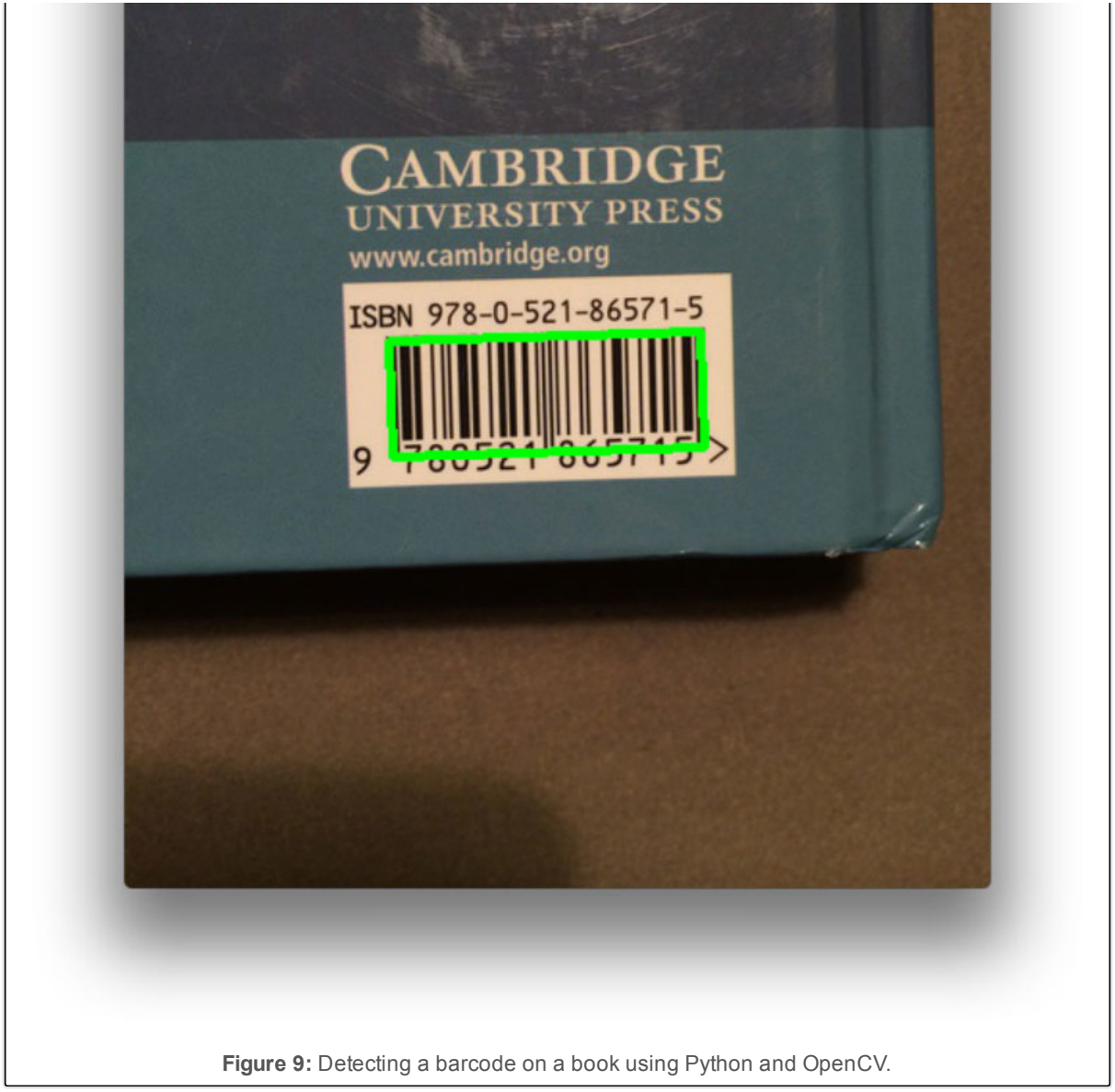


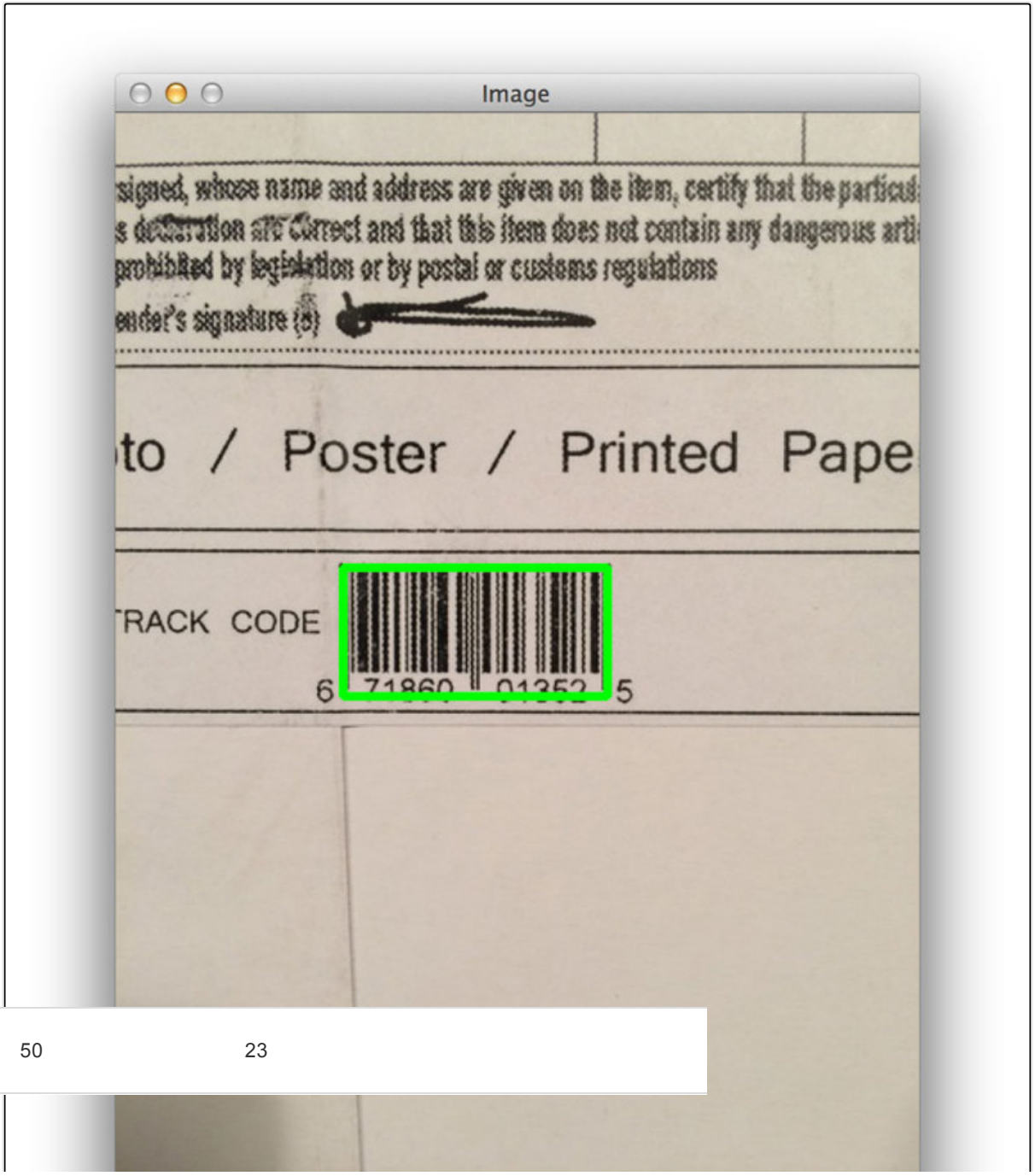
Figure 9: Detecting a barcode on a book using Python and OpenCV.

Again, no problem!

How about the tracking code on a package?

```
Detecting Barcodes in Images with Python and OpenCV                    Shell
```

```
$ python detect_barcode.py --image images/barcode_05.jpg
```



50 23

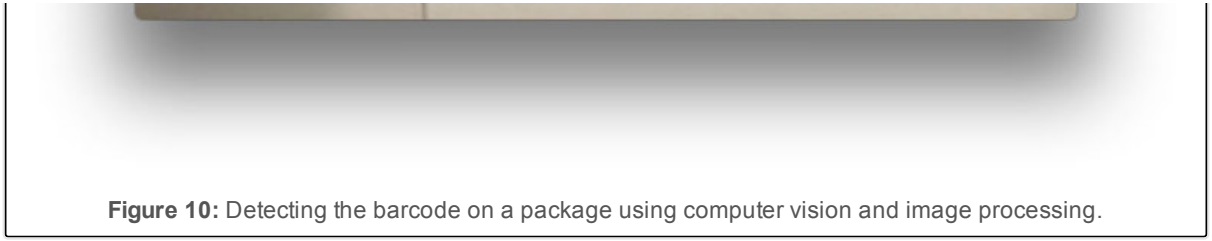
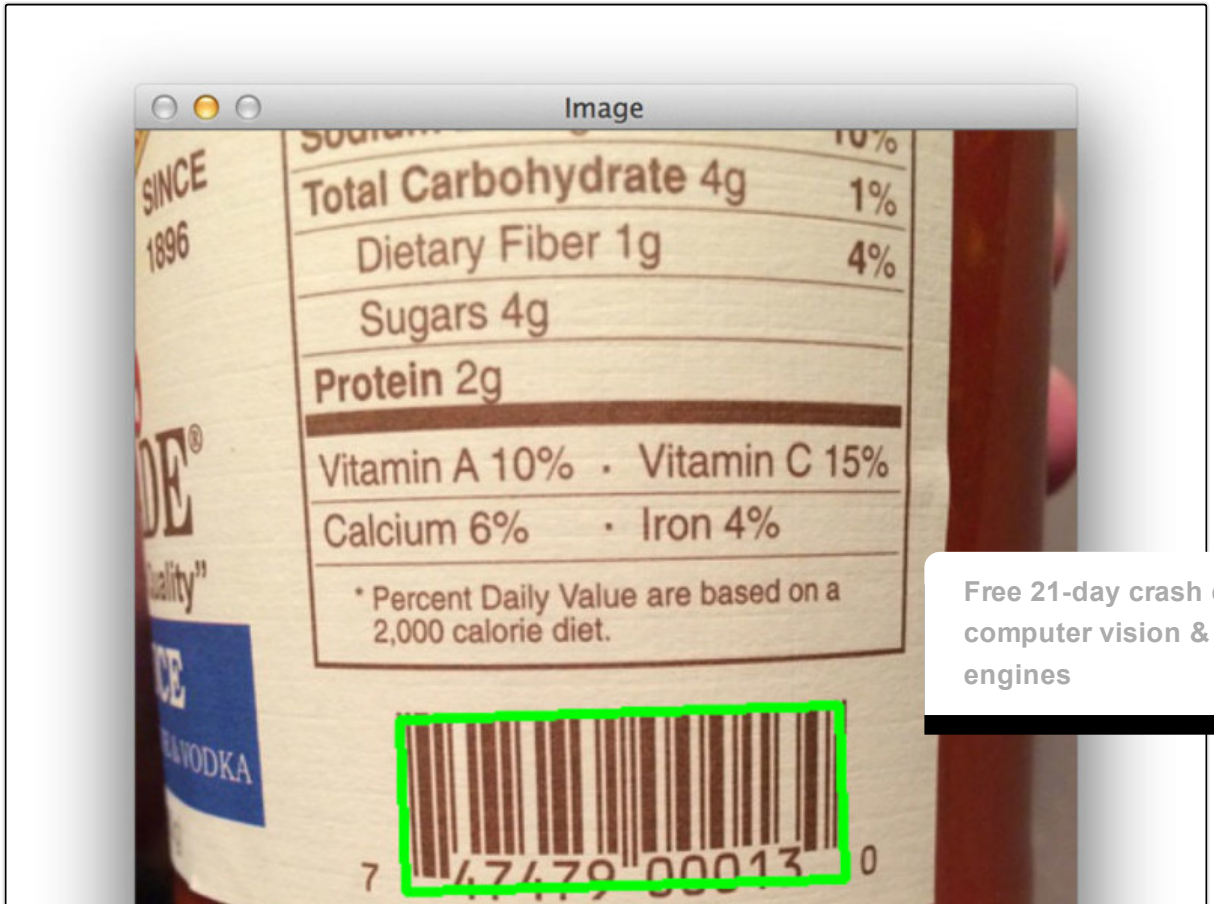


Figure 10: Detecting the barcode on a package using computer vision and image processing.

Again, our algorithm is able to successfully detect the barcode.

Finally, let's try one more image This one is of my favorite pasta sauce, Rao's Homemade Vodka Sauce:

```
Detecting Barcodes in Images with Python and OpenCV                    Shell
1 $ python detect_barcode.py --image images/barcode_06.jpg
```



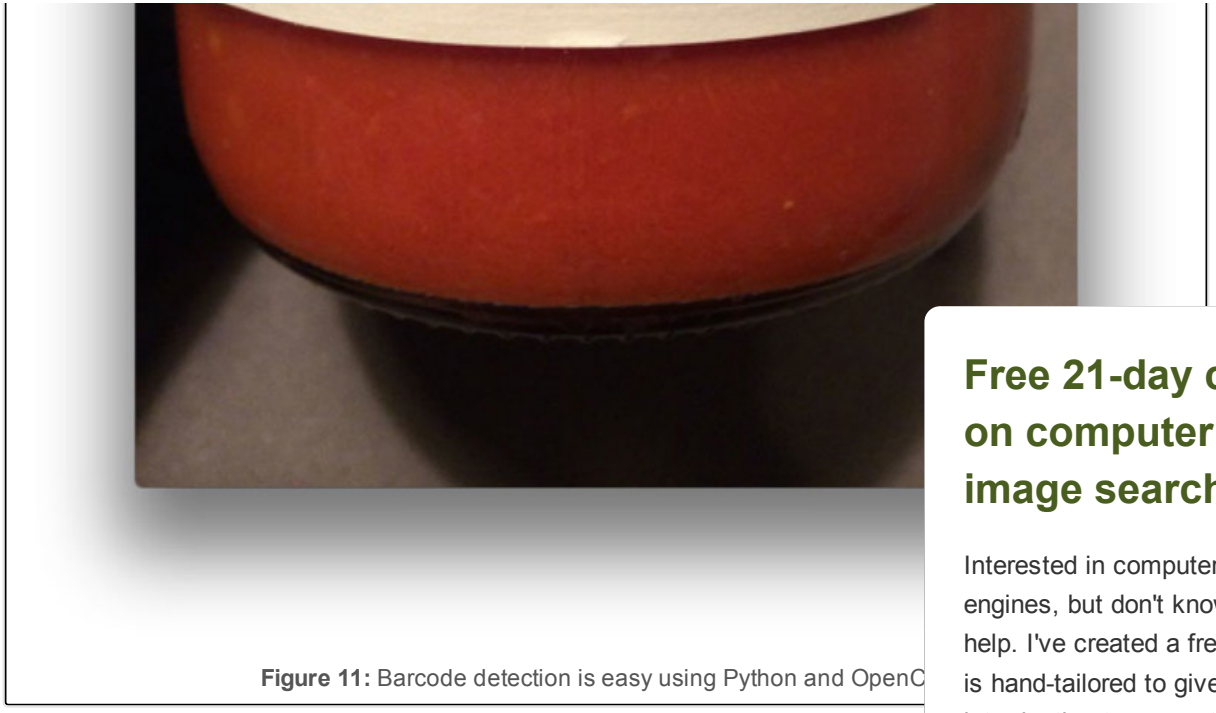


Figure 11: Barcode detection is easy using Python and OpenC

We were once again able to detect the barcode!

Summary

In this blog post we reviewed the steps necessary to detect barcodes in images using computer vision te
Python programming language and the OpenCV library.

The general outline of the algorithm is to:

1. Compute the Scharr gradient magnitude representations in both the *x* and *y* direction.
2. Subtract the *y*-gradient from the *x*-gradient to reveal the barcoded region.
3. Blur and threshold the image.
4. Apply a closing kernel to the thresholded image.
5. Perform a series of dilations and erosions.
6. Find the largest contour in the image, which is now presumably the barcode.

It is important to note that since this method makes assumptions regarding the gradient representations of the image, and thus will only work for horizontal barcodes.

If you wanted to implement a more robust barcode detection algorithm, you would need to take the orientation of the image into consideration, or better yet, apply machine learning techniques such as Haar cascades or HOG + Linear SVM to "scan" the image for barcoded regions.

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!

barcode, **barcode detection**, **gradients**, **image processing**, **magnitude**, **thresholding**, **tutorials**

‹ Non-Maximum Suppression for Object Detection in Python

The complete guide to building an image search engine with Python and OpenCV ›

91 Responses to *Detecting Barcodes in Images with Python and OpenCV*

Brandon November 24, 2014 at 12:32 pm #

REPLY

Hi, nice article. I'd be interested to see a follow-up on actually deciphering the barcode value from the image. One tiny improvement I saw:

c = sorted(cnts, key = cv2.contourArea, reverse = True)[0]

could simply be a min operation to avoid creating a second sorted array, only to take the first element:

c = min(cnts, key = cv2.contourArea)

Adrian Rosebrock November 24, 2014 at 12:59 pm #

REPLY

Hi Brandon, thanks for the reply! I'll definitely consider doing a followup post on deciphering barcode values.

As for the code, I'm actually trying to find the *largest* contour, not the smallest one, which I think is what the code you presented is doing. Did you mean:

```
c = max(cnts, key = cv2.contourArea)
```

If so, I'll definitely give that a shot next time!

NHAS November 24, 2014 at 12:48 pm #

REPLY

Hello, good job on making the article I think its quite useful. Only thing is your zombie reference is a bit out. Its 28 days later 😛 not 24 days later.

Other than that its a brilliant article.

Adrian Rosebrock November 24, 2014 at 12:57 pm #

REPLY

Whoops! My mistake — can't believe I messed that one up. It's fixed now!

Bob November 24, 2014 at 5:25 pm #

REPLY

Nice article. But please don't combine alcohol and tylenol (paracetamol). Even when you're hungover it's really very bad for your liver. Sorry for being off-topic.

Adrian Rosebrock November 24, 2014 at 7:18 pm #

REPLY

Hi Bob, thanks for the comment! You're very right, never mix alcohol and Tylenol. 🙂

Jason November 26, 2014 at 12:25 am #

REPLY

Would be cool to see this done with streaming video.

Adrian Rosebrock November 26, 2014 at 7:13 am #

REPLY

Hi Jason, damn, that's a great idea. I see a followup post coming!

dolaameng November 27, 2014 at 7:57 am #

REPLY

Hi, thanks for the article. Two questions here:

(1) Is using a scharr operator than a sobel crucial to this problem?

(2) In your explanation “By performing this subtraction we are left with regions of the image that have high horizontal gradients and low vertical gradients.”, do you mean LOW horizontal gradients and HIGH vertical gradients? Because that’s how they named vscharr and hscharr in skimage.

REPLY

Adrian Rosebrock November 27, 2014 at 10:04 am #

(1) I don’t think it’s crucial. But the Scharr operator is more accurate for a 3×3 kernel, that’s the reason I used it. Realistically you could use the Sobel operator as well, I don’t think it would make an impact.

(2) Interesting. I’ll have to take a look at that.

REPLY

Niv November 28, 2014 at 1:45 am #

Hi, nice article.

When you subtract the y-gradient from the x-gradient, you end up with:

1. high positive values (i.e. +255) for: {high horizontal gradients and low vertical gradients} and
2. Low negative values (i.e. -255) for: {high vertical gradients and low horizontal gradients},

and since you are further making “convertScaleAbs” then all the high negatives will become high positives, so you end up with an image in which high vertical and low horizontal gradients are also emphasized. Which isn’t what you say you want (you want only the high horizontal and low vertical gradients).

What do I miss here?

Another remark related to the above is that in figure 2 (for example the finger) we can see that he image contains all combinations of high gradients- high horizontals, high verticals, and also gradients in 45 [deg].

Thanks,

REPLY

Adrian Rosebrock November 28, 2014 at 8:17 am #

Hi Niv, shoot me an email and we can chat more about your question. Thanks!

Matt McDonnell November 30, 2014 at 7:48 am #

REPLY

Thanks for the article! I've been playing with OpenCV in Google Chrome Native Client, and put together a version of this post that works in (Chrome) browser.
Demo: http://www.matt-mcdonnell.com/code/NaCl/ImageProc/
Code (BSD): https://github.com/mattmcd/NaCl/tree/master/ImageProc

mastsolis December 13, 2014 at 11:22 pm #

REPLY

Hi,

Wow, thank you for posting this tutorial. I was wondering, if it's possible to somehow save the coordinate location and barcode image size?

Adrian Rosebrock December 14, 2014 at 7:21 am #

REPLY

Sure, absolutely. The line `box = np.int0(cv2.cv.BoxPoints(rect))` will give you the (x, y)-coordinates of the location of the barcode and the barcode size. From there, you can store the coordinates on disk, in a database, etc.

Tony January 30, 2015 at 4:36 am #

REPLY

Hi,

Your articles are good and informative. However, I would like to mention what I perceive as a shortcoming, and this applies to many of your articles. And that is, instead of diving right into code which you seem to have a tendency of doing, I think it is much better to describe in words the algorithm and what you are trying to do. In this case, it would be the characteristics of the barcode region you're trying to detect. You have something which is almost similar in the summary, but it should be even more general and at the beginning. As an example, the article link you provided about how QuaggaJS works is much clearer in this regard, I find. Right in the beginning introduction, the author says we are looking for a barcode region with "lines which are close to each other and have a similar angle", and then proceeds to describe steps and details. It's much clearer that way. Diving into the code without knowing what

you're looking for is confusing. Again, this is meant as constructive criticism. Thanks.

Adrian Rosebrock January 30, 2015 at 6:28 am #

REPLY

Thanks for the feedback Tony.

Arron March 26, 2015 at 8:55 am #

REPLY

Your job is very good and significant,but it can only detect barcode,if it can recognize the barcode in the picture,it will be better.

Richie April 25, 2015 at 12:23 am #

REPLY

Great article Adrian. I'd knew cv was the way to do this but didn't know where to start. I have got this working in c#, you can see the gist here: https://gist.github.com/richardsawyer/62596e14d1ec4d148b28

Changes I made: optimized for vertical barcode; added centring offset to the restructuring element as the final box was shifted up and to the left; experimented with the thresholding to get reliable results in my images; extracted the actual barcode as a bitmap to make reading easy.

Thanks for sharing.

Adrian Rosebrock May 1, 2015 at 7:10 pm #

REPLY

Very nice Richie, thanks for sharing! 🙂

John May 30, 2015 at 3:21 pm #

REPLY

This is cool. Have you come up with how to extract the barcode value yet?

Adrian Rosebrock May 30, 2015 at 6:15 pm #

REPLY

Hi John, using the code above you can most certainly *extract* the barcode from the image. Do you mean actually *recognize* the barcode? If so, I probably won't be covering that directly on the PyImageSearch blog. There are a lot of other resources online that cover the encoding schemes of barcodes far better than I could.

dhruv June 12, 2015 at 6:47 am #

REPLY

Hi thx for program…….I'm New in this……I'm having an error that..usage: detect_barcode.py
[-h] -I IMAGE
detect _barcode.py: error: argument -i/–image is required…so plz say how to solve it

Adrian Rosebrock June 12, 2015 at 9:41 am #

REPLY

Please look at the **Successful Barcode Detections** section where I detail how to run the Python script. Here is an example:

```
python detect_barcode.py --image images/barcode_02.jpg
```

Notice how the `--image` switch points to the `barcode_02.jpg` file in the `images` directory.

zee December 8, 2017 at 6:06 am #

REPLY

usage: detect_barcode.py [-h] -i IMAGE
detect_barcode.py: error: argument -i/–image is required
can you plz briefly expalin how to solve this issue in windows

Adrian Rosebrock December 8, 2017 at 4:39 pm #

REPLY

Please read up on command line arguments and you will be able to resolve the issue.

zee December 9, 2017 at 6:51 am #

sir iam struck with this error i have already checked the size of image its fine

detect_barcode.py –image images/barcode_01.jpg
OpenCV Error: Assertion failed (scn == 3 || scn == 4) in cv::cvtColor, file ..\
.\..\..\opencv\modules\imgproc\src\color.cpp, line 3737
Traceback (most recent call last):
File “C:\Users\zee\detect_barcode.py”, line 17, in
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.error: ..\..\..\..\opencv\modules\imgproc\src\color.cpp:3737: error: (-215)
scn == 3 || scn == 4 in function cv::cvtColor

Adrian Rosebrock December 9, 2017 at 7:22 am #

Your path to your input image is incorrect. The `cv2.imread` function will return None if the image path is incorrect. Since you are using Windows you need to update your `--image` path since Windows uses the “\” path separator rather than “/”:

```
$ detector_barcode.py --image images\barcode_01.jpg
```

Notice how the path separator has been changed.

zee December 9, 2017 at 8:25 am #

sir even after trying the \ same error persists assertion failed

zee December 9, 2017 at 8:26 am #

i tried with both ways / and \ both are giving the same error

SayeedM June 24, 2015 at 1:09 am #

REPLY

Hello, great article. Btw is the procedure to detect 2D barcodes (PDF417) similiar ? I mean for 2D barcodes horizontal gradients wont be comparatively high to vertical gradients, right ? Can you do a followup on this ?

al July 10, 2015 at 5:18 am #

REPLY

I've followed the instructions on installing OpenCV on Ubuntu and I can run the example on that page. I get the following error while trying this example however:

gradX = cv2.Sobel(gray, ddepth=cv2.cv.CV_32F, dx=1, dy=0, ksize=-1)
AttributeError: 'module' object has no attribute 'cv'

Any idea why that might be the case?

Adrian Rosebrock July 10, 2015 at 6:22 am #

REPLY

Hey Al, it sounds like you are using OpenCV 3.0. This code was developed using OpenCV 2.4.

Change `ddepth=cv2.cv.CV_32F` to `ddepth=cv2.CV_32F` (along with the following line for the y gradient) and you'll get ride of the pesky error.

You'll also need to change the `cv2.findContours` call to:

```
(_, cnts, _) = cv2.findContours(closed.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

al July 11, 2015 at 1:52 am #

REPLY

They seem to have updated so now instead of cv2.cv.CV_32F it's just cv2.CV_32F.

also cv2.cv.BoxPoints is now cv2.boxPoints

Adrian Rosebrock July 11, 2015 at 6:30 am #

REPLY

Thanks Al!

Nicolas August 19, 2015 at 8:29 am #

REPLY

Thanks for the great post! I tried it and it worked from the first time, but i was wondering is there a way to get the value of the barcode?

Adrian Rosebrock August 20, 2015 at 7:02 am #

REPLY

Great question. There are a lot of barcode reading packages out there. I would start by looking into zbar.

Chris October 27, 2015 at 9:52 pm #

REPLY

Excellent post and enjoying the e-books too! Just a quick (and I'm sure, easy) question. How difficult would it be to extend this to include multiple barcodes in a single image? Like if you have to catalog a monotonous amount of barcodes but don't really want to scan each one at a time but would rather line up about 5-10 barcodes and extract those contours?

Thanks!

Adrian Rosebrock November 3, 2015 at 10:42 am #

REPLY

It wouldn't be too challenging to extend to multiple barcodes. Instead of taking the largest contour as I do on **Line 40**, just loop over them individually, ensure they have sufficiently large size, and continue to process them.

Inês Martins December 2, 2015 at 11:44 am #

REPLY

How to crop the barcode region out of image?

using box points??

Adrian Rosebrock December 2, 2015 at 3:04 pm #

REPLY

Hey Ines — I would use:

`(x, y, w, h) = cv2.boundingRect(c)` on the contour to grab the bounding box of the barcode. From there, you can extract the region from the image.

Shoo October 24, 2017 at 10:17 pm #

REPLY

Hi can you explain further on this? "How to crop the barcode region out of image?"

Adrian Rosebrock October 25, 2017 at 12:33 pm #

REPLY

Hi Shoo — you can extract a region of interest (ROI) using NumPy array slicing. This is covered in *Practical Python and OpenCV + Case Studies* in Chapter 4.

Vinoth May 9, 2016 at 1:02 pm #

REPLY

Is there a way to compare two different bar codes to check if they are similar?

Adrian Rosebrock May 9, 2016 at 6:44 pm #

REPLY

Sure, there are multiple ways to do this. I would suggest starting with MSE and SSIM.

Scott May 10, 2016 at 12:17 am #

REPLY

do you have something similar for detecting QRCodes?

Adrian Rosebrock May 10, 2016 at 8:07 am #

REPLY

I don't have any tutorials related to detecting QRCodes, but it's certainly something I'll consider in the future!

prasiddha May 22, 2016 at 8:57 am #

REPLY

I m a beginner in image processing and opencv both, i applied almost same algorithm and segmented the barcode from image now i'm lost trying to decode it. can you help me converting those bars into numbers? may be binary?(using opencv)

Adrian Rosebrock May 23, 2016 at 7:25 pm #

REPLY

I personally haven't tried decoding the barcodes themselves, but I would recommend using a library such as zbar.

vaishali masal June 21, 2016 at 3:15 am #

REPLY

I also trying to decode Barcode in opencv and python.But didn't get any idea how to do it. Can you tell me if you get it.

Kevin September 9, 2016 at 2:01 am #

REPLY

Coming in late but I would recommend to use ZXing. Been using for some time now and my colleagues and I are very happy with it.

@Adrian, great post! Really useful.

Adrian Rosebrock September 9, 2016 at 10:51 am #

REPLY

Thanks Kevin, I'm glad it helped! And thanks for the recommending on ZXing.

Em June 22, 2016 at 12:49 pm #

REPLY

Hello!!!! You're the very best Adrian.

I'm getting this error on box points:

```
Python
1 Traceback (most recent call last):
2   File "detect_barcode.py", line 44, in
3     box = np.int0(cv2.BoxPoints(rect))
4 AttributeError: 'module' object has no attribute 'BoxPoints'
```

Do you know what I can do to fix it? Thank you

Adrian Rosebrock June 23, 2016 at 1:17 pm #

REPLY

It sounds like you're using OpenCV 3 (this blog post assumes you're using OpenCV 2.4). You can resolve the issue by using:

```
box = cv2.boxPoints(rect)
```

Mithun S August 9, 2016 at 3:15 am #

REPLY

Is it possible to the same steps in C# .net?

Adrian Rosebrock August 10, 2016 at 9:32 am #

REPLY

Yes, but you would need to convert the code from Python + OpenCV to C# + OpenCV.

Rafi Fadlianto December 5, 2016 at 6:01 pm #

REPLY

Thanks for the Tutorial 😀 I will try it 🙂
So we cant use for OpenCV 3 ?

*note = I dont have any website right now

Adrian Rosebrock December 7, 2016 at 9:51 am #

REPLY

This blog post was written before OpenCV 3 was released. That said, change `cv2.cv.CV_32F` to `cv2.CV_32F` and `(cnts, _) = cv2.findContours(closed.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)` to `(_, cnts, _) = cv2.findContours(closed.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)` and the script should work just fine.

esconda January 18, 2017 at 3:43 am #

REPLY

Is there anyway convert it to c++ code.I just would like to test performance of opencv in c++.I already applied all filtration until this code;

box = np.int0(cv2.cv.BoxPoints(rect))

How can I convert it to c/c++ “np.int0” what is equavalent of this code in c?

Varun February 16, 2017 at 5:28 am #

REPLY

Great work!
Can you provide a source where I could find the tutorial to decipher the bar code? That would be great!

Adrian Rosebrock February 16, 2017 at 9:47 am #

REPLY

I don’t have any tutorials covering barcode decoding (yet), but I would suggest using zbar.

Osama Arif February 19, 2017 at 1:41 pm #

REPLY

the program shows this error:

usage: detect_barcode.py [-h] -i IMAGE
detect_barcode.py: error: argument -i/–image is required

Please respond asap

Adrian Rosebrock February 20, 2017 at 7:42 am #

REPLY

You need to supply command line arguments to your script. Before continuing you need to read up on command line arguments.

Ráfagan Abreu April 3, 2017 at 3:25 am #

REPLY

I built a version of this algorithm using Python 3, OpenCV 3.2 and MatPlot. Follow the code:\

https://gist.github.com/rafagan/5e73867958712eaeeb6d781e868559e6

Thorsten May 26, 2017 at 7:57 am #

REPLY

Hi Adrian,

here again. I changed your code accordingly to your instructions to fit to opencv-3.0.0. The detection itself works and returns following box

[[340. 140.]
[311. 140.]
[311. 121.]
[340. 121.]].

But if this box is taken as input to
cv2.drawContours(frame, [box], -1, (0, 255, 0), 2), it fails with a exception.

cv2.error: /home/pi/opencv-3.0.0/modules/imgproc/src/drawing.cpp:2310:
error: (-215) npoints > 0 in function drawContours.

As I understand the box must be a array of arrays. Even if I exchange '[box]' with 'box' in function call, it fails with same exception.

What's wrong?

Thanks in advance

Adrian Rosebrock May 28, 2017 at 1:12 am #

REPLY

So if I understand correctly, you computed the bounding box and now you are trying to draw the rotated bounding box on the image?

Make sure you do:

```
box = cv2.boxPoints(c)
box = np.array(box, dtype="int")
```

Amrit Kumbhakar August 30, 2017 at 12:20 am #

REPLY

Amazing tutorial. Thanks .
I successfully completed the steps. Now i want to get the numbers from the barcode. What to get that?

Adrian Rosebrock August 31, 2017 at 8:35 am #

REPLY

Take a look at the zbar library. I personally haven't used it but I know other PyImageSearch readers have liked it.

Amrit Kumbhakar September 7, 2017 at 1:38 am #

REPLY

Thanks. Can you suggest some?

uti September 3, 2017 at 5:32 am #

REPLY

Nice tutorial. I compiled it with python 3.6 and opencv 3.3.0 using the mentioned changes to get it up and running. Sadly it's only working for the first barcode in the zip.. The others are not found. Either only parts are detected or some text is marked 😕

Whats interesting is that the results (the rectangle marking the barcode) differ from opencv 3.2.0 to 3.3.0 with opencv 3.2.0 also not finding the barcode. I tested python 2.7 and 3.6 with both opencv 3.2.0 and 3.3.0. Maybe you could revise this tutorial given you have time to spare.

Adrian Rosebrock September 5, 2017 at 9:28 am #

REPLY

Thanks for sharing, Uti!

Atharva Saraf September 20, 2017 at 2:11 am #

REPLY

Hi! I am a python beginner.I tried implementing your code. Ran into a error at c=sorted(…). The index list is growing out of bound. How can this be corrected? Thank you!

Dmytro September 25, 2017 at 12:43 pm #

REPLY

HI Adrian,
Thanks for this article, it is really helps me to solve my solution.
But I have problem while detecting barcode with black background. In this case it works a little bi incorrect. So when I change thresh parameter from 225 to 175, 150 or even 100 , it recognize barcode with black background, but with this parameters it works incorrectly for others images.
Could you please recommend some advise how to solve this issue?

Thanks,
Dmytro

Adrian Rosebrock September 26, 2017 at 8:19 am #

REPLY

Have you considered applying adaptive thresholding or Otsu's automatic thresholding? This would be my first suggestion.

Vinay September 28, 2017 at 2:31 pm #

REPLY

hi i am using python3 and opencv 3 . As you posted in comments i made necessary changes in code but i am getting error as

Traceback (most recent call last):
File "barcoded.py", line 47, in
box = np.int0(cv2.cv.BoxPoints(rect))
AttributeError: module 'cv2' has no attribute 'cv'

how to resolve this error ?

Adrian Rosebrock September 28, 2017 at 6:03 pm #

REPLY

Always check the comments for your error. Please see my reply to “Em” on June 22nd.

Saldor October 26, 2017 at 9:58 pm #

REPLY

HI! Where in this code, that part of it, where it is determined that we should write down the area where the barcode is found (strips and digits)?

Adrian Rosebrock October 27, 2017 at 11:11 am #

REPLY

Hi Saldor– Line 44 gets the coordinates for the box. Let me know if this helps.

hoda December 6, 2017 at 2:39 pm #

REPLY

this code is’nt working. there is not any kind of error but it does’nt open me any images let alone detecting barcodes. i do not know how to fix it can you help me please

Adrian Rosebrock December 8, 2017 at 5:04 pm #

REPLY

Hey Hoda — are you testing on your own images or the images included with the download of the blog post? Make sure you use the “Downloads” section of the post and test with the original images to obtain a baseline first. It sounds like no barcodes are being detected in your input images and thus the script automatically exits.

rushabh December 8, 2017 at 9:31 am #

REPLY

Hey adrian i changed code as you said in comment to EM on 22 june but it still giving samr error

File “detect_barcode.py”, line 47, in
box = cv2.BoxPoints(rect)
AttributeError: ‘module’ object has no attribute ‘BoxPoints’

Adrian Rosebrock December 8, 2017 at 4:37 pm #

REPLY

The code is still incorrect — your “B” is capitalized when it should be a lowercase “b”:

```
cv2.boxPoints(rect)
```

rushabh December 9, 2017 at 1:09 am #

REPLY

oh sorry again but now it giving this problem

it does not showing the right barcode it just showing half part of 4,5 barcode image file.

rushabh December 9, 2017 at 1:10 am #

REPLY

and also i need to make an application to make the for droesiness detection but i cannot having dlib oin windoes so can you help me please.

Adrian Rosebrock December 9, 2017 at 7:27 am #

REPLY

I would suggest doing research on facial landmark detection libraries that can be used on Windows. For what it’s worth, dlib can be installed on Windows.

Surabhi January 13, 2018 at 10:11 pm #

REPLY

This article is really nice. However, I had a question.
How would you justify that the barcode region will have the highest x gradient value? What about the background words in the sentences, even they'll have a high gradient in the x-direction right?

Adrian Rosebrock January 15, 2018 at 9:17 am #

REPLY

We subtract the y-gradient output from the x-gradient output. This subtraction leaves us with regions of the image that have high horizontal gradients and low vertical gradients. A barcode will exhibit lots of changes in the horizontal direction.

Alex January 21, 2018 at 5:51 pm #

REPLY

Hello! Nice article and very informative. However I have a question.

Why did you put [0] on this? c = sorted(cnts, key = cv2.contourArea, reverse = True)[0]

Thanks a lot.

Adrian Rosebrock January 22, 2018 at 6:19 pm #

REPLY

The "[0]" returns the element at the front of the list which will be the contour with the largest area. Alternatively we could write:

```
c = max(cnts, key=cv2.contourArea
```

Trackbacks/Pingbacks

Real-time barcode detection in video with Python and OpenCV - PyImageSearch - December 15, 2014
[…] Today's post is a followup to a previous (extremely popular) article on detecting barcodes in images using Python and OpenCV. […]

Sorting Contours using Python and OpenCV - PyImageSearch - April 20, 2015
[…] Contours enabled us detect barcodes in images. […]

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Search...

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 — OUT NOW!

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 MASTER DEEP LEARNING

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!*

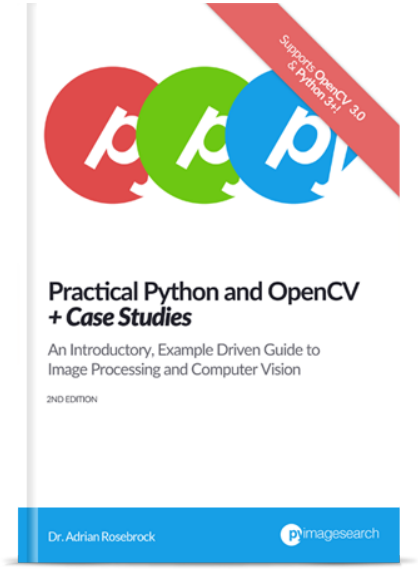
Click the button below to learn more about the course, take a tour, and get 10 (FREE) sample lessons.

TAKE A TOUR & GET 10 (FREE) LESSONS

Hello! I'm Adrian Rosebrock.

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

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3
APRIL 18, 2016

Install OpenCV and Python on your Raspberry Pi 2 and B+
FEBRUARY 23, 2015

Home surveillance and motion detection with the Raspberry Pi, Python, OpenCV, and Dropbox
JUNE 1, 2015

How to install OpenCV 3 on Raspbian Jessie
OCTOBER 26, 2015

Ubuntu 16.04: How to install OpenCV
OCTOBER 24, 2016

Basic motion detection and tracking with Python and OpenCV
MAY 25, 2015

Accessing the Raspberry Pi Camera with OpenCV and Python
MARCH 30, 2015

Find me on **Twitter**, **Facebook**, **Google+**, and **LinkedIn**.