Navigation

pyimagesearch
be awesome at building image search engines

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

by **Adrian Rosebrock** on September 1, 2014 in **Image Processing**, **Tutorials**

70 40

Remember my friend James from my post on skin detection?

Well, we grabbed a cup of coffee and some groceries yesterday at the local Whole Foods, all while discussing the latest and greatest CVPR papers.

After I checked-out, I whipped out my iPhone and took a picture of the

receipt — it's just something that I like to do so I can easily keep track of where my money is going each month (like spending $20 on bacon, for instance).

Anyway, James asked why I wasn't using a document scanning app like Genius Scan or TurboScan to scan the receipt into my phone.

Good question.

I honestly hadn't thought of it.

But then James went too far…

He bet me $100 that I couldn't build a mobile document scanner app of my own.

Well, the joke is on you, James.

I'll be collecting my $100 the next time I see you.

Because honestly, document scanning apps are ***ridiculously easy*** to build.

Genius Scan. TurboScan. Scanner Pro. You name it — the computer vision algorithms running behind the scenes are dead simple to implement.

I bet that after reading this blog post that you could create an app to compete with the big-boy mobile document scanners as well.

You see, scanning a document using your smartphone can be broken down into three simple steps:

- **Step 1:** Detect edges.
- **Step 2:** Use the edges in the image to find the contour (outline) representing the piece of paper being scanned.
- **Step 3:** Apply a perspective transform to obtain the top-down view of the document.

Really. That's it. Only three steps and you're on your way to submitting your own document scanning app to the App Store.

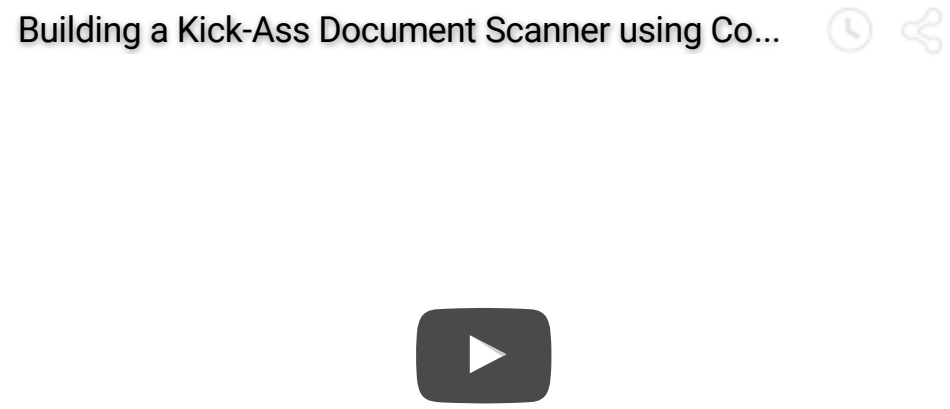
Sound interesting?

Read on. And unlock the secrets to build a mobile scanner app of your own.

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**.

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



Last week I gave you a special treat — my very own `transform.py` module that I use in all my computer vision and image processing projects. You can read more about this module here.

Whenever you need to perform a 4 point perspective transform, you should be using this module.

And you guessed it, we'll be using it to build our very own document scanner.

So let's get down to business.

Open up your favorite Python IDE, (I like Sublime Text 2), create a new file, name it `scan.py` , and let's get started.

Building a Document Scanner App using Python, OpenCV, and C Python

```python
# import the necessary packages
from pyimagesearch.transform import four_point_transform
from pyimagesearch import imutils
from skimage.filter import threshold_adaptive
```

```
5  import numpy as np
6  import argparse
7  import cv2
8
9  # construct the argument parser and parse the arguments
10 ap = argparse.ArgumentParser()
11 ap.add_argument("-i", "--image", required = True,
12     help = "Path to the image to be scanned")
13 args = vars(ap.parse_args())
```

Lines 2-7 handle importing the necessary Python packages that we'll need.

We'll start by importing our `four_point_transform` function which I discussed last week.

We'll also be using the `imutils` module, which contains convenience functions for resizing, rotating, and cropping images. You can read more about `imutils` in my basic image manipulations post.

Next up, let's import the `threshold_adaptive` function from scikit-image. This function will help us obtain the "black and white" feel to our scanned image.

Lastly, we'll use NumPy for numerical processing, `argparse` for parsing command line arguments, and `cv2` for our OpenCV bindings.

Lines 10-13 handle parsing our command line arguments. We'll need only a single switch image, `--image`, which is the path to the image that contains the document we want to scan.

Now that we have the path to our image, we can move on to Step 1: Edge Detection.

Step 1: Edge Detection

The first step to building our document scanner app using OpenCV is to perform edge detection. Let's take a look:

```
Building a Document Scanner App using Python, OpenCV, and C                Python
15 # load the image and compute the ratio of the old height
16 # to the new height, clone it, and resize it
17 image = cv2.imread(args["image"])
18 ratio = image.shape[0] / 500.0
19 orig = image.copy()
20 image = imutils.resize(image, height = 500)
21
22 # convert the image to grayscale, blur it, and find edges
23 # in the image
24 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
25 gray = cv2.GaussianBlur(gray, (5, 5), 0)
26 edged = cv2.Canny(gray, 75, 200)
27
```

```
28 # show the original image and the edge detected image
29 print "STEP 1: Edge Detection"
30 cv2.imshow("Image", image)
31 cv2.imshow("Edged", edged)
32 cv2.waitKey(0)
33 cv2.destroyAllWindows()
```

First, we load our image off disk on **Line 17**.

In order to speedup image processing, as well as make our edge detection step more accurate, we resize our scanned image to have a height of 500 pixels on **Lines 17-20**.

We also take special care to keep track of the `ratio` of the original height of the image to the new height (**Line 18**) — this will allow us to perform the scan on the *original* image rather than the *resized* image.

From there, we convert the image from RGB to grayscale on **Line 24**, perform Gaussian blurring to remove high frequency noise (aiding in contour detection in Step 2), and perform Canny edge detection on **Line 26**.

The output of Step 1 is then shown on **Lines 29-33**.

Take a look below at the example document:

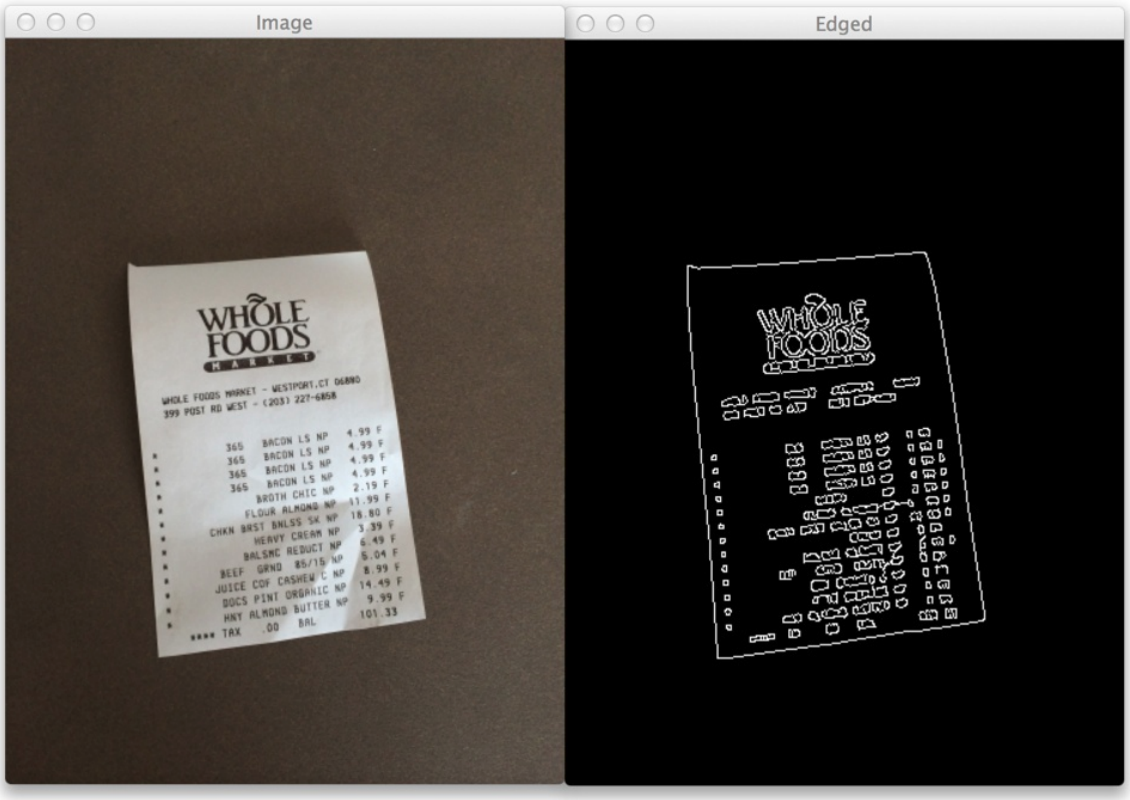


Figure 1: The first step of building a document scanning app. On the *left* we have the original

image and on the ***right*** we have the edges detected in the image.

On the left you can see my receipt from Whole Foods. Notice how the picture is captured at an angle. It is definitely not a 90-degree, top-down view of the page. Furthermore, there is also my desk in the image. Certainly this is not a "scan" of any means. We have our work cut out for us.

However, on the right you can see the image after performing edge detection. We can clearly see the outline of the receipt.

Not a bad start.

Let's move on to Step 2.

Step 2: Finding Contours

Contour detection doesn't have to be hard.

In fact, when building a document scanner, you actually have a ***serious advantage...***

Take a second to consider what we're actually building.

A document scanner simply scans in a piece of paper.

A piece of paper is assumed to be a rectangle.

And a rectangle has four edges.

Therefore, we can create a simple heuristic to help us build our document scanner.

The heuristic goes something like this: we'll assume that the ***largest contour*** in the image ***with exactly four points*** is our piece of paper to be scanned.

This is also a reasonably safe assumption — the scanner app simply assumes that the document you want to scan is the main focus of our image. And it's also safe to assume (or at least should be) that the piece of paper has four edges.

And that's exactly what the code below does:

Building a Document Scanner App using Python, OpenCV, and Compu

```
35 # find the contours in the edged image, keeping only the
36 # largest ones, and initialize the screen contour
37 (cnts, _) = cv2.findContours(edged.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPL
38 cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[:5]
39
```

```
40 # loop over the contours
41 for c in cnts:
42     # approximate the contour
43     peri = cv2.arcLength(c, True)
44     approx = cv2.approxPolyDP(c, 0.02 * peri, True)
45 
46     # if our approximated contour has four points, then we
47     # can assume that we have found our screen
48     if len(approx) == 4:
49         screenCnt = approx
50         break
51 
52 # show the contour (outline) of the piece of paper
53 print "STEP 2: Find contours of paper"
54 cv2.drawContours(image, [screenCnt], -1, (0, 255, 0), 2)
55 cv2.imshow("Outline", image)
56 cv2.waitKey(0)
57 cv2.destroyAllWindows()
```

We start off by finding the contours in our edged image on **Line 37**.

A neat performance hack that I like to do is actually sort the contours by area and keep only the largest ones (**Line 38**). This allows us to only examine the largest of the contours, discarding the rest.

We then start looping over the contours on **Line 41** and approximate the number of points on **Line 43 and 44**.

If the approximated contour has four points (**Line 48**), we assume that we have found the document in the image.

And again, this is a fairly safe assumption. The scanner app will assume that (1) the document to be scanned is the main focus of the image and (2) the document is rectangular, and thus will have four distinct edges.

From there, **Lines 53-57** display the contours of the document we went to scan.

And now let's take a look at our example image:

Figure 2: The second step of building a document scanning app is to utilize the edges in the image to find the contours of the piece of paper.

As you can see, we have successfully utilized the edge detected image to find the contour (outline) of the document, illustrated by the green rectangle surrounding

my receipt.

Lastly, let's move on to Step 3, which will be a snap using my four_point_transform function.

Step 3: Apply a Perspective Transform & Threshold

The last step in building a mobile document scanner is to take the four points representing the outline of the document and apply a perspective transform to obtain a top-down, "birds eye view" of the image.

Let's take a look:

Building a Document Scanner App using Python, OpenCV, and C — Python

```
59 # apply the four point transform to obtain a top-down
60 # view of the original image
61 warped = four_point_transform(orig, screenCnt.reshape(4, 2) * ratio)
62
63 # convert the warped image to grayscale, then threshold it
64 # to give it that 'black and white' paper effect
65 warped = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)
66 warped = threshold_adaptive(warped, 250, offset = 10)
67 warped = warped.astype("uint8") * 255
68
69 # show the original and scanned images
70 print "STEP 3: Apply perspective transform"
71 cv2.imshow("Original", imutils.resize(orig, height = 650))
72 cv2.imshow("Scanned", imutils.resize(warped, height = 650))
73 cv2.waitKey(0)
```

Line 61 performs the warping transformation. In fact, all the heavy lifting is handled by the `four_point_transform` function. Again, you can read more about this function in last week's post.

We'll pass two arguments into `four_point_transform` : the first is our original image we loaded off disk (***not*** the resized one), and the second argument is the contour representing the document, multiplied by the resized ratio.

So, you may be wondering, why are we multiplying by the resized ratio?

We multiply by the resized ratio because we performed edge detection and found contours on the resized image of *height=500* pixels.

However, we want to perform the scan on the *original* image, ***not*** the *resized* image, thus we multiply the contour points by the resized ratio.

To obtain the black and white feel to the image, we then take the warped image,

convert it to grayscale and apply adaptive thresholding on **Lines 65-67**.

Finally, we display our output on **Lines 70-73**.

And speaking of output, take a look at our example document:

Figure 3: Applying step 3 of our document scanner, perspective transform. The original image is on the *left* and the scanned image on the *right*.

On the left we have the original image we loaded off disk. And on the right, we have the scanned image!

Notice how the perspective of the scanned image has changed — we have a top-down, 90-degree view of the image.

And thanks to our adaptive thresholding, we also have a nice, clean black and white feel to the document as well.

We have successfully built our document scanner!

All in less than 5 minutes and under 75 lines of code (most of which are comments anyway).

More Examples

The receipt example was all well and good.

But will this approach work for normal pieces of paper?

You bet!

I printed out page 22 of *Practical Python and OpenCV*, a book I wrote to give you a guaranteed quick-start guide to learning computer vision:

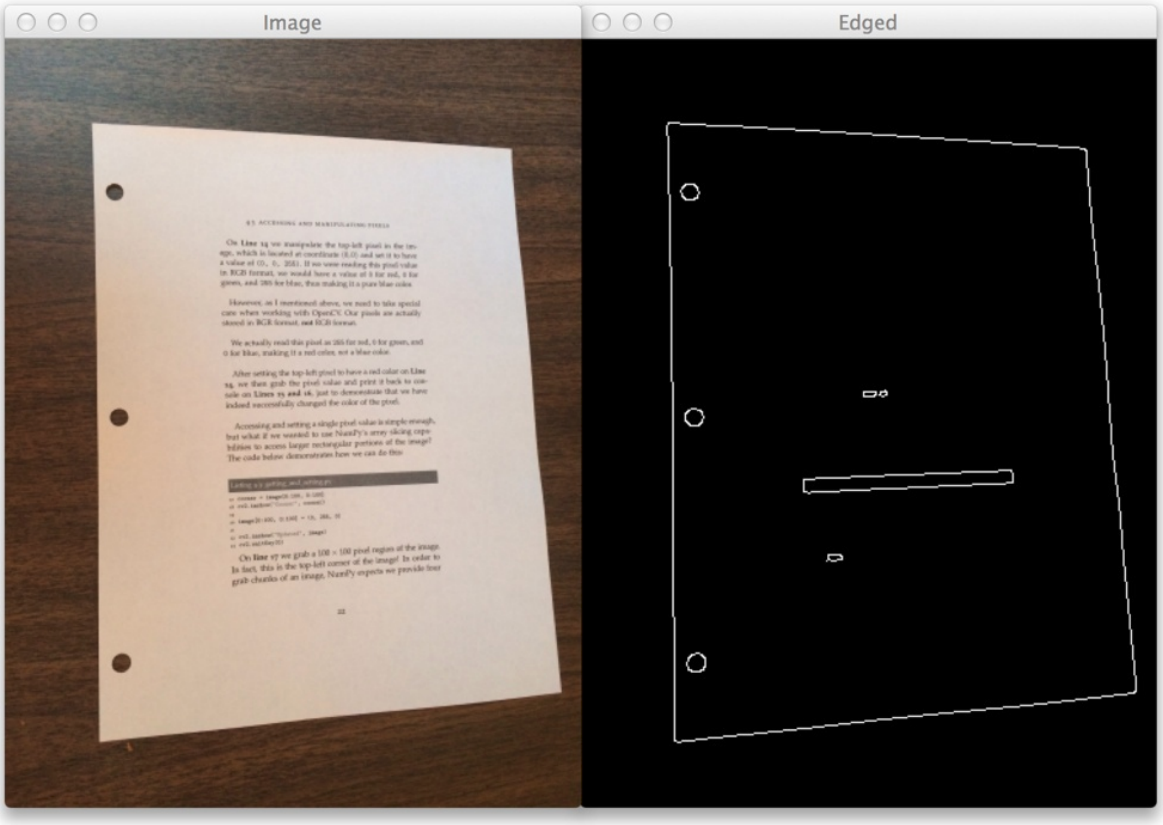


Figure 4: Applying edge detection to scan a document using computer vision.

You can see the original image on the ***left*** and the edge detected image on the ***right***.

Now, let's find the contour of the page:

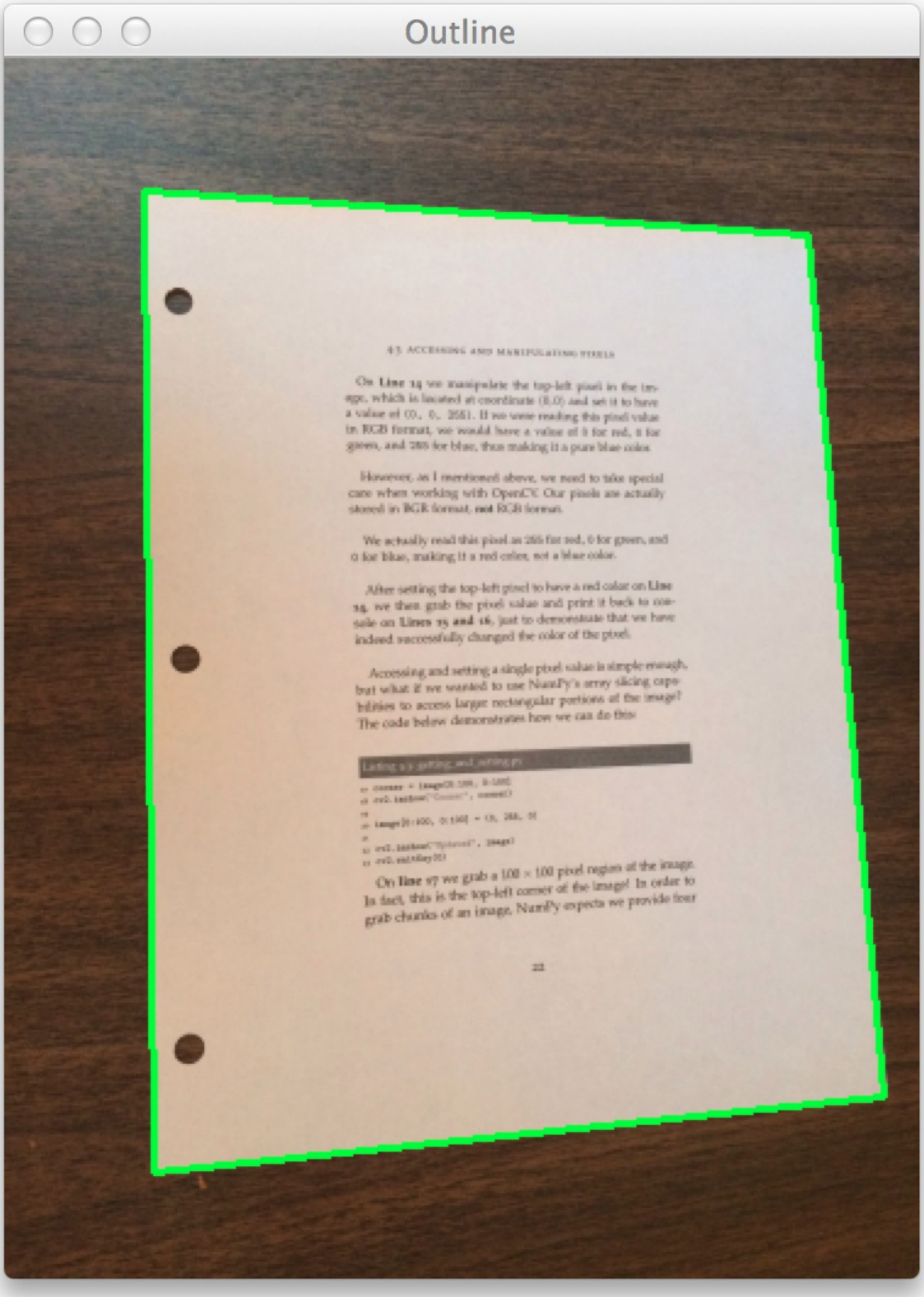


Figure 5: Using the detected images to find the contour and outline of the page to be scanned.

No problem there!

Finally, we'll apply the perspective transform and threshold the image:

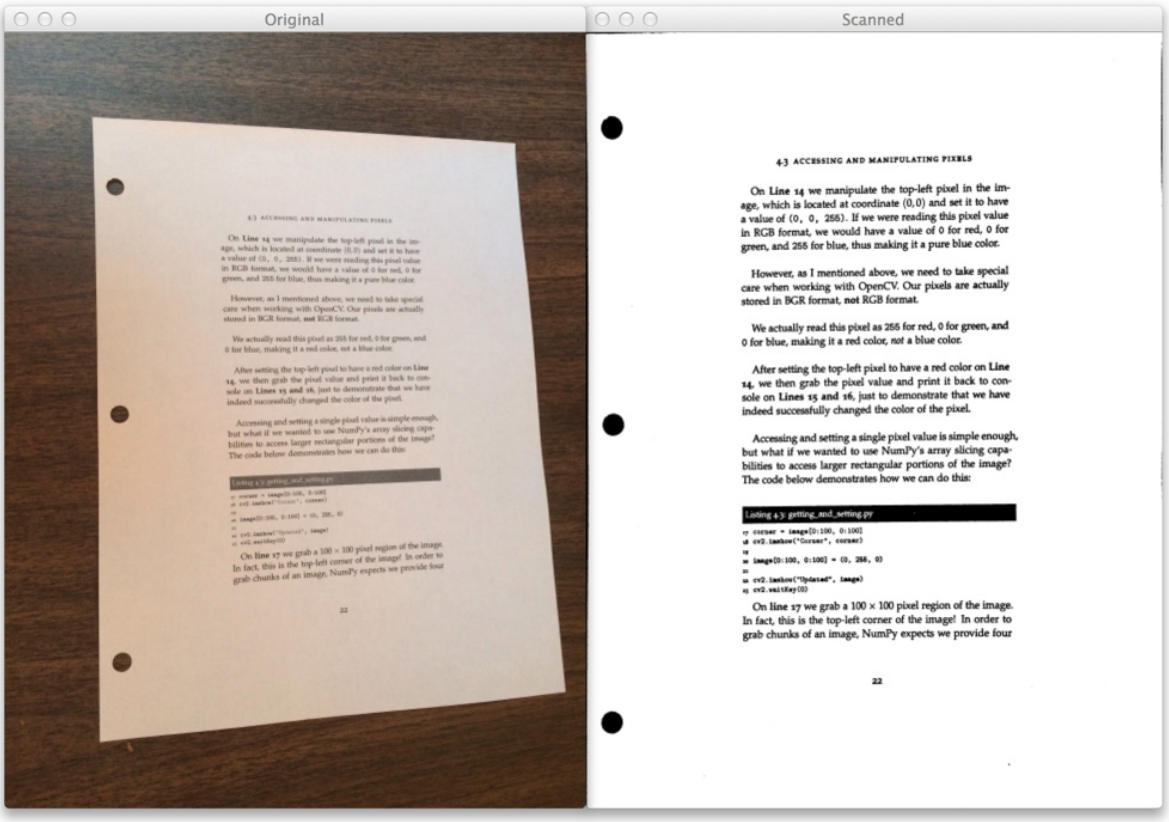


Figure 6: On the *left* we have our original image. And on the *right*, we can see the scanned version. The scan is successful!

Another successful scan!

Where to Next?

Now that you have the code to build a mobile document scanner, maybe you want to build an app and submit to the App Store yourself!

In fact, I think you should.

It would be a great learning experience...

Another great "next step" would be to apply OCR to the documents in the image. Not only could you scan the document and generate a PDF, but you would be able to edit the text as well!

Summary

In this blog post I showed you how to build a mobile document scanner using OpenCV in 5 minutes and under 75 lines of Python code.

Document scanning can be broken down into three distinct and simple steps.

The **first step** is to apply edge detection.

The **second step** is to find the contours in the image that represent the document we want to scan.

And the **final step** is to apply a perspective transform to obtain a top-down, 90-degree view of the image, just as if we scanned the document.

Optionally, you can also apply thresholding to obtain a nice, clean black and white feel to the piece of paper.

So there you have it.

A mobile document scanner in 5 minutes.

Excuse me while I call James and collect my money...

Did You Like this Post?

Hey, did you enjoy this post on building a mobile document scanner?

If so, I think you'll like my book, *Practical Python and OpenCV*.

When you pick up the Case Studies Bundle you'll learn how to **detect faces in images**, **recognize handwriting**, and **utilize keypoint detection and the SIFT descriptors** to build a system to recognize the book covers!

Sound interesting?

Just click here and pickup a copy.

And in a single weekend you'll unlock the secrets the computer vision pros use...*and become a pro yourself!*

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!

document scanner, **mobile app**, **perspective**, **transform**, **warp**

‹ 4 Point OpenCV getPerspective Transform Example

Thresholding: Simple Image Segmentation using OpenCV ›

63 Responses to *How to Build a Kick-Ass Mobile Document Scanner in Just 5 Minutes*

Aaron Altscher September 2, 2014 at 11:03 am #

REPLY

Very informative and detailed explanation. Everything this blogger publishes is gold!

Adrian Rosebrock September 2, 2014 at 12:16 pm #

REPLY