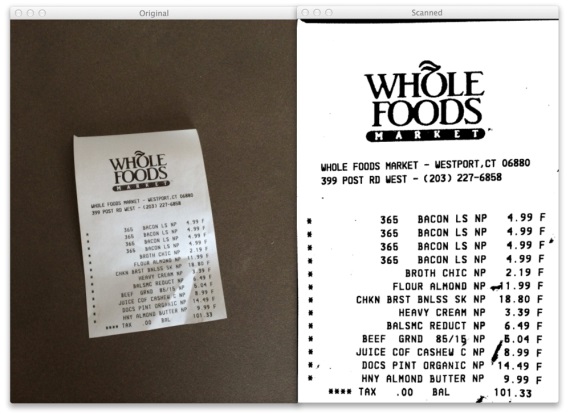
**How to Build a Kick-Ass Mobile Document Scanner in Just 5 Minutes**

[**https://www.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/**](https://www.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/)

by [**Adrian Rosebrock**](https://www.pyimagesearch.com/author/adrian/) on September 1, 2014 in [**Image Processing**](https://www.pyimagesearch.com/category/image-processing/), [**Tutorials**](https://www.pyimagesearch.com/category/tutorials/)

[](https://www.pyimagesearch.com/wp-content/uploads/2014/08/receipt-scanned.jpg)

Building a document scanner with OpenCV can be accomplished in just 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.

**OpenCV and Python versions:**  
This example will run on Python 2.7/3+ and OpenCV 2.4/3+

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](https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/).

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 itscan.py , and let’s get started.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | # import the necessary packages  from pyimagesearch.transform import four\_point\_transform  from skimage.filters import threshold\_local  import numpy as np  import argparse  import cv2  import imutils    # construct the argument parser and parse the arguments  ap = argparse.ArgumentParser()  ap.add\_argument("-i", "--image", required = True,  help = "Path to the image to be scanned")  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.](https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/) 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 [this post](https://www.pyimagesearch.com/2015/02/02/just-open-sourced-personal-imutils-package-series-opencv-convenience-functions/). To install imutils , simply:

|  |  |
| --- | --- |
| 1 | $ pip install --upgrade imutils |

Next up, let’s import the threshold\_local  function from [scikit-image](http://scikit-image.org/" \t "_blank). This function will help us obtain the “black and white” feel to our scanned image.

***Note (15 January 2018):****The threshold\_adaptive  function has been deprecated. This post has been updated to make use of threshold\_local .*

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:

|  |  |
| --- | --- |
| 15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | # load the image and compute the ratio of the old height  # to the new height, clone it, and resize it  image = cv2.imread(args["image"])  ratio = image.shape[0] / 500.0  orig = image.copy()  image = imutils.resize(image, height = 500)    # convert the image to grayscale, blur it, and find edges  # in the image  gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  gray = cv2.GaussianBlur(gray, (5, 5), 0)  edged = cv2.Canny(gray, 75, 200)    # show the original image and the edge detected image  print("STEP 1: Edge Detection")  cv2.imshow("Image", image)  cv2.imshow("Edged", edged)  cv2.waitKey(0)  cv2.destroyAllWindows() |

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

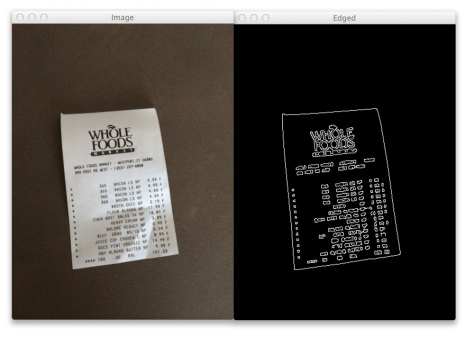
In order to speedup image processing, as well as make our edge detection step more accurate, we resize our scanned image to have a height of 500 pixels on **Lines 17-20**.

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 30 and 31**.

Take a look below at the example document:

[](https://www.pyimagesearch.com/wp-content/uploads/2014/08/receipt-edge-detected.jpg)

**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:

|  |  |
| --- | --- |
| 35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58 | # find the contours in the edged image, keeping only the  # largest ones, and initialize the screen contour  cnts = cv2.findContours(edged.copy(), cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)  cnts = imutils.grab\_contours(cnts)  cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[:5]    # loop over the contours  for c in cnts:  # approximate the contour  peri = cv2.arcLength(c, True)  approx = cv2.approxPolyDP(c, 0.02 \* peri, True)    # if our approximated contour has four points, then we  # can assume that we have found our screen  if len(approx) == 4:  screenCnt = approx  break    # show the contour (outline) of the piece of paper  print("STEP 2: Find contours of paper")  cv2.drawContours(image, [screenCnt], -1, (0, 255, 0), 2)  cv2.imshow("Outline", image)  cv2.waitKey(0)  cv2.destroyAllWindows() |

We start off by finding the contours in our edged image on **Line 37**. We also handle the fact that OpenCV 2.4, OpenCV 3, and OpenCV 4 return contours differently on **Line 38**.

A neat performance hack that I like to do is actually sort the contours by area and keep only the largest ones (**Line 39**). This allows us to only examine the largest of the contours, discarding the rest.

We then start looping over the contours on **Line 42** and approximate the number of points on **Line 44 and 45**.

If the approximated contour has four points (**Line 49**), 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 55 and 56** display the contours of the document we went to scan.

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

[](https://www.pyimagesearch.com/wp-content/uploads/2014/08/receipt-outlined.jpg)

**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:

|  |  |
| --- | --- |
| 60  61  62  63  64  65  66  67  68  69  70  71  72  73  74 | # apply the four point transform to obtain a top-down  # view of the original image  warped = four\_point\_transform(orig, screenCnt.reshape(4, 2) \* ratio)    # convert the warped image to grayscale, then threshold it  # to give it that 'black and white' paper effect  warped = cv2.cvtColor(warped, cv2.COLOR\_BGR2GRAY)  T = threshold\_local(warped, 11, offset = 10, method = "gaussian")  warped = (warped > T).astype("uint8") \* 255    # show the original and scanned images  print("STEP 3: Apply perspective transform")  cv2.imshow("Original", imutils.resize(orig, height = 650))  cv2.imshow("Scanned", imutils.resize(warped, height = 650))  cv2.waitKey(0) |

**Line 62** performs the warping transformation. In fact, all the heavy lifting is handled by thefour\_point\_transform  function. Again, you can read more about this function [in last week’s post](https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/).

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 66-68**.

Finally, we display our output on **Lines 72-74**.

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

[](https://www.pyimagesearch.com/wp-content/uploads/2014/08/receipt-scanned.jpg)

**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).

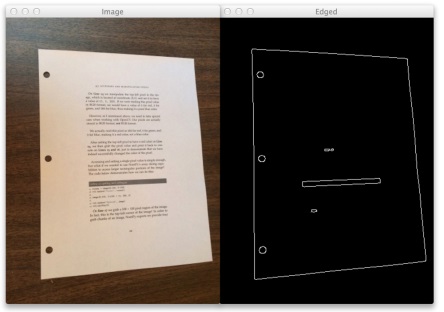
**More Examples**

The receipt example was all well and good.

But will this approach work for normal pieces of paper?

You bet!

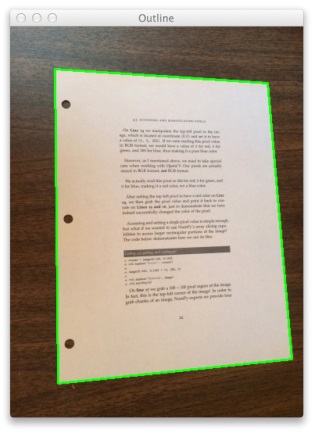
I printed out page 22 of [*Practical Python and OpenCV*](https://www.pyimagesearch.com/practical-python-opencv/), a book I wrote to give you a guaranteed quick-start guide to learning computer vision:

[](https://www.pyimagesearch.com/wp-content/uploads/2014/08/page-edge-detected.jpg)

**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.

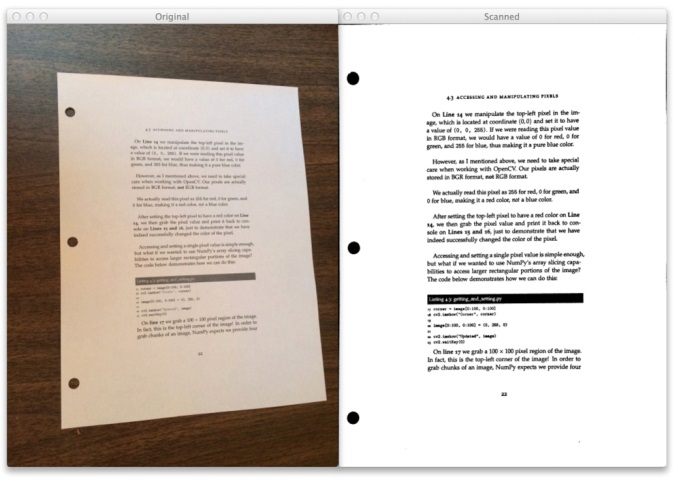
Now, let’s find the contour of the page:

[](https://www.pyimagesearch.com/wp-content/uploads/2014/08/page-outlined.jpg)

**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:

[](https://www.pyimagesearch.com/wp-content/uploads/2014/08/page-scanned.jpg)

**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…