Navigation

Building a Pokedex in Python: OpenCV and Perspective Warping (Step 5 of 6)

by **Adrian Rosebrock** on May 5, 2014 in **Building a Pokedex**, **Examples of Image Search Engines**, **Tutorials**

3 1

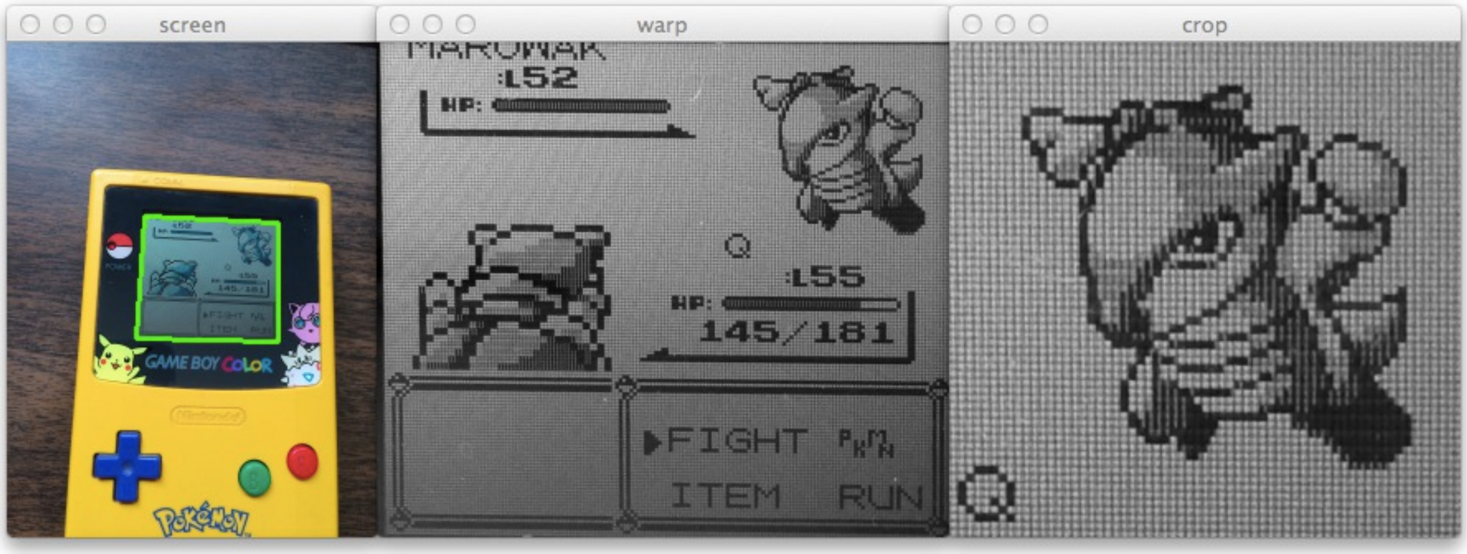


Figure 1: Performing a perspective transformation using Python and OpenCV on the Game Boy screen and cropping out the Pokemon.

We're getting closer to finishing up our real-life Pokedex!

In my previous blog post, I showed you how to find a Game Boy screen in an image using Python and OpenCV.

This post will show you how to apply warping transformations to obtain a "birds-eye-view" of the Game Boy screen. From there, we will be able to crop out the actual Pokemon and feed it into our Pokemon identification algorithm.

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**.

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 you are reading, definitely check it out! But after you give it a read, be sure to go back and review the previous posts — there is a TON of awesome computer vision and image processing content in there.

Finally, if you have have any questions, feel free to shoot me an email. I would be happy to chat.

- **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)
- **Step 3:** Building a Pokedex in Python: Indexing our Sprites using Shape Descriptors (Step 3 of 6)
- **Step 4:** Building a Pokedex in Python: Finding the Game Boy Screen (Step 4 of 6)

Building a Pokedex in Python: OpenCV Perspective Transform Example

When we wrapped up the previous post on building a Pokedex in Python, we were able to find our Game Boy screen by applying edge detection, finding contours, and then approximating the contours, like this:

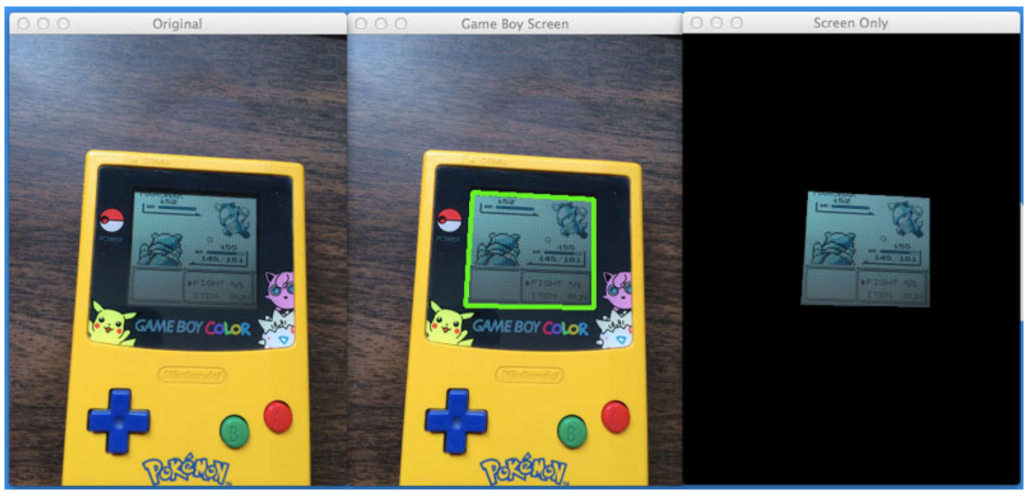


Figure 2: Finding a Game Boy screen in an image using Python and OpenCV.

However, you may notice that the Game Boy screen is slightly skewed — the screen is definitely leaning to the right.

The perspective of the screen is also wrong. Ideally, we would want to have a top-down, birds-eye-view of the Game Boy screen, as in Figure 1.

How are we going to accomplish this?

Let's jump into some code.

We'll be building off the code in the previous post, so if it looks like we are jumping into the middle of a file, it's because we are.

OpenCV and Perspective Warping using Python — Python

```python
45 # now that we have our screen contour, we need to determine
46 # the top-left, top-right, bottom-right, and bottom-left
47 # points so that we can later warp the image -- we'll start
48 # by reshaping our contour to be our finals and initializing
49 # our output rectangle in top-left, top-right, bottom-right,
50 # and bottom-left order
51 pts = screenCnt.reshape(4, 2)
52 rect = np.zeros((4, 2), dtype = "float32")
53
54 # the top-left point has the smallest sum whereas the
55 # bottom-right has the largest sum
56 s = pts.sum(axis = 1)
57 rect[0] = pts[np.argmin(s)]
58 rect[2] = pts[np.argmax(s)]
59
60 # compute the difference between the points -- the top-right
61 # will have the minumum difference and the bottom-left will
62 # have the maximum difference
63 diff = np.diff(pts, axis = 1)
64 rect[1] = pts[np.argmin(diff)]
65 rect[3] = pts[np.argmax(diff)]
66
67 # multiply the rectangle by the original ratio
68 rect *= ratio
```

On **Line 51** we are are reshaping the contour that corresponds to the outline of the screen. The contour has four points, the four points of the rectangular region of the screen. We are simply reshaping the NumPy array of points to make them easier to work with.

In order to apply a perspective transformation, we need to know the top-left, top-right, bottom-right, and bottom-left corners of the contour. However, just because we have the contour that corresponds to the Game Boy screen, we have no guarantee of the *order* of the points. There is no guarantee that the top-left point is the first point in the contour list. It might be the second point. Or the fourth point.

To handle this problem we'll have to impose a strict order on the points. We start on **Line 52** by initializing our rectangle of shape (4, 2) to store the ordered points.

Line 56-58 handles grabbing the top-left and bottom-right points. **Line 56** handles summing the *(x, y)* coordinates together by specifying axis=1. The top-left point will have the smallest sum (**Line 57**), whereas the bottom-right point will have the largest sum (**Line 58**).

Now we need to grab the top-right and bottom-left points on **Line 63-65** by taking the difference between the *(x, y)* coordinates. The top-right point will have the smallest difference (**Line 64**), whereas the bottom-left point will have the largest difference (**Line 65**).

Notice how our points are now stored in an imposed order: top-left, top-right, bottom-right, and bottom-left. Keeping a consistent order is important when we apply our perspective transformation.

If you remember back to the previous post, we resized our image to make image processing and edge detection faster and more accurate. We kept track of this resizing ratio for a good reason — when we crop out of Game Boy screen, we want to crop out the *original* Game Boy screen, *not* the smaller, resized one.

In oder to extract the original, large Game Boy screen, we multiply our rect by the ratio, t size.

Next, we need to calculate the size of the Game Boy screen so that we can allocate memory to store it:

OpenCV and Perspective Warping using Python — Python

```python
70 # now that we have our rectangle of points, let's compute
71 # the width of our new image
72 (tl, tr, br, bl) = rect
73 widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
74 widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
75
76 # ...and now for the height of our new image
77 heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
78 heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
79
80 # take the maximum of the width and height values to reach
81 # our final dimensions
82 maxWidth = max(int(widthA), int(widthB))
83 maxHeight = max(int(heightA), int(heightB))
84
85 # construct our destination points which will be used to
86 # map the screen to a top-down, "birds eye" view
87 dst = np.array([
88     [0, 0],
89     [maxWidth - 1, 0],
90     [maxWidth - 1, maxHeight - 1],
91     [0, maxHeight - 1]], dtype = "float32")
92
93 # calculate the perspective transform matrix and warp
94 # the perspective to grab the screen
95 M = cv2.getPerspectiveTransform(rect, dst)
96 warp = cv2.warpPerspective(orig, M, (maxWidth, maxHeight))
```

Let's take this code apart and see what's going on:

- **Line 72:** Here we are unpacking our rect and grabbing the top-left, top-right, bottom-right, and bottom-left points, respectively.
- **Line 73:** In order to determine the width of the image, we compute the distance between the *x* coordinates of the bottom-right and bottom-left points.
- **Line 74:** Similarly, we compute the distance between the *x* coordinates of the top-right and top-left points.
- **Lines 77 and 78:** Just like we computed the distance between the *x* coordinate points, we now need to do the same for the *y* coordinate points.
- **Lines 82 and 83:** Now that we have our distances, we take the maximum of widthA and widthB to determine the width of our transformed image. We then repeat the process for heightA and heightB to determine the dimensions of the new image.
- **Lines 87-91:** Remember how I said the order of the points is important? In order to compute the birds-eye-view of the Game Boy screen we need to construct a matrix dst to handle the mapping. The first entry in dst is the origin of the image — the top-left corner. We then specify the top-right, bottom-right, and bottom-left points based on our calculated width and height.
- **Line 95:** To compute the perspective transformation, we need the actual transformation matrix. This matrix is calculated by making a call to cv2.getPerspective transformation and passing in the coordinates of the Game Boy screen in the original image, followed by the four points we specified for our output image. In return, we are given our transformation matrix M.
- **Line 96:** Finally, we can apply our transformation by calling the cv2.warpPerspective function. The first parameter is our original image that we want to warp, the second is our transformation matrix M obtained from cv2.getPerspective, and the final parameter is a tuple, used to indicate the width and height of the output image.

If all goes well, we should now have a top-down/birds-eye-view of our Game Boy screen:

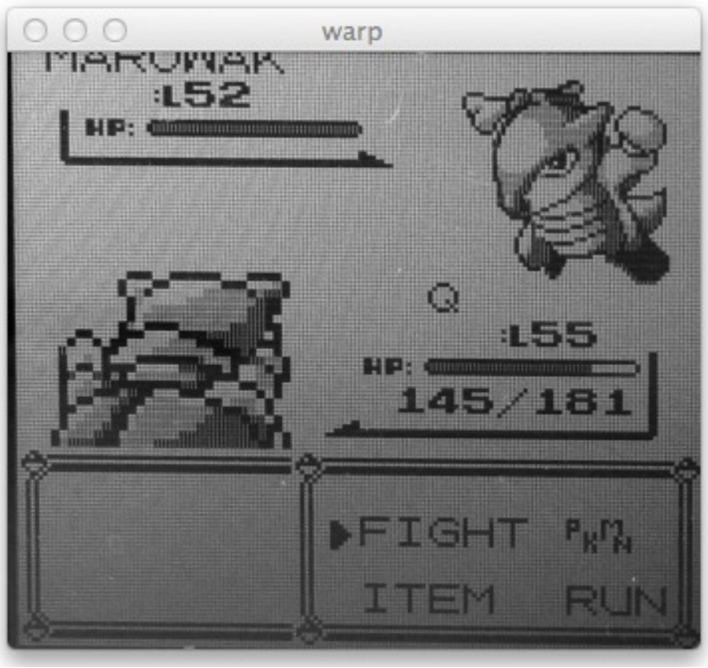


Figure 2: Obtaining a top-down/birds-eye-view of an image using Python, OpenCV, and perspective warping and transform

But we aren't done yet!

We still need to crop out the actual Pokemon from the top-right portion of the screen.

Furthermore, you'll notice that our Marowak seems to be a bit "shadowy" and the screen of the Game Boy itself is darker than we would like it to be. We need to see if we can re-scale the intensity of our image to help mitigate this shadow and make it easier to extract the contour of the Marowak, later allowing us to compute shape features over the Pokemon outline.

OpenCV and Perspective Warping using Python Python

```python
98  # convert the warped image to grayscale and then adjust
99  # the intensity of the pixels to have minimum and maximum
100 # values of 0 and 255, respectively
101 warp = cv2.cvtColor(warp, cv2.COLOR_BGR2GRAY)
102 warp = exposure.rescale_intensity(warp, out_range = (0, 255))
103
104 # the pokemon we want to identify will be in the top-right
105 # corner of the warped image -- let's crop this region out
106 (h, w) = warp.shape
107 (dX, dY) = (int(w * 0.4), int(h * 0.45))
108 crop = warp[10:dY, w - dX:w - 10]
109
110 # save the cropped image to file
111 cv2.imwrite("cropped.png", crop)
112
113 # show our images
114 cv2.imshow("image", image)
115 cv2.imshow("edge", edged)
116 cv2.imshow("warp", imutils.resize(warp, height = 300))
117 cv2.imshow("crop", imutils.resize(crop, height = 300))
118 cv2.waitKey(0)
```

The first thing we'll do is convert our warped image to grayscale on **Line 100**. Then, we make use of the skimage Python library. We make a call to the rescale_intensity method in the exposure sub-package. This method takes our warped image and then re-scales the gray pixel intensities by finding the minimum and maximum values. The minimum value then becomes black (a value of 0) and the maximum value then becomes white (a value of 255). All pixels that fall into that range are scaled accordingly.

The output of this re-scaling can be seen below:

Figure 3: Re-scaling the intensity of pixels using scikit-image.

Notice how that shadow region is much less apparent.

From here, all we need is some simple cropping.

We grab the height and width of the warped Game Boy screen on **Line 106** and then determine a region that is 40% of the width and 45% of the height on **Line 107** — the Pokemon that we want to identify will lie within this region of the image:

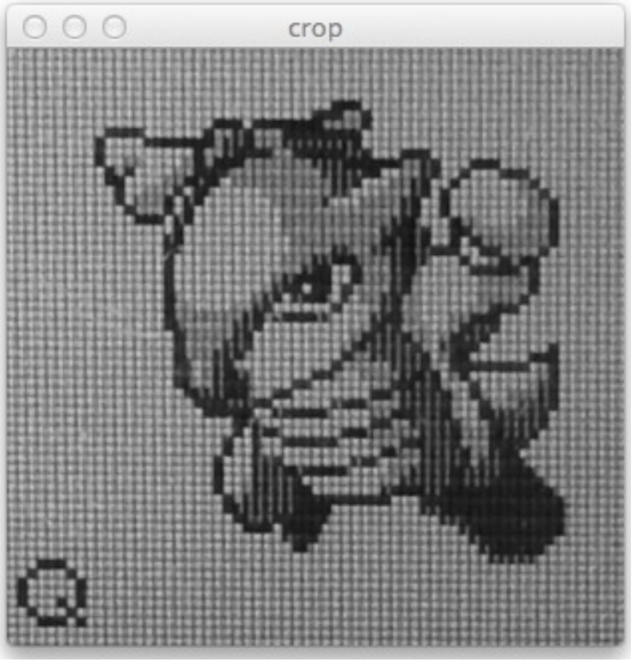


Figure 4: Cropping the Pokemon from our Game Boy screen using Python and OpenCV.

Note: I determined these percentages empirically by trial and error. There is no fancy computer vision magic going on. Just your standard testing and debugging to find the correct percentages.

We crop the Pokemon from the Game Boy screen on **Line 108** and write it to file on **Line 111**. In the next (and final) blog post in this series we'll use this cropped image to perform the actual identification of the Pokemon.

Finally, **Lines 114-118** just show us the results of our labor:

To execute our script to the Pokemon in the Game Boy screen, simply execute the following command:

Applying perspective warping and transformations using Python and OpenCV

```
$ python find_screen.py --query queries/query_marowak.jpg
```

Summary

In this blog post we applied perspective and warping transformations using Python and OpenCV. We utilized the cv2.getPerspectiveTransform and cv2.warpPerspective functions to accomplish these transformations. We then reviewed a perspective transform OpenCV example.

We applied these techniques to obtain a top-down/birds-eye-view of our Game Boy screen, allowing us to crop out the Pokemon we want to identify. This example demonstrated the OpenCV perspective transform.

Finally, we used scikit-image to rescale the pixel intensity of the grayscale cropped image.

My next post will wrap up this series of post and tie everything together. We will take our cropped Pokemon and then run it through our identification algorithm.

From there, we'll have a real-life working Pokedex!

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**

opencv, **perspective**, **pokedex**, **pokemon**, **skimage**, **transformation**, **warp**

‹ Building a Pokedex in Python: Finding the Game Boy Screen (Step 4 of 6) The Best Python Books of 2014 ›

6 Responses to *Building a Pokedex in Python: OpenCV and Perspective Warping (Step 5 of 6)*

jonatslim January 21, 2015 at 11:59 pm # REPLY

Where is guide 6-of-6? This feels like watching a movie when the climax of the story comes, the power goes out ! :d

Adrian Rosebrock January 22, 2015 at 7:17 am # REPLY

The last part of the guide can be found here: http://www.pyimagesearch.com/2014/05/19/building-pokedex-python-comparing-shape-descriptors-opencv/

Niall M ODowd March 31, 2016 at 6:18 pm # REPLY

Hi Adrian,

Your sample code, awesome explanation, and annotation have helped me create a live transforming script that basically finds 4 corners on a piece of paper in the outside world and remaps the points to a perfect square using a webcam.

The transform matrix is used to transform the whole webcam image and display the image as if the webcam was normal to the surface of the square.

My current dilemma is accuracy. it seems that with all of the subpix and goodfeaturetotrack parameter fiddling, I simply cannot get a corner list that does not bounce around. though the shifting of the corners is slight, the transformation matrices vary a lot.

Is there a way to improve accuracy? (maybe use the sidelines of the square to boost orientation accuracy?) I have spent a ton of time trying to improve the shifting, but I just need more information from the webcame frame.

Thanks,
Niall

Adrian Rosebrock April 1, 2016 at 3:19 pm # REPLY

Your project sounds super awesome. Do you mind sending me an email containing the types of images you're working with? That might help me point you in the right direction. I'm not entirely sure I understand what you mean by the corner list "bouncing around".

Trackbacks/Pingbacks

Comparing Shape Descriptors for Similarity using Python and OpenCV - May 30, 2014
[…] We explored what it takes to build a Pokedex using computer vision. Then we scraped 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 finding techniques. And we've performed perspective warping and transformations using the cv2.warpPerspective function. […]
4 Point OpenCV getPerspective Transform Example - PyImageSearch - August 25, 2014
[…] You may remember back to my posts on building a real-life Pokedex, specifically, my post on OpenCV and Perspective Warping. […]

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

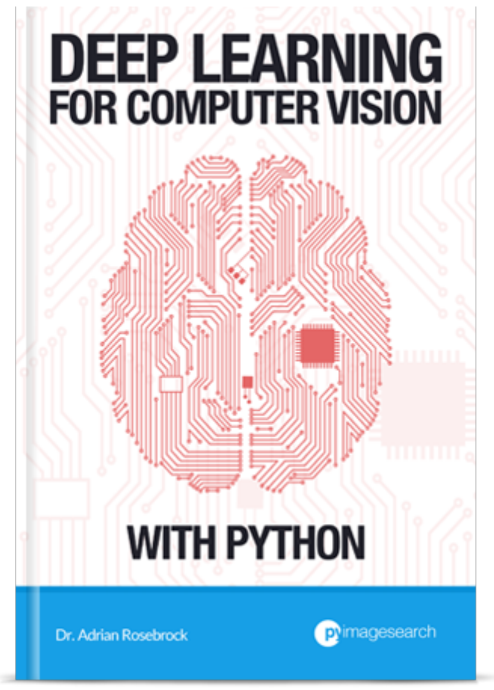
SUBMIT COMMENT

Resource Guide (it's totally free).

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

Download for Free!

Deep Learning for Computer Vision with Python Book



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 yoursel

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.

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

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...

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