PYIMAGESEARCH

Click here to download the source code to this post

IMAGE PROCESSING (HTTPS://PYIMAGESEARCH.COM/CATEGORY/IMAGE-PROCESSING/)

TUTORIALS (HTTPS://PYIMAGESEARCH.COM/CATEGORY/TUTORIALS/)

Ordering coordinates clockwise with Python and OpenCV

by **Adrian Rosebrock (https://pyimagesearch.com/author/adrian/)** *on* March 21, 2016

Today we are going to kick-off a three part series on **calculating the size of objects in images** along with **measuring the *distances* between them**.

These tutorials have been some of the most *heavily requested* lessons on the PyImageSearch blog. I'm super excited to get them underway — *and I'm sure you are too.*

However, before we start learning how to measure the size (and not to mention, the distance

between) objects in images, we first need to talk about something...

A little over a year ago, I wrote one my favorite tutorials on the PyImageSearch blog: **How to build a kick-ass mobile document scanner in just 5 minutes (https://pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/)**. Even though this tutorial is over a year old, its *still* one of the most popular blog posts on PyImageSearch.

Building our mobile document scanner was predicated on our ability to **apply a 4 point cv2.getPerspectiveTransform with OpenCV (https://pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/)**, enabling us to obtain a top-down, birds-eye-view of our document.

However. Our perspective transform has a ***deadly flaw*** that makes it unsuitable for use in production environments.

You see, there are cases where the pre-processing step of arranging our four points in top-left, top-right, bottom-right, and bottom-left order can return *incorrect results!*

To learn more about this bug, *and how to squash it*, keep reading.

Click here to download the source code to this post

Looking for the source code to this post?

JUMP RIGHT TO THE DOWNLOADS SECTION →

Ordering coordinates clockwise with Python and OpenCV

The goal of this blog post is two-fold:

1. The **primary purpose** is to learn how to arrange the *(x, y)*-coordinates associated with a rotated bounding box in top-left, top-right, bottom-right, and bottom-left order. Organizing bounding box coordinates in such an order is a prerequisite to performing operations such as perspective transforms or matching corners of objects (such as when we compute the

distance between objects).

Click here to download the source code to this post

2. The **secondary purpose** is to address a subtle, hard-to-find bug in the `order_points` method of the **imutils package (https://github.com/jrosebr1/imutils)**. By resolving this bug, our `order_points` function will no longer be susceptible to a debilitating bug.

All that said, let's get this blog post started by reviewing the original, flawed method at ordering our bounding box coordinates in clockwise order.

The original (flawed) method

Before we can learn how to arrange a set of bounding box coordinates in (1) clockwise order and more specifically, (2) a top-left, top-right, bottom-right, and bottom-left order, we should first review the `order_points` method detailed in the **original 4 point getPerspectiveTransform blog post (https://pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/)**.

I have renamed the (flawed) `order_points` method to `order_points_old` so we can compare our original and updated methods. To get started, open up a new file and name it `order_coordinates.py`:

Ordering coordinates clockwise with Python and OpenCV

```
# import the necessary packages
from __future__ import print_function
from imutils import perspective
from imutils import contours
import numpy as np
import argparse
import imutils
import cv2

def order_points_old(pts):
    # initialize a list of coordinates that will be ordered
    # such that the first entry in the list is the top-left,
    # the second entry is the top-right, the third is the
    # bottom-right, and the fourth is the bottom-left
    rect = np.zeros((4, 2), dtype="float32")

    # the top-left point will have the smallest sum, whereas
    # the bottom-right point will have the largest sum
    s = pts.sum(axis=1)
    rect[0] = pts[np.argmin(s)]
    rect[2] = pts[np.argmax(s)]

    # now, compute the difference between the points, the
    # top-right point will have the smallest difference,
    # whereas the bottom-left will have the largest difference
    diff = np.diff(pts, axis=1)
    rect[1] = pts[np.argmin(diff)]
```

```
28.     rect[3] = pts[np.argmax(diff)]
29.
30.     # return the ordered coordinates
31.     return rect
```

Click here to download the source code to this post

Lines 2-8 handle importing our required Python packages for this example. We'll be using the `imutils` package later in this blog post, so if you don't already have it installed, be sure to install it via `pip`:

```
Ordering coordinates clockwise with Python and OpenCV
1. $ pip install imutils
```

Otherwise, if you *do* have `imutils` installed, you should upgrade to the latest version (which has the updated `order_points` implementation):

```
Ordering coordinates clockwise with Python and OpenCV
1. $ pip install --upgrade imutils
```

Line 10 defines our `order_points_old` function. This method requires only a single argument, the set of points that we are going to arrange in top-left, top-right, bottom-right, and bottom-left order; although, as we'll see, this method has some flaws.

We start on **Line 15** by defining a NumPy array with shape `(4, 2)` which will be used to store our set of four *(x, y)*-coordinates.

Given these `pts`, we add the *x* and *y* values together, followed by finding the smallest and largest sums (**Lines 19-21**). These values give us our top-left and bottom-right coordinates, respectively.

We then take the difference between the *x* and *y* values, where the top-right point will have the smallest difference and the bottom-left will have the largest distance (**Lines 26-28**).

Finally, **Line 31** returns our ordered *(x, y)*-coordinates to our calling function.

So all that said, can you spot the flaw in our logic?

I'll give you a hint:

What happens when the sum or difference of the two points is the* same*?

In short, tragedy.

If either the sum array `s` or the difference array `diff` have the same values, we are at risk of choosing the incorrect index, which causes a cascade affect on our ordering.

Selecting the wrong index implies that we chose the incorrect point from our `pts` list. And if we

take the incorrect point from `pts`, then our clockwise top-left, top-right, bottom-right, bottom-left ordering will be be destroyed.

Click here to download the source code to this post

So how can we address this problem and ensure that it doesn't happen?

To handle this problem, we need to devise a better `order_points` function using more sound mathematic principles. And that's exactly what we'll cover in the next section.

A better method to order coordinates clockwise with OpenCV and Python

Now that we have looked at a *flawed* version of our `order_points` function, let's review an *updated, correct* implementation.

The implementation of the `order_points` function we are about to review can be found in the **imutils package (https://github.com/jrosebr1/imutils)**; specifically, in the **perspective.py file (https://github.com/jrosebr1/imutils/blob/master/imutils/perspective.py)**. I've included the exact implementation in this blog post as a matter of completeness:

Ordering coordinates clockwise with Python and OpenCV

```
1.  # import the necessary packages
2.  from scipy.spatial import distance as dist
3.  import numpy as np
4.  import cv2
5.
6.  def order_points(pts):
7.      # sort the points based on their x-coordinates
8.      xSorted = pts[np.argsort(pts[:, 0]), :]
9.
10.     # grab the left-most and right-most points from the sorted
11.     # x-roodinate points
12.     leftMost = xSorted[:2, :]
13.     rightMost = xSorted[2:, :]
14.
15.     # now, sort the left-most coordinates according to their
16.     # y-coordinates so we can grab the top-left and bottom-left
17.     # points, respectively
18.     leftMost = leftMost[np.argsort(leftMost[:, 1]), :]
19.     (tl, bl) = leftMost
20.
21.     # now that we have the top-left coordinate, use it as an
22.     # anchor to calculate the Euclidean distance between the
23.     # top-left and right-most points; by the Pythagorean
24.     # theorem, the point with the largest distance will be
25.     # our bottom-right point
26.     D = dist.cdist(tl[np.newaxis], rightMost, "euclidean")[0]
27.     (br, tr) = rightMost[np.argsort(D)[::-1], :]
28.
29.     # return the coordinates in top-left, top-right,
30.     # bottom-right, and bottom-left order
31.     return np.array([tl, tr, br, bl], dtype="float32")
```

Click here to download the source code to this post

Again, we start off on **Lines 2-4** by importing our required Python packages. We then define our `order_points` function on **Line 6** which requires only a single parameter — the list of `pts` that we want to order.

Line 8 then sorts these `pts` based on their *x*-values. Given the sorted `xSorted` list, we apply **array slicing (http://stackoverflow.com/questions/509211/explain-pythons-slice-notation)** to grab the two left-most points along with the two right-most points (**Lines 12 and 13**).

The `leftMost` points will thus correspond to the *top-left* and *bottom-left* points while `rightMost` will be our *top-right* and *bottom-right* points — **the trick is to figure out which is which.**

Luckily, this isn't too challenging.

If we sort our `leftMost` points according to their *y*-value, we can derive the top-left and bottom-left points, respectively (**Lines 18 and 19**).

Then, to determine the bottom-right and bottom-left points, we can apply a bit of geometry.

Using the top-left point as an anchor, we can apply the **Pythagorean theorem (https://en.wikipedia.org/wiki/Pythagorean_theorem)** and compute the **Euclidean distance (https://en.wikipedia.org/wiki/Euclidean_distance)** between the top-left and `rightMost` points. By the definition of a triangle, the hypotenuse will be the largest side of a right-angled triangle.

Thus, by taking the top-left point as our anchor, the bottom-right point will have the largest Euclidean distance, allowing us to extract the bottom-right and top-right points (**Lines 26 and 27**).

Finally, **Line 31** returns a NumPy array representing our ordered bounding box coordinates in top-left, top-right, bottom-right, and bottom-left order.

Testing our coordinate ordering implementations

Now that we have both the *original* and *updated* versions of `order_points`, let's continue the implementation of our `order_coordinates.py` script and give them both a try:

```
Ordering coordinates clockwise with Python and OpenCV

1. # import the necessary packages
2. from __future__ import print_function
3. from imutils import perspective
4. from imutils import contours
5. import numpy as np
6. import argparse
7. import imutils
```

```
7.  import imutils
8.  import cv2
9.
10. def order_points_old(pts):
11.     # initialize a list of coordinates that will be ordered
12.     # such that the first entry in the list is the top-left,
13.     # the second entry is the top-right, the third is the
14.     # bottom-right, and the fourth is the bottom-left
15.     rect = np.zeros((4, 2), dtype="float32")
16.
17.     # the top-left point will have the smallest sum, whereas
18.     # the bottom-right point will have the largest sum
19.     s = pts.sum(axis=1)
20.     rect[0] = pts[np.argmin(s)]
21.     rect[2] = pts[np.argmax(s)]
22.
23.     # now, compute the difference between the points, the
24.     # top-right point will have the smallest difference,
25.     # whereas the bottom-left will have the largest difference
26.     diff = np.diff(pts, axis=1)
27.     rect[1] = pts[np.argmin(diff)]
28.     rect[3] = pts[np.argmax(diff)]
29.
30.     # return the ordered coordinates
31.     return rect
32.
33. # construct the argument parse and parse the arguments
34. ap = argparse.ArgumentParser()
35. ap.add_argument("-n", "--new", type=int, default=-1,
36.     help="whether or not the new order points should should be used")
37. args = vars(ap.parse_args())
38.
39. # load our input image, convert it to grayscale, and blur it slightly
40. image = cv2.imread("example.png")
41. gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
42. gray = cv2.GaussianBlur(gray, (7, 7), 0)
43.
44. # perform edge detection, then perform a dilation + erosion to
45. # close gaps in between object edges
46. edged = cv2.Canny(gray, 50, 100)
47. edged = cv2.dilate(edged, None, iterations=1)
48. edged = cv2.erode(edged, None, iterations=1)
```


Lines 33-37 handle parsing our command line arguments. We only need a single argument, `--new`, which is used to indicate whether or not the *new* or the *original* `order_points` function should be used. We'll default to using the *original* implementation.

From there, we load `example.png` from disk and perform a bit of pre-processing by converting the image to grayscale and smoothing it with a Gaussian filter.

We continue to process our image by applying the Canny edge detector, followed by a dilation + erosion to close any gaps between outlines in the edge map.

After performing the edge detection process, our image should look like this:

Figure 1: Computing the edge map of the input image.

As you can see, we have been able to determine the outlines/contours of the objects in the image.

Now that we have the outlines of the edge map, we can apply the `cv2.findContours` function to actually *extract* the outlines of the objects:

```
Ordering coordinates clockwise with Python and OpenCV
50. | # find contours in the edge map
51. | cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
52. |    cv2.CHAIN_APPROX_SIMPLE)
53. | cnts = imutils.grab_contours(cnts)
54. |
55. | # sort the contours from left-to-right and initialize the bounding box
56. | # point colors
57. | (cnts, _) = contours.sort_contours(cnts)
58. | colors = ((0, 0, 255), (240, 0, 159), (255, 0, 0), (255, 255, 0))
```

We then sort the object contours from left-to-right, which isn't a requirement, but makes it easier to view the output of our script.

The next step is to loop over each of the contours individually:

Click here to download the source code to this post

```
60. | # loop over the contours individually
61. | for (i, c) in enumerate(cnts):
62. |     # if the contour is not sufficiently large, ignore it
63. |     if cv2.contourArea(c) < 100:
64. |         continue
65. |
66. |     # compute the rotated bounding box of the contour, then
67. |     # draw the contours
68. |     box = cv2.minAreaRect(c)
69. |     box = cv2.cv.BoxPoints(box) if imutils.is_cv2() else cv2.boxPoints(box)
70. |     box = np.array(box, dtype="int")
71. |     cv2.drawContours(image, [box], -1, (0, 255, 0), 2)
72. |
73. |     # show the original coordinates
74. |     print("Object #{}:".format(i + 1))
75. |     print(box)
```

Line 61 starts looping over our contours. If a contour is not sufficiently large (due to "noise" in the edge detection process), we discard the contour region (**Lines 63 and 64**).

Otherwise, **Lines 68-71** handle computing the rotated bounding box of the contour (taking care to use `cv2.cv.BoxPoints` [if we are using OpenCV 2.4] or `cv2.boxPoints` [if we are using OpenCV 3]) and drawing the contour on the `image`.

We'll also print the original rotated bounding `box` so we can compare the results after we order the coordinates.

We are now ready to order our bounding box coordinates in a clockwise arrangement:


```
77. |     # order the points in the contour such that they appear
78. |     # in top-left, top-right, bottom-right, and bottom-left
79. |     # order, then draw the outline of the rotated bounding
80. |     # box
81. |     rect = order_points_old(box)
82. |
83. |     # check to see if the new method should be used for
84. |     # ordering the coordinates
85. |     if args["new"] > 0:
86. |         rect = perspective.order_points(box)
87. |
88. |     # show the re-ordered coordinates
89. |     print(rect.astype("int"))
90. |     print("")
```

Line 81 applies the *original* (i.e., flawed) `order_points_old` function to arrange our bounding box coordinates in top-left, top-right, bottom-right, and bottom-left order.

If the `--new 1` flag has been passed to our script, then we'll apply our *updated*

`order_points` function (**Lines 85 and 86**).

Click here to download the source code to this post

Just like we printed the *original bounding box* to our console, we'll also print the *ordered points* so we can ensure our function is working properly.

Finally, we can visualize our results:

```
Ordering coordinates clockwise with Python and OpenCV
92.     # loop over the original points and draw them
93.     for ((x, y), color) in zip(rect, colors):
94.         cv2.circle(image, (int(x), int(y)), 5, color, -1)
95.
96.     # draw the object num at the top-left corner
97.     cv2.putText(image, "Object #{}".format(i + 1),
98.         (int(rect[0][0] - 15), int(rect[0][1] - 15)),
99.         cv2.FONT_HERSHEY_SIMPLEX, 0.55, (255, 255, 255), 2)
100.
101.    # show the image
102.    cv2.imshow("Image", image)
103.    cv2.waitKey(0)
```

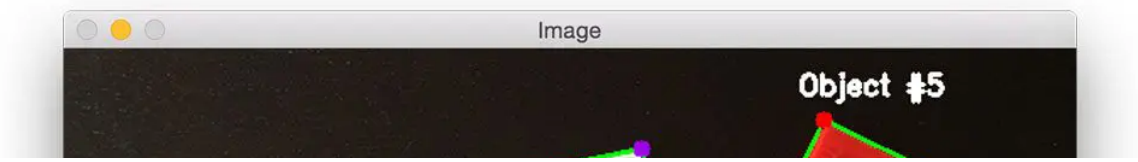
We start looping over our (hopefully) ordered coordinates on **Line 93** and draw them on our `image`.

According to the `colors` list, the top-left point should be *red*, the top-right point *purple*, the bottom-right point *blue*, and finally, the bottom-left point *teal*.

Lastly, **Lines 97-103** draw the object number on our `image` and display the output result.

To execute our script using the *original, flawed* implementation, just issue the following command:

```
Ordering coordinates clockwise with Python and OpenCV
1.  $ python order_coordinates.py
```



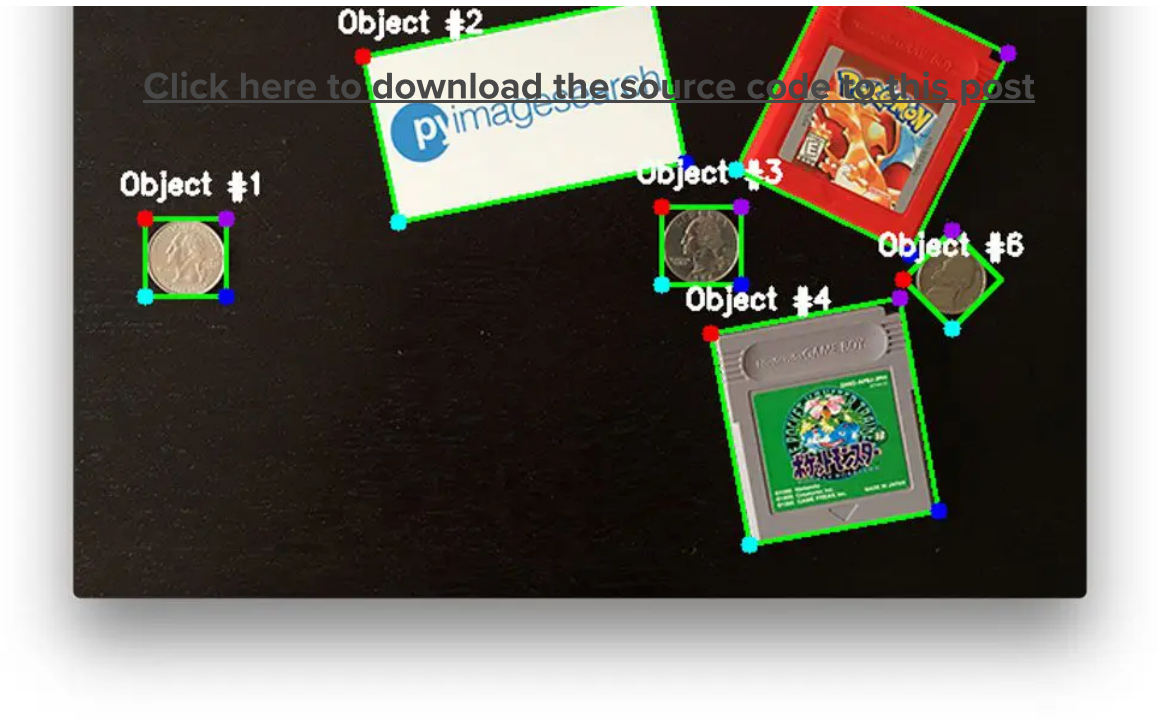


Figure 2: Arranging our rotated bounding box coordinates in top-left, top-right, bottom-right, and bottom-left order...**but with a *major flaw* (take a look at Object #6).**

As we can see, our output is anticipated with the points ordered clockwise in a top-left, top-right, bottom-right, and bottom-left arrangement — ***except for Object #6!***

Note: *Take a look at the output circles — notice how there isn't a blue one?*

Looking at our terminal output for Object #6, we can see why:

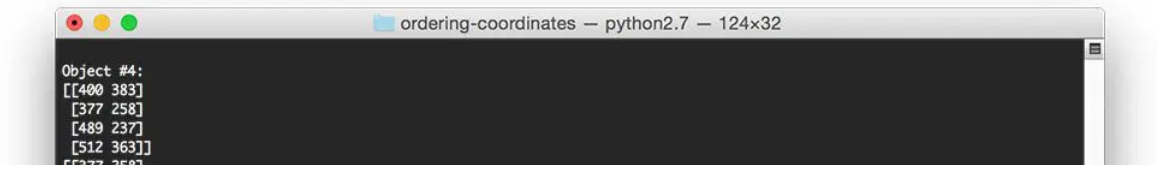


Figure 3: Take a look at the bounding box coordinates for Object #6. And then see what happens what happens when we take their sum and differences.

Taking the sum of these coordinates we end up with:

- 520 + 255 = 775
- 491 + 226 = 717
- 520 + 197 = 717
- 549 + 226 = 775

While the difference gives us:

- 520 – 255 = 265
- 491 – 226 = 265
- 520 – 197 = 323
- 549 – 226 = 323

As you can see, ***we end up with duplicate values!***

And since there are duplicate values, the `argmin()` and `argmax()` functions don't work as we expect them to, giving us an incorrect set of "ordered" coordinates.

To resolve this issue, we can use our updated `order_points` function in the **imutils package**

(https://github.com/jrosebr1/imutils). We can verify that our updated function is working properly by issuing the following command:

Click here to download the source code to this post

```
Ordering coordinates clockwise with Python and OpenCV
$ python order_coordinates.py --new 1
```

This time, all of our points are ordered correctly, including Object #6:

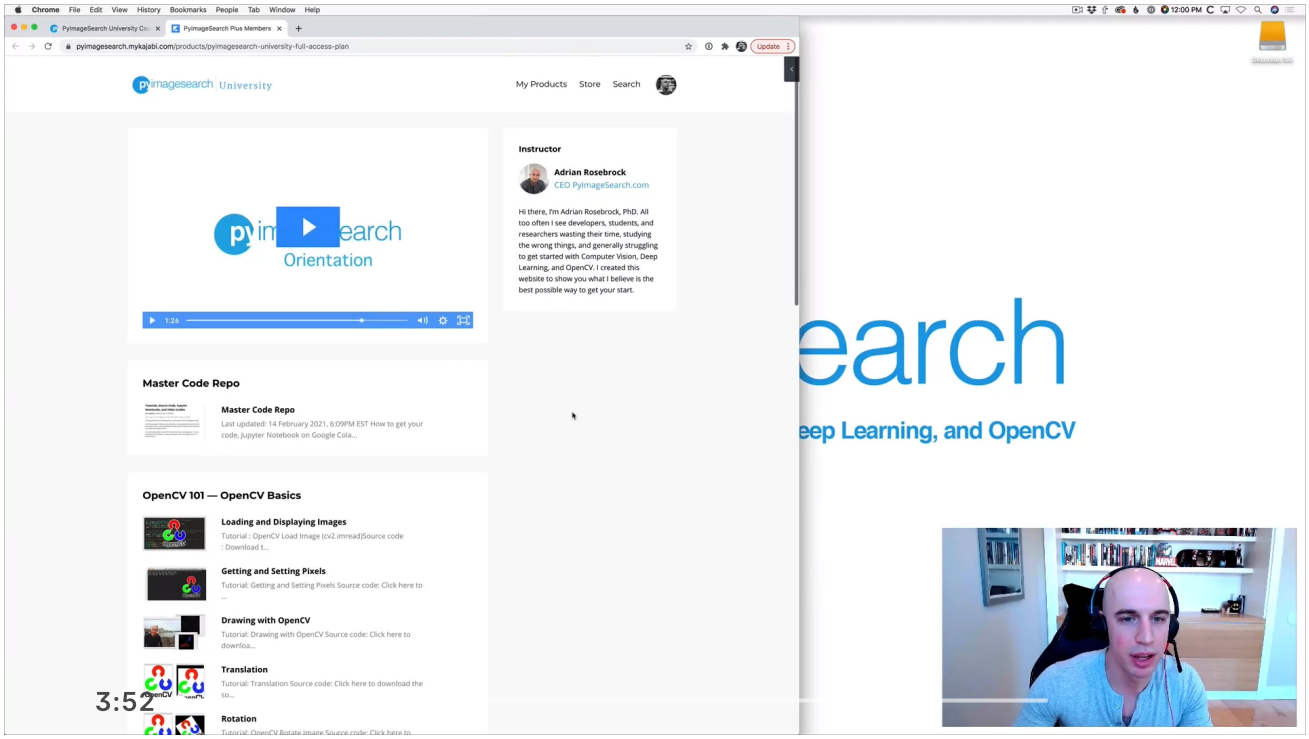
Figure 4: Correctly ordering coordinates clockwise with Python and OpenCV.

When utilizing **perspective transforms (https://pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/)** (or any other project that requires ordered coordinates), *make sure you use our updated implementation!*

What's next? I recommend PyImageSearch University (https://pyimagesearch.com/pyimagesearch-university/?utm_source=blogPost&utm_medium=bottomBanner&utm_campaign=What%27s%20next%3F%20I%20recom

mend).

Click here to download the source code to this post



Course information:
35+ total classes • 39h 44m video • Last updated: April 2022
★★★★★ 4.84 (128 Ratings) • 13,800+ Students Enrolled

I strongly believe that if you had the right teacher you could *master* computer vision and deep learning.

Do you think learning computer vision and deep learning has to be time-consuming, overwhelming, and complicated? Or has to involve complex mathematics and equations? Or requires a degree in computer science?

That's *not* the case.

All you need to master computer vision and deep learning is for someone to explain things to you in *simple, intuitive* terms. *And that's exactly what I do*. My mission is to change education and how complex Artificial Intelligence topics are taught.

If you're serious about learning computer vision, your next stop should be PyImageSearch University, the most comprehensive computer vision, deep learning, and OpenCV course online today. Here you'll learn how to *successfully* and *confidently* apply computer vision to your work, research, and projects. Join me in computer vision mastery.

computer vision mastery.

Inside PyImageSearch University you'll find:

Click here to download the source code to this post

✓ **35+ courses** on essential computer vision, deep learning, and OpenCV topics

✓ **35+ Certificates** of Completion

✓ **39+ hours** of on-demand video

✓ **Brand new courses released *regularly***, ensuring you can keep up with state-of-the-art techniques

✓ **Pre-configured Jupyter Notebooks in Google Colab**

✓ Run all code examples in your web browser — works on Windows, macOS, and Linux (no dev environment configuration required!)

✓ Access to **centralized code repos for *all* 450+ tutorials** on PyImageSearch

✓ **Easy one-click downloads** for code, datasets, pre-trained models, etc.

✓ **Access** on mobile, laptop, desktop, etc.

CLICK HERE TO JOIN PYIMAGESEARCH UNIVERSITY (HTTPS://PYIMAGESEARCH.COM/PYIMAGESEARCH-UNIVERSITY/?UTM_SOURCE=BLOGPOST&UTM_MEDIUM=BOTTOMBANNER&UTM_CAMPAIGN=WHAT%27S%20NEXT%3F%20I%20RECOMMEND)

Summary

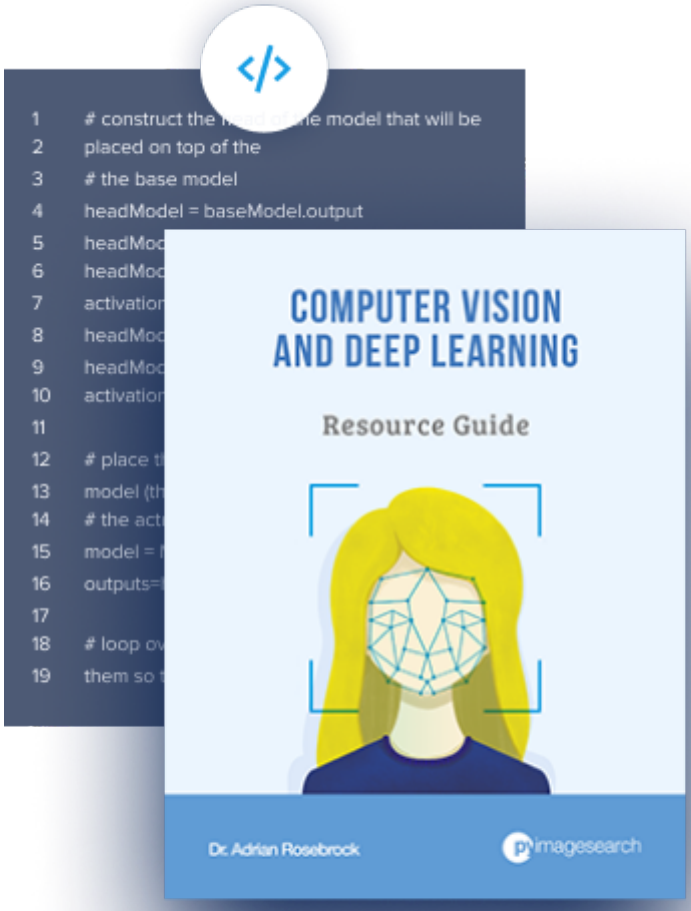
In this blog post, we started a three part series on *calculating the size of objects in images* and *measuring the distance between objects*. To accomplish these goals, we'll need to order the 4 points associated with the rotated bounding box of each object.

We've already implemented such a function in a **previous blog post (https://pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/)**; however, as we discovered, this implementation has a fatal flaw — it can return the *wrong coordinates* under *very specific* situations.

To resolve this problem, we defined a new, updated `order_points` function and placed it in the **imutils package (https://github.com/jrosebr1/imutils)**. This implementation ensures that our points are always ordered correctly.

Now that we can order our *(x, y)*-coordinates in a reliable manner, we can move on to *measuring the size of objects in an image*, which is exactly what I'll be discussing in our next blog post.

Be sure to signup for the PyImageSearch Newsletter by entering your email address in the form below — *you won't want to miss this series of posts!*



Download the Source Code and FREE 17-page Resource Guide

Enter your email address below to get a .zip of the code and a **FREE 17-page Resource**

Guide on Computer Vision, OpenCV, and Deep Learning. Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL!

Click here to download the source code to this post

About the Author

Hi there, I'm Adrian Rosebrock, PhD. All too often I see developers, students, and researchers wasting their time, studying the wrong things, and generally struggling to get started with Computer Vision, Deep Learning, and OpenCV. I created this website to show you what I believe is the best possible way to get your start.

Previous Article:

PyImageSearch Gurus member spotlight: Tuomo Hiippala

(https://pyimagesearch.com/2016/03/14/pyimagesearch-gurus-member-spotlight-tuomo-hiippala/)

Next Article:

Measuring size of objects in an image with OpenCV

(https://pyimagesearch.com/2016/03/28/measuring-size-of-objects-in-an-image-with-opencv/)

54 responses to: Ordering coordinates clockwise with Python and OpenCV

Click here to download the source code to this post

David Hoffman (http://www.twitter.com/drhoffma)

March 21, 2016 at 1:55 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-392497)

This is a great solution to the problem. I have run into this problem before and never was able to devise a reliable solution.

Adrian Rosebrock

March 21, 2016 at 6:31 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-392524)

I'm glad the blog post helped, David! 🙂

Mahed

March 21, 2016 at 2:33 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-392501)

Oh many thanks Adiran !! I knew you would help me out
I will try this and see if my results are better this time

Many Thanks again Adrian !!
Mahed

Adrian Rosebrock

March 21, 2016 at 6:31 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-392523)

No problem Mahed!

Click here to download the source code to this post

Neville

March 21, 2016 at 9:39 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-392535)

Thanks for this post Adrian, but I'm a little confused about one aspect of the new algorithm.

In the new order_points function, lines 18 & 19 sort the left most coordinates in order of the y-value, which gives us the top-left & bottom-left points. That makes perfect sense to me.

My question is, why was it not just as simple for the right side points?
Rather than the Euclidean distance calculation forming the basis of the sort in lines 26 & 27, why could the right most points not have also just been sorted by their y-value to determine the top-right & bottom-right points?

I expect there is a scenario where that wont work (which is the reason for the more complicated solution), but I just cant think of what that scenario would be.

Adrian Rosebrock

March 22, 2016 at 4:40 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-392618)

Yes, you are correct. However, after the previous bug from what looked like an innocent heuristic approach, I decided to go with the Euclidean distance, that way I always knew the bottom-right corner of the rectangle would be based on the properties of triangles (and therefore the correct corner chosen), rather than running into another scenario where the `order_points` function broke (and being left again to figure which heuristic broke it). Consider this more "preemptive strike" against any bugs that could arise.

Jacky

March 27, 2018 at 6:45 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-454372)

I think the y-coordinates order method is more stable. The Euclidean distance method will get wrong order when the object is a trapezoid. For examle, they points are (10,10),(30,10),(20,20),(10,20). The output is incorret when using Eudclidean distance method, but correct when using y-coordinates order method.

Output:

Click here to download the source code to this post

ordered by Euclidean distance
[[10 10]
[20 20]
[30 10]
[10 20]]
orderd by y-coordinates
[[10 10]
[30 10]
[20 20]
[10 20]]

For the test code, please go to gist:
https://gist.github.com/flashlib/e8261539915426866ae910d55a3f9959
(https://gist.github.com/flashlib/e8261539915426866ae910d55a3f9959)

Arif

March 23, 2016 at 10:35 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-392694)

Fantastic as always 🙂

Adrian Rosebrock

March 24, 2016 at 5:18 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-392810)

Thanks Arif! 🙂

Mahed

March 26, 2016 at 10:57 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-392964)

Hi Adrian ,et. everyone ,

I used the code above as video feed on rasPi inorder to zoom in the image containing a ‘white’ text embedded on a ‘red’ square, mounted on a drone as part of a project.

Click here to download the source code to this post

The algorithm works perfectly and the zoomed image appears perfectly upright text even when image is slightly skewed ….. However when the square was turned completely on the l.h.s or r.h.s …. The text appeared sideways too ….

The same situation was when the text was facing bottom

Unfortunately my text recognition software (pytesseract) couldnt read the text sideways/bottomways
There are other recog. engines that can deal with this but are not free

Is there a way i could modify the code so that my embedded text always upright.
I did give myself a thinking but could’nt go that far becoz i thought that for the case if the image is completely sideways … i might say that the distance between top-left and top right corner is less than what was before and hence rotate by 90` but the thing is the case wont work for both situations … i.e. completley l.h.s and r.h.s and i am absolutely clueless on how to solve when the text is facing bottom

---

Mahed

March 26, 2016 at 11:00 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-392965)

Oh crisis !! I just realized my first solution method wont work as well because my target is a red square not a rectangle ….. *facepalms*

jack

April 4, 2016 at 12:06 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-393950)

is it possible to distinguish between real objects and floor lines ? I am trying to detect objects on a floor that has lines all over and I don’t know how to separate the real objects from rectangles/squares on the floor.

Adrian Rosebrock

Click here to download the source code to this post

April 6, 2016 at 9:18 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-394143)

In most cases, yes, this should be possible. Using the Canny edge detector, you can determine lines that run most of the width/height of the image. Furthermore, these lines should also be equally spaced and in many cases intersecting. You can use this information to help prune out the floro lines that you are not interested in.

leena

April 5, 2016 at 12:21 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-393990)

Useful post as always.

Chris

September 13, 2016 at 9:09 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-406081)

If I am reading this right, another special case that may need to be accounted for is skewed four-sided objects where the order of the x values may not reliably give you lefts and rights. Take for example an object with the points {(0,0), (2,0), (3,4), (5,4)}. The top right x is smaller than the bottom left x and the sort by x's will result in top left and top right being identified as top left and bottom left respectively.

solarflare

January 4, 2017 at 9:14 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-415108)

Hi Adrian,

Question on the circular items in the picture. Why is it that the bounding rectangle is upright (that is, not angled) for the two quarters, but is angled for the nickle?

Click here to download the source code to this post

Adrian Rosebrock

January 4, 2017 at 10:38 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-415111)

It's simply due to how the left-most and right-most coordinates are sorted. You can also see results like these if there is noise due to shadowing, lighting spaces, etc.

solarflare

January 6, 2017 at 10:32 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-415250)

It almost seemed that the bounding rectangle detected the rotation angle of the coin (relative to the the head and neck of the President being upright). Just wanted to make sure this was coincidence and not a desired feature of the algorithm.

Adrian Rosebrock

January 7, 2017 at 9:30 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-415328)

Yep, that's a total coincidence.

David Killen

January 24, 2017 at 6:26 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-416356)

You write

```
# now that we have the top-left coordinate, use it as an
# anchor to calculate the Euclidean distance between the
# top-left and right-most points; ...
# ... the point with the largest distance will be
```

our bottom-right point

Click here to download the source code to this post

This may be true for images of quadrilaterals but it's not generally true. Imagine a square oriented with its edges horizontal and vertical and now deform it by sliding the right-hand vertical image straight upwards so that we get a series of parallelograms. At some point it consists of two equilateral triangles stuck together and now the two right-hand points are equidistant from the top-left corner. From now on, the upper-right corner is further from the anchor than is the lower-right corner.

I discovered this the hard way while trying to find the grid-lines on a go board. There were a lot of false lines from diagonals and they created some very skewed parallelograms.

David Killen

January 24, 2017 at 10:07 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-416369)

I think I should have written 'images of rectangles' instead of 'images of quadrilaterals'

David Killen

January 24, 2017 at 10:13 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-416370)

Addendum

I'm now using code that finds the top-left and bottom-left points by your method but then calculates the angle bl->tl->r for r in the rightMost points and assigns tr and br accordingly. It seems to work.

Adrian Rosebrock

January 24, 2017 at 2:19 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-416386)

Thank you for sharing your insights David, I appreciate it.

Varsha

April 27, 2017 at 6:01 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-

April 27, 2017 at 8:01 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-423897)

Click here to download the source code to this post

This approach is not working for Video Frame object, as object from first frame appearing in second frame , its counting as second object. kindly help...

Adrian Rosebrock

April 28, 2017 at 9:31 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-423993)

Hi Varsha — I'm not sure what you mean by "counting as second object". Can you please elaborate?

Diyanpure

July 22, 2017 at 3:55 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-430510)

I need to solve my programs like this but i need in C++ program.. can you give me a suggestion ? Please.. and thanks..

Adrian Rosebrock

July 24, 2017 at 3:44 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-430703)

Hello — I only provide Python on this blog. Best of luck with the project!

Umesh

August 26, 2017 at 9:39 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-433407)

Your article is very good but..
I can't understand what line number 86, perspective. order_points(pts) do..
In perspective manner..
Please guide me..

Thanks for article.

Click here to download the source code to this post

Adrian Rosebrock

August 27, 2017 at 10:35 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-433489)

Hi Umesh — I'm not sure what your question is. What specifically are you trying to understand with the `order_points` function?

Hai

November 6, 2017 at 11:33 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-439859)

Hi, Adiran!
The blogpost is very useful for me, many thanks for sharing!
There is a little problem, I am processing a short video with this method, when there exists objects in video, it perfectly draws rectangles on objects, but when comes to blank (simple black color) part of the video, it gives me an error like:
in line : (cnts, _) = contours.sort_contours(cnts)
not enough values to unpack (expected 2, got 0)

The error seems like becouse of cannot find any contours in blank part of the video, how to i solve this? ... I am a beginner on CV, thank you again!

Adrian Rosebrock

November 9, 2017 at 7:10 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-440102)

I would need to see the full traceback of the error to determine the exact issue; however, I would suggest checking `if len(cnts) == 0`. If this is the case you can skip the frame since no contours can be found. Since you're new to OpenCV and image processing I would

definitely recommend working through **Practical Python and OpenCV (https://pyimagesearch.com/practical-python-opencv/)** where I discuss the fundamentals of OpenCV in detail. By the time you work through the book you'll be able to work through the majority of tutorials here on PyImageSearch with ease.

Click here to download the source code to this post

Waqar Ahmed (http://-)

November 26, 2017 at 7:52 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-441668)

i got an assignment to measure the length and height of the rice grain and i am new to this language and can't understand anything but i am reading this site and it is really helpful but can you tell me how to do that?

Adrian Rosebrock

November 27, 2017 at 1:08 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-441782)

Hey Waqar — take a look at **this blog post (https://pyimagesearch.com/2016/03/28/measuring-size-of-objects-in-an-image-with-opencv/)** where I share more information on measuring object sizes in images.

nwpuxhld

April 11, 2018 at 9:47 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-456258)

I find it doesn't work in some case. The target is not Rect, it is Quadrilateral. For example, the four points are:[[96 263] [98 380] [100 382] [97 263]]. Do you know any solution for this case? Thanks.

Adrian Rosebrock

April 13, 2018 at 6:52 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-456470)

Take a look at my **updated blog post (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/)**.

wellington castro (https://capivararex.wordpress.com/)

April 15, 2018 at 11:25 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-456765)

Adrian thanks for this solution, really! But I was left with a question: Why can't you reorder the rightmost points in respect to y-axis to get TR and BR just as you did for TL and BL ?

Adrian Rosebrock

April 16, 2018 at 2:26 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-456964)

It creates a few edge cases, unfortunately. Refer to the comments section of the **previous post (https://pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/)**.

Artemii

May 9, 2018 at 1:51 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-460585)

My appreciation for such a detailed explanation, a great introduction to the CV.

Code works well while detecting a bounders, but where is an issue with a counting of objects for some reason.. Some of the digits get skipped, like #3, #5, #6 and #7 instead of 1, 2, 3, 4.

Any ideas why it could happed?

Adrian Rosebrock

May 9, 2018 at 9:30 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-460645)

We're not performing digit recognition in this tutorial so I'm not sure what you are referring to the digits being skipped. Could you please clarify?

Francesco Vergentini

June 7, 2018 at 1:05 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-467362)

Thank you for your crystal clear solutions.
I am using your new method for a application where my box is narrow and quite long. When this is rotated approx 45° clockwise happen that the most-left points are the two bottom ones: the result of new algorithm will give as tl point the bl one.

For this special condition the old algorithm works better but fails in the cases you already mentioned.

I cant figure out how could we fix it in order to be always reliable.

Any suggestion would be very much appreciated.

Hola

November 3, 2018 at 7:08 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-485180)

Hi,

I have some objects on a wooden textured background, took a photo of it and tried out this algorithm, it breaks at finding the contour. The contour includes the background texture as well in it, making tiny boxes, I tried around tweaking the contourArea size, but some of the objects have similar size of that some of the texture.

How can i completely remove the textures? any ideas?

Adrian Rosebrock

November 6, 2018 at 1:32 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-485790)

Hey Hola — do you have any example images of what you're working with? That would likely help me provide suggestions

help me provide suggestions.

Click here to download the source code to this post

Shershon

February 15, 2019 at 4:07 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-501507)

Hey Adrian! When I try to execute the program the following error occurs.

ipykernel_launcher.py: error: the following arguments are required: -i/–image, -w/–width

I am using jupyter notebook.

Adrian Rosebrock

February 15, 2019 at 6:12 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-501524)

First make sure you read **this tutorial on command line arguments (https://pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/)** so you can get up to speed on what they are and how they work.

From there you have two options:

1. Execute the script via command line instead of Jupyter notebook.
2. Update the “argparse” code per the recommendations in the argparse tutorial I linked you to.

rosangela

March 31, 2019 at 9:04 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-510238)

HI Adrian, how can a measure the objects(height and widht) in an image knowing the distance of picamera to the object????

Adrian Rosebrock

April 2, 2016 at 3:59 pm (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-510592)

Click here to download the source code to this post

Make sure you follow **this tutorial. (https://pyimagesearch.com/2016/03/28/measuring-size-of-objects-in-an-image-with-opencv/)** Once you have the triangle similarity you can compute the width and height.

Fawzi Sdudah

May 15, 2019 at 10:53 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-517731)

I'm not sure the euclidean method, as used, is fool-proof.

After the anchor point, let's denote the two ordered, right-most points a and b. a will always be closer to the anchor than b as they have been ordered (a_x is smaller than b_x).

For a fixed value of x_y, there are two cases for y_b:

1- when y_b is smaller, (point a) will be both below point b and will be the bottom right-most even though it does not make the longest line from the anchor. Point b will make the longest line from the anchor.

2- when y_b is larger, (point b) will be below a, at the bottom, and will make the longest line from the anchor.

Fawzi Sdudah

Shruti

November 6, 2019 at 1:00 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-570017)

which IDE you are using ?

Adrian Rosebrock

November 7, 2019 at 10:15 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-570729)

Click here to download the source code to this post

I prefer Sublime Text 2 or PyCharm for most projects.

Mahmoud

December 6, 2019 at 10:46 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-590036)

Dear Adrian,
Thanks for your effort.
This blog is working well with me.
The problem is when I use my photo . it draw coordinates in each cell of photo without the real object that I need.
I do not know this is due to resolution of image or what.
I need to know the coordinates or the object only without background.

Thanks for your support

Amine

January 4, 2020 at 10:20 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-626803)

Hello;
Hope you are fine, i need your help, i used your but it doesn't generate ordered pixels i mean in the same piece of countour pixels are not ordred due to the function findcontour. your function return rect not ordered pixels if i'm not wrong.

Thank you very much

Trackbacks

Measuring size of objects in an image with OpenCV - PyImageSearch (https://pyimagesearch.com/2016/03/28/measuring-size-of-objects-in-an-image-with-opencv/) says:

opencv/)

March 28, 2016 at 10:00 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-393138)

[...] Last week, we learned an important technique: how reliably order a set of rotated bounding box coordinates in a top-left, top-right, bottom-right, and bottom-left arrangement. [...]

Measuring distance between objects in an image with OpenCV - PyImageSearch (https://pyimagesearch.com/2016/04/04/measuring-distance-between-objects-in-an-image-with-opencv/) says:

April 4, 2016 at 10:00 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-393942)

[...] weeks ago, we started this round of tutorials by learning how to (correctly) order coordinates in a clockwise manner using Python and OpenCV. Then, last week, we discussed how to measure the size of objects in an image using a reference [...]

Finding extreme points in contours with OpenCV - PyImageSearch (https://pyimagesearch.com/2016/04/11/finding-extreme-points-in-contours-with-opencv/) says:

April 11, 2016 at 10:00 am (https://pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/#comment-394668)

[...] few weeks ago, I demonstrated how to order the (x, y)-coordinates of a rotated bounding box in a clockwise fashion — an extremely useful skill that is critical in many computer vision applications, including [...]

Comment section

Hey, Adrian Rosebrock here, author and creator of PyImageSearch. While I love hearing from readers, a couple years ago I made the tough decision to no longer offer 1:1 help over blog post comments.

At the time I was receiving 200+ emails per day and another 100+ blog post comments. I simply did not have the time to moderate and respond to them all, and the sheer volume of requests was taking a toll on me.

Instead, my goal is to *do the most good* for the computer vision, deep learning, and OpenCV community at large by focusing my time on authoring high-quality blog posts, tutorials, and books/courses.

If you need help learning computer vision and deep learning, I suggest you refer to my full catalog of books and courses (https://pyimagesearch.com/books-and-courses/) — they have helped tens of thousands of developers, students, and researchers *just like yourself* learn Computer Vision, Deep Learning, and OpenCV.

Click here to browse my full catalog. (https://pyimagesearch.com/books-and-courses/)

Click here to download the source code to this post

Similar articles

OPENCV TUTORIALS TUTORIALS

Splitting and Merging Channels with OpenCV

January 23, 2021

(https://pyimagesearch.com/2021/01/23/splitting-and-merging-channels-with-opencv/) →

AMAZON REKOGNITION API AMAZON WEB SERVICES (AWS)

OPTICAL CHARACTER RECOGNITION (OCR) PYTHON TUTORIALS

Text Detection and OCR with Amazon Rekognition API

March 21, 2022

(https://pyimagesearch.com/2022/03/21/text-detection-and-ocr-with-amazon-rekognition-api/)

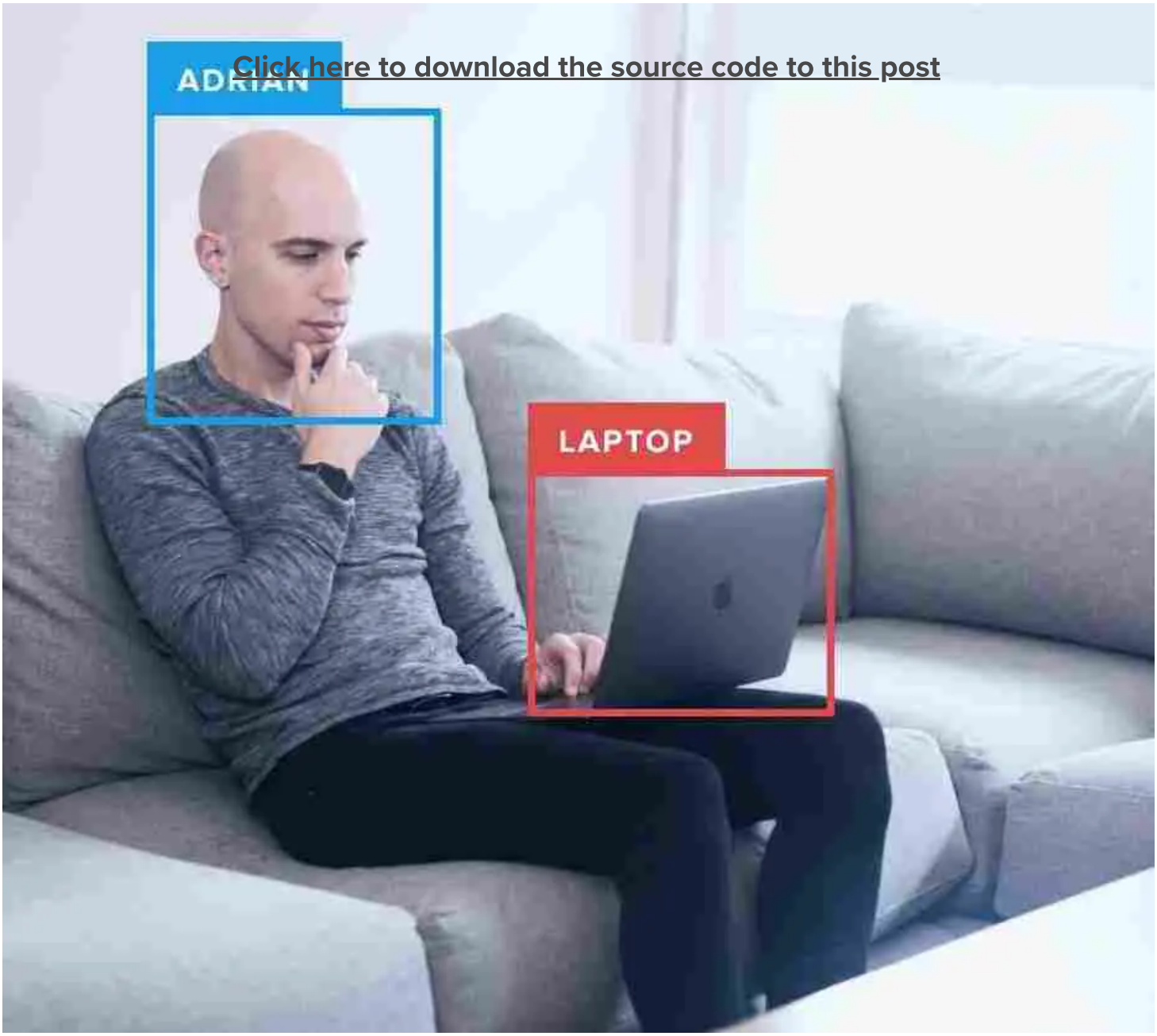
Click here to download the source code to this post

KICKSTARTER

It's time. The Deep Learning for Computer Vision with Python Kickstarter is LIVE.

January 18, 2017

(https://pyimagesearch.com/2017/01/18/its-time-the-deep-learning-for-computer-vision-with-python-kickstarter-is-live/)



You can learn Computer Vision, Deep Learning, and OpenCV.

Get your FREE 17 page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF. Inside you'll find our hand-picked tutorials, books, courses, and libraries to help you master CV and DL.

Topics

Machine Learning and Computer Vision (https://pyimagesearch.com/category/machine-

Books & Courses

PyImageSearch

computer-vision/)

Click here to download the source code to this post