

PyImageSearch Gurus Course

[🏠 \(HTTPS://GURUS.PYIMAGESEARCH.COM\)](https://gurus.pyimagesearch.com/) >

1.11.5: Sorting contours

Topic Progress: (<https://gurus.pyimagesearch.com/topic/finding-and-drawing-contours/>)

(<https://gurus.pyimagesearch.com/topic/simple-contour-properties/>) (<https://gurus.pyimagesearch.com/topic/advanced-contour-properties/>) (<https://gurus.pyimagesearch.com/topic/contour-approximation/>)

(<https://gurus.pyimagesearch.com/topic/sorting-contours/>)

[← Back to Lesson \(https://gurus.pyimagesearch.com/lessons/contours/\)](https://gurus.pyimagesearch.com/lessons/contours/)

Feedback

We've discussed *many* topics related to contours into this module — but let's do just one more. I find the topic of contour sorting extremely useful and applicable to a variety of image processing projects. And perhaps surprisingly, it's not as difficult as you think it would be to accomplish.

For instance, when we build a license plate recognition system in **Module 6** we'll need to find the contours of each character on the license plate (obviously). But what's less obvious is how we **sort** the contours of the characters from left-to-right so we can report them back to the end-user correctly.

Oddly enough OpenCV *does not* provide a built-in function or method to perform the actual sorting of contours.

But no worries.

In the rest of this lesson you'll be sorting contours using Python and OpenCV like a pro.

Objectives:

By the end of this lesson you'll be able to:

1. Sort contours according to their *size/area*, along with a template to follow to sort contours by any other arbitrary criteria.
2. Sort contoured regions from *left-to-right*, *right-to-left*, *top-to-bottom*, and *bottom-to-top* using only a **single function**.

Sorting Contours

So let's go ahead and get started. Open up your favorite code editor, name it `sort_contours.py` and let's get started:

sort_contours.py	Python
<pre>1 # import the necessary packages 2 import numpy as np 3 import argparse 4 import cv2 5 import imutils 6 7 def sort_contours(cnts, method="left-to-right"): 8 # initialize the reverse flag and sort index 9 reverse = False 10 i = 0 11 12 # handle if we need to sort in reverse 13 if method == "right-to-left" or method == "bottom-to-top": 14 reverse = True 15 16 # handle if we are sorting against the y-coordinate rather than 17 # the x-coordinate of the bounding box 18 if method == "top-to-bottom" or method == "bottom-to-top": 19 i = 1 20 21 # construct the list of bounding boxes and sort them from top to 22 # bottom 23 boundingBoxes = [cv2.boundingRect(c) for c in cnts] 24 (cnts, boundingBoxes) = zip(*sorted(zip(cnts, boundingBoxes), 25 key=lambda b:b[1][i], reverse=reverse)) 26 27 # return the list of sorted contours and bounding boxes 28 return (cnts, boundingBoxes)</pre>	

Feedback

We'll start off by importing our necessary packages: NumPy for numerical processing, `argparse` to parse our command line arguments, and `cv2` for our OpenCV bindings.

Instead of starting off by parsing arguments, loading images, and taking care of other normal procedures, let's skip these steps for the time being and jump immediately in to defining our `sort_contours` function which will enable us to sort our contours.

The actual `sort_contours` function is defined on **Line 7** and takes two arguments. The first is `cnts`, the list of contours that we want to sort, and the second is the sorting `method`, which indicates the *direction* in which we are going to sort our contours (i.e. left-to-right, top-to-bottom, etc.).

From there we'll initialize two important variables on **Lines 9 and 10**. These variables simply indicate the sorting order (ascending or descending) and the *index* of the bounding box we are going to use to perform the sort (more on that later). We'll initialize these variables to sort in *ascending* order and along to the x-axis location of the bounding box of the contour.

If we are sorting right-to-left or bottom-to-top, we'll need to sort in **descending** order, according to the location of the contour in the image (**Lines 13 and 14**).

Similarly, on **Lines 18 and 19** we check to see if we are sorting from top-to-bottom or bottom-to-top. If this is the case, then we need to sort according to the y-axis value rather than the x-axis (since we are now sorting vertically rather than horizontally).

The actual sorting of the contours happens on **Lines 23-25**.

We first compute the *bounding boxes* of each contour, which is simply the starting (x, y)-coordinates of the bounding box followed by the width and height (hence the term "bounding box"). (**Line 23**)

The `boundingBoxes` enable us to sort the actual contours, which we do on **Line 24 and 25** using some Python magic that sorts two lists together. Using this code we are able to sort **both** the contours and bounding boxes according to the criteria that we provided.

Finally, we return the (now sorted) list of bounding boxes and contours to the calling function on **Line 28**.

While we're at it, let's go ahead and define another helper function, `draw_contour` :

sort_contours.py	Python
<pre>30 def draw_contour(image, c, i): 31 # compute the center of the contour area and draw a circle 32 # representing the center 33 M = cv2.moments(c) 34 cX = int(M["m10"] / M["m00"]) 35 cY = int(M["m01"] / M["m00"]) 36 37 # draw the countour number on the image 38 cv2.putText(image, "#{}".format(i + 1), (cX - 20, cY), cv2.FONT_HERSHEY_SIMPLEX, 39 1.0, (255, 255, 255), 2) 40 41 # return the image with the contour number drawn on it 42 return image</pre>	

This function simply computes the **center** (x, y)-coordinate of the supplied contour `c` on **Lines 33-35** and then uses the center coordinates to draw the contour ID, `i` , on **Lines 38 and 39**.

Finally, the passed in `image` is returned to the calling function on **Line 42**.

Again, this is simply a helper function that we'll leverage to draw contour ID numbers on our actual image so we can visualize the results of our work.

Now that the helper functions are done, let's put the driver code in place to take our actual image, detect contours, and sort them:

sort_contours.py	Python
<pre>44 # construct the argument parser and parse the arguments 45 ap = argparse.ArgumentParser() 46 ap.add_argument("-i", "--image", required=True, help="Path to the input image") 47 ap.add_argument("-m", "--method", required=True, help="Sorting method") 48 args = vars(ap.parse_args()) 49 50 # load the image and initialize the accumulated edge image 51 image = cv2.imread(args["image"]) 52 accumEdged = np.zeros(image.shape[:2], dtype="uint8") 53 54 # loop over the blue, green, and red channels, respectively 55 for chan in cv2.split(image): 56 # blur the channel, extract edges from it, and accumulate the set 57 # of edges for the image 58 chan = cv2.medianBlur(chan, 11) 59 edged = cv2.Canny(chan, 50, 200) 60 accumEdged = cv2.bitwise_or(accumEdged, edged) 61 62 # show the accumulated edge map 63 cv2.imshow("Edge Map", accumEdged)</pre>	

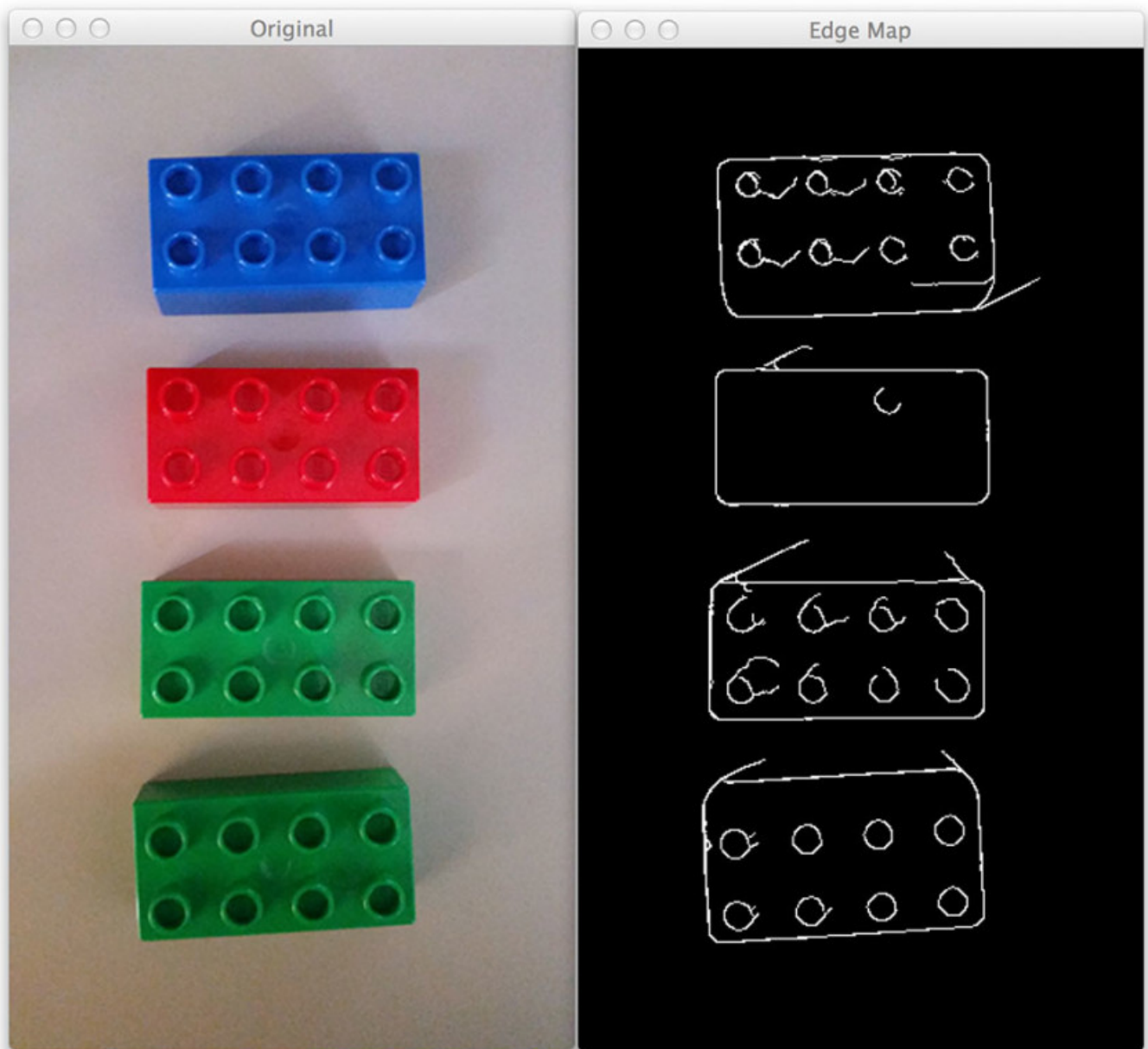
Feedback

Lines 45-48 aren't very interesting — they simply parse our command line arguments, `--image` which is the path to where our image resides on disk, and `--method` which is a text representation of the direction in which we want to sort our contours.

From there we load our image off disk on **Line 51** and allocate memory for the edge map on **Line 52**.

Constructing the actual edge map happens on **Lines 55-60**, where we loop over each Blue, Green, and Red channel of the image (**Line 55**), blur each channel slightly to remove high frequency noise (**Line 58**), perform edge detection (**Line 59**), and update the accumulated edge map on **Line 60**.

We display the accumulated edge map on **Line 63** which looks like this:



https://gurus.pyimagesearch.com/wp-content/uploads/2015/04/sorted_contours_edge_map.jpg

FIGURE 1: (LEFT) OUR ORIGINAL IMAGE. (RIGHT) THE EDGE MAP OF THE LEGO BRICKS.

As you can see, we have detected the actual edge outlines of the Lego bricks in the image.

Now, let's see if we can (1) find the contours of these Lego bricks, and then (2) sort them:

`sort_contours.py`

Python

```

65 # find contours in the accumulated image, keeping only the largest
66 # ones
67 cnts= cv2.findContours(accumEdged.copy(), cv2.RETR_EXTERNAL,
68     cv2.CHAIN_APPROX_SIMPLE)
69 cnts = imutils.grab_contours(cnts)
70 cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:5]
71 orig = image.copy()
72
73 # loop over the (unsorted) contours and draw them
74 for (i, c) in enumerate(cnts):
75     orig = draw_contour(orig, c, i)
76
77 # show the original, unsorted contour image
78 cv2.imshow("Unsorted", orig)
79
80 # sort the contours according to the provided method
81 (cnts, boundingBoxes) = sort_contours(cnts, method=args["method"])
82
83 # loop over the (now sorted) contours and draw them
84 for (i, c) in enumerate(cnts):
85     draw_contour(image, c, i)
86
87 # show the output image
88 cv2.imshow("Sorted", image)
89 cv2.waitKey(0)

```

Quite obviously, the first step here is to find the actual contours in our accumulated edge map image on **Line 67 and 68**. We are looking for the *external* contours of the Lego bricks, which simply corresponds to their *outlines*.

Based on these contours, we are now going to **sort them according to their size** by using a combination of the Python `sorted` function and the `cv2.contourArea` method — this allows us to sort our contours according to their area (i.e. size) from largest to smallest.

We take these sorted contours (in terms of size, *not* location), loop over them on **Line 74** and draw each individual contour on **Line 75** using our `draw_contour` helper function.

This image is then displayed to our screen on **Line 78**. However, you'll notice that these contours are *unsorted*.

We address this problem on **Line 81** where we make a call to our custom `sort_contours` function. This method accepts our list of contours along with sorting direction method (provided via command line argument) and sorts them, returning a tuple of sorted bounding boxes and contours, respectively.

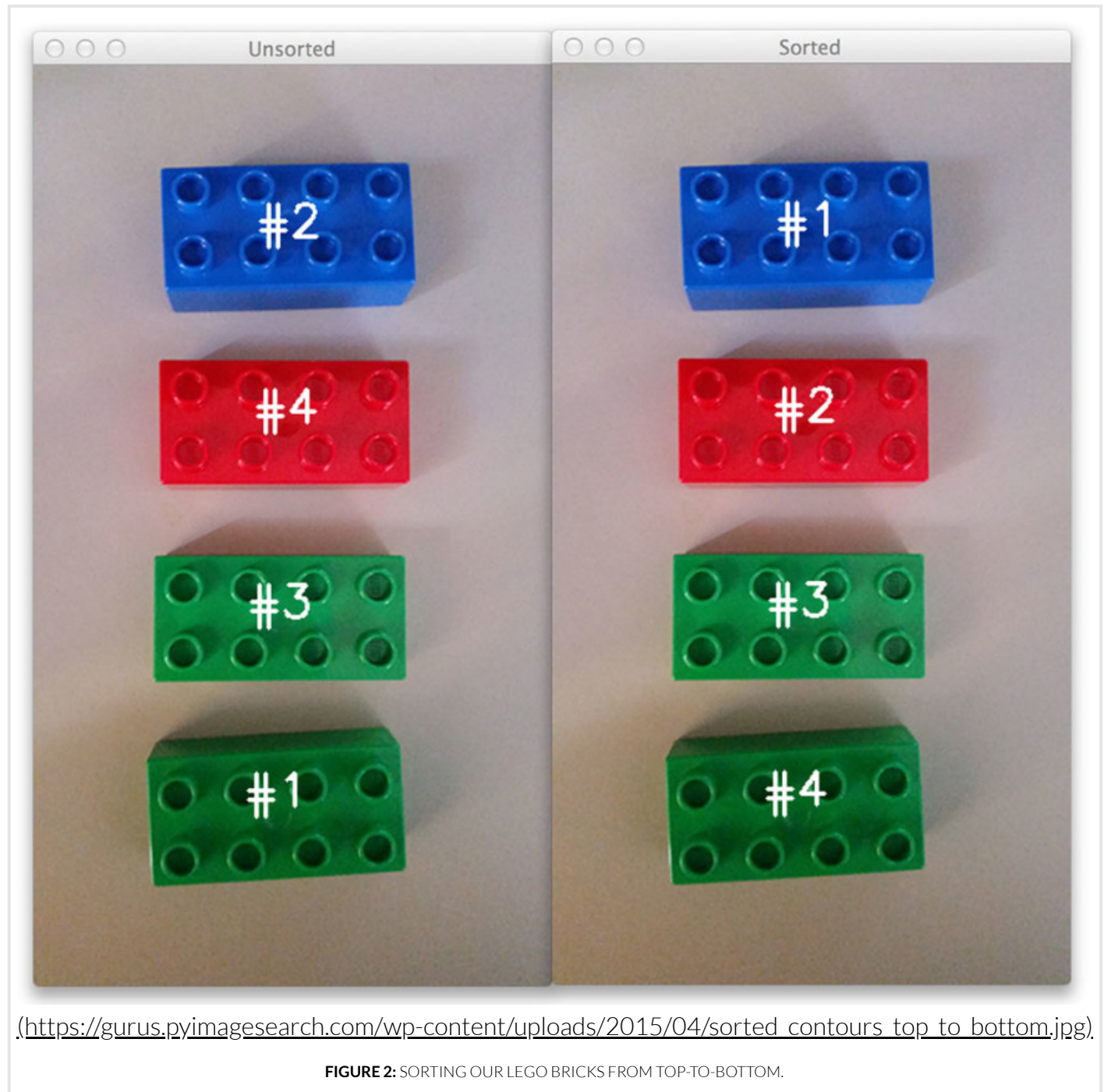
Finally, we take these sorted contours, loop over them, draw each individual one, and finally display the output image to our screen (**Lines 84-89**).

Let's put our hard work to the test.

Open up a terminal, navigate to your source code and execute the following command:

sort_contours.py	Shell
1 \$ python sort_contours.py --image images/lego_blocks_1.png --method "top-to-bottom"	

Your output should look like this:



Feedback

On the *left* we have our original unsorted contours. Clearly, we can see that the contours are very much out of order — the first contour is appearing at the very bottom and the second contour at the very top!

However, by applying our `sorted_contours` function we were able to sort our Lego bricks from top-to-bottom.

Let's take a look at another example.

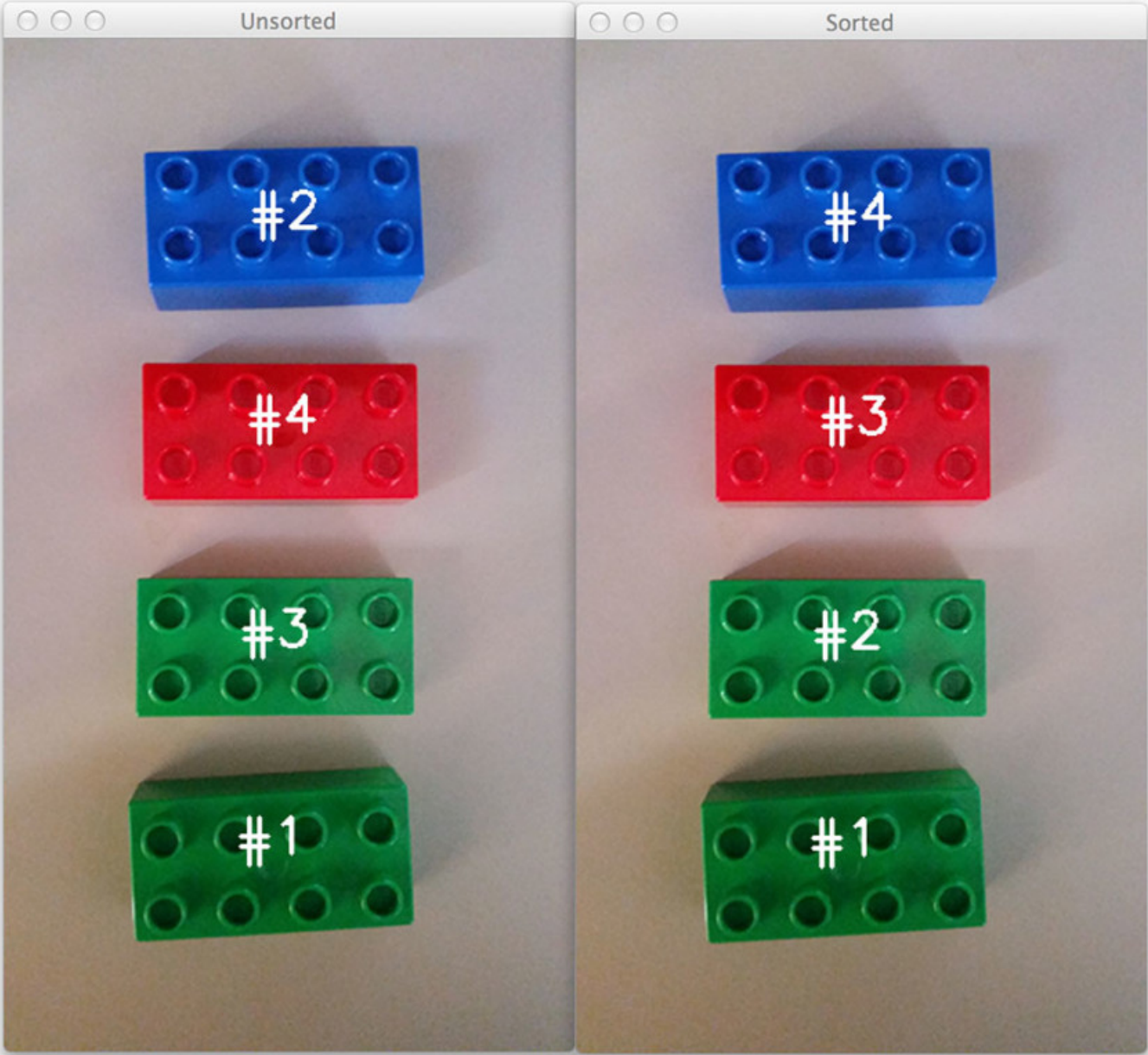
sort_contours.py

Shell

1 \$ python sort_contours.py --image images/lego_blocks_1.png --method "bottom-to-top"

Unsorted

Sorted



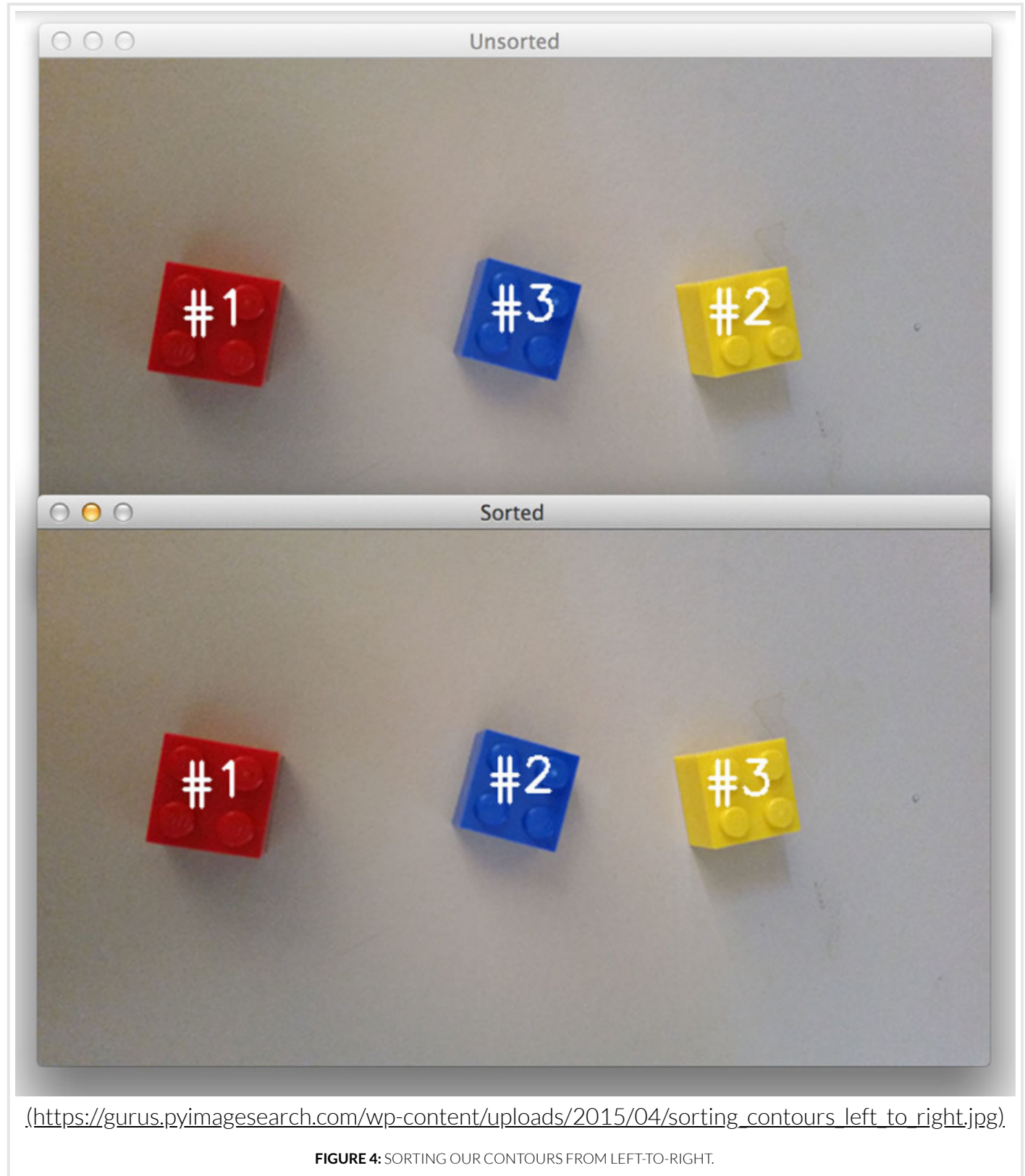
(https://gurus.pyimagesearch.com/wp-content/uploads/2015/04/sorting_contours_bottom_to_top.jpg).

FIGURE 3: SORTING OUR CONTOURS FROM BOTTOM-TO-TOP.

And here we have sorted on contours from bottom-to-top.

Of course, we can also sort contours horizontally as well:

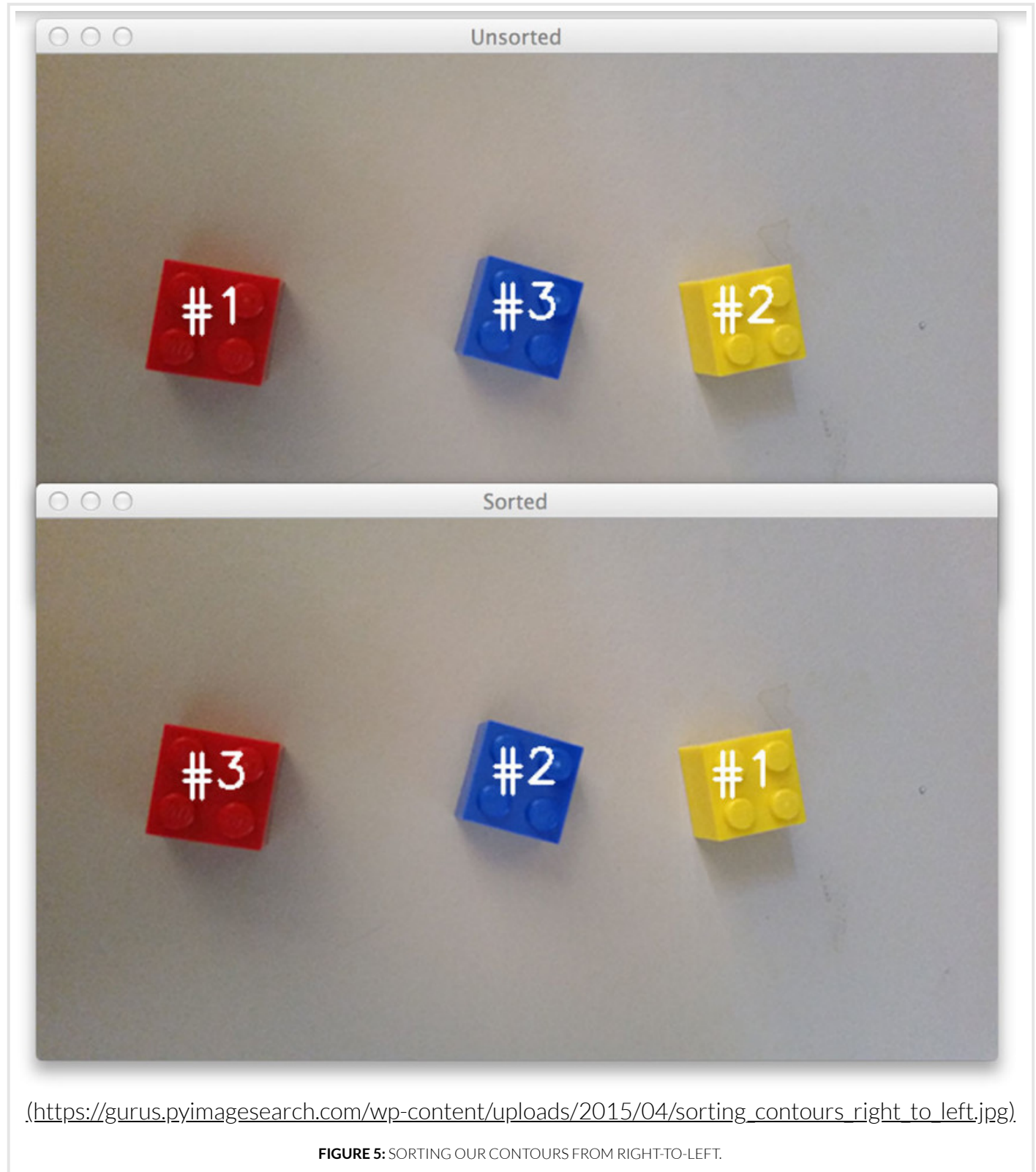
```
sort_contours.py Shell
1 $ python sort_contours.py --image images/lego_blocks_2.png --method "left-to-right"
```



Again, in the *top* image our contours are not in order. But in the *bottom* image we are able to successfully sort our contours without an issue.

One last example:

```
sort_contours.py Shell
1 $ python sort_contours.py --image images/lego_blocks_2.png --method "right-to-left"
```



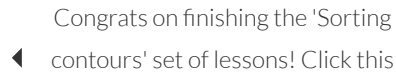
Feedback

As you can see, there's nothing to it – we're simply leveraging the bounding box of each object in the image to sort the contours by direction using Python and OpenCV.

In the future all you need is our trusty `sorted_contours` function and you'll always be able to sort contours in terms of direction without a problem

Summary

In this lesson article we learned how to sort contours from left-to-right, right-to-left, top-to-bottom, and bottom-to-top.



Realistically, we only needed to leverage two key functions.

The first critical function was the `cv2.boundingRect` method which computes the bounding box region of the contour. And based on these bounding boxes, we leveraged some Python magic and used our second critical function, `sorted`, to actually “sort” these bounding boxes in the direction we want.

Whew. We've covered a lot in this module. As you can see, there is a lot to say about contours.

One last reminder: if there is *anything* that you take away from these sections on contouring, it's that **contours are very simple yet extremely powerful**.

Whenever you are working on a new problem, consider how contours and the associated properties of contours can help you solve the problem. More often than not, a **clever use of contours can save you a lot of time and avoid more advanced (and tedious) techniques**.

Feedback

Downloads:

[Download the Code](#)

[.https://gurus.pyimagesearch.com/protected/code/computer vision basics/sortir](https://gurus.pyimagesearch.com/protected/code/computer_vision_basics/sortir)

Mark Complete

[← Previous Topic \(https://gurus.pyimagesearch.com/topic/contour-approximation/\)](#)

Course Progress

Ready to continue the course?

Click the button below to **continue your journey to computer vision guru**.

[I'm ready, let's go! \(/pyimagesearch-gurus-course/\)](/pyimagesearch-gurus-course/)

Resources & Links

- [PyImageSearch Gurus Community \(https://community.pyimagesearch.com/\)](https://community.pyimagesearch.com/)
- [PyImageSearch Virtual Machine \(https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/\)](https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/)
- [Setting up your own Python + OpenCV environment \(https://gurus.pyimagesearch.com/setting-up-your-python-opencv-development-environment/\)](https://gurus.pyimagesearch.com/setting-up-your-python-opencv-development-environment/)
- [Course Syllabus & Content Release Schedule \(https://gurus.pyimagesearch.com/course-syllabus-content-release-schedule/\)](https://gurus.pyimagesearch.com/course-syllabus-content-release-schedule/)
- [Member Perks & Discounts \(https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/\)](https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/)
- [Your Achievements \(https://gurus.pyimagesearch.com/achievements/\)](https://gurus.pyimagesearch.com/achievements/)
- [Official OpenCV documentation \(http://docs.opencv.org/index.html\)](http://docs.opencv.org/index.html)

Your Account

- [Account Info \(https://gurus.pyimagesearch.com/account/\)](https://gurus.pyimagesearch.com/account/)
- [Support \(https://gurus.pyimagesearch.com/contact/\)](https://gurus.pyimagesearch.com/contact/)
- [Logout \(https://gurus.pyimagesearch.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fgurus.pyimagesearch.com%2F&wponce=5736b21cae\)](https://gurus.pyimagesearch.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fgurus.pyimagesearch.com%2F&wponce=5736b21cae)

 Search