Navigation

pyimagesearch
be awesome at building image search engines

Image Stitching with OpenCV and Python

by **Adrian Rosebrock** on December 17, 2018 in **Image Descriptors**, **Tutorials**

Click here to download the source code to this post

In this tutorial, you will learn how to perform image stitching using Python, OpenCV, and the `cv2.createStitcher` and `cv2.Stitcher_create` functions. Using today's code you'll be able to stitch *multiple* images together, creating a panorama of stitched images.

Just under two years ago I published two guides on image stitching and panorama construction:

1. Fundamentals of image stitching
2. Real-time panorama and image stitching

Both of these tutorials covered the fundamentals of the typical image stitching algorithm, which, at a bare minimum, require four key steps:

1. Detecting keypoints (DoG, Harris, etc.) and extracting local invariant descriptors (SIFT, SURF, etc.) from two input images
2. Matching the descriptors between the images
3. Using the RANSAC algorithm to estimate a homography matrix using our matched feature vectors
4. Applying a warping transformation using the homography matrix obtained from Step #3

However, the biggest problem with my original implementations is that they were not capable of handling more than two input images.

In today's tutorial, we'll be revisiting image stitching with OpenCV, including how to stitch more than two images together into a panoramic image.

To learn how to stitch images with OpenCV and Python, *just keep reading!*

> **Looking for the source code to this post?**
> **Jump right to the downloads section.**

Image Stitching with OpenCV and Python

In the first part of today's tutorial, we'll briefly review OpenCV's image stitching algorithm that is baked into the OpenCV library itself via `cv2.createStitcher` and `cv2.Stitcher_create` functions.

From there we'll review our project structure and implement a Python script that can be used for image stitching.

We'll review the results of this first script, note its limitations, and then implement a second Python script that can be used for more aesthetically pleasing image stitching results.

Finally, we'll review the results of our second script and again note any limitations or drawbacks.

OpenCV's image stitching algorithm

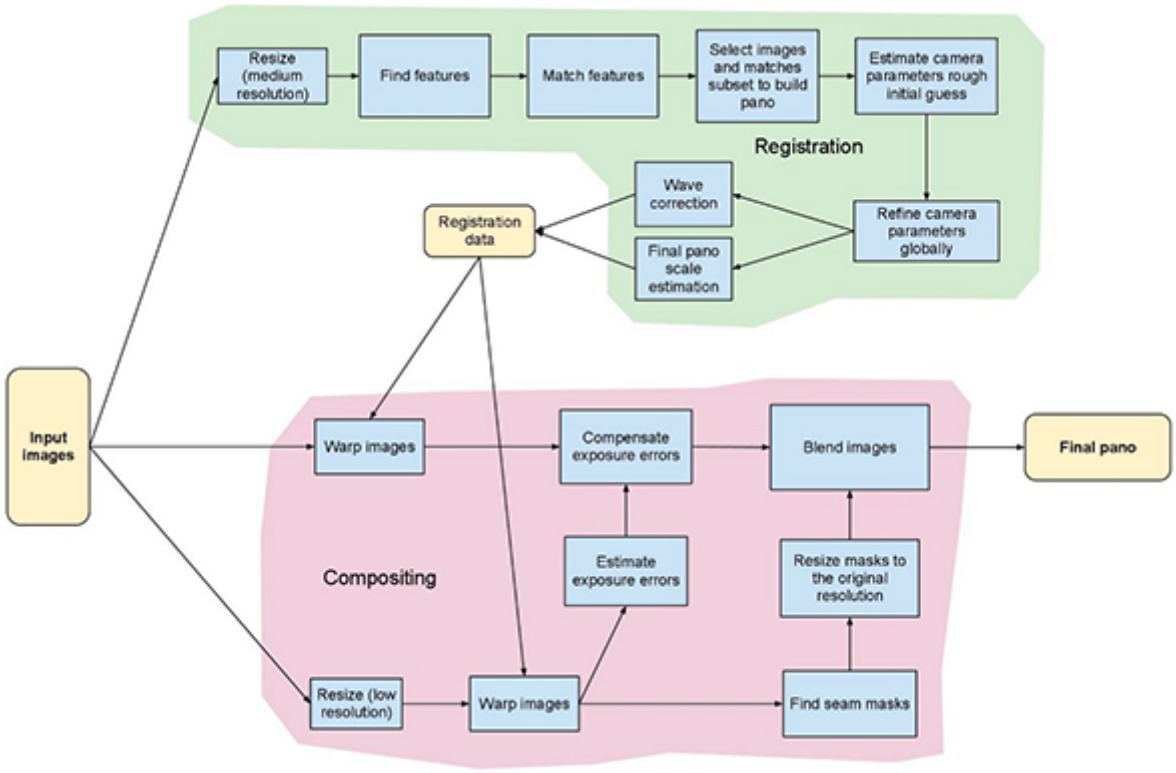


Figure 1: The stitching module pipeline implemented in the Stitcher class (source).

The algorithm we'll be using here today is similar to the method proposed by Brown and Lowe in their 2007 paper, *Automatic Panoramic Image Stitching with Invariant Features*.

Unlike previous image stitching algorithms which are sensitive to the ordering of input images, **the Brown and Lowe method is more robust,** making it *insensitive* to:

- Ordering of images
- Orientation of images
- Illumination changes
- Noisy images that are not actually part of the panorama

Furthermore, their image stitching method is capable of producing more aesthetically pleasing output panorama images through the use of gain compensation and image blending.

A complete, detailed review of the algorithm is outside the scope of this post, so if you're interested in learning more, please refer to the original publication.

Project structure

Let's see how this project is organized with the `tree` command:

```
Image Stitching with OpenCV and Python                                Shell
$ tree --dirsfirst
.
├── images
|   └── scottsdale
|       ├── IMG_1786-2.jpg
```

```
|       ├── IMG_1787-2.jpg
|       └── IMG_1788-2.jpg
├── image_stitching.py
├── image_stitching_simple.py
└── output.png

2 directories, 6 files
```

The input images go in the `images/` folder. I opted to make a subfolder for my `scottsdale/` set of images in case I wanted to add additional subfolders here later.

Today we'll be reviewing two Python scripts:

- `image_stitching_simple.py` : Our simple version of image stitching can be completed in less than 50 lines of Python code!
- `image_stitching.py` : This script includes my hack to extract an ROI of the stitched image for an aesthetically pleasing result.

The last file, `output.png` , is the name of the resulting stitched image. Using command line arguments, you can easily change the filename + path of the output image.

The cv2.createStitcher and cv2.Stitcher_create functions

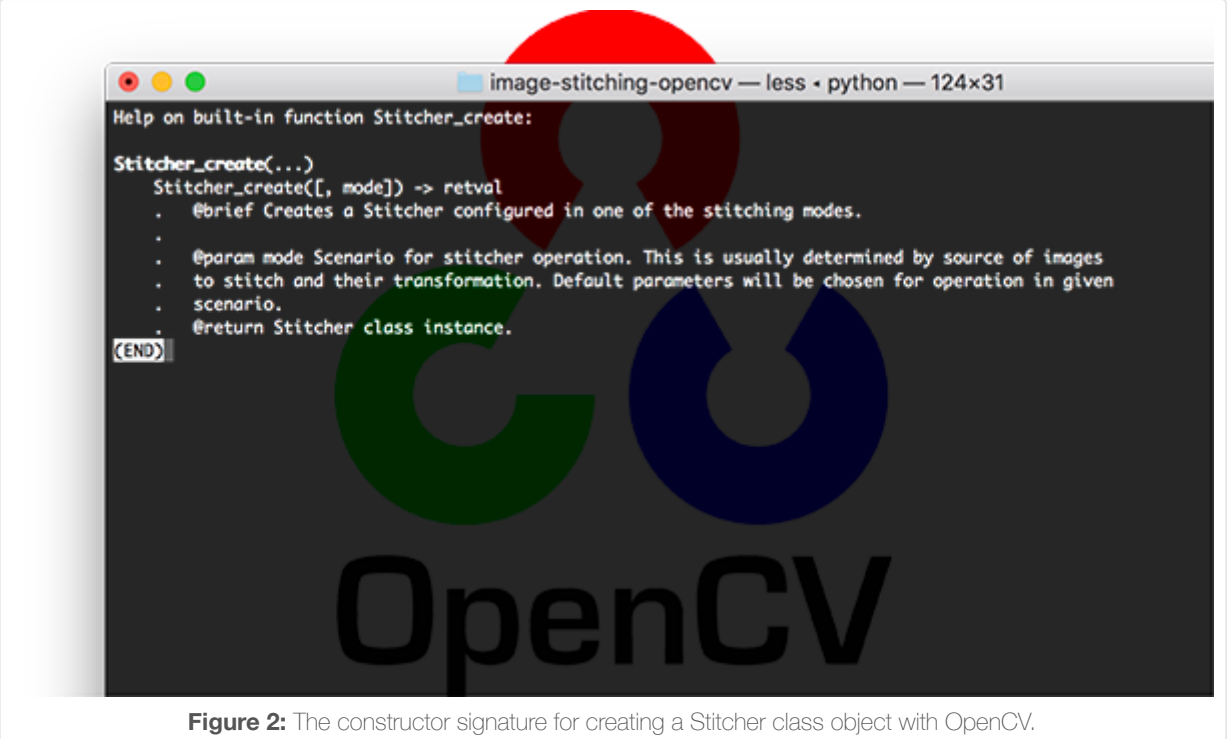


Figure 2: The constructor signature for creating a Stitcher class object with OpenCV.

OpenCV has already implemented a method similar to Brown and Lowe's paper via the `cv2.createStitcher` (OpenCV 3.x) and `cv2.Stitcher_create` (OpenCV 4) functions.

Assuming you have OpenCV properly configured and installed you'll be able to investigate the function signature of `cv2.createStitcher` for OpenCV 3.x:

```
Image Stitching with OpenCV and Python                                C++
createStitcher(...)
```

```
2      createStitcher([, try_use_gpu]) -> retval
```

Notice how this function has only a single parameter, `try_gpu` which can be used to improve your the throughout of your image stitching pipeline. OpenCV's GPU support is *limited* and I've never been able to get this parameter to work so I recommend always leaving it as `False` .

The `cv2.Stitcher_create` function for OpenCV 4 has a similar signature:

```
Image Stitching with OpenCV and Python                                C++
1  Stitcher_create(...)
2      Stitcher_create([, mode]) -> retval
3      .   @brief Creates a Stitcher configured in one of the stitching
4      .   modes.
5      .
6      .   @param mode Scenario for stitcher operation. This is usually
7      .   determined by source of images to stitch and their transformation.
8      .   Default parameters will be chosen for operation in given scenario.
9      .   @return Stitcher class instance.
```

To perform the actual image stitching we'll need to call the `.stitch` method:

```
Image Stitching with OpenCV and Python                                C++
1  OpenCV 3.x:
2  stitch(...) method of cv2.Stitcher instance
3      stitch(images[, pano]) -> retval, pano
4
5  OpenCV 4.x:
6  stitch(...) method of cv2.Stitcher instance
7      stitch(images, masks[, pano]) -> retval, pano
8      .   @brief These functions try to stitch the given images.
9      .
10     .   @param images Input images.
11     .   @param masks Masks for each input image specifying where to
12     .   look for keypoints (optional).
13     .   @param pano Final pano.
14     .   @return Status code.
```

This method accepts a list of input `images` , and then attempts to stitch them into a panorama, returning the output panorama image to the calling function.

The `status` variable indicates whether or not the image stitching was a success and can be one of four variables:

- `OK = 0` : The image stitching was a success.
- `ERR_NEED_MORE_IMGS = 1` : In the event you receive this status code, you will need more input images to construct your panorama. Typically this error occurs if there are not enough keypoints detected in your input images.
- `ERR_HOMOGRAPHY_EST_FAIL = 2` : This error occurs when the RANSAC homography estimation fails. Again, you may need more images or your images don't have enough distinguishing, unique texture/objects for keypoints to be accurately matched.
- `ERR_CAMERA_PARAMS_ADJUST_FAIL = 3` : I have never encountered this error before so I don't have much knowledge about it, but the gist is that it is related to failing to properly estimate camera intrinsics/extrinsics from the input images. If you encounter this error you may need to refer to the OpenCV documentation or even dive into the OpenCV C++ code.

Now that we've reviewed the `cv2.createStitcher` , `cv2.Stitcher_create` , and `.stitch` methods, let's move on to actually implementing image stitching with OpenCV and Python.

Implementing image stitching with Python

Let's go ahead and get started implementing our image stitching algorithm!

Open up the `image_stitching_simple.py` file and insert the following code:

Image Stitching with OpenCV and Python — Python

```python
# import the necessary packages
from imutils import paths
import numpy as np
import argparse
import imutils
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--images", type=str, required=True,
    help="path to input directory of images to stitch")
ap.add_argument("-o", "--output", type=str, required=True,
    help="path to the output image")
args = vars(ap.parse_args())
```

Our required packages are imported on **Lines 2-6**. Notably, we'll be using OpenCV and imutils. If you haven't already, go ahead and install them:

- To install OpenCV, just follow one of my OpenCV installation guides.
- The imutils package can be installed/updated with pip: `pip install --upgrade imutils` . Be sure to upgrade it as new features are often added.

From there we'll parse two command line arguments on **Lines 9-14**:

- `--images` : The path to the directory of input images to stitch.
- `--output` : The path to the output image where the result will be saved.

If you aren't familiar with the concepts of `argparse` and command line arguments then read this blog post.

Let's load our input images:

Image Stitching with OpenCV and Python — Python

```python
# grab the paths to the input images and initialize our images list
print("[INFO] loading images...")
imagePaths = sorted(list(paths.list_images(args["images"])))
images = []

# loop over the image paths, load each one, and add them to our
# images to stitch list
for imagePath in imagePaths:
    image = cv2.imread(imagePath)
    images.append(image)
```

Here we grab our `imagePaths` (**Line 18**).

Then for each `imagePath`, we'll load the `image` and add it to the `images` list (**Lines 19-25**).

Now that the `images` are in memory, let's go ahead and stitch them together into a panorama using OpenCV's built-in capability:

Image Stitching with OpenCV and Python — Python

```
27 # initialize OpenCV's image stitcher object and then perform the image
28 # stitching
29 print("[INFO] stitching images...")
30 stitcher = cv2.createStitcher() if imutils.is_cv3() else cv2.Stitcher_create()
31 (status, stitched) = stitcher.stitch(images)
```

The `stitcher` object is created on **Line 30**. Notice that depending on whether you're using OpenCV 3 or 4, a different constructor is called.

Subsequently, we can pass our `images` to the `.stitch` method (**Line 31**). The call to `.stitch` returns both a `status` and our `stitched` image (assuming the stitching was successful).

Finally, we'll both (1) write the stitched image to disk and (2) display it on the screen:

Image Stitching with OpenCV and Python — Python

```
33 # if the status is '0', then OpenCV successfully performed image
34 # stitching
35 if status == 0:
36     # write the output stitched image to disk
37     cv2.imwrite(args["output"], stitched)
38
39     # display the output stitched image to our screen
40     cv2.imshow("Stitched", stitched)
41     cv2.waitKey(0)
42
43 # otherwise the stitching failed, likely due to not enough keypoints)
44 # being detected
45 else:
46     print("[INFO] image stitching failed ({})".format(status))
```

Assuming our `status` flag indicates success (**Line 35**), we write the `stitched` image to disk (**Line 37**) and display it until a key is pressed (**Lines 40 and 41**).

Otherwise, we'll simply print a failure message (**Lines 45 and 46**).

Basic image stitching results

To give our image stitching script a try, make sure you use the ***"Downloads"*** section of the tutorial to download the source code and example images.

Inside the `images/scottsdale/` directory you will find three photos that I took when visiting Frank Lloyd Wright's famous Taliesin West house in Scottsdale, AZ:

Figure 3: Three photos to test OpenCV image stitching with. These images were taken by me in Scottsdale, AZ at Frank Lloyd Wright's famous Taliesin West house.

Our goal is to stitch these three images into a single panoramic image. To perform the stitching, open up a terminal, navigate to where you downloaded the code + images, and execute the following command:

Image Stitching with OpenCV and Python — Shell

```
$ python image_stitching_simple.py --images images/scottsdale --output output.png
[INFO] loading images...
[INFO] stitching images...
```

Figure 4: Image stitching performed with OpenCV. This image has undergone stitching but has yet to be cropped.

Notice how we have successfully performed image stitching!

But what about those black regions surrounding the panorama? What are those?

Those regions are from performing the perspective warps required to construct the panorama.

There is a way to get rid of them…but we'll need to implement some additional logic in the next section.

A better image stitcher with OpenCV and Python

Figure 5: In this section, we'll learn how to improve image stitching with OpenCV by cropping out the region of the panorama inside the red-dash border shown in the figure.

Our first image stitching script was a good start but those black regions surrounding the panorama itself are not something we would call "aesthetically pleasing".

And more to the point, you wouldn't see such an output image from popular image stitching applications built into iOS, Android, etc.

Therefore, we're going to hack our script a bit and include some additional logic to create more aesthetically pleasing panoramas.

I'm going to again reiterate that this method is a hack.

We'll be reviewing basic image processing operations including threshold, contour extraction, morphological operations, etc. in order to obtain our desired result.

To my knowledge, OpenCV's Python bindings do not provide us with the required information to manually extract the maximum inner rectangular region of the panorama. **If OpenCV does, please let me know in the comments as I would love to know.**

Let's go ahead and get started — open up the `image_stitching.py` script and insert the following code:

Image Stitching with OpenCV and Python — Python

```
# import the necessary packages
from imutils import paths
import numpy as np
import argparse
import imutils
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--images", type=str, required=True,
    help="path to input directory of images to stitch")
ap.add_argument("-o", "--output", type=str, required=True,
```

```
    help="path to the output image")
ap.add_argument("-c", "--crop", type=int, default=0,
    help="whether to crop out largest rectangular region")
args = vars(ap.parse_args())

# grab the paths to the input images and initialize our images list
print("[INFO] loading images...")
imagePaths = sorted(list(paths.list_images(args["images"])))
images = []

# loop over the image paths, load each one, and add them to our
# images to stich list
for imagePath in imagePaths:
    image = cv2.imread(imagePath)
    images.append(image)

# initialize OpenCV's image sticher object and then perform the image
# stitching
print("[INFO] stitching images...")
stitcher = cv2.createStitcher() if imutils.is_cv3() else cv2.Stitcher_create()
(status, stitched) = stitcher.stitch(images)
```

All of this code is identical to our previous script with one exception.

The `--crop` command line argument has been added. When a `1` is provided for this argument in the terminal, we'll go ahead and perform our cropping hack.

The next step is where we start implementing additional functionality:

Image Stitching with OpenCV and Python — Python

```
# if the status is '0', then OpenCV successfully performed image
# stitching
if status == 0:
    # check to see if we supposed to crop out the largest rectangular
    # region from the stitched image
    if args["crop"] > 0:
        # create a 10 pixel border surrounding the stitched image
        print("[INFO] cropping...")
        stitched = cv2.copyMakeBorder(stitched, 10, 10, 10, 10,
            cv2.BORDER_CONSTANT, (0, 0, 0))

        # convert the stitched image to grayscale and threshold it
        # such that all pixels greater than zero are set to 255
        # (foreground) while all others remain 0 (background)
        gray = cv2.cvtColor(stitched, cv2.COLOR_BGR2GRAY)
        thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY)[1]
```

Notice how I've made a new block for when the `--crop` flag is set on **Line 40**. Let's begin going through this block:

- First, we'll add a `10` pixel border to all sides of our `stitched` image (**Lines 43 and 44**), ensuring we'll be able to find contours of the complete panorama outline later in this section.
- Then we're going to create a `gray` version of our `stitched` image (**Line 49**).
- And from there we threshold the `gray` image (**Line 50**).

Here is the result (`thresh`) of those three steps:

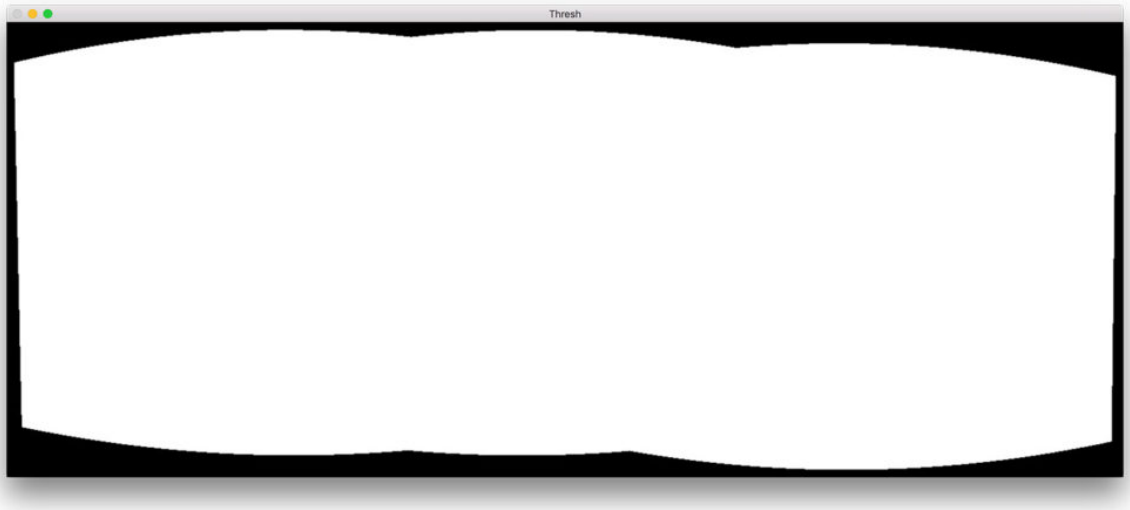


Figure 6: After thresholding, we're presented with this threshold mask highlighting where the OpenCV stitched + warped image resides.

We now have a binary image of our panorama where white pixels (255) are the foreground and black pixels (0) are the background.

Given our thresholded image we can apply contour extraction, compute the bounding box of the largest contour (i.e., the outline of the panorama itself), and draw the bounding box:

Image Stitching with OpenCV and Python — Python

```
52          # find all external contours in the threshold image then find
53          # the *largest* contour which will be the contour/outline of
54          # the stitched image
55          cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
56              cv2.CHAIN_APPROX_SIMPLE)
57          cnts = imutils.grab_contours(cnts)
58          c = max(cnts, key=cv2.contourArea)
59
60          # allocate memory for the mask which will contain the
61          # rectangular bounding box of the stitched image region
62          mask = np.zeros(thresh.shape, dtype="uint8")
63          (x, y, w, h) = cv2.boundingRect(c)
64          cv2.rectangle(mask, (x, y), (x + w, y + h), 255, -1)
```

Contours are extracted and parsed on **Lines 55-57**. **Line 58** then grabs the contour with the largest area (i.e., the outline of the stitched image itself).

Note: *The* `imutils.grab_contours` *function is new in* `imutils==0.5.2` *to accommodate OpenCV 2.4, OpenCV 3, and OpenCV 4 and their different return signatures for* `cv2.findContours` *.*

Line 62 allocates memory for our new rectangular mask. **Line 63** then calculates the bounding box of our largest contour. Using the bounding rectangle information, on **Line 64**, we draw a solid white rectangle on the mask.

The output of the above code block would look like the following:

Figure 7: The smallest rectangular region that the entire OpenCV panorama can fit in.

This bounding box is the *smallest rectangular region* that the *entire panorama* can fit in.

Now, here comes one of the biggest hacks I've ever put together for a blog post:

Image Stitching with OpenCV and Python — Python

```
66          # create two copies of the mask: one to serve as our actual
67          # minimum rectangular region and another to serve as a counter
68          # for how many pixels need to be removed to form the minimum
69          # rectangular region
70          minRect = mask.copy()
71          sub = mask.copy()
72
73          # keep looping until there are no non-zero pixels left in the
74          # subtracted image
75          while cv2.countNonZero(sub) > 0:
76              # erode the minimum rectangular mask and then subtract
77              # the thresholded image from the minimum rectangular mask
78              # so we can count if there are any non-zero pixels left
79              minRect = cv2.erode(minRect, None)
80              sub = cv2.subtract(minRect, thresh)
```

On **Lines 70 and 71** we create two copies of our `mask` image:

1. The first mask, `minMask`, will be slowly reduced in size until it can fit inside the inner part of the panorama (see **Figure 5** at the top of this section).
2. The second mask, `sub`, will be used to determine if we need to keep reducing the size of `minMask`.

Line 75 starts a `while` loop that will continue looping until there are no more foreground pixels in `sub`.

Line 79 performs an erosion morphological operation to reduce the size of `minRect`.

Line 80 then subtracts `thresh` from `minRect` — once there are no more foreground pixels in `minRect` then we can break from the loop.

I have included an animation of the hack below:

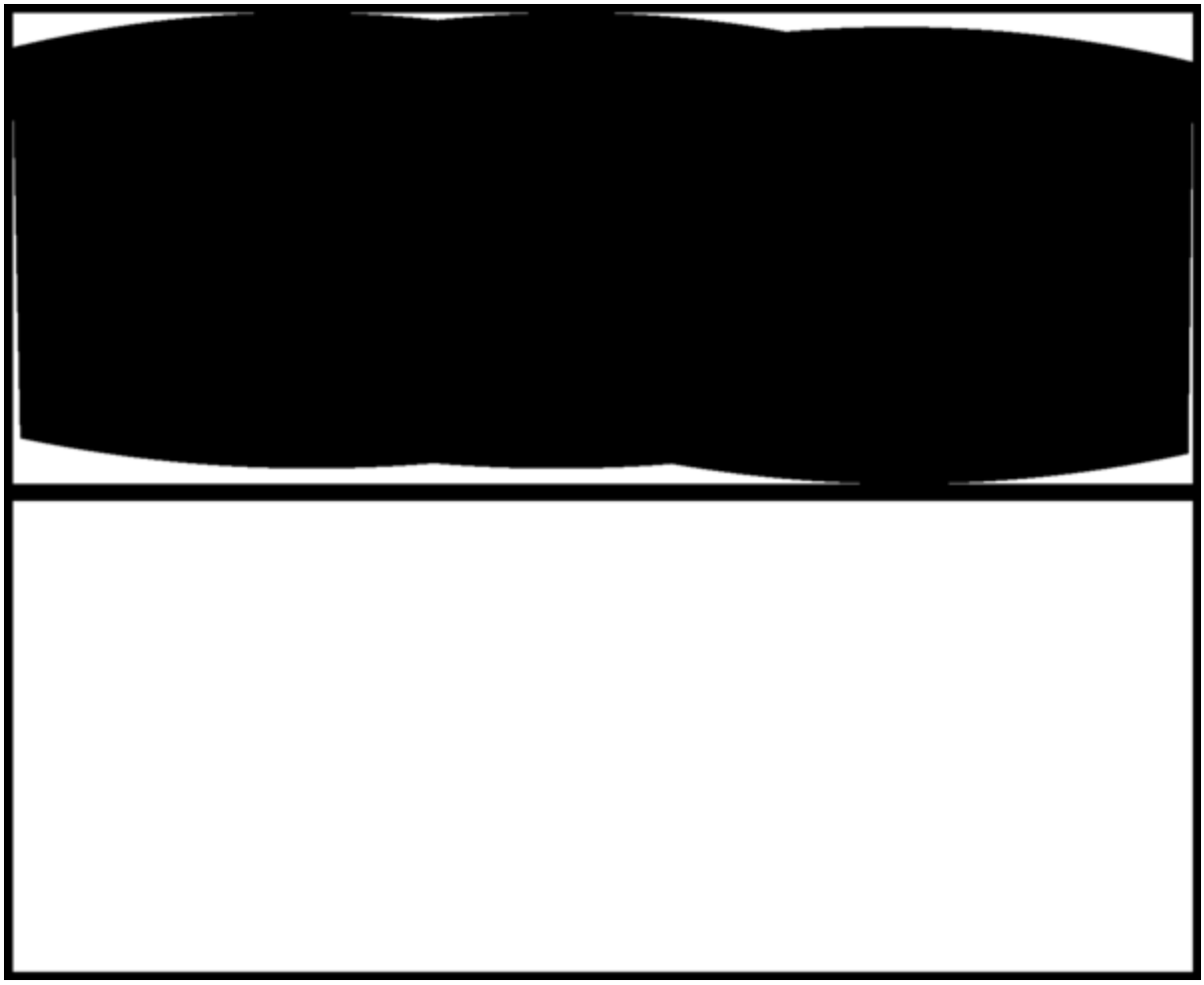


Figure 8: An animation of the hack I came up with to extract the `minRect` region of the OpenCV panorama image, making for an aesthetically pleasing stitched image

On the *top*, we have our `sub` image and on the *bottom* we have the `minRect` image.

Notice how the size of `minRect` is progressively reduced until there are no more foreground pixels left in `sub` — at this point we know we have found the *smallest rectangular mask* that can fit into the largest rectangular region of the panorama.

Given the minimum inner rectangle we can again find contours and compute the bounding box, but this time we'll simply extract the ROI from the `stitched` image:

Image Stitching with OpenCV and Python — Python

```
82          # find contours in the minimum rectangular mask and then
83          # extract the bounding box (x, y)-coordinates
84          cnts = cv2.findContours(minRect.copy(), cv2.RETR_EXTERNAL,
85              cv2.CHAIN_APPROX_SIMPLE)
86          cnts = imutils.grab_contours(cnts)
87          c = max(cnts, key=cv2.contourArea)
88          (x, y, w, h) = cv2.boundingRect(c)
89
90          # use the bounding box coordinates to extract the our final
91          # stitched image
92          stitched = stitched[y:y + h, x:x + w]
```

Here we have:

- Found contours in `minRect` (**Lines 84 and 85**).
- Handled parsing contours for multiple OpenCV versions (**Line 86**). You'll need `imutils>=0.5.2` to use this function.
- Grabbed the largest contour (**Line 87**).
- Computed the bounding box of the largest contour (**Line 88**).
- Extracted the ROI from our `stitched` using the bounding box information (**Line 92**).

The final `stitched` image can be displayed to our screen and then saved to disk:

Image Stitching with OpenCV and Python — Python

```
94      # write the output stitched image to disk
95      cv2.imwrite(args["output"], stitched)
96
97      # display the output stitched image to our screen
98      cv2.imshow("Stitched", stitched)
99      cv2.waitKey(0)
100
101 # otherwise the stitching failed, likely due to not enough keypoints)
102 # being detected
103 else:
104     print("[INFO] image stitching failed ({})".format(status))
```

Lines 95-99 handle saving and displaying the image regardless of whether or not our cropping hack is performed.

Just as before, if the `status` flag didn't come back as a success, we'll print an error message (**Lines 103 and 104**).

Let's go ahead and check out the results of our improved image stitching + OpenCV pipeline.

Improved image stitching results

Again, make sure you have used the ***"Downloads"*** section of today's tutorial to download the source code and example images.

From there, open up a terminal and execute the following command:

Image Stitching with OpenCV and Python — Shell

```
$ python image_stitching.py --images images/scottsdale --output output.png \
    --crop 1
[INFO] loading images...
[INFO] stitching images...
[INFO] cropping...
```

Figure 8: The result of our multiple image stitching with OpenCV and Python.

Notice how this time we have removed the black regions from the output stitched images (caused by the warping transformations) by applying our hack detailed in the section above.

Limitations and drawbacks

In a previous tutorial, I demonstrated how you could build a real-time panorama and image stitching algorithm — this tutorial hinged on the fact that we were manually performing keypoint detection, feature extraction, and keypoint matching, giving us access to the homography matrix used to warp our two input images into a panorama.

And while OpenCV's built-in `cv2.createStitcher` and `cv2.Stitcher_create` functions are certainly capable of constructing accurate, aesthetically pleasing panoramas, one of the primary drawbacks of the method is that it abstracts away any access to the homography matrices.

One of the assumptions of real-time panorama construction is that the scene itself is not changing much in terms of content.

Once we compute the initial homography estimation we should only have to occasionally recompute the matrix.

Not having to perform a full-blown keypoint matching and RANSAC estimation gives us a tremendous boost of speed when building our panorama, so without access to the raw homography matrices, it would be challenging to take OpenCV's built-in image stitching algorithm and convert it to real-time.

Running into errors when performing image stitching using OpenCV?

It is possible that you may run into errors when trying to use either the `cv2.createStitcher` function or `cv2.Stitcher_create` functions.

The two "easy to resolve" errors I see people encounter is forgetting what version of OpenCV they are using.

For example, if you are using OpenCV 4 but try to call `cv2.createSticher` you will encounter the following error message:

```
Image Stitching with OpenCV and Python                                Python
1 >>> cv2.createStitcher
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4 AttributeError: module 'cv2' has no attribute 'createStitcher'
```

You should instead be using the `cv2.Stitcher_create` function.

Similarly, if you are using OpenCV 3 and you try to call `cv2.Sticher_create` you will receive this error:

```
Image Stitching with OpenCV and Python                                Python
1 >>> cv2.Stitcher_create
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4 AttributeError: module 'cv2' has no attribute 'Stitcher_create'
```

Instead, use the `cv2.createSticher` function.

If you are unsure which OpenCV version you are using you can check using `cv2.__version__` :

```
Image Stitching with OpenCV and Python                                Python
1 >>> cv2.__version__
2 '4.0.0'
```

Here you can see that I am using OpenCV 4.0.0.

You can perform the same check on your system.

The final error that you can encounter, and arguably the most common, is related to OpenCV (1) not having contrib support and (2) being compiled without the `OPENCV_ENABLE_NONFREE=ON` option enabled.

To resolve this error you must have the `opencv_contrib` modules installed along with the `OPENCV_ENABLE_NONFREE` option set to `ON` .

If you are encountering an error related to OpenCV's non-free and contrib modules, make sure you refer to my OpenCV install guides to ensure you have the full install of OpenCV.

Note: *Please note that I cannot help debug your own OpenCV install if you did not follow one of my install guides so please make sure you're using my OpenCV install guides when configuring your system.*

Summary

In today's tutorial you learned how to perform multiple image stitching using OpenCV and Python.

Using both OpenCV and Python we were able to stitch multiple images together and create panoramic images.

Our output panoramic images were not only *accurate* in their stitching placement but also *aesthetically pleasing* as well.

However, one of the biggest drawbacks of using OpenCV's built-in image stitching class is that it abstracts away much of the internal computation, including the resulting homography matrices themselves.

If you are trying to perform real-time image stitching, as we did in a previous post, you may find it beneficial to cache the homography matrix and only occasionally perform keypoint detection, feature extraction, and feature matching.

Skipping these steps and using the cached matrix to perform perspective warping can reduce the computational burden of your pipeline and ultimately speed-up the real-time image stitching algorithm, but unfortunately, OpenCV's `cv2.createStitcher` Python bindings do not provide us with access to the raw matrices.

If you are interested in learning more about real-time panorama construction, please refer to my previous post.

I hope you enjoyed today's tutorial on image stitching!

To download the source code to today's post, and be notified tutorials are published here on PyImageSearch, *just enter your email address in the form below!*

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 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! Sound good? If so, enter your email address and I'll send you the code immediately!

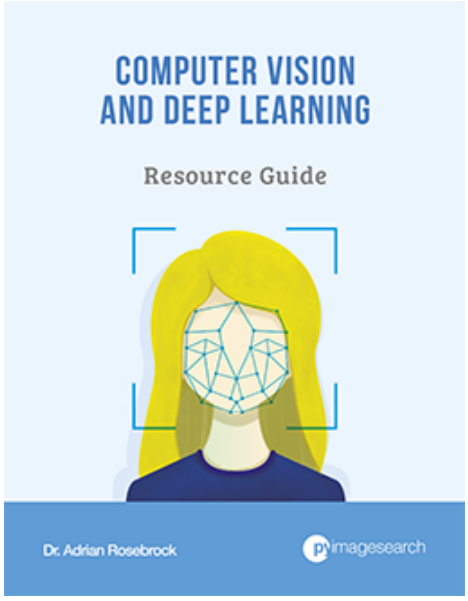
Email address:

rzepaw+pyimages@gmail.com

DOWNLOAD THE CODE!

Resource Guide (it's totally free).

Enter your email address below to get my **free 17-page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF**. Inside you'll find my hand-picked tutorials, books, courses, and Python libraries to



help you master computer vision and deep learning!

Your email address

DOWNLOAD THE GUIDE!

image descriptors, **image stitching**, **keypoint detection**, **keypoint matching**, **panorama**, **ransac**

‹ Keras – Save and Load Your Deep Learning Models

How to use Keras fit and fit_generator (a hands-on tutorial) ›

48 Responses to *Image Stitching with OpenCV and Python*

Manmohan Bishnoi December 17, 2018 at 12:05 pm #

REPLY

Hello Adrian,

Thanks for a great tutorial once again.
Typical steps for panorama creation from multiple images are:

1. Detect Features
2. Compute Descriptors
3. Match features
4. Remove false matches
5. Calculate Homography
6. Stitch images
7. Detect seams
8. Multi-band blend for final panorama
9. Crop for aesthetic final image

I am trying to generate real-time panorama from images taken using burst shots from a mobile camera rotated in horizontal circular direction, similar to most Apps in App store.
But doing all these steps for two adjacent images takes about 10 seconds with OpenCV Stitcher pipeline.

So if I take 32 images at 11.25 degrees apart and try to stitch them in real-time, it is way too slow.
To speed up things I tried doing some tasks in parallel using multi-threading.

Any tips how to speedup this and reduce time for stitching two adjacent images to about 2-4 seconds?

I am trying to generate panorama incrementally by stitching images in sequence.

Thanks

Adrian Rosebrock December 17, 2018 at 12:45 pm #

REPLY

Hi Manmohan — I would suggest looking at the GPU functionality I hinted at in the post. That should ideally help you speedup the pipeline but it may require you going down the rabbit hole and playing with the raw C++ code to get it to work (I haven't been able to).

Tim December 17, 2018 at 7:44 pm #

REPLY

Thanks @Manmohan. I was wondering if I could stitch the frames from 3 cameras together a deliver a panoramic video stream. (I currently use a high-res fisheye camera but even with de-warping the image is not 'right', and I'd like better resolution for zoom in)

Your comment leads me to believe I'll never get ~15 frames per sec via stitching.

(I've been playing inference accelerators (Movidius) but that is no help here)

@Adrian, thoughts?
My use case is a video of a wilderness; ~165 deg FoV. Boars, deer, mnt lion might be, rarely, anywhere, and when they are I like to zoom in.
Note, the object detection is to alert me when something interesting appears. Also this is a personal 'fun' project so time and resources are scarce.

Adrian Rosebrock December 18, 2018 at 8:53 am #

REPLY

Is your wildlife camera stationary and non-moving? If so, yes, you can absolutely achieve 15+ FPS. This tutorial will show you how. You'll need to update the code to work with more than two images or hack the OpenCV C++ source to access the raw homography matrix.

Cenk December 17, 2018 at 1:52 pm #

REPLY

Many thanks for the detailed demo.

Do you think Cython would help to speed up the stitching process?

Thanks

Adrian Rosebrock December 17, 2018 at 2:22 pm #

REPLY

No, mainly because we're calling OpenCV's functions which are C++ compiled functions. There is a small overhead between the Python call and the compiled bindings but that overhead is essentially nothing.

Agus Ambarwari December 17, 2018 at 11:54 pm #

REPLY

Thank you Mr. Adrian, I am from Indonesia and interest in computer vision.
Can stitching image use more than two images and not only from right to left? May, from top to bottom.

Adrian Rosebrock December 18, 2018 at 8:49 am #

REPLY

Technically yes, but I haven't tried with this particular implementation. Be sure to give it a try!

Ankit Agrawal December 18, 2018 at 4:13 am #

REPLY

Hello Adrian,

Thank you for you post, this has been a lot of help for me.

Also, I wanted to ask are you coming up with the post for realtime panorama stitching with more than two cameras, precisely 4 cameras?

Adrian Rosebrock December 18, 2018 at 8:46 am #

REPLY

As I mentioned in the post, the method used here realistically cannot be used for real-time panorama stitching. You would need to hack the OpenCV C++ code to access the homography matrix and only apply new feature matching once every N frames.

Younus December 19, 2018 at 12:23 am #

REPLY

Hello Adrian i am also trying to achieve the same as Ankit mentioned, isn't it possble by combining your this tutorial and real time stitching tutorial by saving cache and applying it to

every frame possible? and if not then what are you telling about hacking opencv c++ code can you guide a little about it. Thank you.

Adrian Rosebrock December 19, 2018 at 1:52 pm #

REPLY

No, make sure you re-read this tutorial as I explain why you cannot cache the homography matrix. I personally have not worked with the C++ code. My suggestion was that you would need to do your own research with the code and so if you could hack the code and compile your own bindings that exposes the matrix. It would be a challenging, non-trivial process.

Rahul Ragesh December 19, 2018 at 8:00 am #

REPLY

Hello Adrian,

I have 3 retinal images which have warped using Homography manually (I couldn't use the built-in function because image qualities were bad). Now I want to blend these images together. I couldn't find any opencv python functions for blending. Am I missing something?

Adrian Rosebrock December 19, 2018 at 1:47 pm #

REPLY

Are you referring to the specific blending functions used by the algorithm in this post? If so, those functions are abstracted by the C++ API. You should refer to the OpenCV docs and source code for the stitching module.

Thomas Paul July 6, 2019 at 3:29 am #

REPLY

Hello Rahul,

Did you figure out how to do this? I'm facing a similar problem.

Thanks,
Thomas.

mjbordalo December 19, 2018 at 2:39 pm #

REPLY

Hello, thanks for another great tutorial.
I would like to do a stitching but with a top-view camera.
Like photos taken from a drone and sequentially stritch this photos.

Which steps should i change?
tnhks in advance

Adrian Rosebrock December 20, 2018 at 5:21 am #

REPLY

Have you tried using the code as-is? If so, what problems did you run into?

mjbordalo January 3, 2019 at 12:14 pm #

REPLY

yes. I tried to take a bunch of photos with my phone pointing downward while i was turning around. (so that the last picture would be similiar to the first one) The output resolt tries to put the images in a landscape mode like all are side by side

William Stevenson December 19, 2018 at 3:28 pm #

REPLY

Hello Adrian

I used pip install opencv_contrib_python, which fetched opencv_contrib_python-3.4.4.19-cp37-cp37m-win_amd64.whl for 64 bit python 3.7. It does not show errors during installation. cv2.__version__ shows 3.4.4. Release date is 27 Nov 2018.

Set OPENCV_ENABLE_NONFREE CMake option and rebuild the library in function ‘cv::xfeatures2d::SURF::create’

I thought that opencv_contrib_modules would contain all the contributed modules, or is this not guaranteed?

Adrian Rosebrock December 20, 2018 at 5:20 am #

REPLY

No, the pip install of opencv-contrib-python does not include the NONFREE modules. To enable them you would need to compile OpenCV from source. You should follow one of my OpenCV install guides to compile from source.

Hure December 19, 2018 at 7:16 pm #

REPLY

Can this sample code, perhaps with some parameter change, also be used to do what I’ll call pixel exact stiching? For example stitch together 5 screen captures of parts of a Google map with some overlap between each screen capture (and same zoom level and so on of course) into a larger map image? In other words find vertical or horizontal one pixel thick lines that are identical and join two

images at each such seam. If this code won't do that, do you know of some other opencv commands to use for that?

REPLY

Adrian Rosebrock December 20, 2018 at 5:16 am #

That is a pretty specific use case. You can try it and see but I don't think you'll be able to obtain that level of accuracy. It's certainly worth a test though!

REPLY

Sebastian December 20, 2018 at 10:19 am #

Great tutorial as usual, but I want to use image stitching for commercial use. But the NONFREE OpenCV functions are unfortionally patented. I'd like (my boss) to pay for them, but I'm without a clue where to ask for permission or how to buy a license for commercial usage. Do you by any chance know how I can obtain a license or use it comercially without breaking the law?

REPLY

Adrian Rosebrock December 27, 2018 at 11:07 am #

You would want to reach out to the patent holders. You can find them via Google's patent search.

REPLY

Emilio January 10, 2019 at 12:34 pm #

Hi Adrian,

Great tutorial as always! I was wondering, back in your OpenCV panorama stitching from 2016, you made a stitcher script and you were able to compute matches yourself. Have you found any way to retrieve matches from this method? I would like to store all the matches computed while stitching.

Thanks!

REPLY

Adrian Rosebrock January 11, 2019 at 9:35 am #

Hi Emilio — could you clarify a bit more what you mean by "retrieve matches from this method"? What specifically are you trying to accomplish?

REPLY

Jerry March 29, 2019 at 5:21 am #

Hi Adrian, Really nice tutorial. I just have one small doubt. I think it may be along the lines of what Emilio was asking. This stitcher_create() method uses SIFT to find key points right? so would it be possible to use ORB method instead of SIFT.

ORB is open source right and according to the Opencv's documentation page, if combined with FLANN matching supposed to be faster that SIFT + RANSAC. Although i may be wrong about be wrong about the performance, I just wanted to know if ORB + FLANN can be used in this method.

chen qu April 4, 2019 at 4:02 am #

REPLY

An excellent article and thank you again !

Another draw-back of this approach I observed is that probably within the stitching algorithm, the input tiles(images) are blurred, so the output of the stitching (image) looses the details from the original input, particularly for those already obscured features in the original images.

Shruti April 5, 2019 at 1:27 am #

REPLY

Hi Adrian,

Thanks for the tutorial. It is incredibly helpful. I was wondering if you could help me out with something.

I am trying to use opencv on Python to make a mosaic of images. So far I have been able to make a mosaic from around 20 images. But I am having trouble going beyond that. I get a status code of 3 when I use more than 20-25 images. Do you know if there is a limit on the number/size of images you can merge using opencv.

My images are incredibly large (4000 * 4000 pixels). I have around 100 such images.

Do you have any suggestions for me? Also, is it possible to see the raw opencv code for the createStitcher() and stitch() functions?

Yusuf April 15, 2019 at 12:58 pm #

REPLY

Hi Adrian, in Image stitching we are finding similarity between two images. so can we use the same concept in background subtraction in scenarios when background frame get slightly changed. So we can find the similar areas and create a new background frame for the new foreground frame. i dont know how to go starting in this direction

Adrian Rosebrock April 18, 2019 at 7:10 am #

REPLY

Sorry, I'm not fully understanding this question. You're using stitching frames and then performing background subtraction/motion detection, correct? I'm not sure how or why the "similar

areas” is being used or why they are important.

Marcel Jaramillo April 17, 2019 at 6:44 pm #

REPLY

Thanks for this amazing demo!

I having a problem with the memory, im getting this error:

Traceback (most recent call last):
Failed to allocate 746307632 bytes in function ‘cv::OutOfMemoryError’

Adrian Rosebrock April 18, 2019 at 6:29 am #

REPLY

Take a look at your error, it shows you what the problem is: you’re machine ran out of RAM, likely because you are trying to stitch together too many images and/or the images are too large (in terms of resolution). Try resizing your images first.

Mohanad April 18, 2019 at 2:50 am #

REPLY

Hello Adrian,
Is it possible to solve jigsaw puzzle using Python and openCV? The link bellow has some ideas and he already tried but no luck.

https://towardsdatascience.com/solving-jigsaw-puzzles-with-python-and-opencv-d775ba730660

Appreciated

Adrian Rosebrock April 18, 2019 at 6:27 am #

REPLY

By my understanding, you could potentially solve the jigsaw problem if the pieces were tiles (i.e., square or rectangular). But if they are actual pieces like a normal jigsaw it would require far too much computation to perform the exhaustive process of shape matching. It’s not really feasible.

Verina May 6, 2019 at 10:45 am #

REPLY

can this code be used to make 360 camera ?? and if it can’t , can you give me any tip or tutorial to help me ,please.
I am working on a 360 camera app.
Thanks in advance.

Adrian Rosebrock May 8, 2019 at 1:05 pm #

REPLY

Sorry, I don't have any code for a full 360 panorama camera.

peacherwu May 14, 2019 at 2:43 am #

REPLY

The Lowe's paper is dated 2007, not 2017.

Adrian Rosebrock May 15, 2019 at 2:41 pm #

REPLY

Whoops, thanks for catching that!

David May 31, 2019 at 4:58 am #

REPLY

Hi Adrian , I am using opencv 4.1.0 but code is not execute after this line stitcher = cv2.createStitcher() if imutils.is_cv3() else cv2.Stitcher_create()

then can you tell me where is my problem ?

Adrian Rosebrock June 6, 2019 at 8:43 am #

REPLY

What is the error you are receiving?

Brad June 18, 2019 at 4:10 pm #

REPLY

I am using the same opencv 4.1.0 and receive a "Bus error" after the call to stitcher.stitch(images). The code then stops.

Adrian Rosebrock June 19, 2019 at 1:42 pm #

REPLY

Thanks Brad. I'll be sure to take a look.

Ángel Ortiz June 5, 2019 at 9:20 am #

REPLY

With how many images can this be done? And what if I have an array of images? Meaning that there are not only made from taking pictures from left to right, but also from up and down.

Abhijit Nathwani June 25, 2019 at 9:10 am #

REPLY

Hi Adrian,

Thanks for the wonderful tutorial. I have been a regular reader of pyimagesearch and found most of your stuff useful!

I have a question regarding your hack that you have neatly applied for the border. Is it possible to control the rectangle dimensions? I'm losing some image data due to this. Can we select the rectangle co-ordinates or know the contour dimensions?

Thanks
Abhijit

Adrian Rosebrock June 26, 2019 at 1:04 pm #

REPLY

What do you mean by selecting the rectangle dimensions? As long as you have the contour itself you can compute the bounding box info via the "cv2.boundingRect" function. If you'd like more information on contours you should refer to Practical Python and OpenCV.

Koa July 31, 2019 at 8:00 pm #

REPLY

Hello,

I am trying to use this method to stitch together multiple images (6 in total). When I run the stitcher (before all of the panoramic image cleanup), it only stitches together 2 out of the 6 images together. I don't get any errors. What should I do?

jason0425 November 12, 2019 at 12:46 am #

REPLY

Hello Adrian!

I am trying to use this method to stitch together multiple 5 images. When I run the stitcher together 5 images together. When the image is stitched, black distortion points occur at the overlapping points. what's the problem…?

Before you leave a comment...

Hey, Adrian here, author of the PyImageSearch blog. I'd love to hear from you, but before you submit a comment, **please follow these guidelines:**

- **If you have a question, *read the comments first.*** You should also search this page (i.e., *ctrl* + *f*) for keywords related to your question. It's likely that I have already addressed your question in the comments.
- **If you are copying and pasting code/terminal output, *please don't.*** Reviewing another programmers' code is a very time consuming and tedious task, and due to the volume of emails and contact requests I receive, I simply cannot do it.
- **Be respectful of the space.** I put *a lot* of my own personal time into creating these free weekly tutorials. On average, each tutorial takes me 15-20 hours to put together. I love offering these guides to you and I take pride in the content I create. Therefore, I will not approve comments that include large code blocks/terminal output as it destroys the formatting of the page. Kindly be respectful of this space.
- **Be patient.** I receive 200+ comments and emails per day. Due to spam, and my desire to personally answer as many questions as I can, I hand moderate all new comments (typically once per week). I try to answer as many questions as I can, but I'm only one person. Please don't be offended if I cannot get to your question
- **Do you need priority support?** Consider purchasing one of my books and courses. I place customer questions and emails in a separate, special priority queue and answer them first. **If you are a customer of mine you will receive a *guaranteed response* from me.** If there's any time left over, I focus on the community at large and attempt to answer as many of those questions as I possibly can.

Thank you for keeping these guidelines in mind before submitting your comment.

Leave a Reply

Name (required)

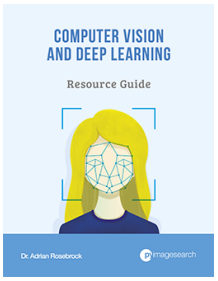
Email (will not be published) (required)

Website

SUBMIT COMMENT

Search...

Resource Guide (it's totally free).



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

Download for Free!

Raspberry Pi for Computer Vision

You can teach your **Raspberry Pi** to "see" using **Computer Vision**, **Deep Learning**, and **OpenCV**. **Let me show you how.**

CLICK HERE TO LEARN MORE

Deep Learning for Computer Vision with Python Book — OUT NOW!

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 MASTER DEEP LEARNING

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 yourself.

CLICK HERE TO MASTER FACE DETECTION

PyImageSearch Gurus: NOW ENROLLING!

The PyImageSearch Gurus course is *now enrolling!* Inside the course you'll learn how to perform:

- Automatic License Plate Recognition (ANPR)
- Deep Learning
- Face Recognition
- *and much more!*

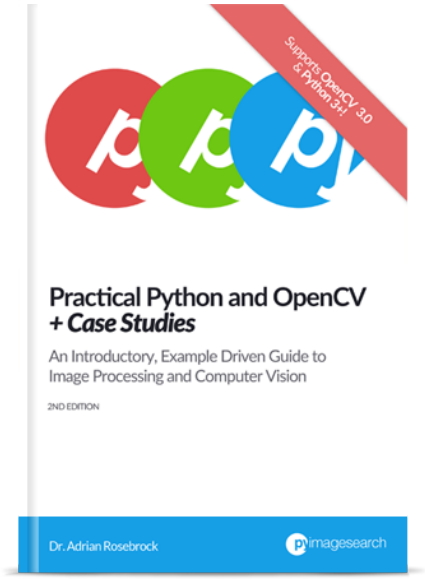
Click the button below to learn more about the course, take a tour, and get 10 (FREE) sample lessons.

TAKE A TOUR & GET 10 (FREE) LESSONS

Hello! I'm Adrian Rosebrock.

I'm Ph.D and entrepreneur who has spent his entire adult life studying Computer Vision and Deep Learning. I'm here to help you master CV, DL, and OpenCV. Learn More

Learn computer vision in a single weekend.



POPULAR

Raspbian Stretch: Install OpenCV 3 + Python on your Raspberry Pi
SEPTEMBER 4, 2017

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3
APRIL 18, 2016

Face recognition with OpenCV, Python, and deep learning
JUNE 18, 2018

Home surveillance and motion detection with the Raspberry Pi, Python, OpenCV, and Dropbox
JUNE 1, 2015

Install OpenCV and Python on your Raspberry Pi 2 and B+
FEBRUARY 23, 2015

Real-time object detection with deep learning and OpenCV
SEPTEMBER 18, 2017

Ubuntu 16.04: How to install OpenCV
OCTOBER 24, 2016