Image stitching with OpenCV and Python

Python Lessons
Feb 19, 2019 · 9 min read ★

You already know that Google photos app has stunning automatic features like video making, panorama stitching, collage making, sorting out images based by the persons in the photo and many others. Have you ever wondered, how all these function work ? So I though, how hard can it be to make panorama stitching on my own by using Python language.

So what is image stitching ? In simple terms, for an input there should be a group of images, the output is a composite image such that it is a culmination of image scenes. At the same time, the logical flow between the images must be preserved.

For example, think about sea horizon while you are taking few photos of it. From a group of these images, we are essentially creating a single stitched image, that explains

the full scene in detail. It is quite an interesting algorithm.

Let's first understand the concept of image stitching. Basically if you want to capture a big scene and your camera can only provide an image of a specific resolution and that resolution is 640 by 480, it is certainly not enough to capture the big panoramic view. So, what we can do is to capture multiple images of the entire scene and then put all bits and pieces together into one big image. Such photos of ordered scenes of collections are called panoramas. The entire process of acquiring multiple image and converting them into such panoramas is called as image stitching. And finally, we have one beautiful big and large photograph of the scenic view.

Firstly, let us install opencv version 3.4.2.16. If you have never version first do "pip uninstall opencv" bofore installing older version. If you will work with never version, you will be required to build opencv library by your self to enable image stitching function, so it's much easier to install older version:

```
pip install opencv-contrib-python==3.4.2.16
```

Next we are importing libraries that we will use in our code:

```
import cv2
import numpy as np
```

For our tutorial we are taking this beautiful photo, which we will slice into two left and right photos, and we'll try to get same or very similar photo back.

So I sliced this image into two images that they would have some kind of overlap region:

So here is the list of steps what we should do to get our final stiched result:

1. Compute the sift-key points and descriptors for left and right images.
2. Compute distances between every descriptor in one image and every descriptor in the other image.
3. Select the top best matches for each descriptor of an image.
4. Run RANSAC to estimate homography.
5. Warp to align for stitching.
6. Finally stitch them together.

So starting from the first step, we are importing these two images and converting them to grayscale, if you are using large images I recommend you to use cv2.resize because if you have older computer it may be very slow and take quite long. If you want to resize image size i.e. by 50% just change from fx=1 to fx=0.5.

```
img_ = cv2.imread('original_image_left.jpg')
img_ = cv2.resize(img_, (0,0), fx=1, fy=1)
img1 = cv2.cvtColor(img_,cv2.COLOR_BGR2GRAY)
```

```
img = cv2.imread('original_image_right.jpg')
img = cv2.resize(img, (0,0), fx=1, fy=1)
img2 = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

We still have to find out the features matching in both images. We shall be using opencv_contrib's SIFT descriptor. SIFT (Scale Invariant Feature Transform) is a very powerful OpenCV algorithm. You can read more OpenCV's docs on SIFT for Image to understand more about features. These best matched features act as the basis for stitching. We extract the key points and sift descriptors for both the images as follows:

```
sift = cv2.xfeatures2d.SIFT_create()
# find the key points and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
```

kp1 and kp2 are keypoints, des1 and des2 are the descriptors of the respective images. If we'll plot this image with features, this is how it will look:

```
cv2.imshow('original_image_left_keypoints',cv2.drawKeypoints(img_,kp
1,None))
```

Image on left shows actual image. Image on the right is annotated with features detected by SIFT:

Once you have got the descriptors and key points of two images, we will find correspondences between them. Why do we do this ? Well, in order to join any two images into a bigger images, we must find overlapping points. These overlapping points will give us an idea of the orientation of the second image according to first one. And based on these common points, we get an idea whether the second image is bigger or smaller or has it been rotated and then overlapped, or maybe scaled down/up and then fitted. All such information is yielded by establishing correspondences. This process is called registration.

For matching images can be used either FLANN or BFMatcher methods that are provided by opencv. I will write both examples prove that we'll get same result. Both examples matches the features which are more similar in both photos. When we set parameter k=2, this way we are asking the knnMatcher to give out 2 best matches for each descriptor. "matches" is a list of list, where each sub-list consists of "k" objects, to read more about this go here. And here is the code:

FLANN matcher code:

```
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
match = cv2.FlannBasedMatcher(index_params, search_params)
matches = match.knnMatch(des1,des2,k=2)
```

BFMatcher matcher code:

```
match = cv2.BFMatcher()
matches = match.knnMatch(des1,des2,k=2)
```

Often in images there may be many chances that features may be existing in many places of the image. So we filter out through all the matches to obtain the best ones. So we apply ratio test using the top 2 matches obtained above. We consider a match if the ratio defined below is greater than the specified ratio.

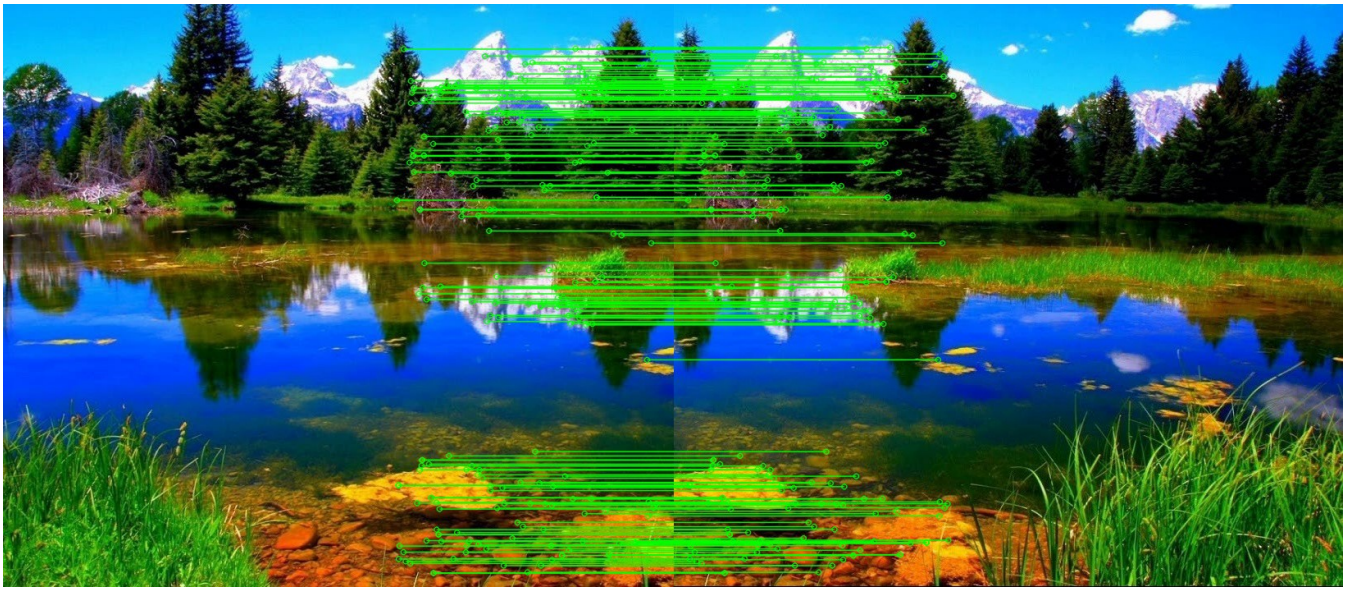
```
good = []
for m,n in matches:
    if m.distance < 0.03*n.distance:
        good.append(m)
```

Now we are defining the parameters of drawing lines on image and giving the output to see how it looks like when we found all matches on image:

```
draw_params = dict(matchColor = (0,255,0), # draw matches in green
color
                   singlePointColor = None,
                   flags = 2)

img3 = cv2.drawMatches(img_,kp1,img,kp2,good,None,**draw_params)
cv2.imshow("original_image_drawMatches.jpg", img3)
```

And here is the output image with matches drawn:



Here is the full code of this tutorial up to this:

```
import cv2
import numpy as np

img_ = cv2.imread('original_image_left.jpg')
#img_ = cv2.resize(img_, (0,0), fx=1, fy=1)
img1 = cv2.cvtColor(img_,cv2.COLOR_BGR2GRAY)

img = cv2.imread('original_image_right.jpg')
#img = cv2.resize(img, (0,0), fx=1, fy=1)
img2 = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

sift = cv2.xfeatures2d.SIFT_create()
# find the key points and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
```

```
kp2, des2 = sift.detectAndCompute(img2,None)
#cv2.imshow('original_image_left_keypoints',cv2.drawKeypoints(img_,k
p1,None))

#FLANN_INDEX_KDTREE = 0
#index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
#search_params = dict(checks = 50)
#match = cv2.FlannBasedMatcher(index_params, search_params)
match = cv2.BFMatcher()
matches = match.knnMatch(des1,des2,k=2)

good = []
for m,n in matches:
    if m.distance < 0.03*n.distance:
        good.append(m)

draw_params = dict(matchColor = (0,255,0), # draw matches in green
color
                   singlePointColor = None,
                   flags = 2)

img3 = cv2.drawMatches(img_,kp1,img,kp2,good,None,**draw_params)
cv2.imshow("original_image_drawMatches.jpg", img3)
```

So, once we have obtained best matches between the images, our next step is to calculate the homography matrix. As we described before, the homography matrix will be used with best matching points, to estimate a relative orientation transformation within the two images.

To estimate the homography in OpenCV is a simple task, it's a one line of code:

```
H, __ = cv2.findHomography(srcPoints, dstPoints, cv2.RANSAC, 5)
```

Before starting coding stitching algorithm we need to swap image inputs. So "img_" now will take right image and "img" will take left image.

So lets jump into stiching coding:

```
MIN_MATCH_COUNT = 10
if len(good) > MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good
]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good
]).reshape(-1,1,2)
```

```
        M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)

        h,w = img1.shape
        pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0]
]).reshape(-1,1,2)
        dst = cv2.perspectiveTransform(pts,M)

        img2 = cv2.polylines(img2,[np.int32(dst)],True,255,3,
cv2.LINE_AA)
        cv2.imshow("original_image_overlapping.jpg", img2)
    else:
        print ("Not enough matches are found - %d/%d" %
(len(good),MIN_MATCH_COUNT))
```

So at first we set our minimum match condition count to 10 (defined by MIN_MATCH_COUNT), and we only do stitching if our good matched exceeds our required matches. Otherwise simply show a message saying not enough matches are present.

So in if statement we are converting our Keypoints (from a list of matches) to an argument for findHomography() function. I can't explain this in details, because didn't had time to chatter this and there is no use for that.

Simply talking in this code line cv2.imshow("original_image_overlapping.jpg", img2) we are showing our received image overlapping area:

So, once we have established a homography we need to to warp perspective, essentially change the field of view, we apply following homography matrix to the image:

```
warped_image = cv2.warpPerspective(image, homography_matrix,
dimension_of_warped_image)
```

So we use this as following:

```
dst = cv2.warpPerspective(img_,M,(img.shape[1] + img_.shape[1],
img.shape[0]))
dst[0:img.shape[0], 0:img.shape[1]] = img
```

In above two lines of code we are taking overlapping area from two given images. Then in “dst” we have received only right side of image which is not overlapped, so in second line of code we are placing our left side image to final image. So at this point we have fully stitched image:

So from this point what is left is to remove dark side of image, so we'll write following code to remove black font from all image borders:

```
def trim(frame):
    #crop top
    if not np.sum(frame[0]):
        return trim(frame[1:])
    #crop bottom
    elif not np.sum(frame[-1]):
        return trim(frame[:-2])
    #crop left
    elif not np.sum(frame[:,0]):
        return trim(frame[:,1:])
    #crop right
    elif not np.sum(frame[:,-1]):
        return trim(frame[:,:-2])
    return frame
```

And here is the final defined function we call to trim borders and at the same time we show that mage in our screen. If you want you can also write it to disk:

```
cv2.imshow("original_image_stiched_crop.jpg", trim(dst))
#cv2.imwrite("original_image_stiched_crop.jpg", trim(dst))
```

With above code we'll receive original image as in first place:

Here is the full final code:

```
import cv2
import numpy as np

img_ = cv2.imread('original_image_right.jpg')
#img_ = cv2.imread('original_image_left.jpg')
#img_ = cv2.resize(img_, (0,0), fx=1, fy=1)
img1 = cv2.cvtColor(img_,cv2.COLOR_BGR2GRAY)

img = cv2.imread('original_image_left.jpg')
#img = cv2.imread('original_image_right.jpg')
#img = cv2.resize(img, (0,0), fx=1, fy=1)
img2 = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

sift = cv2.xfeatures2d.SIFT_create()
# find key points
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

#cv2.imshow('original_image_left_keypoints',cv2.drawKeypoints(img_,k
p1,None))

#FLANN_INDEX_KDTREE = 0
#index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
#search_params = dict(checks = 50)
#match = cv2.FlannBasedMatcher(index_params, search_params)
match = cv2.BFMatcher()
matches = match.knnMatch(des1,des2,k=2)

good = []
for m,n in matches:
    if m.distance < 0.03*n.distance:
        good.append(m)

draw_params = dict(matchColor=(0,255,0),
                       singlePointColor=None,
                       flags=2)

img3 = cv2.drawMatches(img_,kp1,img,kp2,good,None,**draw_params)
#cv2.imshow("original_image_drawMatches.jpg", img3)

MIN_MATCH_COUNT = 10
if len(good) > MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good
]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good
]).reshape(-1,1,2)

    M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

    h,w = img1.shape
    pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0]
```

```
]).reshape(-1,1,2)
    dst = cv2.perspectiveTransform(pts, M)
    img2 = cv2.polylines(img2,[np.int32(dst)],True,255,3,
cv2.LINE_AA)
    #cv2.imshow("original_image_overlapping.jpg", img2)
else:
    print("Not enought matches are found - %d/%d",
(len(good)/MIN_MATCH_COUNT))

dst = cv2.warpPerspective(img_,M,(img.shape[1] + img_.shape[1],
img.shape[0]))
dst[0:img.shape[0],0:img.shape[1]] = img
cv2.imshow("original_image_stitched.jpg", dst)

def trim(frame):
    #crop top
    if not np.sum(frame[0]):
        return trim(frame[1:])
    #crop top
    if not np.sum(frame[-1]):
        return trim(frame[:-2])
    #crop top
    if not np.sum(frame[:,0]):
        return trim(frame[:,1:])
    #crop top
    if not np.sum(frame[:,-1]):
        return trim(frame[:,:-2])
    return frame

cv2.imshow("original_image_stitched_crop.jpg", trim(dst))
#cv2.imsave("original_image_stitched_crop.jpg", trim(dst))
```

. . .

In this tutorial post we learned how to perform image stitching and panorama construction using OpenCV and wrote a final code for image stitching.

Our image stitching algorithm requires four main steps: detecting key points and extracting local invariant descriptors; get matching descriptors between images; apply RANSAC to estimate the homography matrix; apply a warping transformation using the homography matrix.

This algorithm works well in practice when constructing panoramas only for two images.

Original source for this tutorial is here: #part 1 and #part 2

You can find more interesting tutorial on my website: https://pylessons.com

Machine Learning Python Opencv Programming Image