International Journal of Engineering, Science and Mathematics
Vol. 8 Issue 4, April 2019,
ISSN: 2320-0294 Impact Factor: 6.765
Journal Homepage: http://www.ijesm.co.in, Email: ijesmj@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gage as well as in Cabell's Directories of Publishing Opportunities, U.S.A

OVERLAPPING FIELDS WITH IMAGE PROCESSING: IMAGE STITCHING

Jagannath Mahapatro*, Research Scholar*
P.G Department of I & CT Fakiramohan University*, Balasore*
Dr Sabyasachi Pattanik**, Professor **
P.G Department of I & CT Fakiramohan University**, Balasore**

ABSTRACT

Image stitching or photo stitching is the process of combining multiple photographic images with overlapping fields of view to produce a segmented panorama or high-resolution image. Commonly performed through the use of computer software, most approaches to image stitching require nearly exact overlaps between images and identical exposures to produce seamless results, although some stitching algorithms actually benefit from differently exposed images by doing high-dynamic-range-imaging in regions of overlap. Some digital cameras can stitch their photos internally. nce the illumination in two views cannot be guaranteed to be the same, stitching two images could create a visible seam. Other reasons for the seam appearing could be the background changing between two images for the same continuous foreground. In general, the major issues to deal with are presence of parallax, lens distortion, scene motion, and exposure differences. For panoramic stitching, the ideal set of images will have a reasonable amount of overlap (at least 15–30%) to overcome lens distortion and have enough detectable features. The set of images will have consistent exposure between frames to minimize the probability of seams occurring. In the non-ideal real-life case, the intensity varies across the whole scene and so does the contrast and intensity across the frames. Lens distortion, motion in the scene, and misalignment all cause ghosting. Also, the ratio of width to height of a panorama image needs to be taken into account to create a visually pleasing composite.

KEY WORDS: Image stitching, Overlapping,Image Stitching, Panorama

INTRODUCTION

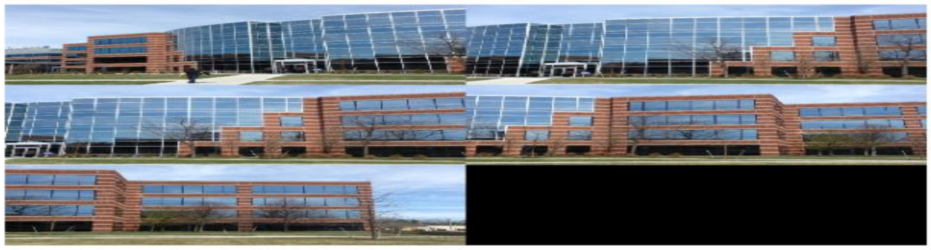
Feature detection is necessary to automatically find correspondences between images. Robust correspondences are required in order to estimate the necessary transformation to align an image with the image it is being composited on. Corners, blobs, harris corners, and difference of gaussian of harris corners (DoG) are good features since they are repeatable and distinct. One of the first operators for interest point detection was developed by Hans P. Moravec in 1977 for his research involving the automatic navigation of a robot through a clustered environment. Moravec also defined the concept of "points of interest" in an image and concluded these interest points could be used to find matching regions in different images. The Moravec operator is considered to be a corner detector because it defines interest points as points where there are large intensity variations in all directions. This often is the case at corners. However, Moravec was not specifically interested in finding corners, just distinct regions in an image that could be used to register consecutive image frames. Harris and Stephens improved upon Moravec's corner detector by considering the differential of the corner score with respect to direction directly. They needed it as a processing step to build interpretations of a robot's environment based on image sequences. Like Moravec, they needed a method to match corresponding points in consecutive image frames, but were interested in tracking both corners and edges between frames. SIFT and SURF are recent keypoint or interest point detector algorithms but a point to note is that these are patented and their commercial usage restricted. Once a feature has been detected, a descriptor method like SIFT descriptor can be applied to later match them.

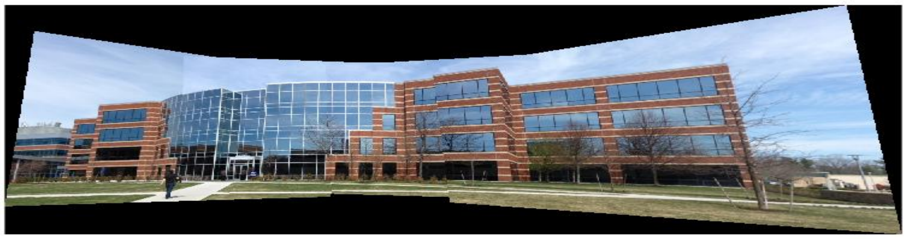
OVERVIEW OF IMAGE STITCHING ALGORITHMS

1. Firstly, algorithms are needed to determine the appropriate mathematical model relating pixel coordinates in one image to pixel coordinates in another. This is for image alignment.
2. Next, we need to estimate the correct alignments relating various pairs (or collections) of images. Algorithms that combine direct pixel-to-pixel comparisons with gradient descent (and other optimization techniques) can be used to estimate these parameters.
3. Distinctive features can be found in each image and then efficiently matched to rapidly establish correspondences between pairs of images. When multiple images exist in a panorama, techniques have been developed to compute a globally consistent set of alignments and to efficiently discover which images overlap one another.
4. For image stitching, we must first decide on a final compositing surface onto which to warp or projectively transform and place all of the aligned images. We also need to develop algorithms to seamlessly blend the overlapping images, even in the presence of parallax, lens distortion, scene motion, and exposure differences.

MULTIPLE IMAGE STITCHING

I must say, even I was enjoying while developing this tutorial . Something about image perspective and enlarged images is simply captivating to a computer vision student (LOL) . I think, image stitching is an excellent introduction to the coordinate spaces and perspectives vision. Here I am going to show how to take an ordered set of many images, **(assuming they have been shot from left to right direction)** .So what is image stitching ? In simple terms, for an input group of images, the output is a composite image such that it is a culmination of scenes. At the same time, the logical flow between the images must be preserved.For example, consider the set of images below. (Taken from matlab examples). From a group of an input montage, we are essentially creating a singular stitched image. One that explains the full scene in detail. It is quite an interesting algorithm !





(Taken from matlab examples).

The implementation will be carried out in python programming language.

- Reading multiple images (in order)
- Finding logical consistencies within images (this will be done using homography).
- Stitching Up images.

Let's first understand the concept of mosaicking or image stitching. Basically if you want to capture a big scene. Now your camera can only provide an image of a specific resolution and that resolution , say 640 by 480 , is certainly not enough to capture the big panoramic view. So , what one can do is capture multiple images of the entire scene and then put together all bits and pieces into one big mat of images. Yes, it seems good .. right ! Such photographs , which pose as an ordered collection of a scene are called as mosaics or panoramas. The entire process of acquiring multiple image and converting them into such panoramas is called as image mosaicking. And finally, we have one beautiful big and large photograph of the scenic view.Another method for achieving this, is by using wide angle lens in your camera. What a wide angle lens does, is effectively increase your field of view.

The output, will differ (obviously). But for the purposes of this tutorial, let's get into how to create panoramas using computers and not lens.

SETTING UP THE ENVIRONMENT

Please note that your system is setup with Python 2.7 (Code implementation is in python2.7 if you have other versions, please modify the code accordingly) and OpenCV 3.0 . We will be using OpenCV's helper utilities for reading images, writing images and conversion of color spaces. Once the images are obtained, the entire computation of the panorama will be done using a home brewed function. This blog article is divided into three major parts.
***** Input, read and process images : image paths from text files. Each textfile contains the list of paths to each image. **Make sure that the paths are in left_to_right order of orientation.**
***** Computation relative orientation of images w.r.t each other : pairwise
***** The stitching / mix and match module : which essentially joins the two images at a time

ALGORITHM

The algorithm for performing image stitching is pretty straightforward.

```
1    images [] <-- Input images
2    Assuming, that the center image is no_of_images/2
3    let centerIdx = length(images)/2
4
5    for each images[] at positions 0 -> centerIdx :
6        perform leftward stitching
7
8    for each images[] at positions centerIdx -> length(images):
9        perform rightward stitching
```

The output will be a complete mosaic of the input images.

Some Constraints The algorithm is time consuming, due to the number of iterations involved, it is best that hte input number of images is not too high or not of very high resolution (eg. 4000x3000). My implementation is based on a 2 GB RAM computer having intel i3 processor (Not tested it on my machine yet !). Feel free to upgrade/scale this model using higher specs, or maybe GPU's .It's never too late to try.

PROJECT ARCHITECTURE

```
|_ code -|
|       |-- pano.py
|       |-- txtlists-|
|                   |--files1.txt ....
|
|_ images - |
|          |- img1.jpg
|          |- abc.jpg
|          .... and so on ...
```

To understand either of the leftward stitching or rightward stitching module, first let's get some vision concepts straight. They being:

- *Homography : Oh I love this !
- *Warping : Cause, without warping, homography would feel a bit lonely
- *Blending : Cause intensity differences are a bit too mainstream

HOMOGRAPHY

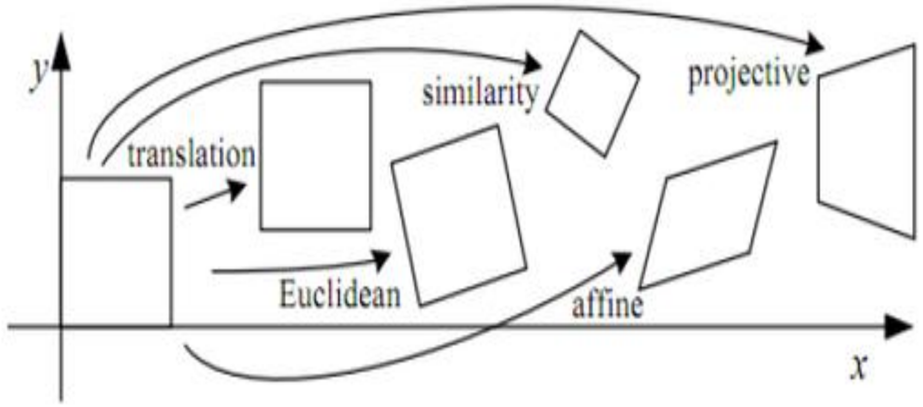
Okay, so assume you're looking at a scenery. You will be having a field of view and that field of view is of what you want to make a panorama. You can rotate your head and cover a big area. But while you are looking straight, looking directly perpendicularly at a sub-scene, the remaining part of the scenery appears slightly inclined or slightly narrowed out. This is due to simple physics. Since you are facing in one direction, the things to your extreme periphery appears unclear, reduced in dimension and not necessarily straight/normal (a bit inclined). This is exactly what we will be exploiting.

Consider the images shown in the above figure. Every image will contain some common portion with the other images. Due to this commonness we are able to say that *image* x

will either lie on to the right or left side of *image* y

. Anyway, now that I've made that clear, let's proceed as to how do we calculate homography. Say you have a pair of images *I*1 , *I*2

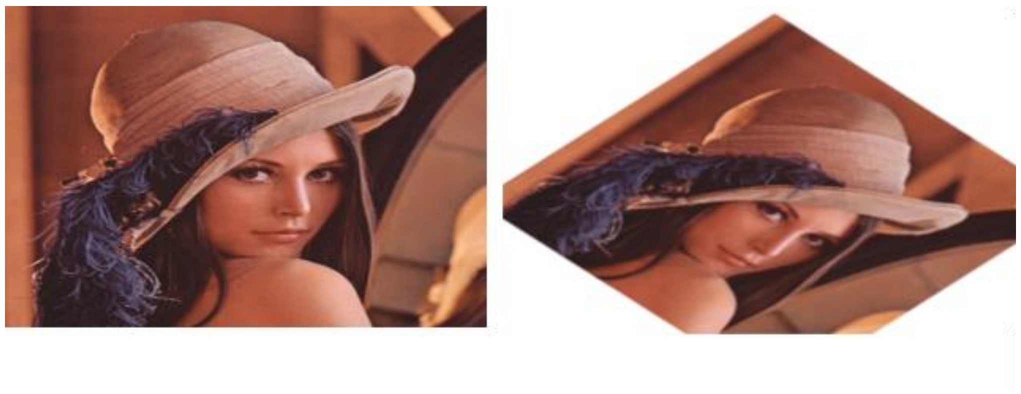
. You capture the first image. Then you decide to rotate your camera, or maybe perform some translation or maybe a combination of rotation / translation motion. Then having update your new camera position , you capture a second image. The problem now at hand is, How do you solve for a system wherein you're required to create a transformation that efficiently maps a point that is being projected in both the images. Or in simple terms, How do you visualize one image w.r.t another point of view, given there is some information available about both your points of views.



Types of transforms

Each of the above transformations performs some sorts of image transformation. For eg. a projective (well in your case,homography) transform will preserve straight lines, .. etc. Moving on, a homography matrix is such that , if applied to any image, transforms image plane P1 to another image plane P2.

See the pic below, you'll understand what i'm talking about.



Source :This SO Post

What you see above, can be an output of applying homography from $I1=H\times I2$

. HOW TO use Homography to transform pictures in OpenCV? (Check out this answer too. these pics have been taken from the aforementioned post).

You can use opencv findhomography ( ) method to solve for homography. For finding $I1=H\times I2$

you will need to pass coordinates of points in original image 1 plane and coordinates of target points in image 2 to the method. Once through, the method will spit out the homography matrix

HOW TO IDENTIFY POINTS TO CALCULATE HOMOGRAPHY !

One of the straight forward methods is as follows

Compute similar features in both images

Out of them , filter out good features (you'll find plenty of tutorials on these)

Make an array sorts of ; featuresofI1 ==> [srcPoints], featuresofI2 ==>[dstPoints] (using opencv nomenclature)

Compute Homography matrix using RANSAC algorithm

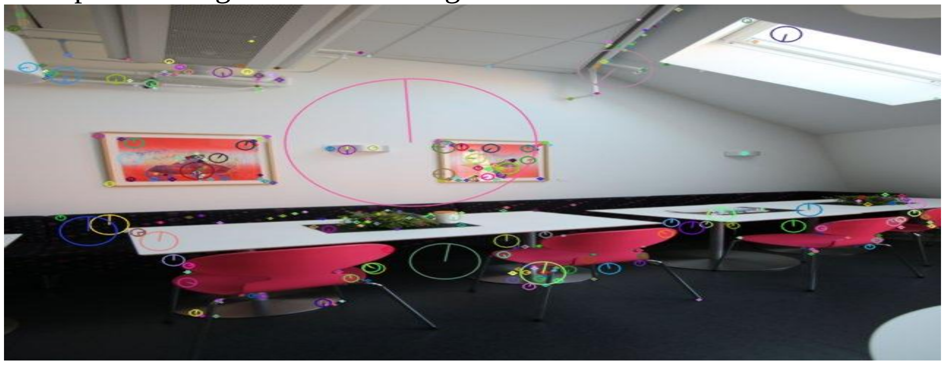
STEP 1: FEATURE EXTRACTION

We shall be using opencv_contrib's SIFT descriptor. SIFT , as in Scale Invariant Feature Transform, is a very powerful CV algorithm. Please read my Bag of Visual Words for Image classification post to understand more about features. Also, check out OpenCV's docs on SIFT. They are a pretty good resource as well!

```
sift_obj = cv2.xfeatures2d.SIFT_create()
descriptors, keypoints = sift_obj.detectAndCompute(image_gray, None)
```

If you plot the features, this is how it will look . (Image on left shows actual image. Image on the right is annotated with features detected by SIFT)

Example 1 : using Lunchroom image



Lunchroom image : PASSTA Dataset

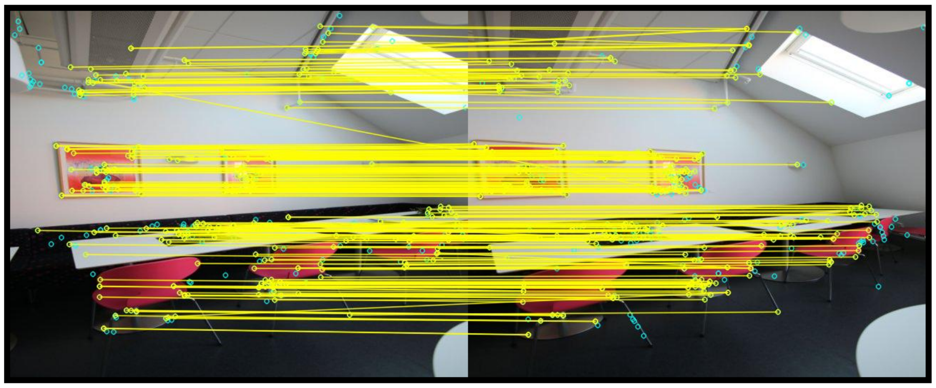
STEP 2: MATCHING CORRESPONDENCES BETWEEN IMAGES

Once you have got the descriptors and keypoints of 2 images, i.e. an image pair, 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 obtain as to what are the overlapping points. These overlapping points will give us an idea of the orientation of the second image w.r.t to the other one. And based on these common points, we get an idea whether the second image has just slid into the bigger image 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** .

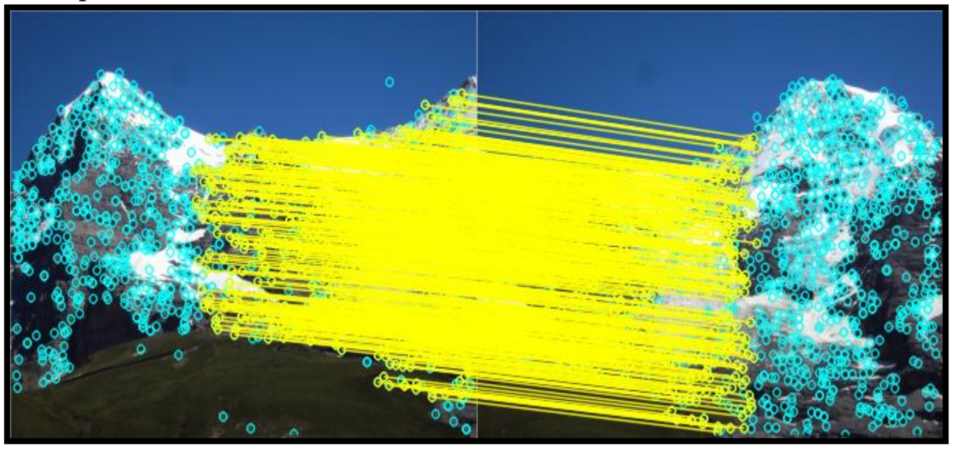
For matching, one can use either FLANN or BFMatcher, that is provided by opencv.

```
# FLANN parameters
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)   # or pass empty dictionary

flann = cv2.FlannBasedMatcher(index_params,search_params)

matches = flann.knnMatch(des1,des2,k=2)

img3 = cv2.drawMatchesKnn(img1c,kp1,img2c,kp2,matches,None,**draw_params)

cv2.imshow("correspondences", img3)
cv2.waitKey()
```

Having computed the matches, you get a similar output :
With the Lunchroom dataset


Feature matching on lunchroom images

With the Hill example:


Feature matching on hill example image

STEP 3: COMPUTE HOMOGRAPHY

Yes, once we have obtained matches between the images, our next step is to calculate the homography matrix. As described above, the homography matrix will use these matching points, to estimate a relative orientation transform within the two images. i.e. it'll solve for the equation

Ix=H×Iy

Hence, it solves for the matrix *H*

Well, to estimate the homography is a simple task. If you are using opencv, it's a two line code. However , I'd recommend that one implements oneself.

H, __ = cv2.findHomography(srcPoints, dstPoints, cv2.RANSAC, 4)

Viola ! Our homography matrix looks something like this …

$\left|\left[h11h21h31h12h22h32h13h23h33 \right]\right|$

Anyway, putting the pretty graphics aside, understand what the homography matrix is . Homography preserves the straight lines in an image. Hence the only possible transformations possible are translations, affines, etc. For example, for an affine transform,

$$[h31h32h33]=[0\ 0\ 1]$$

Also, you can play around with $h13$ and $h23$

for translation

STEP 4 : WARPING & STITCHING

To understand stitching, I'd like to recommend Adrian Rosebrock's blog post on OpenCV Panorama stitching. His blog provides a wonderful explanation as to how to proceed with image stitching and panorama construction using 2 images.So , once we have established a homography, i.e. we know how the second image (let's say the image to the right) will look from the current image's perspective, we need to transform it into a new space. This transformation mimics the phenomenon that we undergo. That is, the slightly distorted, and altered image that we see from our periphery . This process is called warping. We are converting an image, based on a new transformation. In this case, Im using a planar warping. What I'm doing, is essentially change the plane of my field of view. Whereas, the "panorama apps" use something called as a Cylindrical and spherical warps !

Types of warping :

- *Planar : wherein every image is an element of a plane surface, subject to translation and rotations …
- *Cylindrical : wherein every image is represented as if the coordinate system was cylindrical. and the image was plotted on the curved surface of the cylinder.
- *Spherical : the above appends, instead of a cylinder, the reference model is a sphere.

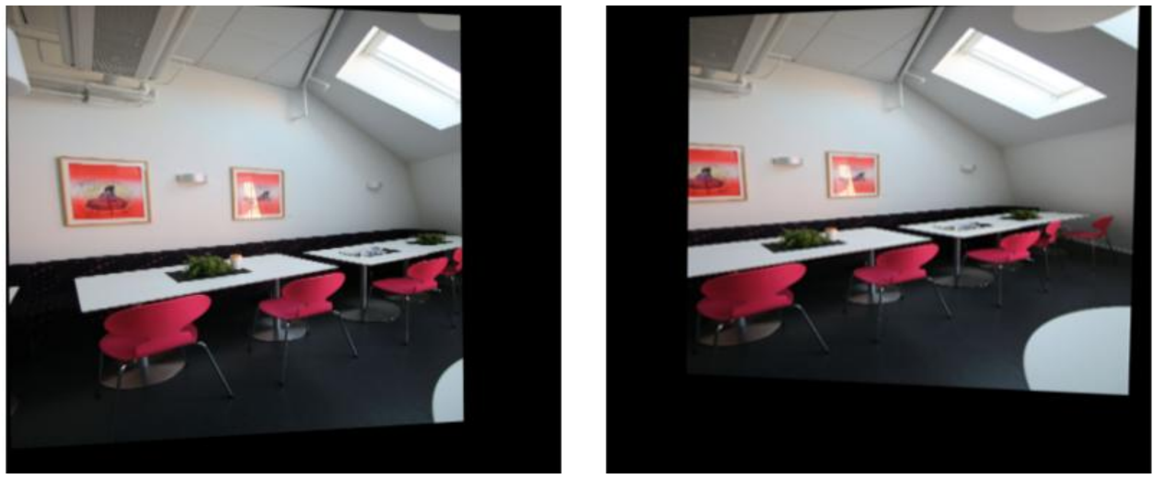
Each model has its' own application. For the purposes of this tutorial, I'll stick to planar homography and warping.

So , to warp, essentially change the field of view, we apply the homography matrix to the image.

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

Here is some visualization …. below are left warped and right warped images Note the orientation and projective-ness of each image.

International Journal of Engineering, Science and Mathematics
Vol. 8 Issue 4, April 2019,
ISSN: 2320-0294 Impact Factor: 6.765
Journal Homepage: http://www.ijesm.co.in, Email: ijesmj@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gage as well as in Cabell's Directories of Publishing Opportunities, U.S.A


Warping Images of Lunchroom !

Stitching 'em up !

Once, we have obtained a warped image, we simply add the warped image along with the second image. Repeat this over through leftward stitching and rightward stitching, and viola! We have our output.

I'll get a bit deeper as to how to perform the image joining part. Say, we have a homography matrix H

. If the starting coordinate of each image is (0,0) and end point is (re,ce) , we can get the new warped image dimension by $startp=H\times[0,0]$ uptill $endp=H\times[re,ce]$. **Note : If start_pt comes out to be negative** , account for a translational shift. i.e. "perform translation shift to the image by $|startp|$

". Also make sure that the homography matrix normalized such that the last row amounts to a unit vector.

You can checkout the above mentioned explanation below This is the implementation snippet from the actual code. Please look at the full code to understand in accordance with the post.

```
def leftstitch(self):
    # self.left_list = reversed(self.left_list)
    a = self.left_list[0]
    for b in self.left_list[1:]:
        H = self.matcher_obj.match(a, b, 'left')
        print "Homography is : ", H
        xh = np.linalg.inv(H)
        print "Inverse Homography :", xh
        # start_p is denoted by f1
        f1 = np.dot(xh, np.array([0,0,1]))
        f1 = f1/f1[-1]
        # transforming the matrix
        xh[0][-1] += abs(f1[0])
        xh[1][-1] += abs(f1[1])
        ds = np.dot(xh, np.array([a.shape[1], a.shape[0], 1]))
        offsety = abs(int(f1[1]))
        offsetx = abs(int(f1[0]))
        # dimension of warped image
        dsize = (int(ds[0])+offsetx, int(ds[1]) + offsety)
        print "image dsize =>", dsize
        tmp = cv2.warpPerspective(a, xh, dsize)
        # cv2.imshow("warped", tmp)
        # cv2.waitKey()
        tmp[offsety:b.shape[0]+offsety, offsetx:b.shape[1]+offsetx] = b
        a = tmp
```

Another method : There is another method, i.e. using basic for looping constructs and overlay the two images. The logic is simple. Input to the method will be the steadyimage and warpedImage. Iterate through both images, and if pixels are equal, put pixel as that value. else give preference to a non black pixel ..

```
def mix_match(self, leftImage, warpedImage)
    i1y, i1x = leftImage.shape[:2]
    i2y, i2x = warpedImage.shape[:2]


    for i in range(0, i1x):
        for j in range(0, i1y):
            try:
                if(np.array_equal(leftImage[j,i],np.array([0,0,0])) and  \
                    np.array_equal(warpedImage[j,i],np.array([0,0,0]))):
                    # print "BLACK"
                    # instead of just putting it with black,
                    # take average of all nearby values and avg it.
                    warpedImage[j,i] = [0, 0, 0]
                else:
                    if(np.array_equal(warpedImage[j,i],[0,0,0])):
                        # print "PIXEL"
                        warpedImage[j,i] = leftImage[j,i]
                    else:
                        if not np.array_equal(leftImage[j,i], [0,0,0]):
```

```
                    bl,gl,rl = leftImage[j,i]
                    warpedImage[j, i] = [bl,gl,rl]
            except:
                pass
    # cv2.imshow("waRPED mix", warpedImage)
    # cv2.waitKey()
    return warpedImage
```

But this method will iterate over soooo many pixels. .. It's very slow, for two reasons. Firstly , it involves heavy iteration. And, well, I'd personally execute such heavy loops in C++ and not python.

So, basically, this is how my main function looks … the heart and core of all implementations

```
if __name__ == '__main__':
    try:
        args = sys.argv[1]
    except:
        args = "txtlists/files1.txt"
    finally:
        print "Parameters : ", args
    s = Stitch(args)
    s.leftshift()
    # s.showImage('left')
    s.rightshift()
    print "done"
    cv2.imwrite("test.jpg", s.leftImage)
    print "image written"
    cv2.destroyAllWindows()
```

International Journal of Engineering, Science and Mathematics
Vol. 8 Issue 4, April 2019,
ISSN: 2320-0294 Impact Factor: 6.765
Journal Homepage: http://www.ijesm.co.in, Email: ijesmj@gmail.com Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gage as well as in Cabell's Directories of Publishing Opportunities, U.S.A

CONCLUSION

Image stitching enables the combination of multiple shots to create a larger picture that is beyond the normal aspect ratio and resolution (super resolution) of the camera's individual shots. The technology enables positioning for dramatically wide shots without duplicated objects or distortion. The most familiar use of image stitching is in the creation of panoramic photographs, often used for landscapes. Wide-angle and super-resolution images created by image stitching are used in artistic photography, medical imaging, high-resolution photo mosaics, satellite photography and more. For best results, image stitching requires that shots have quite precise overlaps and identical exposure settings. Algorithms are required for compositing surface creation, pixel alignment, image alignment and distinctive feature recognition to aid as reference points to software for alignment accuracy. Image stitching is somewhat analogous to the way the human brain performs the task of assimilating the two monocular fields of vision (FOV) from each eye into a single, wider FOV with about 114 degrees of depth perception. The brain's natural image stitching enables humans' enhanced stereoscopic, binocular vision, surrounded by a further 220 degrees of monocular peripheral vision.

REFERENCES

1. *Solomon, C.J.; Breckon, T.P. (2010). Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab. Wiley-Blackwell. doi:10.1002/9780470689776. ISBN 978-0470844731.*
2. *Wilhelm Burger; Mark J. Burge (2007). Digital Image Processing: An Algorithmic Approach Using Java. Springer. ISBN 978-1-84628-379-6.*
3. *R. Fisher; K Dawson-Howe; A. Fitzgibbon; C. Robertson; E. Trucco (2005). Dictionary of Computer Vision and Image Processing. John Wiley. ISBN 978-0-470-01526-1.*
4. *Rafael C. Gonzalez; Richard E. Woods; Steven L. Eddins (2004). Digital Image Processing using MATLAB. Pearson Education. ISBN 978-81-7758-898-9.*
5. *Tim Morris (2004). Computer Vision and Image Processing. Palgrave Macmillan. ISBN 978-0-333-99451-1.*
6. *Milan Sonka; Vaclav Hlavac; Roger Boyle (1999). Image Processing, Analysis, and Machine Vision. PWS Publishing. ISBN 978-0-534-95393-5.*
7. *Alhadidi, Basim; Zu'bi, Mohammad H.; Suleiman, Hussam N. (2007). "Mammogram Breast Cancer Image Detection Using Image Processing Functions". Information Technology Journal.* ***6*** *(2): 217–221. doi:10.3923/itj.2007.217.221.*
8. *Mann, Steve; Picard, R. W. (November 13–16, 1994). "Virtual bellows: constructing high-quality stills from video" (PDF). Proceedings of the IEEE First International Conference on Image Processing. IEEE International Conference. Austin, Texas: IEEE.*

9. *Ward, Greg (2006). "Hiding seams in high dynamic range panoramas". Proceedings of the 3rd Symposium on Applied Perception in Graphics and Visualization. ACM International Conference.* ***153****. ACM. doi:10.1145/1140491.1140527. ISBN 1-59593-429-4.*
10. Steve Mann. "Compositing Multiple Pictures of the Same Scene", Proceedings of the 46th Annual Imaging Science & Technology Conference, May 9–14, Cambridge, Massachusetts, 1993
11. S. Mann, C. Manders, and J. Fung, "The Lightspace Change Constraint Equation (LCCE) with practical application to estimation of the projectivity+gain transformation between multiple pictures of the same subject matter" IEEE International Conference on Acoustics, Speech, and Signal Processing, 6–10 April 2003, pp III - 481-4 vol.3
12. *Breszcz, M.; Breckon, T. P. (August 2015). "Real-time Construction and Visualization of Drift-Free Video Mosaics from Unconstrained Camera Motion" (PDF). IET J. Engineering. IET.* ***2015*** *(16): 1–12. doi:10.1049/joe.2015.0016. breszcz15mosaic.*
13. *Szeliski, Richard (2005). "Image Alignment and Stitching" (PDF). Retrieved 2008-06-01.*
14. *S. Suen; E. Lam; K. Wong (2007). "Photographic stitching with optimized object and color matching based on image derivatives". Optics Express.* ***15*** *(12): 7689–7696. doi:10.1364/OE.15.007689. PMID 19547097.*
15. *d'Angelo, Pablo (2007). "Radiometric alignment and vignetting calibration" (PDF).*
16. *Wells, Sarah; et al. (2007). "IATH Best Practices Guide to Digital Panoramic Photography". Retrieved 2008-06-01.*
17. Hugin.sourceforge.net, hugin manual: Panini
18. Groups.google.com, hugin-ptx mailing list, December 29, 2008
19. Tawbaware.com, PTAssembler projections: Hybrid