ELEC 474 Machine Vision Project

Auto stitch

Paul Li: 20105153

Date: Nov 30th, 2021

Function: LoadFolder	original
Class: Mapping	original
Function: KpsAndDps	original
Function: LoweRatioTest	original
Function: Matching	original
Function: MatchFeatures	original
Function: OpenFile	original
Function: Plot	original
Function: FindDirection	original
Function: EstimateTransformation	original
Function: Merge	original
Function: Wrap	original
Function: DetectObject	original

Statement of Originality

My signature below attest that this submission is my original work.
Following professional engineering practice, I bear the burden of proof for original work. I have read the Policy on Academic Integrity posted on the Faculty of Engineering and Applied Science web site (http://engineering.queensu.ca/policy/Honesty.html) and confirm that this work is in accordance with the Policy.

Individual:

Signature: ____ Paul Li _____ Date: Nov 30th, 2021

Step 1

```
def MatchFeatures(path):
    index = 0
    new_list = []
    new_match = []
    points = []
    pic = OpenFile(path)
    for pic in pic:
        index += 1
        temp = LoadFolder(path, pic)
        new_list.append(temp)
        points.append(KpsAndDps(temp,check=1, dps = dps))
    seeds = 0
    pic_seeds = new_list[seeds]
    points_seeds = points[seeds]
    result = list(range(len(new_list)))
    result.remove(seeds)
    print("Matching the pictures")
    for a in result:
        temp = Matching(points_seeds, points[a], loweRatio)
        if len(temp.matches) > threshold_value:
            temp.Loader(pic_seeds)
            temp.Loader2(new_list[a])
            new_match.append(temp)
    return new_match
```

SIFT was utilised to construct the image's key points and descriptors for the feature extraction procedure. The Lowe's Ratio Test is used to eliminate a number of false positives from the image collection that was initially found. If the image's match value is greater than the threshold value (200) for the matching metric, it is a good image match. In terms of data structures, the matching information (key points and descriptors) for each pair of images has been recorded in a class called Mapping. This class has the ability to containerize data and match behaviour that acts on that data in logical groups.

Step 2

```
def EstimateTransformation(pic1, pic2):
    temp1 = KpsAndDps(pic1,check=0)
    temp2 = KpsAndDps(pic2,check=0)
    match = Matching(temp1, temp2, loweRatio)
    m2 = temp2[0]
    m1 = temp1[0]
    matches = match.matches
    p2 = np.float32([m2[m[0].trainIdx].pt for m in matches]).reshape(-1,1,2)
    p1 = np.float32([m1[m[0].queryIdx].pt for m in matches]).reshape(-1,1,2)
    homo, select = cv.findHomography(p2, p1, cv.RANSAC, 4)
    return homo
```

The matching pairs were discovered using the RANSAC with parameter four. First, the images were divided into right and left halves. The two halves will then use computed homography to perform the viewpoint transform. The homography was to find the metric and compare the inverted result to zeros; if it is higher than zero, it goes to the left; if it is less than zero, it goes to the right.

Step 3

```
def Wrap(pic, pic2):
    temp1, temp2 = pic.shape[:2]
    for a in range(0, temp2):
        for b in range(0, temp1):
            try:
                if(np.array_equal(pic[b,a],np.array([0,0,0])) and  np.array_equal(pic2[b,a],np.array([0,0,0]))):
                    pic2[b,a] = [0, 0, 0]
                else:
                    if(np.array_equal(pic2[b,a],[0,0,0])):
                        pic2[b,a] = pic[b,a]
                    else:
                        if not np.array_equal(pic[b,a], [0,0,0]):
                            blue,green,red = pic[b,a]
                            pic2[b,a] = [blue,green,red]
            except:
                pass
    return pic2
```

I combined the left parts first, then the right halves, using the geometric transformation as the transformation. When combining the images, I utilise the wrap as the image composition first, then combine the entire set of matching images.

Further details of software

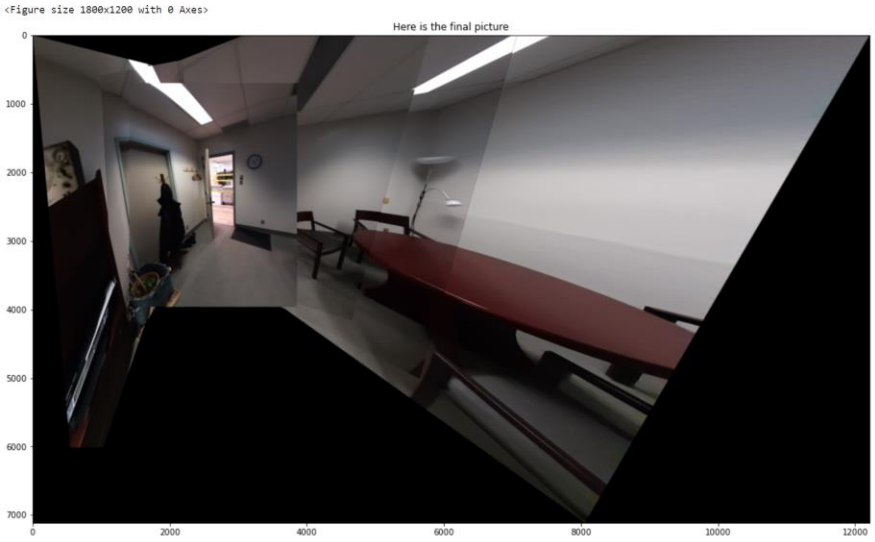
I utilised several of the Numpy library's handy methods, such as "np.linalg.inv" to do the invert and "np.array equal" to discover the matched pictures in the wrap. This Numpy module helped me save a lot of time and improve the efficiency of my code.

Test description

```
Find the file!
Matching the pictures
  Metric value:  251
  Metric value:  385
  Metric value:  740
  Metric value:  845
  Metric value:  467
  Metric value:  220
```

The above is a listing of the matching metric values between each image, for each data set.

The tests results are the two pictures stich and up to six pictures stich. It will take roughly 30 minutes to generate the final result.



Above is my final result for the office set of the pictures.

Correctness and effectiveness

My code can execute a rudimentary stitch, which matches and merges a collection of random photos into a single image with the correct placement and angles. As a result, the majority of my code is right. However, there are several roadblocks and limits in my code that need to be addressed in order for the end result to be more visually appealing. Straight lines are curled outwards from the centre with Pincushion Distortion. As a result, the orientation is slightly wrong for all of the matching, which adds up to a more Pincushion look and a more virtual orientation. Another problem is that the algorithm for combining photographs was not particularly good, thus the programme takes a long time to run, about thirty minutes. The combination also takes up a lot of memory, therefore the code can only combine six photos at a time. As a result, the entire office photos cannot be auto stitched at the same time. The image also has some dark edges and has lost some detail.

Future improvement

The mechanism for extracting features will be improved first. I don't have time to test the ORB, SIFT, SURF, or a combination of them due to time constraints. Because it is the method with which I am most familiar, I only employed the SIFT method. If I have more time, I'll research the differences between those methods and choose the best one for my project.

Because my code can only merge images horizontally, I need to figure out a strategy and method that will allow the auto stitches to merge vertical images as well. In order to fix the photos in the future and offset the Pincushion Distortion, an algorithm will need to be built.

Finally, when matching the photographs, a more effective method would be to only match the main area of the image and not the background. This will reduce the image size, allowing the code to run faster and merge more images.