Imgage Stitching

Amandeep Singh (18010129)

26th May 2020

(Using OpenCV)

1 How to Run the Code

(CODE IN PYTHON LANG)

Just simply unzip the folder and open the folder in terminal and compile and run the shell file using the command ***'bash runme.sh'*** .....thats all everything will be done automatically if you dont have any required packages it will all get downloaded automatically.
Even if you dont have python it will get downloaded from shell command.
NOTE: You must be connected to the internet because all the nescessary files will be downloaded from there.

2 Introduction

What is Image Stitching?
Image Stitching is the process of modifying the perspective of images and blending them, so that the photographs can be aligned seamlessly. In simpler words we can say that combining two or more than two images which has some similarities (i.e. some common part) into one Image. Or reconstructing an image from two or more images which shares a common part.

3 Working

It includes :
1. Keypoint detection
2. Local invariant descriptors (SIFT, SURF, etc)
3. Feature matching
4. Homography estimation using RANSAC
5. Perspective warping

3.1 Feature Detection and Extraction

Moreover, our solution has to be robust even if the pictures have differences in one or more of the following aspects:
1, Scaling
2. Angle
3. Spacial position
4. Capturing devices

The first step in that direction is to extract some key points and features of interest. These features, however, need to have some special properties.

3.2 Keypoints and Descriptors

I have used SIFT to address the limitations of corner detection algorithms. Usually, corner detector algorithms use a fixed size kernel to detect regions of interest (corners) on images. It is easy to see that when we scale an image, this kernel might become too small or too big.
To address this limitation, SIFT uses Difference of Gaussians . It also uses the neighboring pixel information to find and refine key points and corresponding descriptors.
To start, we need to load 2 images, a query image, and a training image. Initially, we begin by extracting key points and descriptors from both. We can do it in one step by using the OpenCV detectAndCompute() function. Note that in order to use detectAndCompute() we need an instance of a keypoint detector and descriptor object. It can be ORB, SIFT or SURF, etc(I have used SIFT). Also, before feeding the images to detectAndCompute() we convert them to grayscale.
We run detectAndCompute() on both, the query and the train image. At this point, we have a set of key points and descriptors for both images. If we use SIFT as the feature extractor, it returns a 128-dimensional feature vector for each key point. If SURF is chosen, we get a 64-dimensional feature vector.

3.3 Feature Matching

As we can see, we have a large number of features from both images. Now, we would like to compare the 2 sets of features and stick with the pairs that show more similarity.
With OpenCV, feature matching requires a Matcher object. Here, we explore two flavors:

1. Brute Force Matcher
2. KNN (k-Nearest Neighbors)
3. Flann Based Matcher

The BruteForce (BF) Matcher does exactly what its name suggests. Given

2 sets of features (from image A and image B), each feature from set A is compared against all features from set B. By default, BF Matcher computes the Euclidean distance between two points. Thus, for every feature in set A, it returns the closest feature from set B. For SIFT and SURF OpenCV recommends using Euclidean distance. For other feature extractors like ORB and BRISK, Hamming distance is suggested.
To create a BruteForce Matcher using OpenCV we only need to specify 2 parameters. The first is the distance metric. The second is the crossCheck boolean parameter.

3.4 Ratio Testing

To make sure the features returned by KNN are well comparable, the authors of the SIFT paper, suggests a technique called ratio test. Basically, we iterate over each of the pairs returned by KNN and perform a distance test. For each pair of features (f1, f2), if the distance between f1 and f2 is within a certain ratio, we keep it, otherwise, we throw it away. Also, the ratio value must be chosen manually.
In essence, ratio testing does the same job as the cross-checking option from the BruteForce Matcher. Both, ensure a pair of detected features are indeed close enough to be considered similar.

4 Goal

Our goal of this final project is to implement a featured based automatic image stitching demo. When we input two images with overlapped fields, we expect to obtain a wide seamless panorama. In this final project, we use scale invariant features transform(SIFT) to extract local features of the input images. Then we use K nearest neighbors algorithms to match these features.

5 The Code/Algo

import cv2
import numpy as np
dim=(1024,768)
left=cv2.imread('left.jpeg',cv2.IMREAD$_COLOR$)
$left = cv2.resize(left, dim, interpolation = cv2.INTER_AREA)$
$right = cv2.imread('right.jpeg', cv2.IMREAD_COLOR)$
$right = cv2.resize(right, dim, interpolation = cv2.INTER_AREA)$

sift=cv2.xfeatures2d.SIFT$_create()$

kp1, des1=sift.detectAndCompute(left,None)
kp2, des2=sift.detectAndCompute(right,None)

cv2.imshow('left$_k$eypoints', $cv2.drawKeypoints(left, kp1, None)$)
$cv2.imshow('right_keypoints', cv2.drawKeypoints(right, kp2, None))$
$FLANN_INDEX_KDTREE = 0$
$index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)$
$search_params = dict(checks = 50)$
$match = cv2.FlannBasedMatcher(index_params, search_params)$

match=cv2.BFMatcher()
matches=match.knnMatch(des1,des2,k=2)
good=[]
for m,n in matches:
if m.distance ¡ 0.1*n.distance:
good.append(m)

draw$_params = dict(matchColor = (0, 255, 0), singlePointColor = None, flags =$
2)
$im3 = cv2.drawMatches(left, kp1, right, kp2, good, None, **draw_params)$

images=[]
images.append(left)
images.append(right)

stitcher=cv2.Stitcher.create()
ret,pano=stitcher.stitch(images)

if ret==cv2.STITCHER$_OK$:
$cv2.imshow('left_img', left)$
$cv2.imshow('right_img', right)$
$cv2.imshow('matching_params', im3)$
$cv2.imshow('stitched_img', pano)$
$cv2.imwrite('original_image_drawmatches.jpeg', im3)$
$cv2.imwrite("output.jpeg", pano)$
$cv2.waitKey(20000)$
$cv2.destroyAllWindows()$
$else$:
$print("ErrorDuringStitching!")$

NOTE : The code is explained in the program itself using comments

THANK YOU!