Image Stitching

Laboni Sarker

Introduction

Image stitching is the process of creating a panaromic image from a given set of images that have some overlapping region in them. Though it sounds like an easy task, there are several steps to reach the final destination. Especially, bringing multiple interleaving images to a common plane to merge them properly and smoothly is a very challenging task. For achieving this task I have followed the steps :

- Designing and implementing algorithms for stitching two images using openCV modules,
- Replace the openCV modules with my implemented warp, RANSAC, homography calculation functions,
- Improvement of warp function to perform better,
- Looping over images to stitch multiple images,
- Improving the stitching by taking the middle image as base.

Data

I have used the datasets provided with the assignment (intersection, ucsb4, west-campus, etc). These folder contains images captured by rotating a camera (the degree of rotation varies). As a beginner, I have started implementation using the **intersection** image set which has less rotational degree. Then I have also tried to see the result in other test cases.

Chanllenges & solutions

For calculation of **homography**, **RANdom SAmple Consensus(RANSAC)** and **warp** functions I have used the traditional algorthims and implemented them. One of the main challenges I faced while implementing the warp function is that when I have tried to interpolate new points from the source image, in some places there are some black lines. I suppose this is because while warping we get the transformed pixels in fraction and converting them to one single integer position results the black lines. Next, I tried to solve this problem by taking round, floor and ceiling of the (x,y) position and filled that up. Finally, I have handled the problem by using [Algorithm 3].
While using two images to stitch, it performed much better than I tried with multiple images. For my first attempt at multiple image stitching, I took the right most image as the base image and then kept stitching images one after another. But the problem was when the degree of rotation of the captured images was too high, the last attached images get zoomed out and the quality of the image gets distorted. For that reason, I have tried stitching images pair wise and then finally stitching them [Algorithm 1]. Moreover, I have taken the middle image as base and tried doing left hand, right hand stitching taking the middle one as the base [Algorithm 2].

Algorithm 1 Image Stitch 1

```
Require: $images$
  $iteration \leftarrow no\_of\_images/2$
  while $iteration \geq 2$ do
    $imageNew \leftarrow []$
    for each consecutive pair $(im1, im2)$ in $images$ do
      $stitched \leftarrow$ stitch$(im1, im2)$
      $imageNew$.append$(stitched)$
    end for
    $images \leftarrow imageNew$
    $iteration \leftarrow iteration/2$
  end while
  return $images[0]$ //final stitched image
```

Libraries used

Python openCV is very rich in image stitching related modules. While digging through the problem, I have found a lot of resources for image

Algorithm 2 Image Stitch 2

```
Require: $images$
  $mid \leftarrow no\_of\_images/2$
  for image ($leftmostimage$ to $mid$) in $images$ do
    $lStitched \leftarrow$ Leftstitch($image, mid$)
  end for
  for image ($mid$ to $rightmostImage$) in $images$ do
    $rStitched \leftarrow$ Rightstitch($mid, image$)
  end for
  $finalImage \leftarrow lStitched + rStitched$
  return $finalimage$
```

Algorithm 3 Warp1

```
Require: $Originalimage, homography$
  $cornerX \leftarrow$ transformed original $cornerX$
  $cornerY \leftarrow$ transformed original $cornerY$
  $minX, maxX \leftarrow$ min and max of $cornerX$
  $minY, maxY \leftarrow$ min and max of $cornerY$
  for $i$ in range($minX, maxX$): do
    for $j$ in range($minY, maxY$): do
      find the position in the original image using $inverseHomography$
      $newImage[i][j] \leftarrow$ pixel value from original image position
    end for
  end for
  for image ($mid$ to $rightmostImage$) in $images$ do
    $stitched \leftarrow$ Rightstitch($mid, image$)
  end for
  $finalImage \leftarrow Leftstitch + Rightstitch$
  return $finalimage$
```

stitching just using the openCV functions. But for my assignment, I have only used cv2.imwrite(), cv2.imread(), cv2.xfeatures2d.SIFTcreate(), cv2.DescriptorMatchercreate() (And some others for cropping images). So, basically openCV is used for matching the points in between the images I want to stitch and for reading, writing images.

Implementation Details

How to compile and run the python file is in detail in the README file in gitlink.

Result

For checking the credibility of my own implemented functions, at first I have run my algorithm with openCV functions for homography (cv2.findHomography()) and warping (cv2.warpPerspective()). Though the homography matrix does not exactly match with my version, the overall impact is same. But the warping function is much more robust in the openCV than my initial warp function (black lines are produced). However, using the second and third version of warp function, I have been able to overcome that issue. If I compare the resultant image from my implementation with the original one which is provided with the dataset, the given one is visually pleasant to watch. On the contrary, though my algorithm stitches the images properly but it is visually understandable that the images have been stitched.

Discussion

While working on this assignment, I have faced a lot of challenges and while trying to solve those, I have learnt a lot of things. Moreover, before implementation, I have had some misconceptions which have got cleared gradually. Like, I thought if I could finish stitching two images, it would just be a simple iterative approach to join multiple images. But this needs much more than that (tuning to stitch from left and right side). Just warping with respect to perspective is not enough. To make an image visually beautiful, it needs other works like color blending, smoothing (which I haven't implemented in my work). Another interesting thing to observe is that each time I run the program, the homography matrix gets changed I believe this is because of the sampling of the RANSAC algorithm.