Project Report for CV Project

Devansh Bansal
B20CS094
bansal.11@iitj.ac.in

Abstract

In this report, I am presenting some basic techniques used for Image Matching, like Harris Corner Detection, Scale Invariant Feature Detection (SIFT); along with some modifications and additions. I have collected images from internet, and ran them through above techniques, and evaluated each technique based on the good matchings, bad matchings, distance (defined later in report). Evaluation results show that my additions to these technique enhance the performance and hence can be used as an add-on applied after the main technique. Also, I performed Image Stitching, using various key-points detectors and various procesudres of stitching. Results of image stitching vary quite-a-lot depending upon key-point detection algorithm, as well as the method used for stitching, as we'll see in the following report.

1. Introduction

Image matching is the process of finding correspondences between two or more images. It is a fundamental problem in computer vision with applications in various fields such as robotics, augmented reality, and satellite imaging. The goal of image matching is to identify common features or points in two or more images, and then use these extracted features or points (called key-points) to align the images or extract relevant information from them. There are various techniques developed for Image Matching, like Harris Corner-Detection, SIFT, SURF, ORB, FAST etc. each having a stronghold in particular domain, and weakness in others. Hence, the choice of technique to be used depends majorly on the task-in-hand. In this report, I am presenting review and analysis of two of the above-mentioned techniques, viz. Harris Corner Detection and SIFT, along with some additions in the steps to achieve better performance. After detecting those key-points, to match them across two images requires different matching algorithm, like Brute-Force Matcher, or Nearest-Neighbor Matcher, or RANSAC. In this report, I have used Brute-Force Matcher and RANSAC. Finally, I have evaluated and describe the results, indicating which technique can be used for better results, and then I am briefly discussing my addition to these techniques.

Image Stitching is a practical use-case of Image Matching (key-point detection and description, to be more precise), in which we intend to create a panorama, from multiple images given that the images have some overlap between them so that key-points can be matched (i.e. the overlapped part of the images can be matched in both images) and then the images can be blended together. Image Stitching is used in various domains like photography, surveillance, and remote sensing applications. Image Stitching is also used in virtual reality and augmented reality applications, where multiple images are combined to create a seamless 360-degree view.

1.1. Contributions

I am doing this project single-handedly, so all of its work, analysis, inferences, results are done by me.

1.2. Harris Corner Detection

Harris Corner Detection is an algorithm used in Computer Vision to detect key-points (here, corners) in an image. Corners are defined as the points where gradient is changing rapidly in more than one direction, and hence they serve as a useful set of key-points. Since, this algorithm is just based on gradient of the image (along both directions), it is sensitive to image noise; like too much variation in brightness or contrast in the original image (and not the noisy image).

As we will see later, Harris Corner Detection works for specific kind of images only, and hence can't be termed as a generic Image Matching Algorithm.

1.3. Scale-Invariant Feature Transform (SIFT)

SIFT was designed to identify distinctive and robust features in an image. These key-points can be edges, corners, or blobs also. These key-points perform quite better as compared to Harris Corner Detection, even though some key-points of SIFT are just corners detected by Harris Corner algorithm. The difference lies in the descriptors gener-

ated by SIFT corresponding to each key-point. These descriptors are generated based on magnitude and direction of the local image gradient; and that too on different scales, which makes the key-points and descriptors scale-invariant and hence they can be matched easily along different images of different scales.

1.4. Binary Robust Invariant Scalable Key-Points (BRISK)

BRISK is designed to extract key-points and corresponding descriptors from images in a fast and efficient manner. The concept of key-points and descriptors remain the same as in SIFT, but here the descriptors are binary, which makes them compact and easy to compare. BRISK is also built to withstand variations in illumination, rotation, and size. It does this by using a scale-space pyramid and smoothing and thresholding the picture at each level. Scale-Space Pyramid is just a term for the procedure mentioned in above section for SIFT, that descriptors are generated on different scales which helps in identifying key-points and their descriptors, which are scale invariant. BRISK has a limitation of performance when the images have repeated patterns, and to overcome this, some variants of BRISK are made, like FREAK and ORB.

1.5. Oriented FAST and Rotated BRIEF (ORB)

ORB was designed to identify distinctive and robust features in an image, irrespective of illumination, rotation or scale. ORB uses FAST key-points and BRIEF descriptors, and it also incorporates orientation information to improve its performance against rotations. ORB also uses multi-scale approach (or scale-space pyramid) just like SIFT and BRISK. But the major advantage of using ORB is its computational efficiency, as it can extract features quite fast (since based on FAST (another key-point detection algorithm, known for its faster execution) key-points) while maintaining good performance (since based on BRIEF descriptors). Owing to the robustness of ORB, it is highly suitable for its usage in real-time scenarios like augmented reality, virtual reality (3D videos) or even Image Stitching.

2. Work and Progress

I implemented the algorithm of Harris Corner Detection from scratch, whereas I used SIFT from the library OpenCV. I used both Harris Corner and SIFT as key-point detector and used SIFT descriptors for image matching and compared the results of image matching. Then I performed Image Stitching using various key-point detectors and descriptors like SIFT, BRISK and ORB. I stitched the images in two different ways, one with stitching from left side and the other with stitching from right side. Comparison of these different key-point detectors-and-desciptors as well as the method is done based on visual output. Then I implemented my modification to improve the performance of Harris Corner Detection, and compared the with and without modification Harris Corner Detection. I also matched and compared the results of image matching between improved Harris Corner and SIFT.

2.1. Scratch Implementation of Harris Corner Detection

Following steps were followed to implement Harris Corner Detection Algorithm:

- Converted RGB image to grayscale image
- Computed gradient of grayscale image, along both directions
- Calculated the Harris Corner Response Function
- Identified Corners

For computing the gradient of grayscale image, I used Sobel Filter along X-direction and along Y-direction, and then I convolved the image with these filters to get the gradient computed.

2.2. Finding Key-Points

Depending on the magnitude of gradient along each direction and value of Harris Corner Response Function, I identified corners in the given image. I defined a value as the ratio of Harris Corner Response Function of the particular pixel and the maximum value of Harris Corner Response Function in the image. If this value is more than a particular threshold set for a particular scenario, then that pixel is termed as Harris Corner. Whereas for key-points detected using SIFT, it was the inbuilt functionality and definition of a pixel being a key-point or not.

2.3. Computing Descriptors

I generated SIFT descriptors for both sets of key-points. Using the library function of OpenCV, I computed the descriptors of key-points detected by Harris Corner (corners) as well as those for key-points detected by SIFT in the previous step.

2.4. Matching of Key-Points

I used the Brute-Force Matcher of OpenCV, which takes descriptors as the input and then create a mapping between corresponding descriptors. Each match is mainly characterized by its parameter 'distance', lower the distance, better the match between corresponding key-points.

I used top-50 matches (50 matches with the least distance values) and termed them as 'good matches'. Then, using the function 'drawMatches', I plotted the matched key-points

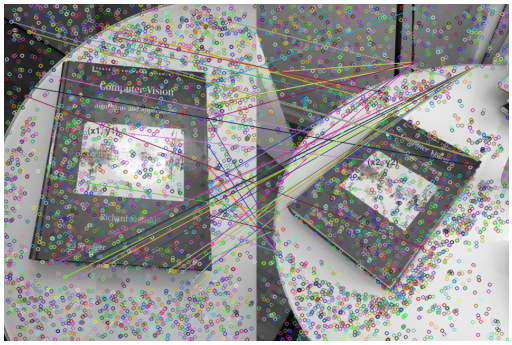

Figure 1. Key-Points and Mappings for Harris Corner Key-Points. Two images are stitched next to each other and colored circles are key-points, whereas lines matching two colored circles is the corresponding mapping.

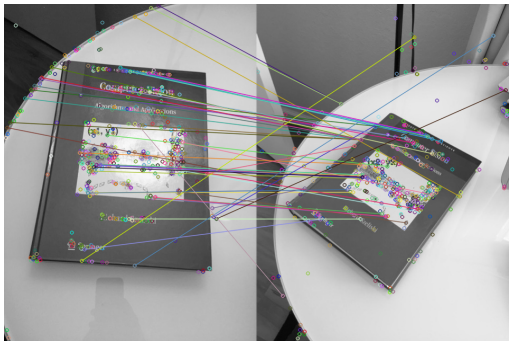

Figure 2. Key-Points and Mappings for SIFT Key-Points. Two images are stitched next to each other and colored circles are key-points, whereas lines matching two colored circles is the corresponding mapping.

and generated a result image. Figure 1 shows the result image for key-points obtained using Harris Corner Detection (i.e. corners as key-points), and Figure 2 shows the result image for key-points obtained using SIFT.

2.5. Comparison

One may think that distance is the effective metric of comparison, as it's the score of similarity of two key-points; but the catch here is that erroneous key-point detection can lead to meaningless key-points being detected, and which might give us less value of distance (or higher similarity score).

Similar thing is observed here when we compare Key-Points detected by Harris Corner Detection Algorithm and by SIFT Algorithm. Among the top-50 matches, as described above, Harris key-points has a minimum value of 27.55 and a maximum value of 55.21 as the distance; whereas for SIFT, these values are 55.17 and 122.48 respectively. We can clearly see that SIFT key-points are performing way better than Harris key-points from Figure 1 and 2.

Here a question arises that if the distance is so low, then why is the matching so poor? It can be understood from the basic working of Harris Corner Detection, which is highly based on the image gradient, which can be similar across the whole image for a number of unrelated pixels, just by the nature of the image. Any two points can have almost the same gradient even though they are absolutely unrelated. Hence, this also gives us a hint why corners are not treated as key-points in general. Moreover, the image is converted to grayscale format, which reduces the chances of variation in gradient as compared to a colored image.

To actually compare the two methodologies, we need to use a metric related to inliers and outliers after applying RANSAC to find the homography, which links the two images. The exact metric is percentage of inliers (i.e. 100*(no. of inliers)/(no. of matches)). For Harris Corner key-points, only 28% of the matches are rermed as inliers, whereas for SIFT key-points, this value is as high as 54%. Now it is clear that SIFT key-points are really performing well and that too their performance is almost twice as good as Harris Corner key-points.

2.6. Image Stitching

To implement Image Stitching, I took the input of the images and used different key-point detectors and descriptors like SIFT, BRISK and ORB to first find the key-points and then their descriptors. After getting the descriptors of both the images, I used Nearest Neighbor Matcher with number of neighbors as 2, to match the key-points based on the descriptors. To choose a particular set of matches that can be proved to be useful as well as meaningful, I performed the thresholding test, in which ratio of distance of nearest neighbor to the distance of second-nearest neighbor is considered as a measure to determine the usefulness and meaningfulness of the key-point. If the ratio comes out to be less than a particular threshold, then that key-point match is considered as a Good Match, and we use them for further procedure, rather than the whole set of matches.

After having the good matches, we want to blend both the images into one another so that they can be stitched together and can be seen as a single image. To perform this blending, I created an empty matrix with number of rows as the maximum of rows in both images and number of columns as the summation of number of columns in both the images. This new empty matrix will serve as the result of stitching the two images in consideration.

Then I computed the homography matrix based on the key-points of good matches, along with the usage of RANSAC. Then I performed a perspective transform of the second image on the first image (due to the way homography matrix was computed, i.e. source points are of the second image and destination points are of the first image) with the help of the homography matrix computed just in

Figure 3. Stitched Image of Image 1 and 2, along with Image 3 stitched together, using SIFT and Left-To-Right Method

Figure 4. Stitched Image of Image 1 and 2, along with Image 3 stitched together, using BRISK and Left-To-Right Method

Figure 5. Stitched Image of Image 1 and 2, along with Image 3 stitched together, using ORB and Left-To-Right Method

Figure 6. Image 1, along with Stitched Image of Image 2 and 3 stitched together, using ORB and Right-To-Left Method

the previous step.

After applying the perspective transform, the transformed image (of the second image) needs to be blended with the first image. If we simply replace the still-empty region of the result matrix with the first image, we will lose continuity in the overlapping region and the stitching would be poor. Hence, either we can take the average of the first image and the overlapping region and fill those values in that region, or we can take the maximum between the first image and transformed image. Calculation and observation tells us that averaging in particular overlapping region leads to slight blurred or visible-mixing of two patches of images, and hence choosing the maximum is a better option as compared to replacing or taking the average.

I opted for 3 different key-point detectors and descriptors, viz. SIFT, BRISK and ORB. And I chose 2 different methods of stitching images provided, viz. left-to-right and right-to-left. In left-to-right method, the leftmost two images are stitched firstly, and then the next image (in order from left to right) is stitched with the resultant image of previous stitching. So if we have 5 images, img1, img2, img3, img4 and img5; then firstly we'll stitch img1 and img2, forming a resulting image named res1. Then we'll stitch res1 and img3, forming res2; then res2 and img4 will form res3 and res3 with img5 will form res4, which will be the final result of stitching of all given 5 images. Whereas in right-to-left, what I did was exactly opposite in terms of order, i.e. img4 and img5 were the first ones to be stitched in the above scenario, if we had chosen right-to-left instead of left-to-right, and following the order, img1 would be the last image to be stitched to the then-result of previous images stitched together.

We can see very clearly that SIFT and BRISK perform quite well in Left-To-Right Method (as well as in Right-To-Left Method, which can be seen in the GitHub Repository), but ORB fails in Left-To-Right Method and not in Right-To-Left Method (Figure 3-6). This is because once we get a stitched image of two or more images, then the region in which the algorithm needs to find the key-points increases in a drastic ratio (almost doubles), and due to such large region, many key-points are detected, and out of them only a few lies in the overlapping region thus leading to poor matching.

One may think that restricting the number of key-points to be detected might solve this problem, but even after doing so, there is no guarantee that the key-points detected will lie in the region overlapping with the next image to be stitched. It may happen that out of top-k (let's say top-100) key-points, only a-few, like 5-10, lie in the overlapping region and hence these key-points will lead to even more poor matching. Restricting the ratio of distances is a better approach as compared to restricting the number of key-points to filter out the good matches from the set of all matches.

But the above problem doesn't arise in the Right-To-Left Method, moreover it gives a much better stitching result for the given 3 images. Right-To-Left method works here because of appropriate number of key-points lying in the region overlapping among Image 1 and result of stitching of Image 2 and 3. Though this might not always be the case, and hence no generic statement can be made about superiority of a method, among the two methods discussed.

Now, if we talk why ORB fails while implementing the Left-To-Right Method on the same image for the same threshold, as compared to SIFT and BRISK, which performs quite well; we can understand this by the following facts:

- ORB has lesser number of Key-Points as compared to SIFT, which uses a Difference-Of-Gaussians Method for various scale and space representations of the Key-Points. Hence, SIFT has large variety of Key-Points as compared to ORB, thereby more detailed descriptors leading to better image stitching even in case of images with complex patterns.

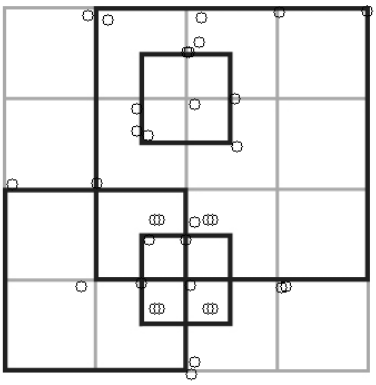

Figure 7. Harris Corners Detected without applying NMS, with a threshold of 97%

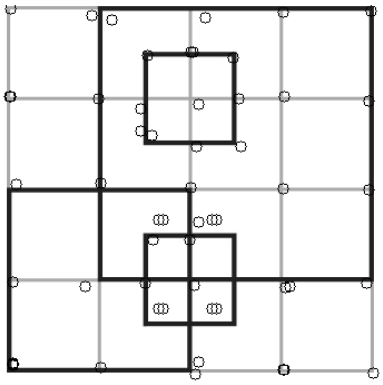

Figure 8. Harris Corners Detected with application of NMS, with a threshold of 40% and a window of size 49x49

- Moreover, BRISK uses a more complex mechanism to find Key-Points, and hence, they are more stable against smaller illumination and rotation changes. But ORB, due to its fast computation, lacks this stability and hence fails where the images have variability in illumination, rotation and complex patterns.
- Overall, ORB was made to compute the Key-Points and descriptors in a fast and effective way, but in complex scenarios, the trade-off between speed and performance becomes highly visible and hence ORB starts performing poor as compared to other algorithms, which perform better, though takes slightly more time.

2.7. My Idea

I noticed that to declare a pixel as a corner in Harris Corner, the Harris Corner Response Function of that pixel needed to be greater than a particular threshold, like 97% of the maximum value of the same or so. As a lower threshold leads to numerous pixels declared as 'corners'. But such a high threshold is also a problem as it misses out some of the actual corners, and also it declares a large number of pixels in a small neighborhood as 'corners', as they all have higher Harris Corner Response Function value, due to a corner, which is present in that neighborhood, and which can be identified by a pixel or two.

To overcome the problem of repetition in neighborhood, I used NMS (Non-Maximum Suppressing), which is generally used in Canny Edge Detection, which suppresses (makes pixel value 0) those pixels who are not having the maximum value in a small neighborhood defined by a window. This makes sure that in a small neighborhood, only one or two pixels are there for a unique corner. Applying NMS also helps us to choose a lower threshold for the final result of same number of corners (key-points).

The results of NMS, applied to Harris Corner Detection, are quite astonishing. Even with a very low threshold of 40% and just a small-sized window of 49x49 in an image of size 353x454, we are able to detect almost all corners present in the image; even those ones which are barely having noticeable gradient change. Figure 11 shows the corners detected by Harris Corner Detection without NMS, and Figure 12 shows the corners detected by Harris Corner Detection with NMS. We can see that without NMS, 34 corners are detected and most of them are around the same region, hence reducing the number of unique corners identified. Whereas when we apply NMS, we get a total of 54 corners and they are almost uniformly spreaded throughout the corners and hence increasing the count of unique corners identified. Without NMS, out of 41 corners present in the image, only 21 unique corners are identified. But when we apply NMS, out of these 41 corners available in the image, an amazing count of 38 unique corners is observed.

References

- Slides Used in the Course for Teaching
- Documentation of Algorithms used