CS 554: Computer Vision Homework Report

Berat Bicer
Department of Computer Engineering
Bilkent University
Ankara, Turkey
berat.bicer@bilkent.edu.tr

Ergun Batuhan Kaynak
Department of Computer Engineering
Bilkent University
Ankara, Turkey
batuhan.kaynak@bilkent.edu.tr

I. Introduction

This report contains the results of the homework and the related discussions. We start of by detailing the common operations for image stitching and disparity maps, then move on to talk about the assignment parts in sections 2 and 3 respectively.

II. Preliminary Operations

A. SIFT Feature Extraction and Matching

SIFT [1] or Scale-invariant feature transform is a scale-invariant image descriptor designed by David Lowe as is frequently used for many computer vision tasks. In this homework, we employ SIFT for feature extraction and point-of-interest matching. SIFT is a complex method to implement in a short period of time, so we employed VLFeat [2] implementation of SIFT for MATLAB. The output we get gives us 128 dimensional vectors for each feature point. After finding SIFT feature points for each image, we match them. Matching algorithm computes the euclidian distance between all features points. Next, having obtained two set of descriptors, we employ nearest neighbor distance ratio (NNDR), in which we compute and denote for each feature point in first image (namely origin point p_o) the following attributes:

$$
\begin{aligned}
D_{o1} &= argmin(|p_o - p_i|) \quad \forall i \neq o \\
D_{o2} &= argmin(|p_o - p_j|) \quad \forall i \neq o, i \\
T &= \frac{D_{o2}}{D_{o1}}
\end{aligned}
$$

Here, D_{o1} denotes the minimum distance between p_o and all other feature points and D_{o2} denotes the second minimum distance between p_o and all other feature points. We then look at the ratio of these two distances. Matches with higher T are more confident matches.

Note that this process may cause some duplicate matches. To ensure every match is unique, we apply post processing to eliminate all but the matching pair with smallest L2 distance apart amongst duplicate matches. Figure 1 shows results obtained when T is enforced to be at 1 and 2. We can see the effects of good NNDR thresholding against just taking the descriptors with the minimum distance to be matches.

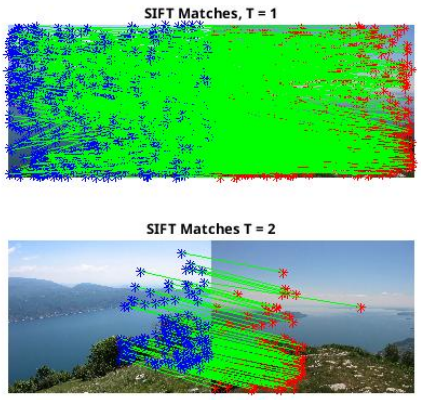


Fig. 1: SIFT NNDR result comparison for thresholds $T = 1$ and $T = 2$.

B. Finding Homography

Two images of the same planar surface are related by a homography. A simple homography between the images can be found with 4 pair of matching points and solving the resulting linear system. However, this is highly susceptible to noise in the matches and usually leads to poor generalization. A more sound approach is to solve the linear least squares problem using the matched points with an approximation algorithm such as Gauss-Newton. This is the approach we take in this part: First, we sample a set of matched pair of points where the number of pairs is bound by a hyperparameter. Next, we construct the least squares problem from these pairs and solve it using Gauss-Newton method. We make two changes to the original algorithm: The parameter update Δx is computed by $\Delta x = (J^{'} * J + \rho(x))^{\ddagger} * J^{'} + r(x)$ rather than the original formulation $\Delta x = (J^{'} * J)^{-1} * J^{'} + r(x)$ where $(\ddagger)$, $(')$ symbolize pseudoinverse and transpose operations respectively and $\rho(x)$ is a small Gaussian white noise. The purpose of using pseudoinverse and a Gaussian noise is to prevent numerical instability caused by inverse operation. Second change is to add a 'learning rate' to the update rule. This is inspired by the concept of learning rate in machine learning and is aimed to stabilize the convergence process. Yet, experiments showed that there is no significant gain in such an addition. The resulting homography is a vector of size 8 as follows: $h = [a_{00}\ a_{01}\ a_{02}\ a_{10}\ a_{11}\ a_{12}\ a_{20}\ a_{21}]$. Normally, homography is a 3x3 matrix with the last element of 1. However, we dropped this 1 for simplicity and directly incorporated into the calculations.

The algorithm described so far suffers heavily from outliers since least squares problem can be interpreted as line fitting, and outliers are known to negatively influence the resulting fit. We implemented RANSAC (Random Sample Consensus)

algorithm to deal with outliers when computing homography as follow: First, we randomly sample a number of pairs from the matching points set where the selected pair is determined by a hyperparameter. Next, we formulate a least squares problem using the selected pairs and run Gauss-Newton algorithm for computing the homography and count the inliers among all matching pairs. An inlier in this context is defined as a pair of matching points which, after first point is transformed by the homography, the L2 distance between the transformed point and the second (target) point is below a threshold. This threshold is a small positive number: as the threshold gets smaller, the inlier error becomes smaller yet the number of inliers lessens. Then, the resulting homography is saved and the process repeats for a certain number of times which is another hyperparameter (namely epochs). As the number of epochs increase, the number of candidate homographies increase which increases the probability of finding a good homography, along with the computation time required. After computing all homographies, we select the one with the smallest average matching error (L2 distance between the transformed point and the target). Another alternative is to select the homography with maximum number of inliers, however, this approach does not guarantee the quality of the inliers. As the number of inliers is the deciding criteria, a homography producing transformations with large L2 error can be selected.

III. Image Stitching

We detail the steps and results from image stitching in this section. We use im1.png and im2.png while detailing our steps, and then give the results for other matches afterwards. Sadly, we cannot show every step due to space constraints, Figure 7 presents the results of other final stitching results. 4-image stitching was done by first stitching images 22-23 / 24-25 and then stitching the products together after cropping black regions (all automatic).

A. Preliminary Operations

We find SIFT descriptors with NNDR threshold $T = 2$. We then run RANSAC algorithm for our matched points to get more confident matching points and therefore a better homography. Figure 2 shows the points we obtain after running RANSAC. A better homogprahy with possibly less points can be obtained, however it is too time consuming to run it for more iterations for so little improvement in performance. Also, removing too many points can result in a homography that fits well to those points, but it is actually erroneous.

B. Stitching

Now that we have a confident homography between our images, we apply affine transformation (warp) on image according to this homography (by multiplying the pixels in the image with the homography) and combine it with the other. Since affine transformation can warp the image dimensions, we first create a black image that can hold both images after warping. Adding these two images results in the most

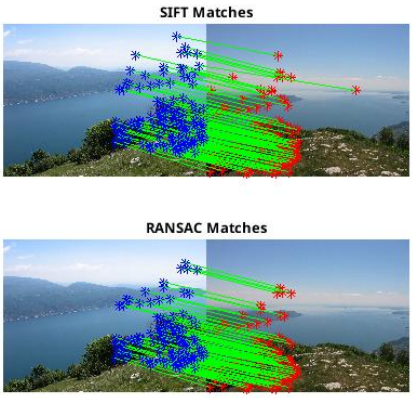


Fig. 2: Comparison between matches obtained by just SIFT-NNDR matching and also passing the matches through RANSAC algorithm.

primitive form of stitching. Since the intersecting pixels have contributions from both images, the intersection looks too bright. We can solve this by averaging the intersection pixels instead of adding them using a mask. See Figure 3 for the results. We obtain good a stitch, implying our homography calculation is good. Our problem right now is the frames of both images (seams) are seen in the images and the transition is not very smooth.

C. Alpha Blending

Instead of averaging the intersection region, we can choose the weight each image will contribute to that intersection pixel. To this aim, we define weight $\alpha = [0, 1]$. Left will contribute α of its pixel values while the right image contributes $1-\alpha$. Note that our averaging method corresponds to $\alpha = 0.5$. Figure 4 shows stitching results for different alpha. Note that extreme alphas remove the edge on one part of the image, while making the other edge look more apparent. We can argue that alpha blending is not very helpful as the seams are still visible even for different alpha values.

D. Gradient Blending (Linear Alpha)

The problem with alpha blending was that the intersection pixel values were calculated by weighted average of two images. This weighted averaging is constant throughout the intersection area, so the transitions are very sudden on image boundaries, and mostly the image with the largest weight overwrites the contribution of the other. To this aim, we research and find a better performing method called gradient blending (sometimes called linear alpha blending). The idea is that instead of a constant weight for each image, the images have variable weights along the intersection area. A weight for a pixel for an image W_{Ip} is high when the intersection pixel p is close to image I. We assume this distance to be horizontal distance. Also including the vertical distances would result in a better output, but we believe the superiority of the method

(a) Left Image (b) Right Image (c) Left and Right Image Combined (d) Normalized (c)

Fig. 3: Images laid out on a black image. (a) Left image without any transformation. (b) Right image after affine transformation of all of its points according to computed homography. (c) Combination of (a) and (b). (d) Bright intersection area in (c) normalized by averaging pixels

(a) $\alpha = 0.9$ (b) $\alpha = 0.1$

Fig. 4: Image stitches blending for $\alpha = 0.9$ and $\alpha = 0.1$. High α values removed the seam close to left image while low α removes the seam close to right image. $\alpha = 0.5$ can be found in Figure 3-d.

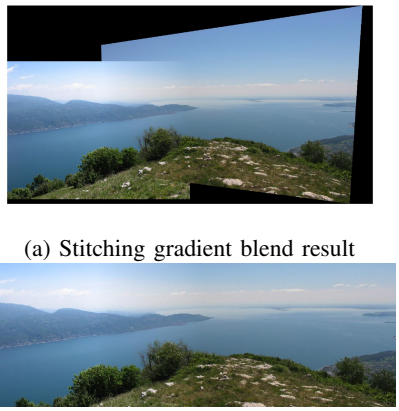
can be show without going to those extra calculations since the test images are moving mostly horizontally. The masks are presented in Figure 5 and the final results are presented in Figure 6. We crop the image so that we get rid of any black background and obtain a horizontal panaroma. Our final results are pretty good except the 4-image stitching has some issues with visible seams and a bit of ghosting. These could be improved with fitting better homogprahies, or implementing seam finding.

(a) Left Image Mask (b) Right Image Mask

Fig. 5: Gradient masks for (a) Left image (b) Right image.

IV. Disparity Maps

In second part of the homework, we are asked to find a dense disparity map for given pairs of images. We shall first describe the algorithm for the whole process, then share the best results obtained for each pair, and lastly discuss the shortcomings of the method and possible improvements.



(a) Stitching gradient blend result

(b) Cropped version of (a)

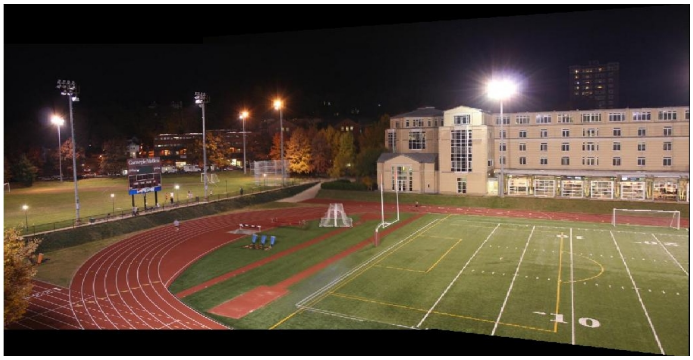
Fig. 6: (a) Stitching gradient blend result after applying individual masks. (b) Cropped version of (a).

Step 1: SIFT matching and homography finding

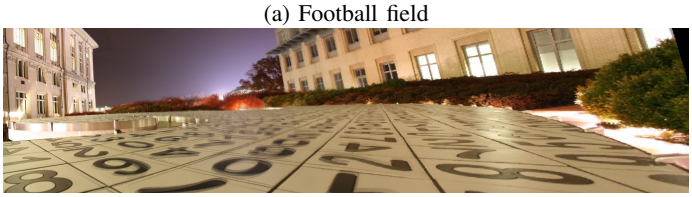
Please refer to Section II, Part A and B.

Step 2: Rectification

Using the computed homography, we can use to align the images so that the epipolar lines passing through the corresponding feature points become horizontal. In this case, in optimal case, the points are located on the same x-coordinate. Hence, by linearly searching over y-axis the matched points can be located. Rectification, therefore, is the process of applying the homography to the first image to obtain horizontal epipolar lines which warps the image to form another. For a point at (i, j) at first image, the corresponding pixel in the rectified image is calculated as $(x^{'}, y^{'})$ where $x^{'} = \frac{a_{00} \times x + a_{01} \times y + a_{02}}{a_{20} \times x + a_{21} \times y + 1}$ and $y^{'} = \frac{a_{10} \times x + a_{11} \times y + a_{12}}{a_{20} \times x + a_{21} \times y + 1}$. The downside of this formulation is the lack of guarantee that every point in the original image will be mapped to a unique point in the resulting image. Thus without postprocessing the resulting image has zero-valued pixels, meaning due to floating point operations and rounding, no pixel was mapped here. We fix this issue by setting intensity values of the zero-pixel to close-



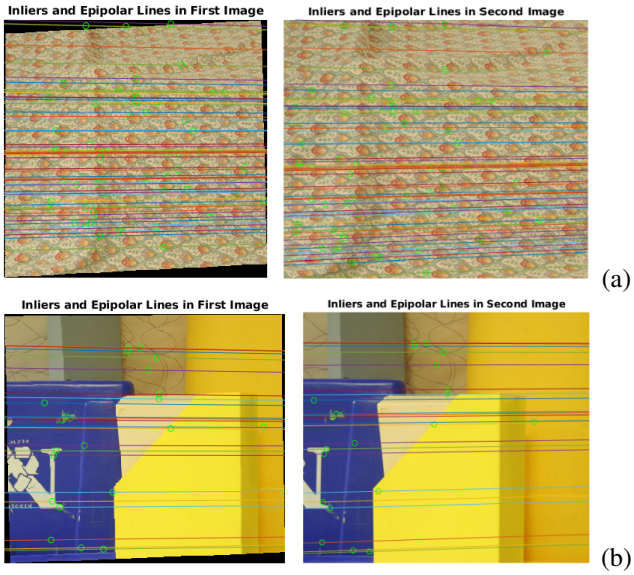
(a) Football field



(b) 4-Images

Fig. 7: Stitching results for (a) Football field. (b) 4-Images

neighbourhood intensity average of nonzero pixels around it. Sample rectifications and resulting epipolar lines can be seen in Figure 8.



(a)

(b)

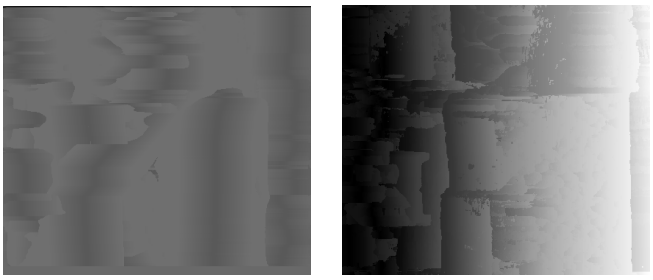
Fig. 8: Rectification results and obtained horizontal epipolar lines. Green circles indicate detected SIFT feature points. Colored lines are epipolar lines connecting corresponding feature points. Lines having the same color in both images correspond to same epipolar lines, and feature points on these lines are the matched feature points. Images on the left are rectified.

Step 3: Matching on epipolar lines and computing pixelwise disparity

After rectification, two images are assumed to be aligned so that matching feature points lie on the same epipolar lines. In this step, we find these points using linear search over epipolar lines. For this purpose, some definitions are required before moving on to the actual algorithm. We define a template as a a NxN rectangular pixel neighbourhood around central point $C_i = (x_i, y_i)$ (namely template center) cropped from target image - the image that is not rectified. A template has three channels: red, green, blue. For a given template, we define several patches as NxN rectangular pixel neighbourhoods around central point $C_j = (x_i, y_j)$ (namely patch center) cropped from rectified image. A patch has the same number of channels as its corresponding template. Notice that a patch is extracted from the rectified image after some horizontal translation. This is in relation to the previous assumption which states that matching points lie on the same horizontal epipolar line. By extracting patches, we seek to determine which patch center has the highest similarity with the template center. To this end, we define similarity between a template T and a patch R as $\Omega(T, R) = \sum_{\forall(P_{ij},Q_{ij})} \sum_{k=1}^{3} [v(i,j,k) \times w(i,j,k))]$ where $v(i,j), w(i,j)$ are vectors of size 3 containing color intensity values for points P_{ij}, Q_{ij} respectively. A point P_{ij} lies in the template T and resides at relative row-column index (i,j) compared to the template center. Q_{ij} is defined similar to P_{ij} except it lies in the patch rather than the template. Note that $-\frac{N-1}{2} \leq i,j \leq \frac{N-1}{2}$. For convenience, we limited the value N to odd numbers. The actual algorithm follows these definitions: For each pixel C_i in target image, we extract a template T and normalize it if necessary. Then, for each pixel in rectified image having column-index in range $[y_i - \Delta, y_i + \Delta]$, where Δ is a hyperparameter limiting search window on the epipolar line, we extract a patch centered at point $C_j = (x_i, y_j)$ and compute the similarity between the template and the patch. Assume the patch with the hightest similarity is centered around point $P_k = (x_i, y_k)$. Then, disparity value d for C_i becomes $d = |y_i - y_k|$. After computing disparity value for each pixel in target image, we apply normalization over the obtained disparity values to minimize extreme changes in disparity values around a local neighbourhood. Note that both the hyperparameter Δ and normalization after disparity map is obtained coincide with the assumption that a pair of stereo images is expected to have a small translation rather than a large one. It follows that pixels in close neighbourhood should have similar disparity values unless there is a boundary in between. By limiting the search window using Δ, we prevent possible distant matches. Normalization after computation fixes the case when a pixel is an extreme point compared to its neighbourhood.

Ω defined above is the correlation function and without normalizing the template (by subtracting the template mean for each channel) the responses scale by the intensity values of patch pixels. Therefore, in the implementation, T is normalized before processing if correlation function is used.

Note that another similarity definition can be made using the sum of squared differences(SSD) between pixel intensities, namely, Φ. For comparison, we implemented both functions and the best results obtained are given in Figure 9. Image (a) uses SSD and postprocess normalization, however, without normalization, the results were strictly worse so it is safe to make comparisons with (a). Image (b) uses correlation function and no posprocess normalization. We observe that the output with correlation function has clearer object boundaries and it is closer to the ground truth. However, results varied for other input pair (cloth). Upon inspection, SSD gave a better disparity map compared to correlation function. Note that this observation may have caused by other hyperparameters. Due to time restrictions, we are unable to verify which hypothesis is true, and thus, the results are omitted from the report.



(a) SSD (b) Correlation

Fig. 9: Best results obtained on plastic pair by using sum of squared differences(SSD) and correlation function as similarity measure.

We now discuss the affects of hyperparameters N and Δ on the disparity maps computed. Our assumption is that, as N grows larger, a pair with high similarity has a higher probability of being a correct match, since the information obtained by inspecting a larger crop grows by size N. This is indeed true, as shown in Figure 10. As N grows, the fluctuations in the disparity map becomes less frequent. Unlike N, as Δ increases, the probability of a false match increases since the number of candidates becomes larger. This implies that the resulting disparity will be less similar to the ground truth. From Figure 11, we observe that object boundaries blend in and shapes stretch in horizontal axis, implying this assumption is also true.

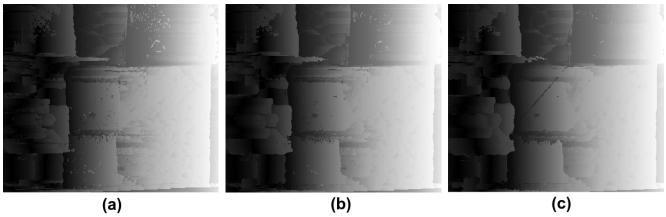


Fig. 10: Disparity maps for (a) N = 5, (b) N = 7, (c) N = 9.

Shortcomings of the algorithm are as follows: regarding homography and rectification, since Gauss-Newton is an approximation algorithm rather than an exact solution, the epipolar lines after rectification aren't exactly parallel everywhere in the image. As matches found at this step aren't exact, the obtained disparity values are approximations. To compensate this, we tried increasing the search window vertically, however, this led to infeasible computation times, up to 30 minutes for a run, and the results did not improve much. It is possible that a different combination of hyperparameters can lead to better results, but, we weren't able to find any such combination. Secondly, as the homography obtained is an approximation, and Gauss-Newton algorithm has many hyperparameters such as number of updates (epochs), it is possible to find a better homography for each pair of input. In the experiments, we tried many such homographies and the results that are shared in the next section are the best we could achieve, yet it is possible to obtained better results through more experimentation. Lastly for some homography vectors, shapes of the rectified image and target image were not the same. We solved this problem by resizing the rectified image, but it is possible that this operation violates the assumption that after rectification matching feature points share the same row index. Yet, as described in before, due to increased computation time we were unable to find a good hyperparameter set to compensate for this change.

(a) Δ = 30 (b) Δ = 40 (c) Δ = 50

Fig. 11: Various values for Δ and the corresponding disparity maps

REFERENCES

[1] G. Lowe, "Sift-the scale invariant feature transform," *Int. J*, vol. 2, pp. 91–110, 2004.

[2] A. Vedaldi and B. Fulkerson, "VLFeat: An open and portable library of computer vision algorithms," http://www.vlfeat.org/, 2008.

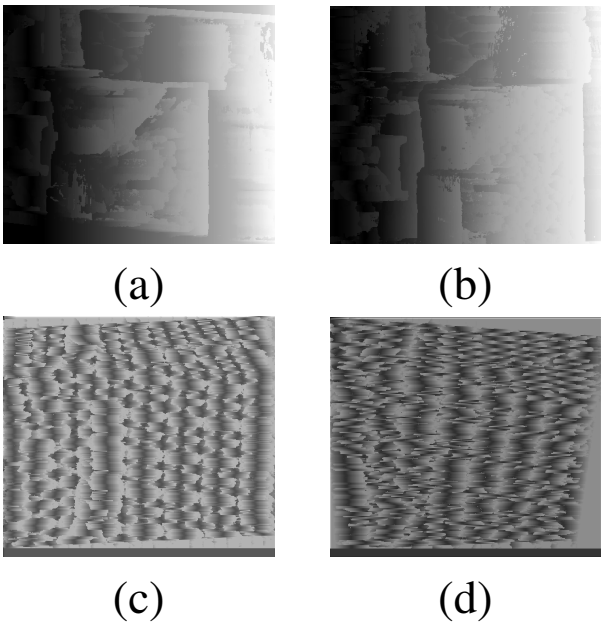


Fig. 12: Final disparity maps obtained for: (a) plastic, 1 to 2; (b) plastic, 2 to 1; (c) cloth, 1 to 2; (d) cloth, 2 to 1