CS 554: Homework Report
Image Stitching and Disparity Estimation *

Aydamir Mirzayev
Computer Engineering
Bilkent University
Ankara, Turkey
aydamir.mirzayev@bilkent.edu.tr

Irmak Türköz
Computer Engineering
Bilkent University
Ankara, Turkey
irmak.turkoz@bilkent.edu.tr

***Index Terms*—Computer Vision, Image Stitching, Disparity Map Estimation**

I. Introduction

In this report, we will discuss the solutions we have overcome with homework questions. The challenges, parameters and methodologies of each step will be explained. We will also describe the logic behind our process with respect to the Computer Vision course.

II. Image Stitching

A. Feature Descriptors

Since we were allowed to use existing code packages, We have used OpenCV's SIFT for feature extraction from the images. One challenge we have faced was the newly released version of OpenCV has removed the SIFT due to license issues, so instead, we had to use the older version's SIFT [1]. We call out the method and it returns the key-points and features of the image. Each of the descriptors has the same size of features, which is fixed to 128 by default. We have not to change the default value of the SIFT descriptor package.

B. Matching Feature Points

To match the features of two descriptors, we calculate two images feature points separately and do one to one match for each of the set in two images. In each iteration, we calculate the distance between two descriptors using "numpy.linalg.norm" which applies n2 normalization to the distances we gather from the feature points [2]. To map every pair of key points together, we call match feature points to function in every step. However, since we have many matches, we calculate the best k best matches and then save those values. Afterward, we use these values to decide if a match is a good match or not. Currently, our system uses k = 2, which means it calculates the first and second-best matches of keypoints. Then, a ratio is given as a hyperparameter. This ratio decides if our first match is relatively much better than the second one when multiplied with ratio. Our experiments concluded that the best value for ratio 0.75. Therefore, we eliminate most of the false matches if they do not have a big difference between their first order match and second-order match. This is called Lowe's ratio test [3] and it has proven to work best with SIFT matches with removing 90% of the false matches. Once we eliminate the matches, we check if the size of raw matches is larger than 4, because we can calculate homography with at least 4 points.

C. Homography

To find homography, we are given a set of points from image1 that match to set of points from image2. One important aspect of homography is to not lose the order of the points because their indices are ordered with respect to the matched points. In the first developed version, a huge mistake was to shuffle the points with different seeds. The system did not work at all, as we were not trying to fit homography between the matched pairs but any of the points in the images. Once we have fixed the seed for both of the points, we have defined a RANSAC size of 4, which means that We always get a random subset of size 4 from the matches. We also define a max iteration size, which decides how many times we will try out different subset to find the best homography matrix. We define two steps of the process to find a homography matrix at each step of the iteration. The first one is the creation of the homography matrix with given points of the first image and second image. We are using the Eq. (1) to calculate homography matrix where the (x_1, y_2), (x_2, y_2), (x_3, y_3), (x_4, y_4) are the points from the first image and (x_1', y_2'), (x_2', y_2'), (x_3', y_3'), (x_4', y_4') are points from the second image.

$$PH = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x_1' & y_1x_1' & x_1' \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y_1' & y_1y_1' & y_1' \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x_2' & y_2x_2' & x_2' \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2y_2' & y_2y_2' & y_2' \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3x_3' & y_3x_3' & x_3' \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3y_3' & y_3y_3' & y_3' \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4x_4' & y_4x_4' & x_4' \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4y_4' & y_4y_4' & y_4' \end{bmatrix} \begin{bmatrix} h1 \\ h2 \\ h3 \\ h4 \\ h5 \\ h6 \\ h7 \\ h8 \\ h9 \end{bmatrix} = 0 \quad (1)$$

Because we should be able to do this for any arbitrary number points where the RANSAC size is a parameter, numpy.linalg.svd is used to compose singular value transformation. The output is a 9x1 matrix, and then we convert it

to a 3x3 matrix which we call homography transformation. However, a proper normalization is needed to ensure h_9 is always 1, so we also divide the each of the element in the homography matrix to the h_9.

Once we find the homography matrix from the 4 points subset, we test out for the rest of the points. To do this, we use Eq. (2) where we put the However because our points set still have false matches; it is not logical to expect that the homography matrix will work for all of the points. The false matches may increase the error and cause us to find punish a good homography matrix. In order to prevent this, we test the points only with the subset of it. This was another hyperparameter, to decide which percentage of the points we are aiming to get the homography matrix accurate for. We have experimented for some values and found out that only testing a random subset of one of the third of the point set is enough to decide the homography matrix is good or not The error is calculated with the least-squares error, as it was stated in course slides. Once we find a less error than the previous minimum error, we update the error and homography matrix to the current iteration. Once we reach the maximum number of iterations, we quit the findHomography function.

Max iteration size has a significant effect to get accurate results. Because we always pick random subsets, the accuracy of the homography matrix increase as we have more trials. Convergence could be checked however, it would take much more time because we need to check every each of the combinations. In our implementation, we are able to find not the global minima in terms of error but we can get very close, or find a local minima. Essentially, to stitch two images; max iteration can be between 30000 - 40000 because it does not have to be a perfect homography matrix; however to stitch four images; max iteration should be at least 12000-13000 to gather more accurate results which will be fed to the next stitching network. Another important point with max iteration is; when we have more repeating patterns in the image, we need more iterations to be more precise about the best homography matrix. This is why, we have only max iteration as 3000 in 3 whereas we have max iteration as 15000 in 2. Moreover, we can also increase the RANSAC size which is equal to 4 by default, when we give the two images that have repeating patterns so that, we are checking correlations between keypoints.

$$x' = f(x,y) = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1}$$

$$y' = g(x,y) = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + 1}$$

(2)

D. Warping Image

To warp the second image with respect to the first image, we use WarpPerspective. The basic idea is to make the first image clipped and apply the homography matrix to the second image to scale, localize, shear and apply any perspective transformation to the second image. One constraint with our program is that we always allow side stitching. This means that we fix the height of the image as a constraint. Since the example images are only side stitched images, we have not described another warping function to warp images from top to bottom.

The warping perspective was most challenging when we have shared objects. Because we stretch the image with a float value; we need to cast it to an integer as pixels do not have float indices. This is either done with ceiling or flooring the results. However, this would cause black marks in the direction of stretching because the in-between pixels are remained empty. To solve this, we have not an efficient but sufficient method. We first construct the new image with only taking the floor of the resulting matrix when we multiply homography with the subject, and then we repaint the ceil - floor combination of x and y coordinates.

We also create an alpha mask for the newly inserted image, to be used in the alpha blending and find the corner points of the new image, however as we will discuss in II-E we could not complete it.

E. Alpha Blend

We have tried to do alpha blend, however, we have faced some problems. The first one was, we could not create a proper alpha mask to the intersection of two images. This is due to the added image may not be always fit a full vertical line from the first image, which causes us to detect every corner point for every newly inserted point. One possible solution was to do the stitched(right-most) image background and stable(leftmost) image foreground, however, this would lose the stitched images warped perspective side. So, the right way was to create the alpha mask only on the intersection region of two images; which we have could not calculate correctly in the given time, so we could not complete the extra time. However, even though we do not use it; our attempt to solve the alpha blending problem has still remained on the code for our further expedition.

F. Stitching Multiple Images

Stitching multiple images is not straightforward as stitching 2 images in the left to right manner. The main problem was that when we start to stitch multiple images, the edges become no vertical lines. So, we needed a different approach than finding the leftmost - rightmost images stitched together and adding images from the only right side would not help us. To solve this, we have a trick to flip the images after we stitch them and after insertion of a new image, we have flipped to get the correct result. Even though, in our implementation the code only works for 4 images; this idea could be applied to any arbitrary number of images. Since we do not have any

top to bottom examples in our given example set, we have implemented a top to bottom stitching. Nevertheless, the idea would be the same. Instead of stitching two images directly, we would first rotate them by 90 degrees in clockwise direction and stitch, and rotate them in the counterclockwise direction to get the final result. Instead of applying a more general stitcher, this trick would also work if we can supervise our implementation.

G. Results

Final Results of the images are given in Fig. 1, Fig. 2 and Fig. 3

Fig. 1: Image 1 and Image 2 Stitching result

Fig. 2: Image 3 through Image 6 Stitching result

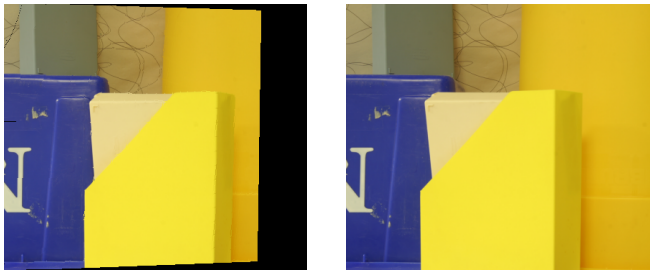
Fig. 3: Image 7 and Image 8 Stitching result

III. Disparity Map Estimation

A. Image Rectification

Similar to the previous parts we find the holography between two images. Procedure used in this part is identical to the previous parts. To avoid repetitiveness rectification steps are more briefly discussed here.

For holography calculation, first feature points and descriptors are tracked in each image. To this end SIFT feature descriptor from cv2 library of Python has been utilized. Then feature points are matched by minimizing respective distances and then fed to RANSAC algorithm to determine inliers. Finally, using Homographies each image is rectified with respect to the other. See F. 4 for a sample rectification result on images of plastic.



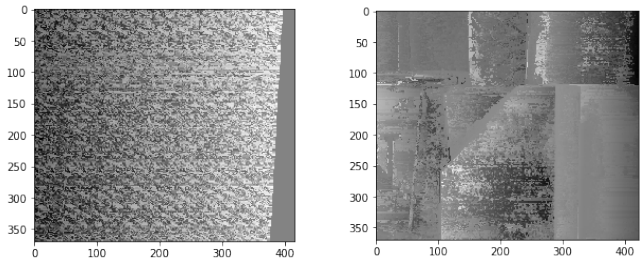
(a) Rectified View 1 (b) View 5

Fig. 4: Rectified image of plastic view 1 with respect to view 5

B. Pixel-to-Pixel Matching

After left image has been rectified with respect to the right image, we move on to matching corresponding pixels in two images. For this section several matching techniques have been employed. As a result we have observed that 'cloth' and 'plastic' images display optimal results under different set of parameters and matching techniques. For all the methods 3x3, 5x5, 7x7 and 9x9, 11x11 window sizes were used during experiments. For all the methods all three channels of the image are being used. For each employed method here, we will display sub-optimal results from view 1 of both images. Global optimal results will be declared at the last section with corresponding and parameters for each method at the end and display results for both views.

Classic Correlation Matching : For this method we conduct linear search along the lines of the rectified and reference images to match corresponding pixels in each row. Metric used for evaluating matrices is correlation. For evaluation, images are first zero padded to allow fitting to outermost pixels. Where size of padding is determined by: $padding = floor(m/2)$ with m being the size of the window. We then normalize corresponding windows in each image and obtain their dot product. This method displays fairly adequate results for the 'plastic' set of images, however, disparity at the very back of the image which corresponds to the furthest points is not perfect as it appears brighter in contrast to the expected darker background. Results for 'cloth' images, on the other hand, were almost uninterpretable. See Fig. 5.



(a) Cloth (b) Plastic

Fig. 5: Results for classical correlation based matching. Parameters: 3x3 windows.

As a result of experiments it was established that poor results for cloth images are due to inability of classic algorithm to deal effectively with repetitive patterns in the image. Which proved problematic with repeating pattern of the bed cover. We address this problem later in the report.

1) Classical Max Distance Matching: This method is fairly similar to the previous one expect as a matching criteria maximum distance between any component of the vectors is used. That is we normalize and flatten each window and obtaining $3m^2$ length vectors, then the maximum distance among all the dimensions of two vectors is recorded as their respective distance: Assuming that the first flattened window is A and the second one is B: $maxdistance = max(|A-B|)$. See Fig. 6 for illustration.

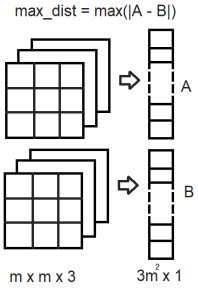
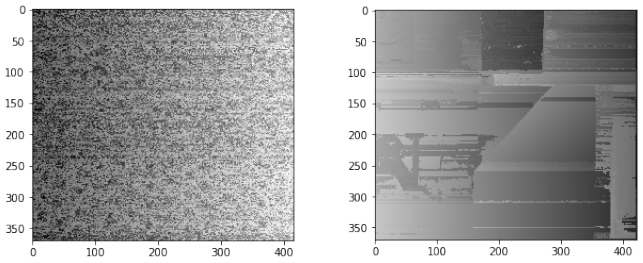


Fig. 6: Max distance method illustration

We traverse entire row of the image and the pixels with the minimum max-distance are then matched. This method displays surprisingly good results with plastic scenery image and especially with uniform surfaces. This is due to the fact that spatial variation in on uniform surfaces are mostly due to slight changes. Max method stresses those variations therefore displaying better matching on uniform surfaces. Better performance with uniform surfaces helps us to obtain good results at the very backside of the image as it is the most problematic region due to low spatial variability See Fig. 7.



(a) Cloth (b) Plastic

Fig. 7: Results for classical max-distance based matching. Parameters: 3x3 windows.

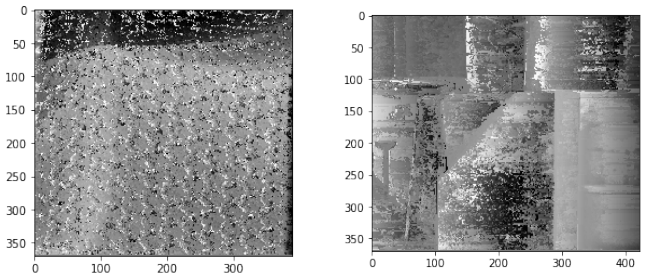
Results for the cloth images do not change significantly as we still do not address the issue of repeating patters. This problem is addressed in the following section.

C. Smarter correlation Matching

In this section we make three improvement to the classical correlation based matching algorithm. i) Firstly we introduce tolerance for rectification imperfections. Classical algorithm assumed perfect rectification, however even with state of the art rectification algorithms there are always slight vertical variations. To account for these imperfections, instead comparing a pixel to all the pixels in one row, we compare with 3 rows. Additional horizontal padding is applied to images for this purpose. This simple trick proves to be very effective in reducing spatial noise. ii) Secondly, instead of using entire row of the the right image for matching we only use a smaller neighborhood. We utilize the fact the not only left image is rectified but also mapped to the right image using holography transform. This little trick proves very effective at discriminating repetitive patters, which was a problem with cloth images. iii) Lastly we introduce confidence threshold parameter. That is if the maximum correlation along traversed row is less than the threshold we use disparity value from previous pixel. If the algorithm moves to a new row, we set disparity back to zero.

This approach proves extremely effective for estimating depth in cloth images as it accounts for highly repetitive patters by confining the search perimeter. Results for the plastic images on the other hand are inferior compared to the max methods. This is due the fact that plastic images contain a lot of uniform surfaces which are not handled well by the third method.

See Fig. 8 for results.



(a) Cloth (b) Plastic

Fig. 8: Results for smarter correlation based matching. Cloth image parameters: 3x3 windows, search at 30 pixel horizontal neighborhood, with 0.1 threshold. Cloth image parameters: 5x5 windows, search at 200 pixel horizontal neighborhood, with 0.1 threshold.

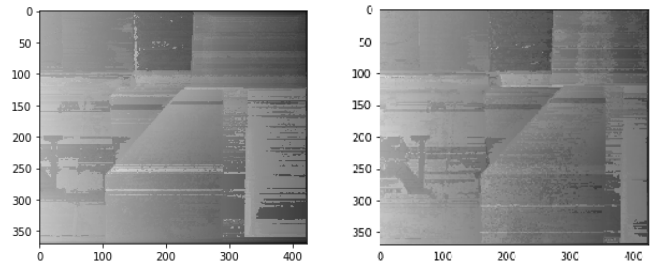
D. Optimal Results

For plastic images optimal results were obtained using blending of two results. We observed that max-distance results are very successful at discriminating background of the image with low level of intensity. On the other hand our results from classical matching were decent on the foreground but suffered on the background. For that reason, to obtain best results possible, we blended two images using formula

outlined below.

$$I = \theta I_m + (1 - \theta) I_c$$

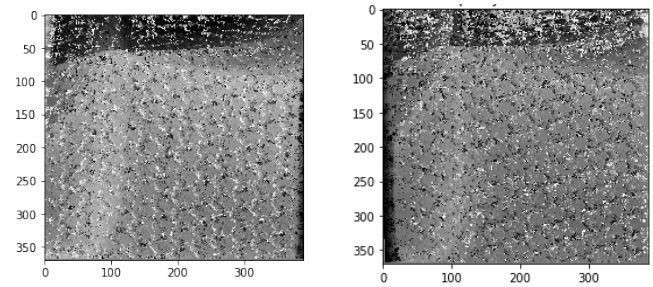
Where θ is the blending parameter that varies in [0,1] range and I_m, I_c are results from max-matching and classical matching respectively. Optimal results obtained for the plastic images are displayed in Fig. 9 with parameters defined in the description.



(a) View 5 (b) View 1

Fig. 9: Optimal results for plastic images. Parameters: max-matching, classic matching, blended with $\theta = 0.7$, 3x3 window in each method.

For cloth images optimal results have been obtained using smarter implementation with a window size of 3x3, a threshold of 0.1 and search range of 15 pixels is each direction with total of 30 pixels. See Fig. 10 for the results.



(a) View 5 (b) View 1

Fig. 10: Optimal results for cloth parameters. Parameters: 3x3 windows, search at 30 pixel horizontal neighborhood, with 0.1 threshold.

REFERENCES

[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] Travis Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006–. [Online; accessed ¡today¿].

[3] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60, 91–110.*