16720 – Computer Vision

Homework 2

Rajat Vikram Singh
rajats@andrew.cmu.edu

2.1. Used the in-built function. Got the following output:

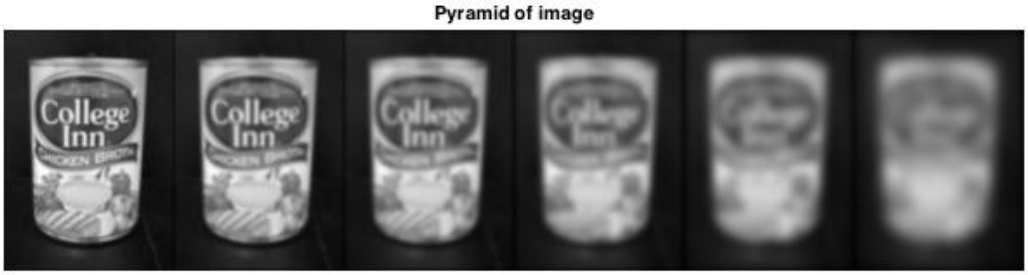

Figure 1. Gaussian Pyramid of chicken broth image

2.1. Wrote the createDoGPyramid function. Got the following output:

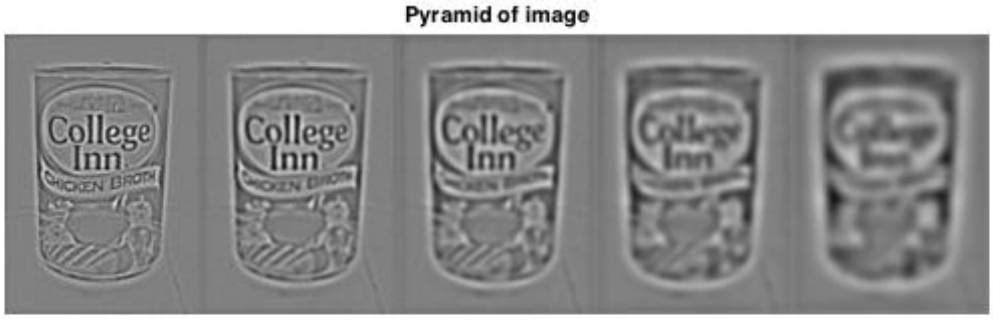

Figure 2. DoG Pyramid of chicken broth image

2.1. Implemented function computePrincipalCurvature, used the Gaussian function as hinted in the assignment. To calculate the trace and determinant used the formula for non-square matrices, since the hessian was not coming out to be a square matrix.

2.1. Wrote the function getLocalExtrema using imregionalmax and imregionalmin, because the function needs both local maximas and minimas. Two loops used but used very little calculation within the loop which makes the function comfortably fast.

2.1. Captured all of the functions together in the function DoGdetector, which computes the features for this image. The detected interest points plotted on an image are as follows:

Figure 3. The detected interest points from the DoGDetector.

2.1. Implemented the makeTestPattern which finds random indexes from the distribution Gaussian(0, (1/25)*S^2)) (S is 9 in this case). Since this function gives us indexes in the patches, used the value S^4 to get values in the range approx. (-40, 40). We get the two index vector compareX and compareY from this function and saved the matrix as testPattern.mat in the submission folder.

2.2. Implemented the function computeBrief which uses the compareX and compareY values to get the index of the patch. The values are from the (-patchWidth/2, patchWidth/2) range. The code iterates through all DoG detected points and finds the brief descriptor for each point. Each point is checked to make sure that the code will not reference any of the points which are outside the boundary of the image. The code then finds the corresponding values as described in the brief paper to describe each point.

2.3. Completed the briefLite function, which uses the DoGDetecter and the computeBrief functions to compute the brief descriptors for the image feature points.

2.4. Wrote the testMatch script which shows the matches between two images. Some of the outputs from the image are as follows, more images are in the results folder:

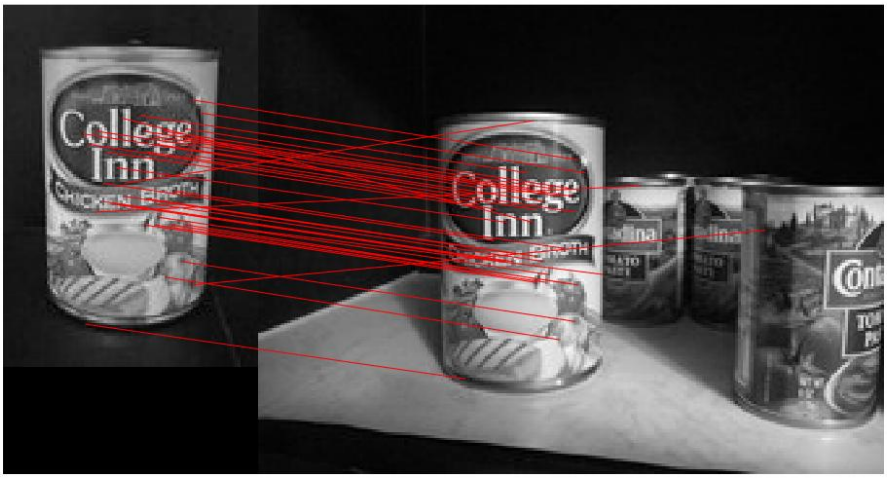

Figure 4: The matches between two ChickenBroth images

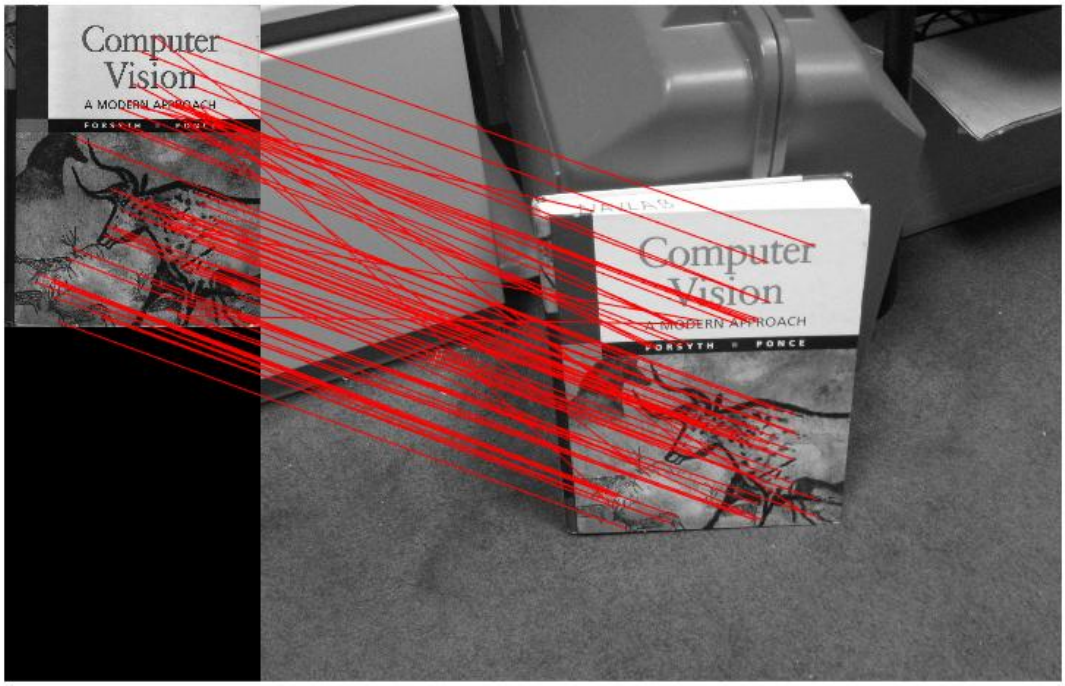

Figure 5: The matches between two CV book images

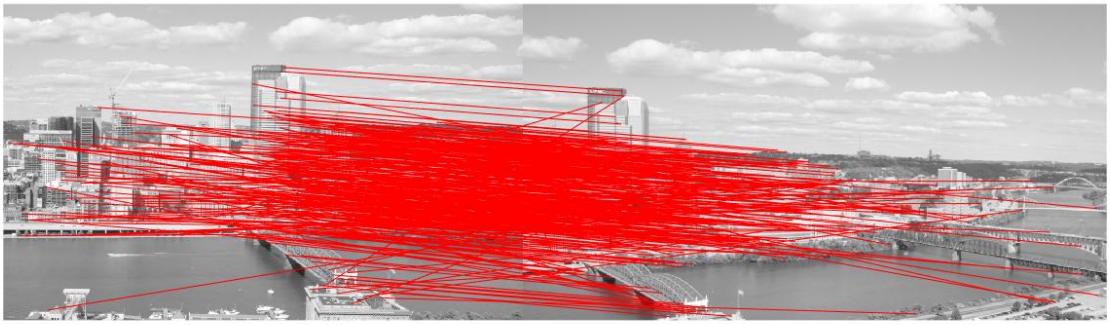

Figure 6: The matches between two incline images

Some bad matching images are:

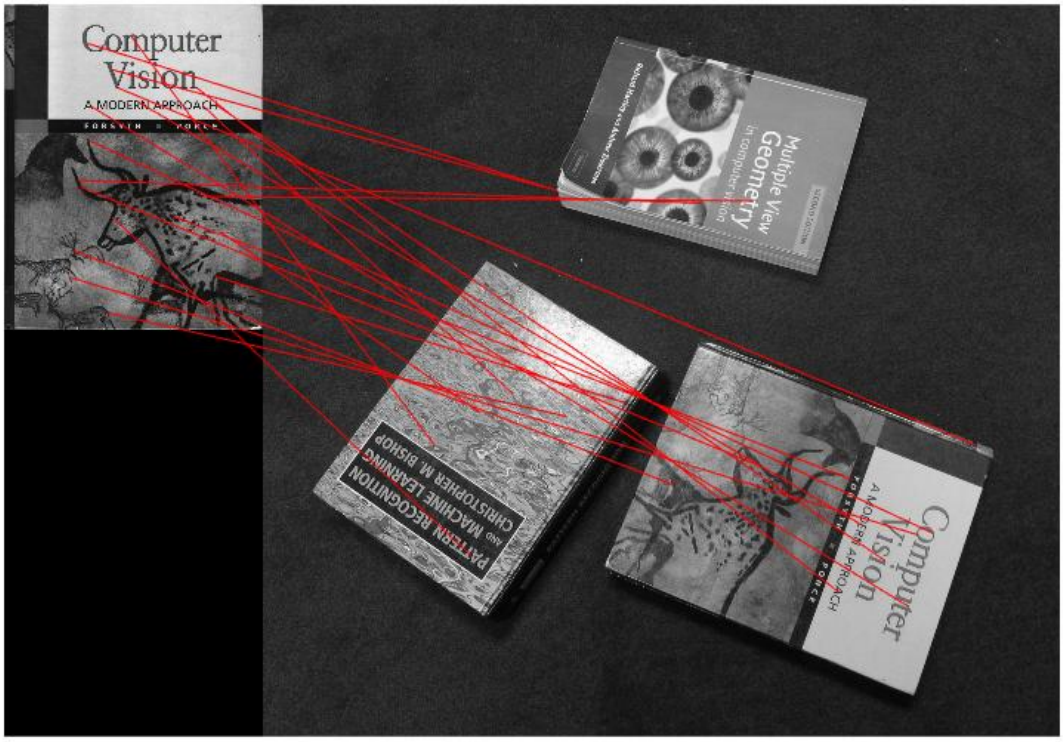

Figure 7: The book is rotated and hence the correct matches are few. We know BRIEF performs poorly in rotation scenarios and this examples shows that.

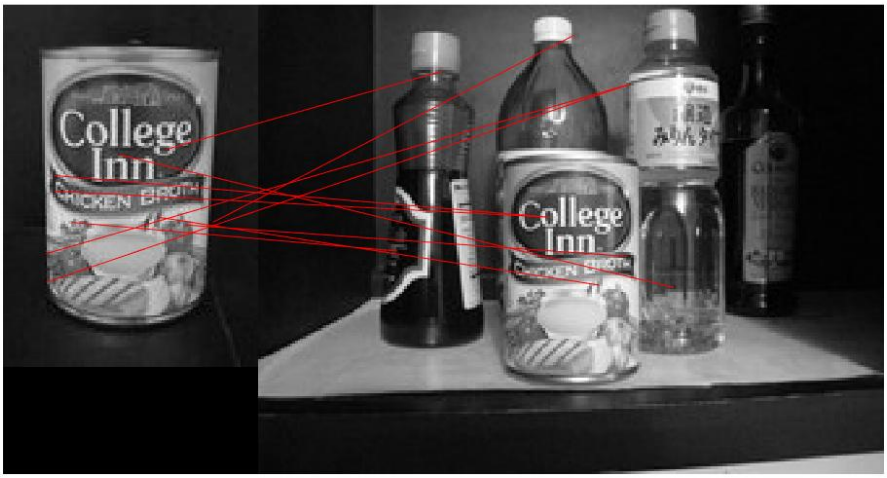

Figure 8: Only few correct matches. Possible reason is the existent of different objects in the background, but it should not ideally affect the performance. The image on the right is a little unclear and that could be the reason for it performing badly. The presence of objects with similar texture also breaks the DoG detector in this case.

2.5. Wrote the script briefRotTest which rotates the image 10 degrees each time and generates the histogram of the matches.

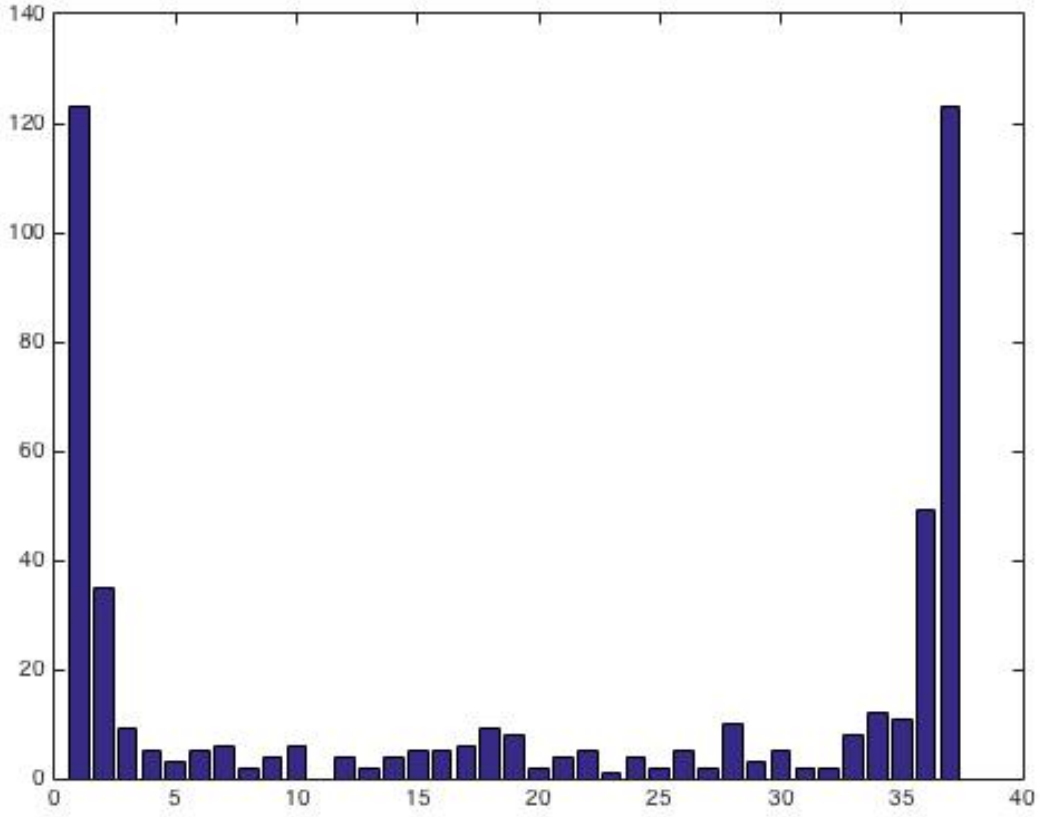

Figure 9: The number of matches when the images is rotated.

The BRIEF paper talks about it being rotational invariant and this is exactly what we are seeing here. The reason BRIEF behaves this way is because there is no orientation component in the descriptor, it works on the relative positioning of the pixels around the point as the way of describing it. When the image is rotated the relative position changes and hence the descriptor changes and we get less matches.

3. Reference for question 3 taken from: https://cseweb.ucsd.edu/classes/wi07/cse252a/homography_estimation/homography_estimation.pdf

HW2

Rajat Vikram Singh
rajatv@andrew.cmu.edu

3.1. $p = \{p^1, p^2, \ldots, p^N\}$

$q = \{q^1, q^2, \ldots, q^N\}$

$p^i = Hq^i$

$Ah = 0$

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}$$

$$\lambda \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}$$

now,

$\lambda x_1 = h_{11}u_1 + h_{12}v_1 + h_{13}$ $\Rightarrow$ $x_1 = \dfrac{h_{11}u_1 + h_{12}v_1 + h_{13}}{h_{31}u_1 + h_{32}v_1 + h_{33}}$ ← homogenous coordinate

$\lambda = h_{31}u_1 + h_{32}v_1 + h_{33}$

$\Rightarrow x_1(h_{31}u_1 + h_{32}v_1 + h_{33}) = h_{11}u_1 + h_{12}v_1 + h_{13}$

$\Rightarrow x_1h_{31}u_1 + x_1h_{32}v_1 + x_1h_{33} - h_{11}u_1 - h_{12}v_1 - h_{13} = 0$ —(1)

rearranging,

$$-h_{11}u_1 - h_{12}v_1 - h_{13} + h_{31}x_1u_1 + h_{32}x_1v_1 + h_{33}x_1 = 0$$

Similarly, for y_1

homogenous coordinate $y_1 = \dfrac{\lambda y_1}{\lambda}$

$\lambda y_1 = h_{21}u_1 + h_{22}v_1 + h_{23}$ $\Rightarrow$ $y_1 = \dfrac{h_{21}u_1 + h_{22}v_1 + h_{23}}{h_{31}u_1 + h_{32}v_1 + h_{33}}$

$\lambda = h_{31}u_1 + h_{32}v_1 + h_{33}$

$\Rightarrow y_1(h_{31}u_1 + h_{32}v_1 + h_{33}) = h_{21}u_1 + h_{22}v_1 + h_{23}$

$\Rightarrow y_1h_{31}u_1 + y_1h_{32}v_1 + y_1h_{33} = h_{21}u_1 + h_{22}v_1 + h_{23}$

$y_1 h_{31} u_1 + y_1 h_{32} v_1 + y_1 h_{33} - h_{21} u_1 - h_{22} v_1 - h_{23} = 0$

rearranging,

$$-h_{21} u_1 - h_{22} v_1 - h_{23} + h_{31} y_1 u_1 + h_{32} y_1 v_1 + h_{33} y_1 = 0 \quad \text{— (2)}$$

we want to solve for matrix h.

if we consider

$$h = (h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33}) \quad \text{— (3)}$$

From (1) & (2), we get

~~$a_x = (-x_1, -y_1, -1, 0, 0, 0, \ldots$~~

$$a_x = (-u_1, -v_1, -1, 0, 0, 0, x_1 u_1, x_1 v_1, x_1) \quad \text{— (4)}$$

$$a_y = (0, 0, 0, -u_1, -v_1, -1, y_1 u_1, y_1 v_1, y_1) \quad \text{— (5)}$$

From (3), (4) & (5), (1) & (2) can be written as :-

$$a_x h = 0$$

$$a_y h = 0$$

So, for two points we can get.

$$\begin{bmatrix} a_x^T \\ a_y^T \end{bmatrix} h = 0$$

$$Ah = 0$$

For, more points A, would be a $n \times 9$ dimensional matrix.

b) h has 9 elements i.e, $h_{11}, h_{12}, h_{13}, h_{21} h_{22}, h_{23}, h_{31}, h_{32}, h_{33}$

c) The matrix H has 8 degree of freedom so, it need 4 point pairs to solve this system.

d) Given $Ah = 0$, the sum squared error can be written as,

$$f(h) = \frac{1}{2}(Ah-0)^T(Ah-0)$$

$$f(h) = \frac{1}{2}(Ah)^T(Ah)$$

$$f(h) = \frac{1}{2}h^TA^TAh$$

$$= \frac{1}{2}h^TA^TAh$$

Taking derivative of f w.r.t h and setting the result to zero, we get

$$\frac{df}{dh} = 0 = \frac{1}{2}(A^TA + (A^TA)^T)h$$

$$= \frac{1}{2}(A^TA + A^TA)h$$

$$= \frac{1}{2} \times A^TAh$$

$$0 = A^TAh$$

We know, that the eginvectors of A^TA correspond to matrix V of the SVD decomposition if matric $A = V\Sigma V^T$. and for A^TAh to be 0 h should ~~belong~~ to be equal to the eginvector of A^TA which has 0 eigenvalue or the minimum value.

$$A = V\Sigma V^T = \sum_{i=1}^{9} \sigma_i u_i V_i^T$$

We get v vector and take the eginvector corresponding to σ_9 (because it is the smallest). If σ_9 is 0 then homography is computed exactly. If there is noise and ~~has~~ σ_9 is ≈ 0 then homography is overdetermined.

Hence, using SVD we can solve the least square problem for homography.

4.1. Implemented the function computeH to calculate the homography matrix H using SVD.

5.1. Implemented the imageStitching function which takes the H computed in computeH and uses it to transform the pixels of the second image in the new plane. The out size of the warped image is fixed to be the addition of the sizes of both the images. Image is stitched using the max function of both the images, so that the image intensities appear seamless. A sample output of the function is:

Figure 10: Image stitching without Ransac and with clipping

5.2. Implemented the imageStitching_noClip function, by using the outermost points of the image to be warped to calculate the translation needed in the image and using the aspect ratio to calculate the scaling of the image. This is the matrix M, which will be used to transform the image to a different plane for both image 1 and image 2. We multiply this 3x3 matrix with the homography matrix for the second image and just the matrix M for the first image. The out_size is also calculated keeping in mind the aspect ratio required by the user. A sample output of the method is:

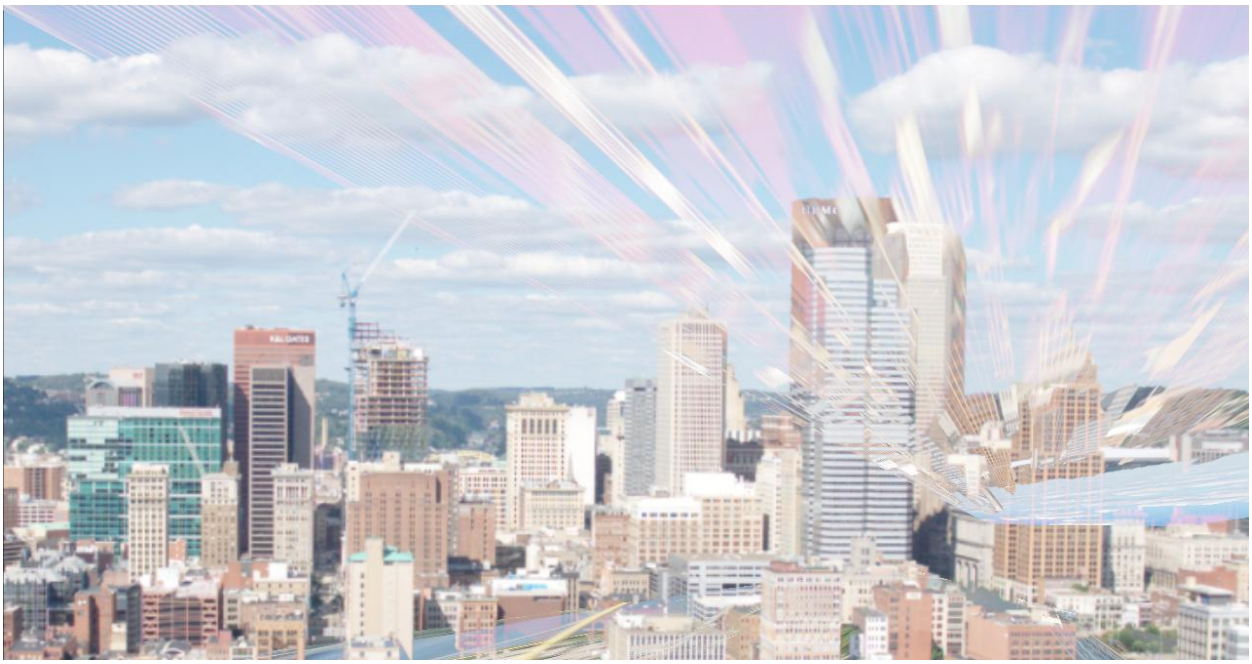

Figure 11: Image stitching without clipping and without RANSAC

6.1. Implemented the ransacH method, which iteratively computes four random matches from all the matches and calculates the homography using these points through computeH. Using this H, all the matched points are transformed to the new plane and then the distance between the transformed point and the original point from image 1 is calculated. The number of inliers is calculated. The number of inliers is the number of points whose distance is below the specified threshold. The method is repeated for the specified iterations and the best H matrix is returned. Using this H matrix gives good results in the image. The outputs from these H are added after the next question. The default value of nIter is set as 1000 and tol as 10, as it was giving a good result for this setting.

6.2. Tied everything together to calculate the panorama image. The final output image from this method are showed below:

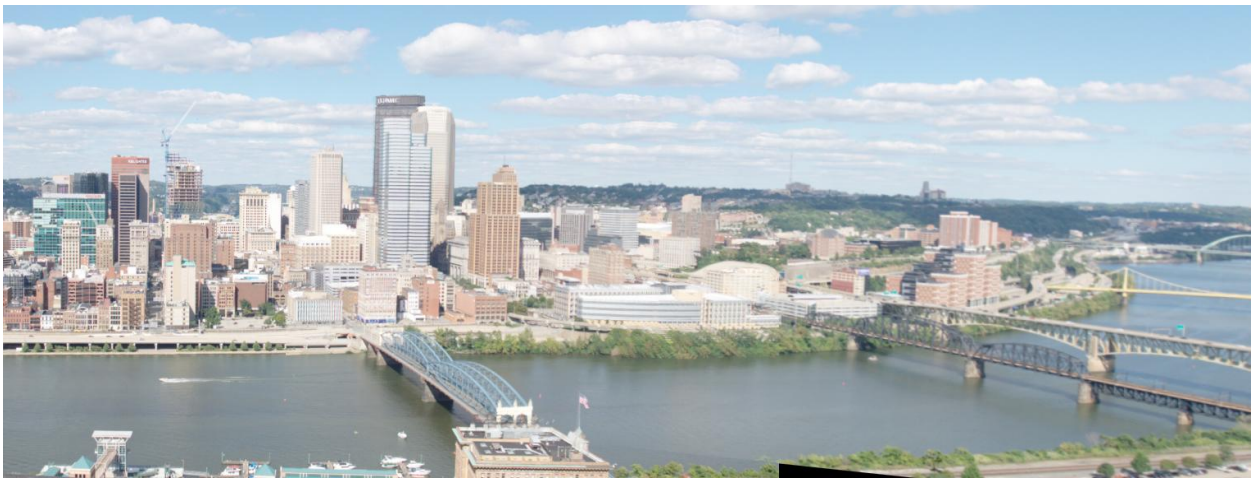

Figure 12: Image stitching with Ransac and with clipping

Figure 13: Image stitching with Ransac and without clipping

Note: I discussed the solutions with classmate Tushar Chugh and he helped me debug my code as well.