Computer Vision Exercise 1

Tomer Geva Maor Naftali

December 2021

Part A: Homography computation

1. Projective Transformation Equation System Derivation

In this section we will write the mathematical development used to find the homography transformation matrix, or the projective transformation matrix. Let us denote the homography as follows:

$$\begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} = \boldsymbol{H_{3X3}} \tag{1}$$

The transformation between the coordinates in the source picture, noted as (u, v) and the coordinates in the destination picture noted as (u', v') is given by:

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ k \end{pmatrix} = \boldsymbol{H_{3X3}} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = \begin{pmatrix} \tilde{u}/k \\ \tilde{v}/k \\ 1 \end{pmatrix} \tag{2}$$

We can see that since the coordinates in the destination coordinates are homogeneous and therefore the projection matrix's degree of freedom (D.O.F) is reduced to 8 instead of 9. Re-writing the matrix equation in Figure (2) as a set of linear equations we get:

$$\tilde{u} = h_{11}u + h_{12}v + h_{13} \tag{3a}$$
$$\tilde{v} = h_{21}u + h_{22}v + h_{23} \tag{3b}$$
$$k = h_{31}u + h_{32}v + h_{33} \tag{3c}$$
$$u' = \frac{h_{11}u + h_{12}v + h_{13}}{h_{31}u + h_{32}v + h_{33}} \tag{3d}$$
$$v' = \frac{h_{21}u + h_{22}v + h_{23}}{h_{31}u + h_{32}v + h_{33}} \tag{3e}$$

After some re-ordering we get:

$$-h_{11}u - h_{12}v - h_{13} + h_{31}u'u + h_{32}u'v + h_{33}u' = 0 \tag{4a}$$
$$-h_{21}u - h_{22}v - h_{23} + h_{31}v'u + h_{32}v'v + h_{33}v' = 0 \tag{4b}$$
$$\tag{4c}$$

We can see that from a transformation of a single point we ended up with two equations. Since the projection transformation matrix has 8 D.O.F we need a minimum of 4 points to find the projection transformation. Let us note the for a noiseless case, 4 point will suffice but for the case where the pairs of coordinates (u, v, u', v') are not perfect and some noise is introduced to the system, more points will allow for a more precise computation of the projection transformation. By denoting the following:

$$\underline{b} = \underline{0} \tag{5a}$$
$$\underline{x} = (h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33})^T \tag{5b}$$

For the i^{th} pair of coordinates noted as (u_i, v_i, u'_i, v'_i) we can substitute into (4). Repeating for a number of coordinate pairs, we create the following matrix:

$$\begin{pmatrix} -u_1 & -v_1 & -1 & 0 & 0 & 0 & u'_1u_1 & u'_1v_1 & u'_1 \\ 0 & 0 & 0 & -u_1 & -v_1 & -1 & v'_1u_1 & v'_1v_1 & v'_1 \\ -u_2 & -v_2 & -1 & 0 & 0 & 0 & u'_2u_2 & u'_2v_2 & u'_2 \\ 0 & 0 & 0 & -u_2 & -v_2 & -1 & v'_2u_2 & v'_2v_2 & v'_2 \\ & & & & \cdot & & & & \\ & & & & \cdot & & & & \\ & & & & \cdot & & & & \\ -u_i & -v_i & -1 & 0 & 0 & 0 & u'_iu_i & u'_iv_i & u'_i \\ 0 & 0 & 0 & -u_i & -v_i & -1 & v'_iu_i & v'_iv_i & v'_i \\ & & & & \cdot & & & & \\ & & & & \cdot & & & & \\ & & & & \cdot & & & & \end{pmatrix} \triangleq \boldsymbol{A} \tag{6}$$

And by using (5) we can get:

$$\boldsymbol{A}\underline{x} = \underline{b} = 0 \tag{7}$$

We want to find $\underline{x}^*$ that holds the following:

$$\underline{x}^* = \underset{\underline{x}}{argmin}||\boldsymbol{A}\underline{x}||^2 \quad s.t.||\underline{x}|| = 1 \tag{8}$$

Noting that the linear equation set in (7) is homogeneous, we can not solve (8) using Least Squares, leading us to use Singular Value Decomposition(SVD). SVD states that any given matrix can be de-composed into:

$$\boldsymbol{A}_{M\times N} = \underbrace{\boldsymbol{U}}_{M \times M} \times \underbrace{\boldsymbol{\Sigma}}_{M \times N} \times \underbrace{\boldsymbol{V}^T}_{N \times N} \tag{9}$$

Where:

- $\boldsymbol{U}$ is an orthogonal matrix such that the columns are the eigenvectors of $\boldsymbol{A}$
- $\boldsymbol{\Sigma}$ is a matrix with non-negative entries on the diagonal(singular values)
- $\boldsymbol{V}$ is an orthogonal matrix such that the columns are the eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$

It holds that the projection transformation matches the column of $\boldsymbol{V}$ which corresponds to the lowest singular value in $\boldsymbol{\Sigma}$, and the reason is as follows: Taking the derivative of (8) w.r.t $\underline{x}$ and checking when it is equal to 0 results in:

$$\frac{d}{d\underline{x}}\left(||\boldsymbol{A}\underline{x}||^2\right) = 2\boldsymbol{A}^T\boldsymbol{A}\underline{x} = 0 \rightarrow \boldsymbol{A}^T\boldsymbol{A}\underline{x} = 0 \tag{10}$$

Looking at (10), we see that $\underline{x}$ holds if it is equal to the eigenvector matching the eigenvalue of 0 for the noiseless case, or the smallest non zero eigenvalue in the presence of noise.

Let us note the in this exercise, the homographies will be shown scaled such that $H_{3,3} = 1$. This is done to enforce a uniform method thus allowing to compare between homographies with ease.

2. Naive Homography Computation

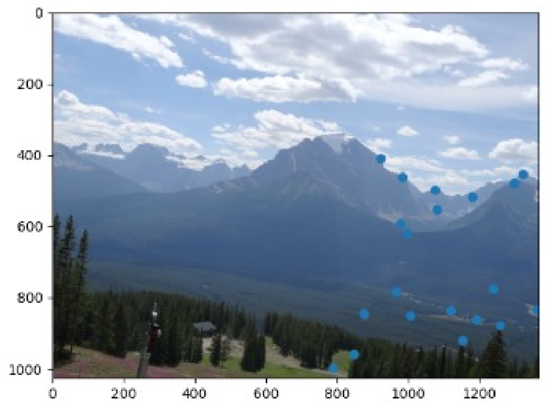
In this section, a function was written to generate the matrix given in (6) using all the given coordinate pairs in the input to reduce noise effects. The Homography is then computed using SVD as explained in the previous sub-section. The resulted function is attached under the API "compute_homography_naive"

3. Noiseless Homography Computation Results

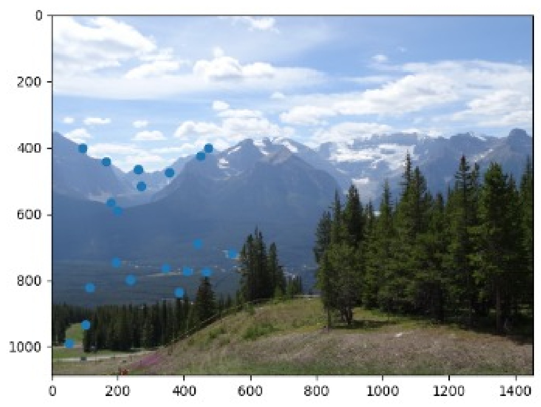
In this section the "matches_perfect.mat" file was loaded and the homography was computed according to it. As the name suggests, the coordinate set does not contain outliers but it is not completely noise-free, which will we expanded later in this sub-section. The resulted homography using this coordinate pairs is shown in (11) below, and the computation time was ≈ 1 $[msec]$

$$\boldsymbol{H}_{noiselss\ naive} = \begin{pmatrix} 1.43457214e+00 & 2.10443232e-01 & -1.27718679e+03 \\ 1.34265153e-02 & 1.34706123e+00 & -1.60455872e+01 \\ 3.79279298e-04 & 5.56523146e-05 & 1.00000000e+00 \end{pmatrix} \tag{11}$$

The images and the point pairs used for the homgraphy are shown in Figures (1a) and (1b) below:



(a) Source photo with points



(b) Destination photo with points

Figure 1: Used images with coordinate pairs shown

Although the source and destination coordinate pairs do not hold outliers, there is still inherent noise in the data, stemming from the fact that the points (u_i, v_i, u_i', v_i') are integers and are essentially quantized. In the case of a very dense grid (high resolution picture) this quantization can be modeled by Additive Gaussian White Noise(AWGN). This is further verified by the fact that the SVD's lowest singular value is not zero, but close to zero (≈ 0.01).

Part A2: Forward Mapping Slow and Fast

4. Forward Homography Slow Computatioon

In this section the rows and columns were iterated in nested for loops and the transformation from the source coordinate points to the destination coordinate points was computed iteratively as depicted in (2) above using the homography in (11). As the exercise requires, this was done in the "compute_forward_homography_slow" API. the resulted post-transformation image in the destination coordinates, in the size of the destination image is seen if Figure (2) below. The computation duration was $\approx 7[sec]$

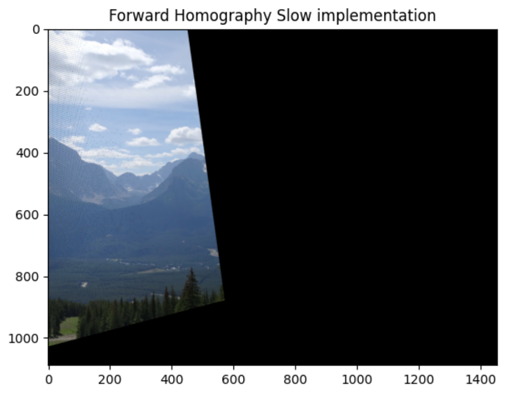


Figure 2: Forward homography slow implementation results

5. Forward homography fast implementation results

In this section the homography was computed vectorically, using matrix multiplication. the source coordinates were organized in a $3 \times (W * H)$ in homogeneous coordinates. This allows us to perform matrix multiplication with the homography in (11) to produce a $3 \times (W * H)$ output holding the destination coordinates. these were then used to interpolate the data onto the grid and return the destination coordinates with the transformed source. The results of the fast method and the slow method are indistinguishable, but for the sake of completeness, the results is shown in Figure (3) below. The fast computation was done in $\approx 300[msec]$ which is a major improvement, more than an order of magnitude faster.

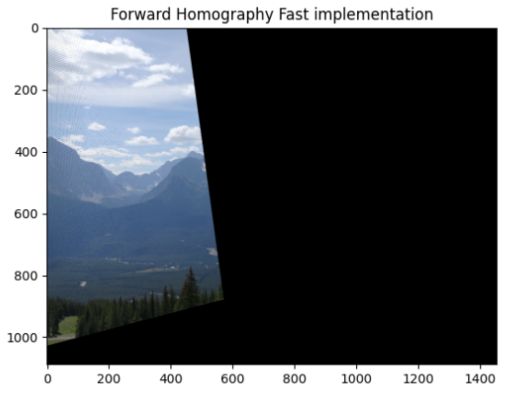


Figure 3: Forward homography fast implementation results

6. Problems with forward mapping

The main problem with forward mapping is caused when pixels in the destination photo does not have a transformed pixel from the source photo. This causes empty holes in the destination, as can be seen in Figure (4) below. One can clearly see black spaces between valid pixels.

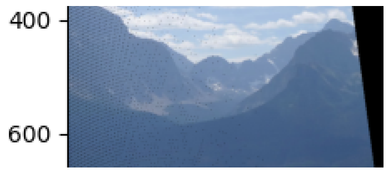
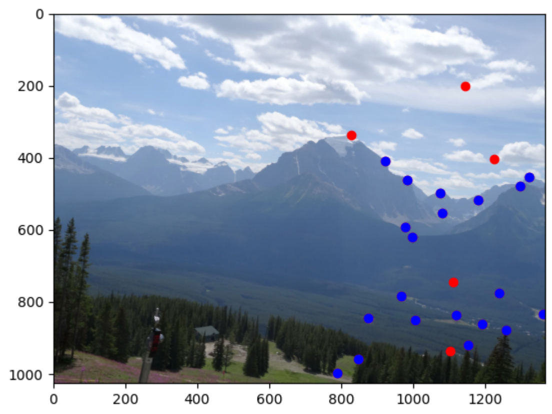


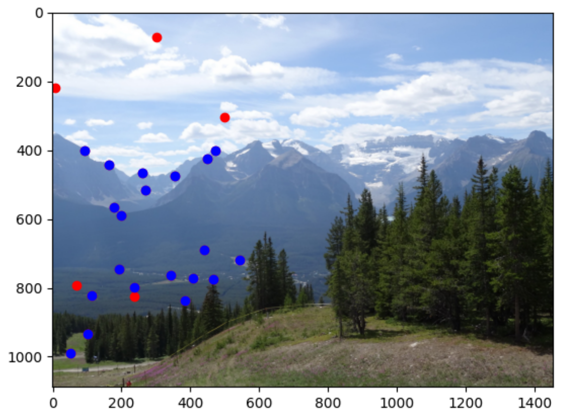
Figure 4: Forward homography problem demonstration

7. Repeat with imperfect matches

In this section we repeat subsections () using the matching points with outliers. the matching points with outliers are shown in Figures (5a) and (5b) below. We can see that on top of the 20 matching coordinate pairs, there are now 5 coordinate pairs with are not matching. These unmatching pairs will introduce heavy noise into the computation of the homography which in turn will distort the transformation of the source picture to the destination.



(a) Source photo, Inliers (blue) and outliers (red)



(b) Destination photo. Inliers (blue) and outliers (red)

Figure 5: Used images with coordinate pairs shown including outliers

The new homography resulted with the computation including the outliers is shown in equation (12) below. we can see that comparing to the noiseless case in equation (11), the outliers have completely changed the homography, resulting in a non-functional transformation.

$$\boldsymbol{H}_{with\ outliers\ naive} = \begin{pmatrix} -5.86018376e-01 & -1.31259752e-01 & 6.25479638e+02 \\ -6.25851768e-01 & -4.85276777e-01 & 8.17143199e+02 \\ -9.48023409e-04 & -3.85199768e-04 & 1.00000000e+00 \end{pmatrix} \tag{12}$$

An example for the resulted transformation is seen if Figure (6) below. We can see that the transformed photo from the source to the destination does not make any sense.

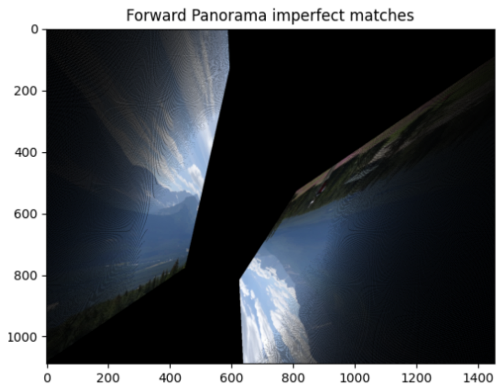


Figure 6: Forward homography result with bad outlier transformation

Part B: Dealing With Outliers

8. Homography testing

In this section we implemented the ”test_homography” API to decide the percent of inliers for a given homography and the MSE of the transformed inliers from their respective target destination. The results for the coordinate pairs without inliers are

- $\textit{inlier percent} \;=\; 1$
- $MSE \;=\; 4.64\ [pixel]$

These results seem to make sense since all the point are inliers and the MSE is relatively low. The results for the imperfect coordinate pairs are

- $\textit{inlier percent} \;=\; 0.16$
- $MSE \;=\; 456.3\ [pixel]$

These results coincide with Figure (6) in the sense that the MSE is bad, and the resulted picture is very far from the original destination. Moreover, the percent of inliers for the homography shown in (12) is very low which is another indicator that this homography should not be used.

9. Points meeting a suggested model

In this section we implemented the ”meet_the_model_points” API to identify which point-pairs are considered as inliers for the given model. That is , given a homography hypothesis, points-pairs, and a threshold - which points are under max_err distance.

10. RANSAC

For the creation of a model (homography) we need 4 pairs of points, as explained above (subsection 1). given that

- $N = 30$
- $\omega = 0.8$
- $n = 4$

We are requested to find, k, the number of iterations for different confidence levels. Using the following

$$1 - p = (1 - w^n)^k \tag{13a}$$
$$k = \frac{\log(1-p)}{\log(1-w^n)} \tag{13b}$$

We can compute for:

$$p = 0.9 \rightarrow k = \frac{log(1-0.9)}{log(1-0.8^4)} \approx 4.369 \tag{14a}$$
$$p = 0.99 \rightarrow k = \frac{log(1-0.99)}{log(1-0.8^4)} \approx 8.739 \tag{14b}$$

This means that for 90% confidence we would need to raffle 5 different sets of points and for 99% confidence we would need to raffle 9 sets of points. The total number of raffles possible is

$$\binom{N}{n} = \binom{30}{4} = 27405 \tag{15}$$

11. RANSAC implementation

In this subsection we implemented the RANSAC function using the "compute_homography" API. The function raffles 4 coordinate pairs from the given pool, computes a homography matching the raffled coordinates and checks if this homography fits the other coordinates in the given pool of coordinates. This process is repeated k times for confidence level of 99% and the inputted probability that a raffled coordinate will be an inlier. After the iterations are over, one last homography is computed using the best raffled model's matching inliers. There are two options:

- In the case where the best raffled model successfully isolated all the inliers from the outliers, the last computed homography will improve in MSE and the last computed homography will be returned
- In the case where in the best homography raffled during RANSAC there was an oulier withing the inlier range, the last homography will perform worse than the RANSAC result and the RANSAC result will be returned

12. RANSAC results

In this section, the database of coordinates was loaded with outliers, and the RANSAC method was used to find the homography. Let us note that this database consists of 25 coordinate pairs, 5 of which are outliers, i.e. 80% inliers. The RANSAC algorithm was able to find a model fitting all the inliers and the resulted homography is shown in (16) below. We can see that the RANSAC algorithm was able to ignore all the outliers and that the resulted homography is identical to the homography computed using only the inliers shown in (11)

$$\boldsymbol{H}_{RANSAC\ naive} = \begin{pmatrix} 1.43457214e+00 & 2.10443232e-01 & -1.27718679e+03 \\ 1.34265153e-02 & 1.34706123e+00 & -1.60455872e+01 \\ 3.79279298e-04 & 5.56523146e-05 & 1.00000000e+00 \end{pmatrix} \tag{16}$$

Let us note that since the RANSAC homography is identical to the outlier-free homography, the transformed source picture should be identical to Figure (3) but for the sake of completeness, the results is also shown in (7) below.

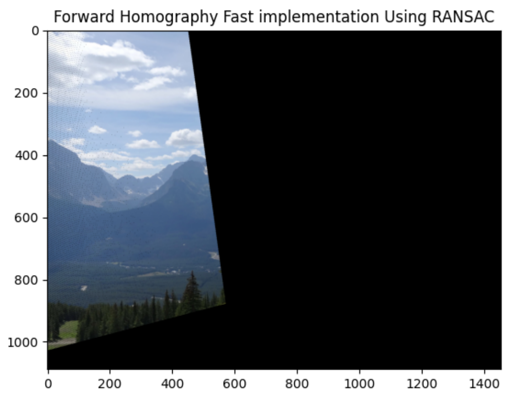
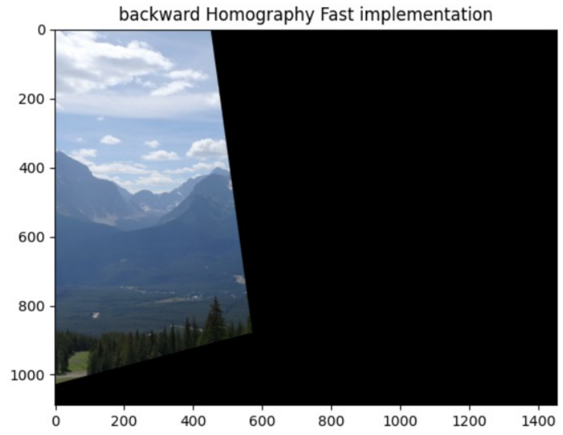


Figure 7: Source image after transformation using RANSAC

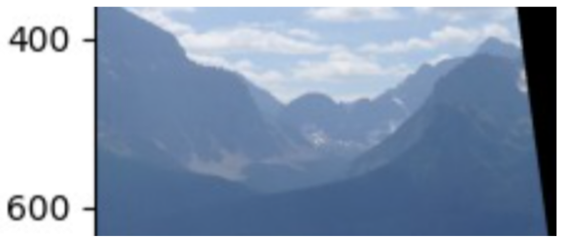
Part C: Panorama Creation

13. Backward mapping

In this part of the exercise the backward mapping was implemented, avoiding the "holes" that the forward mapping creates. This was done by inverting the forward homography and transforming all the coordinates of the destination to the source coordinates. this gave us a floating point representation of the coordinate in the destination. This is not sufficient since the known values in the source are the integer values of the source grid coordinates. Thus, a bi-cubic interpolation was done from the known coordinates of the source grid to the floating point representation of the destination grid in the source coordinates. The points were then planted in their respective place in the destination coordinate image. The result of the backward mapping including the interpolation can be seen in Figure (8a) below. One can see in Figure (10b) that the problem of the forward mapping is solves both by the backward mapping and the interpolation done. This backward mapping was dome using the "comptue_backward_mapping" API, as requested in the exercise.



(a) Backward homography results



(b) Backward zoom in

Figure 8: Backward homography and interpolation solution

14. Adding translation to backward homography

As can be seen in Figure (8a), some of the source pixels are matching negative coordinates which means that a translation should be added after the transformation. this can be seen in (17) below for the example of a single coordinate (u, v) in the source.

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ k \end{pmatrix} = \boldsymbol{H}_{translation}\boldsymbol{H}_{RANSAC} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{17}$$

This means that for the backward mapping, which uses the inverse homography, an inverse translation should be added as follows for coordinate (u', v') in the destination.

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ k \end{pmatrix} = (\boldsymbol{H}_{translation}\boldsymbol{H}_{RANSAC})^{-1} \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = \boldsymbol{H}^{-1}_{RANSAC}\boldsymbol{H}^{-1}_{translation} \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} \tag{18}$$

The translation addition is implemented in the "add_translation_to_backward_homography" API which uses as input the backward projective homography. Thus, the inverse of the needed translation should be added in the form of right matrix multiplication of the inverse homography, as seen in (18), resulting in the complete backward projective and backward translation homography.

15. Panorama creation

In this subsection, the implementation of the panorama creation was done. The implementation was done under the "panorama" API, and the algorithm for the panorama creation is as follows.

1. Compute the forward homography using the coordinate pairs given in the input using RANSAC to overcome outliers
2. Perform forward transformation to the corners of the source image and determine the panorama shape
3. Compute backward homography by inversing the forward homography
4. Add the backward translation needed t plant the transformed source image to the correct location in the panorama as depicted in (18)
5. Create an empty panorama image and place the backward warped source image in the panorama
6. Place the destination image in the correct location, overriding the source image pixels
7. Clip pixel values lower than 0 and higher than 255
8. Convert to uint8

16. Panorama results for given images

The panorama creation algorithm depicted in the previous subsection was executed over the given "src.jpg" and "dst.jpg" images with the imperfect coordinate pairs dictionary from the "matches.mat" file. The inlier percentage was set to 80% and a maximum error of 25 pixels was used. The resulted panorama is shown in (9) below.

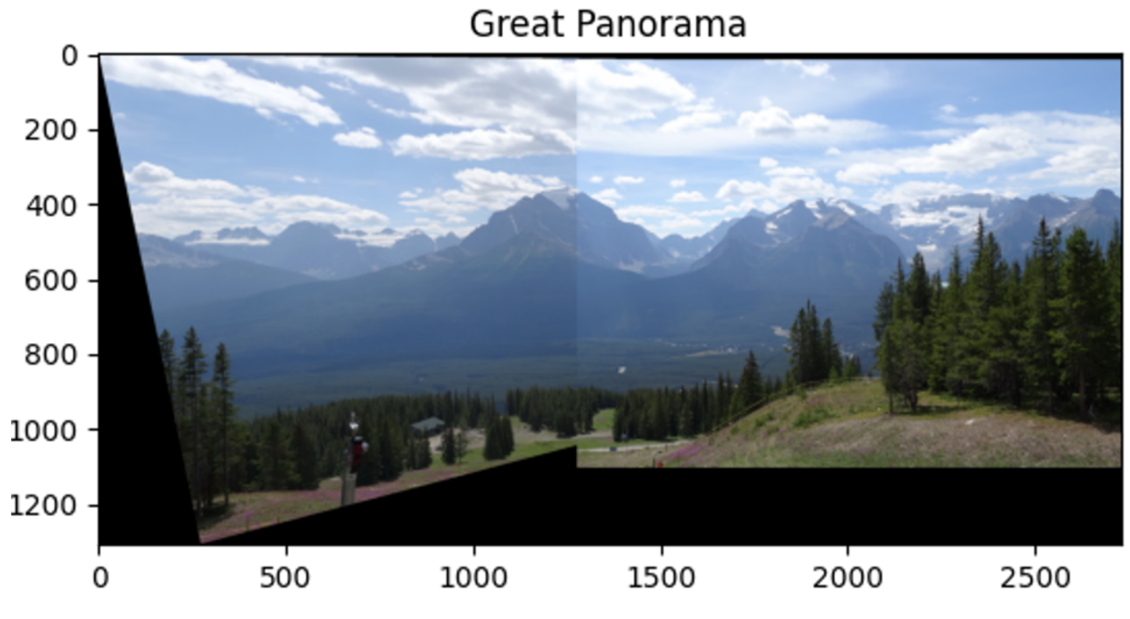
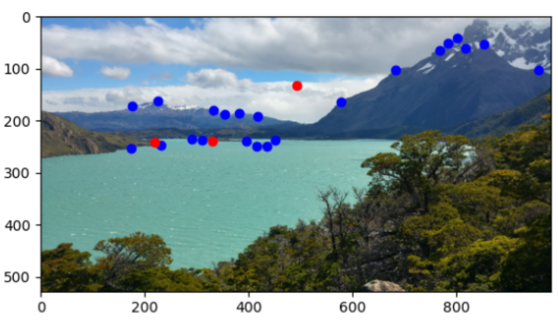


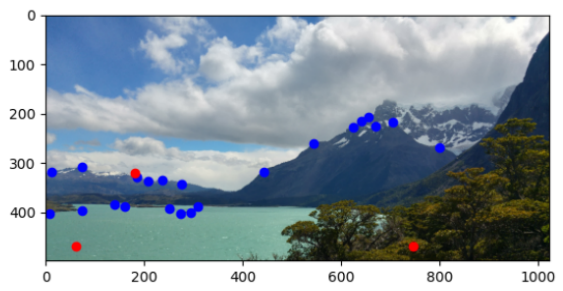
Figure 9: Resulted panorama, given photos

17. Panorama results using our images

The images used in this subsection can be seen in (10) below. On top of the images we can see the selected coordinate pairs where the inliers are colored blue and the outliers and colored red. There are a total of 25 coordinate pairs, 3 of which are outliers. This is more than the required 10% outliers requested in the exercise.



(a) Source image with coordinates scattered



(b) Destination image with coordinates scattered

Figure 10: Custom photos used for panorama

The resulted panorama using the two photos and the coordinate pairs shown in Figure (10) can be seen below.

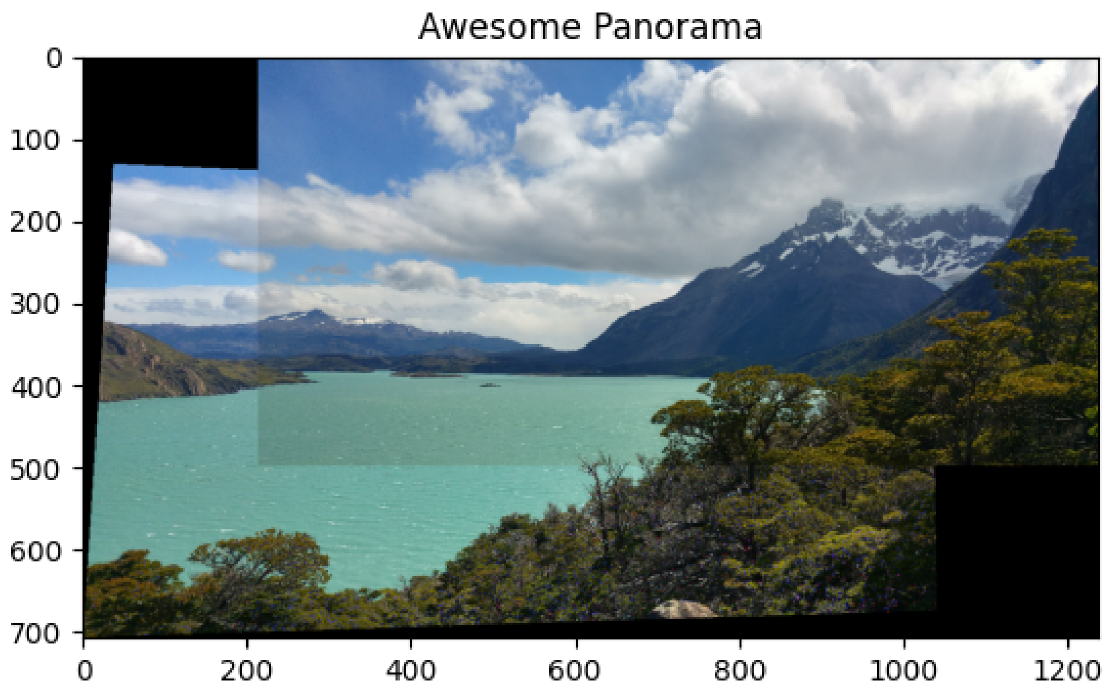


Figure 11: Resulted panorama, custom photos