1 Part A

1.1 Build a system of equations of the form $Ax = b$, as learned in class, for projective transformation. Attach the formula development to your exercise solution. How do we get the conversion matrix from the equation system?

A Homography matrix, H, is a matrix used to transform the pixels of one image (represented by $\{x'\}_i^n$) in its own coordinate system into the coordinate system of another image. This allows the 2 images to be combined into a panorama. The problem is defined as

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} \cong \mathbf{H} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{1}$$

in order to find H we want to make a set of equations, from dividing the 1st row and and 2nd row by the 3rd we get:

let

$$p = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2}$$

so then

$$\frac{u'}{1} = \frac{H_1 p}{H_3 p} \qquad\qquad \frac{v'}{1} = \frac{H_2 p}{H_3 p} \tag{3}$$

$$\Rightarrow u' H_3 p - H_1 p = 0 \qquad\qquad v' H_3 p - H_2 p = 0 \tag{4}$$

This can be written in matrix form as

$$\begin{pmatrix} -p^T & 0 & u'p^T \\ 0 & -p^T & up^T \end{pmatrix} \cdot \begin{pmatrix} H_{11} \\ H_{22} \\ \vdots \\ H_{33} \end{pmatrix} = 0$$

This is an equation of the form $Ax = b$ where

$$A = \begin{pmatrix} -p^T & 0 & u'p^T \\ 0 & -p^T & up^T \end{pmatrix} \qquad\qquad x = \begin{pmatrix} H_{11} \\ H_{12} \\ \vdots \\ H_{33} \end{pmatrix} \tag{5}$$

We see that a pair of matching points gave us a set of 2 equations. To find the homography matrix it is enough to find 8 parameters (Since the transformation is defined up to scale, generally $H_{33} = 1$), so 4 matching points is the minimal number. Writing this explicitly, we find that:

$$\begin{pmatrix} -u & -v' & -1 & 0 & 0 & 0 & uu' & v'u' & u' \\ 0 & 0 & 0 & -u & -v' & -1 & uv' & v'v' & v' \\ & & & & \vdots & & & & \end{pmatrix} \cdot \begin{pmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \\ H_{33} \end{pmatrix} = 0$$

In general, the A matrix will be $2N \times 9$, where N is the number of matching points.

The least squares solution can then be used to find the x vector that minimizes the least squares error, $||Ax||$. The standard choice is to set $||x|| = 1$ to avoid trivial solutions.

To solve this, a loss function is defined as

$$L = \mathbf{x}^T \mathbf{A}^T \mathbf{A} \mathbf{x} - \lambda \left| \mathbf{x}^T \mathbf{x} - 1 \right|$$

Taking the derivative w.r.t. x we find

$$\mathbf{A}^T \mathbf{A}\, \mathbf{x} = \lambda\, \mathbf{x}$$

Then choosing x that is the smallest eigenvalue of $A^T A$ is the x that minimizes the error. Practically, this can be solved using SVD decomposition, or simple search for all eigenvectors and values. Finally, the x vectors elements are refactored into a 3×3 H matrix.

1.2 Write a function that estimates the transformation coefficients from source (src) to destination (dst), from the equation system in section 1.

See code for the function.

1.3 Load the matches_perfect.mat file and calculate the transformation coefficients using the compute_homography_naive function. Present the result.

Results can be seen in figure 1

```
Naive Homography 0.0048 sec
[[ 1.43457189e+00  2.10443271e-01 -1.27718660e+03]
 [ 1.34263476e-02  1.34706110e+00 -1.60454441e+01]
 [ 3.79279048e-04  5.56524152e-05  1.00000000e+00]]
```

Figure 1: Naive Homography Matrix Example

1.4 Implement the transformation function from the source to the destination using the Forward Mapping transform. Use the homography from (3) to display the source image after a projective transformation. The result image should be in the size of the dst image. See function documentation for specific implementation details.

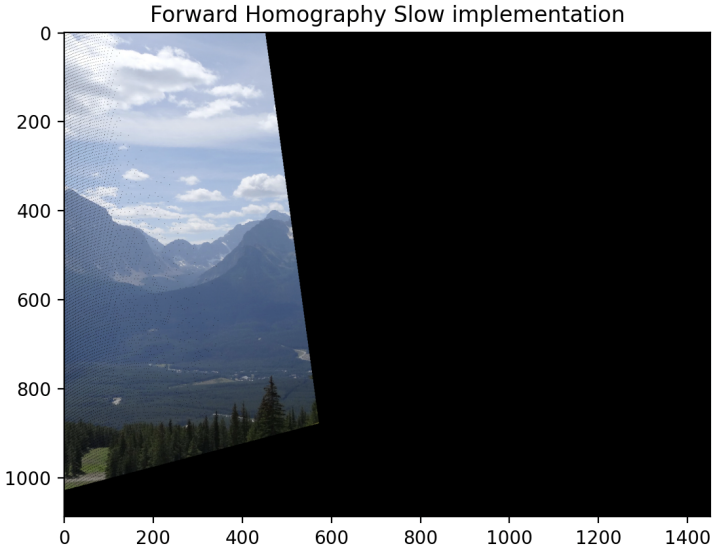
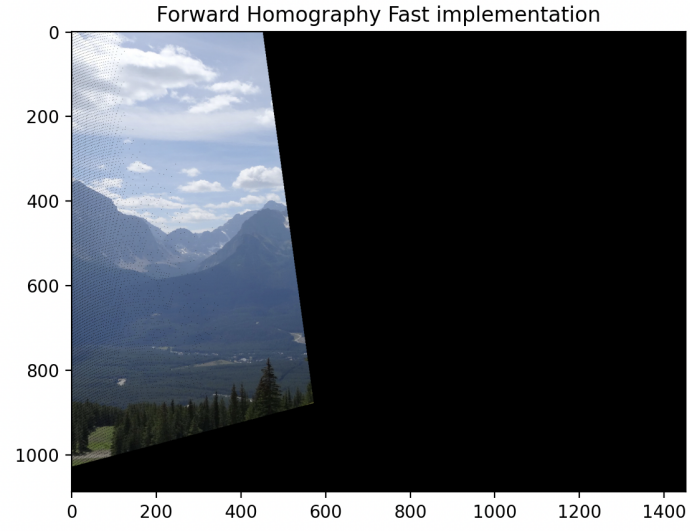


Figure 2: Forward Mapping

1.5 Re-Implement the transformation function from the source to the destination using the Forward Mapping transform. Don't use loops.

See code for the function. Results can be seen in figures 1.5 (fast and slow prduce the same result, just at different computation speeds) and 3




```
Naive Homography 0.0048 sec
[[ 1.43457189e+00  2.10443271e-01 -1.27718660e+03]
 [ 1.34263476e-02  1.34706110e+00 -1.60454441e+01]
 [ 3.79279048e-04  5.56524152e-05  1.00000000e+00]]
Naive Homography Slow computation takes 12.0394 sec
Naive Homography Fast computation takes 0.0803 sec
```

Figure 3: Naive Homography Matrix Example

1.6 What are the problems with Forward Mapping and how are they reflected in the image you received?

Forward mapping moves each pixel from the source image to a new location using the homography. This leads to two main problems:

- Empty Pixels: Some pixels in the destination image are not assigned any value because the mapped coordinates are usually not integers. Rounding leaves gaps, as seen by the black dots in the produced image.

- Overlapping pixels: Multiple source pixels can map to the same destination pixel. The last one overwrites the others, which can make parts of the image look smeared.

1.7 Now load the matches.mat file, and repeat items 1.5 and 1.6. Did you get a different result? Explain. The image may be too large to display, specify. (You can shrink it and then present it)

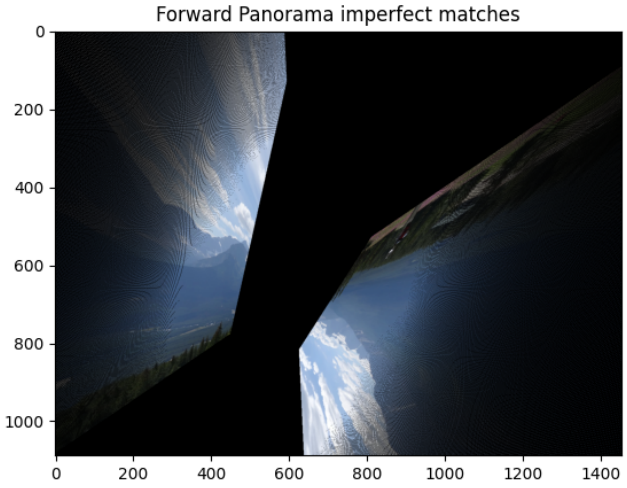


Figure 4: Fast Forward Mapping Using Imperfect Matches

Using an imperfect set of matching points, with no RANSAC algorithm to discount them severely distorts the image, as even a single outlier will cause the whole homography to be incorrect. This can clearly be seen in figure 4;

2 Part B: Dealing with outliers

2.1 Implement a function that calculates the quality of the projective transformation model.

The resulting H matrix can be seen in figure 5

```
[[-5.86021548e-01 -1.31260377e-01  6.25482835e+02]
 [-6.25852108e-01 -4.85279696e-01  8.17144029e+02]
 [-9.48024776e-04 -3.85201052e-04  1.00000000e+00]]
Naive Homography Fast computation for imperfect matches takes 0.0609 sec
Naive Homography Test 0.0004 sec
[0.16, 456.29568120050624]
```

Figure 5: H Matrix Using Imperfect Matches

2.2 Before implementing the function which calculates the source-to-target coefficients that deal with outliers using RANSAC, we will implement a helper function which will tell us, given a homography hypothesis, matching point, and a threshold - which points meet the model. That is, which point-pairs are considered as inliers for the given homography under max_err distance.

See Code.

2.3 Suppose there are 30 match points and it is known that 80% of them are correct. What is the number of randomizations needed in this case to guarantee 90% confidence? Of 99%? How many iterations must be done to cover all options?

The number of iterations needed in RANSAC to guarantee a certain confidence level is given by: (as learned in class)

$$N = \frac{\log(1-p)}{\log(1-w^n)}$$

where:

- p = desired probability of having at least one all-inlier sample,
- w = probability a single point is an inlier,
- n = number of points needed to estimate the model.

Here, $w = 0.8$ and for a homography $n = 4$.
1 $p = 0.9$

$$w^n = 0.8^4 = 0.4096, \quad 1 - w^n = 0.5904$$

$$N = \frac{\log(1-0.9)}{\log(0.5904)} = \frac{\log(0.1)}{\log(0.5904)} \approx \frac{-2.3026}{-0.5277} \approx 4.36$$

Round up, so $N = 5$ iterations.
2 $p = 0.99$

$$N = \frac{\log(1 - 0.99)}{\log(0.5904)} = \frac{\log(0.01)}{\log(0.5904)} \approx \frac{-4.605}{-0.5277} \approx 8.73$$

Round up, so $N = 9$ iterations.
3 To cover all combinations, we choose 4 points out of 30:

$$\binom{30}{4} = \frac{30 \cdot 29 \cdot 28 \cdot 27}{4 \cdot 3 \cdot 2 \cdot 1} = \frac{657720}{24} = 27405$$

So, to try every possible combination, $N = 27,405$ iterations are needed.

2.4 Implement a function that calculates the source-to-target coefficients that deal with outliers by using RANSAC (use the functions you built in previous sections).

See code.

2.5 Load the matches.mat file and calculate the transformation coefficients using the compute_homography function. Present the obtained coefficients, as well as the source image after projective transform using forward mapping. Compare the results you got to the results in sections 5 and 7.

The results for the perfect inputs match those of figure 3, which makes sense, as on the very first pass through all points will be inliers, so the early break condition will be triggered. If all points are inliers, RANSAC has no effect. Conversly, when compared to section 1.9, we see dramatic improvement; the outlier have been ignored, so we successfully return a coherent image. The returned coefficients nearly perfectly match those given with the 'matches_perfect.mat' file, as can be seen in figure 6 and **??**;

```
RANSAC Homography 0.0020 sec
[[ 1.43457189e+00  2.10443271e-01 -1.27718660e+03]
 [ 1.34263476e-02  1.34706110e+00 -1.60454441e+01]
 [ 3.79279048e-04  5.56524152e-05  1.00000000e+00]]
RANSAC Homography Test 0.0001 sec
[0.8, 4.642034506382322]
```

Figure 6: Fast Forward Mapping Using Imperfect Matches, and RANSAC

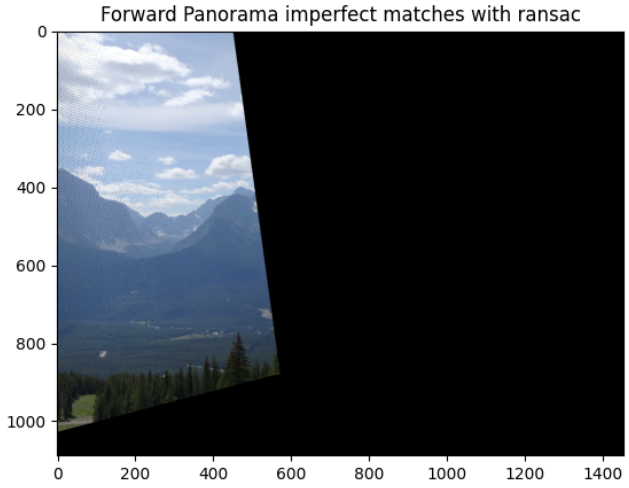


Figure 7: Fast Forward Mapping Using Imperfect Matches, and RANSAC

3 Part C

3.1 Implement the transformation function from the source to the destination using the Backward, which which uses Bi-linear interpolation, and display the source image after a projective transformation, according to the coefficients obtained in section 12. Compare to the image obtained in section 12.

See code.

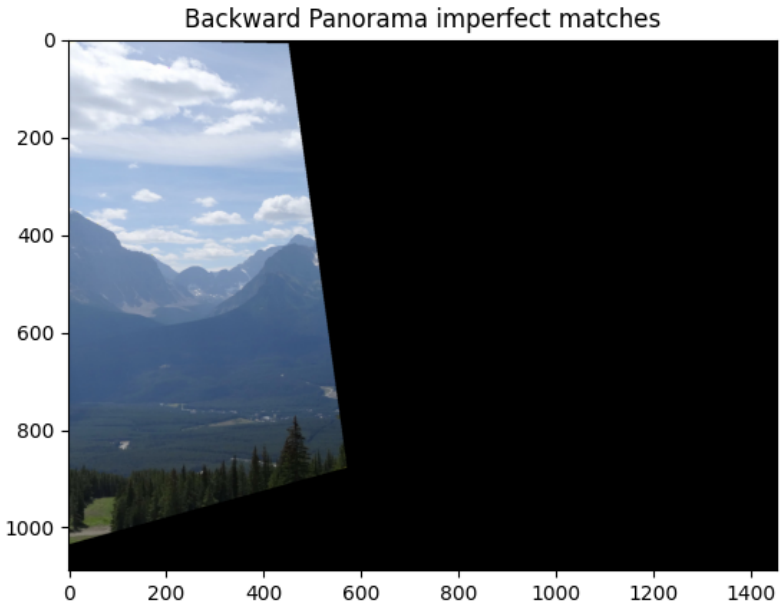


Figure 8: Backward mapping using imperfect matches, and RANSAC

We can see that the homography looks similar, but now there are no holes in it. By using the backward mapping, we overcame these artifacts but with the price of computational complexity in comparison to forward mapping.

3.2 You are given the method which calculates the panorama's shape. Feel free to debug it, if it misses +-1 row and columns. It returns the number of rows and columns of the final panorama, and another struct which holds the padding (in each axis) done to the target image to achieve the panorama's shape. Use this struct to add a translation component to the homography such that the final homography contains both the translation that the source pixels need to undergo before the homography is applied to them

See code.

3.3 Implement a function that produces a panorama image from two images, and two lists of matching points, that deal with outliers using RANSAC (use the functions from previous sections).

See code.

3.4 Run the panorama function for the src.jpg and dst.jpg images, using the points from the matches.mat file. Set 80% inliers and a maximum error of 25 pixels. Present the output panorama.

The final result panorama of the given landscape images:

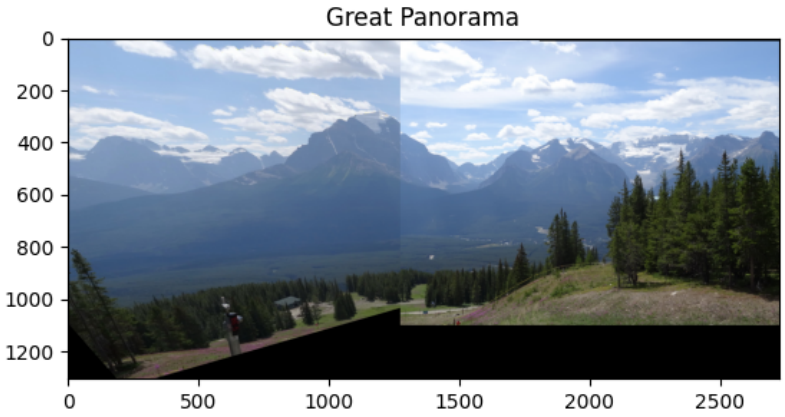


Figure 9: landscape images panorama

3.5 Use a pair of your own images to build a panorama (using the panorama function). Images should be called src_test.jpg and dst_test.jpg. Note that it is preferable to use images of the same size. Run the create_matching_points.py file to produce the matches_test.mat points file with 25 matching points. Make sure there are at least 10Present the input images, along with the marked matching points, and present the output panorama.

The picked images are:

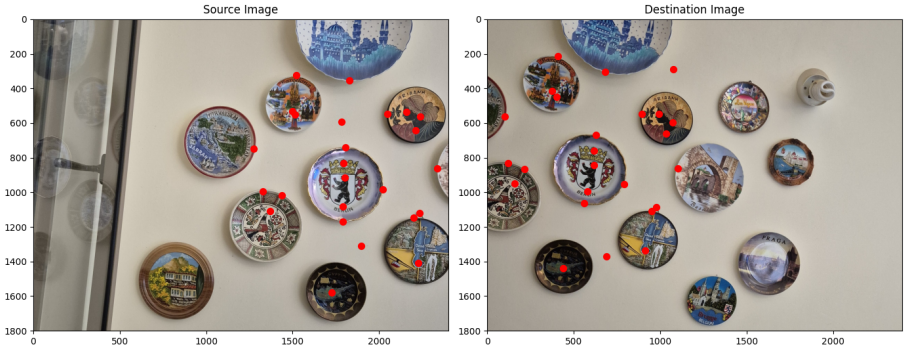


Figure 10: Our images for panorama, with chosen matching points

The panorama result:

Figure 11: Our images' Panorama

Wow it's so cool!
The reversed panorama result:

Figure 12: Our images' reversed Panorama