Computer Vision - Assignment 1

Daniel Soref (204798342) & Amit Haimovich (209804475)

November 13, 2025

1. A homography is a 2D projective transformation that maps points from one image plane (the source) to another (the destination). Let $\underline{x} = \begin{pmatrix} u & v & 1 \end{pmatrix}^T$ denote a set of coordinate in the source image and $\underline{x}' = \begin{pmatrix} u' & v' & 1 \end{pmatrix}$ the corresponding point in the destination. The transformation is done by $\underline{x}' \tilde{=} H\underline{x}$ when H is a 3x3 homography matrix: $H = \begin{pmatrix} -H_1- \\ -H_2- \\ -H_3- \end{pmatrix}$. Since H is defined up to scale, it contains eight degrees of freedom, meaning that at least four point correspondences are required to compute it. Writing the quality explicitly:

$$\begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} \tilde{=} \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} y \\ v \\ 1 \end{pmatrix}$$

representing a set of 2 equations: $\frac{u'}{1} = \frac{H_1 u}{H_3}$, $\frac{v'}{1} = \frac{H_2 v}{H_3}$. Rearranging these equations to form a homogeneous system (where the right-hand side is zero) gives:

$$H_1 u - u' H_3 = 0$$
$$H_2 v - v' H_3 = 0$$

Each point correspondence $(u_i, v_i) \to (u_i', v_i')$ provides two such linear equations. Given N correspondences, we can stack these $2N$ equations into a single matrix system $A\underline{h} = 0$:

$$\begin{pmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1'u_1 & -u_1'v_1 & -u_1' \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -v_1'u_1 & -v_1'v_1 & -v_1' \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -u_2'u_2 & -u_2'v_2 & -u_2' \\ 0 & 0 & 0 & u_2 & v_2 & 1 & -v_2'u_2 & -v_2'v_2 & -v_2' \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_N & v_N & 1 & 0 & 0 & 0 & -u_N'u_N & -u_N'v_N & -u_N' \\ 0 & 0 & 0 & u_N & v_N & 1 & -v_N'u_N & -v_N'v_N & -v_N' \end{pmatrix} \cdot \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ \vdots \\ \vdots \\ h_{33} \end{pmatrix} = 0$$

We want to find a non-trivial (non-zero) solution $\underline{h}$ that solves the homogeneous system $A\underline{h} = 0$. Therefore, we need to find the vector $\underline{h}$ that best minimizes the error $||A\underline{h}||^2 = h^T A^T A h$, subject to a constraint $||\underline{h}|| = 1$, to prevent the trivial solution. As shown in class, this solution $\underline{h}$ is the eigenvector of the 9×9 matrix $A^T A$ corresponding to its smallest eigenvalue. This 9×1 eigenvector is then extracted and reshaped into the final 3×3 homography matrix H.

2. We wrote e a function that estimates the transformation coefficients from source (src) to destination (dst), from the equation system in section 1.

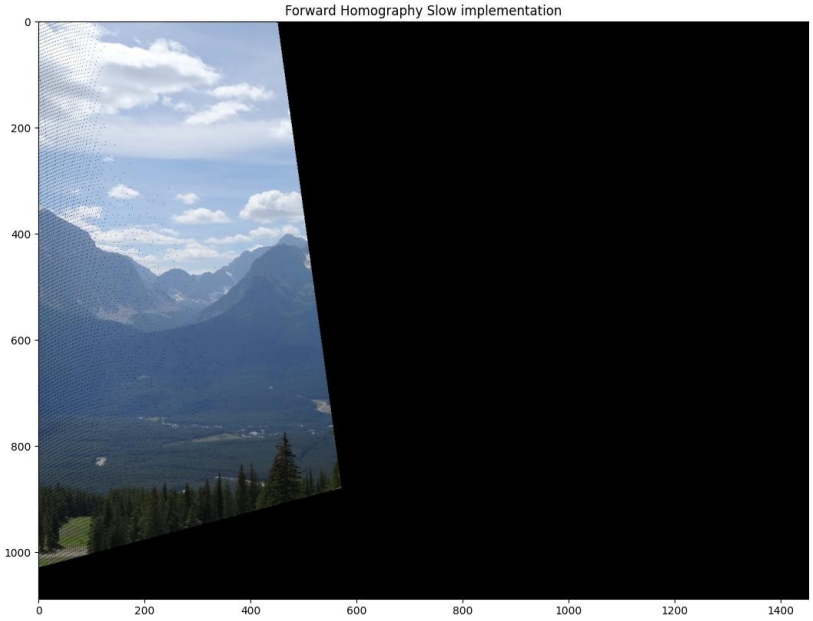
3. We have calculated the transformation coefficients using the 'compute_homography_naive' function:

   [[-1.12313781e-03 -1.64757662e-04 9.99919585e-01]

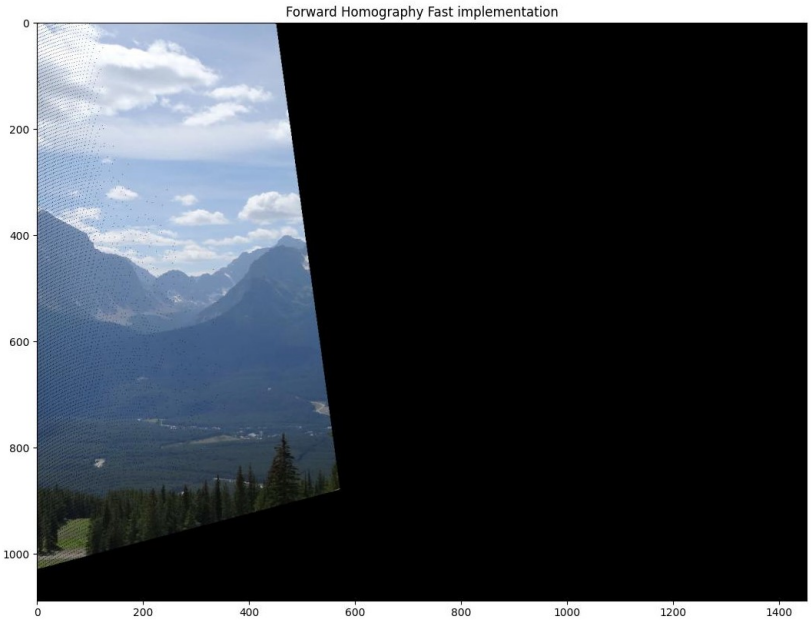
   [-1.05117244e-05 -1.05462483e-03 1.25622164e-02]

   [-2.96940746e-07 -4.35706349e-08 -7.82907867e-04]]

4. Below is the output of 'compute_forward_homography_slow' function:

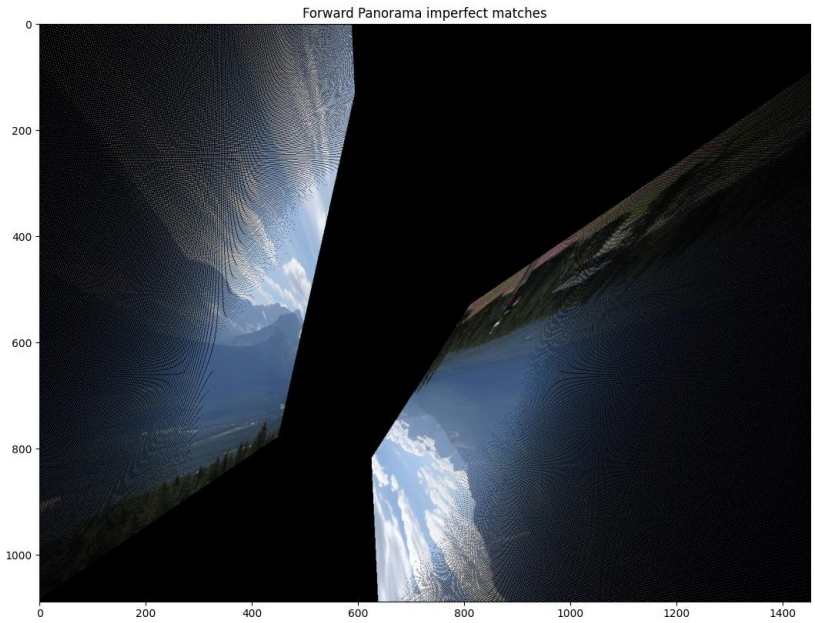


5. Below is the output of 'compute_forward_homography_fast' function:



6. The forward mapping creates a warped image that is "riddled with holes" or looks "speckled" because our loop iterates over the source pixels, not the destination pixels. When the image is stretched, this method doesn't guarantee that every pixel in the destination grid will be filled. A source pixel might map to (10, 10) and the next source pixel might map to (12, 10), completely skipping the destination pixel at (11, 10) and leaving it black. This also has the opposite problem, "splatting," where multiple source pixels map to the same destination pixel when the image is compressed, and the last pixel to be written just overwrites the previous ones.

7. Below is the resulting warped image from using matches.mat, which contains outliers:



As we can see, the result is completely different. When using matches.mat, the resulting image is heavily distorted and unrecognizable, unlike the correct warp we got from matches_perfect.mat. The reason is that matches.mat contains outliers (incorrectly matched points). Our 'compute_homography_naive' function is not robust, it assumes all matches are good. It uses a least-squares method, which tries to find a single best-fit for all points. A few bad outliers with large errors pull the entire calculation away from the true solution, resulting in a completely incorrect homography matrix. When this "wrong" homography is applied, the source image is transformed into a badly distorted shape. On top of this, the speckling and hole-artifacts from forward mapping (Section 6) are still present.

8. We have implement the function 'test_homography'.

9. We have implement the function 'meet_the_model_points'.

10. suppose there are N=30 matching points, and 80% are correct (inliers), so w=0.8. For a homography we need to sample n=4 points. if we require confidence of 90% (p=0.9), we get according to the formula learned in class:

$$k_{90} = \left\lceil \frac{\log(1-p)}{\log(1-w^n)} \right\rceil = \left\lceil \frac{\log(1-0.9)}{\log(1-0.8^4)} \right\rceil = \lceil 4.3696 \rceil \rightarrow \text{we need 5 iterations for 90\%}$$

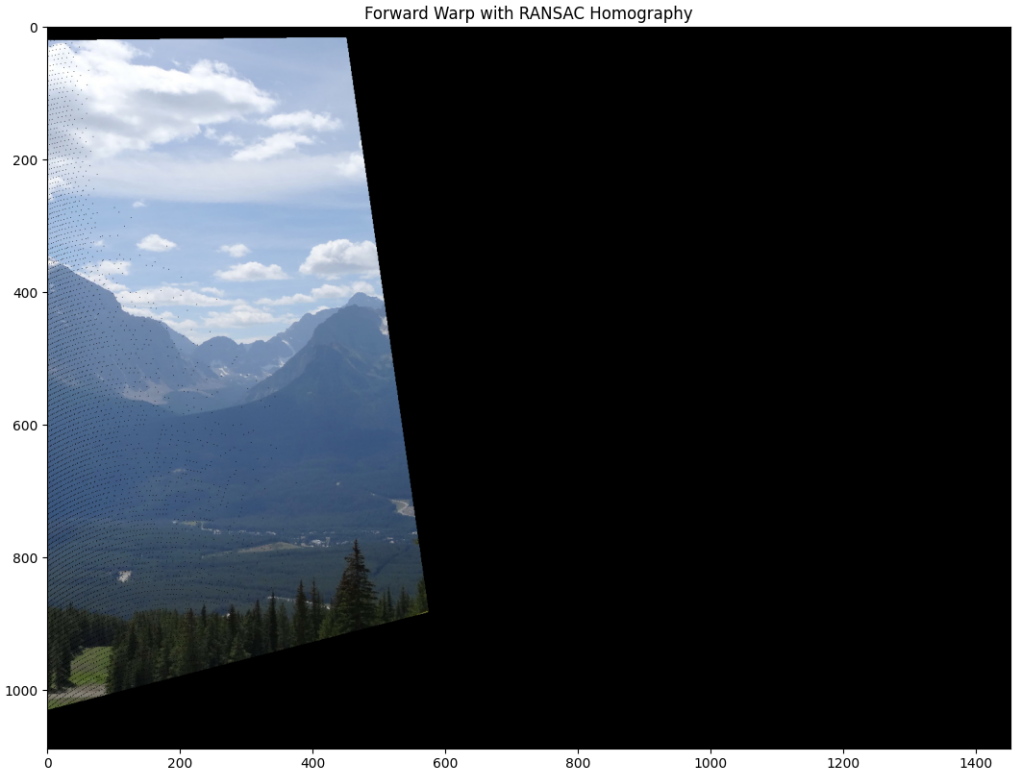
For p=0.99, in the same way we get:

$$k_{99} = \left\lceil \frac{\log(1-p)}{\log(1-w^n)} \right\rceil = \left\lceil \frac{\log(1-0.99)}{\log(1-0.8^4)} \right\rceil = \lceil 8.7392 \rceil \rightarrow \text{we need 9 iterations for 99\%}$$

To cover all options, there needs to be done $\binom{30}{4} = \frac{30!}{4!(30-4)!} = 27,405$ iterations.

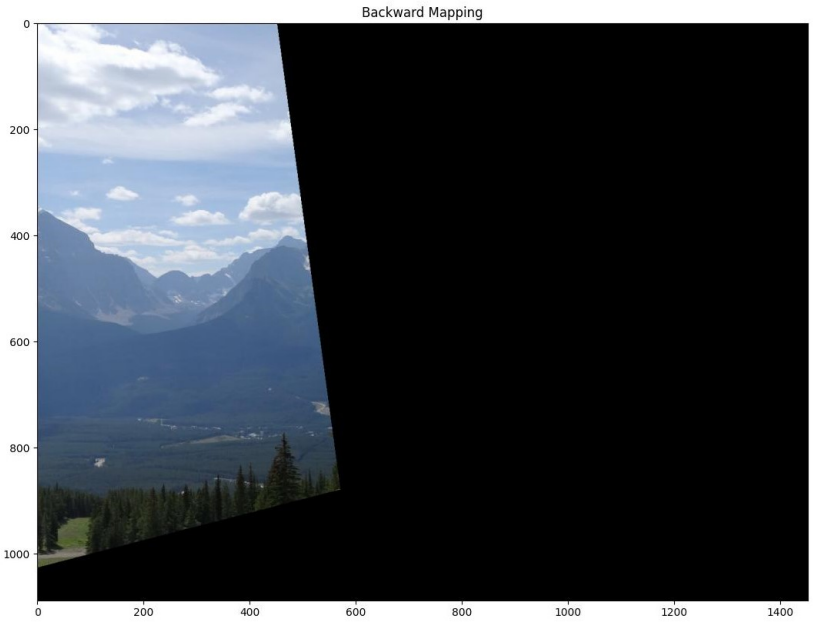
11. We have implement the function 'compute_homography'.

12. We have calculated the RANSAC homography coefficients:
[[1.45025920e+00 2.14193230e-01 -1.29306260e+03]
[1.58369926e-02 1.36054146e+00 -2.00324557e+01]
[3.84527202e-04 6.61047137e-05 1.00000000e+00]]
Source image after projective transform:



We can see that the image no longer appears warped as it did in Section 7. However, the computed homography is slightly different from that in Section 5, likely due to minor mismatch errors within the acceptable range. As expected, black dot artifacts are still present.

13. Below is the output of Backward Mapping:

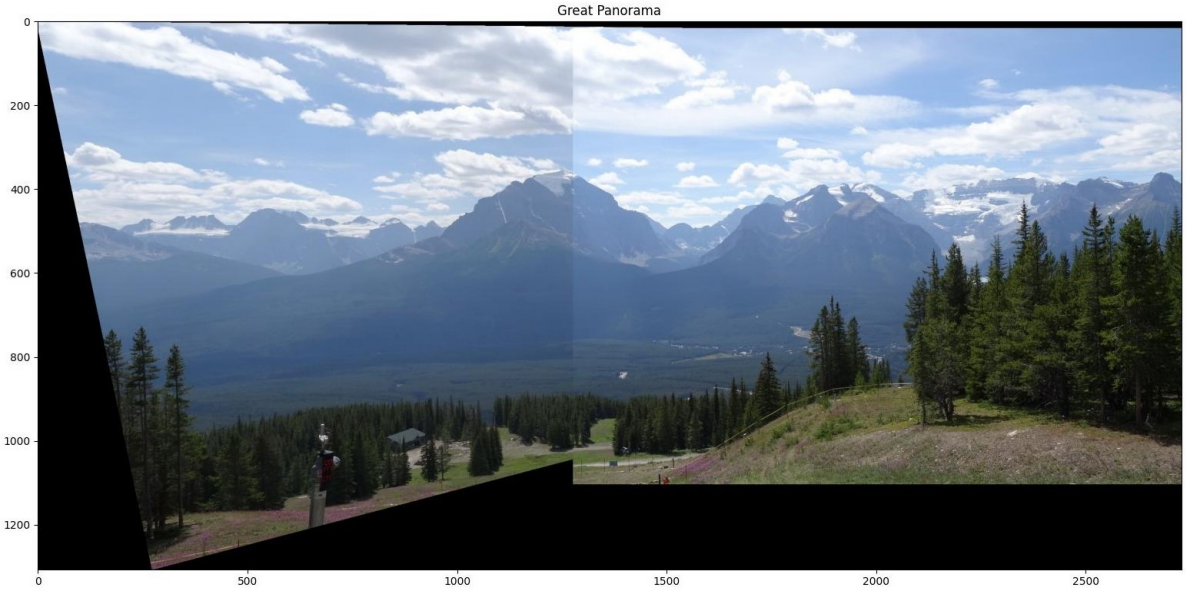


We can see that the result is identical to the one in Section 12 (since the same random seed was used), except that this version no longer contains the artifacts.

14. We have implement the function 'add_translation_to_backward_homography'.

15. We have implement the function 'panorama'.

16. Below is the final panorama, which was created using our RANSAC-based panorama function to stitch the images together:

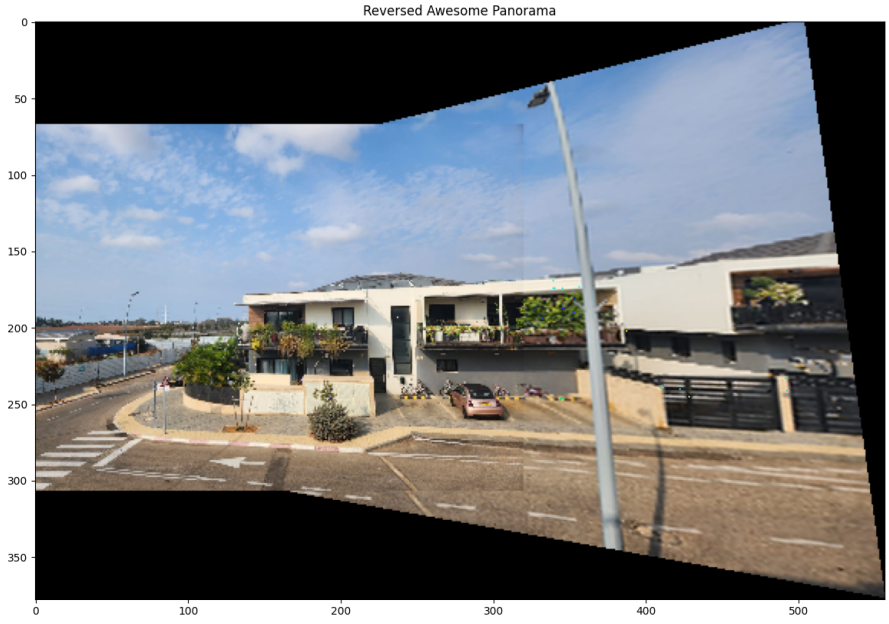


17. We used a pair of our own images to build a panorama, with the following matching points (including inliers and some outliers):

The resulting output from the "your_images_main()" function:


Forward warp example
0
50
100
150
200
0
50
100
150
200
250
300


Backward warp example
0
50
100
150
200
0
50
100
150
200
250
300



These results clearly demonstrate the saaignment progression. The image 'Forward warp example' is filled with "speckling" artifacts, which are solved by the smooth interpolation

shown in the 'Backward warp example'. Using this high-quality backward mapping with our RANSAC homography allowed us to create the final, seamlessly-stitched 'Awesome Panorama' and 'Reversed Awesome Panorama'.