הפקולטה להנדסה
ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

The Iby and Aladar Fleischman
Faculty of Engineering
Tel Aviv University

# Assignment 1 - Panorama

Computer Vision / 0510-6251

By

Nimrod Curtis 311230924

Tel Aviv University

Winter 2024

## Contents:

## A. Homography computation

### 1. Constructing the homography matrix

The homography matrix (also called the projective transformation matrix) is the mapping of one image plane to another through a point.
And can be written by the linear equation as follow:

$$x_2 = Hx_1 \qquad (1)$$

Where:
$x_1 = [\, u_1 \,, v_1 \,, 1 \,]^T$ - a point in image 1 (the source) coordinate system
$x_2 = [\, u_2 \,, v_2 \,, 1 \,]^T$ - the point in image 2 (the destination) coordinate system
$H = \begin{bmatrix} - H_1 - \\ - H_2 - \\ - H_3 - \end{bmatrix}_{3X3}$ - the homography matrix .

Homography can only be define up to scale. If we fix scale such that $\sqrt{\sum h_{ij}^2} = 1$, then we will get **8 free parameters**. Means that for finding the parameters we can provide **4 pairs of matching points**.
For a given pair $i$ of corresponding points, by using the equation (1) :

$$u_2^{(i)} = \frac{h_{11}u_1^i + h_{12}v_1^i + h_{13}}{h_{31}u_1^i + h_{32}v_1^i + h_{33}} \qquad (2)$$

$$v_2^{(i)} = \frac{h_{21}u_1^i + h_{22}v_1^i + h_{23}}{h_{31}u_1^i + h_{32}v_1^i + h_{33}}$$

Rearranging the terms in matrix form:

$$\begin{bmatrix} u_1^1 & v_1^1 & 1 & 0 & 0 & 0 & -u_1^1u_2^1 & -v_1^1u_2^1 & -u_2^1 \\ 0 & 0 & 0 & u_1^1 & v_1^1 & 1 & -u_1^1v_2^1 & -u_1^1v_2^1 & -v_2^1 \\ | & | & | & | & | & | & | & | & | \\ u_1^n & v_1^n & 1 & 0 & 0 & 0 & -u_1^nu_2^n & -v_1^nu_2^n & -u_2^n \\ 0 & 0 & 0 & u_1^n & v_1^n & 1 & -u_1^nv_2^n & -u_1^nv_2^n & -v_2^n \end{bmatrix}_{2nX9} \begin{bmatrix} h_{11} \\ | \\ h_{33} \end{bmatrix}_{1X9} = \begin{bmatrix} 0 \\ | \\ 0 \end{bmatrix} \qquad (3)$$

Now we can solve the equation $Ah = 0$ such that $||h||^2 = 1$. Knowing that $||Ah||^2 = h^TA^TAh$ we can solve the least square problem of finding : $\min\,(h^TA^TAh)$ such that $||h||^2 = h^Th = 1$ . Defining the **Loss function** $\boldsymbol{L(h,\lambda) = h^TA^TAh - \lambda(h^Th - 1)}$ we will get the **Eigenvalue problem of** $\boldsymbol{A^TAh = \lambda h}$. Therefore the **eigen vector** $\boldsymbol{h}$ with the smallest eigen value $\lambda$ of matrix $A^TA$ will minimizes the loss function $L(h,\lambda)$ . This eigen vector is indeed the **homography matrix**.

## 2. Transformation matrix estimation function

**Written in the attached code.**

## 3. Results matrix for the given problem

Given:

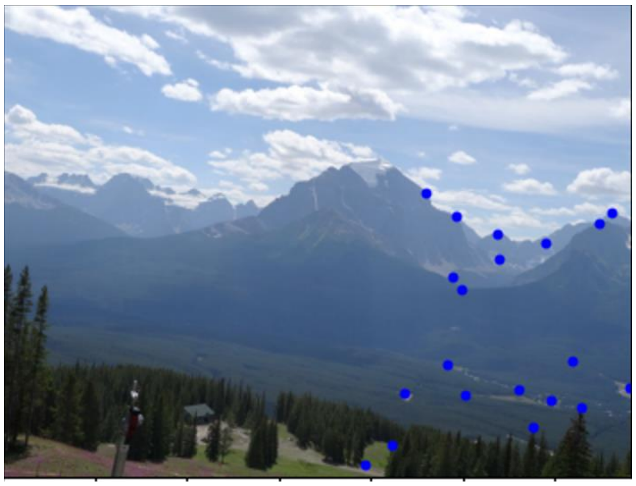
*Figure 1 – source image with matching points*

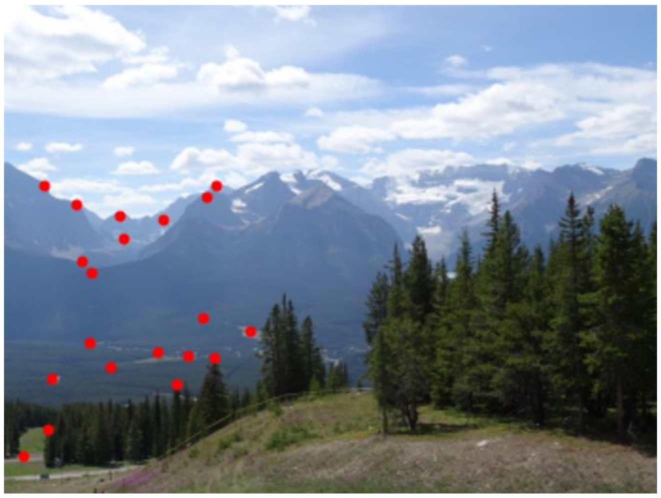
*Figure 2 - destination image with matching points*

The homography matrix is:

$$h = \begin{bmatrix} 1.43457214 & 2.10443232\mathrm{e}^{-1} & -1.27718679\mathrm{e}^{3} \\ 1.34265154\mathrm{e}^{-2} & 1.34706123 & -1.60455873\mathrm{e}^{1} \\ 3.79279298\mathrm{e}^{-4} & 5.56523147\mathrm{e}^{-5} & 1. \end{bmatrix}$$

## B. Forward mapping (slow&fast)

### 4. Slow forward mapping

Result source image tarnsformed to the destination using the projective matrix:

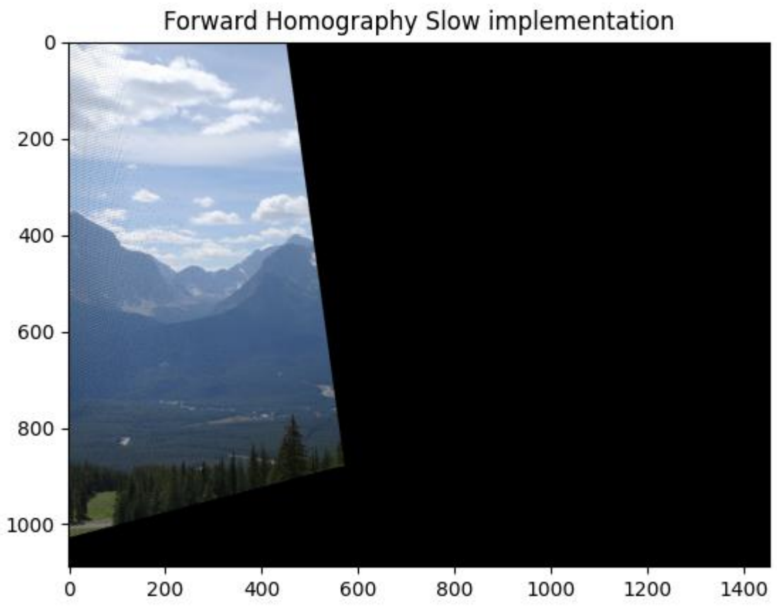

*Figure 3 – source image after forward homography of slow implementation.*

The computation time for this implementation is ~4.0 [sec].

### 5. Fast forward mapping

The result image is exactly the same as the figure of the slow function **but** computation time takes is ~0.093 [sec] .

### 6. Problems of forward mapping

The main problems of the forward mapping are:

1. Interpolation Artifacts - integer coordinates mapped to non-integer coordinates.
2. Sparse Sampling - not all points in the destination image may be covered by the transformed points from the source image. As a result, there may be gaps or regions in the destination image that are not filled, leading to incomplete or distorted results.

In the given problem we can see in the left side of the image a black dots which are gaps in the destination that are not filled.

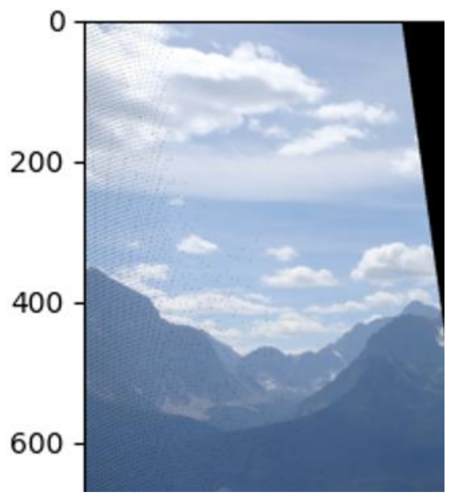

*Figure 4 - forward mapping gaps*

## 7. Imperfect matches – Forward mapping

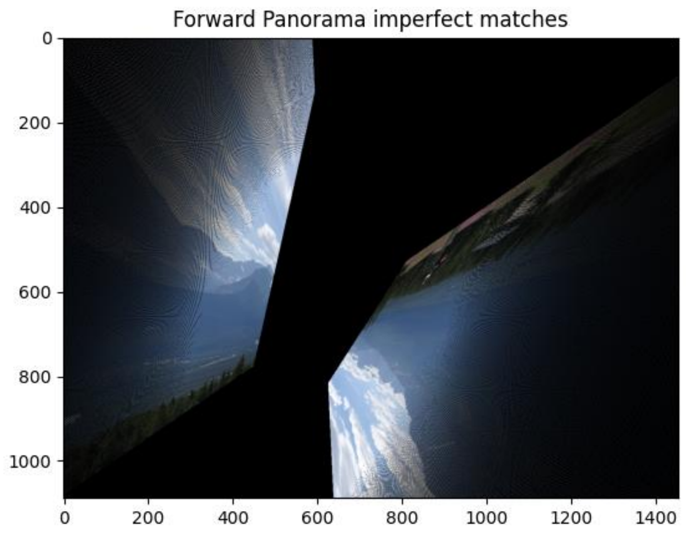

*Figure 5 - imperfect matches panorama*

Above is the image resulting from the raw matches, which were not perfectly aligned due to mismatched correspondence point pairs. The homography calculation this time was leading to wrong transformation of the source image.

## C. Dealing with outliers – RANSAC

### 8. Quality of the projective transformation

**Written in the attached code.**

### 9. Meet the model points

**Written in the attached code.**

### 10. RANSAC formula

The formula for computing the number of iterations in RANSAC is:

$$k = [\frac{\log(1-p)}{\log(1-w^n)}] \tag{4}$$

Where:

$w$ - fraction of inliers

$p$ – target success rate

$n$ – minimum number of points required to fit the model

$k$ – the number of iterations needed based on the parameters above

Given:

$$w = 0.8, p = 0.9, n = 4 \ (for\ determine\ the\ H\ matrix)$$

From equation (4) we will get:

$$k = \left\lceil \frac{\log(1-0.9)}{\log(1-0.8^4)} \right\rceil = \left\lceil \frac{\log(0.1)}{\log(0.59)} \right\rceil = 4.36 + 1 \approx 6 \ [iterations]$$

For $p = 0.99$:

$$k = \left\lceil \frac{\log(1-0.99)}{\log(1-0.8^4)} \right\rceil = \left\lceil \frac{\log(0.01)}{\log(0.59)} \right\rceil = 8.74 + 1 \approx 10 \ [iterations]$$

The number of randomizations for 4 out of 30 elements, without repetition and no importance to order, is: $\frac{m!}{n!(m-n)!} = \frac{30!}{4!26!} = 27405$ [randomizations]

### 11. RANSAC implementation

**Written in the attached code.**

### 12. RANSAC results

Given the imperfect matching points, the homography calculated using RANSAC algorithm for dealing with outliers is the same matrix from question 3 , therefore the warped image is the same. Means the RANSAC algorithm clean the imperfect matches such that it will be equal to the perfect matches.

## D. Panorama creation

### 13. Backward mapping

The result warp image using the backward mapping is:

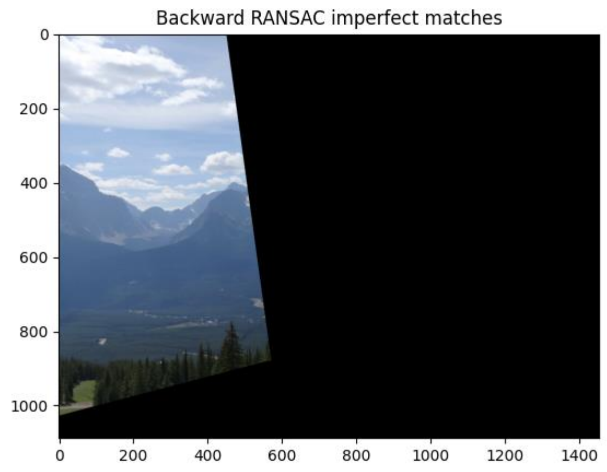

*Figure 6 - backward mapping*

It can be seen that in this implementation the black gaps is not exist due to handling with the problems of the forward mapping by the backward mapping.

### 14. Adding translation to the backward homography

**Written in the attached code.**

### 15. Panorama function

**Written in the attached code.**

## 16. Panorama results

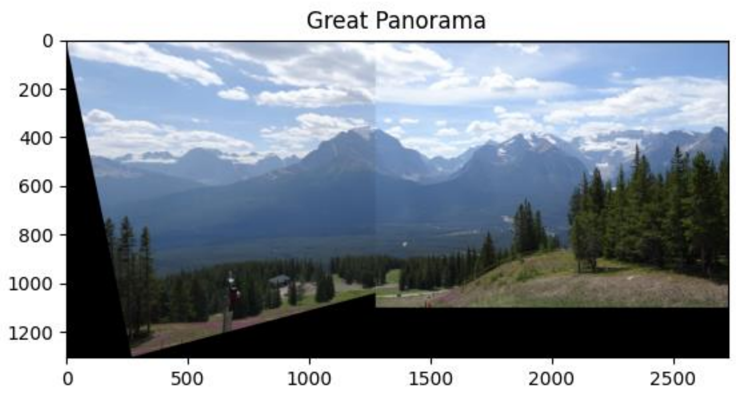

*Figure 7 - panorama*

The above figure is the panorama of the two images using the backward translated mapping and the RANSAC outlier handling.

## 17. Custom images panorama

Using 2 images taken from my balcony by a smartphone, I run the matching point script to match 25 corresponding points and then the panorama creation script with max_err = 2 and inliers = 0.5 , got the results below:

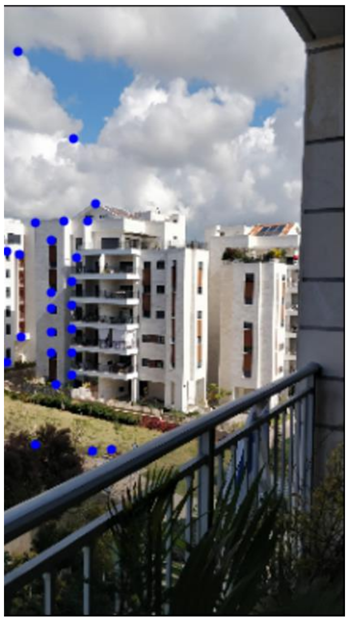

*Figure 8 - source custom image with matching points*

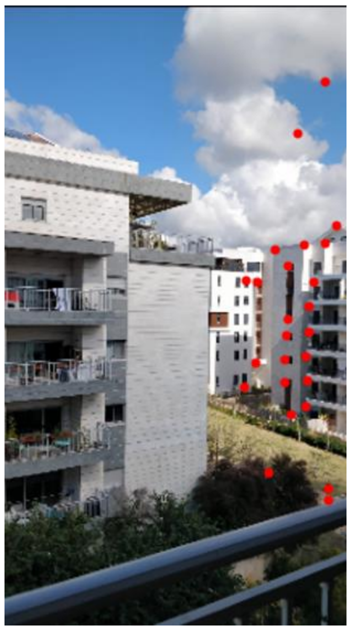

*Figure 9 - destination custom image*

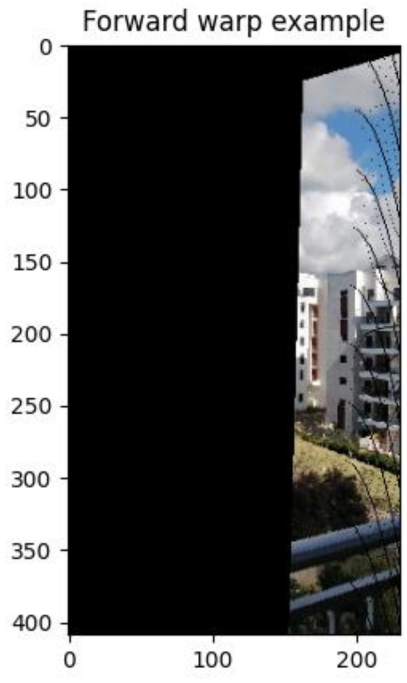

*Figure 10 - forward mapping of custom image*

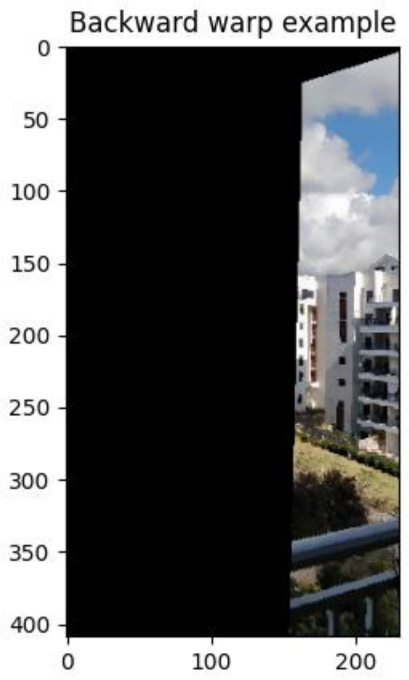

*Figure 11 - backward mapping of custom image*

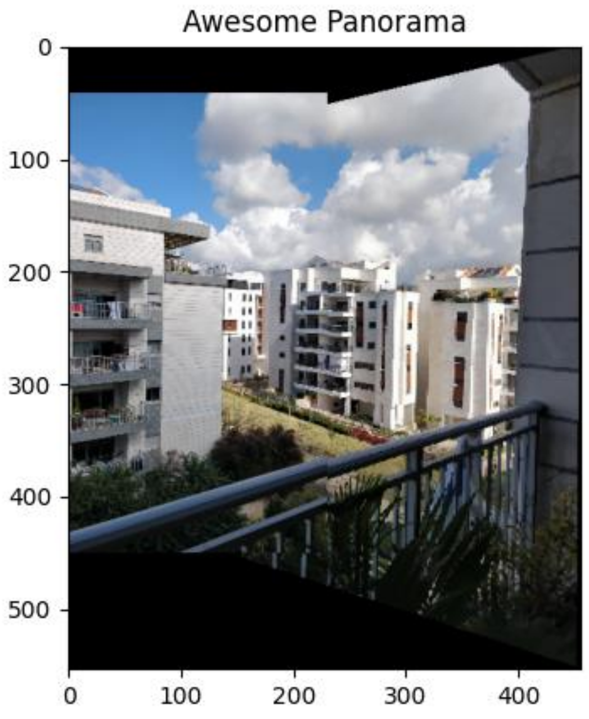

*Figure 12 - panorma of custom image*

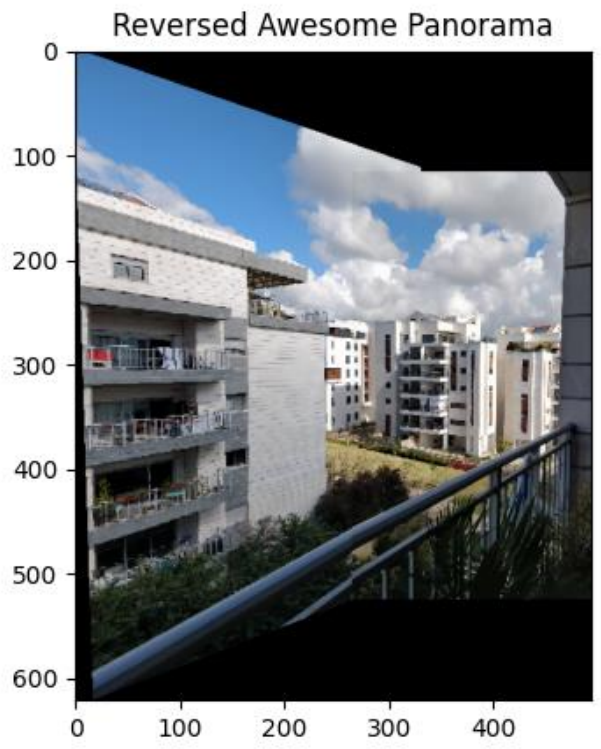

*Figure 13 - reversed panorma of custom image*