ENPM673 Project 2

Vineet Kumar Singh (ID: vsingh03)

March 2023

Contents

1 Problem 1

1.1 Image Processing pipeline

In this section, the paper in the video is extracted and the corners of the paper is detected using Hough transform. Mathematical explanation of Hough transform is given in next subsection.

The pipeline for corner detection is :

1. The video is read and each frame is extracted.
2. The frame is first converted to HSV colour space from BGR and blurred to remove unwanted noise for better processing.
3. Appropriate mask is applied on the frame to extract the white colour paper.
4. The image is then converted to binary image using a threshold.
5. The resulting image is passed through a Canny edge detector to extract the edges in the frame.
6. Hough transform is applied on above image to extract the detected lines.
7. The detected lines with highest intensity is extracted (4 lines are extracted) and solved pairwise to give the corner points.

The above pipeline is implemented for every frame in the video.

1.2 Hough Transform

The Hough Transform is used to detect lines in an image. Given an image, we can represent a line using its polar coordinates, where d is the distance of the line from the origin and θ is the angle between the line and the x-axis. The Hough Transform works by constructing an accumulator array of size (N_d, N_θ), where N_d and N_θ are the number of discrete values of d and θ we want to consider.

For each edge pixel in the image, the set of (d, θ) pairs is computed that correspond to lines passing through that pixel. This is done by evaluating the equation:

$$d = x \cos\theta + y \sin\theta$$

for all possible values of θ and incrementing the accumulator array at the corresponding (d, θ) position.

The lines in the image is found by searching for local maxima in the accumulator array. A threshold is applied to filter out weaker lines. The threshold is iteratively selected which gave best result.

The implementation of Hough Transform in code is done in the sequence as given below:

1. A set of polar coordinates (d, θ) is defined that represent lines in the image.
2. Created an accumulator array of size (N_d, N_θ).
3. For each edge pixel in the image, compute the set of (ρ, θ) pairs that correspond to lines passing through that pixel and increment the accumulator array at the corresponding (ρ, θ) position.

4. Find the local maxima in the accumulator array.

5. Convert the (d, θ) coordinates of the local maxima back to Cartesian coordinates to obtain the lines in the image.

The extracted lines (extracted coefficients of equation of lines) are the sides of the page in the frame. These are arranged as adjacent sides and solved using the **sympy library** *sympy.solve()* function. The solution gives the four corners of the page from the frame. sympy library

The final trajectory plot is as shown in Fig 1.

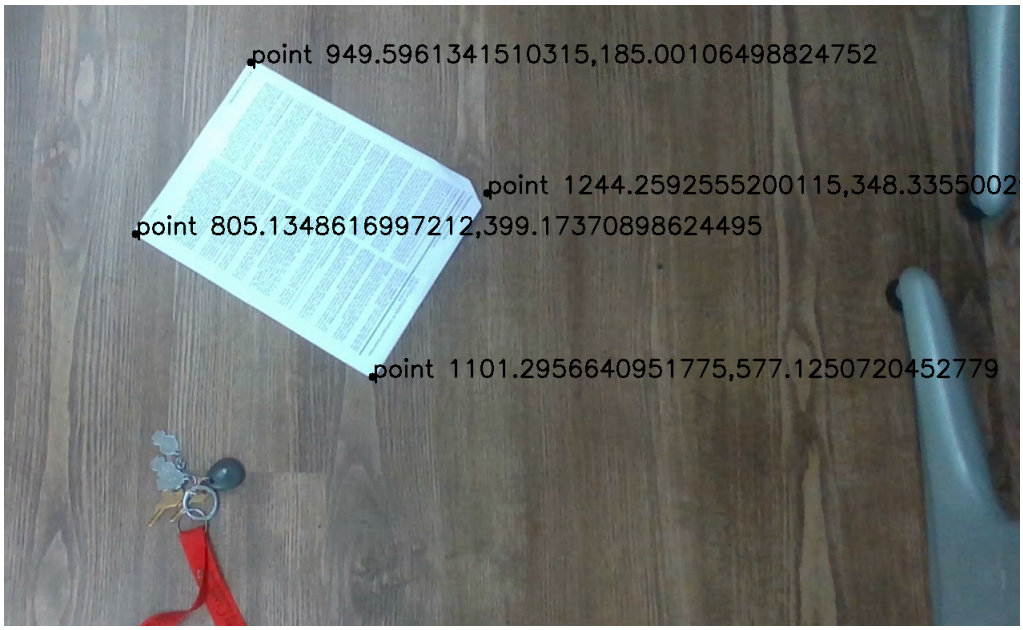

Figure 1: Detected corners from the Hough transform

1.3 Homography matrix

The Homography matrix is used to transform points from one plane to another. Given a set of corresponding points between two planes, these points cna be used to calculate the Homography matrix using a least-squares approach.

The equations to describe the homography matrix is:

The Homography matrix is computed using the detected corners and the original coordinates of the paper considering one corner as origin (as given in problem statement).

1.4 Decomposition of homography matrix

The computed homography matrix is decomposed using SVD to get the Rotation and translation for the frame.

To decompose the Homography matrix, Singular Value Decomposition (SVD) is used. The Homography matrix H is factorized into:

$$H = K[R|t]$$

where K is the intrinsic matrix of the camera, R is the rotation matrix that represents the relative orientation of the two cameras, and t is the translation vector that represents the position of the second camera relative to the first camera.

The K matrix is used as given in the problem statement. The inverse of K is computed and used to normalize the Homography matrix:

$$H' = K^{-1}H$$

Next, H' is decomposed into $H' = U\lambda V^T$, where U, λ, and V are the left singular vectors, singular values, and right singular vectors, respectively. Then:

$$R = UV^T$$

and

$$t = \frac{H'[:, 2]}{|\lambda|}$$

where $H'[:, 2]$ is the third column of H' and $|\lambda|$ is the Frobenius norm of λ. Note that t is normalized by the scale factor $|\lambda|$ to make it a unit vector.

The resulting R and t represent the relative orientation and position of the second camera with respect to the first camera.

The equations used is as given in Figure 2

$$\boldsymbol{X} = (X, Y, 0, 1)$$

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{PX} \\ &= \boldsymbol{K}[\boldsymbol{r}_1\boldsymbol{r}_2\boldsymbol{r}_3\boldsymbol{t}] \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} \\ &= \boldsymbol{K}[\boldsymbol{r}_1\boldsymbol{r}_2\boldsymbol{t}] \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \\ &= \boldsymbol{H} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \end{aligned}$$

$$\boldsymbol{H} = \lambda \boldsymbol{K}[\boldsymbol{r}_1\boldsymbol{r}_2\boldsymbol{t}]$$

$$\boldsymbol{K}^{-1}\boldsymbol{H} = \lambda[\boldsymbol{r}_1\boldsymbol{r}_2\boldsymbol{t}]$$

–r_1 and r_2 are unit vectors ⇨ find lambda

–Use this to compute t

–Rotation matrices are orthogonal ⇨ find r3

$$\boldsymbol{P} = \boldsymbol{K}\begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & (\boldsymbol{r}_1 \times \boldsymbol{r}_2) & \boldsymbol{t} \end{bmatrix}$$

Figure 2: Equations used for Homography decomposition

1.5 Results

The plot of Euler angles (from rotation matrix) vs frames is shown below in Figure 3.

The plotting of Translation matrix (x, y, z) vs frames is shown below in Figure 4.

1.6 Problems Encountered

Few challenges arised while solving the above problem. The significant problems were:

1. Filtering the image to filter out the paper from the video frames was a challenge as there is a lot of noise in the white channel from the surrounding objects. To address this, HSV colour slider is used to filter the image. The images are also eroded for 5 iterations and then dilated to give better result.

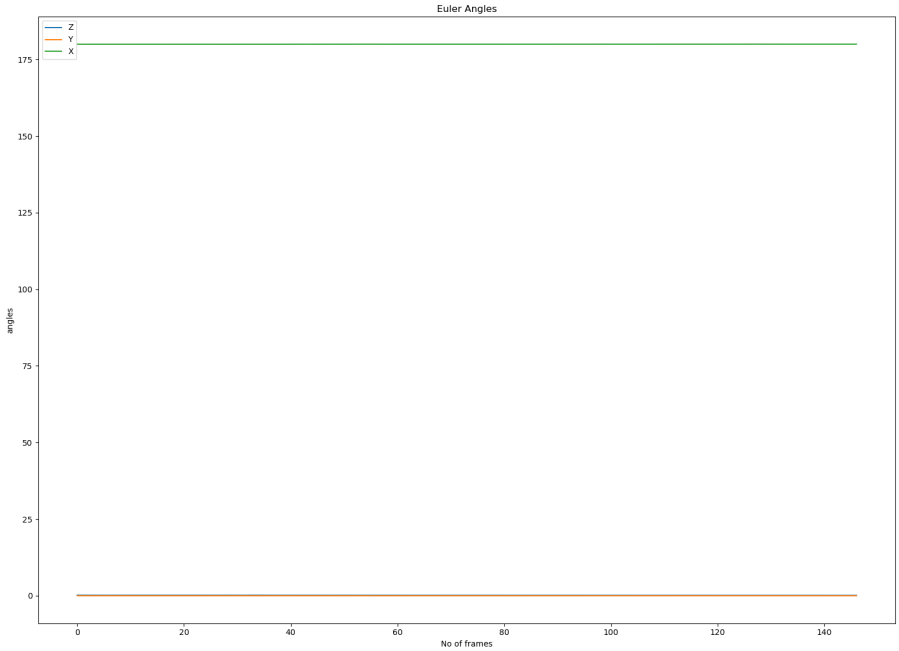

Figure 3: Plot of rotation Euler angles for every frame

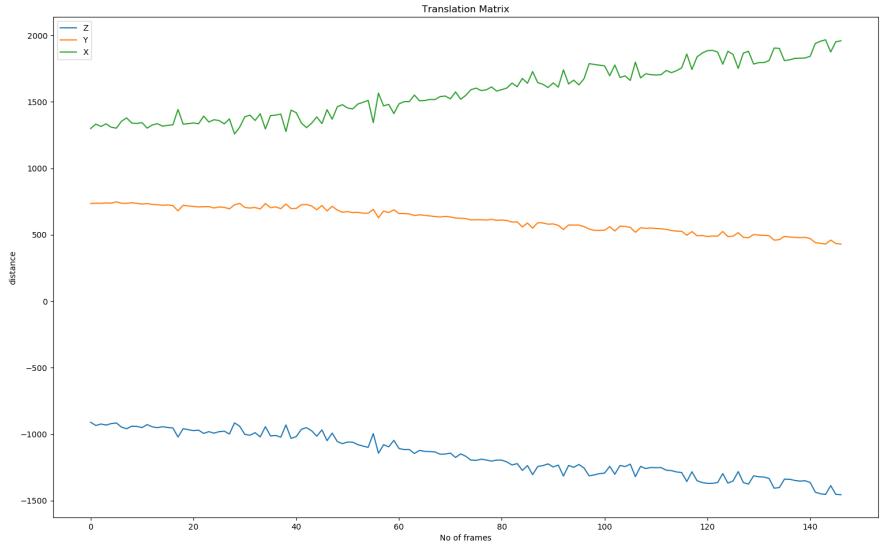

Figure 4: Plot of translation matrix for every frame

2. Filtering the lines from the accumulator in Hough transform as the points detected with highest intensity were usually the neighbours. To address this, the points were grouped together based on their neighbourhood. This reduced the accuracy of the corners on image but gave the four points without multiple detections for the same points.

3. The calculation of scaling value $/lambda$ from the H matrix decomposition was not the same when calculated for r1 and r2 columns. To address this, the values were

averaged to give an estimate, and this estimate is used to calculate the translation matrix.

4. The matching between the corners and world coordinates was tricky as the detected points were not always in the same order for every frame. To address this the detected points are first sorted using x coordinate and then matched with the world coordinate for every frame.

2 Problem 2

2.1 Pipeline

1. All the images are read and converted to Grayscale.

2. SIFT feature detector is used to extract detectable features from all the images.

3. The keypoints and descriptors from SIFT is then used to match the features between 2 images (sequence selected for images from left to right). This is done using OpenCV *BFMatcher* function.

4. All matches are then passed to calculate Homography matrix. To randomly select the matches RANSAC is used and the 4 points with highest inliers is used to calculate the Homography matrix.

5. The second image is then wrapped on the first image using *wrapPerspective* function using the selected Homography matrix.

6. The above step is repeated for the third and fourth image. Both the images are wrapped on the perspective of 1st image to give the final panorama image.

2.2 Results

The output from feature matching is shown in Figure 5
The output from the Feature Matcher is as shown in Figure 6
The result of stitching between first and second images is shown in Figure 7
The final result after stitching all the images is shown in Figure 8

2.3 Problems Encountered

1. The implementation of RANSAC for matched features was a challenge especially for selecting the threshold. To address this the threshold value was selected based on multiple iterations to see the best result.

2. The perspective of the resulting image after stitching was also a major challenge as the last image was getting trimmed on the edges. For solving this, the overall size of the image was increased to have the complete view of the image. The image stitcher function implements this approach.

Figure 5: Features detected in the images 3 and 4

Figure 6: Features matched in the images 1 and 2

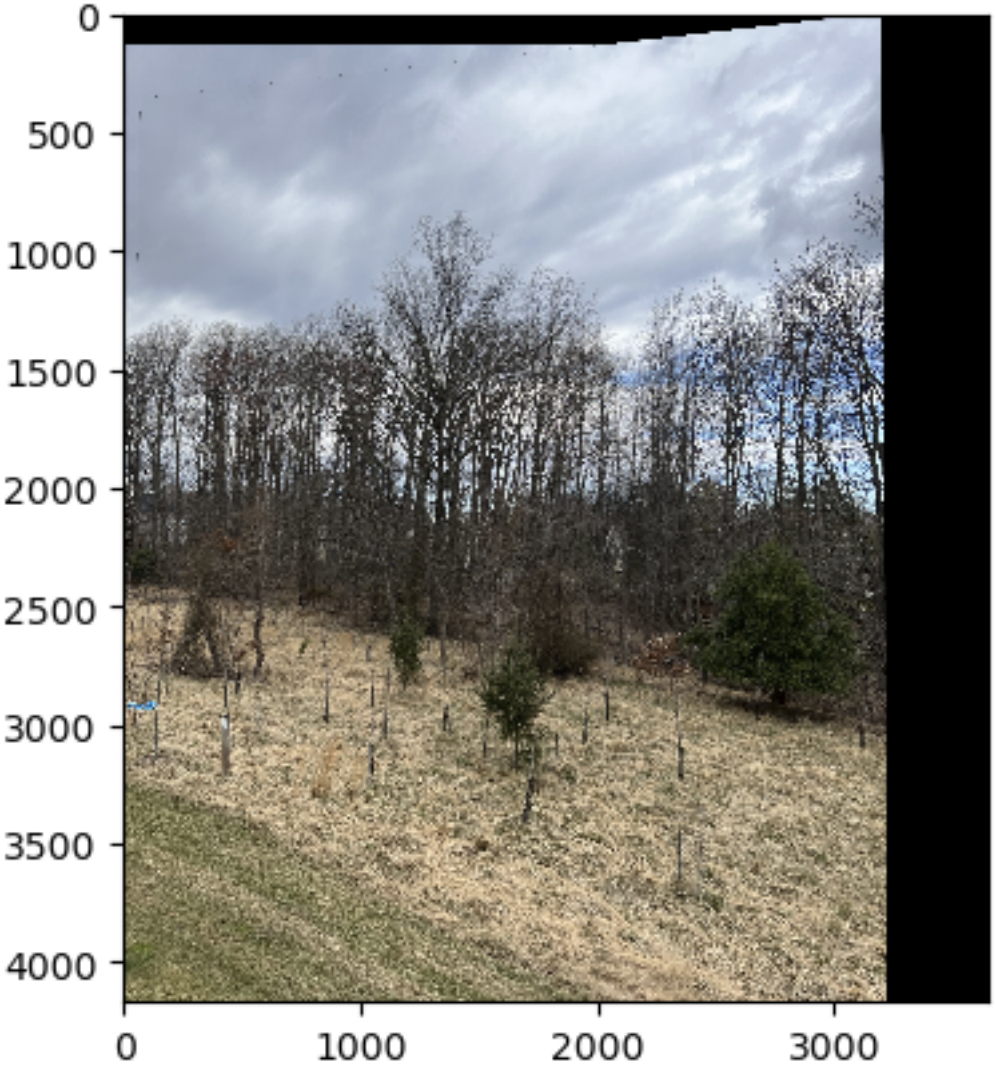

Figure 7: Stitching between images 1 and 2