ENPM673 Project 2

Ankur Chavan (achavan1@umd.edu)

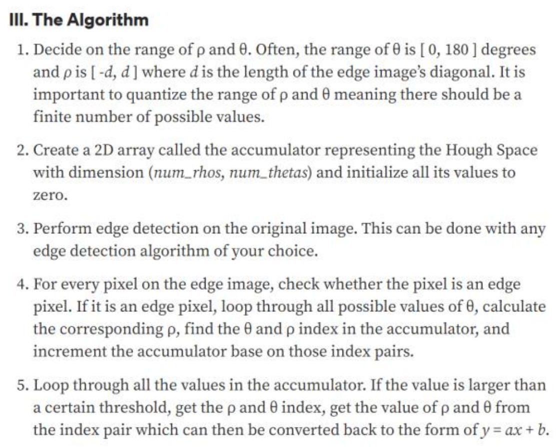
UID: 119284513

Problem1:

Approach or Pipeline:

1. Import necessary libraries.
2. Read the video file frame by frame.
3. Create a while loop that runs as long as the video is opened.
4. Resize the frame using cv2.resize().
5. Apply a Repeated Closing operation on the frame to remove any text present in the document using cv2.morphologyEx().
6. Convert the frame to grayscale using cv2.cvtColor().
7. Apply a threshold to the image to segment the foreground from the background using cv2.threshold().
8. Apply a Repeated Closing operation on the thresholded image to remove any noise present in the document using cv2.morphologyEx().
9. Perform Canny edge detection on the thresholded image using cv2.Canny().

   Hough Transform:



III. The Algorithm

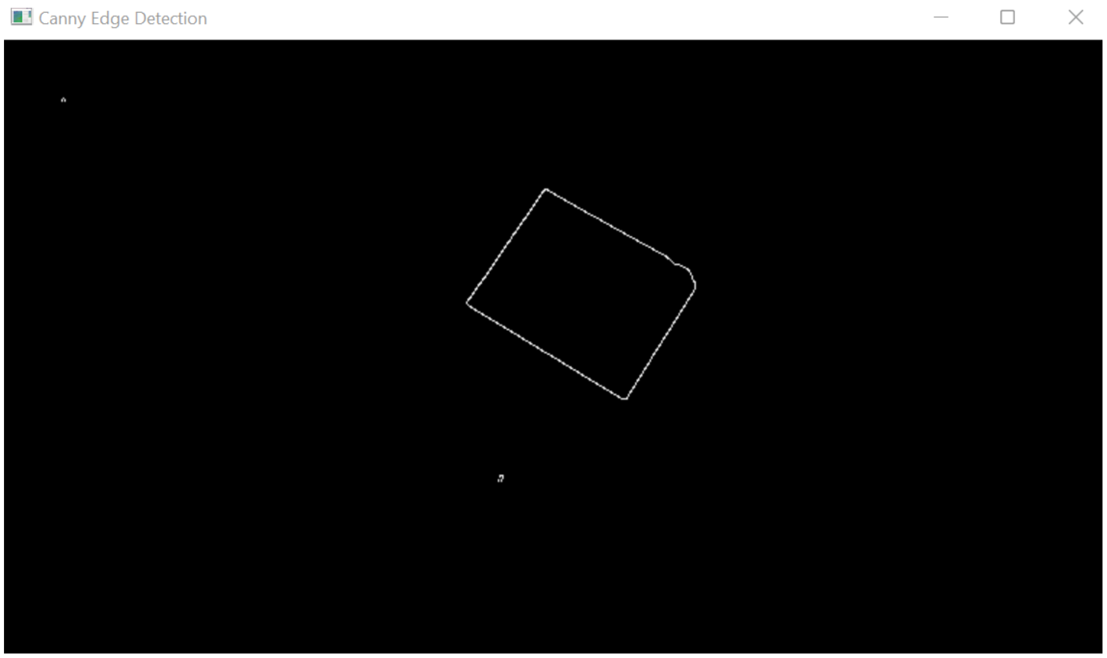
1. Decide on the range of ρ and θ. Often, the range of θ is [0, 180] degrees and *ρ* is [-*d*, *d*] where *d* is the length of the edge image's diagonal. It is important to quantize the range of ρ and θ meaning there should be a finite number of possible values.
2. Create a 2D array called the accumulator representing the Hough Space with dimension (*num_rhos, num_thetas*) and initialize all its values to zero.
3. Perform edge detection on the original image. This can be done with any edge detection algorithm of your choice.
4. For every pixel on the edge image, check whether the pixel is an edge pixel. If it is an edge pixel, loop through all possible values of θ, calculate the corresponding ρ, find the θ and ρ index in the accumulator, and increment the accumulator base on those index pairs.
5. Loop through all the values in the accumulator. If the value is larger than a certain threshold, get the ρ and θ index, get the value of ρ and θ from the index pair which can then be converted back to the form of *y = ax + b*.

10. Define the parameter space grid for the Hough transform.
11. Perform the Hough transform on the edge-detected image and accumulate the Hough space values in a numpy array.
12. Find the peaks in the Hough space accumulator.
13. Find the corresponding line equations for each peak.
14. Draw the lines corresponding to each peak on the original image.
15. Show the lines on morphed frame.
16. Find the intersections of the lines to detect the corner of the paper.

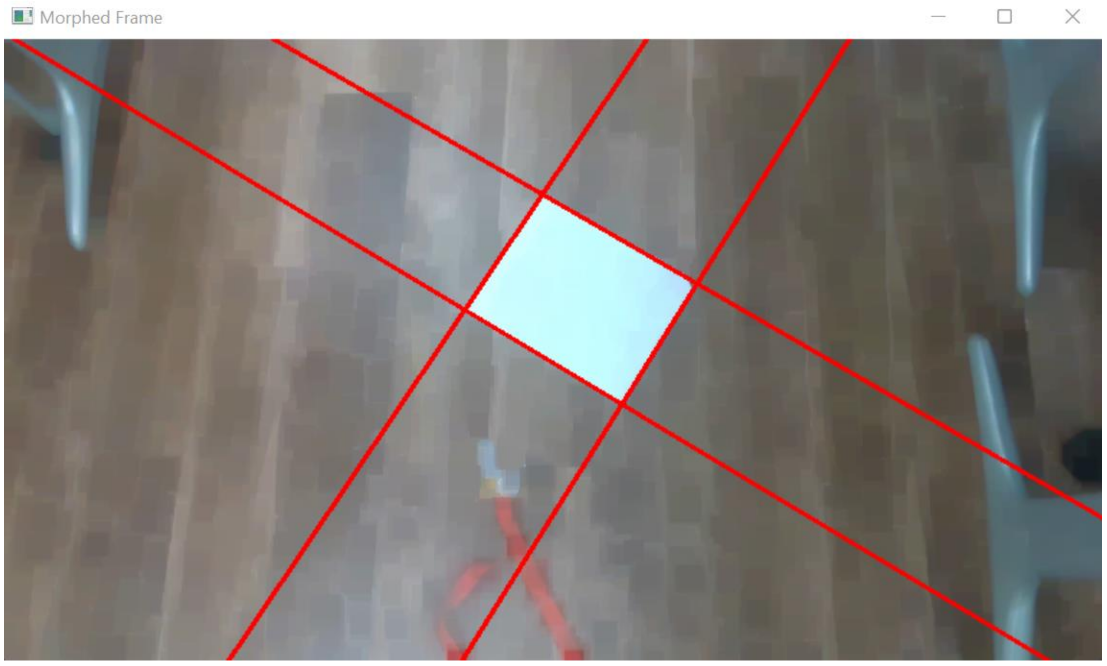
17. Find the homography between the corners of the paper in the video frame and the real corners of the paper.
18. Extracting and plotting the translation and rotation.
19. Wait for keyboard input and check if it is 'q'. If it is, break out of the loop.
20. Release the video and destroy all windows.

Results:

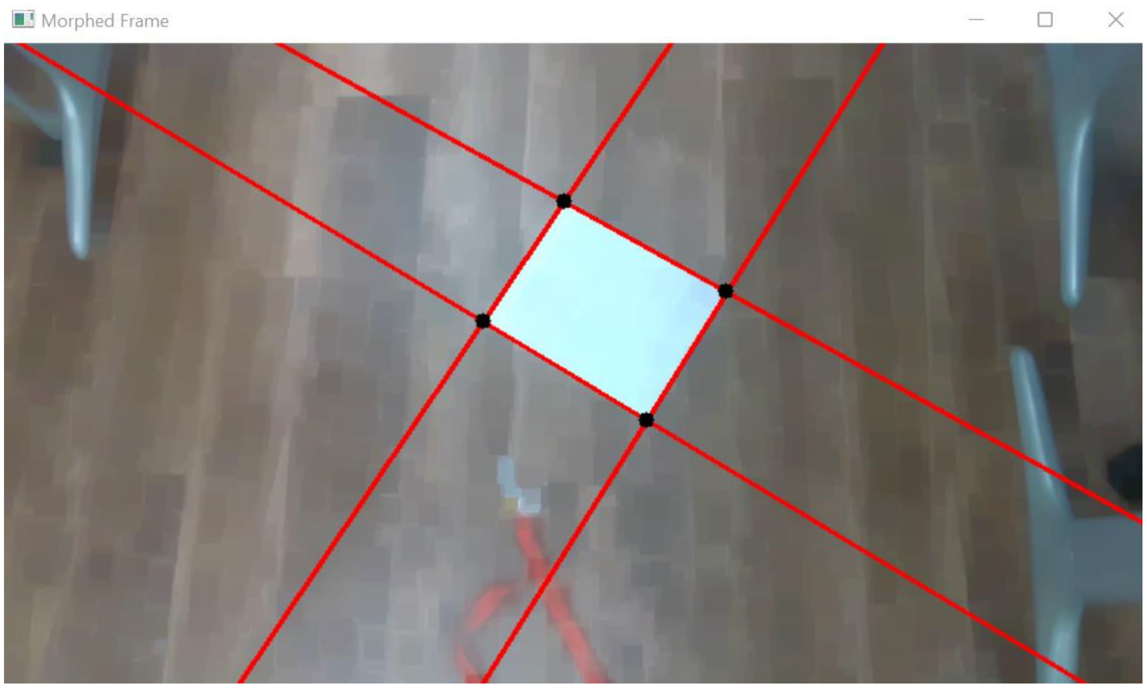
Edge Detection using Canny:



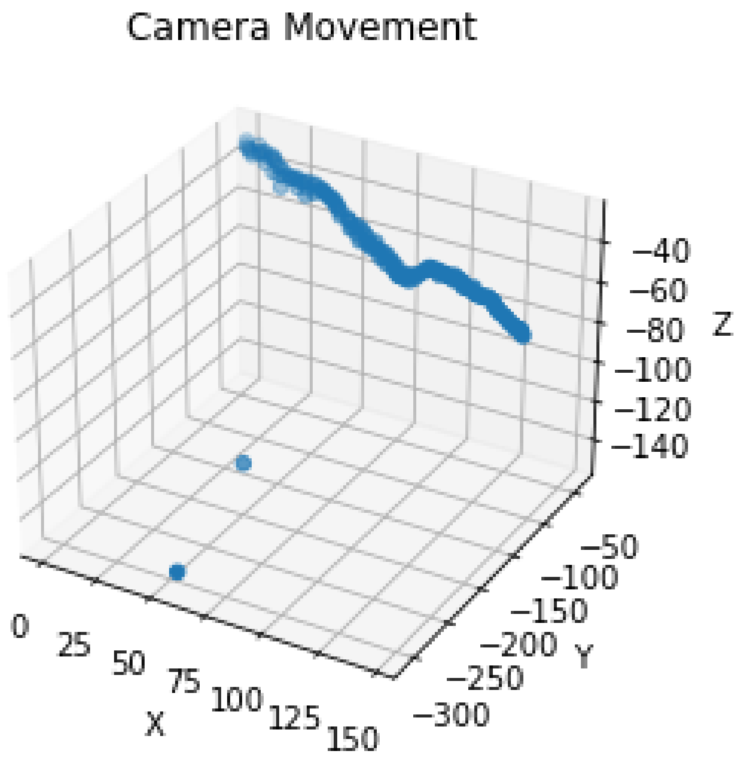
Drawing line at the edges of paper using Hough Transform:



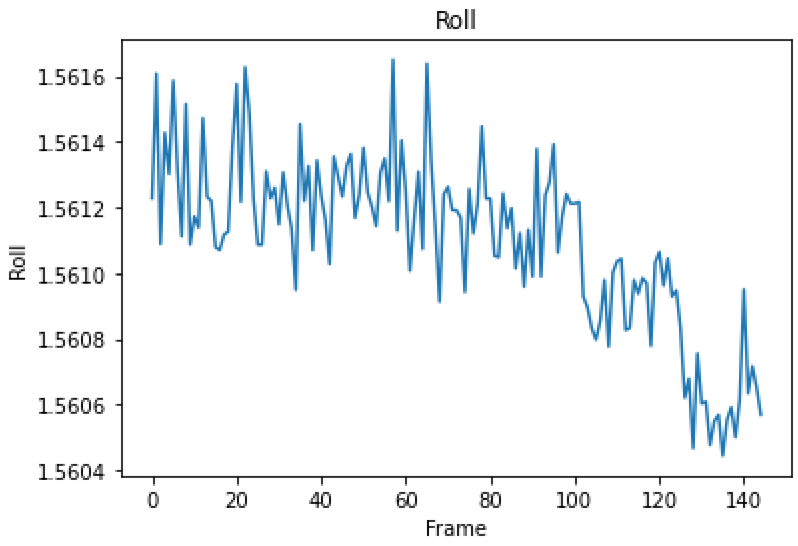
Finding the intersection of the lines to get the corners of the paper:

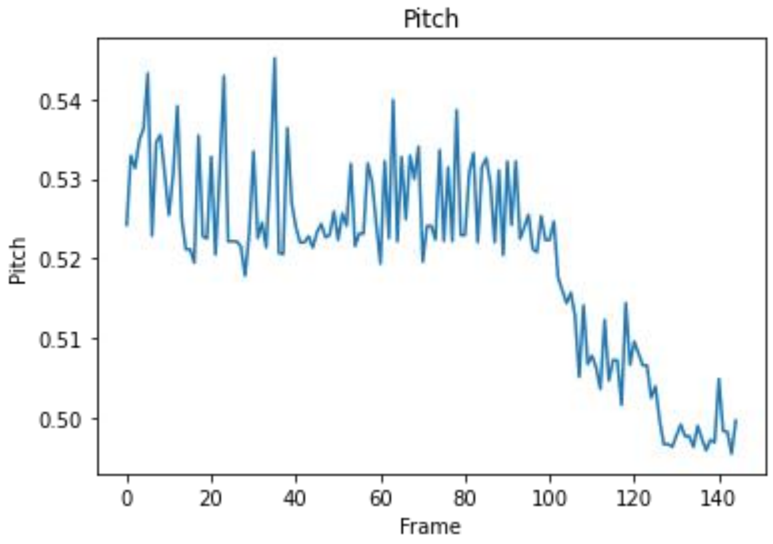


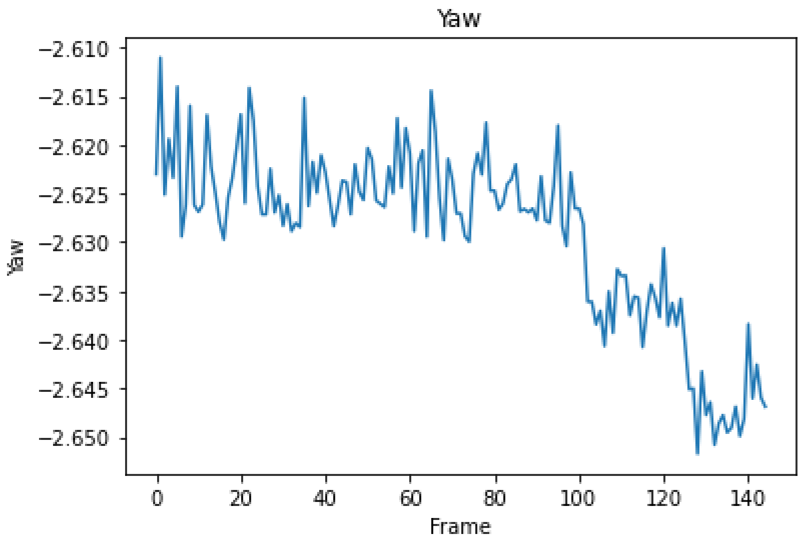
Translation:



Rotation:


Roll
1.5616
1.5614
1.5612
1.5610
1.5608
1.5606
1.5604
Roll
0
20
40
60
80
100
120
140
Frame


Pitch
0.54
0.53
0.52
0.51
0.50
Pitch
0
20
40
60
80
100
120
140
Frame


Yaw
−2.610
−2.615
−2.620
−2.625
−2.630
−2.635
−2.640
−2.645
−2.650
Yaw
0
20
40
60
80
100
120
140
Frame

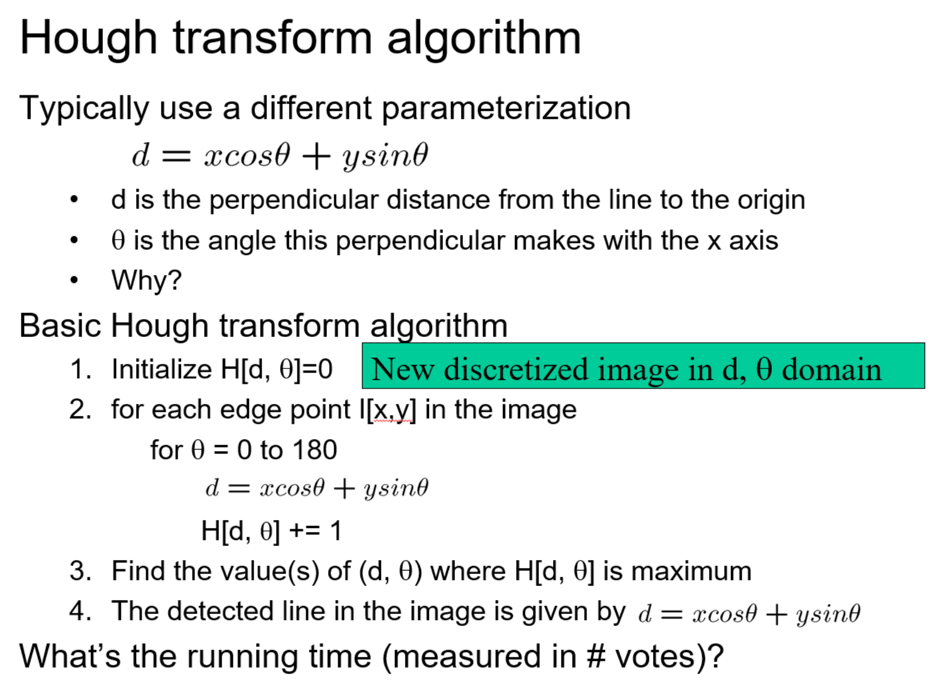
Code:

Problems Encountered:

The first problem that I encountered was in edge detection, I was not getting clear edges of the paper. So, to solve this, I used morphology to get rid of the text on the paper, this helped me to get rid of the false edge that was being detected due to the letters present on the paper.

The second problem I encountered was in the implementation of the Hough transform, which I was able to resolve after referring to various resources.

Class Notes Referred for the problem:



Hough transform algorithm

Typically use a different parameterization

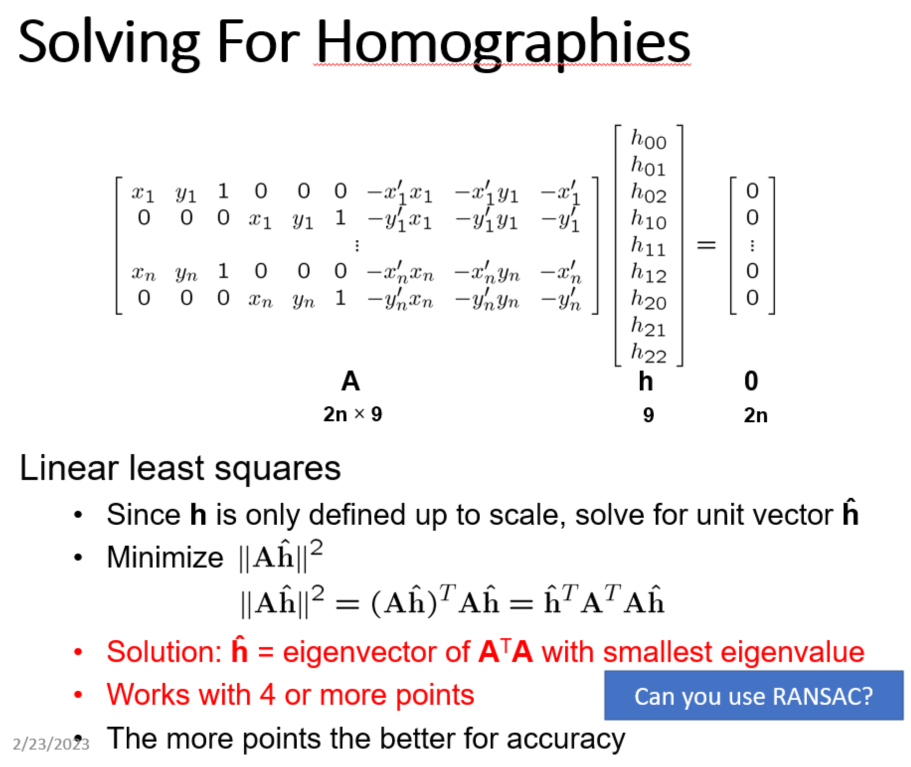
$$d = x cos\theta + y sin\theta$$

- d is the perpendicular distance from the line to the origin
- θ is the angle this perpendicular makes with the x axis
- Why?

Basic Hough transform algorithm

```
1. Initialize H[d, θ]=0     New discretized image in d, θ domain
2. for each edge point I[x,y] in the image
       for θ = 0 to 180
           $d = x cos\theta + y sin\theta$
           H[d, θ] += 1
3. Find the value(s) of (d, θ) where H[d, θ] is maximum
4. The detected line in the image is given by $d = x cos\theta + y sin\theta$
```

What's the running time (measured in # votes)?



Solving For Homographies

$$\begin{array}{ccc}
\begin{bmatrix}
x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\
0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\
 & & & & \vdots & & & & \\
x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\
0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n
\end{bmatrix}
&
\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}
= &
\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \\
\mathbf{A} & \mathbf{h} & \mathbf{0} \\
2n \times 9 & 9 & 2n
\end{array}$$

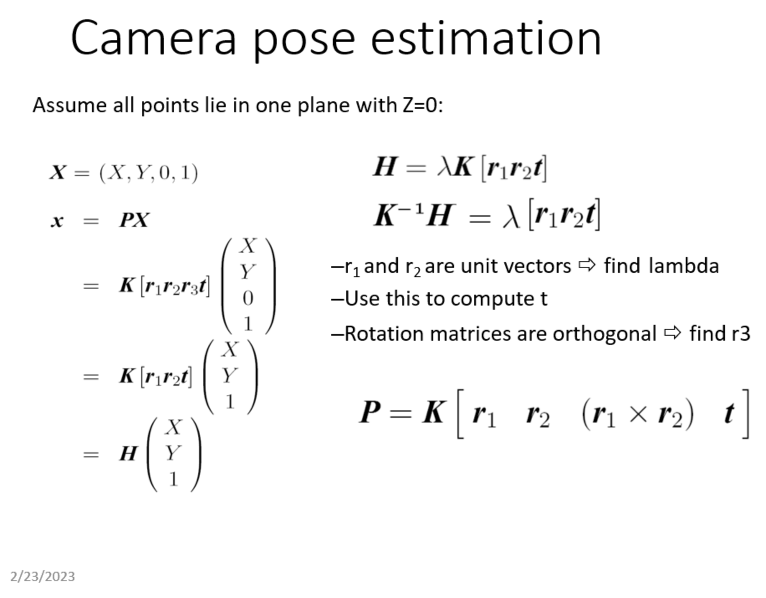
Linear least squares

- Since **h** is only defined up to scale, solve for unit vector $\hat{\mathbf{h}}$
- Minimize $||\mathbf{A}\hat{\mathbf{h}}||^2$

$$||\mathbf{A}\hat{\mathbf{h}}||^2 = (\mathbf{A}\hat{\mathbf{h}})^T\mathbf{A}\hat{\mathbf{h}} = \hat{\mathbf{h}}^T\mathbf{A}^T\mathbf{A}\hat{\mathbf{h}}$$

- Solution: $\hat{\mathbf{h}}$ = eigenvector of $\mathbf{A}^T\mathbf{A}$ with smallest eigenvalue
- Works with 4 or more points   Can you use RANSAC?
- The more points the better for accuracy

2/23/2023



Camera pose estimation

Assume all points lie in one plane with Z=0:

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

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

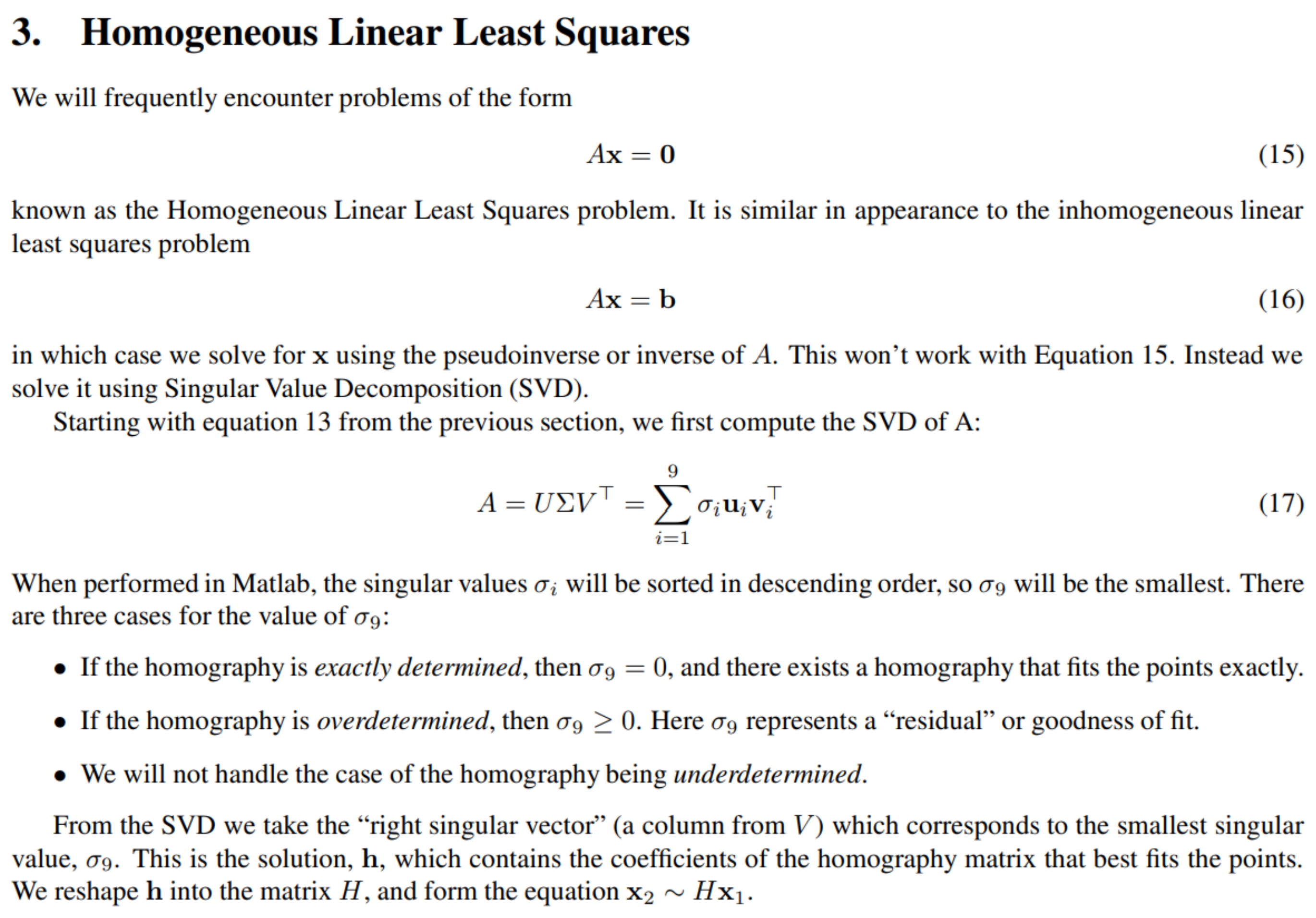
$$\boldsymbol{K}^{-1}\boldsymbol{H} = \lambda[\boldsymbol{r}_1\boldsymbol{r}_2\boldsymbol{t}]$$

–r_1 and r_2 are unit vectors ⇨ find lambda
–Use this to compute t
–Rotation matrices are orthogonal ⇨ find r3

$$\boldsymbol{P} = \boldsymbol{K}\left[\boldsymbol{r}_1 \quad \boldsymbol{r}_2 \quad (\boldsymbol{r}_1 \times \boldsymbol{r}_2) \quad \boldsymbol{t}\right]$$

2/23/2023

Homography using SVD:



3. Homogeneous Linear Least Squares

We will frequently encounter problems of the form

$$A\mathbf{x} = \mathbf{0} \tag{15}$$

known as the Homogeneous Linear Least Squares problem. It is similar in appearance to the inhomogeneous linear least squares problem

$$A\mathbf{x} = \mathbf{b} \tag{16}$$

in which case we solve for $\mathbf{x}$ using the pseudoinverse or inverse of A. This won't work with Equation 15. Instead we solve it using Singular Value Decomposition (SVD).

Starting with equation 13 from the previous section, we first compute the SVD of A:

$$A = U\Sigma V^\top = \sum_{i=1}^{9}\sigma_i\mathbf{u}_i\mathbf{v}_i^\top \tag{17}$$

When performed in Matlab, the singular values σ_i will be sorted in descending order, so σ_9 will be the smallest. There are three cases for the value of σ_9:

- If the homography is *exactly determined*, then $\sigma_9 = 0$, and there exists a homography that fits the points exactly.
- If the homography is *overdetermined*, then $\sigma_9 \geq 0$. Here σ_9 represents a "residual" or goodness of fit.
- We will not handle the case of the homography being *underdetermined*.

From the SVD we take the "right singular vector" (a column from V) which corresponds to the smallest singular value, σ_9. This is the solution, $\mathbf{h}$, which contains the coefficients of the homography matrix that best fits the points. We reshape $\mathbf{h}$ into the matrix H, and form the equation $\mathbf{x}_2 \sim H\mathbf{x}_1$.

source: David Kriegman

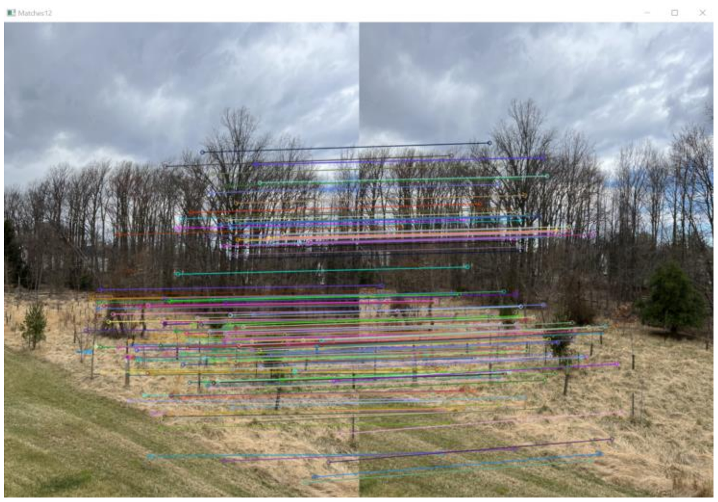
Problem 2 approach:

Step-by-step explanation of the code:

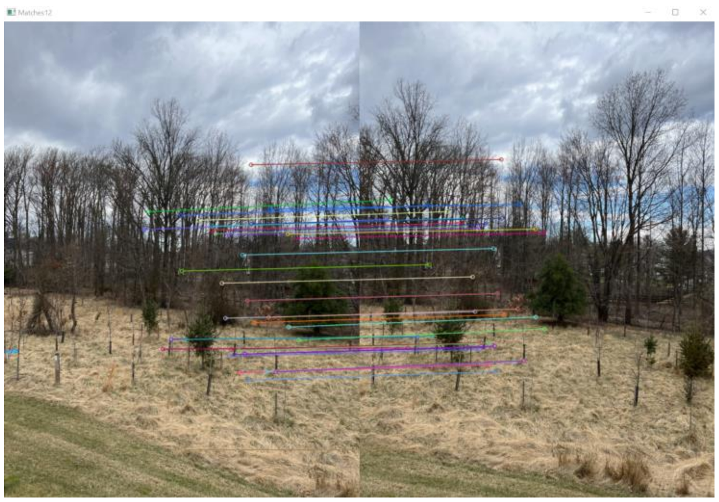
1. First, the necessary libraries are imported.
2. Images are read and resized.
3. A **SIFT** (Scale-Invariant Feature Transform) object is created using **cv2.SIFT_create()**. **SIFT** is a feature detection algorithm that detects keypoints and computes their descriptors.
4. Defined a function named sticher() to create a panorama of two images. The function does the following operations.
   a) Takes two images and the sift object as its arguments
   b) Converts both images to grayscale.
   c) Finds the key points and descriptor pairs for each image.
   d) A **FlannBasedMatcher** object is created using **cv2.FlannBasedMatcher()**. This object is used to match the descriptors of the key points in **img1** with the descriptors of the key points in **img2**.
   e) Key point matches are found using **matcher.knnMatch()** with **k=2**, which specifies that the two best matches for each key point should be found. The matches are stored in the variable **matches**.
   f) A loop is used to iterate through each match in **matches** to find a good match.
   g) Implements RANSAC to find the best homography by iterating randomly and choosing the one with most inliners.
   h) It then stitches the images together.
5. Defined another function named good _matches_ij(), that takes two images and sift object as argument and visualizes the matches between the two images.
6. Called the good _matches_ij() function to visualize the matches between images 1 & 2, images 2 & 3 and images 3 & 4.
7. Then I stitched images 3 & 4 first together and saved it as stich34, then I stitched image 2 and stich34 and saved it as stich234 and finally I stitched image 1 and stitch234 together to form a final panorama image of all four images combined.

Results:

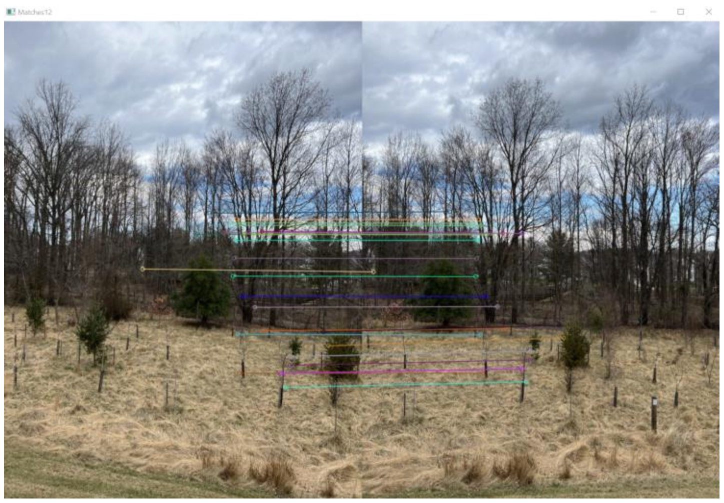
Matches between image 1 and image 2:



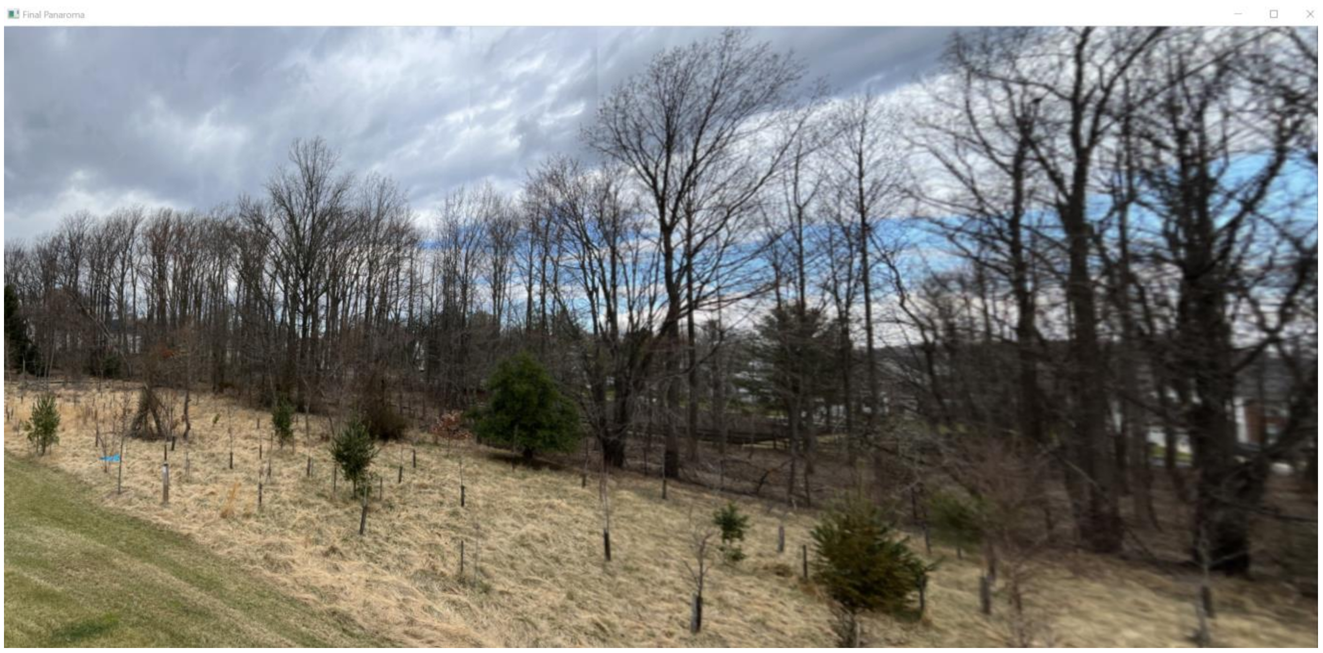
Matches between image 2 and image 3:



Matches between image 3 and image 4:



Final Panoramic Image:



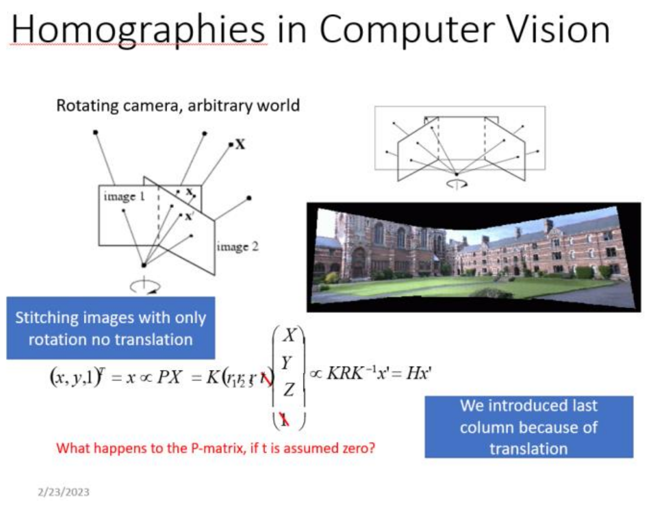
Code: submitted in a separate file.

Problems Encountered:

The first problem I faced was n finding the homography. Initially I computed it using SVD, but I was not getting good results, so I used RANSAC to get the best homography by iterating randomly for a threshold of inliners.

Later, I faced error in stitching both the images together based on the homography. I resolved it after understanding the wrap perspective.

Class Notes Referred:



Homographies in Computer Vision

Rotating camera, arbitrary world

X
image 1
x
x'
image 2

Stitching images with only rotation no translation

$$(x, y, 1)^T = x \propto PX = K(r_1 r_2 r_3 \cancel{t})\begin{pmatrix} X \\ Y \\ Z \\ \cancel{1} \end{pmatrix} \propto KRK^{-1}x' = Hx'$$

We introduced last column because of translation

What happens to the P-matrix, if t is assumed zero?

2/23/2023

RANSAC Pseudocode:

```
Given:
    data – A set of observations.
    model – A model to explain the observed data points.
    n – The minimum number of data points required to estimate the model parameters.
    k – The maximum number of iterations allowed in the algorithm.
    t – A threshold value to determine data points that are fit well by the model (inlier).
    d – The number of close data points (inliers) required to assert that the model fits well to the data.

Return:
    bestFit – The model parameters which may best fit the data (or null if no good model is found).


iterations = 0
bestFit = null
bestErr = something really large // This parameter is used to sharpen the model parameters to the best data fitting as
iterations goes on.

while iterations < k do
    maybeInliers := n randomly selected values from data
    maybeModel := model parameters fitted to maybeInliers
    confirmedInliers := empty set
    for every point in data do
        if point fits maybeModel with an error smaller than t then
             add point to confirmedInliers
        end if
    end for
    if the number of elements in confirmedInliers is > d then
        // This implies that we may have found a good model.
        // Now test how good it is.
        betterModel := model parameters fitted to all the points in confirmedInliers
        thisErr := a measure of how well betterModel fits these points
        if thisErr < bestErr then
            bestFit := betterModel
            bestErr := thisErr
        end if
    end if
    increment iterations
end while

return bestFit
```

Source: Wikipedia