Computer Vision

Assignment II

Abu Shahid

B20CS003

- Part A
  - Question 1

    For this question, 3 different heuristics and 3 different approaches was implemented

    - Heuristic 1

      Idea: higher number of matches means logo is present

      File name— `heuristic1.py`

      Execution: `python3 heuristic2.py --scene <path/to/scene/image> --folder <path/tp/logo/folder>`

      The program also takes technique as an input:

      - After is prompted with the technique, it finds the number of matches between the two images using the given feature detector and BFMatcher and returns the number of matches.
      - The logo that gives the maximum number of matches is the desired logo in the scene.
      - This approach does not work as: just because two images have the same features does not mean they have the same objects.
      - Here, we are also not looking into the quality of matches but just there number.

    - Heuristic 2

      Idea: Higher number of **good** matches means logo is present.

      filename- `heuristic2.py`

execution- `python3 heuristic 2.py --scene <path to image> --folder <path to logo folder>`

In this approach we devise a method to calculate the number of good matches. The pair giving the higher number of good matches is the logo we are looking for.

In this approach, however the definition is ‘good’ matches is dependent on a threshold that was chosen carefully.

```
for m, n in matches:
    if m.distance < 0.5 * n.distance:
        good.append([m])
```

For all three SIFT, BRIEF and ORB

```
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 logoMatch.py --scene Problem-1/Ex2/starbucks.jpeg --folder Problem-1/Ex2/logos/
[ WARN:0@0.042] global shadow_sift.hpp:13 SIFT_create DEPRECATED: cv.xfeatures2d.SIFT_create() is deprecated due SIFT tranfer
o the main repository. https://github.com/opencv/opencv/issues/16736
Best match for sift is starbucks.jpg
Best match for orb is starbucks.jpg
Best match for brief is starbucks.jpg
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 logoMatch.py --scene Problem-1/Ex1/levis.jpg --folder Problem-1/Ex1/logos/
[ WARN:0@0.029] global shadow_sift.hpp:13 SIFT_create DEPRECATED: cv.xfeatures2d.SIFT_create() is deprecated due SIFT tranfer
o the main repository. https://github.com/opencv/opencv/issues/16736
Best match for sift is levis.jpg
Best match for orb is nescafe.jpg
Best match for brief is kfc.jpg
```

As we can see, all the three approaches gave the current result of Starbucks in second example, however; in the first example, only SIFT gave the correct output of levis while ORB and BRIEF failed.

There can be many reasons for this;

- First is the chosen value of multipliers: the affect of chosen multiplier changes the answers as it was observed during the implementation. Very low value gives very less number of matches while a high value gives a high number of matches.

```
Best match for sift is levis.jpg
[0, 0, 34, 0, 6, 0, 6, 21, 4, 3]
```

When multiplier = 0.5

```
Best match for sift is tacobell.jpg
[394, 596, 810, 311, 592, 299, 491, 824, 265, 496]
```

When multiplier = 0.9

```
Best match for sift is hp.jpg
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

When multiplier = 0.1

    - In our implementation, 0.5 was taken as a mere safe choice and gave accurate results in Example 2. At attempt just to get correct output in Example was was not made.
- Another reason can be the internal working of SIFT, ORB and BRIEF

▼ Heuristic 3

This is built on heuristic 2, where we not only take the count of good matches but also their absolute value. If the number of good_matches is less than say a number T, the number of matches are deemed insufficient to draw any conclusion.

The value of T is different for different feature detectors

- For SIFT it is 30
- For ORB it was chosen to be 30
- For BRIEF it was taken to be 250

```
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 logoMatch.py --scene Problem-1/Ex1/levis.jpg --folder Problem-1/Ex1/logos/
[ WARN:0@0.028] global shadow_sift.hpp:13 SIFT_create DEPRECATED: cv.xfeatures2d.SIFT_create() is deprecated due SIFT tranf
o the main repository. https://github.com/opencv/opencv/issues/16736
Best match for sift is levis.jpg
No match found
No match found
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 logoMatch.py --scene Problem-1/Ex2/starbucks.jpeg --folder Problem-1/Ex2/logo
[ WARN:0@0.029] global shadow_sift.hpp:13 SIFT_create DEPRECATED: cv.xfeatures2d.SIFT_create() is deprecated due SIFT tranf
o the main repository. https://github.com/opencv/opencv/issues/16736
Best match for sift is starbucks.jpg
Best match for orb is starbucks.jpg
Best match for brief is starbucks.jpg
```

These values of multiplier and threshold were chosen carefully. In the third heuristic we still see that SIFT seems to outperform ORB and BRIEF

- Computational: SIFT is computationally expensive due to its scale-space extrema detection and keypoint localization. BRIEF is a fast algorithm for feature extraction, but it does not provide scale and rotation invariance. ORB is a combination of SIFT and BRIEF, and it is computationally faster than SIFT and provides rotation and scale invariance.
- Time: SIFT is slow in feature extraction and matching due to its complexity. BRIEF is faster than SIFT, but it does not provide scale and rotation invariance. ORB is faster than SIFT and provides rotation and scale invariance.

```
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 heuristic2.py --scene Problem-1/Ex1/levis.jpg --folder Problem-1/Ex1/logos/
[ WARN:0@0.045] global shadow_sift.hpp:13 SIFT_create DEPRECATED: cv.xfeatures2d.SIFT_create() is deprecated due SIFT tranf
o the main repository. https://github.com/opencv/opencv/issues/16736
Best match for sift is levis.jpg
[0, 0, 34, 0, 6, 0, 6, 21, 4, 3]
Best match for orb is nescafe.jpg
[0, 8, 1, 0, 29, 1, 0, 0, 0, 1]
Best match for brief is kfc.jpg
[39, 222, 87, 31, 137, 101, 59, 60, 12, 218]


Time taken for SIFT: 2.517118148 seconds
Time taken for ORB: 0.344281568 seconds
Time taken for BRIEF: 0.631620437 seconds
```

Results from `heuristic2.py`

- In the above picture, also look at the difference in the number of good match points for each logo for different techniques and the erroneous results produced if they are not properly dealt with.

- Results: SIFT provides high-quality feature extraction and matching, and it is widely used in applications that require high accuracies, such as object recognition and image stitching. BRIEF is good for fast feature extraction, but it does not provide scale and rotation invariance. ORB provides high-quality feature extraction and matching, and it is widely used in applications that require fast processing, such as real-time object recognition and tracking.

Feature matching after ORB. Clearly, there are a lot of mismatches and therefore the threshold of good points needs to be chosen carefully

▼ Question 2

I take the following steps to detect line:

- image is read and converted to grayscale
- Diagonal is calculated and rho and theta list is defined and HoughMatrix is initialized
- image is then blurred and edges are found using cv2.Canny
- All the coordinates that are part of and edge are iterated and for every theta T, rho is calculated
  - H[rho, T] is then incremented by 1
  - H is returned

- argmax of the Hough Matrix is used to calculate rho and theta and finally draw the line

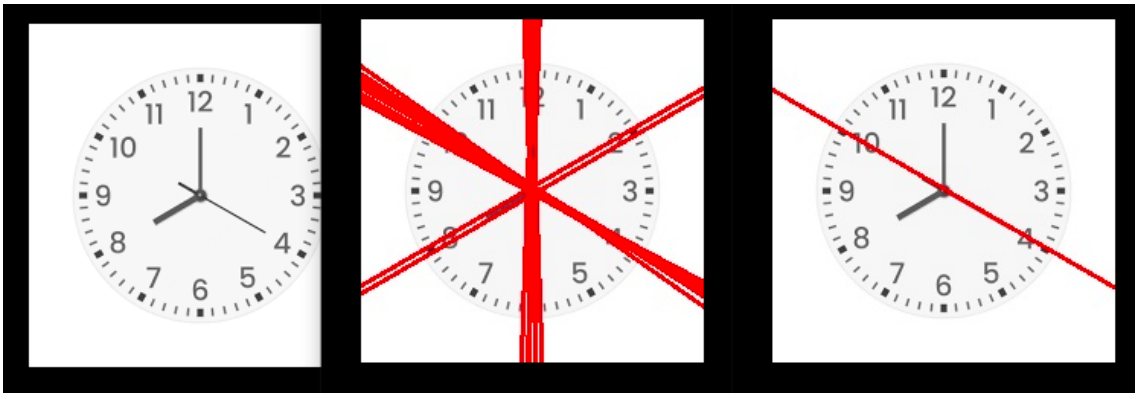
Analysis

- Speed: When it comes to speed, OpenCV is going to be faster anyways because it is natively written in C++.
    - Considering python is 25 times slower than C++, our implementation was found to be 12 times slower in one image while about 8.5 times slower in another image.

```
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 HoughOpen.py --image clock_1.png
OpenCV time: 11.539117 ms
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 HoughTransSimple.py --image clock_1.png
rho:  -40 theta:  -1.0471975511965976
Scratch time: 3427.0151370000003 ms
```

```
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 HoughTransSimple.py --image sea.jpeg
rho:  -102 theta:  -1.5707963267948966
Scratch time: 6856.480093 ms
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 HoughOpen.py --image sea.jpeg
OpenCV time: 32.572632 ms
```

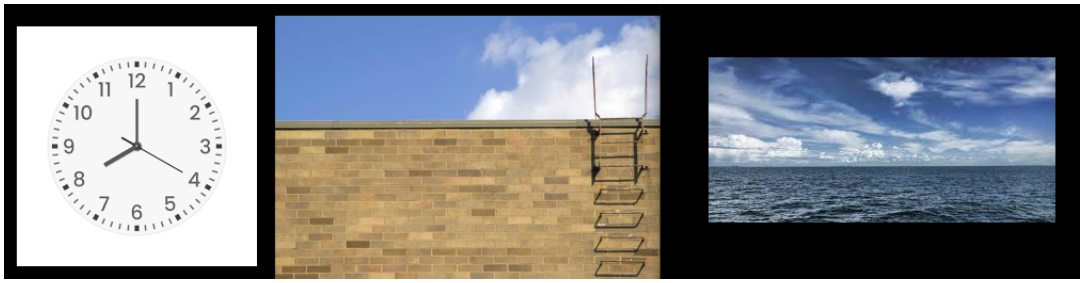
- Performance: By design, the scratch implementation can only detect one line. This line is the most dominant line in the image



Original Image; Using OpenCv; implementation; Scratch

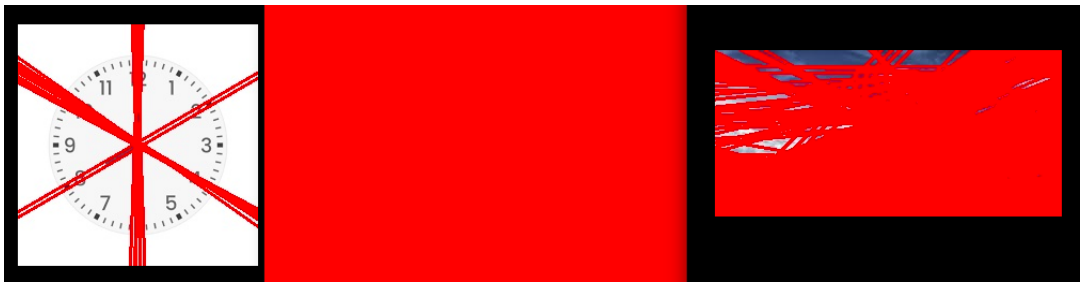
- Robustness: However, a place where our implementation beats OpenCV is robustness.

- In OpenCV, the `votes` parameter in the HoughLines function of OpenCV determines the minimum number of votes or intersections required for a detected line to be considered valid. It is used to filter out weak or spurious lines that may have been detected due to noise or other factors. The exact value of "votes" depends on the specific application and the quality of the input image, with higher values resulting in fewer but more robustly detected lines, and lower values resulting in more lines but with a higher chance of false positives.
- This value of votes needs to be chosen manually, however no such manual basis is needed in our implementation

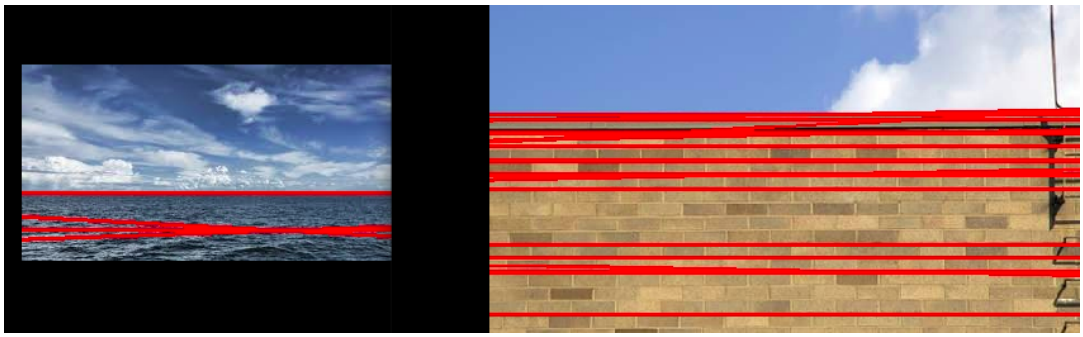


Original images

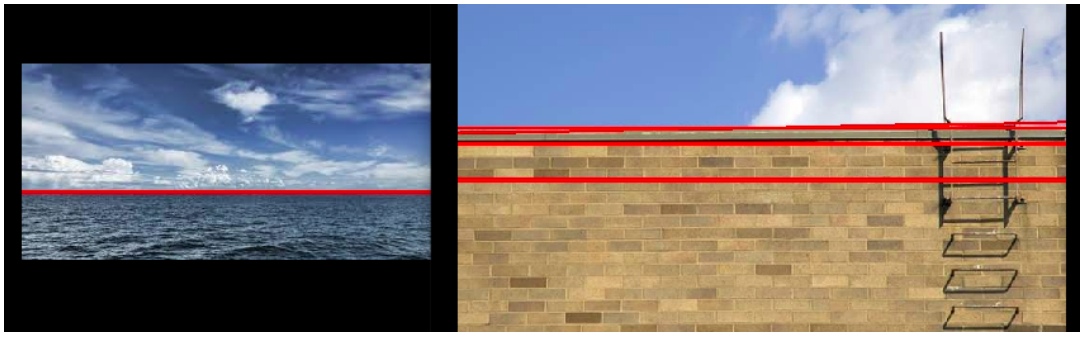
Our approach (no faulty lines)



OpenCV results with num of votes = 45



With 120 votes; Sea is giving some wrong lines



With 150 votes; no lines were detected in the clock

`Hough Line detection for multiple lines was attempted; but could not be implemented in due time`

▼ Question 3

Various approaches were tried

▼ Approach I

Filename: `q3_1.py`

Idea: 2 images are taken and features are found using SIFT. Feature matching is done using bruteforce matcher and number of good matches is found. If the number of matches is greater than a given threshold T, then the probe image is perfect, else faulty.

This approach fails as number of matches for the two perfect images are 500 and 956 while for the faulty images, it is 940 and 921

▼ Approach II and III

We can clearly see that the resolution and the aspect ratio of the two images also happen to be different. In this approach, we first get rid of the white background as much as possible and then reshape the result image to same resolution and absolute difference is found in **approach II** and number of features is found in **approach III**. If the difference is above a certain threshold, its faulty else perfect.

Even this approach fails and gives similar results like before.

This can be concluded that number of feature matches hold no significance.

▼ Approach IV

In this approach, a hypothesis is hypothesized that there might be a perspective difference in the two images. Consequently, the following steps are taken

- White background is removed from both the images
- Aspect ratio of both the probe and reference images is then made same
- They both are then resized to the same smaller size.di
- Features in both the images is found using SIFT and using best 50 matches from a BFMatcher, homography matrix is calculated
- Using the homography matrix, reference image is warped to probe image using warpPerspective() of cv2.
- Absolute difference between the warped reference and the original probe image is found and if it is found to be above a said threshold, the probe is faulty else it is not.
    - If the image is faulty, the region where it is fault is then saved.

This approach worked and here are the results.

Usage:

```
python3 intelligentMatch.py --ref Problem-3/e1/reference.png --img Problem-3/e1/perfect/p2.png
```

```
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 intelligentMatch.py --ref Problem-3/e1/r
eference.png --img Problem-3/e1/faulty/f2.png
[ WARN:0@0.083] global shadow_sift.hpp:13 SIFT_create DEPRECATED: cv.xfeatures2d.SIFT_
create() is deprecated due SIFT tranfer to the main repository. https://github.com/ope
ncv/opencv/issues/16736
probe is faulty
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 intelligentMatch.py --ref Problem-3/e1/r
eference.png --img Problem-3/e1/faulty/f1.png
[ WARN:0@0.091] global shadow_sift.hpp:13 SIFT_create DEPRECATED: cv.xfeatures2d.SIFT_
create() is deprecated due SIFT tranfer to the main repository. https://github.com/ope
ncv/opencv/issues/16736
probe is faulty
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 intelligentMatch.py --ref Problem-3/e1/r
eference.png --img Problem-3/e1/perfect/p1.png
[ WARN:0@0.080] global shadow_sift.hpp:13 SIFT_create DEPRECATED: cv.xfeatures2d.SIFT_
create() is deprecated due SIFT tranfer to the main repository. https://github.com/ope
ncv/opencv/issues/16736
probe is ok
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 intelligentMatch.py --ref Problem-3/e1/r
eference.png --img Problem-3/e1/perfect/p2.png
[ WARN:0@0.035] global shadow_sift.hpp:13 SIFT_create DEPRECATED: cv.xfeatures2d.SIFT_
create() is deprecated due SIFT tranfer to the main repository. https://github.com/ope
ncv/opencv/issues/16736
probe is ok
ceyxasm@pop-os:~/.../A2/Assignment-2$
```

Results of example 1

```
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 intelligentMatch.py --ref Problem-3/e2/r
eference.png --img Problem-3/e2/perfect/p1.png
[ WARN:0@0.064] global shadow_sift.hpp:13 SIFT_create DEPRECATED: cv.xfeatures2d.SIFT_
create() is deprecated due SIFT tranfer to the main repository. https://github.com/ope
ncv/opencv/issues/16736
probe is ok
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 intelligentMatch.py --ref Problem-3/e2/r
eference.png --img Problem-3/e2/perfect/p2.png
[ WARN:0@0.061] global shadow_sift.hpp:13 SIFT_create DEPRECATED: cv.xfeatures2d.SIFT_
create() is deprecated due SIFT tranfer to the main repository. https://github.com/ope
ncv/opencv/issues/16736
probe is ok
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 intelligentMatch.py --ref Problem-3/e2/r
eference.png --img Problem-3/e2/faulty/f2.png
[ WARN:0@0.054] global shadow_sift.hpp:13 SIFT_create DEPRECATED: cv.xfeatures2d.SIFT_
create() is deprecated due SIFT tranfer to the main repository. https://github.com/ope
ncv/opencv/issues/16736
probe is faulty
ceyxasm@pop-os:~/.../A2/Assignment-2$ python3 intelligentMatch.py --ref Problem-3/e2/r
eference.png --img Problem-3/e2/faulty/f1.png
[ WARN:0@0.066] global shadow_sift.hpp:13 SIFT_create DEPRECATED: cv.xfeatures2d.SIFT_
create() is deprecated due SIFT tranfer to the main repository. https://github.com/ope
ncv/opencv/issues/16736
probe is faulty
ceyxasm@pop-os:~/.../A2/Assignment-2$
```

Results of example 2

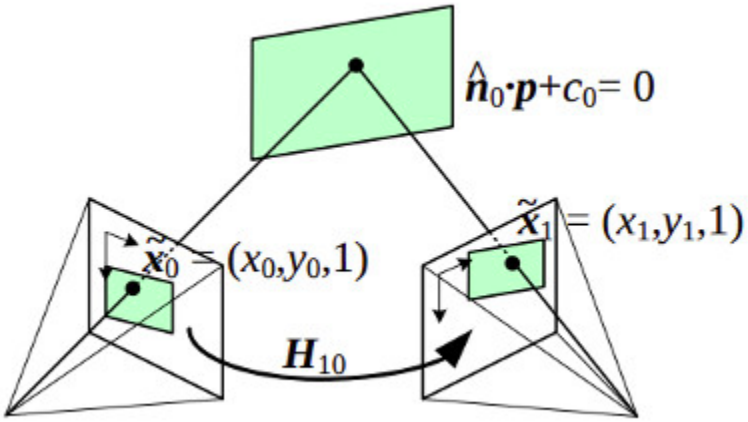
White blobs show regions of error

▼ Part B

▼ Question 1

What is homography and homography matrix?

- Homography enables us to connect two viewpoints that are looking at the same plane. The cameras and the surface they are watching (and creating images of) are both placed in the world's coordinates. In other words, if two 2D images each see the same plane from a different perspective, they are associated by a homography H.
- By multiplying the Homography matrix with the points in one perspective to determine their equivalent locations in another view, it enables us to switch between one view and another of the same scene.
- Briefly, homography relates transformation between two planes having same centre of projection or the scene points should lie in a plane or be distant or imaged from the same point

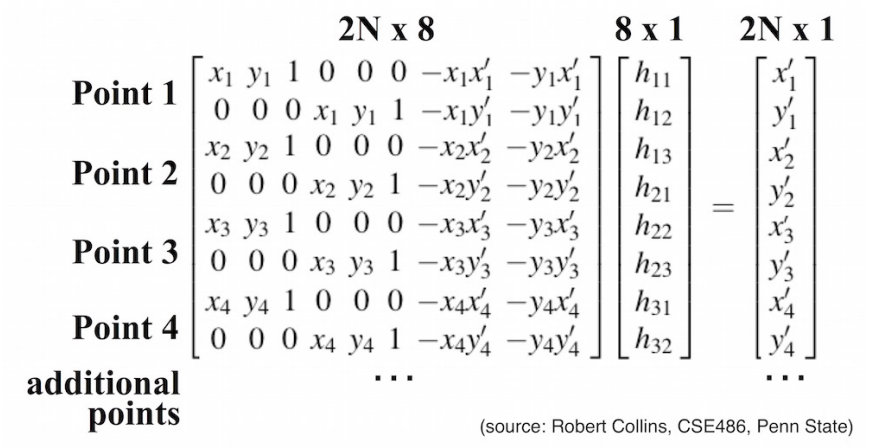


A planar surface viewed by two camera positions (Image taken from here)

Calculating Homography matrix

Homography can be calculated using relative rotation and translation between two viewpoints.

- Before we calculate it, we are going to assume some of its properties as given and will not indulge in its derivation/proof
  - It is a full rank matrix
  - It has degree of freedom =8, even though it has 9 elements.
- Steps
  - Given 2 images, using a feature detector and a matcher, corresponding points are found
    - Atleast 4 pairs are needed
  - For a given pair of points, equations are written
  - They are rearranged, written in matrix form and stacked



- The resulting equation can be solved using constrained least squares
- The loss function is defined, the first derivative of which comes to be a simple **eigenvalue problem** that can then be solved easily

$$min_h ||Ah||_2 \ s.t. ||h||_2 = 1$$
or
$$min_h ||Ah||_2 \text{ s.t. } (1 - h^T h) = 0$$
where
$$A \in \mathbf{R}^{mX9} \ m >= 8$$
$$h \in \mathbf{R}^{9X1}$$

Using Langrange we can write the above optimization as

$$min_h(||Ah||_2^2 + \lambda(1 - h^T h))$$

Taking partial derivative wrt h and setting it as 0

$$(A^T A - \lambda I)h = 0 \text{ and since } h \neq 0$$

$$det(A^T A - \lambda I) = 0$$

Complete derivation is done as below:

Let $X_s \equiv \begin{pmatrix} x_s \\ y_s \\ 1 \end{pmatrix}$ and $X_d \equiv \begin{pmatrix} x_d \\ y_d \\ 1 \end{pmatrix}$ be coordinates of corresponding points in homogeneous system.

And let our homogeneous matrix be

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{21} & h_{32} & h_{33} \end{bmatrix}$$

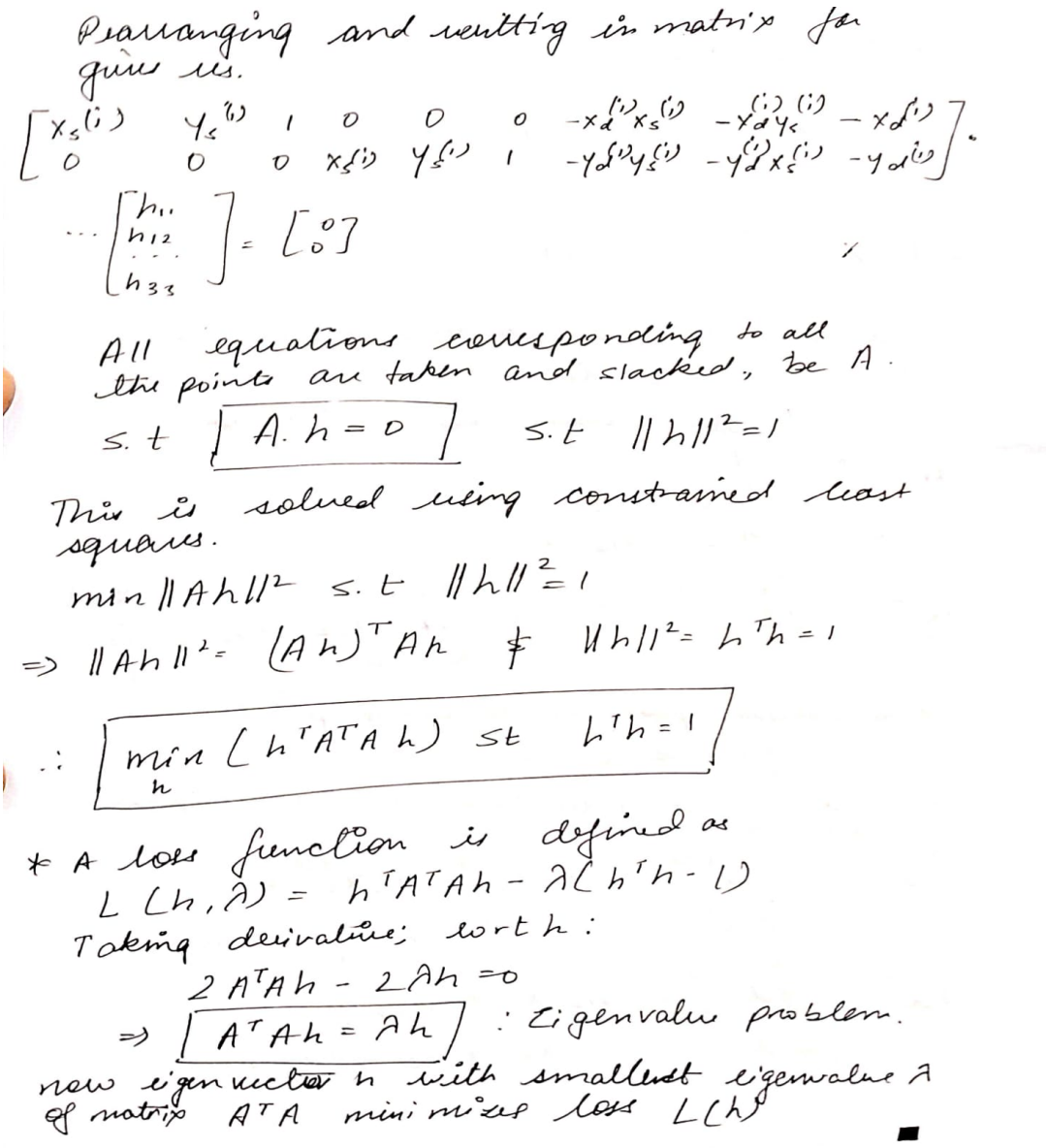
Let $\begin{pmatrix} X_d \\ Y_d \\ 1 \end{pmatrix} \equiv \begin{pmatrix} \bar{x}_d \\ \bar{y}_d \\ \bar{z}_d \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} X_s \\ Y_s \\ 1 \end{pmatrix}$

4 minimum pair of matching points are required to solve this

For a given pair (i) of corresponding points:

$$x_d^{(i)} = \frac{\tilde{x}_d^{(i)}}{\tilde{z}_d^{(i)}} = \frac{h_{11} x_s^{(i)} + h_{12} x_s^{(i)} + h_{13}}{h_{31} x_s^{(i)} + h_{32} y_s^{(i)} + h_{33}}$$

$$y_d^{(i)} = \frac{\tilde{y}_d^{(i)}}{\tilde{z}_d^{(i)}} = \frac{h_{21} x_s^{(i)} + h_{22} y_s^{(i)} + h_{23}}{h_{31} x_s^{(i)} + h_{32} y_s^{(i)} + h_{33}}$$



It can be calculated using OpenCV by

```
h, status = cv2.findHomography(pts_src, pts_dst)
```

▼ Question 2

- Stereo matching compares discrepancies between two or more photographs of the same scene obtained from various angles or viewpoints in order to estimate the depth information of objects in the scene.

- Finding corresponding spots or features in the photos is required.
- Stereo matching is still a difficult challenge, especially when there are occlusions, noise, and complicated scenes present.
- Applications of stereo matching
    - 3D Reconstruction: Stereo matching is frequently used to create 3D representations of objects, buildings, and environments. We can recreate the 3D model using scene depth information using stereo matching.
    - Stereo matching helps autonomous vehicles understand the 3D world and navigate safely. Stereo matching lets autonomous vehicles recognise impediments, estimate vehicle distances, and assess road geometry.
    - Augmented Reality: Stereo matching detects object position and orientation in augmented reality applications. Stereo matching lets us estimate depth and generate realistic 3D overlays.
    - Medical Imaging: CT and MRI use stereo matching to build 3D body images.
    - Robotics: Stereo matching helps robots recognise, grasp, and manipulate objects. Stereo matching allows robots to comprehend the 3D environment and interact with items more naturally.
    - Further uses are in autonomous driving, point cloud construction, volume estimation etc.

▼ Question 3

Panorama stitching

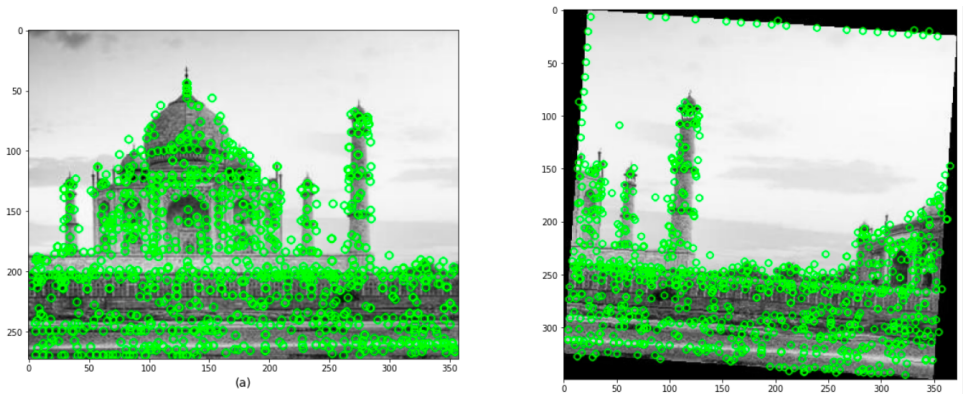
Panorama stitching is the process of stitching images by overlapping common regions. Its involves following principles

- Feature detection
- Feature scoring
- Feature description

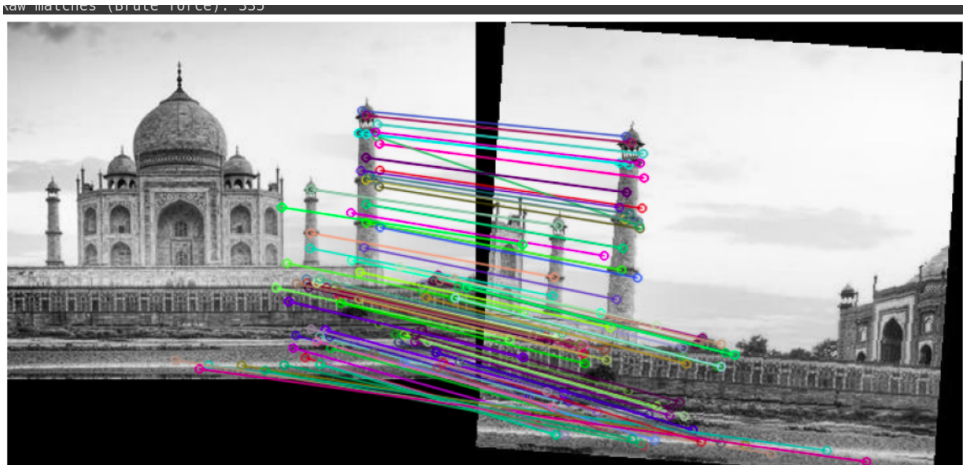
- Feature matching
- Finding best transform
- Composition

Now let us look at the steps:

1. Load the images: The first step is to load the individual photographs that will be stitched together. In order to identify common features, it is crucial to make sure that the photos somewhat overlap. **In all the implementations tried, the ordering of the loaded images is very important.**
2. Feature detection: For feature detection, a number of methods are available, including SIFT, SURF, ORB, and others. These algorithms produce descriptors and recognise keypoints (areas in the image that stand out) (vectors that describe the appearance of the keypoints).



3. Feature matching: Feature matching is the process of comparing keypoints and descriptors between two images once they have been calculated. For feature matching, a variety of algorithms are available, including FLANN-based matching and brute-force matching.



4. Calculate the homography: Following feature matching, you must calculate the homography matrix, which explains how the two pictures were transformed. A 3x3 matrix called the homography matrix connects the pixel coordinates of one image to the pixel coordinates of another image.

```
[[ 1.18620403e+00 -8.22372526e-02 -2.01688775e+02]
 [ 8.09838490e-02  1.18372554e+00 -1.48063097e+01]
 [-8.89310299e-06 -2.17310633e-06  1.00000000e+00]]
```

5. Image warping: Using the homography matrix, you warp one image in this stage such that it lines up with the other image. To do this, each pixel in the second image must be subjected to the homography matrix in order to ascertain where in the first image it belongs.
6. Blending: Next, blending can be applied to make the transition between images seamless
    a. above steps are then repeated to stitch further images.

▼ Implementation

- OpenCV implementation is attached
- Scratch implementation was also attempted however it is failing for some reason for Taj Mahal images
  - Taj Mahal Stitching- here
  - However, scratch stitching works for this other set of images- here
    - The code file demonstration is from CV_project_1
  - Varying feature detectors were used, BF matcher with and without KNN was used, stitching with two images was used
    - Reasons can be so because the shape of **Taj Mahal** images are different and warping is giving error.