COL780 Assignment 2

Harris Corner Detection (Y=0)

Samarth Bhatia
2019CH10124

Details/Formulation

I have implemented a Harris Corner Detector to detect object corners from different scenes of an image so that we can stitch them afterwards to make a panorama, using an affine model.

Harris Corner Detection works on the following principles:

$$M = \sum_{x,y} G(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{1}$$

$$where,\ I_x = \frac{\delta I}{\delta x}\ and\ I_y = \frac{\delta I}{\delta y} \tag{2}$$

$$H = \mathbf{det}(M) - k \cdot \mathbf{tr}(M)^2 \qquad = \lambda_1 \cdot \lambda_2 - k \cdot (\lambda_1 + \lambda_2)^2 \tag{3}$$

$$or \tag{4}$$

$$H = \frac{\mathbf{det}(M)}{\mathbf{tr}(M)} \qquad = \frac{\lambda_1 \cdot \lambda_2}{\lambda_1 + \lambda_2} \tag{5}$$

Here, H is the "Harris Corner Matrix", which tells us whether the point is an edge, corner or a flat region based on the eigenvectors.
G can be chosen to be a gaussian window or a uniform weight window around the point (x, y). Eqn. 4 was the original one from Harris' paper, eqn 5 was given a few years later. However, the original one worked better for me in practice.

I have used the Sobel filter (separable) to efficiently calculate the derivative of the intensities in both directions. Also, I have used a gaussian window in the above.

When the point is a corner, both λ_1 and λ_2 are large. For an edge, only one of them is large while other one is small. For a flat region, both of them are small. Correspondingly, we get:

- Corner: $H > thresh$
- Edge: $H < 0$
- Flat: $|H| \approx 0$

Instead of using a threshold, I have taken the top `N`=500 points with largest value of H as corners. However, we get a large amount of corners by this which are overlapping. So, I have also used NMS (Non-Maximal Suppression) with a window of (5,5). This is all implemented in the `HarrisCornerDetector` function.

To make features in the `extract_decriptors` function for each NMS corner point, I take a patch of (60,60) around a corner, and take pixels at a spacing of 5. This makes a vector of size 432 to match with other points, so that the points detected are very accurate and repeatable. I tried reducing this to (40,40) but then the matches were not that accurate (especially in dataset 1, which has the large

almost uniform yellow region with black dots, so smaller size is unable to locate points.)

Then I match these features for two images (with possibly unequal number of corners each) using a sum of squared distance (Euclidean distance) b/w the vectors. This distance is thresholded below `matching_thresh` to get possible matches. Then all these possible matches are returned.

In the `match_corners` function, I start with the top 3 matches and then check if any 2 of them are equal. (Because if they are equal, we will not be able to get an affine transformation matrix). So I iteratively take the first 3 unequal pairs of points.
These are used to get the affine matrix `M`.

$$
\begin{aligned}
M &= \begin{bmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \end{bmatrix} \\
\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}
\end{aligned} \tag{6}
$$

I have then affine transformed the first image to stitch them with the second image into a panorama, and repeated this. So, I iteratively build the panorama into the first image. The `warp_and_pad` function does this.

I need to use the following parameters which need to be adjusted for getting the best result:

1. `scale`: The image is rescaled to `original_size/scale` to reduce computation time, as results dont change much with high res images. `5`
2. `k`: This is the k used to find the Harris corner matrix. It is set to `0.06`.
3. `patch_size`: This controls the size of the area around the pixel which is taken as features to match. `(60,60)`
4. `patch_spacing`: This controls the spacing at which pixels are selected in the patch around that pixel. `5`
5. `matching_thresh`: This is an upper-threshold on the distance between two features to consider them as (potentially) matching points in two images. Currently, it is `60`.

These can be found in the code.

Implementation

The python file `det.py` implements the Harris Corner detection and image stitching (affine).

```
usage: main.py [-h] [--dataset DATASET] [--show SHOW] [--scale SCALE] [--
patch_size PATCH_SIZE]
               [--patch_spacing PATCH_SPACING]

optional arguments:
  -h, --help            show this help message and exit
  --dataset DATASET     1-6
  --show SHOW           0/1/2, 0=Only Panorama, 1=Panorama+Matching Points,
2=Panorama+Matching Points+All Corners
  --scale SCALE         (H,W)/scale, as a lot of images are very large (more
than 3000*2000)
  --patch_size PATCH_SIZE
                        x: a patch of size (x,x) with patch_spacing is matched
around corners
  --patch_spacing PATCH_SPACING
                        spacing: a patch of size (x,x) with patch_spacing is
matched around corners
```

Results and Analysis of success and failure cases

Dataset 1 : Success till last image

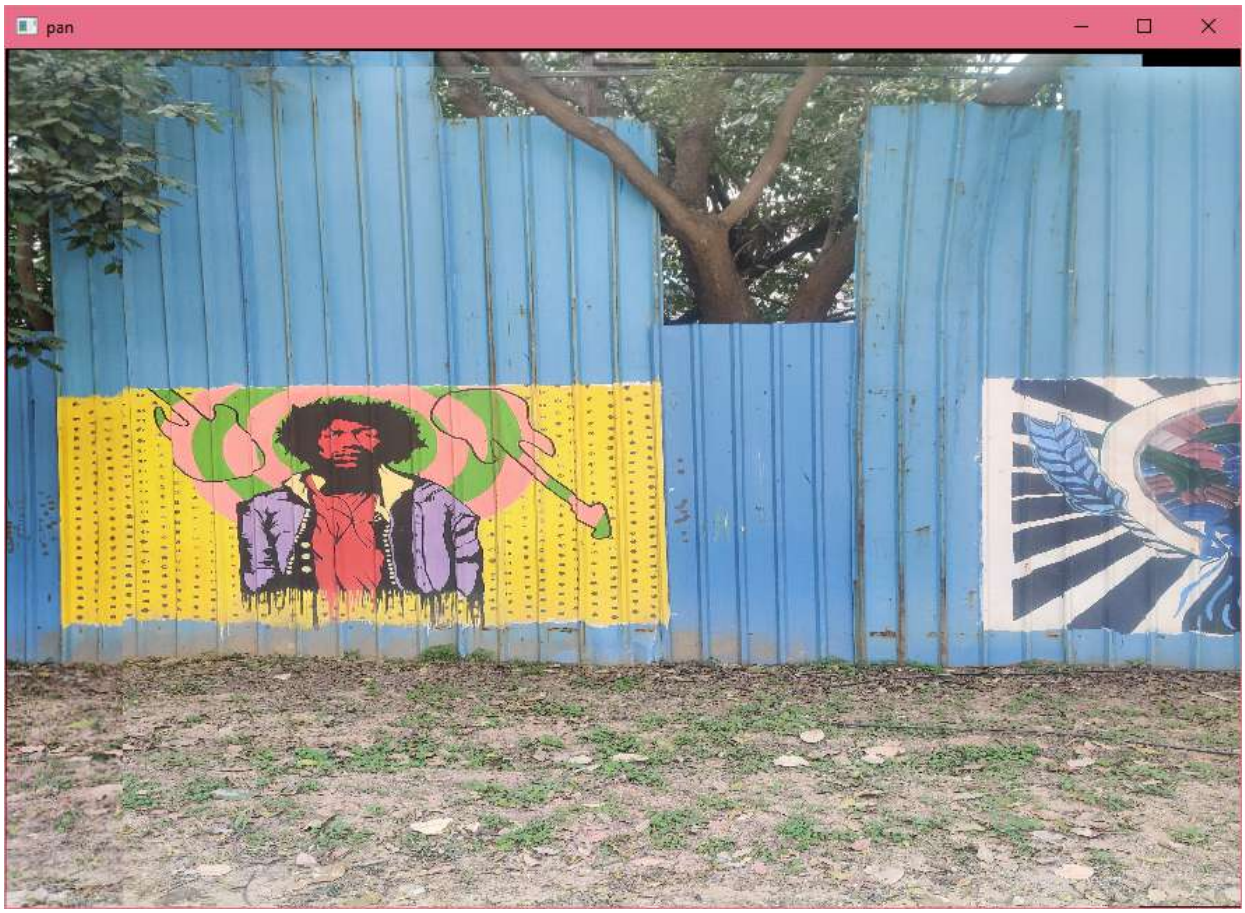
Corner Detection Results

The corners detected by the Harris detector are shown in a purple shade. These results are after NMS. Mainly in the jacket region and the black dots in the yellow region. Note that while taking the derivative the images were converted to grayscale, so corners of the yellow painting are not detected as weighted grayscale intensity does not change much there.

Corner Matching and Panorama Stitching Results

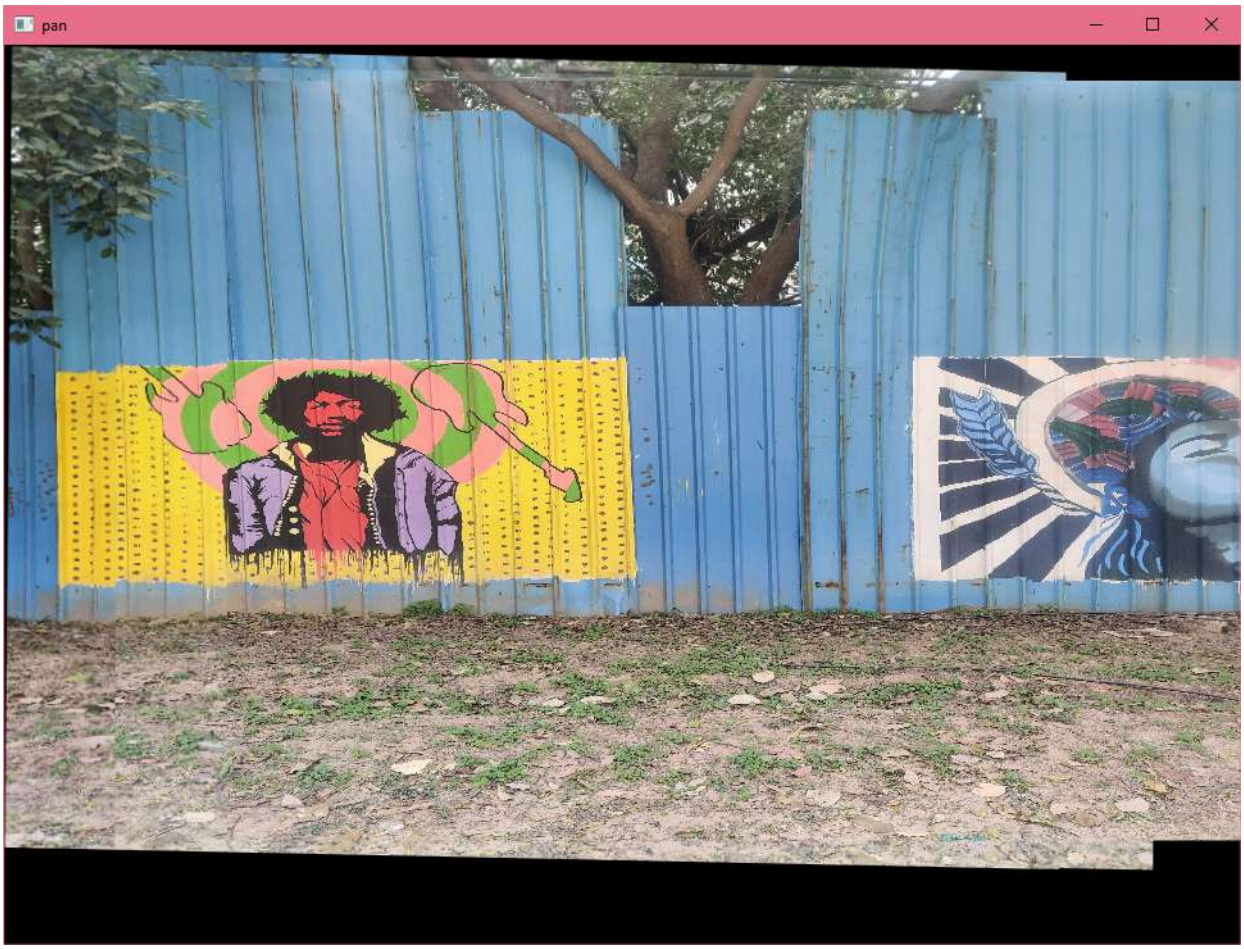
Matching 1st and 2nd images (the 3 points used for affine transformation are shown as small colored squares) [near the jacket buttons]

Resulting panorama (1+2)



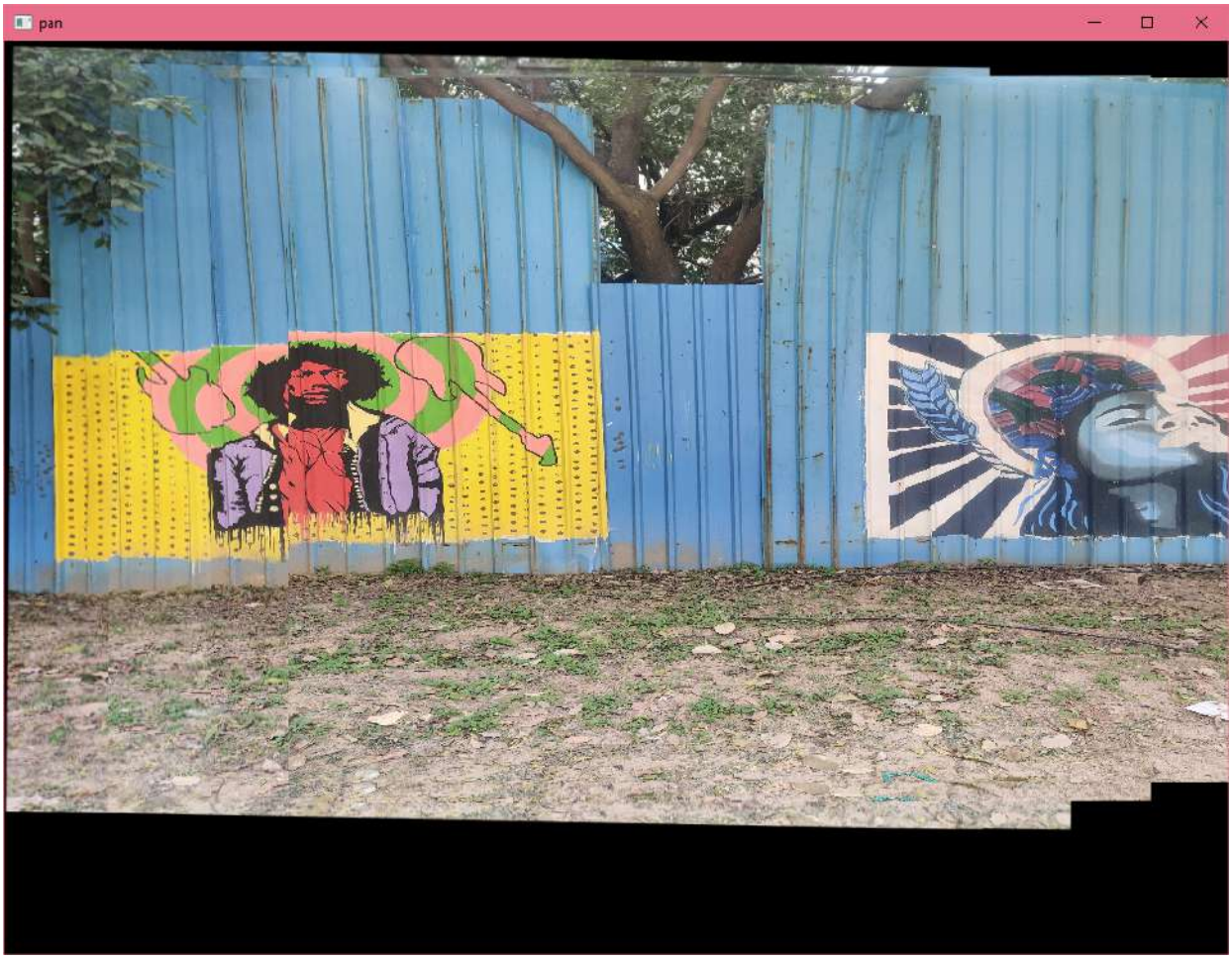
Matching (1+2) with 3

Resulting panorama (1+2+3)



Matching (1+2+3) with 4

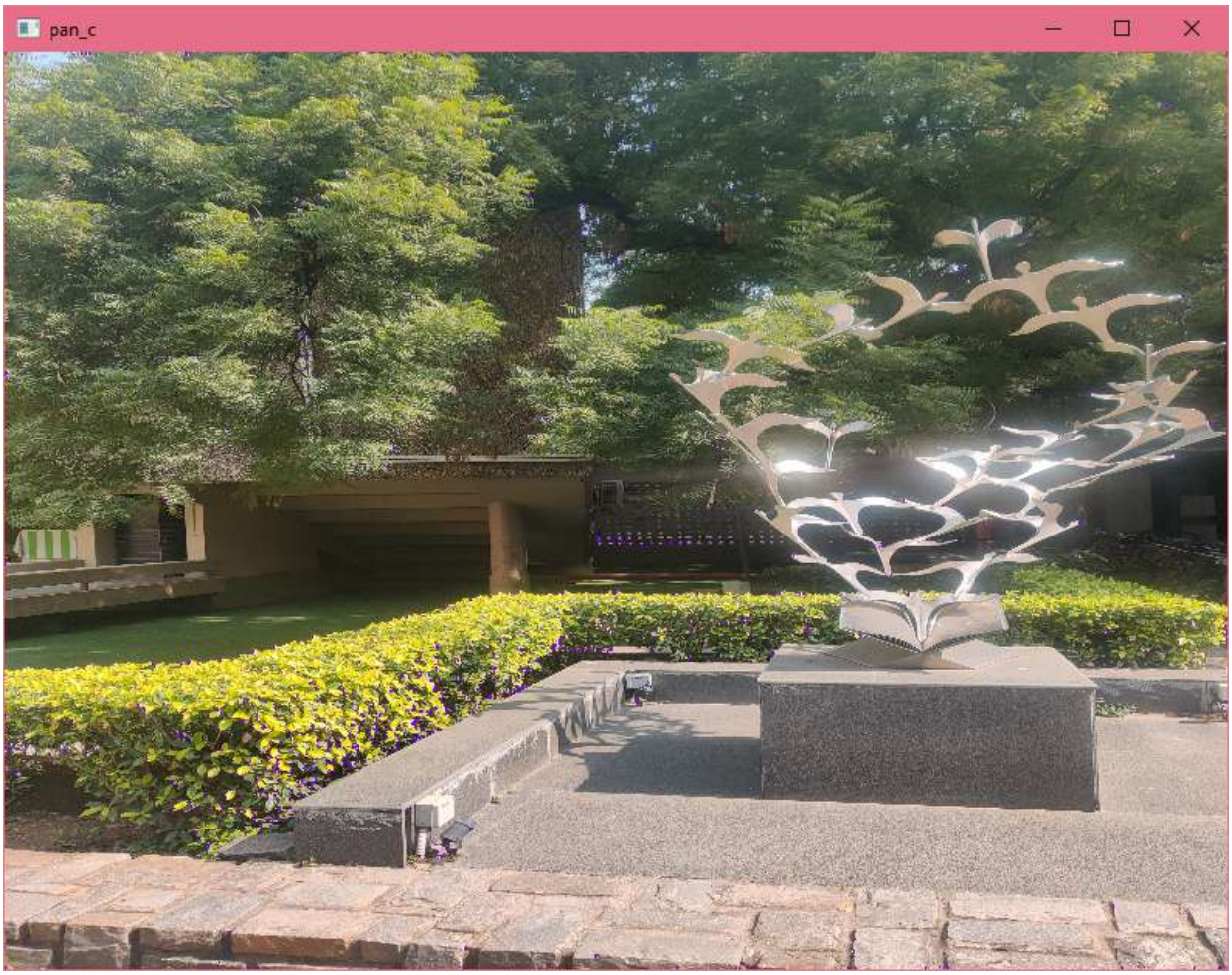
Resulting panorama (1+2+3+4)



Dataset 2 : Almost all perfect, very good result

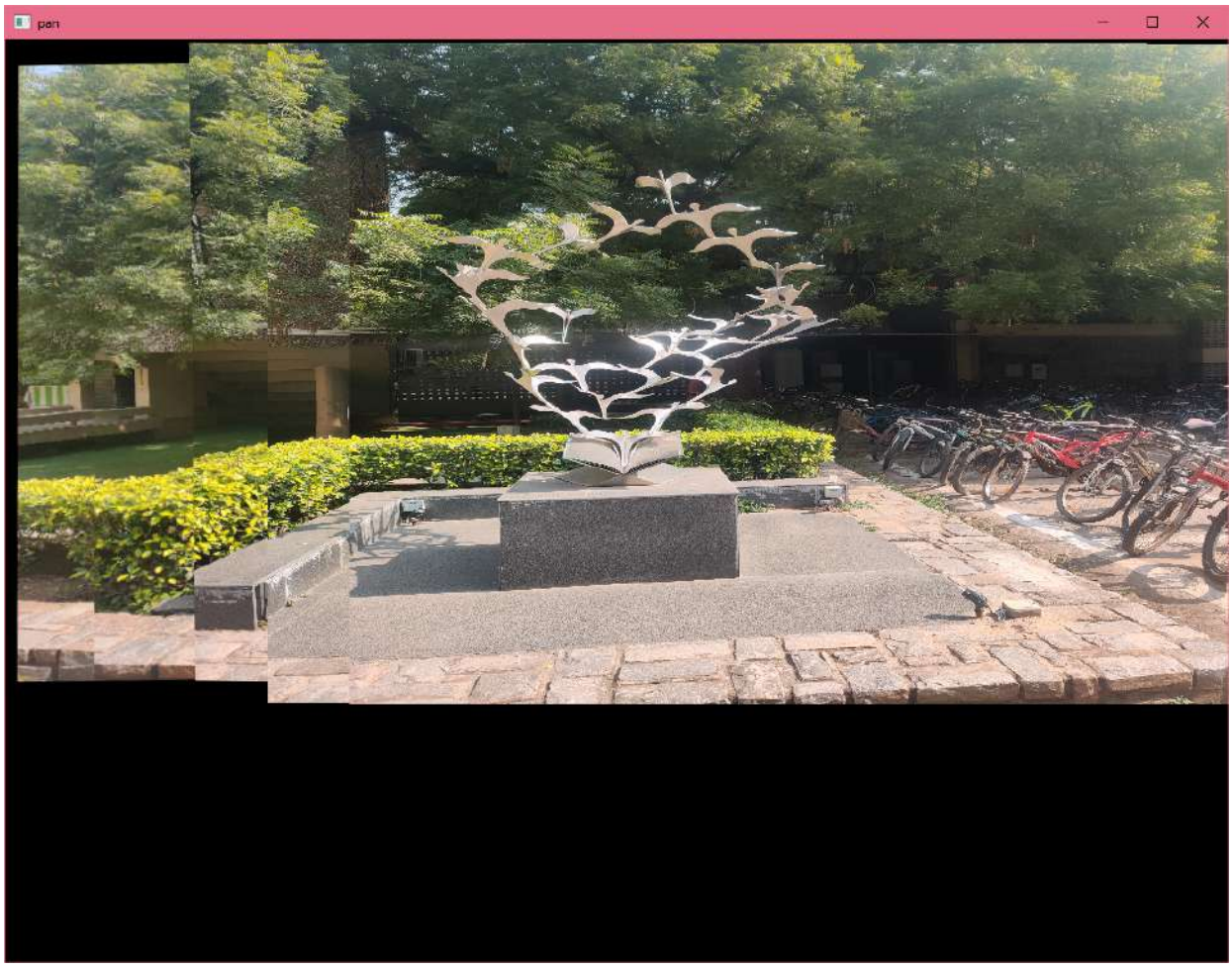
Corner Detection

The corners detected by the Harris detector are shown in a purple shade. (There are corners detected on the statue as well but not visible because of less contrast)



Final Panorama

We see that the leftmost (first) image didn't stitch properly, but all the rest of them did (they stitched very well, we can tell by the line of the bushes and the footpath on the left side, bushes are almost continuous as well as footpath).

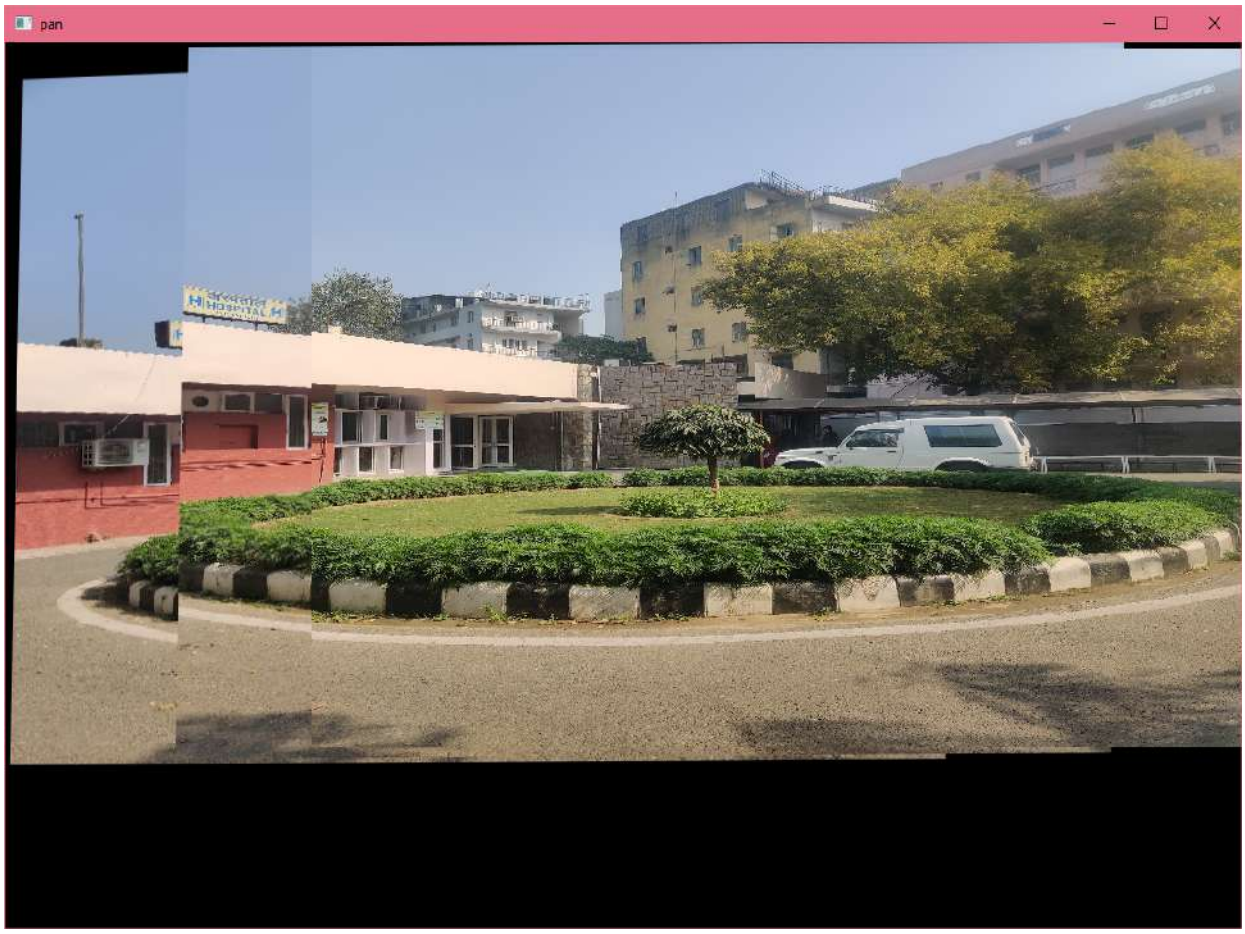


Dataset 4, 5 (matches shown) : Bad stitches

For dataset 4, the panorama is okayish till image 3, but after that a wring matching is done (notice the yellow point is on the first glass window left of pink point instead of the second glass window). This happens because the texture of the 2 glass windows and the surrounding roof and sky is very similar, and not a lot of distinction is present in a (60,60) patch. Increasing the patch size is a way to fix this. The same dataset with a patch size of 100 gives this result with a successful 4th stitch. However, quality of 1st stitch goes down.

Part of the problem with this dataset is that the "roof" of the building is curved - we can see it is closer towards the edges. This means no finite amount of affine stitching will be able to form a perfect circular stitch (as the same point which is higher in one image will be at a lower level in the next image).

Dataset 5:



Conclusion

We conclude that Harris corner detection works decently well on images. Feature extraction, feature matching and affine stitching are another deal and since methods for those were not taught in class (except affine transforms), I assumed that Harris Corner Detection will be worth more.

There can be improvements to this directly. Currently, the corner detection and feature extraction works very well.
For example, while matching the features from 2 images, I have added checks in order to prevent the 3 chosen points (for calculating the affine transform) to form a very scalene triangle, i.e. every angle must be $\geq$ `min_angle`=15°. This ensures that the affine matrix calculated is good and the transformed image also has a good overlap. While I have implemented one that is not complex (which works decently), there could definitely be improvements, for example focusing on something other than the angles of the triangle formed.

Another improvement we could do is calculate the derivative while keeping RGB channels separate, and finding a way to combine them into one gradient s.t. we can find corners. However, that does increase computation time.

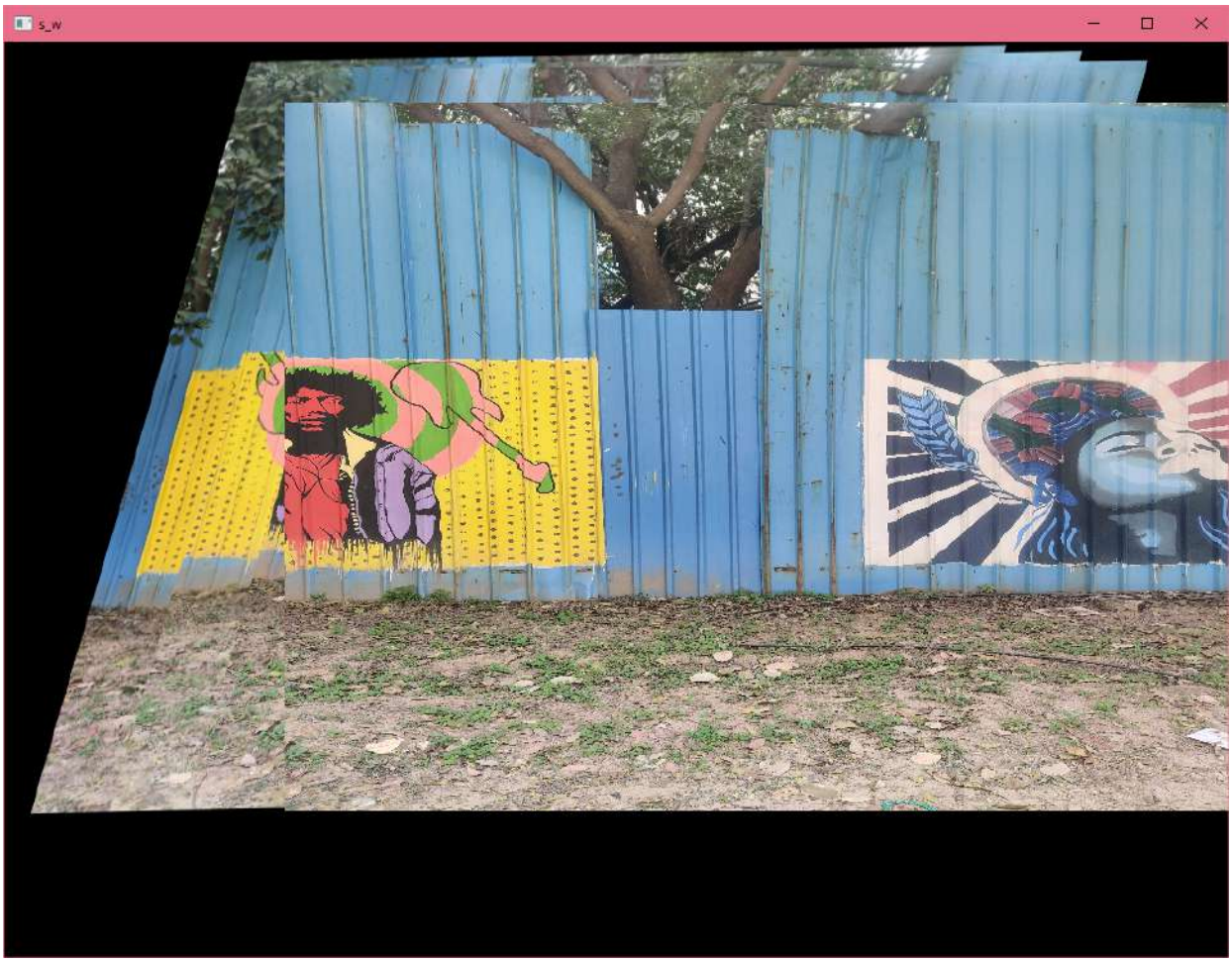
Old Results (and how I got improvements)

Dataset 4 - Collinearity: Result did not generate well because the points detected were (almost) collinear, (so they are not linearly dependent; we can write one in the terms of the other two). This caused the affine transformation matrix to become: $\begin{bmatrix} 2.925 & 0.05 & -575.075 \\ 7.75 & 1.15 & -2415.225 \end{bmatrix}$.
As is obvious, the **shearing term in y is 7.75** ! So the resulting image is degenerated.
Implemented Fix : Check points are not collinear/the triangle formed by them has all angles $\geq$ `min_angle` (set to 15\degree)
This greatly improved the quality of all other datasets as well.

Dataset 1 - Not enough features to match: Matching (1+2+3) with 4 (**problematic, we can see that it matches the "white" point wrongly** under the guitar in the yellow region) This is because in a region of (40,40) around it there are mostly yellow pixels, and so it matched with another corner in that yellow region. This is where corner matching fails and the panorama degenerates.

Implemented Fix : A patch size of (60,60) seems to work well for most images.

Resulting panorama


S_W