MCEN5228 : Project One - Panorama

Team MaxVision

Max Conway
University of Colorado Boulder
Boulder, USA
Maxwell.Conway@colorado.edu

Thanushraam Suresh Kumar
University of Colorado Boulder
Boulder, USA
Thanushraam.Sureshkumar@colorado.edu

***Abstract*—In This paper we document our experience with MCEN5228 Computer Vision Project 1. In this project we are stitching together images to form a panorama. This process is done by first extracting corners of images, then we select a subset of corners that are evenly spread through the image via the ANMS algorithm, subsequently we create feature vectors from images patches around corners. We then match corners between images by a similarity metric (Sum squared differences) to find points that correspond between images. Finally we estimate a homography matrix using RANSAC to handle outliers before stitching the images together with the homography matrix**

I. Process

A. Detecting Corners

Our process begins with us first detecting corners which is a fundamental technique in computer vision to identify points in an image where the intensity changes sharply in multiple directions. Corners are necessary for panorama stitching as it helps locate distinctive points, known as keypoints, which can be matched across overlapping images. We used the cv2.cornerHarris implementation of corner detection. This method scores each pixel in the image where higher scores are more confident corners. We take the top N_{strong} corners from the cv2.cornerHarris output. You can see example outputs in figures 1 2, 3, 4, 5, 6, 7.

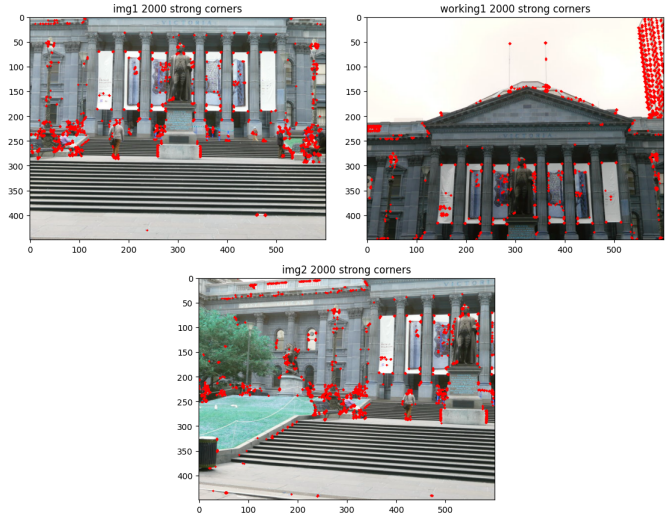

Fig. 1: Corner Detection on Victoria Dataset

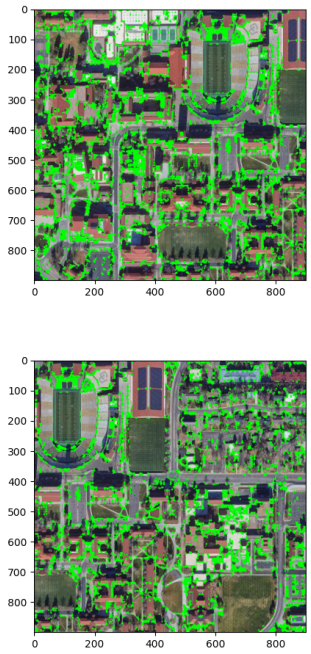

Fig. 2: Corner Detection on CU Satellite Dataset

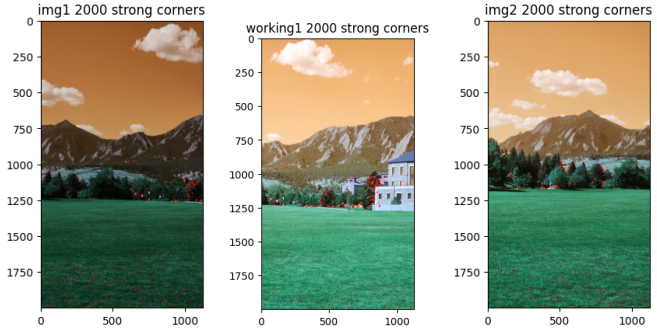

Fig. 3: Corner Detection on Flatirions Challenge Dataset

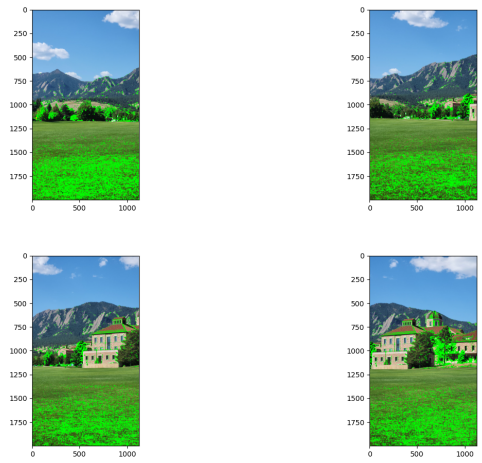

Fig. 4: Corner Detection on Flatirons Dataset

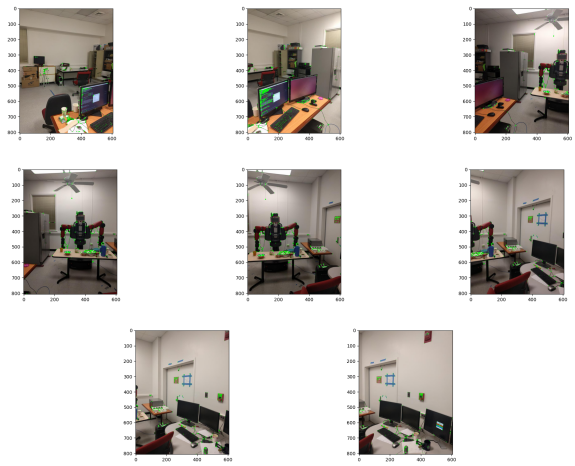

Fig. 5: Corner Detection on Indoors Dataset

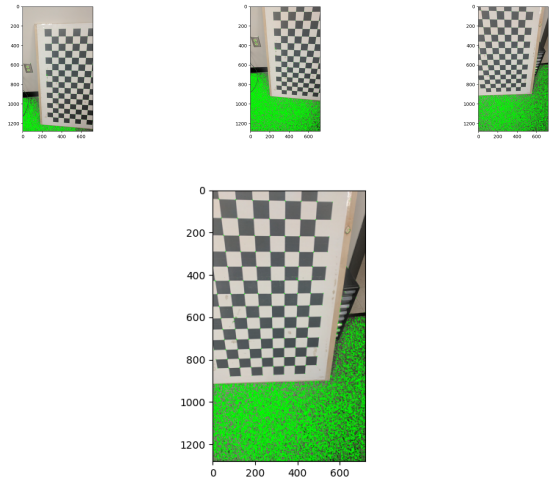

Fig. 6: Corner Detection on Checkerboard Dataset

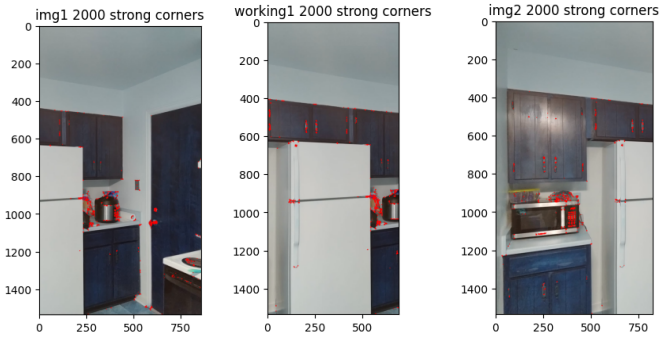

Fig. 7: Corner Detection on Our Dataset

B. ANMS (Adaptive Non-Maximal Suppression)

ANMS is an enhancement of traditional corner detection technique, it is used to select a more uniform distribution of keypoints across as an image. Given a set of N_{strong} corners our goal is to find a smaller set of N_{best} corners as determined by the ANMS algorithm which finds a subset set of a set of points so that the points are roughly equally spaced. Unlike previous corner detection where it relied solely on the strength of detected corners ANMS ensures that the selected keypoints are spatially well-distributed. You can see example outputs in figure 8, 9, 10, 11, 12, 13, 14, 15.

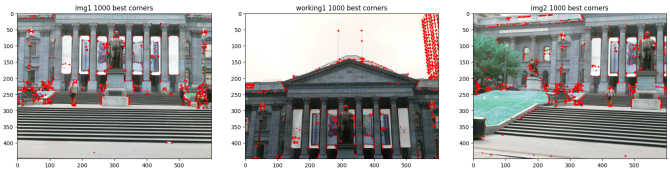

Fig. 8: Output after ANMS on Victoria Dataset

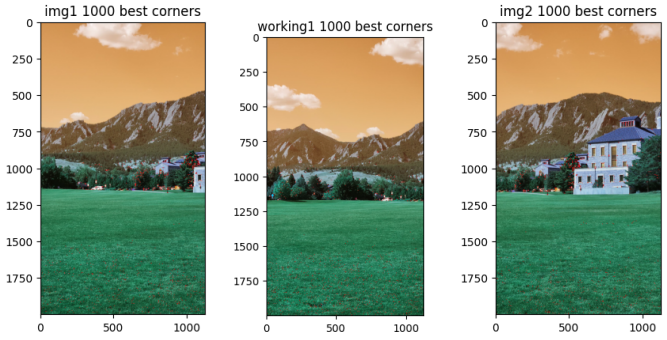

Fig. 9: Output after ANMS on Flatirons Challenge Dataset

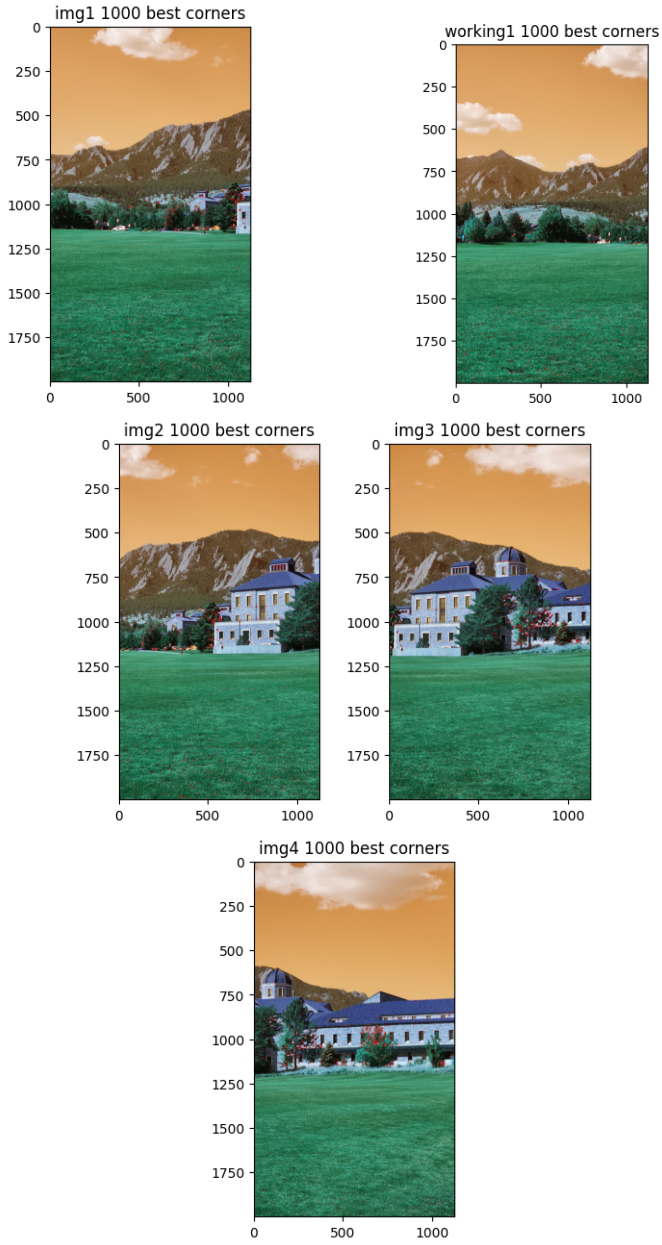

Fig. 10: Output after ANMS on Flatirons Dataset

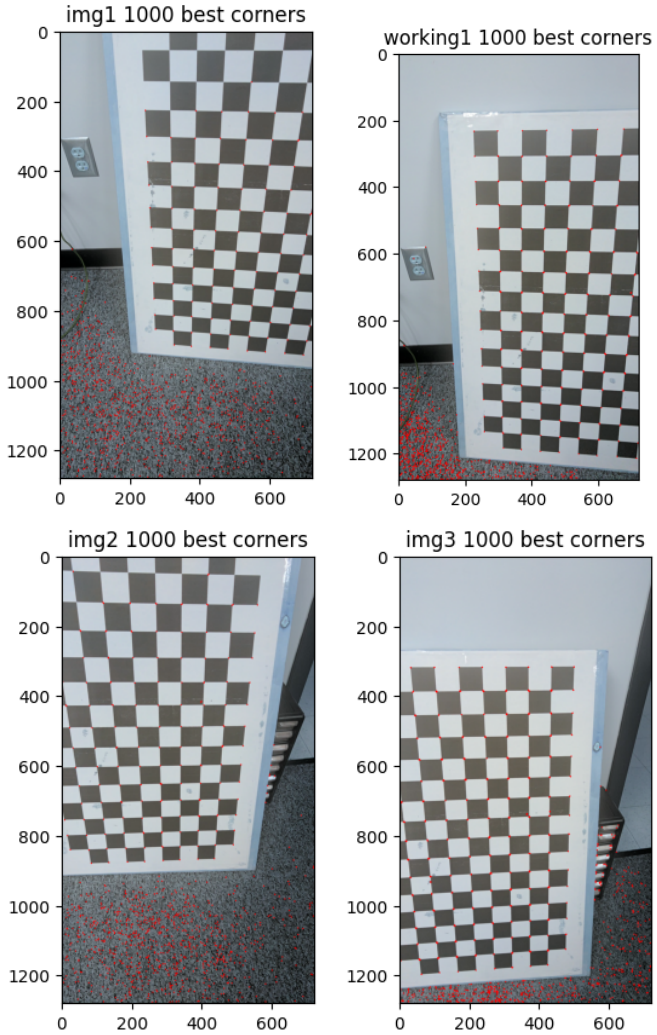

Fig. 11: Output after ANMS on Checkerboard Dataset

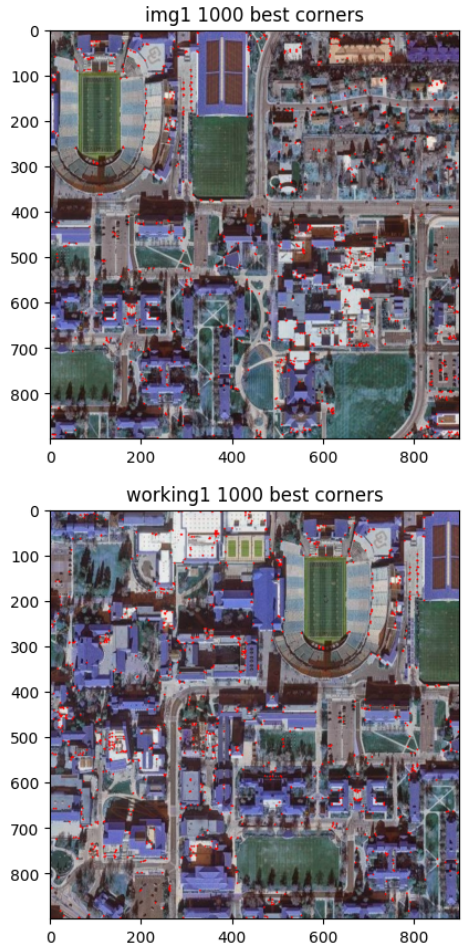

Fig. 12: Output after ANMS on CU Satellite Dataset

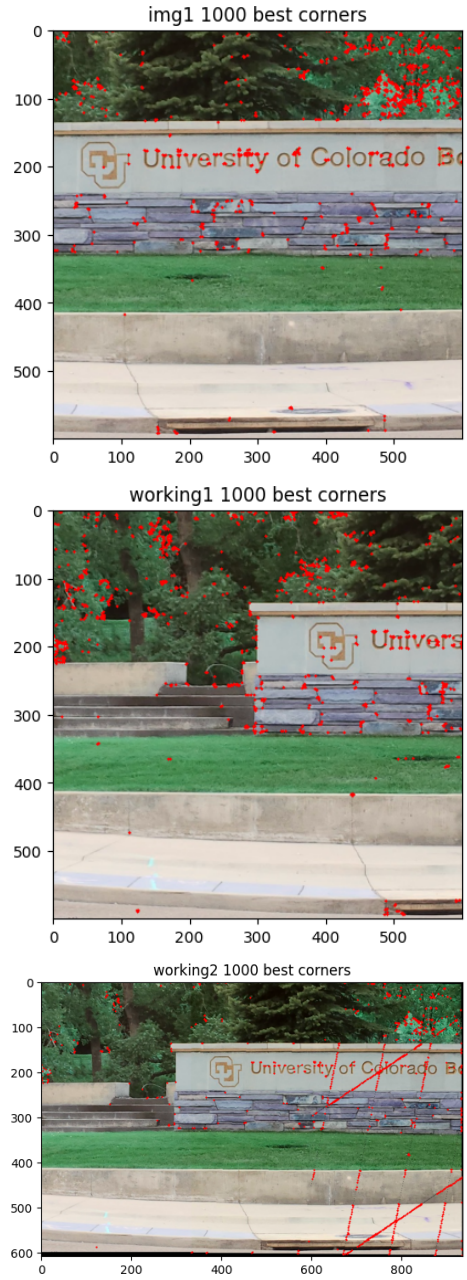

Fig. 13: Output after ANMS on CU Logo Dataset

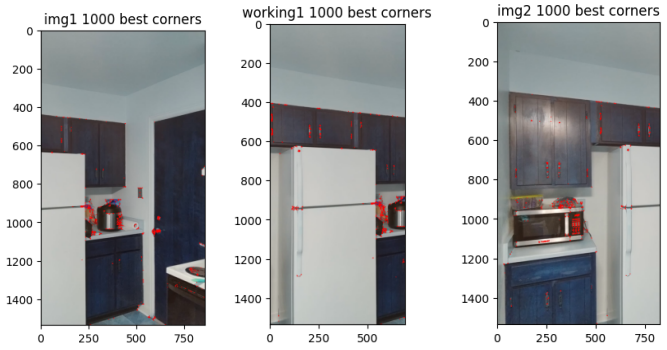

Fig. 14: Output after ANMS on Our Dataset

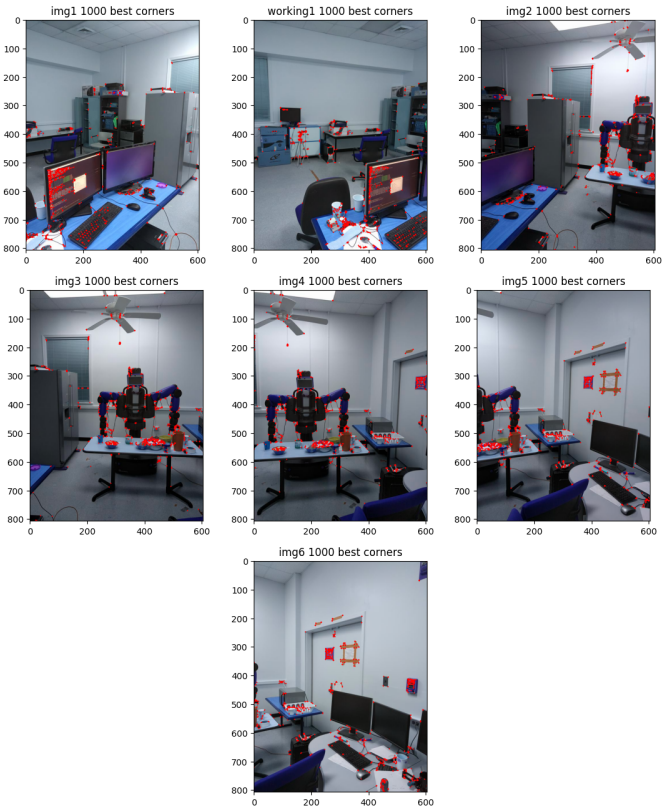

Fig. 15: Output after ANMS on Indoors Dataset

C. Feature Descriptors

Given the set of N_{best} corners produced by the ANMS algorithm we create feature vectors for each corner. These feature vectors are created by taking a patch with height=$Patch_h$ and width=$Patch_w$ centered on the corner. The patch is then blurred with a Gaussian convolution before being reduced in size from $(Patch_h, Patch_w)$ to $(Sample_h, Sample_w)$ using area interpolation. finally the $(Sample_h, Sample_w)$ matrix is flattened into a vector of length $Sample_h \times Sample_w$. This vector is then standardized to have mean 0 and variance 1. You can see example feature vectors output in figures 16 17, 18, 19, 20, 20, 21, 22, 23.

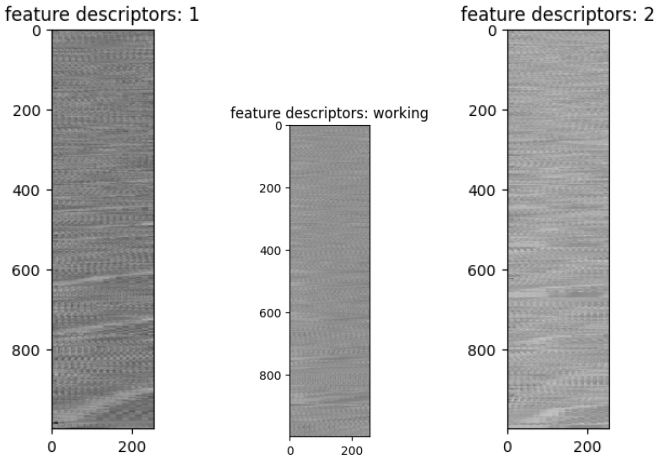

Fig. 16: Feature Descriptors for Victoria Dataset

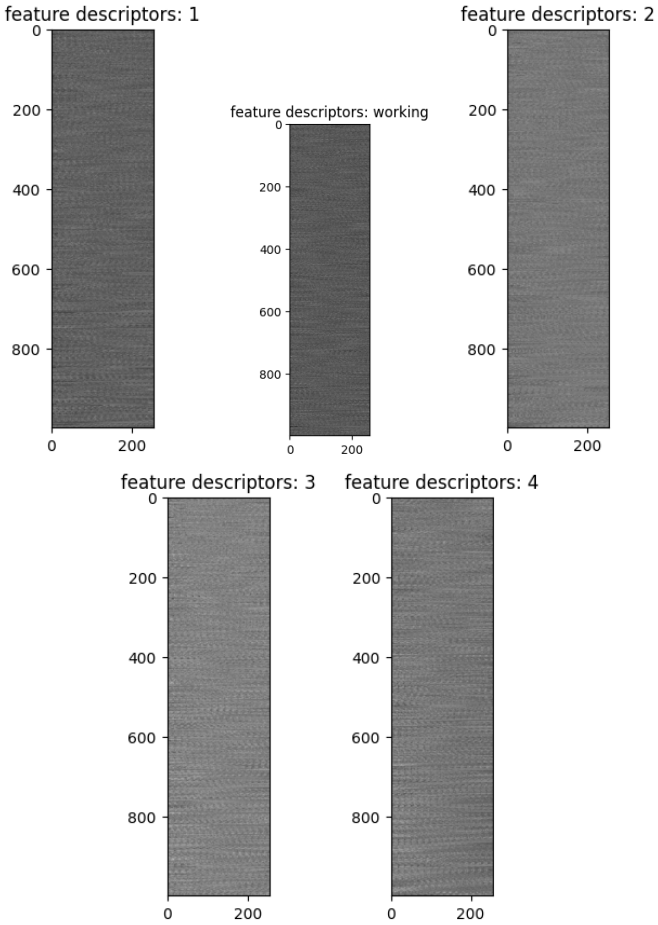

Fig. 17: Feature Descriptors of Flatirons Dataset

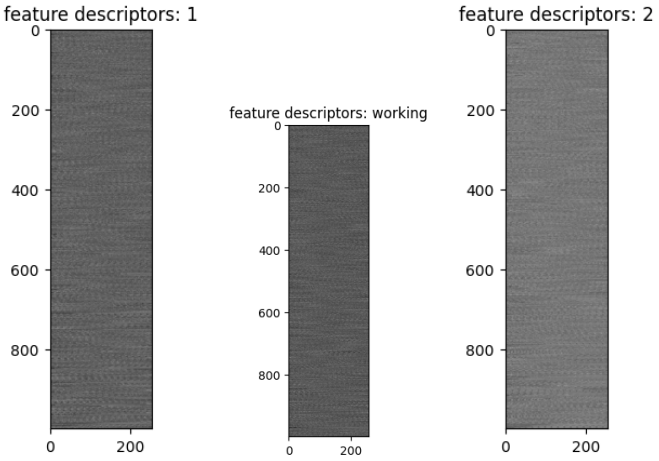

Fig. 18: Feature Descriptors of Flatirons Challenge Dataset

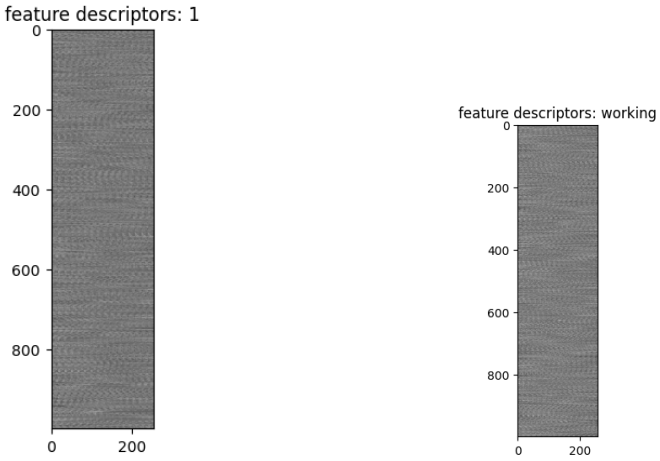

Fig. 20: Feature Descriptors for CU Satellite Dataset

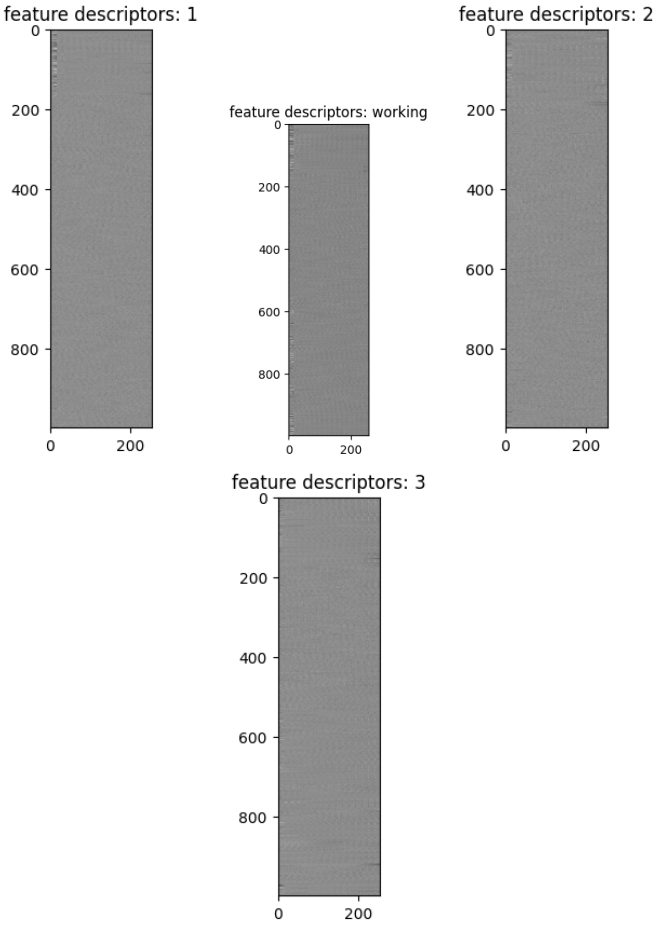

Fig. 19: Feature Descriptors for Checkerboard Dataset

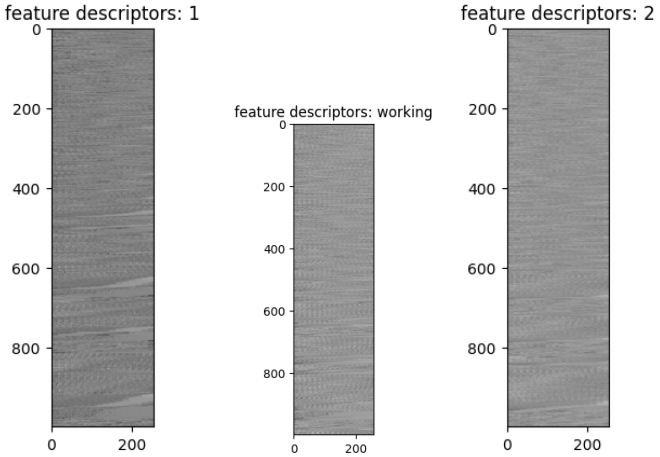

Fig. 21: Feature Descriptors for CU Logo Dataset

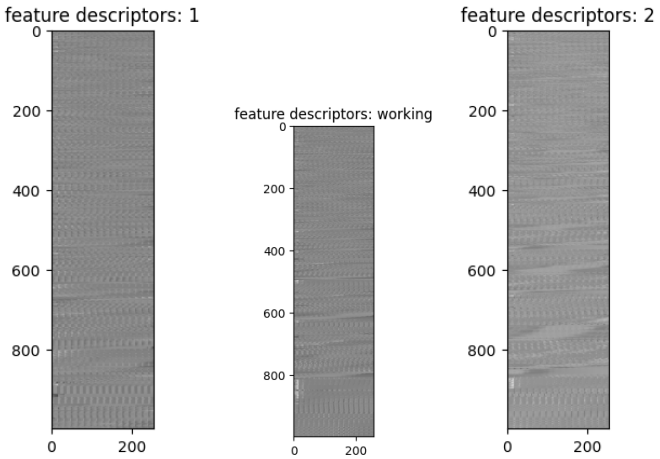

Fig. 22: Feature Descriptors for Our Dataset

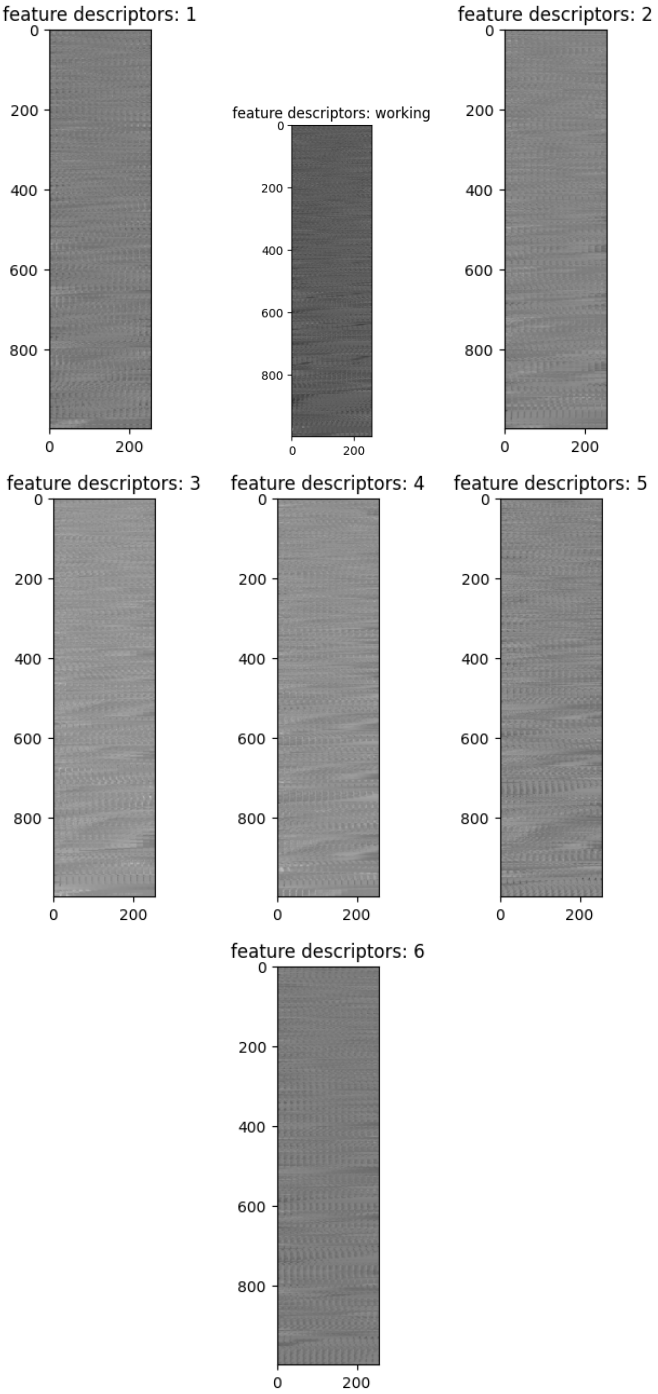

Fig. 23: Feature Descriptors for Indoor Dataset

D. Scale Invariant Feature Transform (SIFT)

SIFT is a powerful and widely used algorithm in Computer Vision for detecting and describing local features in images. In this project, we have used SIFT for Indoor,CU Logo and Checkerboard dataset to get better panorama. We could see the difference that's been discussed in the Result section. Sample output from SIFT from three different sets can be seen in figures 24 25, 26.

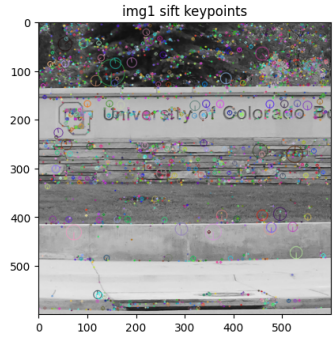

Fig. 24: SIFT for CU Logo Dataset

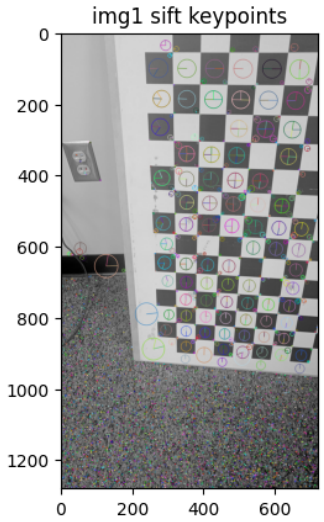

Fig. 25: SIFT for Checkerboard Dataset

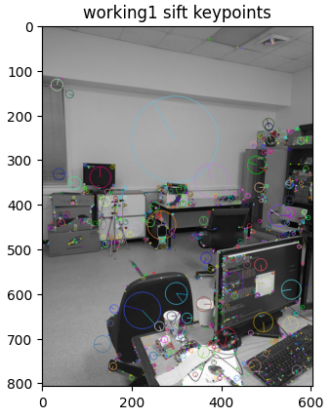

Fig. 26: SIFT for Indoor Dataset

E. Feature Matching

Given a set of feature vectors in image one and a set of feature in image two, our goal is to find features that correspond to one another. This can be achieved with any distance metric, though different distance metrics will yield

different performance. We use the sum squared distance metric $SSD(v_1, v_2) = \sum_{i=1}^{n} (v_1[i] - v_2[i])^2$.

Additionally we impose the restriction that each feature can only match to one other feature, and we introduce a hyper parameters t and require that $\frac{\text{best match metric}}{\text{second best match metric}} < t$. You can see example matchings in figure 27, 28, 29, 30, 31, 32, 33, 34 for $t = 0.9$. This ensures that images without strong matching features are discarded, preventing them from being included in the panorama. Although rejecting these images means we won't be able to form a complete panorama with them, stitching such mismatched images would result in a panorama that is misaligned or out of sync. With more time, we could have developed more refined rejection criteria, such as evaluating the number of matching features. Based on this metric, we could set a threshold to trigger an error, ensuring that images lacking sufficient correspondences are rejected outright, preventing any unsuitable images from being stitched together in the panorama.

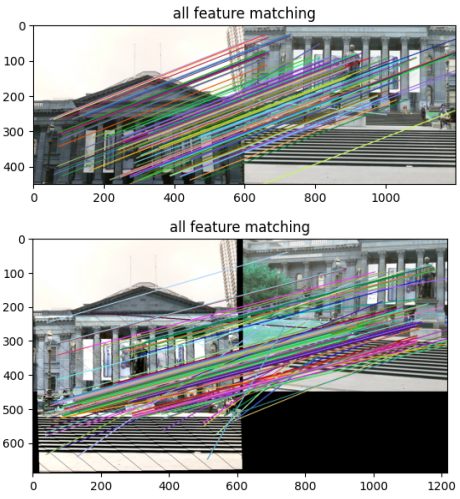

Fig. 27: Feature Matching for Victoria Dataset

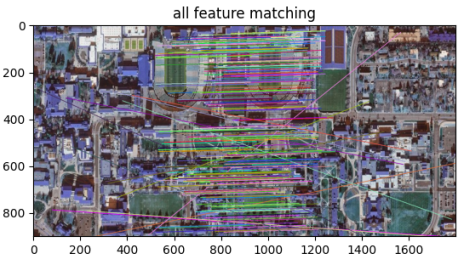

Fig. 28: Feature Matching for CU Satellite Dataset

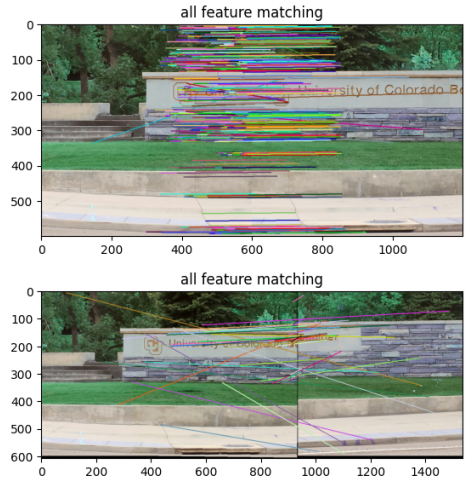

Fig. 29: Feature Matching for CU Logo Dataset

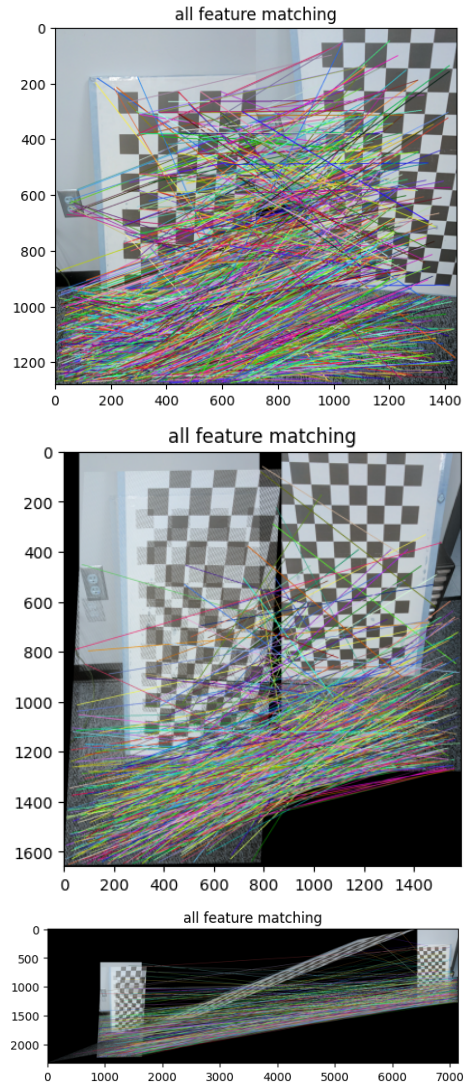

Fig. 30: Feature Matching for Checkerboard Dataset

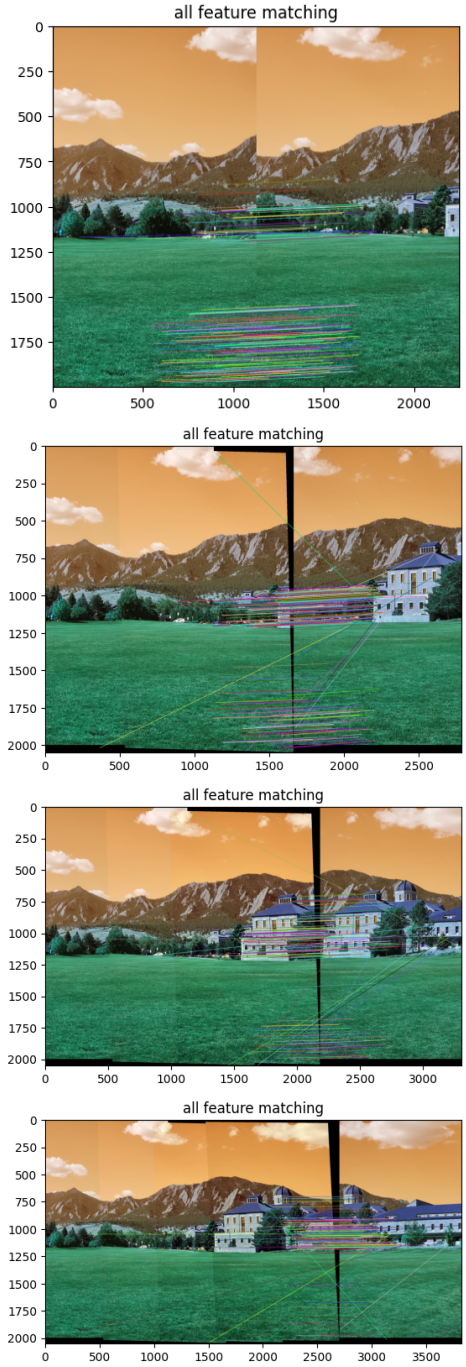

Fig. 31: Feature Matching for Flatirons Dataset

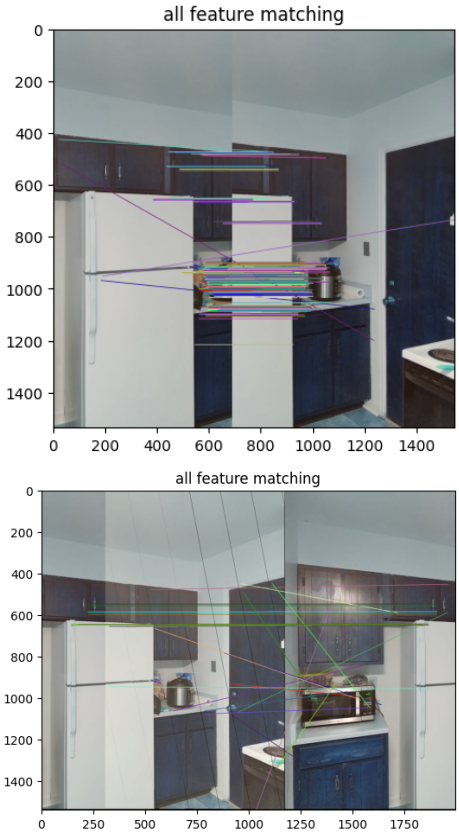

Fig. 32: Feature Matching for Our Dataset

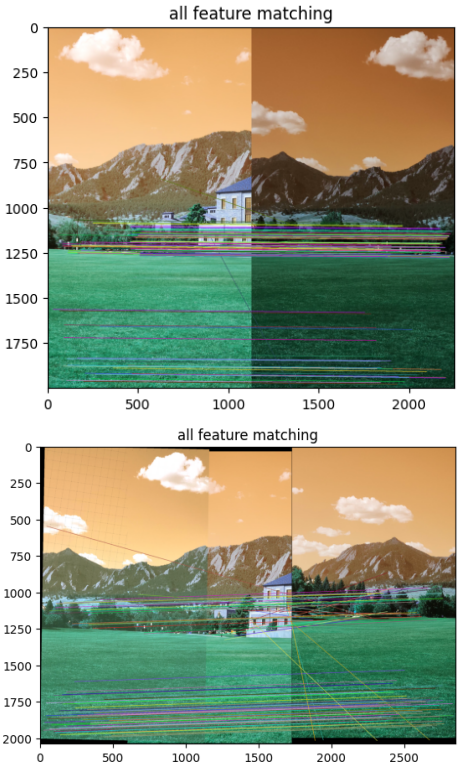

Fig. 33: Feature Matching for Flatiron Challenge Dataset

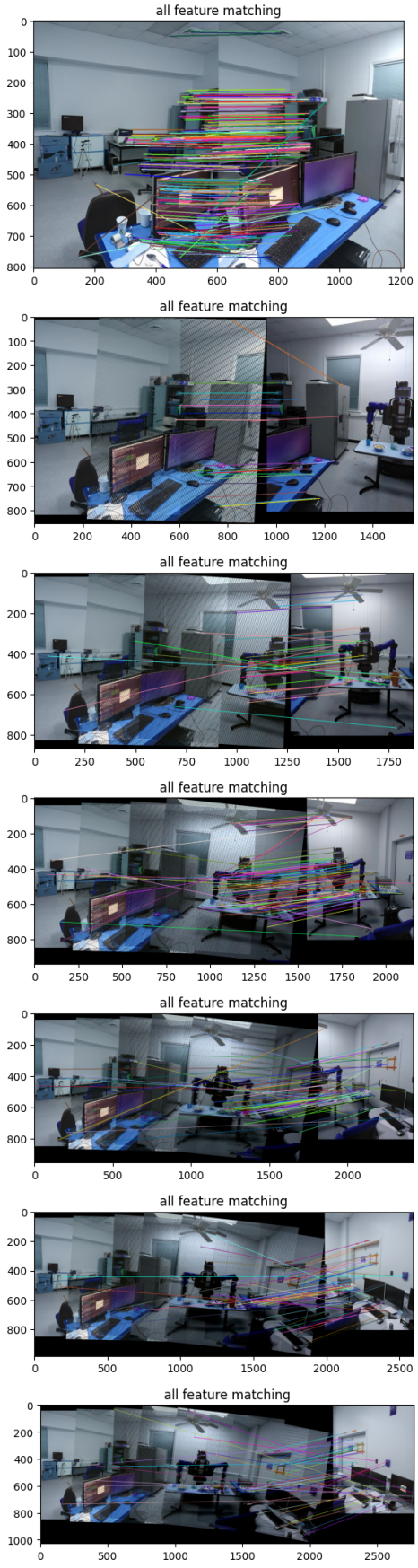

Fig. 34: Feature Matching for Indoor Dataset

F. Refined Feature Matching

From our matched features we have a set of matches between image one and image two given by $X = (y_i, x_i, 1) \mid i < N$ and $Y = (y_i, x_i, 1) \mid i < N$ Where $X[i]$ matches to $Y[i]$ we can estimate the transformation matrix by the closed form solution to linear regression (which minimises the mean squared error in supervised learning) by $H = (X^T X)^{-1} X^T Y$ You can see the homography matrix when all the matches are used in figure 35, However our matches contain many outliers, thus when minimizing the MSE the outliers contribute noise preventing learning an accurate homography matrix. To minimise the effect of outliers we use the RANSAC algorithm in which we randomly sample our matches, learn a homography matrix, then compute how many of our matches follow the proposed homography matrix for N_{iter}. iterations After N_{iter} iterations of RANSAC we use the homography matrix that the most fits the data. You can see the matchings produced by RANSAC in figure 36, 37,38,39,40,39,40,41,42, 43, 44.

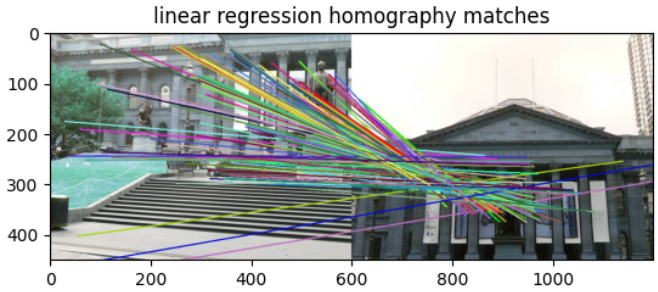

Fig. 35: Homography From all matchings

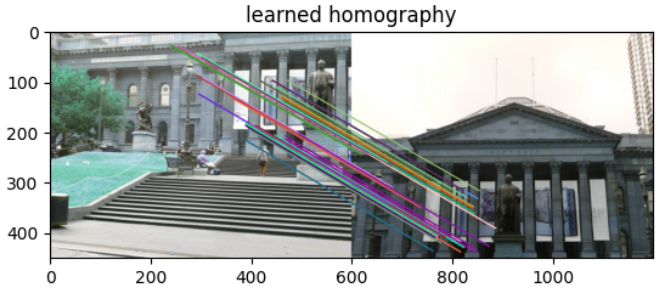

Fig. 36: Homography From RANSAC

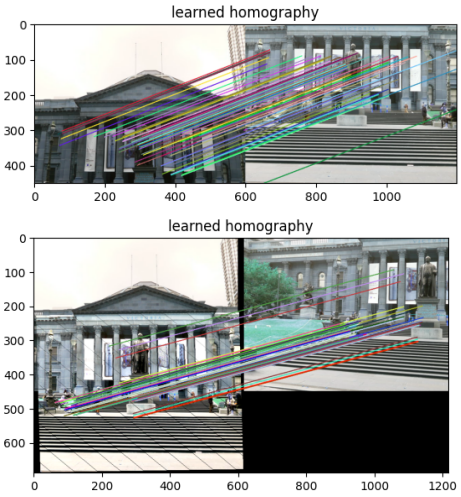

Fig. 37: Feature Matching after RANSAC for Victoria Dataset

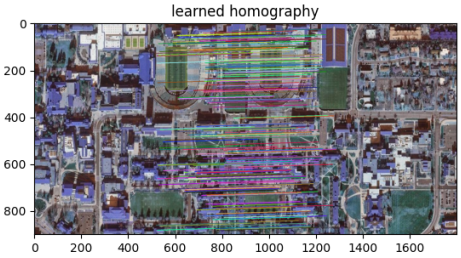

Fig. 38: Feature Matching after RANSAC for CU Satellite Dataset

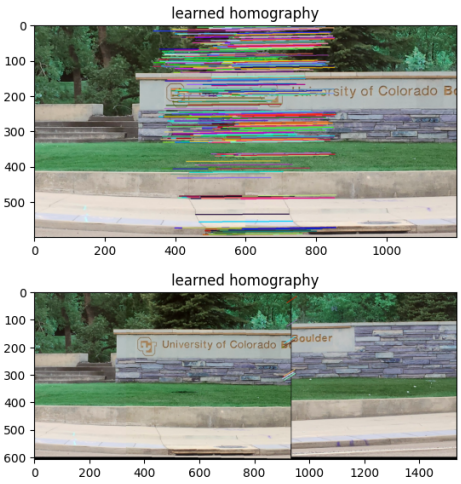

Fig. 39: Feature Matching after RANSAC for CU Logo Dataset

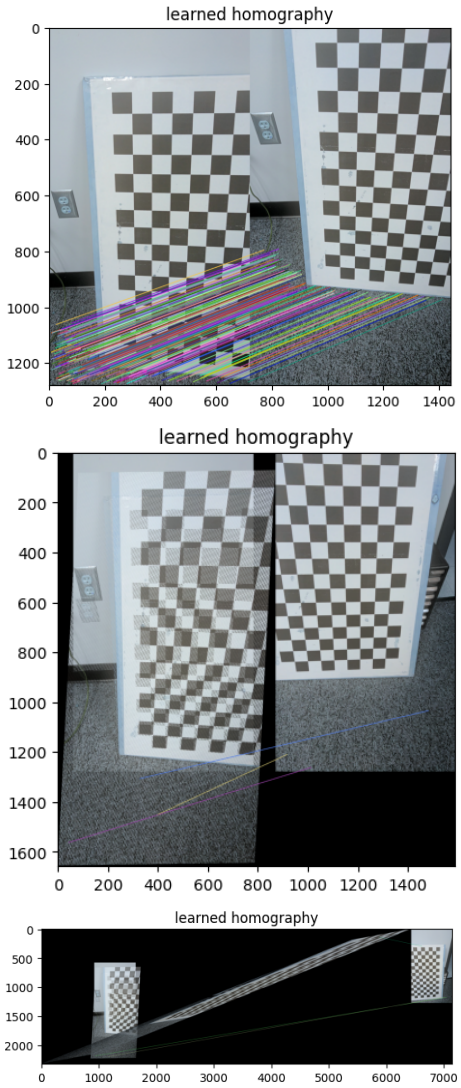

Fig. 40: Feature Matching after RANSAC for Checkerboard Dataset

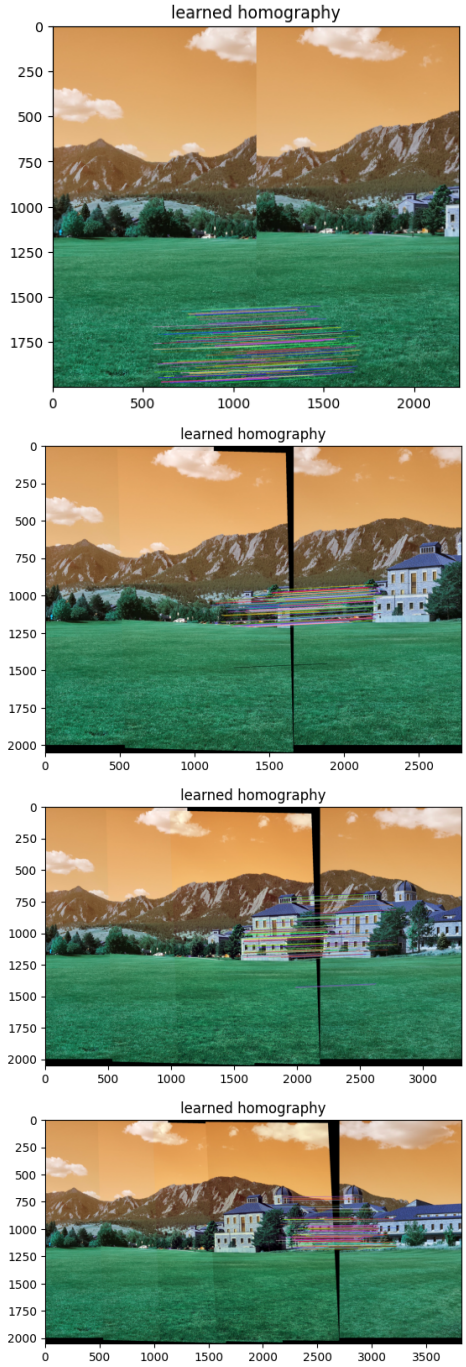

Fig. 41: Feature Matching after RANSAC for Flatirons Dataset

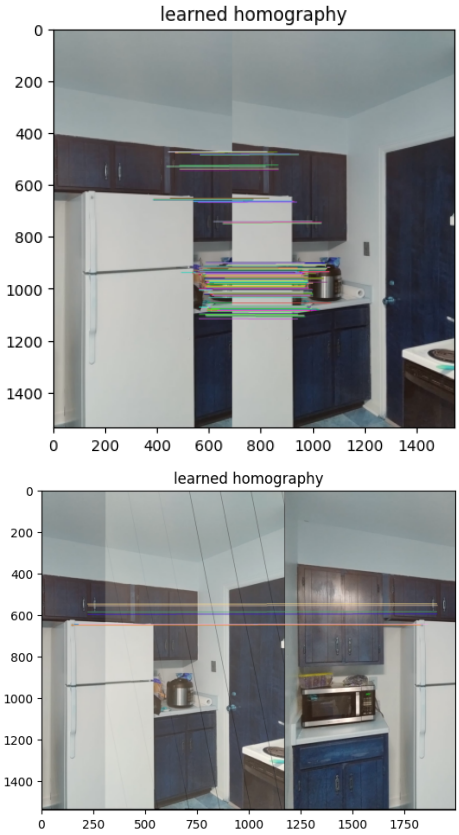

Fig. 42: Feature Matching after RANSAC for Our Dataset

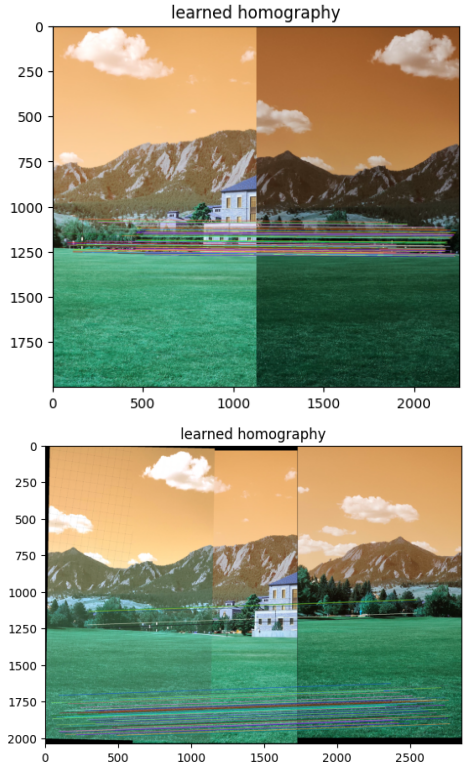

Fig. 43: Feature Matching after RANSAC for Flatiron Challenge Dataset

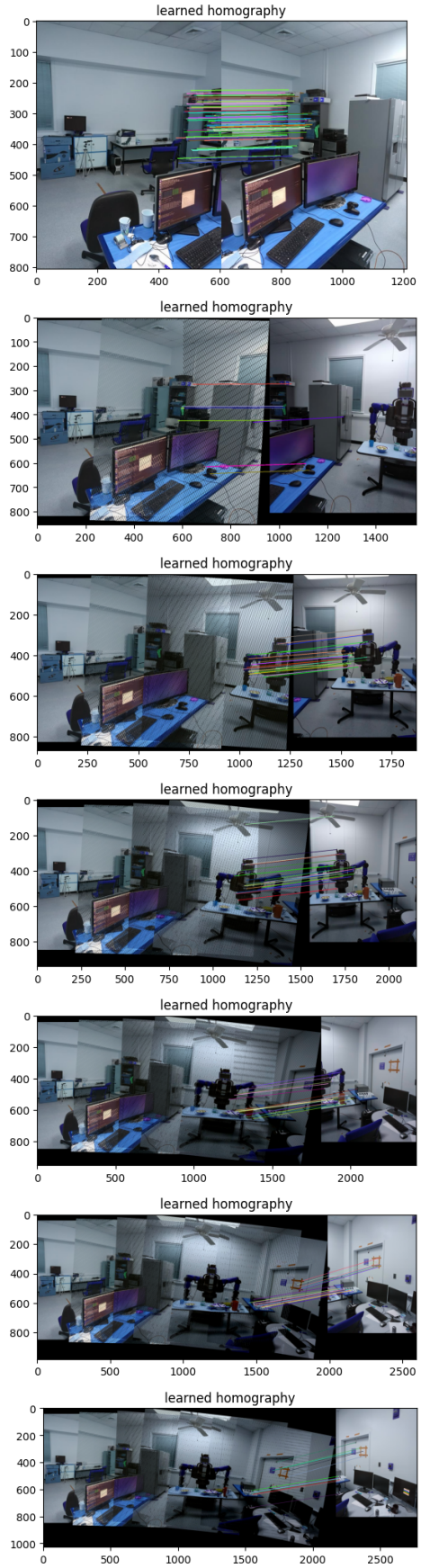

Fig. 44: Feature Matching after RANSAC for Indoor Dataset

G. Warp and blend

To warp and blend we first observe that our homography matrix H transforms points in image one coordinate space to image two coordinate space, because we want to do inverse warping for reasons convered in class we must compute H^{-1} to transform points in image two coordinate space to image one coordinate space.

We then transform the corners of image two into image one coordinate space so we can create a new image that will be able to fit both of our images. With this new image we can simply copy points from image one into this image, and copy points from image two into this image using H^{-1}. We perform blending by making sure the two photos have the same standard deviation and mean for each color channel. You can see the final merging of images in figure 45.

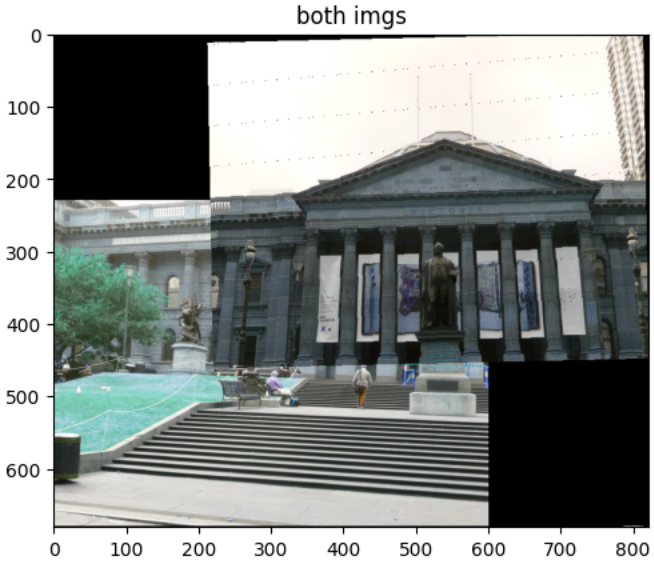

Fig. 45: Homography From RANSAC

H. Numpy acceleration

We are able to significantly accelerate our project by making use of numpy's fast implementation of matrix operations.

The first significant speed up occurs when finding matches between sets of feature vectors. Consider the two feature vector matrices $A, B \mid A, B \in \mathbb{R}^{N \times D}$ where N is the number of feature vectors and D is the number of dimensions in each feature vector. The naive way to find the match for each feature vector would be two nested for loops iterating over A and B, then at each step in the for loop the distance is computed from $A[i]$ to $B[j]$ if this is the best match for the outer loop then we record.

```
best_matches = []
for i in range(N):
    best_match = None
    best_distance = inf
    for j in range(N):
        distance = np.sum((A[i]-B[j])**2)
        if distance < best_distance:
            best_distance = distance
            best_match = B[j]
    best_matches.append(A[i], best_match)
```

However we can speed this up by making use of numpy broadcasting, when subtracting a D dimensional vector from an $N \times D$ matrix numpy automatically subtracts the D dimensional vector from each of the N entries in parallel, effectively computing the difference from a vector to all row vectors of the matrix in parallel.

```
best_matches = []
for i in range(N):
  differences = B - A[i]
  square_differences = differences**2
  ssd = np.sum(square_differences, axis=1)
  match_index = argmin(ssd)
  best_matches.append(A[i], B[match_index])
```

It is possible that with the right formulation we could even remove the first loop using parallelised tensor algebra but this was not a direction we chose to pursue.

The next significant speed up we obtained was using matrix multiplication to quickly transform points, given a set of points $P \in \mathbb{R}^{N \times 3}$ and a transformation matrix $H \in \mathbb{R}^{3 \times 3}$ the naive way to transform the points would be to iterate over P and compute each transform on P like so

```
for p in P:
    transformed_p = Hp
```

here each point is transformed by

$$\begin{bmatrix} y_t \\ x_t \\ 1 \end{bmatrix} = H \begin{bmatrix} y_p \\ x_p \\ 1 \end{bmatrix} = \begin{bmatrix} h_{1,1}(y_p) + h_{1,2}(x_p) + h_{1,3}(1) \\ h_{2,1}(y_p) + h_{2,2}(x_p) + h_{2,3}(1) \\ h_{3,1}(y_p) + h_{3,2}(x_p) + h_{3,3}(1) \end{bmatrix}$$

however we can quickly transform each point in parrallel by making use of the original point matrix P via PH^T

$$P = \begin{bmatrix} p_{1,y} & p_{1,x} & 1 \\ p_{2,y} & p_{2,x} & 1 \\ \vdots & \vdots & \vdots \\ p_{N,y} & p_{N,x} & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ h_{3,1} & h_{3,2} & h_{3,3} \end{bmatrix}$$

then

$$PH^T = \begin{bmatrix} p_{1,y}h_{1,1} + p_{1,x}h_{1,2} + h_{1,3} & p_{1,y}h_{2,1} + p_{1,x}h_{2,2} + h_{2,3} & p_{1,y}h_{3,1} + p_{1,x}h_{3,2} + h_{3,3} \\ p_{2,y}h_{1,1} + p_{2,x}h_{1,2} + h_{1,3} & p_{2,y}h_{2,1} + p_{2,x}h_{2,2} + h_{2,3} & p_{2,y}h_{3,1} + p_{2,x}h_{3,2} + h_{3,3} \\ \vdots & \vdots & \vdots \\ p_{N,y}h_{1,1} + p_{N,x}h_{1,2} + h_{1,3} & p_{N,y}h_{2,1} + p_{N,x}h_{2,2} + h_{2,3} & p_{N,y}h_{3,1} + p_{N,x}h_{3,2} + h_{3,3} \end{bmatrix}$$

which equivalently transforms points in parallel. We use this in conjunction with numpy advanced indexing to quickly copy pixels from one image to another using the homography matrix.

II. Results

A. Victoria Library

Our approach works very well on the Victoria library dataset. There are some bizarre lines that we believe to be caused by the float to int conversion, these lines are discussed in the challenges section. You can see our approaches output in figure 46.

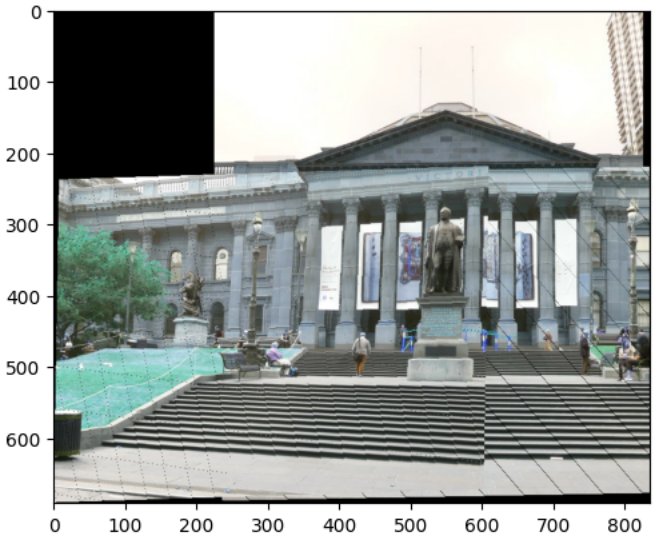

Fig. 46: Progams output on victoria library dataset

B. Flatirons

Our approach works very well on the Flatirons dataset. You can see our approaches output in figure 47.

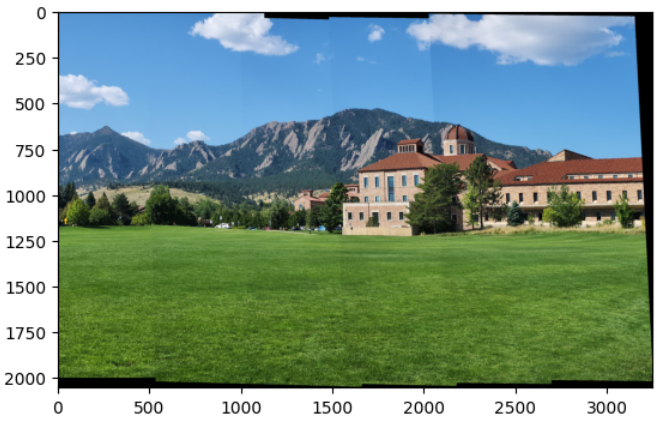

Fig. 47: programs output on flatirons dataset

C. CUBoulderSatView

Our approach works very well on the CUBoulderSatView dataset. You can see our approaches output in figure 48.

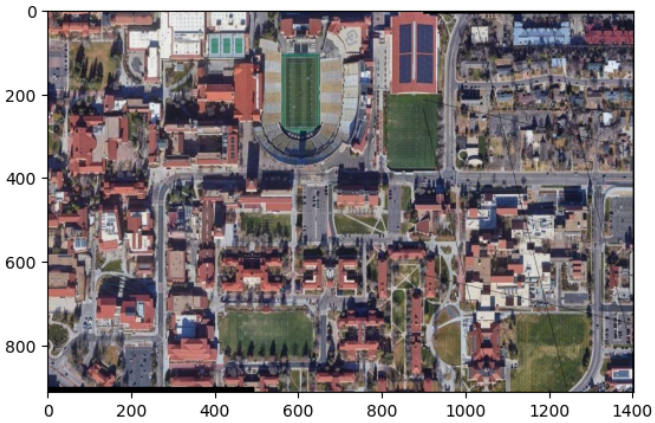

Fig. 48: programs output on satellite view dataset

D. FlatironsChallenge

Our approach works alright on the CUBoulderSatView dataset. You can see our approaches output in figure 49. Note that our images are properly alligned but the coloring is all wrong, this is due to improper blending between the images.

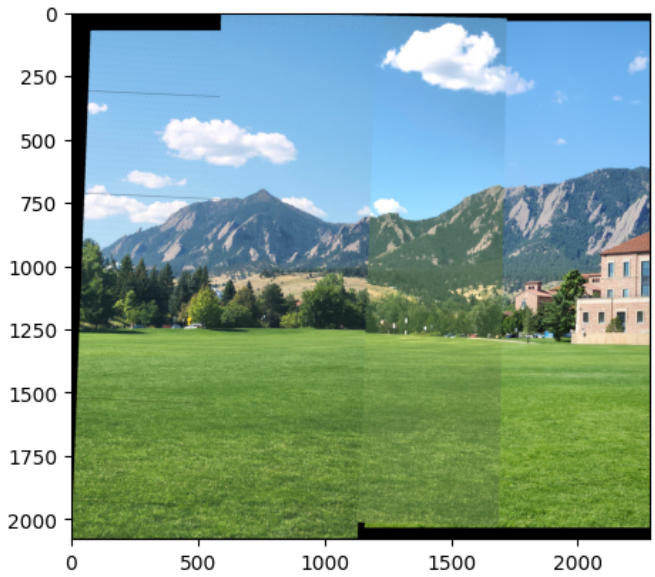

Fig. 49: programs output on flat iron challenge dataset

E. CU Logo

The CU logo dataset is the first dataset that our approach does not perform well on because the sign is relatively featureless our model ends up aligning letters that are not the same. You can see the output of our approach on this dataset in figure 50.

However if we use SIFT feature detection and descriptors instead of our feature detection and descriptors we get much better performance, the output of our approach with sift features and descriptors can be seen in figure 51.

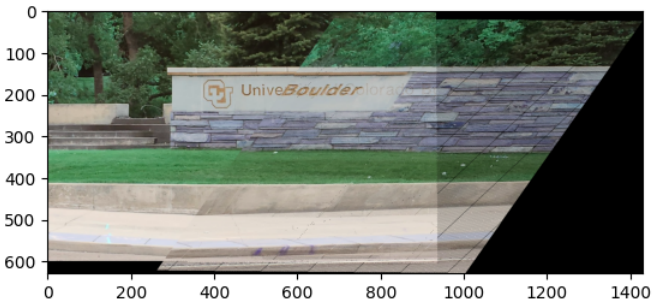

Fig. 50: Our features program output on CULogo dataset

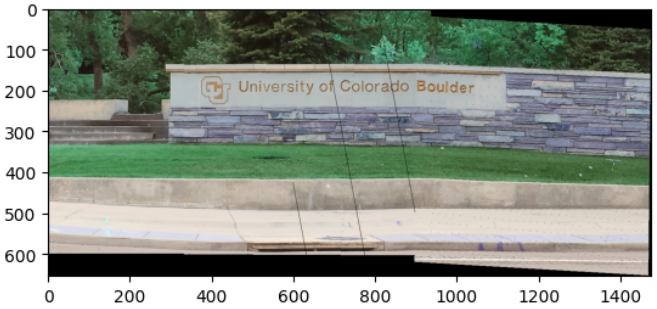

Fig. 51: Sift features program output on CULogog dataset

F. Checkerboard

Using our feature detection and descriptions yields poor results on the checkerboard dataset. These poor results can be seen in figure 52. We can get better matching by using SIFT

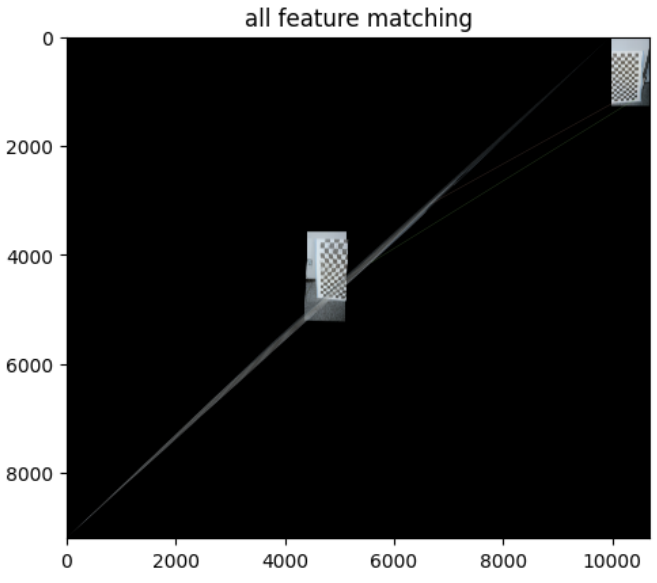

Fig. 52: Our features program output on checkerboard dataset

feature detection and description, but it still does not perform perfectly on the checkerboard dataset. The results using SIFT detection and description can be seen in figure 53

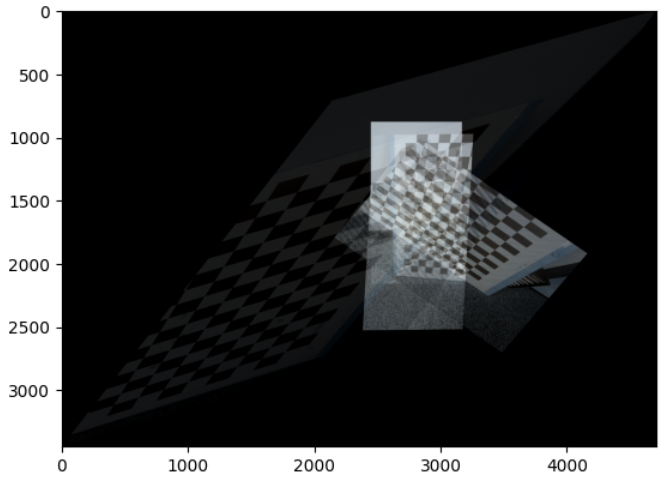

Fig. 53: Sift features program output on checkerboard dataset

G. Indoor

Our approach again under performs on the indoor dataset, this is due to improper feature matchings as well as the planar assumption made by our warp and blend method, using our feature descriptors and detection the result can be seen in figure 54.

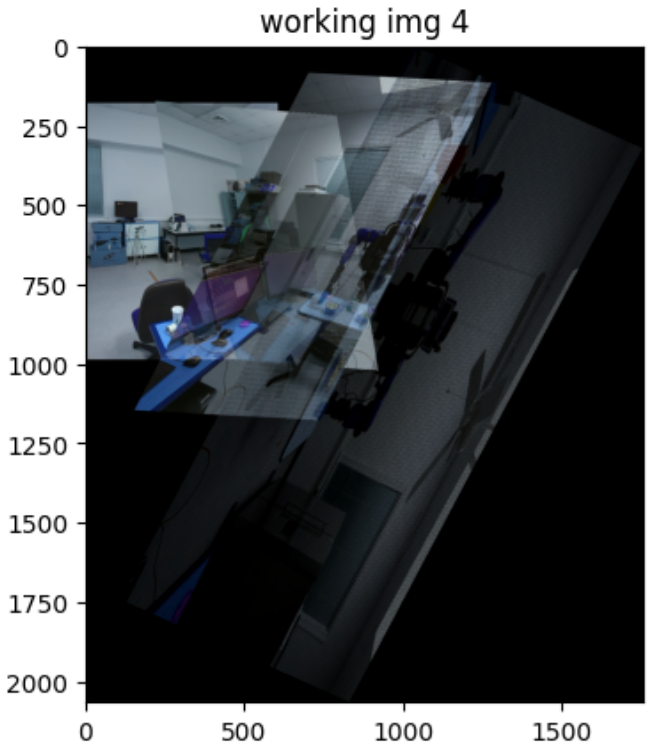

Fig. 54: Our features program output on indoor dataset

Using sift feature detection and descriptors we can get a much better result while still keeping our planar projection

assumption. the results of the using the sift features can be found in figure 55.

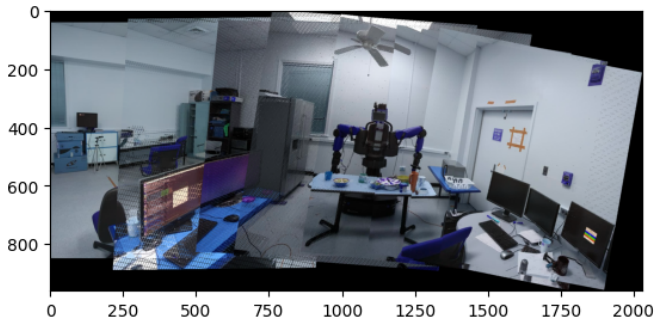

Fig. 55: Sift features program output on indoor dataset

H. Our Own Dataset

We collected a dataset using our phone camera by capturing a single photo and then cropping it into three consecutive images to convert it into a dataset. The Figure 56 represents 3 images of our dataset. In Figure 57, you can observe the success of our approach, which utilized our custom feature descriptor along with the Warp and Blend method.

Fig. 56: Our Dataset Images

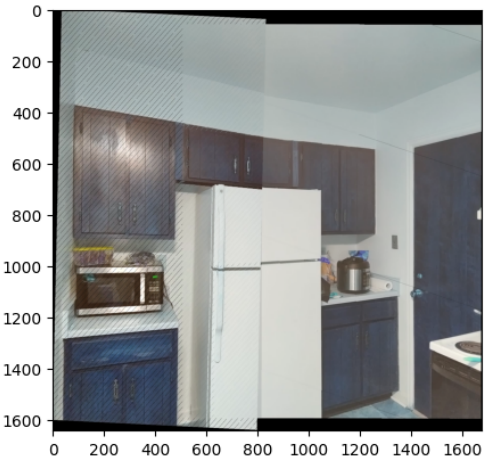

Fig. 57: The output of our dataset after using our feature descriptor method

III. Discussion

A. Challenges

This project featured numerous challenges but when applying to new datasets there are 5 challenges you must consider. The first challenge is brought about by the planar assumption, we did not implement any more advanced kinds of projection so our approach will do its best within the constraints of the planar assumption. The second challenge is hyper parameter tuning, parameters like the ratio threshold for matches, feature patch size, blur kernel, sub sample size, RANSAC inlier threshold, and RANSAC n_iterations can all be tuned to yield optimal performance on a specific dataset. We theorise the feature vector hyper parameters are why our approach does not work on the checkerboard dataset since without sufficient feature vectors any inside corner of the checkerboard can match to any other inside corner of the checkerboard. Blending we have very naive color blending which does not work very well on the flatirons challenge dataset. There is also a bizarre behavior we were not able to diagnose where lines appear on the transformed image, we believe this due to the floating point to integer conversion but were not able to resolve the bug.

B. Future Improvements

Looking ahead, several future improvements could significantly enhance our image stitching and panorama creation process. First, the implementation of cylindrical projection techniques could help better manage perspective distortions, providing a more accurate representation of the scene and improving the overall visual quality of the final panorama. Additionally, we could explore the integration of alternative feature detection algorithms such as SURF (Speeded Up Robust Features) and FAST (Features from Accelerated Segment Test). These methods might offer different advantages in terms of speed and robustness, which could lead to improved feature matching and alignment. Moreover, enhancing the blending process remains a priority. By developing a transformation matrix $C \in \mathbb{R}^{3\times3}$,we could learn to adjust the colors in the

first image to better match those in the subsequent images. This would not only improve the visual coherence between the overlapping regions but also create a more seamless transition across the entire panorama. Overall, these advancements would contribute to more effective and visually appealing image stitching in future projects.