LAB 5

Photomosaicing

Shakeel Ahamed | NUID 002199480 | Team 14 | Section 1

Objective:

The aim of this lab is to calibrate our camera and stitch images using Harris features to create a panorama, we initially try this on the mural on the Latino Students Center building and then use the same algorithm on a cinderblock wall and another graffiti from the university (ideally from around Ruggles).

Camera:

We use an iPhone 13 Pro Max for the photos that need to be stitched.

Method/Data:

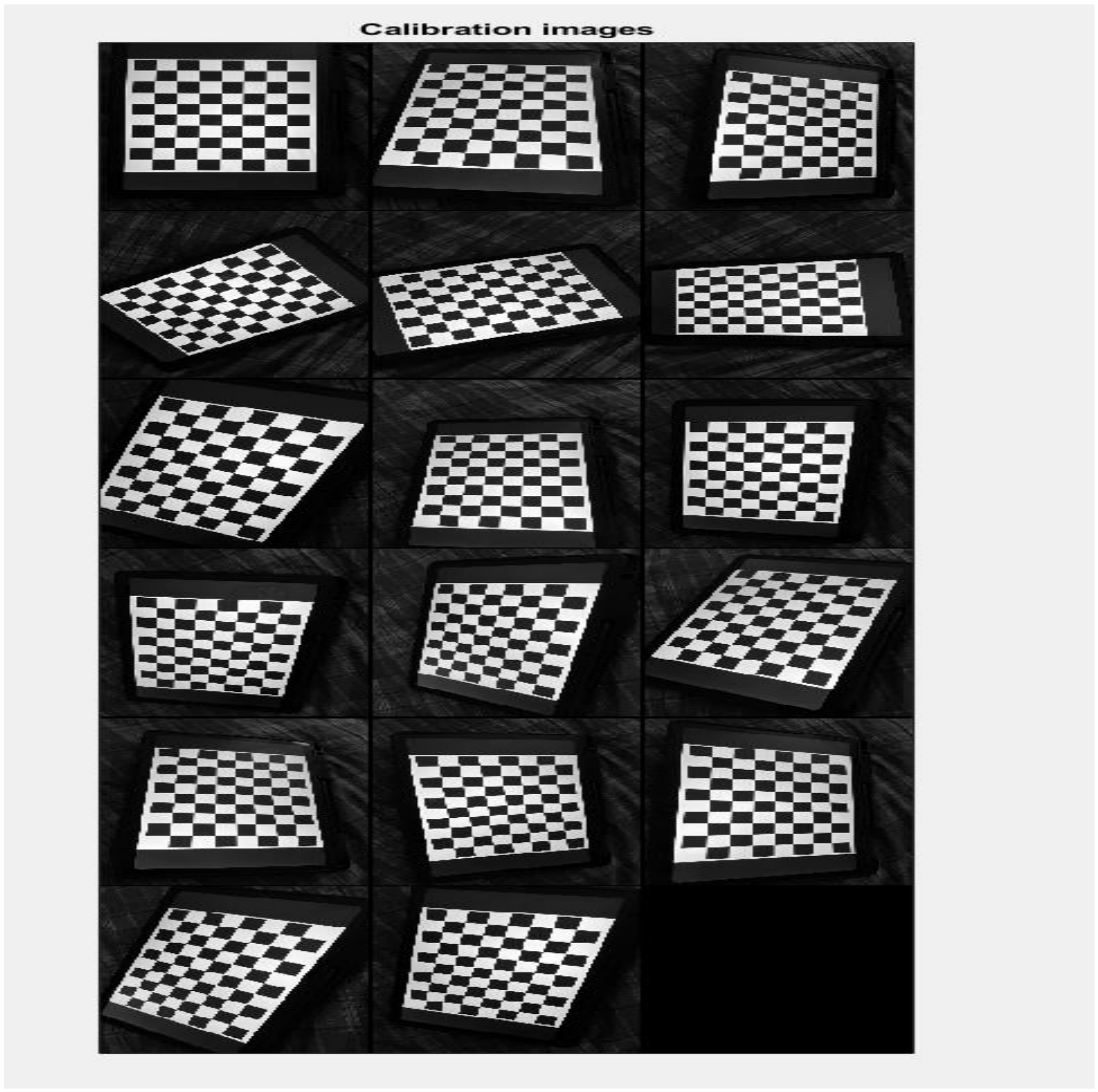
The steps taken to get our readings:

- **Camera Calibration:**
  A calibration pattern was displayed on a tablet and photos of it were taken from various angles and calibrated as instructed in the Caltech Camera Calibration Toolbox.
  About 17 images were used for the calibration(excluding the suppressed images)
- **Forsythe Street, Mural on Latino Student Center building:**
  5-7 photos of the mural were taken from left to right while there were no trespassers to make sure it does not mess with the Harris features.
- **Cinder Block Wall pictures:**
  Cinder Block Wall pictures were obtained from group 2.
- **Graffiti on boundary wall of Ruggles station (campus facing):**
  11 15% overlap photos of the graffiti were taken on the same mobile camera.

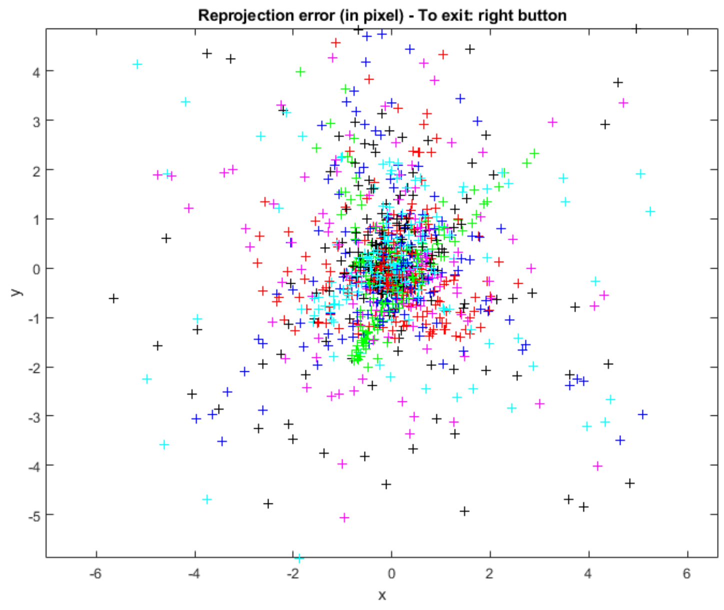
Part 1: Camera Calibration

The first step is to stick a calibration pattern to a flat surface, we do this by a much eco-friendlier method of displaying the image in a flat tablet and taking photos of it from different angles.

The following are the calibration images used (17):



Initial error:



```
Pixel error:          err = [ 1.30119   1.39359] (all active images)
```

All the images were calibrated with default values without any manual distortion entry to extract grid corners since the guesses were very accurate visually. Unfortunately, due to this we have quite an unsatisfactory error after using recompute corners in the toolbox.

To remedy this, we try to reproject errors into images and match the colors to the Reprojection error graph above and remove the ones with most outliers to get better calibration results.

Like so:

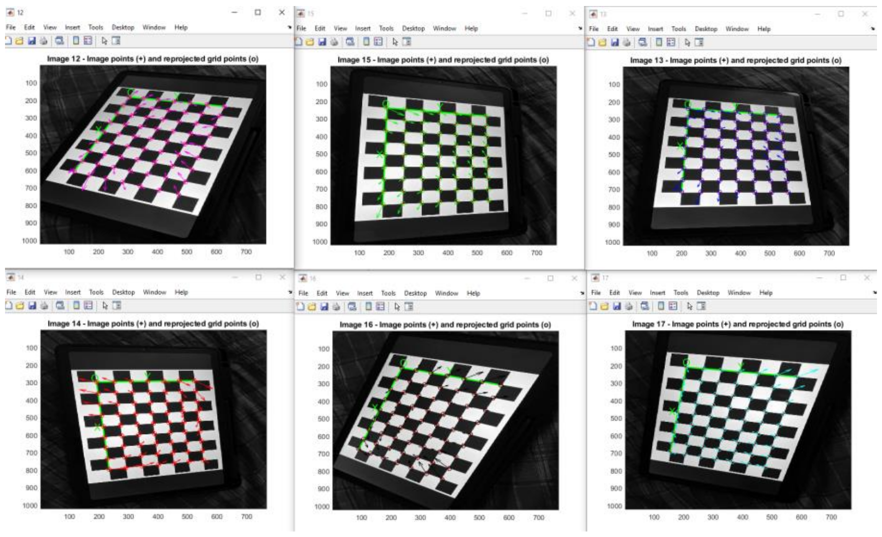
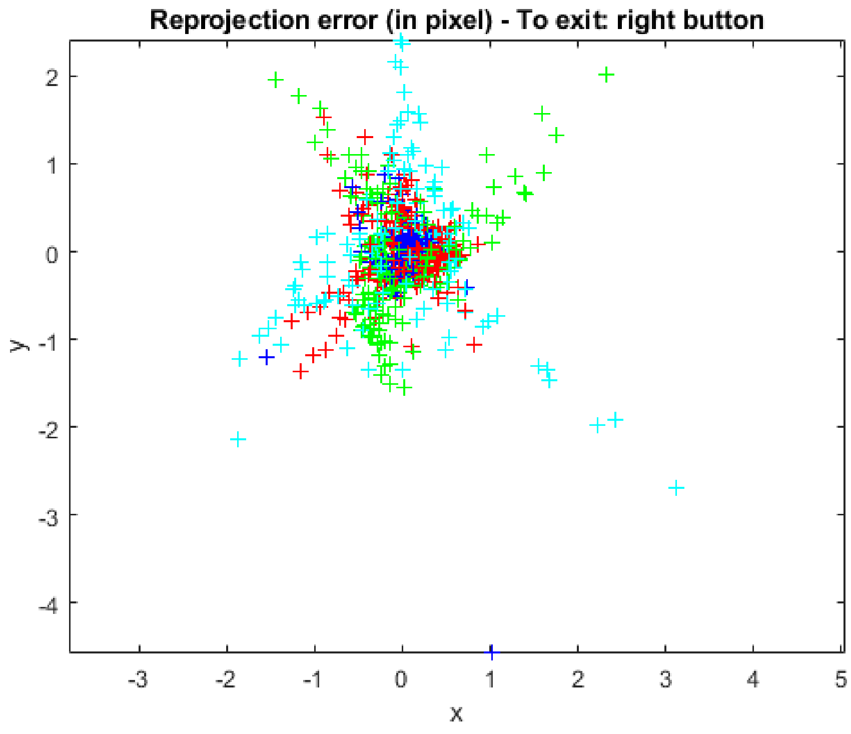


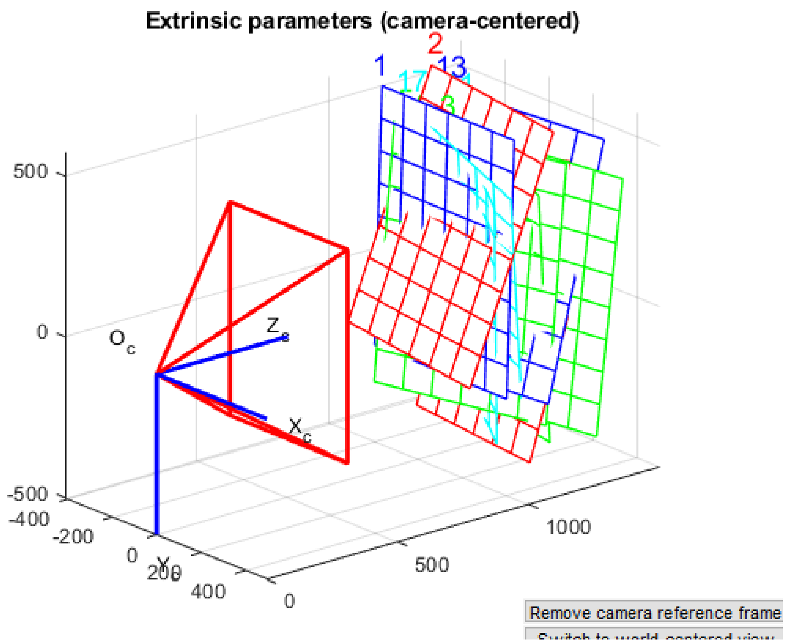
Image 12 - Image points (+) and reprojected grid points (o)
Image 15 - Image points (+) and reprojected grid points (o)
Image 13 - Image points (+) and reprojected grid points (o)
Image 14 - Image points (+) and reprojected grid points (o)
Image 16 - Image points (+) and reprojected grid points (o)
Image 17 - Image points (+) and reprojected grid points (o)

We compare the analyze error graph with reprojection error with color code and remove the following images to get satisfactory errors:

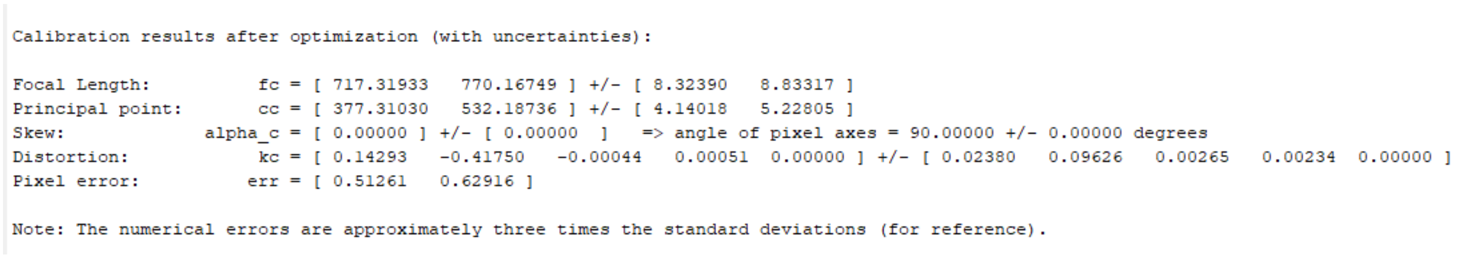
Image4, Image5, Image7, Image16 contained most of the outliers and were removed and the following results were obtained.



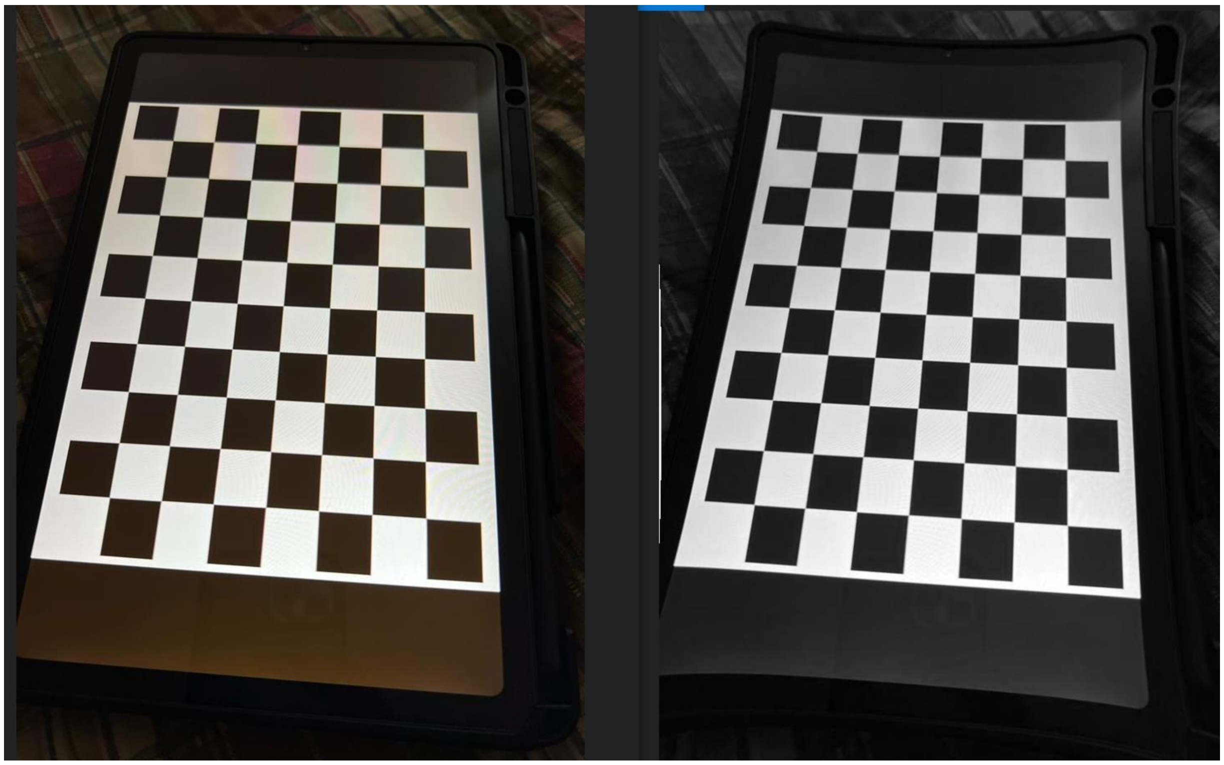
```
Pixel error:          err = [ 0.51261   0.62916] (all active images)
```



With the new adjustments, the following are the calibration parameters:



```
Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 717.31933   770.16749 ] +/- [ 8.32390   8.83317 ]
Principal point:       cc = [ 377.31030   532.18736 ] +/- [ 4.14018   5.22805 ]
Skew:             alpha_c = [ 0.00000 ] +/- [ 0.00000  ]   => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:            kc = [ 0.14293   -0.41750   -0.00044   0.00051  0.00000 ] +/- [ 0.02380   0.09626   0.00265   0.00234  0.00000 ]
Pixel error:          err = [ 0.51261   0.62916 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```



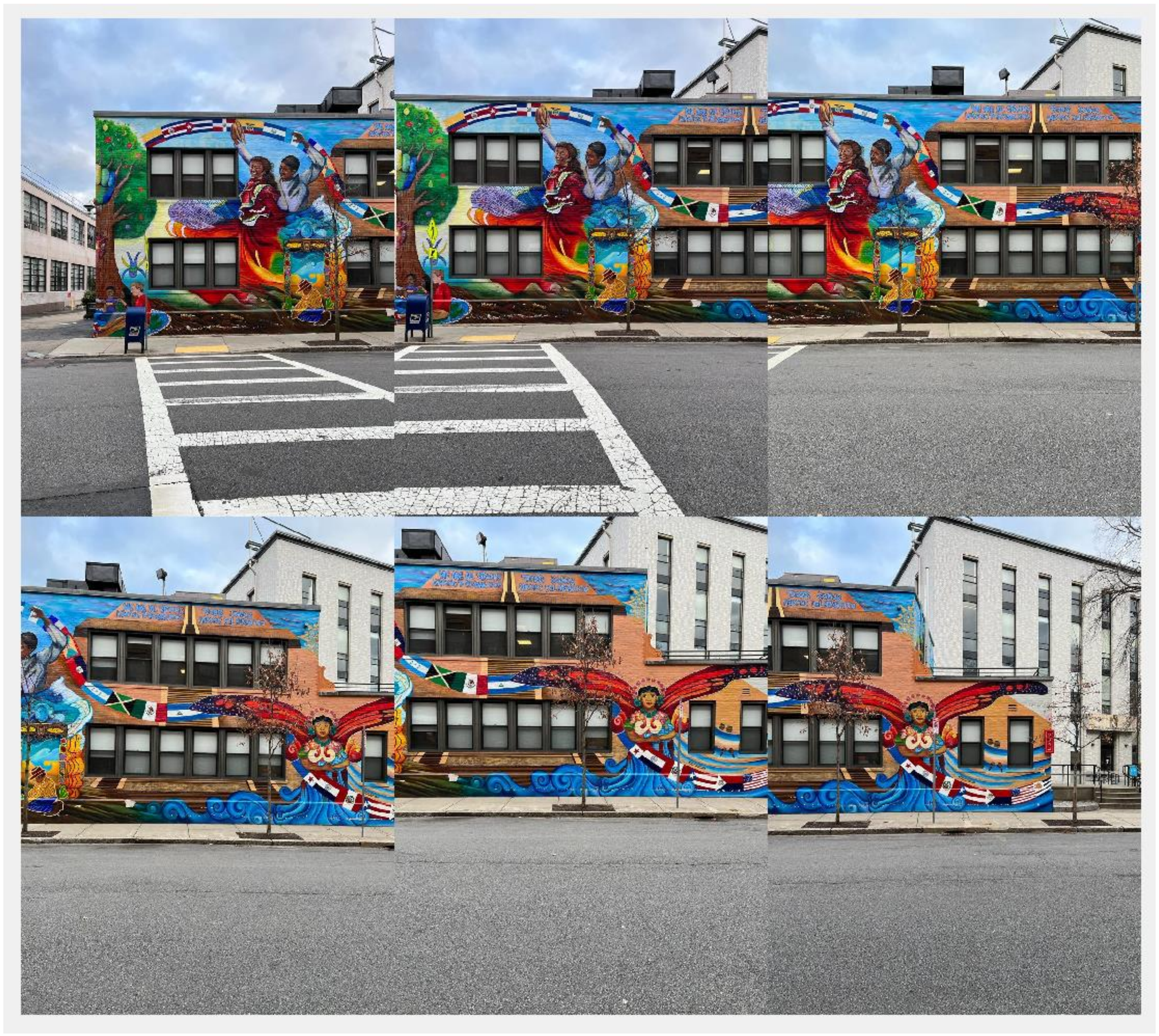
(Right side is post-calibration)

The post-calibration photo is obtained by using the Undistort image function from Camera Calibration Toolbox, but it seems like our original photo was already undistorted and we just ended up adding more distortion by trying to undistort the image which are indicated by the weird looking edges, this could be due to inbuilt functionality in the mobile camera itself and their image-processing algorithms already creating calibrated images, this means we’ll end up getting better results by using original photos rather than the calibrated version using the toolkit to create the panorama in the later parts of this lab.

Part 2: Panorama of mural at the Latino Student Center building

Photo we’ll be using for the panorama stitching:

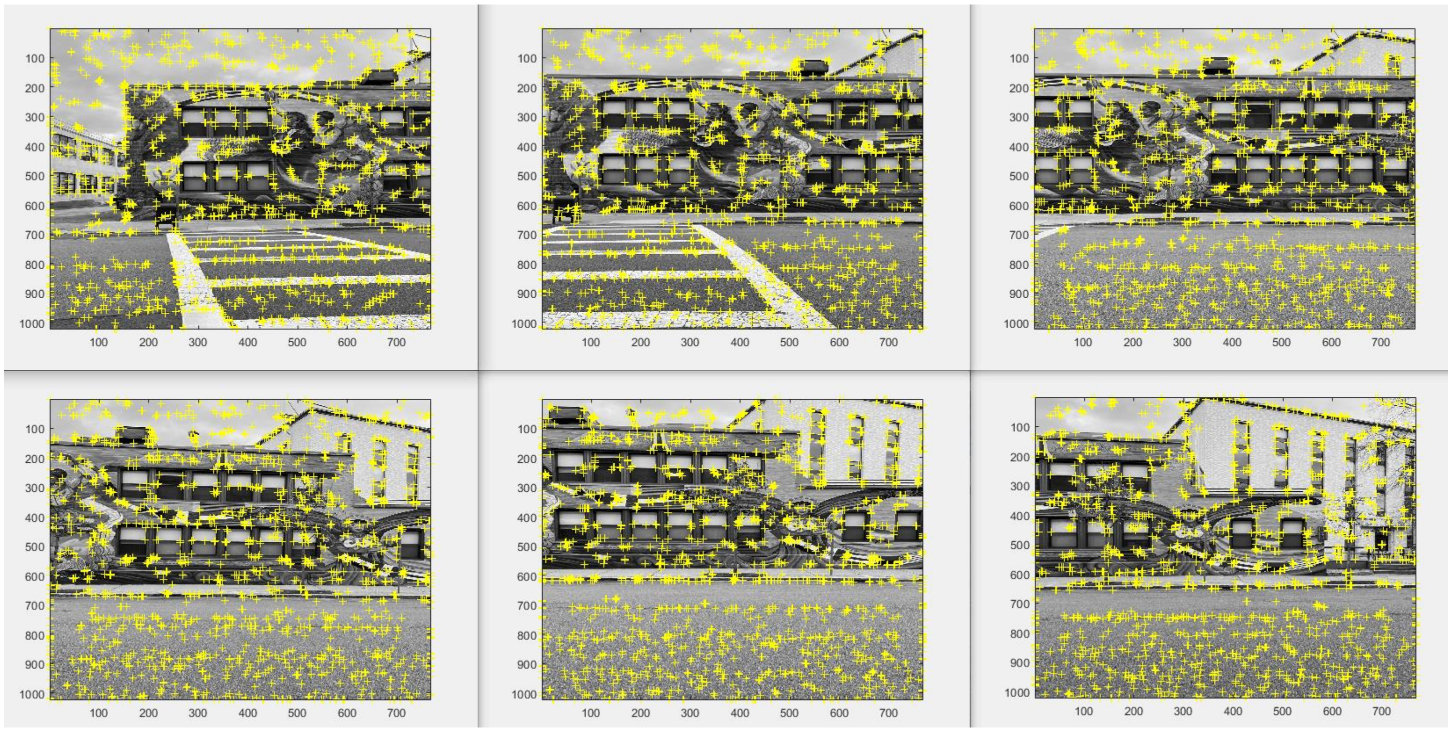
The following photos of the mural were used:



We move left to right to get these photos and try to hold the same orientation, unfortunately they weren’t in the perfectly same orientation which may lead to issues while making the panorama.

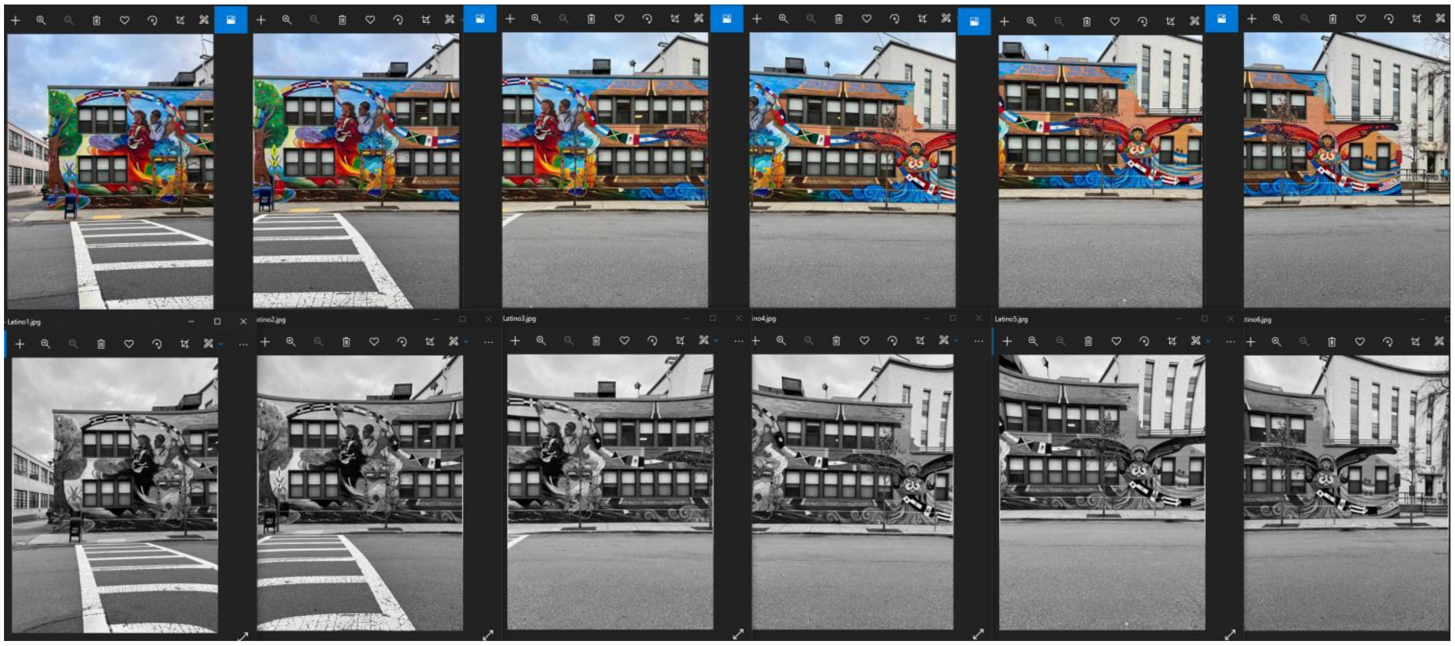
Harris detection feature:

Harris corner detection is a corner detection operator that is commonly used in CV algorithms to extract corners and infer features of an image. Harris' corner detector takes the differential of the corner score into account with reference to direction directly, instead of using shifting patches for every 45-degree angles and has been proved to be more accurate in distinguishing between edges and corners, we use harris.m and convolve2.m function codes given to us to implement this; the following are the harris features:



It is ideal to spread out the features to get more accurate stitching, we tweak around the harris parameter maximum number of interest points and tile, we end up select 1500 as maximum number of interest points and [15 15] as the tile.

Distorted and Undistorted mural images:

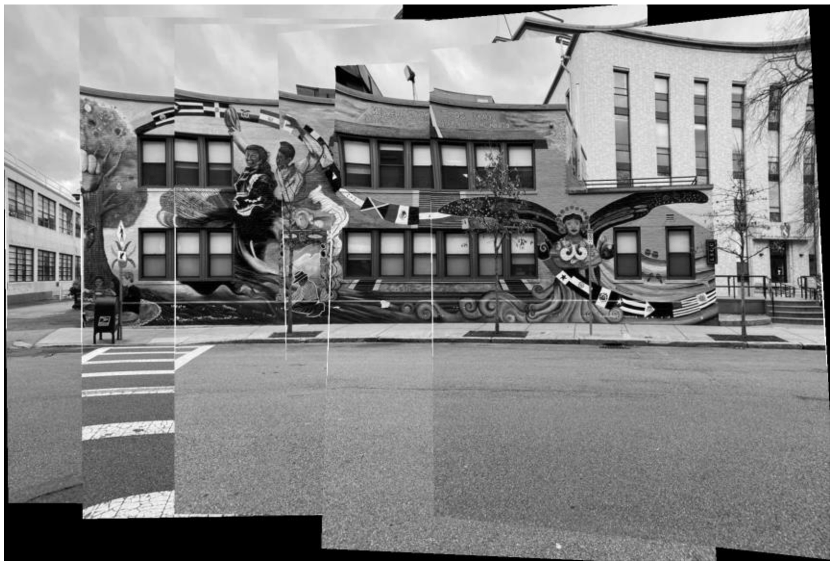


We use the Undistort image function from the calibration toolbox to get the undistorted versions one by one and save them in a separate folder so we can run the stitching process for both the versions to see which comes out better and use accordingly.

Stitching the images:

For this, we follow the mathworks template with one difference, which is using Harris features instead of the SURF features given in the template for feature detection.

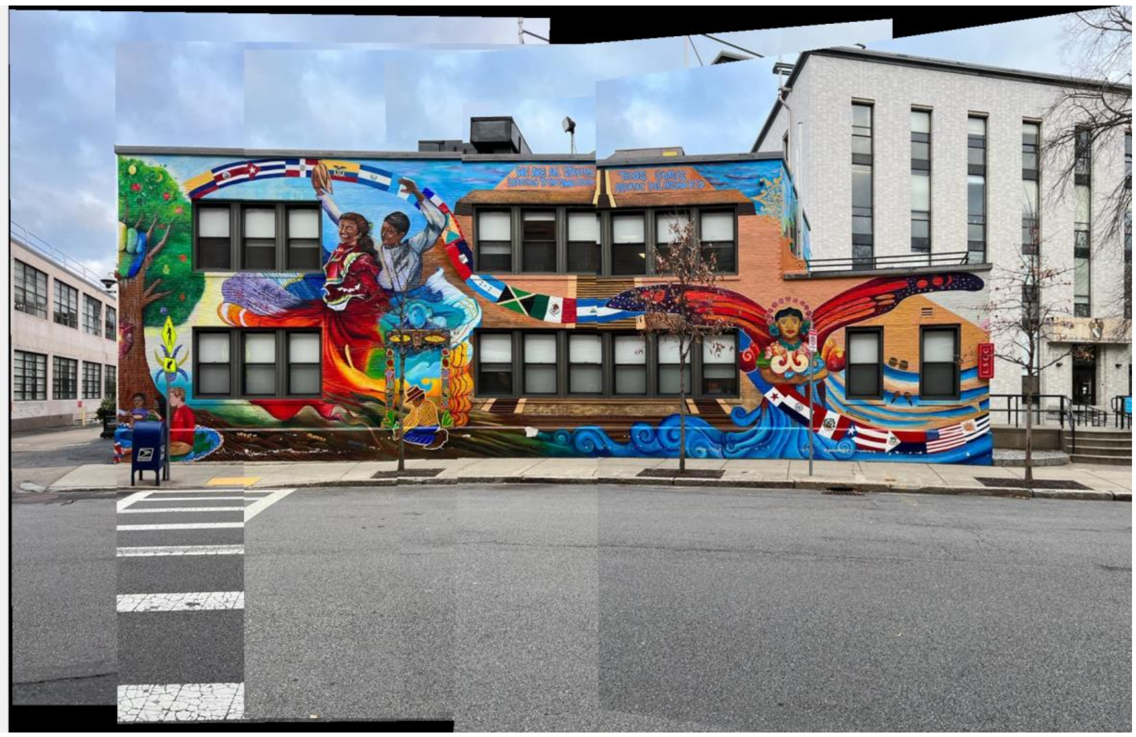
Stitching the calibrated/undistorted images:



The undistorted image panorama creates curvy frame edges which could be due to mistakes while calibrating or the other explanation could be because the photos were already calibrated due to mobile camera inbuilt image processing and we ended up making it worse or un-calibrating the images, it makes more sense to use the original version for the stitching as follows.

Stitching the distorted/uncalibrated images:

The harris feature detection images earlier were for the distorted versions of the mural and the following is the panorama generated by using it.

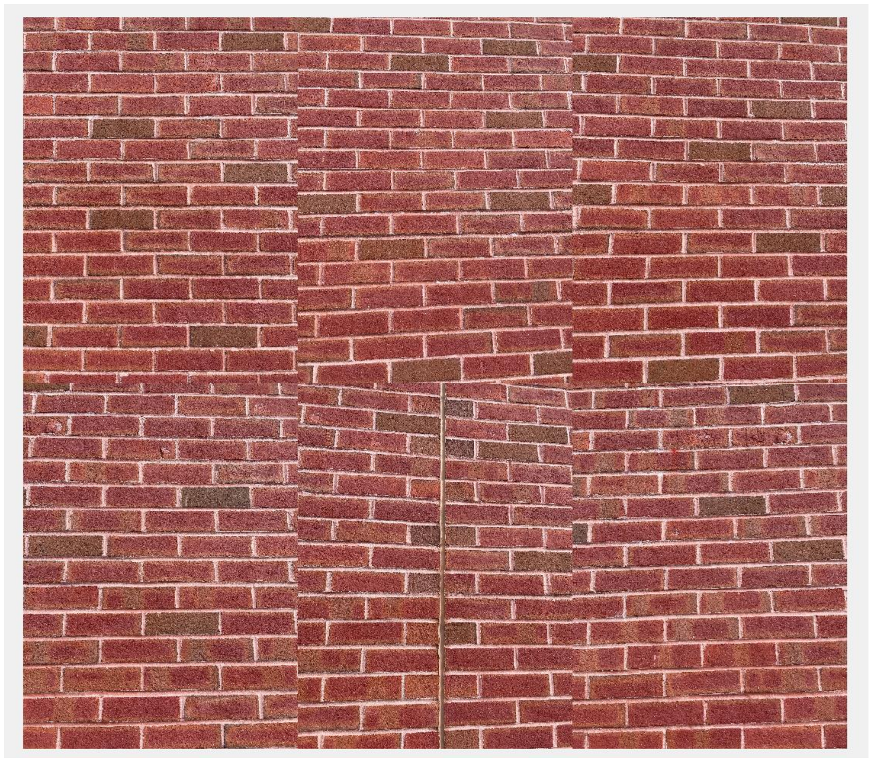


The result comes out a lot better in this case just as expected, we also went with the projective geometric transform method since our photos were not oriented in the same way which would work better with affine method. The building part is blended quite well except for the right most frame which is due to a steep change in oriented relative to the other images which was a data collection error.

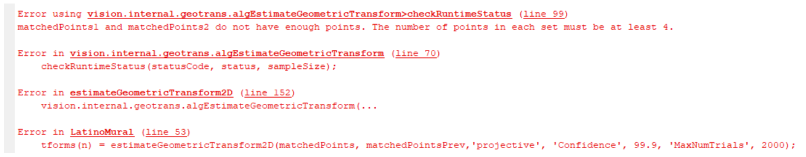
The road part isn't quite accurate, one explanation is due to it not being able to find enough features to blend since other than the crosswalk, the road is very monotonous and not a lot of unique features to stitch, other reason is due to the algorithm which uses 2D methods but the road part makes it 3D and we get compatibility issues which most likely caused the errors.

Part 3: Cinder block wall images (50% overlap)

We use the following images for the stitching process:



We try running the same code used earlier for stitching these images with parameters 1500 [15 15] but the harris detector fails at the 3rd image due to not enough matching points.

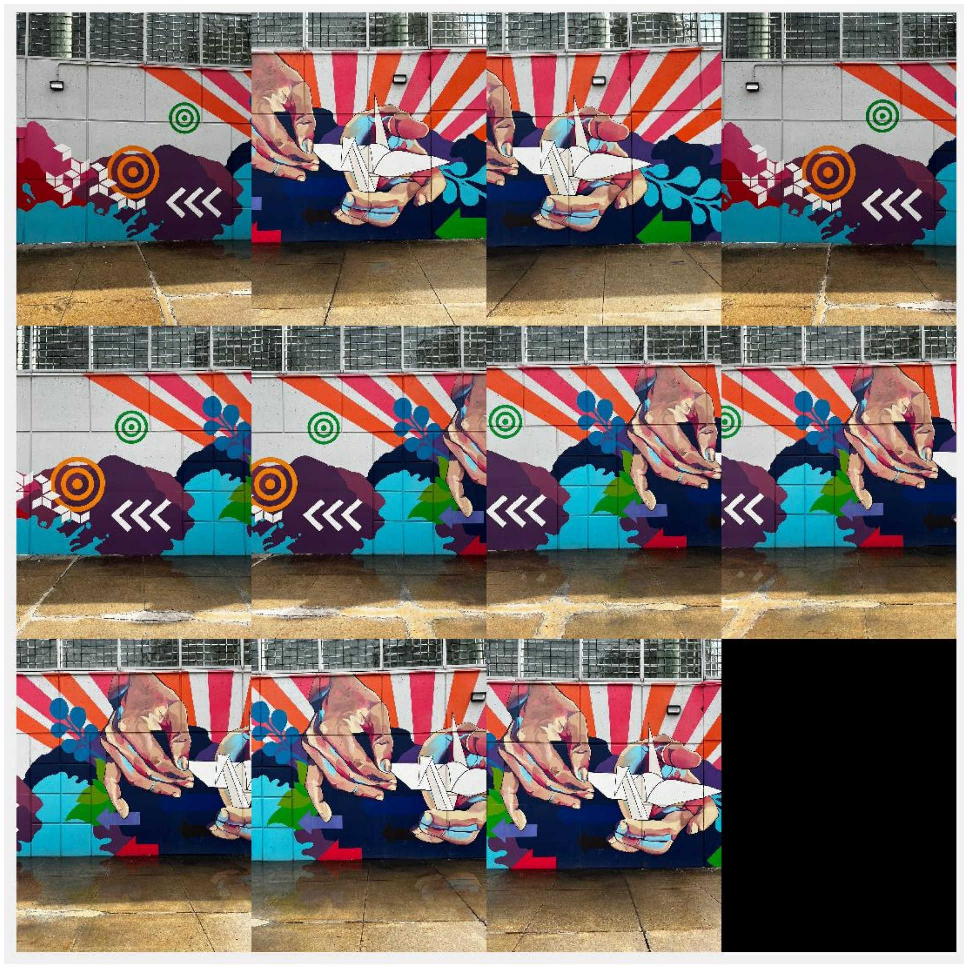


```
Error using vision.internal.geotrans.algEstimateGeometricTransform>checkRuntimeStatus (line 99)
matchedPoints1 and matchedPoints2 do not have enough points. The number of points in each set must be at least 4.

Error in vision.internal.geotrans.algEstimateGeometricTransform (line 70)
    checkRuntimeStatus(statusCode, status, sampleSize);

Error in estimateGeometricTransform2D (line 152)
    vision.internal.geotrans.algEstimateGeometricTransform(...

Error in LatinoMural (line 53)
    tforms(n) = estimateGeometricTransform2D(matchedPoints, matchedPointsPrev,'projective', 'Confidence', 99.9, 'MaxNumTrials', 2000);
```

The reason it’s unable to find matching points is due to lack of unique features, since cinder block walls have a lot of repetitive patterns. The algorithm cannot perform with very high similarity between the images and lack of feature points.

After increasing feature points up to 50000, after painfully long computation times, it doesn’t seem to help, and we end up with the same error hence meaning the code isn’t able to find any matching points between 2 or more of the images.

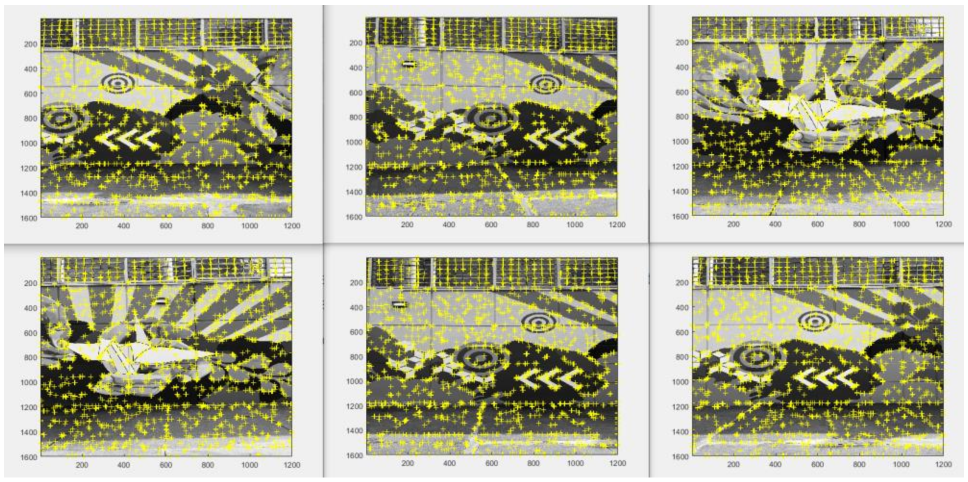
Part 4: Ruggles Graffiti (15% overlap)



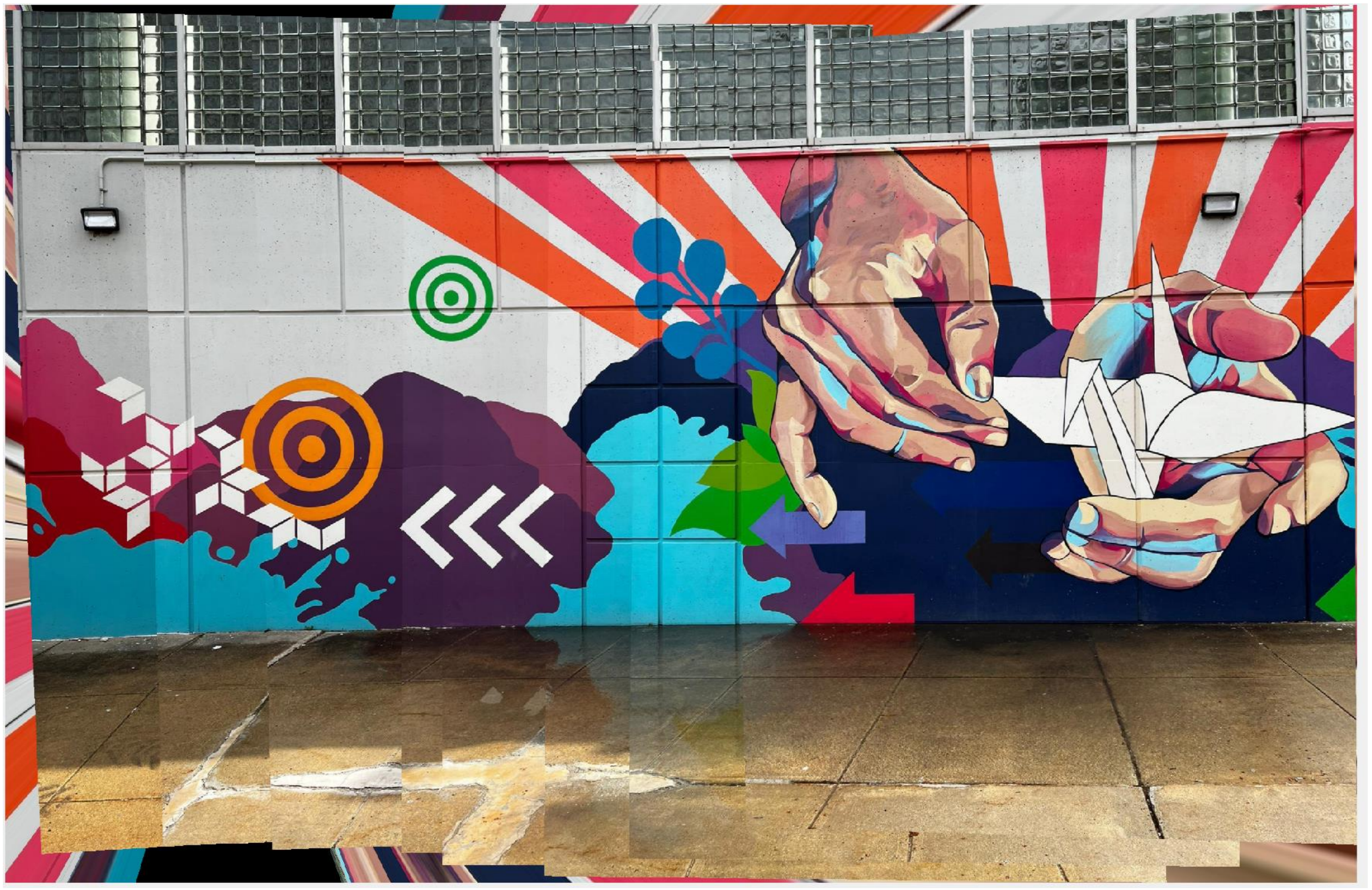
The above are the eleven 15% overlap photos of the graffiti on the Ruggles boundary wall facing the campus.

We tweak around a little with the Harris feature parameters by setting the maximum number of features to 2000 and the tile to [20 20] to get optimal results.

The following are some of the Harris features detected:



The features are well distributed because the images or the graffiti has a lot of unique features or corners to detect, hence the results come out smooth as below:



The initial parameters of 500 [1 1] did not yield optimal results so we started maximum number of feature points from 500 and incremented by 500, eventually we got better results at 2000 since the features were more spread out at this point enabling the stitching process to be way more accurate, since more features = more stitching points which leads to more accuracy.

The other factor was tile size, lesser tile size meant lot of feature points in the overlapping area were missed so we increased tile size all the way to [15 15] to have good, stitched images.

Conclusion:

We now understand how robots in real world stitch images together which are used in various applications such as SLAM or just mapping. The calibration process wasn't as satisfactory but that could be due to the camera sensor used that has pre-processing already rather than your standard cameras used in robotic applications which have to be manually calibrated. We also applied feature extraction to get an idea how it works. Also, the fact that more features are better but at the cost of greater computation time.