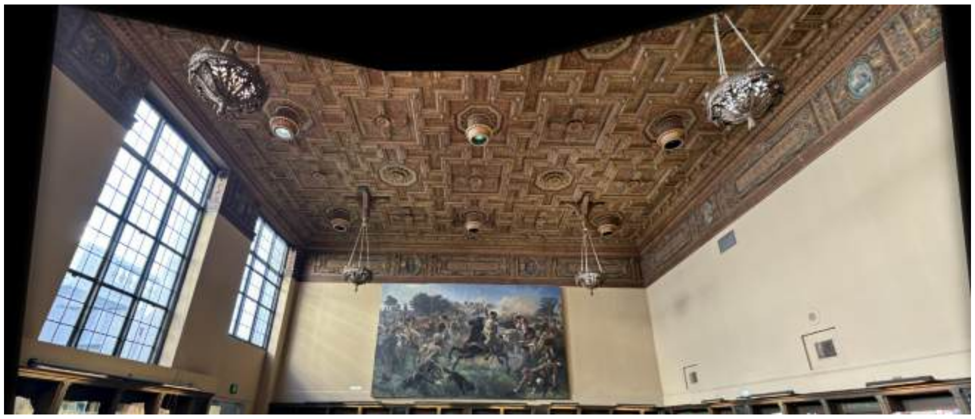
..

image warping and mosaicing



a mosaic of the heyns reading room at uc berkeley

part a:

background

imagine stitching together photos seamlessly, like creating panoramas on your phone or the immersive scenes in vr headsets. this project dives into image mosaicing, where multiple images are registered, projectively warped, resampled, and composited into a single, cohesive mosaic. by computing homographies and warping images, i explored key techniques used in modern camera apps and augmented reality systems.

shooting the pictures

the pictures below were shot with my iphone camera with the camera itself being the center of projection; the camera acted as a pivot as i slowly turned my camera and snapped some photos. this is important for when we warp our images and stitch them together to minimize errors and artifacts. currently, these images are in its **raw and original form** where they will later on be projected onto the center image via the projection transformation matrix.

hearst tennis courts at uc berkeley on a saturday morning

the roger w. heyns reading room at uc berkeley

some pictures of the street outside my apartment

recover homographies

to recover the homographies of the images, we simply find the transformation `p' = Hp` where **H** is a 3x3 matrix with 8 degrees of freedom. in my code, i use the function

```
H = computeH(im1_pts,im2_pts)
```

to compute matrix **H** where *im1_pts* and *im2_pts* are the corresponding **(x, y)** points of two images. the homography matrix can be solved by recovering the 8 parameters as seen by solving the following equation:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix}$$

solving for the projection transformation matrix

we can then expand this out to solve for the parameters *a to h*:

$$\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -xx' & -yx' \\ 0 & 0 & 0 & x & y & 1 & -xy' & -yy' \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

solve via least squares

this solves for a pair of corresponding points. to solve for more corresponding points, we simply just stack the matrix on the left vertically.

warp the images

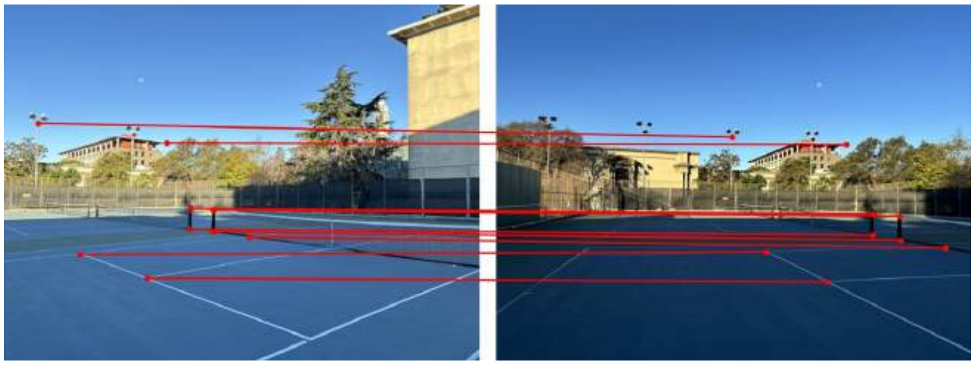
we can use the `computeH()` function to find the homographies of different images. in our case we will project the left and right images onto the middle image. we can then later use these images and stitch them together to form an image mosaic! as seen in lecture, inverse warping produces better results as it doesn’t result in holes in the final image; we use this for our function

```
imwarped = warpImage(im,H)
```

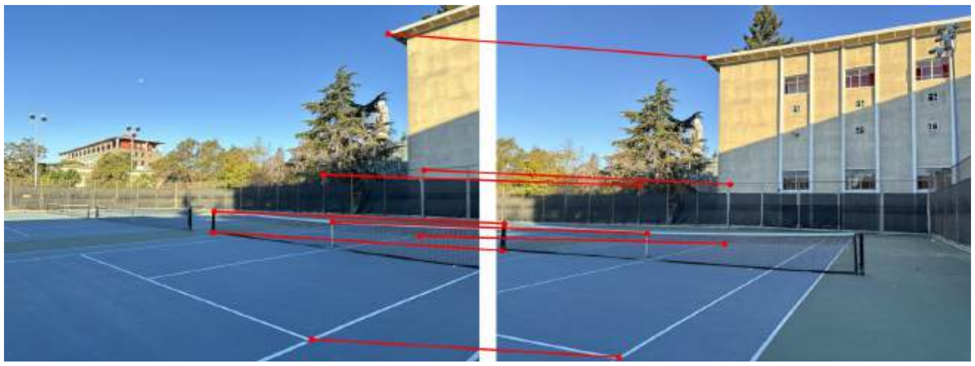
it is important to note that after warping the points of one image to another, we might encounter some values in negative locations. to deal with this, we keep track of the displacement of the warped image from the original image and apply the translation *dx* and *dy* to the warped image. furthermore, the warped image may not necessarily be the same size as the original image which is why we first find the 'boundaries' of the warped corners and then create a new canvas with the newly obtained dimension of the image.

corresponding points

below is the visualization of the corresponding points for one of our soon-to-be mosaic



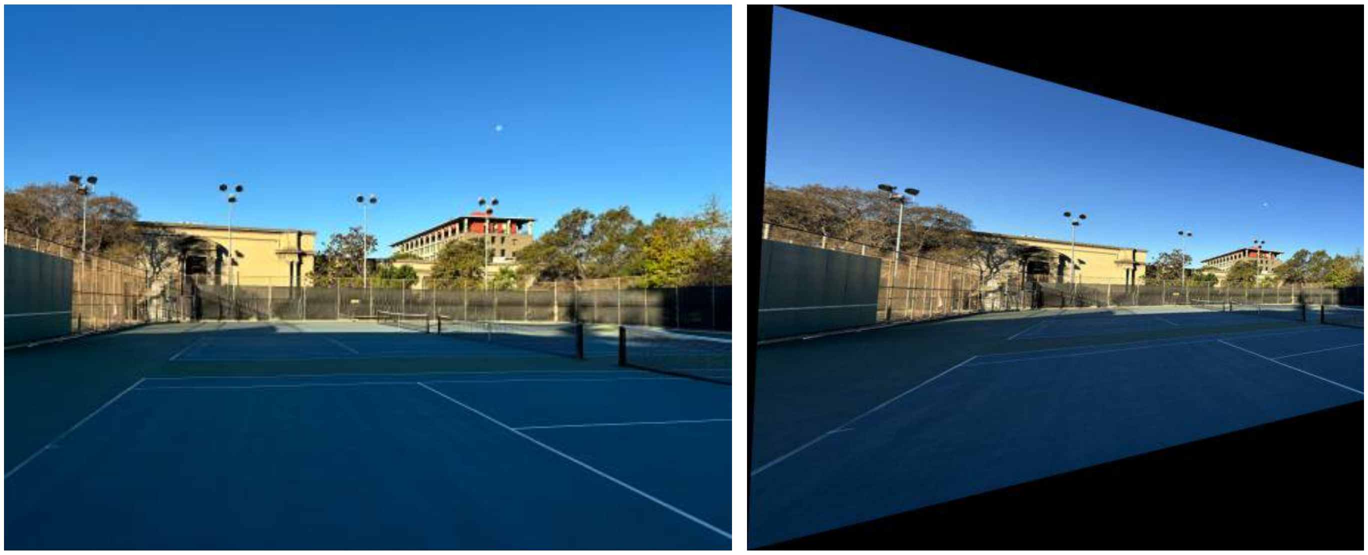
a visualization of the corresponding points of the tennis courts (right to mid)



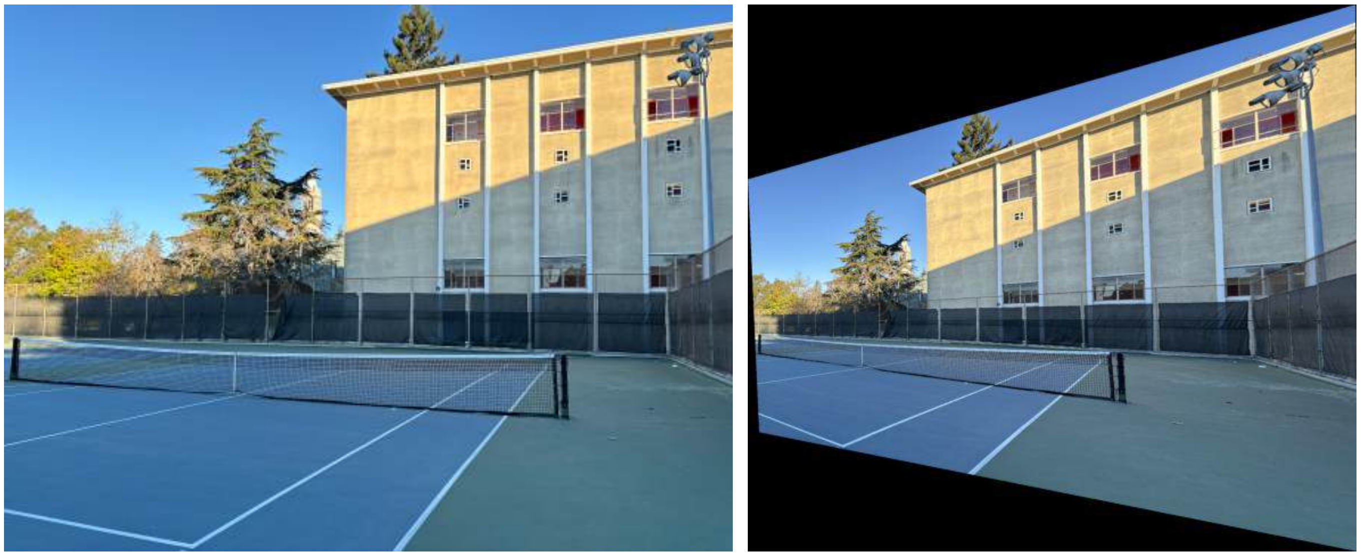
a visualization of the corresponding points of the tennis courts (left to mid)

below are some results of the warped images:

tennis courts:

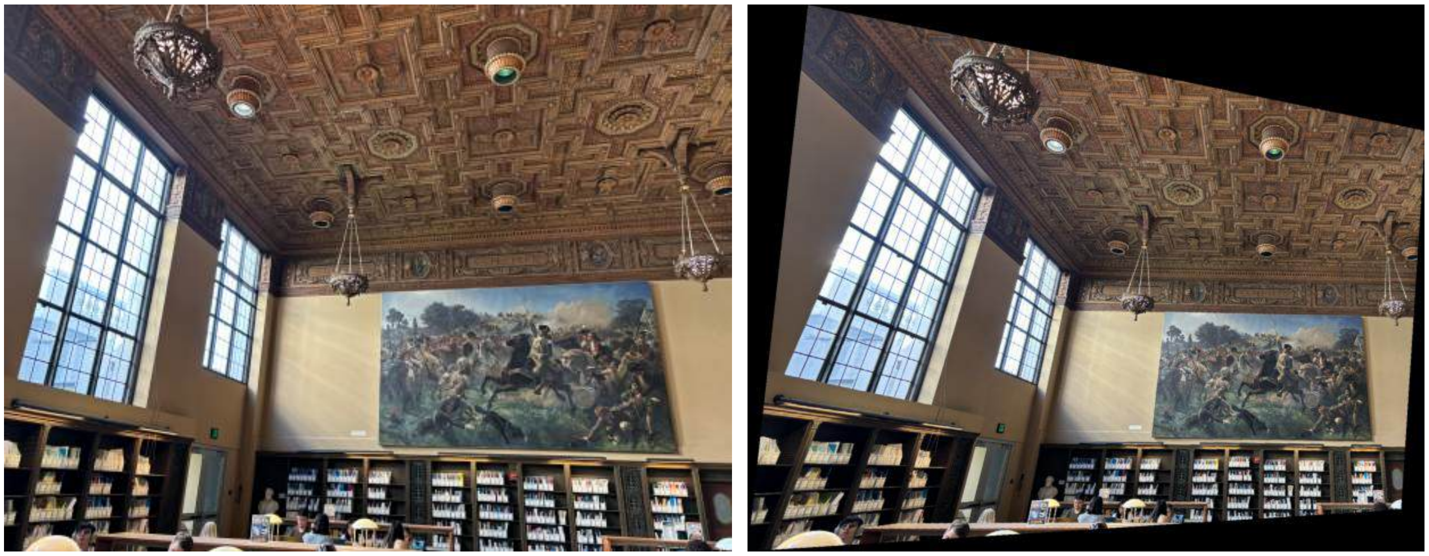


original left image vs warped image

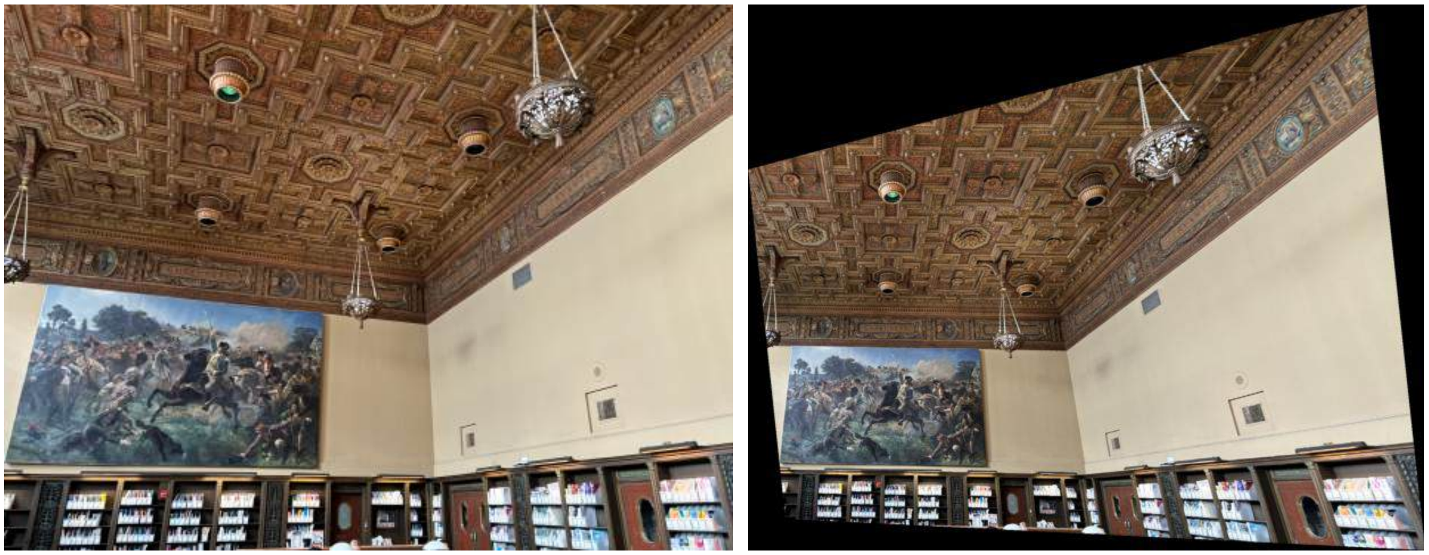


original right image vs warped image

heyns reading room:

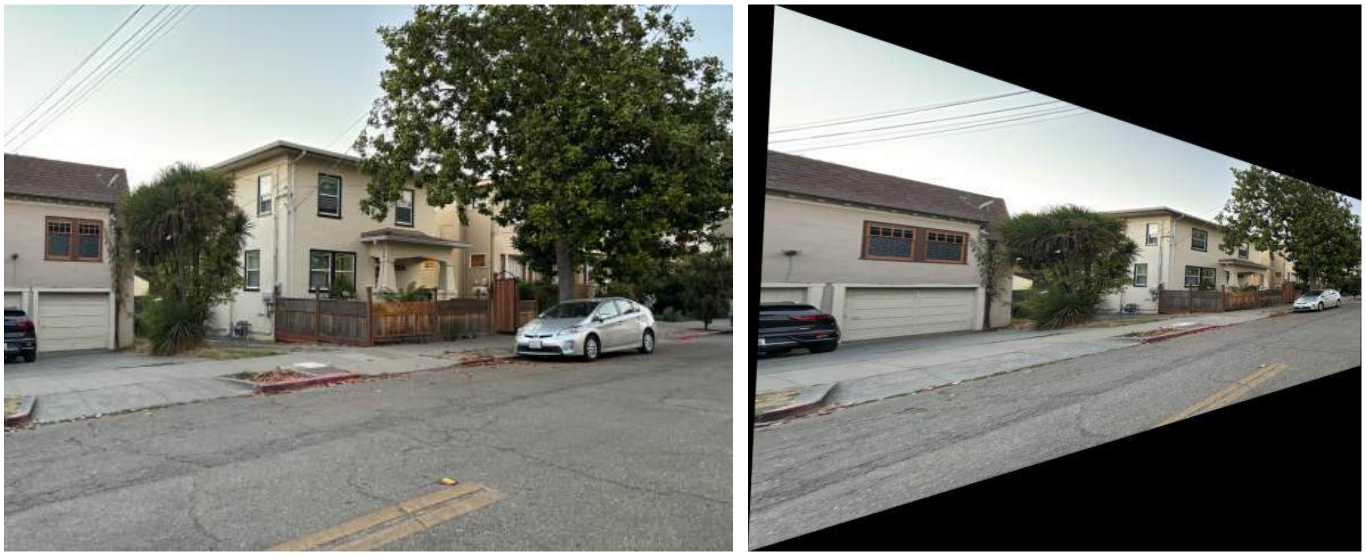


original left image vs warped image



original right image vs warped image

street:


original left image vs warped image

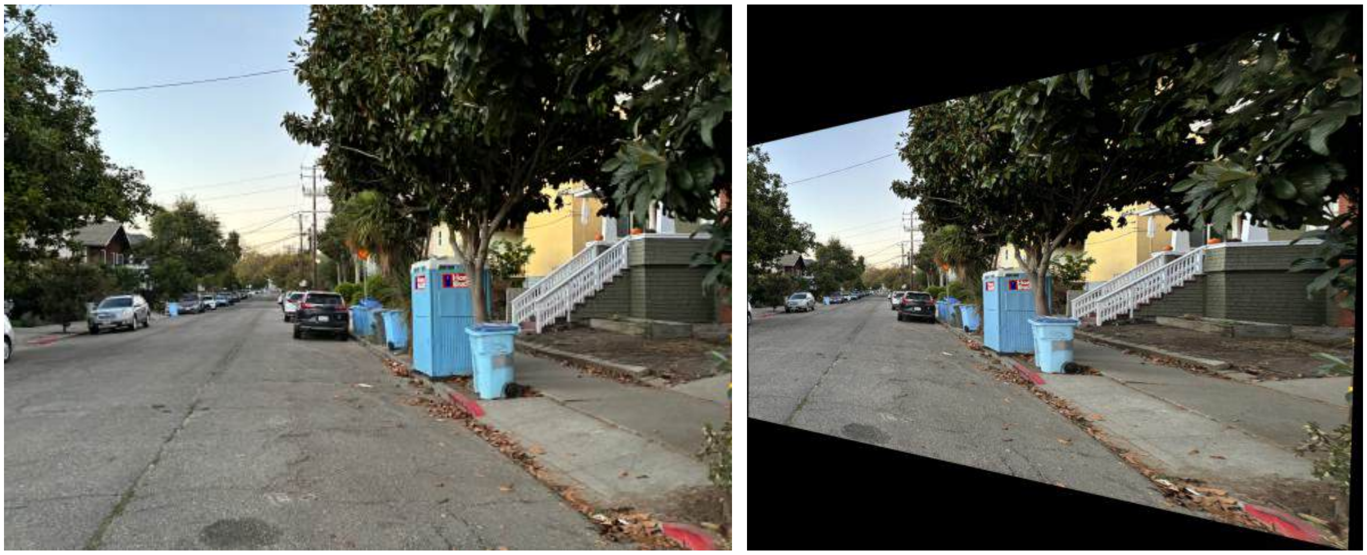
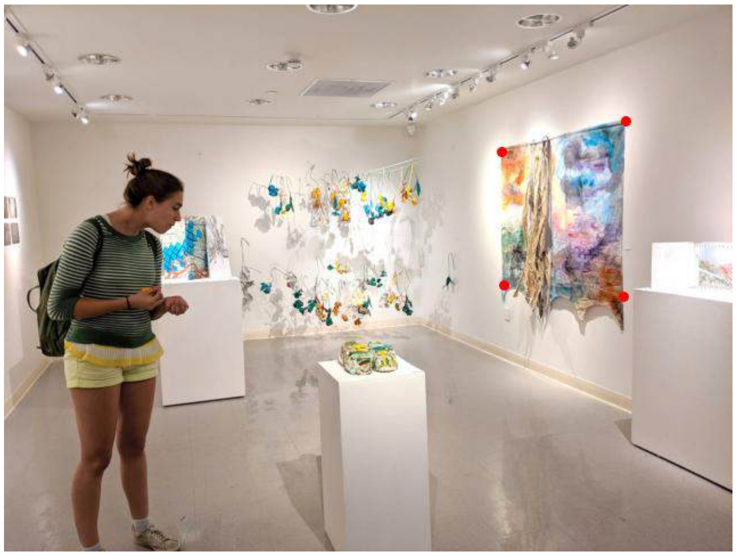

original right image vs warped image

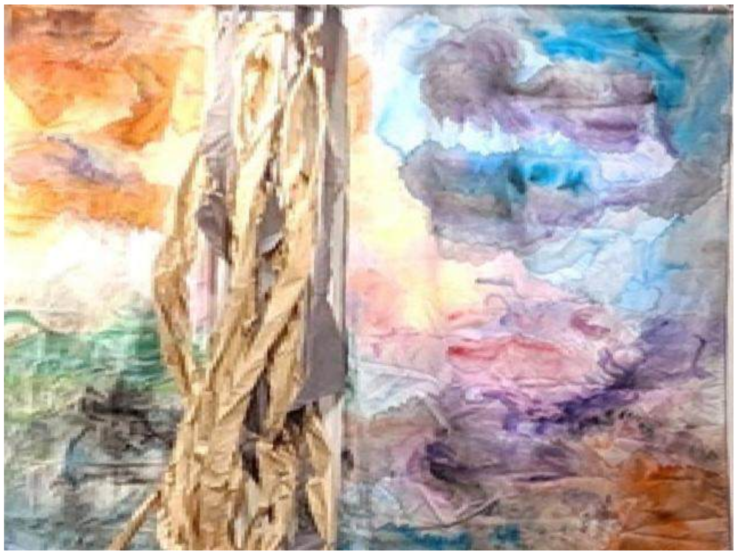
image rectification

to test that our warp function is working so far, i perform rectification on some other images that i took and some from the web. i select corner points from the object in the image that i would like to rectify and since i know the ground plane rectification, i can select the corner points that will be rectified. below are some of the original images, the corresponding points, as well as the rectified (cropped) images.

art gallery (weitman gallery, washington university):

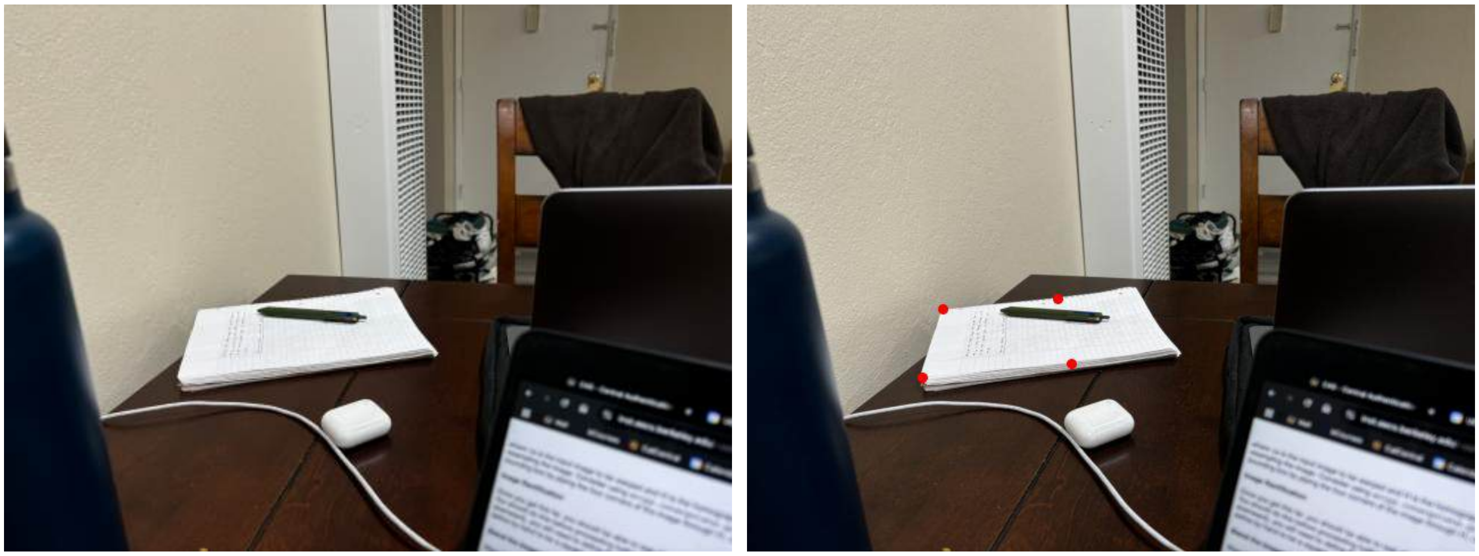


original image vs original image with points

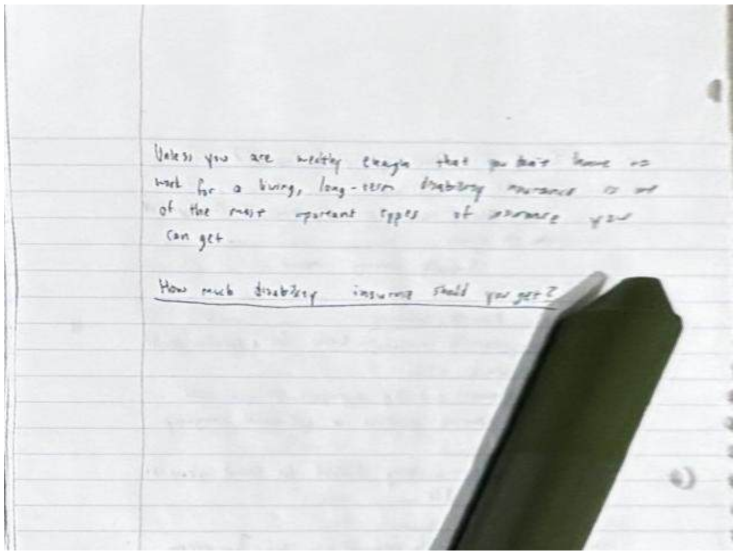


rectified image of the artwork on the wall

taking a peek at my roommate's homework:



original image vs original image with points

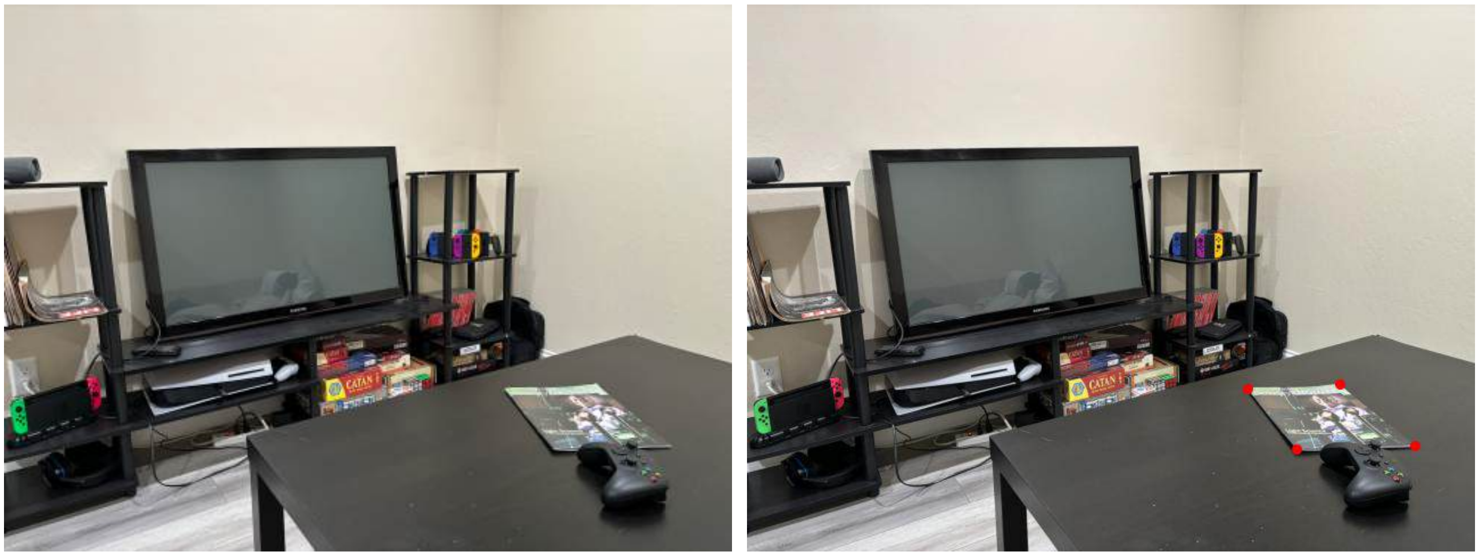


Unless you are wealthy enough that you don't have to work for a living, long-term disability insurance is one of the most important types of insurance you can get

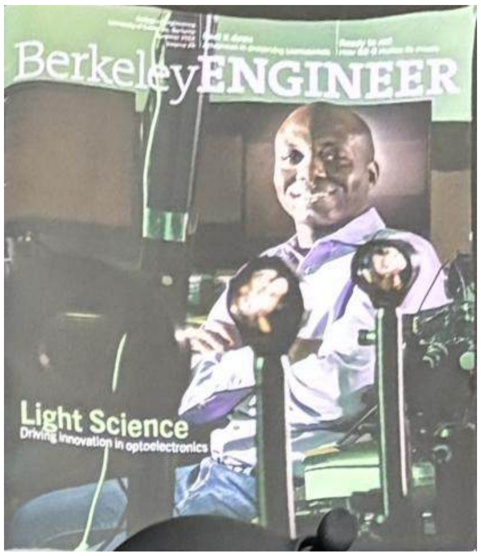
How much disability insurance should you get?

a sneaky rectified image of my roommate's homework

berkeley magazine on the coffee table:



original image vs original image with points



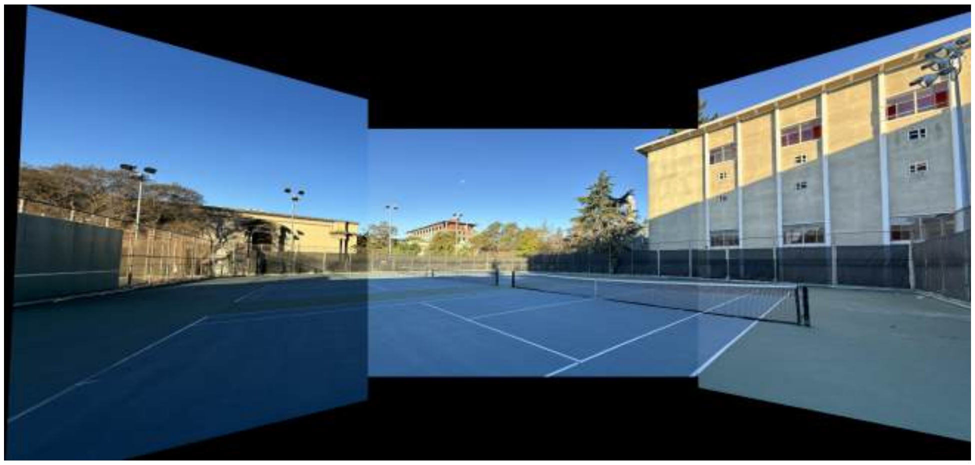
rectified version of berkeley engineering magazine

now that we know our *warpImage()* function works, we can move on to creating our image mosaics of the photographs we took before!

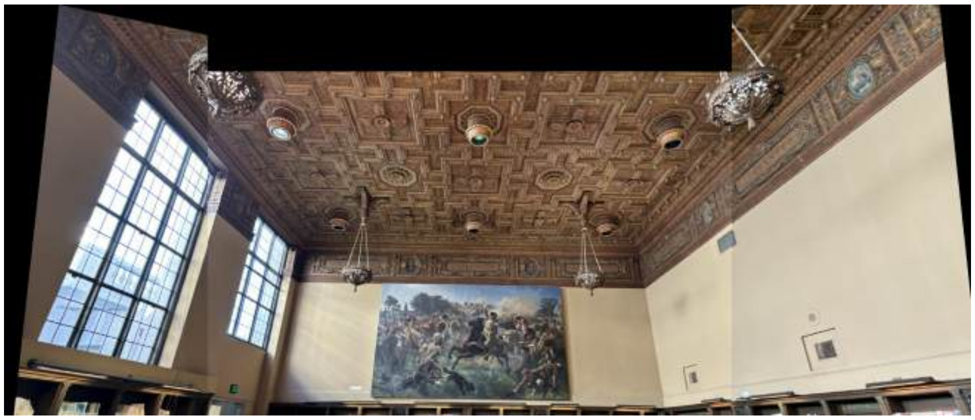
blend the images into a mosaic

using our previously warped images (tennis courts, heyns reading room, and my street view), we can stitch those warped images together, perform some blending, and we will end up with a panorama!

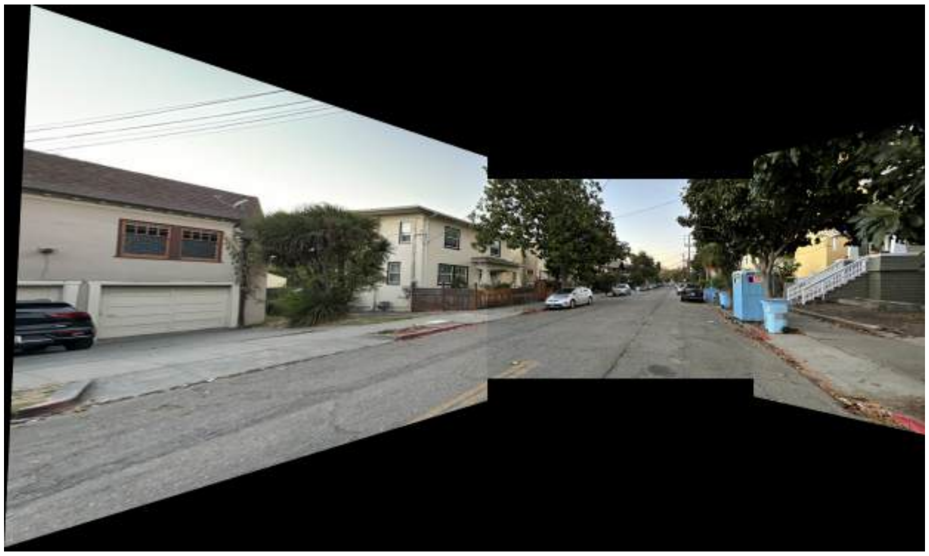
below are the results of placing the warped (left and right) images beside the (unwarped) center image via translation onto a large canvas:



mosaic of tennis courts without blending



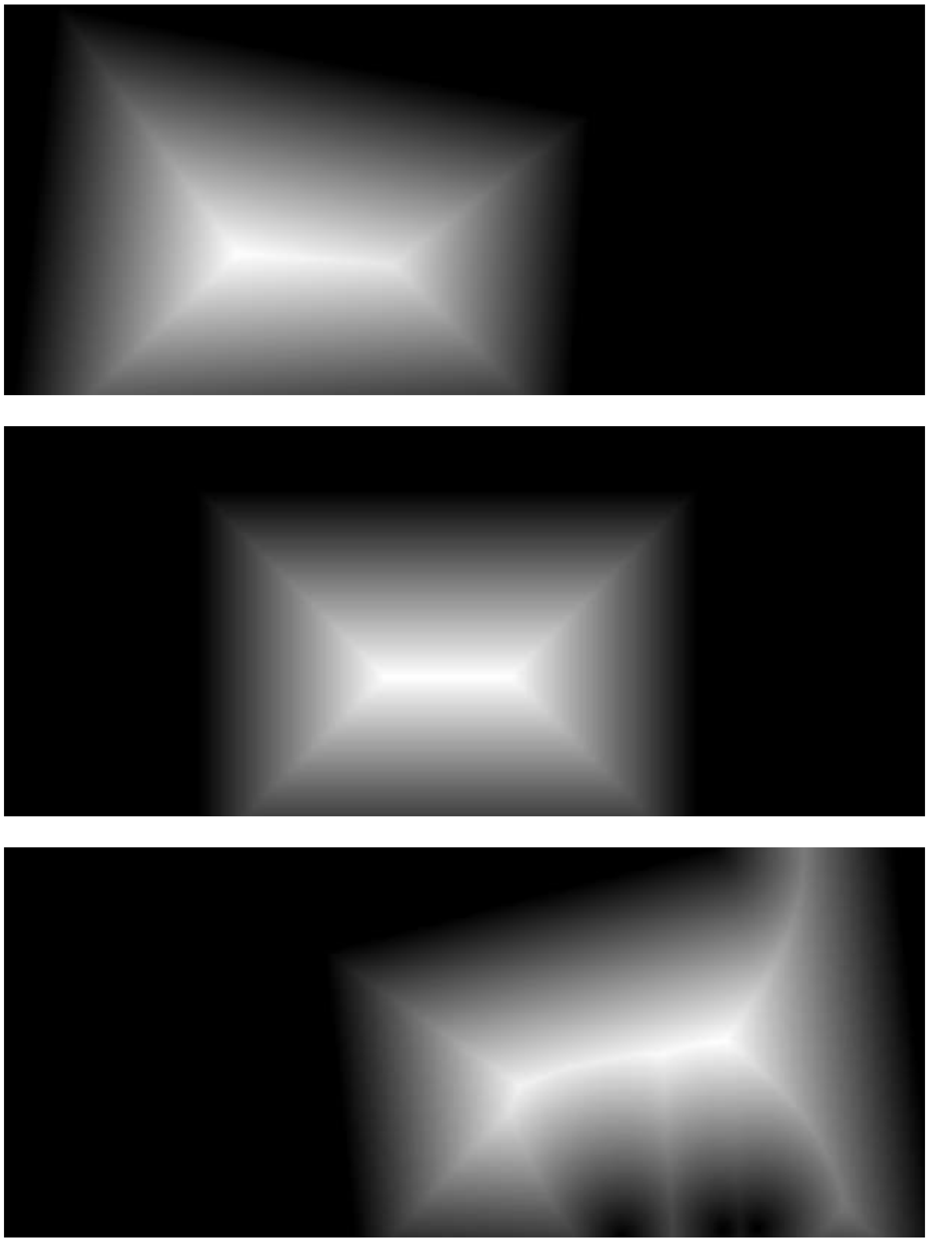
mosaic of heyns reading room without blending



mosaic of the street without blending

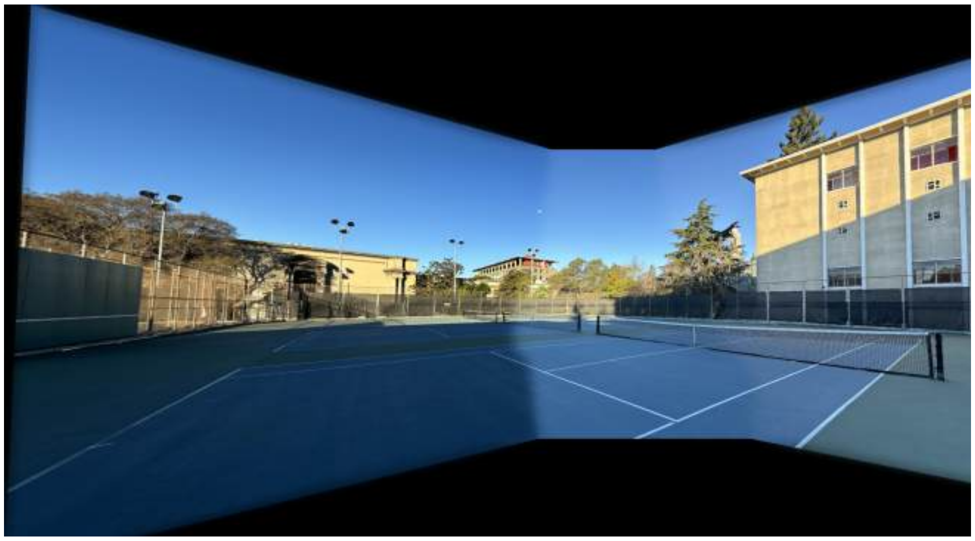
the results are pretty good! we can see that the images are aligned. but, there’s an issue. the edges where the separate images meet are very strong and we can tell that these are separate image put side-by-side, not a mosaic.

to fix this, we use what we did in the image blending project and use a level-2 laplacian pyramid to blend the seams together. i first blend the warped left image to the center, then the warped right image to the center, and then blend both of those blended images together. here are the level-2 laplacian pyramid masks i used for the heyns panoramas:

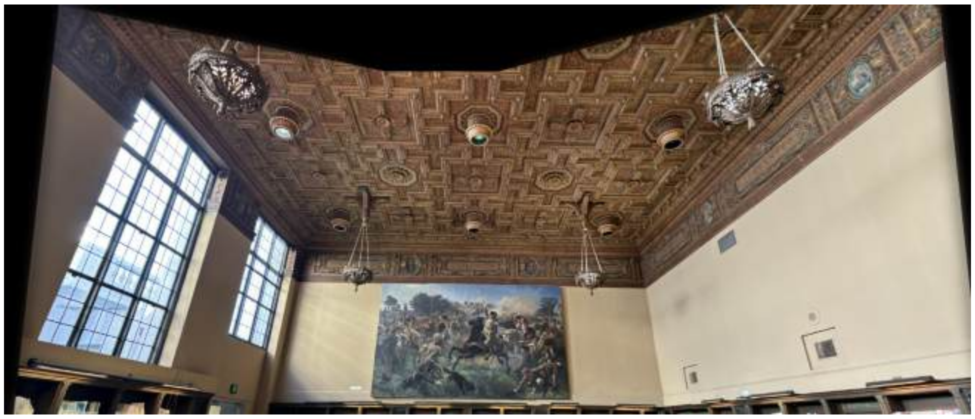


masks of used for blending the heyns images

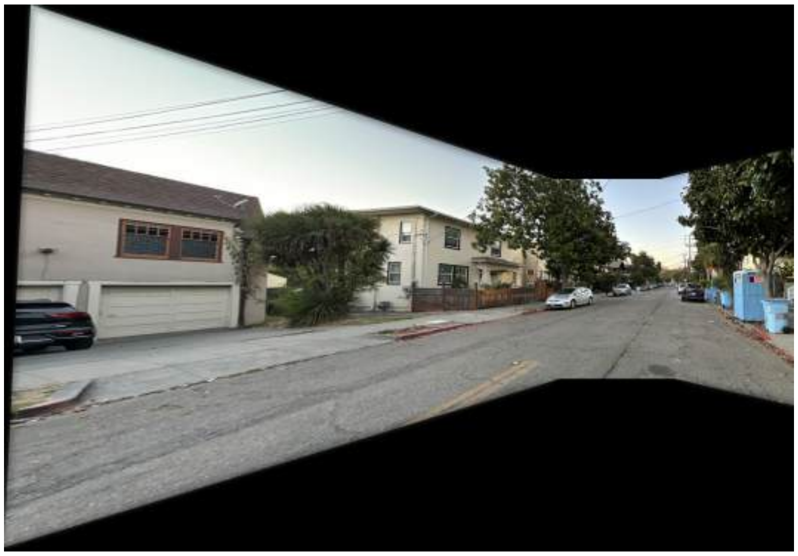
the final results are here:



mosaic of tennis courts [note: the reason for the distinct color difference is due to the images itself and the lighting of when they were taken]



mosaic of heyns reading room



mosaic of the street

you can see the huge improvement in the images! the seams are now gone – we have created panoramas from 3 different photographs.

part b: feature matching and autostitching

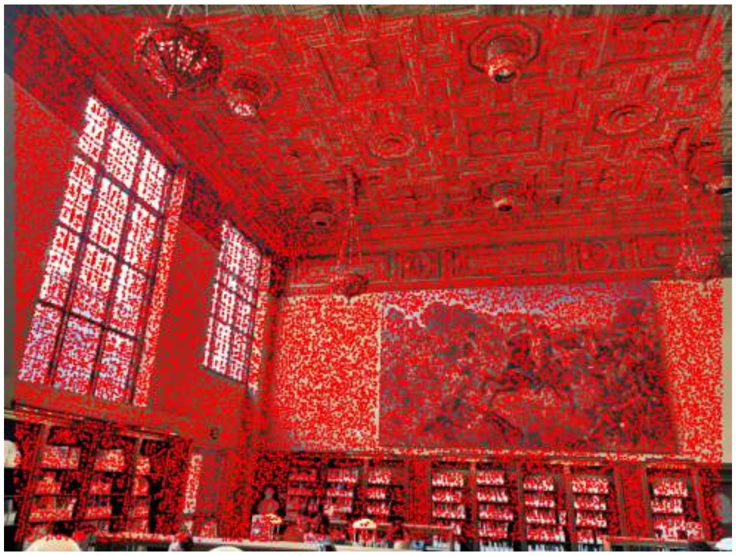
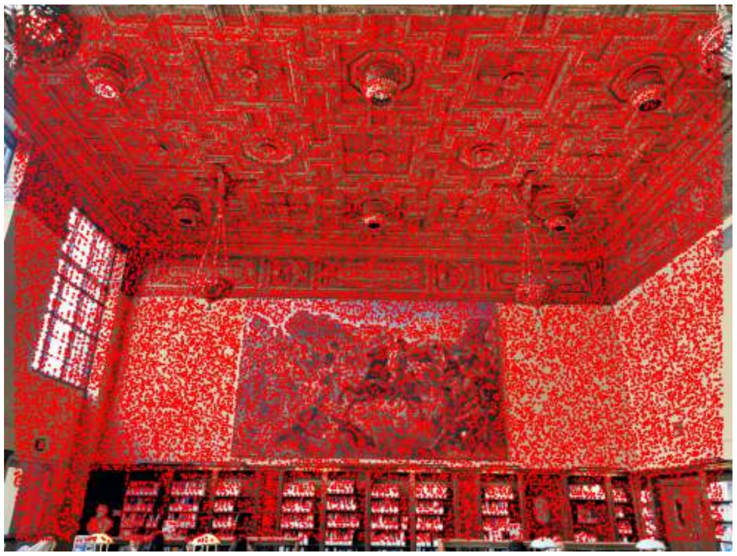
even though our mosaics turned out pretty nice, much of the process is very manual. picking the correspondence points itself takes a while. in this part of the project, i will implement feature matching and autostitching to create our mosaics for us.

corner detection

similar to how i manually selected the points, we want to select corner points by using the harris interest point detector. the harris interest point detector uses peaks in the matrix to find corners.

below are images of harris points overlaid onto my photos:

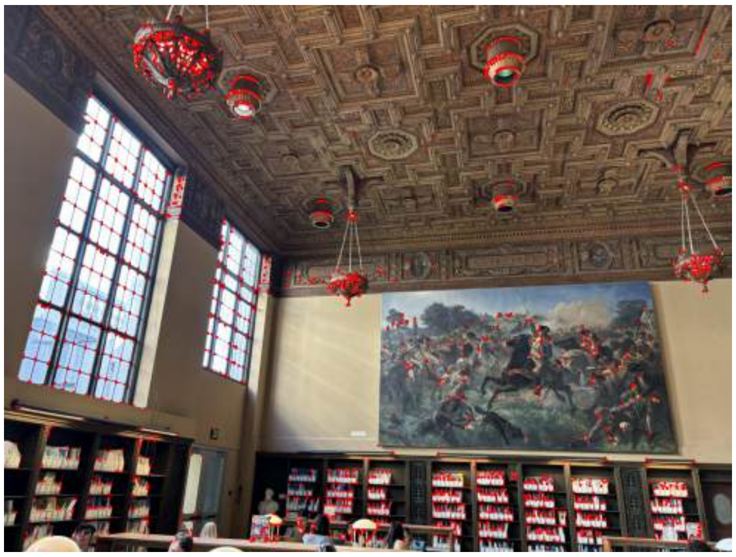
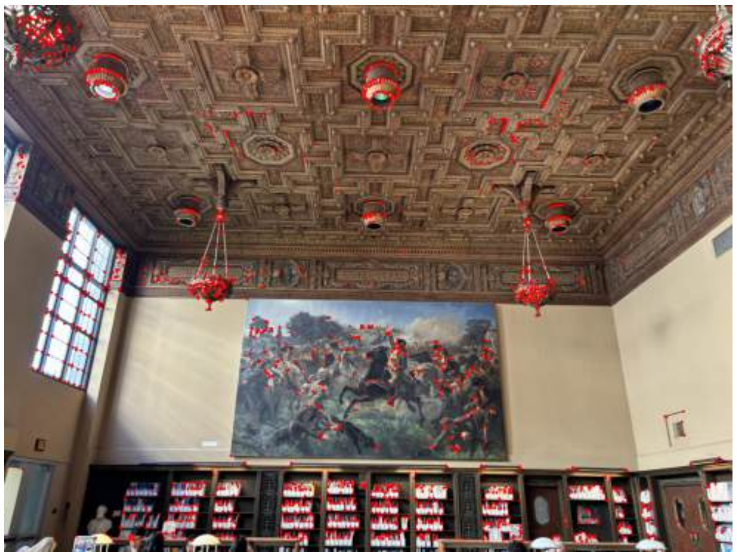
no threshold (all harris points):

heyns (left), heyns (mid)

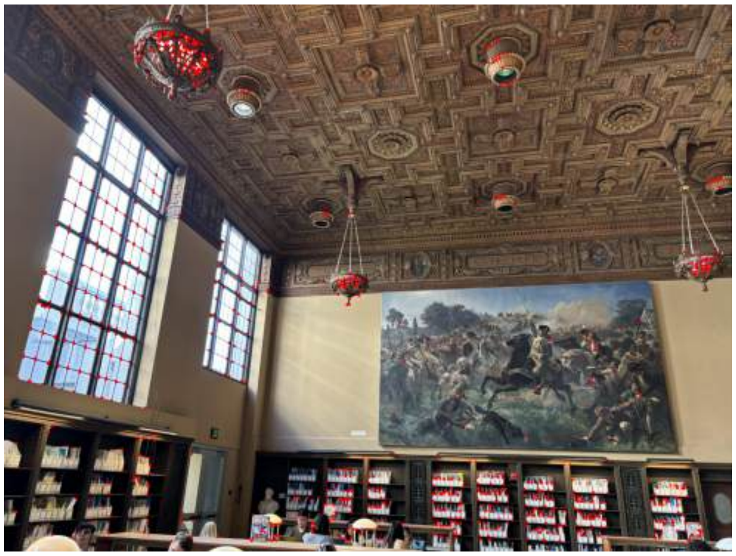
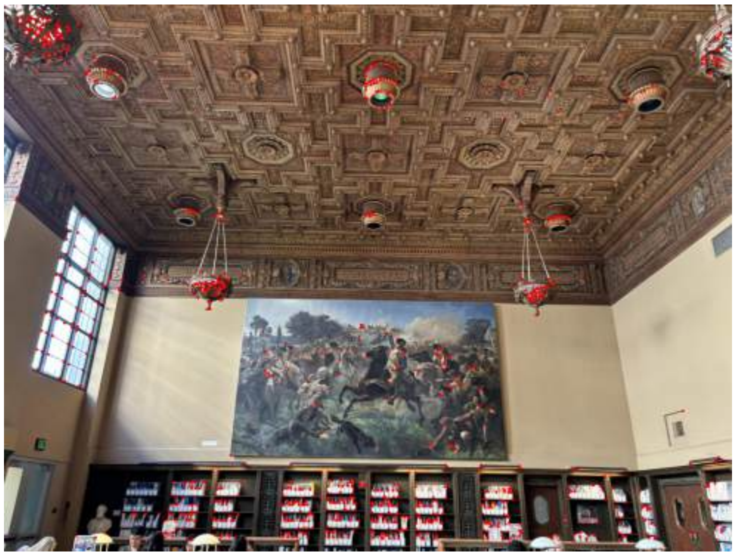
but you can see that there are way too many points. let's set a threshold to only keep the top n points.

threshold = 2000:

heyns (left), heyns (mid)

threshold = 1000:

heyns (left), heyns (mid)

using a threshold to filter out the top n corners is a good idea but it is not very sophisticated. you can see in our above images that our corners are not very spaced out; we get clusters spread out sparsely throughout the image.

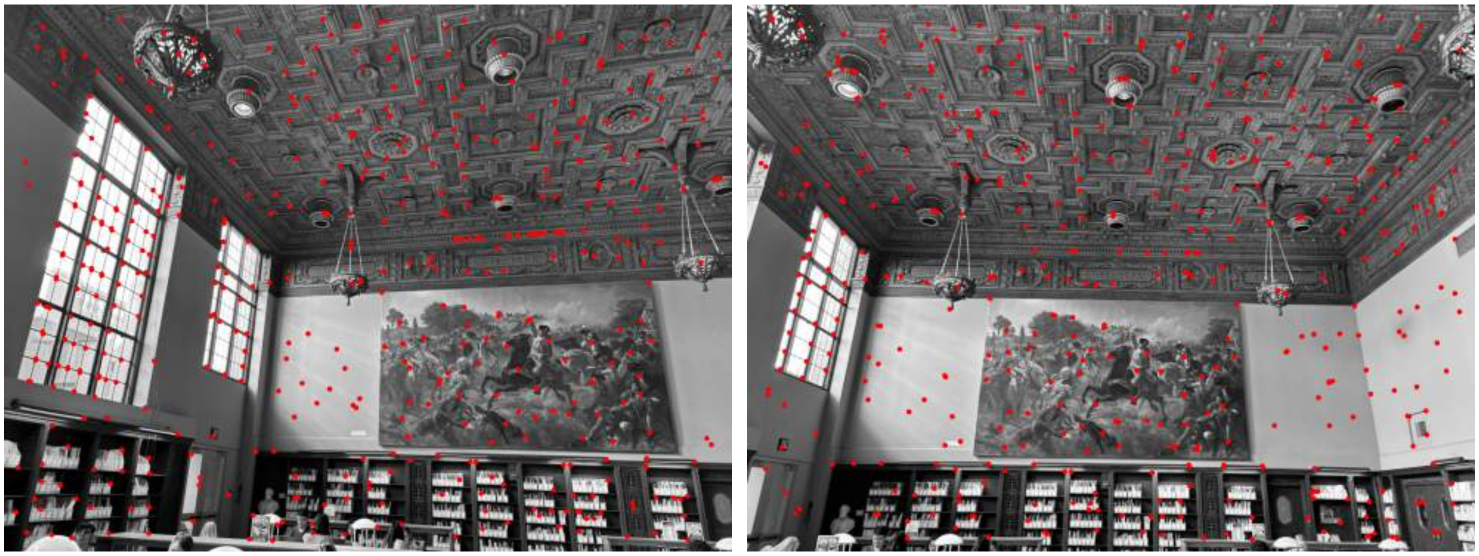
adaptive non-maximal suppression (anms)

with the harris interest points, we get lots of redudant information since many points are clustered together. to space out the points a little more, we use adaptive non-maximal suppression (anms) to make our feature matching more effective.

i implemented the algorithm from section 3 of the paper [“multi-image matching using multi-scale oriented patches” by brown et al.]().

you can see my anms points overlaid onto the images below. you can observe that the points are a lot more evenly spread out compared to just using the threshold:

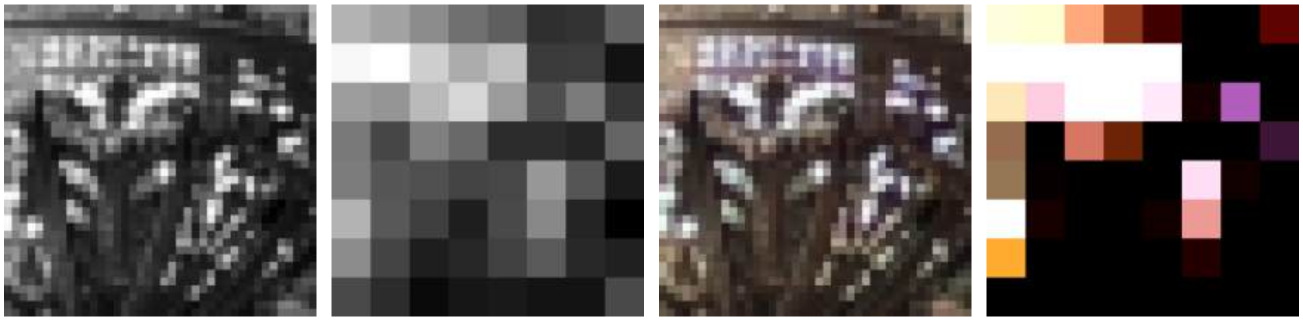
500 anms points:


heyns (left), heyns (mid)

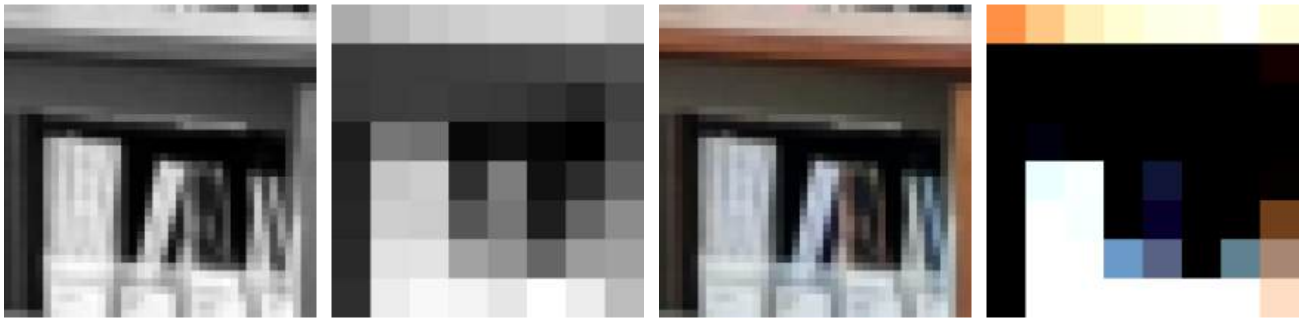
extracting feature descriptors

before we can match our correspondence points, we need to get some context for each point and to do this, we use feature descriptors. for each points, we get its feature descriptor which is a sample of 40x40 pixels centered around that pixel and downsize (space it out) by s=5 (resulting sample will be 8x8). after we do this, we have to bias/gain-normalize our descriptors. here are some of the descriptors from the heyns reading room images.

point at (x=683, y=546):

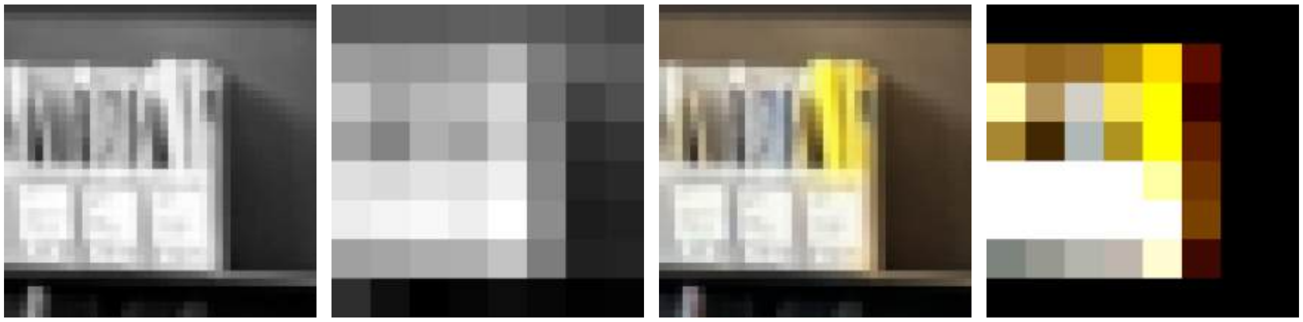

40x40 sample window vs 8x8 descriptor (in both b&w and color)

point at (x=856, y=1041):



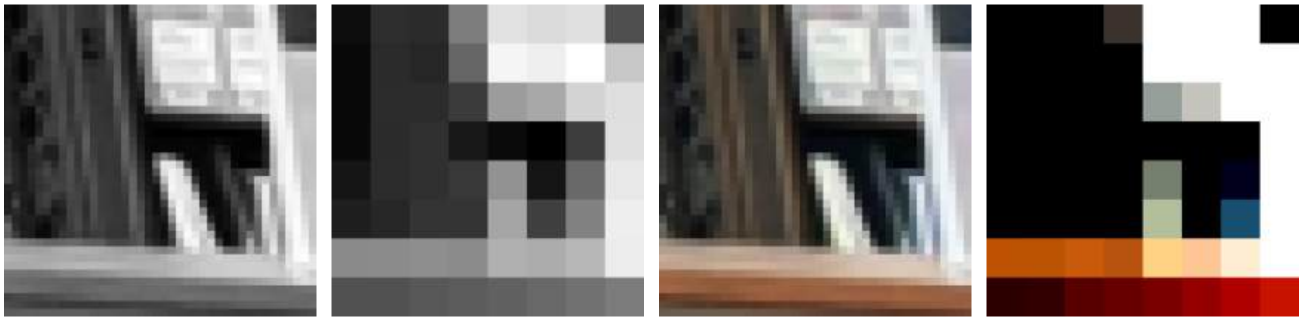
40x40 sample window vs 8x8 descriptor (in both b&w and color)

point at (x=893, y=930):



40x40 sample window vs 8x8 descriptor (in both b&w and color)

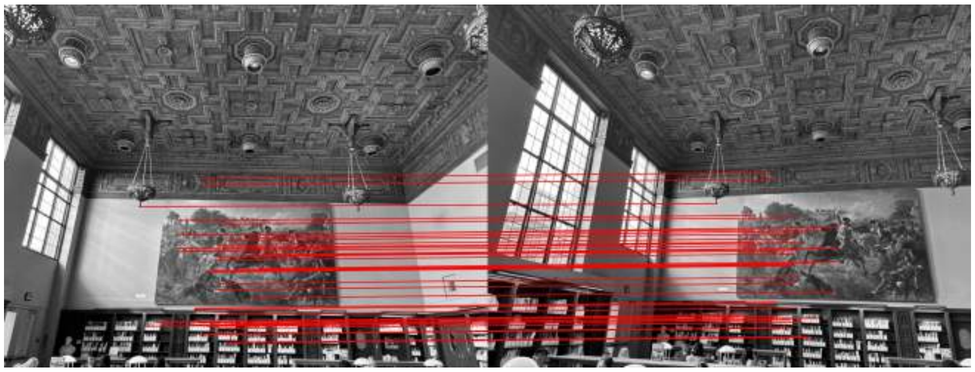
point at (x=1277, y=1034):



40x40 sample window vs 8x8 descriptor (in both b&w and color)

feature matching between two images

we use the feature descriptors from both images and find the closest corresponding descriptors to match to each other. we implement section 5 of brown's paper in this part of the project. we also make use of lowe's trick for thresholding by setting the threshold to the ratio between the nearest and second-nearest neighbor.



corresponding points between both images; # matched points: 41

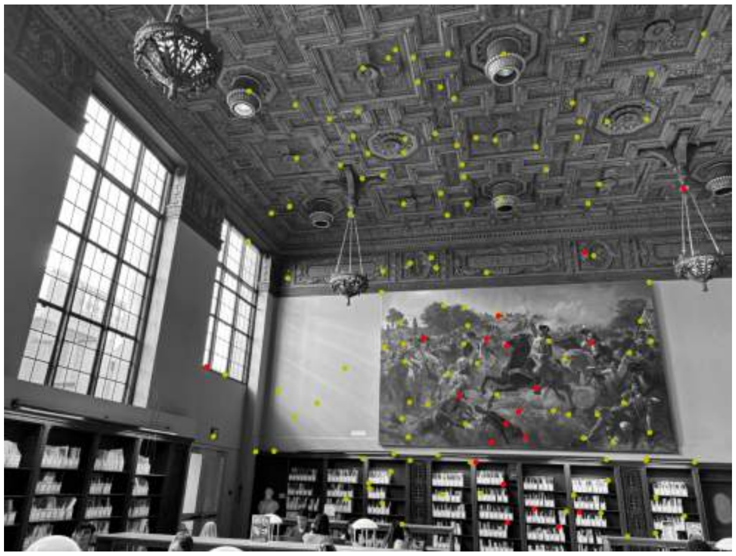
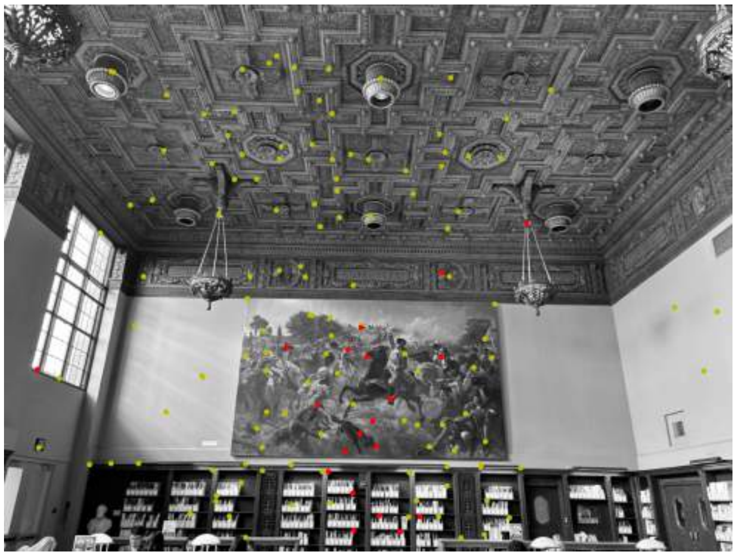
4-point random sample consensus (ransac)

lowe’s trick improves the matching process by reducing the number of outliers. however, we will evidently still have some outliers remaining which can really skew our calculations when finding the least-squares solution. just one pair of incorrect points can really mess up the homography matrix calculations. this is why we need to use ransac.

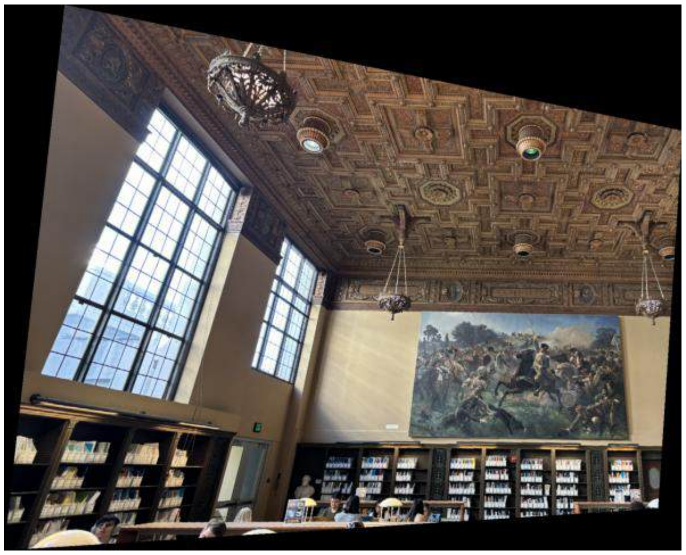
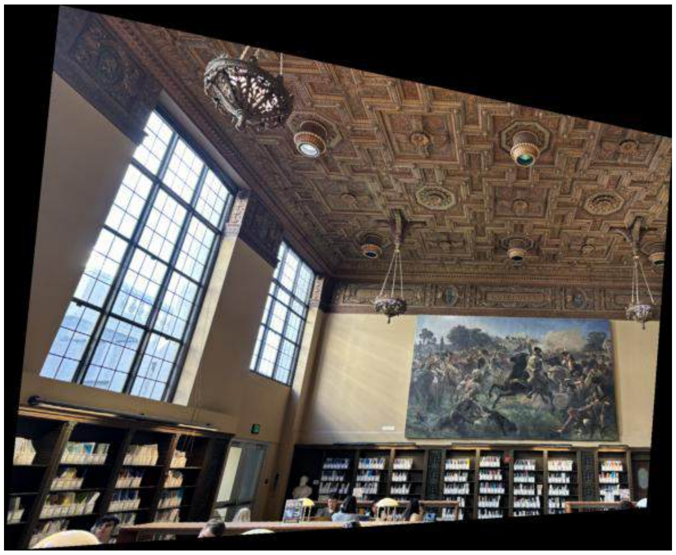
the general idea of ransac:

1. select four feature pairs (at random)
2. compute homography H (exact)
3. compute inliers where `dist(pi’, Hpi) < ε`
4. keep largest set of inliers
5. re-compute least-squares H estimate on all of the inliers

we can see below that ransac does a really good job at eliminating the outliers:

points matched by ransac are in red; you can see that this has significantly reduced the number of outliers, only keeping the ransac points

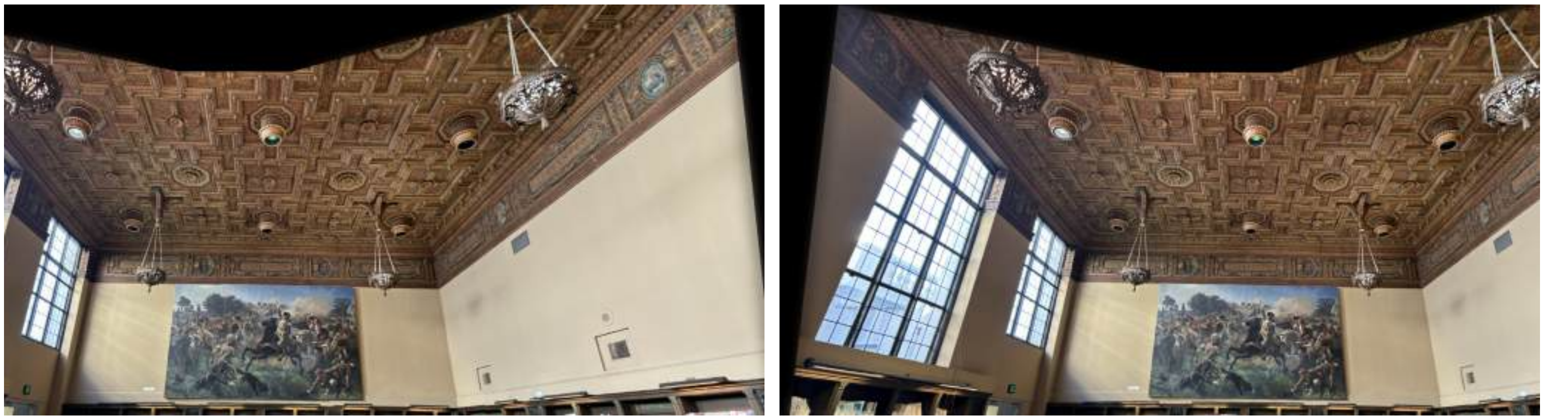



warped image (by manual correspondence), warped image (by ransac)

you can see that the homography **H** produced by ransac is pretty accurate as the warped image from both manually and automatically selected points are the same.

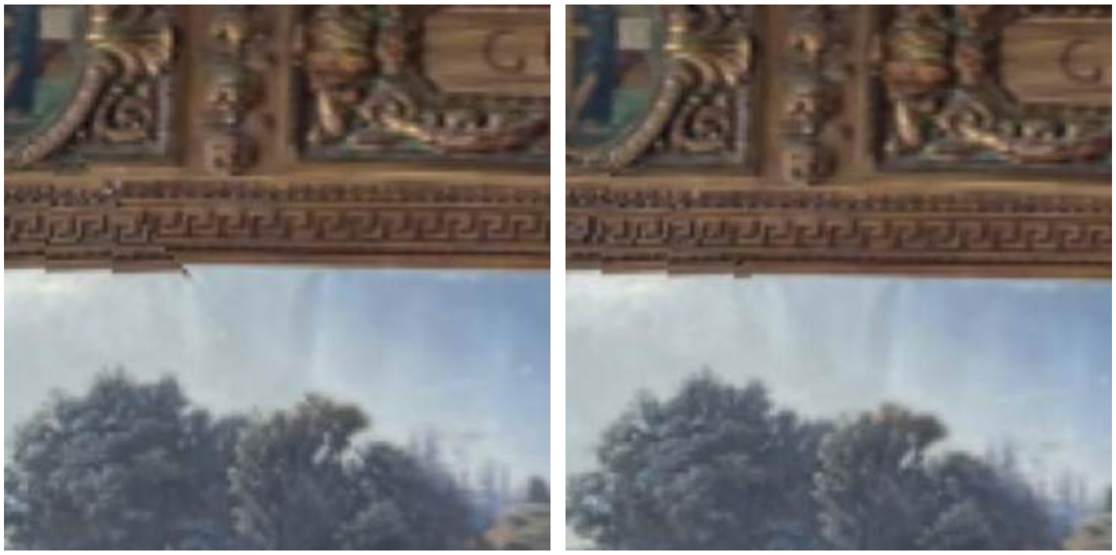
autostitching (putting everything together)

heyns



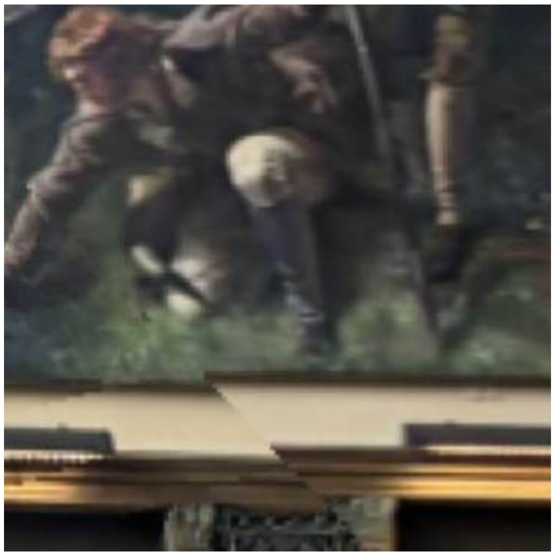
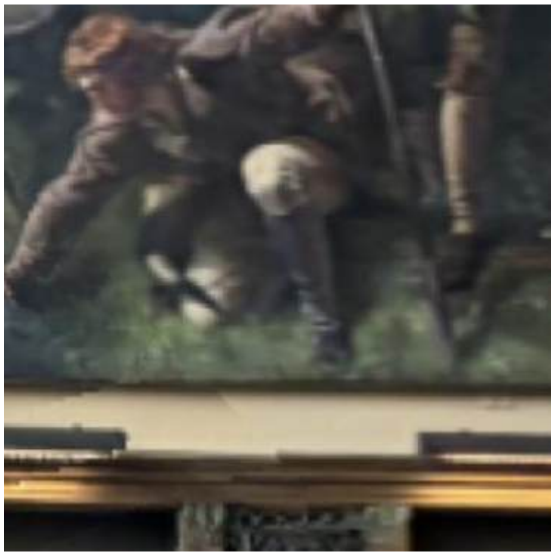
heyns reading room with manual correspondences (left) vs automatic correspondences (right)

you can see that the results are really good! they are basically identical. if we zoom in a little more, into both images, we can see that there are minor differences.



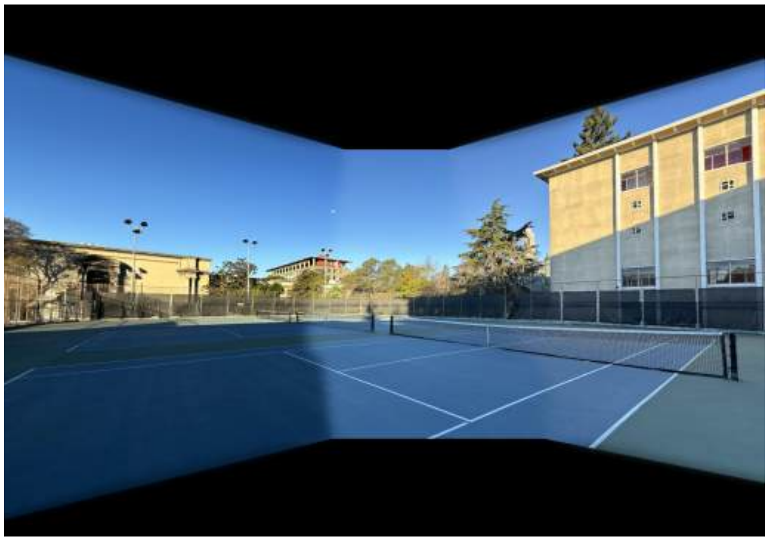
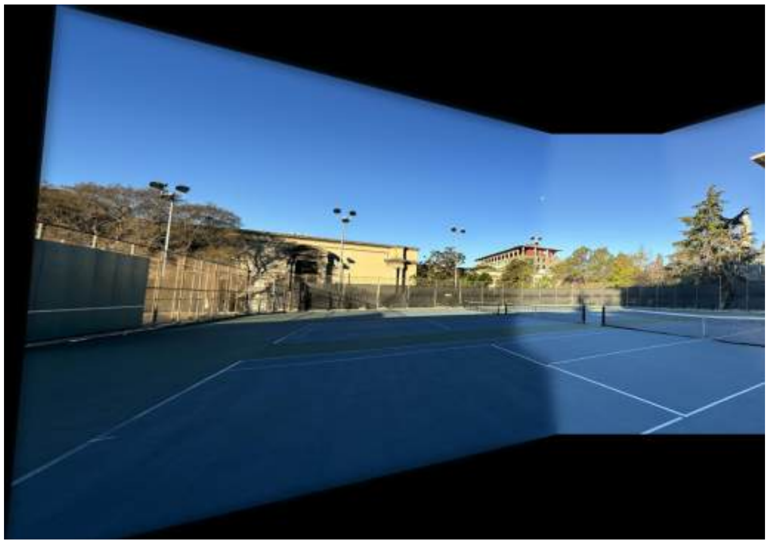
zoomed image of manual correspondence vs automatic correspondence

you can see that the automatic correspondence points using all the techniques used in the second part of the project actually does a better job than manually selected points (as expected). in the image on the left (manual correspondences), there are some noticeable artifacts while there are none in the image on the right.

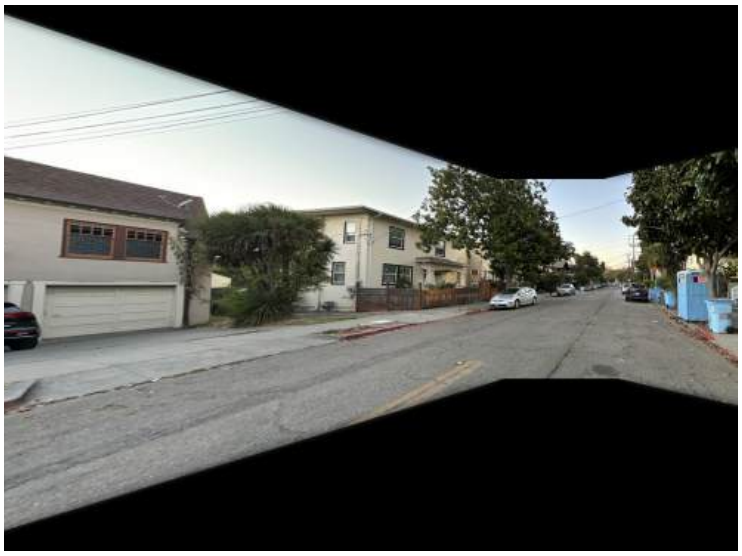
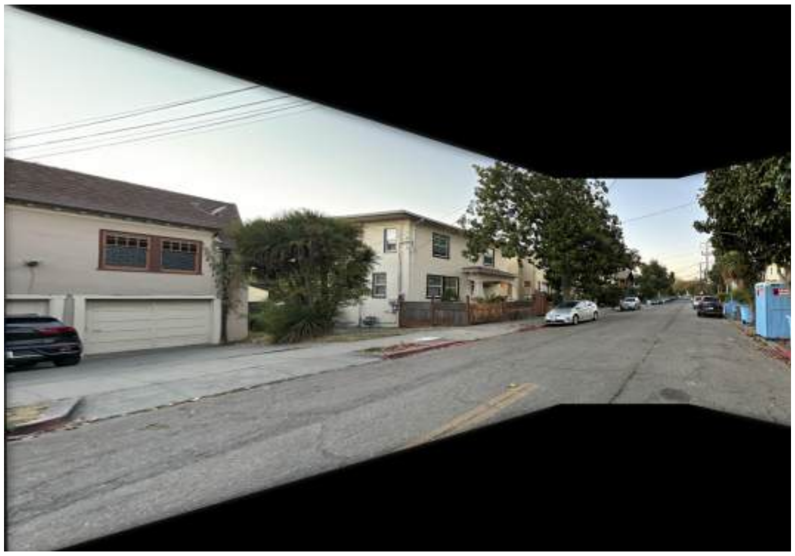
zoomed image of manual correspondence (left) vs automatic correspondence (right)

tennis

tennis courts with manual correspondences (left) vs automatic correspondences (right)

street view

street view with manual correspondences (left) vs automatic correspondences (right)

closing remarks

this project encompasses many techniques we learned in our past assignmnets and combines them all into a single project. it is also quite fascinating to see how much can be done in image processing with just mathematics and smart tricks/techniques.

being able to automate the process of selecting correspondence points and stitching/blending the images together has been both satisfying and rewarding. :)