Image Mosaicing

EECE 5639: Computer Vision

Miquel Alberti Binimelis

Introduction

In this project we will find corner features in multiple images and use them to align the images in a mosaic, by estimating a homography between corresponding features. The homography allows us to warp one image into the coordinate system of the second one to produce a mosaic containing the union of all pixels.

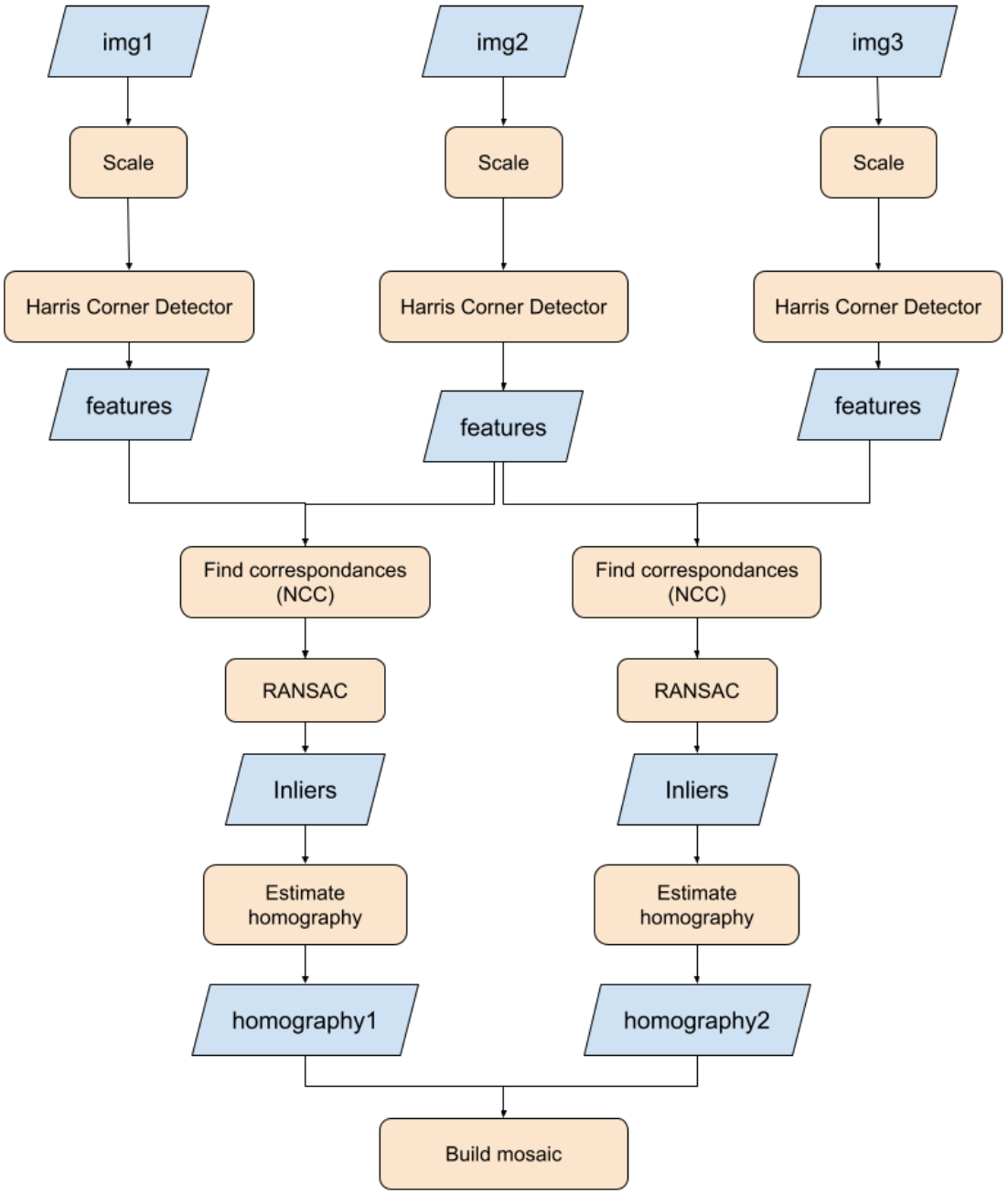
General approach

My algorithm takes three images, that I will call img1, img2, and img3, and outputs a mosaic that contains the three of them. However, the procedure I will explain can be easily generalized to as many images as needed (as long as there is overlapping between them).

The structure of the algorithm is the following:

1. Read the three images, scale them to make computations faster and save a gray version of them. I have used a scaling factor of 0.65.
2. Detect features on each image using the Harris Corner Detector.
3. Compute the two homographies that transform pixel coordinates from img1 and img3 to img2, respectively. In order to do so, for each homography we have to:
    a. Find correspondences between features in both images, using the Normalized Cross Correlation (NCC) to compare each possible pair.
    b. Estimate the homography, using RANSAC algorithm to remove outliers
4. Finally, use the homographies to wrap the three images into a single frame, creating a mosaic.

In the code, I separated each of these processes into different Python files, with the aim of making it clearer. You can see also the flowchart to have a better idea of the general approach. Let's now explain in more detail each of these steps.



Flowchart

Features detection: Harris Corner Detector

To apply the Harris Corner Detector I compute the gradients of the given images using a Sobel mask. The Sobel mask considers a 3x3 neighborhood to reduce the effect of noise, but gives more importance to the center pixel. I also apply a Gaussian smoothing mask, because gradients are very sensitive to noise. The standard deviation used is 0.5.

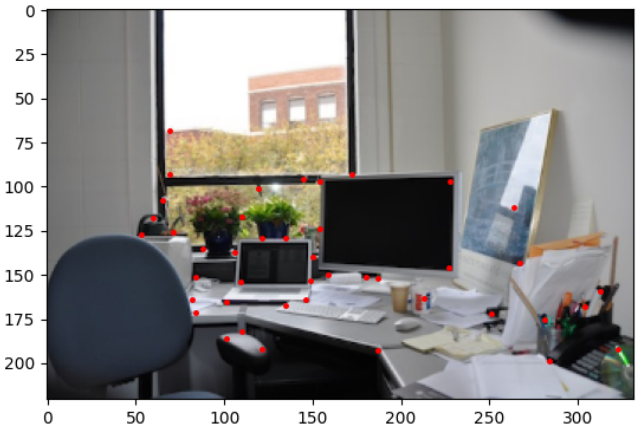
Now we have all we need to compute the C matrix, considering neighborhoods of 7x7. However, instead of computing the eigenvalues, it is far more efficient to use the Harris R function instead.

Finally, I extract the local maximums in R that are above a threshold (non-maximum supression), which will be the coordinates of the desired features.

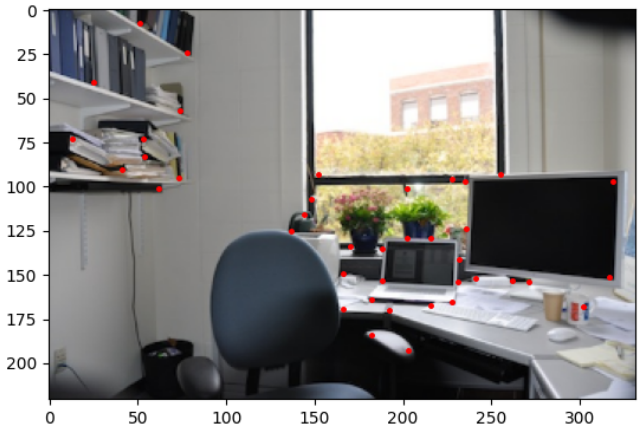
I wanted the threshold value to depend on the values in the image rather than on some “magic number”. What worked for me was considering the sum of the mean and the standard deviations of the matrix with all the R values. I also considered using the maximum value of R as a reference to compute the threshold, but I thought that a big outlier could impact the measure really badly.

Results

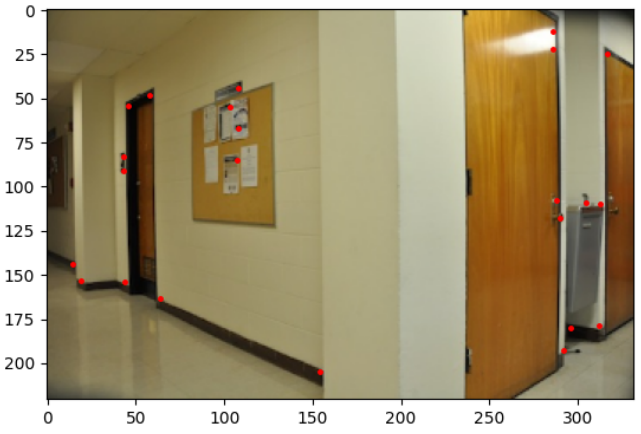
Features in DanaOffice/DSC_0308



Features in DanaOffice/DSC_0309

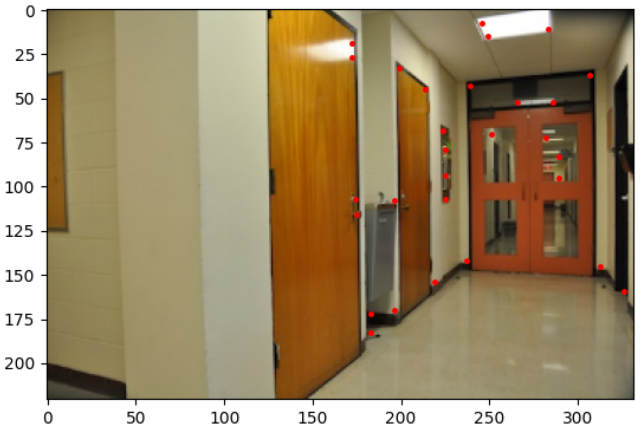


Features in DanaOffice/DSC_0310



Features in DanaHallWay1/DSC_0281

Features in DanaHallWay1/DSC_0282



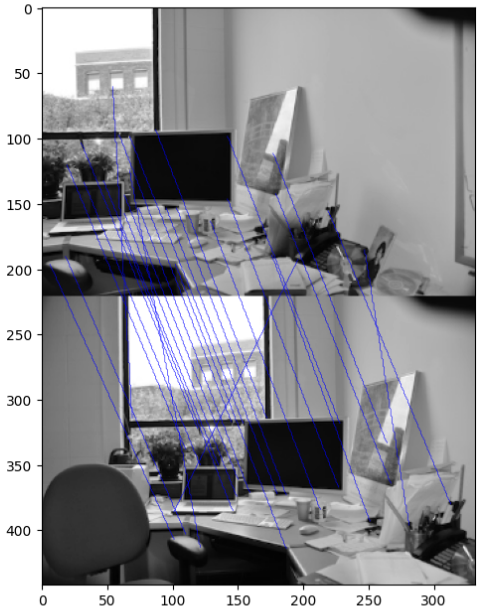
Features in DanaHallWay1/DSC_0283

Find correspondences: NCC

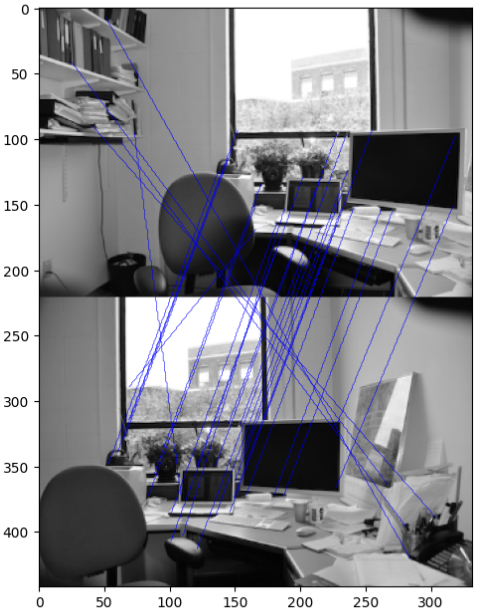
For each feature, I consider a centered patch around it and compare it against all the other patches of the corresponding features in the other image. The comparison is done by computing the Normalized Cross Correlation: the bigger the NCC (above a threshold that I set at 0.5) the more similar those patches are.

We end up with a bunch of good matches, but there can be points that belong to more than one good match. That's why in the end I remove duplicates by comparing NCCs and choosing the best ones for each duplicate.

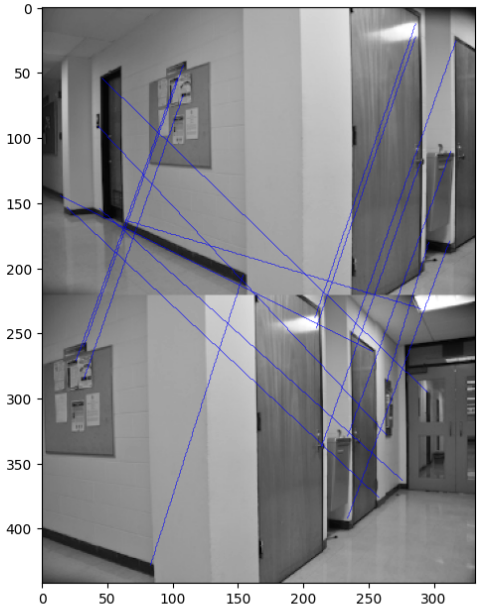
Results



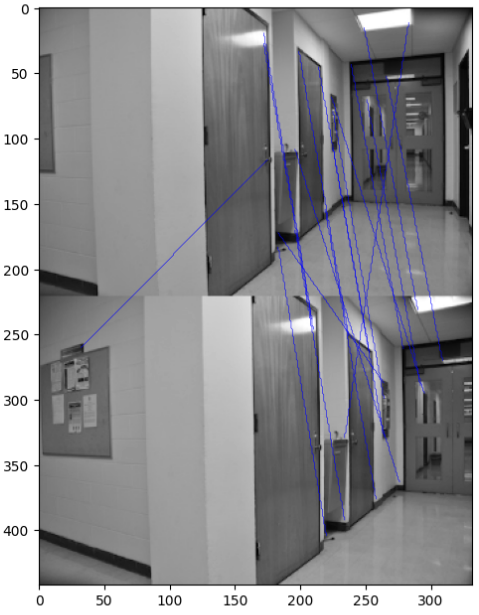
DanaOffice, img1 and img2
27 initial correspondances



DanaOffice, img3 and img2
31 initial correspondances



DanaHallWay1, img1 and img2
17 initial correspondences



DanaHallWay1, img3 and img2
18 initial correspondences

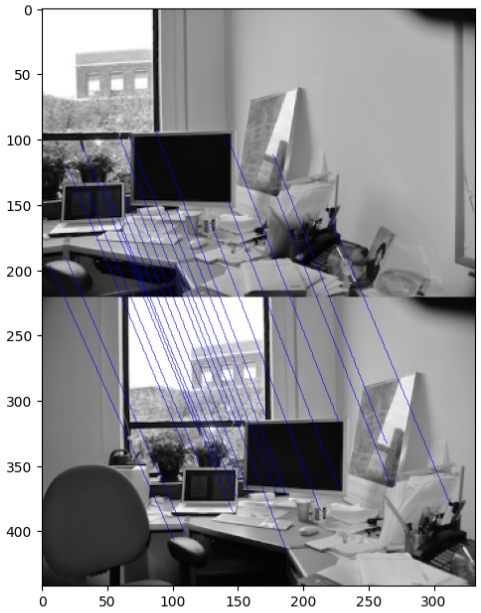
Estimate the homography using RANSAC

Given a list of good matches between features from the previous step, the goal here is to compute a homography that transforms the coordinates of the features from one of the images to the other one. Finding the matrix of the transformation is just a matter of setting a system of equations, writing it in matrix form as a homogeneous system, and using the SVD decomposition to obtain the result.

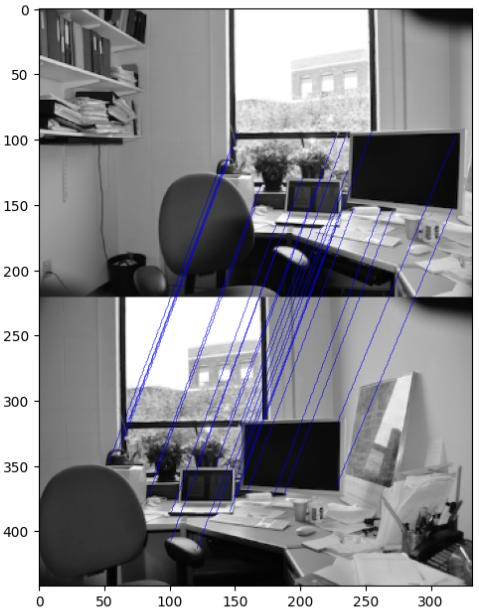
However, there could be outliers in this list, which could have a very negative impact on the transformation, and that's why we use RANSAC to ignore those when computing the homography. On each iteration, we take 4 points, which is the minimum to determine an homography in 2D, compute the corresponding homography, and then check how many of the total correspondences match that homography (inliers). We iterate until we find a homography that matches a big part of the initial correspondences (in my case that's ⅛ of the total correspondences) or until the maximum number of iterations is reached. I set the maximum at 100 iterations, that should be more than enough in terms of probabilities (the iterations are very fast so it is not that important).

Finally, we compute the homography one more time with all the inliers to obtain a better calibrated homography.

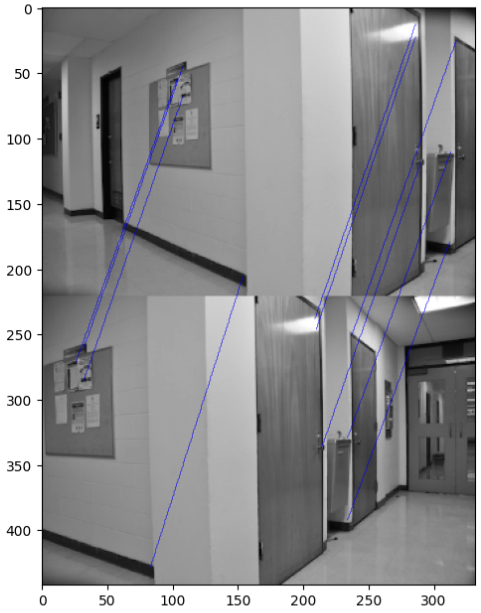
Results



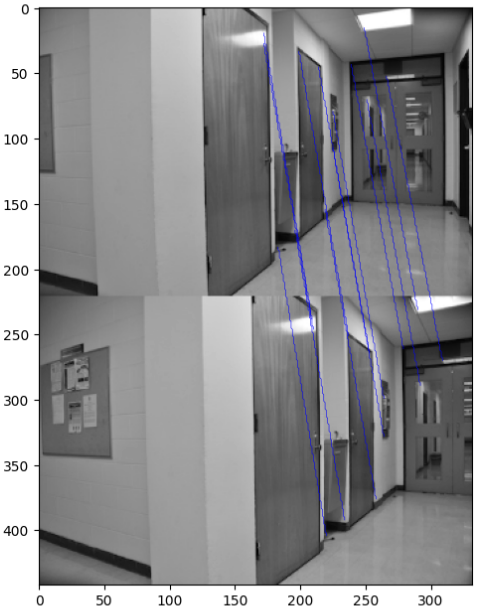
DanaOffice, $h_1 : img1 \rightarrow img2$
22 inliers



DanaOffice, $h_2 : img3 \rightarrow img2$
24 inliers



DanaHallWay1, $h_1 : img1 \rightarrow img2$
10 inliers



DanaHallWay1, $h_2 : img3 \rightarrow img2$
13 inliers

Warp images

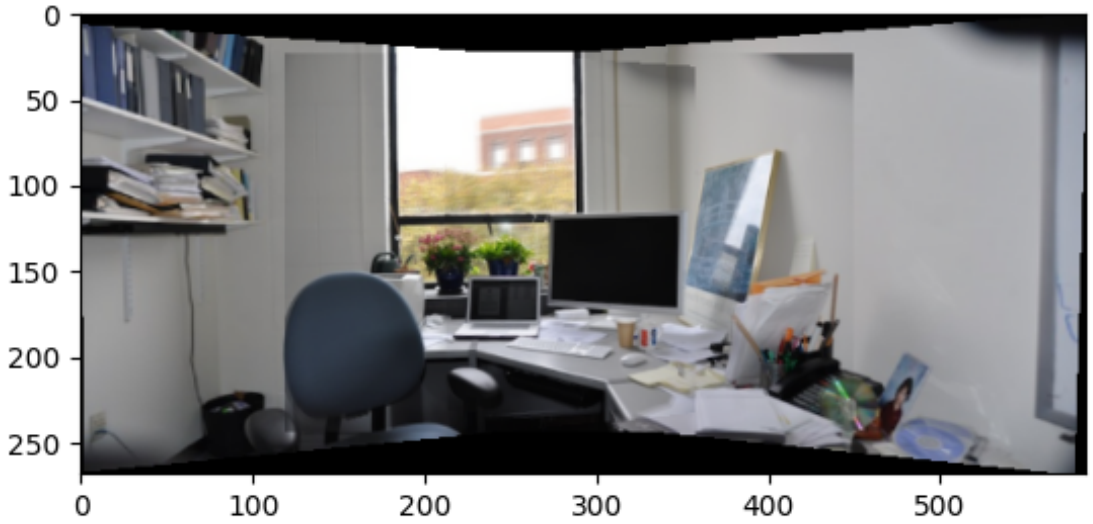
Finally, we have to build the mosaic using the homographies. As the homographies I computed transform img1 and img3 into img2, then my mosaic has to use the coordinate system of img2.

The first step is determining how big the final mosaic has to be. We know that it has to be at least as big as the second image, but we also have to express the coordinates of the corners of the frames of img1 and img3 into the same coordinate system (using the homographies). Once we know the 12 corners of the 3 images, the size is just the difference between the maximum and minimum values of rows and columns among all these 12 points. We can now draw img2 in the final mosaic directly.

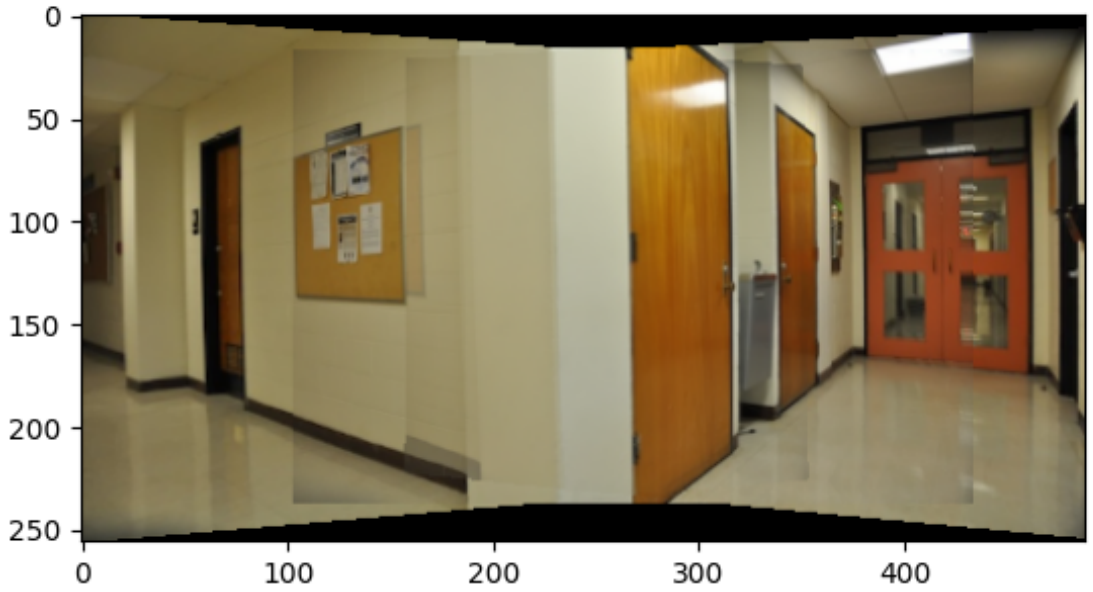
Finally, we draw img1 and img3 using the homographies. For each pixel in the mosaic, we have to:

1. Use the inverse of the homographies to express that coordinate in the coordinate systems of img1 and img3.
2. If the transformed coordinates end up outside of img1 or img3 respectively, we do not consider its color in the final mosaic (because it has no color associated).
3. If the coordinates end up inside the frames, we do linear interpolation of the corresponding image to determine the color of the pixel. We need the interpolation because the coordinate computed has continuous values, but pixel coordinates are discrete.
4. We finally do an equally weighted sum of all color values from the three images that correspond to that pixel.

Results



DanaOffice



DanaHallWay1

As you can see, this implementation has some limitations. First of all, despite doing a weighted average in the pixels with overlapping, if the changes in illumination at the edges are reasonably high then it is easy to tell where one of the images ends and the other starts (look at DanaOffice mosaic, on the right wall). Furthermore, if the original images have some imperfections like the top-right dark corner in DanaOffice, then the imperfection will be present in the final mosaic as many times as the images we use.

Finally, it is interesting to note that, although 4 points are enough, mathematically speaking, to determine the homography, in practice we need far more points to calibrate. This is because we are not working with correspondences 100% accurate, so the more we have, the better the estimation will be. This is easy to appreciate in the two solutions we have. We just have 10 and 13 inliers in DanaHallWay1 and we can see how there are some areas (like the cork board) with some misplaced pixels, while in the first mosaic, the match is much better because we are working with 22 and 24 inliers.

Conclusion

In this project we have applied plenty of different techniques to build a mosaic that, far from being perfect, I think is still impressive. We started with 3 arrays of RGB values, and we ended up with a mosaic by “just” applying pure mathematics: gradients, eigenvalues, correlation, homographies, SVD, and interpolation.

What I find also interesting, and I think my code reflects that, is that the algorithm is really modular. It is a combination of clearly separable modules, and it was really satisfying to see how improvements in one of the modules have a huge impact on the subsequent ones. Furthermore, we could easily improve the result by making improvements in each module. For example, if we substitute the Harris Corner Detector for a better one, I’m sure we would have more and better correspondences and therefore the final mosaic would be much better.