NORTHEASTERN UNIVERSITY

Computer Vision Project #2: Image Mosaicing

Jordan Helderman
Justin Hess

3 November 2017

Under certain conditions, it is possible to write a projective transformation between two image spaces. In these scenarios, it is possible to warp both images into the same image space to create an image that is the union of both images. This process is known as image mosaicing. In this project, we investigate procedures for image mosaicing. We accomplish this by detecting corner points in a image, correlating those points with corresponding points in a similar image, using those points to estimate a homography between the two image spaces, and warping the images into the same image spaces. To accomplish this, we implemented a Harris corner detection function in MATLAB and apply normalized cross correlation to find corresponding corner features. We use RANSAC to estimate the homography in a fashion robust against incorrect correspondences. Finally, we warp images using reverse image warping.

1. INTRODUCTION

Mosaicing is a procedure that takes two or more images and warps them into the same image space to form a composite image. This idea can be applied to stitch together a large number of images into a panoramic image, which is of interest to photography enthusiasts at the very least. Similar techniques can also be applied to overlay images on to surfaces in a target image. This technology is applicable photo editing applications. In this report, we will describe techniques for image mosaicing, show how these techniques can be applied to stitch together images, and qualitaively assess the results.

2. Algorithm Descripion

In this section, we will describe design of the image mosaicing algorithm. The image mosaicing algorithm considers pairs of images, finds a homography between the images, and warps the images so that output contains the union of both images. A necessary step of image mosaicing is the estimation of the projective transformation between the two images. We know from class that it is possible to estimate a projective transformation of dimension N with $N+2$ correspondences between the input and output space of the transformation. Therefore, it is possible to estimate the 2D projective transformation between the two image spaces with 4 correspondences between the images. Corners are an image feature for that has been well explored for this purpose. For that reason, we detect corner features with a well-known detector called the "Harris Corner Detector". The detected corners are compared across images using normalized cross-correlation to detect correspondences between the images. These correspondences likely contain incorrect relations. Consequently, we use RANSAC to estimate the homograpy defined by these correspondences in a manner robust to outliers (.i.e. correspondence mistakes). With the homography estimated, we then use reverse image warping to project the images into the same space. A pseudocode description of this algorithm can be found in algorithm 1. In the following sub-sections, we will discuss the design of each sub-algorithm in detail. MATLAB code for this algorithm can be found in the appendix.

Algorithm 1: Mosaic Algorithm

input : Two Images I_1 and I_2
output: An image, I_{out}, that is the composite of I_1 and I_2

```
// detect corrners and find correspondences
```
1 $C_1 \leftarrow harris(I_1)$;
2 $C_2 \leftarrow harris(I_2)$;
3 $CORR \leftarrow ncc(I_1, C_1, I_2, C_2)$;

```
// estimate homography and warp I2 into I1
```
4 $H \leftarrow homography(CORR)$;
5 $I_{out} \leftarrow warp\_img(I_2, H, I_1)$;
6 **return** I_{out};

2.1. Harris Corner Detection

The Harris function is used to calculate corner points in an input image. This is done first by first converting the image to grayscale, and then computing the image gradients in the horizontal and vertical axes. This is done by smoothing the grayscale image with a gaussian filter in both the x and y directions using the MATLAB function conv2(). This is a gaussian filter with a standard devaiation of 1. One of the parameter inputs to the harris function is the sigma value used for this gaussian filter, usually with a value of 4.5. After we smooth the image, we then apply the convolution of an edge detector filter in both the x and y directions to compute their respective gradients.

```
hx = [1, 0, -1];
hy = hx';
```

```
% compute the gradient/derivative with respect to x
Ix = conv2(I_smoothed, hx, 'same');

% compute the gradient with respect to x at the border pixels
Ix(:,1)   = 2*(I_smoothed(:,2)  -I_smoothed(:,1));
Ix(:,end) = 2*(I_smoothed(:,end)-I_smoothed(:,end-1));

% compute the gradient with respect to y
Iy = conv2(I_smoothed, hy, 'same');

% compute the gradient with respect to y at the border pixels
Iy(1,:)   = 2*(I_smoothed(2,:)  -I_smoothed(1,:));
Iy(end,:) = 2*(I_smoothed(end,:)-I_smoothed(end-1,:));
```

Code Figure 1: Edge Detection Filter, X and Y Gradients

Once we have the gradients in the x and y directions, we can now compute the quadratic terms in order to make the calculation of the corner matrix simpler

```
% Precompute quadratic terms of the derivatives.
Ix2 = Ix.^2;
Iy2 = Iy.^2;
Ixy = Ix.*Iy;
```

Code Figure 2: Gradient Quadratic Terms

Now we can calculate the determinant and the trace of the Corner Matrix. We use the constant k, the sensitivity factor of the response function, and set it to 0.04. Generally it is kept pretty low, between 0.04 and 0.06. After the determinant and the trace of the corner matrix are calculated, we can plug them in with the sensitivity factor into the equation to get R, the response measurement of the corner matrix.

```
% Set sensitivity factor of the response function,
% used to detect sharp corners (usually 0.04 <= k <= 0.06)

k = 0.04;

% Corner response, r
% calculate the determinant and trace of C matrix elements
% and then calculate r

detC = (Ix2.*Iy2 - Ixy.^2);
traceC = Ix2 + Iy2;
r = detC - k * traceC.^2;
```

Code Figure 3: Corner Response Function Calculation

We then can threshold the output using our input parameter of the harris function and non maximal suppression. We chose 27500 for the threshold as this value seemed to be appropriate for enough corner point responses. To apply the non maximal suppression, we loop over all pixels in a 3x3 neighborhood in order to find the local maxima peaks in the image. All other pixels that do not pass the threshold are set to 0.

```
r_thresholded = r .* (r>threshold);

r_final = r_thresholded;

%% Non maximal suppression code

% loop over non-zero pixels that passed the threshold
% pixels that don't pass threshold are set to 0
[idx, idy] = find(r_thresholded>0);

for k = 1:length(idx)
  i_x = idx(k);
  i_y = idy(k);
  % compare to maximum of the 8-pixel neighborhood
  if r_thresholded(i_x, i_y) ~= max(max(r_thresholded(i_x-1 : i_x+1,...
          i_y - 1 : i_y+1)))
  % use -1 here because of the padding
  r_final(i_x-1,i_y-1) = 0;
  end
end
```

Code Figure 4: Non-maximal Suppression

2.2. Correspondence Detection

Now that we have the two corner matrices from the two input images that we are attempting to warp together, we can estimate the correlation points between the two image sets' corner points. To do this, we loop through each row for the first two columns of the first set of corner points, and for every point we loop through the rows in the first two columns of the second corner point set.

We first and foremost want to ignore the border pixels before further calculations. Once that is done, we create a patch, or region, around the corner point from the first corner point set. We initialize our matrix of normalized cross correlation results for later, and for each patch from the first corner set we go through every row for the first two columns of the second set of corner points. We do the same thing for the second set of corner points as the first set; the border pixels are ignored and we create a patch/region around the respective corner points. The key part of the algorithm is calling the MATLAB function, normxcoor2(), which creates

an output matrix of normalized correlation coefficients between the two corner points by calculating a 2D normalized cross correlation between the corner points respective patches. Once this is computed, we obtain the center value of the output matrix for correlation values between the two patches. We then find the peak or local maxima in the cross correlation by finding the max value of the normalized cross correlation matrix and setting its corner index point. Finally, we set a threshold at 0.90, and if the index peak point passes the threshold, we store the two cross correlated points as a matrix used to later input in our homography function.

2.3. Homography Estimation

A homography from one image to another image taken with a camera that has only been rotated can, in general, be represented with the following system of equations.

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

From class, we know that this relation can be used to write the following system of homogeneous equations given a set of N points.

$$A \cdot h = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' & -x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' & -y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' & -x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' & -y_2' \\ & & & & \cdot & & & & \\ & & & & \cdot & & & & \\ & & & & \cdot & & & & \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = 0$$

With this system, we can find the entries of the homography matrix by performing eigenvalue decomposition and selecting the eigenvector corresponding to the smallest eigenvalue. This procedure is described in pseudocode in algoithm 2.

Algorithm 2: Homography Estimation Algorithm

input : A list of correspondences C
output : A homography H

```
// define a system of homogeneous equations
```
$N \leftarrow$ number of correspondences;
for $n \leftarrow 0 : N-1$ **do**
    $(x_1, y_1, x_2, y_2) \leftarrow C(n)$;
    $A(2n,:) \leftarrow [x, y, 1, 0, 0, 0, -x * x', -y * x', -x']$;
    $A(2n+1,:) \leftarrow [0, 0, 0, x, y, 1, -x * y', -y * y', -y']$;
end
```
// perform singular value decomposition
```
$h \leftarrow$ the eigenvector corresponding to the smallest eigenvalue;
$H \leftarrow h$ reshaped appropriately to form the homography matrix;
return H;

As mentioned previously, the corner detection and association procedure can produce erroneous associations. These erroneous associations can corrupt the homography estimate produced by this algorithm. To make the estimation robust to these erroneous correspondences, we apply the RANSAC procedure. In the context of homography estimation, RANSAC dictates that we select the minimum points to estimate a homography, which is four; estimate the homography; check how many points are consistent with the homography; check to see if the number of consistent points exceeds some preset termination condition; and repeat until the condition is met. This algorithm is outlined in the pseudocode in algorithm 3.

Algorithm 3: Homography RANSAC Algorithm

input : A list of correspondences C
output : A homography H

while *iteration less than max tries* **do**
    choose 4 points at random;
    estimate a homography;
    find the number of points that fit the homography;
    if *fit is good enough* **then**
        return a homography fitted to the inlier points;
    end
end
return a homography fitted to the largest set of inliers found;

2.4. IMAGE WARPING

Once the homography is found, we form the mosaic by warping the images into the same image space. We accomplish this via reverse image warping. For a given image, this involves using the homography to transform the points from the target image space to the given image space and sampling that image at those points, using bilinear (or some other) interpolation scheme to evaluate the image at non-integer image indices and adding these values to the

target image. Additionally, it is necessary to blend the output image at pixels that are sampled from more than one image. There are a number of methods for this, but the method we will use is an even-weighted average of the pixel intensities.

2.5. Image Overlay

It is possible to overlay an image on a source image via a similar method. To do the overlay, we specify the homography marking the four points of the quadrilateral we wish to warp the image onto; relate them the four corners of the image we want to warp; estimating the homography; and warping the image into the source image space with this homography. We then just replace the pixels in the target quadrilateral with the warped image instead of blending like we would do in the image mosaicing case.

3. Experimental Results and Discussion

To test this algorithm, we applied the algorithm to a set of test images and qualitatively evaluated the results. In the following sections, we document the results of these tests and discuss the performance of the algorithms on these test data.

3.1. Mosaicing Results

To begin, we will examine the processing flow on a pair of images. The processing begins by reading the images (figure 3.1). These images are then passed through the harris corner detector. The corner response function is in figures 3.2-3, and the detected corners are shown in figures 3.6-7. These corners are associated with each other to form correspondences (figure 3.8). Subsequently, these correspondences are used to estimate a homography between the images and the second image is warped back into the image space of the first. Finally, the images are combined and blended to form the final mosaic. The second image warped by itself, the unblended mosaic, and the final blended mosaic can be seen in figures 3.9-11, respectively.

During the design of the harris corner detection function, it became very clear several features would need to be included other than the main features of calculating the gradients and corner response function. Among these are choices of edge detection filters and smoothing convolutions using gaussian filters.

Several different edge detector filters were also tested to find the optimal corner point outputs from an image. Originally it was proposed to perform the convolution of a 3 x 3 prewitt edge detector, but it was found less realistic corner points were observed as more weight seemed to be given to the more centered pixels within the mask upon convolution (code figure 5). Then a sobel edge detector mask was used, but it ultimately was found a simple 1 x 3 sobel edge detector gave the most optimal results (code Figure 6). This was determined comparing the output corner points with the 1 x 3 filter with the human eye, and seeing it correctly picked up more points accurately as corners than when using the 3 x 3 edge detector filter.

```
% Compute image derivatives by convolution
% in both x and y directions
```

```
hx = [1, 0, -1; 1, 0, -1; 1, 0, -1];
hy = [1, 0, -1; 1, 0, -1; 1, 0, -1].';
```

Code Figure 5: Original Edge Detector Filters

```
% Compute image derivatives by convolution with kernel
% [1 0 -1] in both x and y directions

hx = [1 0 -1];
hy = hx';
```

Code Figure 6: Final Edge Detector Filters

It became pertinent to apply a smoothing gaussian filter on the input image to suppress noise, and it was also found with testing of corner points that another gaussian filter was needed to smooth the gradient values of the images. It was quickly determined that one of the parameter inputs into our harris function was a sigma value, representing the standard deviation of the gaussian filter for smoothing the gradient/derivative outputs. The gaussian filter for smoothing of the input image was less important and used with a default standard deviation of 1, represented by the variable 'sig'. Once the quadratic gradient terms were computed for the corner matrix, we applied a gaussian smoothed gradient smoothing with our input standard deviation, sigma. Using an input sigma of 4.5, and a value of 27500 for the corner response threshold, it was observed that the harris corner outputs were the most accurate compared to when the gaussian filter was not used to smooth the quadratic gradient values.

```
% We then smooth the derivatives/gradients with another gaussian filter
% with a standard deviation of the second parameter input, sigma

gaussian_gradient_filter = exp(-(x).^2/(2*sigma^2)) / (sigma*sqrt(2*pi));

Ix2 = conv2(gaussian_gradient_filter, gaussian_gradient_filter', Ix2, 'same');
Iy2 = conv2(gaussian_gradient_filter, gaussian_gradient_filter', Iy2, 'same');
Ixy = conv2(gaussian_gradient_filter, gaussian_gradient_filter', Ixy, 'same');
```

Code Figure 7: Gaussian Filter Smoothing of Quadratic Gradient Terms

In figures 3.4-5 is the output of the thresholded corner response output. This is after the quadratic gradients are smoothed, as shown above in code figure. This was a pivotal step once the quadratic gradient terms were smoothed and we were able to calculate the corner response function. After we thresholded the corner response values, we applied non maximal suppression and those were the final outputs of the harris function (see figures 3.6-7).

Figure 3.1: Original Images

Figure 3.2: Corner Response for Image #1

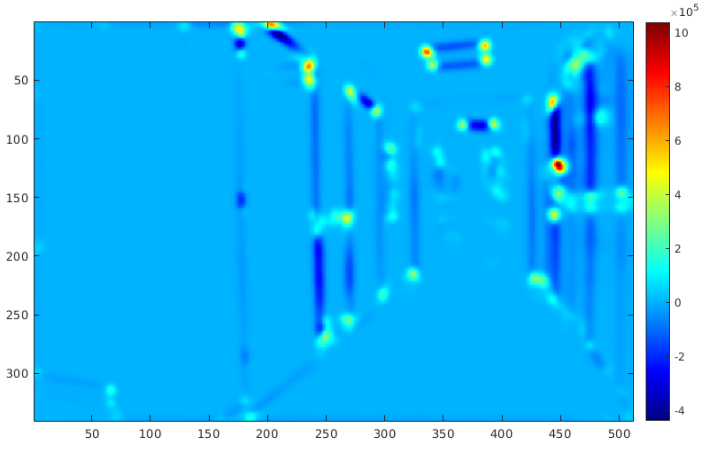


Figure 3.3: Corner Response for Image #2

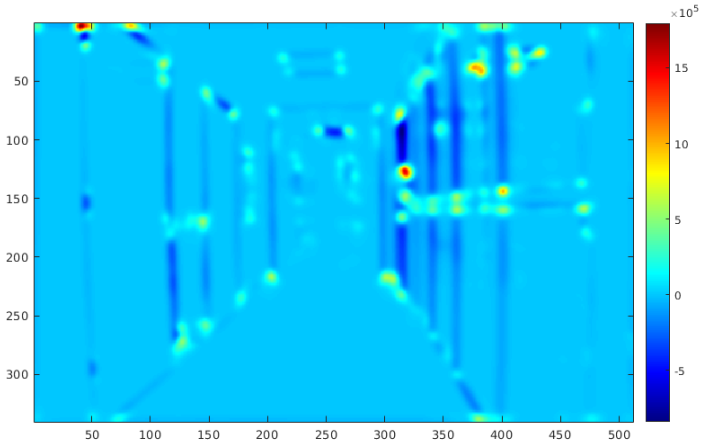


Figure 3.4: Thresholded Corner Response for Image #1

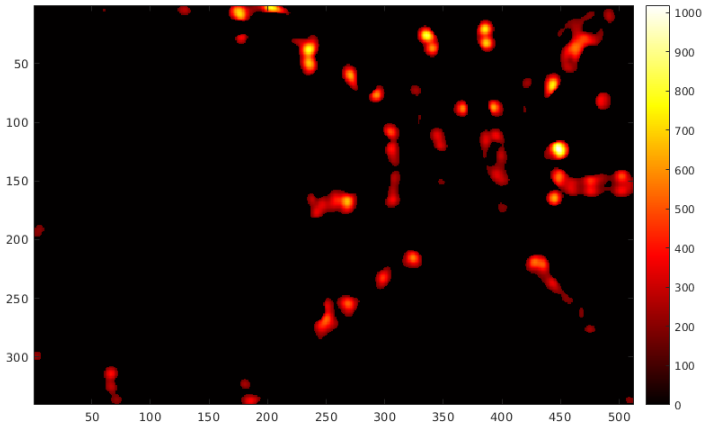


Figure 3.5: Thresholded Corner Response for Image #2

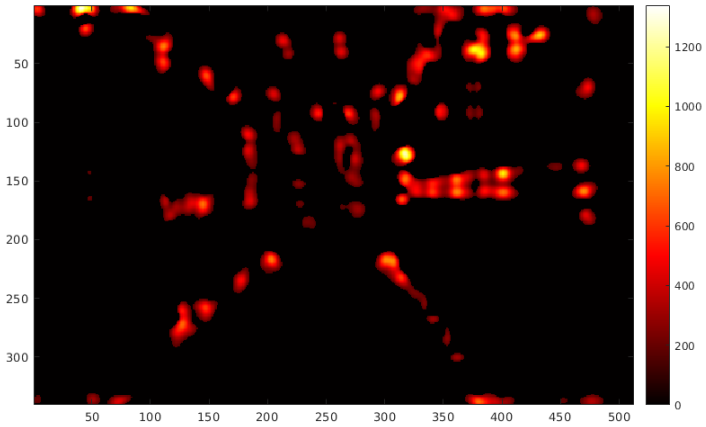


Figure 3.6: Detected Corners for Image #1

Figure 3.7: Detected Corners for Image #2

Figure 3.8: Corner Correspondences

Figure 3.9: Image #2 Warped

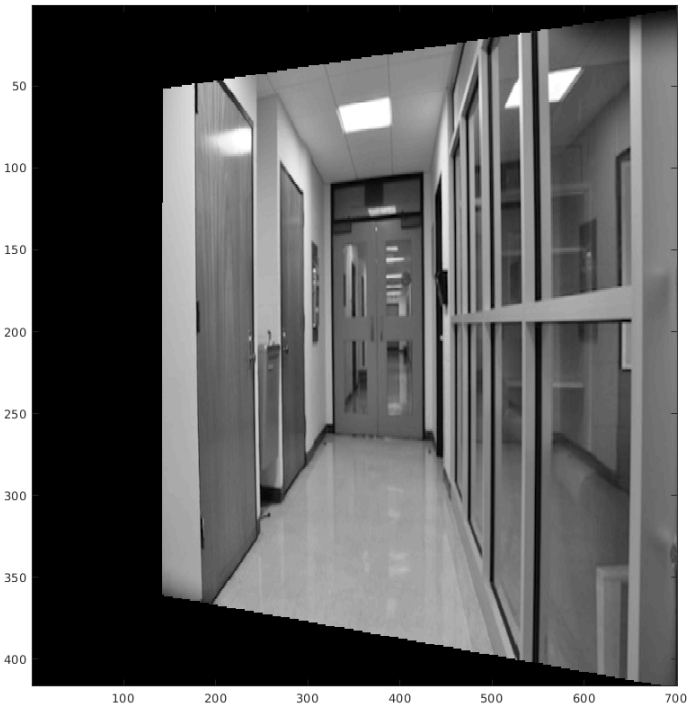


Figure 3.10: Unblended Mosaic

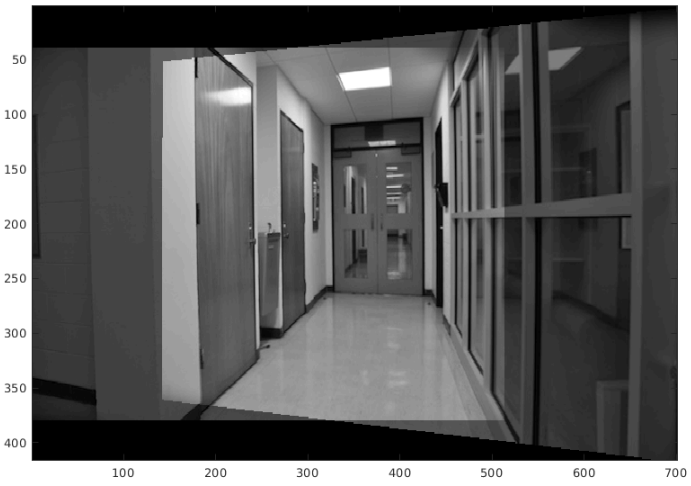


Figure 3.11: Blended Mosaic

We can see from the results in figure 3.11 that the final result is a sensible mosaic of the two images. There are no visible artifacts indicating that the image was warped incorrectly (ghosting artifacts), which indicates that the homography was very close to the actual homography between the images. One area where the mosaic could be improved is in the blend of overlapping edges. We will further discuss this artifact in one of the following sections.

3.2. Image Overlay Results

We also tested the algorithm we implemented for performing image overlay. For this algorithm, we start with two images, one target image and one image we will overlay. These images can be seen in figures 3.12-13. Then, we have the user mark the four correspondences that define the quadrilateral that we wish to warp the image onto. Finally, we warp the image into the marked quadrilateral. The resulting image is shown in figure 3.14.

From the final overlaid image, we can see that the image of Belichick is overlaid on the bulletin board in the right side of the image, which is just as was intended. This indicates that our homography estimation and warping were successful.

3.3. Anomalous Behavior

1. Blending Artifacts

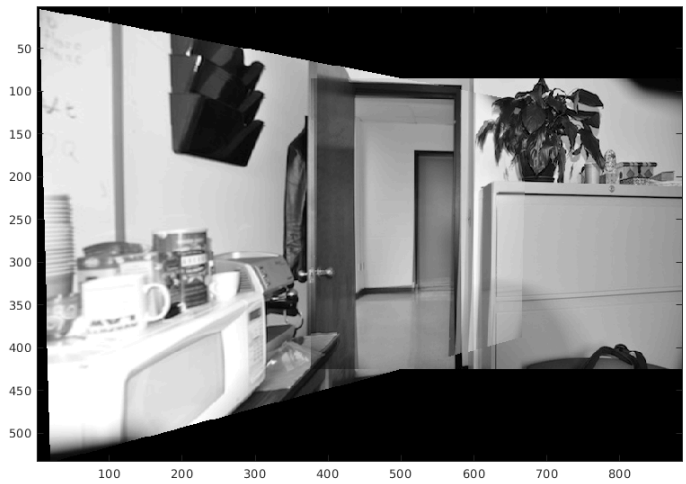
   At times, it was possible to see the edges of the images on the boundary between the shared and unshared pixels in the resulting mosaic. An example of this can be seen in the blended mosaic we presented earlier (figure 3.11). This artifact is likely due to the simplicity of our blending scheme, as it was just a simple average between the two images. We could likely mitigate this artifact by using a more complex blending scheme,

Figure 3.12: Original Image for Image Overlay

Figure 3.13: Bill Belichick

Figure 3.14: Overlaid Image

Figure 3.15: Intensity Difference Artifact



taking into account the distance of the blended pixel from the image edge and/or taking into account the intensity statistics of each image.

2. Differences in Image Intensity

   In cases where the images had differing levels of brightness, the edge of the warped image was more noticable in the mosaic. An example of this can be seen in figure 3.15. One underlying assumption of the blending scheme we used is that regions will have the same intensity in both images. Since this assumption is violated in this case, we get an artifact. This behavior could be mitigated by using a more sophisticated blending scheme that takes into account the intensity statistics of the image.

3. Corner Shadowing

   There was often a noticable decrease in the intensity of the pixels in each image around the corners of the image. This manifested itself in the image mosaic as a noticable edge near the corners on the boundary between the shared and unshared pixels. An example of this behavior can also be seen in the original blended mosaic (figure 3.11). This artifact is similar to the previous artifact, but instead of the brightness varying globally, it varies on a local scale. This artifact can likely be mitigated by taking into account local image intensity statistics. However, it is possible that this artifact in the original images has to do with the hardware used to capture the images. If this is the case, it may be possible to calibrate for this effect and correct it instead of trying to estimate it during the mosaicing, which may yield better results.

4. Conclusion

In this report, we presented a design for a image mosaicing algorithm; we tested this algorithm on a benchmark set of images; and we highlighted some of the challenges of designing such an algorithm. In future work on this topic, we could explore alternate blending techniques in order to address some of the anomalous behavior we observed or, in general, alternative features or homography estimation methods.

Appendix A Mosaic Code

Main Script:

```
clc; close all; clear;
%% parameters
base_path = '/home/jhelderman/Documents/school/computer-vision/projects/project-2/data/'
path1 = [base_path, 'DanaHallWay2/DSC_0286.JPG'];
path2 = [base_path, 'DanaHallWay2/DSC_0287.JPG'];
tau = 100;
max_iter = 100;
alpha = 0.9;

%% read in 2 images
hallway_original1 = imread(path1);
hallway_original2 = imread(path2);

I1 = rgb2gray(hallway_original1);
I2 = rgb2gray(hallway_original2);

%% apply Harris corner detection
c1 = harris(I1, 4.5, 27500);
c2 = harris(I2, 4.5, 27500);

%% show detected corners
figure;
imshow([hallway_original1, hallway_original2]);
figure
imshow(hallway_original1)
hold on
plot(c1(:,2),c1(:,1),'r*')
hold off
figure
imshow(hallway_original2)
hold on
plot(c2(:,2),c2(:,1),'r*')
```

```
hold off

%% Normalized Cross Correlation
corrs = find_correspondences(I1, c1, I2, c2);

%% plot normalized cross correlation
figure
imshow([hallway_original1;hallway_original2])
hold all
plot(c1(:,2), c1(:,1), 'rx');
plot(c2(:,2), c2(:,1) + size(I1,1), 'bx');
for i = 1:size(corrs,1)
    plot([corrs(i,1), corrs(i,3)], ...
        [corrs(i,2), corrs(i,4)+size(I1,1)], 'g');
end
hold off

%% find a homography
H = homography_ransac(corrs, tau, max_iter, alpha);
disp('Homography:');
disp(H);

%% warp the images
I1 = double(I1);
I2 = double(I2);
Iwarp = warp_img(I2, H, I1);
figure;
imagesc(Iwarp);
colormap('gray');
```

Harris Corner Detection Function:

```
function [corners] = harris(I, sigma, threshold)
warning('off','all')
% Usage: corners = harris(I, sigma, threshold)
% Inputs:
%   I   [m x n]   Input image to be processed
%   sigma     Standard deviation of the Gaussian filter
%               used for smoothing the derivatives
%   threshold  Corner response function threshold. Values of the corner
%               response function below this threshold are neglected
% Outputs:
%   corners  [n x 3]  Each row of 3-column matrix represents
%     one corner point. The first two columns describe the position
%     of the corner in im_inp (first column = y , second column = x).
```

```
%     The third column contains the value of the corner response function.

% Conversion of the limiting variables to doubles
sigma = double(sigma);

% Convert im to double precision
I = double(I);

image(I);

% Firstly, the image is smoothed with a simple gaussian filter

sig = 1;
window1 = floor(sig*3);
x = -window1 : 1 : window1;
gaussian1DFilter = exp(-(x).^2 / (2*sig^2)) / (sig*sqrt(2*pi));

% smoothed image
I_smoothed = conv2(gaussian1DFilter, gaussian1DFilter', I, 'same' );

% Compute image derivatives by convolution with kernel
% [1 0 -1] in both x and y directions

hx = [1 0 -1];
hy = hx';
%hx = [1, 0, -1; 1, 0, -1; 1, 0, -1];
%hy = [1, 0, -1; 1, 0, -1; 1, 0, -1].';

% compute the gradient/derivative with respect to x
Ix = conv2(I_smoothed, hx, 'same');

% compute the gradient with respect to x at the border pixels
Ix(:,1)   = 2*(I_smoothed(:,2)  -I_smoothed(:,1));
Ix(:,end) = 2*(I_smoothed(:,end)-I_smoothed(:,end-1));

% compute the gradient with respect to y
Iy = conv2(I_smoothed, hy, 'same');

% compute the gradient with respect to y at the border pixels
Iy(1,:)   = 2*(I_smoothed(2,:)  -I_smoothed(1,:));
Iy(end,:) = 2*(I_smoothed(end,:)-I_smoothed(end-1,:));

% Precompute quadratic terms of the derivatives.
Ix2 = Ix.^2;
```

```
Iy2 = Iy.^2;
Ixy = Ix.*Iy;

% We then smooth the derivatives/gradients with another gaussian filter
% with a standard deviation of the second parameter input, sigma

window2 = floor(3*sigma);
x = -window2:1:window2;
gaussian_gradient_filter = exp(-(x).^2/(2*sigma^2)) / (sigma*sqrt(2*pi));

Ix2 = conv2( gaussian_gradient_filter, gaussian_gradient_filter', Ix2, 'same');
Iy2 = conv2( gaussian_gradient_filter, gaussian_gradient_filter', Iy2, 'same');
Ixy = conv2( gaussian_gradient_filter, gaussian_gradient_filter', Ixy, 'same');

% Set sensitivity factor of the response function,
% used to detect sharp corners (usually 0.04 <= k <= 0.06)

k = 0.04;

% Corner response, r
% calculate the determinant and trace of C matrix elements
% and then calculate r

detC = (Ix2.*Iy2 - Ixy.^2);
traceC = Ix2 + Iy2;
r = detC - k * traceC.^2;

figure(3)
imagesc(r), colormap(hot(256)), axis image, colorbar,
title('Corner Response values')

r_thresholded = r .* (r>threshold);

figure(4)
imagesc(r_thresholded.^(1/2)), colormap(hot(256)), axis image, colorbar,
title('Thresholded Corner Response Output');

%  Non-maximal suppression
%  loop over corner points and only keep those where the corner response
%  is a peak/local maximum in a 3 x 3 window. All other pixels
%  are set to zero if they do not pass the threshold.

r_final = r_thresholded;
```

```
% create a temporary pad and threshold
pad = zeros( size(r_thresholded) + [2 2] );
pad( 2: end-1, 2 : end-1) = r_thresholded;

r_thresholded = pad;

% loop over non-zero pixels that passed the threshold
% pixels that don't pass threshold are set to 0
[idx, idy] = find(r_thresholded>0);

for k = 1:length(idx)
  i_x = idx(k);
  i_y = idy(k);
  % compare to maximum of the 8-pixel neighborhood
  if r_thresholded(i_x, i_y) ~= max(max(r_thresholded(i_x-1 : i_x+1,...
          i_y - 1 : i_y+1)))
  % use -1 here because of the padding
  r_final(i_x-1,i_y-1) = 0;
  end
end

%  The output corner points
%  coordinate values x and y and the
%  value of the corner response function
% is the 3rd value in the matrix
[cornersx, cornersy, response] = find(r_out);

corners = [cornersx cornersy response];

return;

end
```

Normalized Cross Correlation Correspondence Function:

```
function [corrs] = find_correspondences(I1, C1, I2, C2)
% corrs are the correlation points
corrs = nan(size(C1, 1), 4);
for i = 1:size(C1, 1)

    y1 = C1(i, 1);
    x1 = C1(i, 2);
    % p_size = patch size, patch length = 2*patch_size, radius of patch
    p_size = 10;
```

```
% ignore border corners
if x1 <= p_size || y1 <= p_size || x1 >= size(I1,2)-p_size || y1 >= size(I1,1)-p_siz
    continue
end

%creates patch around each corner for both images
%p1 is patch 1
p1 = I1(y1-p_size : y1+p_size,x1-p_size : x1+p_size);

% initialize NCC to a matrix of zeroes
ncc = zeros(1,size(C2,1));

for j = 1:size(C2,1)
    y2 = C2(j, 1);
    x2 = C2(j, 2);

    if x2 <= p_size || y2 <= p_size || x2 >= size(I2,2)-p_size || y2 >= size(I2,1)-p
        continue
    end

    % p2 is patch 2
    p2 = I2(y2-p_size : y2+p_size,x2-p_size : x2+p_size);

    % correlation coefficients in a matrix obtained by
    % 2D normalized cross correlation**
    corr_coeffs = normxcorr2(p1, p2);

    % obtain center value of ncc matrix for correlation value between
    % 2 patches
    ncc(j) = corr_coeffs(1+2*p_size, 1+2*p_size);

    % find the peak in the cross correlation
    if ncc(j) == max(ncc)
        c_index = j;
    end
end

% thresholding
ncc_threshold = 0.90;
if ncc(c_index) > ncc_threshold
    % set the correlation points after if the peak cross correlation
    % passes the threshold
    corrs(i,:) = [x1 y1 C2(c_index,2) C2(c_index,1)];
end
```

```
end
region = not(any(isnan(corrs), 2));
corrs = corrs(region, :);
end
```

Function for Applying a Homography to a Set of Points:

```
function Pout = apply_homography(P, H)
Pout = (H * P.').';
Pout = Pout ./ repmat(Pout(:,3), 1, 3);
end
```

Homography Estimation through Singular Value Decomposition:

```
function H = homography(P)
%% normalize
P1 = [P(:, 1:2), ones(size(P,1), 1)];
H1 = normalization_transform(P1);
P1 = (H1 * P1.').';
P2 = [P(:, 3:4), ones(size(P,1), 1)];
H2 = normalization_transform(P2);
P2 = (H2 * P2.').';
P = [P1(:,1:2), P2(:,1:2)];

%% form the system of linear equations
A = zeros(2 * size(P, 1), 9);
for n = 0:size(P, 1)-1
    %% extract the points
    x1 = P(n + 1, 1);
    y1 = P(n + 1, 2);
    x2 = P(n + 1, 3);
    y2 = P(n + 1, 4);
    %% A matrix entries
    A(2*n + 1, :) = [x1, y1, 1, 0, 0, 0, -x1*x2, -y1*x2, -x2];
    A(2*n + 2, :) = [0, 0, 0, x1, y1, 1, -x1*y2, -y1*y2, -y2];
end

%% solve
[V, D] = eig(A.' * A);
h = V(:,argmin(diag(D)));
h = h / sqrt(h.' * h);

%% determine the transformation matrix
H = reshape(h, 3, 3).';

%% unnormalize
```

```
H = H2^-1 * H * H1;

%% set bottom corner to 1
H = H / H(3,3);
end
```

Function for Measuring the Fit of a Homography on a Set of Correspondences:

```
function N = homography_fit(P, H, tau)
%% pass points through homography
p1 = [P(:,1:2), ones(size(P,1), 1)];
p2hat = apply_homography(p1, H);
p2hat = p2hat(:,1:2);

%% compare to found correspondences
p2 = P(:,3:4);
err = sum((p2 - p2hat).^2, 2);
N = err < tau;
end
```

Robust Estimation of Homography through RANSAC:

```
function H = homography_ransac(P, tau, max_iter, alpha)
%% begin the RANSAC iterations
best_match = 0;
good_enough = @(N) N > alpha * size(P,1);
for n = 1:max_iter
    %% pick points
    idx = randperm(size(P,1), 4);
    Phat = P(idx, :);
    %% find a homography
    Hhat = homography(Phat);

    %% check the number of matching points
    fit = homography_fit(P, Hhat, tau);
    Nhat = sum(fit);

    %% check for termination conditions
    if good_enough(Nhat)
        H = homography(P(fit, :));
        return
    end

    %% update best match
    if Nhat > best_match
        best_match = Nhat;
```

```
            best_idx = fit;
        end
    end
    if best_match == 0
        error('No match found. tau or alpha may be too restrictive.');
    end
    H = homography(P(best_idx,:));
end
```

Function for Warping One Image into Another Image Space:

```
function Iout = warp_img(I, H, Iref)
%% handle exceptional cases for the reference image
if nargin == 2
    Iref = [];
end
if numel(Iref) == 0
    Iref = zeros(2);
end

%% get points for the images
[y, x] = ndgrid(0:size(I,1)-1, 0:size(I,2)-1);
P = [x(:), y(:), ones(numel(x), 1)];
[yref, xref] = ndgrid(0:size(Iref,1)-1, 0:size(Iref,2)-1);
Pref = [xref(:), yref(:), ones(numel(xref), 1)];

%% transform points from the image
Hinv = H ^ -1;
Phat = apply_homography(P, Hinv);

%% find the size for the output image
min_vals = min(Phat(:,1:2),[],1);
min_vals = min([Pref(:,1:2); min_vals], [], 1);
min_vals = floor(min_vals);
max_vals = max(Phat(:,1:2), [], 1);
max_vals = max([Pref(:,1:2); max_vals], [], 1);
max_vals = ceil(max_vals);
out_sz = max_vals - min_vals;
out_sz = [out_sz(2), out_sz(1)];

%% define the output image and coordinate system
Iout = zeros(out_sz);
count = zeros(out_sz);
[yout, xout] = ndgrid(0:size(Iout,1)-1, 0:size(Iout,2)-1);
yout = yout + min_vals(2);
```

```
xout = xout + min_vals(1);

%% warp the reference image
Ioutref = interp2(Iref, xout, yout);
region = not(isnan(Ioutref));
count(region) = count(region) + 1;
Iout(region) = Iout(region) + Ioutref(region);

%% reverse transform the output image coordinates
Pout = [xout(:), yout(:), ones(numel(xout),1)];
Pout = apply_homography(Pout, H);
xout = reshape(Pout(:,1), size(xout));
yout = reshape(Pout(:,2), size(yout));

%% warp the main image
Ioutmain = interp2(I, xout, yout);
region = not(isnan(Ioutmain));
count(region) = count(region) + 1;
Iout(region) = Iout(region) + Ioutmain(region);
figure;
imagesc(Ioutmain);
colormap gray;

%% average
region = count ~= 0;
Iout(region) = Iout(region) ./ count(region);

end
```

Function for Conditioning Points for Homography Estimation:

```
function H = normalization_transform(P)
mu = mean(P, 1);
sigma = std(P, 0, 1);
sigma = sigma / sqrt(2);
H = [1./sigma(1), 0, -mu(1)./sigma(1); 0, 1./sigma(2), -mu(2)./sigma(2); 0, 0, 1];
end
```

Script for Performing Extra Credit Image Overlay:

```
%%
% Extra Credit
insert = imread('belichick.jpg');
hallway = imread('DanaHallWay2/DSC_0285.JPG');
figure(8)
imshow(hallway)
```

```
hold on
% input the 4 corner points where to insert, clockwise, starting from top left
[x,y] = ginput(4);
plot(x, y, 'gx')
hold off
% matrix map of background hallway image
hallway_m = [1 1;size(insert,2) 1;size(insert,2) size(insert,1);1 size(insert,1)];
% create a matrix map of the input corners
insert_m = [x y];
% find the homography matrix
[h, ~, ~] = estimateGeometricTransform(hallway_m, insert_m, 'projective');
warped_mosaic = imwarp(insert,h);
disp(insert_m);
% replace the ginput surface with insert image, and shift the ginput limits
insert_bounds = [insert_m(:,1)-min(insert_m(:,1))+1 insert_m(:,2)-min(insert_m(:,2))+1];
% create preliminary mask of zeroes for warped image
mask = zeros(ceil(fliplr(max(insert_bounds))));
for i = 1:size(mask,1)
    copy_warped = repmat(i,1,size(mask,2));
    size_warped = 1:size(mask,2);
    % check if warped image are within the bounds of insert image bounds first
    % two columns, x and y
    % populate each row of the mask
    mask(i,:) = inpolygon(copy_warped, size_warped, insert_bounds(:,2), insert_bounds(:,
end
% overlay image
coord = round(min(insert_m));%top left
for i = 1:size(mask,1)
    for j = 1:size(mask,2)
        if mask(i,j)==1
            % set the coordinates of hallway image within our bounds to the
            % inserted input image to overlay onto
            hallway(coord(2) + i-1, coord(1) + j-1, :) = warped_mosaic(i, j, :);
        end
    end
end
%show the final background hallway image with the insert picture overlayed
%on it
imshow(hallway)
hold off
```