Computer Vision Project 2

T. A. Rupprecht & Can Uner

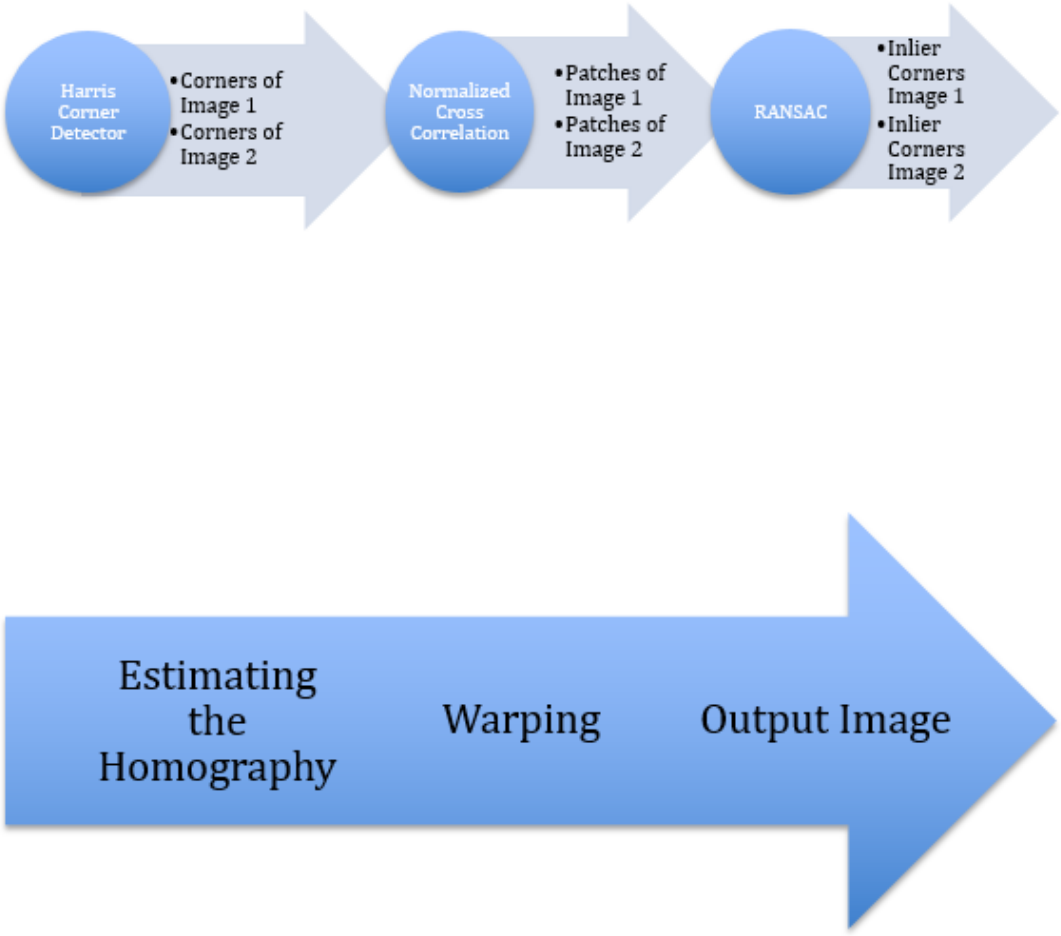
October 23, 2018

Abstract

The main purpose of this project is to mosaic images that are taken from a stationary camera, which is rotated on its axis. Due to the overlapping regions between images, corners can be used as significant features that can be matched after extraction of features. For this purpose, Harris corner detector will be used to find the corners in images, afterwards those features will be used to estimate the homography between the images. The homography estimate will enable us to warp one image into the coordinate system of the second image which will give the result of the mosaic obtained from images with overlapping pixels being blended to each other.

Algorithm Descriptions

Flowchart



Flowchart - Top to Bottom

The Harris Corner Detector

In order to apply the Harris corner detector, we first filter the image with Sobel masks that would give the gradient information in both x and y directions of the image. One other benefit of using Sobel mask is that it also has the property of averaging around the pixel which works as a low pass filter and helps us to smoothen the noise component within the pixels. We square the gradient matrices and also construct the cross gradient product matrix (which results in 3 new matrices: I2x, I2y, Ixy). These matrices are low pass filtered with a Gaussian filter in order to obtain the averaged sum within the Gaussian mask for each pixel. These new matrices give us the results where we can use the R-score test for each pixel, which would provide the necessary information for us to decide whether a pixel is a corner or not (i.e. for a corner the R value should be positive and large; whereas for edges the R value is negative and for flat region the R value is positive but small). The R value is calculated from the value given as R(i,j) = detM - k* trace(M) where k is chosen to

be a value between 0.04-0.06 and the M matrix is constructed as filling the diagonal of the matrix with the filtered and squared gradient pixels and non-diagonal terms are obtained from the cross product pixels. We then apply a non-max suppression for the pixels that are supposed to be corners (due to algorithm and filtering the corners does not occur as a single pixel instead occurs as blobs) so that we can find the real pixels corresponding to the corners within the image.

Normalized Cross Correlation

After obtaining the corner pixels from two images, we want to find the corners that correspond to each other between two images. For this purpose we apply the normalized cross correlation algorithm. The algorithm consists of first obtaining patches from the images whose centers correspond to the corners that were found with the Harris corner detector algorithm. We normalize each patch by first subtracting the mean value of the patch from each pixel value within the patch and then by dividing each pixel value within the patch by the summation of all pixel values within patch. After this process, we calculate the cross correlations between every patch in Image 1 and Image 2. We find the highest normalized cross correlation value for each patch in Image 1 among all the patches in Image 2. We then compare that value with a predetermined threshold value in order to make sure the correspondence between two patches is higher than a minimum value. We will make use of the corners corresponding to those patches (obtained from NCC process) to estimate the homography matrix.

RANSAC

After the NCC part, we have corners from each image that correspond to each other but even though we threshold the NCC scores there are still outliers which will result in a bad estimate of the homography matrix. For this reason, we will be using RANSAC. For this algorithm we choose 4 random corners (the corners that were obtained from NCC part which have a corresponding corner in the Image 2) from the first image and estimate the homography. The homography matrix has 9 components but it has 8 degrees of freedom. In order to estimate the homography, we are using the constraint of $||h||=1$. Thus, from the lecture notes we can estimate the homography matrix from 4 points by constructing the matrix A and using SVD in order to find the corresponding homography matrix. We use homogeneous coordinates for the coordinates of pixels in order to be able to do multiplication with the homography matrix. After estimating the homography matrix we map all the corners of Image 1 (obtained in the NCC part) to corners in Image 2. We compute the distances of the estimated coordinate of the corner in Image 2 with the real coordinate of the corner in Image 2. In order to make a valid comparison we normalize the new coordinates with the third component so that homogeneous coordinates are in the same form. We now compare the distances with a predetermined threshold so that we would be able to find the correspondences that are valid with the homography transformation. We repeatedly apply this procedure, so that we would be able to find the homography matrix that has the highest number of valid correspondences. We then use these inlier corners to construct the new homography (least square homography) matrix that would be the best estimate of the homography matrix.

Warping

We first start by assigning a predetermined size for the final output image (logically it should be smaller than two times the individual images since there is an overlapping between images). We place the first image in this output image and by using the inverse of the estimated homography matrix we warp the second image into the first image where the 'interp2' and 'meshgrid' functions of Matlab is used to apply the bilinear interpolation on the warped image. We used these functions to construct a reference and then build our own warping algorithm which is based on observations of warping the pixel coordinates. For the overlapping regions we applied averaging of the pixels (as our blending scheme) of the first image and the warped image so that we have smoother transition areas.

Experiments

Destination image

Source image

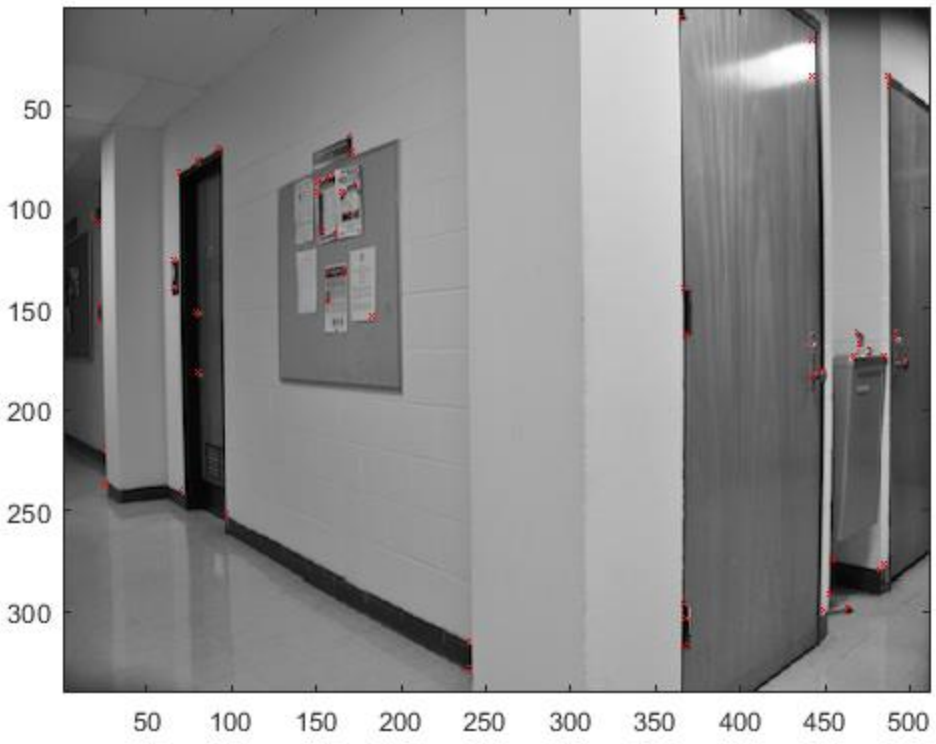
The Harris Corner Detector

We held experiments for each step of the whole algorithm in order to make sure individual and combined parts are working correctly. As a first step, we obtained the results for Harris corner detector. In order to obtain the gradients we used the Sobel given as [-1 -1 -1;0 0 0 ;1 1 1] for y-direction gradients and the transpose of the filter to obtain the x-direction gradients. We used Gaussian filter with sigma equal to 1 and filter size of 5x5 in order to obtain the sum-averaged squared gradients for each pixel. We chose k=0.05 as a constant for calculating the R score. We used the threshold of 5*10^7 for R scores, in order to decide whether a pixel is a corner (after analyzing the output of R scores in different pixels corresponding to different regions). As aforementioned the corner locations occur as blobs of pixels so we used non-max suppression where we find each blob and assigned the centroid of the blob as the real corner. The following image shows the corner positions before and after non-max suppression as a binary image where white points correspond to the corner location:

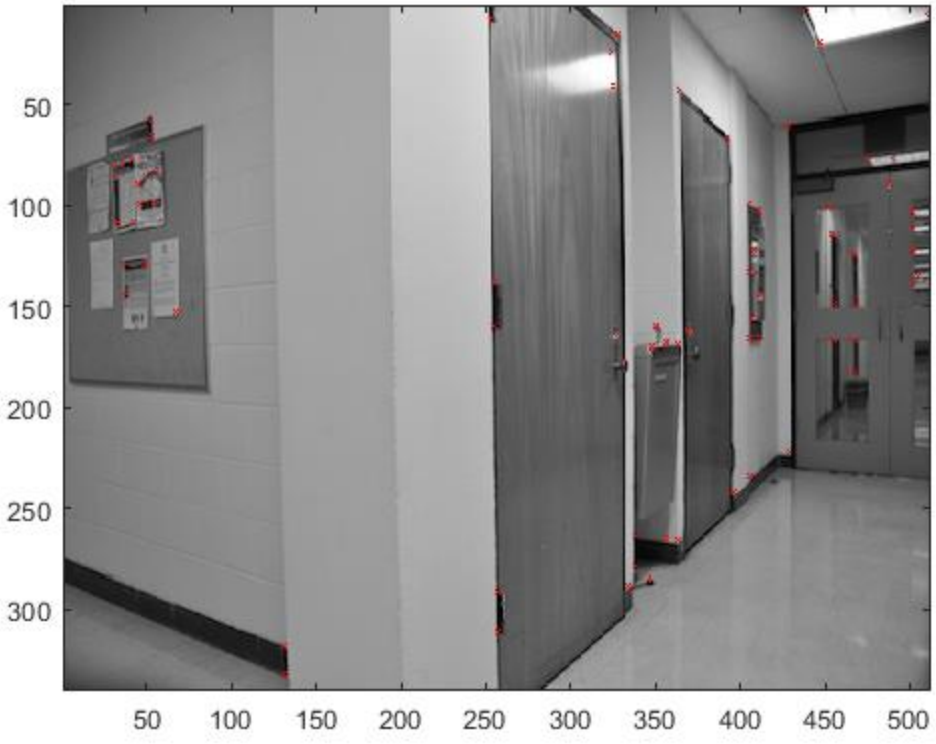
Destination image. corners before & after non-max suppression

After finding all the corner locations obtained from the Harris Corner Detector, the following images shows the locations of corners within the original image marked with a red ‘x’ :

Srouce image corners before & after non-max suppression



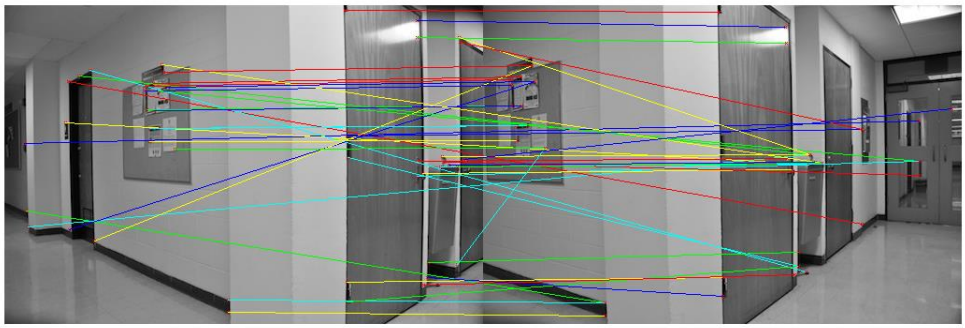
Destination image corners



Source image corners

Normalized Cross Correlation

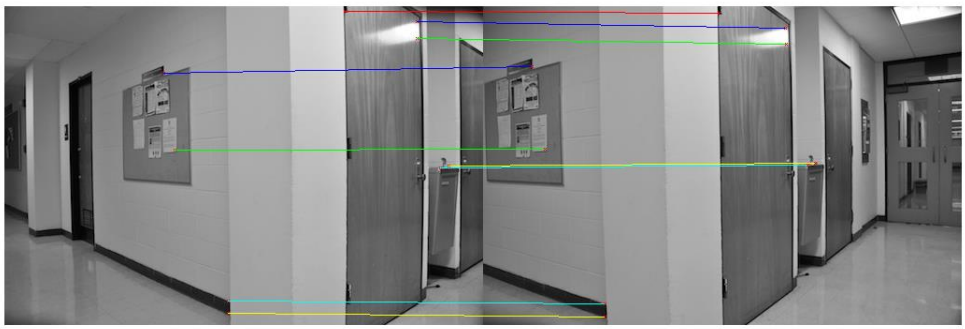
After obtaining the coordinates of the corners from both of the images, now we can apply normalized cross correlation for the patches obtained from images. We chose the patch sizes to be 5x5. Since the patches are normalized the highest correlation between them can be 1. After experimenting and analyzing the results, we assigned the threshold for correlation between two patches to be 0.998 a good fit between a pair of corners from each image. We inserted a marker for the corners that have a correspondent corner in the other image and we draw a line to visualize the findings of NCC algorithm for the corresponding corners in each image, which is shown in the image below:



Correlating all the corners

RANSAC

In this part, we estimated homographies from randomly chosen 4 points obtained from the NCC part that correspond to corners in first image. As described before we mapped every corner in first image to the corners corresponding in the second image with the homography matrix. We calculated the distances between the mapped corner and the real corner coordinates and compared with a threshold which is assigned to be 10 which result in the number of valid correspondences. We iterated this random 4 corner selection 400 times and chose the homography that has the highest number of correspondences. Using the those corners, we estimated the final homography which would be the least squares homography from the all the inliers in the largest set of inliers. The following image shows the inlier corners from the set of correspondences (the set that is used to estimate the least-squares homography) and a line is drawn between the corresponding corners from images:



Using RANSAC to find inlier correlated corners

Warping

After obtaining the finalized homography, we can warp the second image on to first one. For this purpose, we decided to have final output image's size to be 340x669 after experimenting with several different sizes. We placed the first image within the output

image and warped the second image onto that by using the homography matrix obtained in the previous part. We averaged the values of the overlapping parts as blending scheme and the output image is shown below for several different datasets:

Source Image brought into destination image

Source Image brought into destination image

Observations

In Harris Corner Detector, we were able to detect the corners within the images. Before applying the non-max suppression we were getting a region of corners instead of a pixel for the corner, thus non-max suppression was an important step of obtaining the resulting corners. The normalized cross correlation algorithm enabled us to find the corners that are corresponding to each other but still there were wrong correspondences (outliers) which

could be seen from the lines drawn, thus, RANSAC was also an important step to get rid of the outliers and construct the best estimate of the homography matrix for the warping step. We can see that after RANSAC applied the lines drawn between corresponding corners from each image becomes far more accurate compared to just normalized cross correlation outputs. Changing the threshold for both RANSAC and NCC decreases our accuracy since the values for threshold were experimentally observed and chosen. Thus, we see that corners are good features of a image that can be used for various applications including mosaicking.

We struggled for longer than we should have because we were unaware of certain matlab commands that made implementing these algorithms easier. Ironically we experienced the opposite situation when implementing the mosaic portion of the project as we implemented our own interp2 function rather than use the matlab command.

Extra Credit Experiments

Our extra credit work was done mostly by recycling code snippets used throughout the rest of the project. The key difference was rather than warping a portion of a scene into another, we warped an entire scene, the entire square picture, into a self contained plane within the destination scene.

Here are some of our attempts for making extra credit images.

Extra credit #1

Extra credit #2

Conclusions

The main purpose of this project was to mosaic images that are taken from a stationary camera, rotated on its axis. Due to the overlapping regions between images, corners can be used as significant features that can be matched after extraction of features. The Harris corner detector was used to find the corners in images, afterwards those features were used to estimate the homography between the images. The homography estimate enabled us to warp one image into the coordinate system of the second image which will give the result of the mosaic obtained from images with overlapping pixels being blended to each other. We learned during this project the steps required to integrate several algorithms we learned about in theory during class. These algorithms combine to form a robust system that can be reused in the future to perform mosaicking for different imagesets.

Apendix

Main.m

```
clear all
close all
clc
% Ima1 = rgb2gray(imread('DSC_0282.JPG'));
% corners = HarrisCornerDetector2(Ima1);
% %[positionx,positiony] = find(corners~=0);
% color = {'red'};
% out1 =
insertMarker(Ima1,[corners(:,1),corners(:,2)],'x','color',color,'size',1);
% image(out1)
%%
clear all
clc
Ima1 = rgb2gray(imread('DSC_0309.JPG'));
Ima2 = rgb2gray(imread('DSC_0308.JPG'));

%%
% [res,cor1,cor2] = NCC2(Ima1,Ima2,0.99);
% % [positionx] = find(res~=0);
% % corners = HarrisCornerDetector(Ima);
% % [positionx,positiony] = find(corners~=0);
%  color = {'red'};
%  out1 = insertMarker(Ima1,cor1,'x','color',color,'size',1);
%  out2 = insertMarker(Ima2,cor2,'x','color',color,'size',1);
% imshowpair(out1,out2,'montage');
% hold on
% %line([cor1(1,1),cor2(1,1)+512],[cor1(1,2),cor2(1,2)]);
% for m = 1: size(cor1,1)
%     colors = {'red','blue','green','cyan','yellow'};
%
line([cor1(m,1),cor2(m,1)+512],[cor1(m,2),cor2(m,2)],'color',colors{mod(m,5)+
1});
%     hold on;
% end
% image(out)

%%
clear all
close all
clc
Ima1 = rgb2gray(imread('DSC_0309.JPG'));
Ima2 = rgb2gray(imread('DSC_0308.JPG'));

[out,offhomog,incor1,incor2] = RANSAC2(Ima1,Ima2,400,10,0.998);
% color = {'red'};
% out1 = insertMarker(Ima1,incor1,'x','color',color,'size',1);
% out2 = insertMarker(Ima2,incor2,'x','color',color,'size',1);
% imshowpair(out1,out2,'montage');
% hold on
% %line([cor1(1,1),cor2(1,1)+512],[cor1(1,2),cor2(1,2)]);
% for m = 1: size(incor1,1)
%     colors = {'red','blue','green','cyan','yellow'};
```

```
%
line([incor1(m,1),incor2(m,1)+512],[incor1(m,2),incor2(m,2)],'color',colors{m
od(m,5)+1});
%     hold on;
% end
% image(out)
[maxv loc] = max([out{:,2}]);
firsthomo = out{loc,1}
offhomog

% warping part
Ima1 = double(Ima1);
Ima2 = double(Ima2);
finim = zeros(340,668);
[w a] = size(finim)
[xi , yi ] = meshgrid(1:w, 1:a);
xi = xi';
yi = yi';
h = inv(offhomog);
xx = round( h(1,1)*xi+h(1,2)*yi + h(1,3))./(h(3,1)*xi+h(3,2)*yi+h(3,3));
yy = round( h(2,1)*xi+h(2,2)*yi + h(2,3))./(h(3,1)*xi+h(3,2)*yi+h(3,3));
k=uint8(round(interp2(Ima2,yy,xx)));

imshow(k)
for indexX = 1:668
    for indexY = 1:340
        newCoords = offhomog * [indexY; indexX; 1];
        newCoords = newCoords / newCoords(3,1);

        X = floor(newCoords(2,1)) + 1;
        Y = floor(newCoords(1,1)) + 1;

        if X >= 1 && Y >= 1
            if indexX > 512
                finim(indexY,indexX) = double(Ima2(Y,X));
            else
                if Y > 340
                    finim(indexY,indexX) = double(Ima1(indexY,indexX));
                else
                    finim(indexY,indexX) = (double(Ima1(indexY,indexX))+
double(Ima2(Y,X)))/2;
                end
            end
        elseif (X < 1)
            finim(indexY,indexX) = double(Ima1(indexY,indexX));
        elseif Y < 1 && indexY > 1
            finim(indexY,indexX) = double(Ima1(indexY,indexX));
        end
    end
    imshow(uint8(finim))
end
imshow(uint8(finim))
```

HarrisCornerDetector

```
function [ fincorncoor ] = HarrisCornerDetector2( Ima )
```

```
[h,w] = size(Ima);
sob = [-1 -1 -1; 0 0 0; 1 1 1];

Ix = filter2(sob',Ima,'valid');
Iy = filter2(sob,Ima,'valid');

Ix2 = Ix.*Ix;
Iy2 = Iy.*Iy;
Ixy = Ix.*Iy;

Ix2_filted = imgaussfilt(Ix2,1);
Iy2_filted = imgaussfilt(Iy2,1);
Ixy_filted = imgaussfilt(Ixy,1);

k = 0.05;
threshold =5*10^7;
R = zeros(h,w);

for i = 1:h-2
    for j =1:w-2
        detM = Ix2_filted(i,j) * Iy2_filted(i,j)-Ixy_filted(i,j)^2;
        R(i,j) = detM -k*(Ix2_filted(i,j) + Iy2_filted(i,j))^2;
    end

end

corners_coor = R.*(R>threshold);
corners_coor(corners_coor~=0) =1;
%imshow(corners_coor)
%fincorncoor =zeros(size(corners_coor));

CC = bwconncomp(corners_coor);
S = regionprops(CC,'centroid');
fincorncoor = zeros(size(S,1),2);

for u = 1:size(S,1)
   fincorncoor(u,:) = round(S(u).Centroid);
end

imshowpair(corners_coor,fincorncoor,'montage')
end
```

NCC

```
function [ bestfit,cornersncc1,cornersncc2 ] = NCC2( Im1,Im2,nccthre )

corners1 = HarrisCornerDetector2(Im1);
corners2 = HarrisCornerDetector2(Im2);
ps =2;
px1 = corners1(:,2);
py1 = corners1(:,1);
px2 = corners2(:,2);
py2 = corners2(:,1);

temp = zeros(1,length(px2));
bestfit = zeros(1,length(px1));
Im1 = [circshift(Im1(1:ps,:),ps-1,1); Im1 ;circshift(Im1(end:end-ps,:),ps-
1,1)];
Im1 = [circshift(Im1(:,1:ps),ps-1,2) Im1  circshift(Im1(:,end-ps:end),ps-
1,2)];
Im2 = [circshift(Im2(1:ps,:),ps-1,1); Im2 ;circshift(Im2(end:end-ps,:),ps-
1,1)];
Im2 = [circshift(Im2(:,1:ps),ps-1,2) Im2  circshift(Im2(:,end-ps:end),ps-
1,2)];
Im1 = double(Im1);
Im2 = double(Im2);
for i = 1:length(px1)
    patch1 = Im1(px1(i):px1(i)+2*ps,py1(i):py1(i)+2*ps);
    patch1mm = patch1 -sum(sum(patch1))/(ps+1)^2;
    patch1n = patch1mm/sqrt(sum(sum(patch1mm.*patch1mm)));

    for j = 1: length(px2)

        patch2 = Im2(px2(j):px2(j)+2*ps,py2(j):py2(j)+2*ps);
        patch2mm = patch2 -sum(sum(patch2))/(ps+1)^2;
        patch2n = patch2mm/sqrt(sum(sum(patch2mm.*patch2mm)));

        temp(j) = sum(sum(patch1n.*patch2n));


    end
    [maxv, patch2no] = max(temp);

    if maxv > nccthre
        bestfit(i) = patch2no;
    end


end

[positionx] = find(bestfit~=0);
cornersncc1 = [py1(positionx) px1(positionx)];
cornersncc2 = [py2(bestfit(positionx)) px2(bestfit(positionx))];

end
```

RANSAC

```
function [ homocount,offchomog,incor1,incor2 ] = RANSAC2(
Ima1,Ima2,iter,ransthreshold,nccthre )

[ bestfit,cornersncc1,cornersncc2 ] = NCC2( Ima1,Ima2,nccthre );
cornersncc1 = circshift(cornersncc1,1,2);
cornersncc1 = [cornersncc1 ones(size(cornersncc1,1),1)];
cornersncc2 = circshift(cornersncc2,1,2);
cornersncc2 = [cornersncc2 ones(size(cornersncc2,1),1)];

homocount = cell(iter,2);

for y = 1:iter

    randcorloc = floor(size(cornersncc1,1)*rand(4,1))+1;
    A = zeros(8,9);
    for p=1:4
            2*p-1;
            A(2*p-1,1) = cornersncc1(randcorloc(p),1);
            A(2*p-1,2) = cornersncc1(randcorloc(p),2);
            A(2*p-1,3) =1;
            A(2*p-1,7) = -
cornersncc1(randcorloc(p),1)*cornersncc2(randcorloc(p),1);
            A(2*p-1,8) = -
cornersncc1(randcorloc(p),2)*cornersncc2(randcorloc(p),1);
            A(2*p-1,9) = -cornersncc2(randcorloc(p),1);
            A(2*p,4) = cornersncc1(randcorloc(p),1);
            A(p*2,5) = cornersncc1(randcorloc(p),2);
            A(p*2,6) =1;
            A(p*2,7) = -
cornersncc1(randcorloc(p),1)*cornersncc2(randcorloc(p),2);
            A(p*2,8) = -
cornersncc1(randcorloc(p),2)*cornersncc2(randcorloc(p),2);
            A(p*2,9) = -cornersncc2(randcorloc(p),2);


    end
    [U, S, V] =svd(A'*A);
    homog = U(:,9);
    homog = transpose(reshape(homog,3,3));

    count =0;

    mapped1to2 = zeros(size(cornersncc1));
    distances = zeros(size(cornersncc1,1),1);

    for n = 1: size(cornersncc1,1)
        mapped1to2(n,:) = transpose(homog*transpose(cornersncc1(n,:)));
        mapped1to2(n,:) = mapped1to2(n,:)/mapped1to2(n,3);
        distances(n) = norm( mapped1to2(n,:)-cornersncc2(n,:));

        if distances(n)< ransthreshold
            count = count +1;
```

```
        end
    end
    homocount{y,1} = homog;
    homocount{y,2} = count;
end

[maxv loc] = max([homocount{:,2}]);
realhomo = homocount{loc,1};
inliercorners = zeros(size(cornersncc1,1),1);
for j = 1: size(cornersncc1,1)
        mapped1to2(j,:) = transpose(realhomo*transpose(cornersncc1(j,:)));
        mapped1to2(j,:) = mapped1to2(j,:)/mapped1to2(j,3);
        distances(j) = norm( mapped1to2(j,:)-cornersncc2(j,:));

        if distances(j)< ransthreshold
            inliercorners(j) = 1;

        end
end
inlierloc1 = find(inliercorners~=0);
incor1 = cornersncc1(inlierloc1,:);
incor1 = circshift(incor1(:,1:2),1,2);

incor2 = cornersncc2(inlierloc1,:);
incor2 = circshift(incor2(:,1:2),1,2);
A = zeros(length(inlierloc1)*2,9);

for i=1:length(inlierloc1)

            A(2*i-1,1) = cornersncc1(inlierloc1(i),1);
            A(2*i-1,2) = cornersncc1(inlierloc1(i),2);
            A(2*i-1,3) =1;
            A(2*i-1,7) = -
cornersncc1(inlierloc1(i),1)*cornersncc2(inlierloc1(i),1);
            A(2*i-1,8) = -
cornersncc1(inlierloc1(i),2)*cornersncc2(inlierloc1(i),1);
            A(2*i-1,9) = -cornersncc2(inlierloc1(i),1);
            A(2*i,4) = cornersncc1(inlierloc1(i),1);
            A(i*2,5) = cornersncc1(inlierloc1(i),2);
            A(i*2,6) =1;
            A(i*2,7) = -
cornersncc1(inlierloc1(i),1)*cornersncc2(inlierloc1(i),2);
            A(i*2,8) = -
cornersncc1(inlierloc1(i),2)*cornersncc2(inlierloc1(i),2);
            A(i*2,9) = -cornersncc2(inlierloc1(i),2);

end


    [U, S, V] =svd(A'*A);
    %[~, loc] =min(S);
    homog = U(:,9);
    offchomog = transpose(reshape(homog,3,3));
end
```

Extra Credit – main.m

```
targetImg = double(rgb2gray(imread('extracredit.png')));
figure
image(targetImg);
computerScreenPlane = floor(ginput());

flagImg = rgb2gray(imread('flag.png'));
flagImg = imresize(flagImg,[340 512]);
% figure
% image(flagImg);
% flagPlane = double(floor(ginput()))
flagPlane = [1 1; 1 340; 512 1; 512 340]

for i=1:4

            A(2*i-1,1) = computerScreenPlane(i,1);
            A(2*i-1,2) = computerScreenPlane(i,2);
            A(2*i-1,3) =1;
            A(2*i-1,7) = -computerScreenPlane(i,1)*flagPlane(i,1);
            A(2*i-1,8) = -computerScreenPlane(i,2)*flagPlane(i,1);
            A(2*i-1,9) = -flagPlane(i,1);
            A(2*i,4) = computerScreenPlane(i,1);
            A(i*2,5) = computerScreenPlane(i,2);
            A(i*2,6) =1;
            A(i*2,7) = -computerScreenPlane(i,1)*flagPlane(i,2);
            A(i*2,8) = -computerScreenPlane(i,2)*flagPlane(i,2);
            A(i*2,9) = -flagPlane(i,2);

end

[U, S, V] =svd(A'*A);
h = U(:,9);
H = transpose(reshape(h,3,3));

for indexX = 1:686
    for indexY = 1:433
        newCoords = H * [indexX; indexY; 1]; % during testing this and line
"Y = floor(newCoords(2,1)) + 1;" were switched with their repsective
coordinate... It may be due to the reshaping occuring at the beginning.
        newCoords = newCoords / newCoords(3,1);
        indexX;
        indexY;
        Y = floor(newCoords(2,1)) + 1;
        X = floor(newCoords(1,1)) + 1;

        if (X >= 1 && Y >= 1) && (X < 512 && Y < 340)
            finim(indexY,indexX) = double(flagImg(Y,X));
        else
            finim(indexY,indexX) = double(targetImg(indexY,indexX));
        end
    end
    imshow(uint8(finim))
end
```