Matthew Vaysfeld
I pledge my honor that I have abided by the Stevens Honor System

Code

```
function main = main(image1,image2,inliers, threshold)
    arguments
    image1 string = "uttower_left.jpg";
    image2 string = "uttower_right.jpg";
    inliers string = "15";
    threshold string = "10";
    end
    inliers = str2double(inliers);
    threshold = str2double(threshold);
    f1 = figure('Name',"Left Harris");
    f2 = figure('Name',"Right Harris");
    f3 = figure('Name',"Left NMS");
    f4 = figure('Name',"Right NMS");
    f5 = figure('Name',"Patch Similarity");
    f6 = figure('Name',"Warped Image for 20 points");
    f7 = figure('Name',"Warped Image for 50 points");


    %Read images and find harris (1a)
    image1 = imread(image1);
    image2 = imread(image2);
    image1g = rgb2gray(image1);
    image2g = rgb2gray(image2);
    harris1 = harrisdetector(double(image1g));
    harris2 = harrisdetector(double(image2g));

    figure(f1);
    imshow(harris1);
    figure(f2);
    imshow(harris2);

    % Do nms (1b)
    maxima1 = maxes(harris1);
    maxima2 = maxes(harris2);
    answer1 = non_maxsuppression(maxima1);
    answer2 = non_maxsuppression(maxima2);

    figure(f3);
    imshow(answer1);
```

```
    figure(f4);
    imshow(answer2);

    %Do matching on images (1c)
    points1 = getpoints(answer1);
    points2 = getpoints(answer2);
    [pointsleft,pointsright,pointsall] =
ssd_patch(image1g,image2g,points1,points2,40);
    figure(f5);

showMatchedFeatures(image1,image2,pointsleft,pointsright,'m
ontage');

    %Get Random points (2a)
    pointsrandom = correspondences(pointsall);
    disp(pointsrandom);

    %Find Affine transformation (2b)
    pointsall = pointsall(1:20,1:4);
    [Affine,Average,Inliers,Iter] =
ransac(pointsall,inliers,threshold);
    disp(Affine);
    disp(Average);
    disp(Inliers);
    disp(Iter);

    pointsall = cat(1,pointsall,pointsrandom(1:30,1:4));
    [AffineR,AverageR,InliersR,IterR] =
ransac(pointsall,inliers,threshold);
    disp(AffineR);
    disp(AverageR);
    disp(InliersR);
    disp(IterR);


    %Warp Image (2c)

    warped = warp(image1, image2, Affine);
    figure(f6);
    imshow(uint8(warped));


    warpedR = warp(image1, image2, AffineR);
    figure(f7);
```

```
    imshow(uint8(warpedR));
end
%Reused code
%Pads Matrix
function padded_array = pad(image,sigma)
    %original rows, original colums
    [or,oc] = size(image);
    new_sigma = 3*sigma;
    m = zeros(or+(2*new_sigma), oc+(2*new_sigma),
"double");

m(new_sigma+1:or+new_sigma,new_sigma+1:oc+new_sigma)=image;

    %Top Left Corner
    for r = 1:new_sigma+1
        for c = 1:new_sigma+1
            m(r,c) = m(new_sigma+1,new_sigma+1);
       end
    end
    %Left side
    for r = new_sigma+1:or+new_sigma-1
        for c = 1:new_sigma+1
            m(r,c) = m(r,new_sigma+1);
        end
    end
     %Bottom Left Corner
     for r = or+new_sigma:or+(2*new_sigma)
        for c = 1:new_sigma+1
            m(r,c) = m(or+new_sigma,new_sigma+1);
       end
    end
     %Top
    for r = 1:new_sigma
         for c = new_sigma+1:oc+new_sigma
             m(r,c) = m(new_sigma+1,c);
         end
     end
     %Top Right Corner
     for r = 1:new_sigma+1
        for c = oc+new_sigma+1:oc+(2*new_sigma)
             m(r,c) = m(new_sigma+1,oc+new_sigma);
        end
     end
     %Right Side
```

```
     for r = new_sigma+1:or+new_sigma-1
         for c = oc+new_sigma+1:oc+(2*new_sigma)
             m(r,c) = m(r,oc+new_sigma);
         end
     end
     %Bottom Right Corner
     for r = or+new_sigma:or+(2*new_sigma)
         for c = oc+new_sigma+1:oc+(2*new_sigma)
             m(r,c) = m(or+new_sigma,oc+new_sigma);
        end
     end
     %Bottom
     for r = or+new_sigma:or+(2*new_sigma)
         for c = new_sigma+1:oc+new_sigma
             m(r,c) = m(or+new_sigma,c);
        end
     end
    padded_array=m;

end

%Applies any filter to an image
%Takes in a filter, image, and the extended image
function apply_filter = app(filter,image,ext_image)
    [row,col] = size(image);
    [re,ce] = size(ext_image);
    [~,len] = size(filter);
    newx = re - row;
    newy = ce - col;
    image2 = zeros(row, col);
    for r = 1:row
        for c = 1:col
            newr = r +((newx)/2);
            newc = c +((newy)/2);
            piece = ext_image((newr)-((len-
1)/2):(newr)+((len-1)/2),(newc)-((len-1)/2):(newc)+((len-
1)/2));
            %mult_matrix = mult_matrices(filter,piece);
            mult_matrix = filter .* piece;
            image2(r,c) = sum(mult_matrix(:));
        end
    end
    apply_filter = image2;
end
```

```
%Gets the Gaussian Filter
function get_gaussian = gauss(sigma)
   size = 6*sigma;
   filter = zeros(size-1,size-1);
   for r = 1:size-1
        for c = 1:size-1
             x = abs(r - size/2);
             y = abs(c - size/2);
             filter(r,c) = (1/(2*pi*sigma))*exp((-(x^2 +
y^2))/(2*sigma^2));
        end
   end
   sumof = sum(filter(:));
   get_gaussian = filter;
end

%Applies the Gaussian filter
function appgauss = appgauss(image,sigma)
    %Do the Gauss
     gaussian = gauss(sigma);
     padded_matrix = pad(image,sigma);
     appgauss = app(gaussian,image,padded_matrix);
end

%Applies the Vertical Sobel Filter
function sob1 = sobel1(gaussimage)
     single_pad = pad(gaussimage,1);
     sobelfilt1 = [-1 0 1; -2 0 2; -1 0 1];
     sob1 = app(sobelfilt1,gaussimage,single_pad);
     sob1 = double(sob1);

end

%Applies the Horizontal Sobel Filter
function sob2 = sobel2(gaussimage)
     single_pad = pad(gaussimage,1);
     sobelfilt2 = [1 2 1; 0 0 0; -1 -2 -1];
     sob2 = app(sobelfilt2,gaussimage,single_pad);
     sob2 = double(sob2);
end
```

```
%Parts 1 a/b
function harris = harrisdetector(image)
    gaussimage = appgauss(image,3);
    [row,col] = size(image);
    sobx = sobel1(gaussimage);
    sobxx = sobel1(sobx);
    soby = sobel2(gaussimage);
    sobyy = sobel2(soby);
    sobxy = sobel2(sobx);
    image2 = zeros(row, col);
    for r = 1:row
        for c = 1:col
            mm = gaussimage(r,c) * [sobxx(r,c) sobxy(r,c);
sobxy(r,c) sobyy(r,c)];
            image2(r,c) = det(mm) - (.05*(trace(mm)^2));
        end
    end
    harris = image2;
end

function maximum = maxes(harris)
    [row,col] = size(harris);
    image2 = zeros(row, col);
    count = 0;
    while(count < 1000)
        biggest = max(harris(:));
        for r= 1:row
            for c = 1:col
                if(harris(r,c) == biggest)
                    image2(r,c) = biggest;
                    harris(r,c) = 0;
                    count= count + 1;
                    r=row;
                    break;
                end
            end
        end
    end
    maximum = image2;
end

function non_max = non_maxsuppression(image)
    [row,col] = size(image);
    m = zeros(row,col, "double");
```

```
    image = pad(image,(1/3));
    for r = 2:row+1
         for c= 2:col+1
             if(max(image(r-1:r+1,c-1:c+1), [], 'all') ==
image(r,c))
                    m(r-1,c-1) = image(r,c);
                else
                    m(r-1,c-1) = 0;
             end
         end
    end
    non_max = m;
end

function getpoints = getpoints(matrix)
    [row,col] = size(matrix);
    getpoints = [];
    for r= 1:row
        for c = 1:col
            if(matrix(r,c) ~= 0)
                getpoints = cat(1,getpoints,[r c]);

            end
        end
    end
end

%Part 1c

function [pointsleft,pointsright,pointsall] =
ssd_patch(image1,image2,points1,points2,r)
    [row1,~] = size(points1);
    [row2,~] = size(points2);
    pointsall = [];
    harris1 = pad(image1,r/3);
    harris2 = pad(image2,r/3);
     for r1= 1:row1
        for r2 = 1:row2
            p1 = points1(r1,1:2);
            p2 = points2(r2,1:2);
            patch1 = harris1((p1(1)+r)-
r:(p1(1)+r)+r,(p1(2)+r)-r:(p1(2)+r)+r);
            patch2 = harris2((p2(1)+r)-
r:(p2(1)+r)+r,(p2(2)+r)-r:(p2(2)+r)+r);
```

```
            X=patch1-patch2;
            ssd = sum(X(:).^2);
            p1 = p1(end:-1:1);
            p2 = p2(end:-1:1);
            combined = cat(2,p1,p2,ssd);
            pointsall = cat(1,pointsall,combined);
        end
    end
    pointsall = sortrows(pointsall,5);
    pointsleft = pointsall(1:20,1:2);
    pointsright = pointsall(1:20,3:4);


end

%Part 2a
function pointsrandom = correspondences(pointsall)
    pointsrandom = [];
    [row,~] = size(pointsall);
    for i = 1:30
        r = randi(row);
        pointsrandom =
cat(1,pointsrandom,pointsall(r,1:end));
    end
end

%Part 2b
function [RANSAC,average,inlierlist,iter] =
ransac(pointsall,inliers,threshold)
    iter = 0;
    while(1==1)
        inlierlist = [];
        inliersc = 0;
        points = [];
        [row,~] = size(pointsall);
        r = randperm(row,3);
        points = cat(1,points,pointsall(r(1),1:4));
        points = cat(1,points,pointsall(r(2),1:4));
        points = cat(1,points,pointsall(r(3),1:4));
        A = [points(1,1) points(1,2) 1 0 0 0;
                0 0 0 points(1,1) points(1,2) 1;
                points(2,1) points(2,2) 1 0 0 0;
                0 0 0 points(2,1) points(2,2) 1;
                points(3,1) points(3,2) 1 0 0 0;
```

```
                0 0 0 points(3,1) points(3,2) 1];
        B = [points(1,3);
                points(1,4);
                points(2,3);
                points(2,4);
                points(3,3);
                points(3,4)];
        X= linsolve(A,B);
        %Apply to each first point, check distance between
affined point and
        %second point
        totaldist = 0;
        for r = 1 : row
            first = [pointsall(r,1);
                    pointsall(r,2);
                    1];
            second = [X(1,1) X(2,1) X(3,1);
                      X(4,1) X(5,1) X(6,1)];
            point = second * first;
            distance = sqrt(((point(1,1) -
pointsall(r,3))^2) + ((point(2,1) - pointsall(r,4))^2));
            if(distance < threshold)
                inliersc = inliersc + 1;
                combined =
cat(2,pointsall(r,3),pointsall(r,4));
                inlierlist = cat(1,inlierlist,combined);
                totaldist = totaldist + distance;
            end
        end
        iter = iter + 1;
        if(inliersc >= inliers)
            RANSAC = X;
            average = totaldist/inliersc;
            break;
        end
    end
end


function warped = warp(image1, image2, affine)
    change = maketform('affine', [affine(1,1) affine(4,1)
0; affine(2,1) affine(5,1) 0; affine(3,1) affine(6,1) 1]);
    [transformation, x, y] = imtransform(image1,change);
    [row, col,~] = size(image2);
```

```
    [rowt, colt,~] = size(transformation);
    xshift = ceil(abs(x(1)));
    yshift = ceil(abs(y(1)));
    newimage = zeros(row+yshift,col+xshift,3,'uint8');
    newimage(yshift+1:end,xshift+1:end,:) = image2;

    for r = 1:rowt
        for c = 1:colt
            if(transformation(r,c,:) == [0 0 0])

            else
                if(newimage(r,c,:) == [0 0 0])
                    newimage(r,c,:) =
transformation(r,c,:);
                else
                    newimage(r,c,:) =
(transformation(r,c,:) .* (.5)) + (newimage(r,c,:) .*
(.5));
                end

            end


        end
    end
    warped = newimage;

end
```

Resulting Images

Part 1a

Harris Corner Detector

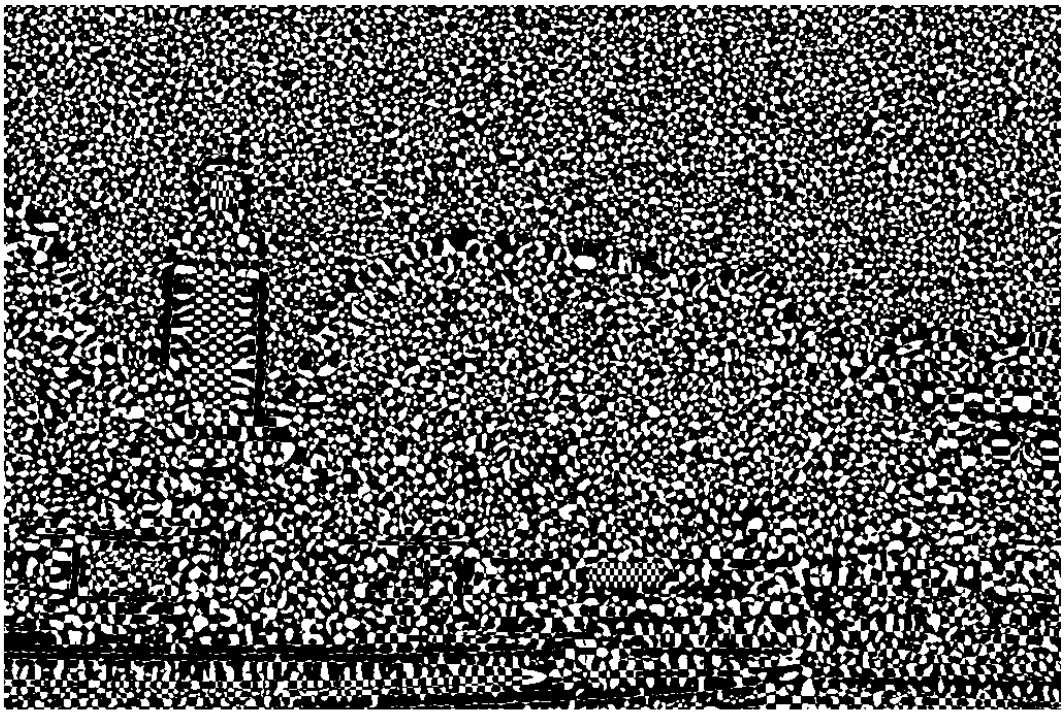


Image: Left Tower

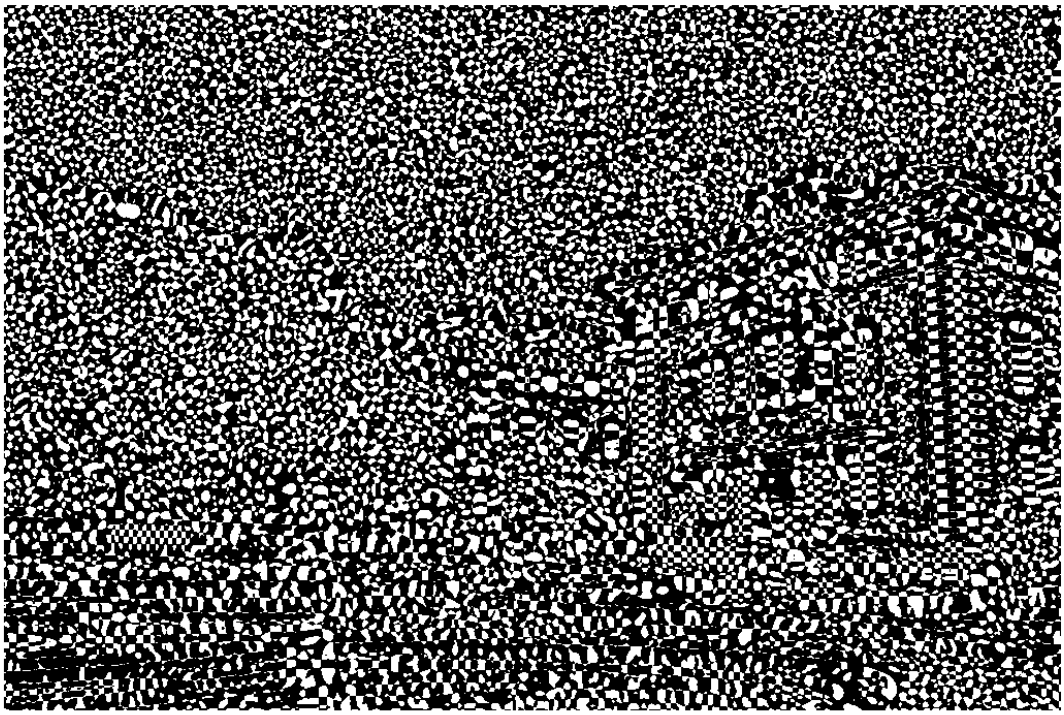
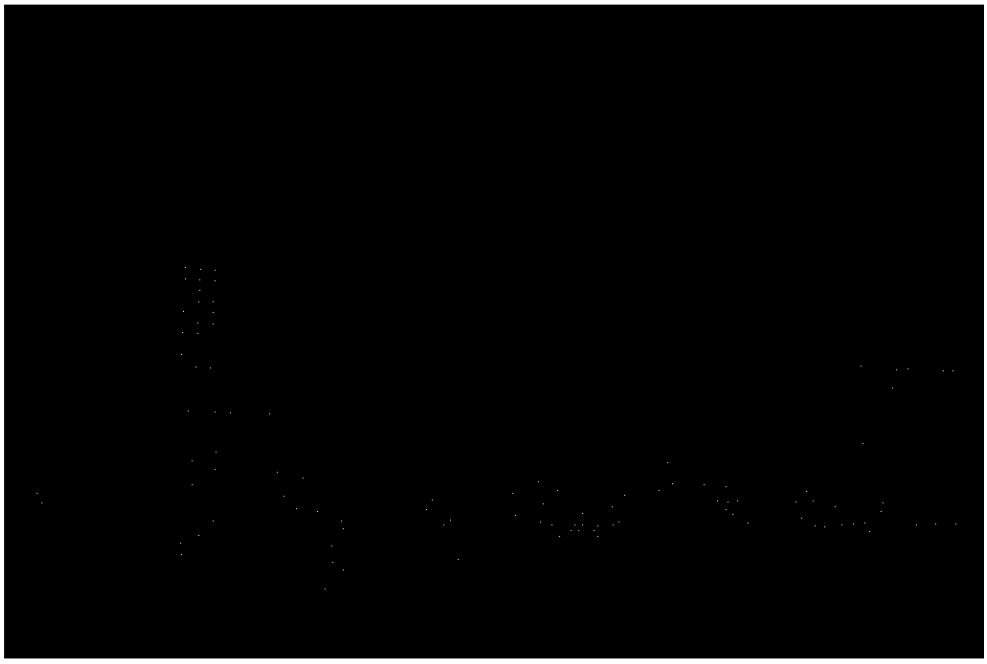
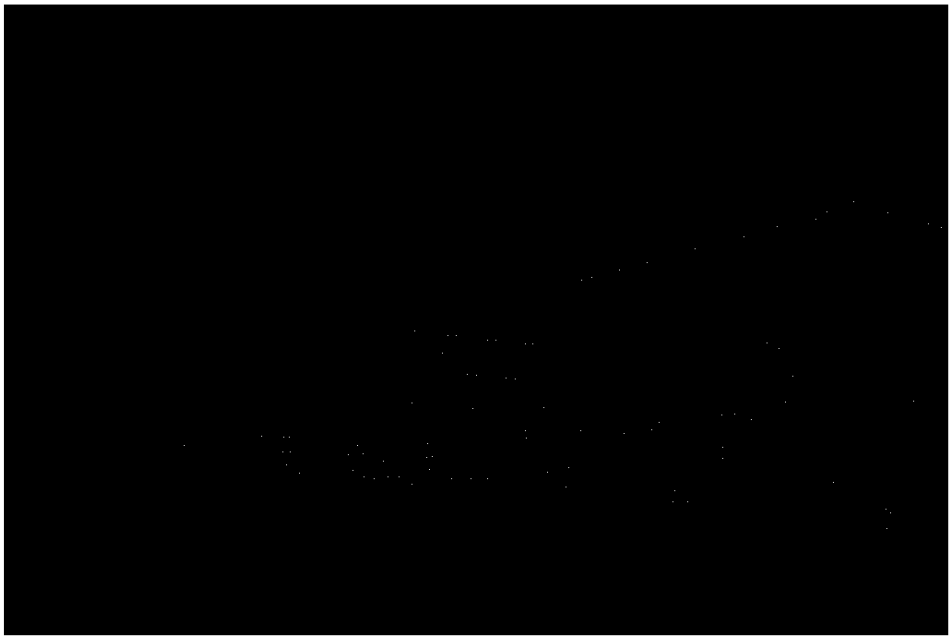


Image: Right Tower

Part 1b Non-maximum Suppression



Left Tower NMS



Right Tower NMS

Part 1c Patch Similarity

Difference between SSD and NCC:

SSD (sum of square differences) calculates the similarity score by using the sum of the differences of two patches and squaring it. This produces worse results than NCC (Normalized Cross Correlation) because unlike SSD, NCC is invariant to local average intensity and contrast. However, NCC is also slower than SSD, so I decided to use SSD to calculate the patch similarity.

Part 2a

Corresponding points and random points

995	543	922	214	25148623
858	546	626	462	63568883
848	545	893	225	53566412
190	287	566	462	44337265
186	575	793	444	53657166
898	459	525	514	44187646
699	501	987	430	127285127
467	539	402	514	87674426
293	514	574	368	68295065
846	519	509	438	81108145
448	518	962	551	72049718
982	383	443	520	58116830
561	541	828	367	39976647

187	343	412	495	99578566
201	379	307	499	98370368
328	530	1017	242	50785645
186	366	958	568	74069415
621	556	491	359	70663087
605	532	304	469	85870039
306	527	780	492	56138059
974	543	311	485	70364406
954	544	486	514	22657726
649	513	459	491	46327793
561	541	742	539	70330274
858	546	280	468	74770913
778	542	390	487	87655338
460	544	280	468	106265281
564	522	803	252	41309394
982	383	962	551	59793303
186	575	379	505	108987089

Random points and their similarity scores

Part 2b

RANSAC based method to find affine transformation

Experiment A1/ For 20 points

Affine Transformation:

0.9690

-0.0603

-399.0205

0.0273

0.9540

-30.9093

Average Distance/Error:

1.6684

Inliers:

443 432

525 364

534 364

379 505

446 354

491 359

476 378

429 512

374 488

443 520

482 359

402 514

280 468

390 487

417 512

384 478

391 512

Iterations:

1

For Experiment A1+A2/50 points

Affine Transformation:

0.8924

-0.0757

-322.3745

0.0274

0.9411

-23.4024

Average Distance/Error:

1.8760

Inliers:

443 432

525 364

534 364

379 505

446 354

491 359

476 378

429 512

374 488

443 520

482 359

402 514

390 487

417 512

384 478

391 512

486 514

Iterations:

1

(this one is lucky, many RANSACs usually take around 10 iterations)

Discuss what is the expected number of RANSAC iterations for each experiment and what is the actual number observed in practice:

Typically when RANSAC is run on the 20 corresponding points (experiment a1) it typically has around 1-4 iterations, a very small number as it finds the correct 3 points for the affine transformation quickly. This is larger than the expected amount of inliers as typically a RANSAC based method would need to iterate through a lot of different trios of points to find the correct ones. With our current assumptions this would most likely be around half of the total points (10) which is much larger than what is actually observed. For the second experiment (a1 +a2) this iteration count was increased to around 10 which is more like what we would have seen in a normal RANSAC method. Sometimes it even goes to a lower value (in this case 1 but it is very luck reliant), showing that even with random points RANSAC is still a good method of finding an affine transformation. This leads to the conclusion that if we get rid of the lower corresponding points it significantly reduces the time RANSAC needs to run.

Part 2c

Warped Images

Experiment A1/ For 20 points

For Experiment A1+A2/50 points

Explanation of Code

1a. For 1a I calculate the partial derivatives for the image using sobel filters (reusing code from previous homeworks). Using the partial derivatives I calculate the second moment matrix, and then using that calculate R (det(M) -a trace(M)) and use it as the new point of the new matrix. I then also calculate the 1000 biggest points in the harris matrix and use that for the input of the NMS.

1b. I used the same Non-Maximum suppression function from my previous homework on the image in order to leave only the points that are of interest to us.

1c. To find the corresponding points of the left and right image I use the SSD between all of the points of interest of the left and right nms and calculate their similarity score. I then sort these similarity scores in order to find the points with the lowest SSD scores (highest correlation) and I use the showMatchedFeatures to display theses matched points.

2a. I already calculate the 20 top points in 1c so I just use that for 2a1. For 2a2 I also generate random points from the points of all correspondences.

2b.In order to do 2b I first generate three random points from the set I am given (either 20 or 50 points) and then find the affine transformation for those points. Then for each first point I apply the affine transformation and check the distance between it and the second point (from pointsall). Then, since it is a RANSAC method, I check if the amount of points that are a distance smaller than the threshold distance are equal to the inliers (user inputted). If they are then this affine transformation works, otherwise keep running the function until a good affine transformation is found.

2c. After we generate the affine transformation we need to actually apply it to the image and generate 2 images that are “stitched together”. To accomplish this I use the maketform and imtransform methods. I then create a new image that will hold the right image and the left image transformed onto it. I then composite the two images where they overlap by finding rhe average of the two pixels in that point.

Main Function:
Main function takes in image1,image2,inliers, threshold where the two images are the left and right image, inliers are the number of inliers that the affine transformation needs to have to be picked, and the threshold is the distance maximum that the inliers can to be from the transformed point in order to be considered an inlier. It returns 7 figures, all correlating to each respective part. I also display the affine transformation, average, inliers, and iterations for each RANSAC. I display these values in the console