Homework 12

Meta information	
Name	Savan Kiran
Program	Masters in Computer Science
Questions skipped	N/A
Questions substituted	N/A
Extra credit questions	N/A

HW 11 – Part E (grad question)

The last week of classes was hectic for me. I had finished up the undergrad part earlier but had the grad part left for the final day. Despite all the attempt, I couldn't complete the task on time. I'm doing Part E for approx. 20% of HW11 (which is worth 2 points of this make-up).

For image stitching, we need to first identify features from the image using SIFT and match them. We will then find Homography using RANSAC. For stitching, we draw the first image first and then for each pixel in the second image, we calculate the new position using homography. If the pixel overlaps with the first image, we calculate the mean of RGB value between both images else we only take RGB value from the second image. Because of planar distortion, there are black areas between pixels where second image is getting drawn. However, we can see that the homography perfectly maps the pixels from the second image w.r.t the first image. We see the roads, road lines, buildings, align perfectly. I tried stitching with two pairs of images and the results are shown below.

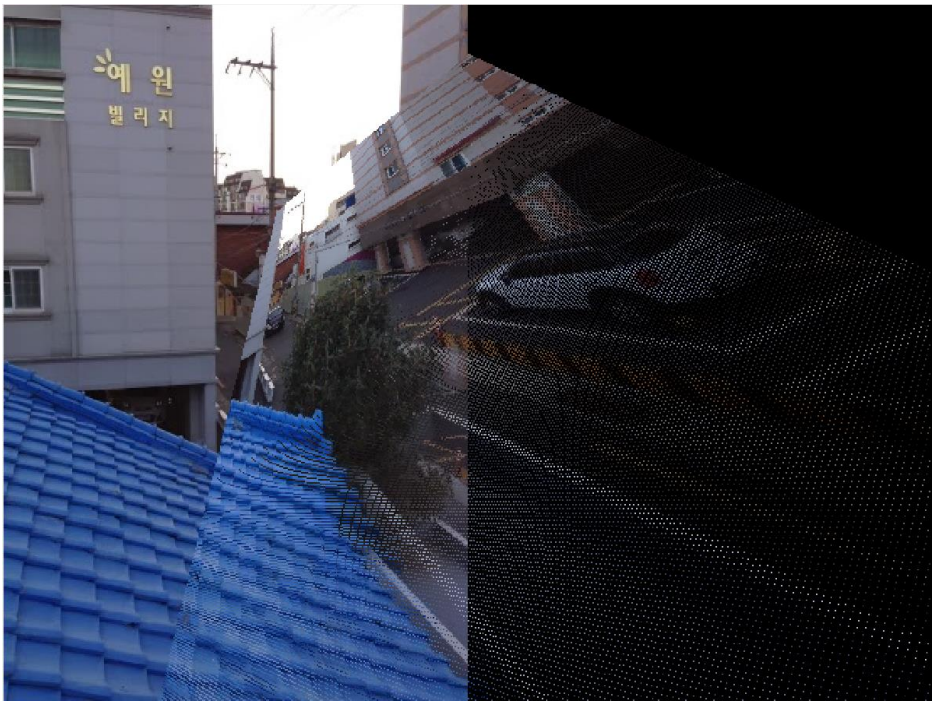

Figure 1. Image 1 (buildings and road) and Image 2 (buildings, road and car) stitched together

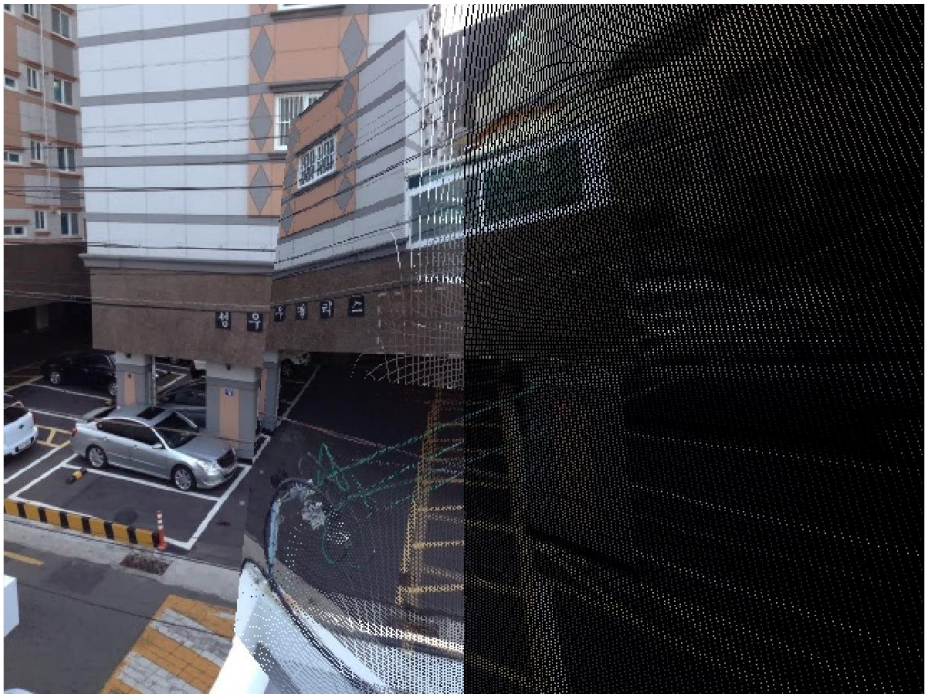

Figure 2. Image 1 (buildings, road and vehicles) and Image 2 (buildings, road and vehicles) stitched together

Below is the code snippet,

```
function hw11_partE()
im1=imread('S1.jpg');
im2=imread('S2.jpg');
H=homo_ransac('S1.jpg','S2.jpg');

im=zeros(max(size(im1,1), size(im2,1)), size(im1,2)+size(im2,2));
for i=1:size(im1,1)
    for j=1:size(im1,2)
        im(i,j,1)=im1(i,j,1);
        im(i,j,2)=im1(i,j,2);
        im(i,j,3)=im1(i,j,3);
    end
end

for i=1:size(im2,1)
    for j=1:size(im2,2)
        m=map([i,j,1],H);
        r=uint16(m(1));
        c=uint16(m(2));
        if r > 0 && r <= size(im,1) && c > 0 && c < size(im,2)
            if c+size(im1,2) > size(im1,2)
                im(r,c,1)=im2(i,j,1);
                im(r,c,2)=im2(i,j,2);
                im(r,c,3)=im2(i,j,3);
            else
```

```
                im(r,c,1)=mean([im(r,c,1) im2(i,j,1)]);
                im(r,c,2)=mean([im(r,c,2) im2(i,j,2)]);
                im(r,c,3)=mean([im(r,c,3) im2(i,j,3)]);
            end
        end
    end
end
im=uint8(im);

figure;
imshow(im);
end
```

HW 9 – Part A Q5

Here, I misunderstood the question. Instead of looking for the best match over all the slides for every keypoint in a frame, I was looking for best match only in the corresponding slide match. I'm re-doing Part A Q5 for approx. 10% of HW9 (which is worth 1 points of this make-up).

From previous knowledge, I knew the min distance (angular) between keypoints was roughly 0.1-0.2, the mean was roughly 0.4-0.6 and max was roughly 1.0-1.1. With this information, I set the threshold for a good match to be T=0.85. Any match <=T is considered a good match and others are discarded in the measurement.

For each keypoint in the frame, I calculate the best match in each slide and note their distances. We pick the best match, i.e., match with least distance and increment the count for that frame-slide pair. We repeat this for all the frames and the resulting confusion matrix is shown below,

	Slide 1	Slide 2	Slide 3
Frame 1	134	46	33
Frame 2	150	98	16
Frame 3	59	18	406

Table 1. Confusion matrix for keypoint matches between frame & slide

From the above table, we can see that clearly, frame 1 resembles slide and frame 3 resembles slide 3 because they have the maximum match count. However, for frame 2, it strangely resembles slide 1 rather than slide 2.

```
%HW 9 - Part A - Q5
[f1s1 f1s2 f1s3]=hw9_partA_q5('frame1.pgm');
[f2s1 f2s2 f2s3]=hw9_partA_q5('frame2.pgm');
[f3s1 f3s2 f3s3]=hw9_partA_q5('frame3.pgm');

CM=[f1s1 f1s2 f1s3; f2s1 f2s2 f2s3; f3s1 f3s2 f3s3];
disp('Match count confusion matrix: ');
disp(CM);

function [s1_match, s2_match, s3_match]=hw9_partA_q5(frame_pgm)
if1=imread(frame_pgm);
```

```
if1=single(if1);
[ff, df] = vl_sift(if1);

is1=imread('slide1.pgm');
is1=single(is1);
[fs1, ds1] = vl_sift(is1);

is2=imread('slide2.pgm');
is2=single(is2);
[fs2, ds2] = vl_sift(is2);

is3=imread('slide3.pgm');
is3=single(is3);
[fs3, ds3] = vl_sift(is3);

euclidean=0;
s1_match=0;
s2_match=0;
s3_match=0;
T=0.85;
for i=1:size(df,2)
    d1 = df(:,i)';
    [n1, n1_dist, x, y] = find_nearest_neighbor(d1, ds1, euclidean);
    [n2, n2_dist, x, y] = find_nearest_neighbor(d1, ds2, euclidean);
    [n3, n3_dist, x, y] = find_nearest_neighbor(d1, ds3, euclidean);
    n_dist=min([n1_dist n2_dist n3_dist]);
    if n_dist <= T
        if n_dist == n1_dist
            s1_match=s1_match+1;
        elseif n_dist == n2_dist
            s2_match=s2_match+1;
        else
            s3_match=s3_match+1;
        end
    end
end
end
```

HW 9 – Part B (grad question)

I implemented the Harris corner detection and provided an instance to show that the corner detection is rotation invariant. However, I missed to provide a reasonable justification for it. I'm re-doing Part B for approx. 5% of HW9 (which is worth 0.5 points of this make-up).

When the image rotates, the corner just rotates but doesn't change the shape. This can be confirmed from eigen vectors and eigen values. The ellipse (eigen vectors) changes but the shape (eigen values) remains the same. Hence the corner response R is rotation invariant.

HW 7 – Part A Q5 and Q6

There was an error in r,g RMS calculation for Q5. I should have squared (r2-r) and (g2-g) separately and added them rather than adding them and squaring the sum. For Q6, my angular error calculation for problem 6 is computing the error between the max rgb values for two images, rather than the max RGB for the image with unknown light and the known value of our canonical light. I'm re-doing Part A Q5 and Q6 for approx. 10% of HW7 (which is worth 1 points of this make-up).

The fix for RMS error calculation was straight forward and the code snippet below depicts the same with the fix highlighted in green.

```
function[rms]=rms_error(image,newImage)
SquaredErrSum=0;
pixelCount=0;
for i=1:size(image,1)
    for j=1:size(image,2)
        p=image(i,j,:);
        p=[p(1,1,1) p(1,1,2) p(1,1,3)];
        p2=newImage(i,j,:);
        p2=[p2(1,1,1) p2(1,1,2) p2(1,1,3)];
        if((p(1,1)+p(1,2)+p(1,3))>10 && (p2(1,1)+p2(1,2)+p2(1,3))>10)
            r=double(p(1,1)/(p(1,1)+p(1,2)+p(1,3)));
            g=double(p(1,2)/(p(1,1)+p(1,2)+p(1,3)));
            r2=double(p2(1,1)/(p2(1,1)+p2(1,2)+p2(1,3)));
            g2=double(p2(1,2)/(p2(1,1)+p2(1,2)+p2(1,3)));
            e=(r2-r)^2+(g2-g)^2;
            SquaredErrSum=SquaredErrSum+e;
            pixelCount=pixelCount+1;
        end
    end
end
rms=sqrt(SquaredErrSum/pixelCount);
end
```

The resultant RMS error is shown below.

```
RMS error(r,g) b/w original & canonical image: 0.24982
RMS error(r,g) b/w corrected & canonical image: 0.87282
```

I fixed the angular error now to compute the error between maxRGB of the image under unknown light and the maxRGB of the image under canonical light.
Below is the code snippet that does that,

```
function hw7_partA_q6(origImagePath,canonImagePath,text)
origImage=imread(origImagePath);
canonImage=imread(canonImagePath);
[lr1,lg1,lb1]=max_RGB(origImage);
[lr2,lg2,lb2]=max_RGB(canonImage);
D=[lr1/lr2 0 0; 0 lg1/lg2 0; 0 0 lb1/lb2];
newImage=correct_image(D,origImage);
```

```
[lr1,lg1,lb1]=max_RGB(newImage);
l1=[lr1 lg1 lb1];
[lr2,lg2,lb2]=max_RGB(canonImage);
l2=[lr2 lg2 lb2];
ae=angular_error(l1,l2);
disp(['Angular error for ' text ': ' num2str(ae)]);
end
```

Below are the results of the angular error measurements,

```
Angular error for apple: 20.1833
Angular error for ball: 36.9687
Angular error for blocks: 14.9115
```

HW 8 – Part B1 (grad question)

I had implemented edge detection with non-maximal suppression and edge linking. However, I forgot to discuss the effect of sigma and thresholds on edge detection. I'm re-doing Part B1 for approx. 10% of HW8 (which is worth 1 points of this make-up).

We implement edge detection using non-maximal suppression and edge linking which is explained in detail in HW8 partB1. Here we will discuss the effects of sigma and threshold on the edge detection.

If we do edge detection without using gaussian blur, we end up with many probable edges and it is very difficult to distinguish between the actual edge and noise. Gaussian blur reduces noise and helps improve edge detection. However, as we increase sigma, the edges that are close by merge to form thicker edges and we might lose subtle edges. The right way is to experiment with different values of sigma and see what works for the image considering what edges do you want to be highlighted.

Similarly, at each image point, we can apply a threshold on whether an edge is present or not. With lower threshold, more edges are found however, we also include a lot of noise. With higher threshold we might lose some subtle edges and hence lose edge connectivity.

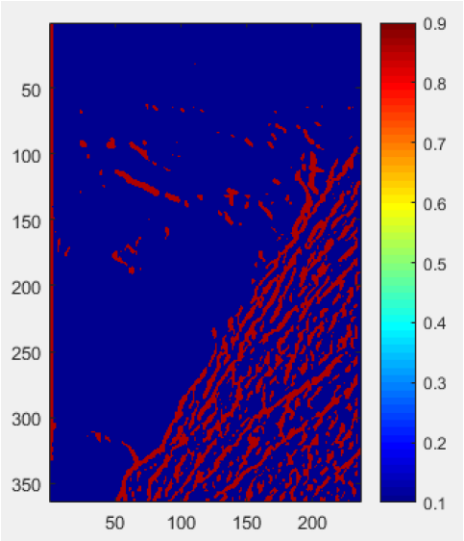

Figure 3. Edge detection using non-maximal suppression and edge linking

HW 3 – Part B2 (grad question)

Verify if the image is in perspective or not by drawing parallel lines over the image. The lines I had drawn were not parallel. I had looked up the solution when it was posted (long ago). I'm attempting it again hoping I remember my mistakes from then. I'm re-doing Part B2 for approx. 15% of HW8 (which is worth 1.5 points of this make-up).

The parallel lines drawn below intersect at a vanishing. We make an approximate judgement here for the vanishing points. The horizon line passes through all the vanishing points exactly.

1. Each set of parallel lines (yellow, blue and green) meets at different vanishing points.
2. The set of parallel lines indicated by blue, yellow and orange are all on the same plane and their vanishing points are collinear. The line joining all 3 vanishing points forms one of the horizons for the image and is indicated in black in Figure 3.

The scale and perspective hold correct because of the satisfaction of above two conditions. Therefore, the image is real.

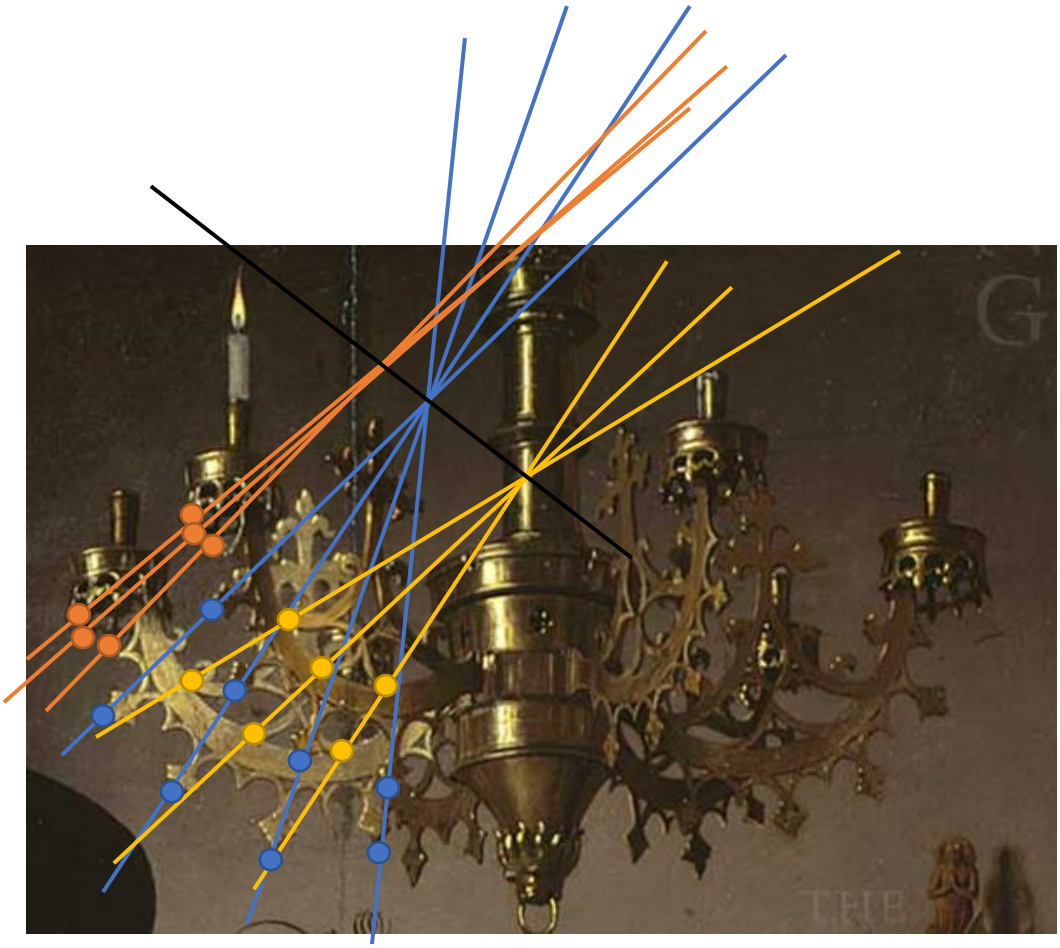

Figure 4. Set of parallel lines showing the vanishing points and the horizon marked in black for the image

HW 3 – Part A2 (grad question)

I was not able to provide a formal mathematical argument earlier. I'm re-doing Part A2 for approx. 10% of HW8 (which is worth 1 points of this make-up).

We know that Non-homogenous LLS gives us the global minimum for the vertical error. Shown below,

$$Non-homogeneous\ LLS\ error = \min\left(\sum\left[y_i - \left(m_{opt}x_i + b_{opt}\right)\right]^2\right)$$

On the other hand, homogenous LLS gives us the global minimum for the perpendicular error.

$$Homogeneous\ LLS\ error = \min\left(\sum\left(d_{opt} - a_{opt}x_i - b_{opt}y_i\right)^2\right)$$

Which means, for all possible lines,

In case of Non-homogenous LLS,

$$\left(\sum[y_i - (mx_i + b)]^2\right) \geq \left(\sum\left[y_i - \left(m_{opt}x_i + b_{opt}\right)\right]^2\right)\ \forall m, b$$

In case of Homogenous LLS,

$$\left(\sum(d - ax_i - by_i)^2\right) \geq \left(\sum\left(d_{opt} - a_{opt}x_i - b_{opt}y_i\right)^2\right) \forall d, a, b$$

If we consider H_{opt} line with Non-homogenous measurement,

$$y = \frac{d_{opt} - a_{opt}x}{b_{opt}}$$

$$\left(\sum\left[y_i - \frac{d_{opt} - a_{opt}x}{b_{opt}}\right]^2\right) \geq \left(\sum\left[y_i - \left(m_{opt}x_i + b_{opt}\right)\right]^2\right)\ from\ above\ eqn$$

Similarly, for NH_{opt} line with Homogeneous measurement, it is error with NH_{opt} line >= Homogenous LLS error.

The question was asked for RMS of Non-homogenous error which is nothing but Non-homogenous error divided by 'n' -> number of points. And same for RMS Homogeneous error. Since, 'n' is a constant, the above proof still holds.

Therefore, both non-homogeneous and homogeneous LLS can only find local minimum for 'D' (perpendicular) RMS error and 'Y' (vertical) RMS error respectively.