CE4003 Computer Vision Lab 2 Assignment Report

Name: Gavin Neo Jun Hui

Matric number: U1921265L

3.1 Edge Detection

3.1a. Download and display macritchie.jpg

```
Pc = imread('macritchie.jpg');
P = rgb2gray(Pc);
figure;imshow(P);
```

3.1b. Creating 3x3 Sobel masks and filter the image. Display the edge-filtered images.

Creating 3x3 horizontal and vertical Sobel masks:

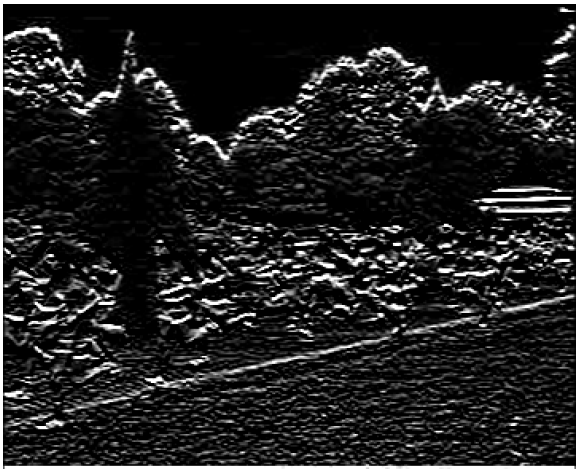
```
%horizontal mask
hh = [-1,-2,-1;
       0,0,0;
       1,2,1];

%vertical mask
hv = [-1,0,1;
      -2,0,2;
      -1,0,1];
```

Filter the image using conv2:

```
%filter the images
Ph = conv2(double(P), double(hh));
Pv = conv2(double(P), double(hv));
```

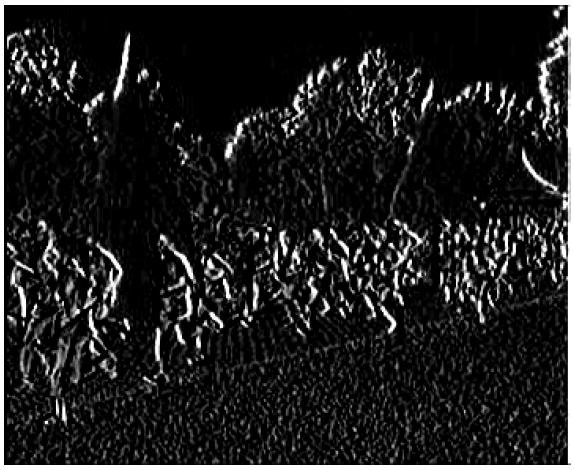
Display image filtered using horizontal mask filter:

```
%image with horizontal mask filter
figure;imshow(uint8(Ph));
```



Display image filtered using vertical mask filter:

```
%image with vertical mask filter
figure;imshow(uint8(Pv));
```



What happens to edges which are not strictly vertical nor horizontal, i.e. diagonal?

It could still be detected, as diagonal is made of horizontal and vertical vectors. However, if the diagonal edges are closer to the horizontal than vertical edges, the horizontal Sobel mask can detect better and vice versa. An example is the line across the running track, which is better detected using the horizontal Sobel mask as compared to the vertical Sobel mask.

3.1c. Generating combined edge image.

```
Pcombined = imadd((Ph.^2),(Pv.^2));
Pcombined = sqrt(Pcombined);
```

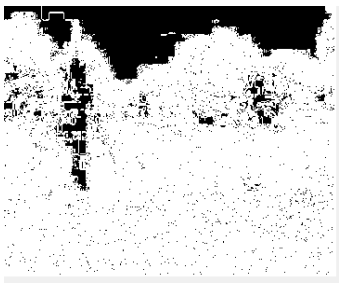
Suggest a reason why a squaring operation is carried out.

This is to take the negative value into consideration. Calculation will be easier when squaring first, followed by addition and lastly, square root the result.

3.1d. Creating binary images with different threshold values t.

t=10:

```
E10 = Pcombined>10;
figure;imshow(E10);
```



t=50:

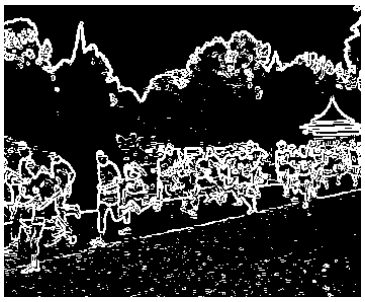
```
E50 = Pcombined>50;
figure;imshow(E50);
```

t=100:

```
E100 = Pcombined>100;
figure;imshow(E100);
```

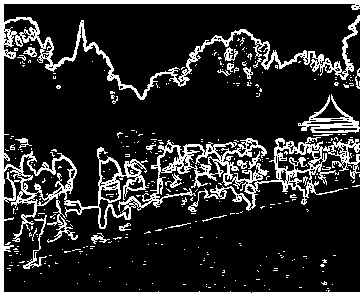
t=150:

```
E150 = Pcombined>150;
figure;imshow(E150);
```



t=200:

```
E200 = Pcombined>200;
figure;imshow(E200);
```



t=300:

```
E300 = Pcombined>300;
figure;imshow(E300);
```

t=400:

```
E400 = Pcombined>400;
figure;imshow(E400);
```

What are the advantages and disadvantages of using different thresholds?

A suitable threshold will allow us to retrieve the required information and understand it clearly. If the threshold value selected is too low, there will be too many details extracted, which is hard visualize the image. On the other hand, if the threshold value selected is too high, too much information will be omitted from the binary image.

3.1e. Using Canny edge detection algorithm.

tl=0.04, th=0.1, sigma=1:

```
tl = 0.04;
th = 0.1;

%sigma = 1.0
E1 = edge(P, 'canny', [tl th], 1);
figure; imshow(E1);
```

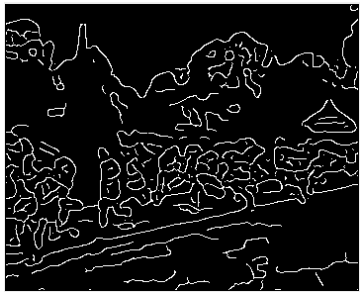
3.1e(i). Trying different values of sigma from 1 to 5

Sigma=2:

```
%sigma = 2.0
E2 = edge(P, 'canny', [tl th], 2);
figure; imshow(E2);
```

Sigma=3:

```
%sigma = 3.0
E3 = edge(P, 'canny', [tl th], 3);
figure; imshow(E3);
```

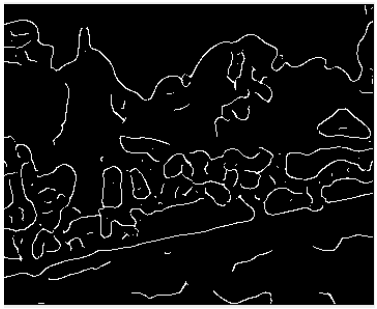


Sigma=4:

```
%sigma = 4.0
E4 = edge(P, 'canny', [tl th], 4);
figure; imshow(E4);
```

Sigma=5:

```
%sigma = 5.0
E5 = edge(P, 'canny', [tl th], 5);
figure; imshow(E5);
```



Try different values of sigma ranging from 1.0 to 5.0 and determine the effect on the edge images. What do you see and can you give an explanation for why this occurs? Discuss how different sigma are suitable for (a) noisy edgel removal, and (b) location accuracy of edgels.

Sigma is the standard deviation of the Gaussian filter. At low sigma, the resultant image are less blurry and allow detection of small, sharp lines. At large sigma, less details are recognized and the image will be more blurry.

Thus, a high sigma is better for noisy edgel removal, as it blurs the image and remove noise meanwhile. A low sigma value will be better for the location accuracy of the edgels, detecting the small, sharp lines.

3.1e(i). Trying different values of tl.

Tl = 0.01:

```
%tl = 0.01
F1 = edge(P, 'canny', [0.01 th], 1);
figure; imshow(F1);
```

Tl=0.02

```
%tl = 0.02
F2 = edge(P, 'canny', [0.02 th], 1);
figure; imshow(F2);
```

Tl=0.03:

```
%tl = 0.03
F3 = edge(P, 'canny', [0.03 th], 1);
figure; imshow(F3);
```

Tl=0.04:

```
%tl = 0.04
F4 = edge(P, 'canny', [0.04 th], 1);
figure; imshow(F4);
```

Tl=0.05:

```
%tl = 0.05
F5 = edge(P, 'canny', [0.05 th], 1);
figure; imshow(F5);
```

Tl=0.08:

```
%tl = 0.08
F6 = edge(P, 'canny', [0.07 th], 1);
figure; imshow(F6);
```

Tl=0.10:

```
%tl = 0.1
F7 = edge(P, 'canny', [0.09 th], 1);
figure; imshow(F7);
```

Try raising and lowering the value of tl. What does this do? How does this relate to your knowledge of the Canny algorithm?

This sets the minimum threshold of the algorithm. If it sets too low, falsely identified irrelevant information such as noise will be included into the image. If is too high, the image might be missing some important information.

3.2 Line Finding using Hough Transform

3.2a. Reusing Canny algorithm with sigma = 1.0

```
E = edge(P, 'canny', [0.04 0.1], 1);
```

3.2b. Using Radon transform

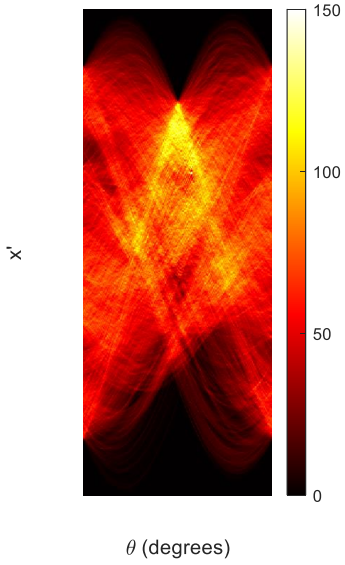
```
theta = 0:179;
[H, xp] = radon(E);
```

As there is no function available to compute the Hough transform in MATLAB, we will use the Radon transform, which for binary images is equivalent to the Hough transform. Read the help manual on Radon transform, and explain why the transforms are equivalent in this case. When are they different?

The transforms are equivalent as the image is binary. The Radon function will return the Radon transform if the intensity image I for the angel theta degrees, like how Hough transform works. However, if is not a binary image, Radon transform will compute using the projection of the image intensity along a radical line oriented at specific angle. Thus the 2 transforms will then be different from each other.

Displaying the image H:

```
figure; imshow(H,[],'Xdata',theta,'Ydata',xp,'InitialMagnification','fit');
xlabel('\theta (degrees)');
ylabel('x''');
colormap(gca,hot), colorbar
```

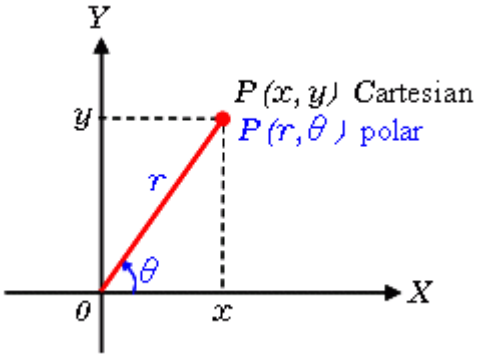


3.2c. Finding the location of maximum pixel intensity in the Hough image.

```
[a,b] = max(H(:));
[x_max, y_max] = ind2sub(size(H), b);
theta = theta(y_max);
radius = xp(x_max);
```

3.2d. Derive equation Ax + By = C in image coordinates. Show how A and B can be obtained.

```
[A, B] = pol2cart(theta*pi/180,radius);
B = -B;
```



B needs to be negated as unlike the image above, the y-axis will be pointing downwards.

Finding C with respect to origin in the top-left corner of the image.

```
%find C
C = -(radius/sin(theta*pi/180));

%convert back to image coordinates
[Height, Width] = size(E);
x_origin = -Width/2;
y_origin = (-A/B)*x_origin + C;
C_origin = y_origin + (Height/2);
```

3.2e. Compute yl and r for xr=0 and xr = width of image – 1.

```
xl = 0;
xr = Width-1;
yl = (-A/B)*xl + C_origin;
yr = (-A/B)*xr + C_origin;
```

3.2f. Display image and superimpose the estimated line.

```
figure;imshow(Pc);
line([xl xr], [yl yr])
```

Does the line match up with the edge of the running path? What are, if any, sources of errors? Can you suggest ways of improving the estimation?

Yes, the line matches with the edge of the running path.

3.3 3D Stereo

3.3a. Write disparity map algorithm function script.

```
function D = disparity_map(Pl, Pr, y, x)

%retrieve paameters
half_h = floor(y / 2);
half_w = floor(x / 2);
[Pl_h, Pl_w] = size(Pl);

%initialize D with matrixx of ones
D = ones(Pl_h - y + 1, Pl_w - x + 1);

%Calculate disparity
for i = half_h+1:Pl_h - half_h
    for j = half_w+1:Pl_w - half_w

        T = Pl(i-half_h:i+half_w,j-half_h:j+half_w);
        l = j-14;
        r = j;
        if l < half_w+1
            l = half_w+1;
        end

        min_xr = l;
        min_ssd = Inf;

        for xr = l:r
            I = Pr(i-half_h:i+half_w,xr-half_h:xr+half_w);
            ssd1 = ifft2(fft2(I) .* fft2(rot90(I, 2)));
            ssd1 = ssd1(y, x);
            ssd2 = ifft2(fft2(T) .* fft2(rot90(I, 2)));
            ssd2 = ssd2(y, x) * 2;
            ssd = ssd1 - ssd2;
            if ssd < min_ssd
                min_ssd = ssd;
                min_xr = xr;
            end
        end
        D(i, j) = j - min_xr;
    end
end
end
```

3.3b. Download and display corridorl.jpg and corridor.jpg.

```
Pcl = imread('corridorl.jpg');
Pl = rgb2gray(Pcl);
figure;imshow(Pl);

Pcr = imread('corridorr.jpg');
Pr = rgb2gray(Pcr);
figure;imshow(Pr);
```

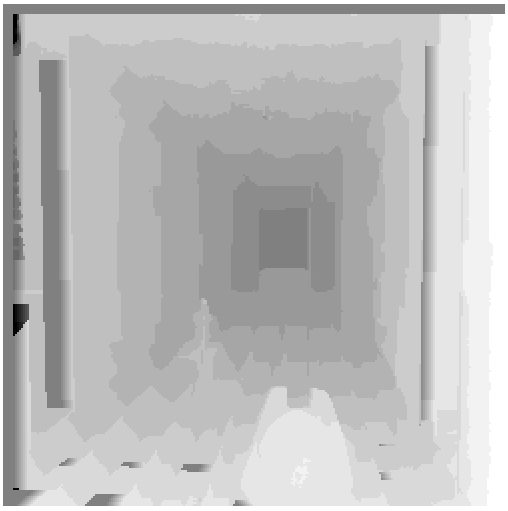
corridorl.jpg:

corridor.jpg:

3.3c. Obtain the disparity map.

```
%referece image
ref = imread('corridor_disp.jpg');
figure;imshow(ref);

%disparity map
D = disparity_map(Pl, Pr, 11, 11);
figure;imshow(D, [-15 15]);
```

Reference image corridor_disp.jpg:



Disparity map:

Comment on how the quality of the disparities computed varies with the corresponding local image structure.

Both the images are almost like one another, where the points nearer to the camera have lighter colours as compared to points further away. However, at the deepest part in the disparity map, it should show the darkest colour, like how the reference image is. Since the deepest part is a white wall, it is very hard to find the depth as the neighbouring pixels have the same colour intensity, leading to an incorrect result.

3.3d. Rerun algorithm on triclops-i2l.jpg and triclops-i2r.jpg.

```
%imread images
Tcr = imread('triclopsi2l.jpg');
Tl = rgb2gray(Tcr);
figure;imshow(Tl);

Tcr = imread('triclopsi2r.jpg');
Tr = rgb2gray(Tcr);
figure;imshow(Tr);

%reference image
ref = imread('triclopsid.jpg');
figure;imshow(ref);

%disparity map
D = disparity_map(Tl, Tr, 11, 11);
figure;imshow(D, [-15 15]);
```

triclops-i2l.jpg:

triclops-i2r.jpg:

Reference image triclops-id.jpg:

Disparity map:

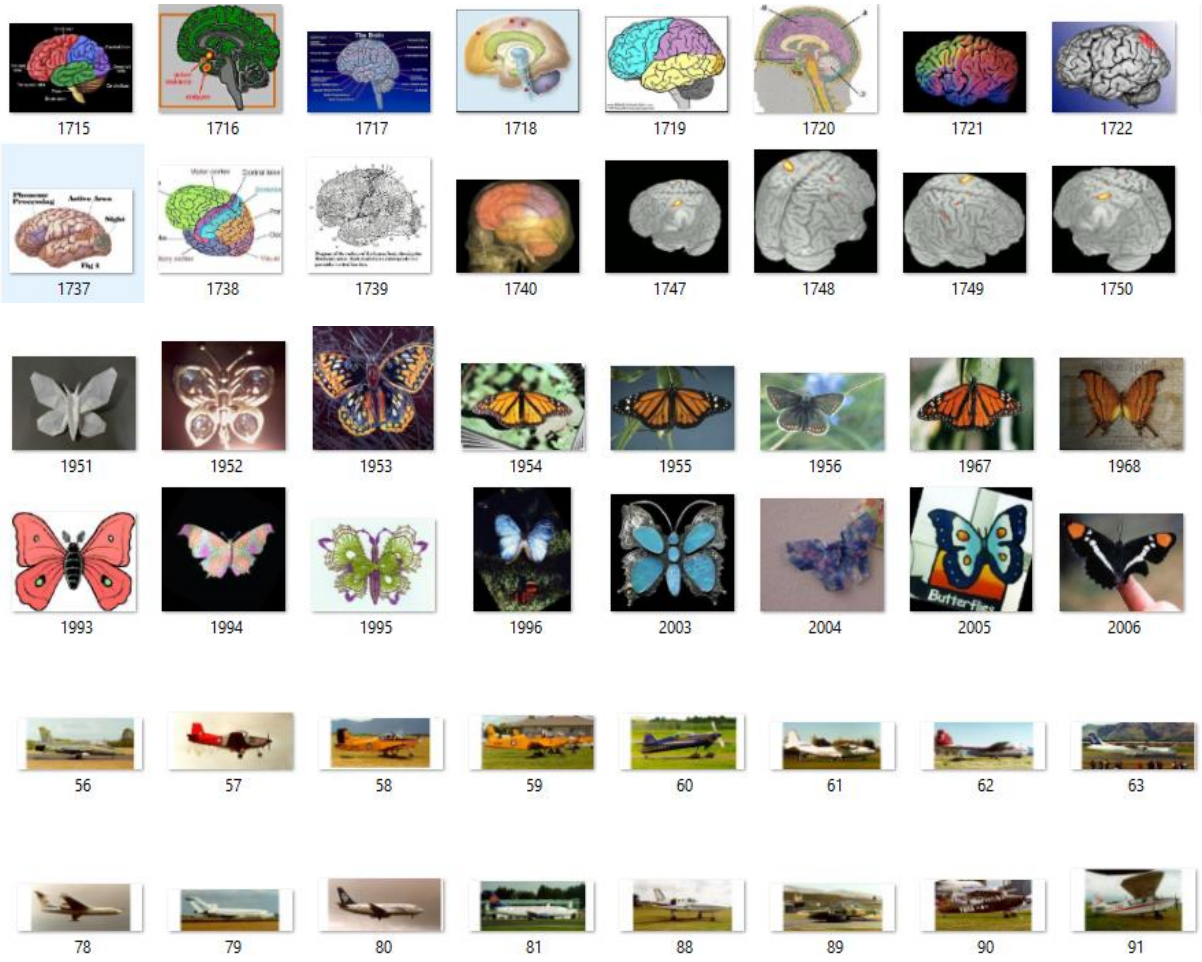
How does the image structure of the stereo images affect the accuracy of the estimated disparities?

The disparity map shows error on the pavement area. This is due to the slight geometric distortion. The slight distortion can affect the minimum SSD calculation, which causes slight error to the resultant disparity map. In order to have better accuracy, the 2 image patches must look almost identical, with minimum geometric distortion, Lambertian reflectance and no occlusion. Also, the cameras must be close, with a small baseline.

3.4 Optional

You will need to implement the algorithm in the CVPR 2006 paper entiled “Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories”. You will need to use the benchmark Caltech-101 dataset and compare the classification results of Spatial Pyramid Matching (SPM) and the bag-of-words (BoW) method as in Table 2 of the paper by following the experimental setting in Section 5.2 of the paper.

Preparing Caltech-101 dataset:



Have prepared 4 classes (brain, butterfly, plane and plant) of images for both training and testing. Each class consists of 30 images, thus having total of 120 images for both training and testing.

Source code:

The code is sourced from github: https://github.com/lipiji/PG_BOW_DEMO.

It implements the algorithm for bag-of-words (BoW) and Spatial Pyramid Matching (SPM).

Main:

```
%% initialize the settings
display('********** start *********')
clc;
clear;
ini;
detect_opts=[];descriptor_opts=[];dictionary_opts=[];assignment_opts=[];ada_opts=[];
```

```
%% Descriptors
descriptor_opts.type='sift';                                        % name descripto
descriptor_opts.name=['des',descriptor_opts.type]; % output name (combines detector and descrtiptor name)
descriptor_opts.patchSize=16;                                       % normalized patch size
descriptor_opts.gridSpacing=8;
descriptor_opts.maxImageSize=1000;
GenerateSiftDescriptors(pg_opts,descriptor_opts);
```

```
%% Create the texton dictionary
dictionary_opts.dictionarySize = 300;
dictionary_opts.name='sift_features';
dictionary_opts.type='sift_dictionary';
CalculateDictionary(pg_opts, dictionary_opts);
```

```
%% assignment
assignment_opts.type='1nn';                                    % name of assignment method
assignment_opts.descriptor_name=descriptor_opts.name;          % name of descriptor (input)
assignment_opts.dictionary_name=dictionary_opts.name;          % name of dictionary
assignment_opts.name=['BOW_',descriptor_opts.type];            % name of assignment output
assignment_opts.dictionary_type=dictionary_opts.type;
assignment_opts.featuretype=dictionary_opts.name;
assignment_opts.texton_name='texton_ind';
do_assignment(pg_opts,assignment_opts);
```

```
%% CompilePyramid
pyramid_opts.name='spatial_pyramid';
pyramid_opts.dictionarySize=dictionary_opts.dictionarySize;
pyramid_opts.pyramidLevels=3;
pyramid_opts.texton_name=assignment_opts.texton_name;
CompilePyramid(pg_opts,pyramid_opts);
```

```
%% Classification
do_classification_rbf_svm
```

```
%% pyramid bow rbf
do_p_classification_rbf_svm
```

BoW:

```
train_labels    = labels(trainset);          % contains the labels of the trainset
train_data      = BOW(:,trainset)';          % contains the train data
[train_labels,sindex]=sort(train_labels);    % we sort the labels to ensure that the first label is '1', the second '2' etc
train_data=train_data(sindex,:);
test_labels     = labels(testset);           % contains the labels of the testset
test_data       = BOW(:,testset)';           % contains the test data
```

SPM:

```
%% sift
load([pg_opts.globaldatapath,'/',pyramid_opts.name])
train_labels    = labels(trainset);          % contains the labels of the trainset
train_data      = pyramid_all(:,trainset)';          % contains the train data
[train_labels,sindex]=sort(train_labels);    % we sort the labels to ensure that the first label is '1', the second '2' etc
train_data=train_data(sindex,:);
test_labels     = labels(testset);           % contains the labels of the testset
test_data       = pyramid_all(:,testset)';           % contains the test data
```

Results:

```
Classification using BOW rbf_svm
Accuracy = 83.3333% (100/120) (classification)

Classification using Pyramid BOW rbf_svm
Accuracy = 88.3333% (106/120) (classification)
```

The results show that the SPM generally has a higher accuracy than BoW.

References

1. https://github.com/lipiji/PG_BOW_DEMO.