

Lab 2 Report

1. Introduction

In this lab, several advanced image processing and computer vision techniques were explored. Firstly, edge processing methods including Sobel and Canny were explored. Then, Hough Transform was used to join broken edge pixels on the result of Canny edge detector. Secondly, sum-of-squares difference (SSD) was being experimented for the estimation of disparity map. Lastly, using the benchmark Caltech-101 dataset, the performance of image classification task using the bag-of-words method was compared with the spatial pyramid matching (SPM) method.

2. Tools

For the experiments on edge processing, Hough Transform and sum-of-squares difference (SSD), **MATLAB** was being used. For the experiment on the bag-of-words method and the spatial pyramid matching (SPM) method, **python** was being used.

3. Experiments

3.1 Edge Detection

1. Download “macritchie.jpg” and convert the image to grayscale.

```
img = imread("images/maccropped.jpg");  
img = rgb2gray(img);
```

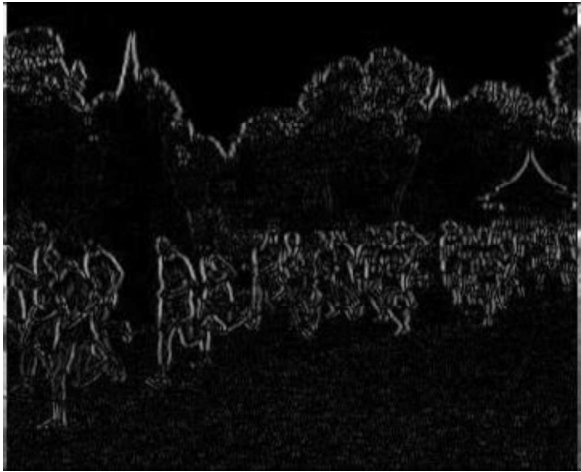
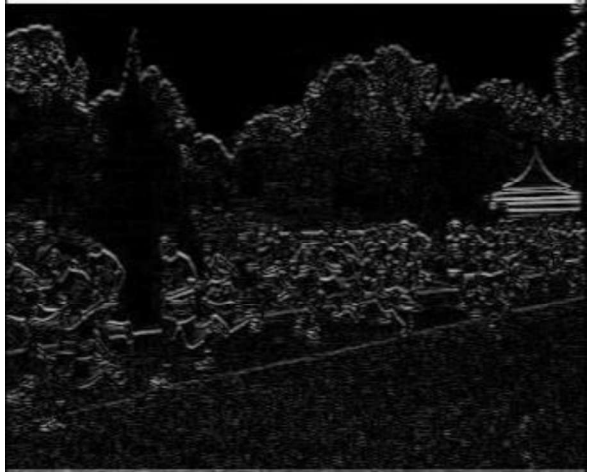


2. Create 3×3 horizontal and vertical Sobel masks and filter the image. Display the edge-filtered images. What happens to the edges which are not strictly vertical nor horizontal?

```
function edges = normalize_edge(edges)
    edges = abs(edges);
    edges = edges ./ max(edges(:));
    edges = edges .* 255;
end

x_sobel = [-1, 0, 1; -2, 0, 2; -1, 0, 1];
y_sobel = x_sobel';
gx = conv2(double(img), double(x_sobel));
gy = conv2(double(img), double(y_sobel));
imwrite(uint8(normalize_edge(gx)), "results/01_gx.png");
imwrite(uint8(normalize_edge(gy)), "results/01_gy.png");
```

The result of convolution between the image and the filters G_x and G_y may have negative values and may have absolute values greater than 255. Therefore, the absolute value of the filtered image was taken and normalized to the range of 0 – 255 before displayed.

G_x	G_y
$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Calculate gradient in the horizontal direction, which results in vertical edges	Calculate gradient in the vertical direction, which results in horizontal edges
	

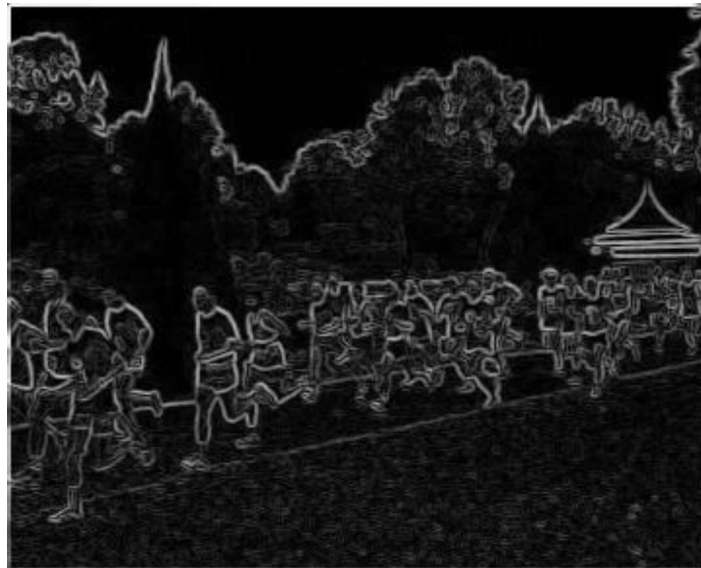
G_x filtered image showed clear vertical edges whereas G_y filtered image showed clear horizontal edges. However, most diagonal edges had lower intensity level. therefore, not clearly shown in the filtered images. This was due to the fact that Sobel filters were designed for filtering vertical and horizontal edges separately.

3. Generate a combined edge image by using the formula below. Suggest a reason for the squaring operation.

$$G = \sqrt{G_x^2 + G_y^2}$$

```
g = sqrt(gx .^ 2 + gy .^ 2);  
g = normalize_edge(g);  
imwrite(uint8(g), "results/01_grad.png");
```

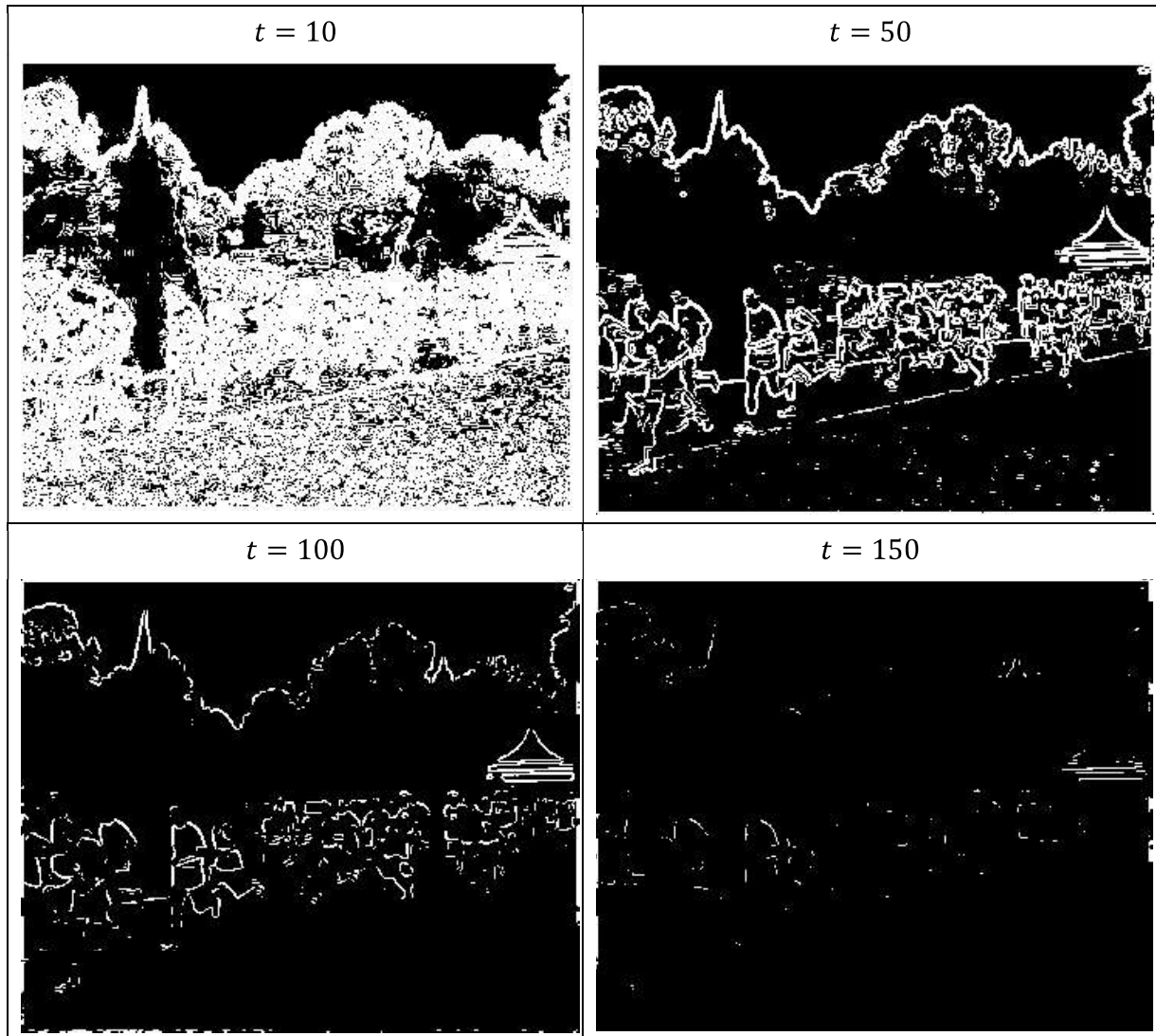
After performing filtering using the Sobel filters, some edge pixels may have negative values. The squaring operation was performed to obtain the magnitude of edge pixels. The result G may have values greater than 255, thus the combined edge image was normalized to the range of 0 – 255.



4. Threshold the image at value t . This creates a binary image. Try different thresholds and explain their advantages and disadvantages.

$$E_t = G > t$$

```
T = [10, 50, 100, 150];  
for i=1:length(T)  
    t = T(i);  
    e = g > t;  
    imwrite(e, sprintf("results/01_t_%d.png", t));  
end
```



From the images above, the lower the threshold, the more the edge being detected. Choosing small threshold tend to include more unnecessary edges, causing more noise in the detected edges. Choosing a threshold that is too large resulted in very little edges detected as it removed noise and also valid edges. Thus, a good threshold was needed to keep a balance between noise and valid edges.

5. Recompute the edge image using Canny edge detection algorithm, with the following parameters. The resultant image is a binary image without the need of thresholding.

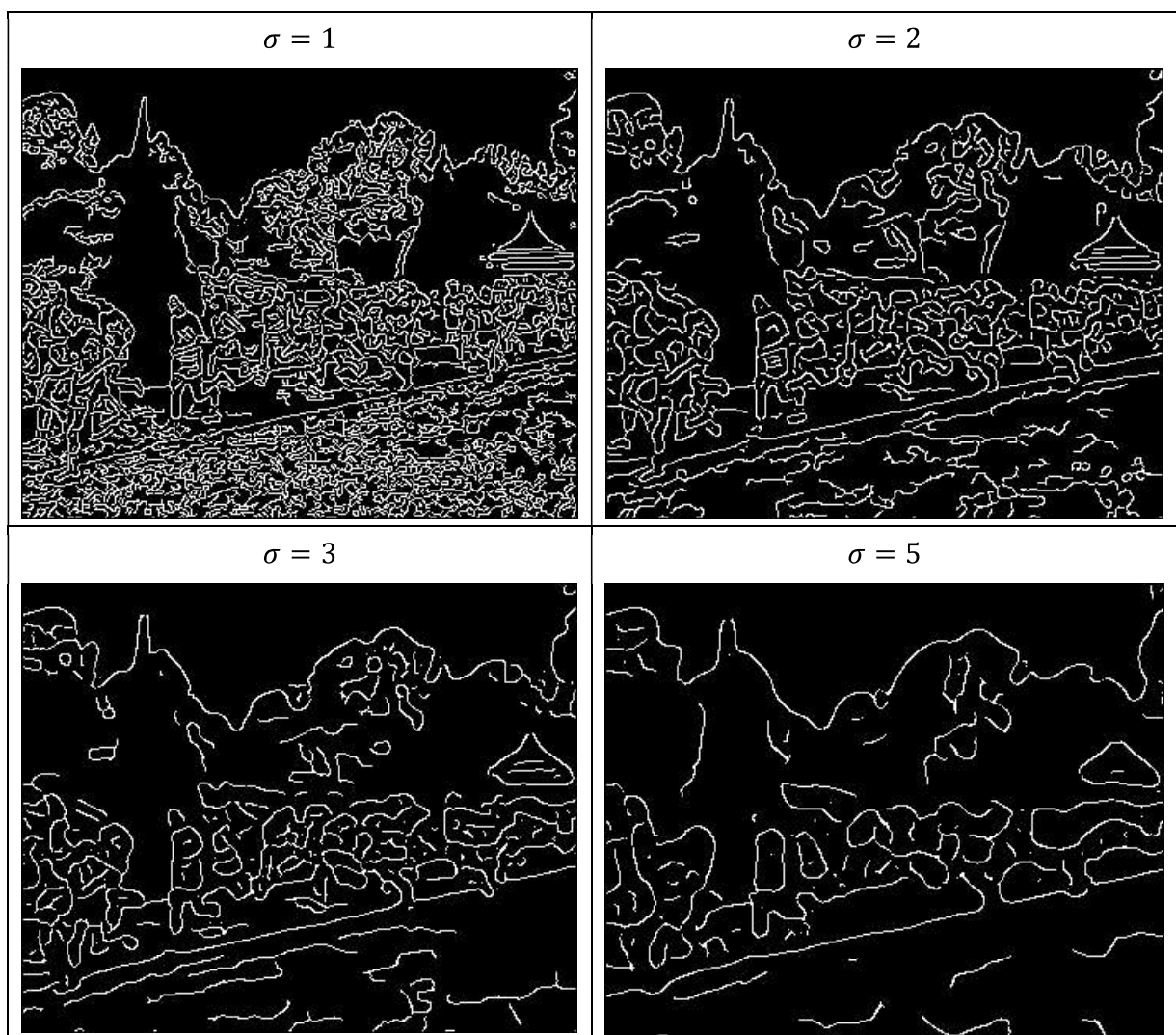
$$t_l = 0.04$$

$$t_h = 0.1$$

$$\sigma = 1.0$$

6. Try different values of σ ranging from 1 to 5 and determine the effect on edge images. How different σ are affects the removal of noisy edgels and the location accuracy of edgels.

```
t1 = 0.04;  
th = 0.1;  
sigma = 1.0;  
  
S = [1, 2, 3, 4, 5];  
for i=1:length(S)  
    s = S(i);  
    e = edge(img, 'canny', [t1, th], s);  
    imwrite(e, sprintf("results/01_canny_sigma_%d.png", s));  
end
```

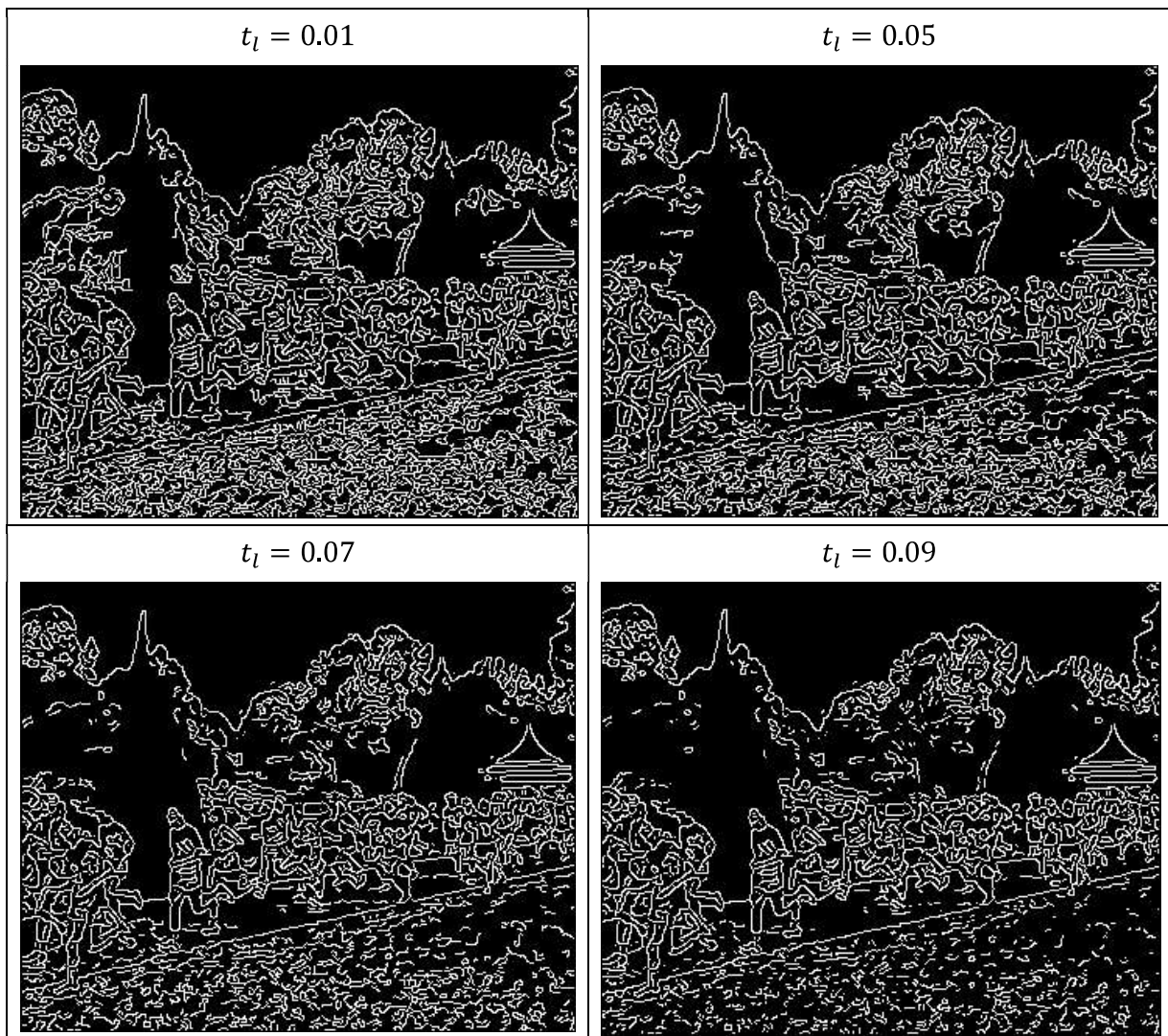


From the results above, the larger the value of σ , the less noisy the edgels detected, the less accurate the location of edgels. This was because Canny edge detector used gaussian filters

to smooth edges, when σ is larger, more noise were being removed but at the same time, the edges were blurred and edge details were lost.

7. Try different values of t_l , what are the effects on the edges detected?

```
TL = [0.01, 0.03, 0.05, 0.07, 0.09];  
for i=1:length(TL)  
    tl = TL(i);  
    e = edge(img, 'canny', [tl, th], sigma);  
    imwrite(e, sprintf("results/01_canny_tl_%d.png", round(tl * 100)));  
end
```



Canny edge detection used hysteresis thresholding such that:

$$e_t = \begin{cases} 1 & e \geq t_h \\ 1 & t_l \leq e < t_h \text{ and neighbouring pixels perpendicular to gradient} = 1 \\ 0 & e < t_l \end{cases}$$

When t_l is small, more edgels were being detected thus more noisy edge image is obtained.

When t_l is high, less edgels were being detected thus less noisy edge image is obtained.

3.2 Line Finding using Hough Transform

1. Reuse the edge image computed using Canny algorithm with the following parameters:

$$t_l = 0.04$$

$$t_h = 0.1$$

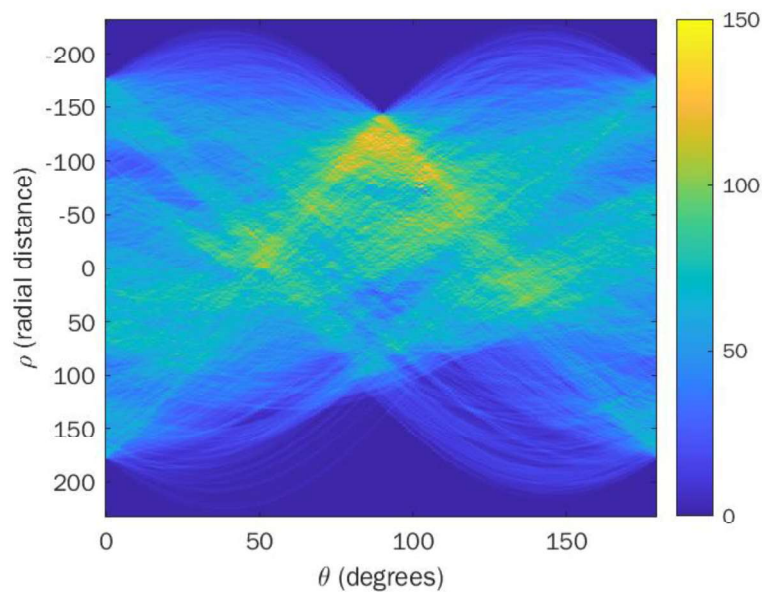
$$\sigma = 1.0$$



2. Use Radon transform on the binary image and explain why the transform is equivalent to Hough transform? When are they different?

```
THETA = 0:179;
[H, xp] = radon(e);
imagesc(THETA, xp, H);
xlabel('\theta (degrees)');
ylabel('\rho (radial distance)');
colormap("default");
colorbar;
saveas(gcf, "results/02_radon.png");
```

Radon transform is a projection of an image matrix along specific directions, represented by a set of line integrals in their parametric space, which is similar to Hough transform that represent lines in the spatial domain in their parametric form. The same line will be mapped to the same point in the parametric domain for both Radon and Hough transform. The difference is when the image is not in binary format, as Radon transform considers the intensity of pixels along the lines whereas Hough transform only counts the number of pixels along the line. In another words, Hough transform is equivalent to the discretization form of Radon transform.



Conventional Hough Transform	Radon Transform in MATLAB
Cartesian coordinate system has its origin located at the top left corner of image	Cartesian coordinate system has its origin located at the center of image
x-axis pointing right y-axis pointing down	x-axis pointing right y-axis pointing up

- Find the location of the maximum pixel intensity in the Hough image in the form of [theta, radius]. These are the parameters corresponding to the line in the image with strongest edge support.

```
[~, idx] = max(H(:));
[radius_max, theta_max] = ind2sub(size(H), idx);
theta = THETA(theta_max)
radius = xp(radius_max)
```

theta = 103, radius = -76

- Derive the equations to convert the [theta, radius] line representation to the normal line equation form $Ax + By = C$ in **image coordinates**.

Let $\theta = \text{theta}$ and $\rho = \text{radius}$. Let (x', y') be the Cartesian coordinate with origin at the center of image and (x, y) be the image coordinates. From the maximum pixel intensity (θ, ρ) obtained from Radon transform, we can obtain the equation:

$$x' \cos \theta + y' \sin \theta = \rho$$

Let $A = \rho \cos \theta$ and $B = \rho \sin \theta$, we can rewrite the above equation as follows:

$$Ax' + By' = \rho^2 \quad (1)$$

In order to convert the above equation into the form of image coordinate system, we have to derive equations that represents the relationship between (x', y') and (x, y) . Let X be the image width and Y be the image height, we will get:

$$x' = x - \frac{X}{2} \quad (2)$$

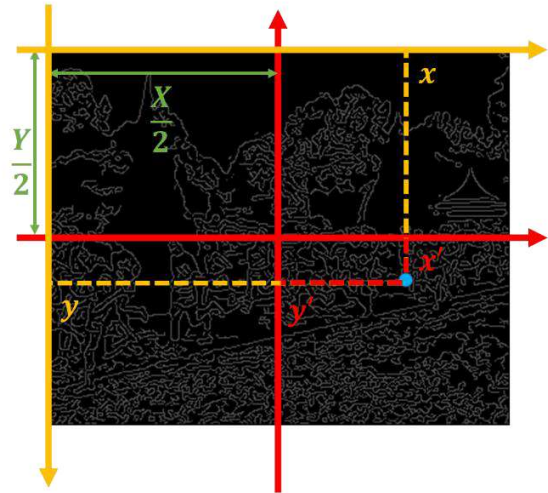
$$y' = y - \frac{Y}{2} \quad (3)$$

Substituting (2) and (3) into (1), we get

$$A \left(x - \frac{X}{2} \right) + B \left(y - \frac{Y}{2} \right) = \rho^2$$

$$Ax + By = \rho^2 + \frac{AX}{2} + \frac{BY}{2}$$

$$C = \rho^2 + \frac{AX}{2} + \frac{BY}{2}$$



```
[A, B] = pol2cart(theta * pi / 180, radius);
A
B = -B % B is negated because y-axis is pointing downwards
[Y, X] = size(img)
C = radius ^ 2 + A * X / 2 + B * Y / 2
```

Take note that B is negated because the y -axis direction is different between the Cartesian coordinate system used by Radon transform (pointing upwards) and the image coordinate system (pointing downwards).

$$X = 358, \quad Y = 290$$

$$A = 17, \quad B = 74, \quad C = 19574$$

5. Based on the equation of line $Ax + By = C$, compute y_l and y_r values for corresponding $x_l = 0$ and $x_r = X - 1$. Superimpose the estimated line that passes through (x_l, y_l) and (x_r, y_r) . Does the line match up with the edge of running path?

```
xl = 0;  
xr = X - 1;  
yl = (C - A * xl) / B  
yr = (C - A * xr) / B  
  
imshow(img);  
line([xl, xr], [yl, yr], 'color', [1, 0, 0]);  
saveas(gcf, "results/02_superimpose.png");
```

$$y_l = 264, \quad y_r = 182$$



Yes, the line matches the running path.

3.3 3D Stereo

1. Write a function that takes in the 4 arguments below and return a disparity map.
 - a. Left image
 - b. Right image
 - c. Template width
 - d. Template height

```

function disparity = disparity_map(image_l, image_r, x_temp, y_temp)
    dim_x = floor(x_temp / 2);
    dim_y = floor(y_temp / 2);
    constraint = 15;
    disparity = ones(y_l, x_l);

    for y=1:y_l
        for x=1:x_l
            y_l_start = max(1, y - dim_y);
            y_l_end = min(y_l, y + dim_y);
            x_l_start = max(1, x - dim_x);
            x_l_end = min(x_l, x + dim_x);
            T = image_l(y_l_start:y_l_end, x_l_start:x_l_end);
            % To calculate S(x, y), we need to perform correlation
            % correlation = convolution with inverted kernel
            T = double(rot90(T, 2));

            y_r_start = y_l_start;
            y_r_end = y_l_end;
            x_r_start = max(x - dim_x - constraint, 1);
            x_r_end = min(x + dim_x + constraint, x_r);
            I = double(image_r(y_r_start:y_r_end, x_r_start:x_r_end));

            S = conv2(I.^2, ones(size(T)), 'valid') - 2 * conv2(I, T, 'valid');
            [~, idx] = min(S);
            d = x_r_start - x_l_start + idx - 1;
            disparity(y, x) = d;
        end
    end
end

```

Explanation of algorithm

1. Loop through every pixel of the left image (PL).
2. For each pixel, extract a template T comprising the 11×11 neighborhood around that pixel.
3. Use the template and calculate the SSD on the right image (PR) along the same horizontal scanline, the scan region is limited to 15 pixels left and right.

$$SSD = \sum_j \sum_k (I(x+j, k) - T(j, k))^2$$

$$SSD = \sum_j \sum_k I^2(x+j, k) + \sum_j \sum_k T^2(j, k) - 2 \sum_j \sum_k I(x+j, k)T(j, k)$$

4. The SSD matched pixel's x-coordinate \widehat{x}_r is obtained by taking the x value that results in the lowest SSD.

$$\widehat{x}_r = \arg \min_x SSD$$

5. The disparity at pixel coordinate (x_l, y_l) is given by $d(x_l, y_l) = x_l - \widehat{x}_r$

2. Download the synthetic stereo pair images “corridorl.jpg” and “corridor.jpg”, converting both to grayscale and run the disparity map algorithm on the 2 images.

```
img_l = imread('images/corridorl.jpg');  
img_l = rgb2gray(img_l);  
  
img_r = imread('images/corridorr.jpg');  
img_r = rgb2gray(img_r);  
  
D = disparity_map(img_l, img_r, 11, 11);  
imagesc(D);  
colormap(flipud(gray));  
colorbar;  
saveas(gcf, "results/03_corridor_disp.png");
```

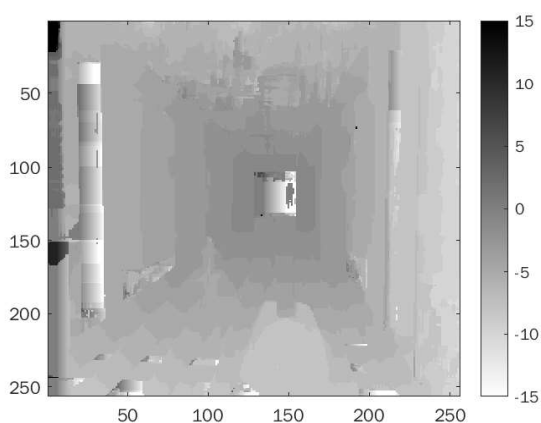
Left image



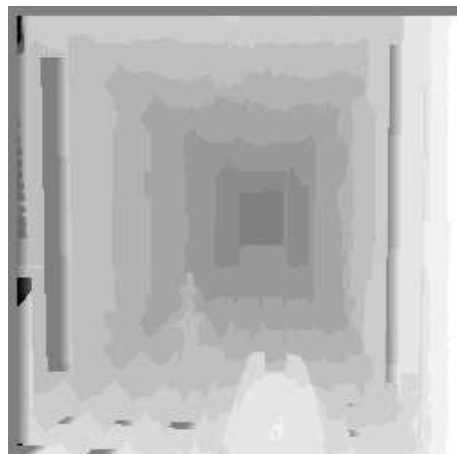
Right image



Disparity Map Obtained

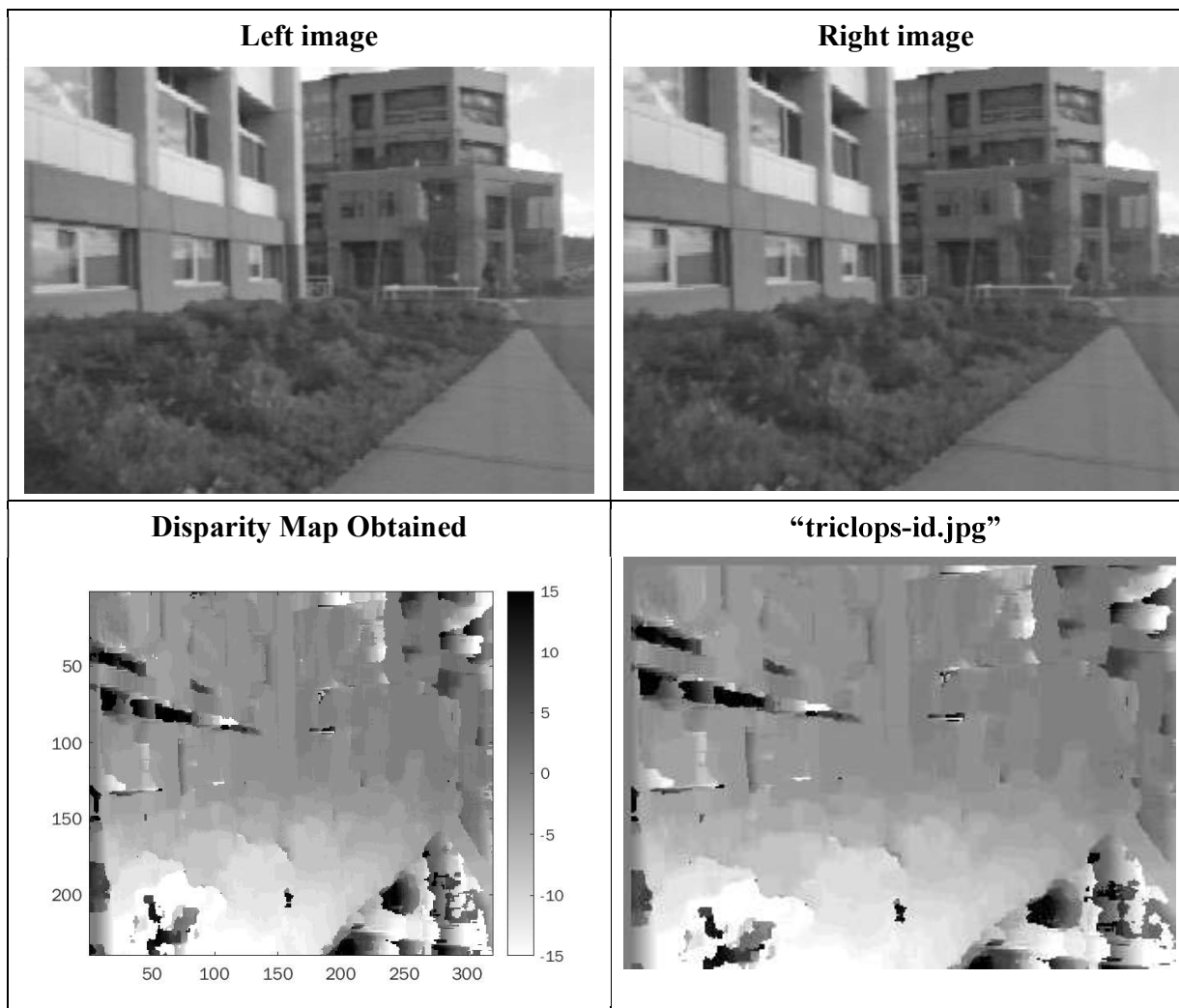


“corridor_disp.jpg”



From the disparity map obtained, pixels for region at further distance were darker, which is similar to “corridor_disp.jpg”. However, the quality of the disparity map obtained using the implemented algorithm was not as good as the result in “corridor_disp.jpg”. This is because the implemented algorithm only scan the template along the horizontal scanline and within 15 pixel difference only. The bright region at the center was supposed to have dark color, this may be due to the large white region which caused the SSD values to be very similar regardless of the value of \hat{x}_r .

3. Rerun your algorithm on the real images of “triclops-i2l.jpg” and “triclops-i2r.jpg”. How does the image structure of the stereo images affect the accuracy of the estimated disparities?



From the results above, the quality of the disparity map obtained is worse than the one obtained using “corridor” images above. This might be due to the lack of good contrast in the images, causing the SSD value obtained to be very similar and lack distinct information to determine the best \hat{x}_r .

3.4 Bag-of-Words (BoW) and Spatial Pyramid Matching (SPM)

Bag of Words (BoW)

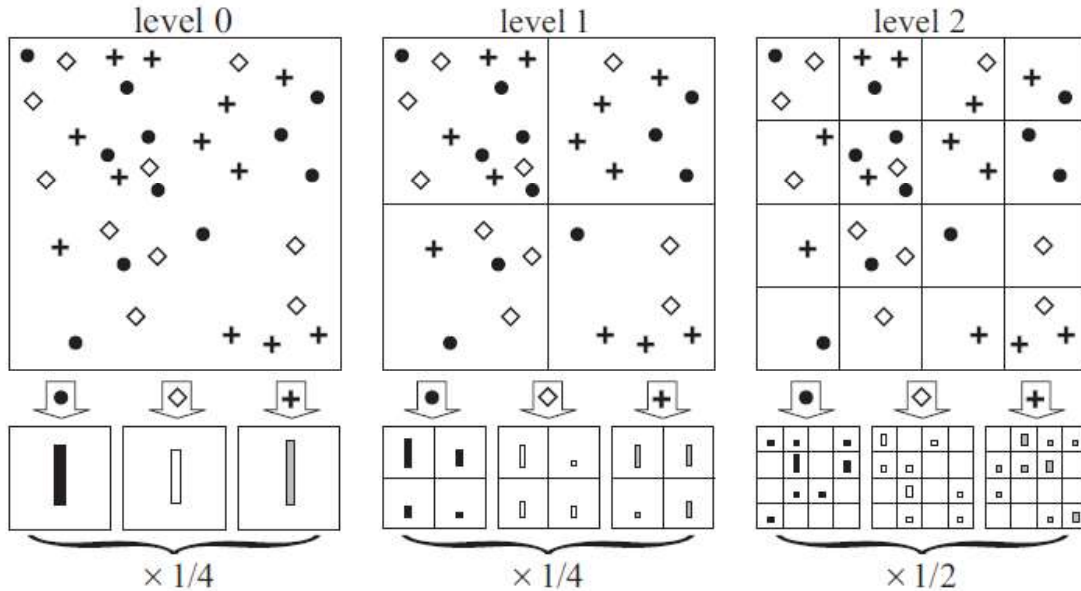
Bag of Words (BoW), also known as Bag of Features, is an algorithm to derive a feature vector from the extracted features. The feature vector is in the form of histogram storing the count of different features present in an image. Let N be the number of different features extracted, the feature vector of an image is $u = \{u_1, \dots, u_N\} \in \mathbb{R}^N$ will be a N -dimensional vector.

Spatial Pyramid Matching (SPM)

Spatial Pyramid Matching (SPM) is an extension over Bag of Words algorithm. The problem with Bag of Words representation is that it does not contain spatial information regarding the extracted. In other words, only the count of the extracted feature is involved in the formation of histogram, the location of the extracted feature is completely ignored.

Spatial Pyramid Matching divides an image into 4^l region ($2^l \times 2^l$ grid) where l is the number of level. The feature vector now stores the count of each feature in each grid. Let N be the number of different features extracted, and L be the total number of levels to be extracted ($0 \leq l \leq L$), the dimensionality of the feature vector u will be:

$$N \sum_{l=0}^L 4^l = \frac{(4^{L+1} - 1)N}{3}$$



The figure above shows how the grids are formed at different levels. When $l = 0$, no grids are formed and therefore the feature vector derived from SPM is equivalent to BoW.

Training Steps

1. Let $x_i \in X$ be an image where $X = \{x_1, \dots, x_N\}$ is the training dataset containing all training images and N is the size of training dataset
2. Extract raw features (keypoints and descriptors) from each image. This can be done with SIFT. The result returned by SIFT for a single image includes
 - a. A set of descriptors $F = \{f_1, \dots, f_n\}$ where $f_i \in \mathbb{R}^{128}$ is a descriptor (128-dimensional vector) and n is the number of descriptors which varies for different images.
 - b. A set of keypoint coordinates for each of the descriptor.
3. Collect all descriptors $F = \{f_1, \dots, f_M\}$ from all training images where $M \gg N$ and cluster them into k clusters using K-means clustering algorithm. At the end of clustering, we will have a set of centroids $C = \{c_1, \dots, c_k\}$ where $c_i \in \mathbb{R}^{128}$. These centroids represent the main k features that are present in the whole training dataset.

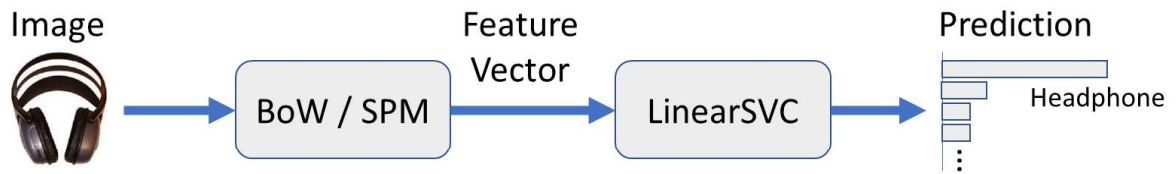
Feature Vector Generation

1. Extract SIFT features from an image, let the descriptors be $U = \{u_1, \dots, u_{|U|}\}$
2. For each descriptor, obtain its nearest cluster by comparing their distance to the centroids C , eventually we will get a vector $P = \{p_1, \dots, p_{|U|}\} \in \mathbb{R}^{|U|}$ where $0 \leq p_j < k$
3. Let L be the maximum number of level, for each l where $0 \leq l \leq L$:
 - a. Create a vector $v_l \in \mathbb{R}^{4^l k}$ to store the count of each feature in each grid.
 - b. For each descriptor u_i , obtain its keypoint coordinates and derive which grid g_i does the keypoint lies within, where $0 \leq g_i < 4^l$, then update the count in vector v_l according to g_i and p_i .
4. Concatenate all vectors v_l into a single vector $V \in \mathbb{R}^{\frac{(4^{L+1}-1)k}{3}}$

Experiment and Results

The objective of this experiment was to compare the image classification performance using image features extracted from BoW and SPM. Caltech-101 dataset was used for this experiment, this dataset consists of 102 categories, each contains 31 to 800 images with medium resolution (about 300×300 pixels).

30 images for each category were used for training and up to 50 images per category were used for testing. Linear support vector classifier (LinearSVC) model was trained to perform classification task. The feature vector extracted using BoW or SPM algorithm was passed into the model as its input to predict the category of an image. The classification experiment is carried out using different L where $0 \leq L \leq 3$. For each L , the experiment is repeated using 10 different seeds for train test split.



L	Accuracy (%)
0 (BoW)	23.59 ± 0.61
1	33.02 ± 0.44
2	34.29 ± 0.42
3	26.86 ± 0.54

From the results above, SPM in general performed better compared to BoW. However, the performance of SPM reduced when $L = 3$. This might be due to the over-sparsity of feature vector when the number of grids is too large, leading to noisy high-dimensional feature vector with only a few important features, causing the model to be more prone to overfitting.