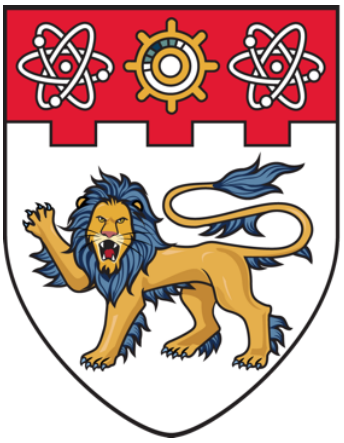


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4003: Computer Vision

LAB 2: Advance Image Processing and Computer Vision Techniques

Gede Bagus Bayu Pentium (U1420090G)

School Of Computer Science and Engineering

1. Introduction

1.1 Objectives

This laboratory aims to provide further exposure to advanced image processing and computer vision techniques in MATLAB. In this laboratory you will:

- Explore different edge detection methods
- Employ the Hough Transform to recover strong lines in the image
- Experiment with pixel sum-of-squares difference (SSD) to find a template match within a larger image. Estimate disparity maps via SSD computation
- Optionally, compare the bag-of-words method with spatial parymid matching (SPM) on the benchmark Caltech-101 dataset.

1.2 Tools

In this experiments we will be using **MATLAB** and **Image Processing Toolbox software package**. For the last (optional) experiment (Compare Spatial Pyramid Matching and Bag-of-Words) we will be using **python**.

2. Experiments and Results

2.1 Edge Detection

a. Display grayscaled image

- **Code:**

```
P = imread('image/maccropped.jpg');
I = rgb2gray(P);
imshow(I);
```

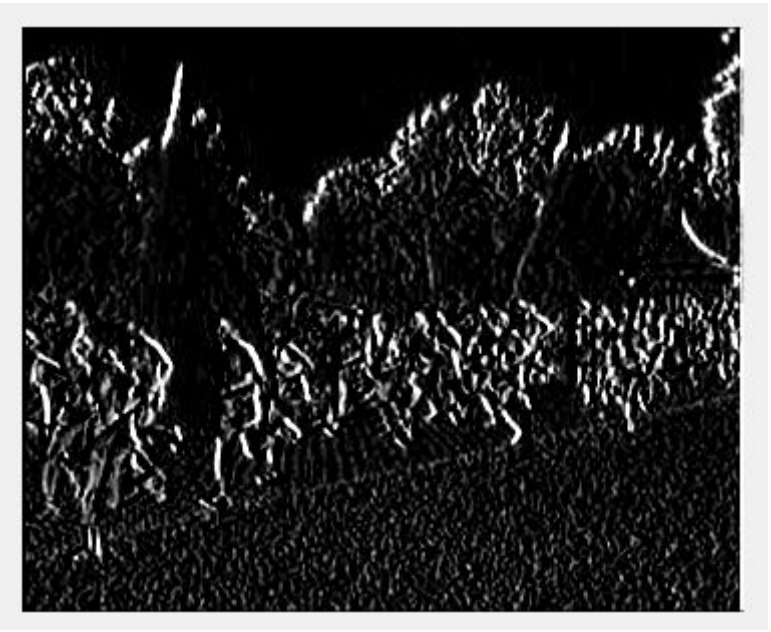
- **Result:**

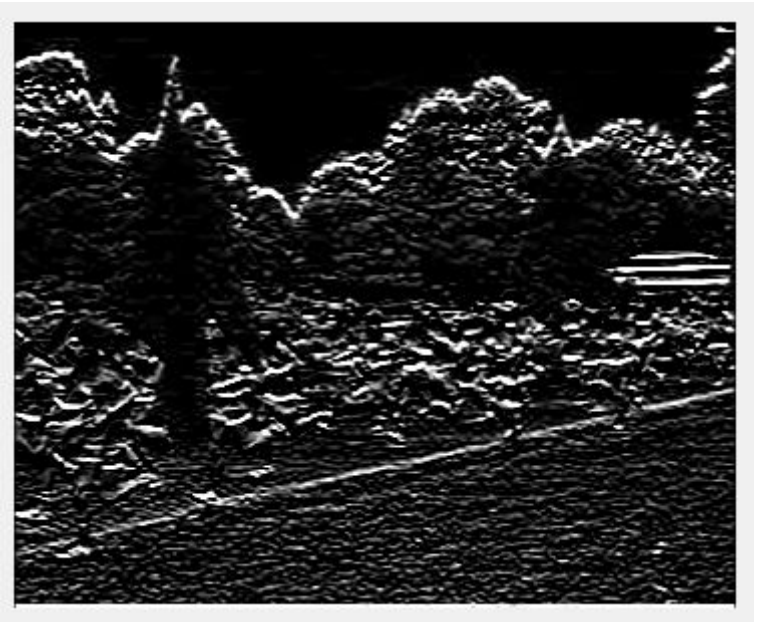
b. Create 3x3 horizontal and vertical Sobel masks and filter the image. Display the edge-filtered images.

- **Code:**

```
sobel_v = [
    -1 0 1;
    -2 0 2;
    -1 0 1;
];
sobel_h = [
    -1 -2 -1;
    0 0 0;
    1 2 1;
];

res1 = conv2(I, sobel_v);
imshow(uint8(res1))
res2 = conv2(I, sobel_h);
imshow(uint8(res2))
```

- **Result:**





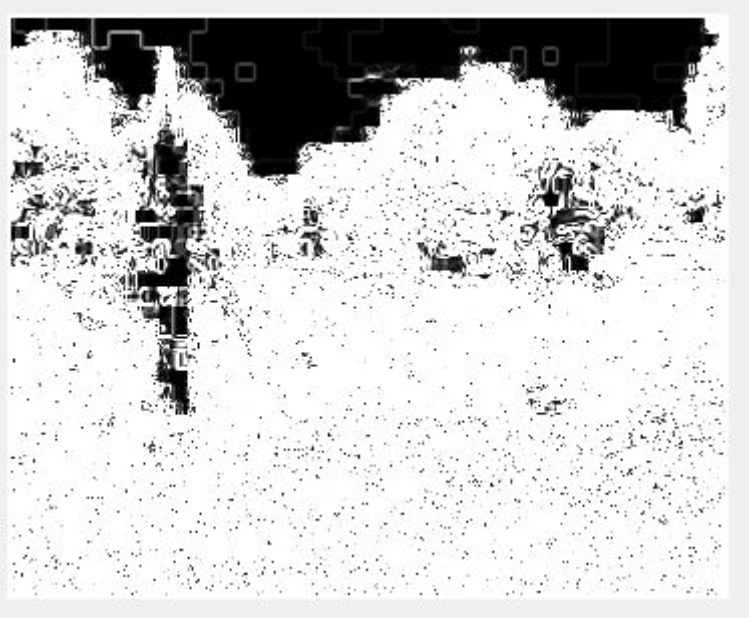
From the images above we can see vertical line (top image) and horizontal line (bottom image) are detected after applying the image. The non vertical or horizontal image (i.e. diagonal) is not fully detected. because the individual solbel filter designed only filter vertical and horizontal image separately. If we observe further some diagonal is detected in vertical and some is detected in horizontal. This is because if the image shown diagonal line but if zoomed has 3*3 vertical or horizontal line it will be detected. For example the roof line on right part of the image is shown in vertical filter but not on horizontal filter. This because if the image is zoomed it has many small vertical lines.
Further observation that the vertical image detected is only from right to left and horizontal image from top to bottom. This is because the result is not squared or absoluted. So, some of the result is still in negative.

c. Generate a combined edge image

- **Code:**

```
E = res1.^2 + res2.^2;
imshow(uint8(E));
```

- **Result:**



The vertical and horizontal filtered image shows vertical and horizontal difference of the image considering 3*3 kernel size. So, the sum of square from 2 horizontal or vertical edge image will give the magnitude of the gradient. So this will allow to both detect horizontal, vertical and even diagonal line.
The square operation is carried because we want to get the magnitude. Also from original egde image the edge is one direction (right → left, top → bottom), hence some of the line is actually becomes negative. To get the line (magnitude of the gradient) square operation need to be done.
From the image we can see almost all of the pixel is filled with value, this is because the treshold is not applied yet to the edge image.

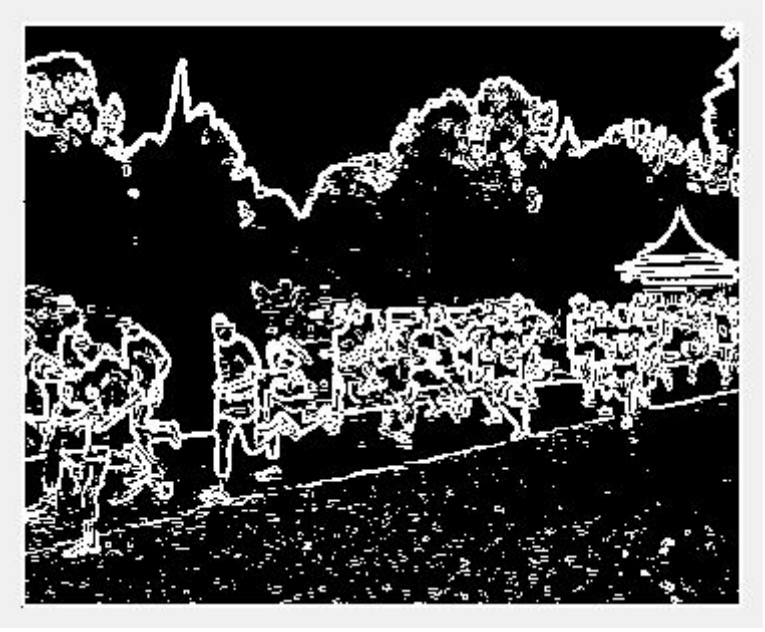
d. Threshold the edge image E at value t

- **Code:**

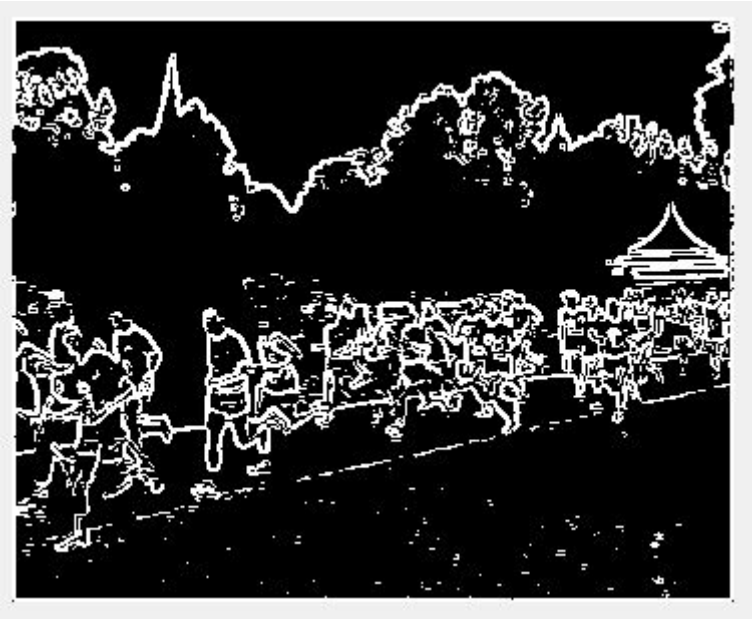
```
t = [10000, 20000, 40000, 80000];
Et = E>t(1);
imshow(Et)
Et = E>t(2);
imshow(Et)
Et = E>t(3);
imshow(Et)
Et = E>t(4);
imshow(Et)
```

- **Result:**

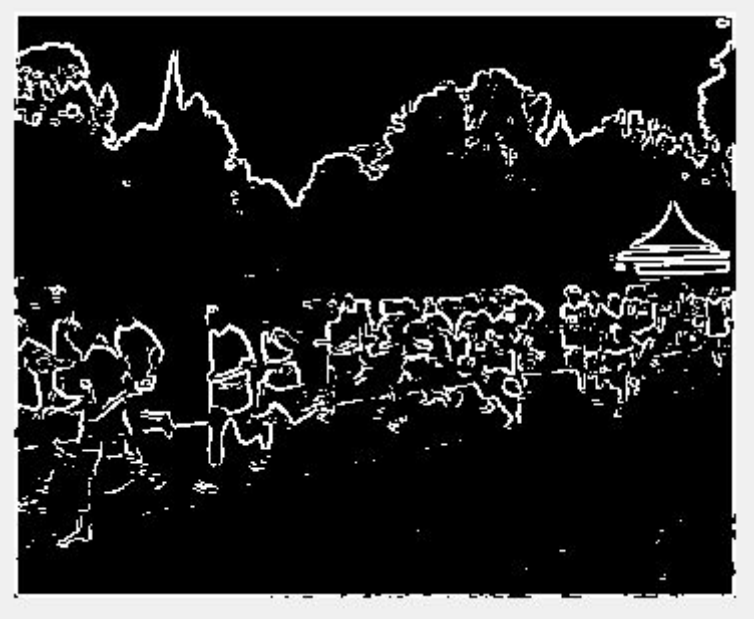
Treshold = 10000



Treshold = 20000



Treshold = 40000



Treshold = 80000

From the images above we can see more edge is detected if we are using small treshold. But at the same time more noise also created (small edge). Because the edge detection using sobel matrix also create noise if the treshold is too small.
In contrary if the treshold is big, most of the noise is filtered. But at the same time some edge is failed to be detected because the magnitude is not very big (the contrast is not very big in the line).
So, the right treshold need to be choosen to filter the noise and detect the line. Balance between the trade-off of having more noise (small edge) and detecting more edge need to be considered.

e. Recompute the edge image using Canny edge detection
    i. Try different value of `sigma`

- **Code:**

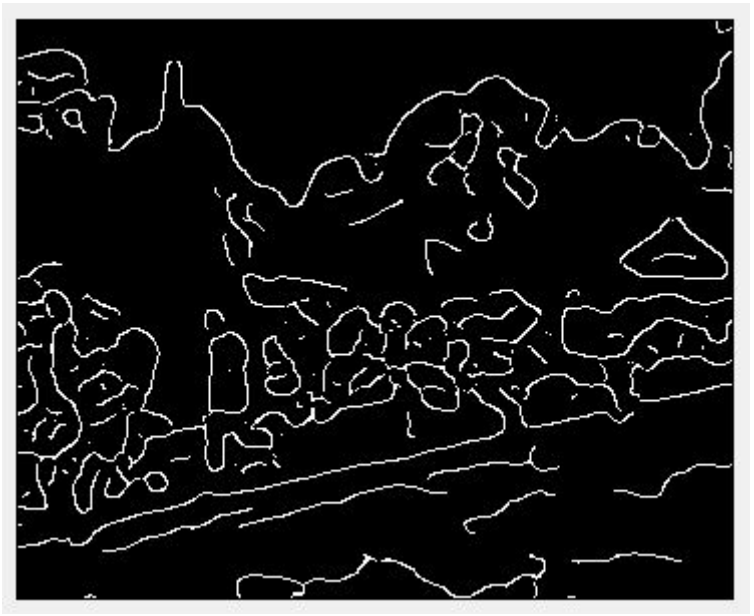
```
tl = 0.04;
th = 0.1;
sigma = [1.0, 2.0, 3.0, 4.0, 5.0];
E = edge(I, 'canny', [tl th], sigma(1));
imshow(E)
E = edge(I, 'canny', [tl th], sigma(2));
imshow(E)
E = edge(I, 'canny', [tl th], sigma(3));
imshow(E)
E = edge(I, 'canny', [tl th], sigma(4));
imshow(E)
E = edge(I, 'canny', [tl th], sigma(5));
imshow(E)
```

- **Result:**

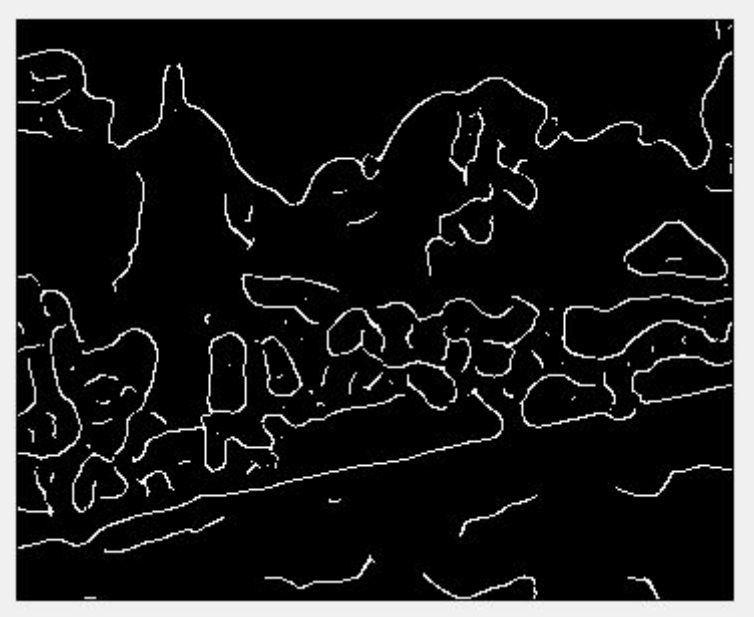
sigma = 1.0

sigma = 2.0

sigma = 3.0



sigma = 4.0



sigma = 5.0

From the result above we can see that changing `sigma` values change the result of edge detection. With small value of `sigma` we get more noisy edgel (small edge) but the edgel location accuracy is better. In contrary with high value of `sigma` we get less noisy edgel with worse edgel location accuracy.
The reason is because the canny algorithm are using gaussian filter to smooth the edge. If the sigma is too big the image will have less noise but also lost its details. This resulting in less noisy edgel but less accuracy for the edge location.

ii. Try different value of `tl`

- **Code:**

```
tl = [0.09, 0.04, 0.01];
th = 0.1;
sigma = 1.0;
E = edge(I, 'canny' ,[tl(1) th], sigma);
imshow(E);
E = edge(I, 'canny' ,[tl(2) th], sigma);
imshow(E);
E = edge(I, 'canny' ,[tl(3) th], sigma);
imshow(E);
```

- **Result:**

tl = 0.09

sigma = 0.04

sigma = 0.01

As we know Canny edge detection using hysteresis tresholding. And `tl` is the lower bound of the hysteresis tresholding. So, Canny in this case will filter the edgel that have magnitude smaller than `tl`. With that reason when `tl` is small, more edge will be detected as well as more noise. And when the `tl` is large, less edge will be detected and less noise.

2.2 Line Finding using Hough Transform

a. Reuse edge with `sigma = 1.0`.

- **Code:**

```
P = imread('image/maccropped.jpg');
I = rgb2gray(P);
tl = 0.04; th = 0.1; sigma = 1.0;
E = edge(I,'canny',[tl th],sigma);
imshow(E);
```

- **Result:**

b. Simulate Hough trasnform using Radon transform.

The Radon transform is a function to computes the projections of an image matrix. The porjection of two-dimensional function *f(x,y)* is a set of line integrals.

$$(x(z), y(z)) = \Big((z\sin\alpha + s\cos\alpha), (-z\cos\alpha + s\sin\alpha)\Big)$$

First of all the radon transform will first formulate the every line in spatial domain into parameterization

$$\begin{aligned} Rf(\alpha, s) &= \int_{-\infty}^{\infty} f(x(z), y(z))\, dz \\ &= \int_{-\infty}^{\infty} f\big((z\sin\alpha + s\cos\alpha), (-z\cos\alpha + s\sin\alpha)\big)\, dz \end{aligned}$$

Then the value of Radon transform is calculated by finding the integral value of the line (projections) into x and y.

- **Code:**

```
[H, xp] = radon(E);
imshow(uint8(H));
```

- **Result:**

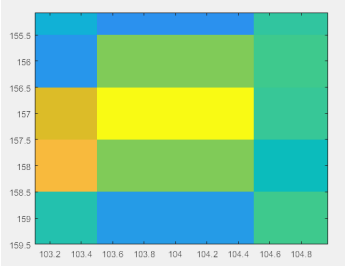
As a result redon transform will have the same value as Hough transform. This is because the image is applying discrete radon transform. With this both of the transform will map each pixel into equivalent sinusodial function. However the Radon transform will result differently if we consider a continuous radon transform. In continuous radon transform is obviously different because hough transform works discretely. Hence the density of the line will have different value.

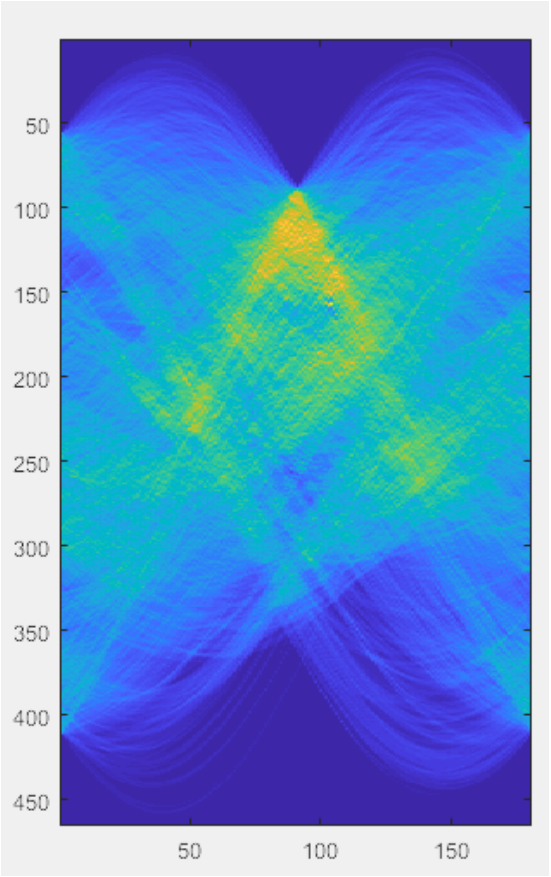
c. Find location of maximum pixel intensity.

- **Code:**

```
imagesc(uint8(H));
colormap('default');
```

- **Result:**





From the image above we can find that the peak is located at

- `theta = 103;`
- `radius = 157;`

d. Derive the line equation.

Function `pol2chart` is used to convert polar coordinate to cartesian coordinate. Which is the value of A and B in equation *Ax* + *By* = *C* when *C* = *0*. So using `pol2chart` we can find the value of A and B. in Matlab `pol2chart` using the following relation to convert polar coordinate to cartesian coordinate:

```
theta = atan2(y,x)
rho = sqrt(x.^2 + y.^2)
```

The value *A* is not changed after we move the center of the image back to the top left of the image. The value *B* is changed to *-B*. Then the value of *C* is changed by the same amount of the change in center. The center is changed from `(145, 179)` (center of the image) in the cartesian coordinate. Hence the value of *C* is changed into *A*(A+179) + B*(B+145)*

- **Code:**

```
theta = 103;
radius = xp(157);
[A, B] = pol2cart(theta*pi/180, radius);
B = -B;

C = A*(A+179) + B*(B+145);
```

- **Result:**

```
A =

   17.0963


B =

   74.0521


C =

   1.9574e+04
```

e. Compute `yl` and `yr` with `xl = 0`, `xr` = width of image - `1`.

- **Code:**

```
xl = 0;
yl = (C - A * xl) / B;
xr = 357;
yr = (C - A * xr) / B;
```

- **Result:**

```
yl =

  264.3245


yr =

  181.9046
```

f. Display image with line.

- **Code:**

```
imshow(I)
line([xl xr], [yl yr])
```

- **Result:**

From the image we can see that the line is almost perfectly aligned with the edge of running path (which here is the strongest line in the image). However if looked carefully the end line is not perfectly aligned.
There are several possibility for source of error in forming the line:

- First, the line in the image is not necessarily a straight line. So, in this case finding a line that 100% match the running path will require non linner function
- Second, the calculation of Canny edge detection and Radon transform will lose some precision. So even the line is a very strict line the calculated line equation is not always in the best precision. Because the Radon transformation also supposed to be working on continuous function. But the our calculations are using discrete functions.

2.3 3D Stereo

a. Write desparity map function

The disparity map function take 4 parameters:

- Matrix of grayscaled left image
- Matrix of grayscaled right image
- Template x dimension
- Template y dimension

It will search along the same scan line the template with smallest sum of squared difference. And it also limit the search up to 15 pixel along the scan line (to the left).

- **Code:**

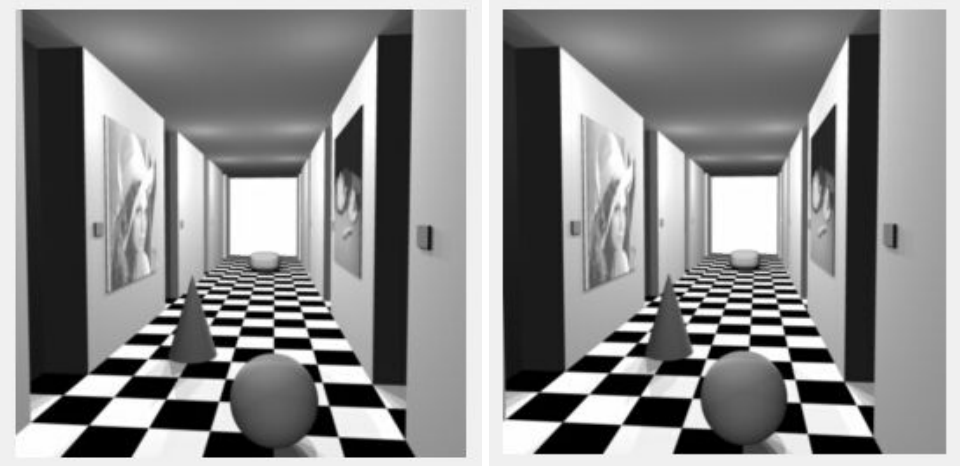
```
function ret = map(image_l, image_r, x_temp, y_temp)
% Fuction to calculate disparity map

% calculate half size of template
dim_x = floor(x_temp/2);
dim_y = floor(y_temp/2);

% image_l and image_r must have the same size
[x, y] = size(image_l);

%initialization
ret = ones(x - x_temp + 1, y - y_temp + 1);

for i = 1+dim_x : x-dim_x
    for j = 1+dim_y : y-dim_y
        cur_r = image_l(i-dim_x: i+dim_x, j-dim_y: j+dim_y);
        cur_l = rot90(cur_r, 2);
        min_coor = j;
        min_diff = inf;

        % search for simmilar pattern in right image
        % limit search to 15 pixel to the left
        for k = max(1+dim_y , j-14) : j
            T = image_r(i-dim_x: i+dim_x, k-dim_y: k+dim_y);
            cur_r = rot90(T, 2);

            % Calculate ssd and update minimum diff
            conv_1 = conv2(T, cur_r);
            conv_2 = conv2(T, cur_l);
            ssd = conv_1(x_temp, y_temp) - 2 * conv_2(x_temp, y_temp);
            if ssd < min_diff
                min_diff = ssd;
                min_coor = k;
            end
        end
        ret(i - dim_x, j - dim_y) = j - min_coor;
    end
end
```

b. Display greyscaled image of *corridorl.jpg* and *corridorr.jpg*.

- **Code:**

```
l = imread('image/corridorl.jpg');
l = rgb2gray(l);
imshow(l);
r = imread('image/corridorr.jpg');
r = rgb2gray(r);
imshow(r);
```

- **Result:**

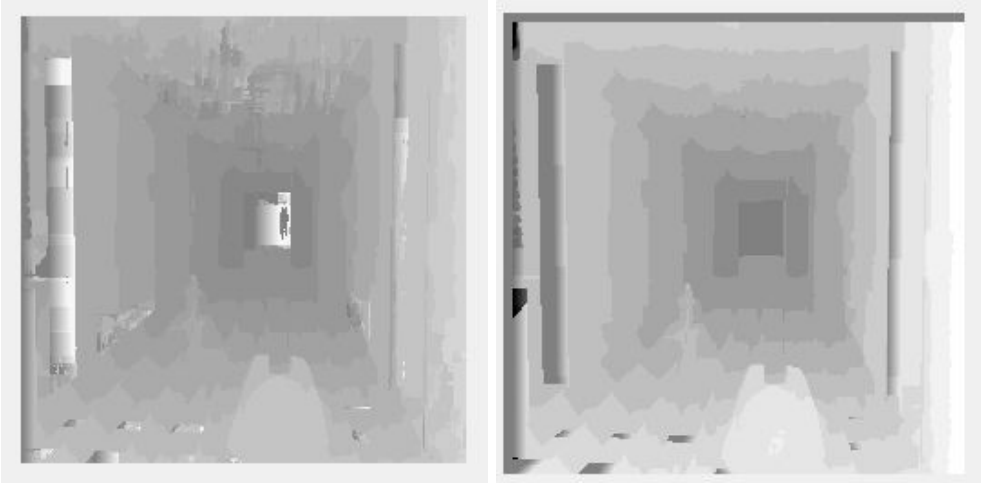


c. Run disparity map on the images.

- **Code:**

```
D = map(l, r, 11, 11);
imshow(D,[-15 15]);
res = imread('image/corridor_disp.jpg');
imshow(res);
```

- **Result:**



The left image is the result of the disparity map. From the result we can see further distance resulting in darker disparity map. But because the comparison only with the same scanline and search only within 15 pixel difference the result is a bit different compared with the *corridor_disp.jpg*. We can see that the corridor is correctly mapped (becomes darker) and the object (sphere and cone) also mapped.

d. Re-run disparity map on *triclops-i2l.jpg* and *triclops-i2r.jpg*.

- **Code:**
- **Result:**

From the results above we can see that the image is not correctly mapped at some region. The result can be said worse compared with the corridor. The reason is because:

- First, the image in the second experiment has less contrast compared with the corridor. With less contrast the disparity map detection become not very accurate.
- Second, the image in the second image is having different scan line. So some part is not correctly mapped (The same thing happen in the first experiment)

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

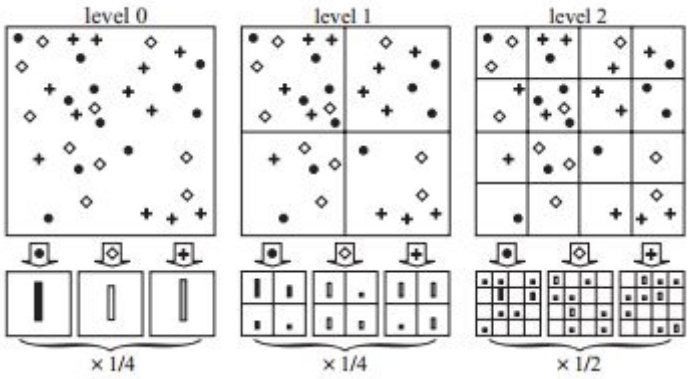
Algorithms

Bag of Word

Bag of words is algorithms of feature extraction that count the number of different feature and store it in the form of histogram. This approach has been widely used in machine learning specially in natural language processing (hence the name bags-of-words).

Spatial Pyramid Matching

The spatial pyramid matching is an algorithm that improve the bags-of-words by considering the location of the features. As mentioned before bags-of-words only use the count of each feature, which means the location of the feature is ignored (orderless). In spatial pyramid matching the image is further divided into 4 region and each region number of features is counted. With this the algorithm also consider the location of the feature (spatial order).



Experiments

NOTE: Due to technical problems in this experiments only half is done (only strong feature)

The experiments consist of the following steps:

- Dataset is loaded from “Caltech-101” (30 train and up to 50 test images per class)
- Generate features usign SIFT algorithm
- Extract the features by using pyramid spatial
- Train the model using SVM and train data feature
- Test the model using test data feature

Implementations

Utility Functions (utils.py)

```
""" Utils for SPM """
import cv2
import matplotlib.pyplot as plt

def image_to_gray(color_img):
    """ Convert image to gray """
    gray = cv2.cvtColor(color_img, cv2.COLOR_BGR2GRAY)
    return gray

def get_sift_features(gray_img):
    """ Generate sift features """
    sift = cv2.xfeatures2d.SIFT_create()
    kp, description = sift.detectAndCompute(gray_img, None)
    return kp, description
```

Feature Extraction (spm.py)

```
"""
Function to get spatial pyramid features of an images
"""

def get_grid(level, x, y, cols, rows):
    """
    Get the specified grid from the images
    """
    denom = 1<<level
    nx, ny = 0, 0
    for numer in range(denom):
        if (numer/denom)*cols <= x <= ((numer+1)/denom)*cols:
            nx = numer
        if (numer/denom)*rows <= y <= ((numer+1)/denom)*rows:
            ny = numer
    return ny * denom + nx

def get_spatial_pyramid(descriptions, coordinates, dimensions, clusters, L=3, M=200):
    """
    Transforms the images into spatial pyramid
    """
    X = []
    Y = []
    for k in descriptions:
        for idx in range(len(descriptions[k])):
            if descriptions[k][idx] is None:
                X.append([0 for i in range((M * (-1 + 4**(L+1)))//3)])
                Y.append(k)
                continue
            cols = dimensions[k][idx][0]
            rows = dimensions[k][idx][1]
            v = []
            channels = {}
            channels_coordinates = {}
```

```
            preds = clusters.predict(descriptions[k][idx])
            for i in range(len(preds)):
                if preds[i] not in channels:
                    channels[preds[i]] = []
                    channels_coordinates[preds[i]] = []
                channels[preds[i]].append(descriptions[k][idx][i].tolist())
                channels_coordinates[preds[i]].append(coordinates[k][idx][i])
            for c in range(M):
                if c not in channels:
                    v += [0 for i in range(((-1 + 4**(L+1)))//3)]
                    continue
                for l in range(L+1):
                    w = 0
                    if l == 0: w = 1/(1<<L)
                    else: w = 1/(1<<(L-l+1))
                    hist = [0 for i in range(4**l)]
                    for i in range(len(channels_coordinates[c])):
                        x = channels_coordinates[c][i][0]
                        y = channels_coordinates[c][i][1]
                        grid = get_grid(l, x, y, cols, rows)
                        hist[grid] += 1
                    hist = [it * w for it in hist]
                    v += hist
            X.append([it/(((M/200) * 25) * (1<<L)) for it in v])
            Y.append(k)
    return X, Y
```

Training and Testing (start.py)

```
"""
Start script
"""
import os
import cv2
from sklearn.metrics import accuracy_score
from sklearn.svm import LinearSVC
from sklearn.cluster import KMeans
import numpy as np

import utils
import spm

def main():
    """ load data, extract feature (spatial pyramid), train model, test """
    rootdir = os.getcwd() + '/101_ObjectCategories'
    categories = os.listdir(rootdir)
    descriptions = {}
    descriptions_test = {}
    coordinates = {}
    coordinates_test = {}
    sizes = {}
    sizes_test = {}
        for cat in categories:
        files = os.listdir(rootdir + '/' + str(cat))
        descriptions[cat] = []
```

```
            coordinates[cat] = []
            sizes[cat] = []
            descriptions_test[cat] = []
            coordinates_test[cat] = []
            sizes_test[cat] = []
            for i in range(len(files)):
                each = files[i]
                if i <= 29:
                    img = cv2.imread(rootdir + '/' + str(cat) + '/' + str(each))
                    img_gray = utils.image_to_gray(img)
                    kp, descriptions = utils.get_sift_features(img_gray)
                    kp = [it.pt for it in kp]
                    descriptions[cat].append(descriptions)
                    coordinates[cat].append(kp)
                    sizes[cat].append((img_gray.shape[1], img_gray.shape[0]))
                elif 30 <= i <= 30+50-1:
                    img = cv2.imread(rootdir + '/' + str(cat) + '/' + str(each))
                    img_gray = to_gray(img)
                    kp, descriptions = utils.get_sift_features(img_gray)
                    kp = [it.pt for it in kp]
                    descriptions_test[cat].append(descriptions)
                    coordinates_test[cat].append(kp)
                    sizes_test[cat].append((img_gray.shape[1], img_gray.shape[0]))
        features = []
        for k in descriptions:
            for it in descriptions[k]:
                if it is None:
                    continue
                for i in range(len(it)):
                    features.append(it[i])
        features = np.array(features)
        M = 200
        clusters = KMeans(n_clusters=M, n_jobs=-1)
        clusters.fit(features)
        for L in range(4):
            X_train, Y_train = spm.get_spatial_pyramid(descriptions, coordinates, sizes, clusters, L, M)
            X_test, Y_test = spm.get_spatial_pyramid(descriptions, coordinates, sizes, clusters, L, M)
            model = LinearSVC()
            model.fit(X_train, Y_train)
            print("Level " + str(L) + ": " + str(accuracy_score(Y_test, model.predict(X_test))))

if __name__ == "__main__":
    main()
```

Results and Discussions

The following are the result of the experimens

- Level 0 (bags-of-words): 0.151544991511
- Level 1: 0.218149405772
- Level 2: 0.252139219015
- Level 3: 0.24408149405772

As can be seen the result using spatial pyramid matching is better compared using bags-of-words (Level 0). But we can see the result of the spatial pyramid matching is reduced at Level 3. This is may be because the sift feature generator used only generate sift at several position. The ideal case (in the paper) the sift feature is generated for every region. This resulting in a very low variation of train features.

3. Conclusion

After the experiment the following is learned:

- Exploring Sobel and Canny edge detection
- Learning Hough Transformation to detect strongest line (using Raden transforms)
- Estimate disparity maps using convolution
- Learn spatial-pyramid-matching and compare with bags-of-words

4. Reference

- https://en.wikipedia.org/wiki/Edge_detection
- https://en.wikipedia.org/wiki/Radon_transform
- http://tnw.home.tudelft.nl/fileadmin/Faculteit/TNW/Over_de_faculteit/Afdelingen/Imaging_Science_and_Technology/Research/Research_Groups/Quantitative_Imaging/Publications/Technical_Reports/doc/mvanginkel_radonandhough_tr2004.pdf
- http://www.ece.northwestern.edu/local-apps/matlabhelp/techdoc/ref/pol2cart.html
- https://en.wikipedia.org/wiki/3D_stereo_view
- CZ4003 - Lecture Notes
- CZ4003 - Lab Manuals
- Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories