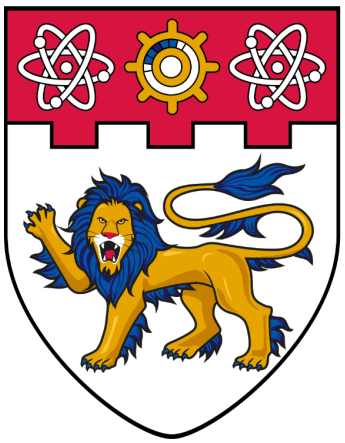


NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

CZ4003 Computer Vision

Lab 2: Advanced Image Processing and Computer Vision Techniques

Sherwin Samson

U2020911J

07/11/2022

3.1 Edge Detection

(a) Display grayscale image of macritchie.jpg:

<<Input>>

```
img = imread('macritchie.jpg');
img = rgb2gray(img);

figure
imshow(img)
```

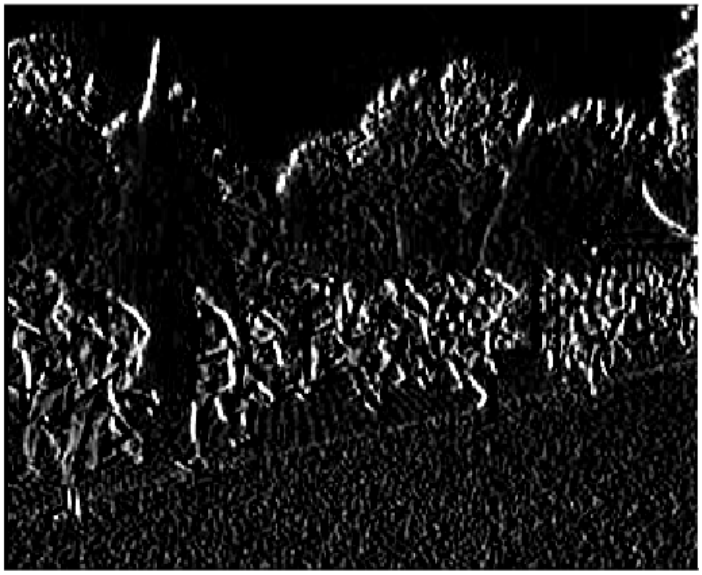
<<Output>>

(b) Edge filtering with 3x3 horizontal and vertical Sobel masks:

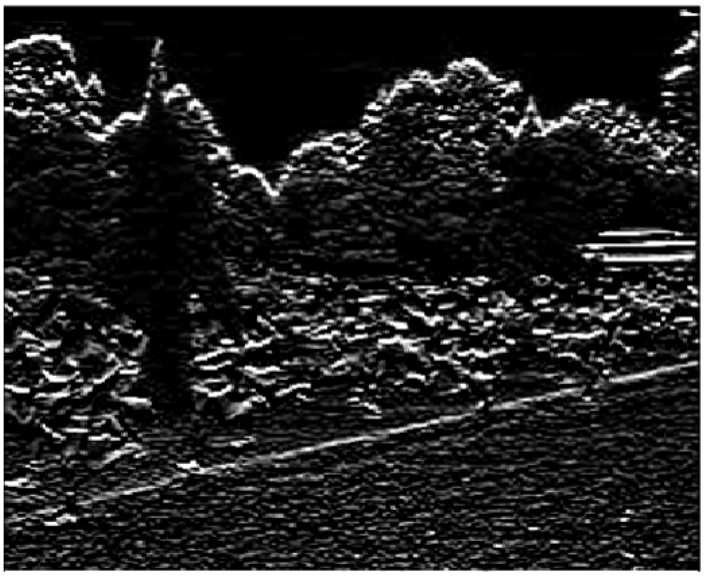
<<Input>>

```
sobel_vertical = [-1 0 1; -2 0 2; -1 0 1;];
sobel_horizontal = [-1 -2 -1; 0 0 0; 1 2 1;];

sobel_v_img = conv2(img, sobel_vertical);
figure
imshow(uint8(sobel_v_img))

sobel_h_img = conv2(img, sobel_horizontal);
figure
imshow(uint8(sobel_h_img))
```

<<Output>>



Vertical Sobel Mask

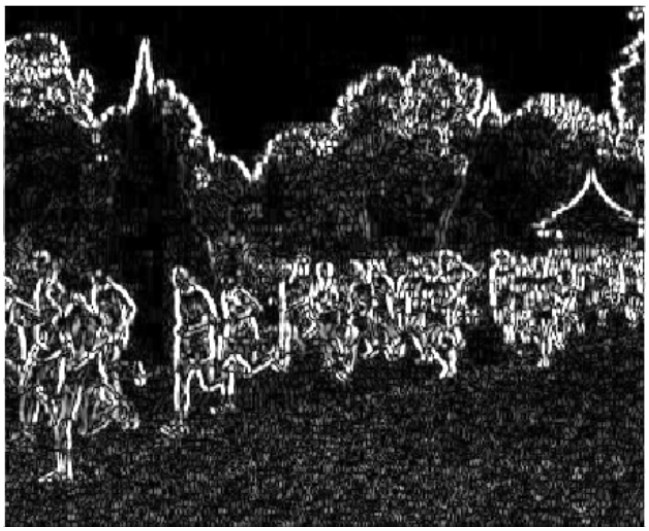


Horizontal Sobel Mask

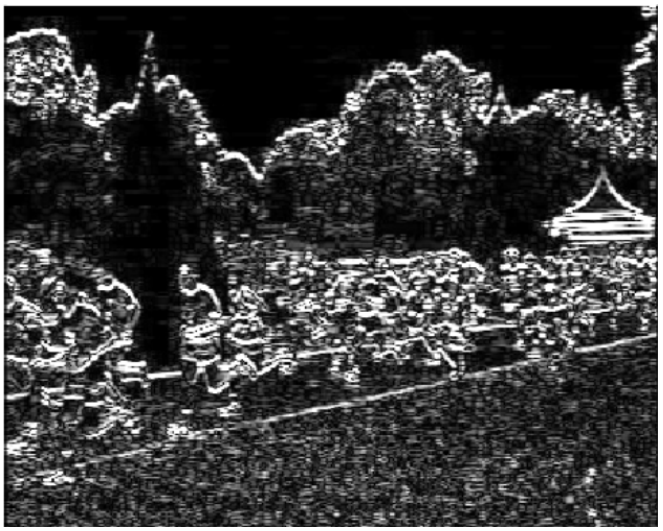
1. On applying the vertical Sobel Mask, vertical edges of the image are detected while the horizontal Sobel Mask detects the horizontal edges.
2. On observing further, we can see that the vertical and horizontal sobel masks actually did not detect half of the vertical and horizontal edges. This is due to detecting the edges in one direction only which will set the edges of the opposite direction as negative and undetected. The vertical mask detects from top to bottom while the horizontal mask detects edges from right to left.

Hence, to better detect the edges, we must take the absolute or squared value of the result.

```
figure
imshow(uint8(abs(sobel_v_img)))

figure
imshow(uint8(abs(sobel_h_img)))
```



Vertical Sobel Mask with absolute value



Horizontal Sobel Mask with absolute value

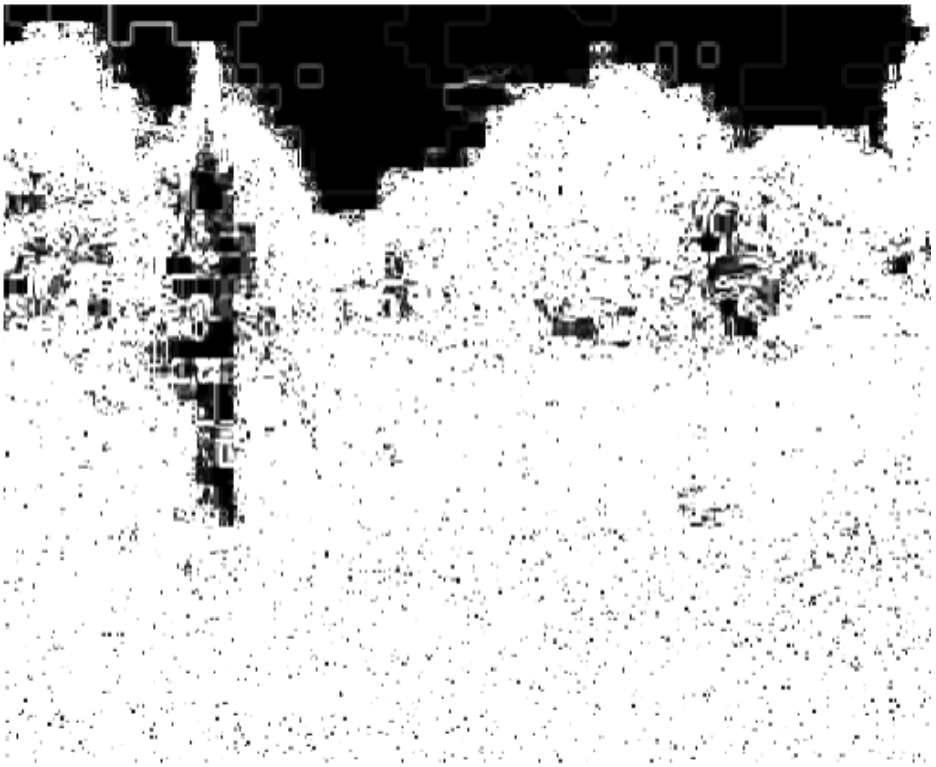
- Although the Sobel masks were meant to detect horizontal and vertical lines, some diagonal edges are detected.
- For the edges that are not perfectly horizontal or vertical, they differ because of their angle/ gradient. If the gradient is small or the edge only has a small angle and is mostly straight, these edges will be detected by these Sobel masks.
  For example :
  - Diagonal edges of the trees and of the dome-like roof at the top right hand corner are identified with the vertical Sobel Mask.
  - Some diagonal edges of the trees and runners' legs are detected through the horizontal Sobel Mask.
  - Edges with larger gradients/angles are not detected, like the bottom parts of the trees where the lines are much steeper.

(c) Combined Edge Image:

<<Input>>

```
E = sobel_v_img.^2 + sobel_h_img.^2;
imshow(uint8(E));
```

<<Output>>



- The combined edge image seems to be made up of mostly high intensity pixels. This is caused by adding the squared values which amounted to large values for pixels.
- To obtain the true absolute gradient, from our lecture notes we see that a square root must be applied to these combined Gx^2 and Gy^2 values.

Applying square root to the obtained (Gx^2 + Gy^2):
<<Input>>

```
E = sqrt(E);
imshow(uint8(E));
```

<<Output>>

- After using the square root value, the horizontal,vertical and diagonal edges are displayed clearly.
- By squaring, the actual magnitude of edges in opposite directions are detected, despite their negative values.
- Applying square root to the combined squares of Gx and Gy values, will give the gradient magnitude's absolute value.
- Some noisy pixels are observed as most pixels are non-zero since thresholding hasn't been applied yet.

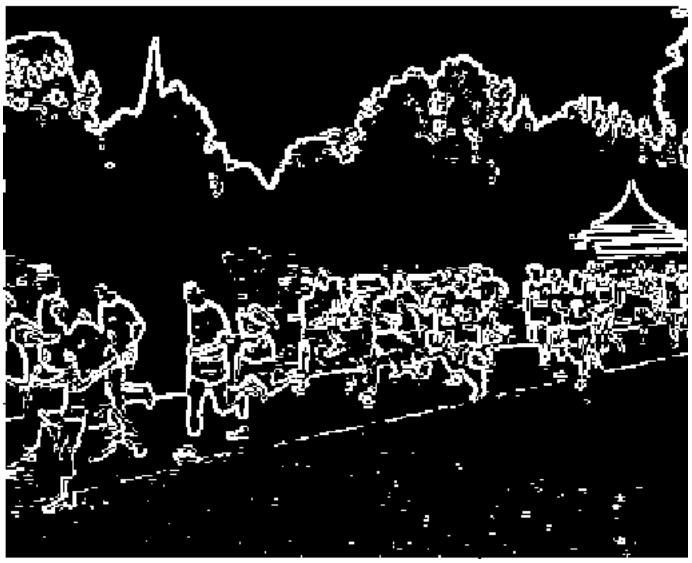
(d) Threshold the edge image E at value t:
<<Input>>

```
t1 = 10000;
Et1 = E>t1;
imshow(Et1)
figure

t2 = 30000;
Et2 = E>t2;
imshow(Et2)
figure

t3 = 50000;
Et3 = E>t3;
imshow(Et3)
figure

t4 = 80000;
Et4 = E>t4;
imshow(Et4)
```

<<Output>>

Threshold = 10,000

Threshold = 30,000



Threshold = 50,000

Threshold = 80,000

1. Thresholding the image allows the pixels to have equal brightness on a binary level.
2. With smaller thresholds, most crucial edges are detected. However, there is also more noise detected from these small edges.
3. With larger thresholds, most of the noise is not detected. However, some actual edges are undetected and edges also get separated into smaller edges if their magnitude is not as high as the threshold values.
4. Due to this tradeoff between detecting more edges or more noise, a balance must be found in choosing an appropriate threshold value.

(e) Canny Edge Detection:

(i) Sigma values from 1.0 to 5.0

<<Input>>

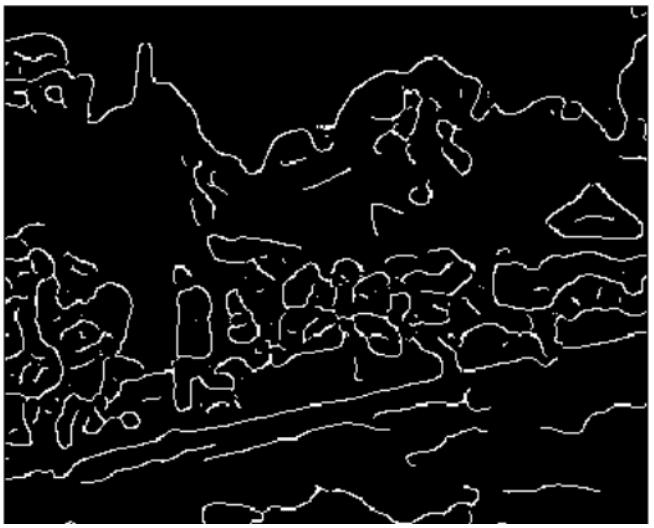
```
tl = 0.04;
th = 0.1;

sigma1 = 1.0;
E = edge(img, 'canny', [tl th], sigma1);
figure
imshow(E)

sigma2 = 2.0;
E = edge(img, 'canny', [tl th], sigma2);
figure
imshow(E)

sigma3 = 3.0;
E = edge(img, 'canny', [tl th], sigma3);
figure
imshow(E)

sigma4 = 4.0;
E = edge(img, 'canny', [tl th], sigma4);
figure
imshow(E)

sigma5 = 5.0;
E = edge(img, 'canny', [tl th], sigma5);
figure
imshow(E)
```

<<Output>>

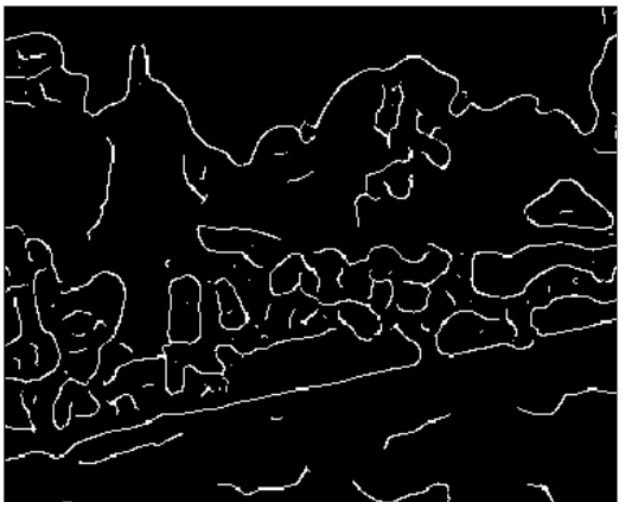
Sigma = 1.0

Sigma = 2.0

Sigma = 3.0



Sigma = 4.0



Sigma = 5.0

Sigma refers to the standard deviation of the Gaussian kernel used to convolve with the image. Higher sigma value denotes a higher contribution of pixels for edge detection, while lower sigma value denotes lesser pixels contributing to the edge detection.

1. Smaller Sigma values detect more noisy edges, while larger Sigma values detect less noisy edges
2. Location accuracy is better for smaller Sigma values while it gets worse for larger Sigma values. As the sigma values increase, the edges are simplified

The Canny Edge detector uses gaussian filters to remove noise and smoothen the edges. With large Sigma values, the image will have lesser noise due to higher smoothing but also losing the fine details and definitions. Hence, the larger sigma values have lower noise and lower location accuracy for edge detections. The tradeoff in choosing an appropriate sigma value will either be to reduce noise or increase location accuracy.

(ii) Changing the values of tl:

<<Input>>

```
th = 0.1;
sigma = 1.0;

tl1 = 0.01;
E1 = edge(img, 'canny' ,[tl1 th], sigma);
figure
imshow(E1);

tl2 = 0.02;
E2 = edge(img, 'canny' ,[tl2 th], sigma);
figure
imshow(E2);

tl3 = 0.04;
E3 = edge(img, 'canny' ,[tl3 th], sigma);
figure
imshow(E3);

tl4 = 0.08;
E4 = edge(img,'canny',[tl4 th],sigma);
figure
imshow(E4);
```

<<Output>>

tl = 0.01

tl = 0.02

tl = 0.04

tl = 0.08

Canny Edge detector adopts hysteresis thresholding as a part of its algorithm. The tl value is the lower bound of this hysteresis thresholding and is used to determine if a pixel can be detected as an edge. Any edges with magnitude below this lower bound tl value, will not be detected.

1. With smaller tl values, more edges are detected. Although the edges are more defined, more noise is also detected.
2. With larger tl values that approach th, there are lesser edges detected. Although the edges are not as defined, lesser noise is also detected.

The tradeoff here will hence be between having more defined edges or lesser noise.

3.2 Line finding using Hough Transform

(a) Reuse edge image with Sigma = 1.0 :

<<Input>>

```
P = imread('macritchie.jpg');
I = rgb2gray(P);
tl = 0.04; th = 0.1; sigma = 1.0;

E = edge(I,'canny',[tl th],sigma);
imshow(E);
```

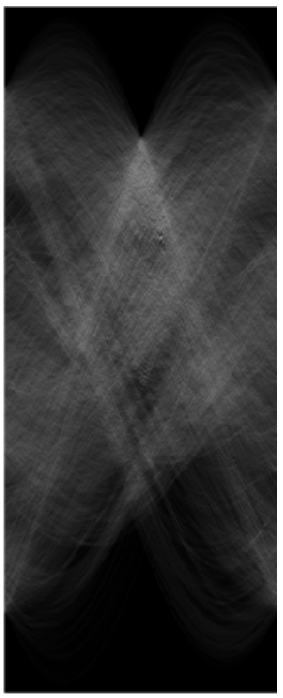
<<Output>>

(b) Compute Hough Transform with Radon Transform :

<<Input>>

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

<<Output>>



1. From the definition of Radon function, it computes the projection of image intensity on a radial line at a specific angle.
2. Points in a spatial domain that lie on a line with a specific angle, will be mapped to an equivalent point of the theta domain. This is essentially equivalent to Hough Transform.
3. The bright spots in the transformed image describe the various points that lie on a line with an angle.
4. Radon transform takes the pixel intensities of the image into account for a non-binary image. Hence, if the pixel intensity is between [0,255] as opposed to [0,1], Radon transform and Hough transform will produce different results.

(c) Location of maximum pixel intensity

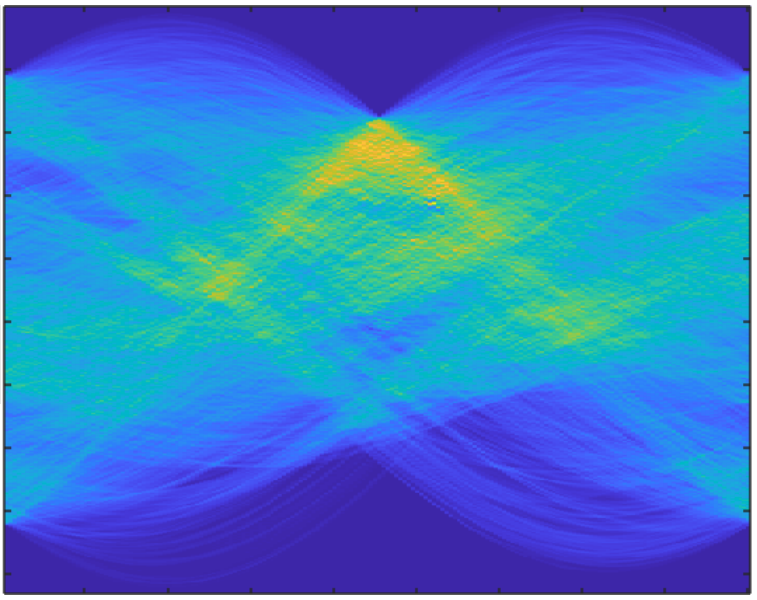
<<Input>>

```
[radius, theta] = find(H>=max(max(H)));
radius; theta;
```

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

<<Output>>

```
radius =

   157


theta =

   104
```



- The maximum pixel intensity is found at location [157,104].

(d) Equations to convert [theta,radius] line representation to normal line equation:

- From the definition of the pol2cart function, with the input of the angle in radians, it transforms polar coordinates into cartesian coordinates.
- So, we pass the value of theta into pol2cart after converting it into radians.
- From the definition of Hough point transformation:

  C = x * cos(theta * pi/180) + y * sin(theta * pi/180)
- B = -B is done to negate the y axis which is pointing downwards in the image as opposed to the y-axis pointing upwards.

<<Input>>

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

- The theta and radius will now be transformed into the same coordinate system adopted for the image, so a line can be superimposed on the same plane.
- To convert the Hough transform with origin as the centre and make the origin to be at the top left corner. The center of the image is calculated by dividing the respective size of X and Y axes by two.

```
[y_size, x_size] = size(I);
x_centre = x_size/2;
y_centre = y_size/2;
C = A*(A+x_centre) + B*(B+y_centre);
```

<<Output>>

x_centre	179	1×1	double
y_centre	145	1×1	double
A	17.0963	1×1	double
B	74.0521	1×1	double
C	1.9574e+04	1×1	double

(e) Compute yl and yr values for xl = 0 and xr = (width of image - 1) = 357

<<Input>>

- On rearranging the line equation Ax+By=C to By = C-Ax we can obtain yl & yr.

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

xr = x_size -1;
yr = (C - A * xr) / B;
```

<<Output>>

yl	264.3245	1×1	double
yr	181.9046	1×1	double

(f) Superimpose estimated line on initial image

<<Input>>

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

<<Output>>

1. The superimposed line appears to fit pretty well on to the nearer edge of the running path in the image. After further observation, the right end of the superimposed line seems to be slightly misaligned.
2. The reasons for misalignment might be:
    a. Use of integers from the x_center of [0,179], may not produce the most precise theta value.
    b. The actual path in the image may not be perfectly straight and could be slightly curved. In this case, a linear function will not be able to trace it.
    c. The calculation for the Canny edge detection and Hough Transforms could have lost some precision. Despite being a strict line, the calculated line equation may not perfectly represent this. This is caused due to the Radon transformation's application on discrete functions as opposed to continuous functions.

3.3 3D Stereo

(a) Disparity map algorithm:

- The idea behind the disparity map algorithm is to search along the scan line template, with the smallest sum of squared difference. The search is limited up to 15 pixels along the scan line (to the left).
- Input the left and right Matrix of the given grayscale image.

```
function res_map = Dmap(I_l, I_r)
[x,y] = size(I_l);
temp_x = 11; temp_y = 11;
center_x = floor(temp_x/2); center_y = floor(temp_y/2);
search_lim = 14;
res_map = ones(x-temp_x + 1, y-temp_y + 1);

for i = 1+center_x: x-center_x
    for j = 1+center_y: y-center_y
        curIpatch_r = I_l(i-center_x : i+center_x, j-center_y : j+center_y);
        curIpatch_l = rot90(curIpatch_r, 2);
        min_diff = inf; min_coor = j;
        for k = max(1+center_y, j-search_lim) : j
            T = I_r(i-center_x : i+center_x, k-center_y : k+center_y);
            curIpatch_r = rot90(T,2);

            c_1 = conv2(T,curIpatch_r); c_2 = conv2(T,curIpatch_l);
            ssd = c_1(temp_x, temp_y)-2*c_2(temp_x, temp_y);
            if ssd < min_diff
                min_diff = ssd;
                min_coor = k;
            end
        end
        res_map(i-center_x, j-center_y) = j-min_coor;
    end
end
```

(b) Download synthetic stereo pair images and convert to grayscale:

<<Input>>

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

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

<<Output>>

(c) Run the algorithm to obtain disparity maps of the stereo pair images:

<<Input>>

```
D = Dmap(l, r);
figure
imshow(D,[-15 15]);

res = imread('corridor_disp.jpg');
figure
imshow(res);
```

<<Output>>

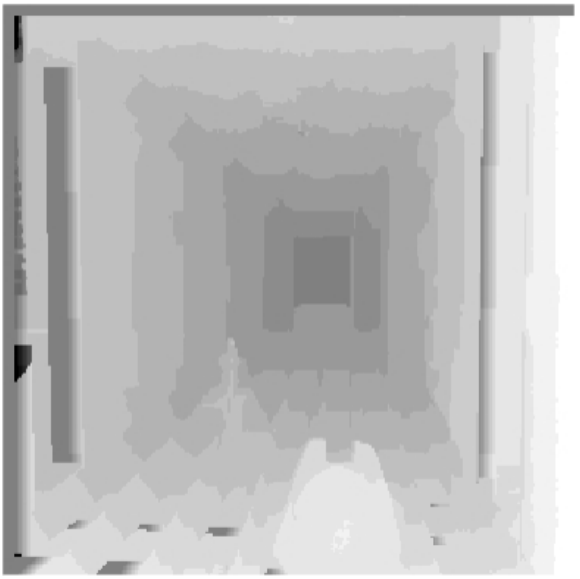


Image from the disparity map *corridor_disp.jpg*

1. The disparity map had mapped the corridor pretty much as expected. The corridor gradually appears darker along with the sphere and cone objects mapped.
2. The quality of the disparity mapping is also similar to that of the expected image.
3. The image obtained from the disparity map shows a further distance which starts with bright spots and leads to be darker, also resulting in a darker disparity map.
4. The biggest difference is seen at the end of the corridor, where the disparity mapped image appears much darker. This is due to the centre portion of the original images having no disparities as they are pretty much a plain white wall.
5. The actual image having a lower contrast might cause the disparity map detection to be not too accurate..

(d) Rerun the algorithm to obtain disparity maps on the triclops images.

<<Input>>

```
l = imread('triclopsi2l.jpg');
l = rgb2gray(l);
figure
imshow(l);

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

D = Dmap(l, r);
figure
imshow(D,[-15 15]);

res = imread('triclopsid.jpg');
figure
imshow(res);
```

<<Output>>

triclopsi2l.jpg

triclopsi2r.jpg

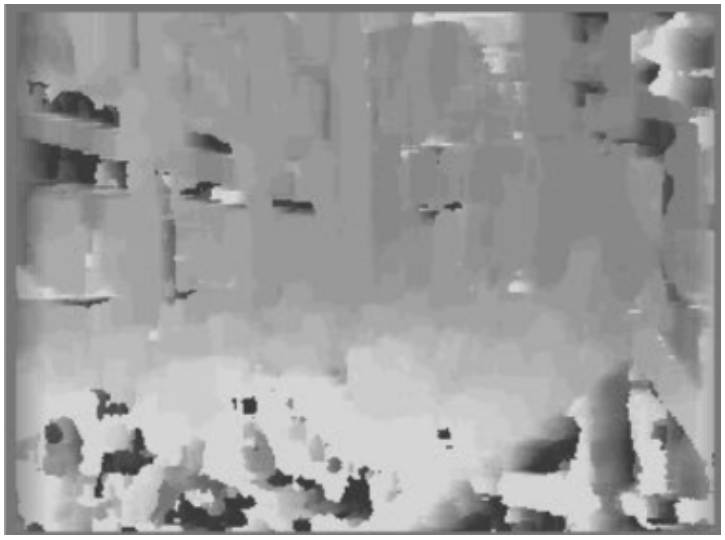
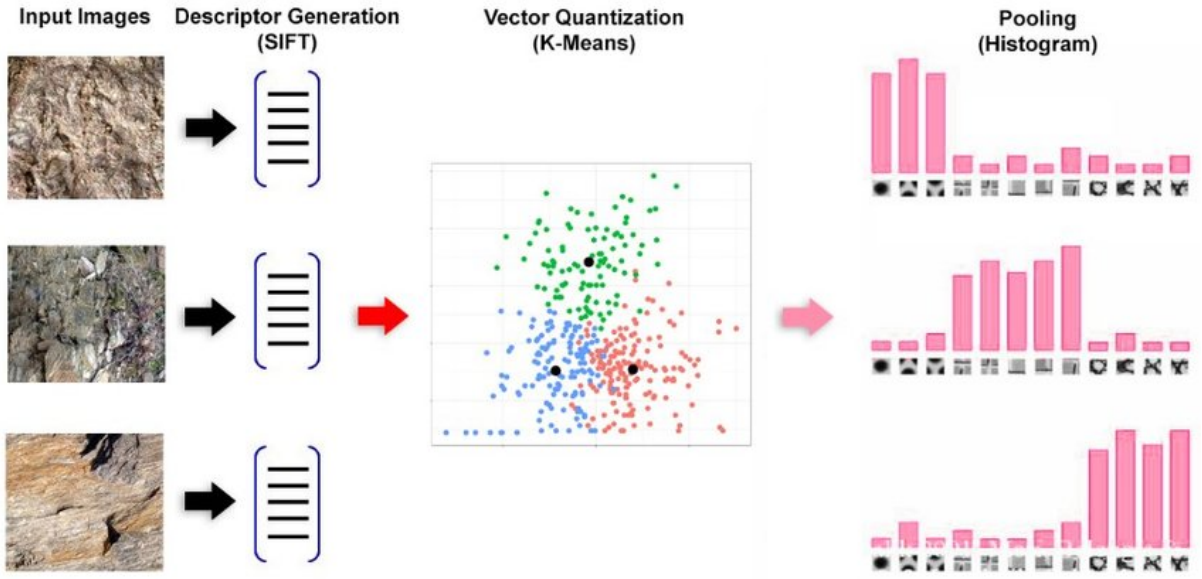


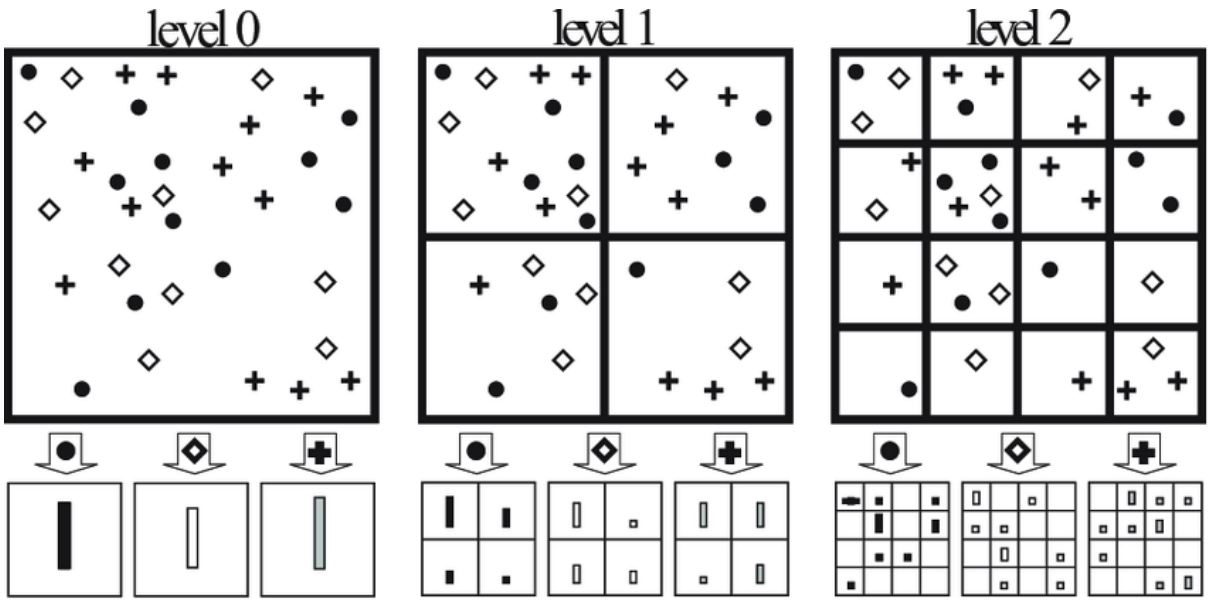
Image from disparity map *triclopsid.jpg*

1. The image from disparity mapping appears to have been correctly mapped and is similar to the expected image.
2. The quality of the image from disparity mapping can also be said to be good, similar to the corridor's image.
3. On further observation, some parts of the disparity mapped image are different from the expected image:
    a. The pathway of the disparity mapped image and expected image are slightly different. This could have been due to the pixels having almost similar colours and hence not much disparity to be mapped by SSD.
    b. The actual image has less contrast compared to the corridor's image. This lower contrast had caused the disparity map detection to be inaccurate.
    c. As similar to the corridor image, the original triclops image having a different scan line, some parts will not be correctly mapped.

3.4 Spatial Pyramid Matching (SPM) & Bag-of-Words(BoW): (OPTIONAL)



The bag-of-words algorithm and SPM models are used for feature extraction and are very famous for machine learning and NLP models. The number of SIFT features /unique features are counted and stored in a histogram using K means clustering. The similar SIFT features are grouped together to uniquely separate from other features.



The Spatial Pyramid Matching algorithm is a better version of the bag-of-words method by considering the location of the features. Bag-of-words only uses the count of the features and ignores the location of the feature.
In SPM, the image is split into different regions and the number of features in each region is counted separately using K-means clustering and finding the key features through their histograms. Due to the division of regions, the spatial order or location of features are extracted. Thus, based on the highest magnitude of a feature, we can tell the type of object .

Set dataset + Training and Test splits

```
class_names = [name.split('\\')[1] for name in glob.glob('caltech-101/*')]
class_names = dict(zip(range(0,len(class_names)), class_names))
numOfClasses = len(class_names)

def set_Dataset(path, num_per_class, classes):
    data = []
    labels = []

    for id, class_name in classes.items():
        img_path_class = glob.glob(path + class_name + '/*.jpg')

        if num_per_class > 0:
            img_path_class = img_path_class[:num_per_class]

        labels.extend([id]*len(img_path_class))

        for filename in img_path_class:
            data.append(cv2.imread(filename, 0))

    return data, labels

data, label = set_Dataset('caltech-101/*', 80, class_names)

X_train, X_test, y_train, y_test = train_test_split(data, label, test_size = 50/80, random_state = 42)
print(f"train_data_size: {len(X_train)}, test_data_size: {len(X_test)}")
```

```
train_data_size: 2291, test_data_size: 3820
```

Extracting SIFT features and put in Histogram

```
def putSPMinHistogram(L, data, kmeans, k):
    x = []
    for i in range(len(data)):
        hist = SPMImageFeatures(L, data[i], kmeans, k)
        x.append(hist)
    return np.array(x)

def extract_dSIFT(img):
    ds_Step_Sz = 2
    sift = cv2.xfeatures2d.SIFT_create()
    disft_size = ds_Step_Sz
    keypoints = [cv2.KeyPoint(x, y, disft_size)
            for y in range(0, img.shape[0], disft_size)
                for x in range(0, img.shape[1], disft_size)]
    desc = sift.compute(img, keypoints)[1]
    return desc
```

Perform K means clustering based on the respective SIFT features extracted

```
d_train = []
for i in tqdm(range(len(x_train_SIFT))):
    for j in range(x_train_SIFT[i].shape[0]):
        d_train.append(x_train_SIFT[i][j,:])

d_train = np.array(d_train)
```

```
100% 2291/2291 [00:00<00:00, 6125.66it/s]
```

```
def Feature_k_clustering(d_train, k):
    kmeans = KMeans(n_clusters=k, random_state=0).fit(d_train)
    return kmeans
```

Compute the dense SIFT

```
def SIFT_comp(data):
    x = []
    for i in range(len(data)):
        sift = cv2.xfeatures2d.SIFT_create()
        img = data[i]
        step_size = 15
        kp = [cv2.KeyPoint(x, y, step_size) for x in range(0, img.shape[0], step_size) for y in range(0, img.shape[1], step_size)
        dense_feat = sift.compute(img, kp)
        x.append(dense_feat[1])

    return x

x_train_SIFT = SIFT_comp(X_train)
```

Spatial Pyramid Matching

```
def SPMImageFeatures(L, img, kmeans, k):
    W = img.shape[1]
    H = img.shape[0]
    h = []
    for l in (range(L+1)):
        w_step = math.floor(W/(2**l))
        h_step = math.floor(H/(2**l))
        x, y = 0, 0
        for i in range(1,2**l + 1):
            x = 0
            for j in range(1, 2**l + 1):
                desc = extract_dSIFT(img[y:y+h_step, x:x+w_step])
                predict = kmeans.predict(desc)
                histo = np.bincount(predict, minlength=k).reshape(1,-1).ravel()
                weight = 2**(l-L)
                h.append(weight*histo)
                x = x + w_step
            y = y + h_step
    hist = np.array(h).ravel()
    dev = np.std(hist)
    hist = hist - np.mean(hist)
    hist = hist - dev
    return hist
```

Conclusion

As defined above, BoW has lower classification accuracy when compared to SPM. Without taking into account the spatial area of each SIFT feature, BoW was lower than SPM. The final obtained accuracy of classification was:

BoW: 39% while **SPM: 48%.**