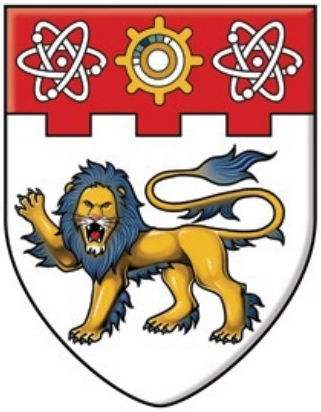


NANYANG TECHNOLOGICAL UNIVERSITY

**CZ4003:**
**Computer Vision**

# *Lab 2:*

*Edge Detection + Hough Transform + SSD + SPM*

by
Chulpaibul Jiraporn (U1822666H)

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**
**NANYANG TECHNOLOGICAL UNIVERSITY**

**Experiment 1: Edge Detection**

a)

**Code**

```
% download `macritchie.jpg, convert to grayscale, display the image.
I = imread('macritchie.jpg');
I = rgb2gray(I);
imshow(I);
```

**Result**


*Fig. 1*

b)

**code**

```
% create 3x3 horizontal & vertical Sobel masks,
sobel_h = [
    -1 -2 -1;
    0 0 0;
    1 2 1;
];

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

% filter the image using conv2
f_h = conv2(I, sobel_h);
f_v = conv2(I, sobel_v);

% display the edge-filtered images
```

```
subplot(1,2,1); imagesc(f_h); axis off; title('Convoluted with horizontal
filter');
subplot(1,2,2); imagesc(f_v); axis off; title('Convoluted with vertical
filter');
colormap(gray);
```

**result**

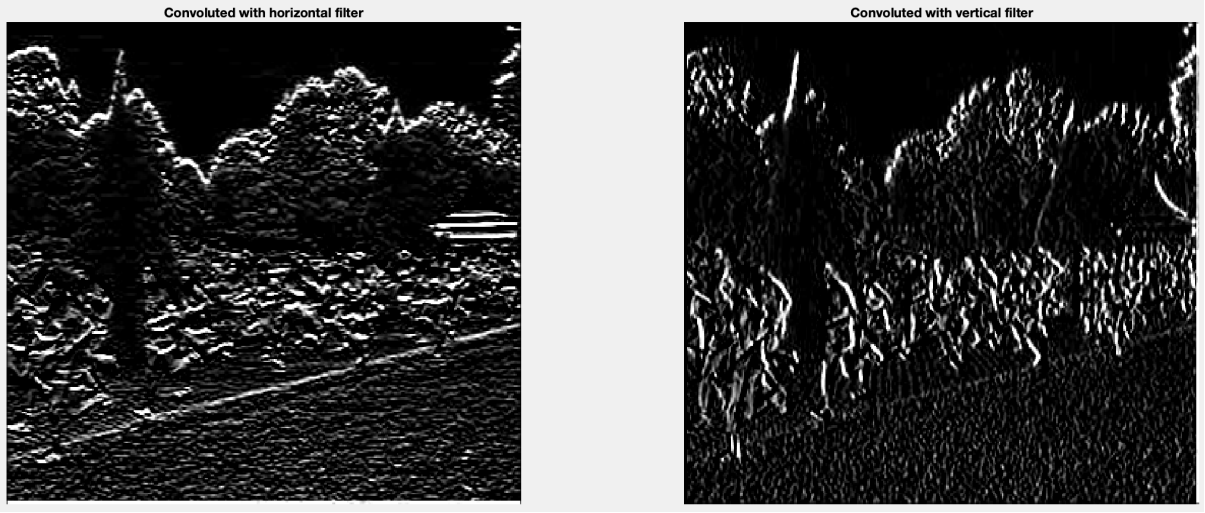



*Fig. 2*

**Discussion**

From Fig.2 above, it can be observed that horizontal and vertical Sobel masks detect the horizontal and vertical edges respectively. Sobel filters calculate the gradient for each pixel to detect edges. The image on the left of Fig.2 shows the result after convolution of the image with horizontal Sobel filter. We can clearly see the horizontal line of the pavilion in the background, horizontal lines of people's arms and the edges between the path and the grass. While on the right of Fig.2, we can observe the vertical lines of the pavilion, vertical edges of people and the vertical edges of the tree are more visible. When there is a diagonal edges, they can be decomposed into horizontal and vertical component which can still be detected by the horizontal and vertical Sobel mask respectively. Thus, we can combined the two results to detect diagonal edges.

c)

**Code**

```
% combined edge image by squaring and adding the squared images
E = f_h.^2 + f_v.^2;
imshow(uint8(E));
```

**Results**

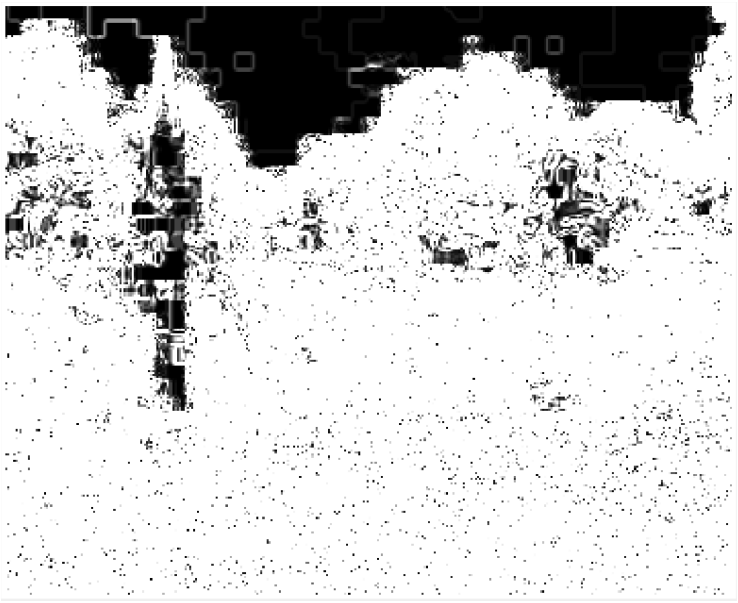


*Fig. 3*

**Discussion**

Fig.3 shows the combined result of horizontal and vertical edge images. Firstly, it can be observed that most of the pixels are white as compared to Fig.2. This is because the combined result contains horizontal, vertical and diagonal edges. We can also infer that almost every pixel is an edgel except the top part where the region has a uniform colour in the original image. Squaring operation is applied to obtain only the magnitude of the gradient because when Sobel filters are applied, the gradient of each pixel can be negative. However, we are only concerned about the magnitude when detecting edges. The direction of the gradient can be found using inverse tangent (arctan). We can then apply thresholding to select edgels with gradient greater than a threshold.

d)

**Code**

```
% threshold the edge image E at value t by
t = 50;
Et = E>t;

% try different threshold values
t2 = 100;
Et2 = E>t2;
t3 = 5000;
Et3 = E>t3;
t4 = 10000;
Et4 = E>t4;
t5 = 50000;
Et5 = E>t5;
t6 = 100000;
Et6 = E>t6;

% display the binary edge images
subplot(3,2,1); imagesc(uint8(Et)); axis off;
subplot(3,2,2); imagesc(uint8(Et2)); axis off;
subplot(3,2,3); imagesc(uint8(Et3)); axis off;
subplot(3,2,4); imagesc(uint8(Et4)); axis off;
subplot(3,2,5); imagesc(uint8(Et5)); axis off;
subplot(3,2,6); imagesc(uint8(Et6)); axis off;
colormap(gray);
```

**Results**

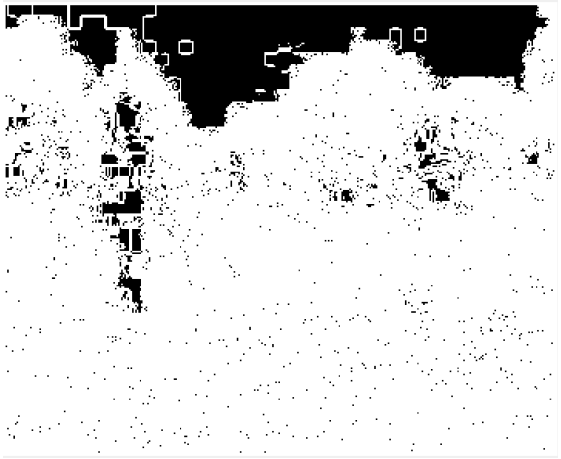


*Fig. 4*

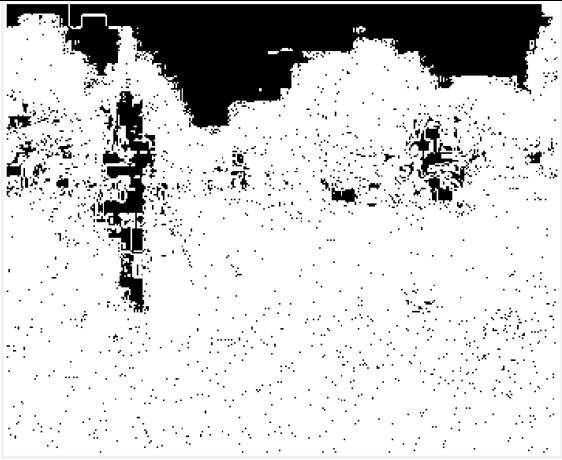


*Fig. 5*

*Fig. 6*

*Fig. 7*

*Fig. 8*

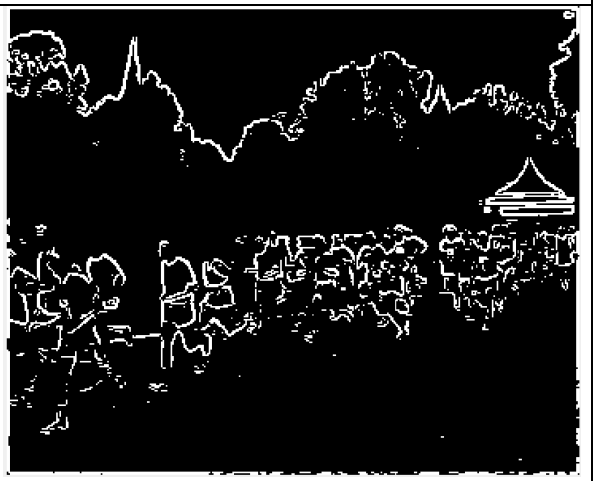


*Fig. 9*

**Discussion**

Fig. 4, 5, 6, 7, 8, 9 shows the edge image with threshold t equal to 50, 100, 5000, 10000, 50000, 100000 respectively. We can observe that as the threshold increases, the number of edgels decreases. We can also observed that the thickness of some edgels decrease with larger threshold. A small threshold means that most of the edgels will be accepted including those that caused by noise.

A large threshold means that only edgels with gradient magnitude larger than the threshold will be accepted, resulting in some desired edges being left out.
Advantage of small threshold: Edgels with low gradient magnitude can be retained. These edgels maybe important in some cases.
Disadvantage of small threshold: Noisy edgels are also retained as these edgels often have small magnitude of gradient due to Gaussian noise.

Advantage of large threshold: Noisy edgels are mostly removed as they often have small magnitude of gradient.
Disadvantage of large threshold: Some important edgels maybe removed if they have low gradient magnitude.
Hence, the threshold values will have to be selected while keeping in mind the trade-off between noise and desired edgels.

ei)
**Code**

```
tl=0.04;
th=0.1;
sigma=1.0;
E = edge(I,'canny',[tl th],sigma);
imagesc(uint8(E));
colormap(gray);
axis off;

% try different values of sigma ranging from 1.0 to 5.0
sigma2=2.0;
E2 = edge(I,'canny',[tl th],sigma2);
sigma3=3.0;
E3 = edge(I,'canny',[tl th],sigma3);
sigma4=5.0;
E4 = edge(I,'canny',[tl th],sigma4);

% display edge images
subplot(2,2,1); imagesc(uint8(E)); axis off;
subplot(2,2,2); imagesc(uint8(E2)); axis off;
subplot(2,2,3); imagesc(uint8(E3)); axis off;
subplot(2,2,4); imagesc(uint8(E4)); axis off;
colormap(gray);
```

**Results**

*Fig. 10 (sigma = 1.0)*

*Fig. 11 (sigma = 2.0)*

*Fig. 12 (sigma = 3.0)*

*Fig. 13 (sigma = 5.0)*

**Discussion**

Fig. 10, 11, 12, 13 show edge images as a result of canny edge detection using sigma values of 1, 2, 3, 5 respectively. Canny edge detector uses Guassian filter to reduce noise and sigma is a parameter used to determine the standard deviation of Gaussian filter. As can be observed, as sigma increases, more noise are removed from the original image, resulting in fewer noisy edgels. However, a high sigma not only remove noise, it also removed details from the image. Thus, in Fig. 13, there are only a few edgels left and it can also be observed that the location of edgels is less accurate as details are removed. Hence, a lower sigma results in less noisy edgel removal but a more accurate location of edgel while a higher sigma results in more noise edgel removal but a less accurate location of edgel.

eii)

**Code**

```
tl2 = 0.01;
tl3 = 0.08;
E5 = edge(I,'canny',[tl2 th],sigma);
E6 = edge(I,'canny',[tl3 th],sigma);

% display edge images
subplot(2,2,1); imagesc(uint8(E5)); axis off;
subplot(2,2,2); imagesc(uint8(E6)); axis off;
colormap(gray);
```

**Results**

*Fig. 14 (sigma = 1.0, tl = 0.01)*

*Fig. 15(sigma = 1.0, tl = 0.08)*

**Discussion**

Fig. 14 and 15 show the edge image when tl = 0.01 , sigma = 1.0 and tl = 0.01. sigma = 1.0 respectively. As can be observed, as tl increases, the number of edgel in edge image decreases. Canny algorithm uses hysteresis thresholding where the edgel will be discarded if the gradient magnitude is lower than tl while edgel with gradient magnitude greater than th will be kept. The edgel with values from tl to th will be kept if a pixel has neighboring pixels perpendicular to the edge gradient which have already been set to 1. Thus, when tl increases, the lower threshold is higher and more edgel are discarded, resulting in fewer edges in Fig. 15.

**Experiment 2: Line Finding using Hough Transform**

a) Edge map from experiment 1 is reused.

b) MATLAB help about Radon transform:

--- help for radon ---

radon Radon transform.
The radon function computes the Radon transform, which is the projection of the image intensity along a radial line oriented at a specific angle.

….if you omit THETA, it defaults to 0:179.

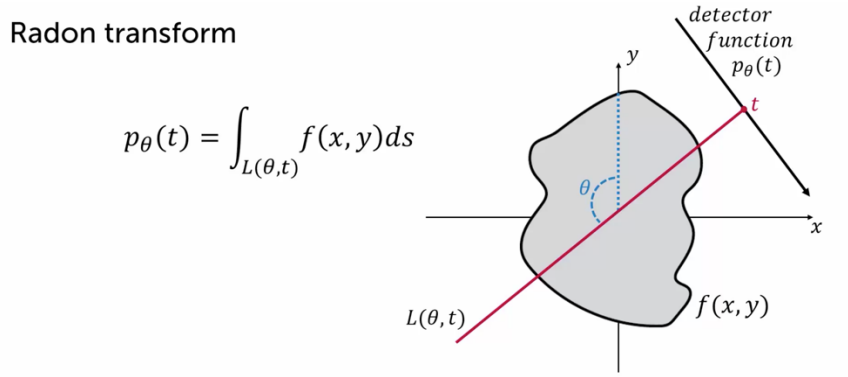



*Fig. 16*[1]

From MATLAB help function, it can be understand that radon transform is a projection of image intensity along radial line with oriented at a specific angle. The radial line can be uniquely defined by the parameter θ, t. In the case of binary image like the edge map in this case where each pixel is either 0 (black) or 1 (white), value of each point in the Radon space equal to the sum of the total number of pixel that are white (because black is 0) which is equivalent to Hough transform. In addition, Fig. 16 shows how radon transform works and its equation but the angle θ should be measured from x-axis (ignore the axis because it is different from MATLAB). As can be seen, if points that lie on the same straight line will be mapped to the one point after radon transform because they have the same θ and t. In the case of application of radon transform on edge image, if there are many pixels lie on the same straight line then the point θ, t in the radon space will be brighter which makes Radon transform equivalent to Hough transform. Both Radon and Hough transform map the line in cartesian coordinates into a point in parametric space and has same range of θ . While the Radon transform derives a point in parametric space (θ, t) to from image space, the Hough transform maps pixels from image space to parametric space. However, Radon transform and Hough transform are not equivalent in non-binary image.

**Code**

```
[H, xp] = radon(E);
subplot(1,1,1); imagesc(0:179, xp, H); colormap(gray);
xlabel('\theta (degrees)')
ylabel('x''')
```

**Results**

[1] https://youtu.be/MA2y_2YySq0

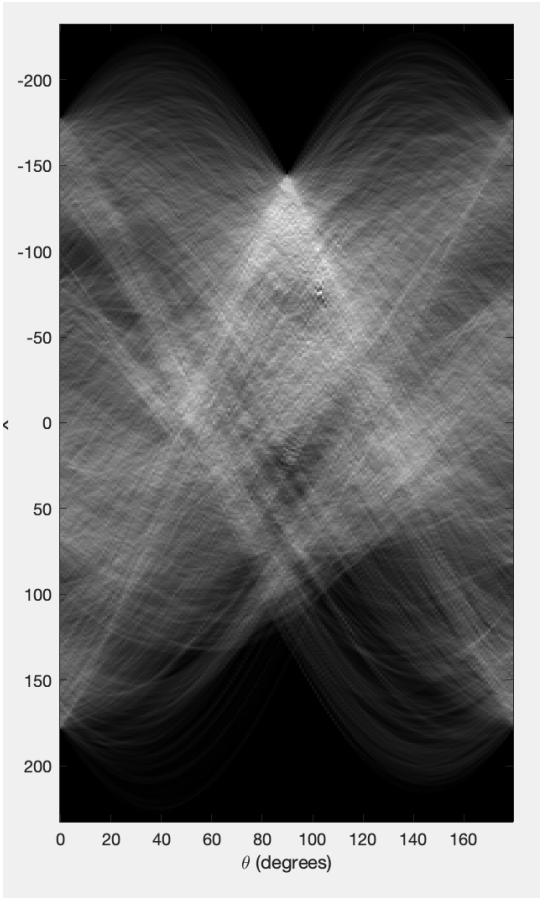



*Fig. 17*

**Discussion**

Fig. 17 shows the image of H which is the Radon transform of the original image.

c)

**Code**

```
maximum = max(max(H));
[d,theta]=find(H==maximum);
radius = xp(d);
theta = theta - 1;
```

**Results**

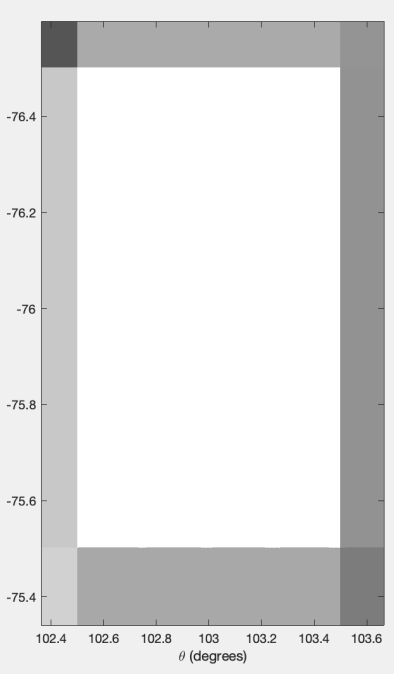



*Fig. 18*

**Discussion**
Fig. 18 shows the brightest pixel where theta = 103 and radius = -76 [103, -76] in the Radon transform of Fig. 17.

d)
With reference to Fig. 19 $\rho$ and $\theta$ correspond to radius and theta by definition respectively. N is a normal vector from origin O to the blue line which can be represented as $\rho(\cos\theta, \sin\theta)$. Since N is perpendicular to the blue line,
$N\cdot[(x,y)-N] = 0$ where x,y can be the coordinate of any point on blue line. We can then expand the equation as $\rho(\cos\theta,\sin\theta)\cdot[(x,y) - \rho(\cos\theta,\sin\theta)] = 0$. By simplifying the equation, we can get
$x\cos\theta + y\sin\theta = \rho$ where $(\rho,\theta)$ corresponds to a unique line. Thus, we convert the [theta, radius] line representation to the normal line equation form *Ax + By = C in image coordinates* where
$A = \cos\theta$, $B = \sin\theta$ and $\rho = C$.
Fig. 20 shows how function pol2cart in MATLAB works which is exactly the same as our equations above. Thus, we can use pol2cart to find A and B:

$$A = \rho\cos\theta$$
$$B = \rho\sin\theta$$

where $C = \rho^2$

To solve for C, since Hough transform is done with respect to an origin at the centre of the image, and we will need to convert back to image coordinates where the origin is in the top-left corner of the image. We will derive the following equation:

$$A(x-179) + B(y-145) = \rho^2$$

$$Ax + By = \rho^2 + 179A + 145B$$

Since $\rho^2 = A^2 + B^2$, we can get:

$$Ax + By = A^2 + B^2 + 179A + 145B$$

Hence, the new C is $A^2 + B^2 + 179A + 145B$.

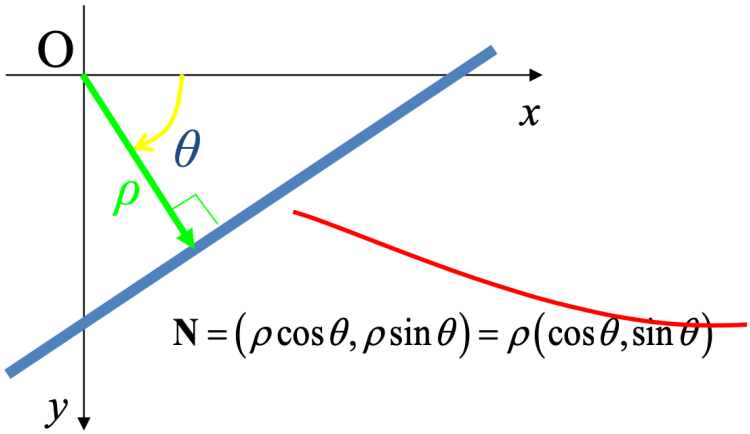
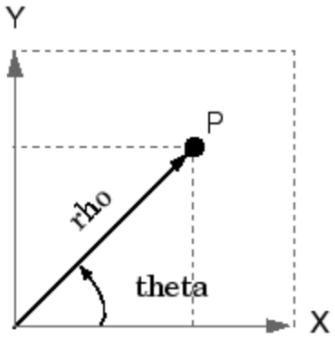



*Fig. 19*[2]




**Polar to Cartesian Mapping**

```
x = rho*cos(theta)
y = rho*sin(theta)
```

*Fig. 20*[3]

**Code**

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

C = (A^2 + B^2) + 179*A + 145*B
```

**Results**

[2] From lecture notes L4 Edges

[3] https://www.mathworks.com/help/matlab/ref/pol2cart.html

```
A =

    17.0963

B =

   74.0521

C =

   1.9574e+04
```

e)

**Code**

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

**Results**

```
yl =

  342.3236

yr =

  259.9037
```

*Fig. 21*

**Discussion**

From the linear equation Ax + By = C, we can change it to y = (C – Ax)/B which can be used to solve for y value. The width of the image is 358, therefore width -1 = 357.

f)

**Code**

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

**Results**

*Fig. 22*

**Discussion**

From part c) to part e), we get the brightest pixel (straight line with most votes) in the Radon transform and convert it to the straight line in the image as shown in Fig.22. The result shows a line that match up the edge of the running path but it did not match up exactly the right of the image. This is because there can be small precision error conversion from Radon transform parameters to the coordinate in image space. In addition, it could be due to the actual path edge may not be exactly straight.

## Experiment 3

1) The algorithm is implemented as a function disparityMap(pl, pr, template_x, template_y) where pl, pr, template_x and template_y are left image, right image, dimension of template in x direction and dimension of template in y direction respectively. Note that in the function, we use the equation below but omitting the second convolution calculation because it does not affect the result and this will reduce computation resource.

$$S(x,y)=\sum_{j=0}^{M}\sum_{k=0}^{N}I^2(x+j,y+k).1+\sum_{j=0}^{M}\sum_{k=0}^{N}T^2(j,k)-2\sum_{j=0}^{M}\sum_{k=0}^{N}I(x+j,y+k)T(j,k)$$

Rotation needs to be performed before convolution to achieve the same effect as correlation.

```
function map = disparityMap(pl, pr, template_x, template_y)
    [m, n] = size(pl);
    map = ones(m - template_y + 1 , n - template_x + 1);
    x_half = floor(template_x/2);
    y_half = floor(template_y/2);

    for j = 1+y_half:m-y_half
        for i = 1+x_half:n-x_half
            patch_l = pl(j-y_half:j+y_half,i-x_half:i+x_half);
            lower_x = max(1+x_half,i-15);
            upper_x = min(m-x_half,i+15);
            min_ssd = inf;
            min_ssd_idx = i;
            for x = lower_x:upper_x
                patch_r = pr(j-y_half:j+y_half,x-x_half:x+x_half);
                temp_r = rot90(patch_r,2);
                i_square = conv2(patch_r,temp_r, 'same');
                c = 2.*conv2(patch_l,temp_r, 'same');
                ssd = i_square(y_half+1,x_half+1) - c(y_half+1,x_half+1);
                if ssd < min_ssd
                    min_ssd = ssd;
                    min_ssd_idx = x;
                end
            end
            if j == 115
                if i == 129
                    i - min_ssd_idx
                end
            end
            map(j-y_half, i-x_half) = i - min_ssd_idx;
        end
    end
end
```

2)

**Code**

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

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

disp = imread('corridor_disp.jpg');
imagesc(disp);

%%
D = disparityMap(l, r, 11, 11);
imshow(D,[-15 15]);colormap(gray);
```

## Results

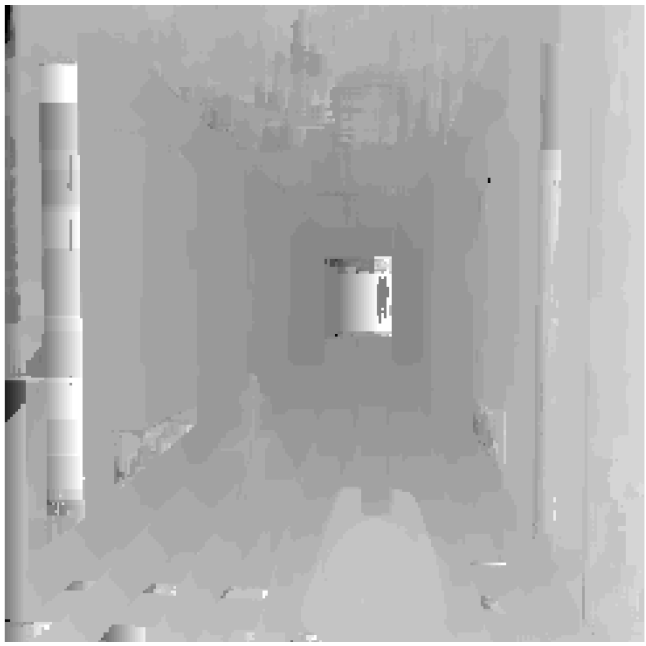


*Fig. 23: Results*

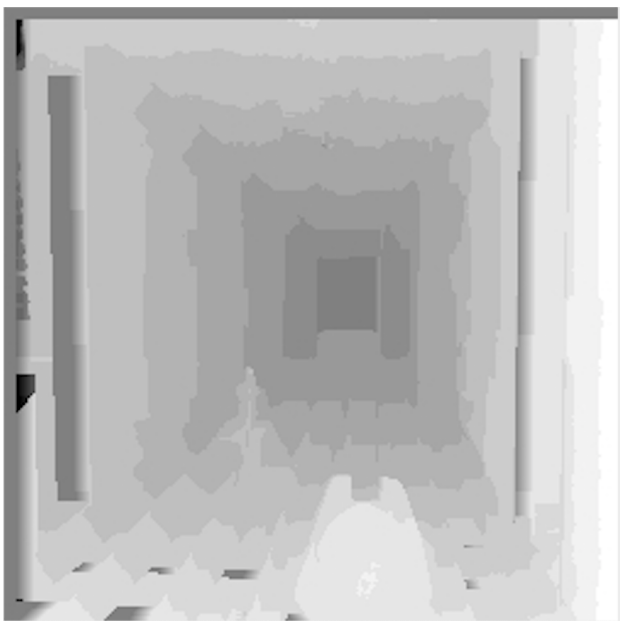


*Fig. 24: corridor_disp.jpg*

## Discussion

Fig.23 shows the result from the function disparityMap while Fig.24 shows ‘corridor_disp.jpg’. Compared to ‘corridor_disp.jpg. The result is similar to corridor_disp.jp However, we can observe that some parts are different especially the center of the disparity images are different. This is because in the original image there is a white rectangle in the center. Thus when we compute minimum sum of square (ssd), some pixels in the white rectangle may have correspondences(same ssd) with multiple pixels. Thus, as the ssd is computed from left to right and ssd is only updated if the new ssd is greater than the minimum ssd, this may result in wrong correspondence of pixel between left and right image. In addition, we only compute ssd along the scanline within 15 pixels from the target pixel. This can result in wrong correspondence if the disparity is more than 15 or is not only along the scanline.

d)

**Code**

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

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

disp = imread('triclopsid.jpg');

D = disparityMap(l, r, 11, 11);
imshow(D,[-15 15]);colormap(gray);
```

**Results**

*Fig. 25: Result*

*Fig. 26: triclopsid.jpg*

Fig. 25 shows the result from disparityMap() while Fig.26 shows triclopsid.jpg. We can see that the two figures are very similar. However, the accuracy of the disparity map is not good as compared to the corridor disparity map. The image structure has a significant impact on the accuracy of the estimated disparities. Since we use appearance-based matching to compute the disparities, if there are pixels with similar appearance (similar surrounding pixels), this can results in incorrect mapping of the correspondence pixels between the left and right image. This is evident in the noisy disparity of the walking path on the right bottom where most of the pixels are similar to each other so the pixels on the left are mapped to the wrong pixels on the right image. The actual disparity should linearly decrease as the depth increases. The disparity is also affected by the translation of the two images. If there is a translation is vertical direction, the mapping of pixels will be incorrect because we only compare the pixels along the scanline. Other image structure that can affect the performance of parity map are reflectance and occlusion.

**Optional**

Comparison between Bag Of Words and Spatial Pyramid Matching.

**Code**

```
def computeSIFT(data):
    x = []
    for i in range(len(data)):
        sift = cv2.xfeatures2d.SIFT_create()
        img = data[i]
        step_size = 24
        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

def extractSIFT(img):
    sift = cv2.xfeatures2d.SIFT_create()
    step_size = 24
```

```
    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)
    return dense_feat[1]


def getImageFeaturesSPM(L, img, clusters, 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 _ in range(2**l):
        x = 0
        for _ in range(2**l):
          desc = extractSIFT(img[y:y+h_step, x:x+w_step])
          predict = clusters.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().astype('float64')
    dev = np.std(hist)
    hist -= np.mean(hist)
    hist /= dev
    return hist

def getImageFeaturesBOW(l, img, clusters, k):
    W = img.shape[1]
    H = img.shape[0]
    h = []
    w_step = math.floor(W/(2**l))
    h_step = math.floor(H/(2**l))
    x, y = 0, 0
    for _ in range(2**l):
      x = 0
      for _ in range(2**l):
        desc = extractSIFT(img[y:y+h_step, x:x+w_step])
        predict = clusters.predict(desc)
        histo = np.bincount(predict, minlength=k).reshape(1,-1).ravel()
        h.append(histo)
        x = x + w_step
      y = y + h_step
```

```
    hist = np.array(h).ravel().astype('float64')
    dev = np.std(hist)
    hist -= np.mean(hist)
    hist /= dev
    return hist

def getHistogramSPM(L, data, clusters, k):
    x = []
    for i in range(len(data)):
        hist = getImageFeaturesSPM(L, data[i], clusters, k)
        x.append(hist)
    return np.array(x)


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

**Results**

LinearSVC (C=0.007)

| L | Bag of Words | Spatial Pyramid |
|---|---|---|
| 0 | 0.4020033388981636 | 0.4020033388981636 |
| 1 | 0.42070116861435725 | 0.4651085141903172 |
| 2 | 0.462771285475793 | 0.5225375626043406 |
| 3 | 0.4617696160267112 | 0.5499165275459098 |

SVC OneVSRestClassifier

| L | Bag of Words | Spatial Pyramid |
|---|---|---|
| 0 | 0.42237061769616024 | 0.42237061769616024 |
| 1 | 0.4604340567612688 | 0.48981636060100164 |
| 2 | 0.48146911519198665 | 0.5275459098497496 |
| 3 | 0.4651085141903172 | 0.5308848080133556 |

**Discussion**

The functions computeSIFT and extractSIFT are used to extract SIFT features from images. The functions getImageFeaturesBow and getHistogramBow are used to get the histograms of Bag of Word representations. The functions getImageFeaturesSPM and getHistogramSPM are used to get the histograms of Spatial Pyramid representations. The results above shows the accuracies of the techniques. We can observe that the Spatial Pyramid always perform better than the Bag of Words representation for L > 0. When L = 0, the Spatial Pyramid and Bag of Words representations are the same.