CZ4003 Computer Vision

School of Computer Science and Engineering

AY23/24 Semester 1

Lab 2

Lee Yew Chuan Michael
U2021372J
November 2, 2023

Contents

1 Edge Detection

a Convert to Grayscale

Input:

```
Pc = imread('macritchie.jpg');
P = rgb2gray(Pc); % convert to grayscale

imshow(P); % View this image
```

Output:

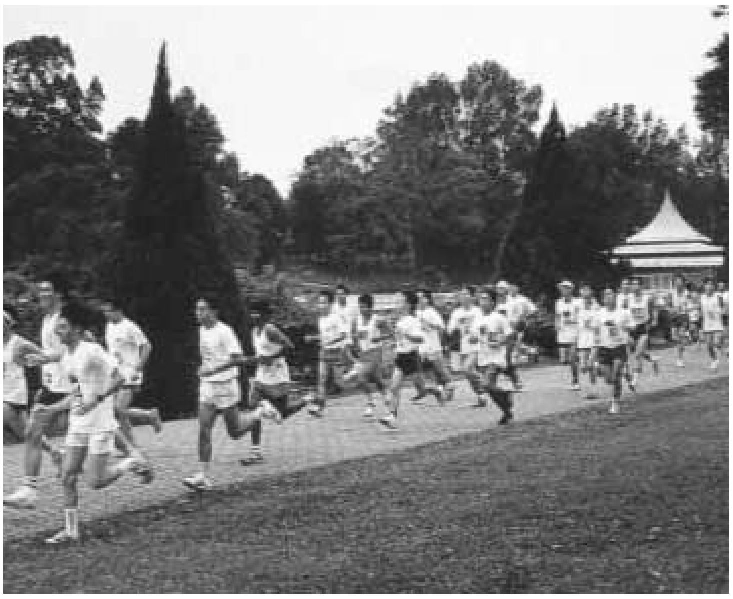

Figure 1: Grayscale image of macritchie.jpg.

b Create Sobel Masks and Filter Image

Input:

```
% Create 3x3 vertical and horizontal Sobel filters
sobel_ver = [-1 0 1;
           -2 0 2;
           -1 0 1];

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

% vertical filtered img
ver_filtered = conv2(P, sobel_ver); % apply vertical sobel
imshow(uint8(ver_filtered));

% horizontal filtered img
hor_filtered = conv2(P, sobel_hor); % apply horizontal sobel
imshow(uint8(hor_filtered));
```

Output:

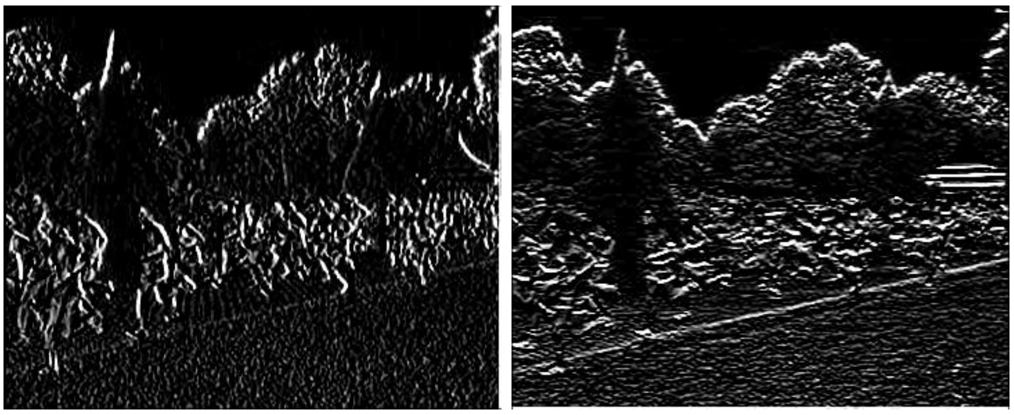

Figure 2: **Left:** Filtered with vertical filter. **Right:** Filtered with horizontal filter.

b.1 Question

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

While the vertical Sobel filter captures strictly vertical edges well, the horizontal Sobel filter captures strictly horizontal edges well. However, neither filters were able to capture diagonal edges well. While some faint diagonal edges could be captured in both resultant images, they are not as effectively captured as strictly horizontal and vertical edges. Thus, vertical and horizontal filters can only capture diagonal edges faintly, and are not able to capture diagonal edges well.

c Generate a Combined Edge Image

Input:

```
E = ver_filtered.^2 + hor_filtered.^2;
imshow(uint8(E));
```

Output:

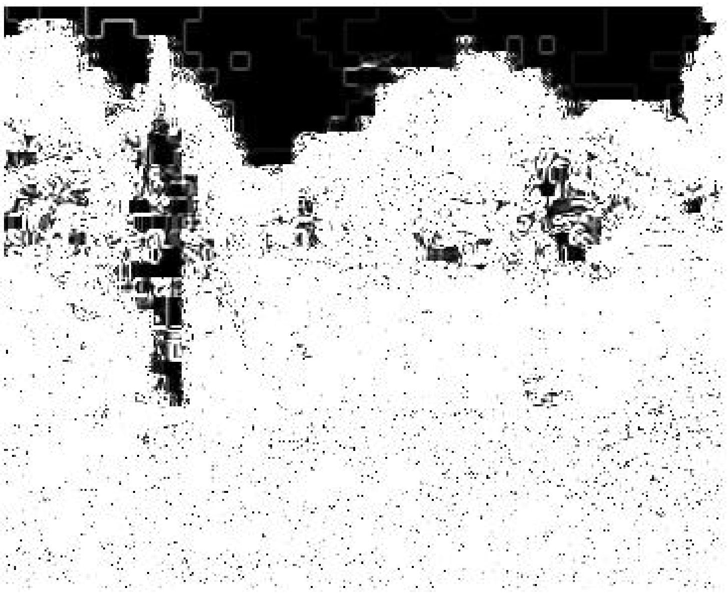

Figure 3: Combined edge image.jpg.

c.1 Question

Suggest a reason why a squaring operation is carried out.
When applying the Sobel filters to identify edges, we might encounter negative values when calculating the gradient. Hence, a squaring operation is performed to get rid of the negative gradient values. This allows us to perform the thresholding operation.

d Threshold the edge image

Input:

```
thresh_list = {100, 500, 1000, 5000, 10000, 50000, 100000, 500000};

for t = 1:length(thresh_list)
    Et = E>thresh_list{t};
    figure;
    imshow(Et);
    title('thresh'+string(thresh_list{t}));
end
```

Output:

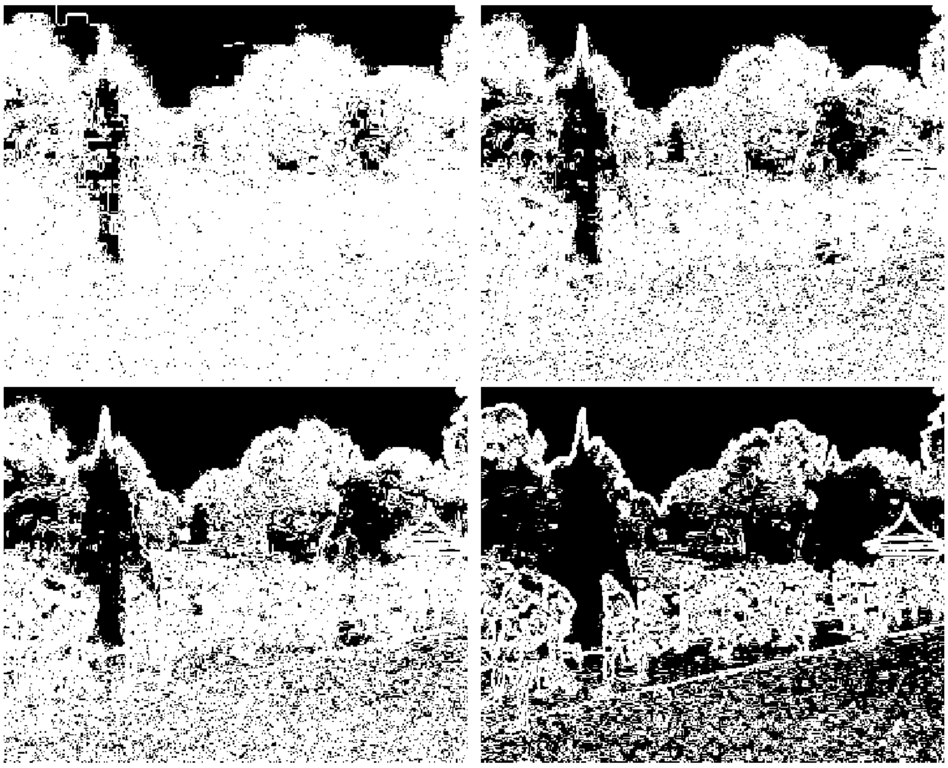

Figure 4: **Top left:** Threshold value t at 100. **Top right:** Threshold value t at 500. **Bottom left:** Threshold value t at 1000. **Bottom right:** Threshold value t at 5000.

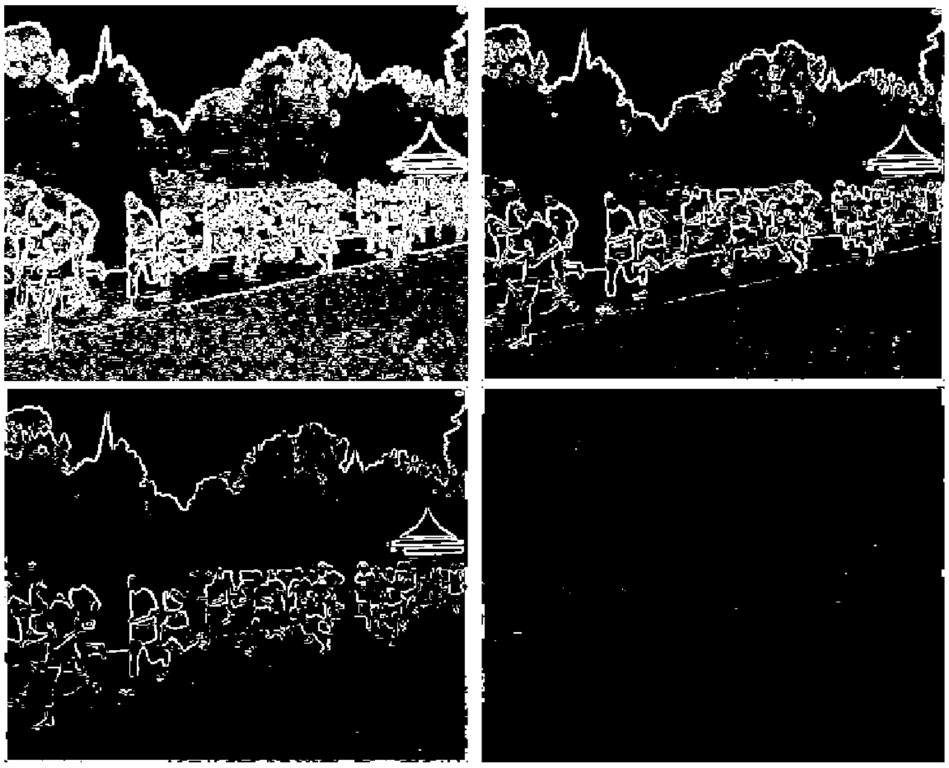

Figure 5: **Top left:** Threshold value t at 10000. **Top right:** Threshold value t at 50000. **Bottom left:** Threshold value t at 100000. **Bottom right:** Threshold value t at 500000.

d.1 Question

What are the advantages and disadvantages of using different thresholds?
The lower the threshold t is, the more noisy edges remains in the edge image. The higher the threshold t is, the less noisy edges remains in the edge image. As seen from Figure 4, when threshold values of 100 and 500 were used, a significant portion of noise in the images remained. However, as threshold value t increased from 5000 to 100000, we can see that at each interval, an increasing amount of edges are removed. An extreme case is provided in Figure 5, when threshold value t=500000 was used. When t= 500000 was used, almost no edge information was retrieved, making it rather useless.

The advantages and disadvantages of using different thresholds is that with higher thresholds, more noisy edges are removed, and we may have a higher chance of getting the relevant edges. However, setting threshold too high might result in us losing important edge information (edge details).

On the other hand, lower threshold allows one to get more edges. However, this also means getting more noisy edges (irrelevant edges), as edges with low magnitude above the threshold would also be shown.

e Canny Edge Detection

Input:

```
tl = 0.04;
th = 0.1;
sigma = 1.0;

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

Output:

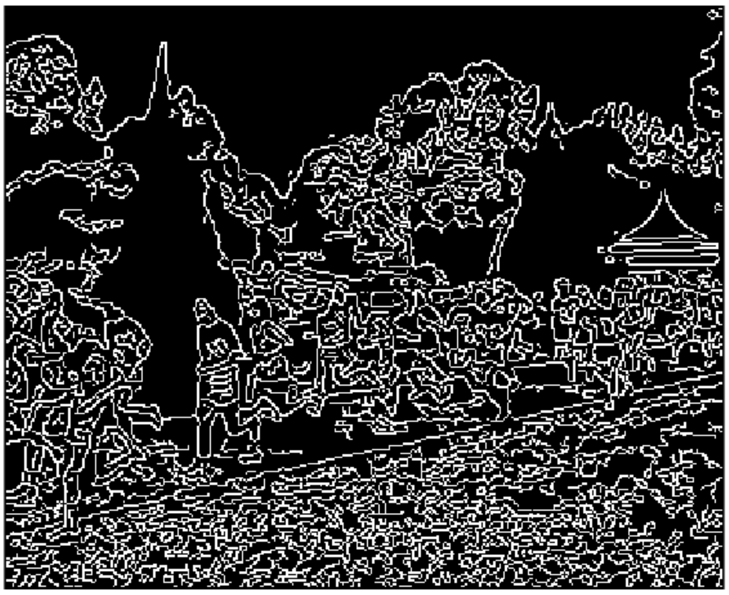

Figure 6: Canny edge detection with default settings

e.1 Varying sigma values

Input:

```
sigma_list = {1.0, 2.0, 3.0, 4.0, 5.0};
for s = 1:length(sigma_list)
    E = edge(P , 'canny', [tl, th], s);
    figure;
    imshow(E);
    title('Sigma'+string(sigma_list{s}));
end
```

Output:

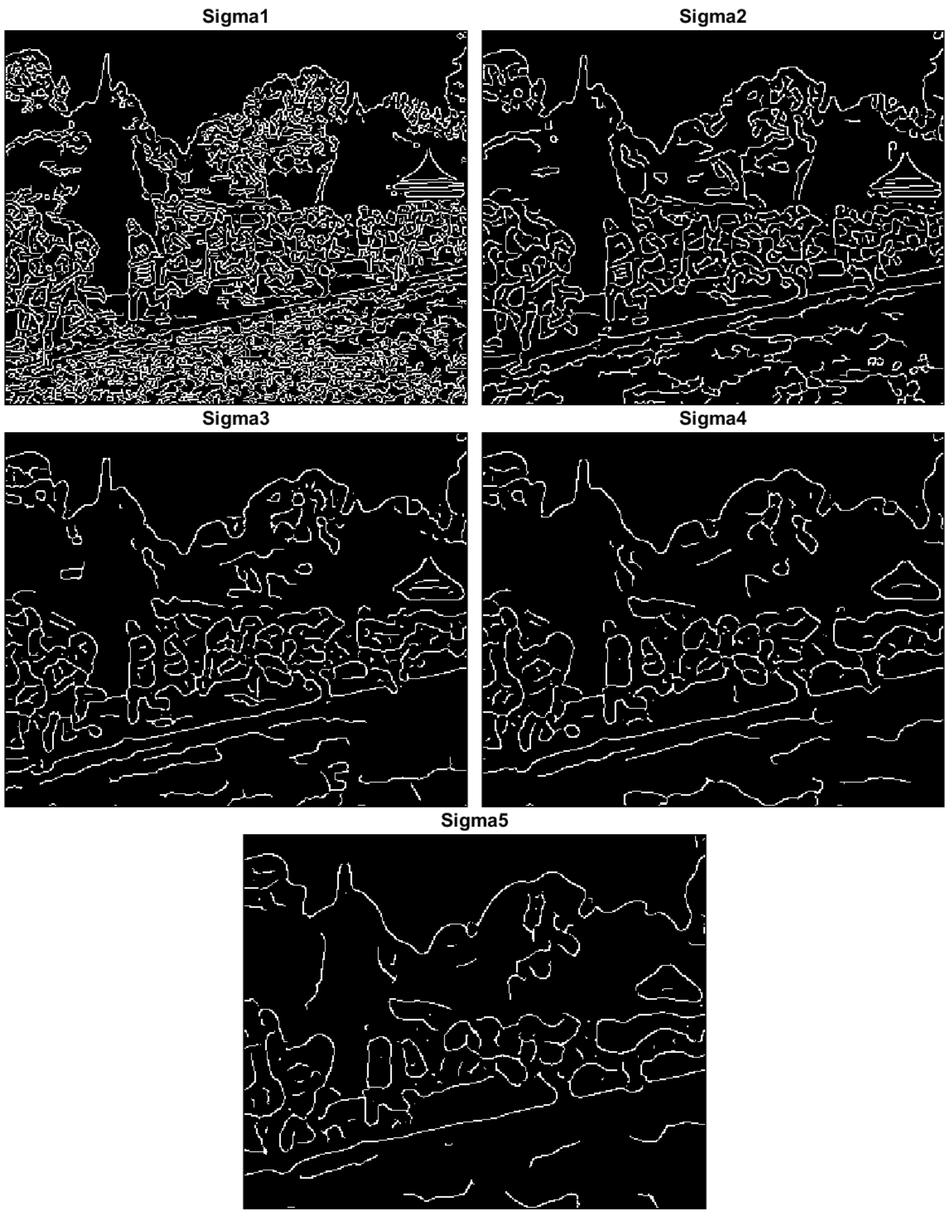

Figure 7: Varying sigma values for canny edge detection

What do you see and can you give an explanation for why this occurs?
As sigma increase from 1.0 to 5.0, more noisy edgels are removed. However, increasing Sigma also causes the location accuracy of edgels to decrease.

Sigma controls the smoothing factor of the Gaussian blur applied as part of the canny edge detection process. As sigma increases, the smoothing factor increases, causing more noise (edgels) to be removed. However, this also leads to more information lost, causing a drop in location accuracy of edgels.

Discuss how different sigma are suitable for:
(a) noisy edgel removal: Higher Sigma values are more suitable for removing noisy edgels.

(b) location accuracy of edgels: Lower Sigma values are more suitable for obtaining higher location accuracy of edgels.

e.2 Varying tl value

Input:

```
% varying tl
tl_list = {0.01, 0.02, 0.06, 0.08};
for t = 1:length(tl_list)
    E = edge(P , 'canny', [tl_list{t}, th], sigma);
    figure;
    imshow(E);
    title('tl'+string(tl_list{t}));
end
```

Output:

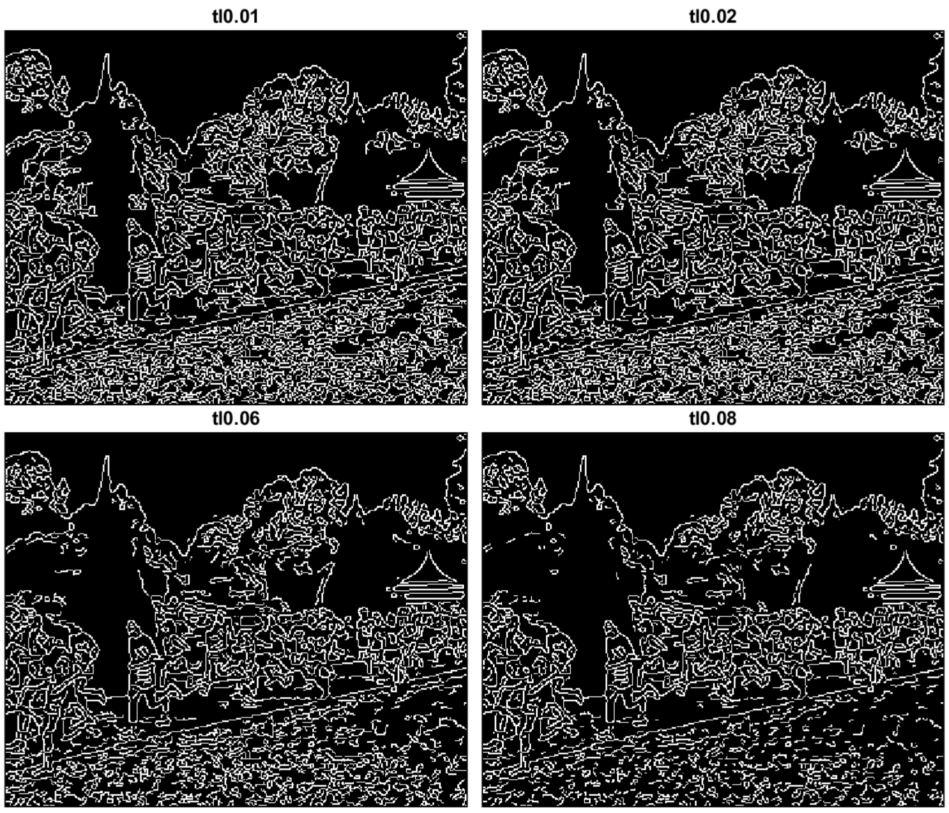

Figure 8: Varying tl values for canny edge detection

Try raising and lowering the value of tl. What does this do? How does this relate to your knowledge of the Canny algorithm?
Upon lowering tl below the default value of 0.04, more edgels were observed. In other words, the image became more noisy. On the other hand, when tl was increased above 0.04, less edgels were observed, the image became less noisy.

In the Canny algorithm, the variable tl refers to the lower bound threshold used during hysteresis thresholding. Pixels with gradient magnitude above tl and below th which has neighbouring pixels perpendicular to the edge gradient which have already been set as an edgel will be set as an edgel too. On the other hand, when the pixel gradient magnitude is lower than tl, they would be filtered out and not considered as edgels. Thus, the higher tl is, the more edgels get filtered out and the less noisy the image is.

2 Line Finding using Hough Transform

a Reuse the Edge Image with Sigma Set To 1.0

Input:

```
tl = 0.04;
th = 0.1;
sigma = 1.0;
Pc = imread('macritchie.jpg');
P = rgb2gray(Pc); % convert to grayscale
E = edge(P , 'canny', [tl, th], sigma);
imshow(E);
```

Output:

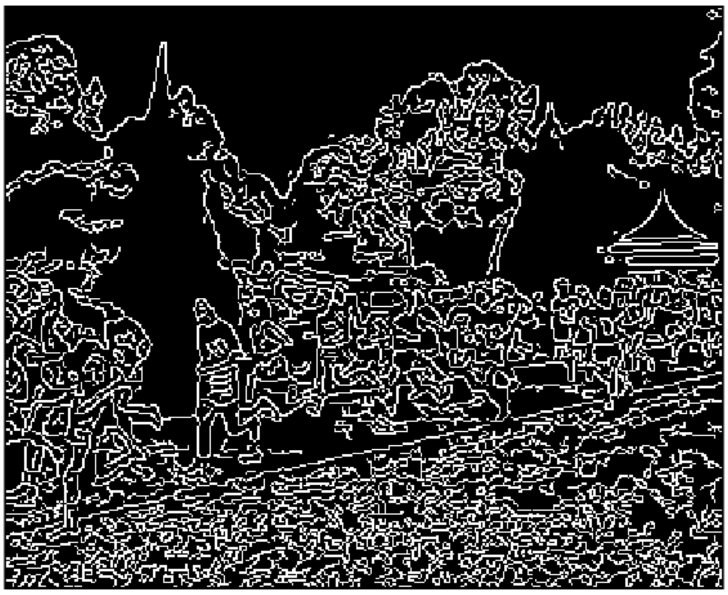

Figure 9: Canny edge detection with sigma=1.0

b Radon Transform

b.1 Question

Explain why the Radon transform and Hough transform are equivalent in the case

From the manual, Radon transform performs projections of image intensities on a radial line that is oriented at a specific angle. Radon transform is the integral transform defined for continuous functions on R(n) on hyperplanes in R(n). On the other hand, Hough transform is a discrete algorithm that can detect lines in an image via a voting procedure.

In this case, the Radon transform performed ends up being discrete, as a binary image is used. As such, both Radon and Hough transform will both behave similarly and end up mapping each pixel in the image into the same sinusoidal curve in the theta, rho space. Thus, becoming equivalent and resulting in the same outcome.

When are they different?

They are different when we use non-binary images. This refers to images whose pixels values lie between [0,255]. This is opposed to binary images whose pixel values are either 1s or 0s. As Radon transform uses pixel intensities in its calculations, we would get different results compared to when Hough transform is used on non-binary images. Additionally, Radon transform tends to be more accurate compared to Hough transform. We tend to see less artifacts when Radon transform is used.

b.2 Perform Radon Transform and Display H as an Image

Input:

```
[H, xp] = radon(E);
figure;
imagesc(uint8(H));
colormap(gca,hot);
```

Output:

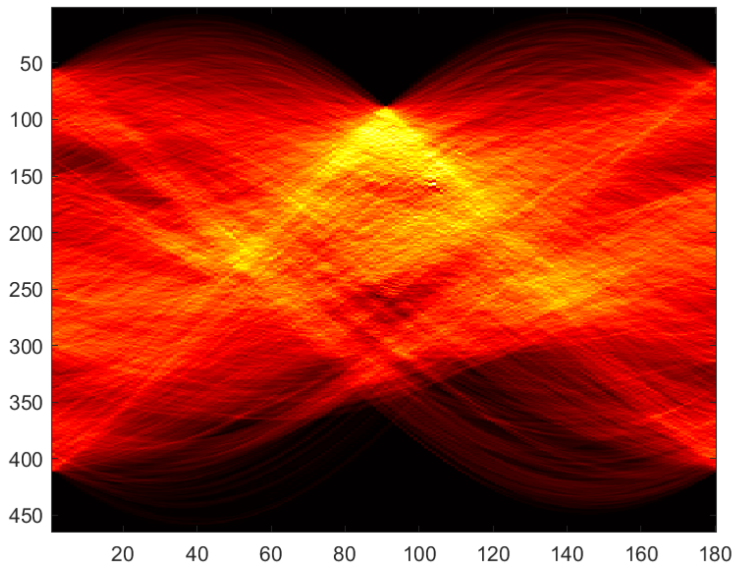

Figure 10: Radon transform

c Find the location of the maximum pixel intensity in the Hough image

Input:

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

Output:

```
radius =

   157

theta =

   104
```

d Derive the equations

Input:

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

[numRows, numCols] = size(P);
x_center_c = numCols/2;
y_center_c = numRows/2;

C = A*(A+x_center_c) + B*(B+y_center_c);
```

Output:

```
C =

   1.9760e+04
```

e Compute yl and yr Values

Input:

```
[numRows, numCols] = size(P);
xl = 0;
xr = numCols - 1;

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

Output:

```
yl =

  267.9563

yr =

  178.9463
```

f Display the original image and superimpose your estimated line

Input:

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

Output:

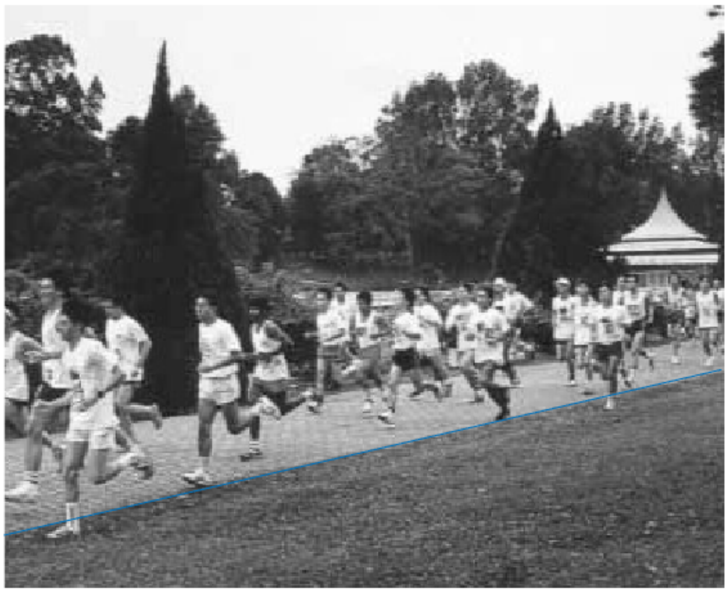

Figure 11: Identified Edge of the Running Path

f.1 Question

Does the line match up with the edge of the running path?
At first glance, the line obtained almost matches up with the edge of the running path perfectly. However, upon closer analysis we can observe that this line does not align with the path perfectly. Parts of the line deviates from the edge of the running path. A zoomed in view of this can be seen in Figure 12.

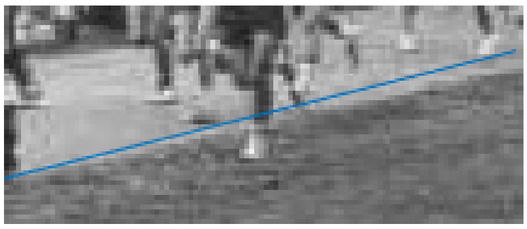

Figure 12: Zoomed in Edge of the Running Path

What are, if any, sources of errors?

1. The edge of the running path may not be perfectly straight. There might have been some curvature to it. As such, when the equation for straight edges is applied to find the edge of the running track, it cannot be identified perfectly.
2. Inaccuracies may have arose when performing Radon transform. Radon is typically used for continuous functions. However, we used it as a discrete function in our experiments. This might have led to a drop in precision.
3. Noisy edgels present in the image may have resulted in the inaccurate representation of the edge of the running path.

Can you suggest ways of improving the estimation?

1. Use a non-linear function to perform the edge detection rather than the linear one used previously. This might enable us to discover the curved edge of the running path.
2. A larger sigma value when applying the Canny edge detection algorithm could have been used. This could have allowed us to remove more noisy edgels in the image. This could in turn lead to a more accurate identification of the running path edge.

3 3D Stereo

a Write the Disparity Map Algorithm

Input:

```
function map = disparity_map(imgl, imgr, dim1, dim2)

[height, width] = size(imgl);
max_range = 14;
row = floor(dim1/2);
col = floor(dim2/2);

map = ones(size(imgl));

for h = 1+row:height
    min_row = max(1, h - row);
    max_row = min(height, h + row);

    for c = 1+col:width
        min_col = max(1, c - col);
        max_col = min(width, c + col);

        I = imgl(min_row:max_row, min_col:max_col);

        min_ssd = inf;
        min_diff = 0;
        lower_bound = max(-max_range, 1 - min_col);
        upper_bound = min(max_range, width - max_col);
        for k = lower_bound:upper_bound
            T = imgr(min_row:max_row, min_col + k:max_col + k);

            ssd = (I - T).^2;
            ssd = sum(ssd(:));
            if ssd < min_ssd
                min_ssd = ssd;
                min_diff = k + c;
            end
        end
        map(h, c) = min_diff - c;
    end
end
```

b Download the Synthetic Stereo Pair Images and Convert Both to Grayscale

Input:

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

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

Output:

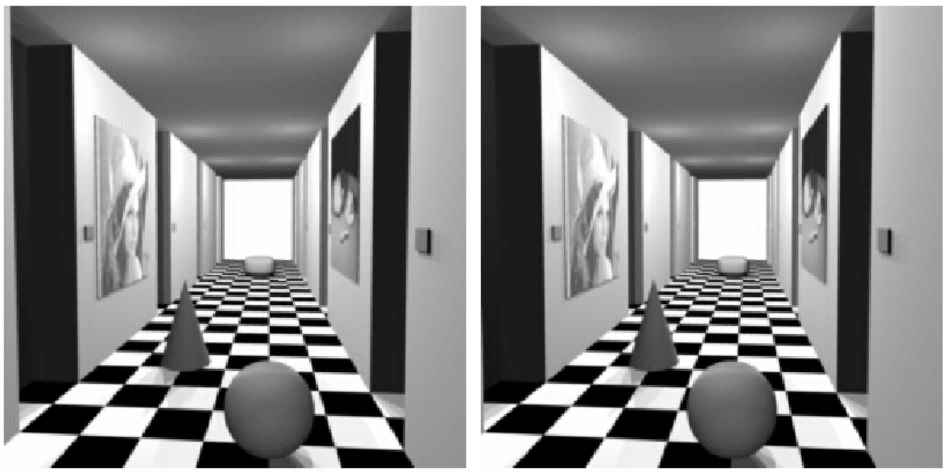

Figure 13: **Left:** Grayscale corridorl.jpg **Right:** Grayscale corridorr.jpg

c Run your algorithm on the two images to obtain a disparity map D

Input:

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

Output:

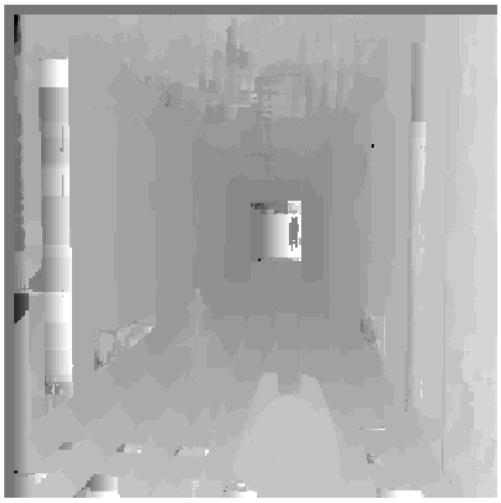

Figure 14: Computed Disparty map of corridor

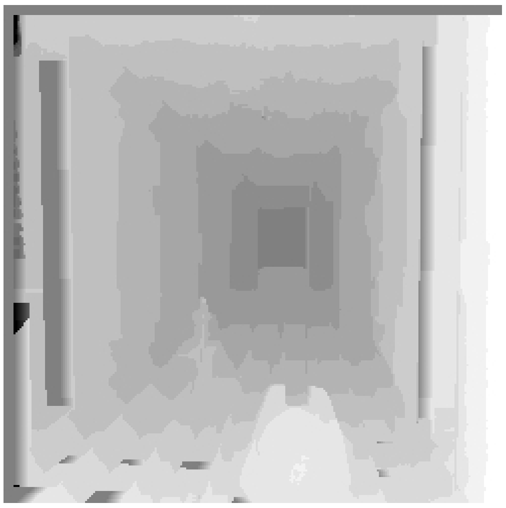

Figure 15: corridor_disp.jpg for reference

c.1 Question

Comment on how the quality of the disparities computed varies with the corresponding local image structure
As seen from Figure 13, while both images do capture the same scene, ie, the corridor, the right image has been taken at an angle. As such, the points in the left and right images are displaced with respect to each other. In general, closer points have higher displacement, indicating high disparity. On the other hand, points which are further away have lower displacement, indicating low disparity. When reflected on a disparity map, this means closer points would appear brighter. Further points on the other hand would appear darker.

This phenomenon is largely observed in our computed disparity map in Figure 14. At the start of the corridor, which represents points which are closer, the disparity map is bright. This is attributed to the fact that the displacement due to the images at and angle from each other was more significant. Thus, leading to higher disparity and brighter regions. However, as we moved further down the corridor, displacement was less significant. Thus, leading to lower disparity and darker regions.

There is however an anomaly to this trend. As seen from Figure 14, a bright spot is observed at the end of the corridor. Indicating that this was a region of high disparity. This differs from the reference image provided in Figure 15 which expected the end of the corridor to be a dark region, reflecting low disparity.

Such an inaccuracy in our computed disparity map may have arose due to the use of Sum-of-Squares Difference (SSD) as the measure of disparity.

d Rerun your algorithm on the real images of triclops

Input:

```
Tl = imread("triclopsi2l.jpg");
Tl = rgb2gray(Tl);
figure
imshow(Tl);

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

D_triclops = disparity_map(double(Tl), double(Tr), 11, 11);
figure
imshow(-D_triclops,[-15 15]);
```

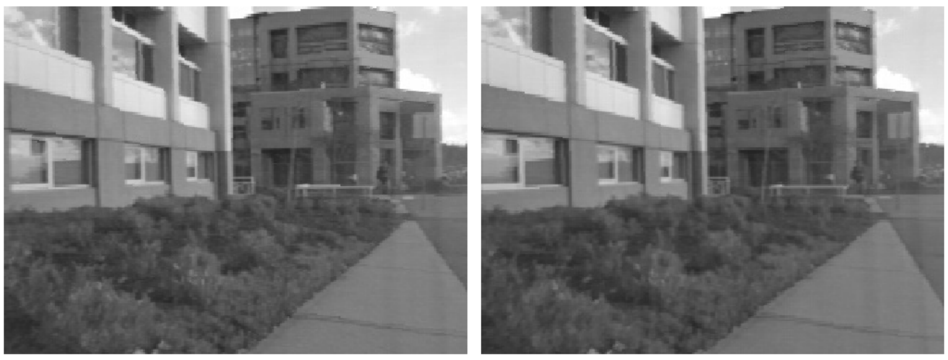

Figure 16: **Left:** triclopsi2l.jpg **Right:** triclopsi2r.jpg

Output:

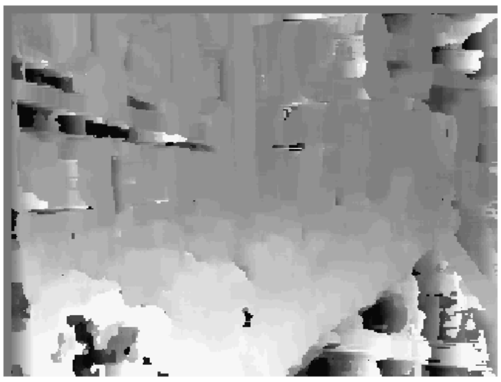

Figure 17: Computed Disparty map of triclops

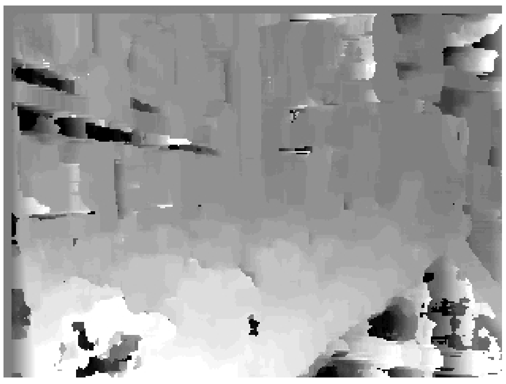

Figure 18: triclopsid.jpg for reference

d.1 Question

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

As seen from Figure 17, the computed disparity map is quite similar to the reference map. However, such a disparity map is not very desirable, inconsistencies are observed in some regions of the disparity map. In particular, the regions in the computed disparity map corresponding to the pathway is rather irregular and no clear patterns can be observed. This is in contrast to the bushes beside the pathway where we can observe that for bushes which are closer, they correspond to brighter regions in the disparity map. On the other hand, bushes which are further correspond to darker regions. A similar pattern should have been exhibited for the regions corresponding to the pathway.

The inconsistency in the calculation of disparity for the pathway can be attributed to the homogeneity of the pathway surface and the use of SSD to calculate the disparity. SSD is a Point matching-Appearance based method for computing disparity. When it is used to calculate the disparity in regions of the image with homogeneous surfaces that have pixels of similar intensities (such as the pathway), SSD will find it difficult to find the corresponding points with the global minimum. As such, the quality of the regions in the computed disparity map corresponding to homogeneous regions such as the pathway would be poor.