CZ4003 Computer Vision

School of Computer Science and Engineering

AY23/24 Semester 1

Lab 2

Lee Yew Chuan Michael
U2021372J
October 24, 2023

Contents

1 Edge Detection **2**

a Convert to Grayscale . . . 2

b Create Sobel Masks and Filter Image . . . 3

b.1 Question . . . 3

c Generate a Combined Edge Image . . . 4

c.1 Question . . . 4

d Threshold the edge image . . . 5

d.1 Question . . . 6

e Canny Edge Detection . . . 7

e.1 Varying sigma values . . . 7

e.2 Varying tl value . . . 9

2 Line Finding using Hough Transform **11**

a Reuse the Edge Image with Sigma Set To 1.0 . . . 11

b Radon Transform . . . 12

b.1 Question . . . 12

b.2 Perform Radon Transform and Display H as an Image 12

c Find the location of the maximum pixel intensity in the Hough image . . . 13

d Derive the equations . . . 14

e Compute yl and yr Values . . . 15

f Display the original image and superimpose your estimated line 16

f.1 Question . . . 16

3 3D Stereo **18**

a Write the Disparity Map Algorithm . . . 18

b Download the Synthetic Stereo Pair Images and Convert Both to Grayscale . . . 19

c Run your algorithm on the two images to obtain a disparity map D . . . 20

c.1 Question . . . 21

d Rerun your algorithm on the real images of triclops . . . 22

d.1 Question . . . 23

4 Optional **23**

1 Edge Detection

a Convert to Grayscale

Input:

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

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

Output:

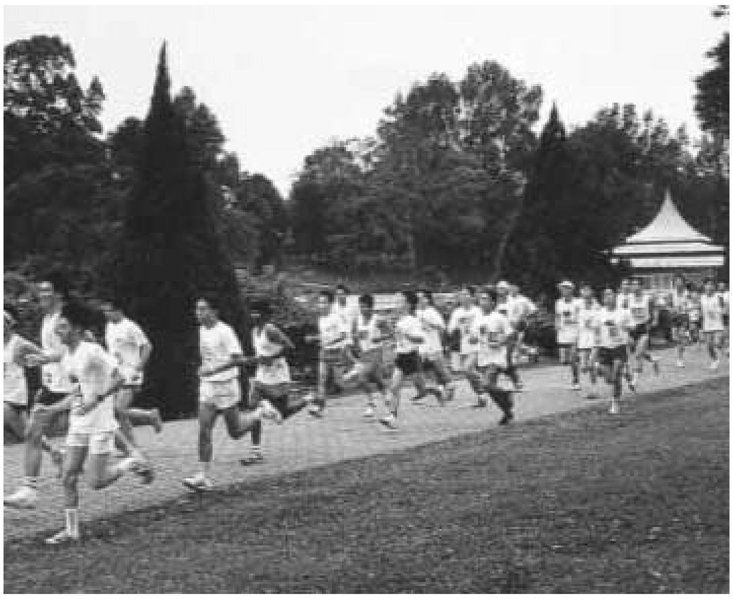

Figure 1: Grayscale macritchie.jpg.

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:

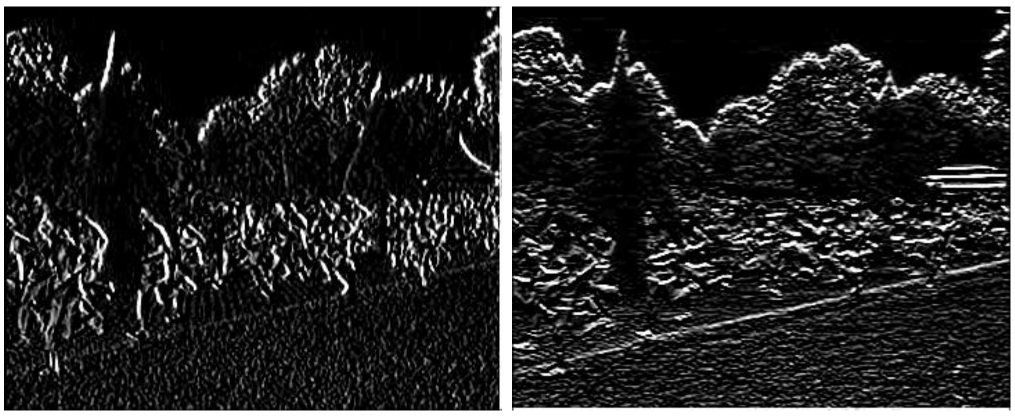

Figure 2: **Left:** Filtered with vertical filter. **Right:** Filtered with horizontal filter.

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 observed in both resultant images, they are not as effectively captured as strictly horizontal and vertical edges.

c Generate a Combined Edge Image

Input:

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

Output:

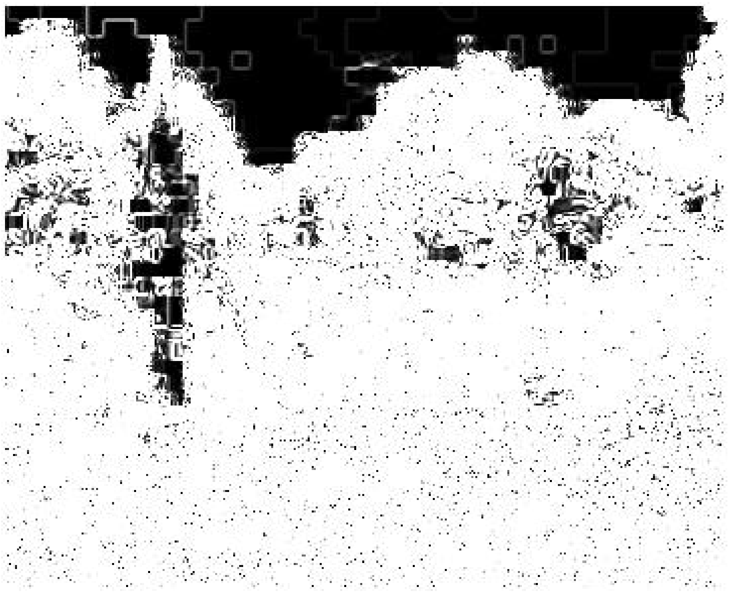

Figure 3: Combined edge image.jpg.

c.1 Question

Suggest a reason why a squaring operation is carried out.
During the calculation of the gradient, which is performed when the Sobel filters are applied, there might be negative calculations. Hence, a squaring operation is performed to get rid of the negative values to get the magnitude.

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:

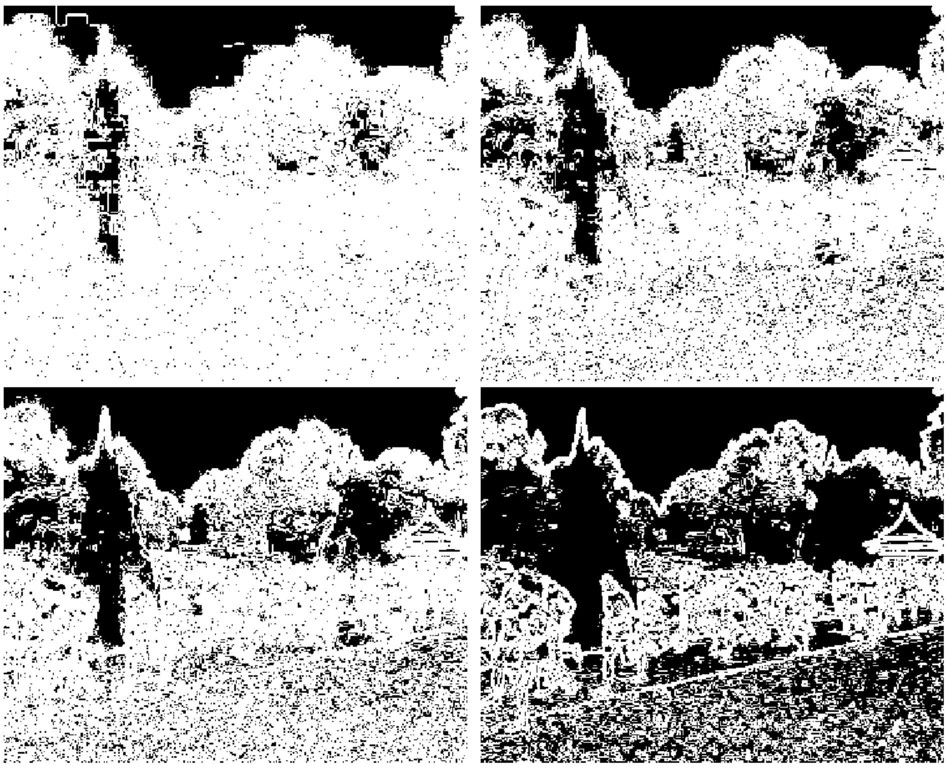

Figure 4: **Top left:** Threshold value t at 100. **Top right:** Threshold value t at 500. **Bottom left:** Threshold value t at 1000. **Bottom right:** Threshold value t at 5000.

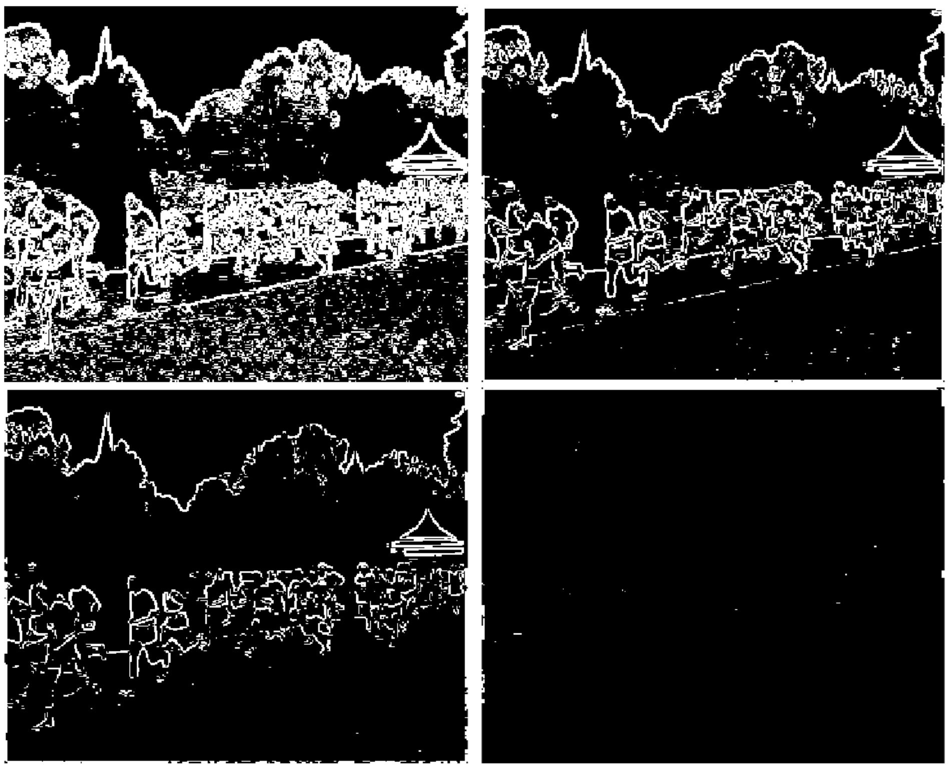

Figure 5: **Top left:** Threshold value t at 10000. **Top right:** Threshold value t at 50000. **Bottom left:** Threshold value t at 100000. **Bottom right:** Threshold value t at 500000.

d.1 Question

What are the advantages and disadvantages of using different thresholds?
The lower the threshold t is, the more noise remains in the 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 increases, more edges are removed. An extreme case is provided in Figure 5 when threshold value t=500000 was used.

The advantages and disadvantages of using different thresholds is that with higher thresholds, more noise is removed, and we may have a higher chance of getting the relevant edges. On the other hand, lower threshold allows one to get more edges. However, this also means getting more noise (irrelevant edges), as edges with low magnitude above the threshold would 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:

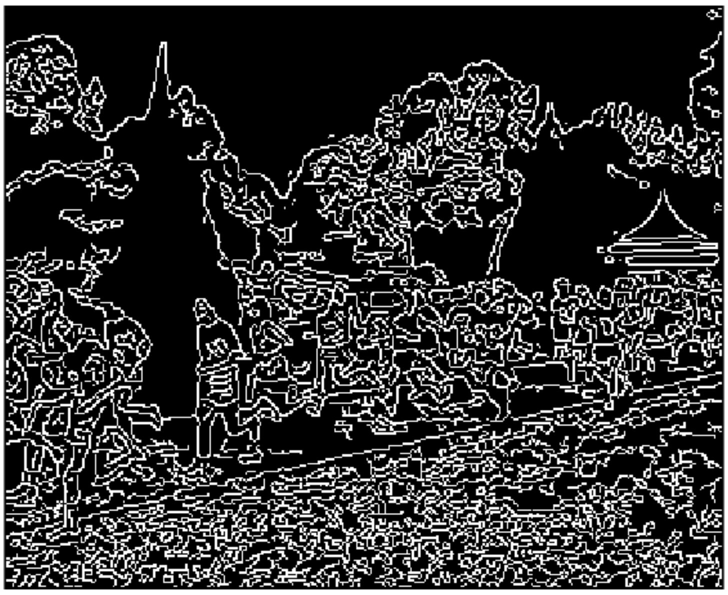

Figure 6: Canny edge detection with default settings

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:

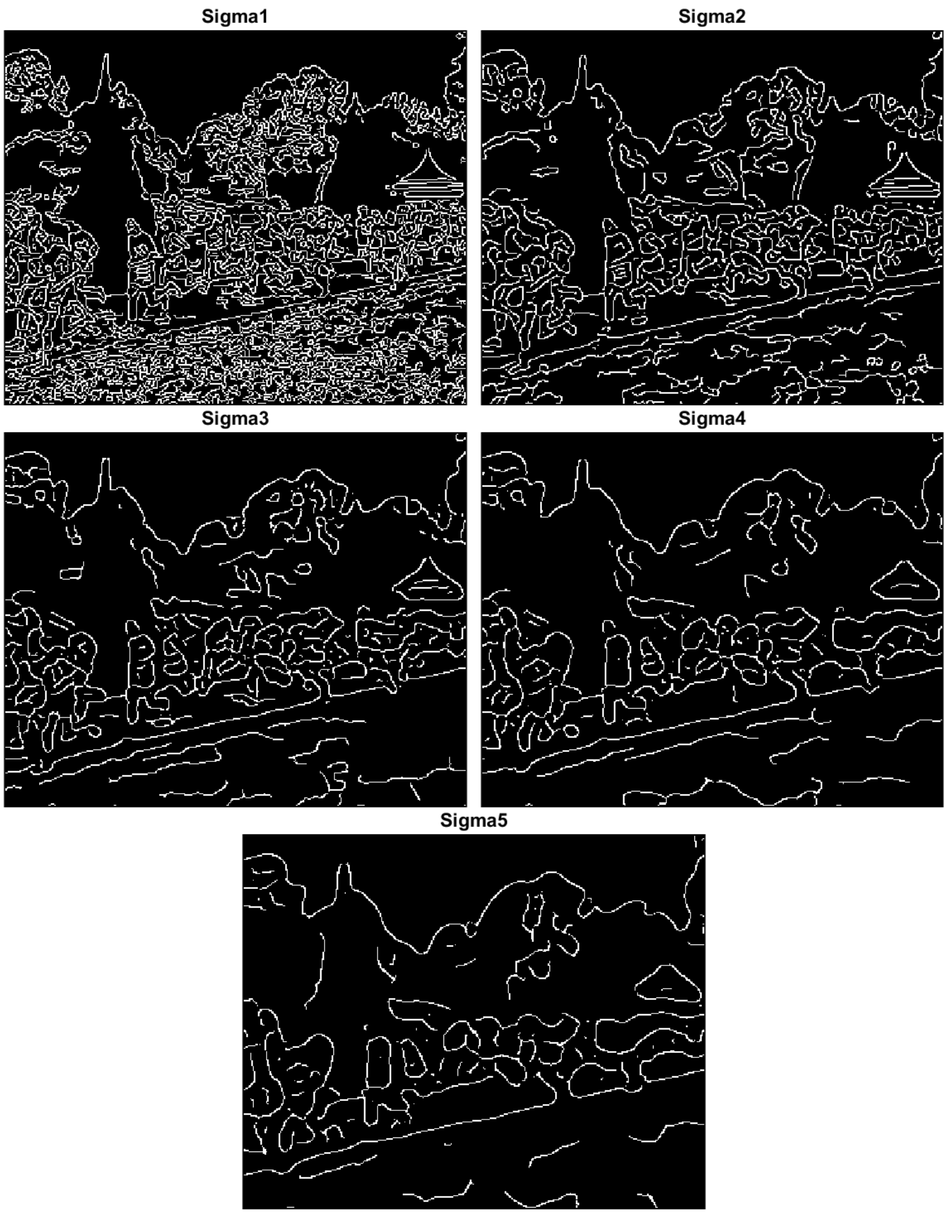

Figure 7: Varying sigma values for canny edge detection

What do you see and can you give an explanation for why this occurs?
As sigma increase from 1.0 to 5.0, more edgels are removed but location accuracy of edges also 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: To remove more noisy edgels, increase sigma.

(b) location accuracy of edgels: To improve location accuracy of edgels, decrease sigma.

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:

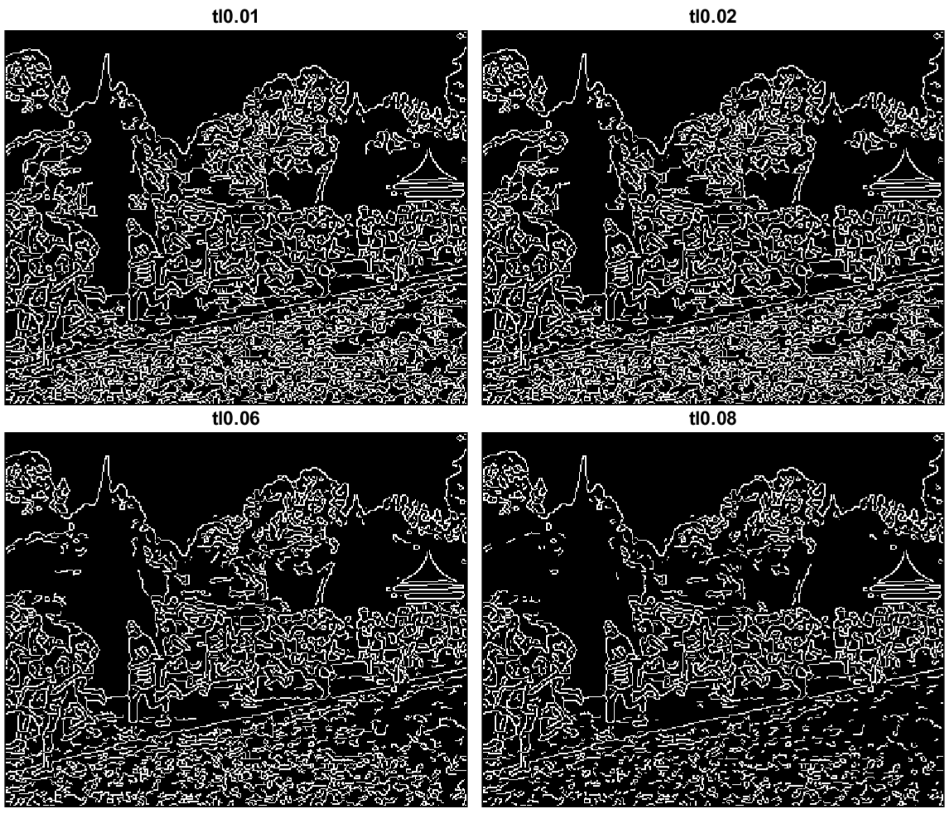

Figure 8: Varying tl values for canny edge detection

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 abocve 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. Edgels with magnitude above tl would be displayed, while those with magnitude lower than tl would be filtered out. 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:

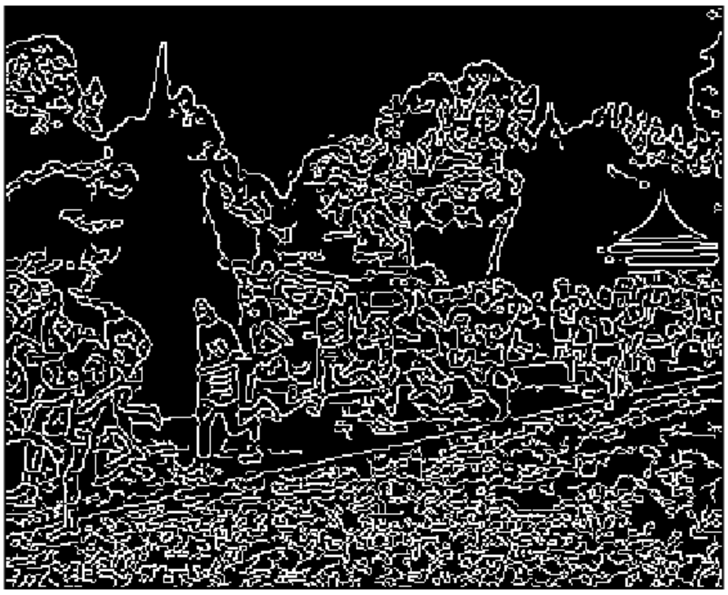

Figure 9: Canny edge detection with sigma=1.0

b Radon Transform

b.1 Question

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

Radon transform is the integral transform defined for continuous functions on R(n) on hyperplanes in R(n). On the other hand, the 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 map each pixel into the same sinusoidal function. Thus, resulting in the same outcome.

When are they different?

They are different when continuous Radon transform is performed instead. Continuous Radon transform tends to be more accurate compared to Hough transform. Less artifacts tend to be observed 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);
axis off;
```

Output:

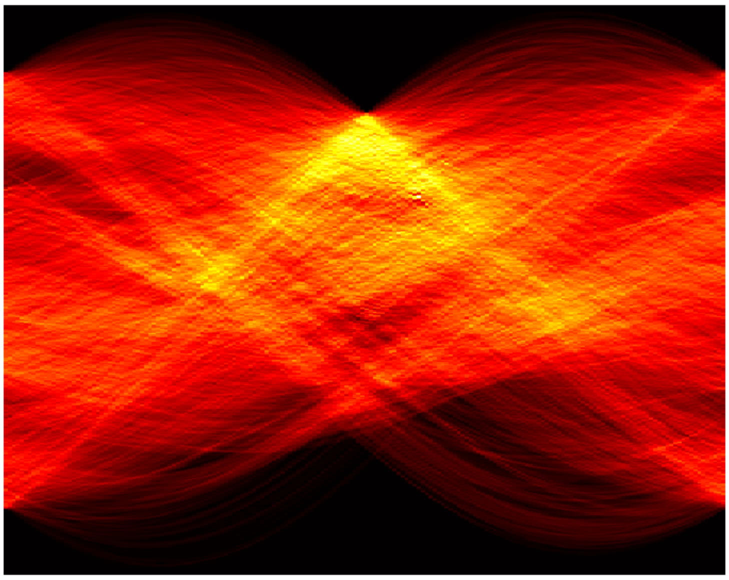
Figure 10: Radon transform

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:

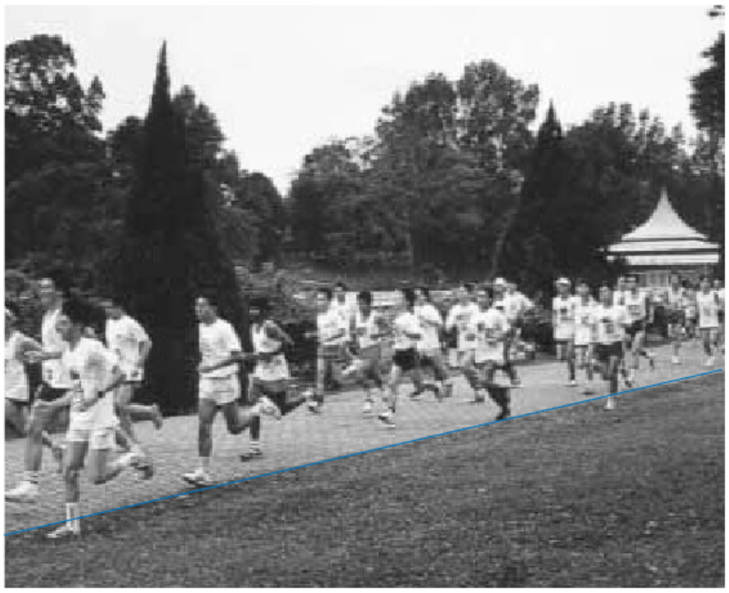

Figure 11: Identified Edge of the Running Path

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 vuew of this can be seen in Figure 12.

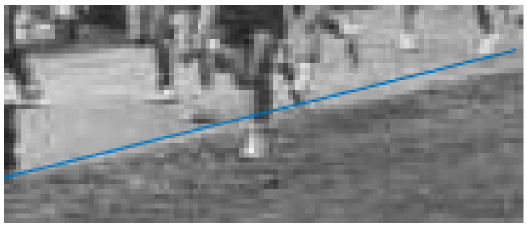
Figure 12: Zoomed in Edge of the Running Path

What are, if any, sources of errors?

1. The running track edge 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 arisen when performing Radon transform.
3. Noise present in the image might have interfered with the identification of high intensity pixels.

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 could have been used. This could have allowed us to remove more noise from the image, potentially allowing us to identify the peaks more accurately. 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)

imgl = double(imgl); % Convert imgs to double
imgr = double(imgr);

[height, width] = size(imgl);
max_range = 15;
row = floor(dim1/2);
col = floor(dim2/2);

map = ones(size(imgl));

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

    for j = 1+col:width
        min_col = max(1, j - col);
        max_col = min(width, j + 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
            min_col_ = min_col + k;
            max_col_ = max_col + k;

            T = imgr(min_row:max_row, min_col_:max_col_);

            ssd = (I - T).^2;
            ssd = sum(ssd(:));
            if ssd < min_ssd
                min_ssd = ssd;
                min_diff = k - lower_bound + 1;
            end
        end
        map(i, j) = min_diff + lower_bound - 1;
    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:

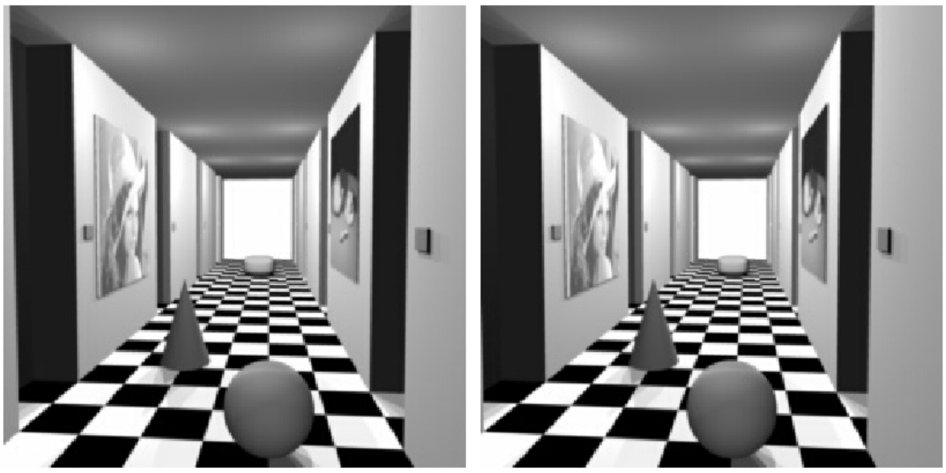

Figure 13: **Left:** Grayscale corridorl.jpg **Right:** Grayscale corridorr.jpg

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

Input:

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

Output:

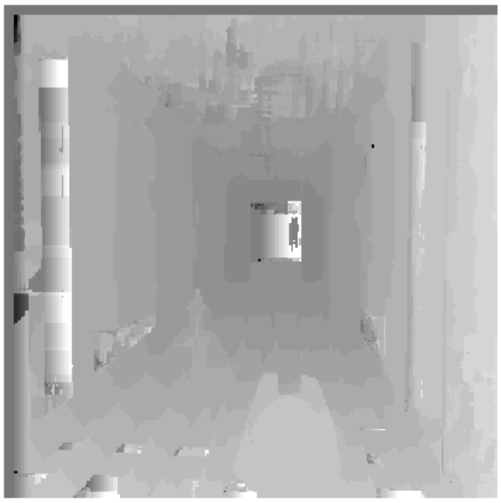

Figure 14: Computed Disparty map of corridor

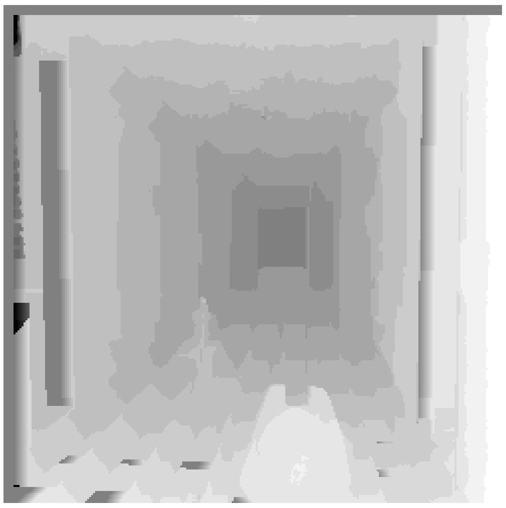

Figure 15: corridor_disp.jpg for reference

c.1 Question

Comment on how the quality of the disparities computed varies with the corresponding local image structure

As seen in Figure 14, the computed disparity map does largely capture the trends in the image. Nearer points are brighter while further parts are darker. However, noticeable inaccuracies arise in areas where the images have high similarities. For example, as the wall at the end of the corridor are near identical matches in both images, the disparity is not captured well in the computed disparity map. While we should have expected to see a dark square near the centre of the image representing that wall, we instead see a discoloured square that has some portions bright and dark.

d Rerun your algorithm on the real images of triclops

Input:

```
Tl = imread("triclopsi2l.jpg");
Tl = rgb2gray(Tl);
Tr = imread("triclopsi2r.jpg");
Tr = rgb2gray(Tr);

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

Output:

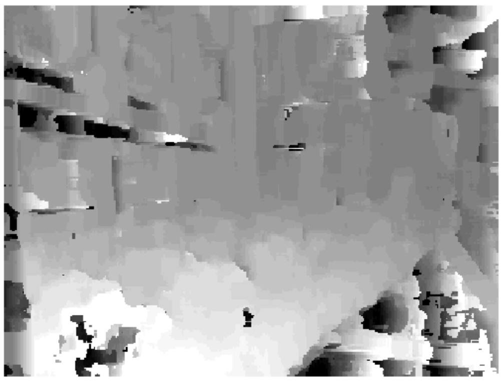

Figure 16: Computed Disparty map of triclops

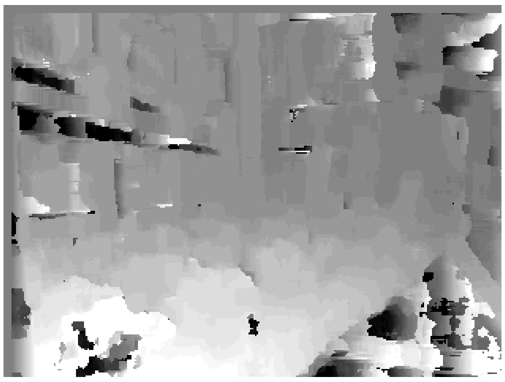

Figure 17: triclopsid.jpg for reference

d.1 Question

How does the image structure of the stereo images affect the accuracy of the estimated disparities?
Once again, we can see that in areas where the images are significantly different, the estimated disparity computed is more accurate. On the other hand, in areas of high similarity, the estimated disparity computed loses accuracy.

4 Optional