CZ4003 Computer Vision, Lab 2 Report

Lee Zhen Wei U1922813B

3.1 Edge Detection

3.1a)

Imported macritchie.jpg and converted to grayscale, displayed with imshow()

3.1b)

Created 3x3 horizontal and vertical Sobel masks

```
sobelMask1 = [-1,0,1;-2,0,2;-1,0,1]; %x kernel
sobelMask2 = [-1,-2,-1;0,0,0;1,2,1]; %y kernel
```

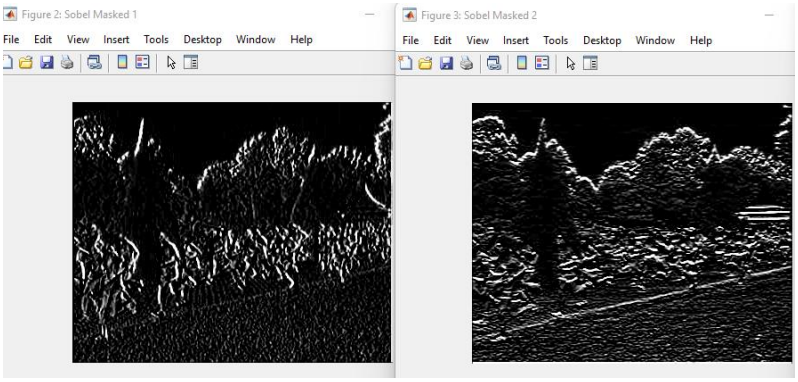
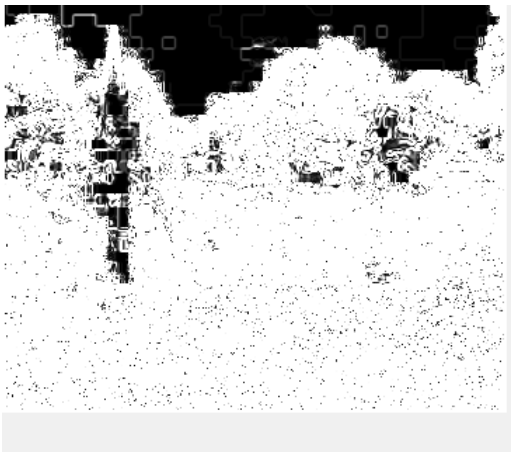


Image filtered with sobelMask1 can be observed to have preserved edges that are vertical, likewise the image filtered with sobelMask2 preserves edges that are perceived as horizontal.
These horizontal and vertical kernels are used to perform a convolution(weighted sum of neighbouring pixels) to find horizontal or vertical edges specifically by calculated the weighted sum of its neighbouring 3x3 pixels to get the output, this allows the horizontal kernel to subtract the brightness of the pixels neighbouring the right side with the brightness of neighbouring pixels on the left to detect the gradient. To detect vertical gradient, we then perform a simple 90 degree rotation to the kernel to detect vertical gradient, by finding gradients by comparing the top and bottom neighbouring pixels.

Edges which are not strictly vertical nor diagonal are not consistently detected as their individual pixel values are still calculated based on which 3x3 kernel is being used, and are highly dependent on their independent local weighted sum of pixels, and because these kernels are not tuned to pick-up

diagonals, it is a hit or miss, as seen by the diagonal pavement edges, where the horizontal kernel is unable to detect the pavement edge while the vertical kernel is able to detect the pavement edge.

3.1c)



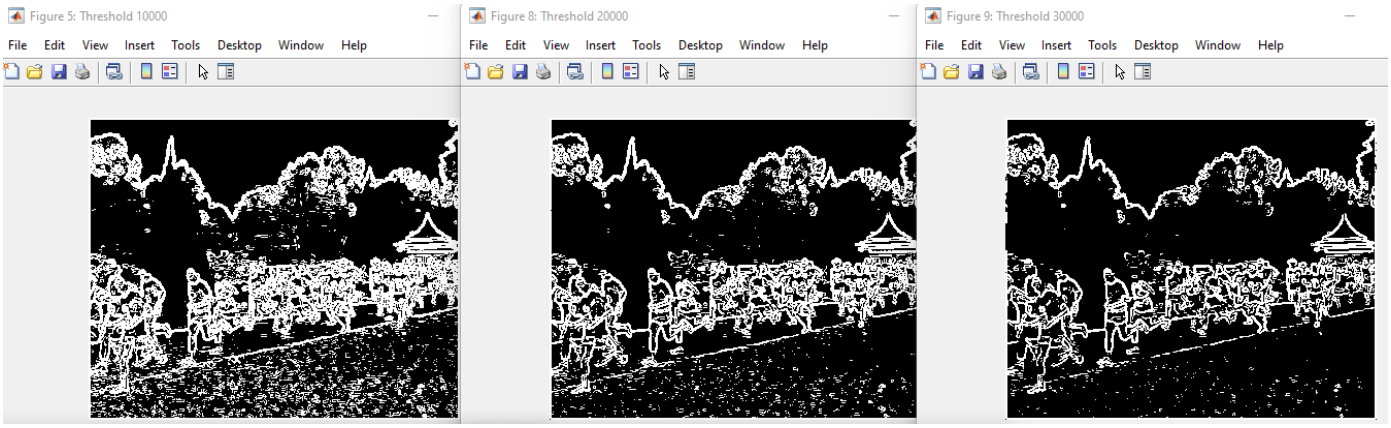
```
squaredSobelImage =sobelImageMask1.^2 + sobelImageMask2.^2;
figure('Name','Sobel after squaring')
imshow(uint8(squaredSobelImage));
```

The output images after both kernels are squared then summed together, it has resulted in an amplification of all pixel values, and when summed together, results in the image appearing to be almost entirely lighted up as all pixel values have been amplified, a thresholding can then be performed to obtain a better result based on this image. I observe that this squaring is done to obtain the magnitude of a line and remove negative values. i.e for pixel 1,1, horizontal filter = (-2) vertical filter = (3) we want to obtain the magnitude , hence square to remove negative, then sum to obtain magnitude. (-2)^2 + 3^2 = 4+9 = 13, unfortunately this step also amplifies non-consequential pixel values, so we then perform a thresholding to filter out the edges that we want to keep.

3.1d)

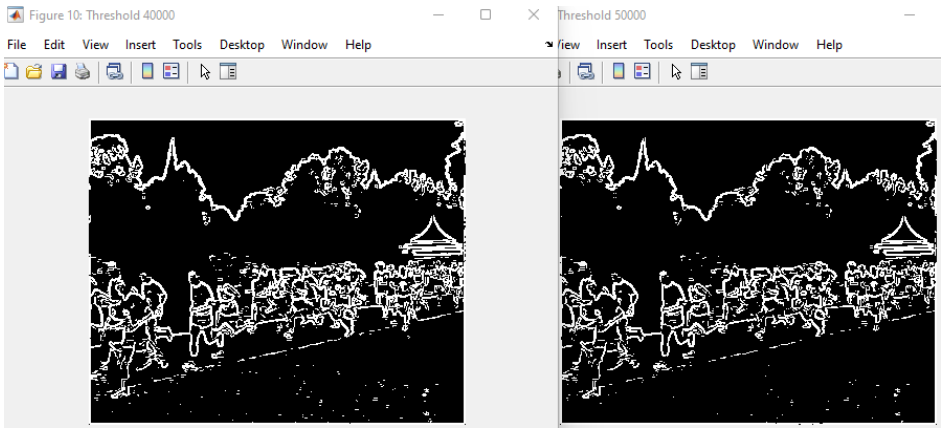
```
t = 10000;
squaredSobelImageThresholded = squaredSobelImage>t;
imshow(squaredSobelImageThresholded)
```

Pixels are filtered with a simple thresholding value to display images with different binary edges.

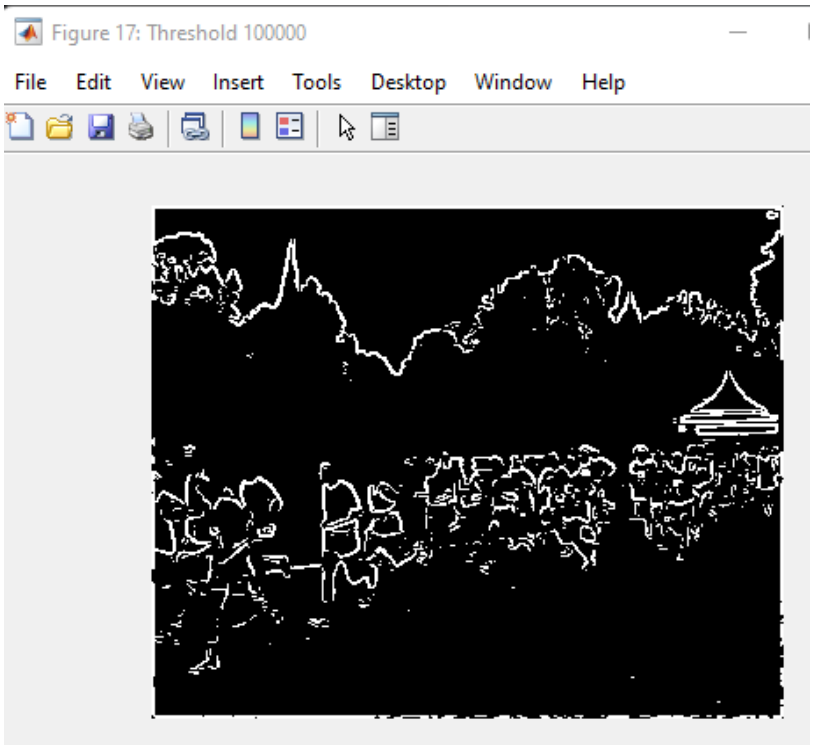


Thresholds 10000 to 50000 have been chosen for observation, I observe that when threshold is at 10000, there is already good improvement from the sum squared image in 3.1c where the whole image was filled up, now at t = 10000 the image has cleared up to pick up pixels that are edges based

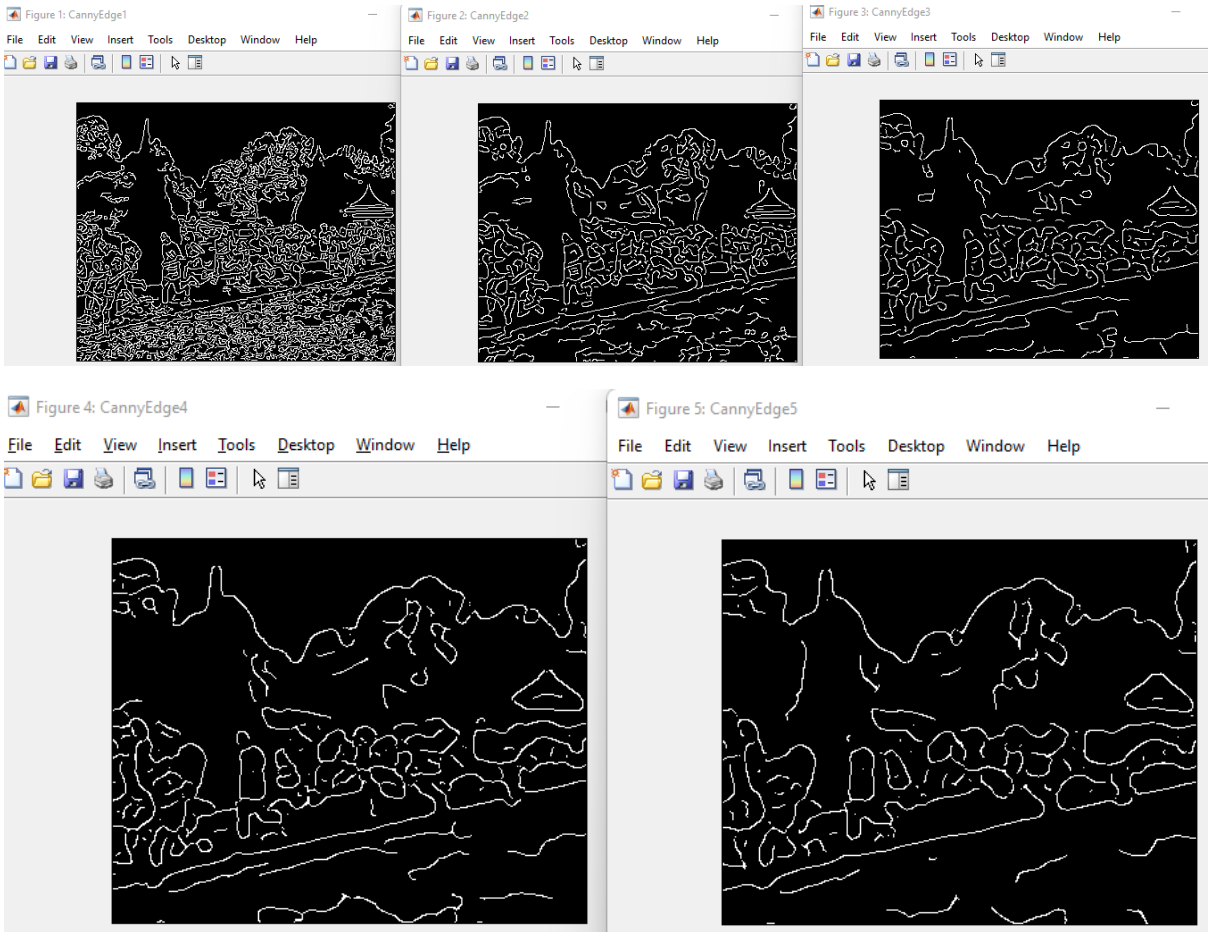
on both the horizontal and vertical gradients. Diagonal edges have also been observed to be successfully identified and displayed. As the threshold value t is gradually increased from 10000 to 20000 the image is observed to be less noisy as pixels with less strong edges have been further filtered out due to the increased threshold value, this behaviour and trend continues onwards until t= 50000 which is a more aggressive threshold, and has manged to filter out most of the noise, however, this also means that edge details that are not strong(magnitude values below threshold) are lost. It is thus important to experiment with different threshold values to find the best threshold parameter that preserves details yet filters out the most noise.



I can see in the picture below at t = 100000 , there is very little noise left and the edges left are very clear but many details have been lost such as the edge that represents the pavement road that the runners are on, so if detection of that edge is important, in application such as for self driving car, I will need to pick a threshold that is able to identify the edge, yet minimize the amount of noise.



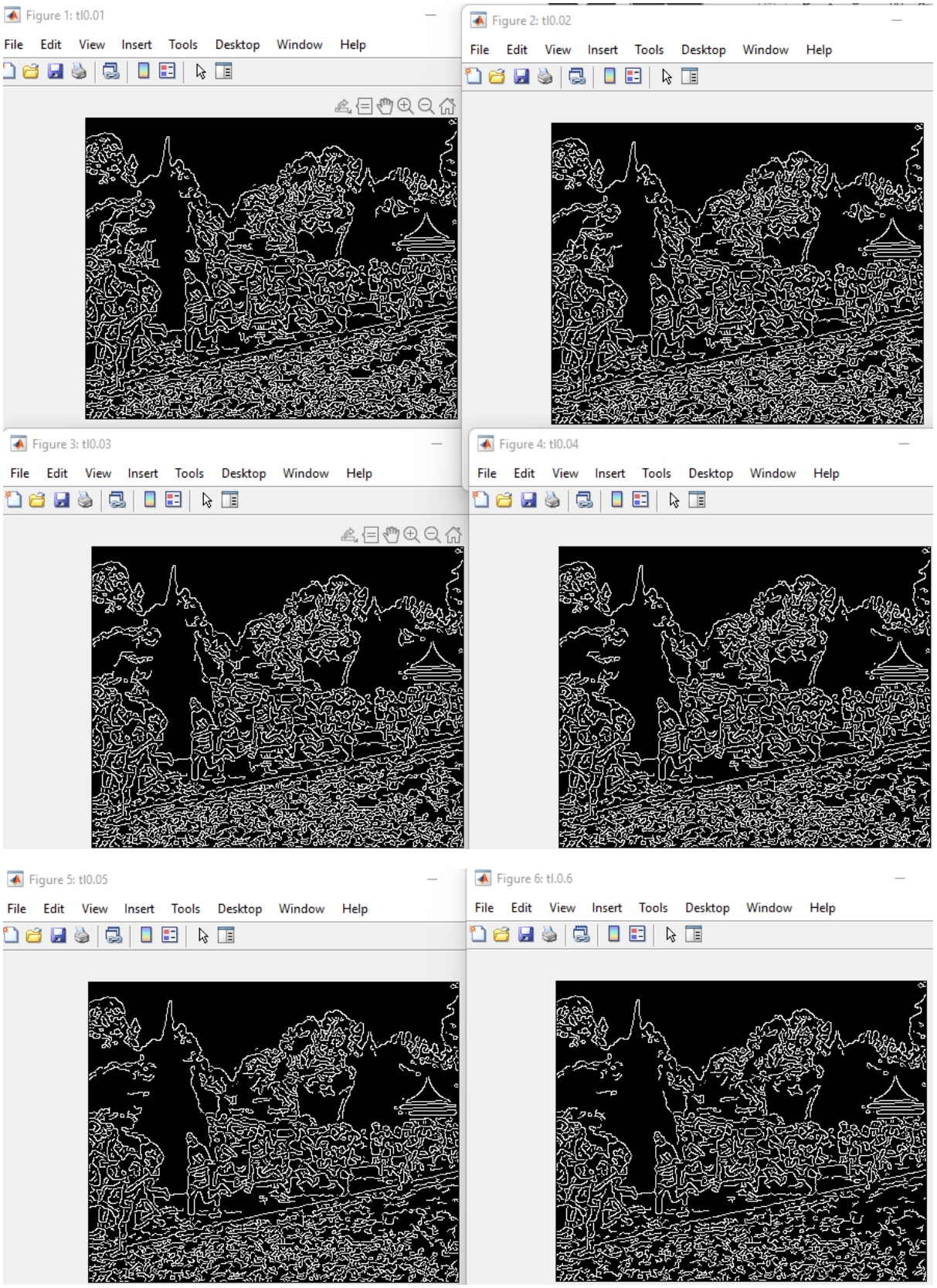
3.1e)



Images above have been computed by canny edge detector with sigma values from 1.0-5.0

The canny edge detector manages to detect all the edges well, including the diagonal edges. When the sigma value is first set at 1.0, many small edges are detected this is because the sigma value decides the size of the gaussian filter, when the sigma value is small, the gaussian filter is small hence there will be less blurring after filtering through this small gaussian filter, this leads to small sharper lines being detected by the canny filter. As the sigma value increases, the size of the gaussian filter increases as well which causes more blurring of the pixel value, this blurring actually spreads/ smears out the pixel value over a larger area , it can then be observed that when the sigma value increases in our images, less edgels are picked up as the canny filter operates on a higher level of detail, due to the pixel values having been smoothed out, the edges are also longer/larger and more connected, hence the canny detector is better at identifying and processing objects and edgels that are larger when the sigma value is increased, but it is also important to note that the edgels identified are more gradually smoothed and finer details, with smaller shorter edgels are lost, as observed by the roof of the temple/ pavilion on the right becoming more smooth, and the loss of the horizontal edge details on the roof.

3.1eii)



I have experimented with tl values from 0.01 to 0.04, from matlab documentation

```
edge(I,method,threshold,sigma)
```

“You can specify `threshold` as a 2-element vector of the form `[low high]` with `low` and `high` values in the range [0 1].”

Because tl in our code represents threshold low

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

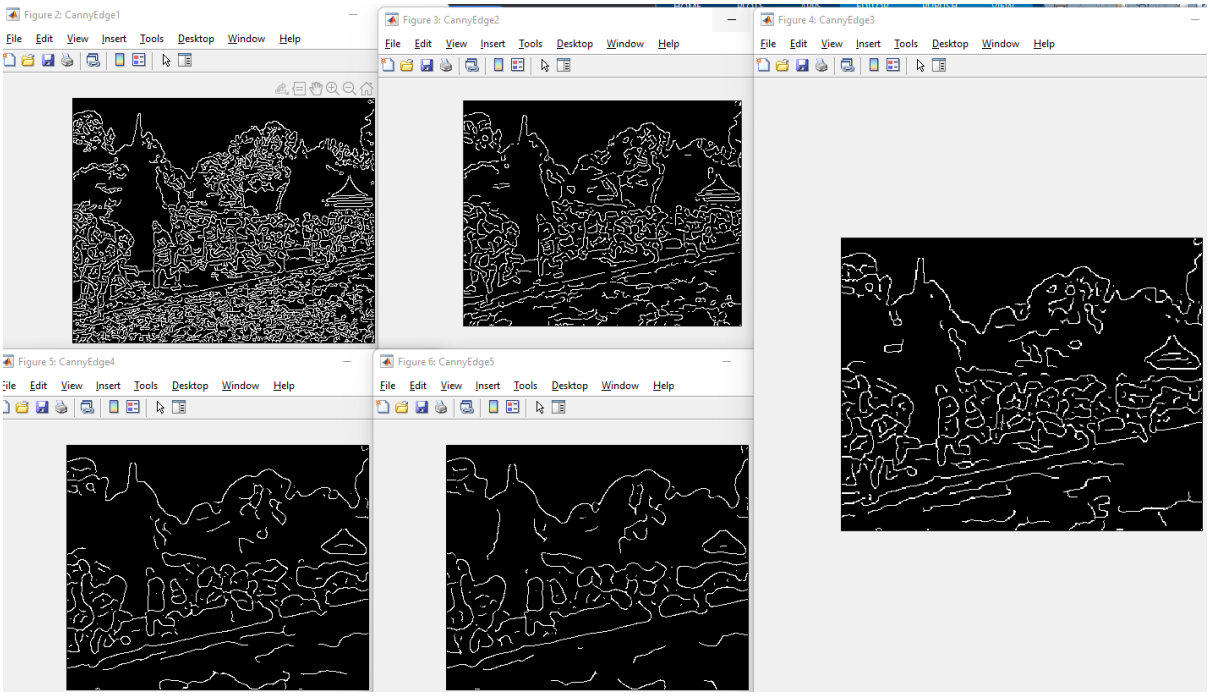
It can be understood that tl is the lower threshold of the canny edge detection’s hysteresis thresholding, and the expected result is that when the edges are below the threshold, they will be filtered out. Similarly in the result from experimentation, Figure 1 with tl =0.01 has the lowest threshold, hence filters out the least edgels, as compared to tl =0.06 where we can notice that more edgels have been filtered out as they do not pass the higher minimum threshold value.

3.2) Line Finding using Hough Transform

3.2a) Reusing edge image computed with canny , sigma = 1.0

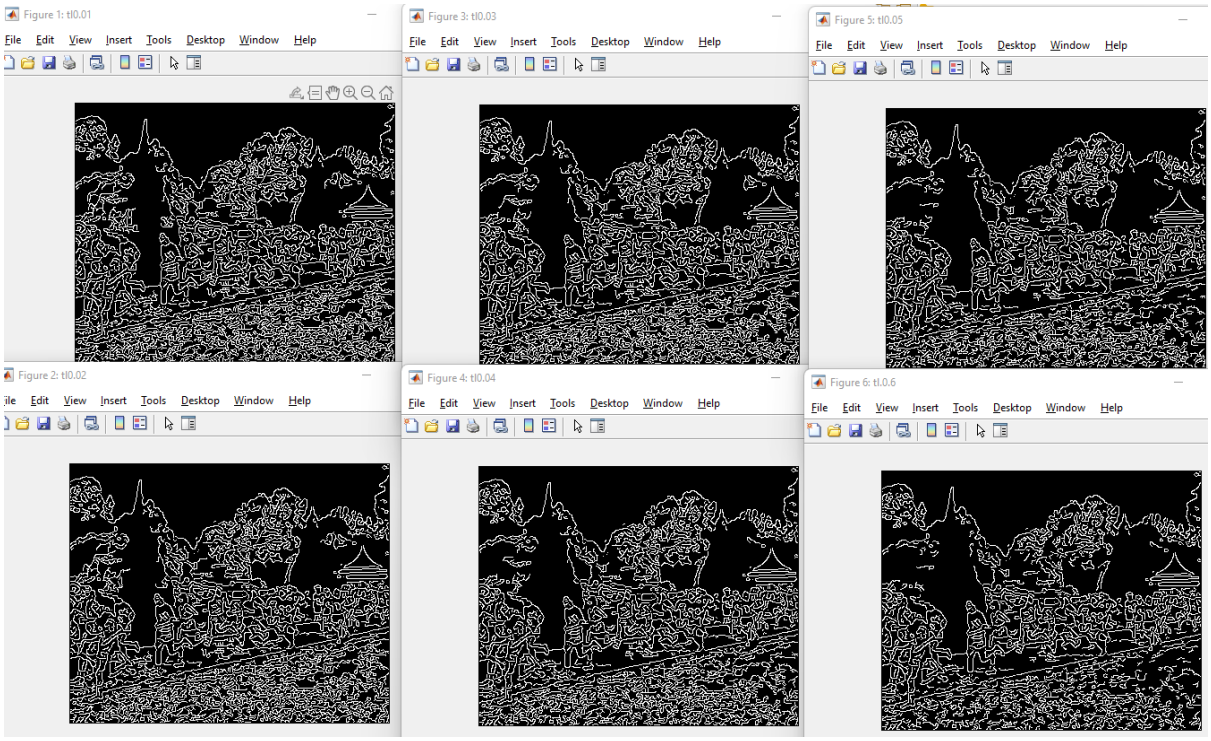
3.1e)

tl=0.04, th=0.1, sigma=1.0-5.0



We can observe that as the sigma value increases, less edges are preserved, and when sigma =5 only the strongest edges are preserved

3.1eii

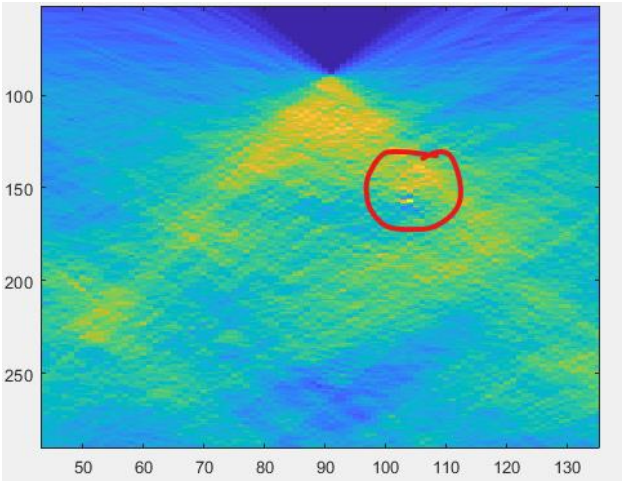


Tl changes are negligible.

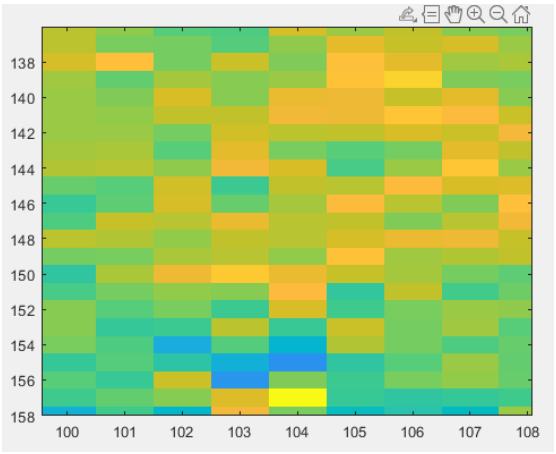
3.2b)

The hough and random transforms are similar as they are both mapping from image space to parameter space in terms of p and theta. They differ in terms of their implementation, while the hough transform maps data from the image to parameter space, the radon transform derives the similar point in parameter space from the image space instead. The hough transform takes every image pixel and adds this intensity to all corresponding parameter space bins while the radon transform takes the image points on a particular line, along with its angle theta to project into the parameter space into buckets of size p. This leads to the trade off where the radon transform will take longer than the hough transform but is more precise as it has a higher resolution due to the use of p sized buckets to split each subpixel's contribution proportionally according to distance between the projected location and bin centres

3.2c)



After transforming the image with the radon transform, the peak can be easily found by either using code or visually inspecting the image display.



When zoomed in, it is obvious that the max coordinates are theta =104 and radius = 157.

```
maxVal = max(H(:));
[xRadiusMax, thetaMax] = find(H== maxVal)
```

Evaluation using the max function similarly returns the max values as theta = 104 and radius = 157.

3.2d)

`[x,y] = pol2cart(theta,rho)` transforms corresponding elements of the polar coordinate arrays `theta` and `rho` to two-dimensional Cartesian, or *xy*, coordinates.

To convert [theta, radius] to image coordinates, we need to utilize the pol2cart function which returns 2D cartesian coordinates, X Y , mapped as A B , which will be used to substitute for C using the Ax+By=C line equation.

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

B values needs to be negated due to the y axis being inverted across the different coordinate system

```
A =

   18.3861


B =

   73.7425
```

With A, B values of the max theta and radius coordinates found, I now need to obtain C.
Since I already have(A,B) Ax+By = C, x and y values are also needed to solve for C, these x and y values can be obtained by first calculating the cartesian coordinates of the center of the image.

```
center = floor((size(greyImage)+1)/2)
center
```

Using the center coordinates allow us to calculate the appropriate transformation with respect to the original origin(center coordinates) of the image.I then proceed to solve for C

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

```
center =

   145   179


A =

   18.3861


B =

   73.7425


C =

   1.9760e+04
```

3.2e)

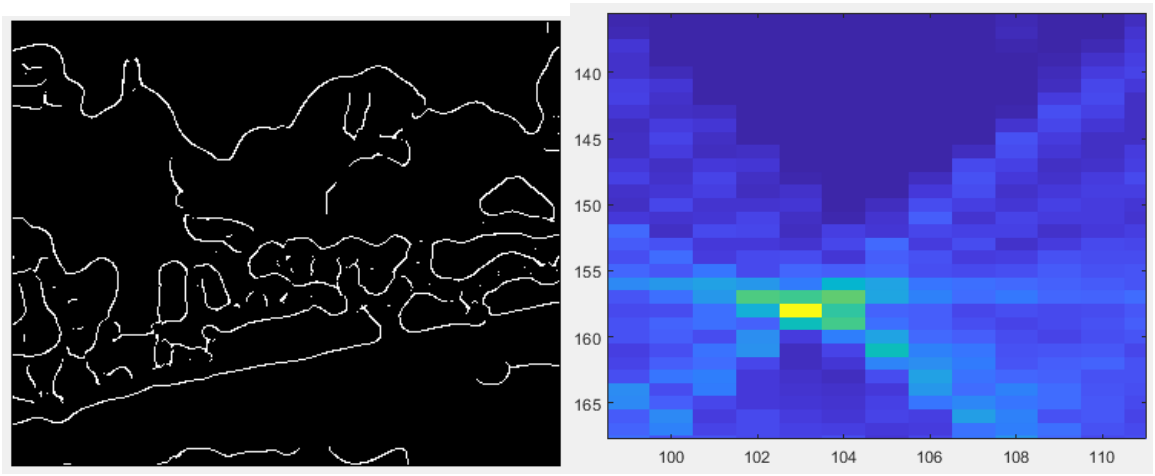
The yl and yr values can then be computed by plugging in the xl and xr values into the formula Ax+By=c

```
width=357
xl =0;
yl = (C -A *xl) /B;
xr =width-1;
yr = (C -A *xr) /B;
```

```
yl =

  267.9563


yr =

  179.1956
```

3.2f)

The line roughly matches with the edge of the running track, there is a slight deviation from the actual visible edge as it seems that the theta angle the line is drawn at is slightly off, this may be due to some deviations in the pre-processing step while extracting the edge information. I suspect that the initial edges identified with the current canny edge detector parameters may have caused these deviations , hence I went to tweak the parameters to attempt to better isolate the strong edge of the running track.



With Canny edge detector sigma value set to 6, the edge image looks a lot cleaner and the line representation has shifted somewhat to the new max values of theta = 103, radius = 157

Hence tweaking the canny edge detector allows us to better isolate the edge of the running track which is strongly represented in the radon transform as the max value because many of the edges are in a straight line and are hence lie in the same space in the hough space.

3.3)3D Stereo

3.3a)

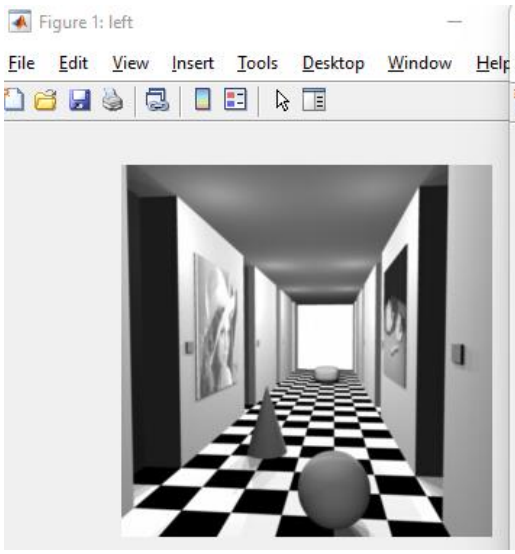
```
21  %%
22  function template= disparity_map(imLeft,imRight,xTemp,yTemp)
23
24  templateSize=11;
25  templateMid = ceil(templateSize/2);
26  templateOffset = templateMid-1;
27  [leftRow leftCol] = size(imLeft);
28  template = ones(leftRow-2*templateOffset:leftCol-2*templateOffset);
29
30
31  for x = templateMid : leftRow-templateMid+1
32      for y = templateMid : leftCol-templateMid+1
33          leftSlice = imLeft(x-templateOffset:x+templateOffset,y-templateOffset:y+templateOffset);
34          leftSliceRotated = rot90(leftSlice,2);
35          %global variable
36          min_coord = y;
37          min_diff = inf;
38          for z = max(templateMid, y-14) :y
39              sliceTemplate = imRight(x-templateOffset:x+templateOffset,z-templateOffset:z+templateOffset);
40              sliceTemplateRotated = rot90(sliceTemplate,2);
41              %correlate sliceTemplate with rotated slices to do matching
42              %and calculate SSD
43              convresult1 = conv2(sliceTemplate,sliceTemplateRotated);
44              convresult2 = conv2(sliceTemplate,leftSliceRotated);
45              ssd = convresult1(xTemp,yTemp)-2*convresult2(xTemp,yTemp);
46              if ssd<min_diff
47                  min_diff = ssd;
48                  min_coord = z;
49              end
50
51          end
52      template(x-templateOffset,y-templateOffset) =  y-min_coord;
53  end
54  end
55  end
```

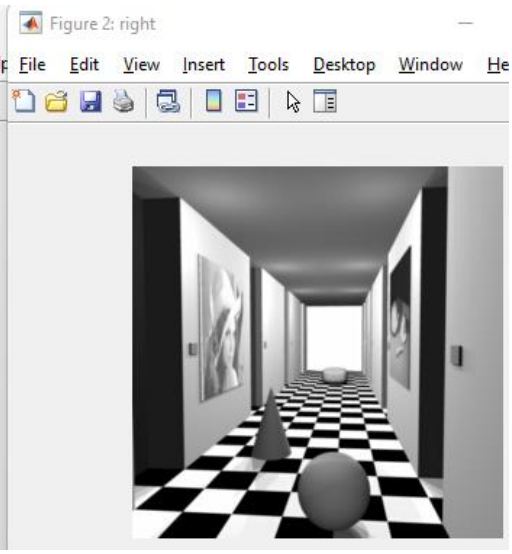
3.3i) line 28, template is initialized with a matrix of ones, excluding bordering pixels(used as padding) since the search required an 11x11 grid of valid pixels to perform the SSD computation.
3.3ii) line 31-52, SSD matching along scanline is done by looping through all pixels in the left image, grabbing an 11x11 slice of the pixels around each pixel in the left image, and comparing these slices with each and every pixel slice of 11x11 of the right image along the scanline (axis y). These slices are compared using convolution and the ssd score is extracted and stored in the template matrix.
3.3ii) line 52, disparity map is stored in template

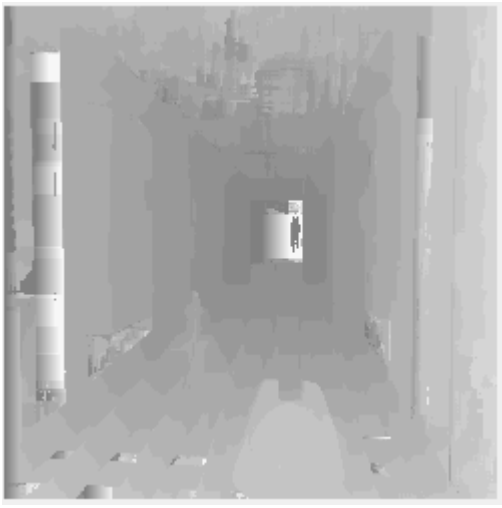
3.3b,c)

```
%%
templateSize=11;
templateMid = ceil(templateSize/2)
templateOffset = templateMid-1;
leftImg = imread('images\corridor1.jpg');
leftImg = rgb2gray(leftImg);
imshow(leftImg);
rightImg = imread('images\corridorr.jpg');
rightImg = rgb2gray(rightImg);
imshow(rightImg);
map =disparity_map(leftImg,rightImg,11,11);
figure('Name','test')
imshow(map,[-15 15]);
disp("end");
```

Reading in corridor and cooridorr and converting them into greyscale.
Calling function to read in the images and generate the disparity map using the disparity_map function.







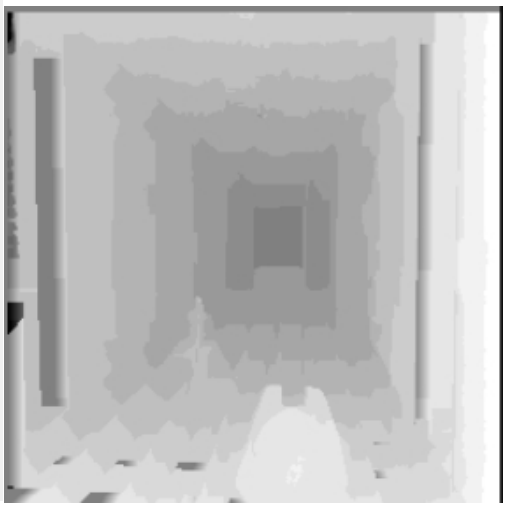
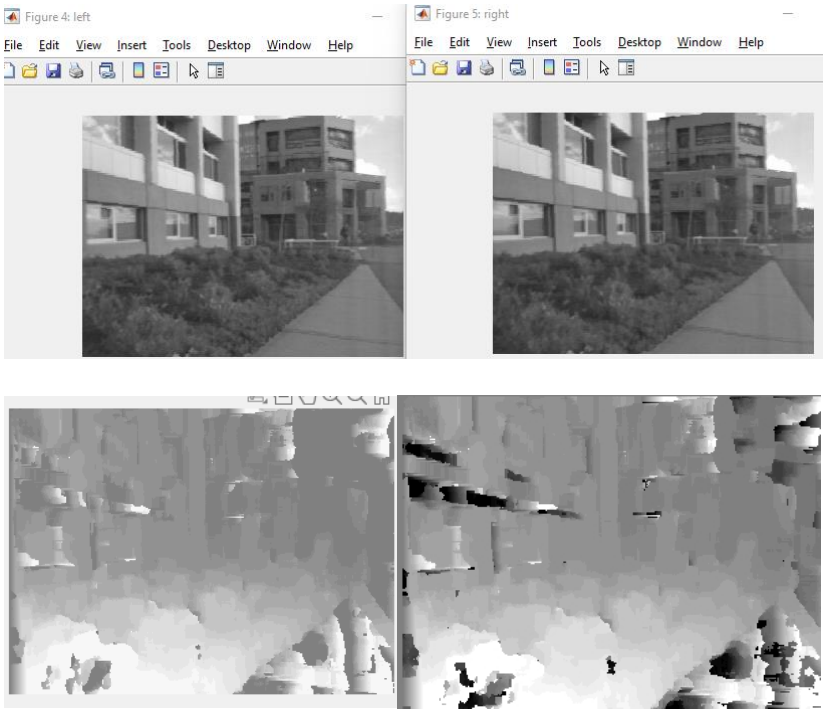


Image is of similar quality with corridor_disp and show nearer points as bright and further points as dark.

3d)



My result | Provided template result

The disparity map for this image can be observed to be poorer in result as compared to the previous image, this is because, the image has less structure to begin with as an outdoor image as compared to the previous indoor generated image. The depth is hence not as pronounced and the algorithm has difficulty identifying parts of the image with very poor contrasts such as the pavement, as it depends heavily on change in contrasts to detect the depth, resulting in very poor mapping that does not reflect depth at all on the pavement.

The End.