Lab 2: Edge Detection + Hough Transform + 3D Stereo

3.1 Edge Detection

3.1.1 Sobel Edge Detection

The first method of edge detection that is used is the Sobel filter. The vertical Sobel filter and the horizontal Sobel filter is convoluted with the image to from Gx and Gy respectively as shown in figure 1, where 'A' denotes the grayscale version of the 'macritchie.jpg' image. In MATLAB, the conv2 function can be used to achieve this. The keyword 'same' is used to return the central part of the convolution, which is the same size as the original image, 'A' {14,15}*.

*{} indicate the line number(s) in the main source code [Refer to Appendix A]

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

Figure. 1. Vertical and Horizontal Sobel filter application

From figure 2, it is observed that Gx contains the vertical lines filtered out by the vertical Sobel filter while Gy contains the horizontal lines filtered out by the horizontal Sobel filter. However, Gx and Gy are cluttered with too many edges and the main objects such as the trees, runners and pavement in the picture cannot be differentiated. In addition, the vertical and horizontal Sobel filters are unable to filter out diagonal edges. This means important edge information might potentially have been lost. Hence, some extra processing is required.

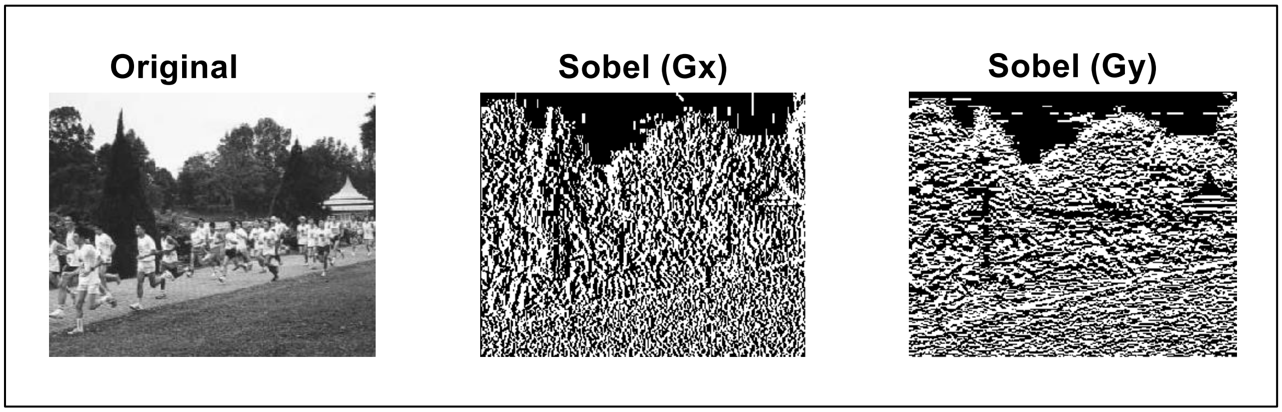

Figure. 2. Result vertical and horizontal Sobel filtering on the image 'macritchie.jpg'

A combined edge image, Sobel(x,y) is generated by adding the squared images, Gx and Gy {16}. The values are squared to remove any possible negative values in the image. As an example, the strength of an edge of ‘A’ intensity and the strength of an edge of ‘-A’ intensity is the same. Hence, squaring it will make both edges the same intensity value at ‘A^2’. The Sobel(x,y) image in figure 3 displays an image full of edges of varying intensities, too many to visualize any valuable information. Hence, a threshold added to filter out the strong edges from the noise {17}. The lower the threshold, the more details will be shown. However, it might include less important edges that will add noise to your important ones. A threshold of ‘200’ results in the image Sobel(threshold), with the edges of the trees, humans and pavement being clearly visible.

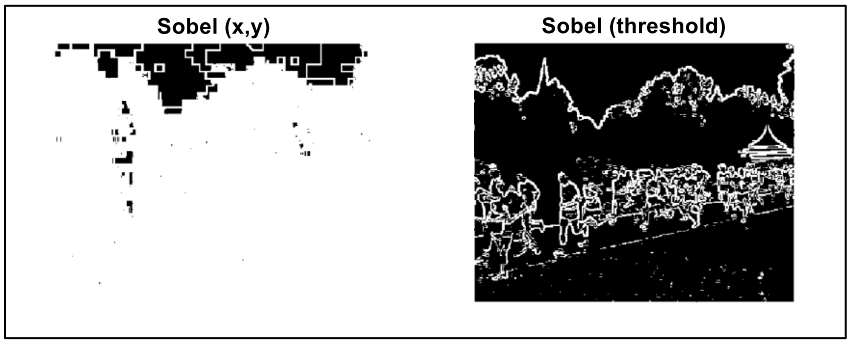

Figure. 3. Combination of vertical and horizontal Sobel filtering and thresholding

3.1.2 Canny Edge Detection

For Canny edge detection, the edge() function in MATLAB can be used {29-31}. It takes in the original image, lower and upper threshold limits, and the sigma value as input parameters and outputs a edge map. In the first step of the canny algorithm, gaussian derivative filtering is done, with sigma controlling the scale parameter. Smaller values of sigma (1-2) will return finer scales edges while larger values of sigma (3-5) will return coarser scale edges, as reflected in figure 4. As a result, larger values of sigma have greater noise suppression capabilities. As for accuracy, it depends on what is the point of interest. If minor details like finding out the number of people running are important, a lower sigma value is preferred. However, if the environmental details like trees and pavement are important, then a high sigma value is preferred. The next step is nonmaximal suppression, which transform all edges into 1 wide width along the direction of the intensity gradient.

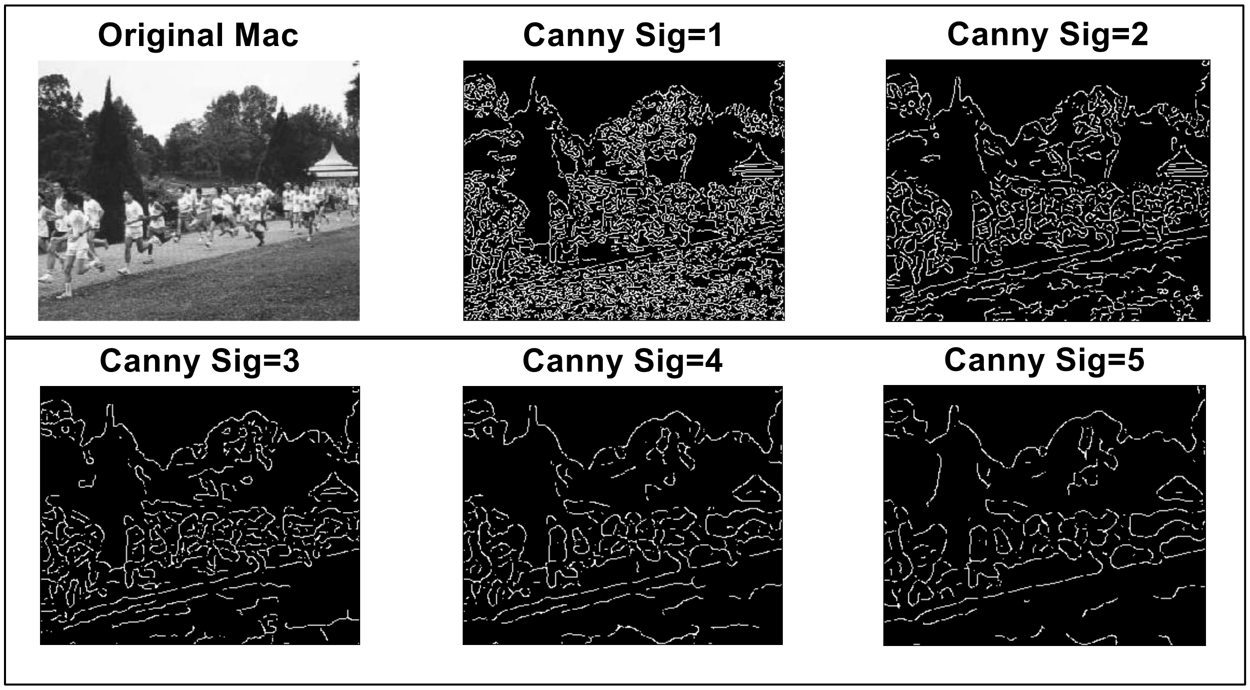

Figure. 4. Result of Canny edge detection with varying sigma value

The last step is hysteresis thresholding. For my experiment, the lower threshold is adjusted from 0.015 to 0.075 while the upper limit is 0.1 {33-35}. As most of the edges have intensities are higher than the upper threshold of 0.1, they have been remapped to maximum intensity. Hence, there is little change when adjusting the lower limit, as seen from figure 5. Theoretically, all those below the lower threshold will be remapped to 0 intensity. From the results, we can observe that there exist some edges between 0.015 to 0.075 threshold, mainly those in the tree in front. As tl increases, more edges are filtered out, resulting in more black spots in the tree.

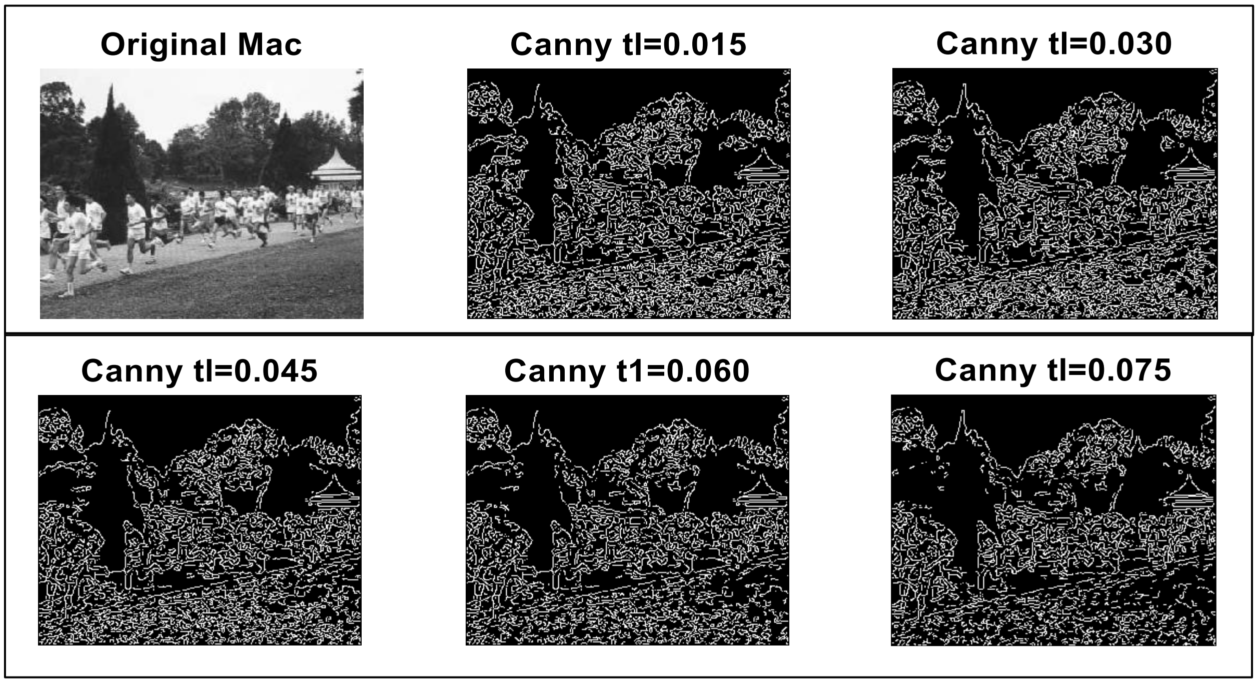

Figure. 5. Result of Canny edge detection with varying lower threshold (tl) value

3.2 Line Finding using Hough Transform

3.2.1 Radon Transform vs Hough Transform

Hough Transform and Radon Transform both maps an image from the image space to a parameter space of ρ and θ, which is also known as the Hough space. AltHough the result of transformation is similar, the way they transform is different, and it affects the accuracy and speed of the algorithm. Due to their unique properties, both transformation methods are still relevant and used in different fields. The differences are as detailed in table 1 below.

Table 1. Differences between Radon Transform and Hough Transform

	Radon Transform	**Hough Transform**
Definition	A mathematically integral transform, defined for continuous functions on Rn on hyperplanes in Rn	A discrete algorithm that detects lines in an image by polling and binning
How it transforms	Calculates the characteristic function of a random variable as the Fourier transform of its probability density function (PDF)	Generates a random sequence, calculating its empirical PDF by histogram binning and then transforming it appropriately
Main advantage	More accurate	Less accurate

Main disadvantage	Slower	Faster
Application	Tomography (medical, seismic, etc.), microscopy, etc.	Image processing, computer vision, etc.

As time is a big factor in image processing, Hough transform will be used instead. There are other methods in image processing that can be utilized to combat the loss in accuracy, which will be discussed in part 3.2.2.

3.2.2 Hough Transform

As of now, there exists functions on MATLAB for Hough Transform, namely the Hough(), Houghpeaks() and Houghlines() functions. The Hough() function takes in the edge imagine from 3.1 as an input parameter and maps each point into its corresponding curve in the Hough domain {55}. It then returns the image in the Hough domain, and a 2D vector of for theta points and a 2D vector for rho points. Figure 6 is a plot of each point in the edge image in the Hough domain.

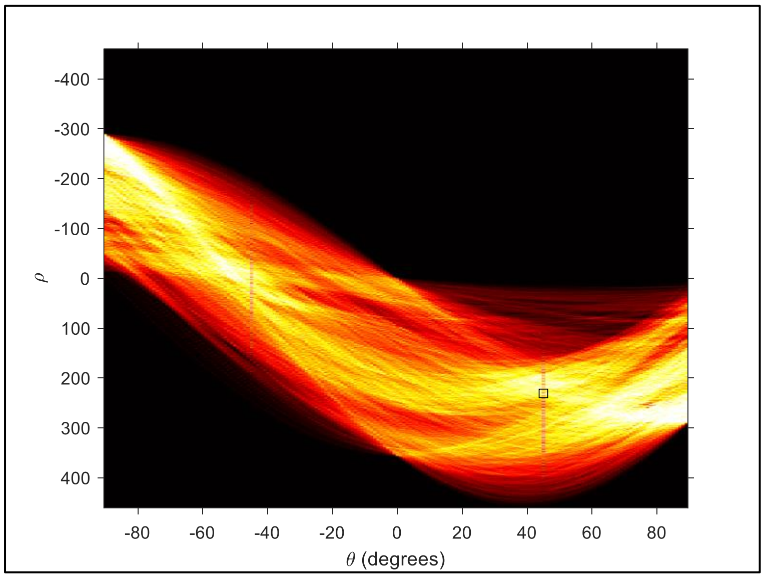

Figure. 6. Result of Canny edge detection with varying lower threshold (tl) value

Each intercept of curves in the Hough domain represent points that can form a line in the spatial domain. Higher number of intercepts means those points have a higher probability to form a line in the spatial domain. By connecting those points, the gaps between would have been filled.

For my experiment, only the most probable line is of interest. Hence, the function Houghpeaks(H,1) is utilized to find the point with the most intercepts in the Hough domain {69}. Those lines that intercept at that point are then converted back to the points in the spatial domain using the theta() and rho() functions {71-72}. The vector of points is transformed into a line using the Houghlines() function {75}. Lastly, the line is overlayed onto the original image to check if it has correctly detected the pavement edge {79-84}.

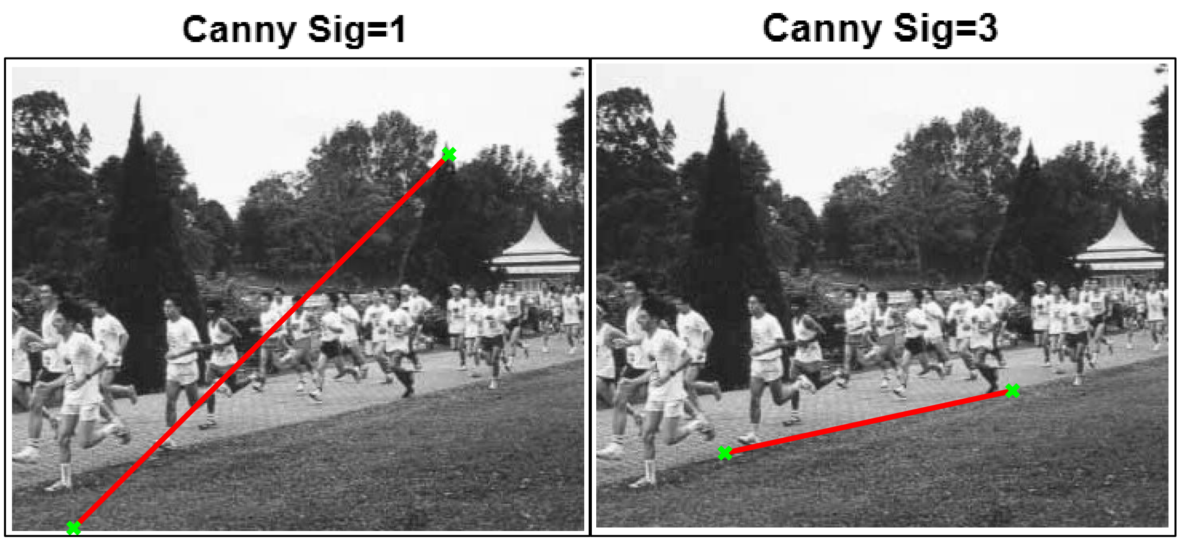

Figure. 7. Overlay of line with most intercepts on original image for sigma of 1 and 3

Unfortunately, due to the large amount of noise edges for sigma=1, the resultant line was inaccurate, as in figure 7. One solution is to increase the sigma value to 3, removing many of the noise edges, which results in the line along the pavement to be found. The lower threshold of hysteresis thresholding can also be adjusted to moving the edges with low thickness.

3.3 3D Stereo

In 3D Stereo, 3D depth information is extracted from two 2D digital images. This is done using triangulation of points, examining the relative positions of objects in the two images, and generating a disparity map. The map() function is written to take in the two images and template dimensions and output the resultant disparity map [1]*. For our purpose, the template is a 11x11 square, hence only 1 variable is needed for template dimensions. As benchmark, the disparity map is generated using the disparityBM() function provided in MATLAB. This is done for both the ‘corridor’ and ‘triclops’ image pairs.

*[] indicate the line number(s) in the map() function [Refer to Appendix B]

The procedure of generating the disparity map in map() is as follows:

1. Initialize variables [2-5]
2. Select a point in first image and extract 11x11 template from it [13]
3. Initialize local variables for minimum difference and coordinate of points with minimum difference [15,16]
4. Search 14 points left of selected point and calculate Sum of Squared Difference (SSD) for each [20-27]
5. Find the point with minimum SSD out of the 14 points and record the disparity in the final matrix ret [28-33]
6. Repeat steps 2-5 for all points in the image except the 5 width corner pixels

The results are presented in figure 8.

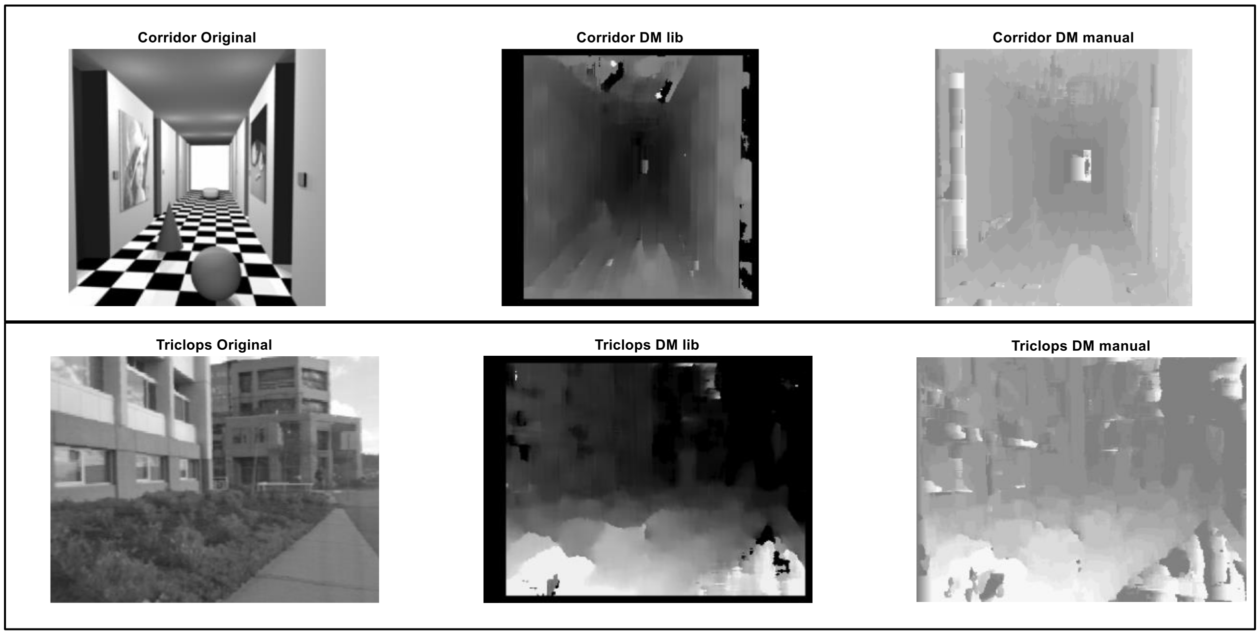

Figure. 8. Disparity map for 'Corridor' and 'Triclops' image pairs using disparityBM() and map() functions

The first observation is that the one generated by disparityBM() is much faster than that of map(). One reason could be the space where convolution is performed. Performing the convolution in real space is significantly more computationally expensive than convolution by fast fourier transform (FFT), which is especially the case as the template size increases. Overall, the pattern for both results are similar, being lighter in the foreground and darker in the background. However, certain places like the black walls in the corridor image and the pavement in the triclops image. This is due to the correspondence problem, where surfaces have similar appearance or features, resulting in a difficulty in finding the exact point in the first image that corresponds to the point in the second image.

Appendix A: Main Source Code

```
clc;
clear;
close all;

%% Edge Detection

mac = imread('maccropped.jpg');

mac_g=rgb2gray(mac);

 sob_x = double([-1 0 1;-2 0 2;-1 0 1]);
 sob_y = sob_x';

 Gx = conv2(mac_g,sob_x,'same');
 Gy = conv2(mac_g,sob_y,'same');
 Gxy = sqrt(Gx.^2 + Gy.^2);
 Gt=Gxy>200;

 figure('name','Sobel filtering');
 subplot(131),imshow(mac_g),title('Original Mac');
 subplot(132),imshow(Gx),title('Sobel (Gx)');
 subplot(133),imshow(Gy),title('Sobel (Gy)')
 figure('name','Sobel filtering w post processing');
 subplot(121),imshow(Gxy),title('Sobel (x,y)')
 subplot(122),imshow(Gt),title('Sobel (threshold)')

```

```
%Canny Edge Detection

for i = 1:5
    mac_canny1(:,:,i) = edge(mac_g,'canny',[0.04 0.1],i);
end

for j = 1:5
    mac_canny2(:,:,j) = edge(mac_g,'canny',[(0.015*j) 0.1],1);
end

figure('name','Canny Edge Detection (Changing sigma)')
subplot(231),imshow(mac_g),title('Original Mac');
subplot(232),imshow(mac_canny1(:,:,1)),title('Canny Sig=1');
subplot(233),imshow(mac_canny1(:,:,2)),title('Canny Sig=2')
subplot(234),imshow(mac_canny1(:,:,3)),title('Canny Sig=3')
subplot(235),imshow(mac_canny1(:,:,4)),title('Canny Sig=4')
subplot(236),imshow(mac_canny1(:,:,5)),title('Canny Sig=5')

figure('name','Canny Edge Detection (Changing tl)')
subplot(231),imshow(mac_g),title('Original Mac');
subplot(232),imshow(mac_canny2(:,:,1)),title('Canny tl=0.015');
subplot(233),imshow(mac_canny2(:,:,2)),title('Canny tl=0.030')
subplot(234),imshow(mac_canny2(:,:,3)),title('Canny tl=0.045')
subplot(235),imshow(mac_canny2(:,:,4)),title('Canny t1=0.060')
subplot(236),imshow(mac_canny2(:,:,5)),title('Canny tl=0.075')

%% Line Finding using Hough Transform

```

```
[H,theta,rho] = Hough(mac_canny1(:,:,3));

figure('name','Picture in Hough domain')
imshow(imadjust(rescale(H)),[],...
      'XData',theta,...
      'YData',rho,...
      'InitialMagnification','fit');
xlabel('\theta (degrees)')
ylabel('\rho')
axis on
axis normal
hold on
colormap(gca,hot)

P = Houghpeaks(H,1);

x = theta(P(:,2));
y = rho(P(:,1));
plot(x,y,'s','color','black');

lines = Houghlines(mac_canny1(:,:,1),theta,rho,P);

figure('name','Line Finding using Hough Transform'), imshow(mac_g), hold on

xy = [lines(1).point1; lines(1).point2];
plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','red');

% Plot beginnings and ends of lines
```

```
plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','green');
plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','green');

%% 3D Stereo

%Using disparity function
corrL = imread('corridorl.jpg');
corrR = imread('corridorr.jpg');

corrL_g=rgb2gray(corrL);
corrR_g=rgb2gray(corrR);

disparityRange = [0 16];
disparityMap =
disparityBM(corrL_g,corrR_g,'DisparityRange',disparityRange,'UniquenessThreshold',0);

% figure('name','Disparity Map 1')
% imshow(disparityMap,disparityRange)

triL = imread('triclopsi2l.jpg');
triR = imread('triclopsi2r.jpg');

triL_g=rgb2gray(triL);
triR_g=rgb2gray(triR);

disparityRange2 = [0 16];
disparityMap2 =
disparityBM(triL_g,triR_g,'DisparityRange',disparityRange2,'UniquenessThreshold',0);

```

```
% figure('name','Disparity Map 2')
% imshow(disparityMap2,disparityRange2)

figure('name','Disparity Map overall')
subplot(231),imshow(corrL_g),title('Corridor Original');
subplot(232),imshow(disparityMap,disparityRange),title('Corridor DM lib');
subplot(234),imshow(triL_g),title('Triclops Original')
subplot(235),imshow(disparityMap2,disparityRange2),title('Triclops DM lib')

%Manual try 2

n= 11;

res = map(corrL_g, corrR_g, n);

subplot(233),imshow(res, [-15 15]),title('Corridor DM manual')

res2 = map(triL_g, triR_g, n);

subplot(236),imshow(res2, [-15 15]),title('Triclops DM manual')
```

CE4003_lab2.m

Appendix B: Map Function

```
function ret = map(img_l, img_r, s)
  n = s;
  n1 = floor(n/2);
  [x,y]=size(img_l);
  ret = ones(x - n + 1, y - n + 1);

  msg = 'Creating Disparity Map ...';
  xw=0;
  fw = waitbar(xw, msg);

   for i = 1+n1 : x-n1
       for j = 1+n1 : y-n1
           cur_r = img_l(i-n1: i+n1, j-n1: j+n1);
           cur_l = rot90(cur_r, 2);
           min_coor = j;
           min_diff = inf;

           % search for simmilar pattern in right image
           % limit search to 15 pixel to the left
           for k = max(1+n1 , j-14) : j
               T = img_r(i-n1: i+n1, k-n1: k+n1);
               cur_r = rot90(T, 2);

               % Calculate ssd and update minimum diff
               conv_1 = conv2(T, cur_r);
               conv_2 = conv2(T, cur_l);
```

```
                ssd = conv_1(n, n) - 2 * conv_2(n, n);
                if ssd < min_diff
                    min_diff = ssd;
                    min_coor = k;
                end
            end
            ret(i - n1, j - n1) = j - min_coor;
        end
        xw = i/(x-n1);
        waitbar(xw,fw);
    end
    close(fw);
end
```

map.m