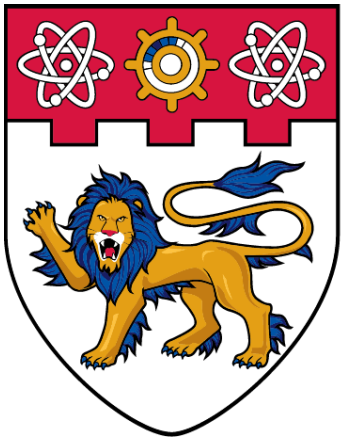


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Lab Report 2 :
Advanced Image Processing in MATLAB

(Edge Detection, Template Matching, Disparity Mapping & Spatial Pyramid matching)

CZ4003 – COMPUTER VISION
(Semester 1, AY 2020/2021)

WILSON THURMAN TENG
U1820540H

1. Introduction

1.1 Overview

In this report, we to explore and implement different algorithms used for image processing in MATLAB. In this report, we will explore and implement the following techniques used in image processing

(a) Edge detection using Sobel filter;

(b) Employ Hough Transform to recover dominant line edges in the image;

(c) Experiment with pixel sum-of-squares difference (SSD) to find a template match within a larger image. Estimate disparity maps via SSD computation;

(d) Compare the bag-of-words method with Spatial Pyramid Matching (SPM) on the benchmark Caltech-101 dataset.

1.2 Dependencies

The following tools will be used in the implementations of this lab report:

- MATLAB R2020a (Source code tested on this version)
- Image Processing Toolbox software package

2. Experiments

2.1 Edge Detection

(a) Display original and grayscale image

The whos command is useful for finding out the image size as well as the pixel datatype. We also obtain the minimum and maximum intensity of the image.

```
  Name        Size                Bytes  Class    Attributes

  Pc        290x358x3            311460  uint8


min_intensity =

  uint8

   13


max_intensity =

  uint8

   254
```

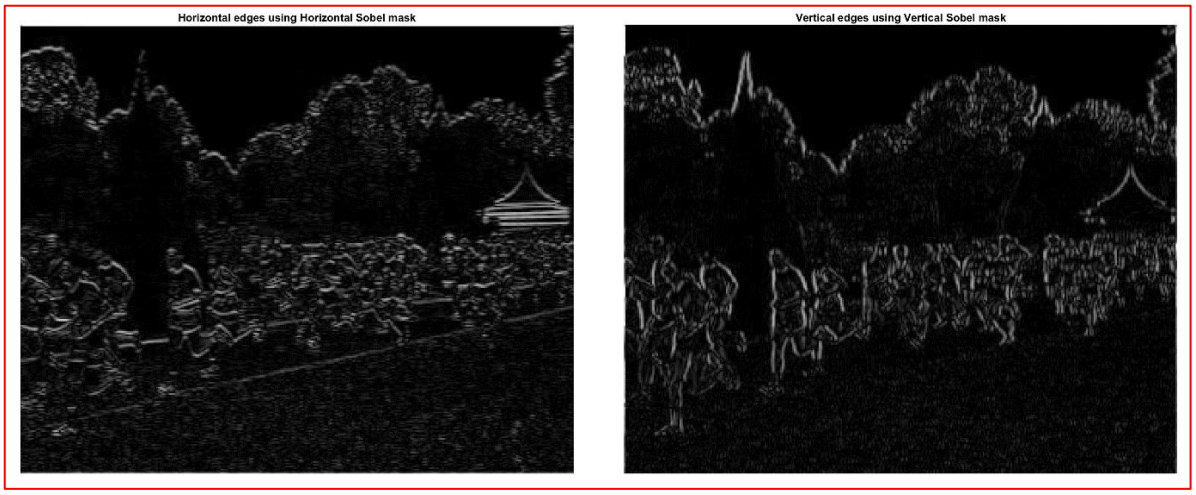
(b) Generate Horizontal & Vertical edges using Sobel Filter

Horizontal and Vertical Sobel Masks:

```
sobel_h_mask = [
    -1 -2 -1;
    0 0 0;
    1 2 1
];
```

```
sobel_v_mask = [
    -1 0 1;
    -2 0 2;
    -1 0 1
];
```

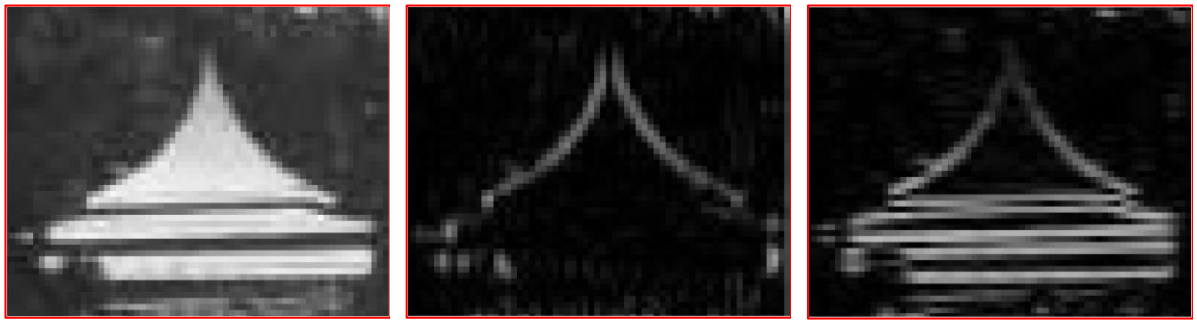
Output after convolution:



:

In this implementation, contrast stretching was applied to the image as a pre-processing technique before applying the masks. This allowed for edge detection to be easier due to the larger pixel intensity differences.

Observation for diagonal edges:



Original Grayscale Image | Vertical Edges | Horizontal Edges

For edges that are not strictly vertical or horizontal (i.e. diagonal), the edges are fainter than their strictly vertical or horizontal counterparts. This is more evident after zooming into the pavilion part of the image. We can observe that the strictly horizontal edges are absent after applying the vertical Sobel mask and similarly, the strictly vertical edges are absent after applying the horizontal Sobel mask. However, diagonal edges are still present in both derived images, although much fainter. This will be apparent in the next stage when we combine both types of edges.

(c) Combined Edge Image

A squaring operation was applied on both the vertical and horizontal edge images and summed together. The resulting image allows us to see the both types of edges as well as horizontal edges without the faint line issue as before for prominent edges.

Rationale for square operation before summing both images:

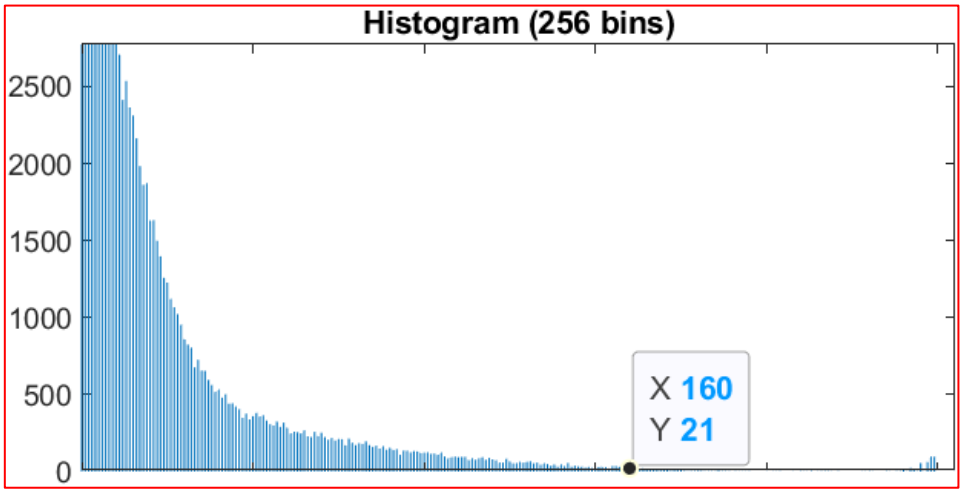
The vertical and horizontal Sobel masks allow us to obtain gradient approximations which may result in negative values. Both the negative and positive values are just as important as they give us the complete information of the edges detected. By squaring both edge images, we take their values into account by converting them into positive values.

An added benefit is that it allows us to extract diagonal edges as both horizontal and vertical components are considered.

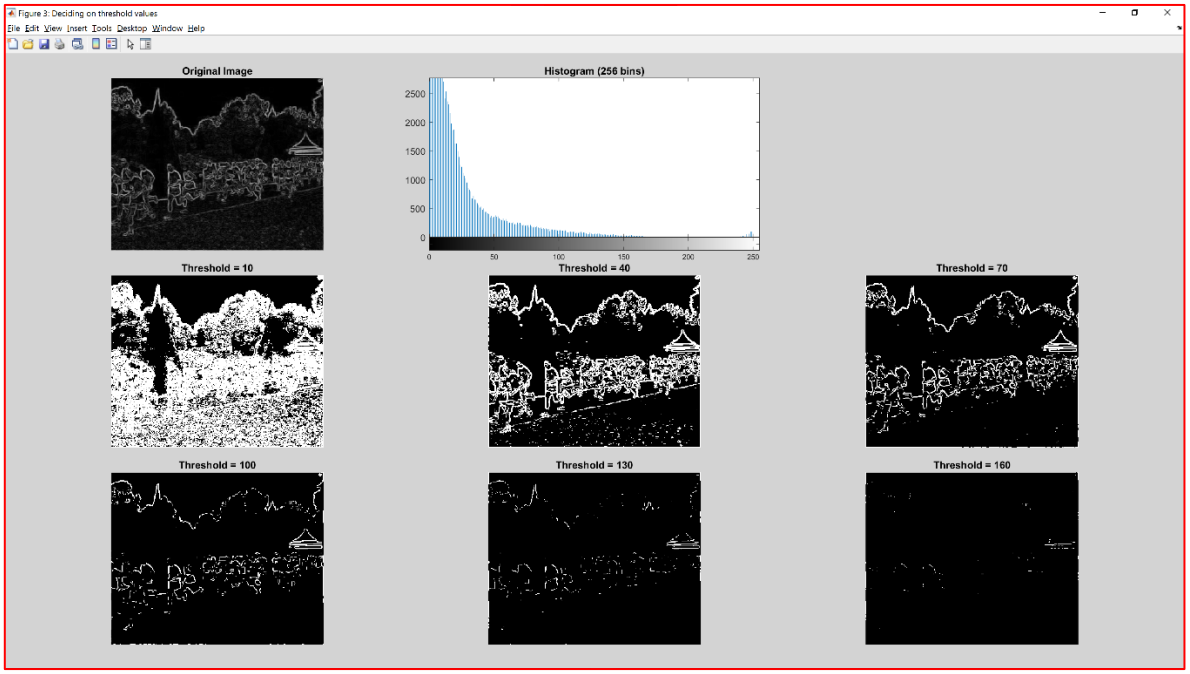
(d) Thresholding

In the earlier stage, we have already performed contrast stretching. This allows us to choose a threshold value more easily as the range of possible threshold values are reduced. Although this may cause some contrast information to be lost, it is a non-issue for this simple thresholding experiment.

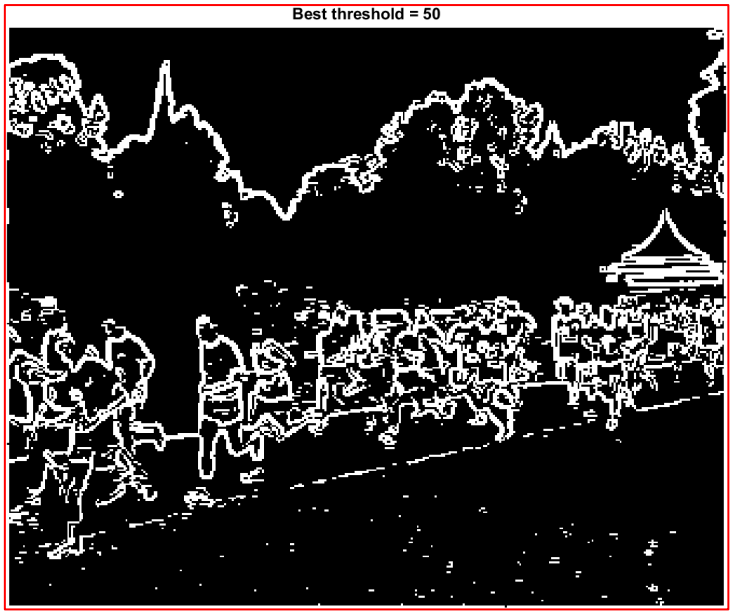
To aid us in selecting a suitable threshold value, we plot a histogram to investigate the pixel intensity distribution of our edge image obtained from the previous section. From a preliminary observation, we can observe that 40 and 160 seem to be promising candidates for the threshold. 40 is a good threshold value as it is where a steep drop in pixel intensity is observed while 160 is a good threshold value as it separates the high pixel intensities with low count which could be the prominent edges that we may be interested in.



To compare the effects of using different thresholds, the images for 6 different thresholds are computed.



From this comparison of results, we observe that a threshold value of 40 captures most of the important edges in the image with some noise while a threshold of 70 has almost no noise while allowing us to retain important edges.



After some experimentation, we find that a threshold value of 50 is the best as it minimizes noise while retaining important edges (i.e. humans, track boundary, buildings & tree canopy boundary).

Insights on the advantage and disadvantages of using different threshold values:

Low Thresholds:

+ Detailed edge information about the image is retained. This is useful if the goal is to detect noise.

- Low thresholds are sensitive to noise. For example, when the threshold value is 10, edges are detected in almost the entire image.

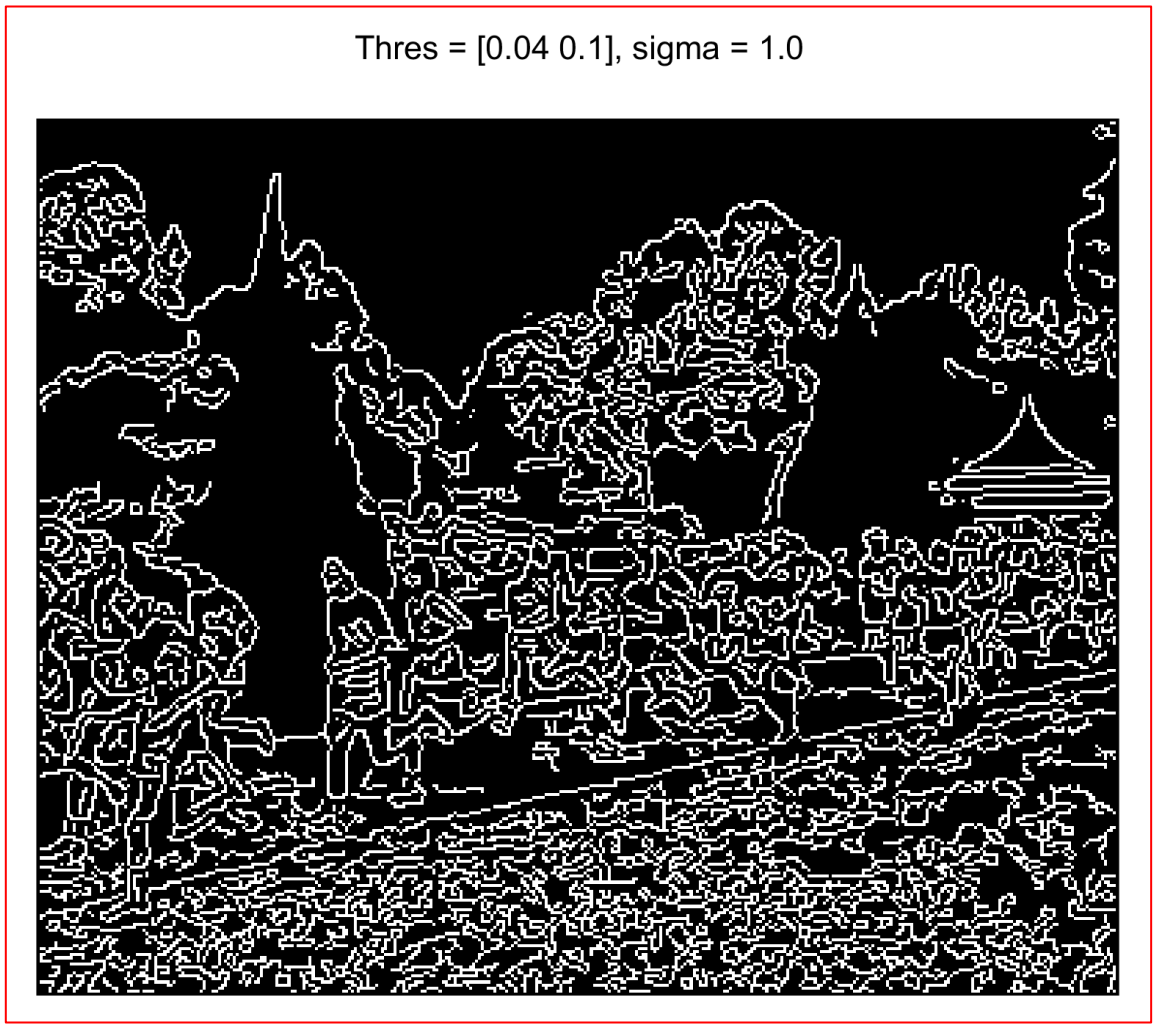
High Thresholds:

+ Although still susceptible to noise, we obtain edges that are more prominent, which are more likely to be useful.

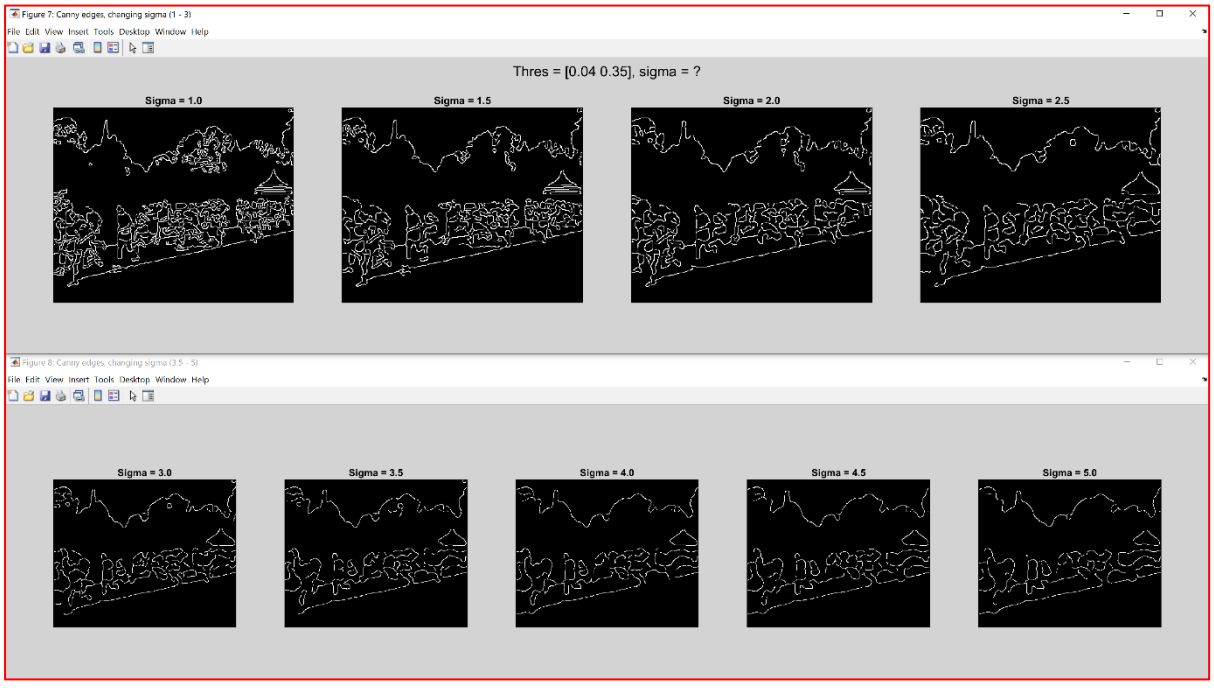
- If the threshold value is too high, we may not detect any edges at all. In the histogram of the matrichie.jpg combined edges image, the number of pixels with an intensity value of 160 or greater is low. Hence, we were unable to see any contiguous detected edges with semantic significance when the threshold value was set at 160.

(e) Canny Edge Detection algorithm

Canny edge detection algorithm applied on the combined edge image.



(e)(i) Canny Edge Detection algorithm (Varying Sigma)

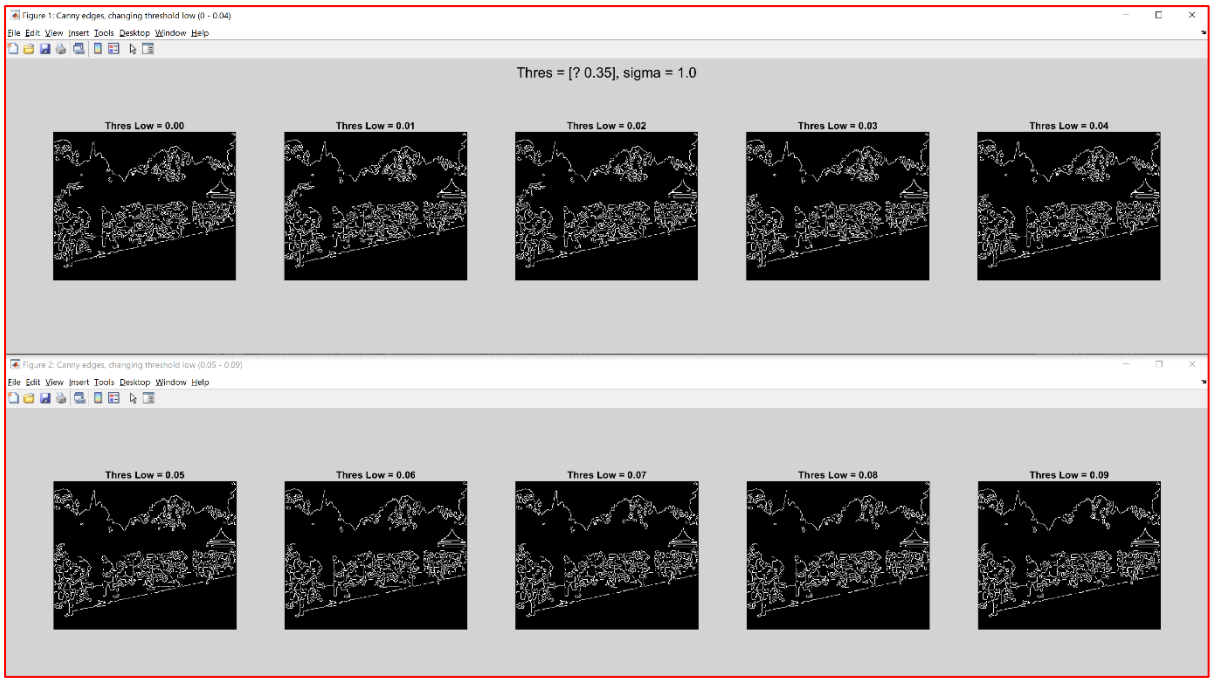


We can observe that as sigma increases, the noise is gradually removed. However, the edges are also less defined as some of the important edges are lost. (i.e. humans look more like blobs at high sigma values).

Conclusion:
High sigma values are suitable for **noisy edgel removal** while low sigma values are required to preserve the **location accuracy of edgels**.

This is so as the sigma parameter corresponds to the strength of the gaussian filter applied. A larger sigma value corresponds to a flatter gaussian curve which results in more smoothing.

(e)(ii) Canny Edge Detection algorithm (Varying Threshold Low)



The canny edge detector uses hysteresis thresholding and *tl* corresponds to the lower bound threshold value used. Any value lesser than the set *tl* value will be disregarded as an edge.

In this experiment, as the *tl* value is increased, the lesser the amount of edges detected. (i.e. details in the tree line). This results in less noise overall as more weak edges are removed.

2.2 Line finding using Hough Transform

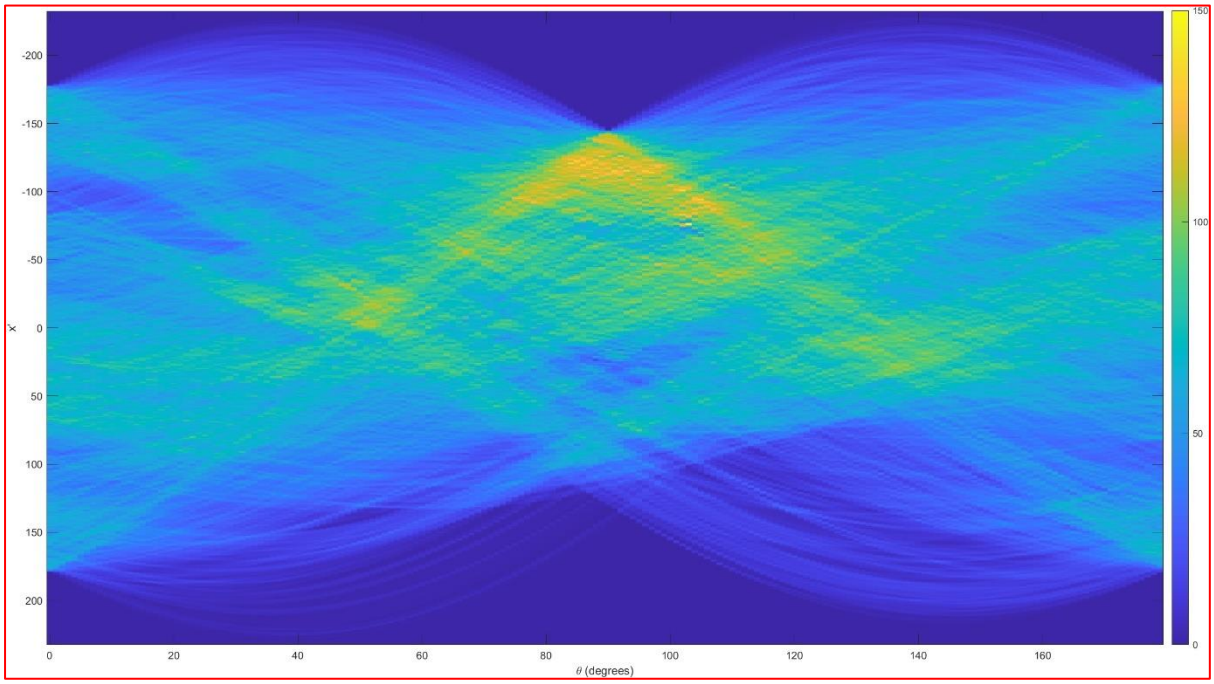
(a) Reuse Canny edges image with *tl* = 0.04, *th* = 0.1 & *sigma* = 1.0

(b) Performing Radon transform (equivalent to Hough transform for binary images)

Source code:

```
help radon
default_theta = 0:179;
[H, xp] = radon(cannyedges, default_theta);
```

Using Radon transform to compute Hough transform:



To visualize the projection better, a colour map with a colour bar on the right is added. The y-axis corresponds to x' (distance in pixels from the centre of the image) while the x-axis corresponds to θ (projection angle in degrees). The colour intensities of the different (θ, x') points in the graph corresponds to the how strong a particular line edge is. This strong line edge is often the intersection of 2 or more sinusoidal projections, which combines the votes for that edge.

Why is the Radon transform equivalent to the Hough transform for binary images?

```
>> help 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.

    R = radon(I,THETA) returns the Radon transform of the intensity image I
    for the angle THETA degrees. If THETA is a scalar, the result R is a
    column vector containing the Radon transform for THETA degrees. If
    THETA is a vector, then R is a matrix in which each column is the Radon
    transform for one of the angles in THETA. If you omit THETA, it
    defaults to 0:179.

    [R,Xp] = radon(...) returns a vector Xp containing the radial
    coordinates corresponding to each row of R.

    Class Support
    -------------
    I can be of class double, logical or of any integer class and must be
    two-dimensional. THETA is a vector of class double.  Neither of the
    inputs can be sparse.

    Remarks
    -------
    The radial coordinates returned in Xp are the values along the x-prime
    axis, which is oriented at THETA degrees counterclockwise from the
    x-axis. The origin of both axes is the center pixel of the image, which
    is defined as:

         floor((size(I)+1)/2)

    For example, in a 20-by-30 image, the center pixel is
    (10,15).
```

Hough Transform formulation:

$$x\cos\theta + y\sin\theta = \rho$$

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x^*,y^*)\ \delta(x-x^*)\ \delta(y-y^*)\ dx^*dy^* \ \Rightarrow$$

$$\check{g}(\rho,\theta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x^*,y^*)\ \delta(\rho - x^*\cos\theta - y^*\sin\theta)\ dx^*dy^*$$

Discrete Radon Transform formulation:

$$\check{g}(\rho_r,\theta_t) = \int_{-\infty}^{\infty} g(\rho_r\cos\theta_t - s\sin\theta_t, \rho_r\sin\theta_t + s\cos\theta_t)\ ds$$

$$\approx \Delta s \sum_{j=0}^{J-1} g(\rho_r\cos\theta_t - s_j\sin\theta_t, \rho_r\sin\theta_t + s_j\cos\theta_t)$$

When Radon Transform is applied on a binary image, discrete Radon Transform is performed. In the Hough and discrete Radon formulation shown above, we can see that while Hough transform takes every point in the image and determines its contribution by performing template matching specific to each (θ, ρ) line. Radon transform collects the entire image, applies template matching according to θ & ρ and does a summation. These 2 operations are essentially different interpretations to solve the same problem, hence, the Radon and Hough transform are equivalent when Radon transform is used on binary images.

(c) Location of maximum pixel intensity in Hough Image in [θ, radius] form

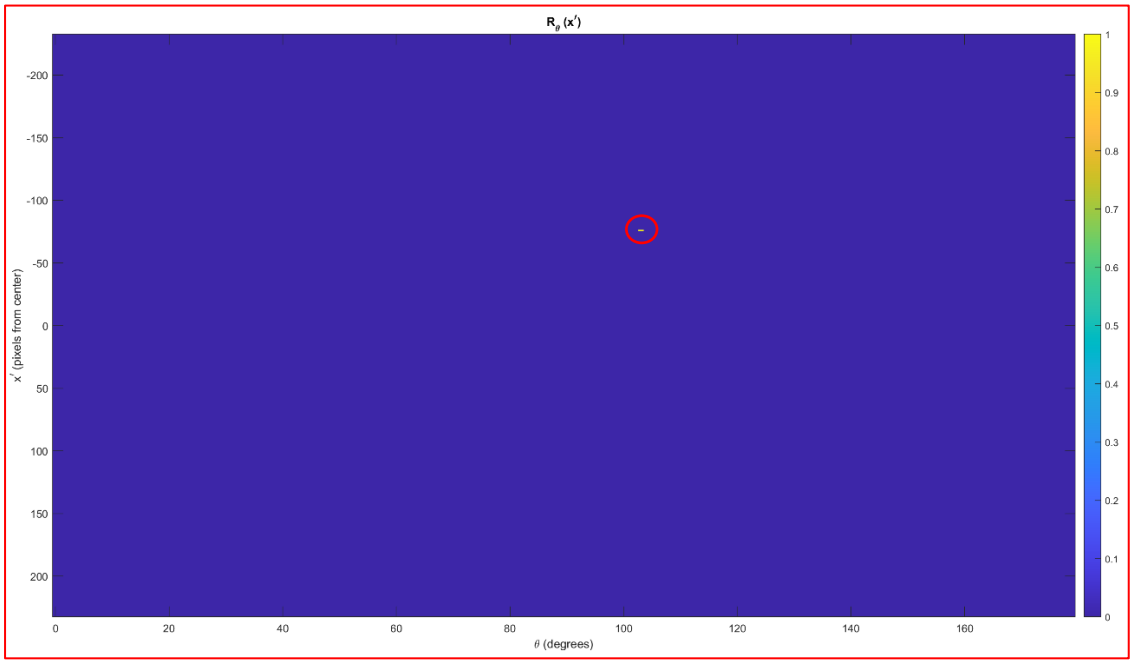
We first obtain the maxH value and find the coordinates of the maxH point by applying thresholding.

Source code:

```
maxH = max(H(:))
maxH_img = H>=int32(maxH);
figure('Name', 'maximum pixel intensity in Hough Projection', 'Color', '#D3D3D3');
imagesc(maxH_img,'Xdata',default_theta,'Ydata',xp)
title('R_{\theta} (x^{\prime})');
xlabel('\theta (degrees)'); ylabel('x^{\prime} (pixels from center)');
colormap('default'), colorbar
```

Result:

```
maxH =

  150.0116
```



In this implementation, we apply an offset of -1 to θ as the x-axis of θ starting from 0. We also apply an offset using the y-axis range, *xp* on the radius. This allows us to obtain the correct values of [θ, radius] for maxH.

Source code:

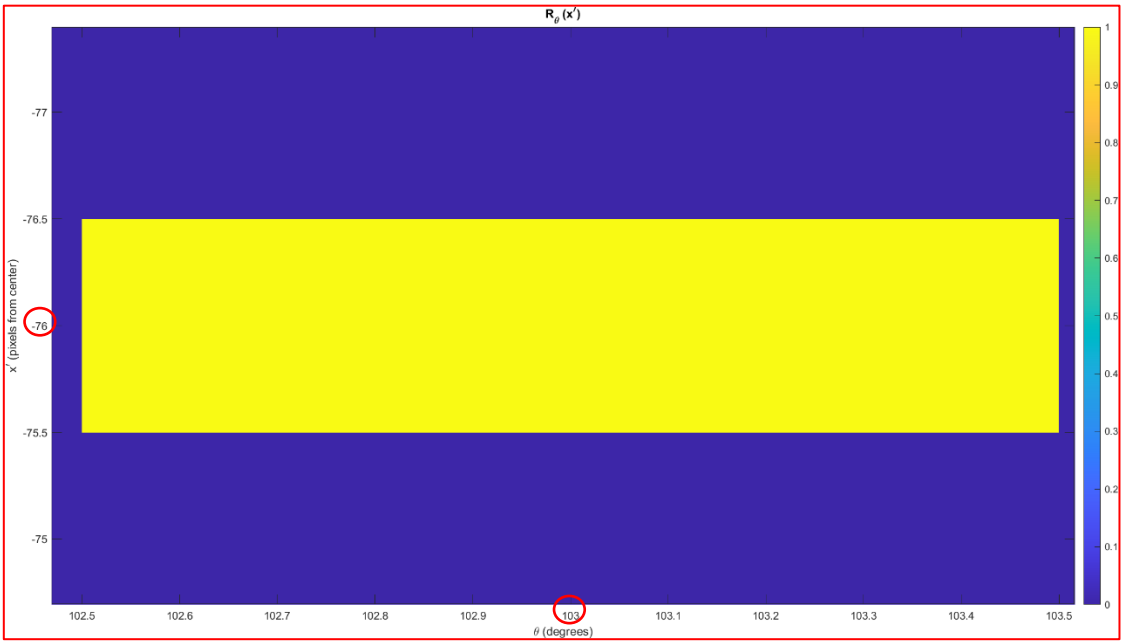
```
[radiusOfMaxWithoutOffset, thetaOfMaxWithoutOffset] = find(H == maxH)
thetaOfMax = thetaOfMaxWithoutOffset - 1
radiusOfMax = xp(radiusOfMaxWithoutOffset)
```

Result:

```
radiusOfMaxWithoutOffset =

   157


thetaOfMaxWithoutOffset =

   104
```

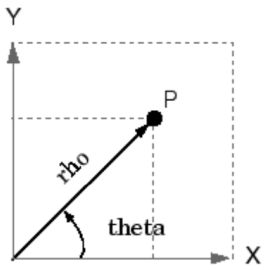
```
radiusOfMax =

   -76


thetaOfMax =

   103
```

Alternatively, we can zoom in into the image and obtain the [θ, radius] of maxH manually since we have set our x-axis and y-axis to the correct range previously. We can observe that we obtained the same value as before.



(d) Convert the [θ, radius] line representation to the normal line equation Ax + By = C representation

To convert the polar coordinates of [θ in radians, radius] to a cartesian form, we apply the pol2cart function in MATLAB where:



A = radius * cos(θ) = -76 * cos(1.7977) = 17.0963
B = radius * sin(θ) = -76 * sin(1.7977) = -74.0521

To obtain C, we need to perform a translation in both the x-axis and y-axis from the centre of the image *(179, 145)* back to the origin *(0, 0)*. After accounting for this translation, we obtain the new equation: **C = A*(A+179) + B*(B+145)**.

Image dimensions: *(358, 290)* | **Centre of image**: *(179, 145)*

```
% After accounting for offset, thetaOfMax = 103 & radiusOfMax = -76
[img_height, img_width] = size(P1a);
[A, B] = pol2cart(thetaOfMax*pi/180, radiusOfMax);
A, B = -B % B needs to be negated because the y-axis is pointing downwards for image coordinates.

C = A * (A + img_width/2) + B * (B + img_height/2)
```

```
A =                B =                C =

   17.0963            74.0521                1.9574e+04
```

The value of C is **19574**.

(e) Find value of y when x = 0 and x = width of image - 1

```
yl =        yr =

  264.3245    181.9046
```

By substituting x=0 and x=179-1=178 into **"17.0963*x + 74.0521*y = 19574"**, we obtain the 2 cartesian points **(0, 264)** and **(178, 182)** which corresponds to the most dominant line edge.

(f) Superimpose estimated line on original image

We can observe from the resulting image that the line is almost perfectly aligned with the racing track with a slight deviation nearing the furthest path of the track (Circled in green). This can be due to a few reasons:

1) The θ value is not precise enough. This could be due to the range of 0 to 179 with discrete intervals of 1 being too large and therefore not precise enough to capture the specific angle of the track path.
2) The track in this picture is slightly curved, hence the deviation. To mitigate this, a different algorithm would have to be used to detect this non-linearity.

Therefore, a range of 0 to 179 with intervals of 0.01 for θ was used to perform Radon transform. The steps are then repeated to obtain the new θ which corresponds to the highest value. The left image shows the zoomed in original result with a θ value of 103 while the right image shows the zoomed in new results with a θ value of 102.72. We can observe that the new line generated from the more precise θ value hugs the track path more closely.

However, although a more accurate result can be obtained, it is worth noting that decreasing the interval range increases computational time as the Radon transform would have to compute for more θ values.

2.3 3-Dimensional Stereo

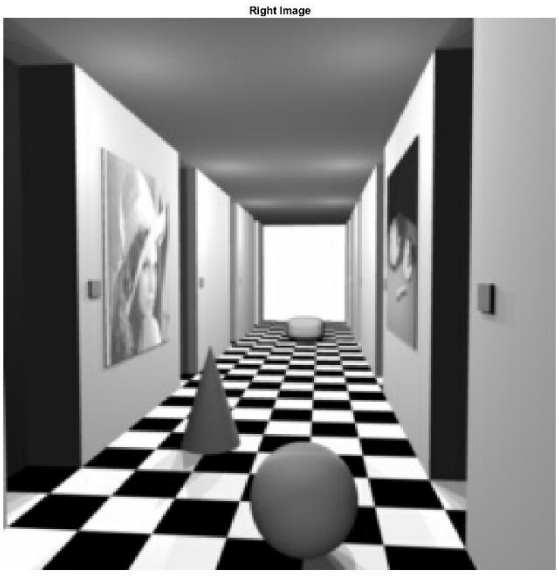
(a) Disparity map algorithm as a MATLAB function script

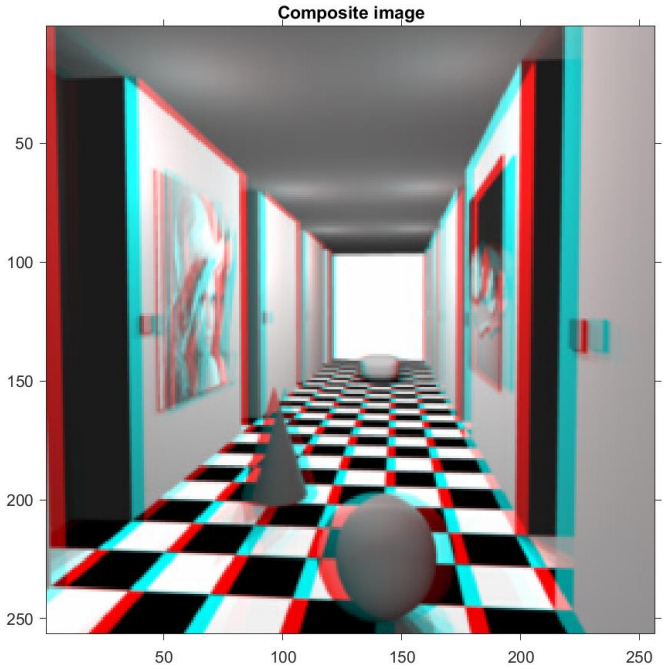
```
function [ret] = disparityMap(img_left, img_right, temp_x, temp_y)
%DISPARITYMAP Summary of this function goes here
% Description
% (i) Extract a template comprising the 11x11 neighbourhood region
%     around that pixel.
% (ii) carry out SSD matching in Pr, but only along the same scanline.
% (iii) Input the disparity into the disparity map with the same Pl pixel
%       coordinates.
    tic();

    img_left = im2double(img_left); img_right = im2double(img_right);
    [height, width] = size(img_left);
    [img_right_height, img_right_width] = size(img_right);

    % ensure both images have the same dimensions
    assert(width == img_right_width)
    assert(height == img_right_height)
    % ensure template x size == template y size
    assert(temp_x == temp_y)

    % calculate half size of template
    offset = floor(temp_x/2);
    % initialize
    searchRng = 15;
    % initialize ret
    ret = zeros(size(img_left));

    % for each row of pixels
    for (row = 1 : height)
        % set min/max row bounds for the template and blocks.
        min_row = max(1, row - offset);
        max_row = min(height, row + offset);

        % for each column of pixels
        for (col = 1 : width)
            % Set the min/max column bounds for the template.
            min_col = max(1, col - offset);
            max_col = min(width, col + offset);
            % Define search boundary limits from current (row, col)
            % 'mind' & 'maxd' is the the max displacement of pixels
            % we can search to the left and right repectively.
            left_displacement = max(-searchRng, 1 - min_col);
            right_displacement = min(searchRng, width - max_col);
            % Select the block from the right image to use as template.
            template = img_right(min_row:max_row, min_col:max_col);
            ssd_min = inf;
            smallestDiffIndex = 0;
            % Calculate the difference for each of the blocks.
            for i = left_displacement : right_displacement
                % Select a block from the left image at displacement 'i'.
                block = img_left(min_row:max_row, ...
                    (min_col + i):(max_col + i));
                % Compute the sum of squared difference (SSD)
                ssd = sum(sum((template - block).^2));
                if ssd < ssd_min
                    ssd_min = ssd;
                    smallestDiffIndex = i - left_displacement + 1;
                end
            end

            % Disparity value produced by templates matching.
            ret(row, col) = smallestDiffIndex + left_displacement - 1;
        end

        % Update progress every 10th row.
        if (mod(row, 10) == 0)
            fprintf('  Image row %d / %d done. (%.0f%%)\n', ...
                row, height, (row / height) * 100);
        end

    end
    % Display computation time.
    elapsed = toc();
    fprintf('Calculation %.2f seconds.\n', elapsed);
end
```

```
  Image row 180 / 256 done. (70%)
  Image row 190 / 256 done. (74%)
  Image row 200 / 256 done. (78%)
  Image row 210 / 256 done. (82%)
  Image row 220 / 256 done. (86%)
  Image row 230 / 256 done. (90%)
  Image row 240 / 256 done. (94%)
  Image row 250 / 256 done. (98%)
Calculation 3.29 seconds.
```

After experimenting with the naïve implementation as well as the FFT implementation, it is found that the naïve implementation took significantly faster to compute compared to the FFT implementation. On average, the FFT implementation took around 40-50 seconds while the naïve implementation took between 3-4 seconds. Therefore, we have decided to use the naïve implementation for this report.

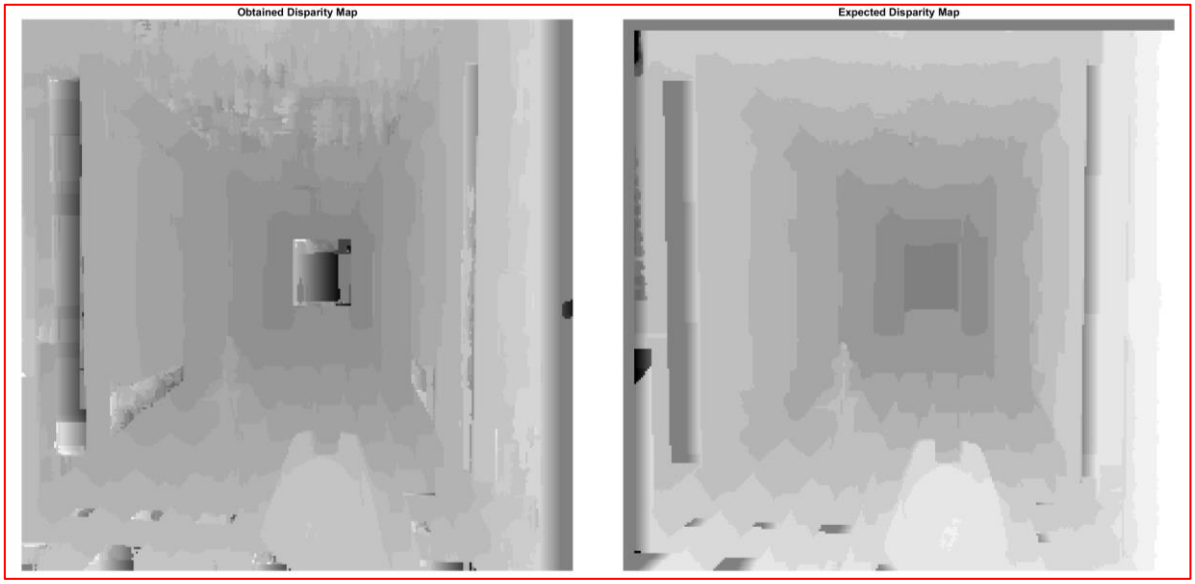
(b) Download the synthetic stereo pair images of 'corridorl.jpg' and 'corridorr.jpg'.





We are not able to observe any differences using the original left and right images. Hence, a composite image is generated to see the differences more clearly. The red channel corresponds to the left image while the blue and green channels correspond to the right image.

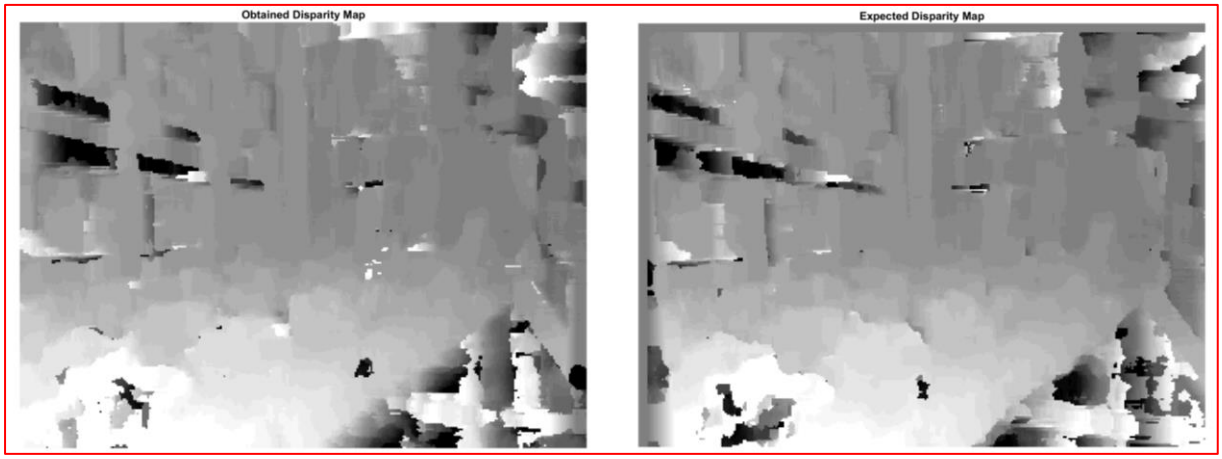
(c) Obtain corridor.jpg disparity map



The output is acceptable as it is like the expected disparity map. As expected, the pixel intensity decreases (gets darker) as we reach the centre of the image since the disparity decreases for far away objects. One area that can be improved upon is the centre of the image. Although the pixel intensities there are dark indicating low disparity which is correct, the results are not uniform across that section. This is due to the homogenous colour of that part of the image.

(d) Obtain triclops.jpg disparity map

Again, we obtain the composite image to see the differences between the two images clearly.



As we can see, the disparity map obtained is very close to the expected disparity map. However, this result is not desirable as the original building structure edge information are lost in the disparity map. To conclude, appearance-based disparity maps like the one we

have implemented have trouble calculating the correct disparity if there is a homogenous surface with the same intensity.

2.4 Spatial Pyramid (Optional)