CZ4003 COMPUTER VISION - 2020

Lab 2: Edge Detection + Hough Transform + 3D Stereo + Spatial Pyramid Matching

by

B L

U-H

Supervised by

Prof Kemao &Prof Shijian

School of Computer Science and Engineering

50, Nanyang Avenue, Singapore 639798, Tel:65 790 5488 Fax:65 792 4062

Contents

1 Edge Detection (Part 3.1a,b,c,d,e) 5
1.1 Edge Detection: Sobel (Part 3.1a,b,c,d) 5
1.1.1 Code (Part 3.1b,c) 5
1.1.2 Results (Part 3.1b, Part 3.1c) 6
1.2 Edge Detection: Canny edge detector (Part 3.1 e i ii) 9
1.2.1 Code (Part 3.1e) 9
1.2.2 Results (Part 3.1e) 9
1.2.3 Vary σ: Code (Part 3.1ei) 9
1.2.4 Vary σ: Results (Part 3.1ei) 10
1.2.5 Vary tl: Code (Part 3.1eii) 11
1.2.6 Vary tl: Results (Part 3.1eii) 11
2 Line Finding using Hough Transform (Part 3.2) 12
2.1 (Part 3.2 b) Read the help manual on Radon transform and explain why the transforms are equivalent in this case. When are they different? [2, 3, 4] 12
2.2 Radon transform Code (Part 3.2 b and c) 15
2.3 Radon transform Results (Part 3.2 b and c) 15
2.3.1 (Part 3.2 c) Find the location of the maximum pixel intensity in the Hough image in the form of [theta, radius] 15
2.4 (Part 3.2 d) Derive the equations to convert the [theta, radius] line representation to the normal line equation form *Ax + By = C in image coordinates* 16
2.4.1 (Part 3.2 d) Show that A and B can be obtained via [6] 16
2.4.2 Code: Line in Image plane (Part 3.2 d, e, f) 17
2.4.3 Results: Line in Image plane (Part 3.2 d, e, f) 17
3 3D Stereo (Part 3.2) 18
3.1 Disparity Map Function (Part 3.2a) 18
3.2 Disparity Map Function on ‘corridorl.jpg’ and ‘corridorr.jpg’ (Part 3.2b, c, d) 19
3.2.1 Code 19
3.2.2 Results (Part 3.2 c) 19
3.2.3 (Part 3.2 c) Comment on how the quality of the disparities computed varies with the corresponding local image structure. 20
3.3 Rerun your algorithm on the real images of ‘triclops-i2l.jpg’ and triclops-i2r.jpg’. 20

3.3.1 Results (Part 3.2 d) 20
3.3.2 How does the image structure of the stereo images affect the accuracy of the estimated disparities? 21
3.4 Optional Bag of Words and Spatial Pyramid Matching 21
3.4.1 Summary 21
3.4.2 Algorithm 21
3.4.3 CODE 22
3.4.4 Results Bag of SIFT Representation + (KNN) Nearest Neighbor Classifier 28
3.4.5 Results Bag of SIFT Representation + one-vs-all SVMs 29
3.4.6 Results Spatial Pyramid Matching (SPM) Scheme one-vs-all SVMs ...... **Error! Bookmark not defined.**
4 Appendix 31
4.1 OTSU_B.m 31
5 References 31

1 Edge Detection (Part 3.1a,b,c,d,e)

1.1 Edge Detection: Sobel (Part 3.1a,b,c,d)

Sobel filter is an edge detector that works by determining the gradient ("the jump") of image intensity. Sobel filter edge detector works because edges typically has the largest increase from light to dark.

Sobel filter works by using a vertical and horizontal edge detector:

Vertical Gradient Filter (detect Horizontal edge)	Horizontal Gradient Filter (detect Verticle edge)
$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
Results: G_y =Magnitude of Verticle Gradient	Results: G_x =Magnitude of Horizontal Gradient
Resultant Gradient Magnitude	
$G = \sqrt{{G_y}^2 + {G_x}^2}$	
Resultant Gradient Direction	
$\theta = \tan^{-1}(\frac{G_y}{G_x})$	

After convolution with the filter separately, we will have a map of the magnitudes of verticle gradient (from Vertical Edge Filter) and horizontal gradient (from Horizontal Edge Filter). The resultant gradient magnitude and orientation of the gradient can be determined as shown in the formula above. [1]

1.1.1 Code (Part 3.1b,c)

```
% 3.1 b)
%Sobel Verticle and Horizontal Gradient Filter
s_h=[-1 0 1;
     -2 0 2;
     -1 0 1;]
s_v=[-1 -2 -1;
     0 0 0;
     1 2 1;]
%Verticle Filter
Pv=conv2(P,s_v);
%Horizontal Filer
Ph=conv2(P,s_h);

% 3.1 c)
%Magnitude of Gradient Map
P2=(Pv.^2.+Ph.^2).^0.5
```

1.1.2 Results (Part 3.1b, Part 3.1c)

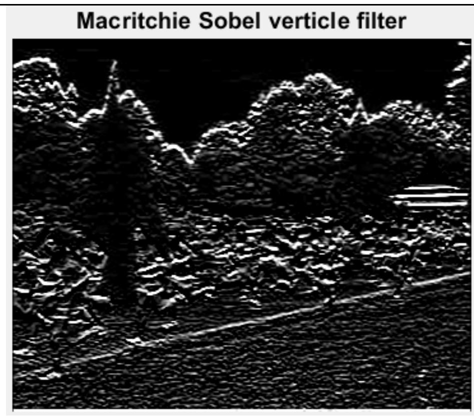


Figure 1: Convolution with Vertical Filer (Part 3.1b)

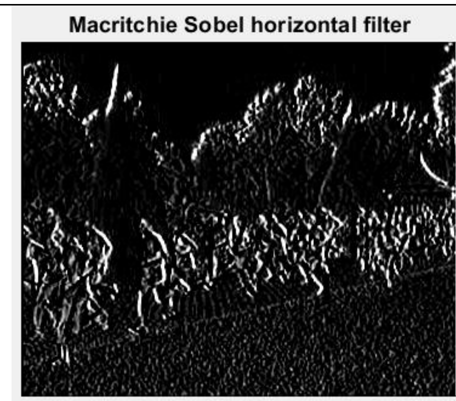


Figure 2:Convolution with Horizontal Filer (Part 3.1b)

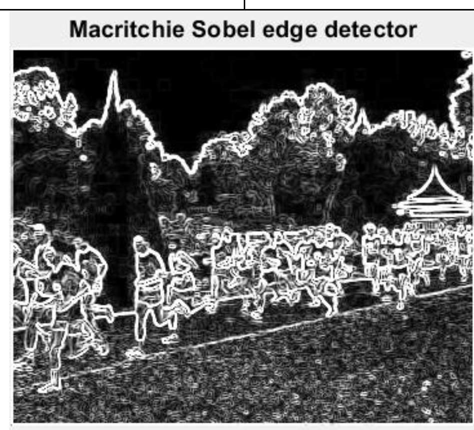


Figure 3: Map of the Magnitude of the Resultant Gradient

1.1.2.1 (Part 3.1b) What happens to edges which are not strictly vertical nor horizontal, i.e. diagonal?

After convolution with the **verticle gradient filter**, the image is a map of the magnitudes of verticle gradient (Figure 1).

Horizontal lines appear brighter while perfectly vertical lines are dark. The intensity of the **resultant verticle gradient map** changes from brightest when the line is horizontal to darkest when the line is verticle or if there is no change intensity of the original map. Edges that are diagonal appear less bright than a horizontal line.

After convolution with the **horizontal gradient filter**, the image is a map of the magnitudes of horizontal gradient (Figure 2).

Vertical lines appear brighter while perfectly horizontal lines are dark. The intensity of the **resultant verticle gradient map** changes from brightest when the line is Vertical to darkest when the line is horizontal or if there is no change intensity of the original map. Edges that are diagonal appear less bright than a Verticle line.

In the final resultant gradient magnitude map (Figure 3), all edges are detected both vertical and horizontal. Edges that are diagonal are also detected.

1.1.2.2 (Part 3.1c) Suggest a reason why a squaring operation is carried out.

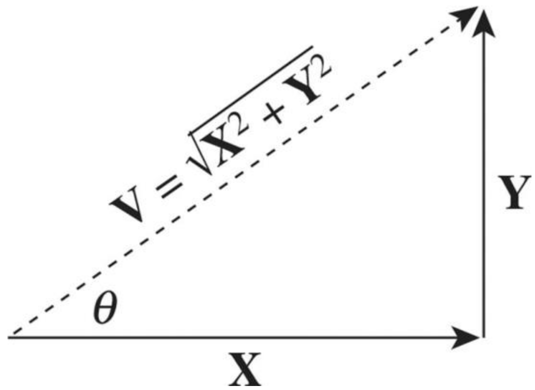


Figure 4: Resultant Vectors derived from horizontal and vertical components

The Squaring operation is carried and summed to obtain the resultant magnitude of the gradient. Because convolution with the **horizontal gradient filter** give the horizontal gradient vector. Convolution with the **verticle gradient filter** gives the verticle gradient vector. Therefore, we can use Pythagorean theorem to determine the resultant gradient (Figure 4).

1.1.2.3 (Part 3.1d) Vary Threshold: Try different threshold values and display the binary edge images.

1.1.2.3.1 Code

<table>
<tr><td>

```
% 3.1 d) Thresholding
min_P2=double(min(P2(:)))%Min intensity=13
max_P2=double(max(P2(:)))%Max intensity=204

%contrast stretching
P2C = (double(P2(:,:))-min_P2).*(255/(max_P2-min_P2));
%checking min max of P2
min(P2C(:)), max(P2C(:)) %0, 255
```

</td><td>Contrast stretch to make sure resultant gradient magnitude map is between 0 to 255</td></tr>
<tr><td>

```
t=OTSU_B(P2C,true);
P2t=P2C>t;
[count,bins]=imhist(uint8(P2C),256);
```

</td><td>Use OTSU to determine optimal threshold with minimal intra class variance/ maximum interclass variance
Refer to Appendix OTSU_B.m.</td></tr>
<tr><td>

```
figure( 'Position', [10 10 900 600]);
colormap('gray');imshow(P2t);title("Macritchie Sobel edge
detector")
hold off;

t_varry=linspace(10,220,8)
t_varry=[t_varry(1:2),t,t_varry(3:7)]
figure( 'Position', [10 10 1440 960]);
for i = 1 : size(t_varry,2)
    P2t=P2C>t_varry(i);
    hold on

subplot(2,size(sigma_varry,2)/2,i);colormap('gray');imshow(P2
t);
    if i==3
        title({"Sobel" "t= " t_varry(i),"OTSU"})
    else
        title(["Sobel" "t= " t_varry(i)])
    end
end
hold off
```

</td><td>Plot out resulting sober filter after using different level of thresholding for binarization</td></tr>
</table>

1.1.2.3.2 Results

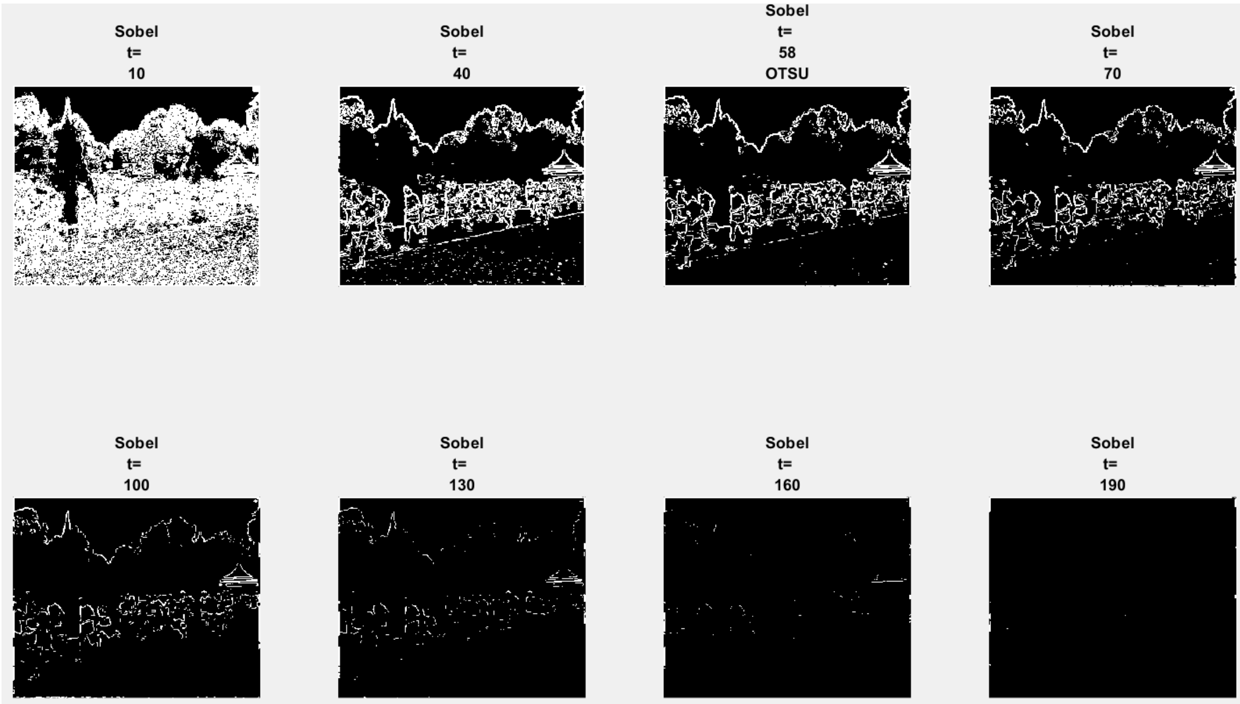


Figure 5: Sobel with different threshold

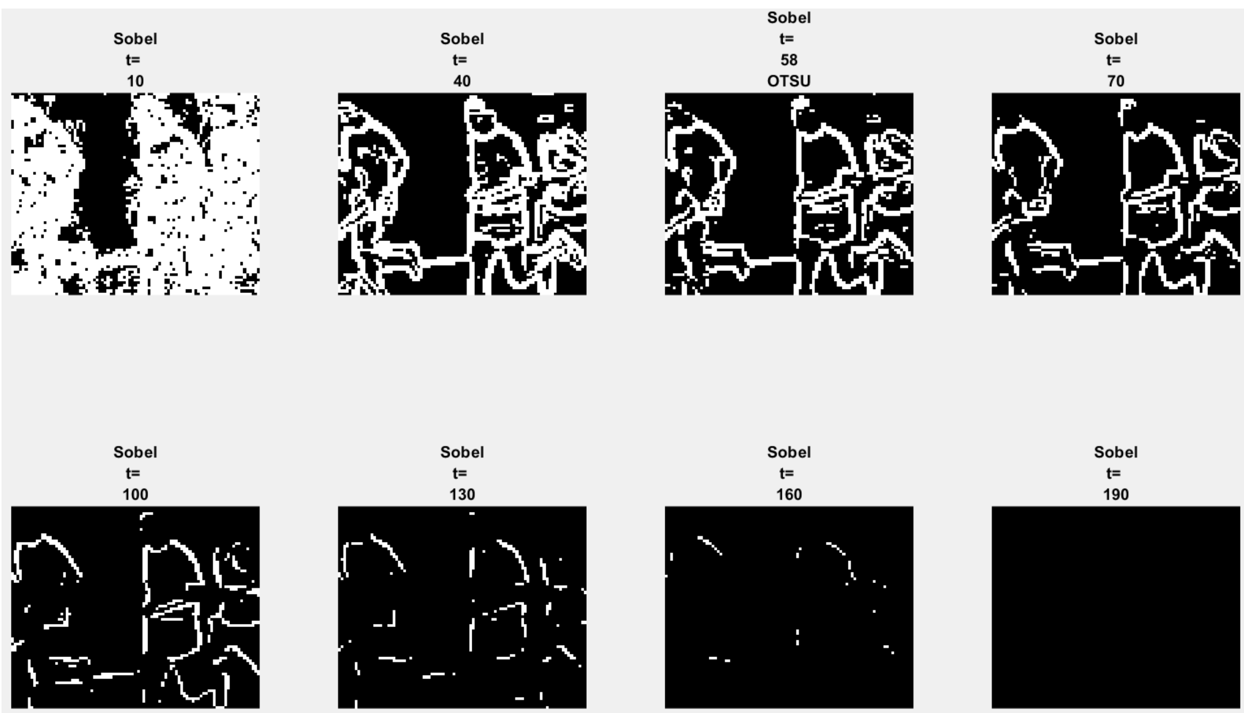


Figure 6: Sobel with different threshold zoom in

1.1.2.4 (Part 3.1 d)What are the advantages and disadvantages of using different thresholds?

Thresholding will help us to binarize the gradient map and help us identify likely edges. At lower threshold we obtain thicker lines (Figure 5, t=10)(Figure 6, t=10) while at higher thresholds we get thinner lines, and more lines is not detected(Figure 5, t=190)(Figure 6, t=190). To get the 'optimal' thresholding OTSU thresholding(Figure 5, t=58)(Figure 6, t=58). is implemented as a function

1.2 Edge Detection: Canny edge detector (Part 3.1 e i ii)

1.2.1 Code (Part 3.1e)

```
% 3.1 e)
tl=0.04, th=0.1, sigma=1.0;

E = edge(P,'canny',[tl th],sigma);
figure;colormap('gray');imshow(E);title("Macritchie Sobel edge detector")
```

1.2.2 Results (Part 3.1e)

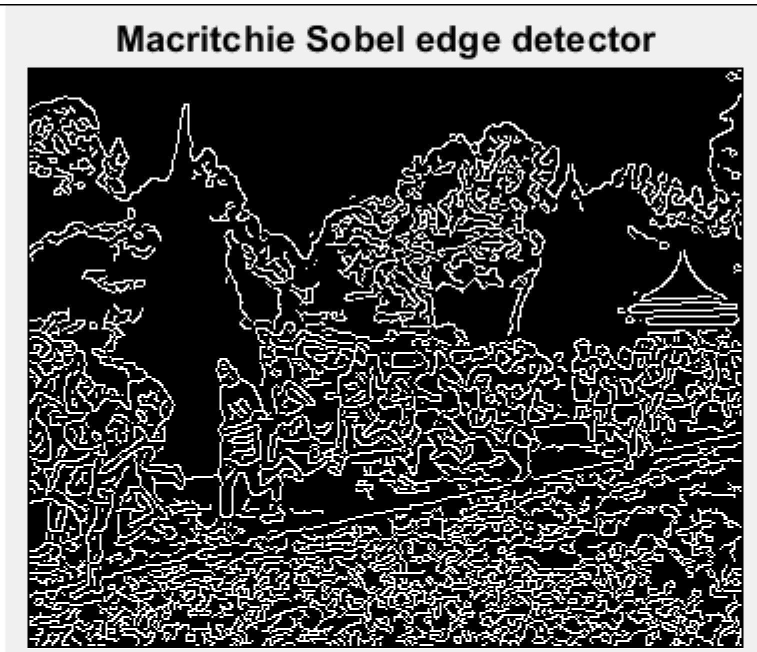


Figure 7: Canny edge Detector $tl = 0.04, th = 0.1, \sigma = 1.0$;

1.2.3 Vary σ: Code (Part 3.1ei)

```
% 3.1 e)i) Varry Sigma
sigma_varry=linspace(1,5,5);
figure( 'Position', [10 10 1440 960]);
for i = 1 : size(sigma_varry,2)
    E = edge(P,'canny',[tl th],sigma_varry(i));
    hold on
    subplot(2,3,i);colormap('gray');imshow(E);title(["Canny" "\sigma= " sigma_varry(i)])
end
subplot(2,3,6);colormap('gray');imshow(P);title(["Original"])
hold off
```

1.2.4 Vary σ: Results (Part 3.1ei)

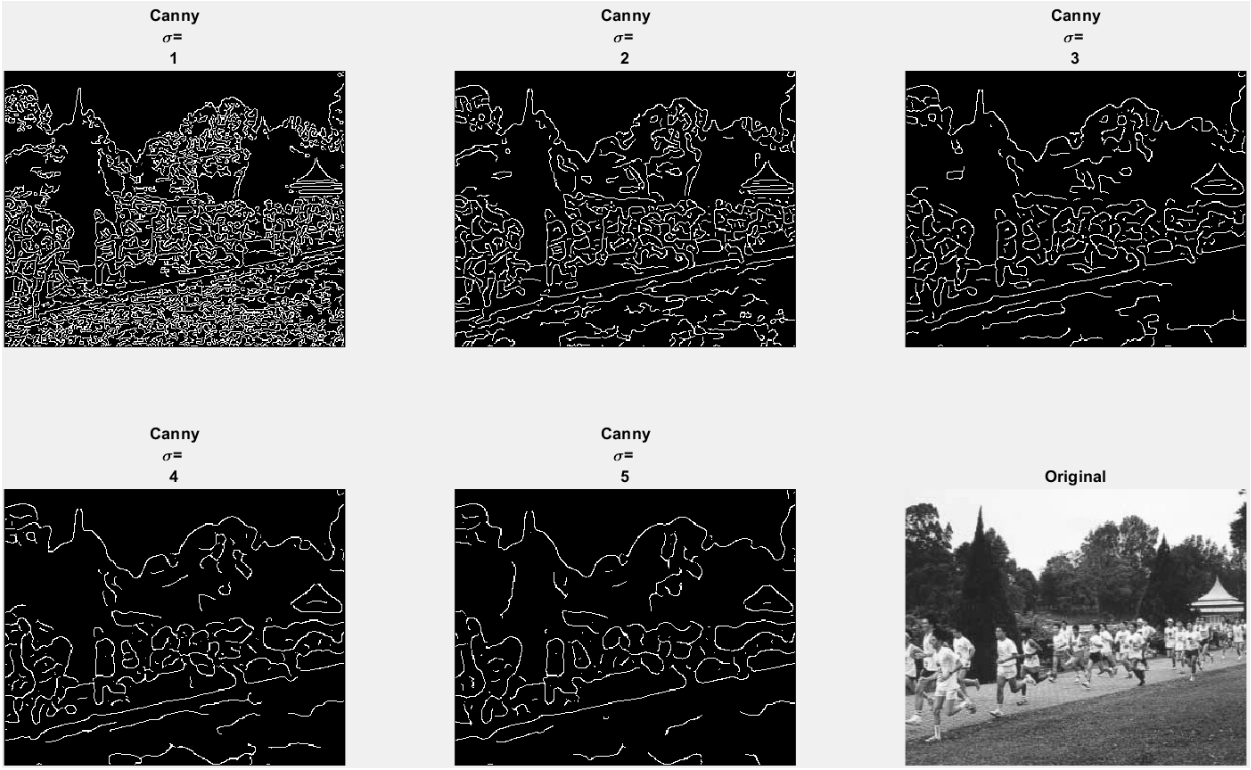


Figure 8: Canny Edge Detector varying σ

1.2.4.1 (Part 3.1 e i) What do you see and can you give an explanation for why this occurs? Discuss how different sigma are suitable for (a) noisy edgel removal, and (b) location accuracy of edgels.

As σ increases from 1 to 5 the sensitivity towards the edgel decreases. In other words, less edgel is detected. In terms of noise, less noisy edgel is detected. The location accuracy of the edgel also decreases. The reason for this is because gaussian edge filtering is able to remove noise. As, σ increases more **noise is removed(less noisy edgel is detected)** but it also means more details such as **edges is lost (location accuracy of the edgel also decreases)**.

If the original image is noisy and a lot of noisy edgel is detected, a larger sigma should be used to remove noisy edgel.
If the original image is not noisy and we want to detect more edges with higher location accuracy, a σ of 1 should be sufficient.

1.2.5 Vary tl: Code (Part 3.1eii)

```
% 3.1 e)i)Varry tl
tl_varry=linspace(0.01,0.09,8);
figure( 'Position', [10 10 1440 960]);
for i = 1 : size(tl_varry,2)
    E = edge(P,'canny',[tl_varry(i) th],sigma);
    hold on
    subplot(2,size(tl_varry,2)/2,i);colormap('gray');imshow(E);title(["Canny" "th= "
tl_varry(i)])
end
hold off
```

1.2.6 Vary tl: Results (Part 3.1eii)

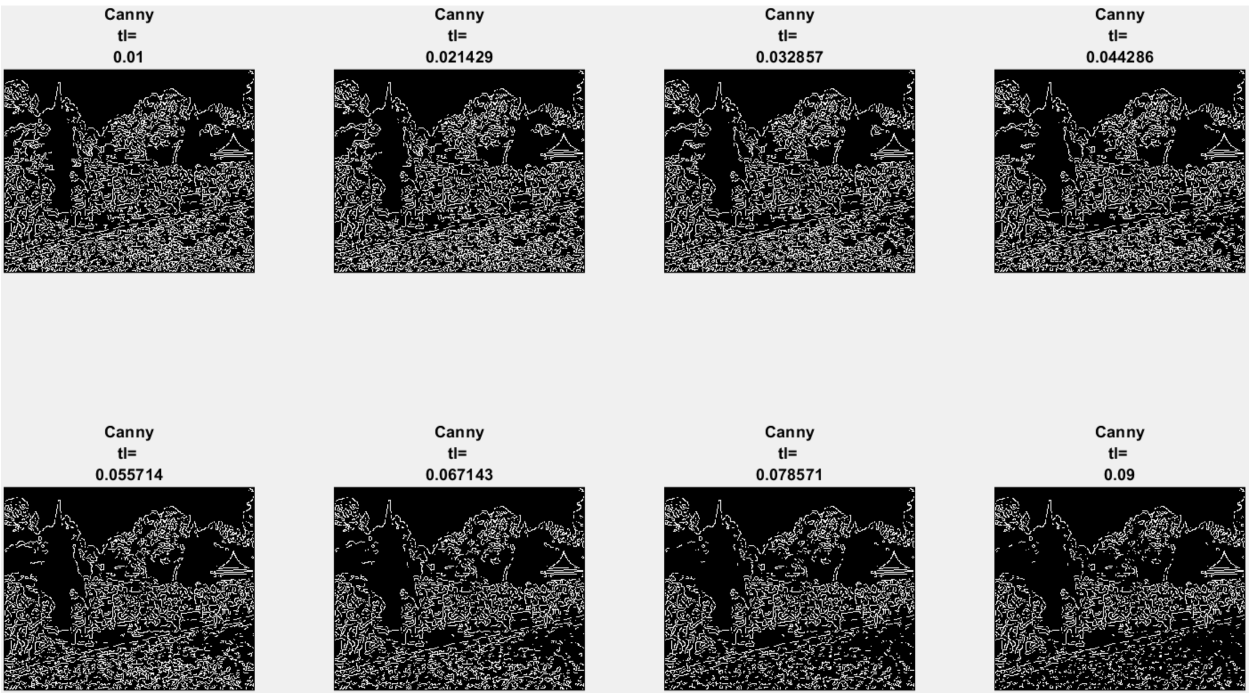


Figure 9: Canny Edge Detector varying tl

1.2.6.1 (Part 3.1 e ii) Try raising and lowering the value of tl. What does this do? How does this relate to your knowledge of the Canny algorithm?

Raising the tl decreases the number of edges detected (Figure 9)

In Canny's edge detector, hysteresis thresholding is employed.

$$f_t = \begin{cases} 1, & f > t_H \\ 1, & t_L < f < t_H \ and \ Condition \\ 0, & f < t_L \end{cases}$$

- If a pixel is bellow tl, the pixel is set to 0.
- If a pixel is above tl and bellow th, the pixel is set to 1 if (Condition) its neighboring pixel perpendicular to the edge gradient have already been set to 1 .
- If a pixel is above th, the pixel is set to 0.

Therefore, increasing tl increases the threshold for edge detection and thus lesser edge is detected and lesser noisy edge is detected as well.

2 Line Finding using Hough Transform (Part 3.2)

2.1 (Part 3.2 b) Read the help manual on Radon transform and explain why the transforms are equivalent in this case. When are they different? [2, 3, 4]

Radon Transform and Hough transform are mathematically similar

<table>
<tr>
<td>

Radon Transform

The Radon Projection is computed for a set of θ angle. “In general, the Radon transform of *f(x,y)* is the line integral of *f* parallel to the *y´*-axis. ” [2]

$$R\theta(x') = \int_{\infty}^{\infty} f(x'cos\theta - y'sin\theta, x'sin\theta + y'cos\theta)dy'$$

$$where, \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

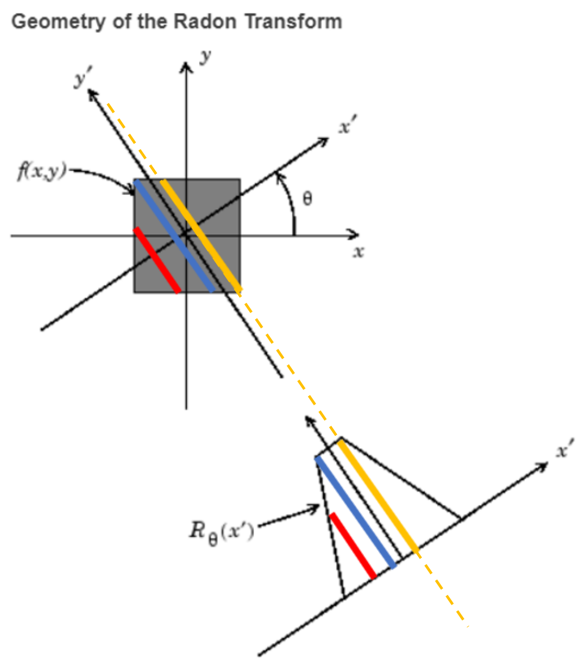


Figure 10: Geometry for Radon Transform

In other words, Radon Projection is like the **number of votes for a specific line represented in the parameter space in Hough Transform**.

“The Radon transform of an image is the sum of the Radon transforms of each individual pixel.” [2]

The magnitude of radon projection $R\theta(x')$ tell us the number of points that lie on the line perpendicular to x’

For instance, from Figure 10, “yellow amount” of points lie on the yellow dotted line which is a line perpendicular to x’.

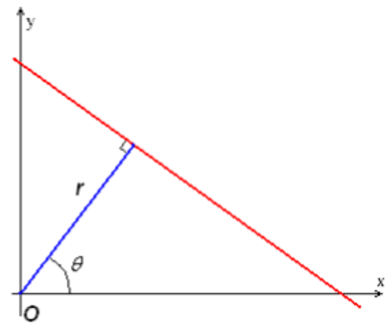
It is important to note that

</td>
<td>

Hough transform

$$\rho = xcos\theta + ysin\theta$$

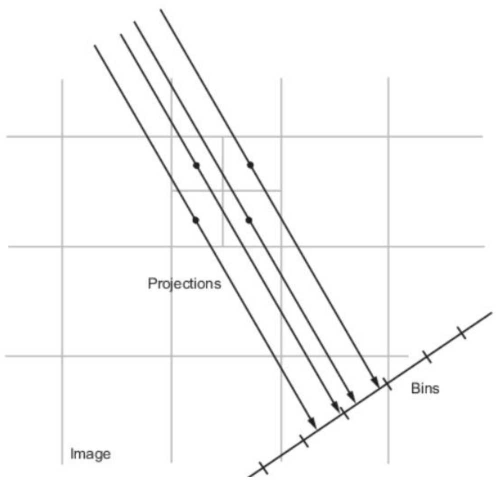
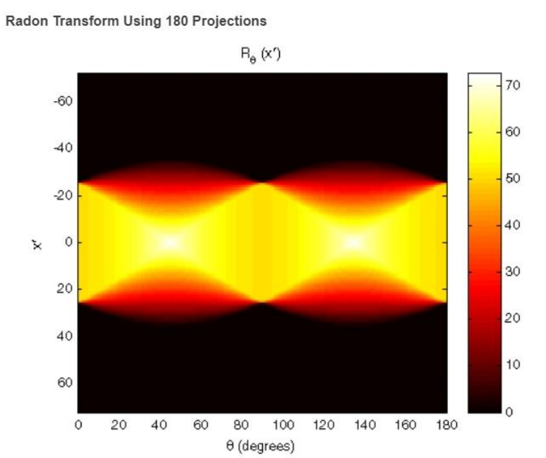
“*where* ρ *is the distance from the origin to the closest point on the straight line, and* θ *(theta) is the angle between the* x *axis and the line connecting the origin with that closest point*.” [5]



The fundamental idea is that any point (x, y) can be described using a set of (ρ, θ) that describes a set of lines that intersects the point (x, y).

“*Given a single point in the plane, then* ***the set of all straight lines going through that point corresponds to a sinusoidal curve in the*** $(\boldsymbol{\rho}, \boldsymbol{\theta})$ ***plane****, which is unique to that point. A set of two or more points that form a straight line will produce sinusoids which cross at that specific* (ρ, θ) *for that line. Thus, the problem of detecting collinear points can be converted to the problem of finding concurrent curves*.” [5]

</td>
</tr>
</table>

θ *correspond to* θ *in Hough transform* x' *correspond to* ρ *in Hough transform* $R\theta(x')$ *correspond to* **number of votes for a specific line represented in the parameter space** *in Hough transform*	
Algorithm ***Step 1:*** The Radon transform is computed for a set of θ angle. **For each θ angle,** the Radon transform of an image is the sum of the Radon transforms of each individual pixel.  The algorithm first divides pixels in the image into four subpixels and projects each subpixel separately, as shown in the following figure.  *Figure 11: Radon projection magnitude map in a theta and x' array*	Algorithm *The linear Hough transform algorithm uses a two-dimensional array, called an accumulator, to detect the existence of a line described by* $\rho = x\cos\theta + y\sin\theta$ *The dimension of the accumulator equals the two quantized values of* ρ *and* θ *in the pair* (ρ, θ). ***Step 1: For each pixel** at (x,y) the Hough Algorithm determines if there is enough evidence of a straight line at that pixel.* ***Step 2:** If there is a line, it will calculate the parameters* (ρ, θ) *of lines that pass through that pixel and then increment the value of that corresponding bin in the accumulator* ***Step 3:** Highest Values in the accumulator bin likely indicates a line can be extracted. The (approximate) geometric definition of that line in the x,y space can be derived from* (ρ, θ) *and* $\rho = x\cos\theta + y\sin\theta$ *Due to **imperfection errors** in the edge detection step **(Step 2),** there will usually be errors in the accumulator space, which may make it not easy to find the appropriate peaks, and thus the appropriate lines.* *"The final result of the linear Hough transform is a two-dimensional array (matrix) similar to the accumulator—one dimension of this matrix is the quantized angle* θ *and the other dimension is the quantized distance* ρ. *Each element of the matrix has a value equal to the sum of the points or pixels that*

Step 2: Radon projection magnitude can be map in a theta and x' array [2]	*are positioned on the line represented by quantized parameters* (ρ, θ) *. So the element with the highest value indicates the straight line that is most represented in the input image."* [5]

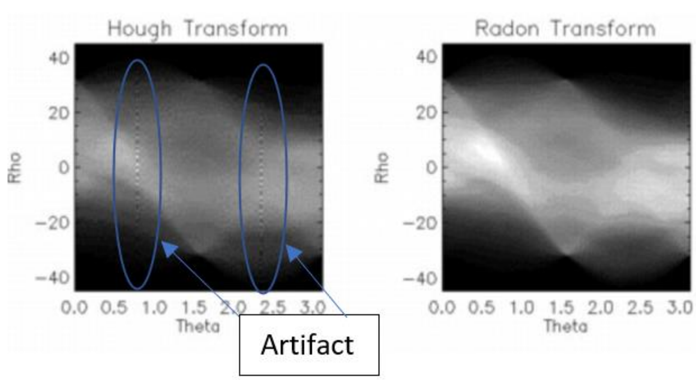
From this we see that magnitude in the parameter space is derived continuously (implemented discretely) using integration for Radon while, magnitude in the parameter space is derived discretely when using Hough Transform.

The difference in the Algorithm approach of mapping from the image space to a parameter space of ρ and θ is in their point of view.

- While the Radon transform derives the magnitude of a point in parameter space from image space (the *reading paradigm*) by using the radon projection transform on the image space,
- the Hough transform explicitly maps data points from image space to parameter space (the *writing paradigm*) when a line is detected. Magnitude of a point in a parameter space is derived by the accumulation of mapped points.

"Because of this, it is clear by comparing their (discrete) algorithms. For each θ in parameter space

- Radon projects the image points on a line described by (its angle) θ using buckets of size Δρ.
- Hough on the other hand takes each image point (x,y) and adds the appropriate intensity to all corresponding parameter space bins.



As a result Hough suffers artifacts whereas Radon allows for high resolution in parameter space (by adjusting Δθ and Δρ and dividing pixels into subpixels)." [4]

Therefore, in this case the transform is essentially equivilant because X_p is discretized by a unit of 1. They will differ slightly if X_p is discretized to smaller units.

(Note at the background, X_p is discretized to smaller units but the radon Transform is presented with X_p discretized to a unit of 1).

2.2 Radon transform Code (Part 3.2 b and c)

```
%3.2 b
theta = 0:180;
[H, xp] = radon(E,theta);

figure( 'Position', [10 10 900 350]);
subplot(1,2,1);imagesc(theta,xp,H),colormap(gca), colorbar;
title('R_{\theta} (X\prime)');
xlabel('\theta (degrees)')
ylabel('x''')
hold on;

%3.2 c
H_max = max(H,[],'all')
[y,x]=find(H == H_max)
hold off;

zoom=3
subplot(1,2,2);imagesc(theta(x-zoom:x+zoom),xp(y-zoom:y+zoom),H(y-zoom:y+zoom,x-zoom:x+zoom))
colormap(gca), colorbar
title({'Max R_{\theta} (X\prime) =' num2str(H_max),strcat("@ \theta= ", num2str(theta(x)), "
and xp= ", num2str(xp(y)))});
xlabel('\theta (degrees)')
ylabel('x''')
hold on;
plot(theta(x),xp(y),'o',...
             'MarkerEdgeColor','red',...
             'MarkerFaceColor',[1 .6 .6])
```

2.3 Radon transform Results (Part 3.2 b and c)

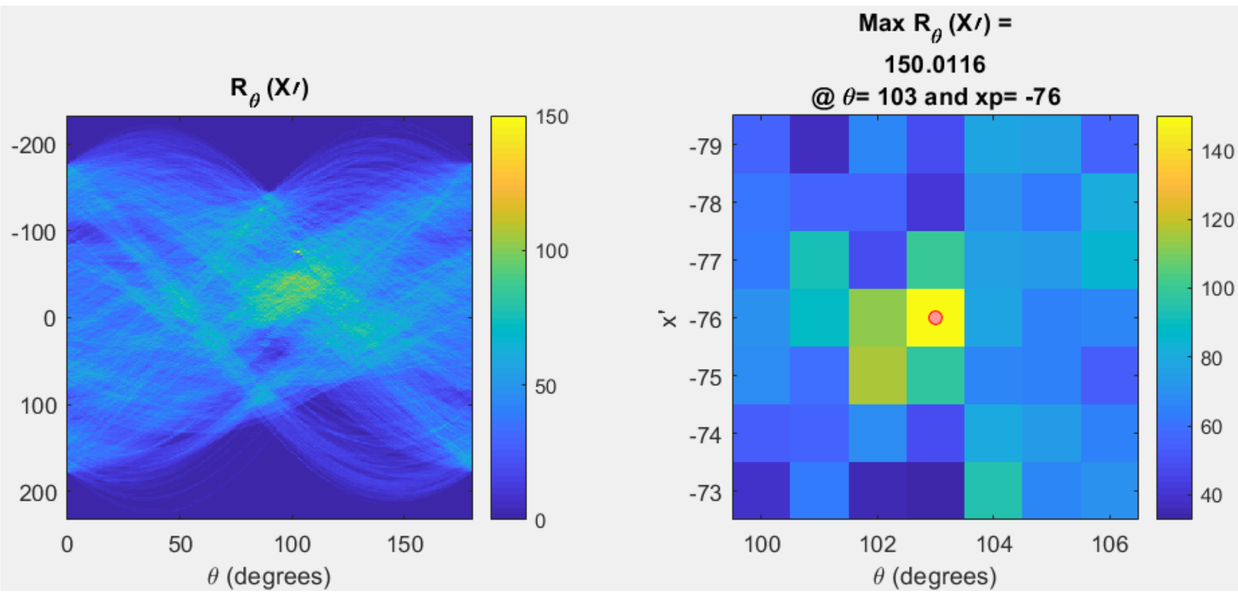


Figure 12: Results from Radon Transform (left) max intensity (right)

2.3.1 (Part 3.2 c) Find the location of the maximum pixel intensity in the Hough image in the form of [theta, radius].

From the graph Figure 12, the maximum pixel intensity is 150.0116 at $\theta = 103\ and\ xp = -76$.

2.4 (Part 3.2 d) Derive the equations to convert the [theta, radius] line representation to the normal line equation form *Ax + By = C in image coordinates*.

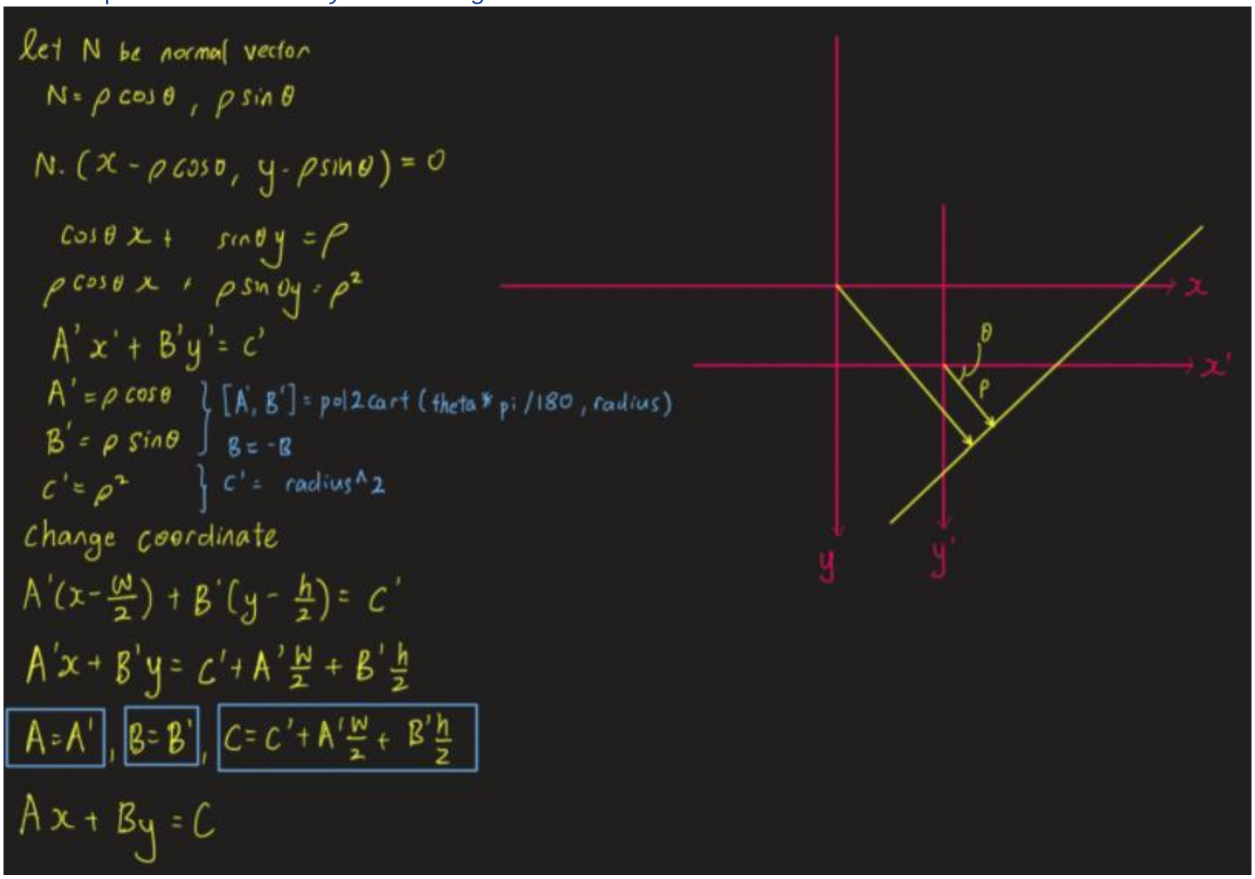
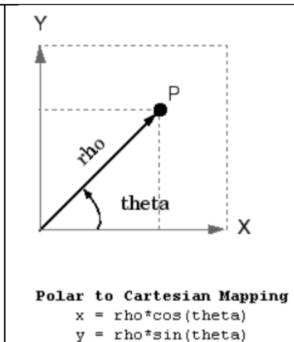


Figure 13: Derive the equations to convert the [theta, radius] line representation to the normal line equation form Ax + By = C in image coordinates.

2.4.1 (Part 3.2 d) Show that A and B can be obtained via [6]

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



Since pol2cart(theta, radius) converts polar coordinates (ρ, θ) to cartesian coordinate(x, y) and it is shown in Figure 13 that A and B is the vector coordinates of P, therefore A and B can be obtained via the code above.

2.4.2 Code: Line in Image plane (Part 3.2 d, e, f)

```
[A, B] = pol2cart(theta(x)*pi/180, xp(y));
B=-B;
C=xp(y)^2+A*size(P,2)/2+B*size(P,1)/2;
y_line=@(x) -A/B.*x+C/B;
xl = 0,xr = size(P,2) - 1;
yl=y_line(xl)
yr=y_line(xr)

figure;colormap('gray');imshow(uint8(P));title("Macritchie (Finding Line)")
line([xl xr], [yl yr],'LineWidth',2);
```

2.4.3 Results: Line in Image plane (Part 3.2 d, e, f)

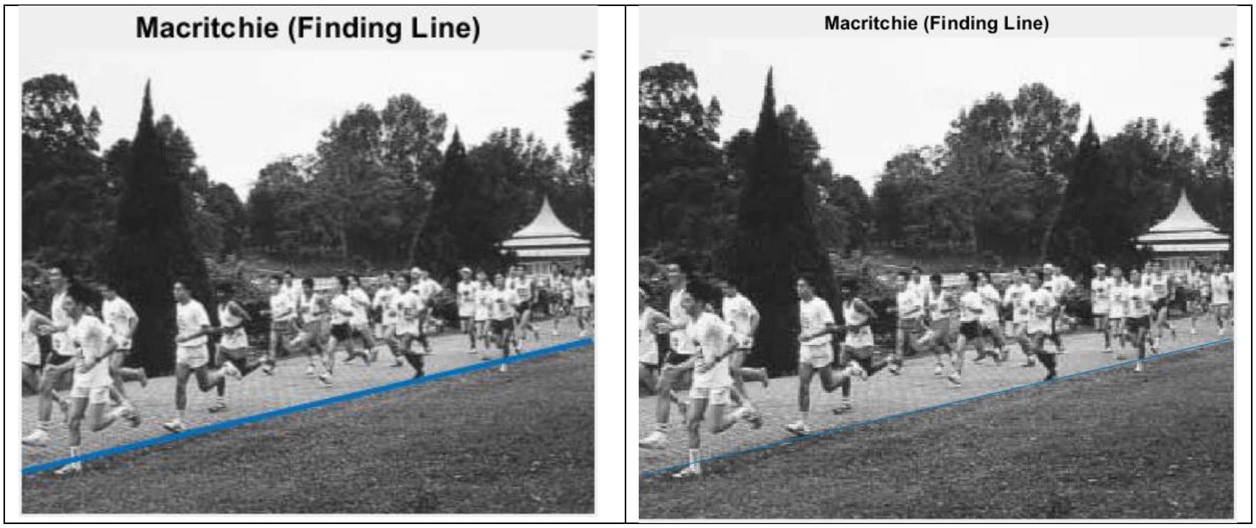


Figure 14: Line in Image Plane

2.4.3.1 *(Part 3.2 f) Results: Does the line match up with the edge of the running path? What are, if any, sources of errors? Can you suggest ways of improving the estimation?*

Yes. it matches up almost perfectly with the edge of the running path. But on closer inspection there is some deviation from the line on the right-hand side of the Image. This is likely because the running path is not a perfect straight line and therefore does will not match up with the perfect line.

The **possible sources of error** and suggested **improvements** are

1. The use of Radon to identify straight line introduced error in the detection of the straight line and this can be avoided by using a Generalized Hough Transform that can be used to detect curves
2. Another source of error may come from the discretization of the X_p. This can be improved by using Radon transform with Xp discretized to smaller units rather than Hough transform.

3 3D Stereo (Part 3.2)

3.1 Disparity Map Function (Part 3.2a)

$$S(x,y)=\sum_{j=0}^{M}\sum_{k=0}^{N}I^2(x+j,y+k).1+\sum_{j=0}^{M}\sum_{k=0}^{N}T^2(j,k)-2\sum_{j=0}^{M}\sum_{k=0}^{N}I(x+j,y+k)T(j,k)$$

```
function map = disparity_map_Barn(Pl, Pr, tempx,tempy)

[h, w] = size(Pl);
% Empty Map
map = ones(h, w);

%Half the size of template
half_tx=floor(tempx/2);
half_ty=floor(tempy/2);

max_disparity=20;

for row = half_tx+1:h-half_tx
    for xl = half_ty+1:w-half_ty
        %Template
        T = Pl(row-half_ty:row+half_ty,xl-half_tx:xl+half_tx);
        %Since right point image will be more left than left point image
        left = xl-max_disparity;
        right = xl;
        if left<half_tx+1
            left = half_tx+1;
        end
        ssd_min = Inf;
        xr_min = left;
        for xr = left:right
            I = Pr(row-half_ty:row+half_ty,xr-half_tx:xr+half_tx);
            ssd=sum(double(I).*double(I),"All")-2*sum(double(I).*double(T),"All");
            if ssd<ssd_min
                ssd_min=ssd;
                xr_min = xr;
            end
        end
        d = xl - xr_min;
        map(row, xl) = -d;
    end
end
map=map(half_tx+1:h-half_tx, half_ty+1:w-half_ty);
end
```

Function parameter

- Pl: gray scale left stereo image
- Pr: gray scale right stereo image
- tempx: width of template (left image)
- tempy: height of template (right image)

Avoided additional computation from fft and conv2 by using sum(, “All”) and element wise multiplication. Didn’t compute middle term because the middle term is a constant.

3.2 Disparity Map Function on ‘corridorl.jpg’ and ‘corridorr.jpg’ (Part 3.2b, c, d)

3.2.1 Code

```
%3.3 3D stereo
clear all;
%3.3 b
Pl = rgb2gray(imread('resource\corridorl.jpg'));
Pr = rgb2gray(imread('resource\corridorr.jpg'));
temp_size=11;
half=floor(temp_size/2);
whos Pl
whos Pr
figure;
subplot(1,2,1);colormap('gray');imshow(uint8(Pl));title("corridor left")
subplot(1,2,2);colormap('gray');imshow(uint8(Pr));title("corridor right")

%3.3 c
map=disparity_map_Barn(Pl,Pr,11,11);
figure;imshow(-map,[-15 15]);title({"Disparity Map","Max D=15"})
```

3.2.2 Results (Part 3.2 c)

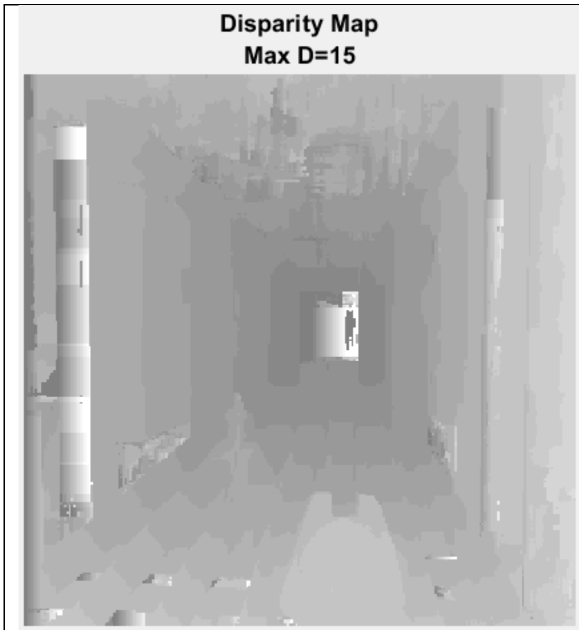


Figure 15: Disparity Map

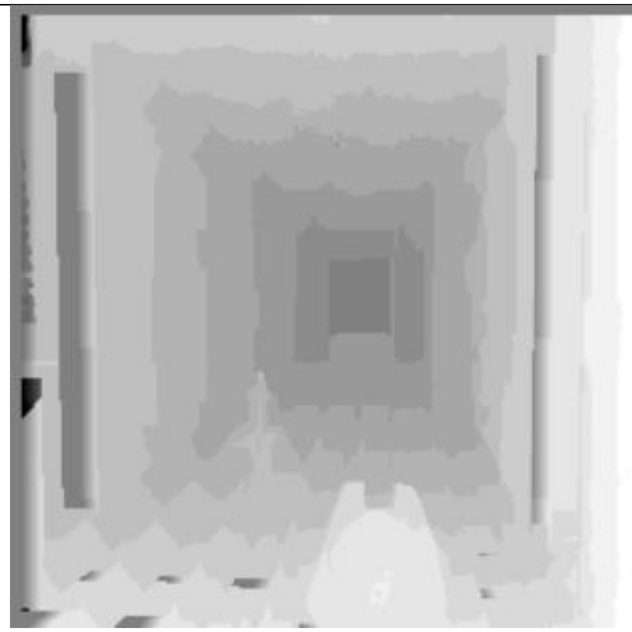


Figure 16: corridor_disp.jpg

Figure 17: corridorl.jpg

3.2.3 (Part 3.2 c) Comment on how the quality of the disparities computed varies with the corresponding local image structure.

Generally, as we move further away from the corridor, the disparity map becomes darker. This corresponds to reality because the distance is further away from the lens and thus disparity is smaller. The cone and sphere can be identified with the sphere being lighter and having a larger disparity than the cone. For the furthest most wall, because it seems the most homogeneous with even lighting, there seems to be some error in the where there is large disparity detected. The disparity map derived from SSD works best when there are more details or texture on the object.

3.3 Rerun your algorithm on the real images of 'triclops-i2l.jpg' and triclops-i2r.jpg'.

3.3.1 Results (Part 3.2 d)

Figure 18: Disparity Map

Figure 19: triclopsid.jpg

Figure 20: triclopsi2l.jpg

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

Generally, the results show that objects closer to the lens have larger disparity and thus appear brighter while objects further away from the lens have smaller disparity and thus appear darker. There pavement does not seem to be properly mapped. One possible reason for this because we constrain the algorithm to only horizontal search. As you can see, there is **some rotation** between the left and the right image. Another possible reason for the poor disparity map of the pavement is due to the lack of very distinguishable features or edges that can help improve the robustness of the SSD this is in contranst to the vegetation on the left that seems to have a proper disparity map.

Also due to minimum disparity limit due to pixel size, the any object that is very far away will not be differentiated on the disparity map.

3.4 Optional Bag of Words and Spatial Pyramid Matching

The section will reference the original article on SPM [7] and reference code from [8] to implement BOW and SPM.

3.4.1 Summary

Algorithm	BOW+ KNN	BOW+SVM	SPM (L=2) +SVM
Accuracy	60%	67%	67-68%

Refer to the sections bellow for proof of results

- *Results Bag of SIFT Representation + (KNN) Nearest Neighbor Classifier*
- *Results Bag of SIFT Representation + one-vs-all SVMs*
- *Results for Spatial Pyramid Matching +* one-vs-all SVMs

3.4.2 Algorithm

3.4.2.1 Bag of words

“Bag of words models are a popular technique for image classification inspired by models used in natural language processing. The model ignores or downplays word arrangement (spatial information in the image) and classifies based on a histogram of the frequency of visual words. The visual word "vocabulary" is established by clustering a large corpus of local features. See Szeliski chapter 14.4.1 for more details on category recognition with quantized features. In addition, 14.3.2 discusses vocabulary creation and 14.1 covers classification techniques.” [8]

Commonly in bag of words, huge data images is first vectorized using **SIFT. The vocabulary (features)** is extracted using K means clustering of the SIFT. Each K means represents a feature. Note the number of K means is selected by the programmer. In the code bellow we study how K means affect the accuracy of prediction. (Typically larger K, better accuracy because there is more features for prediction)

Using the set of K means/ features, feature histogram is formed for each image.

Using label training data set feature histogram and their corresponding label, a model can be trained either using KNN(K nearest neighbour) or SVM(Support vector Machine). Using the Trained Model, a prediction can be made on any input image. Note Typically (SVM models perform better than KNN)

3.4.2.2 Spatial Pyramid Matching

SPM builds on the same concepts of back of words by considering location of features. This is different from Bag of Words that sole depend on the histogram of features. In SPM, the same image is further sub divided into smaller images (Level 0=1 image, Level 1=4 image, Level 2=16 image, total 21 images). Each image is vectorized using SIFT and using K mean clustering, histogram features is formed for them. The histogram of features for the different level is then appended in a specific order.

3.4.3 CODE

Note

- Code was fully extracted from [8] with minor edits to reproduce accuracy of SPM for different levels L=0, L=1, L=2.
- Data was taken from [9] (source Caltect 101 [10])
- Explanations of flow of code for both BOW and SP and evaluations were added by me.

In the following code

- BOW was implemented with KNN(K nearest neighbour) and SVM(Support vector Machine)
    - (Not related of KNN) Varying number of K-means clustering/ feature selected
    - Varying C in SVM to find optimal parameter for SVM
- Spatial Pyramid Matching was implemented with SVM(Support vector Machine)

Source: [8]	Explanations of flow of code for both BOW and SP and evaluations by me

```python
# import packages here
import cv2
import numpy as np
import matplotlib.pyplot as plt
import glob
#from scipy.misc import imresize  # resize images
import copy
print('OpenCv Version:',cv2.__version__)

class_names = [name[11:] for name in glob.glob('data/train/*')]
class_names = dict(zip(range(0,len(class_names)), class_names))
print (class_names)
```

```python
###########################################################################
def load_dataset(path, num_per_class=-1):
    data = []
    labels = []
    for id, class_name in class_names.items():
        img_path_class = glob.glob(path + class_name + '/*.jpg')
        if num_per_class > 0:
            img_path_class = img_path_class[:num_per_class]
        labels.extend([id]*len(img_path_class))
        for filename in img_path_class:
```

This section load data

1. train data, train label
2. test data, test label

```python
            data.append(cv2.imread(filename, 0))
    return data, labels

# load training dataset
train_data, train_label = load_dataset('data/train/', 100)
train_num = len(train_label)
print (train_num)
# load testing dataset
test_data, test_label = load_dataset('data/test/', 100)
test_num = len(test_label)
```

```python
###########################################################################
from sklearn.neighbors import KNeighborsClassifier

# train model
def trainKNN(data, labels, k):
    neigh = KNeighborsClassifier(n_neighbors=k, p=2)
    neigh.fit(data, labels)
    return neigh
###########################################################################
#Bag of SIFT Representation + Nearest Neighbor Classifer

from sklearn.cluster import KMeans
from sklearn import preprocessing

# compute dense SIFT
def computeSIFT(data):
    x = []
    for i in range(0, len(data)):
        sift = cv2.xfeatures2d.SIFT_create()
        img = data[i]
        step_size = 15
        kp = [cv2.KeyPoint(x, y, step_size) for x in range(0, img.shape[0], step_size) for y in range(0, img.shape[1], step_size)]
        dense_feat = sift.compute(img, kp)
        x.append(dense_feat[1])

    return x

# extract dense sift features from training images
x_train = computeSIFT(train_data)
x_test = computeSIFT(test_data)

all_train_desc = []
for i in range(len(x_train)):
    for j in range(x_train[i].shape[0]):
        all_train_desc.append(x_train[i][j,:])

all_train_desc = np.array(all_train_desc)
###########################################################################
# build BoW presentation from SIFT of training images
def clusterFeatures(all_train_desc, k):
    kmeans = KMeans(n_clusters=k, random_state=0).fit(all_train_desc)
    return kmeans
```

3.4.3.1 *Bag of SIFT Representation + Nearest Neighbor Classifier*

1. ***Vectorise Image*** by converting image to dense SIFT
   - computeSIFT(data)
     - return SHIFT data x
2. ***build BoW*** presentation from SIFT of training images
   - clusterFeatures(all_train_desc, k)
     - return k means
     - K means clustering to for k number of means (k features) from SIFT
3. ***form training set histograms*** for each training image using BoW representation
   - def formTrainingSetHistogram(x_train, kmeans, k):
     - return histogram of features
     - using SIFT predict closest k mean and fill up feature histogram
4. ***build histograms for test set*** and ***predict using KNN***
   - def predictKMeans(kmeans, scaler, x_test, train_hist, train_label, k):
     - return test set prediction
     - Use KNN(K nearest neighbours) different from K mean clustering.
       - k-NN is a supervised algorithm used for classification
       - For each evaluation matrix, Decide upon the value of k. Here k refers to the number of closest neighbors we will consider while doing the majority voting of target labels.
       - Run k-NN a few times, changing k and checking the evaluation measure.
       - In each iteration, k neighbors vote, majority vote wins and becomes the ultimate prediction
       - Optimize k by picking the one with the best evaluation measure.
       - Once you've chosen k, use the same training set and now create a new test set with the people's ages and incomes that you have no labels for, and want to predict.
     - Train KNN using train_hist and train_label
     - Use trained KNN to predict test_hist's label
5. ***evaluate accuracy*** of predict test_hist's label

```python
#############################################################################
# form training set histograms for each training image using BoW representation
def formTrainingSetHistogram(x_train, kmeans, k):
    train_hist = []
    for i in range(len(x_train)):
        data = copy.deepcopy(x_train[i])
        predict = kmeans.predict(data)
        train_hist.append(np.bincount(predict, minlength=k).reshape(1,-1).ravel())

    return np.array(train_hist)

# build histograms for test set and predict
def predictKMeans(kmeans, scaler, x_test, train_hist, train_label, k):
    # form histograms for test set as test data
    test_hist = formTrainingSetHistogram(x_test, kmeans, k)

    # make testing histograms zero mean and unit variance
    test_hist = scaler.transform(test_hist)

    # Train model using KNN
    knn = trainKNN(train_hist, train_label, k)
    predict = knn.predict(test_hist)
    return np.array([predict], dtype=np.array([test_label]).dtype)


def accuracy(predict_label, test_label):
    return np.mean(np.array(predict_label.tolist()[0]) == np.array(test_label))
#############################################################################

k = [10, 15, 20, 25, 30, 35, 40]
for i in range(len(k)):
    kmeans = clusterFeatures(all_train_desc, k[i])
    train_hist = formTrainingSetHistogram(x_train, kmeans, k[i])

    # preprocess training histograms
    scaler = preprocessing.StandardScaler().fit(train_hist)
    train_hist = scaler.transform(train_hist)

    predict = predictKMeans(kmeans, scaler, x_test, train_hist, train_label, k[i])
    res = accuracy(predict, test_label)
    print("k =", k[i], ", Accuracy:", res*100, "%")
```

- def accuracy(predict_label, test_label):
    - return correct label/ total test label

```python
#Bag of SIFT Representation + one-vs-all SVMs

from sklearn.svm import LinearSVC

k = 60
kmeans = clusterFeatures(all_train_desc, k)

# form training and testing histograms
```

3.4.3.2 Bag of SIFT Representation + one-vs-all SVMs

1. (Same)***Vectorise Image*** by converting image to dense SIFT
    - computeSIFT(data)
        - return SHIFT data x

```python
train_hist = formTrainingSetHistogram(x_train, kmeans, k)
test_hist = formTrainingSetHistogram(x_test, kmeans, k)

#############################################################################
# normalize histograms
scaler = preprocessing.StandardScaler().fit(train_hist)
train_hist = scaler.transform(train_hist)
test_hist = scaler.transform(test_hist)
#############################################################################
#Train one-vs-all SVMs using sklearn

for c in np.arange(0.0001, 0.1, 0.00198):
    clf = LinearSVC(random_state=0, C=c)
    clf.fit(train_hist, train_label)
    predict = clf.predict(test_hist)
    print ("C =", c, ",\t Accuracy:", np.mean(predict == test_label)*100, "%")
#############################################################################
#We can train 15 SVM classifiers manually and get same result


y_train_global = np.zeros((len(train_label), 1))
y = copy.deepcopy(y_train_global)

y_predict = np.zeros((len(test_label), 1))
for i in range(len(test_label)):
    index = 0
    test = np.array([test_hist[i,:]]).T
    for j in range(len(class_names)):
        y = copy.deepcopy(y_train_global)
        y[index:index+100, 0:1] = np.ones((100,1))
        clf = LinearSVC(random_state=0, C=0.06148)
        clf.fit(train_hist, y.ravel())
        if j == 0:
            maxScore = np.dot(clf.coef_, test) + clf.intercept_
            y_predict[i, 0:1] = j
        elif np.dot(clf.coef_, test) + clf.intercept_ > maxScore:
            maxScore = np.dot(clf.coef_, test) + clf.intercept_
            y_predict[i, 0:1] = j
        index = index + 100
print ("Accuracy:", np.mean(y_predict.ravel() == test_label)*100, "%")
```

2. (Same)***build BoW*** presentation from SIFT of training images
    - clusterFeatures(all_train_desc, k)
        - return k means
        - K means clustering to for k number of means (k features) from SIFT
3. (Same)***form training and test set histograms*** for each training image using BoW representation
    - def formTrainingSetHistogram(x_train, kmeans, k):
        - return histogram of features
        - using SIFT predict closest k mean and fill up feature histogram
4. ***Use 1 v All SVM (Support Vector Machine) to train and predict***
    - clf = LinearSVC(random_state=0, C=c)
    - clf.fit(train_hist, train_label)
    - predict = clf.predict(test_hist)
        - Large Value of parameter C => small margin
        - Small Value of paramerter C => Large margin
            - find optimal smallest c (largest margin) value for training data set so that misclassification of test data set is lower. In order words, it gerneralised the SVM model for other data sets other than training data set source: Medium
5. (Same)***evaluate accuracy*** of predict test_hist's label
    - def accuracy(predict_label, test_label):
        - return correct label/ total test label

```python
#############################################################################
import itertools
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
```

3.4.3.3 Plot Confusion Matrix for Bag of SIFT Representation + one-vs-all SVMs

REFER TO RESULTS : *"Results Bag of SIFT Representation + one-vs-all SVMs"*

```
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        #print("Normalized confusion matrix")

    #print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Compute confusion matrix
cnf_matrix = confusion_matrix(np.array([test_label]).T, y_predict)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure(figsize=(18, 6))
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plt.figure(figsize=(18, 6))
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```python
#Improve performance with Spatial Pyramid Matching

import math

def extract_denseSIFT(img):
    DSIFT_STEP_SIZE = 2
    sift = cv2.xfeatures2d.SIFT_create()
    disft_step_size = DSIFT_STEP_SIZE
    keypoints = [cv2.KeyPoint(x, y, disft_step_size)
            for y in range(0, img.shape[0], disft_step_size)
                for x in range(0, img.shape[1], disft_step_size)]

    descriptors = sift.compute(img, keypoints)[1]

    #keypoints, descriptors = sift.detectAndCompute(gray, None)
    return descriptors


# form histogram with Spatial Pyramid Matching upto level L with codebook kmeans and k codewords
def getImageFeaturesSPM(L, img, kmeans, k):
    W = img.shape[1]
    H = img.shape[0]
    h = []
    for l in range(L+1):
        w_step = math.floor(W/(2**l))
        h_step = math.floor(H/(2**l))
        x, y = 0, 0
        for i in range(1,2**l + 1):
            x = 0
            for j in range(1, 2**l + 1):
                desc = extract_denseSIFT(img[y:y+h_step, x:x+w_step])
                #print("type:",desc is None, "x:",x,"y:",y, "desc_size:",desc is None)
                predict = kmeans.predict(desc)
                histo = np.bincount(predict, minlength=k).reshape(1,-1).ravel()
                weight = 2**(l-L)
                h.append(weight*histo)
                x = x + w_step
            y = y + h_step

    hist = np.array(h).ravel()
    # normalize hist
    dev = np.std(hist)
    hist -= np.mean(hist)
    hist /= dev
    return hist


# get histogram representation for training/testing data
def getHistogramSPM(L, data, kmeans, k):
    x = []
    for i in range(len(data)):
        hist = getImageFeaturesSPM(L, data[i], kmeans, k)

        x.append(hist)
```

3.4.3.4 Improve performance with Spatial Pyramid Matching

1. (Same done)***Vectorise Image*** by converting image to dense SIFT
   - computeSIFT(data)
     - return SHIFT data x
2. (Same 200 features SHIFT data)***build BoW*** presentation from SIFT of training images
   - clusterFeatures(all_train_desc, k)
     - return k means
     - K means clustering to for k number of means (k features) from SIFT
3. (different)***form training and test set feature histograms*** for each training image using BoW representation
   - ***getHistogramSPM(L, data, kmeans, k)***
     - return SPM Histogram
       - return combined SPM Histogram
       - ↑ ***getImageFeaturesSPM(L, img, kmeans, k)***
         - return SPM Histogram for each all level of spaces in the spacial pyramid.
         - For L=0: 1 image, L=1: 4image, L=2: 16image
         - ↑ ***extract_denseSIFT(img)***
4. (same) ***Use 1 v All SVM (Support Vector*** *Machine) to train and predict*
   - clf = LinearSVC(random_state=0, C=c)
   - clf.fit(train_hist, train_label)
   - predict = clf.predict(test_hist)
     - Large Value of parameter C => small margin
     - Small Value of paramerter C => Large margin
       - find optimal smallest c (largest margin) value for training data set so that misclassification of test data set is lower. In order words, it gerneralised the SVM model for other data sets other than training data set source: Medium
5. (Same)*evaluate accuracy* of predict test_hist's label
   - def accuracy(predict_label, test_label):
     - return correct label/ total test label

```
    return np.array(x)

#############################################################################
k = 200
kmeans = clusterFeatures(all_train_desc, k)
#############################################################################
train_histo = getHistogramSPM(2, train_data, kmeans, k)
test_histo = getHistogramSPM(2, test_data, kmeans, k)
#############################################################################
# train SVM
for c in np.arange(0.000307, 0.001, 0.0000462):
    clf = LinearSVC(random_state=0, C=c)
    clf.fit(train_histo, train_label)
    predict = clf.predict(test_histo)
    print ("C =", c, ",\t\t Accuracy:", np.mean(predict == test_label)*100, "%")
```

3.4.4 Results Bag of SIFT Representation + (KNN) Nearest Neighbor Classifier

```
k = 10 , Accuracy: 52.800000000000004 %
k = 15 , Accuracy: 51.93333333333333 %
k = 20 , Accuracy: 55.266666666666666 %
k = 25 , Accuracy: 55.60000000000001 %
k = 30 , Accuracy: 58.4 %
k = 35 , Accuracy: 58.86666666666667 %
k = 40 , Accuracy: 60.0 %
```

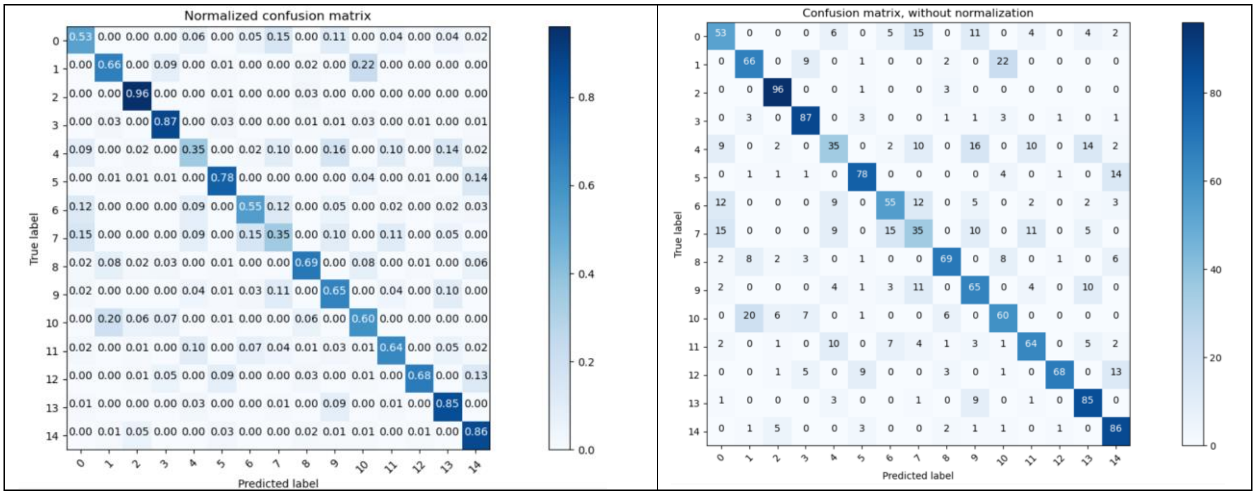
3.4.5 Results Bag of SIFT Representation + one-vs-all SVMs

```
C = 0.00208 ,    Accuracy: 61.33333333333333 %
C = 0.00406 ,    Accuracy: 63.0 %
C = 0.00604 ,    Accuracy: 63.46666666666667 %
C = 0.00802 ,    Accuracy: 63.93333333333333 %
C = 0.009999999999999998 ,     Accuracy: 64.4 %
C = 0.01198 ,    Accuracy: 64.73333333333333 %
C = 0.01396 ,    Accuracy: 65.06666666666666 %
C = 0.01594 ,    Accuracy: 65.46666666666667 %
C = 0.01792 ,    Accuracy: 65.8 %
C = 0.019899999999999998 ,     Accuracy: 65.93333333333334 %
C = 0.02188 ,    Accuracy: 65.86666666666666 %
C = 0.02386 ,    Accuracy: 65.86666666666666 %
C = 0.02584 ,    Accuracy: 65.93333333333334 %
C = 0.02782 ,    Accuracy: 66.0 %
C = 0.0298 ,     Accuracy: 66.06666666666666 %
C = 0.03178 ,    Accuracy: 66.26666666666667 %
C = 0.033760000000000005 ,     Accuracy: 66.2 %
C = 0.03574 ,    Accuracy: 66.4 %
C = 0.037720000000000004 ,     Accuracy: 66.4 %
C = 0.0397 ,     Accuracy: 66.66666666666666 %
C = 0.04168 ,    Accuracy: 66.8 %
C = 0.043660000000000004 ,     Accuracy: 66.73333333333333 %
C = 0.04564 ,    Accuracy: 66.86666666666666 %
C = 0.04762 ,    Accuracy: 66.93333333333334 %
C = 0.049600000000000005 ,     Accuracy: 66.93333333333334 %
C = 0.05158 ,    Accuracy: 66.93333333333334 %
C = 0.05356 ,    Accuracy: 66.93333333333334 %
C = 0.055540000000000006 ,     Accuracy: 67.0 %
C = 0.05752 ,    Accuracy: 67.06666666666666 %
C = 0.059500000000000004 ,     Accuracy: 66.86666666666666 %
C = 0.06148 ,    Accuracy: 66.8 %
C = 0.06346 ,    Accuracy: 66.86666666666666 %
C = 0.06544 ,    Accuracy: 66.93333333333334 %
C = 0.06742000000000001 ,      Accuracy: 66.93333333333334 %
```

```
C = 0.0694 ,     Accuracy: 66.93333333333334 %
C = 0.07138 ,    Accuracy: 67.0 %
C = 0.07336000000000001 ,      Accuracy: 67.06666666666666 %
C = 0.07534 ,    Accuracy: 67.0 %
C = 0.07732 ,    Accuracy: 67.06666666666666 %
C = 0.0793 ,     Accuracy: 67.06666666666666 %
C = 0.08128 ,    Accuracy: 67.06666666666666 %
C = 0.08326 ,    Accuracy: 67.2 %
C = 0.08524 ,    Accuracy: 67.13333333333334 %
C = 0.08722 ,    Accuracy: 67.2 %
C = 0.0892 ,     Accuracy: 67.2 %
C = 0.09118 ,    Accuracy: 67.2 %
C = 0.09316 ,    Accuracy: 67.2 %
C = 0.09514 ,    Accuracy: 67.33333333333333 %
C = 0.09712 ,    Accuracy: 67.33333333333333 %
C = 0.09910000000000001 ,      Accuracy: 67.33333333333333 %
Accuracy: 66.8 %
```

```
{0: 'Bedroom', 1: 'Coast', 2: 'Forest', 3: 'Highway', 4: 'Indus
trial', 5: 'InsideCity', 6: 'Kitchen', 7: 'LivingRoom', 8: 'Mou
ntain', 9: 'Office', 10: 'OpenCountry', 11: 'Store', 12: 'Stree
t', 13: 'Suburb', 14: 'TallBuilding'}
```

0	Bedroom	8	Mountain
1	Coast	9	Office
2	Forest	10	OpenCountry
3	Highway	11	Store
4	Industrial	12	Street
5	InsideCity	13	Suburb
6	Kitchen	14	TallBuilding
7	LivingRoom		



3.4.5.1 Results for Spatial Pyramid Matching + one-vs-all SVMs

```
C = 0.000307 ,              Accuracy: 67.93333333333334 %
C = 0.00035319999999999997 ,          Accuracy: 67.93333333333334 %
C = 0.00039939999999999995 ,          Accuracy: 68.0 %
C = 0.00044559999999999993 ,          Accuracy: 67.93333333333334 %
C = 0.0004917999999999999 ,           Accuracy: 67.73333333333333 %
C = 0.0005379999999999998 ,           Accuracy: 67.66666666666666 %
C = 0.0005841999999999999 ,           Accuracy: 67.73333333333333 %
C = 0.0006303999999999999 ,           Accuracy: 67.80000000000001 %
C = 0.0006765999999999999 ,           Accuracy: 67.73333333333333 %
C = 0.0007227999999999998 ,           Accuracy: 67.46666666666667 %
C = 0.0007689999999999998 ,           Accuracy: 67.4 %
C = 0.0008151999999999999 ,           Accuracy: 67.26666666666667 %
C = 0.0008613999999999998 ,           Accuracy: 67.4 %
C = 0.0009075999999999997 ,           Accuracy: 67.26666666666667 %
C = 0.0009537999999999998 ,           Accuracy: 67.2 %
C = 0.0009999999999999998 ,           Accuracy: 67.33333333333333 %
```

4 Appendix

4.1 OTSU_B.m

```
function threshold =OTSU_B(gray, Analysis)
%OTSU Thresholding
%Maximise interclass variance
size(count)
inter_class_var=zeros(size(bins,1),1);
for n=1:size(bins,1)
    %Mean of L(backgnd)and H(foregnd)

mean_L=dot(count(1:n),bins(1:n))/sum(count(1:n));

mean_H=dot(count(n+1:256),bins(n+1:256))/sum(count(n
+1:256));
    weight_L=sum(count(1:n))/sum(count);
    weight_H=sum(count(n+1:256))/sum(count);
    inter_class_var(n)=weight_L*weight_H*(mean_L-
mean_H)^2;
end
[M,I] = max(inter_class_var)
%Analysis Report
```

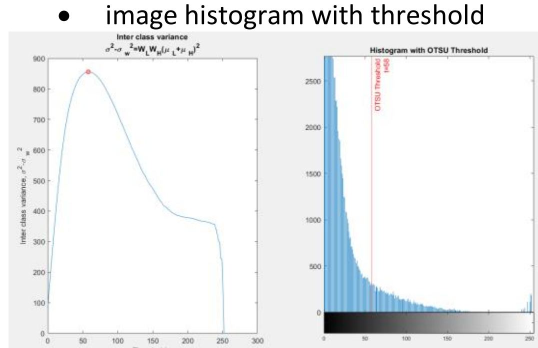
Maximize intraclass variance

$$\sigma^2 - \sigma_w{}^2 = W_L W_H(\mu_L + \mu_H)^2$$

```
if Analysis ==true
        %Plot Interclass variance
        figure( 'Position', [10 10 900 600]);

        subplot(1,2,1);plot(bins,inter_class_var);
        hold on;
        plot(bins(I),inter_class_var(I),'o',...
            'MarkerEdgeColor','red',...
            'MarkerFaceColor',[1 .6 .6])
        title({'Inter class variance','{\sigma}^2-
{\sigma _w}^2={W}_L{W}_H({\mu _L}+{\mu _H})^2'})
        xlabel('Threshold, t');
        ylabel('Inter class variance, {\sigma}^2-
{\sigma _w}^2');

        %Plot histogram with OTSU threshold


subplot(1,2,2);imhist(uint8(gray),256);title("Histog
ram with OTSU Threshold");
        hold on;
        xline(bins(I),'-r',{'OTSU
Threshold',strcat('t= ',num2str(bins(I)))})
    end

threshold=bins(I)
end
```

Analysis produce

- interclass variance vs threshold plot and
- image histogram with threshold



5 References

[1] "EDGE DETECTION," The University of Auckland, [Online]. Available: https://www.cs.auckland.ac.nz/compsci373s1c/PatricesLectures/Edge%20detection-Sobel_2up.pdf.

[2] "Radon Transform," MathWorks, [Online]. Available: https://www.mathworks.com/help/images/radon-transform.html#:~:text=The%20radon%20function%20computes%20projections,beams%2C%20in%20a%20certain%20direction..

[3] "Radon transform," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Radon_transform#:~:text=In%20mathematics%2C%20the%20Radon%20transform,the%20function%20over%20that%20line..

[4] "What's the difference between the Hough and Radon transforms?," [Online]. Available: https://dsp.stackexchange.com/questions/470/whats-the-difference-between-the-hough-and-radon-transforms.

[5] "Hough transform," [Online]. Available: https://en.wikipedia.org/wiki/Hough_transform.

[6] "pol2cart," MathWorks, [Online]. Available: https://www.mathworks.com/help/matlab/ref/pol2cart.html.

[7] C. S. J. P. Svetlana Lazebnik1, "Beyond Bags of Features: Spatial Pyramid Matching," 2006. [Online]. Available: https://inc.ucsd.edu/~marni/Igert/Lazebnik_06.pdf.

[8] TrungTVo, "spatial-pyramid-matching-scene-recognition," [Online]. Available: https://github.com/TrungTVo/spatial-pyramid-matching-scene-recognition/blob/master/spatial_pyramid.ipynb.

[9] [Online]. Available: https://drive.google.com/u/0/uc?id=0B446EB1iI6_Qc0Q1NTRTajdUVTg&export=download.

[10] "Caltech 101," [Online]. Available: http://www.vision.caltech.edu/Image_Datasets/Caltech101/.