1. (35 pts) In class, you used the Hough transform to detect lines in an image of edge points. Modify the Hough transform algorithm developed in lecture to make use of the gradient direction (see slide 36 on Lecture 14). Apply it to the “diamond.tif” image and show that you detect the four peaks corresponding to the four lines in the image. The new algorithm should visit far fewer accumulator cells than the old algorithm. Compare the number of accumulator cells visited with the new algorithm to the old algorithm (note that this is just the sum of the values in the accumulator array).

Solution: The old program, developed in class is as follows.

```matlab
% clear all
% close all
I = imread('diamond.tif');
E = edge(I, 'canny');
[iHeight,iWidth] = size(E);
distMax = sqrt(iHeight^2 + iWidth^2);  % Max possible distance from origin
theta = -90:1:89;           % range of theta values
rho = -distMax:1:distMax;   % range of rho values
H = zeros(length(rho),length(theta));  % Allocate accumulator array

% Scan through edge image
for ix=1:iWidth
    for iy=1:iHeight
        if E(iy,ix) ~= 0
            % Fill in accumulator array
            for iTheta = 1:length(theta)
                t = theta(iTheta)*pi/180;      % get angle in radians
                dist = ix*cos(t) + iy*sin(t);
                [d, iRho] = min(abs(rho-dist));
                if d <= 1
                    H(iRho,iTheta) = H(iRho,iTheta) + 1;  % Inc accumulator
                end
            end
        end
    end
end
```
peaks = houghpeaks(H,4); % Find 4 peaks

% Draw lines
% Equation of a line: rho = x*cos(theta) + y*sin(theta)
figure, imshow(E, []);
for i=1:size(peaks,1)
    r = rho(peaks(i,1));
    t = theta(peaks(i,2));
    fprintf('Peak %d: theta=%f, rho=%f, count=%d
', i, t, r, ...
    H(peaks(i,1), peaks(i,2)));

    t = t * pi/180; % convert to radians
    if t == 0
        x0 = r;    x1 = r;
        y0 = 1;     y1 = size(E,2);
    else
        x0 = 1;
        y0 = (r - x0*cos(t)) / sin(t);
        x1 = size(E,2);
        y1 = (r - x1*cos(t)) / sin(t);
    end

    line([x0 x1], [y0 y1], 'Color', 'r');
end

fprintf('Total counts in accumulator array = %d
', sum(H(:)));
Total counts in accumulator array = 32400

The new program is as follows:

```matlab
clear all
close all

I = double(imread('diamond.tif'));

% Compute the gradient components of the image. 
% (For noisy images, you should first apply a smoothing operator.)
[gx,gy] = gradient(I);

Gmag = sqrt(gx.^2 + gy.^2);
figure, imshow(Gmag,:), title('gradient magnitude');

% Angles range from -pi/2 to +pi/2
Gangle = atan(gy./gx);
figure, imshow(Gangle,:), title('gradient angles'), impixelinfo;

E = edge(I, 'canny');

[iHeight,iWidth] = size(E);
distMax = sqrt(iHeight^2 + iWidth^2); % Max possible distance from origin

theta = -90:1:89;           % range of theta values
rho = -distMax:1:distMax;   % range of rho values
H = zeros(length(rho),length(theta));  % Allocate accumulator array

% Scan through edge image
for ix=1:iWidth
    for iy=1:iHeight
        if E(iy,ix) ~= 0
            t = Gangle(iy,ix);
            % Calculate distance from origin, given this angle
            dist = ix*cos(t) + iy*sin(t);

            % Find theta value that is closest to this
            [d, iTheta] = min(abs(theta-t*180/pi));

            % Find rho value that is closest to this
            [d, iRho] = min(abs(rho-dist));

            if d <= 1
                H(iRho,iTheta) = H(iRho,iTheta) + 1; % Inc accumulator
            end
        end
    end
end

(the rest of the program, which finds the peaks and draws lines, is the same)
The output of the program is:

- Peak 1: \( \theta = -45.000000, \rho = -32.038672, \text{count}=45 \)
- Peak 2: \( \theta = -45.000000, \rho = 30.961328, \text{count}=44 \)
- Peak 3: \( \theta = 45.000000, \rho = 212.961328, \text{count}=44 \)
- Peak 4: \( \theta = 45.000000, \rho = 148.961328, \text{count}=43 \)

Total counts in accumulator array = 180

So the new program visits far fewer accumulator cells than the old program (180 vs. 32400).

2. (35 pts) Program the Otsu automatic thresholding algorithm to find two optimal thresholds, as described in section 10.3.6 of the textbook. Apply your program to the image “iceberg.tif”.
   a. Give the two thresholds that were found.
b. At the thresholds found above, give the resulting values of the between-class variance, and the within-class variance.

c. Give the program and the output thresholded image.

Solution:

```matlab
clear all, close all;

I = imread('iceberg.tif');
[H,W] = size(I);
figure, imshow(I);

% Get histogram. hist(i) is the count of pixels with value x(i).
[hist,x] = imhist(I);
figure, imhist(I);

% Convert to probability
p = hist/(H*W);

% Compute mean of entire image
ug = sum( x .* p );
fprintf('Global mean = %f
', ug);

% We will search for two thresholds, t1 and t2, where each threshold is % between 1..255. This array holds the between-class variance for each % threshold pair of thresholds.
vb = zeros(255,255);    % variances for each combination of t1,t2
vw = 1e9 * ones(255,255);

for t1 = 1:253
    % Group 1 is all pixels with intensity <= t1
    m1 = (x <= t1);     % create a logical mask; true where x<=t1
    P1 = sum(p(m1));    % total probability for those values

    % Compute mean of group 1
    u1 = sum( x(m1) .* p(m1) ) / P1;

    for t2 = t1+1:254
        % Group 2 is all pixels with intensity >t1 and <= t2
        m2 = (x > t1 & x <= t2);   % true where x>t1 and x<=t2
        P2 = sum(p(m2));          % total probability for those values

        % Compute mean of group 2
        u2 = sum( x(m2) .* p(m2) ) / P2;

        % Group 3 is all pixels with intensity >t2
        m3 = (x > t2);            % true where x>t2
        P3 = sum(p(m3));          % total probability for those values

        % Compute mean of group 3
```

\[ u_3 = \frac{\sum x(m_3) \cdot p(m_3)}{P_3}; \]

% Between-group variance
\[ vb(t_1,t_2) = P_1*(u_1-ug)^2 + P_2*(u_2-ug)^2 + P_3*(u_3-ug)^2; \]

% Let's also compute the within-group variance.
\[ v_1 = \frac{\sum (x(m_1)-u_1)^2 \cdot p(m_1)}{P_1}; \quad \% \text{Variance of group 1} \]
\[ v_2 = \frac{\sum (x(m_2)-u_2)^2 \cdot p(m_2)}{P_2}; \quad \% \text{Variance of group 2} \]
\[ v_3 = \frac{\sum (x(m_3)-u_3)^2 \cdot p(m_3)}{P_3}; \quad \% \text{Variance of group 3} \]
\[ vw(t_1,t_2) = P_1*v_1 + P_2*v_2 + P_3*v_3; \]

end

% Find the maximum between-group variance and where it occurs
vmax = max(vb(:));
[t1,t2] = find(vb == vmax);
fprintf('Maximum between-group variance=%f at t1=%d, t2=%d
', vmax, t1, t2);

% Find the minimum within-group variance and where it occurs
vmin = min(vw(:));
[t1,t2] = find(vw == vmin);
fprintf('Minimum within-group variance=%f at t1=%d, t2=%d
', vmin, t1, t2);

% Threshold the image using t1,t2
Iout = zeros(size(I));
Iout(I<=t1) = 0;
Iout(I>t1 & I<=t2) = 1;
Iout(I>t2) = 2;

figure, imshow(Iout,[]);

Here is the output:

\[
\text{Global mean} = 87.251125 \\
\text{Maximum between-group variance}=5325.723512 \text{ at } t_1=80, t_2=177 \\
\text{Minimum within-group variance}=256.549183 \text{ at } t_1=80, t_2=177
\]

Usually you don’t need to compute both the between-group variance and the within-group variance in order to find the thresholds ... one or the other will do. I just wanted to show that you get the same answer if you do it either way.

Below are shown: Input image, histogram, output thresholded image:
3. (30 pts) Start an “annotated bibliography” for your final project. Namely, find at least 5 papers from the literature that are most relevant to the project and read them. Then write a paragraph (in your own words) for each, summarizing the paper and how it relates to your project. (Note – you can often find papers on websites, but citing a website is not acceptable … it must be from a journal or a conference; a book is ok too.) Below is an example from a previous student project.


This paper is about lane detection and is useful for my project since I want to do lane detection. One key feature of this paper is that it is for surveillance application and thus they want to find multiply lanes instead of just one single lane. First they use edge detection to simplify the image. Due to the orientation of the camera in this application, which is in the perspective view, the camera parameters need to be estimated to get better results. Then using orientation and length discrimination the lane information can be determined.
This paper performs lane detection assuming a Catmull-Rom spline line model, and is useful for my project since I want to do lane detection. First they go into detail of what a Catmull-Rom spline is. They then break down lanes into three types: straight, left turn, and right turn. For the turns three points are needed to use the spline model. To find the lanes in the image a canny edge detector was used, along with other image processing techniques. Once a general area of the lanes was found control points for the spline had to be fit so that the model and lanes would overlap.