# Homework 5

ECE 253
Digital Image Processing

November 29, 2015

**Instructions :**

- Homework 5 is due by 11:59 PM, December 11, 2015.
- Submit your homework electronically by email to arangesh@ucsd.edu with the subject line *ECE 253 HW5*.
- The email should have one PDF file of your writeup attached. Make sure it includes your full name, PID, and email. This file must be named ECE_253_hw5_lastname_studentid.pdf.
- All problems are to be solved using MATLAB unless mentioned otherwise.
- Append source code, floats, matrices, and image outputs, to your writeup where applicable. Simply pasting your code in the report should suffice, but make sure it is indented correctly.

## Problem 1. Canny Edge Detection

In this problem, you are required to write a function that performs *Canny Edge Detection*. The function has the following specifications:

- It takes in two inputs: a grayscale image, and a threshold $t_e$.
- It returns the edge image.
- You are allowed the use of loops.

A brief description of the algorithm is given below. Make sure your function reproduces the each step as given.

1. **Smoothing**: It is inevitable that all images taken from a camera will contain some amount of noise. To prevent noise from being mistaken for edges, noise must be reduced. Therefore the image is first smoothed by applying a Gaussian filter. A Gaussian kernel with standard deviation $\sigma = 1.4$ (shown below) is to be used.

$$k = \frac{1}{159} \cdot \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

2. **Finding Gradients** The next step is to find the horizontal and vertical gradients of the smoothed image using the *Sobel* operators. The gradient images in the x and y-direction, $G_x$ and $G_y$ are found by applying the kernels $k_x$ and $k_y$ given below:

$$k_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, k_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

The corresponding gradient magnitude image is computed using:

$$|G| = \sqrt{G_x^2 + G_y^2},$$

and the edge direction image is calculated as follows:

$$G_\theta = arctan(\frac{G_y}{G_x}).$$

3. **Non-maximum Suppression (NMS)**: The purpose of this step is to convert the thick edges in the gradient magnitude image to "sharp" edges. This is done by preserving all local maxima in the gradient image, and deleting everything else. This is carried out by recursively performing the following steps for each pixel in the gradient image:

   - Round the gradient direction $\theta$ to nearest $45^\circ$, corresponding to the use of an 8-connected neighbourhood.
   - Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient direction i.e. if the gradient direction is north ($\theta = 90^\circ$), then compare with the pixels to the north and south.
   - If the edge strength of the current pixel is largest; preserve the value of the edge strength. If not, suppress (remove) the value.

4. **Thresholding**: The edge-pixels remaining after the NMS step are (still) marked with their strength. Many of these will probably be true edges in the image, but some may be caused by noise or color variations. The simplest way to remove these would be to use a threshold, so that only edges stronger that a certain value would be preserved. Use the input $t_e$ to perform thresholding on the non-maximum suppressed magnitude image.

Evaluate your canny edge detection function on *car.png* for a suitable value of $t_e$ that retains the edges on the car, while discarding most other edges.

*Things to turn in:*

- The original gradient magnitude image, the image after NMS, and the final edge image after thresholding.
- The value for $t_e$ that you used to produce the final edge image.
- MATLAB code for the function.

## Problem 2. Hough Transform

(The first two parts of this problem is borrowed from Professor Belongie's past CSE 166 class.)

(i) Implement the Hough Transform (HT) using the ($\rho$, $\theta$) parameterization as described in GW Third Edition p. 733-738 (see 'HoughTransform.pdf' under resources section on Piazza). Use accumulator cells with a resolution of 1 degree in $\theta$ and 1 pixel in $\rho$.

(ii) Produce a simple $11 \times 11$ test image made up of zeros with 5 ones in it, arranged like the 5 points in GW Third Edition Figure 10.33(a). Compute and display its HT; the result should look like GW Third Edition Figure 10.33(b). Threshold the HT by looking for any $(\rho, \theta)$ cells that contains more than 2 votes then plot the corresponding lines in (x,y)-space on top of the original image.

(iii) Load in the image 'lane.png'. Compute and display its edges using the Sobel operator with default threshold settings, i.e.,

```
E = edge(I,'sobel')
```

Now compute and display the HT of the binary edge image $E$. As before, threshold the HT and plot the corresponding lines atop the original image; this time, use a threshold of 75% maximum accumulator count over the entire HT, i.e. `0.75*max(HT(:))`.

(iv) We would like to only show line detections in the driver's lane and ignore any other line detections such as the lines resulting from the neighboring lane closest to the bus, light pole, and sidewalks. Using the thresholded HT from the 'lanes.png' image in the previous part, show only the lines corresponding to the line detections from the driver's lane by thresholding the HT again using a specified range of $\theta$ this time. What are the approximate $\theta$ values for the two lines in the driver's lane?

*Things to turn in:*

- HT images should have colorbars next to them
- Line overlays should be clearly visible (adjust line width if needed)
- HT image axes should be properly labeled with name and values (see Figure 10.33(b) for example)
- 3 images from 2(ii): original image, HT, original image with lines
- 4 images from 2(iii): original image, binary edge image, HT, original image with lines
- 1 image from 2(iv): original image with lines
- $\theta$ values from 2(iv)
- Code for 2(i), 2(ii), 2(iii), 2(iv)

## Problem 3. Circular Hough Transform

For this problem, we will not use the circular Hough Transform with $(x - c_1)^2 + (y - c_2)^2 = c_3^2$. We will instead use the circular hough transform similar to the method described in E.R. Davies's Machine Vision: Theory, Algorithms, Practicalities 3rd Edition p. 285 (see 'CircularHoughTransform.pdf' under resources section on Piazza).

(i) Load in the image 'coins.png' as $I$. Compute the gradient images along horizontal ($g_x$) and vertical ($g_y$) directions using the sobel kernels, and also compute the gradient magnitude image ($g$), i.e.

$$g_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I, \qquad g_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I, \qquad g = \sqrt{g_x^2 + g_y^2}$$

where $*$ is the convolution operator.

We will use the graythresh(...) function to automatically threshold the gradient magnitude image (g) into a binary edge image, i.e.

```
gmax = max(g(:));
edgeThresh = graythresh(g/gmax);
edgeThresh = gmax * edgeThresh;
e = gradientImg > edgeThresh; % edge image
```

(ii) Given an edge pixel from image $e$ and assuming it is from a circle of radius R, the center of the circle can be estimated by moving a distance R along the direction of the edge pixel's gradient, i.e.

$$x_c = x - \frac{g_x(x,y)}{g(x,y)}R, \qquad y_c = y - \frac{g_y(x,y)}{g(x,y)}R$$

(from eq. 10.3 and 10.4 in E. R. Davies, Machine Vision: Theory, Algorithms, Practicalities 3rd Edition p. 285)

Note that for other applications, the '-' sign may need to be changed to a '+' sign depending on whether the circle's edge is transitioning from dark to light or light to dark. The '-' sign will work for the images provided in this assignment.

Generate the circular Hough Transform accumulation matrix (CHT) (same dimensions as the image) by estimating the center coordinate $(x_c, y_c)$ for each edge pixel in the binary edge image $e$ (round the coordinates to nearest integer), i.e.

$$CHT(x_c, y_c) = CHT(x_c, y_c) + 1$$

Ignore any coordinates that are outside the image boundaries. Do this for $R = 15, 16, ..., 35$ so that the CHT accumulates estimated center votes of varying radii 15-35 pixels. As in the line Hough Transform problem, threshold the CHT using a threshold of 50% maximum accumulator count over the entire CHT, i.e. `0.5*max(CHT(:))`, then plot markers at the corresponding centers atop the original image, i.e.

```
imshow(im);
hold on;
plot(xc,yc,'rx','linewidth',2)
hold off;
```

You may use for loops, but you may find the following functions useful for implementing this without for loops: accumarray(...), bsxfun(...), find(...), ind2sub(...), and sub2ind(...)

(iii) Repeat 3(i) and 3(ii) for 'wheels.png', for $R = 10, 11, ...20$.

*Things to turn in:*

- CHT images should have colorbars next to them
- Marker overlays at the estimated circular centers should be clearly visible.
- 4 images from 3(i): original image, $g_x$, $g_y$, $g$, $e$
- 2 images from 3(ii): CHT, original image with markers
- 6 images from 3(iii): original image, $g_x$, $g_y$, $g$, $e$, CHT, original image with markers
- Code for 3(i), 3(ii), 3(iii)

**Problem 4. Object Detection**

(i) You are given a folder with the following:

- Libsvm folder
- Positives folder that contains positive samples of vehicles
- Negatives folder "negatives"
- Negatives folder 2 "negatives2"
- MATLAB script named "script_training_testing_SVM_assignment.m"

Go through the MATLAB script. Try to understand it.

(a) HOG features are being extracted in the code. What is the function that is extracting the HOG features?

(b) What is the `cellSize` and `blockSize` for the HOG feature extractor?

(c) Run the script and determine the accuracy of the SVM classifier for the parameters already set. What is the accuracy rate that you get after running the code? Explain the accuracy rate (why is it high or low)?

(ii) Change the cellSize to $16 \times 16$ for all the feature extractors. Run the script again and determine the accuracy of the SVM under the same conditions as in (i). Do you see any changes? Explain.

(iii) Now, replace "negatives" folder with "negatives2" folder that is in the script given to you (you could either change "negatives" to "negatives2" in the script (or) rename the folders and keep "negatives" in the script). Rerun the code with the `cellSize` and `blockSize` that you identified in (i)(b). What is the accuracy rate? Explain your answer.

(iv) Now change the number of positive training examples `num_tr_pos` and negative training examples texttttnum_tr_neg in the following manner.

`num_tr_pos` changes from 10 to 200 in steps of 20, and `num_tr_neg` changes from 300 to 750 in steps of 50, i.e.

```
num_tr_pos = [10, 30, 50, 70, 90, 110, 130, 150, 170, 190];
num_tr_neg = [300, 350, 400, 450, 500, 550, 600, 650, 700, 750];
```

For each combination of the two parameters, you will get an accuracy value. Plot the accuracy values and comment on the plot that you get. (You should use the parameters for `cellSize` and `blockSize` and negatives folder in (iii)).

*Things to turn in:*

- Response for 4(i)(a-c), 4(ii), 4(iii)
- Plot for 4(iv)
- Code for 4(iii)