http://cs.ucf.edu/~bagci/

[Programming Assignment] (1)

Computer Vision

Dr. Ulas Bagci • (Fall) 2016 • University of Central Florida (UCF)

Coding Standard and General Requirements

Code for all programming assignments should be **well documented**. A working program with no comments will receive **only partial credit**. Documentation entails writing a description of each function/method, class/structure, as well as comments throughout the code to explain the program flow. Programming language for the assignment is **Python**. You can use standard python built-in IDLE, or CANOPY for the working environment. Other commonly used IDLEs are the following: PyCharm Community Edition, PyScripter, CodeSculptor, Eric Python, Eclipse plus PyDev.

Following libraries will be used extensively throughout the course:

- PIL (The Python Imaging Library), Matplotlib, NumPy, SciPy, LibSVM, OpenCV, VLFeat, python-graph.

If you use CANOPY, make sure that you use version 2.7, which already includes many libraries. If you are asked to implement "Gaussian Filtering", you are not allowed to use a Gaussian function from a known library, you need to implement it from scratch.

Submit by **18th of September 2016**, 11.59pm.

Question 1: Canny Edge Detection Implementation [4pts]

In 1986, John Canny defined a set of goals for an edge detector and described an optimal method for achieving them. Canny specified three issues that an edge detector must address:

- **Error rate:** Desired edge detection filter should find all the edges, there should not be any missing edges, and it should respond only to edge regions.
- **Localization:** Distance between detected edges and actual edges should be as small as possible.
- **Response:** The edge detector should not identify multiple edge pixels where only a single edge exists.

Remember from the lecture that in Canny edge detection, we will first smooth the images, then compute gradients, magnitude, and orientation of the gradient. This procedure is followed by non-max suppression, and finally hysteresis thresholding is applied to finalize the steps. Briefly, follow the steps below for practical implementation of *Canny Edge detector*:

1. Read a gray scale image you can find from Berkeley Segmentation Dataset, Training images, store it as a matrix named I.
2. Create a one-dimensional Gaussian mask G to convolve with I. The standard deviation(s) of this Gaussian is a parameter to the edge detector (call it $\sigma > 0$).

3. Create a one-dimensional mask for the first derivative of the Gaussian in the x and y directions; call these G_x and G_y. The same $\sigma > 0$ value is used as in step 2.

4. Convolve the image I with G along the rows to give the x component image (I_x), and down the columns to give the y component image (I_y).

5. Convolve I_x with G_x to give I'_x, the x component of I convolved with the derivative of the Gaussian, and convolve I_y with G_y to give I'_y, y component of I convolved with the derivative of the Gaussian.

6. Compute the magnitude of the edge response by combining the x and y components. The magnitude of the result can be computed at each pixel (x, y) as: $M(x, y) = \sqrt{I'_x(x, y)^2 + I'_y(x, y)^2}$.

7. Implement **non-maximum suppression** algorithm that we discussed in the lecture. Pixels that are not local maxima should be removed with this method. In other words, not all the pixels indicating strong magnitude are edges in fact. We need to remove false-positive edge locations from the image.

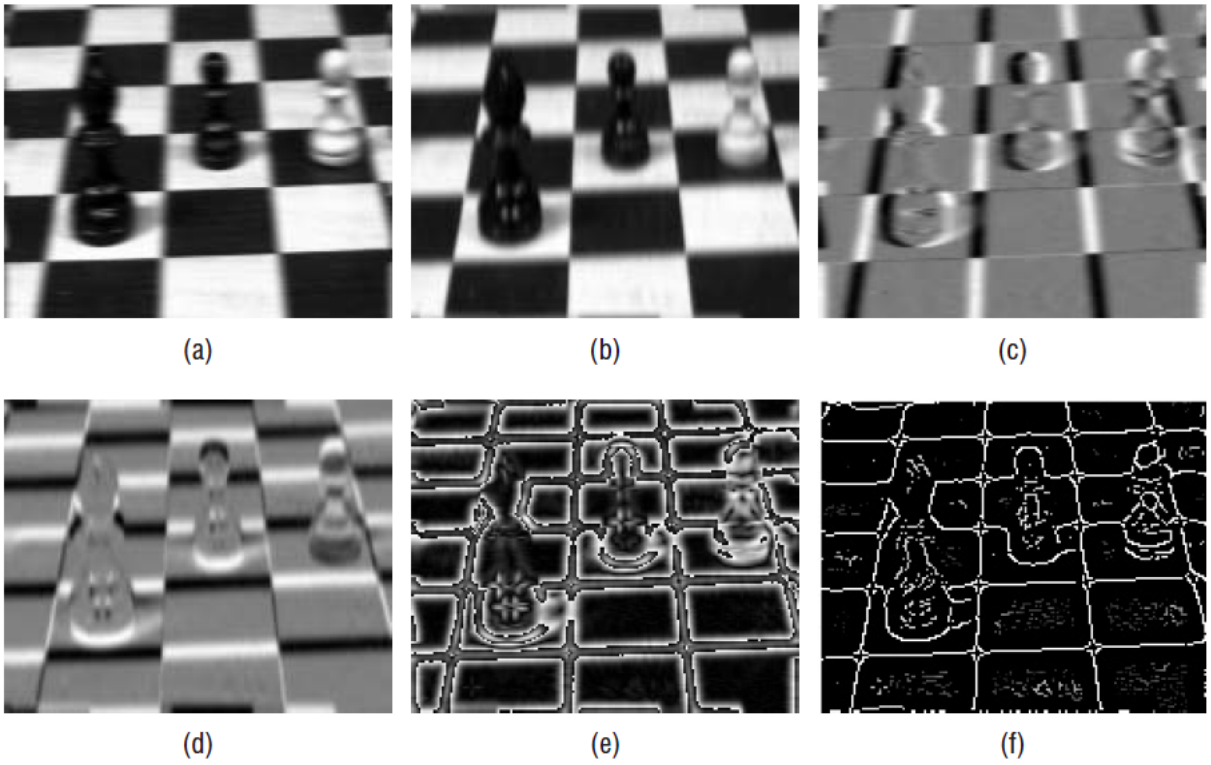
8. Apply **Hysteresis thresholding** to obtain final edge-map.

Definition: Non-maximal suppression means that the center pixel, the one under consideration, must have a larger gradient magnitude than its neighbors in the gradient direction. That is: from the center pixel, travel in **the direction of the gradient** until another pixel is encountered; this is the first neighbor. Now, again starting at the center pixel, travel in the direction opposite to that of the gradient until another pixel is encountered; this is the second neighbor. Moving from one of these to the other passes though the edge pixel in a direction that crosses the edge, so the gradient magnitude should be largest at the edge pixel.

Algorithmically, for each pixel p (at location x and y), you need to test whether a value $M(p)$ is maximal in the direction $\theta(p)$. For instance, if $\theta(p) = pi/2$, i.e., the gradient direction at $p = (x, y)$ is downward, then $M(x, y)$ is compared against $M(x, y - 1)$ and $M(x, y + 1)$, the values above and below of p. If $M(p)$ is not larger than the values at both of those adjacent pixels, then $M(p)$ becomes 0. For estimation of the gradient orientation, $\theta(p)$, you can simply use $atan2(I'_y, I'_x)$.

Hint: It is assumed that the gradient changes continuously as a function of position, and that the gradient at the pixel coordinates are simply sampled from the continuous case. If it is further assumed that the change in the gradient between any two pixels is a linear function, then the gradient at any point between the pixels can be approximated by a linear interpolation.

For a sample output, please refer to figure below: chessboard image's X component of the convolution with a Gaussian (a), Y component of the convolution with a Gaussian (b), X component of the image convolved with the derivative of a Gaussian (c), Y component of the image convolved with the derivative of a Gaussian (d), resulting magnitude image (e), and canny-edge image after non-maximum suppression (f) are shown.



Your tasks:

- Choose three example gray-scale images from Berkeley Segmentation Dataset (Training Images) CLICK HERE. When executed, your algorithm should plot intermediate and final results of Canny Edge Detection process as similar to the figure illustrated above.
- Please show the effect of σ in edge detection by choosing three different σ values when smoothing. Note that you need to indicate which σ works best as a comment in your assignment.

Question 2: Quantitative Evaluation of Edge Detector [2 pts]

[1 pt] Download two input images (and corresponding edge maps of those two images) from the WebCourses. After using your own implementation of Canny edge detection output on input images, you will have three output images. Use the following metrics to evaluate your edge detector with respect to the edge maps provided to you along with input images.
Your task is to report quantitative evaluation of your edge detection results with the following functions:

- **Sensitivity** (or true positive rate TPR) as $\mathsf{TP}/(\mathsf{TP}+\mathsf{FN})$
- **Specificity** (or true negative rate TNR) as $\mathsf{TN}/(\mathsf{TN}+\mathsf{FP})$
- **Precision** (or positive predictive value PPV) as $\mathsf{TP}/(\mathsf{TP}+\mathsf{FP})$
- **Negative Predictive Value** as $\mathsf{TN}/(\mathsf{TN}+\mathsf{FN})$
- **Fall-out** (or false positive rate FPR) as $\mathsf{FP}/(\mathsf{FP}+\mathsf{TN})$
- **False Negative Rate FNR** as $\mathsf{FN}/(\mathsf{FN}+\mathsf{TP})$
- **False Discovery Rate FDR** as $\mathsf{FP}/(\mathsf{FP}+\mathsf{TP})$
- **Accuracy** as $(\mathsf{TP}+\mathsf{TN})/(\mathsf{TP}+\mathsf{FN}+\mathsf{TN}+\mathsf{FP})$
- **F-score** as $2\mathsf{TP}/(2\mathsf{TP}+\mathsf{FP}+\mathsf{FN})$
- **Matthew's Correlation Coefficient** as $\frac{\mathsf{TP}*\mathsf{TN}-\mathsf{FP}*\mathsf{FN}}{\sqrt{((\mathsf{TP}+\mathsf{FP})*(\mathsf{TP}+\mathsf{FN})*(\mathsf{TN}+\mathsf{FP})*(\mathsf{TN}+\mathsf{FN}))}}$

[1 pt] Select the first input image, and add **Gaussian** and **Salt-Pepper** noise to the input image. Now, **your task** is to repeat the previous step (find Canny edges of the noisy images), and compute the aforementioned metrics with respect to the ground truth image. For Gaussian noise parameters, the mean should be set to 0 and standard deviation be 10. For Salt-pepper noise, the amount of standard deviation is interpreted as a percentage, and it should be set to 10%.

Question 3: Can we use Entropy for thresholding? [3 pts]

This question examines your knowledge on probability density function (pdf) construction from gray-scale histogram, and the use of Entropy information for mapping image intensity values into two classes: white (foreground) and black (background).

Definition 1: Entropy (i.e., H) is a measure of disorder (uncertainty), and it is zero for a perfectly ordered system. The entropy statistic is high if a variable is well distributed over the available range, and low if it is well ordered and narrowly distributed. (Entropy formulation is given in Lecture 3 notes).

Definition 2: The **Histogram** h of a digital image I is a plot or graph of the frequency of occurrence of each gray level in I. Hence, h is a one-dimensional function with domain $0, ..., L-1$ and possible range extending from 0 to the number of pixels in the image, N.
Histogram and *pdf* (probability density function) have the following direct relationship. Assume that the image I has a grayscale resolution of L gray levels. The number of pixels with gray level i is written as n_i, so the total number of pixels in the image is $N = n_0 + n_1 + ... + n_{L-1}$. Thus, the probability of a pixel having gray level i is:

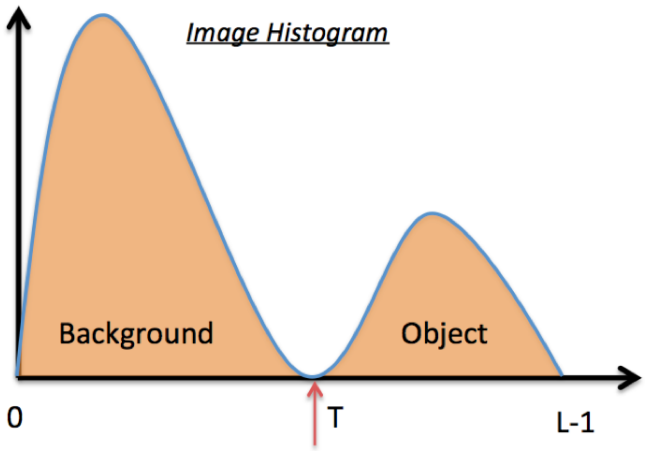
$$p_i = \frac{n_i}{N},$$

where $p_i \geq 0$ and $\sum_{i=1}^{L} p_i = 1$. Thus, normalized histogram is simply *pdf*.

Definition 3: Thresholding is an extreme form of gray level quantization. Suppose that a gray level image I can take L possible gray levels (i.e., for 8-bit, L=256). The process of thresholding is a simple comparison: each pixel value in I is compared to T, where $T \in 0, ..., L-1$. Based on this comparison, a binary decision is made that defines the value of the corresponding pixel (at location x,y) in an output binary image:

$$O(x, y) = \begin{cases} 1 & \text{if } I(x, y) \geq T \\ 0 & \text{otherwise.} \end{cases}$$

The threshold T is of critical importance, since it controls the particular abstraction of information that is obtained. Indeed, different thresholds can produce different valuable abstractions of the image. Image thresholding can often produce a binary image result that is quite useful for simplified processing, interpretation, or display. However, some gray level images do not lead to any interesting binary result regardless of the chosen threshold T. Several questions arise: given a gray level image, how can one decide whether binarization of the image by gray level thresholding will produce a useful result? Can this be decided automatically by a computer algorithm? Assuming that thresholding is likely to be successful, how does one decide on a threshold level T? These are apparently simple questions pertaining to a very simple operation. However, the answers to these questions turn out to be quite difficult to answer in the general case. In all cases, however, the basic tool for understanding the process of image thresholding is the image histogram (see figure below).



The concept of entropy thresholding is to threshold at an intensity for which the sum of the entropies of the two intensity probability distributions thereby separated is maximized. The reason for this is to obtain the greatest reduction in entropy–i.e., the greatest increase in order–by applying the threshold: in other words, the most appropriate threshold level is the one that imposes the greatest order on the system, and thus leads to the most meaningful result. To proceed, the intensity probability distribution is again divided into two classes (i.e., A and B)–those with gray levels above the threshold value T and those with gray levels above T. This leads to two probability distributions A and B:

$$A : \frac{p_0}{P_T}, \frac{p_1}{P_T}, ..., \frac{p_T}{P_T},$$
$$B : \frac{p_{T+1}}{1 - P_T}, \frac{p_{T+2}}{1 - P_T}, ..., \frac{p_{L-1}}{1 - P_T}.$$

where $P_T = \sum_{i=0}^{T} p_i$.

Your tasks are:

- Choose three example gray-scale images from Berkeley Segmentation Dataset (Training Images) CLICK HERE.
- For each possible $T \in 0, ..., L-1$, compute summation of entropy A and entropy B, called *total entropy*=H(A)+H(B). Note that A and B correspond to background and foreground of the image.
- Find the value of T corresponding to maximum total entropy $H(T) = H(A) + H(B)$.
- Plot each of three gray scale images (I) and corresponding binary images (O) on the screen (binary images are obtained through thresholding gray-scale images at threshold level T) .

Question 4: Corner Detection [3 pts]

Definition: A corner in an image I is a pixel p where two edges from different directions intersect. Corners can be characterized by high curvature of intensity values.

In this question, you will implement three different versions of the corner detection algorithms for given three input images (input1.png, input2.png, and input3.png).

[1 pts] Implement corner detection algorithm based on **Hessian matrix** (H) computation. Note that Hessian matrix is defined for a given image I at a pixel p as

$$H_1(p) = \begin{bmatrix} I_{xx}(p) & I_{xy}(p) \\ I_{xy}(p) & I_{yy}(p) \end{bmatrix}, \tag{0.1}$$

such that eigen-decomposition (spectral decomposition) of this matrix yields two eigenvalues as: λ_1 and λ_2. If both λ_1, λ_2 are large, we are at a corner. Provide the detected corners in the resulting output images in color.

[1 pt] Implement Harris Corner Detection algorithm for the same input images you used in previous question. Rather than considering the Hessian of the original image I (i.e. second-order derivatives), we use the first-order derivatives of the smoothed version $L(p, \sigma)$ for some Gaussian filter with standard deviation $\sigma > 0$. Note that you need to construct the following matrix for every pixel p,

$$H_2(p, \sigma) = \begin{bmatrix} L_x^2(p, \sigma) & L_x(p, \sigma)L_y(p, \sigma) \\ L_x(p, \sigma)L_y(p, \sigma) & L_y^2(p, \sigma) \end{bmatrix}, \tag{0.2}$$

where L is obtained after smoothing I with Gaussian filter G. Now, instead of calculating those eigenvalues we computed in previous question, we will consider **the cornerness measure** as

$$\textbf{Cornerness}(p, \sigma, \alpha) = \mathsf{Det}(H_2) - \alpha.\mathsf{Tr}(H_2), \tag{0.3}$$

where Det and Tr indicate the determinant and trace of the matrix H_2, respectively. Please use non-negative $\alpha \approx 1/25$ as a starting value and try to optimize it by trying different values and comment about it. Provide the detected corners in the resulting output image in color.

[1 pt] In the previous question, replace **cornerness measure** with the following:

$$\textbf{Cornerness}(p, \sigma, \alpha) = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2), \tag{0.4}$$

and determine the efficiency of this system and the system in the previous question by measuring and reporting the time. You are supposed to get the same results in accuracy but different results in efficiency.

Question 5: SUSAN Corner Detection [3 Pts]

Smallest Univalue Segment Assimilating Nucleus (SUSAN): Smith and Brady (IJCV, Vol. 23(1), pp. 45–78, 1997) presented an entirely different approach to the 1D and 2D feature detection in images, such as edges and corners, respectively. The SUSAN corner detection algorithm does not require derivative operation; hence, it can work well with noisy images. The main idea of the SUSAN is the use of a mask to count the number of pixels having the same brightness as the center pixel. By comparing the number of pixel having the same brightness as the center pixel with a threshold, the detector can determine whether the center pixel is a corner.

Circular mask M consisting of 37 pixels should be used. The central pixel of the mask is called a nucleus. Then intensities of all pixels within a mask are compared with an intensity of a nucleus and an area of "similar" pixels is marked. This area is called USAN (Univalue Segment Assimilating Nucleus) and it conveys the most important information on a local structure of an image. Analyzing the size, centroid and the second moments of USAN the exact information on a type of local structure around a nucleus is inferred, such as edges or corners. For those regions, inverted USAN area shows strong peaks –thus the term SUSAN – i.e. the smallest USAN. This approach has an additional advantage of not using any derivatives which are cumbersome to use in the presence of noise.

Computing USAN for every pixel in the digital image leads to detection of edges or corners. The number of pixels of USAN $n(r_0)$ is computed as

$$n(r_0) = \sum_{r \in M} e^{-(\frac{I(r)-I(r_0)}{t})^6}, \tag{0.5}$$

where t is a threshold for a difference of brightness and r and r_0 are distances to a pixel and to a nucleus of M, respectively. The value of USAN gets smaller near the point of interest which are located at local maxima of the following value (a SUSAN principle):

$$R(r_0) = \begin{cases} g - n(r_0) \text{ for } n(r_0) \leq g \\ 0 \text{ for } n(r_0) > g, \end{cases} \tag{0.6}$$

where g is half of n_{max} value of a mask M. The SUSAN corner detection procedure is outlined as follows:

- Place a circular mask around a pixel (i.e., nucleus)
- Calculate the number $n(r_0)$ of pixels within the circular mask which have similar brightness to the nucleus in accordance with Equation 3.1. Such pixels constitute the USAN.
- Compute the strength of a corner from Equation 3.2.
- Use non-max suppression to find corners.

[1 pt] Implement SUSAN corner detection algorithms and test the provided images (*susan-input1.png*).Find the optimal (near-optimal) values of free parameters yourself.
[1 pt] Test the provided image *susan-input2.png* for corner detection procedure. Comment on the success of the SUSAN method with respect to noise.
[1 pt] Preprocess test image *susan-input2.png* before applying SUSAN algorithm, and comment on the results. Preprocessing steps can include denoising, smoothing, and normalization. Demonstrate how you can handle noisy images when detecting corners.