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

[Programming Assignment] (1)

Robot Vision

Dr. Ulas Bagci • (Spring) 2019 • 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 other IDLEs such as CANOPY, PyCharm Community Edition, PyScripter, CodeSculptor, Eric Python, Eclipse plus PyDev, etc.

Following libraries can be used when necessary:

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

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 **10th of February 2019**, 11.59pm. PA1 is 15% of your total grade.

Question 1: Box Filtering [1 pt]

Implement two box filters (one with 3 by 3, the other one is 5 by 5 kernel size), and apply them to the given images (i.e., image1.png and image2.png) separately. Show the resulting images, and explain the resulting images, and their differences. (For convolution operation, you can use built-in function. Do not use built-in function for box filtering.)

Question 2: Median Filtering [1 pt]

Implement three median filters (3 by 3, 5 by 5, and 7 by 7 kernel size), and apply these filters to image1.png and image2.png separately. Show (and discuss as comments) the resulting differences for each kernel on the screen, explain where median filters are most effective. (For convolution operation, you can use built-in function. Do not use built-in function for Median filtering.)

Question 3: Gaussian Filtering [1 pt]

Implement a two-dimensional Gaussian kernel with a variation (sigma) equal to 3, 5, and 10. Apply these three Gaussian kernels to image1.png and image2.png separately, show them on the screen, discuss the differences of Gaussian operations with different sigmas (as comments on the code). Also, compare your results with question 1 and question 2: what are the differences between these three filters, what do you observe (as comments on the code)? Which filtering is the most effective in which images ? Why ? (For convolution operation, you can use built-in function. Do not use built-in function for Gaussian filtering.)

Question 4: Gradient Operations [1 pt]

Write a derivative operations (forward, backward, and central difference) that can be applied to any given image (f) and produces two images: gradient x, and gradient y images (i.e., f_x, f_y). Also, calculate gradient magnitude as $\sqrt{(f_x^2 + f_y^2)}$. The image that you should use for this problem is called image3.png. Show the results on the screen (Do not use built-in gradient functions, create difference operator yourself as stated clearly in the question.)

Question 5: Sobel Filtering [1 pt]

Implement Sobel filtering with 3 by 3 kernel size. Apply it to image1.png and image2.png. Show the results on the screen, and discuss the resulting images (as comments on the code). (For convolution operation, you can use built-in function. Do not use built-in function for Sobel filtering.)

Question 6: Fast Gaussian Filtering [1 pt]

Implement question 3, but this time, use only 1D Gaussian to do filtering in 2D. The trick is the following: since Gaussians are separable, you can use 1D Gaussian to filter the image in one direction first, and then the other direction can be filtered. Show the results on the screen and compare the efficiency of both methods (question 3 and question 6) as comments on the code. Use image1.png and image2.png for smoothing and show results too. (For convolution operation, you can use built-in function. Do not use built-in function for fast Gaussian filtering.)

Question 7: Histogram [1 pt]

Implement histogram function from scratch, and show the resulting bar-graph (histogram). Use 256, 128, and 64 bins to visualize histograms. Comment on the resulting differences with respect to bins. Use image4.png to conduct this experiment. (Do not use built-in histogram function, create it yourself.)

Question 8: Can we use Entropy for thresholding? [1 pt]

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 for a given N samples can be formulated as follows: $H = \sum_{i=1}^{N} p_i . \log(p_i)$, where p_i means probability of the sample i.).

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:

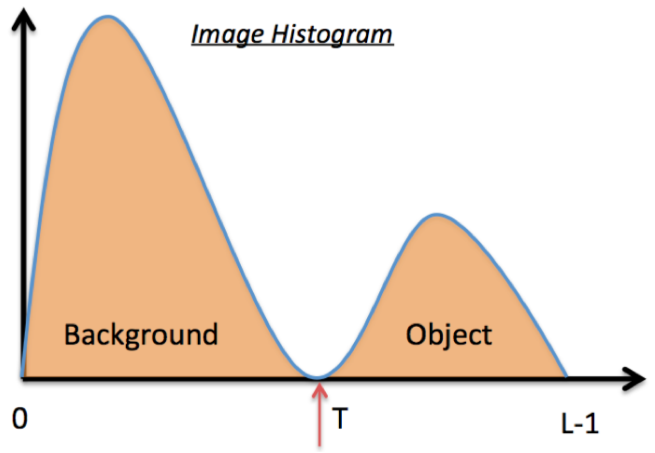
$$p_i = \frac{n_i}{N},$$

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

<u>**Your tasks**</u> are:

- Use the attached image for conducting this experiment (image4.png).
- 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) .

Canny Edge Detection [7 pt]

<u>**Your tasks:**</u>

0 pt Use two different images (canny1.jpg and canny2.jpg) to perform the following steps for getting Canny edges of the input image.

1 pts Use 1-dimensional Gaussians to implement 2D Gaussian filtering yourself (do not use built-in functions).

1 pts Obtain gradient images (x-dim, y-dim, gradient magnitude, and gradient orientation) by following the Canny algorithm that we have seen in the class. Show resulting gradient images on screen and in the report.

2 pts Implement non-max suppression algorithm to reduce some of the falsely detected edges in the gradient images (from the previous step). Show the improved edge map on the screen and in the report.

1 pts Implement hysteresis thresholding algorithm and use it to further enhance the edge map obtained from the previous step. Show the final Canny edge map on the screen and in the report.

1 pt 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.

1 pt Discuss about the different filtering approaches you took for four pictures. Since pictures are the same scene but different noise and smoothing patterns, you need to adjust your Canny edge filtering parameters to show similar results to Canny edges of the output-canny1.png and output-canny2.png.