CIS581, Computer Vision
Project 1, Image Edge Detection
Written part due on Sept. 12, 4:30pm
Programming part due on Sept. 19, 4:30pm

Overview

This project focuses on understanding image convolution and edge detection. The first 4 questions are to be done individually. Question 5 can be done with a partner. Unless otherwise stated the problems should be done by hand. All matlab functions should follow the names and arguments stated in the problems in order for them to run properly with the grading script. A test script will be provided shortly that will call your functions to ensure they will run inside the grading script.

To submit the assignment, zip your files into a folder named "1_<penn user name>" and email it to cis581.fall2013@gmail.com

Recall the definition of convolution, $J = I \otimes g$ in 1D as

$$J(i) = \sum_k I(i-k)g(k), \tag{1}$$

and in 2D as

$$J(i,j) = \sum_{k,l} I(i-k, j-l)g(k,l). \tag{2}$$

Typically, I is an image, g is a filter, and J is the filter response of I under g.

In the following exercise, we will use the following 10×10 image:

$$I = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0.5 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0.5 & 0.5 & 0.5 & 0.5 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 1 & 1 & 1 & 1 & 1 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3}$$

Let G be a 1D smoothing filter, with

$$\begin{aligned} G(0) &= a \\ G(1) = G(-1) &= 1/4 \\ G(-2) = G(2) &= 1/4 - a/2 \end{aligned}$$

You can verify that for any choice of a, $\sum_k G(k) = 1$. We can apply this 1D filter either in the horizontal or vertical direction, with $Gx(0,k) = G(k)$, or $Gy(k,0) = G(k)$. Let δ be a difference filter, with $\delta(0) = 1, \delta(1) = -1$, and δ_x, δ_y similarly defined.

Written Part

1. *Images reading and display.*

   This is a warm up exercise for Matlab image processing. Write a Matlab function J = myImcrop(I) that

   (a) reads in an image

   (b) displays the image in figure 1

   (c) prompts the user to crop out a sub-image

   (d) displays and returns the sub-image

Make sure to use the function heading: Function J = myImcrop(I);

Note "imcrop" is a built-in Matlab function. The goal of this exercise is to replace this matlab function with your own, using the "ginput" function.

2. *Verifying Convolution is indeed associative.*

   For the following filtering operation, use $a = 0.4$(Gaussian). Let $J_1 = (I \otimes G_x) \otimes G_y$. Let $G_{xy} = Gx \otimes Gy$, and $J_2 = I \otimes G_{xy}$. Compute J_1, J_2 using the image I defined above, and verify they are the same.

   In the convolution computation, you will encounter negative indexes, you should use the assumption: $I(-i, -j) = I(i, j)$. How many operations(addition and multiplication) are required for computing J_1, J_2?

   This exercise is intended to familiarize you with the convolution operation. You should hand compute some of the entries. However, carrying out the entire computation by hand might be over exhausting. You should consider using Matlab to work the remainder out.

3. *Image gradient*

   (a) Compute the image gradient $\Delta I = [I_x, I_y]$, where

   $$\begin{aligned} I_x &= I \otimes G_x \otimes \delta_x \otimes G_y \\ I_y &= I \otimes G_y \otimes \delta_y \otimes G_x \end{aligned} \tag{4}$$

   (b) Compute the magnitude of the image gradient $||\Delta I||_2$.

4. *How edges move after image smoothing*

   We can think of the images on our eyes as a noisy version of the ideal image signal from the scene. Consider a ray from a point on the scene which ideally should impinge the image at location x. Because of imperfections of the visual apparatus, the point in the image is with large probability at location x, with smaller probability it could be at the neighboring location, $x + dx$, and with even smaller probability at the even farther location, $x+2dx$. The uncertainties at every image point is mathematically similar to a Gaussian smooth function. Therefore, we can create the most likely image the vision system by smoothing the image. The main question is: does such smoothing affect our perception of the edges in the image? The following three cases are simplified images in the world:

(a) I = [0 0 0 0 0 1 1 1 1 1], $G_1 = G$.

(b) I = [0 0 0 0 0 1 1 0 0 0 0 0], $G_2 = G$.

(c) I = [0 0 0 0 0 0.5 0.5 0.5 1 1 1 1 1], $G_3 = G \otimes G$.

The edge locations is where the image gradient magnitude is at its local maximum. In the original signals, the edges are between 0 and 1, 0 and 0.5, 0.5 and 1. For each case, compute the smoothed image with the provided smoothing kernel. Compute the image gradient magnitude, and determine the edge location(s) after smoothing. Are the edge locations the same as before smoothing? If not, how do they change?

FAQ of the Written Part

Q: For the 'imcrop' question, does the prompt that the user sees to crop the sub-image have to be the same as 'imcrop'? Meaning the rectangle that is displayed is adjustable? Or do we just have the user select two points using "ginput" and whatever two points they select, will be the boundary of the cropped image.

A: Just having the user select two points is fine. There's no need to make the rectangle adjustable.

Q: For question 3 and 4, can we use matlab to get the final image gradient, assuming we show how to get a few points first?

A: You can use matlab after showing a few points by hand. In general, when it's asking for a convolution you can just do 3-5 points by hand in areas of interest (image boundaries, places where there's some kind of change in the image, etc) and do the rest in matlab.

Programming Part

5. *Edge detection*

   Write a Matlab function E = cannyEdge(I)

Compute the Canny edges for image I described above. Canny edges are defined as local maxima of the image gradient. Following the steps described in the lecture notes:

(a) compute local edge normal orientation,

(b) seek local maximum in the edge normal orientation,

(c) continue search in the edge orientation of detected edge point.

Pick three face images from your own image collection, and compute Canny edges for those images.

The function header should be function E = cannyEdge(I) where I is an image and E is a binary edge map in logical type with 1 on edge pixels and 0 elsewhere. You can use logical() for conversion.

Good test images can be found in the Berkeley Segmentation Dataset at http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/. Additionally they provide code to benchmark edge detectors.

If the image I is not smoothed, we expect the edges to be exactly at the boundary between the squares in image I.

Note Matlab has also a build-in edge detector called "edge" in the image processing toolbox. You can compare your result with Matlab's.

We have provided a test script for project 1 on the course website `https://alliance.seas.upenn.edu/~cis581/wiki/index.php?title=Projects`. Extract the contents of the file to the same directory as your functions and run "TestScript1.m" in matlab. This does not test the correctness of your code itself, only whether the output is in the expected format. Additionally, when grading we'll be calling your functions in the same manner, so make sure they work as you'd expect on the sample in the test script.

FAQ of the Programming Part

Q: How to set thresholds of hysteresis in Canny edge detection?

A: Vision is all about picking the 'right thresholds'. For this project, you should pick thresholds yourselves to get good performance. Since the thresholds aren't passed into the function, you either choose the thresholds hat work well across a wide range

of images or use some method to automatically determine what thresholds would be good based on the input image. Also, some implementations might have a lower threshold that is a function of the current edge length.

Q: Are we allowed to use functions like bwselect and bwmorph?

A: There's no restriction on what functions you can use. However, the edge linking step will be more involved than just calling those functions. Rather than checking all adjacent pixels to see if they're above a threshold, the edge linking should check in the direction perpendicular to the image gradient. In general we're not expecting everyone to implement the edge linking in the exact same way so if you have a method without recursion that performs well that's fine.