Homework 2, Computer Vision Fall 2018

Prof. Dr. Francesco Maurelli, Jacobs University Bremen

Handed out 10.10.2018, due 20.10.2018 23:55:00

Please use the moodle system to upload your homework (`moodle.jacobs-university.de`). The system will shut down at the deadline and homeworks will not be accepted at a later time. No other form of submission is allowed.
Please upload one single zip file, with a main pdf file which can guide the instructors through your work and the various Matlab files and images generated. Be as clear as possible.
Use of LaTeXis recommended, though not compulsory. Scans of hand-written papers are accepted as long as the writing is clearly understandable. Add your source code and result images when requested and properly reference them in the pdf file.

1 Linear Filters

In class, we introduced 2D discrete space convolution. Consider an input image $I[i; j]$ and an mxn filter $F[i; j]$. The 2D convolution $I * F$ is defined as

$$(I * F)[i, j] = \sum_{k,l} I[i - k; j - l]F[k, l]$$

Hint: The above operation is run for each pixel (i; j) of the result.

(a) Convolve the following I and F. Assume zero-padding where necessary.

$$I = \begin{bmatrix} 2 & 0 & 1 \\ 1 & -1 & 2 \end{bmatrix} F = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

*Important: do **not** use Matlab for this question. Please show all necessary steps to receive full credit.*

(b) Note that the F is separable; that is, it can be written as a product of two 1D filters: $F = F_1F_2$.

$$F_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} F_2 = \begin{bmatrix} 1 & -1 \end{bmatrix}$$

Compute $(I * F1)$ and $(I * F1) * F2$, i.e. first perform 1D convolution on each column, followed by another 1D convolution on each row.

*Important: do **not** use Matlab for this question. Please show all necessary steps to receive full credit.*

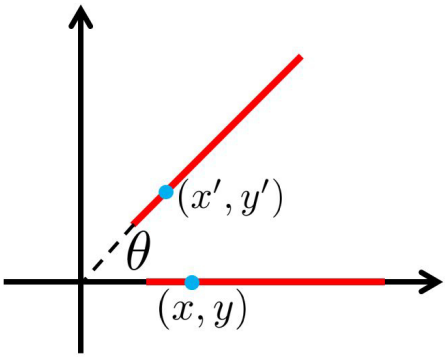
2 Normalized Cross-correlation

Background: In class, we covered cross-correlation, in which a template image is multiplied with sections of a larger image to measure how similar each section is to the template. Normalized cross-correlation is a small refinement to this process. In rough terms, it works like this: before multiplying the template with each small section of the image, the image section is scaled and offset so it has zero mean and variance of 1. This increases accuracy by penalizing image sections which have high intensity but do not match the pattern of the template. The MATLAB function `normxcorr2` performs this entire process for you.

(a) Use the provided `crossCorrelation.m` file to load the provided photo and template. Read the MATLAB documentation for `normxcorr2`, and use it to perform cross-correlation to find the section of the image that best matches the template. Include your MATLAB code in your writeup, and describe why the peak occurs where it does. Also explain the straight-line artifacts in the cross-correlation. You don't need to include the cross-correlation image itself in your report, as it may not print well.

(b) In `crossCorrelation.m`, perform cross-correlation using the larger template. Note that the larger template does not exactly match the image. Describe your results, and why they are different from part (a). What does this tell you about the limitations of cross-correlation for identifying objects in real-world photos? Also justify what applications cross-correlation would be good at. *Hint: The provided code auto-scales image brightness so that it covers the full range. Therefore, same colors do not necessarily have the same value in the two cross-correlation images.*

3 Canny Edge Detector



(a) Suppose that the Canny edge detector successfully detects an edge in an image. The edge (see Figure 1) is then rotated by θ, where the relationship between a point on the original edge (x, y) and a point on the rotated edge (x', y') is defined as:

$$x' = xcos\theta; y' = xsin\theta$$

Will the rotated edge be detected using the same Canny edge detector? *Hint: The detection of an edge by the Canny edge detector depends only on the magnitude of its derivative. The derivative at point (x, y) is determined by its components along the x and y directions. Think about how these magnitudes have changed because of the rotation.*

(b) After running the Canny edge detector on an image, you notice that long edges are broken into short segments separated by gaps. In addition, some spurious edges appear. For each of the two thresholds (low and high) used in hysteresis thresholding, explain how you would adjust the threshold (up or down) to address both problems. Assume that a setting exists for the two thresholds that produces the desired result. Briefly explain your answer.

4 Difference-of-Gaussian (DoG) Detector

(a) The 1-D Gaussian is

$$g_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/2\sigma^2}$$

Calculate its 2nd derivative with respect to x, and use Matlab to plot it (use $\sigma = 1$).

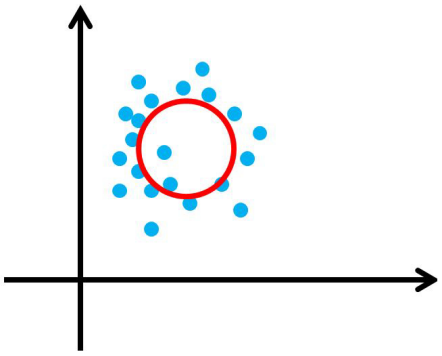
(b) Use Matlab to plot the difference of Gaussians in 1-D given by

$$D(x; \sigma; k) = \frac{g_{k\sigma}(x) - g_\sigma(x)}{k\sigma - \sigma}$$

using $k = 1.2; 1.4; 1.6; 1.8$ and 2.0. State which value of k gives the best approximation to the 2nd derivative with respect to x. You may assume that $\sigma = 1$.

Important: Submit your Matlab code either as m file or in the pdf as a code block.

5 RANSAC for Fitting Circles



In class, we discussed how to fit a line to a series of points using RANSAC. In this problem, you are going to develop an algorithm that uses RANSAC for fitting a circle to a group of points in two dimensional space (see Figure). *Important: Please submit your code and plots for this problem.*

(a) A circle C in 2D space is given by its center (c_x, c_y) and its radius R. Before we implement RANSAC, we need a way to fit a circle (c_x, c_y, R) to a chosen set of points (we will need at least 3 points to specify a circle, and with more points we can get an "average circle" that provides a better fit). Read the provided handout on least-squares model fitting. We will use least-squares to fit a circle to 3 or more points. Remember that, for each point i, a circle should satisfy the simple scalar equation:

$$R^2 = (x_i - c_x)^2 + (y_i - c_y)^2$$

From basic algebra, this can be rewritten as

$$R^2 = x_i^2 - 2x_i c_x + c_x^2 + y_i^2 - 2y_i c_y + c_y^2$$
$$2x_i c_x + 2y_i c_y + R^2 - c_x^2 - c_y^2 = x_i^2 + y_i^2$$

Note that our equations contain squares of unknown variables (R, c_x and c_y). Least squares can only handle linear equations, so this is a problem. However, we can define a new variable $q = R^2 - c_x^2 - c_y^2$. Using this trick, we have:

$$2x_i c_x + 2y_i c_y + q = x_i^2 + y_i^2$$

This is indeed a linear equation (note x_i^2 and y_i^2 are constants), so a set of these equations can be solved by least squares! Afterwards, we can recover the value of R^2 from q. Your task is to edit the provided `FitCircle.m` to set up a system of the above equations (one equation per input point) and solve the system using least squares. Run the provided `TestFit.m` to check your solution. The fit should be good on the first plot, but may be sensitive to outliers for the second plot. Include your code and the plots from a run of (TestFit.m) in your submitted solution.

(b) Using your `FitCircle()` function as a subroutine, complete the `RANSAC.m` function. Run the provided `TestRansac.m` to test your function. Include your code and the plots from a run of `TestRansac.m` in your submitted solution. *Hint: review the lecture slides on RANSAC and read through `RANSAC.m` from the beginning. Be sure to understand the meaning of each input argument before coding. Use the given functions to help you. Each "Your Code Here" spot should only take one or a few lines of code. Look at our default value for each variable to understand the format expected.*

(c) Above, we applied RANSAC to $N = 10$ points. Edit `TestRansac.m` to set $N = 1000$ and run it a few times to see the results. Briefly explain how the results are different and why. What RANSAC parameters could you change to improve the solution? You can modify the parameters in `TestRansac.m` to test your answer.