Problem Set 2

Assigned: 03/01/2005
Due: 03/10/2005

Please submit an electronic copy of your writeup and code for each problem to `6869-submit@csail.mit.edu`.

Problem 1 *Implement and Evaluate the DoG detector*

Background Reading

Read the reference paper, "Distinctive Image Features from Scale-Invariant Keypoints" by David Lowe. To understand how the DOG detector works, you should read at least Sections 3 and 4. The remaining sections talk about the SIFT descriptor and applications to object recognition. You don't have to implement "Accurate Keypoint Localization", the first part of Section 4, although you should read about and understand it.

(a) The Difference-of-Gaussian is a function of what variables? Explain what they are, don't just state their names.

(b) What trick is used to speed up scale selection in this paper?

(c) Suppose you are given σ_{cur}, or the current scale of the image. Derive an expression for σ_Δ, or the width of the Gaussian with which the image must be blurred to obtain the next scale sub-level in an octave. The expression should be in terms of σ_{cur} and k, the constant factor separating the scale sub-levels.

Hint: Use the Gaussian semi-group property:

$$g(\sigma_1^2) * g(\sigma_2^2) = g(\sigma_1^2 + \sigma_2^2)$$

where $g(\sigma)$ is a two-dimensional Gaussian with covariance σI_2.

Implementation

Implement the following functions in MATLAB. You should use the optimized parameters given in the paper, namely, use an initial blurring parameter `InitSigma` $= 1.6$, scale resolution `Scales` $= 3$ intervals per octave, Difference-of-Gaussian peak threshold `PeakThresh` = 0.03 and eigenvalue ratio threshold `R` = 10.

`function [InitSigma, Scales, PeakThresh, R] = get_dog_params`

Returns the constant parameters as described above. Call this function from inside the other functions to get the parameter values.

`function Points = compute_dog_points(Image)`

This is the main detector function. It should detect and display keypoints in the input image `Image`. First, double the image size and blur it to obtain the initial spatial resolution of `InitSigma`. You can assume that the doubled image already has blurring equivalent to $\sigma = 1.0$. Then, use the following functions as subroutines to search for the interest points. Finally, display the points on the image, using the provided function `draw_sigma` to show a circle with width proportional to the scale of the detected point. `Points` should be an $N \times 3$ matrix with the rows corresponding to the detected points (row, column and scale). Note, duplicate points that are detected within 1/2 a pixel of an existing point should be discarded.

`function [P, nextOctave] = process_octave(Octave, octSize)`

This function forms the sub-levels of the given octave and creates the difference images, or the Difference-of-Gaussian function. Octave is the first sub-level of the current octave, while `octSize` is the size of the octave relative to the original image. For example, in the first octave, the `octSize` is 0.5 because the image was doubled. You should use the `octSize` parameter to scale the points to the size of the original image. The function returns all the detected points (`P`) and the first level of the next octave (subsampled by 2.)

`function P = find_dog_peaks(DoG, octSize)`

Search over all given scales and locations in the DoG for one octave, checking if each point value is above a threshold. You must also check that the point is a local extremum and not an edge as discussed below. Return the detected points, P. Don't look too close to the border (minimum 5 samples) to avoid unstable points. Also, divide the `PeakThresh` parameter by `Scales`, since more closely spaced scale samples produce smaller DoG values. Use the following to functions as subroutines two `find_dog_peaks`:

`function bool = is_local_extremum(N)`

This function returns true if the given location is a local extremum at the given scale in a $3 \times 3 \times 3$ neighborhood of the point (see the paper for details.) You should try to make this check efficient.

$$\texttt{function A} = \texttt{not\_an\_edge}(\texttt{DoG}, \texttt{row}, \texttt{col}, \texttt{R})$$

Returns true if the eigenvalue ratio of the Hessian matrix for the given location and scale of the DoG is above the threshold `R`. To compute the 2×2 Hessian, use point differences computed around the 3×3 neighborhood of the given point.

$$\texttt{function Im} = \texttt{blurGauss}(\texttt{Im}, \texttt{sigma})$$

Returns the image after convolving it with a Gaussian kernel of width (standard deviation) `sigma`. You should use the `mkGaussian` and `rconv2` functions from the matlabPyrTools package. The Gaussian kernel should be truncated at 4 sigmas from the center.

In your submission, please include the m-file for each function named as `[function_name]-[last_name].m`*.*

Evaluation

You are given a test image `einstein.jpg` and the images `einstein0*.jpg`, which are scaled versions of the first image. Please answer the following questions.

(a) Run the detector on the test image `einstein.jpg` with the suggested parameter values provided above. Change the peak threshold `PeakThresh` to 0.08 and re-run your code on this image. Explain how this parameter affects the number of detected points. For the remainder of this problem use `PeakThresh` = 0.08.

(b) How many points are detected in each of the images? Hand in a printout of each image with the keypoints superimposed. Also include in your writeup the point location and scale in table format of each point detected in the original image `einstein.jpg`.

(c) How many of the points detected in the original image are still detected in the scaled images at the correct location and scale, i.e. what is the repeatability rate? Plot the number of points detected along with the number of correct detections for each test image. On a separate graph, plot the repeatability rate (# correct / # detected) for each test image as a function of its scale with respect to the original image.

(d) Examine how the `Scales` parameter influences repeatability. For `Scales` = 2,3,4,5 plot the resulting repeatability rate across all images on a single plot. What are your conclusions?

Problem 2 *Lukas-Kanade Optical Flow Tracker*

Lucas and Kanade's optical flow technique proposes to solve the brightness constant constraint equation (BCCE) by assuming constant flow over a patch:

$$\begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = - \begin{pmatrix} \sum I_x I_t \\ \sum I_y I_t \end{pmatrix} \quad (1)$$

Implement a single-iteration Lucas-Kanade tracker. Your function should have the following syntax:

$$\texttt{function}[\texttt{dx},\texttt{dy}] = \texttt{lucaskanade}(\texttt{I1},\texttt{I2},\texttt{xl},\texttt{yt},\texttt{xr},\texttt{yb},\texttt{wsize})$$

where `I1` and `I2` are the iamges, `(xl, yt)` and `(xr, yb)` are the top-left and bottom-right corners of the region where you want the optical flow computed, and `wsize` is the size of the squared image patch used to solve equation (1).

Please include the file `lucaskanade-[last_name].m` *in your submission.*

Test your implemenation using `LK-0001.bmp` and `LK-0002.bmp` with different window sizes: 5, 9, 13, and 17.

(a) For each window size, plot the optical flow for the region `(xl, yt)`;`(xr, yb)` = `(80,170)`;`(150,220)` using the MATLAB function `quiver`. To better visualize the flow field `(dx,dy)`, sub-sample it by keeping only every fifth flow value in both the x and y directions.

(b) Is the estimated optical flow constant over the whole region? Should the true optical flow differ from your estimate? Why or why not?

(c) Does the window size change the estimated optical flow? Why or why not?

Hint: To compute the image gradient you should first blur image `I1` *and then use the MATLAB function* `gradient`*. The temporal gradient can be approximated by the difference between blurred versions of* `I1` *and* `I2`*.*