http://crcv.ucf.edu/people/faculty/Bagci/

[Programming Assignment] (2)

Computer Vision

Dr. Ulas Bagci • (Fall) 2015 • 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 **29th of September 2015**, 11.59pm.

1 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

$$\mathsf{H}_1(\mathsf{p}) = \begin{bmatrix} \mathsf{I}_{xx}(\mathsf{p}) & \mathsf{I}_{xy}(\mathsf{p}) \\ \mathsf{I}_{xy}(\mathsf{p}) & \mathsf{I}_{yy}(\mathsf{p}) \end{bmatrix}, \tag{1.1}$$

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 pts] 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,

$$H_2(p, \sigma) = \begin{bmatrix} L_x^2(p, \sigma) & L_x(p, \sigma)L_y(p, \sigma) \\ L_x(p, \sigma)L_y(p, \sigma) & L_y^2(p, \sigma) \end{bmatrix}, \tag{1.2}$$

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

$$\textbf{Cornerness}(p, \sigma, \alpha) = Det(H_2) - \alpha.Tr(H_2), \tag{1.3}$$

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 pts] In the previous question, replace **cornerness measure** with the following:

$$\textbf{Cornerness}(p, \sigma, \alpha) = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2), \tag{1.4}$$

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.

2 SIFT [7 pts]

Scale-invariant feature transform (SIFT) is an algorithm for detecting and describing local features in images (Lowe, D., IJCV Vol 64(2), pp.91–110, 2004). Key stages for SIFT are the following: (i) scale-space extrema detection, (ii) keypoint localization, (iii) Orientation assignment, and (iv) keypoint descriptor. In this question, you are asked to implement SIFT and show the results for given images *SIFT-test1.png* and *SIFT-test2.png*.

The steps for implementing the SIFT are given as follows (please show output of every step separately, make necessary comments in the source code):

i **Scale-space extrema detection (1 pt):** Point of interests are called keypoints in the SIFT framework. For an input image $I(x, y)$, you will implement Difference of Gaussian (DoG) as

$$D(x, y, \sigma) = L(x, y, \alpha_i\sigma) - L(x, y, \alpha_j\sigma),$$

where $L(x, y, \alpha_i\sigma)$ is the convolution of the original image $I(x, y)$ with Gaussian blur $G(x, y, \alpha_i\sigma)$ at scale $\alpha_i\sigma$. Figure 2.1 (left) shows scale-space representation in Gaussian pyramid (5-level). The convolved images are grouped by octave (an octave is constructed from set of images convolved with different values of σ), and the value of α_i is selected so that we obtain a fixed number of convolved images per octave. Then the DoG images are taken from adjacent Gaussian-blurred images per octave. Once DoG images have been obtained, keypoints are identified as local minima/maxima of the DoG images across scales. This is done by comparing each pixel in the DoG images to its eight neighbors at the same scale and nine corresponding neighboring pixels in each of the neighboring scales. If the pixel value is the maximum or minimum among all compared pixels, it is selected as a candidate keypoint (Figure 2.2).

ii **Keypoint localization (2 pts):** This stage attempts to eliminate more points from the list of keypoints by finding those that have low contrast or are poorly localized on an edge. Low contrast based removal: for each candidate keypoint, interpolation of nearby data is used to accurately determine its position. The initial approach was to just locate each keypoint at the location and scale of the candidate keypoint.

- The interpolation is done using the quadratic Taylor expansion of the DoG scale-space function $D(x, y, \sigma)$ with candidate keypoint as the origin. This Taylor expansion is given by

$$D(\mathbf{u}) = D + \frac{\partial D^T}{\partial \mathbf{u}} + \frac{\mathbf{u}^T}{2}\frac{\partial^2 D^2}{\partial \mathbf{u}^2}\mathbf{u},$$

  where D and its derivatives are evaluated at the candidate keypoint and $\mathbf{u} = (x, y, \sigma)$ is the offset from this point.
- The location of the extremum $\mathbf{u}'$ is determined by taking the derivative of this function with respect to $\mathbf{u}$ and setting it to zero:

$$\mathbf{u}' = -\left(\frac{\partial^2 D^{-1}}{\partial \mathbf{u}}\right)\left(\frac{\partial D}{\partial \mathbf{u}}\right).$$

  Note that, the Hessian and derivative of D are approximated by using differences of neighboring sample points. The resulting 3x3 linear system can be solved with minimal cost. If the offset $\mathbf{u}'$ is larger than 0.5 in any dimension, then it means that the extremum lies closer to a different sample point. In this case, the sample point is changed and the interpolation performed instead about that point. The final offset $\mathbf{u}'$ is added to the location of its sample point to get the interpolated estimate for the location of the extremum. The function value at the extremum, $D(\mathbf{u}')$, is useful for rejecting unstable extrema with low contrast. In this question, all extrema with a value of $|D(\mathbf{u}')| < 0.03$ will be discarded.
- To eliminate extrema based on poor localization, it is noted that in these cases there is a large principle curvature across the edge but a small curvature in the perpendicular direction of the DoG function. If this difference is less than the ratio of largest to smallest eigenvector, from the Hessian matrix (H) at the location and scale of the keypoint, the keypoint is rejected. Simply, compute $R = Tr(H)/Det(H)$, and then if R for a candidate keypoint is larger than $(r+1)^2/r$, that keypoint is poorly localized and hence rejected. For practical purposes, $r = 10$ can be used.

iii **Orientation assignment (2 pts):** Each keypoint is assigned one or more orientations based on local image gradient directions. This is the key step in achieving invariance to rotation as the keypoint descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation. After computing gradient magnitude m and orientation θ as

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

and

$$\theta(x, y) = atan(\frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}),$$

respectively, orientation histogram should be formed. Note that orientation histogram will include 8 angles (bins). Then, locate the highest peak in the histogram, and use this peak and any other local peak within 80% of the height of this peak to create a keypoint with that orientation. Some points may be assigned multiple orientations. Finally, fitting a parabola to the 3 histogram values closest to each peak (to interpolate the peaks position) will finish the orientation assignment.

iv **Keypoint descriptor (2 pts):** The local gradient data, used above, is also used to create keypoint descriptors. The gradient information is rotated to line up with the orientation of the keypoint and then weighted by a

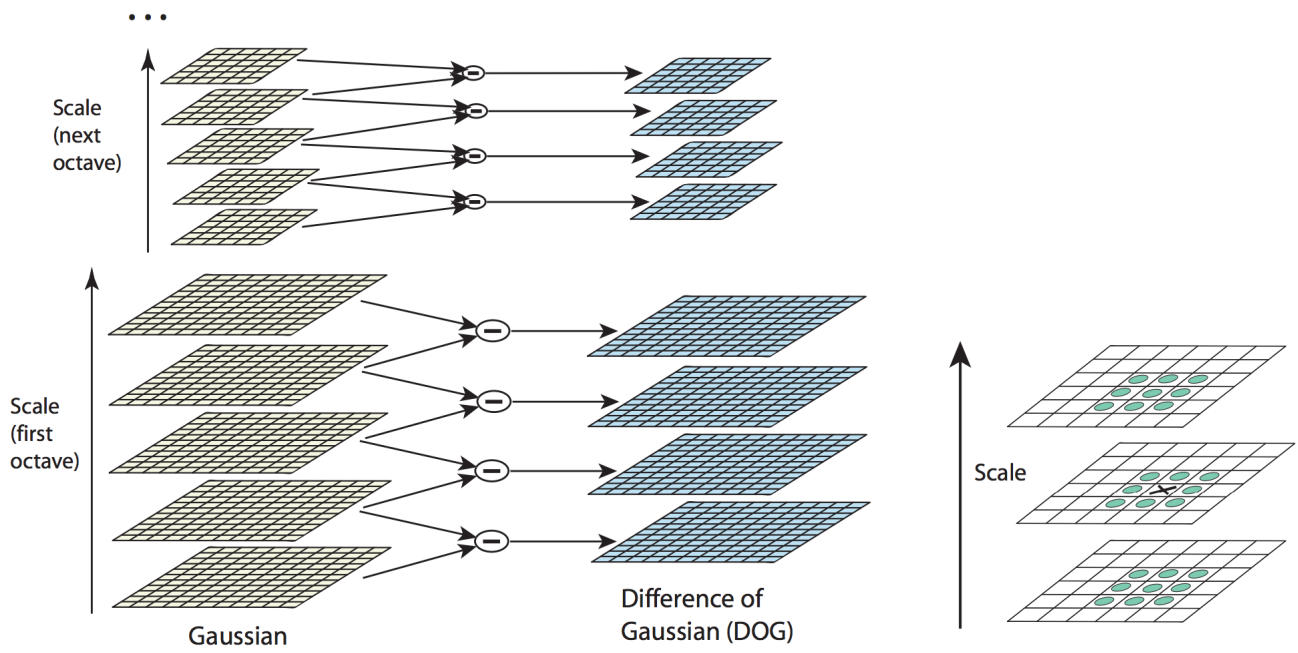

Figure 2.1: Left: For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the DoG images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeated. **Right:** Maxima and minima of the DoG images are detected by comparing a pixel (marked with X) to its 26 neighbors in 3x3 regions at the current and adjacent scales (marked with circles).

Gaussian with variance of $1.5 * \sigma$. This data is then used to create a set of histograms over a window centered on the keypoint. Keypoint descriptors typically uses a set of 16 histograms, aligned in a 4x4 grid, each with 8 orientation bins. This results in a feature vector containing 128 elements.

These resulting vectors are known as SIFT keys. Your task is to use the SIFT-input1.png and SIFT-input2.png images to identify SIFT keys first, then comment on the results. You are not going to match the keys, but visually check the keys in both images. Notice that SIFT-input2.png image is rotated, scaled, translated version of the SIFT-input1.png file, and there are some illumination differences as well.

3 BONUS [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

$$n(r_0) = \sum_{r \in M} e^{-(\frac{I(r)-I(r_0)}{t})^6}, \tag{3.1}$$

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):

$$R(r_0) = \begin{cases} g - n(r_0) \text{ for } n(r_0) \leq g \\ 0 \text{ for } n(r_0) > g, \end{cases} \tag{3.2}$$

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 pts] Implement SUSAN corner detection algorithms and test the provided images (*susan-input1.png*).Find the optimal (near-optimal) values of free parameters yourself.
[1 pts] Test the provided image *susan-input2.png* for corner detection procedure. Comment on the success of the SUSAN method with respect to noise.
[1 pts] 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.