Assignment_2 Solution

1. Noise and filtering

a. The signal-to-noise power ratio is defined as the ratio of the variances of the signal and noise:

$$\mathrm{SNR} = \frac{E_s}{E_n} = \frac{\sigma_s^2}{\sigma_n^2} = \frac{\frac{1}{n}\sum_{i,j}(I(i,j) - \bar{I})^2}{\sigma_n^2},$$

where σ_s^2 is the variance of pixels in a sequence of images and σ_n^2 is the variance for multiple frames of a static scene or the variance in uniform image area.

b.

- The Gaussian noise is statistical noise having a probability density function equal to that of the normal distribution. The impulsive noise is caused by sharp and sudden disturbances in the image signal.
- The median filter handles better impulsive noise because it can avoid outliers of noise.

c. $(1 \times 2) \times (3 \times 3) = 18$ (If using zero padding: boundary corner = 8, other boundary pixels = 12)

d. Instead of computing the image derivative first and then convolving with the filter, we take the derivative of the filter first, and then convolve the result with the image:

$$\frac{d(I * f)}{dx} = \frac{dI}{dx} * f = I * \frac{df}{dx}$$

(Also accept) Get the filter equivalent to the two filters, that is the derivative filter D and the other one G, the result is the convolution of the two filters $D * G$.

e.

- Zero padding: extend by padding zeros along the borders.
- Replication padding: extend by replicating the values along the borders.
- Ignore: not compute the convolution of the boundary pixies.
- Reflection padding: reflect the input values across the border axis.

f.

- $\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$
- The sum of all entries should be 1.
- If the sum is not 1, the image intensity will become darker or lighter than before.

g.

- Instead of convolving with a 2D Gaussian directly, we convolve with 1D Gaussian along rows then along columns: $I_G = I * G = I * G_x * G_y$
- Two 1D filters option is more efficient. Suppose the image shape is $H \times W$ and filter shape is $m \times n$. The cost of two 1D filters is $O(H \times W \times (m + n))$, while the cost of one 2D filter is $O(H \times W \times m \times n)$.
- No, it is not possible to implement any 2D filter in this way. (SVD) If only the first singular value σ_0 is non-zero, the kernel is separable and $\sqrt{\sigma_0}u_0$ and $\sqrt{\sigma_0}v_0^\top$ provide the vertical and horizontal kernels.

h. $\text{Size} \geq 5\sigma = 10$. We prefer an odd-sized filter, so the size should be 11.

i.

- The process of each layer is image $\rightarrow$ smoothing $\rightarrow$ downsample.
- It is hard to choose an appropriate window size in detection. However, if we use Gaussian pyramid, we can use the same size of window with different sizes of the image.
- The cost of a Gaussian pyramid is $m^2 + m^2/4 + m^2/16 + \cdots < 4m^2/3$ and the cost of a single image is m^2. Therefore, the amount of additional processing is $m^2/3$.

j.

- A Laplacian pyramid is very similar to a Gaussian pyramid, but saves the difference image of the blurred versions between each level. Only the smallest level is not a difference image to enable reconstruction of the high-resolution image using the difference images on higher levels.
- This technique is useful for image compression.

---

2. Edge detection

a.

- Because it can detect the change in the image which can help get the feature of the image.
- Desired properties:
  - Edges are corresponding to scene elements.
  - Invariant to illumination, pose, viewpoint, scale.
  - Consistent detection.

b.

- Basic steps of edge detection:
  1. Smoothing.
  2. Enhancement edges.
  3. Detection edges.
  4. Localization edges.
- The need for:
  - Smoothing: smoothing filters convolve with image to reduce noise in the image.
  - Enhancement: emphasizing pixels where there is a significant change in local intensity values and is usually performed by computing the gradient magnitude.
  - Localization: find the exact location of the edge.

c.

- Filters for computing the gradient: forward difference, central difference, Sobel, and Gaussian derivative.
- An image gradient is a directional change in the intensity or color in an image.
- The image gradient can detect the change of image such as edge and corner.

d. Sobel filter can be produced by convolving a smoothing filter with a directional derivative filter.

$$\underbrace{\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}}_{\text{smoothing}} * \underbrace{\begin{bmatrix}-1 & 1\\ -1 & 1\end{bmatrix}}_{x \text{ direction derivate}} = \underbrace{\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}}_{x \text{ direction of Sobel filter}}$$

$$\underbrace{\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}}_{\text{smoothing}} * \underbrace{\begin{bmatrix}1 & 1\\ -1 & -1\end{bmatrix}}_{y \text{ direction derivate}} = \underbrace{\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}}_{y \text{ direction of Sobel filter}}$$

e.

- Suppose we have a discrete function $f[x]$ and we want to compute its derivative, $f'[x]$. To obtain a more accurate result,

  1. Reconstruct the continuous function $f(x)$ from its discrete function by $f(x) = f[x] * h(x)$, where $h(x)$ is the sinc function.
  2. Calculate the derivative of this continuous function as $f'(x) = (f[x] * h(x))' = f[x] * h'(x)$.
  3. Finally, we sample $f[x] * h'(x) \Rightarrow f[x] * h'[x]$ to get the derivative of $f[x]$.

  In practice, we use Gaussian $G(x)$ to approximate $h(x)$ and $h'(x) \approx G'(x)$ which is the derivative of Gaussian function.

- $I_x = I * G'[x] * G[y]$ and $I_y = I * G'[y] * G[x]$

  $G[x] = e^{-x^2/2\sigma^2}$ and $G'[x] = -\frac{x}{\sigma^2}e^{-x^2/2\sigma^2}$

  $G[y] = e^{-y^2/2\sigma^2}$ and $G'[y] = -\frac{y}{\sigma^2}e^{-y^2/2\sigma^2}$

  Given $\sigma = 2$, the kernel size is $5\sigma = 10$. We prefer an odd-sized filter, so the size should be 11:

  $G[x] = (G[y])^\top = (1/4.98) \times [0.04, 0.14, 0.32, 0.61, 0.88, 1, 0.88, 0.61, 0.32, 0.14, 0.04]$

  $G'[x] = (G'[y])^\top = [0.055, 0.135, 0.243, 0.303, 0.221, 0, -0.221, -0.303, -0.243, -0.135, -0.055]$

f.

The first-order derivative captures a significant rise or fall in pixel values. The derivative will be at a max or min around the edge. A threshold can be used to identify that an edge is somewhere in the vicinity. Depending upon the threshold, it can be a rather wide vicinity.

The second-order derivative captures the actual peak of the change of value. It is observed as a root, or zero-crossing, of the second-order derivative. This gives a reliable and standardized way of declaring exactly where the edge is.

g.

- LoG filter: $\nabla^2 G = \frac{r^2 - 2\sigma^2}{\sigma^4}e^{-r^2/2\sigma^2}$, where $r^2 = x^2 + y^2$.

  Given $\sigma = 1$, the kernel size is $5\sigma = 5$ and $\nabla^2 G = (r^2 - 2)e^{-r^2/2}$

$$\nabla^2 G = \begin{bmatrix}0.110 & 0.246 & 0.271 & 0.246 & 0.110\\ 0.246 & 0 & -0.607 & 0 & 0.246\\ 0.271 & -0.607 & -2 & -0.607 & 0.271\\ 0.246 & 0 & -0.607 & 0 & 0.246\\ 0.110 & 0.246 & 0.271 & 0.246 & 0.110\end{bmatrix}$$

- Steps of using LoG to detect edges:

  1. Compute LoG: $H = I * \nabla^2 G$
  2. Threshold to classify each pixel $E(i, j) = 1$ if $H(i, j) \geq 0$ or $E(i, j) = 0$ if $H(i, j) < 0$

3. Scan the result binary image left-to-right and top-to-down, and mark edges at transitions 0 to 1 and 1 to 0.

h.

- The Canny detection uses the directional derivative of the image, and the standard detection uses the gradient of the image.
- If $|\nabla(I * G)| > \tau$, then detect edges at the maximum of $|\nabla(I * G)|$ along the direction of $\nabla(I * G)$.

i.

- Non-Maximum Suppression: The image is scanned along the image gradient direction, and if pixels are not part of the local maxima they are set to zero. This has the effect of suppressing all image information that is not part of local maxima.
- Hysteresis thresholding: We set two thresholds τ_h and τ_l, where $\tau_h > \tau_l$. A candidate edge is considered an edge if either its strength is above τ_h or its strength is above τ_l and it is (recursively) connected to a previously detected edge:

  1. Make array of visited pixels, $v(i, j) = 0$
  2. Scan image from left-to-right and top-to-down:

     If $!v(i, j) \wedge |\nabla(I * G)| > \tau_h$:

     set $v(i, j) = 1$ and track an edge using τ_l

     Track means searching neighbors orthogonal to gradient.

---

3. Corner detection

a.

- Check the local neighborhood to see if there are more than one dominant direction or orientation of gradient.
- The number of principal directions is derived as follows:

  1. Build correlation matrix of gradient in local neighborhood.
  2. Compute eigenvalues of correlation matrix.
  3. Count the number of sufficiently large eigenvalues.

  OR Plot the gradient orientation histogram and count how many peaks it has.

b. Given $\{g_i\}_{i=1}^n$, we want to find the direction v that will minimize the projection of all gradients $\{g_i\}$:

$$
\begin{cases} E(v) = \sum_{i=1}^n (g_i \cdot v)^2 = v^\top \left(\sum_{i=1}^n g_i g_i^\top\right) v \\ v^* = \arg\min_v E(v) \end{cases}
$$

$$
\Longrightarrow
$$

$$
\nabla E(v) = 0
$$

$$
Cv = 0, \text{where } C \equiv \sum_{i=1}^n g_i g_i^\top
$$

The solution is the eigenvector of matrix C belonging to the smallest eigenvalue.

c. $\begin{bmatrix} 4 & 6 \\ 6 & 44 \end{bmatrix}$

d. $\lambda_1 \lambda_2 > \tau$, where λ_1 and λ_2 are the eigenvalue of gradient correlation matrix, and τ is the threshold.

e. How Non-Maximum Suppression works:

1. Get λ_1 and λ_2 for each point.
2. Compute $\lambda_1\lambda_2$ and sort in decreasing order.
3. Select top point as corner and delete the neighboring points.
4. Stop deleting corner candidates after a desired percentage of the corners have been detected.

f. $\text{Cornerness}(C) = \det(C) - k \times \text{tr}^2(C)$, where $\det(C)$ is $\lambda_1\lambda_2$ and $\text{tr}(C)$ is $\lambda_1 + \lambda_2$

g.

- The formula for better localization: $p^* = C^{-1}\sum_i \nabla I(x_i)\nabla I(x_i)^\top x_i$
- Explanation: Connect each point x_i to the corner point p and project the gradient at x_i onto $(x_i - p)$. The optimal p will minimize the sum of all projections:

$$
\begin{cases} E(p) = \sum_i (\nabla I(x_i) \cdot (x_i - p))^2 \\ p^* = \arg\min_p E(p) \end{cases}
$$
$$
\Longrightarrow
$$
$$
\nabla E(p) = 0
$$
$$
p^* = C^{-1}\sum_i \nabla I(x_i)\nabla I(x_i)^\top x_i,
$$

  where x_i is the position, $\nabla I(x_i)$ is the gradient at position x_i, and C is the gradient correlation matrix.
- C must be a non-singular matrix.

h.

- Feature points characterization using HoG:
  1. Take a window and split it into blocks.
  2. Compute the histogram of gradient orientations.
  3. Concatenate histogram.
- A good characterization of feature points should be translation, rotation, scale, and illumination invariant.

i. SIFT features are formed by computing the gradient at each pixel in a 16×16 window around the detected keypoint. The 16×16 window is further divided into sixteen 4×4 windows. Within each 4×4 window, gradient magnitudes and orientations are calculated. These orientations are put into an 8-bin histogram. The magnitudes are further weighted by a Gaussian function with σ equal to one-half the width of the descriptor window. The descriptor then becomes a vector of all the values of these histograms. Since there are $4 \times 4 = 16$ histograms each with 8 bins, the SIFT feature has 128 elements.