CS512
AS2

Khalid Saifullah
A20423546

17th Feb 2023
Spring 2023

1-a) Signal to noise ratio (SNR) can be estimated as

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

Here, $I(i,j)$ is the pixel intensity
$\bar{I}$ is the average intensity of all pixels.

There are two ways to calculate variance of the noise (σ_n^2):

i) If the scene is static: Take multiple images of the scene. Look at the same pixel at some location across the images. The variance can be computed along this dimension. First i calculate the mean of all the images then i compute the differences from ~~me~~ the mean, sum the square of differences. Then divide it by n-1 or N (no. of images).

ii) If the scene is not static: Find the region that is roughly uniform. Compute $\bar{I}$ from this region. Use $\bar{I}$ to compute σ_n^2.

b) In gaussian noise, a normally distributed random value is added to each pixel. But in case of impulsive noise random pixels are replaced by extremely dark or bright values.

Median filter handles better the impulsive noise since median is not sensitive to the outlier.

c)

1	1	1
1	1	1
1	1	1

Filter

$*$

2	2	2	2
2	2	2	2
2	2	2	2
2	2	2	2

Image

$=$

18	18	18	
18	18	18	
18	18	18	

d) Computing the derivative of an image (I) convolved with filter F is the same as computing the convolution of the image with the derivative of the filter.

e) Three different ways to handle boundaries during convolution:
1. Zero Pad
2. Mirror/Replicate
3. Ignore

1) During zero padding i create extra row and column of zeros in the original image so as to maintain original image size after convolution. The amount of zero padding depends on the size of the filter.

2) During Mirror/Replicate, the first and last rows and columns are replicated.

3) During Ignore, no padding is applied, and the size of the image decreases after convolution.

f) A basic 3x3 smoothing filter:

$\frac{1}{9}$ | 1 | 1 | 1 |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |

$\Leftrightarrow$

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Sum of all the entries in the filter is 1. When the filter is applied to the image it averages the intensifies of pixels under the filter.

The reason for the sum to be selected as it is, it looks at the immediate neighbors in image and produces a smoother image.

g) A 2D convolution with a gaussian can be implemented using two 1D covolution filters. This separate implementation is the efficient method. Here, we convolve with 1D Gaussian along rows and then columns, or vice versa.

$$I_G = I * G = \sum_i \sum_j I(i,j)\, e^{-\frac{i^2+j^2}{2\sigma^2}}$$

$$= \sum_i e^{-\frac{i^2}{2\sigma^2}} \sum_j I(i,j) e^{-\frac{j^2}{2\sigma^2}}$$

$$= (I * G_y) * G_x$$

Let the image size be $M \times N$ and the filter size be $m \times n$.

for one 2D pass, no. of ~~parameters~~ operations $= M \times N \times m^2$

For a two 1D pass, no. of operations $= 2 \times M \times N \times m$

Hence, the no. of operations of two 1D filters is less than one 2D filter.

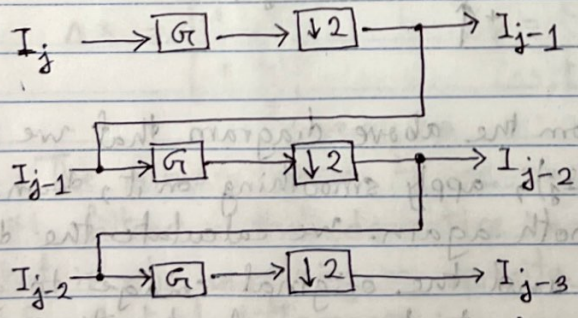
h) $m = 5\sigma$

$= 5 \times 2$

$= 10$

Hence, filter size is 10×10

i)



Pyramid

I_j	$m \times m$
I_{j-1}	$m/2 \times m/2$
I_{j-2}	
⋮	
I_0	

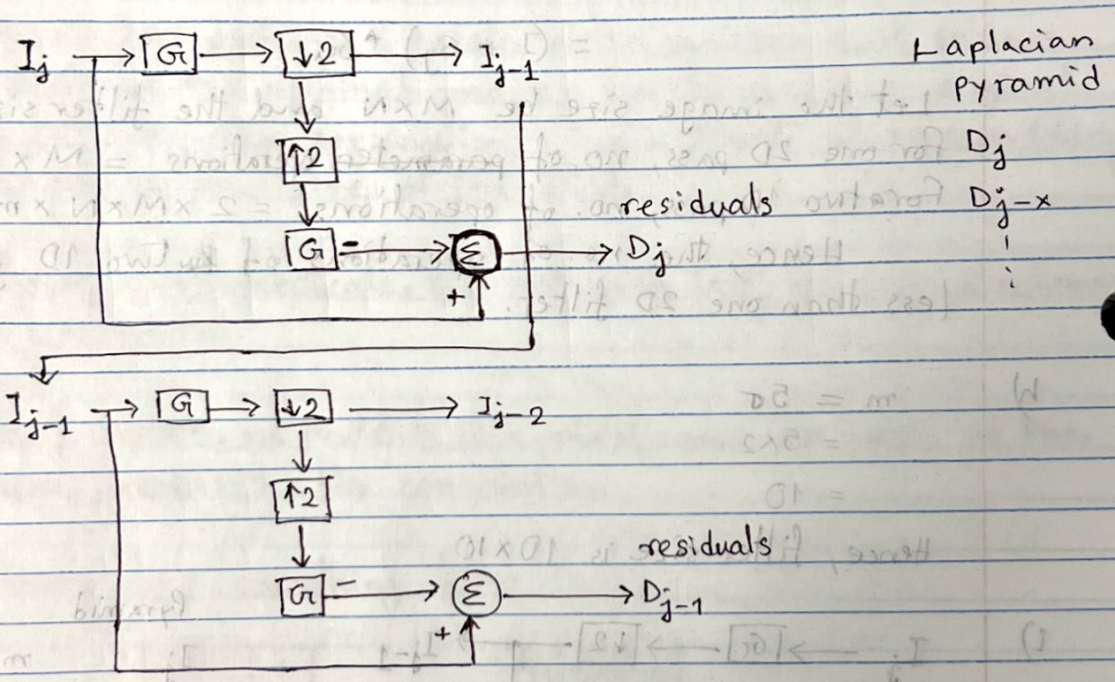
From the above diagram, we can see that the input image is smoothed with a gaussian filter and then downsampled to get I_{j-1}. This process is continued for a number of steps and we end up producing an image pyramid.

Gaussian image pyramid is helpful to do multiscale analysis.

We don't require to use different filter sizes to detect objects of different sizes. One filter size is sufficient for this purpose. Suppose we have a filter of size 3×3 in I_{j-1}, this is equivalent to filter of size 6×6 in the original image (I_j).

Number of pixels in the pyramid is only 33% higher than the number of pixels in the original image.

i)



Laplacian pyramid

D_j

D_{j-x}

It can be seen from the above diagram that we start with the original image (I_j), apply smoothing on it, downsample it, upsample it and smooth again. We calculate the difference between this image and the original image to get D_j (residual). In the next step, we start with I_{j-1} and perform the same operations to get a new ~~image~~ residual image (D_{j-1}). This process is repeated a certain number of times to generate a pyramid of residuals.

Laplacian pyramid is useful for image compression

2. a) Edge detection is useful to detect the shape of the object. Desired properties of edge detection are:
   i) It should correspond to seene elements
   ii) It should be invariant to illumination, pose, viewpoint, scale.
   iii) Reliable detection.

b) Basic steps of image detection are:
   i) Smoothing: In this step we remove noise from the image without blurring the edges.
   ii) Enhancement: In this step, a filter is used to enchance the edge contrast of an image.
   iii) Detection: In this step, we detect edges in the image using first order derivatives, second order derivatives or directional derivatives.
   iv) Localization: In this step, we find the exact location of the edge in the image.

c) Two filters for computing the image gradient are:
   i) For forward difference:

$$\Delta x = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \quad \text{(derivative →, averaging ↓)} \qquad \Delta y = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \quad \text{(averaging →, derivative ↓)}$$

The primary reason to use 2×2 filter instead of using 1×2 and 2×1 filters is that derivative has the tendency to amplify noise. That is why smoothing in opposite direction gives more stable results.

   ii) For Central differences:

$$\Delta x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{(derivative →, averaging ↓)} \qquad \Delta y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad \text{(averaging →)}$$

Image gradient shows the magnitude and direction of the

change of intensity. It is used to create edge map as follows:

$$E(i,j) = \begin{cases} 1 & \text{if } |\nabla I(i,j)| > \tau \\ 0 & \text{otherwise} \end{cases}$$

N.B.: If the image is dark, τ should be low.
If the image is bright, τ should be high.

d)

$$\Delta x = \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}}_{\text{smooth filter}} * \underbrace{\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}}_{\text{derivative filter}} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \begin{matrix} \downarrow \text{average} \\ \rightarrow \text{derivative} \end{matrix}$$

$$\Delta y = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \begin{matrix} \downarrow \text{derivative} \\ \rightarrow \text{average} \end{matrix}$$

e) Let $F(x)$ denotes original image
$F[x]$ denotes sampled image
$F'(x)$ denotes derivative of original image
$F'[x]$ denotes derivative of sampled image
$h(x)$ denotes perfect interpolator.

We can convolve $F[x]$ with $h(x)$ to get $F(x)$. We can then take the derivative of $F(x)$ to get $F'(x)$. $F'(x)$ can be sampled to get $F'[x]$.

But this process is not efficient because:
i) $h(x)$ is infinite and convolution with infinite filter is expensive.
ii) Instead of taking the derivative of the convolution with a filter, we can convolve with the derivative of the filter.

Gaussian function satisfies both the above requirements. Hence, it is used as an approximation for perfect interpolator.

Using separable properties of Gaussian:

$I_x = I * G'[x] * G[y]$

$I_y = I * G'[y] * G[x]$

$G(x) = e^{-\frac{x^2}{2\sigma^2}}$ | $G(y) = e^{-\frac{y^2}{2\sigma^2}}$

$G'(x) = -\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$ | $G'(y) = -\frac{y}{\sigma^2} e^{-\frac{y^2}{2\sigma^2}}$

For $\sigma = 2$, we have:

x	-2	-1	0	1	2
G(x)	$e^{-1/2}$	$e^{-1/8}$	1	$e^{-1/8}$	$e^{-1/2}$

x	-2	-1	0	1	2
G'(x)	$\frac{1}{2}e^{-1/2}$	$\frac{1}{4}e^{-1/8}$	0	$-\frac{1}{4}e^{-1/8}$	$-\frac{1}{2}e^{-1/2}$

y	-2	-1	0	1	2
G(y)	$e^{-1/2}$	$e^{-1/8}$	1	$e^{-1/8}$	$e^{-1/2}$

y	-2	-1	0	1	2
G'(y)	$\frac{1}{2}e^{-1/2}$	$\frac{1}{4}e^{-1/8}$	0	$-\frac{1}{4}e^{-1/8}$	$-\frac{1}{2}e^{-1/2}$

f) To detect an edge using first order derivative of the image we use a threshold τ. If the gradient is bigger than τ, we detect an edge. In this approach, edges are a bit thick.

To detect an edge using second order derivative of the image we look at the zero crossing of the second derivative. In this case, we don't require a threshold (τ) but second order derivatives are more ~~second~~ sensitive to noise than first order derivative.

g) Given $G = e^{-\frac{r^2}{2\sigma^2}}$ $(r^2 = x^2 + y^2)$

$$\nabla^2 G = \frac{r^2 - 2\sigma^2}{\sigma^4} e^{-\frac{r^2}{2\sigma^2}}$$

For $\sigma = 1$,

$$\nabla^2 G = \frac{r^2 - 2}{1} e^{-\frac{r^2}{2}} = (r^2 - 2) e^{-\frac{r^2}{2}}$$

Alternatively, we can convolve laplacian with Gaussian as follows.

$$LOG = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} * \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Laplacian — Gaussian with $\sigma = 1$

Steps to detect edge using LOG:

i) Compute LOG: $H = (\nabla^2 G) * I$

ii) Threshold:

$$E(i,j) = \begin{cases} 0 & \text{if } H(i,j) < 0 \\ 1 & \text{if } H(i,j) \geq 0 \end{cases}$$

iii) Mark edges at transition $0 \to 1$, $1 \to 0$

scan left-to-right and top-to-bottom

	0	1	1		
0	0	0→1	1→0	0	0
0	1	0	0	0	0

b) In canny edge detection, we calculate second order directional derivative along the gradient. In order to detect edge either checks zero crossing of second derivative or just do ~~maximo~~ maximum of the first derivative.

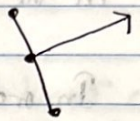
In the standard edges detection, we either calculate first order derivative along x and y direction and if the magnitude of gradient is higher than threshold then mark an edge, or we can calculate second order derivative along x and y direction and check for zero crossing to detect an edge.

i) In non-maximum suppression, we try to find local maximum of gradient magnitude in direction of gradient.

First, we calculate $\nabla I (I * G_\sigma) \equiv$ ~~[illegible]~~ (I_x, I_y)

$$\theta = \tan^{-1}\left(\frac{I_y}{I_x}\right)$$

$$\theta^* = \text{round}\left(\frac{\theta}{45}\right) * 45$$

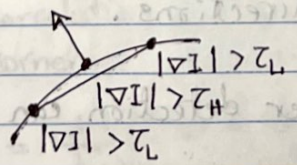


Now, we look at the neighbors in opposite direction. If there is maximum in the gradient value then that becomes an edge.

$$E(i,j) = \begin{cases} 1 & \text{if } \nabla(I * G_\sigma) \text{ is a local maximum} \\ 0 & \text{otherwise} \end{cases}$$

In Hysteresis thresholding, we use two thresholds τ_H and τ_L. τ_H is used to start tracking and τ_L is used to continue tracking. $\tau_H > \tau_L$) because we don't want to start tracking an edge at any of the noise point.

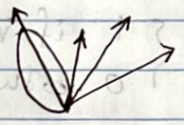
Once we start tracking an edge at a point with $|\nabla I| > \tau_H$, we look at the neighbor in the direction perpendicular to the gradient and check if $|\nabla I| > \tau_L$ for the neighbors.



3. a) In order to detect corner, we take a sliding window in the image. At each location of the window, we will ~~eke~~ check if there are multiple orientations of gradients. If there are more than one peak in the orientation histogram then there is a corner.

In PCA, first principal direction is the direction that maximizes the sum of projection of all the gradient vector. Second principal direction is orthogonal to first one. We continue in this manner until we capture enough variance in the data.

b) In order to find the principal directions of gradient orientations in a local path, we find the direction that minimizes the projections. We find the direction v such that the projection of $\{g_i\}$ onto v is minimized.



$$E(v) = \sum_i (g_i \cdot v)^2 = \sum_i (g_i^T v)(g_i^T v)$$

$$= \sum_i (v^T g_i)(g_i^T v)$$

$$= \sum_i v^T g_i g_i^T v$$

$$= v^T \left(\sum_i g_i g_i^T\right) v$$

$$= v^T C v$$

Additional directions minimize projection such that being orthogonal to previous directions.

c) Correlation matrix for corner detection can be written as:

$$C = \sum_i \underbrace{g_i}_{2\times1} \underbrace{g_i^T}_{1\times2} = \underbrace{\begin{bmatrix} \sum x_i^2 & \sum x_i y_i \\ \sum x_i y_i & \sum y_i^2 \end{bmatrix}}_{2\times2} \qquad g_i = [x_i \; y_i]^T$$

Gradient vectors are given as:

$\{(0,0), (0,1), (0,2), (0,3), (0,4), (1,0), (1,1), (1,2), (1,3)\}$

$\sum x_i^2 = 0+0+0+0+0+1+1+1+1 = 4$

$$\sum Y_i^2 = 0+1+4+9+16+0+1+4+9 = 44$$

$$\sum X_i Y_i = 0+0+0+0+0+0+1+2+3 = 6$$

$$C = \begin{bmatrix} 4 & 6 \\ 6 & 44 \end{bmatrix}$$

d) The condition on the eigenvalues of the gradient correlation matrix for corner detection is that their product should be greater than a certain threshold. If λ_1 and λ_2 are eigenvalues of the correlation matrix then

$$\lambda_1 \cdot \lambda_2 > \tau$$

then a corner is present in the window.

e) Algorithm for non-maximum suppression for corner detection:

i) Compute $\lambda_1 \cdot \lambda_2$ for all windows

ii) Select windows with $\lambda_1 \cdot \lambda_2 > \tau$ and sort in decreasing order

iii) Select the top of the list as corner, and delete all other corners in its neighborhood from the list

iv) Stop once detecting x % of the points as corners.

f) Algorithm for Harris Corner detection:

i) Compute correlation matrix C for window

ii) Compute cornerness measure

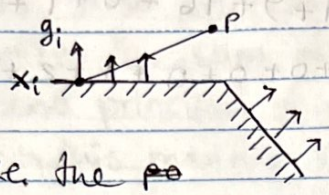
$$C(c) = \underbrace{\det(C)}_{\lambda_1 \cdot \lambda_2} - \underbrace{k\,tr^2(C)}_{k(\lambda_1+\lambda_2)^2}$$

K is a hyper-parameter

iii) Detect corners where $C(c)$ is high

We don't directly compute the eigenvalues but we calculate the determinant of C and trace of C.

g) In order to localize a corner, we follow the following procedure:



Let's take the point x_i and let g_i be its gradient. If $g_i \neq 0$ then we connect x_i with candidate point P. Now we take the dot product between g_i and $(x_i - P)$. If point P is really the corner, they should be perpendicular.

Similarly, we connect each point x_i to P and project the gradient at x_i onto $(x_i - P)$. The best P will minimize the sum of all projections.

$$E(P) = \sum_i \left(\nabla I(x_i) \cdot (x_i - P)\right)^2$$

$$= \sum_i (x_i - P)^T \nabla I(x_i) \nabla I(x_i)^T (x_i - P)$$

$$= \sum_i (x_i - P)^T \left(\nabla I(x_i) \nabla I(x_i)^T\right)(x_i - P)$$

$$P^* = \arg\min_P E(P)$$

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

$$-2 \sum_i \left(\nabla I(x_i) \nabla I(x_i)^T\right)(x_i - P) = 0$$

$$\underbrace{\sum \nabla I(x_i) \nabla I(x_i)^T P}_{C} = \underbrace{\sum \nabla I(x_i) \nabla I(x_i)^T x_i}_{Q}$$

$$CP = Q$$

$$P = C^{-1} Q$$

$$P^* = \left(\sum \nabla I(x_i) \nabla I(x_i)^T\right)^{-1} \sum \nabla I(x_i) \nabla I(x_i)^T x_i$$

$$= C^{-1} \sum \nabla I(x_i) \nabla I(x_i)^T x_i$$

Condition for the solution to exist: C must be non-singular.

h) We take a window (10x10), for instance, that we want to characterize. Break them into overlapping subwindow. In each subwindow, we calculate the orientation histogram (using edge or gradient directions, possibly weighted by distance from center or gradient magnitude. Then concatenate the orientation histograms.

If we use 8 directions in each subwindow and we have 9 subwindows, we have a vector of 72 elements.

Requirements for a good characterization of feature points:
i) Translation invariance
ii) Rotation invariance
iii) Scale invariance
iv) Illumination invariance

i) SIFT breaks the window into subparts (non-overlapping). In each subpart, we compute gradient direction. It builds orientation histogram in each subpart. Then align each one based on principal direction. After this, we will concatenate.