FILTRAGGIO SPAZIALE

Il **filtraggio** è una tecnica per il dominio delle frequenze che consente di far **passare** o **bloccare** alcune frequenze.
Le tecniche di **filtraggio spaziale** operano sui pixel di un'immagine prendendo in considerazione i valori di intensità in un **intorno**.
La **regola di trasformazione** è descritta da una **matrice** definita **filtro** della stessa dimensione dell'intorno.
Per poter applicare il filtro a tutti i pixel, si inseriscono opportuni valori su tutti i lati dell'immagine (**padding**).

CORRELAZIONE E CONVOLUZIONE

Con **correlazione** si intende il progressivo scorrimento di una maschera sull'immagine e il calcolo della somma dei prodotti in ogni posizione.

$$g(x,y) = w(x,y) * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

w indica la **matrice peso** (filtro) e **a** e **b** che sono le dimensioni del filtro.
Con **convoluzione** si intente la correlazione con filtro ruotato di 180 gradi.
Per realizzarla, bisogna pre ruotare il filtro di 180 gradi.

$$g(x,y) = w(x,y) * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t)$$

In OpenCV è possibile effettuare la correlazione con **filter2D()**.
Per la convoluzione, è necessario usare la **rotate()** per ruotare il filtro di 180 gradi.

FILTRI DI SMOOTH

Sono utilizzati per sfocare le immagini e ridurre il rumore.
Operano rimuovendo i dettagli più piccoli, permettendo di evidenziare gli oggetti più grandi e di riempire i piccoli gap.
Nei filtri **di media**, definiti anche **passa basso** in quanto attenuano le alte frequenze a discapito delle basse, il valore di ogni pixel è sostituito con la media dei livelli di intensità nell'intorno definito dalla maschera.
Se si usa un **operatore di media**, tutti i coefficienti della maschera sono uguali a 1.
Se si usa un **operatore di media pesata**, i pesi dei pixel sono inversamente proporzionali alla distanza dal centro (il centro ha il peso maggiore).
Il compito dei **filtri non lineari** è ordinare i pixel contenuti nell'intorno definito dal filtro e sostituire il valore del pixel centrale con un valore dell'insieme come il mediano (**medianBlur()**), il massimo o il minimo.
In OpenCV esistono diverse funzioni di smoothing come **blur()**, **boxFilter()** e **GaussianBlur()**.
Per effettuare la sogliatura si usa **double threshold()**.
In pratica, se si vuole estrarre gli oggetti più importanti di un'immagine, si esegue un'operazione di smoothing, si applica una soglia, si sovrappone la maschera all'immagine di input e si esegue il prodotto puntuale.
Maggiore è la dimensione del filtro, maggiore è la dimensione degli oggetti estratti.

SHARPENING

Col termine **sharpening** si intende una serie di tecniche utilizzate per evidenziare le **transizioni di intensità**.
Gli oggetti percepiti per il cambio di intensità nelle immagini sono i **bordi**: più nette sono le transizioni, più l'immagine è definita.
La transizione di intensità tra pixel adiacenti è strettamente collegata alla **derivata dell'immagine** in quella posizione, in quanto permette di valutare la variazione di intensità tra due pixel.
In aree di intensità costante la variazione sarà nulla, per diventare più netta se si ha un bordo.

DERIVATA PRIMA DI UN'IMMAGINE

Definiamo l'**operatore di derivazione naturale** che equivale alla differenza tra le intensità dei pixel vicini e che goda delle principali proprietà della derivata prima:

- Uguale a 0 quando l'intensità è costante;
- Diversa da 0 per le transizioni di intensità;
- Costante sulle rampe in cui la transizione di intensità è costante;
- $\frac{\mathbf{Df}}{\mathbf{Dx}} = \mathbf{f(x+1) - f(x)}$: differenza di intensità tra due pixel vicini.

DERIVATA SECONDA DI UN'IMMAGINE

Analogamente, definiamo l'**operatore di derivata seconda**, definita usando il pixel precedente e successivo:

- Uguale a 0 dove l'intensità è costante;
- Diversa da 0 all'inizio di una rampa di intensità;
- Uguale a 0 sulle pendenze costanti delle rampe;
- $\frac{\mathbf{D^2f}}{\mathbf{D^2x}} = \mathbf{f(x+1) - 2f(x) + f(x-1).}$

Il segno della derivata seconda può essere usato per determinare se un edge è una **transizione chiaro/scuro** (negativa) o **scuro/chiaro** (positiva).
I punti singoli sono associati al **rumore**.

FILTRO LAPLACIANO

Il **filtro Laplaciano** implementa una **derivata del secondo ordine**.
Si tratta di un **filtro isotropico**, ossia **indipendente dalla rotazione** dell'immagine.
Viene utilizzato per trovare le variazioni di intensità in un'immagine (edge).
Sommando il suo contributo all'immagine si mettono in risalto i dettagli.
Esso è definito come:

- Invariante alle rotazioni di 90 gradi:
$$\nabla^2\mathbf{f} = \frac{\mathbf{D^2f}}{\mathbf{D^2x}} + \frac{\mathbf{D^2f}}{\mathbf{D^2y}} = \mathbf{f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)}$$
- Invariante alle rotazioni di 45 gradi:
$$\nabla^2\mathbf{f} = \frac{\mathbf{D^2f}}{\mathbf{D^2x}} + \frac{\mathbf{D^2f}}{\mathbf{D^2y}} = \mathbf{f(x-1,y-1) + f(x+1,y-1) + f(x-1,y+1) + f(x+1,y+1) - 4f(x,y)}$$

Siccome assume spesso valore negativo, per visualizzarlo occorre scalarlo nell'intervallo di rappresentazione [0, L-1].
Utilizzando il Laplaciano per verificare la presenza di edge, vengono restituite due risposte, che corrispondono all'inizio e alla fine di una rampa di intensità (un valore negativo e uno positivo).
Se utilizzo una **sogliatura**, perdo i valori negativi.
Se prendo il **valore assoluto**, non perdo i valori negativi ma ottengo un edge più spesso.

LAPLACIAN OF GAUSSIAN

Consiste nello sfruttare il Laplaciano per verificare le variazioni di intensità (edge), mentre il Gaussiano per applicare lo smoothing (ridurre il rumore).

$$\nabla^2 G(x,y) = \frac{\partial^2 G(x,y)}{\partial^2 x^2} + \frac{\partial^2 G(x,y)}{\partial^2 y^2}$$

In OpenCV ciò è realizzato usando il risultato di **getGaussianKernel()** come argomento di **filter2D()**.
Lo **zero-crossing** è il punto intermedio tra un valore negativo e quello positivo in cui ci si aspetta che passi l'edge (approssimazione).
Per trovarlo, si considera un intorno 3x3 centrato in un pixel p dell'immagine filtrata.
Uno zero-crossing in p implica che i **segni** di almeno due pixel vicini opposti siano **diversi**.
Inoltre, il **valore assoluto** della loro differenza deve superare una **soglia**.

LAPLACIAN OF GAUSSIAN

Consiste nello sfruttare il Laplaciano per verificare le variazioni di intensità (edge), mentre il Gaussiano per applicare lo smoothing (ridurre il rumore).

$$\nabla^2 G(x,y) = \frac{\partial^2 G(x,y)}{\partial^2 x^2} + \frac{\partial^2 G(x,y)}{\partial^2 y^2}$$

In OpenCV ciò è realizzato usando il risultato di **getGaussianKernel()** come argomento di **filter2D()**.
Lo **zero-crossing** è il punto intermedio tra un valore negativo e quello positivo in cui ci si aspetta che passi l'edge (approssimazione).
Per trovarlo, si considera un intorno 3x3 centrato in un pixel p dell'immagine filtrata.
Uno zero-crossing in p implica che i **segni** di almeno due pixel vicini opposti siano **diversi**.
Inoltre, il **valore assoluto** della loro differenza deve superare una **soglia**.

GRADIENTE

Il **gradiente** è un **vettore** formato dalle **derivate parziali** che punta nella **direzione** di **massima variazione.**
La **magnitudo** del gradiente è l'informazione sull'intensità della variazione.

$$M(x,y) = mag(\nabla f) = \sqrt{(g_x^2 + g_y^2)}$$

La massima variazione corrisponde ad un edge.

UNSHARP MASKING

Metodo per rendere le immagini più nitide nel caso non ci sia rumore:

1. Sfocare l'immagine originale;
2. Ottenere una maschera come differenza tra l'immagine originale e quella sfocata;
3. Aggiungere la maschera all'immagine originale.

g = f + k * (f - f * h).

OPERATORI DI SOBEL

Il **filtraggio di Sobel** riduce la visibilità delle regioni in cui l'intensità varia lentamente, permettendo di **evidenziare i dettagli**.
Sono utili anche per trovare la direzione dell'edge.

$$\mathbf{g_x}(\mathbf{x},\mathbf{y}) = -\mathbf{f}(\mathbf{x}-\mathbf{1},\mathbf{y}-\mathbf{1}) - \mathbf{2f}(\mathbf{x}-\mathbf{1},\mathbf{y}) - \mathbf{f}(\mathbf{x}-\mathbf{1},\mathbf{y}+\mathbf{1}) + \mathbf{f}(\mathbf{x}+\mathbf{1},\mathbf{y}-\mathbf{1}) + \mathbf{2f}(\mathbf{x}+\mathbf{1},\mathbf{y}) + \mathbf{f}(\mathbf{x}+\mathbf{1},\mathbf{y}+\mathbf{1})$$

$$\mathbf{g_x}(\mathbf{x},\mathbf{y}) = -\mathbf{f}(\mathbf{x}-\mathbf{1},\mathbf{y}-\mathbf{1}) - \mathbf{2f}(\mathbf{x},\mathbf{y}-\mathbf{1}) - \mathbf{f}(\mathbf{x}+\mathbf{1},\mathbf{y}-\mathbf{1}) + \mathbf{f}(\mathbf{x}-\mathbf{1},\mathbf{y}+\mathbf{1}) + \mathbf{2f}(\mathbf{x},\mathbf{y}+\mathbf{1}) + \mathbf{f}(\mathbf{x}+\mathbf{1},\mathbf{y}+\mathbf{1})$$

In OpenCV, **Sobel()**.

COLORE

Il **colore** è un **descrittore** per estrarre oggetti da una scena.
L'elaborazione delle immagini a colori si divide in **full-color** (colori acquisiti da un sensore) e **falsi colori** (colori assegnati a particolari valori di intensità).
La **luce acromatica** ha come unico attributo l'**intensità** (grigio), mentre quella **cromatica** è descritta da:

- **Radianza:** quantità di energia uscente;
- **Luminanza:** quantità di energia percepita;
- **Luminosità:** descrittore di intensità.

L'occhio umano vede i **colori** come **combinazioni** di **colori primari**.
Per distinguere tra loro i colori, sono necessarie la **luminosità**, la **tonalità** e la **saturazione**, con le ultime due che compongono la **cromaticità**.
Un colore è specificato mediante tali **coefficienti tricromatici: x + y + z = 1**, con **x**, **y** e **z** che consistono nelle quantità di **rosso**, **verde** e **blu** necessarie per formare un colore.
Un altro modo per specificare il colore è il **diagramma di cromaticità**, che mostra la composizione del colore in funzione di x (rosso) e y (verde). Dalla relazione **x + y + z = 1**, si ricava il blu.
Il punto in cui i tre componenti assumono valore uguale è detto **punto di uguale energia** (**bianco**).

MODELLI COLORE

Un **modello colore** è un **sistema di coordinate** in cui ogni colore è rappresentato da un punto.
I modelli più utilizzati sono:

- **RGB:** ogni colore è rappresentato dalle componenti primarie (rosso, verde e blu). Si basa su un sistema di coordinate cartesiane in cui lo spazio di interesse è un cubo. Le immagini rappresentate in tale modello sono formate da tre immagini, una per ogni colore primario. In genere, ognuna delle immagini è a **8 bit**, per cui la profondità dei pixel RGB è **24 bit**;
- **CMY:** si basa sui colori ciano, magenta e giallo. Uguali quantità dei pigmenti non producono un nero puro, per questo le stampanti usano un colore aggiuntivo per il nero;
- **HSI:** si basa sui concetti di **tonalità** (colore puro), **saturazione** (quantità di bianco) e **luminosità** (intensità). Per determinare l'**intensità** di un punto colore RGB, si fa passare un **piano perpendicolare** all'asse di intensità che contenga il punto colore. Il punto di intersezione tra asse di intensità e piano è il valore di intensità. La **saturazione** è data dalla distanza del punto colore dall'asse di intensità. La **tonalità** di un punto è determinata dall'angolo rispetto ad un punto. Un vantaggio di questo modello è decorrelare le informazioni dei colori dall'intensità.

ELABORAZIONI FULL-COLOR

Il primo metodo consiste nell'elaborare separatamente ogni componente e combinarla con le altre.
Il secondo metodo prevede di elaborare tutti i pixel colore insieme.
Lo **smoothing** di un'immagine in scala di grigio è realizzabile con un'operazione di filtraggio spaziale ed una maschera.
Il valore di ogni pixel è sostituito con la **media** dei valori dei pixel nell'intorno definito dalla maschera.
Lo **sharpening** si realizza mediante **Laplaciano**.
In OpenCV, la funzione **imread()** determina il tipo di file ed alloca la memoria necessaria per la struttura dati che conterrà i dati dell'immagine.
Le immagini sono caricate in matrici a tre canali, ognuna di profondità 8 bit.
Se l'immagine di input è **a colori**, viene allocata una matrice con tre canali RGB.
Se l'immagine di input è in **scala di grigio**, viene allocata una matrice con tre canali in scala di grigio.
La funzione **cvtColor()** modifica lo spazio colore con cui è rappresentata un'immagine, conservando il tipo.

SEGMENTAZIONE

Data una regione **R** occupata dall'immagine, la segmentazione consiste nel partizionare R in **sotto regioni**:

- La loro unione consista nell'intera regione $U_{i=1}^{n} R_i = R$;
- R_i sia un insieme connesso;
- Le regioni siano tra loro disgiunte;
- Ogni regione rispetti un predicato (regola);
- L'unione tra due regioni restituisca come risultato falso (predicati diversi), altrimenti significherebbe che le due regioni non sono disgiunte.

Un **percorso** da p a q è una **sequenza di pixel adiacenti** da p e q che si dicono **connessi**.
I pixel connessi di una regione S formano una **componente connessa**.
Se in S esiste una sola componente connessa S è una **regione**.
L'obiettivo della **segmentazione** è suddividere un'immagine nelle **regioni/oggetti** che la compongono.
Le tecniche si basano su:

- **Discontinuità:** trovare i gruppi di pixel che si trovano in un bordo. Si assume che i bordi siano sufficientemente diversi tra le regioni e dallo sfondo. Si hanno problemi in caso di presenza di rumore;
- **Similarità:** raggruppare i pixel che presentano caratteristiche in comune (colore, intensità). L'idea è che se in una zona i pixel non variano, allora appartengono alla stessa regione. Il bordo sarà frastagliato perché non si ha una risposta puntuale.

Problema: per ogni immagine esistono varie segmentazioni.

SEGMENTAZIONE EDGE BASED

Si sfruttano tre caratteristiche di un'immagine:

- **Bordo:** insiemi di pixel di edge in cui si presenta una repentina variazione di intensità;
- **Linee:** segmenti di edge in cui l'intensità ai lati è minore dell'intensità dei pixel della linea;
- **Punti:** linee di lunghezza e larghezza pari a 1 pixel.

Per individuare le variazioni di intensità si utilizzano **derivata prima** e **seconda**.
Per i punti isolati, dopo aver applicato il filtro Laplaciano, si utilizza una **soglia** sulla risposta del filtro per determinare i punti di discontinuità.

$$g(x,y) = \begin{cases} 1 \; se \; |R(x,y)| \geq T \\ 0 \; altrimenti \end{cases}$$

dove **g** è l'immagine di output, **T** una soglia non negativa ed R la risposta.
L'intensità di un punto isolato sarà diversa da quella dei suoi vicini.
Un problema nell'utilizzo del Laplaciano è la perdita delle informazioni riguardo le variazioni delle intensità.
Per risolvere, si prende il **valore assoluto** della risposta (edge doppi) o solo i **valori positivi** (edge sottili).
Quando le linee dell'immagine sono più ampie rispetto alla dimensione del filtro, si ottiene una **valle**.
Nel caso ci fosse rumore, come un'immagine ben dettagliata, bisogna prima applicare uno **smoothing** per ridurlo, per poi individuare i punti di edge e selezionare solo quelli che fanno parte di un edge.

CANNY EDGE DETECTOR

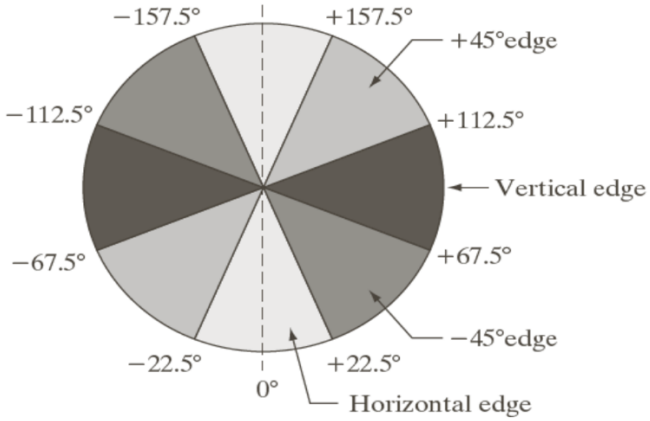
L'algoritmo di Canny è utilizzato per l'individuazione degli edge in tre obiettivi:

- **Basso tasso di errore:** minimizzare la probabilità di falsi positivi (quando l'algoritmo rileva come pixel di edge uno che non lo è) e falsi negativi;
- **Punti di edge ben localizzati:** gli edge rilevati devono essere il più vicino possibile a quelli reali;
- **Risposta uguale per edge singolo:** restituire un solo punto per ogni punto di edge.

Canny dimostrò che una buona approssimazione per l'individuazione di un edge a gradino è la **derivata prima della Gaussiana**.
Poiché la direzione dell'edge non è conosciuta a priori, bisognerebbe applicare l'operatore in tutte le direzioni possibili.
Questo processo è approssimabile effettuando la **convoluzione** dell'immagine con una **Gaussiana 2D** e calcolando il **gradiente** del risultato.
Magnitudo e **gradiente** saranno utilizzati per calcolare **intensità** e **orientazione** dell'edge.

NON MAXIMA SUPPRESSION

Siccome l'**immagine magnitudo** presenta **molti picchi** intorno ai massimi locali (variazioni forti), è necessario sopprimere tali valori, considerando un numero finito di orientazioni in un intorno 3x3.
Poiché bisogna **quantizzare** tutte le possibili direzioni in quattro settori, si definiscono degli **intervalli** e la direzione dell'edge è calcolata dalla direzione del gradiente, che è ortogonale rispetto all'edge.
Ad esempio, se il gradiente punta ad un angolo di fase compreso tra -22.5 e 22.5, si ha un edge orizzontale.



Date le 4 direzioni orizzontale, verticale, +45 e -45, lo schema per un intorno 3x3 prevede i seguenti passi:

- Trovare la direzione più vicina a a(x,y);
- Se M(x,y) è **minore** di almeno uno dei due vicini lungo la direzione del gradiente, si esegue la **soppressione** $\boldsymbol{g_n(x,y) = 0}$. Altrimenti, $\boldsymbol{g_n(x,y) = M(x,y)}$. In pratica sopprimo il pixel se nel suo intorno si ha un pixel con variazione maggiore;
- **Sogliare** l'immagine per evitare di sopprimere picchi che sono poco minori rispetto ai massimi. Con una soglia troppo bassa si ottengono falsi positivi e viceversa.

THRESHOLDING CON ISTERESI

L'**isteresi** usa due soglie: una bassa $\boldsymbol{T_L}$ ed una alta $\boldsymbol{T_H}$.
In pratica, scarto tutti i picchi con valore minore della soglia bassa ed elevo a forti tutti quelli con valore maggiore della soglia alta.
Per quanto riguarda i pixel che si trovano tra le due soglie, si analizza il pixel adiacente per stabilire se vanno elevati a forti.
Per visualizzare l'operazione di **thresholding** si creano due immagini ausiliarie:
$\boldsymbol{g_{NH}(x,y) = g_n(x,y) \geq T_H}$ solo i pixel che superano la soglia alta
$\boldsymbol{g_{NL}(x,y) = g_n(x,y) \geq T_L}$ solo i pixel che superano la soglia bassa
Siccome tutti i pixel non nulli di $g_{NH}(x,y)$ saranno in $g_{NL}(x,y)$, eseguo la sottrazione tra insiemi per trovare i pixel che stanno tra le due soglie $\boldsymbol{g_{NL}(x,y) = g_{NL}(x,y) - g_{NH}(x,y)}$
I pixel di edge in $\boldsymbol{g_{NH}(x,y)}$ sono detti **forti**, mentre quelli in $\boldsymbol{g_{NL}(x,y)}$ **deboli**.
Per riempire i **vuoti** negli edge:

- Si localizza il prossimo pixel di edge p in $g_{NH}(x,y)$;
- Si etichettano come edge forti tutti i pixel di edge di $g_{NL}(x,y)$ in $N_8(p)$;
- Si ripete il primo passo finché non sono stati visitati tutti i pixel in $g_{NH}(x,y)$.

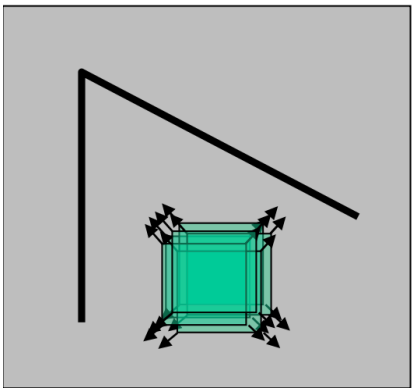
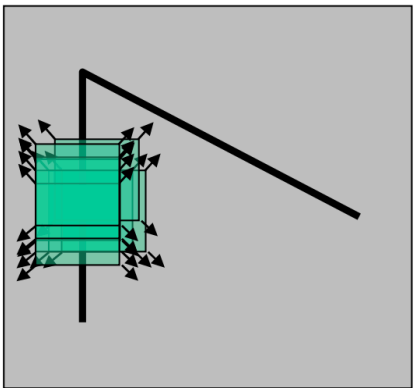
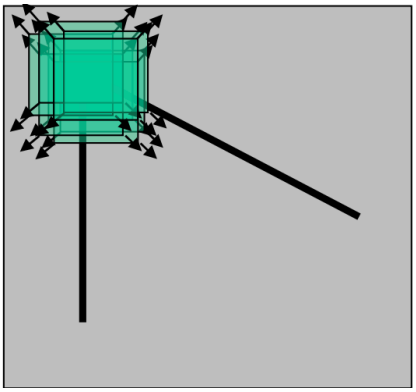
Per l'implementazione, non è necessaria la creazione delle due immagini ausiliarie, in quanto il thresholding con isteresi è eseguibile direttamente sull'immagine $g_{NH}(x,y)$.

PASSI ALGORITMO CANNY

- Eseguire una convoluzione con un filtro Gaussiano per lo smoothing;
- Calcolare magnitudo e orientamento del gradiente;
- Applicare la non maxima suppression;
- Applicare il thresholding con isteresi.

HARRIS CORNER DETECTOR

L'algoritmo di Harris è utilizzato per rilevare i **corner** di un'immagine.
Un **corner** rappresenta l'intersezione di linee all'interno di un'immagine.
Una caratteristica importante dei corner è che sono **stabili** in sequenze di immagini, motivo per il quale possono essere utilizzati per tracciare gli oggetti nei video.
L'idea per rilevare un corner è che in sua corrispondenza si avranno variazioni significative in più direzioni, a differenza di un edge in cui si ha una variazione che si sviluppa in un'unica direzione.

CARATTERIZZAZIONE DEGLI EDGE

Prendiamo una **finestra** e consideriamo uno **spostamento [u, v]**:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2$$

La funzione **w(x,y)** assegna un peso ad ogni pixel dell'immagine e può essere di due tipi:

- assume il valore 1 se si trova nella finestra e 0 se all'esterno;
- assume un valore 0 se all'esterno della finestra o un valore maggiore in base alla vicinanza ai pixel centrali (gaussiana).

Il secondo contributo alla funzione è definito **Sum of Squared Distance** e calcola la distanza, in termini di valori di intensità, tra i pixel di posizione (x, y) e il suo valore shiftato della quantità (u, v).
Esso permette di valutare la variazione di intensità sull'asse x e sull'asse y (in pratica è il gradiente).
L'elevazione al quadrato permette di esaltare ulteriormente le variazioni.
Per calcolare un piccolo **spostamento**, la funzione può essere **approssimata al gradiente**.
La matrice è ottenuta dal prodotto tra il gradiente e la sua trasposta (per calcolare il quadrato).

$$E(u,v) = \sum_{x,y} w(x,y)[\nabla I(u,v) \nabla I(u,v)^T]$$

Il risultato è una matrice che contiene lungo la diagonale principale i quadrati delle variazioni di intensità lungo l'asse x e l'asse y.

$$E(u,v) = \begin{bmatrix} \sum_{x,y} w(x,y) I_u^2 & \sum_{x,y} w(x,y) I_u * I_v \\ \sum_{x,y} w(x,y) I_u * I_v & \sum_{x,y} w(x,y) I_v^2 \end{bmatrix}$$

Notiamo che la matrice è **simmetrica** e può essere scomposta, mediante **SVD** (decomposizione a valori singolari) nel prodotto di tre sottomatrici:

- **U:** contiene gli auto vettori di E sulle colonne;
- U^T**:** contiene gli auto vettori di E sulle righe;
- **S**: contiene sulla diagonale gli autovalori di E.

Ogni autovalore indica in che direzione e con che valore si ha la differenza di intensità.

$$\boldsymbol{E = USU^T = U\begin{bmatrix}\gamma_1 & 0\\ 0 & \gamma_2\end{bmatrix}U^T}$$

- $\boldsymbol{\gamma_1 \gg 0}$ e $\boldsymbol{\gamma_2 \approx 0}$**:** nell'intorno del punto considerato si ha una sola grande variazione di intensità nella direzione dell'auto vettore associato a $\boldsymbol{\gamma_1}$ e quindi si ha un edge;
- $\boldsymbol{\gamma_1 \gg 0}$ e $\boldsymbol{\gamma_2 \gg 0}$**:** nell'intorno del punto considerato si ha una grande variazione di intensità in entrambe le direzioni degli auto vettori associati a $\boldsymbol{\gamma_1}$ e $\boldsymbol{\gamma_2}$ e quindi si ha un corner;
- $\boldsymbol{\gamma_1 \approx 0}$ e $\boldsymbol{\gamma_2 \approx 0}$**:** si ha una zona di **intensità uniforme**.

OTTIMIZZAZIONE

Il calcolo della SVD andrebbe ripetuto per ogni punto dell'immagine ma si tratta di un'operazione costosa. Per ridurre il costo computazionale, si utilizzano degli indici più veloci da calcolare, utilizzando i concetti di **traccia** e **determinante**.

$$\boldsymbol{R(u,v) = det(C(u,v)) - ktrace^2(C(u,v))}$$

Se $\boldsymbol{R \gg 0}$ allora si ha un corner.
La matrice è facilmente calcolabile usando Sobel e applicando un filtro Gaussiano.
Il valore k è generalmente impostato a 0,04.

$$E(u,v) = \begin{bmatrix}\sum_{x,y} w(x,y)\, I_u^2 & \sum_{x,y} w(x,y)\, I_u * I_v\\ \sum_{x,y} w(x,y)\, I_u * I_v & \sum_{x,y} w(x,y)\, I_v^2\end{bmatrix}$$

$$E(u,v) = \begin{bmatrix}C_{00} & C_{01}\\ C_{10} & C_{11}\end{bmatrix}$$

- $\text{determinante} = C_{00} * C_{11} - C_{01} * C_{10}$
- $\text{traccia} = C_{00} + C_{11}$
- $R = \text{determinante} - k * \text{traccia}^2$

PASSI DELL'ALGORITMO DI HARRIS

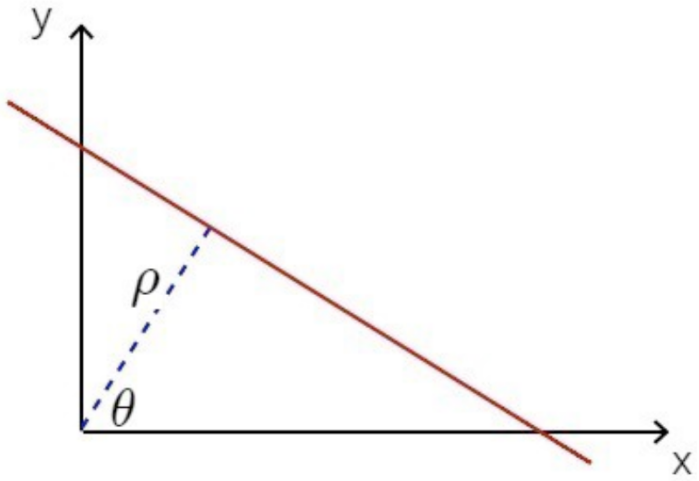
- Calcolare le derivate parziali rispetto a x e y, ovvero I_u e I_v;
- Calcolare le derivate seconde rispetto a x e y, ovvero I_u^2 e I_v^2 e Dx * Dy, ovvero $I_u * I_v$;
- Applicare un filtro Gaussiano alle derivate seconde e a Dx * Dy, ovvero moltiplicando per $w(x,y)$;
- Calcolare l'indice R utilizzando le componenti della matrice ottenute e i concetti di determinante e traccia;
- Normalizzare l'indice R in [0, 255] per evitare problemi in caso di valori elevati;
- Sogliare l'indice R.

TRASFORMATA DI HOUGH

Tecnica che permette di riconoscere particolari configurazioni di punti in un'immagine, come rette e cerchi.
È utilizzabile se la forma cercata è esprimibile tramite una **funzione parametrica** e l'insieme dei parametri si dice che definisce una **istanza** della forma dell'oggetto.
Viene definito un **operatore globale** perché non opera sull'intorno di un pixel ma analizza le sue proprietà rispetto agli altri.

HOUGH PER RETTE

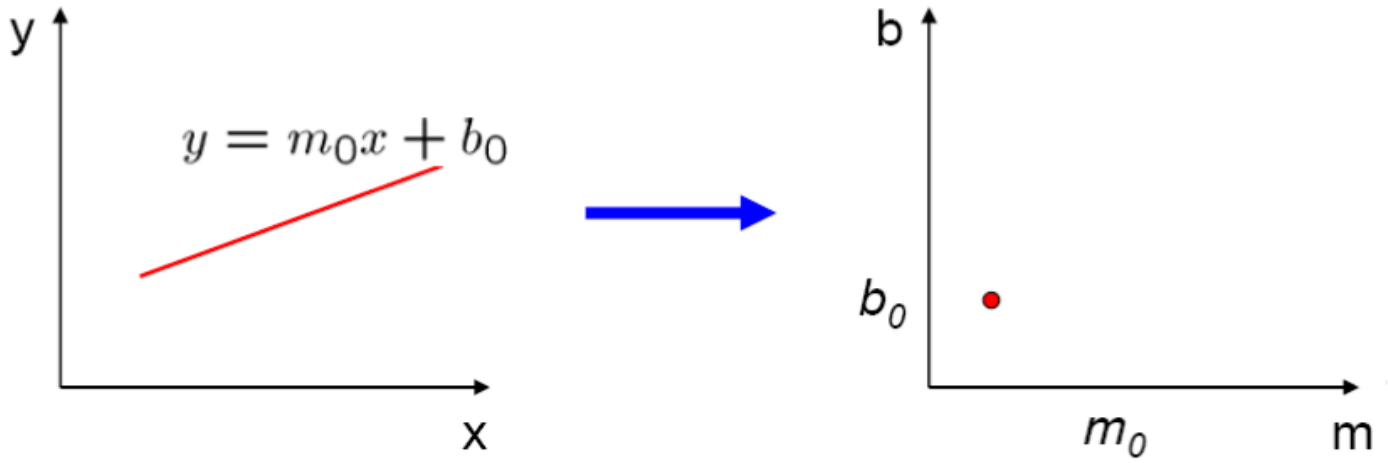
La rappresentazione canonica **y = mx + b** (**coefficiente angolare** ed **intercetta**) presenta problemi.
Perciò si usa la rappresentazione in forma normale **p = xcosθ + ysinθ**.



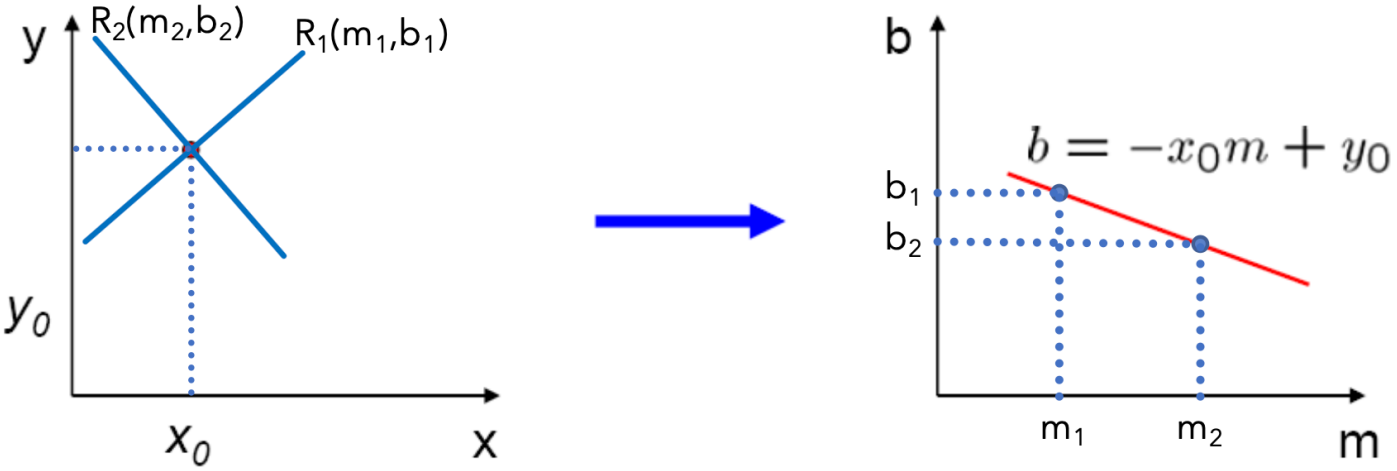
L'obiettivo è trovare, dati pixel di edge dell'immagine, i parametri delle rette su cui giacciono.

SPAZIO DEI PARAMETRI

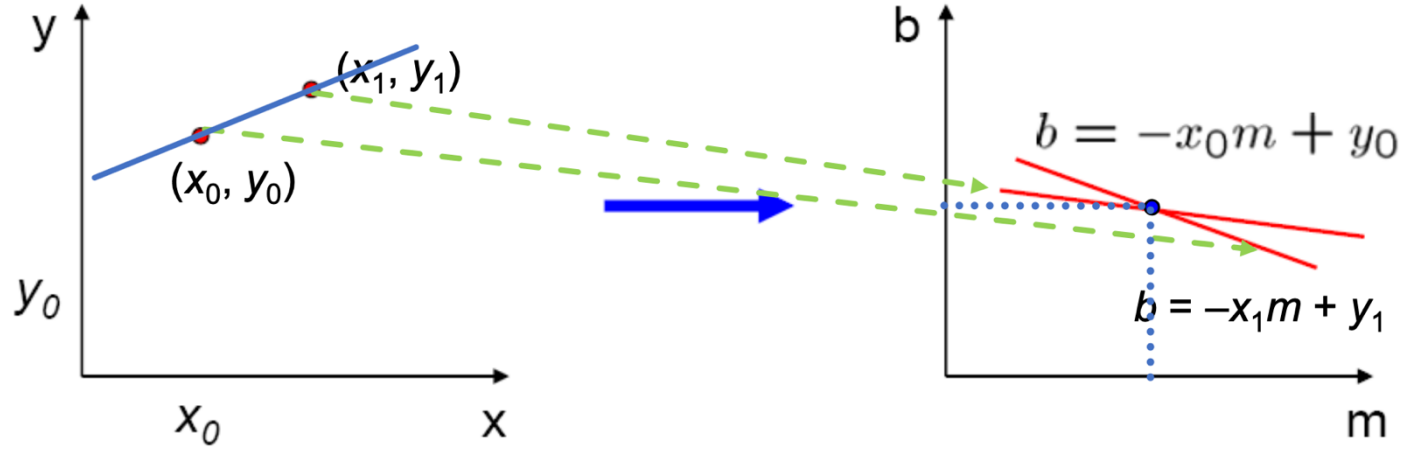
Fissata una retta e la sua rappresentazione, definiamo la sua trasformazione nello **spazio dei parametri**.
Nello **spazio dei parametri**, gli assi corrispondono ai parametri b ed m ed una istanza di retta è rappresentata da un punto.



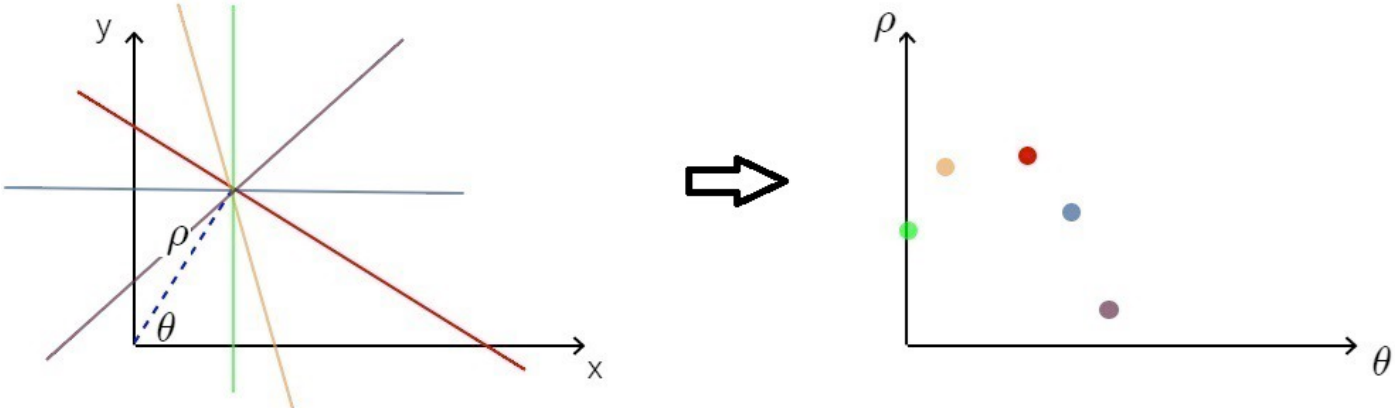
Per un punto passano infinite rette, ognuna identificata da una coppia di parametri.
Ciò che hanno in comune, sono le coordinate del punto (x_0, y_0).
Esse sono coefficiente angolare e intercetta della retta nello spazio dei parametri.
Perciò, un punto nello **spazio immagine** corrisponde ad una retta nello spazio dei parametri.

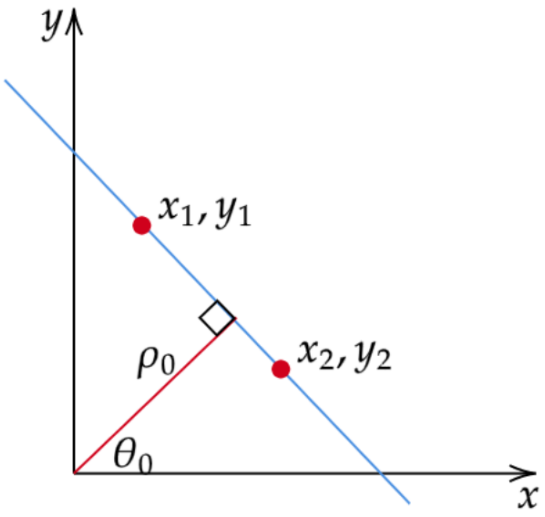


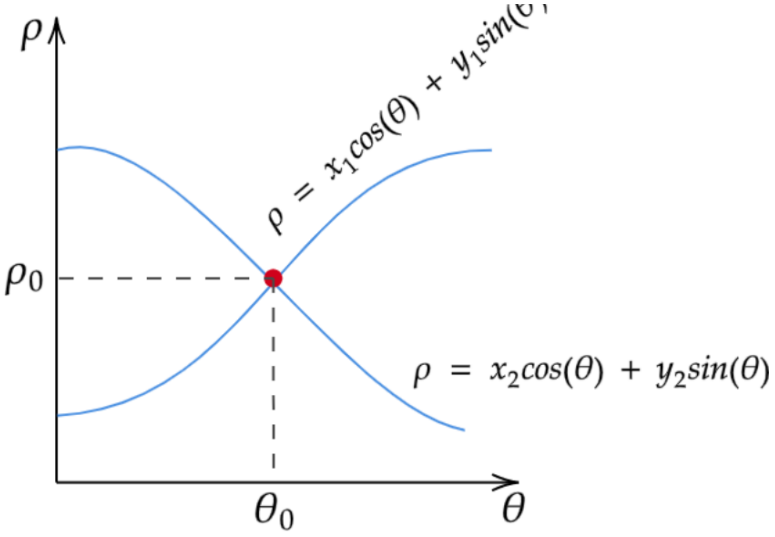
Se si considerano due punti nello spazio immagine, la retta che li contiene è identificata dai parametri **m** e **q**, che identificano il punto di intersezione delle rette corrispondenti ai due punti nello spazio dei parametri.



Si ha un problema: quando la retta è verticale, **m = ∞.**
Si utilizza la **rappresentazione polare**, con la retta che viene rappresentata da un punto nello spazio dei parametri di coordinate (p, θ).
In questo caso, un punto nello spazio immagine corrisponde ad una sinusoide nello spazio dei parametri.
Il punto di intersezione di tutte le sinusoidi nello spazio dei parametri individua la retta su cui giacciono i punti nello spazio immagine.

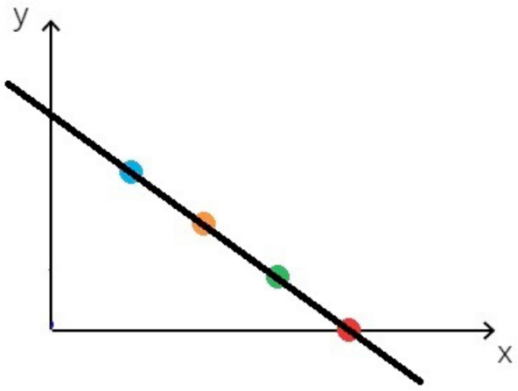


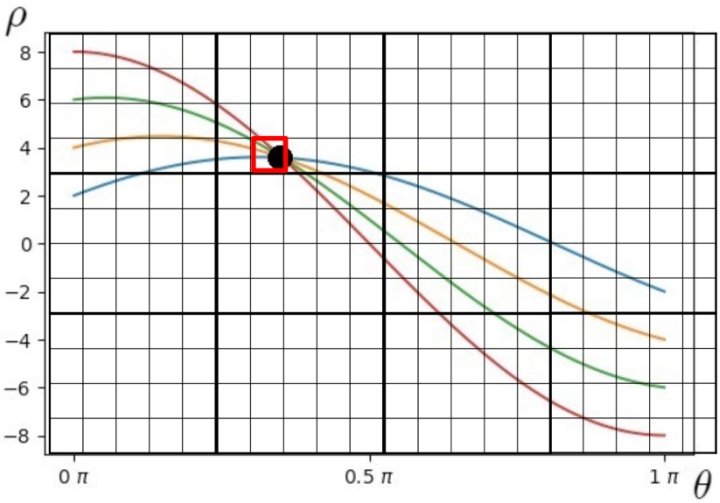




SCHEMA DI VOTO

Quantizziamo lo spazio dei parametri, con lo spazio che viene definito **spazio dei voti**.
Si definisce una **matrice di accumulatori** inizializzati a 0.
Per ogni punto si aggiunge un **voto** nelle **celle** corrispondenti ai parametri delle rette che passano per quel punto (incremento).
Una volta che tutti i punti hanno votato, le celle con un numero di voti superiore ad una certa soglia corrispondono alle rette nell'immagine.
Ogni cella può assumere al massimo un valore corrispondente al numero di pixel sulla diagonale dell'immagine, che corrisponde alla distanza massima.



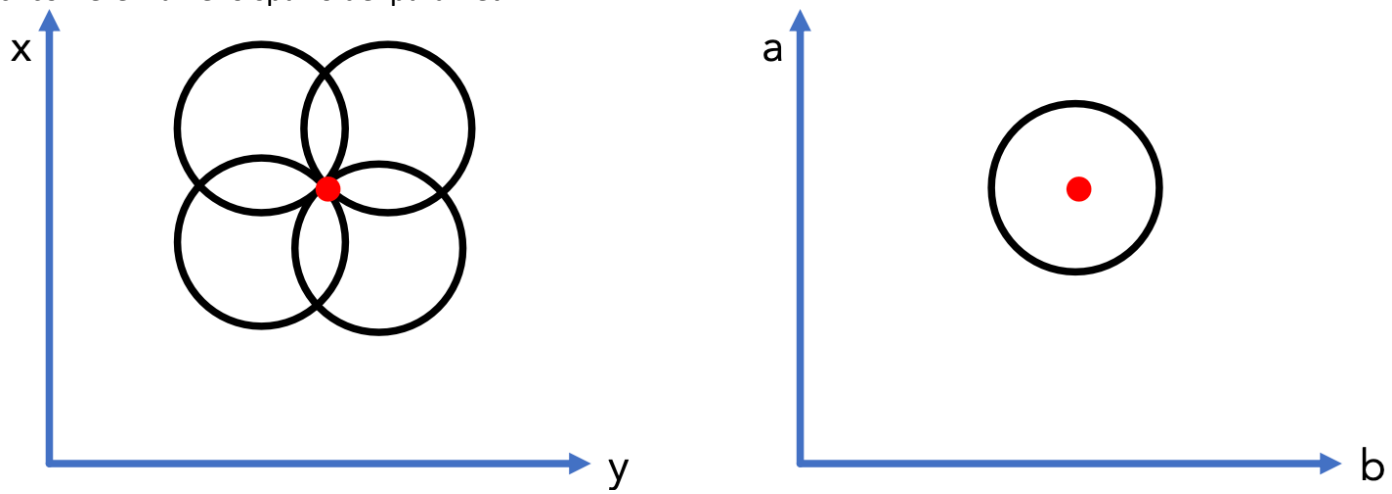


PASSI ALGORITMO HOUGH PER RETTE

- Creare l'accumulatore H (bidimensionale);
- Applicare l'algoritmo di Canny per trovare i punti di edge;
- Per ogni punto di edge, calcolare **p = xcosθ + ysinθ** e incrementare H(p, θ) di 1 (voto);
- Tutte le celle con valore superiore alla soglia corrispondono alle rette.

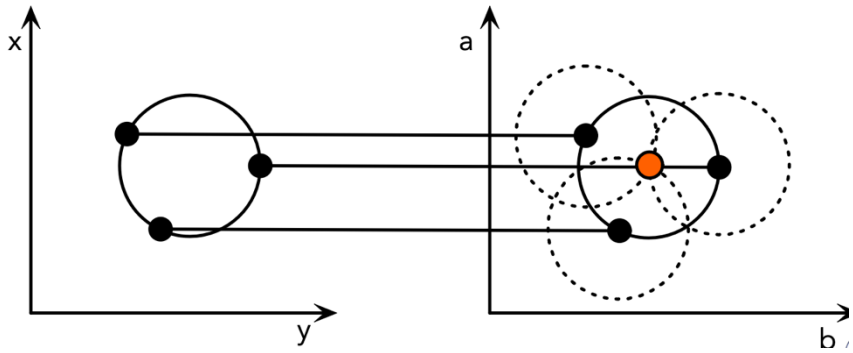
HOUGH PER CERCHI

Una circonferenza con centro (a, b) e raggio R è rappresentata dall'equazione parametrica **x = a + Rcosθ** e **y = b + Rsinθ**.
In un'immagine possono essere presenti cerchi con differenti centri e raggi.
Ipotizzando di conoscere la lunghezza del raggio delle circonferenze da trovare, restano variabili solo **a** e **b**.
Per questo motivo, lo spazio dei parametri è un piano bidimensionale.
Per un punto nello spazio immagine passano infinite circonferenze di raggio R, ognuna identificata dai parametri a e b e tutti accomunati dallo stesso raggio.
Il punto nello spazio immagine per il quale passano tutte le circonferenze, corrisponde ad una circonferenza nello spazio dei parametri.

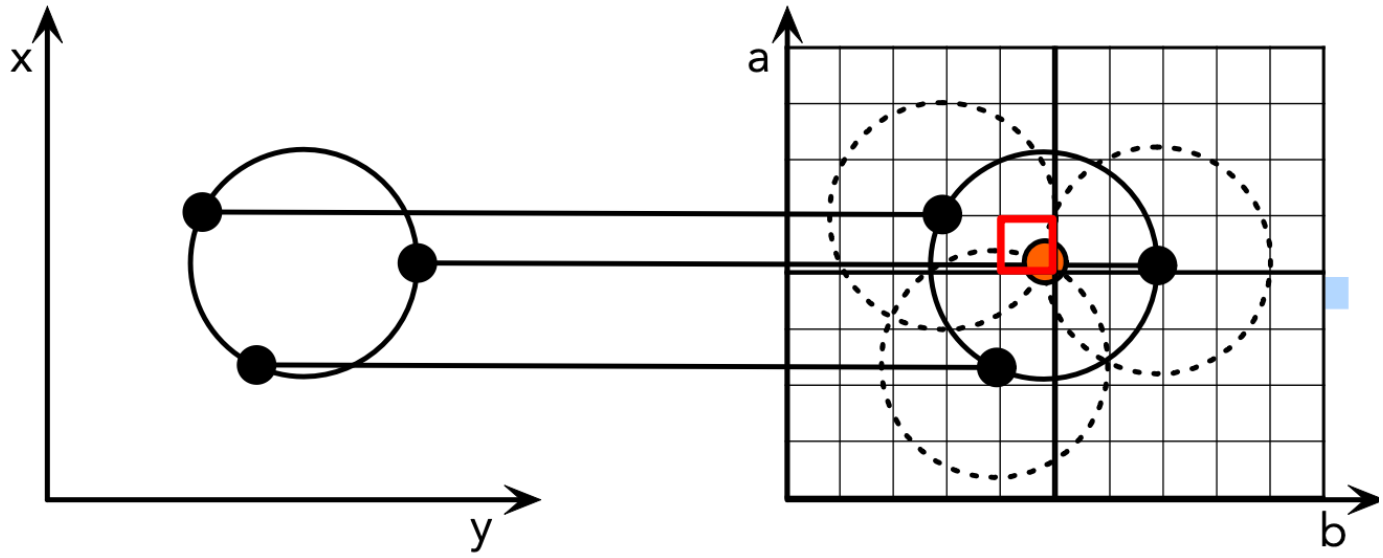


Per tre punti non allineati, invece, passa una sola circonferenza le cui coordinate del centro corrispondono al punto di intersezione nello spazio dei parametri.
I tre punti nello spazio immagine corrispondono ai centri di tre circonferenze nello spazio dei parametri.
La circonferenza nello spazio dei parametri passante per i centri delle circonferenze ha il centro equivalente al centro della circonferenza nello spazio immagine.

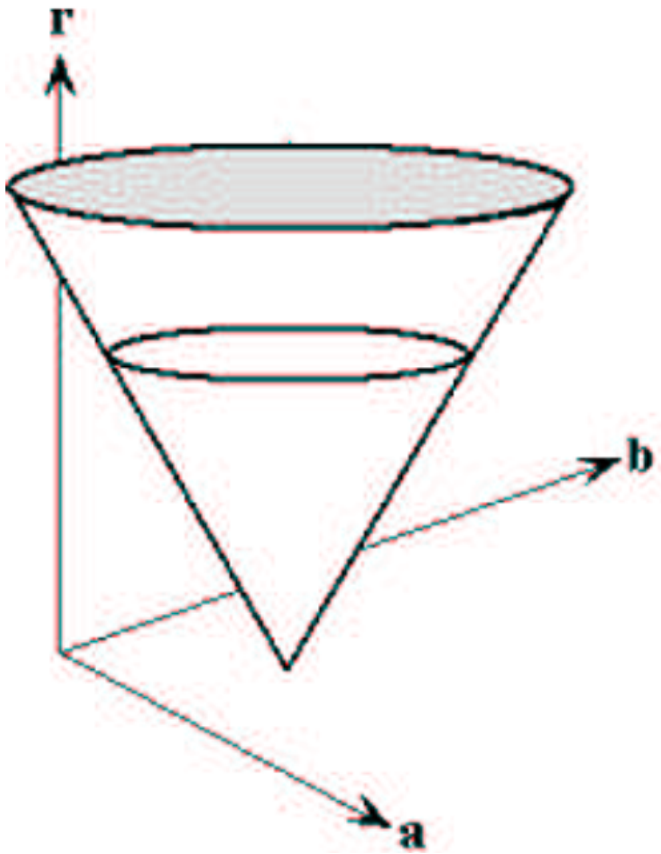
- $a = x - r * \cos\left(\theta * \frac{\pi}{180}\right)$
- $b = y - r * \sin\left(\theta * \frac{\pi}{180}\right)$



Ottenuto lo spazio dei parametri, si effettua una **quantizzazione** per limitare i valori ed ottenere lo **spazio dei voti** e si imposta una soglia come nel caso delle rette.
Ogni punto nello spazio immagine vota per una circonferenza nello spazio dei voti e le celle con numero di voti superiori alla soglia corrispondono alle circonferenze nell'immagine.



Nel caso in cui non si conosca il raggio delle circonferenze da cercare, tutti i parametri potrebbero variare e si parlerebbe di spazio tridimensionale.
Ogni punto nello spazio immagine corrisponderebbe alla superficie di un cono nello spazio dei parametri, la cui dimensione dipende dal range di raggi stabilito.
L'intersezione tra i coni corrisponde ad una circonferenza nell'immagine.



PASSI ALGORITMO HOUGH PER CERCHI

- Creare l'accumulatore H (tridimensionale);
- Applicare l'algoritmo di Canny per individuare i punti di edge;
- Per ogni punto di edge, calcolare **a = x - r * cos($\theta * \frac{\pi}{180}$), b = x - r * sin($\theta * \frac{\pi}{180}$)** e incrementare H(a, b, r) (voto);
- Tutte le celle con valore maggiore della soglia corrispondono alle circonferenze.

CARATTERISTICHE

Un'idea per ridurre il numero dei parametri è utilizzare la direzione del gradiente.
La trasformata di Hough si dimostra **robusta al rumore** e ad **interruzioni** nelle forme cercate.
Inoltre, può individuare più istanze di una forma in una singola esecuzione ed è **parallelizzabile**.
L'immagine può essere divisa in blocchi ed ognuno di essi assegnato ad un processore.
In questo caso, ogni core ha il proprio spazio dei voti e al termine si esegue la somma dei voti.
In OpenCV, per applicare Hough per rette bisogna passare l'immagine dopo aver applicato l'algoritmo di Canny per trovare gli edge.
Hough per cerchi richiede invece l'immagine originale, siccome esegue implicitamente Canny.

SOGLIATURA

Le tecniche di **sogliatura** si basano sull'analisi dei valori di intensità e su alcune proprietà per partizionare l'immagine in regioni.
L'analisi, tipicamente, è effettuata sull'**istogramma** dei valori di intensità.
Dato l'istogramma di un'immagine f(x, y), composta di oggetti chiari su sfondo scuro e con i valori raggruppati in due **mode**, per segmentare l'immagine bisogna scegliere una **soglia T** che separi le due mode, tale che:

$$g(x,y) = \begin{cases} 1, f(x,y) > T \\ 0, f(x,y) \leq T \end{cases}$$

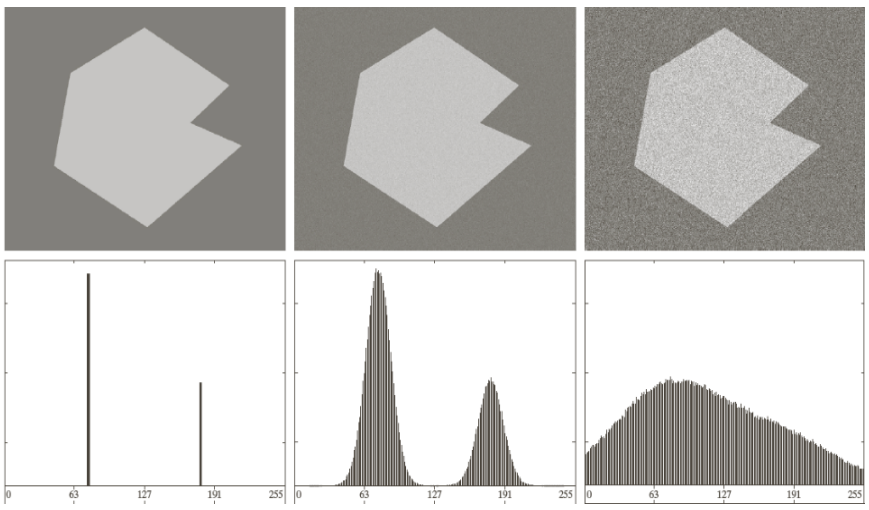
In una regione si avranno tutti i pixel dell'immagine con valore di intensità minore della soglia e viceversa.
g(x, y) restituisce un'**etichetta** che indica a quale regione appartiene un pixel.
Si distinguono le seguenti tipologie di sogliatura:

- **Globale:** applicata a tutta l'immagine;
- **Variabile:** modificata durante il processo;
- **Locale:** dipendente dall'intorno di ogni pixel;
- **Dinamica:** dipendente dalla posizione del pixel;
- **Multipla:** più soglie.

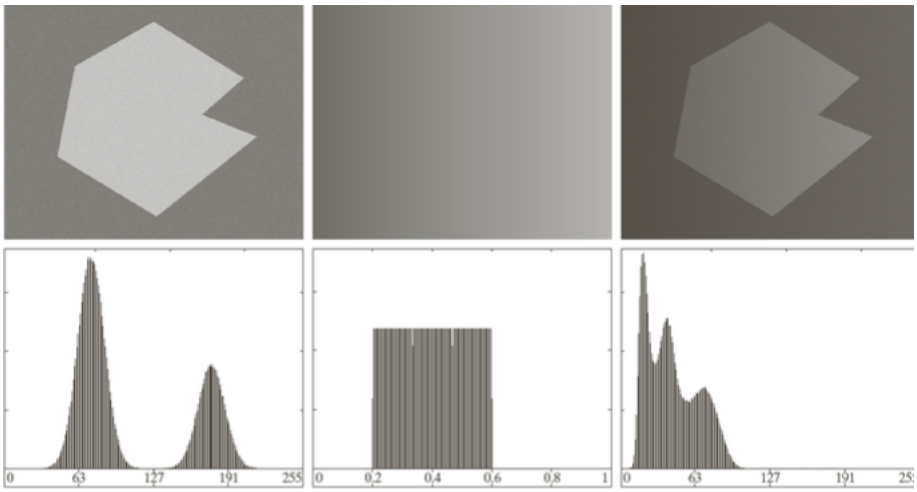
Ad esempio, utilizzando due soglie si avrà:

$$g(x,y) = \begin{cases} a, f(x,y) > T_2 \\ b, T_1 < f(x,y) \leq T_2 \\ c, f(x,y) \leq T_1 \end{cases}$$

I fattori che influenzano la sogliatura sono la distanza tra i picchi, il rumore, la dimensione degli oggetti rispetto allo sfondo e l'uniformità dell'illuminazione.
Se si fa riferimento al rumore, in un istogramma con due mode a "chiodo" qualsiasi valore di soglia compreso tra le due mode permette di ottenere la segmentazione.
Nel caso di un'immagine corrotta da rumore Gaussiano e ampia deviazione è addirittura impossibile trovare una soglia.



Anche l'illuminazione è importante.
Come è notabile dalla seguente immagine, non è possibile trovare una soglia per segmentare l'immagine, in quanto la separazione tra i due picchi non è più visibile.



SOGLIATURA GLOBALE

Quando le distribuzioni di intensità dei pixel di background e quelli di foreground sono distinte, è possibile applicare una **soglia globale**.
Il procedimento da seguire è il seguente:

- Stimare il valore iniziale della soglia T scegliendo un valore casuale tra massima e minima intensità;
- Segmentare l'immagine usando T in modo da ottenere due gruppi di pixel $\boldsymbol{G_1}$ e $\boldsymbol{G_2}$**,** tale che in un gruppo ci siano tutti i pixel con intensità minore della soglia e viceversa;
- Calcolare l'intensità media m_1 e m_2 dei due gruppi. È possibile farlo usando l'istogramma e moltiplicando l'intensità del picco per la frequenza per la quale si presenta;
- Ricalcolare la soglia come $\boldsymbol{T = (m_1 + m_2)/2}$;
- Ripetere i passi finché le soglie in due iterazioni successive non differiscono di un valore minore di un valore $\Delta \boldsymbol{T}$.

Per calcolare l'istogramma dell'immagine, si scandiscono tutti i pixel dell'immagine e si incrementa la cella della matrice che rappresenta l'istogramma nella posizione del valore di intensità del pixel analizzato.

$$\boldsymbol{H(f(x,y))++}$$

ALGORITMO DI OTSU

Il **metodo di Otsu** è un'alternativa valida alla **regola di decisione di Bayes** per **massimizzare la varianza interclasse** e **minimizzare l'errore medio**.
Opera esclusivamente sull'istogramma dell'immagine e separa i valori di intensità in due classi.
Data un'immagine di MxN pixel a L - 1 livelli di intensità, definiamo $\boldsymbol{n_i}$ come il numero di pixel dell'immagine con **intensità i** tale che $\boldsymbol{MN = \sum_{i=0}^{L-1} n_i}$.
L'**istogramma normalizzato** si ottiene dividendo ogni $\boldsymbol{n_i}$ per il numero di pixel **MN**, ovvero $\boldsymbol{p_i = \frac{n_i}{MN}}$ e quindi $\boldsymbol{\sum_{i=0}^{L-1} p_i = 1}$. Il risultato è la probabilità che un pixel assuma il valore di intensità **i** (tra 0 e 1).
Selezionando un'opportuna **soglia k**, compresa tra 0 e L - 1, si segmenta l'immagine ottenendo due **classi** $\boldsymbol{C_1}$ e $\boldsymbol{C_2}$, contenenti i pixel con intensità minore e maggiore della soglia.
La **probabilità** che un pixel appartenga a $\boldsymbol{C_1}$ è $\boldsymbol{P_1(k) = \sum_{i=0}^{k} p_i}$.
La **probabilità** che un pixel appartenga a $\boldsymbol{C_2}$ è $\boldsymbol{P_2(k) = \sum_{i=k+1}^{L-1} p_i = 1 - P_1(k)}$.
Il **valore di intensità medio** dei pixel assegnati a $\boldsymbol{C_1}$ è $\boldsymbol{m_1(k) = \frac{1}{P_1(k)} \sum_{i=0}^{k} (i+1) p_i}$.
Analogamente, quello dei pixel in $\boldsymbol{C_2}$ è $\boldsymbol{m_2(k) = \frac{1}{P_2(k)} \sum_{i=k+1}^{L-1} (i+1) p_i}$.
La **media cumulativa** fino al livello **k** è $\boldsymbol{m(k) = \sum_{i=0}^{k} (i+1) p_i}$.
L'**intensità media** dell'intera immagine è $\boldsymbol{m_G = \sum_{i=0}^{L-1} (i+1) p_i}$.

Per stimare l'**efficienza della soglia** al livello **k** si utilizza il valore $\boldsymbol{\eta = \frac{\sigma_B^2}{\sigma_G^2}}$, ovvero la **varianza interclasse** rispetto a quella **globale** (dispersione dei valori sull'intero istogramma).

$$\boldsymbol{\sigma_G^2 = \sum_{i=0}^{L-1} (i + 1 - m_G)^2 p_i}$$

$$\boldsymbol{\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2 = \frac{(m_g P_1 - m)^2}{P_1(1 - P_1)}}$$

Al crescere della distanza delle due medie, la varianza interclasse aumenta.
Essa è una **misura della separabilità** tra le classi.
L'obiettivo è trovare un **k** che **massimizzi** $\boldsymbol{\sigma_B^2}$, in quanto il valore al denominatore è costante e non cambia al variare di k.
La **soglia ottimale** è il valore **k*** che **massimizza** $\boldsymbol{\sigma_B^2(k)}$.
Se $\boldsymbol{\sigma_B^2(k)}$ è massimo per più di un valore, si calcola la **media** dei valori di k e, una volta trovato **k***, si procede a **segmentare** l'immagine.

PASSI DELL'ALGORITMO DI OTSU

- Calcolare l'istogramma normalizzato dell'immagine;
- Calcolare le somme cumulative $P_1(k)$;
- Calcolare le medie cumulative m(k);
- Calcolare l'intensità media globale m_G;
- Calcolare la varianza interclasse $\sigma_B^2(k)$;
- Calcolare la soglia k*;
- Calcolare il valore di separabilità $\eta(k*)$.

SMOOTHING E EDGE

La presenza di **rumore** rende difficile la sogliatura.
Effettuare lo **smoothing** dell'immagine prima dell'operazione aumenta le prestazioni.
Nel caso in cui la regione da segmentare è così piccola che il suo contributo all'istogramma è trascurabile rispetto a quello del rumore, lo smoothing non risolve il problema.
Per migliorare l'istogramma, al fine dell'immagine è possibile considerare solo i pixel che si trovano sugli **edge** o in loro prossimità.
In questo modo, l'istogramma è caratterizzato da picchi di stessa altezza e la **probabilità** che un pixel appartenga al foreground è la stessa che appartenga al background.
Per estrarre i pixel si utilizza il **Laplaciano** o la **magnitudo del gradiente**.

- Calcolare un'immagine di edge;
- Individuare un valore di soglia T per individuare gli edge;
- Applicare la soglia all'immagine di edge ottenendo un'immagine g_T;
- Calcolare l'istogramma utilizzando solo i pixel dell'immagine di input che corrispondono alle posizioni dei pixel con valore 1 in g_T;
- Utilizzare l'istogramma per segmentare l'immagine con il metodo di Otsu.

SOGLIE MULTIPLE

Il metodo di Otsu può essere **generalizzato** per **K classi**.
Tuttavia, con troppe classi il metodo perde di significato.

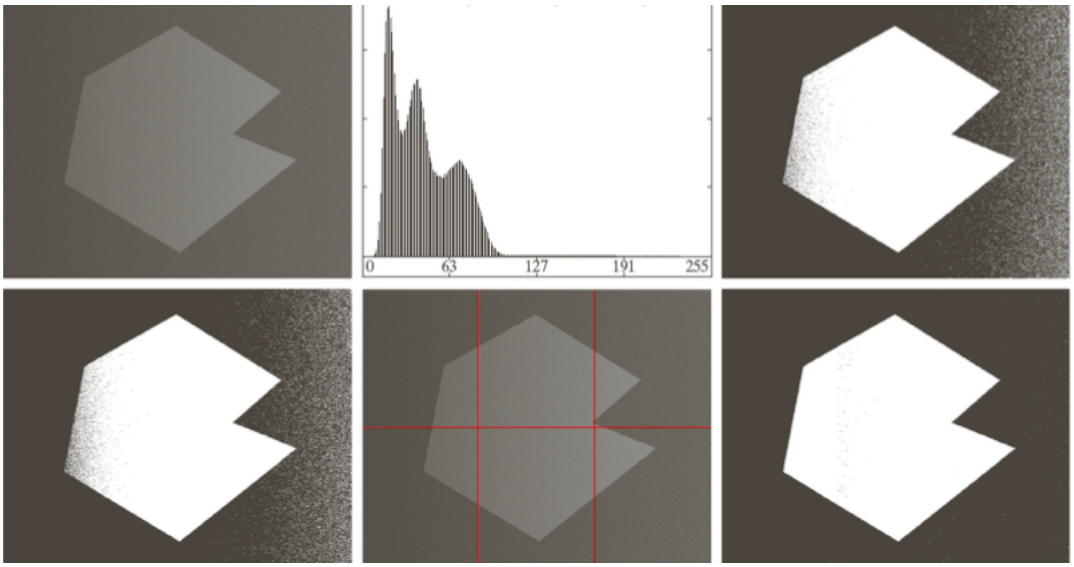
$$\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2 + P_3(m_3 - m_G)^2$$

La procedura inizia selezionando una **prima soglia** $k_1 = 1$ per poi **incrementare la soglia** k_2 a partire dai valori maggiori di k_1 e minori di **L - 1**.
Successivamente si **incrementa** k_1 e si **incrementa** k_2 a partire dai valori maggiori di k_1.
Al termine della procedura si ottiene una **matrice** all'interno del quale si trova il valore **massimo** in corrispondenza di k_1^* e k_2^*.
L'immagine segmentata si ottiene da:

$$g(x,y) = \begin{cases} a \text{ se } f(x,y) \leq k_1^* \\ b \text{ se } k_1^* < f(x,y) \leq k_2^* \\ c \text{ se } f(x,y) > k_2^* \end{cases}$$

SOGLIATURA VARIABILE

Uno dei metodi più semplici consiste nel suddividere un'immagine in **rettangoli non sovrapposti**.
Ogni rettangolo corrisponde ad un'immagine e ci applico Otsu.



Un approccio più generale prevede di calcolare una soglia considerando una o più proprietà dell'intorno di ogni pixel, ad esempio calcolando la **media** e la **deviazione standard** dei valori di intensità nell'intorno di ogni pixel.

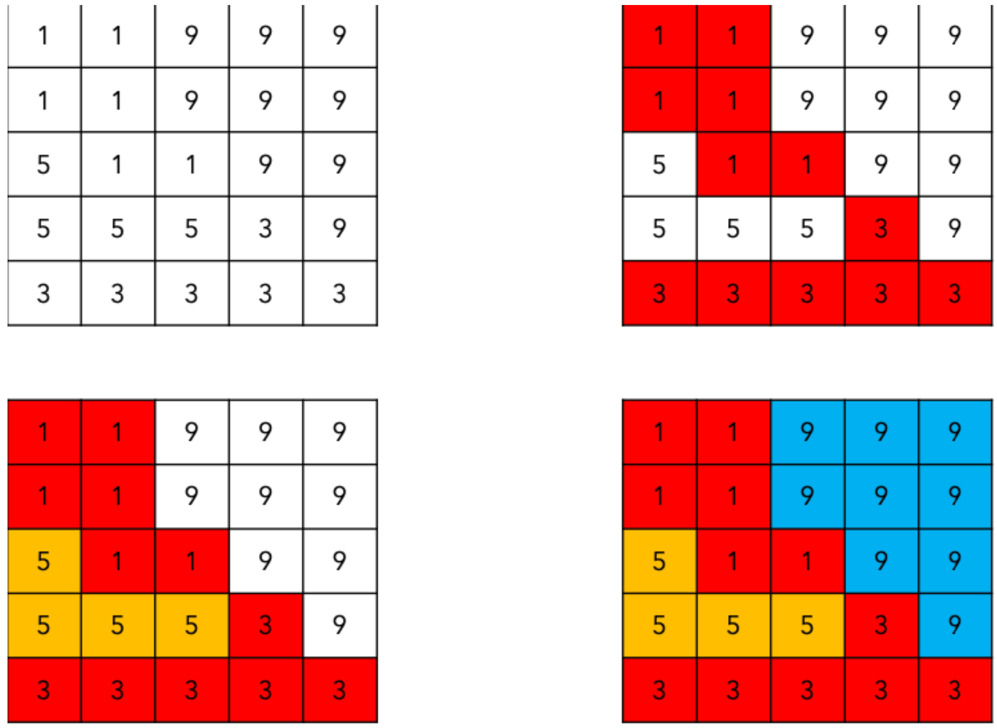
REGION GROWING

La tecnica del **region growing** consiste nel **raggruppare** i **pixel** in regioni più grandi in base a criteri.
Il procedimento inizia da pixel iniziali detti **seed** e si propaga ai pixel **adiacenti** che rispettano delle **proprietà**, i quali vengono aggiunti alla regione (una sorta di BFS).
Una volta che la regione non è più espandibile, ripeto il procedimento su di un pixel non già presente in una regione.
Il **criterio di similarità** dipende dal tipo di immagine (per immagini in scala di grigio si usano descrittori basati sui livelli di intensità).
Problema: raggruppare pixel simili ma non adiacenti causa **segmentazioni inconsistenti** (**oversegmentazione**).
Altro problema è la **regola d'arresto**, utile per arrestare l'accrescimento della regione quando i pixel non soddisfano i criteri di inserimento.
Considerare solo i **descrittori locali** non è sufficiente, ma bisogna considerare le caratteristiche di tutti i pixel già inseriti nella regione (la **storia**).

PASSI DELL'ALGORITMO REGION GROWING

Sia **f(x, y)** l'**immagine** di input.
Sia **S(x, y)** la **matrice dei seed** che assegna il valore 1 alle posizioni dei seed e 0 alle altre.
Sia **Q** un **predicato** da applicare ad ogni pixel.
I passi dell'algoritmo sono:

- Formare l'immagine f_Q che nel punto (x, y) contiene il valore 1 se Q(f(x, y)) è vero e 0 altrimenti;
- Aggiungere ad ogni seed i pixel impostati ad 1 in f_Q che risultano [4, 8] connessi al seed;
- Marcare ogni componente connessa con un'etichetta diversa.



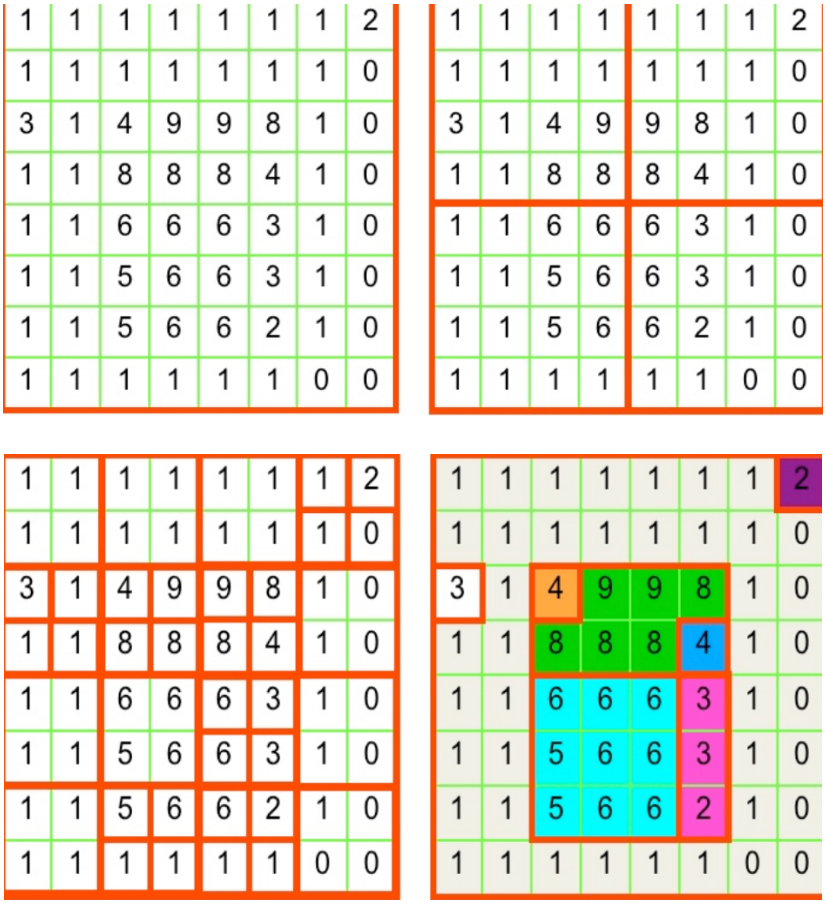
SPLIT AND MERGE

Un approccio alternativo consiste nel **dividere** l'immagine in **regioni disgiunte** di forma e dimensione arbitrarie e successivamente **fonderle** in base a **criteri di similarità**.
Sia **R** la regione corrispondente all'intera immagine e **Q** un predicato, è possibile dividere R in regioni sempre più piccole finche il predicato non risulta vero.
Una strategia utilizzata è dividere le regioni in **quadranti**.
Partendo da R, se **Q(R) = falso** si **divide** R in quattro quadranti.
I quadranti per cui **Q** è falso si dividono ulteriormente in quadranti e così via finché il predicato non risulta vero o non si è raggiunta una soglia minima (una regione di 1 pixel non ha valenza).
Al termine della fase di splitting, la partizione finale potrebbe contenere regioni adiacenti con caratteristiche simili.
Per questo motivo, queste regioni possono essere **fuse**.
Questo è possibile se, date due regioni adiacenti R_i e R_j, si ha $Q(R_i \ U \ R_j)$ = vero.
Per semplicità (le operazioni insiemistiche sono costose) se il predicato è vero su entrambe le regioni, assumo che sia vero per la loro unione.

PASSI DELL'ALGORITMO SPLIT AND MERGE

- Dividere in quattro quadranti tutte le regioni per cui il predicato risulta falso;
- Quando non è possibile dividere le regioni, applicare il processo di merging a tutte le regioni adiacenti per cui $Q(R_i \ U \ R_j)$ = vero;
- Il processo termina quando non sono effettuabili unioni.

Solitamente si stabilisce una **dimensione minima** della regione oltre la quale non si effettua lo split.
La fase di merge può essere effettuata se il predicato è vero per le singole regioni adiacenti e non per la loro unione.



CLUSTERING

Il **clustering** consiste nel raggruppare pixel in base a caratteristiche di **intensità**, **colore** e **posizione**.
Nell'algoritmo **k-means** si creano **k cluster disgiunti** rappresentati da **k centri** equivalenti alle medie dei valori di intensità dei pixel.
Ogni pixel è assegnato al cluster rappresentato dalla media ad esso più vicina.
I pixel di ogni cluster sono **etichettati** con il valore del centro del cluster a cui appartengono.
La scelta migliore per il centro è quella che minimizza la **Sum of Squared Distances** tra tutti i punti ed il centro più vicino.
L'obiettivo è minimizzare la **varianza** in ogni cluster.

$$c^*, \delta^* = \boldsymbol{argmin} \frac{1}{N} \sum_{j}^{N} \sum_{i}^{K} \delta_{ij} (c_i - x_j)^2$$

Se si conoscono i centri dei cluster, si possono assegnare i pixel al cluster con centro più vicino.
In pratica, dato un pixel, lo assegno al cluster la cui media è più simile ad esso.
Se si conosce a quale cluster appartiene ogni pixel è possibile calcolare il centro di ogni cluster.
Il problema è come calcolare i gruppi: in maniera casuale o secondo un ragionamento.

1. **Inizializzare** i centri di ogni cluster;
2. **Assegnare** ogni pixel al cluster con centro più vicino;
3. Per ogni pixel calcolare la distanza dai centri di ogni cluster ed assegnare il pixel al cluster con centro più vicino;
4. Aggiornare i centri calcolando la media dei pixel di ogni cluster;
5. Ripetere i passi 2 e 3 finché il centro di ogni cluster non è più modificato.

L'algoritmo potrebbe terminare in un **minimo locale**, ragion per cui lo si può rieseguire più volte.

K-MEANS++

I centri iniziali possono essere scelti **casualmente** o in maniera più precisa.
Se sono **troppo vicini** il risultato non è ottimale.
Il primo centro è scelto casualmente ed ogni altro deve essere **distante** da quello scelto precedentemente.
Come distanza si utilizza la **distanza euclidea** ma l'algoritmo diventa molto costoso.
L'algoritmo di inizializzazione prevede:

- Scegliere il primo centro c_1 in maniera **casuale**;
- Calcolare la distanza di ogni pixel da c_1;
- Scegliere tra i restanti pixel il prossimo centro c_2 con probabilità direttamente proporzionale alla sua distanza da c_1 (pixel con valore più **lontano**);
- Ripetere fino a scegliere i **k** centri iniziali.

Lo stesso procedimento è utilizzato per immagini a colori.
Sebbene l'obiettivo sia creare gruppi di pixel compatti, pixel con valori simili potrebbero trovarsi distanti all'interno dell'immagine.
In questo caso, i cluster non sono componenti connesse.
Per evitare il problema, si utilizza la **posizione** dei pixel per ottenere regioni più compatte.
Ogni pixel è rappresentato da un **vettore** le cui componenti sono le **coordinate spaziali** e le **componenti del colore** in caso di immagini a colore o il **livello di intensità** in caso di immagini in scala di grigio.
La **complessità** è **O(kNd)**, dove **k** è il numero di cluster, **N** il numero di pixel e **d** il numero di feature (5 nel caso in cui si considera anche la posizione).

MEAN SHIFT

L'algoritmo **mean-shift** cerca, per ogni valore, la **moda più vicina** nello spazio delle intensità.
Per ogni valore si considera una **finestra di ampiezza W**, si calcola la **media** in tale finestra e si **sposta** la finestra sulla media calcolata, ripetendo il procedimento finché la media non cambia.
Dato l'istogramma, la media, che va presa sull'asse x, è calcolabile mediante il prodotto tra la **frequenza** e il **valore di intensità**.
L'algoritmo non richiede di specificare il numero di cluster, l'ampiezza della finestra determina il risultato e converge naturalmente al numero naturale di cluster.
La **complessità** è **O(kN^2d)**, dove **k** è il numero di cluster, **N** il numero di pixel e **d** il numero di feature (5 nel caso in cui si considera anche la posizione).

```
for p in f(x,y)
   while |p_out^i-p_out^i-1|>th
      shift=n=0
      for pt in f(x,y)
         if pt ∈ W
            shift+=pt
            n++
      p_out=shift/n
```

```
for p in f(x,y)
   while |p_out^i-p_out^i-1|>th
      shift=n=0
      for pt in f(x,y)
         d=dist(p,pt)
         w=kernel(d,W)
         shift+=pt * w
         n+=w
      p_out=shift/n
```

MORFOLOGIA MATEMATICA

La **morfologia** di un'immagine descrive le forme rappresentate.
Ad un basso livello, le immagini a colore sono considerate **insiemi di pixel** e gli oggetti sono rappresentati come **gruppi di pixel di foreground** distribuiti in base a delle caratteristiche.
I **processi morfologici** sono operazioni su insieme di punti del piano, generalmente definite rispetto ad un insieme chiamato **elemento strutturante ES** (array di pixel nelle immagini).
Bisogna fare attenzione che l'ES sia interamente contenuto nell'immagine (altrimenti padding).
Essi sono descritti mediante una **convenzione**:

- **Cella riempita:** appartenente all'elemento strutturante;
- **Cella vuota:** non appartenente all'elemento strutturante;
- **Cross:** non importa.

La **riflessione**, ovvero la rotazione di 180 gradi di un ES intorno alla sua origine, è definita come: $\widehat{\boldsymbol{B}} = \{-\boldsymbol{b}|\boldsymbol{b} \in \boldsymbol{B}\}$**.**
La **traslazione** di B tramite z è: $(\boldsymbol{B})_{\boldsymbol{z}} = \{\boldsymbol{b} + \boldsymbol{z}|\boldsymbol{b} \in \boldsymbol{B}\}$ e consiste nel traslare tutti i punti di B di un valore z.

EROSIONE

Dato un insieme di pixel di foreground A, l'**erosione** di **A** attraverso **B** è: $\boldsymbol{A \ominus B = \{z|(B)_z \subseteq A\}}$.
In pratica, tutti i punti z di B che appartengono ad A.
Essa è utilizzata per estrarre i pixel di foreground da un'immagine.
Si sposta l'origine dell'elemento strutturante in ogni pixel dell'immagine A e si valuta se la definizione dell'operazione è valida.
Se l'origine di B è **traslata** su un pixel a appartenente ad A e tutti gli elementi di B sono coperti da un elemento di A, il pixel a appartiene al foreground.
L'erosione può essere utilizzata anche per effettuare un **filtraggio morfologico:** cancella i dettagli dell'immagine più piccoli dell'elemento strutturante e riduce il rumore.

DILATAZIONE

Dati gli insiemi A e B, la **dilatazione** di A attraverso B è: $\boldsymbol{A \oplus B = \{z|(\widehat{B})_z \cap A \neq 0\}}$.
La dilatazione ha effetti simili al filtraggio passa-basso: i dettagli sono assorbiti ed è utile per riempire le interruzioni.
Erosione e dilatazione sono **operazioni duali** rispetto al **complemento** e la **riflessione**.
Se l'elemento strutturante è simmetrico ($\boldsymbol{\widehat{B} = B}$), l'erosione di A è ottenibile dilatando il background con lo stesso elemento strutturante e complementando il risultato.

APERTURA E CHIUSURA

L'**apertura** di un insieme A attraverso B è definita come: $\boldsymbol{A \circ B = (A \ominus B) \oplus B}$, ovvero come un'operazione di erosione seguita da una di dilatazione.
La **chiusura** dell'insieme A attraverso B è definita come: $\boldsymbol{A \cdot B = (A \oplus B) \ominus B}$, ovvero come un'operazione di dilatazione seguita da una di erosione.
Supponendo di avere un'immagine di un'impronta digitale, con l'erosione eliminiamo il rumore esterno e con la dilatazione quello interno.
L'apertura ha causato l'interruzione di alcune dorsali, per questo motivo dilatiamo per recuperarle ed erodiamo per ripristinarne gli spessori.

TRASFORMAZIONE HIT OR MISS

La **trasformazione hit or miss** è utilizzata per individuare la posizione di un oggetto all'interno di un'immagine.
Gli oggetti devono essere **disgiunti**, ovvero essere separati da almeno un pixel di background.
Si effettua l'intersezione tra l'erosione di A attraverso B1 e l'erosione del complemento di A attraverso B2.

MORFOLOGIA IN SCALA DI GRIGIO

Nel caso di immagini in scala di grigio, sono considerate **funzioni f(x, y)** e non insiemi e gli elementi strutturanti **funzioni b(x, y)**.
Essi si dividono in **flat** e **non flat**.
Un **ES flat** rappresenta una **matrice binaria** e prende in considerazione solo la presenza o meno del pixel.
Un **ES non flat** rappresenta una **matrice tridimensionale** e prende in considerazione anche il valore del pixel.
Per quanto riguarda le operazioni di apertura e chiusura, esse sono definite come nel caso delle immagini a colore. L'apertura limita i picchi e la chiusura riempie le valli (filtraggio passa-alto/passa-basso).
Dalla combinazione di tali operazioni si ottiene uno **smoothing morfologico.**
Le **trasformazioni top-hat** e **bottom-hat** sono utilizzate per preservare l'informazione delle operazioni di apertura e chiusura in immagini con oggetti chiari su sfondo scuro o viceversa.

$$\boldsymbol{T_{hat} = f - (f \circ b) \; e \; B_{hat} = (f \cdot b) - f}$$