"Algoritmi Elaborazione delle Immagini"
Codici C++ OpenCV

2018

Indice

Capitolo 1

Introduzione

In questo documento vengono descritti brevemente tutti gli algoritmi presenti nella collezione di *GitHub*
(https://github.com/BesTeam-ing/ELIMcollection).
Prima di cominciare a consultare il seguente documento e per la totale comprensione dei codici, si consiglia **vivamente** di averne studiato il loro funzionamento da un vero e proprio libro di Elaborazione delle Immagini.

Capitolo 2

Dominio delle Frequenze

L'obbiettivo dell'elaborazione nel dominio delle frequenze é quello di digitalizzare l'immagine. I passi da effettuare per filtrare nel dominio delle frequenze sono i seguenti, supponendo che si voglia filtrare una frequenza compresa tra 5 Hz e 20 Hz.

1. Data una immagine in input $f(x, y)$ di dimensione **MxN**, ricavo i parametri di *Padding* **P** e **Q**, utilizzati per centrare l'immagine. Di solito $P = 2M$ e $Q = 2N$.

2. Formo l'immagine padded $f_p(x, y)$ di dimensione **PxQ**, estendendo *f(x,y)* con il necessario numero di 0.

3. Moltiplico $f_p(x, y)$ per $(-1)^{x+y}$ per centrare la trasformata.

4. Calcolo il DFT di *F(u,v)* dell'immagine.

5. Genero una funzione filtro e moltiplico il filtro per la DFT.

6. In questo caso, applico un filtro di *Smoothing* (Studiare i vari filtri e relative formule). Ció per permettere il passaggio delle basse frequenze.

$$H(u, v) = \begin{cases} 1 & \text{se } D(u, v) \leq D_0 \\ 0 & \text{se } D(u, v) > D_0 \end{cases}$$

   Dove: $D_0 = 20Hz$

7. Ricavo l'immagine nel *Dominio Spaziale*:
   $g_p = I_{DFT}(F(u, v) * H(u, v)) * (-1)^{x+y}$

8. L'immagine si ottiene dall'estrazione del primo quadrante in alto-sinistra di dimensioni **MxN**.

9. Utilizzando l'immagine ricavata nel passo precedente, ripeto i passaggi 1-6 ed applico un filtro di *Sharpening* per filtrare le alte frequenze.

$$H(u, v) = \begin{cases} 0 & \text{se } D(u, v) < D_0 \\ 1 & \text{se } D(u, v) \geq D_0 \end{cases}$$

   Dove: $D_0 = 5Hz$

10. Ricavo l'immagine nel *Dom. Spaziale* come precedentemente nel punto 7.

- Il vantaggio dell'utilizzo del filtro ideale é che taglia perfettamente la frequenza al di sotto e/o al di sopra di una determinata frequenza. Mentre tutte le altre frequenze vengono totalmente annullate.
  Lo svantaggio, invece, é che provoca *Ringing*.

- Utilizzare il *Padding* prima della DFT é un metodo computazionalmente efficiente per interpolare un gran numero di punti. Aggiungere uno *zero-padding*, ovvero centrare l'immagine, permette poi di ottenere una DFT piú grande, che potrá contenere piú valori e a sua volta risulterá piú precisa.
  Essa, in particolare, ci semplifica la risoluzione di un singolo picco di segnale, é quindi utile quando abbiamo una frequenza isolata.

Capitolo 3

Segmentazione Colori

3.1 K-Means

É un algoritmo di clustering che permette di suddividere gruppi di oggetti in K partizioni. L'obiettivo é quello di minimizzare la somma delle distanze di ciascun oggetto dal centroide a cui é assegnato.

Algoritmo

1. Scelgo i K punti e creo K clusters.
2. Calcolo il centroide per ogni gruppo.
3. Scorro l'immagine e cerco la distanza minore tra il pixel considerato ed i K clusters.
4. Controllo che il valore del nuovo centroide sia diverso dal vecchio, in caso contrario l'algoritmo termina.

3.2 Isodata

É un algoritmo di clustering che, per formare i clusters, utilizza molte medie tra i clusters.

Algoritmo

1. Calcolo la distanza che intercorre tra ogni punto e tutte le medie scelte. Attribuendo alla media la distanza minima.
2. Suddivido i clusters la cui *varianza* supera una certa soglia.
3. Se, invece, i clusters sono troppo piccoli, li raggruppo.
4. Ricalcolo le medie e ripeto finché i passi 2 e 3 possono essere applicati.

3.3 Ohlander

Il metodo si applica ad una immagine a 3 colori RGB considerando l'istogramma per ogni banda di colore. A partire da un istogramma, questo viene clusterizzato in piú clusters e vengono determinate le componenti dell'immagine, in pratica, se abbiamo come immagine una palla rossa, sapremo per certo che i pixel di quella palla rossa faranno parte dell'istrogramma della palla rossa.
Tra i pixel prelevati, peró, potrebbero esserci anche quelli di altri oggetti con componenti rosse. Analizzati i vari istogrammi delle bande e create le necessarie maschere che raccolgono i pixel degli oggetti desiderati, bisogna inserire le maschere in uno *stack* e procedere iterativamente.

Algoritmo

Dopo aver inserito le maschere nello stack posso procedere iterativamente:

1. Estraggo una maschera dallo stack e la applico a quella dell'immagine iniziale.
2. Analizzo gli istogrammi delle tre bande dell'immagine ottenuta dall'applicazione della maschera. Se questa nuova maschera ha prelevato piú di un oggetto, realizzo la maschera che mi individua quel particolare oggetto e la carico nello stack. Altrimenti i pixel prelevati dalla maschera faranno parte tutti dello stesso cluster.
3. Salvo i cluster e passo ad analizzare le altre maschere nello stack.

3.4 Tagli Minimali

Rappresento l'immagine come un grafo nel quale i vertici rappresentano i pixel e gli archi rappresentano le connessioni tra i pixel. Ogni arco possiede un peso che rappresenta la similiaritá tra i pixel connessi.
Se si considerano due insiemi di nodi **A** e **B**, il taglio sará dato dalla rimozione dei collegamenti da A a B e dalla loro sommatoria. Ovvero dalla sommatoria dei pesi dei vertici connessi tra A e B. Da questa sommatoria si ricava il taglio minimale, ovvero la sommatoria di peso minore tra tutte.
$\sum_A \sum_B w(u, v)$

3.5 Tagli Normalizzati

Si utilizza poiché il taglio minimale favorisce il taglio di piccoli nodi, ovvero puó portare ad una serie di problemi, potrebbe non considerare gruppi di pixel di ugual valore ma molto distanti tra loro (Visti, dunque, come cluster a parte). Il *Taglio Normalizzato* risolve questo problema, esso tiene conto anche dell'*associazione* tra i nodi, permettendoci di considerare nel dettaglio anche nodi che non sarebbero stati inclusi nel taglio minimale, poiché troppo distanti.
$Ncut = \frac{cut(A,B)}{asso(A,V)} + \frac{cut(A,B)}{asso(B,V)}$ Dove:
$asso(A, V) = \sum_A w(u, v)$

Capitolo 4

Trasformata di Hough

4.1 Trasformata di Hough per Rette

La trasformata di Hough é una tecnica che ci consente di individuare rette,cerchi e forme all'interno di una immagine.
Nel caso delle rette, si vuol stabilire la presenza di rette all'interno dell'immagine. L'equazione della retta piú conosciuta é quella $y = mx + q$ dove m é il coefficiente angolare e q é l'intercetta, entrambe servono a rappresentare univocamente la retta su di un piano.
In questo caso, peró, se si vuol rappresentare una retta perpendicolare all'asse delle ascisse, o parallela all'asse delle ordinate, si ha che m o q tenderanno ad infinito.
Per ovviare a questo problema si utilizzano le coordinate *polari* date dalla seguente equazione:
$r = x\sin(\theta) + y\cos(\theta)$ (Nel caso in cui si scorrano prima le X e poi le Y).
Segue un processo di *voting* per verificare che il punto preso in considerazione faccia parte di una retta. Il voting verrá infine confrontato con una soglia arbitraria per decidere se quel punto appartiene alla retta. In caso affermativo, la retta viene disegnata.

Algoritmo

1. Utilizzo un *Filtro di Smoothing Gaussiano* per eliminare il rumore dall'immagine e agevolare la ricerca di edges.

2. Utilizzo l'algoritmo *Canny Edge Detection* per rilevare i bordi ed escludere lo sfondo (Il quale diventa nero).

3. Poiché utilizzo le coordinate polari per rette calcolo ρ e θ:
   $\rho = \frac{Cols}{Rows} * \sqrt{2}$
   $0° < \theta < 180°$ Poiché non ci interessa il verso della retta.

4. Dichiaro una matrice *Accumulatore* di dimensioni [ρ **x** θ] inizializzata a 0, che utilizzo per il processo di voting.

5. Scorro l'immagine e, se sono in presenza di un punto di edge, sparo un fascio di rette aggiornando l'accumulatore per ognuna della rette.

6. Scorro l'accumulatore confrontando ciascun valore con una soglia arbitraria. Nel caso in cui il valore sia maggiore della soglia, tale punto fará parte di una retta nell'immagine.

4.2 Trasformata di Hough per Cerchi

La trasformata di Hough é una tecnica che ci consente di individuare rette,cerchi e forme all'interno di una immagine.
Nel caso delle circonferenze, vogliamo trovare la presenza di circonferenze all'interno dell'immagine. Pertanto la formula utilizzata per la ricerca di una circonferenza é:
$x_c = x - \rho \sin(\theta)$
$y_c = y - \rho \cos(\theta)$
Verranno quindi dichiarate le due variabili. Inoltre, per evitare, di lavorare in uno spazio 3D, computazionalmente molto oneroso, decidiamo di assegnare un set di raggi entro i quali lavorare, avremo quindi le varibili r_{min} e r_{max}

Algoritmo

1. Utilizzo un *Filtro di Smoothing Gaussiano* per eliminare il rumore dall'immagine e agevolare la ricerca di edges.

2. Utilizzo l'algoritmo *Canny Edge Detection* per rilevare i bordi ed escludere lo sfondo (Il quale diventa nero).

3. Fisso un *Range* che avrá per estremi r_{min} e r_{max}.

4. Dichiaro una matrice *Accumulatore* di dimensioni [**raggio x DimX x DimY**] inizializzata a 0, che utilizzo per il processo di voting.

5. Scorro l'immagine e, se sono in presenza di un punto di edge, disegno una serie di circonferenze che vanno da r_{min} a r_{max} e centrate in quel punto, aggiornando l'accumulatore.

6. Scorro l'accumulatore confrontando ciascun valore con una soglia arbitraria. Nel caso in cui il valore sia maggiore della soglia posso disegnare la circonferenza in quel punto.

Capitolo 5

Edge Detection

5.1 Harris Corner Detection

L'algoritmo di Harris é un modello di *Edge Detection*, ovvero quegli algoritmi che si occupano di individuare bordi all'interno di una immagine mediante i bruschi cambiamenti locali di intensitá.
In base al loro profilo di intensitá essi possono essere classificati in: *a Gradino , a Rampa, a Radice*.
Harris, in particolare, si occupa dell'individuazione di *Corners* ovvero di un contorno comune tra due edge. Per calcolare il punto di corner consideriamo un intorno $7x7$ di un punto e ne calcoliamo la **Matrice di Covarianza**:

$$C = \begin{bmatrix} \sum lx^2 & \sum lxly \\ \sum lxly & \sum ly^2 \end{bmatrix}$$

In seguito si calcola il **Determinante** e la **Traccia** della matrice C ed applico la formula:
$R = det - k(Traccia)^2$
Se la R ottenuta é maggiore di una determinata soglia, quel punto é un corner.

Algoritmo

1. Applico, dove necessario, una conversione in *B/N*
2. Applico un filtro di *Sharpening*, in questo caso preferisco usare *Roberts* per evidenziare i bordi e soprattutto per trovare il *Gradiente* di X e Y.
3. Scorro l'immagine utilizzando una finestra di dimensioni $[7x7]$ per ogni punto.
4. Calcolo la matrice di *Covarianza* C in ognuno di questi punti.
5. Trovo il *Determinante* e la *Traccia* della matrice C e ricavo R applicando la formula.
6. Confronto R con una soglia arbitraria, se il valore risulta maggiore, allora segno quel punto come punto di corner.

NB. Se volessi rendere piú pulita la stampa dei punti di corner, dovrei filtrare tra di loro i punti vicini in modo da inglobarli in un solo punto.

5.2 Canny Edge Detection

L'algoritmo Canny é un modello di *Edge Detection*, ovvero quegli algoritmi che si occupano di individuare bordi all'interno di una immagine mediante i bruschi cambiamenti locali di intensitá.
In base al loro profilo di intensitá essi possono essere classificati in: *a Gradino , a Rampa, a Radice*.
Canny, in particolare, si occupa dell'individuazione di *edges*, ovvero dei bordi.
A differenza dei filtri tradizionali (Prewitt,Sobel,etc) che producono immagini dove gli edges sono si ben visibili, ma non direttamente utilizzabili in quanto sono spezzettati in molti frammenti, il Canny risolve questo problema individuando bordi ben connessi tra di loro.

Algoritmo

1. Applico un *Filtro di Smoothing Gaussiano* per rendere piú piane le zone uniformi e ridurre il rumore.

2. Applico un *Filtro di Sharpening Passa-Alto*, preferendo quello di *Roberts* in quanto computazionalmente meno oneroso, per stimare il *modulo* e la *direzione* del gradiente. (Posso farlo utilizzando o la formula o l'apposita maschera).

3. Sopprimo i punti di non massimo locali, ovvero quei punti che non sono massimi locali rispetto alla direzione del gradiente. Controllo che il valore del pixel sull'edge sia il massimo tra il suo intorno 4-connesso.

4. Utilizzo una *Sogliatura con Isteresi* per meglio estrarre il contorno e far si che esso risulti ben connesso (A differenza di Roberts,etc). Definisco quindi due punti $T1$ e $T2$ con $T1 > T2$ che confronteró con ogni punto in questa maniera:

   - Se $punto < T1$ lo scarto.
   - Se $punto > T2$ lo prendo.
   - Se $T1 < punto < T2$ lo prendo soltanto se i 4-connessi soddisfano $punto > T2$.

Capitolo 6

Segmentazione di Regioni

6.1 Region Growing

Il Region Growing é una tecnica orientata a segmentazione di regioni. L'immagine R viene partizionata in sottoregioni R1, R2, ... , RN le quali soddisfano un predicato di veritá che le accumuna.
Esistono diversi approcci che segmentano la regione, il Region Growing, in particolare, é un approccio *Bottom-Up* ovvero parte dai singoli pixel per formare poi una regione completa. Il pixel da cui partire é chiamato *Seed*.
In una immagine a colori un possibile predicato di veritá é appunto il colore. Ovvero per ogni regione associo un colore e verifico, attraverso il calcolo della *Distanza Euclidea* se é soddisfatto.

Algoritmo

1. Definisco i *Seed*.
2. Inizializzo una matrice di *bool* che porterá il conto dei pixel visitati per ogni regione.
3. Ciclicamente, parto da un seed e costruisco la regione che li accomuna per predicato di veritá:
   - Verifico di non trovarmi sui lati dell'immagine.
   - In caso affermativo, calcolo la distanza euclidea tra il pixel su cui mi trovo ed il seed. Se questa distanza é minore di un threshold, esso fa parte della regione associata al seed.
   - Ripeto fintanto che la matrice *Visited* non risulterá tutta *true*.

6.2 Split and Merge

Lo Split and Merge é una tecnica orientata a segmentazione di regioni. A differenza del Region Growing, esso adotta un approccio *Top-Down* ovvero a partire

dall'immagine si determinano le regioni.
L'idea é quella di dividere l'immagine in 4 parti di uguale dimensione, In queste sottoregioni si verifica se esse soddisfano un determinato predicato. In caso negativo, vengono a loro volta divise in 4 sottoregioni e cosí via. Il processo termina nel momento in cui il predicato assume valore *true* oppure la dimensione della sottoregione risulta essere troppo piccola rispetto ad una dimensione arbitraria.
In seguito avviene il processo di **Merging**: tutte le regioni adiacenti che soddisfano uno stesso predicato vengono unite in un'unica regione piú grande. Il processo termina nel momento in cui non é piú possibile unire.

Algoritmo

1. Dichiaro una *struct Region* per meglio definire le informazioni di ogni regione.
2. Procedo allo *splitting* e se una regione non rispetta il predicato, la divido in ulteriori 4 sottoregioni di uguale dimensione, salvando le relative informazioni.
3. Utilizzo il *merging* e analizzo le sottoregioni adiacenti, verificando se esse rispettano il predicato. In caso affermativo le si unisce in un'unica regione piú grande.

Il predicato che prendono in considerazione, verifica le seguenti proprietá:

- La *Deviazione Standard* deve essere piú piccola di un dato K.
- Le dimensioni minime consentite per ogni sottoregione devono essere maggiori di un determinato *threshold*.

Capitolo 7

Trasformata della Distanza

La Trasformata della Distanza produce un'immagine in scala di grigio partendo da una immagine binaria. Questo perché il valore di grigio puó essere visto come la distanza che il pixel ha dallo sfondo.
Essa puó essere vista come una applicazione di morfologia matematica, in particolare come successione tra *Erosione* e *Somma*.
La trasformata della distanza puó essere 4-connessa oppure 8-connessa. Esistono diversi algoritmi per il calcolo della DT, quello **Sequenziale** prevede due scansioni:

- **Diretta** $= min(n_1, n_3) + 1$
- **Inversa** $= min(n_7, n_5, dp) + 1$

Dove :

$$\begin{bmatrix} n_2 & n_3 & n_4 \\ n_1 & pixel & n_5 \\ n_8 & n_7 & n_6 \end{bmatrix}$$

Inoltre l'algoritmo di **Media Pesata** risulta essere simile al metodo Sequenziale, ma aggiunge una certa media w:

$$w = \begin{bmatrix} 2 & 3 & 2 \\ 3 & 0 & 3 \\ 2 & 3 & 2 \end{bmatrix}$$

Algoritmo

1. Se necessario, converto l'immagine in B/N.
2. Dichiaro una *Mask [3x3]* contenente i pesi (Nel caso di algoritmo *Sequenziale* la inizializzo tutta a 1).
3. Scorro l'immagine ed applico la formula della *Scansione Diretta*.
4. Scorro l'immagine al contrario, da destra verso sinistra e dal basso all'alto ed applico la formula della *Scansione Inversa*.

Capitolo 8

Stima del Movimento

8.1 Sottrazione dello Sfondo

Con la sottrazione dello sfondo ho una stima del movimento ma non dello spostamento, ovvero non conosco di quanto un oggetto si é spostato e dove si sia spostato. Dunque esso viene utilizzato per stimare il modo globale del movimento. Semplicemente, il movimento si ottiene mediante la differenza tra il pixel x al tempo T e al tempo $T-1$. Se essa risulta essere diversa da 0 allora é avvenuto del movimento. Per *Sfondo* si intende le medie dell'immagine, ovvero, dopo aver effettuato la media dei valori dei pixel tra tutte le immagini, i pixel = 0 sono considerati sfondo.

Algoritmo

1. Prendo un set di immagini in input (Almeno 2) e ne calcolo la media ottenendo, quindi, uno sfondo.
2. Sottraggo lo sfondo al primo frame, ottenendo soltanto i pixel che rappresentano il movimento.

8.2 Flusso Ottico

!! DA AGGIUNGERE !!

Capitolo 9

Equalizzazione dell'Istogramma

Un istogramma é la rappresentazione grafica della quantitá di pixel presenti nell'immagine per ciascun livello di grigio. L'asse X rappresenta i *valori di grigio*, mentre quella delle T rappresenta il numero *totale* di pixel per quel determinato valore di grigio. Immagini diverse, peró, potrebbero avere lo stesso istogramma.
L'equalizzazione dell'istogramma é una tecnica che mira a modificare la forma dell'istogramma redistribuendo i valori di grigio in modo che esso sia quanto piú possibile uniforme. Il suo obiettivo é quello di migliorare le immagini a debole contrasto, distribuendo in modo uniforme i valori di grigio, in modo che il numero di pixel ad ogni livello di grigio sia il piú possibile costante.

Algoritmo

1. Disegno l'istogramma.
    - Scorro l'immagine incrementando il vettore *Occorrenze* di ogni pixel.
    - Creo una matrice *istogramma* avente come dimensione il numero di livelli di grigio (256) e il massimo numero di occorrenze per ogni livello.
    - Scorro la matrice *istogramma* colorando, nella colonna opportuna, un pixel per ogni occorrenza.
2. Equalizzo l'istogramma.
    - Dichiaro un array $S_K = \frac{(L-1)}{MN} \sum_{k=0}^{k} (n_k)$
      Questa formula rappresenta la *Formula di Equalizzazione*, dove:
      $n_k = istogramma[k]$
3. Applico l'istogramma equalizzato all'immagine originale.