Progetto MNI
“2D Interpolation for Edge Detection”

Giovanni Stea

July 1, 2024

Contents

1 Introduzione ed approccio al problema

Nel campo dell'elaborazione delle immagini, l'"**edge detection**" (rilevamento dei contorni) è una tecnica fondamentale per l'analisi e la comprensione di un'immagine, ad esempio per un agente intelligente che debba *muoversi autonomamente in uno spazio* riconoscendo ed evitando gli ostacoli.

Rilevare i contorni degli oggetti presenti in un'immagine consiste nell'identificare **discontinuità nell'intensità luminosa** dei pixel dell'immagine: queste discontinuità spesso corrispondono ai contorni degli oggetti che vediamo nell'immagine.

Al fine di intercettare le variazioni di intensità luminosa (che varia tra 0 e 1) dell'immagine è però necessario disporre di una **superficie bidimensionale** ottenuta *interpolando* i valori di luminosità dell'immagine, la quale è una collezione di pixel discreti.
Una volta ottenuta la *superficie interpolante* a partire dall'immagine si impiegano le *derivate* per individuare automaticamente i punti di maggiore variazione di intensità luminosa: in particolare si costruisce un'**approssimazione dell'immagine "gradiente"**, ottenuta a sua volta tramite un'approssimazione delle derivate parziali che fa uso delle *differenze finite*.

Si è realizzato a tale scopo un codice matlab, presente nel file `EdgeDetection.m`, che implementa una classe la quale incapsula e comprende *tutte le routines* necessarie al processing dell'immagine, dall'interpolazione della stessa fino alla sua binarizzazione.

Di seguito se ne descrivono brevemente i metodi, utilizzati poi nel file `main.m`.

- `EdgeDetection(img)`: costruttore della classe che imposta l'attributo `obj.Img` e gli attributi `obj.xg` e `obj.yg` che corrispondono alle coordinate dei pixel dell'immagine nelle direzioni x e y rispettivamente.

- `interpolate(obj, xx, yy, method)`: incapsula sia la routine `griddedInterpolant` fornita dal matlab, che la routine `lagrange2d`, da me implementata, selezionabile impostando il metodo `'lagrange'`; i parametri `xx` e `yy` corrispondono ai punti di query.

- `lagrange2d(obj, xx, yy)`: calcola una *superficie interpolante quadratica* con l'uso del polinomio di Lagrange; i parametri `xx` e `yy` corrispondono ai punti di query.
  Il metodo contiene altre due funzioni di lavoro:

  - `find_grid3x3(x, y)` che trova 3 punti nella direzione x e 3 nella direzione y che disegnano una griglia che circondi il punto di interpolazione.
  - `evaluate(x, y)` che valuta la superficie interpolante nel punto dato.

- `gradient(obj, method, order, h)`: calcola l'*immagine "gradiente"* sull'immagine interpolata con la routine `interpolate` chiamata passando il metodo di interpolazione scelto; è inoltre possibile specificare l'ordine delle differenze finite da utilizzare (`'#1'` o `'#2'`), e il passo di discretizzazione `h`.

- `binarize(obj, grad, thresh)`: binarizza l'immagine "gradiente" `grad` passata, secondo la soglia `thresh` che viene specificata.

2 Interpolazione 2D

L'**interpolazione 2D** è una tecnica numerica utilizzata per *stimare i valori di una funzione ignota* $f(x, y)$ in punti non campionati, partendo da un insieme di dati noti nella forma di triple $(x_i, y_i, f(x_i, y_i))$, i quali possono essere distribuiti secondo una certa struttura "a griglia" (*gridded*), oppure no, ed essere "sparsi".

In questo caso di studio si sono applicati metodi di interpolazione 2D di tipo *gridded*, poiché si necessitava, partendo dai valori di colore (in scala di grigi) $f(x_i, y_i)$ di un'immagine per ogni pixel in posizione (x_i, y_i), di valutare la luminosità dell'immagine anche in punti arbitrari del tipo $(x_i + h, y_i)$ o $(x_i, y_i + h)$, con h diverso da un multiplo di 1.

I metodi di interpolazione 2D impiegati per ottenere una **superficie interpolante** sono stati quelli forniti dalla routine `griddedInterpolant` di matlab (in particolare il metodo *lineare* e *Akima modificato*), confrontati con un metodo implementato da zero nella funzione `lagrange2d` che utilizza il *polinomio di Lagrange* di grado 2.

L'**interpolazione di Lagrange** è un tipo di interpolazione polinomiale che, dati $n + 1$ punti $(x_i, f(x_i))$, costruisce il polinomio interpolante $P_n(x)$ come segue:

$$P_n(x) = \sum_{j=0}^{n} f(x_j)\ell_j(x) \qquad \text{con } \ell_j(x) = \prod_{i=1, i\neq j}^{n} \frac{(x - x_i)}{(x_j - x_i)}$$

Nel caso di studio si è implementata l'interpolazione di Lagrange *bidimensionale quadratica*, o **interpolazione biquadratica**, applicata a dati "gridded", ovvero i pixel di un'immagine.
Di seguito è illustrato il procedimento implementato dal metodo `lagrange2d`.

Dato un punto di interpolazione (x, y), il metodo usa la funzione `find_grid3x3` per trovare i 3 indici $[\, x_0, x_1, x_2 \,]$ e i 3 indici $[\, y_0, y_1, y_2 \,]$ che disegnano un *quadrato che circonda* il punto di interpolazione. Una volta ottenuto ciò si procede alla valutazione del punto (x, y) come segue.

Viene calcolata $f(x, y_i)$ per ciascun y_i: ovvero per $i = 0, 1, 2$

$$f(x, y_i) =$$
$$f(x_0, y_i)\frac{(x - x_1)}{(x_0 - x_1)}\frac{(x - x_2)}{(x_0 - x_2)} + f(x_1, y_i)\frac{(x - x_0)}{(x_1 - x_0)}\frac{(x - x_2)}{(x_1 - x_2)} + f(x_2, y_i)\frac{(x - x_0)}{(x_2 - x_0)}\frac{(x - x_1)}{(x_2 - x_1)}$$

Una volta calcolati $[\, f(x, y_0), f(x, y_1), f(x, y_2) \,]$ viene calcolato il *valore finale dell'interpolante*:

$$f(x, y) =$$
$$f(x, y_0)\frac{(y - y_1)}{(y_0 - y_1)}\frac{(y - y_2)}{(y_0 - y_2)} + f(x, y_1)\frac{(y - y_0)}{(y_1 - y_0)}\frac{(y - y_2)}{(y_1 - y_2)} + f(x, y_2)\frac{(y - y_0)}{(y_2 - y_0)}\frac{(y - y_1)}{(y_2 - y_1)}$$

Questo procedimento è racchiuso nella funzione `evaluate`, a sua volta utilizzata dal metodo `lagrange2d`, il quale richiama la prima per ogni punto di interpolazione.

Il confronto con i metodi di interpolazione implementati da `griddedInterpolant` verrà fatto quando si commenterà il loro utilizzo per il *calcolo dell'immagine "gradiente"*.

3 Calcolo dell'immagine "gradiente"

Una volta definito il procedimento per ottenere la superficie interpolante a partire dall'immagine iniziale, veniamo all'effettiva *individuazione dei bordi* degli oggetti presenti.

A tal fine analizziamo l'immagine in scala di grigi, nella quale, per ogni pixel, l'*intensità luminosa* varia tra 0 e 1 : se consideriamo l'immagine come una **superficie bidimensionale** $f(x, y)$, allora possiamo cercare di individuare i punti dell'immagine nei quali l'intensità luminosa *varia in misura maggiore*, cioè quei punti che probabilmente corrispondono al bordo di un oggetto.

Per intercettare i punti con maggiore variazione di luminosità utilizziamo il concetto di derivata: in particolare andiamo ad approssimare la così detta **immagine "gradiente"** ∇f. A questo scopo necessitiamo di un'*approssimazione delle derivate parziali* rispetto ad x e y per ogni (x, y) corrispondenti alle coordinate di un pixel dell'immagine f.

Nel metodo `gradient` sono stati implementati i metodi delle **differenze finite** (sia in avanti che all'indietro) per approssimare l'immagine "gradiente"; tramite il parametro `order` è possibile specificare l'ordine delle differenze finite da utilizzare (`'#1'` o `'#2'`).
Nel caso di studio è stato utilizzato l'ordine 2 in quanto permetteva una migliore approssimazione, la quale ha condotto a *migliori risultati* nell'individuazione dei bordi.

Le differenze finite *di ordine 1* (in avanti) sono calcolate come segue:

$$\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \approx \frac{f(x+h, y) - 2f(x, y) + f(x, y+h)}{h}$$

Mentre quelle *di ordine 2* (in avanti) sono calcolate come segue:

$$\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \approx \frac{4f(x+h, y) - f(x+2h, y) - 3f(x, y)}{2h} + \frac{4f(x, y+h) - f(x, y+2h) - 3f(x, y)}{2h}$$

Di seguito si mostra come, con lo stesso passo h (0.9), variasse l'approssimazione dell'immagine "gradiente" ottenuta usando le differenze finite di ordine 1 (a sinistra) e 2 (a destra).

Nella formula h rappresenta il **passo di discretizzazione**, il quale è stato fatto variare sperimentalmente tra 0.1 e 2.0, osservando poi gli effetti sull'approssimazione ottenuta.

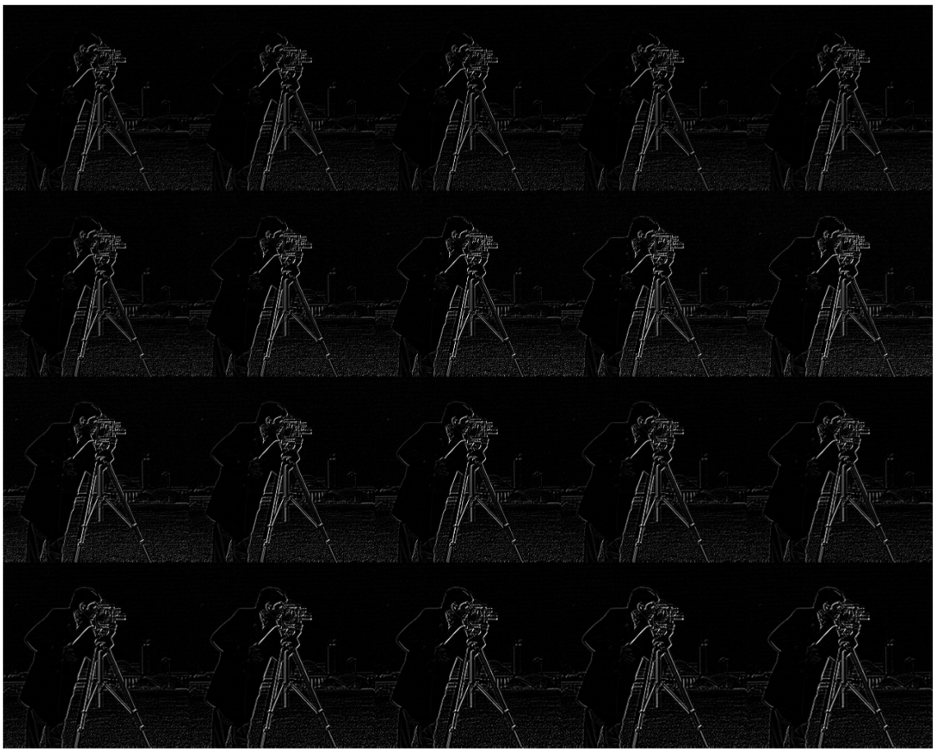
Nel calcolo delle differenze finite per ottenere l'immagine "gradiente" è fondamentale disporre di una *superficie interpolante dell'immagine iniziale* di modo da poter valutare f anche in punti (x, y) che non corrispondono alle coordinate di pixel dell'immagine.

Nel caso di studio si sono sperimentati tre tipi di costruzione dell'interpolante f:

- **interpolante lineare**, implementata dal metodo `'linear'` di `griddedInterpolant`
- **interpolante cubica**, implementata dal metodo `'makima'` di `griddedInterpolant`
- **interpolante quadratica**, di cui si è fornita una *propria implementazione* nel metodo `lagrange2d`, o selezionabile chiamando `interpolate(xx, yy, 'lagrange')`

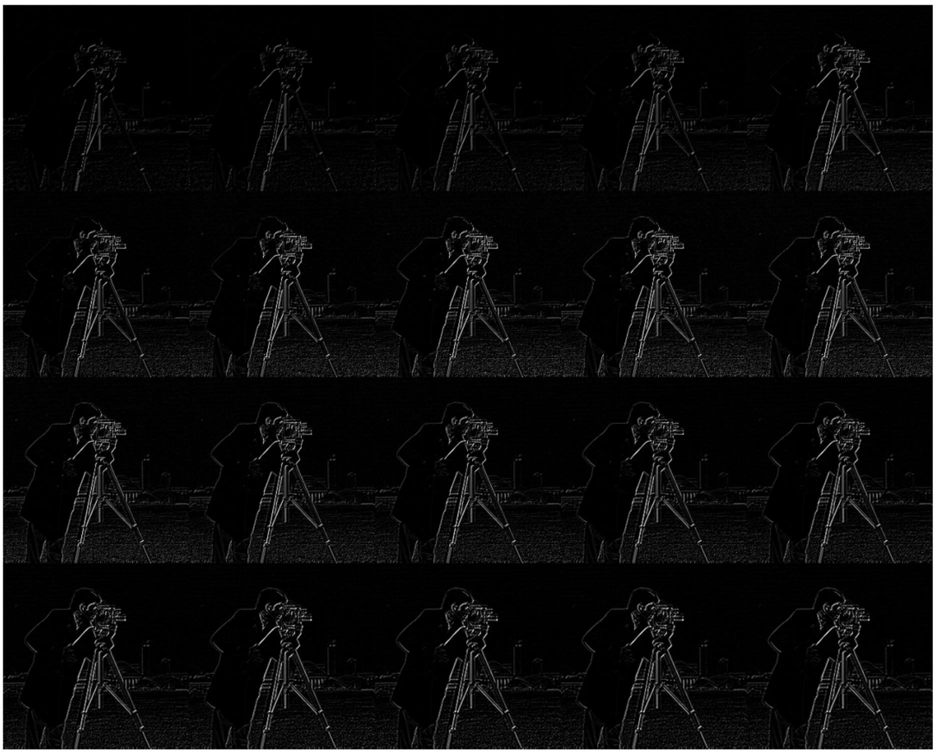
Di seguito si mostra un confronto tra i metodi di interpolazione considerati, nello specifico si mostrano i risultati al variare del passo di discretizzazione h per ciascuno dei tre metodi.

- Immagine "gradiente" (con `'linear'`) al variare di h:



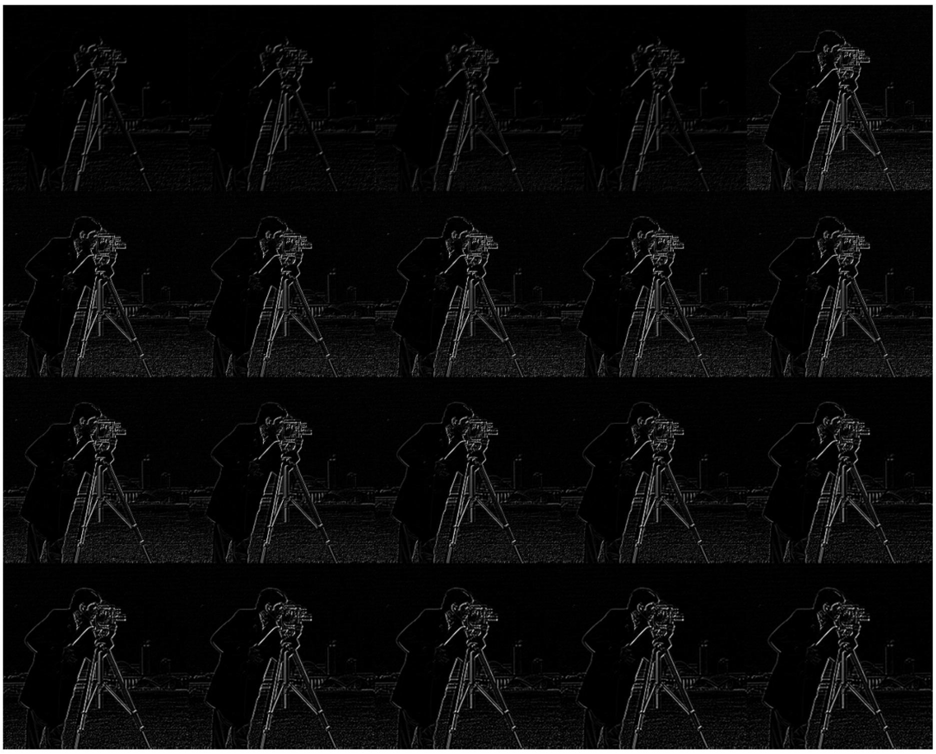
Si nota come a partire da un passo h pari a $0.7 - 0.8$ i bordi del soggetto nell'immagine cominciano ad essere individuati in modo completo. Si è scelto un passo $h = 1.8$ in quanto si otteneva una migliore definizione del soggetto senza che fosse evidenziata erroneamente parte del background.

- Immagine "gradiente" (con `'makima'`) al variare di h:



Si nota come, in questo caso, per h minore di $0.4 - 0.5$ si ha addirittura una peggiore approssimazione rispetto all'interpolante lineare con lo stesso passo, mentre a partire da 0.8 l'approssimazione migliora. Si è scelto anche qui un passo h vicino a 2 ($h = 1.9$) poiché si otteneva una migliore definizione del soggetto rispetto al background.

- Immagine "gradiente" (con `'lagrange'`) al variare di h:



In quest'ultimo caso vediamo come l'approssimazione migliori drasticamente a partire da $h = 0.5$ per poi crescere nel dettaglio in modo molto simile all'interpolante cubica con `'makima'`. Come per i precedenti due interpolanti, si è scelto un passo h vicino a 2, ovvero $h = 2.0$, per gli stessi motivi già citati.

4 Binarizzazione dell'immagine "gradiente"

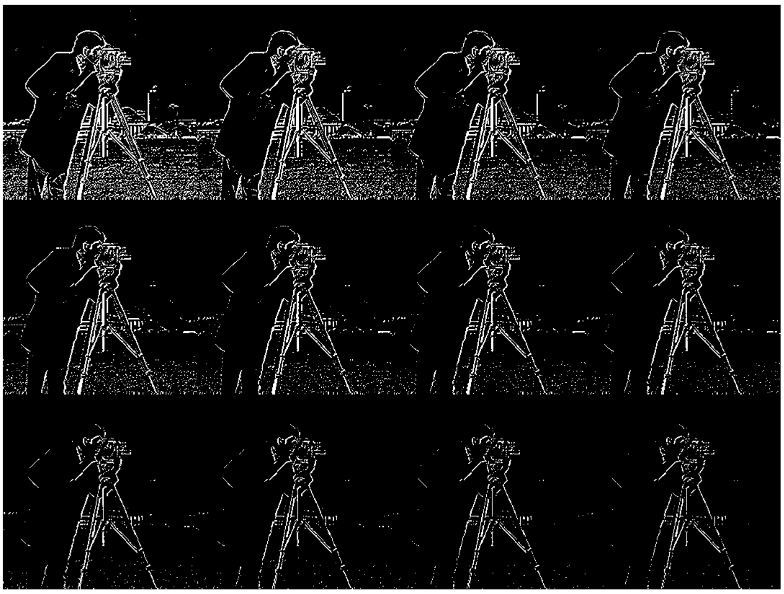
Una volta ottenuta l'*immagine "gradiente"* è stata in ultimo applicata una **binarizzazione** dell'immagine risultante, così da *rendere più evidenti i bordi* di soggetti/oggetti individuati.

In particolare i valori di ogni pixel dell'immagine "gradiente" ∇f sono stati prima scalati in modo da variare tra 0 e 1, e successivamente si è variato su un certo intervallo di *valori soglia* τ tra 0.5 e 0.6, modificando l'immagine come segue:

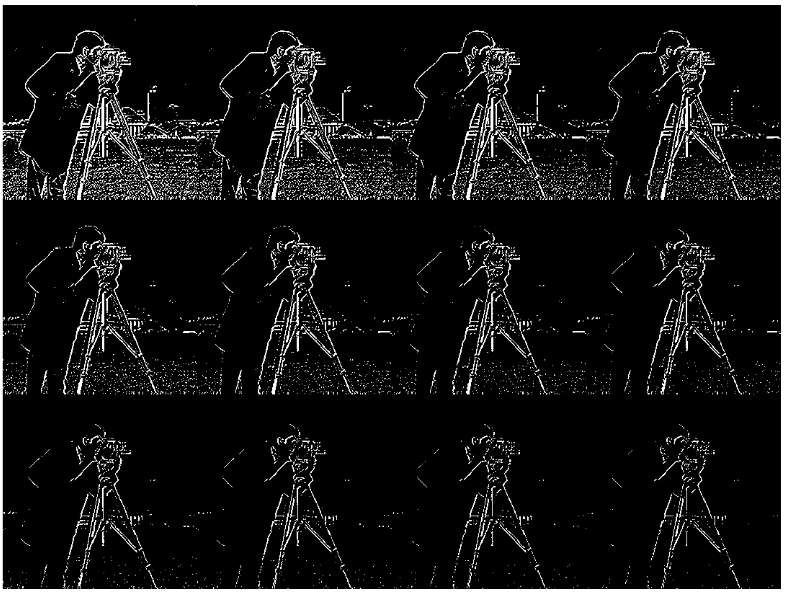
$$\nabla f(x_i, y_j) = \begin{cases} 1 & \text{se } \nabla f(x_i, y_j) \geq \tau \\ 0 & \text{se } \nabla f(x_i, y_j) < \tau \end{cases} \qquad \text{per } i = 1, ..., m \text{ e per } j = 1, ..., n$$

Di seguito si mostra la successione di immagini binarizzate (a partire dall'immagine "gradiente" calcolata con il passo h scelto in precedenza) per ogni metodo di interpolazione utilizzato.

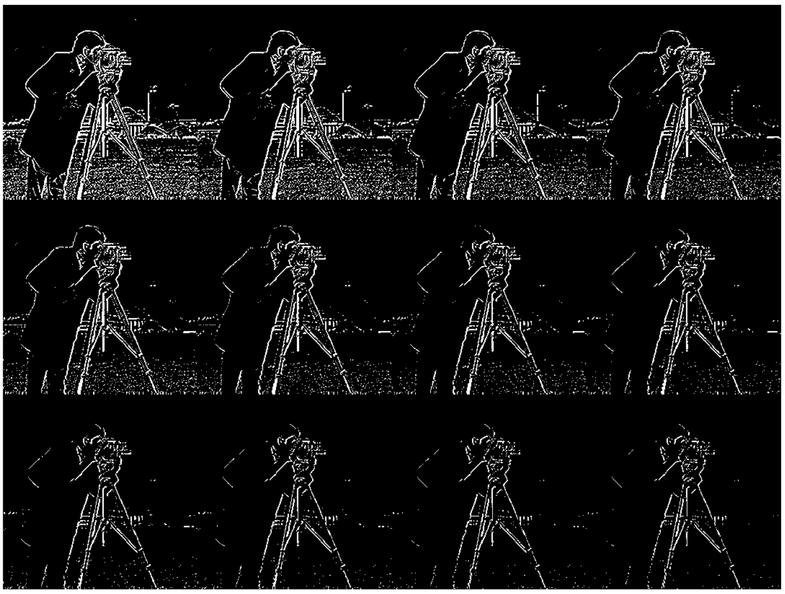
- Immagine binarizzata (con `'linear'` e $h = 1.8$) al variare di τ



- Immagine binarizzata (con `'makima'` e $h = 1.9$) al variare di τ



- Immagine binarizzata (con `'lagrange'` e $h = 2.0$) al variare di τ



In tutti e tre i casi si è scelto τ in modo che preservasse al meglio i bordi senza però catturare troppo rumore dal background: in particolare si è scelto $\tau = 0.52625$ nel primo caso, mentre $\tau = 0.5175$ per gli altri due.

5 Conclusioni

In conclusione si può dire di aver ottenuto dei buoni risultati di *edge-detection* in tutti e tre i casi, con un leggero miglioramento dell'approssimazione ottenuto tramite interpolanti di grado maggiore di 1 (`'makima'` di grado 3 e `'lagrange'` di grado 2).

L'*interpolante quadratica*, di cui si è fornita una propria implementazione nel metodo `lagrange2d`, si è inoltre dimostrata all'altezza dell'approssimazione fornita dall'interpolante cubica (nella versione "Akima modificata") fornita da matlab.

Di seguito si mostrano i risultati finali.