



Università degli studi di Cagliari

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in Ingegneria Elettrica ed Elettronica

**RILEVAMENTO DI OGGETTI IN MOVIMENTO
ATTRAVERSO METODI BASATI SULL'OPTICAL FLOW**

Tesi di Laurea

**Relatore:
Prof. Giuseppe Rodriguez**

**Candidato:
Mirko Sanna**

Anno Accademico 2016/2017

“Dedicata a tutte le persone che
mi hanno sostenuto durante
questo percorso.”

Indice

Introduzione	6
Capitolo 1: Elementi di matematica applicata	8
1.1 Problemi mal posti o ben posti e condizionamento	8
1.2 Definizione di algoritmo	9
1.3 Problemi ai minimi quadrati	9
1.4 Sviluppo in serie di Taylor	16
1.5 Trasformata di Fourier	18
1.6 Interpolazione	19
Capitolo 2: Tecniche per la stima del moto denso	23
2.1 Translational alignment	24
2.2 Stima gerarchica del moto	27
2.3 Allineamento basato su Fourier	28
2.4 Spline-based motion	30
2.5 Spostamento relativo ed Optical Flow	34
2.6 Il problema dell'apertura ed il gradiente di flusso	38
Capitolo 3: Algoritmi per la stima del moto con le tecniche di Optical Flow	42
3.1 Algoritmo di Horn-Schunck	42
3.2 Algoritmo di Lucas-Kanade	49

3.3 Algoritmo BBPW	54
Capitolo 4: Valutazione delle performance ed applicazioni in Matlab	60
4.1 Strategie dei test	60
4.2 Errori di misura con la realtà di base disponibile	62
4.3 Tracciamento delle auto utilizzando il flusso ottico	64
4.4 Algoritmo di Horn-Schunck su Matlab	66
Bibliografia	70

Introduzione

In questo lavoro di tesi si affronteranno le tematiche sulla stima del moto di un oggetto attraverso delle tecniche basate sull'optical flow (flusso ottico). Nella prima parte, dopo una breve introduzione di alcuni elementi fondamentali di matematica applicata, come ad esempio la trasformata di *Fourier* o l'*interpolazione*, si illustreranno svariate tecniche che si utilizzano nella *Computer Vision* per avere una stima del moto; in alcune di esse verranno anche indicate delle applicazioni tipiche. Nella seconda parte invece, l'attenzione si concentrerà esclusivamente sulle tecniche e gli algoritmi basati sull'optical flow. L'optical flow ha origine dalla psicologia sperimentale della visione, *James Jerome Gibson*, fu uno dei primi psicologi a studiarne gli effetti. I suoi studi furono effettuati per conto dell'aeronautica militare degli USA, infatti utilizza l'esempio del pilota che gestisce l'atterraggio di un aereo con successo. In questo suo trattato *Gibson* scoprì che le informazioni necessarie per la corretta manovra di atterraggio venivano fornite dalle *configurazioni di flusso ottico*. All'interno di queste configurazioni il punto verso il quale l'aereo si dirigeva risultava immobile agli occhi del pilota, mentre tutti gli altri punti sembravano allontanarsi da esso. Inoltre, egli ha notato come il punto immobile del flusso ottico cambiasse in conseguenza al cambio di direzione dell'aereo. *Gibson* arrivò dunque alla conclusione che le configurazioni di flusso ottico forniscono informazioni dettagliate sulla direzione dell'aereo, sulla distanza dalla pista e sulla sua velocità. Su queste conclusioni la *Computer Vision* utilizza i suoi strumenti per poter avere delle stime quanto più attendibili possibile del moto, anche attraverso video o immagini. Infine, nell'ultimo capitolo verranno trattati alcuni algoritmi applicati al software Matlab ed i relativi risultati.

Capitolo 1

Elementi di matematica applicata.

Prima di iniziare a parlare di Computer Vision e di tecniche per la stima del moto è opportuno introdurre alcuni concetti che verranno ripresi nei capitoli successivi. Lo scopo di questo capitolo è quello di fornire gli strumenti necessari per la comprensione dei capitoli seguenti. Di seguito si useranno le seguenti notazioni:

- \mathbf{u} = vettore;
- \mathbf{U} = matrice;
- u = scalare;
- $\mathbf{u} \cdot \mathbf{v}$ = prodotto scalare;
- $\mathbf{u} \times \mathbf{v}$ = prodotto vettoriale;
- $*$ = operatore di convoluzione;

1.1 Problemi mal posti o ben posti e condizionamento

Quando si vuol studiare un fenomeno, il metodo più utilizzato è quello di costruirne il *modello matematico*, ovvero un insieme di formule che ne descrivono il comportamento. La creazione del metodo non è per nulla banale, richiede infatti un'ottima conoscenza della fisica e della matematica che concerne il fenomeno. Le formule così ottenute però risultano spesso troppo complicate per la risoluzione classica, in questi casi intervengono i metodi numerici, ovvero metodi che, accettando qualche semplificazione o approssimazione accettabile (ad esempio un errore in percentuale tollerato), rendono il metodo risolvibile numericamente con l'utilizzo di un computer.

Posizionamento: nei metodi numerici gioca un ruolo fondamentale il *posizionamento*. Un problema si dice *ben posto* se esso possiede, in un prefissato campo di definizione, una ed una sola soluzione, e questa dipende con continuità dai dati. Nel caso ciò non avvenga il problema si dice *mal posto*. Degli esempi di *problemi mal posti* sono i sistemi con matrici rettangolari, infatti la soluzione potrebbe o non esistere, oppure non essere unica.

Condizionamento: un'altra caratteristica fondamentale di un modello matematico, analizzato con metodi numerici, è il condizionamento, ovvero quantifica quanto l'errore sui dati abbia ripercussioni sui risultati. Sia δd una perturbazione dei dati nel problema,

espressi mediante un vettore \mathbf{d} , sia invece δs la perturbazione che affligge la sua soluzione s . Il numero di *condizionamento assoluto* $K=K(\mathbf{d})$ è definito come:

$$\|\delta s\| \leq K \|\delta \mathbf{d}\| \quad (1)$$

Invece il numero di *condizionamento relativo* $k=k(\mathbf{d})$ è:

$$\frac{\|\delta s\|}{\|s\|} \leq k \frac{\|\delta \mathbf{d}\|}{\|\mathbf{d}\|} \quad (2)$$

dove con $\|\mathbf{x}\|$ si è intesa una qualsiasi norma vettoriale. Da queste formule si evince che il problema di un elevato condizionamento non dipende dal metodo utilizzato per la risoluzione del problema.

1.2 Definizione di algoritmo

Un'altra definizione che sarà utile più avanti, soprattutto nel capitolo 3, è la definizione di *algoritmo*. Un algoritmo viene definito come una sequenza univoca di un numero finito di operazioni elementari che stabilisce come valutare la soluzione di un problema una volta assegnati dei valori iniziali. Negli algoritmi si usa una sorta di linguaggio di programmazione per esprimerli. Molti algoritmi adottano infatti strutture di controllo del flusso comuni a più linguaggi (dal linguaggio C alla programmazione su Matlab), come ad esempio 'if', 'for', 'while' e simili. Un algoritmo si dice *stabile* quando la successione delle operazioni non amplifica eccessivamente gli errori presenti sui dati, in caso contrario viene definito *instabile*. Un'altra caratteristica importante è la *complessità computazionale*, ovvero il numero delle operazioni in virgola mobile necessarie per risolvere un problema mediante l'algoritmo dato (questa definizione è la definizione utilizzata in analisi numerica, differisce infatti con quella utilizzata tipicamente in informatica).

1.3 Problemi ai minimi quadrati

I problemi ai minimi quadrati nascono nelle varie applicazioni pratiche dove si riscontrano sistemi lineari (nella forma $\mathbf{Ax}=\mathbf{b}$, dove \mathbf{A} è la matrice dei coefficienti, \mathbf{x} il vettore delle incognite e \mathbf{b} il vettore dei termini noti) in cui la matrice dei coefficienti \mathbf{A} non è quadrata bensì rettangolare.

$$\begin{cases} x + y + z = 2 \\ 2x - 5y = 3 \\ 3y + 7z = 1 \end{cases} \rightarrow \mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 2 & -5 & 0 \\ 0 & 3 & 7 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} \quad \text{\textit{Sistema lineare determinato}}$$

$$\begin{cases} x + y + z = 3 \\ 3x - 7y = 9 \end{cases} \rightarrow \mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 3 & -7 & 0 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 3 \\ 9 \end{bmatrix} \quad \text{\textit{Sistema lineare con A rettangolare}}$$

Mentre nel primo caso si ha una, ed una sola, soluzione, se il determinante di $\mathbf{A} \neq 0$, come nell'esempio, nel secondo caso le soluzioni dipendono dalle dimensioni di \mathbf{A} . Se \mathbf{A} è rettangolare non si può esprimere il sistema come $\mathbf{Ax}=\mathbf{b}$, perciò non si userà $\|\mathbf{Ax} - \mathbf{b}\|$, e si minimizzerà la norma, il problema vale per qualunque norma, tipicamente si usa la norma 2. Quindi il problema ai minimi quadrati nasce da:

$$\min_{x \in \mathbb{R}^n} \|\mathbf{Ax} - \mathbf{b}\|_2$$

Ipotizzando \mathbf{A} di rango pieno, quindi con determinante della matrice di dimensione più grande $\neq 0$, si dividono i casi in 2. Un caso dove matrice \mathbf{A} del tipo $m \times n$, con m numero di righe ed n di colonne, in cui $m > n$ e l'altro dove $m < n$.

Quando $m > n$ si avranno più equazioni che incognite, quindi si otterrà un sistema che è detto sovradeterminato. In queste condizioni si potrebbe non avere la soluzione del sistema, ad esempio:

$$\begin{cases} x - y = 2 \\ x + y = 3 \\ x - 3y = 2 \end{cases}$$

Questo caso però è tipico nelle applicazioni in cui si fanno molte misurazioni, poiché essendo i dati affetti da errore è conveniente fare più misurazioni per vedere in che intervallo è più probabile che cada il valore effettivo della misurazione. Un esempio può essere la misura di una tensione ai capi di una resistenza.

Quando $m < n$ il sistema avrà più incognite che equazioni, nella geometria classica si hanno ∞^{n-m} soluzioni, ed il sistema è detto sottodeterminato. Con $m < n$ il sistema avrà sempre soluzione.

$$\begin{cases} x - 3y + 2z = 5 \\ 8x + y + 3z = 4 \end{cases}$$

Questo caso è tipico nelle applicazioni dove non è possibile fare troppe misurazioni, per vari fattori: costo, sicurezza, tempo, ecc.

La differenza sostanziale tra le due tipologie è il modo di affrontarle, entrambe danno vita a dei problemi ai minimi quadrati, ma mentre nel caso $m > n$ si ha un problema ai minimi quadrati standard, dove devo solo minimizzare la norma $\|Ax = b\|$, nel caso di $m < n$ invece mi serviranno informazioni aggiuntive, ad esempio Matlab usa l'approccio con il vettore x con più 0 possibili; un'altra soluzione è invece quella di minimizzare anche la norma di x , ottenendo così un problema ai minimi quadrati alla minima norma.

1.3.1 Metodi risolutivi per i problemi ai minimi quadrati

Metodo delle equazioni normali: esistono vari metodi per la risoluzione dei problemi ai minimi quadrati, tra i più utilizzati c'è il metodo che opera risolvendo il sistema normale associato al problema.

Volendo minimizzare una quantità non negativa, è consigliabile considerare la norma quadra:

$$\|Ax = b\|_2^2 = (Ax - b)^T (Ax - b) = x^T A^T Ax - x^T A^T b - b^T Ax + b^T b \quad (3)$$

Sommando i termini simili ($x^T A^T b = b^T Ax$) si ottiene infine:

$$\|Ax = b\|_2^2 = x^T A^T Ax - 2x^T A^T b + b^T b \quad (4)$$

Per minimizzare la norma bisogna imporre che la derivata prima sia nulla, in questo caso non si ha solo un'incognita, bisogna dunque usare il gradiente, che è definito come:

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right)^T \quad (5)$$

Delle proprietà utili del gradiente sono:

- $\nabla(\mathbf{b}^T \mathbf{x}) = \mathbf{b}$, dove $f(\mathbf{x}) = (\mathbf{b}^T \mathbf{x})$, con $\mathbf{x}, \mathbf{b} \in R^n$
- $\nabla(\mathbf{x}^T \mathbf{A} \mathbf{x}) = \mathbf{A} \mathbf{x} + \mathbf{A}^T \mathbf{x}$, dove \mathbf{A} è una matrice quadrata $n \times n$
- $\nabla(\mathbf{x}^T \mathbf{A} \mathbf{x}) = 2\mathbf{A} \mathbf{x}$, se \mathbf{A} oltre ad essere quadrata è anche simmetrica

Dunque tornando alla minimizzazione si può scrivere:

$$\frac{1}{2} \nabla (\|Ax = b\|_2^2) = A^T Ax - A^T b = 0 \quad (6)$$

Si giunge così al sistema normale:

$$A^T Ax = A^T b \quad (7)$$

Se A è a rango pieno, la matrice data dalla sua trasposta per se stessa risulta invertibile, ed il sistema ha una soluzione unica data da:

$$x = (A^T A)^{-1} A^T b \quad (8)$$

Se A invece non ha rango pieno, $A^T A$ risulta singolare, dunque non è possibile invertirla, il sistema però rimane consistente, poiché il vettore $A^T b$ è appartenente all'immagine di A^T , che coincide con l'immagine di $A^T A$. Tra le infinite soluzioni, in questo caso, si adotta la soluzione alla minima norma euclidea.

La matrice $A^+ = (A^T A)^{-1} A^T$ è detta pseudo-inversa, in quanto è un'inversa sinistra di A , ma non destra.

Se la matrice A è a rango pieno allora la sua pseudo-inversa risulta definita positiva ed è possibile risolvere il sistema normale con la fattorizzazione di Cholesky¹, che ha un basso costo computazionale.

Una volta applicato Cholesky si può calcolare la soluzione del sistema come la soluzione di due sistemi triangolari (quindi di soluzione semplice):

$$\begin{cases} R^T y = A^T b \\ R x = y \end{cases} \quad (9)$$

Un lato negativo di questo approccio è che la matrice $A^T A$ ha un numero di condizionamento pari al quadrato della matrice A , cosa che potrebbe minarne la stabilità.

Il metodo della fattorizzazione QR: la fattorizzazione QR consiste nello scrivere la matrice A come il prodotto di due matrici: una matrice ortogonale Q , ed una matrice triangolare superiore R . La particolarità della matrice ortogonale è che la sua inversa coincide con la trasposta, inoltre Q è trasparente alla norma euclidea, infatti se viene moltiplicata per un vettore non ne altera la norma.

¹ La fattorizzazione di Cholesky è un tipo di fattorizzazione $A=LU$, nel suo caso però è $A=R^T R$, dove R è una matrice triangolare superiore.

I due algoritmi più utilizzati per calcolare questa fattorizzazione sono: l'algoritmo di *Givens* e l'algoritmo di *Householder*; il primo viene preferito quando la matrice \mathbf{A} ha molti zeri ed è in forma quasi triangolare. Si basa sulla matrice di *Givens*, detta \mathbf{G} . Essa fa compiere una rotazione piana che lascia invariato il sottospazio ortogonale al piano definito dai vettori e_i, e_j . Il secondo invece è il più utilizzato nella maggior parte dei casi, pertanto si vedrà nel dettaglio.

Il metodo di *Householder* si fonda sull'usare delle matrici di *Householder* per la costruzione della matrice ortogonale \mathbf{Q} , una matrice di *Householder* è formata da:

$$\mathbf{H} = \mathbf{I} - 2\mathbf{w}\mathbf{w}^T \quad (10)$$

con $\mathbf{w} \in \mathbb{R}^n$, $\|\mathbf{w}\|=1$ (norma euclidea).

$2\mathbf{w}\mathbf{w}^T$ rappresenta una matrice $n \times n$ di rango 1, la matrice \mathbf{H} è simmetrica ed ortogonale.

Inserendo un vettore \mathbf{x} , determiniamo \mathbf{w} in modo che:

$$\mathbf{H}\mathbf{x} = k\mathbf{e}_1 \quad (11)$$

dove e_i sta ad indicare l' i -esimo vettore della base canonica \mathbb{R}^n e $k \in \mathbb{R}$.

La costruzione di \mathbf{H} è possibile dividerla in 3 fasi:

1. Dato che \mathbf{H} è ortogonale :

$$\|\mathbf{H}\mathbf{x}\| = \|\mathbf{x}\| = |k| \quad (12)$$

da cui ricaviamo $k = \pm\sigma$, con $\sigma = \|\mathbf{x}\|$.

2. Per la definizione di \mathbf{H} , si ottiene:

$$\mathbf{H}\mathbf{x} = \mathbf{x} - 2\mathbf{w}\mathbf{w}^T\mathbf{x} = \mathbf{x} - 2(\mathbf{w}^T\mathbf{x})\mathbf{w} = k\mathbf{e}_1 \quad (13)$$

che implica:

$$\mathbf{w} = \frac{\mathbf{x} - k\mathbf{e}_1}{2\mathbf{w}^T\mathbf{x}} = \frac{\mathbf{x} - k\mathbf{e}_1}{\|\mathbf{x} - k\mathbf{e}_1\|} \quad (14)$$

in quanto \mathbf{w} ha norma unitaria.

Con questa fase si è già determinato \mathbf{w} , la terza fase serve per fare delle considerazioni numeriche.

3. Usando la relazione $\mathbf{x}^T\mathbf{H}\mathbf{x} = \mathbf{x}^T\mathbf{x} - 2(\mathbf{w}^T\mathbf{x})^2 = kx_1$ si ottiene:

$$(\mathbf{w}^T\mathbf{x})^2 = \frac{\sigma^2 - kx_1}{2} \quad (15)$$

ed utilizzando l'espressione per \mathbf{w} :

$$\|\mathbf{x} - k\mathbf{e}_1\| = 2\mathbf{w}^T\mathbf{x} = \sqrt{2(\sigma^2 - kx_1)} \quad (16)$$

Con questa operazione si riduce il calcolo computazionale sulla normalizzazione di \mathbf{w} . Per evitare problemi di cancellazione si sceglie il segno di k , in modo che $-kx_1$ sia positivo.

Una volta costruite le matrici \mathbf{H} è possibile costruire la fattorizzazione **QR** di \mathbf{A} . Ipotizzando che \mathbf{A} sia quadrata (non è necessario, in quanto la fattorizzazione funziona anche su matrici rettangolari, serve solamente a rendere più semplici le spiegazioni), si può generare una successione di matrici $\mathbf{A}^{(i)}$, con i che varia da 1 alla dimensione della matrice, in modo che $\mathbf{A}^{(n)}$ sia una matrice triangolare superiore.

Scrivendo la matrice \mathbf{A} in termini delle sue colonne:

$$\mathbf{A} = \mathbf{A}^{(1)} = \left[\mathbf{a}_1^{(1)} \quad \mathbf{a}_2^{(1)} \quad \dots \quad \mathbf{a}_n^{(1)} \right] \quad (17)$$

Seguendo i 3 passi scritti prima, si crea una matrice elementare di *Householder* tale che:

$$\mathbf{H}_1 \mathbf{a}_1^{(1)} = k_1 \mathbf{e}_1 \quad (18)$$

Applicando questa matrice alla sinistra della matrice $\mathbf{A}^{(1)}$, si genera la seconda matrice della successione:

$$\mathbf{A}^{(2)} = \mathbf{H}_1 \mathbf{A}^{(1)} = \left[\mathbf{a}_1^{(2)} \quad \mathbf{a}_2^{(2)} \quad \dots \quad \mathbf{a}_n^{(2)} \right] \quad (18)$$

dove:

$$\mathbf{a}_1^{(2)} = k_1 \mathbf{e}_1 \quad (19)$$

$$\mathbf{a}_j^{(2)} = \mathbf{H}_1 \mathbf{a}_j^{(1)} \quad (20)$$

con $j=2, \dots, n$.

La matrice $\mathbf{A}^{(2)}$ appena generata ha la struttura seguente:

$$\mathbf{A}^{(2)} = \begin{bmatrix} k_1 & \mathbf{a}_{12}^{(2)} & \dots & \mathbf{a}_{1n}^{(2)} \\ 0 & \mathbf{a}_{22}^{(2)} & \dots & \mathbf{a}_{2n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \mathbf{a}_{n2}^{(2)} & \dots & \mathbf{a}_{nn}^{(2)} \end{bmatrix} = \begin{bmatrix} k_1 & \mathbf{v}_1^T \\ \mathbf{0} & \widehat{\mathbf{A}}^{(2)} \end{bmatrix} \quad (21)$$

con $\mathbf{0}$ vettore colonna di dimensioni tali che le dimensioni della matrice siano rispettate, con componenti tutte nulle, ed $\widehat{\mathbf{A}}^{(2)}$ una sottomatrice di dimensione $n-1$.

Dopo di che si considera la sottomatrice ricavata $\widehat{\mathbf{A}}^{(2)}$:

$$\widehat{\mathbf{A}}^{(2)} = \left[\widehat{\mathbf{a}}_2^{(2)} \widehat{\mathbf{a}}_3^{(2)} \dots \widehat{\mathbf{a}}_n^{(2)} \right] \quad (22)$$

Si costruirà quindi la matrice elementare di *Householder* di dimensione n-1 in modo che:

$\widehat{\mathbf{H}}_2 \widehat{\mathbf{a}}_2^{(2)} = k_2 \mathbf{e}_1$, in questo caso \mathbf{e}_1 ha le dimensioni adeguate a fare il prodotto tra vettori, quindi ha lunghezza n-1.

Dopo questo passaggio si aggiunge una riga ed una colonna alla matrice $\widehat{\mathbf{H}}_2$, la riga e la colonna sono le prime della matrice identità, affinché raggiunga la dimensione n:

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & & & \\ \vdots & \widehat{\mathbf{H}}_2 & & \\ 0 & & & \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & \widehat{\mathbf{H}}_2 \end{bmatrix} \quad (23)$$

E moltiplicando a sinistra per la matrice $\mathbf{A}^{(2)}$ si ottiene:

$$\mathbf{A}^{(3)} = \mathbf{H}_2 \mathbf{A}^{(2)} = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & \widehat{\mathbf{H}}_2 \end{bmatrix} \begin{bmatrix} k_1 & \mathbf{v}_1^T \\ \mathbf{0} & \widehat{\mathbf{A}}^{(2)} \end{bmatrix} = \begin{bmatrix} k_1 & \mathbf{v}_1^T \\ \mathbf{0} & \widehat{\mathbf{H}}_2 \widehat{\mathbf{A}}^{(2)} \end{bmatrix} \quad (24)$$

questi processi si iterano fino al passo n-1, dove ci si ritrova con la matrice triangolare superiore \mathbf{R} :

$$\mathbf{R} = \mathbf{A}^{(n)} = \mathbf{H}_{(n-1)} \mathbf{A}^{(n-1)} = \begin{bmatrix} k_1 & * & \dots & * \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & * \\ 0 & \dots & 0 & k_n \end{bmatrix} \quad (25)$$

Dato che $\mathbf{A} = \mathbf{QR}$, avendo l'espressione di \mathbf{R} , è facile ricavare \mathbf{Q} , che risulta:

$$\mathbf{Q} = \mathbf{H}_1 \mathbf{H}_2 \dots \mathbf{H}_{n-1} \quad (26)$$

ovvero la produttoria di tutte le matrici elementari di *Householder*.

Fattorizzazione QR nei problemi ai minimi quadrati: la fattorizzazione QR è un altro metodo efficace per la risoluzione dei problemi ai minimi quadrati. In questo metodo, come per il metodo delle equazioni normali, è più pratico usare il quadrato della norma del residuo, sostituendo \mathbf{A} con la sua fattorizzazione, $\mathbf{A}=\mathbf{QR}$, ottenendo:

$$\|\mathbf{Ax} - \mathbf{b}\|^2 = \|\mathbf{QRx} - \mathbf{b}\|^2 = \|\mathbf{Q}(\mathbf{Rx} - \mathbf{Q}^T \mathbf{b})\|^2 = \|\mathbf{Rx} - \mathbf{c}\|^2 \quad (27)$$

in cui $\mathbf{c} = \mathbf{Q}^T \mathbf{b}$, e \mathbf{Q} sparisce dai calcoli in quanto una matrice ortogonale non influenza la norma euclidea di un vettore.

La matrice \mathbf{R} ha forma:

$\mathbf{R} = \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix}$, dove \mathbf{R}_1 è una matrice quadrata triangolare superiore, e se \mathbf{A} è a rango pieno, non singolare.

Dividendo anche \mathbf{c} in modo coerente (facendo combaciare le dimensioni dei blocchi con \mathbf{R}) si ottiene

$$\mathbf{c} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix} \quad (28)$$

con $\mathbf{c}_1 \in \mathbb{R}^n$ e $\mathbf{c}_2 \in \mathbb{R}^{m-n}$, ottenendo dunque:

$$\|\mathbf{Ax} - \mathbf{b}\|^2 = \|\mathbf{Rx} - \mathbf{c}\|^2 = \left\| \begin{bmatrix} \mathbf{R}_1 \mathbf{x} \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix} \right\|^2 = \|\mathbf{R}_1 \mathbf{x} - \mathbf{c}_1\|^2 + \|\mathbf{c}_2\|^2 \quad (29)$$

Se \mathbf{R}_1 non è singolare, il sistema $\mathbf{R}_1 \mathbf{x} = \mathbf{c}_1$, ammette una ed una sola soluzione, per la quale si ha che $\|\mathbf{R}_1 \mathbf{x} - \mathbf{c}_1\|^2 = 0$. Da questa soluzione si ricava che:

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{Ax} - \mathbf{b}\| = \|\mathbf{c}_2\| \quad (30)$$

dove nell'eventualità che $\mathbf{c}_2 = \mathbf{0}$, ci si ritrova nel caso di un sistema classico, con n equazioni ed n incognite. Se non è nullo invece è la soluzione del problema di minimizzazione, e la norma di \mathbf{c}_2 è la misura del residuo. Il vantaggio del metodo della fattorizzazione \mathbf{QR} sta nell'operare direttamente sulla matrice \mathbf{A} , il cui condizionamento è la radice quadrata di quello del metodo delle equazioni normali, ovvero di $\mathbf{A}^T \mathbf{A}$. Questo aumenta la stabilità del metodo, inoltre la sua complessità computazionale è di $O(\frac{n^3}{3})$ moltiplicazioni, usando l'algoritmo di *Householder*.

1.4 Sviluppo in serie di Taylor

Lo sviluppo in serie di *Taylor*, se esiste, permette di approssimare una funzione in un intorno del punto x_0 come un polinomio di infiniti termini. Ovviamente, nelle applicazioni non si può considerare un polinomio infinito, perciò lo sviluppo si ferma ad un ordine N , esprimendo gli altri termini sotto forma di resto.

1.4.1 Teorema di Taylor

Sia $f: [a,b] \rightarrow \mathbb{R}$ una funzione, sia $x_0 \in (a,b)$ e supponiamo che esistano le derivate di f fino alla derivata $(n-1)$ esima. Preso h tale che f sia definita in $[x_0-h, x_0+h]$, vale la formula:

$$f(x_0 + h) = \sum_{i=0}^{n-1} \frac{f^{(i)}(x_0)}{i!} h^i + R_n(h) \quad (31)$$

o, in maniera equivalente, ponendo $x=x_0+h$:

$$f(x) = \sum_{i=0}^{n-1} \frac{f^{(i)}(x_0)}{i!} (x - x_0)^i + R_n(x) \quad (32)$$

dove R_n è un'opportuna funzione, detta *resto di ordine n*, ed x_0 è detto *centro dello sviluppo*. La forma di R_n non è conosciuta a priori, ci interessano però di più informazioni di tipo qualitativo/quantitativo.

1.4.2 Sviluppo di Taylor con resto di Peano

Nell'ipotesi che esista la derivata di ordine n di f il termine R_n viene espresso come:

$$R_n(x) = O[(x - x_0)^n] \quad (33)$$

dove R_n prende il nome di *resto di Peano*. Il termine $O[(x - x_0)^n]$ è dato da una qualsiasi funzione $g(x)$ tale che:

$$\lim_{x \rightarrow x_0} \frac{g(x)}{(x-x_0)^n} = 0 \quad (34)$$

Il resto di *Peano* da un'informazione di tipo qualitativo sul comportamento di $R_n(x)$.

1.4.3 Sviluppo di Taylor con resto di Lagrange

Nell'ipotesi che esista la derivata di ordine n di f , allora esiste un punto $c \in (a,b)$ che permetta di esprimere R_n come:

$$R_n(x) = \frac{f^{(n)}(c)}{n!} (x - x_0)^n \quad (35)$$

con R_n detto *resto di Lagrange*. Il resto di Lagrange da un'informazione di tipo quantitativo.

1.5 Trasformata di Fourier

La trasformata di Fourier nasce per sopperire ai limiti della serie di Fourier, ovvero per poter rappresentare anche funzioni non periodiche. In questo caso la serie viene sostituita da un integrale. I valori a_k e b_k che determinano le ampiezze di ciascuna armonica, diventano i valori della trasformata di Fourier $\mathcal{F}\{f\}$ calcolata in k , la quale risulta definita sull'intera retta reale.

La trasformata di Fourier di una funzione f definita su tutto lo spazio reale è:

$$\mathcal{F}\{f\} = F(k) = \hat{f}(k) = \int_{-\infty}^{\infty} e^{-ikx} f(x) dx \quad (36)$$

definita nello spazio delle frequenze. Per la trasformata inversa di Fourier si intende la funzione:

$$\mathcal{F}^{-1}\{F\} = f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{ikx} F(k) dk \quad (37)$$

definita nello spazio ordinario. La definizione può variare a seconda dell'ambito in cui si utilizza, in alcuni ambiti il fattore $\frac{1}{2\pi}$ si trova nella formula per la trasformata, oppure i segni dei due esponenziali vengono cambiati. Tutte le definizioni sono comunque equivalenti a quella esposta.

1.5.1 Proprietà della trasformata di Fourier

Nel seguito verranno enunciate le proprietà della trasformata di Fourier. Dato che lo scopo del capitolo è solo quello di far comprendere al meglio lo strumento matematico che poi verrà ripreso nel capitolo successivo, si eviterà la dimostrazione delle proprietà qui discusse.

Linearità: dalla definizione di trasformata, essendo l'operatore integrale un operatore lineare, segue che, se f e g sono dotate di trasformata di Fourier:

$$\mathcal{F}\{cf(x)+kg(x)\} = c\mathcal{F}\{f(x)\} + k\mathcal{F}\{g(x)\} = cF(k) + kF(k) \quad (38)$$

per ogni coppia di numeri complessi c e k .

Traslazione nello spazio ordinario:

$$\mathcal{F}\{f(x-x_0)\} = e^{-ikx_0}F(k) \quad (39)$$

Traslazione nello spazio delle frequenze:

$$\mathcal{F}\{e^{ik_0x} f(x)\} = F(k-k_0) \quad (40)$$

Variazione di scala: per ogni $a \in \mathbb{R}$, purché $a \neq 0$

$$\mathcal{F}\{f(ax)\} = \frac{1}{|a|} F\left(\frac{k}{a}\right) \quad (41)$$

Simmetria: Se una funzione f può essere considerata come la trasformata di Fourier di una funzione g , allora la trasformata di f è semplicemente 2π che moltiplica $g(-k)$.

Modulazione: se $\mathcal{F}\{f\} = F(k)$ allora:

$$\mathcal{F}\{f(x)\cos(k_0x)\} = \frac{1}{2}\{F(k - k_0) + F(k + k_0)\} \quad (42)$$

$$\mathcal{F}\{f(x)\sin(k_0x)\} = \frac{1}{2i}\{F(k - k_0) - F(k + k_0)\} \quad (43)$$

Trasformata di Fourier di una derivata: se f è una funzione continua e derivabile in \mathbb{R} , con f' continua a tratti in ogni intervallo $[-L, L]$ ed assolutamente integrabile, vale:

$$\mathcal{F}\{f'\} = ikF(k) \quad (44)$$

Derivazione nello spazio delle frequenze: se una funzione f ammette la trasformata di Fourier ed inoltre la funzione $xf(x)$ è assolutamente integrabile in \mathbb{R} , allora:

$$\mathcal{F}\{xf(x)\} = iF'(k) \quad (45)$$

Convoluzione: se f e g sono dotate di trasformata di Fourier, anche la loro convoluzione è dotata di trasformata di Fourier e tra esse vale la seguente relazione:

$$\mathcal{F}\{f * g\} = F(k)G(k) \quad (46)$$

1.6 Interpolazione

Un altro elemento che tornerà utile nel corso del lavoro di tesi è il concetto di interpolazione. L'interpolazione è un metodo per individuare nuovi punti del piano cartesiano a partire da un insieme finito di punti dati. Questo si può fare partendo dall'ipotesi che tutti i punti si possano riferire ad una funzione $f(x)$ di una famiglia di funzioni di variabile reale.

1.6.1 Definizione del problema

Sia data una sequenza di n numeri reali distinti x_k chiamati nodi, e per ciascuno di questi x_k sia dato un secondo numero y_k . Si vuole dunque individuare una funzione f di una certa famiglia tale che sia:

$$f(x_k) = y_k \text{ per } k=1,\dots,n \quad (47)$$

Una coppia (x_k, y_k) viene detta *punto dato* ed f è la *funzione interpolante* per i punti dati. Esistono varie tecniche di interpolazione, qui si tratteranno solo l'interpolazione lineare e l'interpolazione *spline*.

1.6.2 Interpolazione lineare

L'interpolazione lineare, come suggerisce il nome, unisce i vari punti dati consecutivi con una retta. In generale, l'interpolazione lineare per ogni coppia di punti dati consecutivi, denotandoli (x_a, y_a) e (x_b, y_b) , si definisce come funzione interpolante nell'intervallo $[x_a, x_b]$ la:

$$f(x) = \frac{x-x_b}{x_a-x_b} y_a - \frac{x-x_a}{x_a-x_b} y_b \quad (48)$$

Questa formula può essere interpretata come valutazione della media ponderata. Un vantaggio dell'interpolazione lineare è senza dubbio la rapidità, mentre la precisione non è tra le migliori. Un altro svantaggio è che l'interpolante non è differenziabile nei punti dati. La seguente stima dell'errore indica che l'interpolazione lineare non è molto precisa. Indichiamo con $g(x)$ la funzione interpolante e supponiamo che la x sia compresa fra x_a, x_b e che $g(x)$ sia due volte derivabile. Allora l'errore dell'interpolazione lineare è:

$$|f(x) - g(x)| \leq C(x_b - x_a)^2 \quad (49)$$

dove

$$C = \frac{1}{8} \max_{y \in [x_a, x_b]} g''(y) \quad (50).$$

Quindi, l'errore è proporzionale al quadrato della distanza fra i punti dati.

1.6.2 Interpolazione spline

L'interpolazione spline è un particolare metodo di interpolazione che si basa sulle funzioni spline.² Sia tratta di uno strumento dell'analisi numeri molto utilizzato, come ad esempio in fisica o statistica. L'interpolazione spline è ottenuta suddividendo l'intervallo in più sotto-intervalli ($I_k = [x_k, x_{k+1}]$ con $k = 1, \dots, N-1$) e scegliendo per ciascuno di essi un polinomio di grado d (solitamente piccolo) Verrà in seguito imposto che due polinomi successivi si saldino in modo liscio, ovvero, osservando la continuità delle prime $d-1$ derivate. La funzione che si ottiene con questo procedimento è detta *funzione spline*.

² In analisi matematica, una spline è una funzione costituita da un insieme di polinomi raccordati tra loro, il cui scopo è interpolare in un intervallo un insieme di punti, in modo che tale funzione si continua almeno fino ad un dato ordine di derivate in ogni punto dell'intervallo.

Capitolo 2

Tecniche per la stima del moto denso.

Gli oggetti in una scena possono apparire in movimento, anche se nessuno di essi si muove e questo dipende dal moto dell'osservatore o del sistema di visione (biologico, come gli occhi, oppure non-biologico come una videocamera). Questi sistemi danno un'uscita che può essere considerata pressoché continua, questa sequenza di informazioni è detta Optical Flow. Nella Computer Vision, il principale problema è elaborare questo flusso di dati in tempi rapidi a sufficienza per poter notare le variazioni in esso contenute. Tipicamente ci sono diverse tecniche per poter stimare il moto in 3D a partire dalle sue proiezioni in 2D, queste tecniche si possono dividere in 2 macro-categorie: quelle che stimano il moto basandosi sull'analisi dell'Optical Flow e quelle che si basano sulle corrispondenze discrete. Volendo è possibile fare un'ulteriore suddivisione nelle sopracitate categorie, per cui i metodi possono essere dipendenti o meno dal dominio, ovvero, possono basarsi o non basarsi su informazioni a priori dell'ambiente. Tuttavia queste tecniche fanno tutte parte di quelli che in matematica vengono chiamati *Problemi inversi*, ossia problemi dove non è garantita né l'esistenza, né l'unicità della soluzione, oltre ad essere soggetti ad un'elevata sensibilità di propagazione dell'errore, come spiegato nel capitolo precedente. Gli algoritmi per l'allineamento delle immagini e la stima del moto in sequenze video, sono tra i più diffusi ed usati nell'ambito della Computer Vision. Un primo esempio di algoritmo diffuso per la registrazione delle immagini è il *patch-based translational alignment (Optical Flow) technique*, sviluppato da Lucas e Kanade, che verrà discusso nel capitolo successivo, nel quale verranno trattati gli algoritmi. Varianti di quest'algoritmo sono usate per la compressione video, come il MPEG o l'H.263. Metodi di parametrizzazione del moto simili hanno vari campi d'applicazione, come ad esempio l'ambito medico. Per stimare il moto tra due immagini bisogna prima scegliere un *errore di misura* accettabile per comparare le due immagini. Una volta stabilito ciò, bisogna escogitare una tecnica di *search* (ricerca) adeguata. La tecnica più semplice consiste nel provare tutti i possibili allineamenti, ad esempio, fare una *full search* (ricerca completa). Nella pratica però risulterebbe troppo lenta, infatti vengono utilizzate altre tecniche, dette *hierarchical coarse-to-fine technique*, letteralmente tecniche gerarchiche dal grossolano al raffinato, basate su piramidi di immagini. In

alternativa, per velocizzare il calcolo si può usare la trasformata di Fourier. Ogni quantità, informazione e caratteristica, che può essere estratta da un'immagine è limitata dalla quantizzazione in pixel dell'immagine stessa. Nonostante questo però, esaminando un intorno del punto da estrarre, è possibile fornire una stima, approssimata, della posizione della caratteristica con una precisione inferiore al pixel (*sub-pixel*). Questi approcci sono tutti basati sul tentativo di modellare localmente la funzione immagine, quantizzata, cercando di ricostruire l'informazione distrutta dalla quantizzazione spaziale. Per avere una precisione *sub-pixel* nell'allineamento si usano spesso metodi incrementali basati sullo sviluppo in serie di Taylor. Possono essere usate anche nei *modelli di moto parametrici*. La stima del moto può venir resa più affidabile dall'apprendimento delle dinamiche tipiche o dal moto statico della scena oppure dall'oggetto monitorato, come ad esempio la naturale andatura della camminata. Per moti più complessi vengono usati i modelli parametrici *spline motion*. In presenza di moti multipli indipendenti (e probabilmente non rigidi), si usano invece le tecniche sull'*optical flow*.

2.1 Translational alignment (Allineamento traslazionale)

Il modo più facile per stabilire un allineamento tra due immagini, o un patch di immagini, è di traslare un'immagine sull'altra. Dato un modello di immagine $I_0(\mathbf{x})$, campionato a locazioni di pixel discrete $\{\mathbf{x}_i = (x_i, y_i)\}$, vorremo trovare dove è posizionata nella immagine $I_1(\mathbf{x})$. Una soluzione ai minimi quadrati di questo problema, è quello di trovare il minimo della funzione *somma di differenze quadratiche (SSD)*:

$$E_{SSD}(\mathbf{u}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2 = \sum_i e_i^2 \quad (1)$$

dove $\mathbf{u} = (u, v)$ è lo spostamento, $I_0(\mathbf{x}_i)$ e $I_1(\mathbf{x}_i)$ sono le due immagini campionate a locazioni di pixel discrete, mentre $e_i = I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)$ è chiamato *errore residuo* (o *differenza di spostamento tra frame* nella letteratura di codifica video)³. L'assunzione che i corrispondenti valori dei pixel rimangano gli stessi nelle due immagini è spesso chiamata *vincolo di costanza della luminosità*. Tipicamente lo spostamento \mathbf{u} può essere frazionato, perciò bisogna applicare all'immagine $I_1(\mathbf{x})$ un'interpolazione adatta. Nella pratica è spesso usata un'interpolazione bilineare, anche se in alcuni casi un'interpolazione bicubica può dare dei risultati lievemente migliori. I colori dell'immagine possono essere processati sommando le differenze attraverso tutti i tre canali di colore, nonostante sia

³ Per il momento ignoriamo la possibilità che parti di $I_0(\mathbf{x})$ possano trovarsi fuori dai confini di $I_1(\mathbf{x})$, oppure non siano visibili.

possibile trasformare l'immagine in uno spazio di colori differente, oppure usare la luminanza (approccio tipico della codifica video). La luminanza è una grandezza fotometrica vettoriale definita come il rapporto tra l'intensità luminosa emessa da una sorgente nella direzione dell'osservatore e l'area apparente della superficie emittente, così come vista dall'osservatore.

Robustezza agli errori di misurazione: possiamo rendere l'errore residuo più robusto ai valori anomali ponendo una funzione di robustezza insieme ai termini di errore quadratico, ottenendo:

$$E_{SRD}(\mathbf{u}) = \sum_i \rho(I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)) = \sum_i \rho(e_i) \quad (2)$$

dove $\rho(e_i)$ è la funzione di robustezza.

Il valore assoluto di $\rho(e_i)$ è una funzione che cresce più lentamente della funzione quadratica associata ai minimi quadrati. Una funzione simile viene occasionalmente usata nella stima del moto nell'ambito della codifica video. Viene utilizzata questa variante grazie alla sua velocità, la quale è detta *misura della somma delle differenze assolute* (*SAD*) o norma L_1 ⁴:

$$E_{SAD}(\mathbf{u}) = \sum_i |I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)| = \sum_i |e_i| \quad (3)$$

Tuttavia, dato che la funzione non è derivabile nell'origine, non è molto indicata per approcci a gradiente decrescente. Invece, risulta più indicata un'ulteriore variante che si comporta da quadratica per piccoli valori, ma cresce più lentamente delle classiche quadratiche come ci si allontana dall'origine. La variante presentata è quella di Geman-McClure:

$$\rho_{GM}(x) = \frac{x^2}{1 + x^2/a^2} \quad (4)$$

dove a è una costante che viene assunta come un valore anomalo di soglia.

Spatially varying weights (variabilità spaziale dei pesi): l'errore di misura precedentemente usato, si basava sull'ignorare che fosse possibile, per un dato allineamento, che alcuni pixel confrontati potessero trovarsi fuori dai confini

⁴ Si dice che una funzione f appartiene ad $L_1(\mathbb{R})$ se esiste finito l'integrale $\int |f(y)| dy < \infty$.

dell'immagine originale. Inoltre, ci piacerebbe diminuire, parzialmente o completamente, il peso del contributo di certi pixel. Per applicazioni come la stabilizzazione dello sfondo, è auspicabile diminuire il peso della parte centrale dell'immagine, dove spesso vengono ripresi dalla videocamera oggetti estranei in movimento. Questo può essere possibile associando una variabilità spaziale al valore del peso per-pixel, per ognuna delle due immagini confrontate. L'errore di misura quindi diventa:

$$E_{WSSD}(\mathbf{u}) = \sum_i \omega_0(\mathbf{x}_i) \omega_1(\mathbf{x}_i + \mathbf{u}) [I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2 \quad (5)$$

dove le funzioni del peso ω_0 e ω_1 valgono zero fuori dai confini dell'immagine. Se è permesso un gran numero di moti potenziali, l'errore appena enunciato può avere una polarizzazione verso soluzioni sovrapposte più piccole. Per contrastare questa polarizzazione la (5) può essere divisa per l'area di sovrapposizione

$$A = \sum_i \omega_0(\mathbf{x}_i) \omega_1(\mathbf{x}_i + \mathbf{u}) \quad (6)$$

per calcolare un valore di media dell'errore quadrato. La radice quadrata di questa quantità è la *radice dell'intensità media dell'errore*

$$RMS = \sqrt{E_{WSSD}/A} \quad (7)$$

Bias and gain con exposure difference (polarizzazione e guadagno con la differenza d'esposizione): spesso le due immagini allineate non vengono prese con la stessa esposizione. Un modello lineare della variazione di intensità tra le due immagini è il *modello di polarizzazione e guadagno*

$$I_1(\mathbf{x} + \mathbf{u}) = (1 + \alpha)I_0(\mathbf{x}) + \beta \quad (8)$$

dove α è il guadagno e β è la polarizzazione. La formula ai minimi quadrati diventa:

$$E_{BG} = \sum_i [I_1(\mathbf{x}_i + \mathbf{u}) - (1 + \alpha)I_0(\mathbf{x}_i) - \beta]^2 = \sum_i [\alpha I_0(\mathbf{x}_i) + \beta - e_i]^2 \quad (9)$$

Per un'immagine a colori è necessario assegnare valori di α e β per ogni canale di colore, al fine di compensare la correzione di colore automatica implementata in alcune fotocamere.

Correlazione: un'alternativa al prendere la differenza d'intensità è fare la correlazione, ovvero, massimizzare il prodotto delle due immagini allineate

$$E_{CC} = \sum_i I_0(\mathbf{x}_i) I_1(\mathbf{x}_i + \mathbf{u}) \quad (10)$$

A prima vista sembrerebbe rendere inutile il modello di polarizzazione e guadagno, dato che le immagini preferiranno disporsi a prescindere dai loro valori di scala e contrasto. Tuttavia ciò non avviene. Difatti, se esiste in $I_1(\mathbf{x})$ una zona molto luminosa, il massimo del prodotto potrebbe giacere in quell'area. Per questa ragione viene utilizzato più comunemente la *cross-correlazione normalizzata*

$$E_{NCC}(\mathbf{u}) = \frac{\sum_i [I_0(\mathbf{x}_i) - \bar{I}_0][I_1(\mathbf{x}_i + \mathbf{u}) - \bar{I}_1]}{\sqrt{\sum_i [I_0(\mathbf{x}_i) - \bar{I}_0]^2} \sqrt{\sum_i [I_1(\mathbf{x}_i + \mathbf{u}) - \bar{I}_1]^2}} \quad (11)$$

$$\text{dove } \bar{I}_0 = \frac{1}{N} \sum_i I_0(\mathbf{x}_i), \quad \bar{I}_1 = \frac{1}{N} \sum_i I_1(\mathbf{x}_i + \mathbf{u})$$

sono le immagini medie nella zona corrispondente ed N è il numero di pixel nella zona. Il risultato della cross-relazione normalizzata sarà sempre all'interno dell'intervallo $[-1;1]$, a meno che le due zone non abbiano varianza zero, in tal caso risulta indefinito. Questo metodo funziona particolarmente bene quando le immagini sono prese con un'esposizione diversa.

2.2 Stima gerarchica del moto.

Abbiamo una funzione costo da ottimizzare, dobbiamo dunque trovarne il minimo; la soluzione più semplice è fare una *full search* su alcuni intervalli di traslazione usando step a numeri interi oppure sub-pixel. Per accelerare questo processo, viene spesso utilizzata la gerarchia per la stima del moto, dove un'immagine piramidale viene costruita e si cerca un minor numero discreto di pixel (che corrispondano allo stesso intervallo di moto). La prima iterazione darà dei risultati 'grezzi'. La stima del moto da un livello della piramide viene utilizzato per inizializzare una ricerca locale più piccola nel livello successivo, dando risultati migliori. In alternativa, alcune risorse (soluzioni buone) del livello grezzo, possono essere utilizzate per iniziare una ricerca nei livelli più raffinati. Anche se non è garantito lo stesso risultato di una *full search*, solitamente si ottengono risultati comparabili, ma molto più velocemente. Formalmente abbiamo

$$I_k^l(\mathbf{x}_j) \leftarrow \tilde{I}_k^{(l-1)}(2\mathbf{x}_j) \quad (12)$$

l'immagine decimata al livello l , ottenuta campionando una più adeguata versione dell'immagine al livello $l-1$.⁵ Nel livello più grezzo, ricerchiamo il miglior spostamento $\mathbf{u}^{(l)}$ che minimizzi la differenza tra le immagini $I_0^{(l)}$ e $I_1^{(l)}$. Questo si ottiene facendo una full search in alcuni range di spostamento $\mathbf{u}^{(l)} \in 2^{-l}[-S, S]^2$, dove S è l'intervallo di ricerca desiderato al miglior livello di risoluzione. Una volta definito un vettore di moto adeguato, si 'predice' uno spostamento verosimile

$$\hat{\mathbf{u}}^{(l-1)} \leftarrow 2\mathbf{u}^{(l)} \quad (13)$$

per il prossimo livello più raffinato. La ricerca degli spostamenti viene iterata fino al livello migliore, dove ci sarà un limitato range di spostamenti, del tipo $\hat{\mathbf{u}}^{(l-1)} \pm 1$.

2.3 Allineamento basato sulla trasformata di Fourier.

Quando l'intervallo di ricerca corrisponde ad una frazione significativa dell'immagine più grande, il *metodo gerarchico* non si adatta bene alla situazione. Spesso, infatti, un peggioramento eccessivo della rappresentazione (inteso come scendere nei livelli più grezzi), potrebbe causare un offuscamento di caratteristiche fondamentali dell'immagine. In casi simili è più indicato un approccio che utilizza la trasformata di Fourier. L'allineamento con Fourier si basa sulla proprietà della trasformata di un segnale traslato, ovvero, sul fatto che la trasformata di un segnale traslato avrà lo stesso modulo della trasformata del segnale originario e risulterà differente solo la fase:

$$\mathcal{F}\{I_1(\mathbf{x} + \mathbf{u})\} = \mathcal{F}\{I_1(\mathbf{x})\}e^{-j\mathbf{u}\boldsymbol{\omega}} = I_1(\boldsymbol{\omega})e^{-j\mathbf{u}\boldsymbol{\omega}} \quad (14)$$

Dove $\boldsymbol{\omega}$ è il vettore della velocità angolare della trasformata di Fourier.⁶ Un'altra proprietà utile è quella sulla convoluzione, che risulta una moltiplicazione nel dominio di Fourier.⁷ Perciò, la trasformata della *cross-correlazione* E_{CC} può essere scritta come

$$\mathcal{F}\{E_{CC}(\mathbf{u})\} = \mathcal{F}\{I_0(\mathbf{u}) * I_1(\mathbf{u})\} = I_0(\boldsymbol{\omega})I_1^*(\boldsymbol{\omega}) \quad (15)$$

⁵ Decimare l'immagine ne riduce la risoluzione. Per poterla decimare dobbiamo prima fare una convoluzione con un filtro passa-basso, al fine di evitare fenomeni di aliasing, poi prendere ogni r -esimo campione, tutto questo si fa a livello concettuale. Nella pratica invece si valuta solo la convoluzione per ogni r -esimo campione

$$g(i, j) = \sum_{k, l} f(k, l)h(r_i - k, r_j - l)$$

⁶ Si utilizza anche qui la notazione $I_1(\boldsymbol{\omega}) = \mathcal{F}\{I_1(\mathbf{x})\}$ per indicare la trasformata di Fourier del segnale.

⁷ Argomento che viene affrontato nel capitolo precedente.

Dunque per calcolare E_{CC} nell'intervallo di tutti i possibili valori di \mathbf{u} , è più pratico usare Fourier e poi anti-trasformare per ottenere il risultato. L'algoritmo per *la trasformata veloce di Fourier* può calcolare la trasformata di un'immagine $N \times M$ in $O(NM \log NM)$, dunque molto più rapidamente rispetto alle operazioni richieste per una *full search* nell'intervallo di sovrapposizione delle immagini. Inoltre possiamo utilizzare Fourier anche nella funzione della *somma di differenze quadrate* (1):

$$\mathcal{F}\{E_{SSD}(\mathbf{u})\} = \delta(\mathbf{u}) \sum_i [I_0^2(\mathbf{x}_i) + I_1^2(\mathbf{x}_i)] - 2I_0(\boldsymbol{\omega})I_1^*(\boldsymbol{\omega}) \quad (16)$$

Correlazione di inquadatura: sfortunatamente, la proprietà sulla convoluzione, può essere applicata solo se la sommatoria degli \mathbf{x}_i è fatta su tutti i pixel di entrambe le immagini, usando una traslazione circolare dell'immagine quando dei pixel accessibili si trovano fuori dai confini dell'originale. Benché questo sia accettabile per piccole traslazioni e per immagini confrontabili, perde di significato quando l'immagine si sovrappone di poco, oppure, se una è un sottoinsieme dell'altra. In quel caso, la *funzione di cross-correlazione* dovrebbe essere sostituita con una *funzione di cross-correlazione inquadrate (pesata)*

$$E_{WCC}(\mathbf{u}) = \sum_i \omega_0(\mathbf{x}_i)I_0(\mathbf{x}_i)\omega_1(\mathbf{x}_i + \mathbf{u})I_1(\mathbf{x}_i + \mathbf{u}) = [\omega_0(\mathbf{x})I_0(\mathbf{x})] * [\omega_1(\mathbf{x})I_1(\mathbf{x})] \quad (17)$$

dove le funzioni di peso ω_0 e ω_1 valgono zero fuori dall'intervallo valido delle immagini, ed entrambe le immagini sono piene, cosicché la traslazione circolare restituisca come valore zero fuori dai confini dell'immagine originale.

Correlazione di fase: una variante della normale correlazione (15) che talvolta viene usata per la stima del moto è la *correlazione di fase*. Qui, lo spettro dei due segnali associati viene "sbiancato" (*whitened*) dividendo ogni prodotto tra frequenze della (15) per il modulo delle trasformate di Fourier

$$\mathcal{F}\{E_{PC}(\mathbf{u})\} = \frac{I_0(\boldsymbol{\omega})I_1^*(\boldsymbol{\omega})}{\|I_0(\boldsymbol{\omega})\| \|I_1(\boldsymbol{\omega})\|} \quad (18)$$

prima di antitrasformare. Nel caso di segnali privi di rumore con un traslamento (ciclico) perfetto, otteniamo $I_1(\mathbf{x} + \mathbf{u}) = I_0(\mathbf{x})$ e quindi dalla (14) otteniamo

$$\mathcal{F}\{I_1(\mathbf{x} + \mathbf{u})\} = I_1(\boldsymbol{\omega})e^{-2\pi j\mathbf{u}\boldsymbol{\omega}} = I_0(\boldsymbol{\omega}) \text{ ed anche}$$

$$\mathcal{F}\{E_{PC}\} = e^{-2\pi j\mathbf{u}\boldsymbol{\omega}} \quad (19).$$

Il risultato della correlazione di fase (sotto condizioni ideali) è dunque un singolo impulso posizionato all'esatto valore di \mathbf{u} , che (in teoria) rende più semplice arrivare ad una stima esatta. La correlazione di fase sembra “battere” la classica correlazione, tuttavia il suo comportamento dipende dalle caratteristiche del segnale e dal rumore. Se le immagini originali sono affette da rumore in una banda di frequenze ristretta, allora il processo di “sbiancamento” smorza effettivamente il rumore in quella regione. Però, se il segnale ha un basso *signal-to-noise ratio* a qualche frequenza, il processo di “sbiancamento” può ridurre le performance. Un'alternativa recente alla correlazione di fase è la *cross-correlazione di gradiente*.

2.4 Spline-based Motion (metodo basato sulle funzioni spline)

Molte immagini di moti sono troppo complicate per essere acquisite con metodi a poche dimensioni, come ad esempio quelli di tipo parametrico. Tipicamente, gli algoritmi per il flusso ottico calcolano una stima del moto indipendente per ogni pixel, perciò, il numero di vettori di flusso calcolati è uguale al numero di pixel in ingresso. L'equazione generale del flusso ottico, omologa alla (1) può essere scritta come:

$$E_{SSD_{OF}}(\{\mathbf{u}_i\}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u}_i) - I_0(\mathbf{x}_i)]^2 \quad (20).$$

Da notare come nell'equazione il numero delle variabili $\{\mathbf{u}_i\}$ è il doppio del numero delle misure effettuate, dunque il problema risulta *sottodeterminato*.⁸ Il metodo *spline* (linguetta) si trova a metà strada tra il *metodo del flusso ottico* (flusso indipendente per ogni pixel) e il *metodo del flusso parametrico* (avente un piccolo numero di variabili globali). L'approccio è quello di rappresentare il campo del moto come una *spline* bidimensionale controllata da un piccolo numero di vertici di controllo $\{\hat{\mathbf{u}}_j\}$ (Figura 1)

$$\mathbf{u}_i = \sum_j \hat{\mathbf{u}}_j B_j(\mathbf{x}_i) = \sum_j \hat{\mathbf{u}}_j \omega_{ij} \quad (21)$$

⁸ Per la definizione si rimanda al capitolo precedente.

dove $B_j(x_i)$ sono le funzioni di base, che assumono valore diverso da zero solo in un piccolo intervallo di supporto.

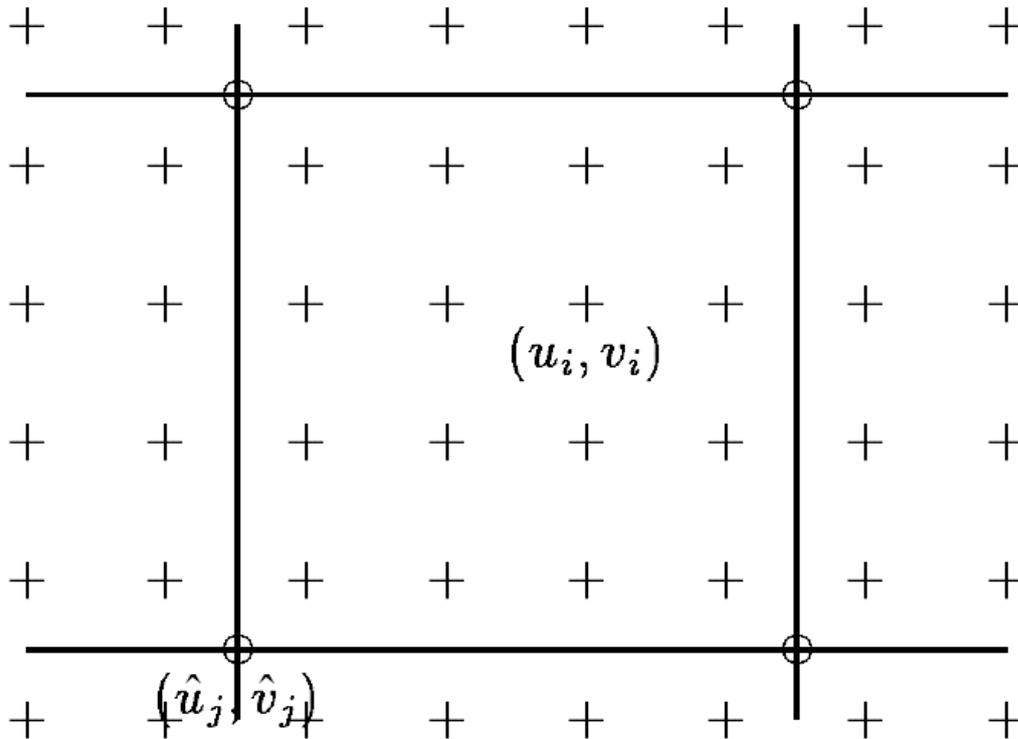


Figura 1: Campo del moto 'spline': i vettori spostamento $u_i = (u_i, v_i)$ vengono rappresentati con le croci (+), e vengono controllati da un minor numero di vertici di controllo $(u_j, v_j) = (\hat{u}_j, \hat{v}_j)$, rappresentati con i cerchi (o).

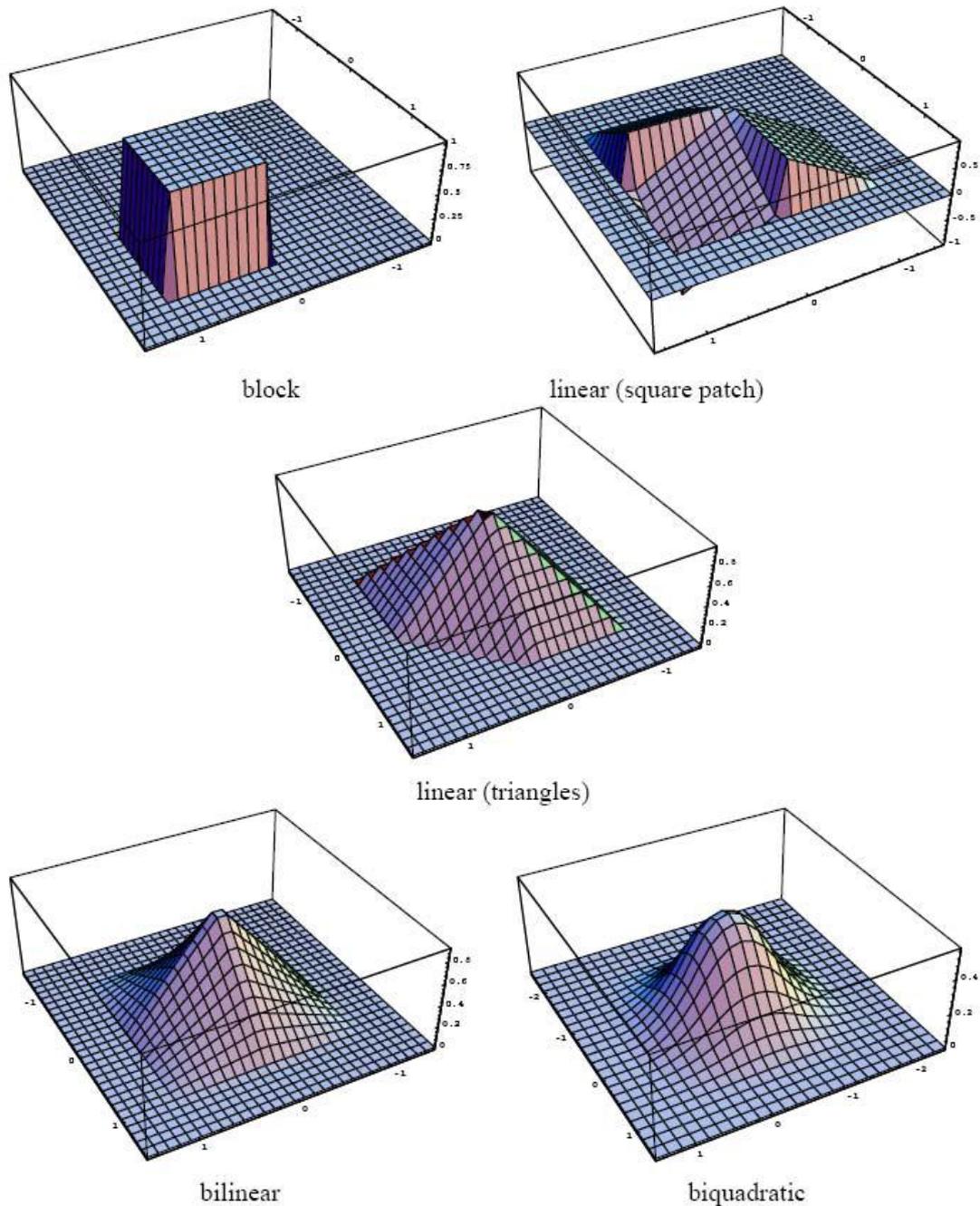


Figura 2: Alcune funzioni di spline utilizzate di frequente.

Chiamiamo inoltre $\omega_{ij} = B_j(\mathbf{x}_i)$ pesi, per accentuare il concetto che ogni $\{\mathbf{u}_i\}$ è una combinazione lineare nota di $\{\hat{\mathbf{u}}_j\}$. Comunemente vengono utilizzate delle *spline* di base, alcune sono mostrate in figura 2. Sostituendo la formula per i vettori di flusso individuali di ogni pixel (21) nella (20) per l'errore di misura, otteniamo una formula per il moto

parametrico simile all'espressione per i moti parametrici.⁹ La più grande differenza è che il Jacobiano $J_1(\mathbf{x}'_i)$ ¹⁰ ora è formato da inserimenti sparsi nella matrice di peso $\mathbf{W}=[\omega_{ij}]$. In molti casi, il numero ridotto di *spline* rendono la stima del moto un problema ben condizionato. Invece, se una vasta regione senza struttura, oppure degli spigoli allungati soggetti al problema d'apertura, persistono nonostante le diverse ricostruzioni con le *spline*, può essere necessario aggiungere un termine di regolazione per rendere il problema ben posto. Se una strategia multisoluzione (del tipo *coarse-to-fine*) viene usata, è importante riscaldare questi termini rifiniti mentre si passa da un livello all'altro. Il sistema lineare corrispondente al modello di stima del moto con le *spline* è di tipo sparso e regolare. Dato che è di dimensioni moderate, spesso viene utilizzato il metodo di Cholesky per la sua risoluzione. Invece, se la matrice del sistema diventa troppo grande, si prediligono metodi iterativi, come ad esempio il *metodo gerarchico del gradiente coniugato pre-condizionato*. Grazie alla sua robustezza, il calcolo del moto attraverso il metodo delle *spline*, è utilizzato in molte applicazioni, anche in ambito medico. Uno svantaggio di questa tecnica (base) è che il modello non lavora bene in condizioni di moti discontinui, a meno che non si usino un numero eccessivo di nodi. Per rimediare a questo inconveniente, Szeliski e Shum, proposero di utilizzare una rappresentazione *quadtree* implementata nella griglia di controllo della *spline* (figura 3a). Le celle più grandi sono utilizzate per rappresentare regioni di moto fluido, mentre le regioni più piccole, sono adibite alla rappresentazione di moti discontinui (figura 3c). Per stimare il moto viene utilizzato il metodo *coarse-to-fine*. Partendo dalla *spline* classica, utilizzandola in un'immagine a bassa risoluzione, viene stimato un calcolo iniziale del moto. I riempimenti a *spline* dove il moto è inconsistente, ovvero il residuo quadrato (20) è oltre una soglia accettabile, vengono suddivisi in riempimenti più piccoli. Per evitare spaccature (*cracks*) nel campo di moto risultante (figura 3b), i valori di certi nodi nella maglia rifinita, cioè quelli adiacenti a celle più grandi, necessitano di essere ridotti, cosicché possano dipendere dal loro valore iniziale. Questo è il metodo più semplice utilizzando una rappresentazione gerarchica base, in riferimento alla *spline quadtree*, la quale imposta selettivamente alcune funzioni gerarchiche di base a zero.

⁹ $E_{LKPM}(\mathbf{p} + \Delta\mathbf{p}) \approx \sum_i [I_1(\mathbf{x}'_i) + J_1(\mathbf{x}'_i)\Delta\mathbf{p} - I_0(\mathbf{x}_i)]^2$, dove \mathbf{J} = jacobiano.

¹⁰ $J_1(\mathbf{x}'_i) = \frac{\partial I_1}{\partial \mathbf{p}} = \nabla I_1(\mathbf{x}'_i) \frac{\partial \mathbf{x}'_i}{\partial \mathbf{p}}(\mathbf{x}_i)$

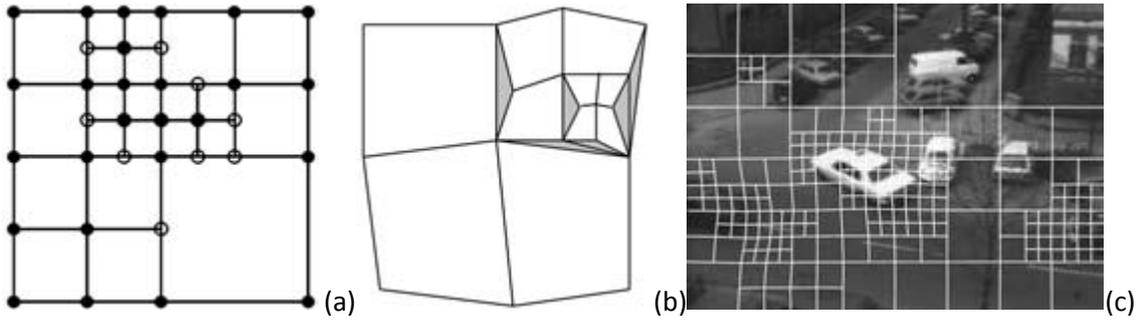


Figura 3: calcolo del moto con il metodo quadtree. a) rappresentazione del metodo quadtree. b) spaccatura causata poichè i nodi bianchi non dipendevano dai loro valori iniziali. c) quadtree spline deforme implementato in un'immagine in bianco e nero.

2.5 Spostamento relativo e Optical Flow.

Il metodo più generale ed impegnativo per la stima del moto è quello che calcola un moto indipendente per ogni pixel. Questo metodo è chiamato *Optical Flow*. Come detto precedentemente, di solito, comporta la minimizzazione di luminosità, o la minimizzazione della differenza di colori fra la somma dei pixel sull'immagine, (20). Inoltre, come gli altri metodi è un problema sottodeterminato. I due approcci classici sono: la sommatoria relativa sulla regione di sovrapposizione (approccio sull'inquadratura) oppure aggiungere termini di fluidità sul campo dei $\{\mathbf{u}_i\}$ usando, o delle regolazioni, oppure il campo casuale di Markov, al fine di trovare un minimo assoluto. Invece di risolvere il sistema in modo indipendente per ogni moto associato, Horn e Schunck, svilupparono un *framework* basato sulle regolarizzazioni dove la (20) è minimizzata contemporaneamente su tutti i vettori di flusso $\{\mathbf{u}_i\}$. Affinché il problema sia vincolato, le incertezze quadrate sulla derivata di flusso vengono aggiunte al calcolo dell'errore di misura su ogni pixel. Dato che la tecnica fu sviluppata per piccoli moti in *framework* variabili, il vincolo di costanza luminosa linearizzato¹¹ può venir scritto in forma integrale

$$E_{HS} = \int (I_x u + I_y v + I_t)^2 dx dy \quad (22)$$

¹¹ $E_{LKSSD}(\mathbf{u} + \Delta\mathbf{u}) = \sum_i [J_1(x_i + \mathbf{u})\Delta\mathbf{u} + e_i]^2$

dove $(I_x, I_y) = \nabla I_1 = \mathbf{J}_1$ ed $I_t = e_i$ è la derivata in base al tempo, quindi in breve, la luminosità cambia nelle due immagini. Il modello Horn-Schunck può venir visto anche come il caso limite della tecnica *spline*, dove le *spline* diventano di dimensioni 1x1 pixel. È possibile inoltre combinare le due idee di calcolo del flusso relativo ed assoluto, in modo da ottenere un unico *framework*, utilizzando Hessiani aggregati localmente come termine di costanza di luminosità. L'energia analitica totale sarà:

$$E_{HSD} = \sum_i \mathbf{u}_i^T [\mathbf{J}_i \mathbf{J}_i^T] \mathbf{u}_i + 2e_i \mathbf{J}_i^T \mathbf{u}_i + e_i^2 \quad (23).$$

Se sostituiamo l'Hessiana (di rango 1) associata al singolo pixel, $\mathbf{A}_i = [\mathbf{J}_i \mathbf{J}_i^T]$, ed i residui $\mathbf{b}_i = \mathbf{J}_i e_i$ con la versione per aree aggregate¹², otteniamo un algoritmo di minimizzazione globale, nel quale ci sono regioni che utilizzano vincoli di luminosità.

Moto in 2D: Avendo un punto nello spazio, $\mathbf{P} = (X, Y, Z)$, immaginiamo un suo movimento lineare tra l'istante $t \cdot \delta t$ e l'istante $(t+1) \cdot \delta t$, con velocità

$\mathbf{v} = (v_x, v_y, v_z)^T$ definendo un moto 3D $\mathbf{v} \cdot \delta t$ che parte da $\mathbf{P} = (X, Y, Z)$ e finisce a $\mathbf{P}' = (X+v_x \cdot \delta t, Y+v_y \cdot \delta t, Z+v_z \cdot \delta t)$. Il vettore $\mathbf{d} = (\xi, \psi)^T$, è la proiezione del moto 3D del punto P, tra le immagini $\mathbf{I} = (\cdot, \cdot, t)$ e $\mathbf{I} = (\cdot, \cdot, t+1)$ figura 4.

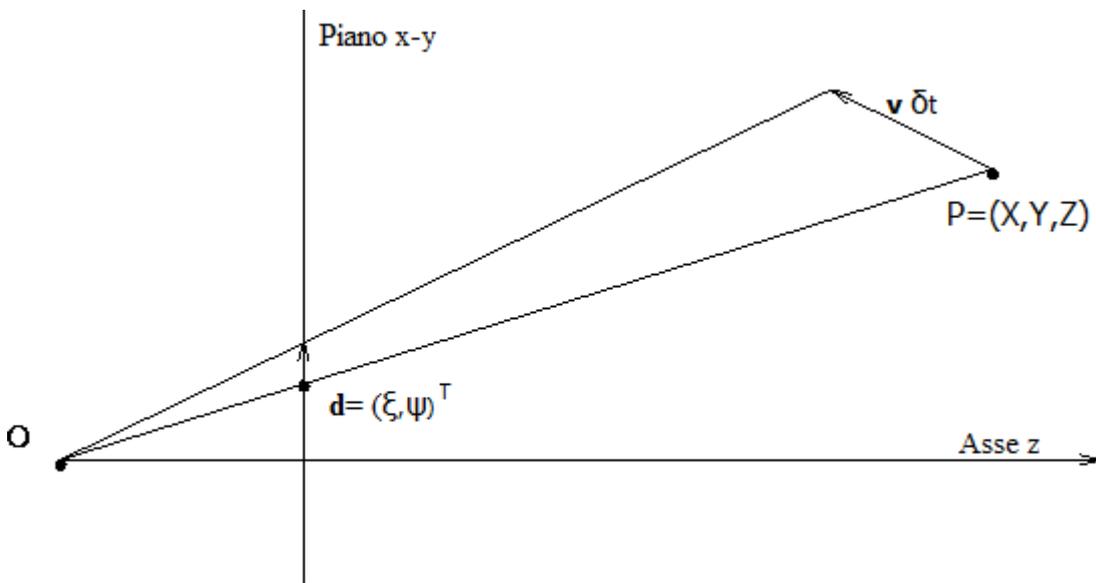


Figura 4: proiezione della velocità v nello spostamento d nel piano dell'immagine.

¹² $\mathbf{b} = -\sum_i e_i \mathbf{J}_1^T (x_i + \mathbf{u})$, $\mathbf{A} = \sum_i \mathbf{J}_1^T (x_i + \mathbf{u}) \mathbf{J}_1 (x_i + \mathbf{u})$, dove \mathbf{A} e \mathbf{b} , sono le componenti di un problema ai minimi quadrati: $\mathbf{A} \Delta \mathbf{u} = \mathbf{b}$.

Questo moto in 2D viene geometricamente definito come spostamento relativo del pixel p dal centro ottico del sistema O , assumendo la conoscenza del moto 3D. Lo spostamento visibile in 2D, è definito come l'Optical Flow $\mathbf{u} = (u, v)^T$, che inizia al pixel $p = (x, y)$ e termina al pixel $p' = (x+u, y+v)$, che spesso è diverso dall'attuale spostamento relativo.

Il calcolo dell'Optical Flow mira a stimare il valore del moto in 2D, come nell'esempio in *figura 5*. Un'analisi dell'algoritmo del *moto denso (dense motion)*, potrebbe ricavare dal flusso ottico verticale \mathbf{u} , l'effettivo moto 2D \mathbf{d} .

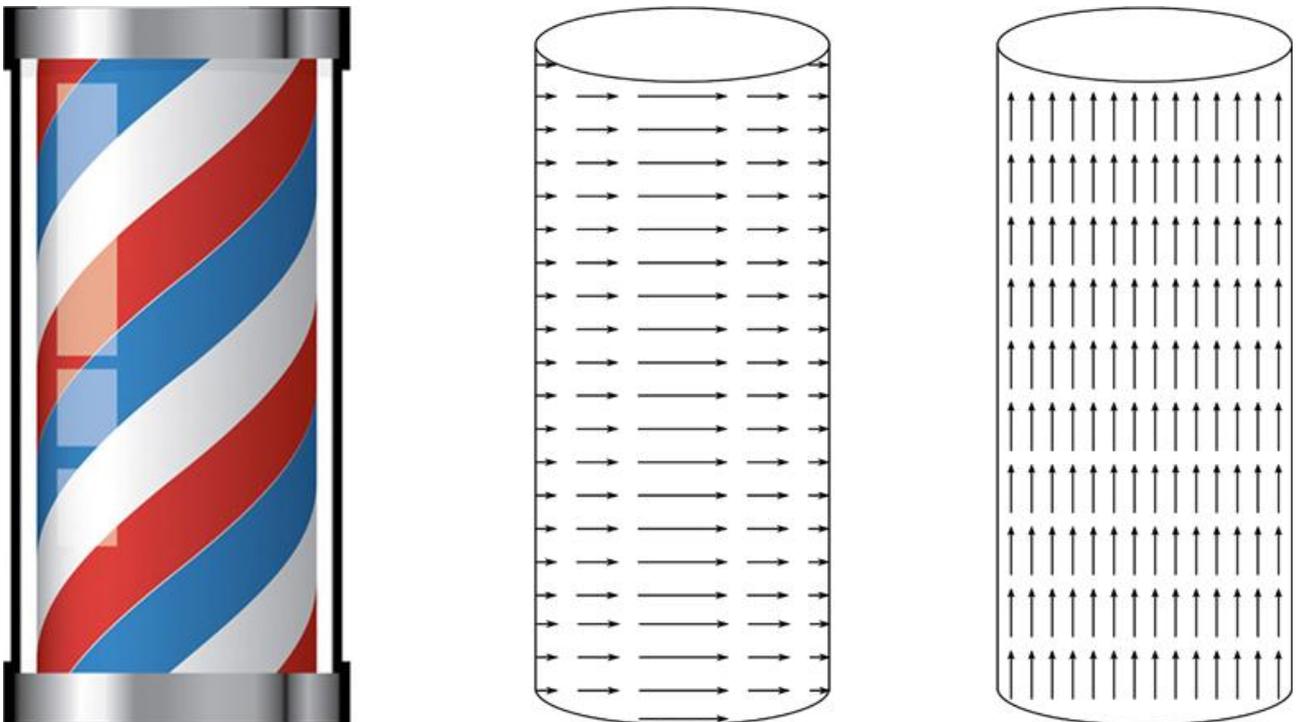


Figura 5: A sinistra abbiamo il classico palo del barbiere che si vede tipicamente nei film, dove abbiamo un moto 2D verso destra. Al centro abbiamo uno schematico del moto 2D (senza dimensionamento dei vettori). A destra infine abbiamo una bozza dell'Optical Flow, che tende a fluire verso l'alto.

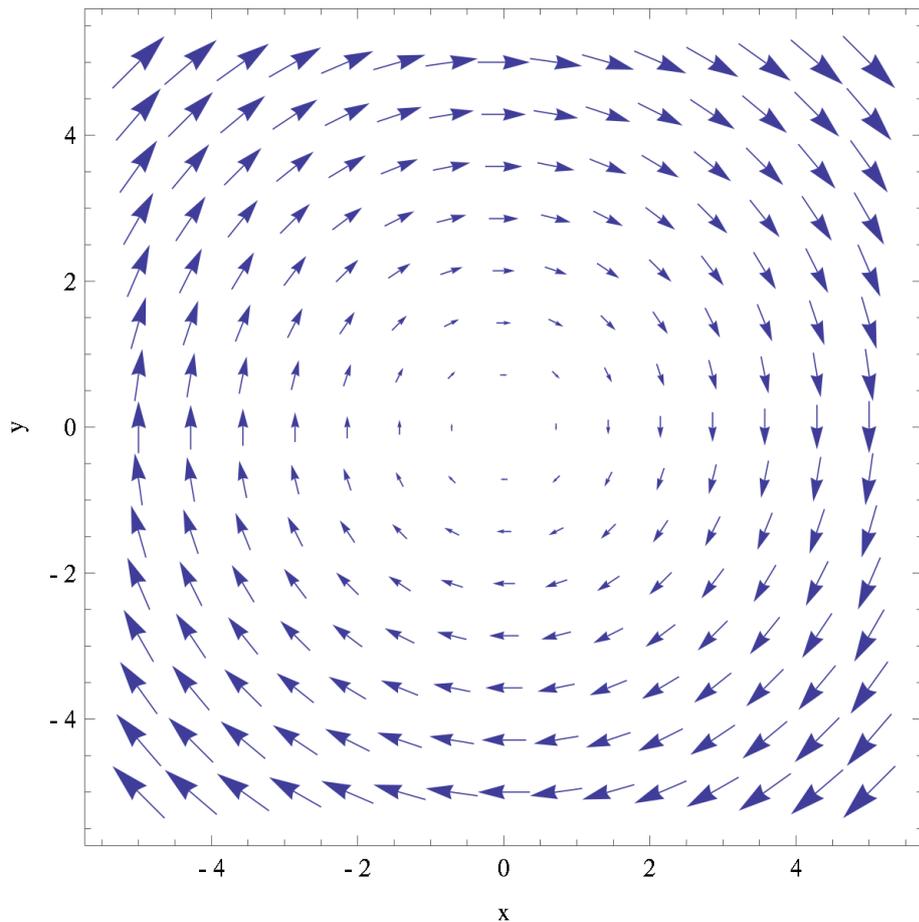


Figura 6: Campo vettoriale della rotazione di un quadrato.

Campo vettoriale: Per capire il moto in 3D, si può adoperare l'approccio dell'analisi delle proiezioni di vari punti $\mathbf{P}=(X, Y, Z)$ della superficie dell'oggetto. Il risultato consisterà in oggetti in movimento, oppure, con la sagoma di oggetti rigidi in movimento. I vari vettori del moto 2D formano dunque un campo vettoriale (nella *figura 6* si propone un esempio). Un corpo rigido semplifica l'analisi del campo vettoriale del moto 2D, ma anche in questo caso, la lettura dei vettori non è per nulla semplice. Un campo vettoriale di moto è detto

denso se contiene un vettore nelle vicinanze di ogni pixel, in caso contrario è detto *sparso*. Per dare un'idea intuitiva dei vettori di Optical Flow, in *figura 7*, le tonalità di colore rappresentano la direzione del movimento, mentre la saturazione, il modulo del vettore del moto.

Un esempio visivo lo si dà invece in *figura 8*, nella quale si è utilizzato l'algoritmo di Horn-Schunck, che verrà presentato nel terzo capitolo.

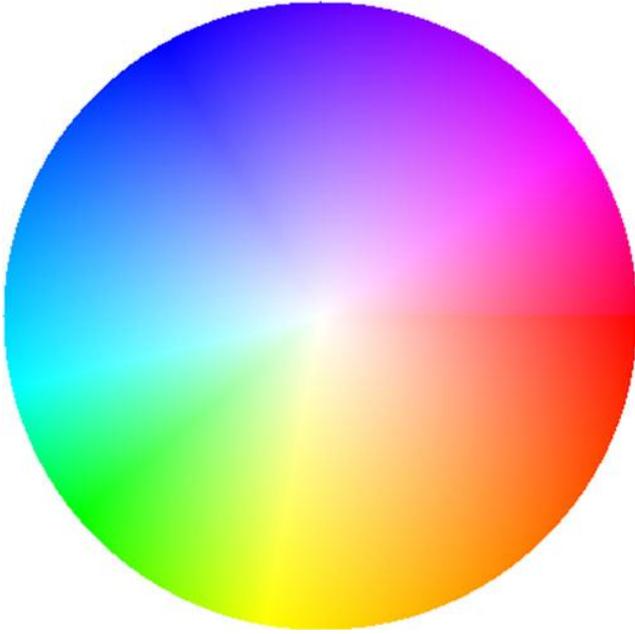


Figura 7: Una trasposizione in colori per individuare l'Optical Flow. I colori si presume partano dal centro del cerchio, e rappresentano le direzioni in cui si propaga il vettore. La saturazione corrisponde alla grandezza del vettore, dove si codifica col colore bianco uno stato di 'non movimento'.



Figura 8: A sinistra ed al centro 2 frame successivi di un video girato a 25fps. A destra la codifica dei vettori del moto coi colori, ottenuta utilizzando l'algoritmo Horn-Schunck.

2.6 Il problema dell'apertura ed il Gradiente di Flusso.

Dall'analisi delle proiezioni delle immagini, riscontriamo che la visibilità del moto risiede solo in un'area limitata, definita dall'apertura della fotocamera, oppure dall'algoritmo usato per l'analisi del moto.

Il problema dell'apertura: è causato dalla visuale limitata su una scena dinamica, ed aggiunge incertezze ulteriori alla stima del moto. Un esempio nella vita quotidiana di questo problema può essere dato dal treno. Quando siamo sul treno in stazione, e guardiamo dal finestrino, possiamo avere la sensazione che il treno si stia muovendo, mentre in realtà, è il treno affianco a muoversi. Nella *figura 9*, si vedono 3 frame successivi di un'auto, se guardassimo solo il rettangolo, avremo una sensazione di avanzamento dell'auto, in realtà la foto è stata presa mentre l'auto era in una rotonda, quindi stava svoltando a sinistra. Ovviamente, non si può avere completa conoscenza dell'informazione, però un incremento dell'apertura porta senza dubbio benefici per la comprensione del moto.

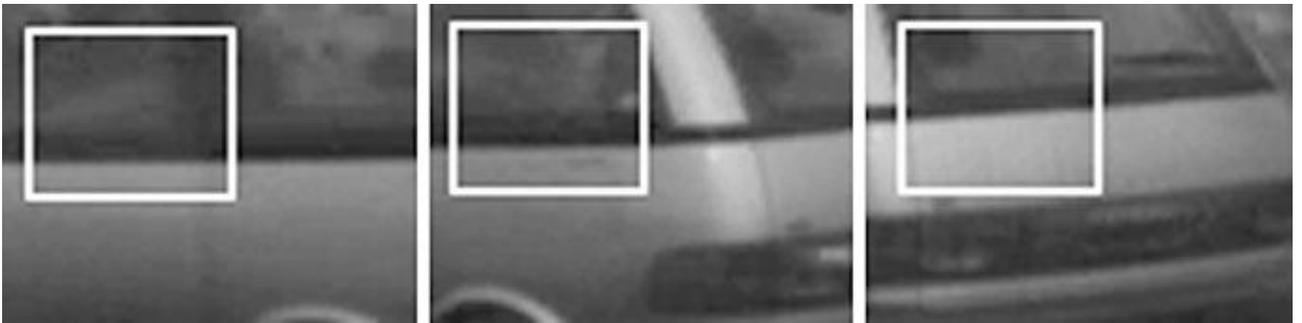


Figura 9: 3 immagini prese da un video, rispettivamente al tempo t , $t+1$ e $t+2$. Osservando il rettangolo interno concluderemmo che l'auto sta avanzando, in realtà l'auto sta svoltando a sinistra.

Gradiente di flusso: Il gradiente di flusso, o *gradient flow*, è la proiezione del reale moto in 2D nel gradiente del pixel dato. Questo flusso è ottenuto da metodi relativi, ed è una causa del problema dell'apertura. Il gradiente in 2D, rispetto alle coordinate x ed y è:

$$\nabla_{x,y} \mathbf{I} = [I_x(x, y, z), I_y(x, y, z)]^T \quad (24)$$

È ortogonale alla discontinuità dell'oggetto (uno spigolo) come si nota dalla *figura 10*, ipotizzando che l'intensità dell'immagine decresca dalla zona sulla sinistra a quella sulla destra. La I_x e la I_y della (24), sono le derivate parziali del frame $\mathbf{I} = (\cdot, \cdot, t)$, rispetto ad x ed y . Il risultato può essere migliorato dai valori dei pixel adiacenti.

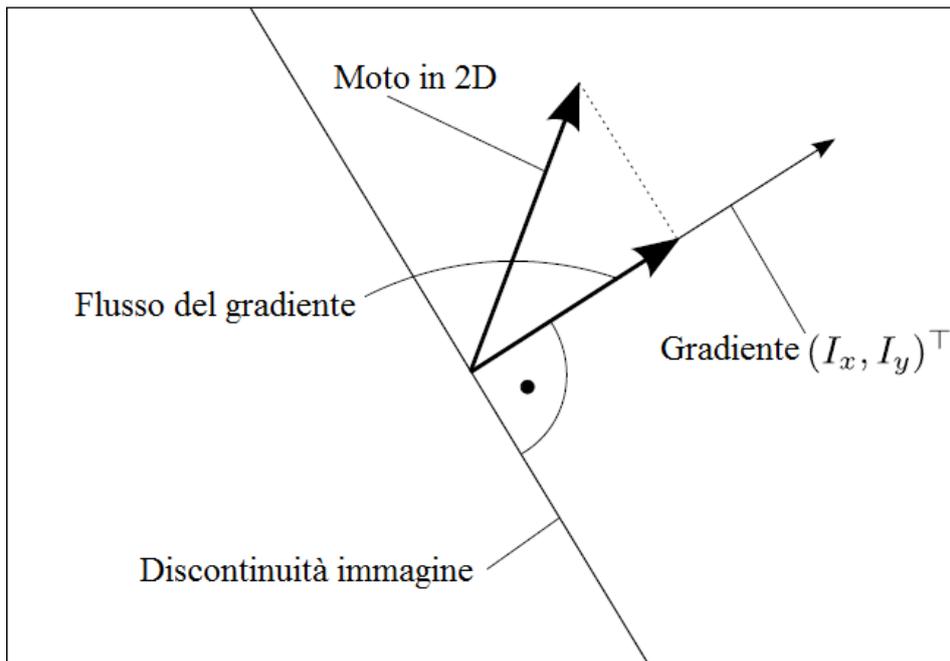


Figura 10: Gradiente di flusso. Il vero moto in 2D va verso l'alto con una certa angolazione, ma il moto che vediamo è dato dalla proiezione del moto vero, sul vettore del gradiente.

Capitolo 3

Algoritmi per la stima del moto con le tecniche di Optical Flow.

In questo capitolo introdurremo alcuni algoritmi per la stima del moto, sebbene alcuni possano essere usati in differenti tecniche, qui verranno trattati facendo riferimento solo ed esclusivamente alla tecnica dell'*Optical Flow*, introdotta nel capitolo precedente. Verranno affrontati tre algoritmi: l'algoritmo di *Horn-Schunck*, l'algoritmo di *Lucas-Kanade* ed infine l'algoritmo *BBPW*.

3.1 Algoritmo di *Horn-Schunck*

Ci piacerebbe definire una relazione tra i valori nei frame $I(\bullet, \bullet, t)$ e $I(\bullet, \bullet, t+1)$. Un buon inizio è quello di considerare lo sviluppo di Taylor del primo ordine della funzione $I(\bullet, \bullet, t+1)$.

3.1.1 Preparazione all'algoritmo

Avendo una funzione terziaria dobbiamo sviluppare $I(\bullet, \bullet, \bullet)$ applicando lo sviluppo di Taylor del primo ordine in 3D, ottenendo:

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \delta x \frac{\partial I}{\partial x}(x, y, t) + \delta y \frac{\partial I}{\partial y}(x, y, z) + \delta t \frac{\partial I}{\partial t}(x, y, t) + e \quad (1)$$

dove il termine e rappresenta tutti gli ordini dello sviluppo di Taylor dal secondo in poi. Come valore di δt prendiamo il lasso di tempo che intercorre tra due frame consecutivi $I(\bullet, \bullet, t)$ e $I(\bullet, \bullet, t+1)$. Con δx e δy vogliamo invece modellare il moto di un pixel, dall'istante di tempo t , in una certa posizione, all'istante di tempo $t+1$ in un'altra posizione. Utilizzando l'ipotesi di intensità costante (ICA)

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) \quad (2)$$

Prima e dopo il moto, abbiamo che

$$0 = \delta x \frac{\partial I}{\partial x}(x, y, t) + \delta y \frac{\partial I}{\partial y}(x, y, t) + \delta t \frac{\partial I}{\partial t}(x, y, t) \quad (3)$$

Dividendo tutto per δt otteniamo:

$$0 = \frac{\delta x}{\delta t} \frac{\partial I}{\partial x}(x, y, t) + \frac{\delta y}{\delta t} \frac{\partial I}{\partial y}(x, y, t) + \frac{\partial I}{\partial t}(x, y, t) \quad (4).$$

I cambiamenti delle coordinate x e y durante δt rappresentano l'*Optical flow* $\mathbf{u}(x, y, t) = (u(x, y, t), v(x, y, t))^T$. Ci interessa calcolare:

$$0 = u(x, y, t) \frac{\partial I}{\partial x}(x, y, t) + v(x, y, t) \frac{\partial I}{\partial y}(x, y, t) + \frac{\partial I}{\partial t}(x, y, t) \quad (5).$$

L'equazione (5) è conosciuta come *equazione del flusso ottico* o *vincolo di Horn-Schunck (HS)*. Un'osservazione è doveroso fare, l'equazione (5) deriva dal considerare solo piccoli istanti e dall'ipotesi di intensità costante. Logicamente questi presupposti definiranno il vincolo dell'algoritmo finale.

Il piano delle velocità uv : esprimendo la (5) in forma compatta, e tralasciando le coordinate (x, y, t) :

$$-I_t = \mathbf{u} \cdot \mathbf{I}_x + v \cdot \mathbf{I}_y = \mathbf{u} \cdot \nabla_{x,y} \mathbf{I} \quad (6)$$

\mathbf{I}_x e \mathbf{I}_y sono le derivate parziali rispettivamente rispetto a x e y , mentre \mathbf{I}_t è la derivata parziale rispetto al tempo. Lo scalare $\mathbf{u} \cdot \nabla_{x,y} \mathbf{I}$ è il prodotto scalare del vettore del flusso ottico per il vettore gradiente (nel quale non compare la derivata rispetto a t). Le tre derivate parziali sono valutate nei frame dati usando approssimazioni discrete delle derivate. Le componenti del flusso ottico u e v sono invece le incognite della (6). Quindi, questa equazione definisce una retta nel piano delle velocità uv (figura 11).

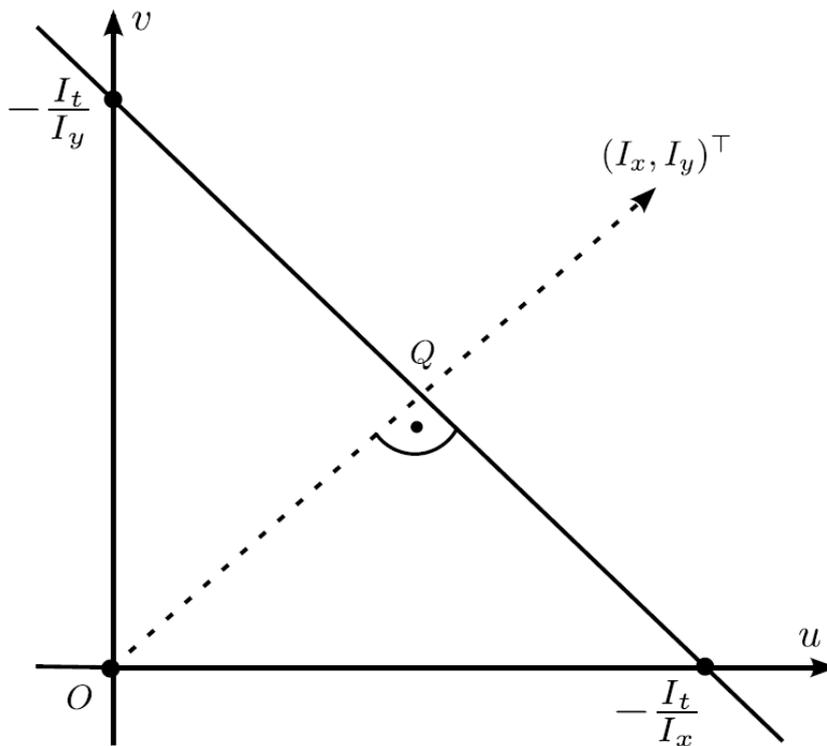


Figura 11: retta sul piano uv .

Calcolo dell'Optical flow come un problema di labelling. La computer vision di basso livello è spesso caratterizzata dal compito di assegnare ad ogni pixel un'etichetta (*label*) l da un set di possibili etichette L . Dobbiamo dunque utilizzare le etichette $(u,v) \in R^2$ in uno spazio 2D continuo. Le etichette vengono assegnate a tutti i pixel in Ω da una funzione di labelling

$$f: \Omega \rightarrow L \quad (7)$$

Risolvere un problema di labelling equivale ad identificare la funzione f che approssima in maniera ottimale.

Primo vincolo: sia f una funzione di labelling che assegna l'etichetta (u,v) ad ogni pixel $p \in \Omega$ in un'immagine $I(\bullet, \bullet, t)$. A causa della (6) siamo interessati in una soluzione di f che minimizzi gli errori nei dati:

$$E_{data}(f) = \sum_{\Omega} [u \cdot I_x + v I_y + I_t]^2 \quad (8)$$

con $u \cdot I_x + v I_y + I_t = 0$ nel caso ideale per pixel appartenenti a Ω nell'immagine $I(\bullet, \bullet, t)$.

Secondo vincolo: moto spaziale costante. Oltre alla (6) si è scelto di formulare un secondo vincolo per le soluzioni (u,v) nella speranza che conducano ad un'unica soluzione sulla retta $-I_t = u \cdot I_x + v \cdot I_y$. Si ipotizzi un moto costante tra i pixel vicini al tempo t , il che comporta che i pixel adiacenti in $I(\bullet, \bullet, t)$ hanno tutti gli stessi vettori di flusso ottico. Per una funzione che è pressoché costante nelle vicinanze, la sua derivata prima sarà vicina allo zero. Perciò, la costanza del moto può venir formulata come un vincolo di fluidità che minimizza, per tutti i pixel $p \in \Omega$ in $I(\bullet, \bullet, t)$, la somma dei quadrati delle derivate del primo ordine. Anche nella formula che seguirà si eviterà di appesantire la formulazione omettendo gli argomenti (x,y,t) . Sia f una funzione di labelling che assegna l'etichetta (u,v) ad ogni pixel $p \in \Omega$ in $I(\bullet, \bullet, t)$. L'errore di fluidità è definito come:

$$E_{smooth}(f) = \sum_{\Omega} \left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2 + \left(\frac{\partial v}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial y} \right)^2 \quad (9).$$

Il problema dell'ottimizzazione: si vorrebbe calcolare una funzione di labelling f che minimizzi l'errore totale:

$$E_{total}(f) = E_{data}(f) + \lambda E_{smooth}(f) \quad (10)$$

dove $\lambda > 0$ è un coefficiente di peso. In generale si usa $\lambda < 1$ per evitare un'eccessiva fluidità. Il parametro introdotto dipenderà dalle applicazioni e da altri dati importanti dell'immagine. Si cerca una f ottimale, nell'insieme di tutte le possibili funzioni di labelling, che definisca una variazione totale (TV). Un valore di $\lambda > 1$ darebbe alla fluidità un peso maggiore di quello concesso dal *vincolo di Horn-Schunck* (5), dunque non è una scelta consigliata. Un buon valore iniziale che viene spesso utilizzato è $\lambda = 0.1$. Per l'ottimizzazione si utilizza un metodo ai minimi quadrati. L'unico punto stazionario della (10) è un suo minimo. Le incognite sono tutte le u e v in tutte le $N_{colonne}$ ed N_{righe} delle posizioni dei pixel; il che implica il dover calcolare $2N_{colonne}N_{righe}$ derivate parziali della funzione ai minimi quadrati. Per alleggerire, di seguito, si ometterà t dalle formule. Per le considerazioni che seguiranno il frame $I(\bullet, \bullet, t)$, con i frame adiacenti $I(\bullet, \bullet, t+1)$ e $I(\bullet, \bullet, t-1)$, si intenderanno fissati. Per semplificazione si assumerà che ogni pixel abbia tutti i 4 pixel adiacenti nell'immagine. Approssimando le derivate nei termini dell'*errore di fluidità* usando le *differenze asimmetriche*, si ottiene:

$$E_{smooth}(f) = \sum_{x,y} (u_{x+1,y} - u_{xy})^2 + (u_{x,y+1} - u_{xy})^2 + (v_{x+1,y} - v_{xy})^2 + (v_{x,y+1} - v_{xy})^2 \quad (11)$$

Le differenze asimmetriche non sono la scelta migliore, le differenze simmetriche difatti avrebbero garantito risultati migliori, però l'intento rimane quello di presentare l'originale algoritmo di *Horn-Schunck*. Avendo le seguenti derivate parziali di $E_{data}(u, v)$:

$$\frac{\partial E_{smooth}}{\partial u_{xy}}(u, v) = 2[I_x(x, y)u_{xy} + I_y(x, y)v_{xy} + I_t(x, y)]I_x(x, y) \quad (12)$$

e

$$\frac{\partial E_{smooth}}{\partial v_{xy}}(u, v) = 2[I_x(x, y)u_{xy} + I_y(x, y)v_{xy} + I_t(x, y)]I_y(x, y) \quad (13)$$

Le derivate parziali di $E_{smooth}(u, v)$ valgono:

$$\begin{aligned} \frac{\partial E_{smooth}}{\partial u_{xy}}(u, v) = & -2[(u_{x+1,y} - u_{xy}) + (u_{x,y+1} - u_{xy})] + 2[(u_{xy} - u_{x-1,y}) + \\ & (u_{xy} - u_{x,y-1})] = 2[(u_{xy} - u_{x+1,y}) + (u_{xy} - u_{x,y+1}) + (u_{xy} - u_{x-1,y}) + \\ & (u_{xy} - u_{x,y-1})] \quad (14) \end{aligned}$$

Questi sono gli unici termini che contengono l'incognita u_{xy} . Semplificando si ottiene:

$$\frac{1}{4} \frac{\partial E_{smooth}}{\partial u_{xy}}(u, v) = 2 \left[u_{xy} - \left[\frac{1}{4} (u_{i+1,j} + u_{x,y+1} + u_{i-1,j} + u_{x,y-1}) \right] \right] \quad (15)$$

Usando \bar{u}_{xy} come valore medio dei quattro pixel adiacenti, si ottiene:

$$\frac{1}{4} \frac{\partial E_{smooth}}{\partial u_{xy}}(u, v) = 2[u_{xy} - \bar{u}_{xy}] \quad (16)$$

analogamente

$$\frac{1}{4} \frac{\partial E_{smooth}}{\partial v_{xy}}(u, v) = 2[v_{xy} - \bar{v}_{xy}] \quad (17)$$

Considerando tutto, ed usando λ anziché $\lambda/4$, dopo aver imposto le derivate uguali a zero, si giunge finalmente al sistema di equazione:

$$0 = \lambda[u_{xy} - \bar{u}_{xy}] + [I_x(x, y)u_{xy} + I_y(x, y)v_{xy} + I_t(x, y)]I_x(x, y) \quad (18)$$

$$0 = \lambda[v_{xy} - \bar{v}_{xy}] + [I_x(x, y)u_{xy} + I_y(x, y)v_{xy} + I_t(x, y)]I_y(x, y) \quad (19)$$

Questo è uno schema discreto per la minimizzazione. Questo è un sistema lineare con $2N_{colonne}N_{righe}$ incognite u_{xy} e v_{xy} .

Schema iterativo della soluzione: l'equazione del sistema appena trovato contiene i termini medi \bar{u}_{xy} e \bar{v}_{xy} , che son calcolati sulla base dei valori delle incognite. La dipendenza tra le incognite u_{xy} e v_{xy} ed i valori medi \bar{u}_{xy} e \bar{v}_{xy} contenuti nell'equazione è effettivamente un beneficio. Questo infatti, consente di definire uno schema iterativo partendo da alcuni valori iniziali. Lo schema è il seguente:

1. *Inizializzazione:* si inizializzano i valori u_{xy}^0 e v_{xy}^0 .
2. *Iterazione 0:* calcola i valori medi \bar{u}_{xy}^0 e \bar{v}_{xy}^0 usando i valori inizializzati nel punto precedente, inoltre calcola u_{xy}^1 e v_{xy}^1 .
3. *Iterazione n:* usando i valori u_{xy}^n e v_{xy}^n calcola \bar{u}_{xy}^n e \bar{v}_{xy}^n ; usando quei dati poi ricava u_{xy}^{n+1} e v_{xy}^{n+1} .

Si procede per $n \geq 1$ finché non viene soddisfatto un criterio per l'interruzione delle iterazioni. Le soluzioni del sistema (18) e (19) sono rispettivamente:

$$u_{xy}^{n+1} = \bar{u}_{xy}^n - I_x(x, y) \frac{I_x(x, y)\bar{u}_{xy}^n + I_y(x, y)\bar{v}_{xy}^n + I_t(x, y)}{\lambda^2 + I_x^2(x, y) + I_y^2(x, y)} \quad (20)$$

$$v_{xy}^{n+1} = \bar{v}_{xy}^n - I_y(x, y) \frac{I_x(x, y)\bar{u}_{xy}^n + I_y(x, y)\bar{v}_{xy}^n + I_t(x, y)}{\lambda^2 + I_x^2(x, y) + I_y^2(x, y)} \quad (21)$$

Detto ciò ora si può procedere con la discussione sull'algorithm.

3.1.2 Algoritmo

Le soluzioni date e lo schema di iterazioni consentono il calcolo dei valori di u_{xy}^n e v_{xy}^n al passo di iterazione n per tutte le posizioni dei pixel (x,y) nell'immagine $I(\bullet, \bullet, t)$. Per calcolare il valore di I_t serve almeno un'immagine adiacente, $I(\bullet, \bullet, t+1)$ oppure $I(\bullet, \bullet, t-1)$. Sia:

$$\alpha(x,y,n) = \frac{I_x(x,y)\bar{u}_{xy}^n + I_y(x,y)\bar{v}_{xy}^n + I_t(x,y)}{\lambda^2 + I_x^2(x,y) + I_y^2(x,y)} \quad (22)$$

L'algorithm è mostrato in figura 12.

```

1: for  $y = 1$  to  $N_{rows}$  do
2:   for  $x = 1$  to  $N_{cols}$  do
3:     Compute  $I_x(x, y)$ ,  $I_y(x, y)$ , and  $I_t(x, y)$  ;
4:     Initialize  $u(x, y)$  and  $v(x, y)$  (in even arrays);
5:   end for
6: end for
7: Select weight factor  $\lambda$ ; select  $T > 1$ ; set  $n = 1$ ;
8: while  $n \leq T$  do
9:   for  $y = 1$  to  $N_{rows}$  do
10:    for  $x = 1$  to  $N_{cols}$  {in alternation for even or odd arrays} do
11:      Compute  $\alpha(x, y, n)$ ;
12:      Compute  $u(x, y) = \bar{u} - \alpha(x, y, n) \cdot I_x(x, y, t)$  ;
13:      Compute  $v(x, y) = \bar{v} - \alpha(x, y, n) \cdot I_y(x, y, t)$  ;
14:    end for
15:  end for
16:   $n := n + 1$ ;
17: end while

```

Figura 12: Algoritmo di Horn-Schunck

\bar{u} e \bar{v} sono i valori medi dei quattro pixel adiacenti. Si utilizzano vettori “pari” e “dispari” per i valori di u e v . Ad esempio, nell'iterazione $n=1$, si calcolano i valori in un vettore “dispari”, ad $n=2$ in un vettore “pari” e così via, alternando i due vettori. La soglia T serve per il criterio d'interruzione, e corrisponde con il numero massimo di iterazioni possibili. L'inizializzazione col valore zero per tutte le posizioni di u_{xy} e v_{xy} fu consigliato

nell'algoritmo originale di *Horn-Schunck*. Questo consente di avere valori di u_{xy}^1 e v_{xy}^1 diversi da zero, finché $I_x(x, y)I_t(x, y)$ e $I_y(x, y)I_t(x, y)$ non sono diversi da zero in tutte le locazioni di pixel. L'algoritmo originale usava per I_x , I_y e I_t le seguenti approssimazioni asimmetriche:

$$I_x(x, y, t) = \frac{1}{4}[I(x + 1, y, t) + I(x + 1, y, t + 1) + I(x + 1, y + 1, t) + I(x + 1, y + 1, t + 1)] - \frac{1}{4}[I(x, y, t) + I(x, y, t + 1) + I(x, y + 1, t) + I(x, y + 1, t + 1)] \quad (23)$$

$$I_y(x, y, t) = \frac{1}{4}[I(x, y + 1, t) + I(x, y + 1, t + 1) + I(x + 1, y + 1, t) + I(x + 1, y + 1, t + 1)] - \frac{1}{4}[I(x, y, t) + I(x, y, t + 1) + I(x + 1, y, t) + I(x + 1, y, t + 1)] \quad (24)$$

$$I_t(x, y, t) = \frac{1}{4}[I(x, y, t + 1) + I(x, y + 1, t + 1) + I(x + 1, y, t + 1) + I(x + 1, y + 1, t + 1)] - \frac{1}{4}[I(x, y, t) + I(x, y + 1, t) + I(x + 1, y, t) + I(x + 1, y + 1, t)] \quad (25)$$

L'algoritmo richiede solo una coppia di immagini successive come input. Ci sono però molte alternative per modificare l'algoritmo. In figura 13 si mostra un esempio di applicazione dell'algoritmo originale di *Horn-Schunck*.

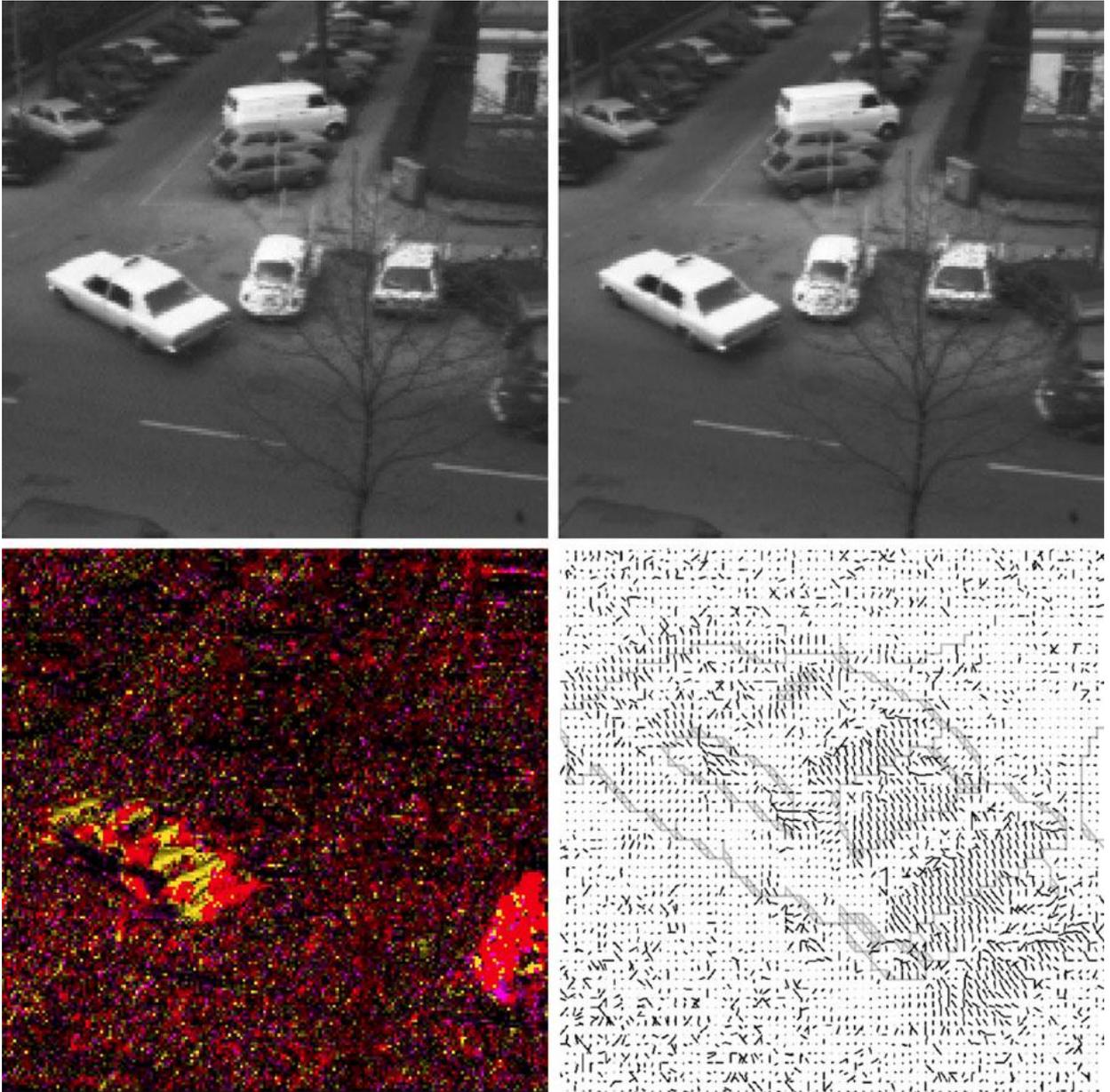


Figura 13: In alto: due frame successive della "sequenza del taxi di Hamburg", pubblicato nel 1983 per testare gli algoritmi sul flusso ottico. In basso, a sinistra: si utilizza la tecnica di visualizzazione coi colori (introdotta nel capitolo precedente, ma in chiave differente) per visualizzare direzione e modulo dei vettori di flusso ottico calcolati. In basso, a destra: illustrazione dei vettori di flusso (conosciuta come needle map, letteralmente mappa d'aghi) in una porzione dell'immagine.

3.2 Algoritmo di *Lucas-Kanade*

L'equazione del flusso ottico definisce una retta $u \cdot I_x + v I_y + I_t = 0$, nel piano delle velocità u e v , per ogni pixel $p \in \Omega$. Si considerino tutte le rette definite da tutti i pixel nelle vicinanze. Si assuma dunque che non siano parallele, bensì definite da un moto 2D simile, perciò si intersecheranno vicino al vero moto 2D (figura 14). Questo algoritmo ha una

trattazione matematica molto più semplice rispetto all'algoritmo di *Horn-Schunck*. Serve però per mostrare una valida alternativa per la rilevazione del moto.

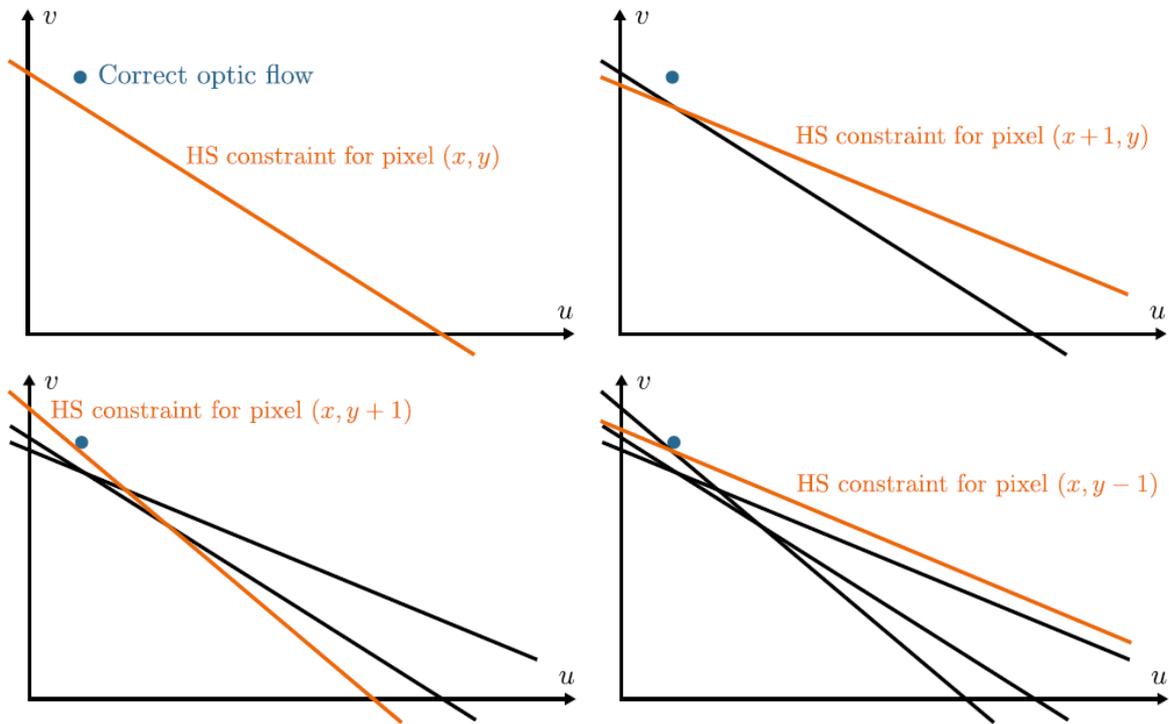


Figura 14: prendendo sempre più pixel nelle vicinanze ed analizzando le intersezioni delle rette, è possibile ricostruire l'andamento della retta del reale flusso ottico.

3.2.1 Soluzione lineare ai minimi quadrati

L'algoritmo per il flusso ottico di *Lucas-Kanade* utilizza un metodo lineare ai minimi quadrati per analizzare le intersezioni di più di due rette. Questo metodo è utilizzato quando il sistema è *sovradeterminato*¹³. Inizialmente si analizzano solo due rette, una passante per il pixel in posizione $p1$ e l'altra che passa per il pixel nella posizione adiacente $p2$. Si assuma che il moto 2D in entrambi i pixel adiacenti sia identico ed uguale ad \mathbf{u} . Inoltre, si presumano due differenti gradienti unitari \mathbf{g}_1° e \mathbf{g}_2° per i due pixel. Sia in aggiunta $\mathbf{u}^T \cdot \mathbf{g}^\circ = -\frac{I_t}{\|\mathbf{g}\|_2}$ ¹⁴ l'equazione del flusso ottico per entrambi i pixel.

Si ottengono quindi due equazioni in due incognite, u e v , per $\mathbf{u} = (u, v)^T$:

¹³ Per la definizione si rimanda al primo capitolo.

¹⁴ $\|\mathbf{x}\|_2$ indica la norma 2, ovvero la norma che calcola il modulo del vettore.

$$\mathbf{u}^T \cdot \mathbf{g}_1^\circ(p1) = -\frac{I_t}{\|\mathbf{g}\|_2}(p1) \quad (26)$$

$$\mathbf{u}^T \cdot \mathbf{g}_2^\circ(p2) = -\frac{I_t}{\|\mathbf{g}\|_2}(p2) \quad (27)$$

Usando b_i al secondo membro, possiamo scriverle in forma di sistema lineare con due equazioni e due incognite:

$$u \cdot g_{x1} + v \cdot g_{y1} = b_1 \quad (28)$$

$$u \cdot g_{x2} + v \cdot g_{y2} = b_2 \quad (29)$$

Nei vettori unitari $\mathbf{g}_1^\circ = [g_{x1}, g_{y1}]^T$ e $\mathbf{g}_2^\circ = [g_{x2}, g_{y2}]^T$. Ponendo tutto in forma matriciale:

$$\begin{bmatrix} g_{x1} & g_{y1} \\ g_{x2} & g_{y2} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad (30)$$

È possibile risolvere il sistema se la matrice sulla sinistra è invertibile:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} g_{x1} & g_{y1} \\ g_{x2} & g_{y2} \end{bmatrix}^{-1} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad (31)$$

Questo è il caso in cui non abbiamo un sistema sovradeterminato. Si trova infine l'intersezione fra le due rette, figura 15. Ci sono errori quando si calcolano I_x , I_y e I_t , inoltre i dati dell'immagine sono sempre soggetti a rumore, seppur minimo, quindi è meglio risolvere \mathbf{u} utilizzando il metodo dei minimi quadrati, assegnando una vicinanza di pixel $k > 2$. Il sistema ora risulterà sovradeterminato. Il valore k non deve essere preso troppo grande, poiché una delle assunzioni iniziali era che i pixel avessero lo stesso moto in 2D \mathbf{u} . Si giunge dunque ad un sistema:

$$\begin{bmatrix} g_{x1} & g_{y1} \\ g_{x2} & g_{y2} \\ \vdots & \vdots \\ g_{xk} & g_{yk} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} \quad (32)$$

scrivendolo come:

$$\mathbf{G} \cdot \mathbf{u} = \mathbf{B} \quad (33)$$

con \mathbf{G} di dimensioni $k \times 2$, \mathbf{u} di dimensioni 2×1 e \mathbf{B} di dimensioni $k \times 1$, per $k > 2$ il sistema può essere risolto ai minimi quadrati. Per farlo prima bisogna renderlo 'quadrato':

$$\mathbf{G}^T \cdot \mathbf{G} \cdot \mathbf{u} = \mathbf{G}^T \cdot \mathbf{B} \quad (34)$$

Infine risolverlo come errore ai minimi quadrati:

$$\mathbf{u} = (\mathbf{G}^T \mathbf{G})^{-1} \mathbf{G}^T \mathbf{B} \quad (35)$$

Una volta fatto, $\mathbf{G}^T \mathbf{G}$ sarà una matrice quadrata di dimensioni 2x2, mentre $\mathbf{G}^T \mathbf{B}$ sarà una matrice rettangolare 2x1. Ad esempio, ponendo:

$$\mathbf{G}^T \mathbf{G} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad (36)$$

avremo

$$(\mathbf{G}^T \mathbf{G})^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \quad (37)$$

Il resto è semplicemente prodotto matriciale.

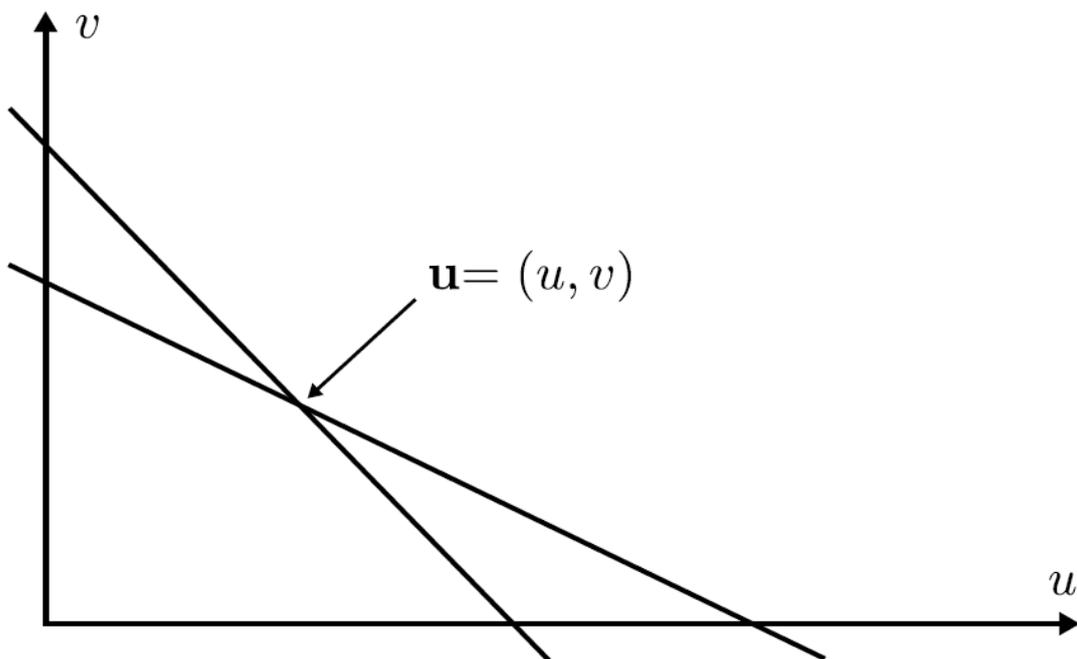


Figura 15: caso semplice considerando solo due pixel

3.2.2 Algoritmo originale ed algoritmo pesato

Algoritmo originale: l'algoritmo originale è il seguente:

1. Decidere le dimensioni di k ed applicarlo uniformemente in ogni frame.

2. Al frame t , calcolare I_x , I_y e I_t .
3. Per ogni locazione di pixel p nel frame t , ottenere il sistema (32) e risolverlo ai minimi quadrati, come definito nella (35).

Algoritmo pesato: si associ ω_i con $i=1,\dots,k$, come peso per tutti i k pixel che contribuiscono alla stima del moto. Tipicamente il pixel attuale avrà il peso massimo, mentre tutti i pixel adiacenti (contribuendo a quei k pixel) avranno pesi più piccoli. Sia $\mathbf{W}=\text{diag}[\omega_1, \dots, \omega_k]$ la matrice diagonale $k \times k$ di quei pesi. Una matrice diagonale soddisfa la relazione:

$$\mathbf{W}^T \mathbf{W} = \mathbf{W} \mathbf{W} = \mathbf{W}^2 \quad (38)$$

L'obiettivo ora è quello di risolvere il sistema:

$$\mathbf{W} \mathbf{G} \mathbf{u} = \mathbf{W} \mathbf{B} \quad (39)$$

invece del sistema (33) mostrato in precedenza. Anche il secondo membro della (39) deve essere “pesato” come il primo. La soluzione si ottiene manipolando le matrici come segue:

$$(\mathbf{W} \mathbf{G})^T \mathbf{W} \mathbf{G} \mathbf{u} = (\mathbf{W} \mathbf{G})^T \mathbf{W} \mathbf{B} \quad (40)$$

$$\mathbf{G}^T \mathbf{W}^T \mathbf{W} \mathbf{G} \mathbf{u} = \mathbf{G}^T \mathbf{W}^T \mathbf{W} \mathbf{B} \quad (41)$$

$$\mathbf{G}^T \mathbf{W} \mathbf{W} \mathbf{G} \mathbf{u} = \mathbf{G}^T \mathbf{W} \mathbf{W} \mathbf{B} \quad (42)$$

$$\mathbf{G}^T \mathbf{W}^2 \mathbf{G} \mathbf{u} = \mathbf{G}^T \mathbf{W}^2 \mathbf{B} \quad (43)$$

Ottenendo infine:

$$\mathbf{u} = [\mathbf{G}^T \mathbf{W}^2 \mathbf{G}]^{-1} \mathbf{G}^T \mathbf{W}^2 \mathbf{B} \quad (44)$$

Il significato di queste trasformazioni è di avere una matrice $\mathbf{G}^T \mathbf{W}^2 \mathbf{G}$ di dimensioni 2×2 , per la quale si può utilizzare l'inversa al fine di calcolare il valore di \mathbf{u} attuale. Paragonando l'algoritmo pesato all'originale bisogna aggiornare il punto 3 così:

3. Per ogni locazione di pixel p nel frame t , ottenere il sistema (39) e risolverlo ai minimi quadrati, come definito nella (44).

In figura 6 si mostrano i risultati dell'algoritmo originale di *Lucas-Kanade* per un insieme 5×5 (quindi $k=25$). Quando si risolve il sistema (35) le soluzioni sono considerate accettabili solo nei casi in cui gli autovalori della matrice $\mathbf{G}^T \mathbf{G}$ siano maggiori di un certo

valore di soglia. Facendo ciò si filtreranno i risultati “rumorosi”. La matrice $\mathbf{G}^T \mathbf{G}$ è una matrice 2×2 simmetrica e positiva. Ha due autovalori λ_1 e λ_2 . Sia $0 \leq \lambda_1 \leq \lambda_2$. Ipotizziamo una soglia $T > 0$. Saranno prese in considerazione le soluzioni della (35) solo se sarà verificato che $\lambda_1 \geq T$. Questo porta ad un campo di flusso sparso, come mostrato in figura 16. Si noti come quei pochi vettori mostrati siano vicini al vero spostamento generale.



Figura 16: flusso ottico calcolato con l'algorithmo originale di Lucas-Kanade, con $k=25$ per un insieme 5×5 .

3.3 Algoritmo BBPW

L'algoritmo BBPW estende l'algoritmo di *Horn-Schunck*, in un tentativo di migliorare i risultati, soprattutto per quanto riguarda grandi spostamenti, cercando anche di superare le limitazioni date dall'ipotesi di intensità costante (ICA). Per di più, l'uso dei quadrati nell'ottimizzazione permette ai valori anomali di avere un impatto significativo. Si usano

dunque degli approcci di ottimizzazione L1, in alternativa a quelli di tipo L2 utilizzati per l'algoritmo di *Horn-Schunck*.

3.3.1 Ipotesi usate e funzioni di energia

L'ipotesi di intensità costante viene rappresentata dal contesto considerato come

$I(x, y, t) = I(x+u, y+v, t+1)$, con u e v che rappresentano gli spostamenti rispettivamente nelle direzioni x ed y , durante un intervallo di tempo δt . Linearizzandolo si ottiene il vincolo di *Horn-Schunck* $u \cdot I_x + v I_y + I_t = 0$. L'intensità costante non vale per scenari esterni; di fatti gli algoritmi visti finora spesso lavorano male per cambiamenti di pixel, ad esempio, passando da 5 a 10 pixel.

Costanza del gradiente: l'ipotesi di costanza dell'intensità dei gradienti (GCA) sugli spostamenti è rappresentata come:

$$\nabla_{x,y} I(x, y, t) = \nabla_{x,y} I(x + u, y + v, t + 1) \quad (45)$$

dove $\nabla_{x,y}$ è limitato, come in precedenza, alle derivate rispetto ad x e y , ma non a t . L'informazione del gradiente viene considerata abbastanza robusta contro i cambi d'intensità.

Ipotesi di fluidità: usando solo la (ICA) e (GCA) non si forniscono informazioni sufficienti per identificare univocamente il flusso ottico; si devono perciò coinvolgere i pixel adiacenti ed introdurre inoltre della consistenza nel campo del flusso ottico calcolato. Bisogna prestare attenzione ai confini degli oggetti. I moti discontinui non saranno completamente eseguiti. La fluidità a tratti potrebbe essere una valida opzione per evitare vettori di flusso dall'esterno, in breve, un oggetto influenza il flusso entro l'oggetto stesso e viceversa.

Un primo abbozzo per una formula dell'errore: per la funzione dell'errore (o dell'energia) si continuerà ad usare l'ipotesi (ICA) (ma senza inoltrarsi nel vincolo di *Horn-Schunck*), combinandola però con la GCA. Si considerino le locazioni di pixel (x, y, t) nel frame $I(\bullet, \bullet, t)$, nel vettore di flusso ottico $\mathbf{w}=(u, v, 1)^T$, che contiene una terza componente per andare dal piano dell'immagine al tempo t , in quella dell'immagine al tempo $t+1$, e per usare il gradiente nello spazio 3D $\nabla=(\partial x, \partial y, \partial t)$. L'equazione (8) diventa:

$$E_{data}(f) = \sum_{\Omega} ([I(x+u, y+v, t+1) - I(x, y, t)]^2 + \lambda_1 [\nabla_{x,y} I(x+u, y+v, t+1) - \nabla_{x,y} I(x, y, t)]^2) \quad (46)$$

dove $\lambda_1 > 0$ è un peso che deve essere specificato. Per tutti i termini di fluidità, fondamentalmente, ci si trova nella (9), ma per un gradiente spazio-temporale nello spazio 3D, la (9) diventa:

$$E_{smooth}(f) = \sum_{\Omega} \|\nabla_u\|_2^2 + \|\nabla_v\|_2^2 \quad (47)$$

Quindi, alla fine, l'obiettivo sarà quello di identificare una funzione di labelling che assegni il flusso ottico u e v ad ogni pixel nell'immagine $I(\bullet, \bullet, t)$ così che la somma

$$E_{total}(f) = E_{data}(f) + \lambda_2 E_{smooth}(f) \quad (48)$$

sia minimizzata per un peso $\lambda_2 > 0$.

Regolarizzazione L_1 di variazione totale: un allontanamento dall'ottimizzazione L_2 rispetto alla L_1 è utilizzare la formula:

$$\Psi(s^2) = \sqrt{s^2 + \varepsilon} \approx |s| \quad (49)$$

utilizzando anziché $|s|$ la funzione dell'errore. Questo porta ad una funzione dell'errore più robusta, dove si può considerare continua la derivata per $s=0$, persino per ε piccoli, purché siano positivi. La costante ε ad esempio potrebbe assumere il valore di 10^{-6} . Non ci sono derivate continue di $|s|$ per $s=0$, ma servono per poter applicare lo schema di minimizzazione dell'errore. La funzione (49) è una funzione concava crescente, ed è applicata nelle (46) e (47) per ridurre l'influenza dei valori anomali. Questo definisce un termini di regolarizzazione di variazione totale in L_1 . Si ottengono dunque:

$$E_{data}(f) = \sum_{\Omega} \Psi([I(x+u, y+v, t+1) - I(x, y, t)]^2 + \lambda_1 [\nabla_{x,y} I(x+u, y+v, t+1) - \nabla_{x,y} I(x, y, t)]^2) \quad (50)$$

e

$$E_{smooth}(f) = \sum_{\Omega} \Psi((\nabla_u)^2 + (\nabla_v)^2) \quad (51)$$

La formula per ottenere l'errore totale rimane invariata.

3.3.2 Delineazione dell'algoritmo

Bisogna trovare una funzione di labelling f che minimizzi la funzione dell'errore totale (48), usando i termini d'errore introdotti nelle (50) e (51). Minimizzare una funzione non lineare come la (48) non è per nulla semplice. Quando si cerca un minimo assoluto, un algoritmo di minimizzazione potrebbe rimanere bloccato in un minimo relativo.

Equazioni di Eulero-Lagrange: sia Ψ' la derivata rispetto al suo argomento di Ψ . La divergenza div denota la somma di derivate, in questo caso la somma di tre derivate parziali, componenti di:

$$\nabla \mathbf{u} = \left[\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial t} \right]^T \quad (52)$$

Una funzione di labelling f che minimizza la (48) deve soddisfare le seguenti equazioni di Eulero-Lagrange. Le equazioni risultanti sono:

$$\Psi'(I_t^2 + \lambda_1(I_{xt}^2 + I_{yt}^2))(I_x I_t + \lambda_1(I_{xx} I_{xt} + I_{xy} I_{yt})) - \lambda_2 div(\Psi'(\|\nabla u\|^2 + \|\nabla v\|^2) \nabla u) = 0 \quad (53)$$

$$\Psi'(I_t^2 + \lambda_1(I_{xt}^2 + I_{yt}^2))(I_y I_t + \lambda_1(I_{yy} I_{yt} + I_{xy} I_{xt})) - \lambda_2 div(\Psi'(\|\nabla u\|^2 + \|\nabla v\|^2) \nabla v) = 0 \quad (54)$$

Come sempre I_x , I_y e I_t sono le derivate rispetto alle coordinate del pixel o del tempo, I_{xx} , I_{xy} e simili, sono invece le derivate seconde. Tutte quelle derivate possono considerarsi costanti, per essere calcolate con approssimazioni nella sequenza di frame. Quindi, entrambe le equazioni sono del tipo:

$$c_1 - \lambda_2 div(\Psi'(\|\nabla u\|^2 + \|\nabla v\|^2) \nabla u) = 0 \quad (55)$$

$$c_2 - \lambda_2 div(\Psi'(\|\nabla u\|^2 + \|\nabla v\|^2) \nabla v) = 0 \quad (56)$$

La soluzione di queste equazioni per u e v per ogni pixel $p \in \Omega$ possono essere supportate utilizzando un approccio piramidale.

Algoritmo piramidale: un approccio efficiente dell'algoritmo è quello di usare copie ricampionate dei frame processati in un'immagine piramidale, per poi calcolare i vettori di flusso. Successivamente si usano i risultati ottenuti in copie ad alta risoluzione per ottenere ulteriori miglioramenti. Questo algoritmo supporta anche l'identificazione di vettori di flusso lunghi. In figura 17 viene mostrato un esempio dell'algoritmo piramidale che implementa l'algoritmo BBPW. La chiave dei colori è la stessa della figura 7 del capitolo precedente.

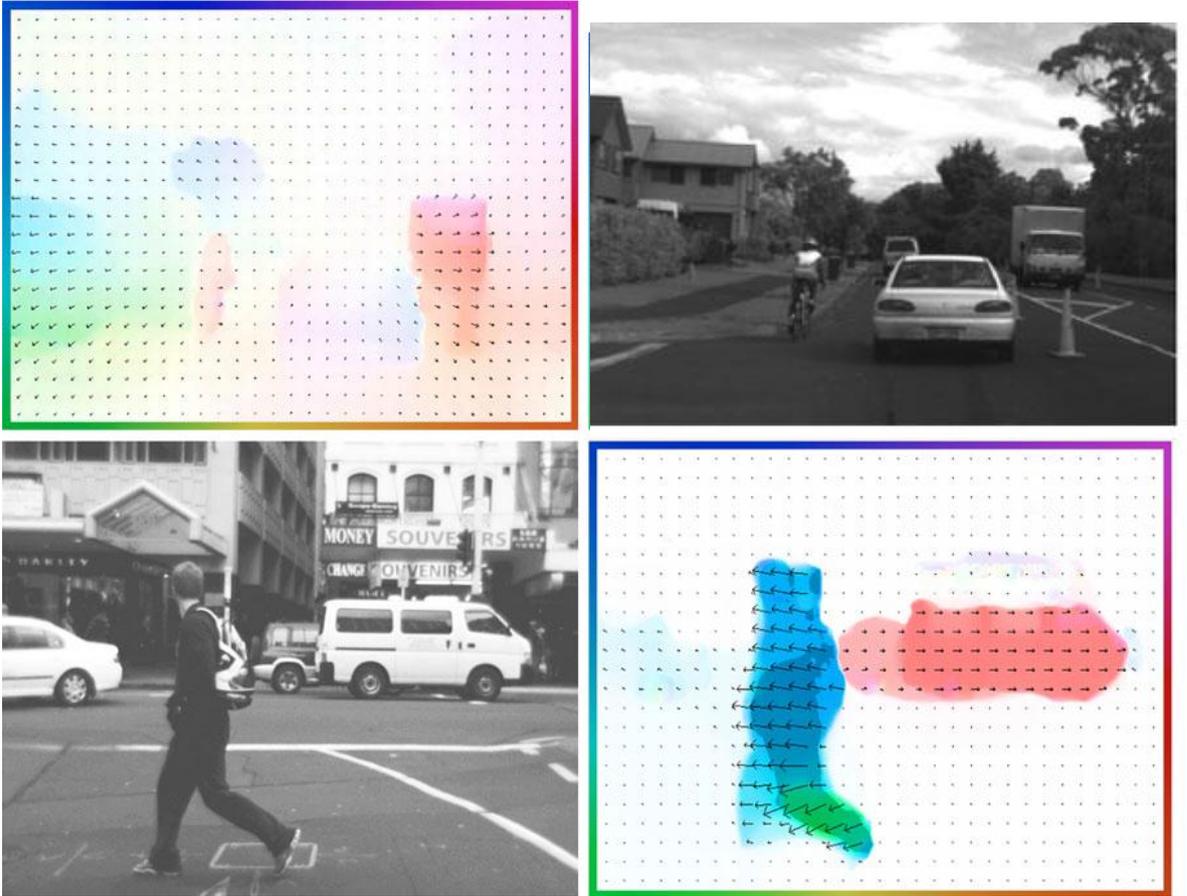


Figura 17: calcolo del flusso ottico con l' algoritmo BBPW

Capitolo 4

Valutazione performance ed applicazioni su Matlab.

L' algoritmo di *Horn-Schunck* fu il pioniere degli algoritmi per il calcolo del flusso ottico; molti altri algoritmi sono stati sviluppati ad oggi. L' analisi del moto è infatti un campo fertile della *Computer Vision*. In questo capitolo prima si discuterà brevemente delle performance delle tecniche per il calcolo del flusso ottico, poi si illustreranno alcune applicazioni del flusso ottico su *Matlab*, sia attraverso un modello Simulink, sia attraverso l' implementazione di alcuni algoritmi visti finora. Le applicazioni su Matlab, ed i relativi risultati, inseriti in questo paragrafo sono state reperite nel sito del software *mathwork.com*.

4.1 Strategie dei test

L' algoritmo di *Horn-Schunck* generalmente si allontana da un gradiente di flusso. L' esempio riportato in seguito però mostra che ciò non capita se la vicinanza di pixel non ammette il calcolo di un moto corretto.

Esempio: si assuma un' immagine 16×16 I_1 (come mostrato in *figura 18*), con $G_{max} = 7$. L' immagine contiene un bordo lineare verticale; il gradiente nel punto dei pixel del bordo punta verso sinistra. L' immagine I_2 viene generata da uno dei tre moti abbozzati sulla sinistra:

- (A) È una traslazione di un pixel (0,1) a destra;
- (B) È una traslazione diagonale (1,1);
- (C) È una traslazione di un pixel (1,1) in alto.

Per semplicità si considereranno i valori addizionali sulla sinistra uguali a zero, mentre quelli sul fondo uguali ai valori della colonna data. Questo semplice esempio permette di calcolare manualmente i valori prodotti dall' algoritmo di *Horn-Schunck*.

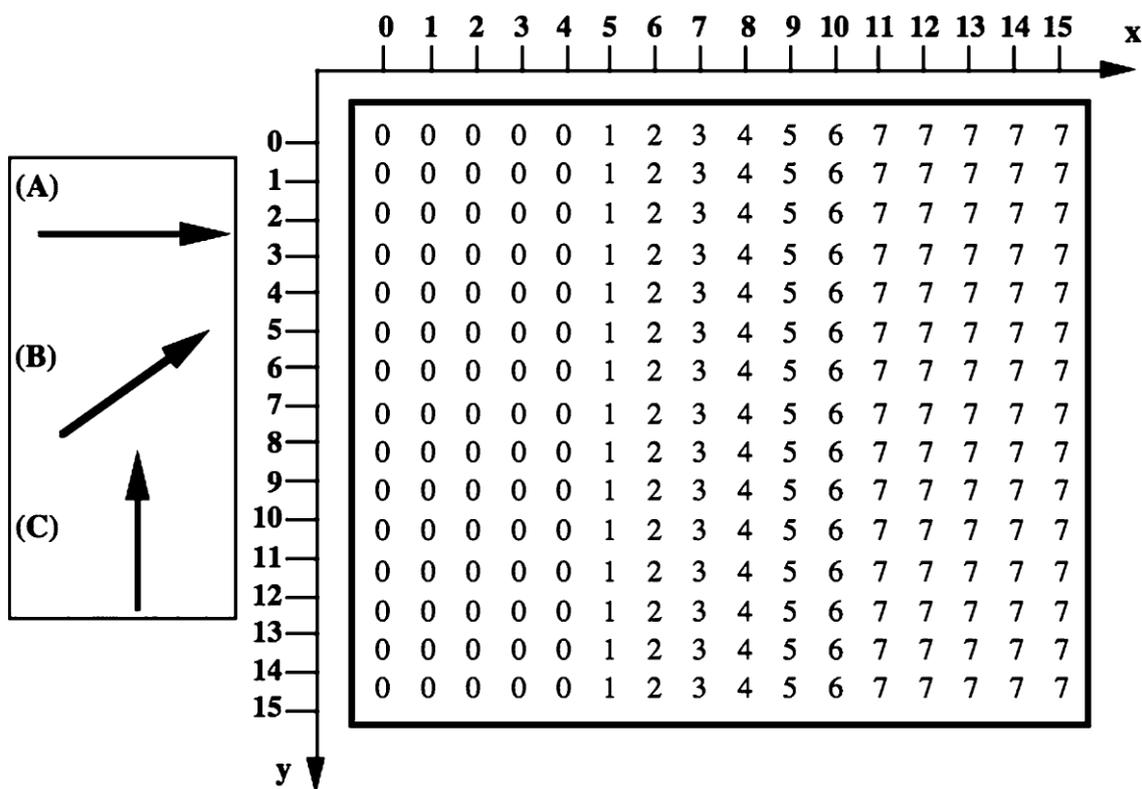


Figura 18: un input semplice per l'iterazione dell' algoritmo di Horn-Schunck in maniera manuale.

Valutazione delle performance per gli algoritmi per il flusso ottico: per valutare un algoritmo per il flusso ottico in una sequenza di immagini reali ci sono diverse opzioni:

- 1) Ipotizzando che la registrazione delle immagini avvenga ad alta velocità (almeno 100Hz), è possibile usare *frame* alternativi per l'analisi di predizione dell'errore. Stimato il flusso ottico per il *frame t* ed il *frame t+2*, calcolare l'immagine virtuale per *t+1* interpolando lungo i vettori calcolati e comparando l'immagine virtuale con il *frame t+1*. Un modo per comparare i due frame è quello di utilizzare la *cross-correlazione normalizzata*.¹⁵
- 2) Se la realtà di base è disponibile sull'attuale moto 2D, la valutazione del flusso ottico potrebbe essere affetta da errori di misura. Questi errori sono generati dalla comparazione dei veri vettori (permettendo errori di misura del modulo quando si genera la realtà di base) con i vettori stimati.

¹⁵ Introdotta e spiegata nel capitolo 2.

- 3) Ci sono inoltre metodi per confrontare i risultati di più metodi differenti. In tal modo è possibile capire quali metodi generano risultati simili, e quali invece danno risultati differenti.
- 4) Un metodo può essere valutato su dati reali dall'introduzione di una *differenza incrementale dei gradi di vari tipi di rumore* nei dati di input. Fare ciò permette di capire la robustezza dei risultati delle varie tecniche. I rumori possono essere, ad esempio, variazioni di intensità, rumore gaussiano, oppure sfocamento.
- 5) Conoscendo la geometria della scena, ed approssimativamente, il moto della camera, è possibile confrontare i risultati ottenuti con i campi di flusso desiderati.

4.2 Errori di misura con la realtà di base disponibile

Realtà di base: immagini registrate in aeroplano sono spesso utilizzate per valutare la distanza dal suolo, ma anche per avere ricostruzioni in 3D di paesaggi o città. Per la valutazione dei risultati, è una pratica diffusa quella di utilizzare dei punti di riferimento sul suolo, come ad esempio negli angoli degli edifici, e misurare le distanze o le posizioni di questi punti. Questa tecnica è chiamata *realtà di base (ground truth)* e viene spesso utilizzata per il confronto con i valori calcolati dalle immagini prese dall'aeroplano. Il termine viene ormai utilizzato come definizione di dati di misura che vengono considerati abbastanza accurati.

La realtà di base per l'analisi del moto è difficile da ottenere, specialmente per sequenze reali. Le sequenze sintetizzate sono una valida alternativa. L'interpretazione fisica delle immagini può indirizzare verso delle sequenze di immagini sintetiche abbastanza realistiche, perfino nell'ipotesi di studiare il comportamento degli algoritmi per il flusso ottico per parametri variabili. La figura 19 mostra i risultati per una realtà di base fornita per una sequenza di immagini sintetica (100 frame) ed i risultati dell'algoritmo piramidale di *Horn-Schunck*. Oltre alla chiave dei colori si mostrano anche dei vettori sparsi sia nell'immagine della realtà di base che in quella dell'algoritmo. Questi vettori servono a far capire la direzione del moto.

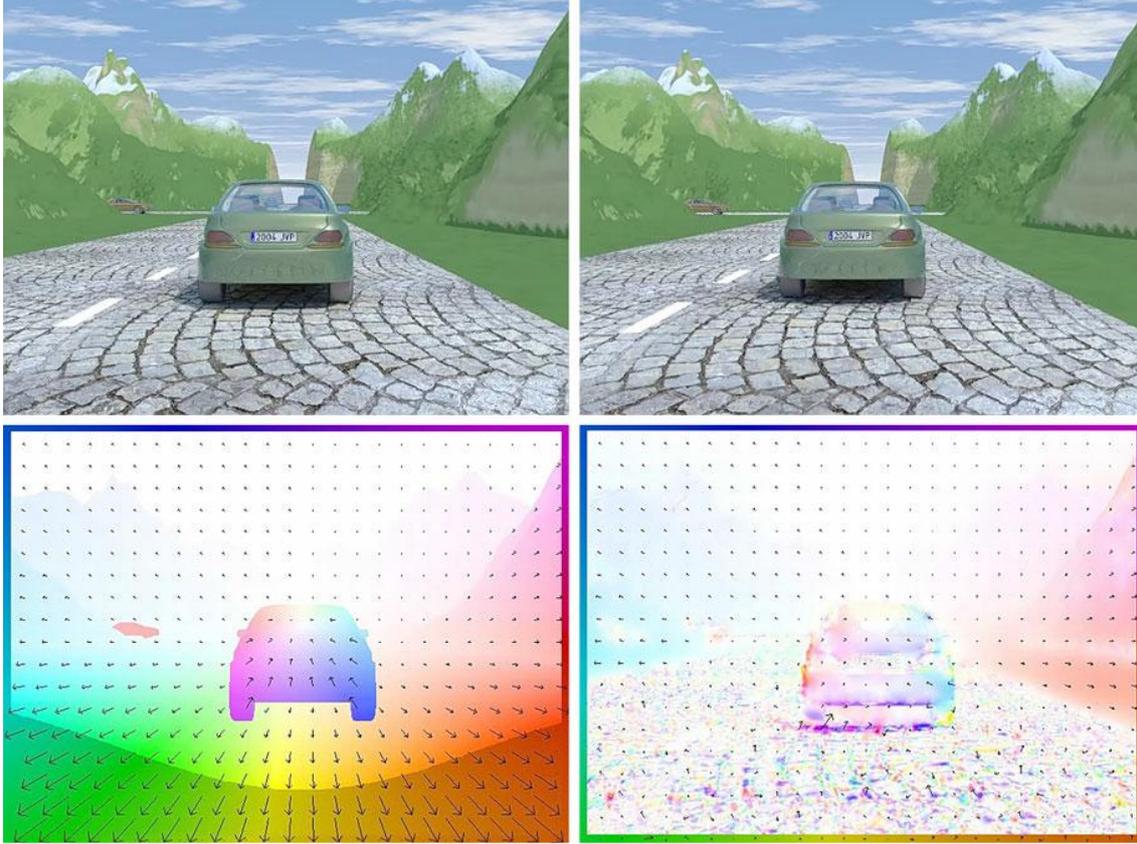


Figura 19: in alto 2 frame consecutivi. In basso a sinistra: il risultato utilizzando la realtà di base. In basso a destra: il risultato dell' algoritmo piramidale di Horn-Schunck.

Errori di misura in presenza di realtà di base: avendo calcolato il flusso $\mathbf{u} = (u, v)$ ed il flusso $\mathbf{u}^* = (u^*, v^*)$ fornito dalla realtà di base, è possibile calcolare l'errore di endpoint. Che sarà nel caso L2:

$$\mathbf{E}_{ep2}(\mathbf{u}, \mathbf{u}^*) = \sqrt{(u - u^*)^2 + (v - v^*)^2} \quad (1)$$

mentre nel caso L1:

$$\mathbf{E}_{ep1}(\mathbf{u}, \mathbf{u}^*) = |u - u^*| + |v - v^*| \quad (2)$$

Questi due errori sono ottenuti confrontando entrambi i vettori nello spazio 2D. L'errore angolare tra le direzioni spazio-temporali del flusso stimato e la realtà di base alla locazione di pixel p è il seguente:

$$\mathbf{E}_{ang}(\mathbf{u}, \mathbf{u}^*) = \arccos\left(\frac{\mathbf{u}^T \cdot \mathbf{u}^*}{|\mathbf{u}| |\mathbf{u}^*|}\right) \quad (3)$$

dove $\mathbf{u} = (u, v, 1)$ è il flusso ottico calcolato, ma esteso di una coordinata¹⁶ nello spazio 3D, ed $\mathbf{u}^* = (u_t, v_t, 1)$ è il flusso provvisto di realtà di base, anch'esso esteso allo spazio 3D. Questo errore misura le accuratezze esaminate in entrambe le direzioni dello spazio 3D. Gli errori di misura vengono applicati a tutti i pixel nel frame, permettendo di calcolare il valore medio dell'*errore angolare, dell'errore di endpoint L2*, e così via.

4.3 Tracciamento delle auto utilizzando il flusso ottico

Nei vari tool per la Computer Vision presenti in Matlab, uno che suscita particolare interesse per questo lavoro di tesi è il 'Tracking Cars Using Optical Flow'. Il modello Simulink del 'Tracking Cars Using Optical Flow' utilizza due degli algoritmi visti nel capitolo 2 per calcolare i vari vettori di flusso ottico. Dopo di che, fissata una regione limite dello spazio dell'immagine (in figura 20, nell'immagine in fondo, corrisponde alla linea bianca orizzontale), mette in evidenza gli oggetti (auto in questo caso) che la superano contrassegnandoli con un rettangolo verde, e ne tiene il conteggio. Nella figura 21 si presenta invece lo schema a blocchi del modello Simulink. Nel primo blocco sulla sinistra viene caricato un video, ad esempio le riprese di una telecamera in un tratto di strada extraurbano. Nel secondo blocco invece ci sono le impostazioni per la conversione del colore, nel caso mostrato si passa da RGB (che rappresentano i tre canali di colore) all'intensità. Il terzo blocco invece permette di scegliere l'algoritmo con cui calcolare il flusso ottico sono possibili solo l'algoritmo di Horn-Schunck, e quello di Lucas-Kanade. Nel penultimo blocco si possono scegliere le soglie di velocità (un utilizzo pratico potrebbe essere quello di un autovelox, in cui se la velocità calcolata risulta maggiore di quella di soglia allora scatta la foto del mezzo) e di filtraggio del rumore. L'ultimo blocco serve invece per mostrare su display i risultati del modello.

¹⁶ L'uno rappresenta la distanza tra l'immagine registrata all'istante di tempo t e quella registrata all'istante di tempo $t+1$.



Figura 20: esempio del funzionamento del tool, nella seconda immagine a partire dall'alto si può notare come venga calcolato il flusso ottico delle auto in movimento.

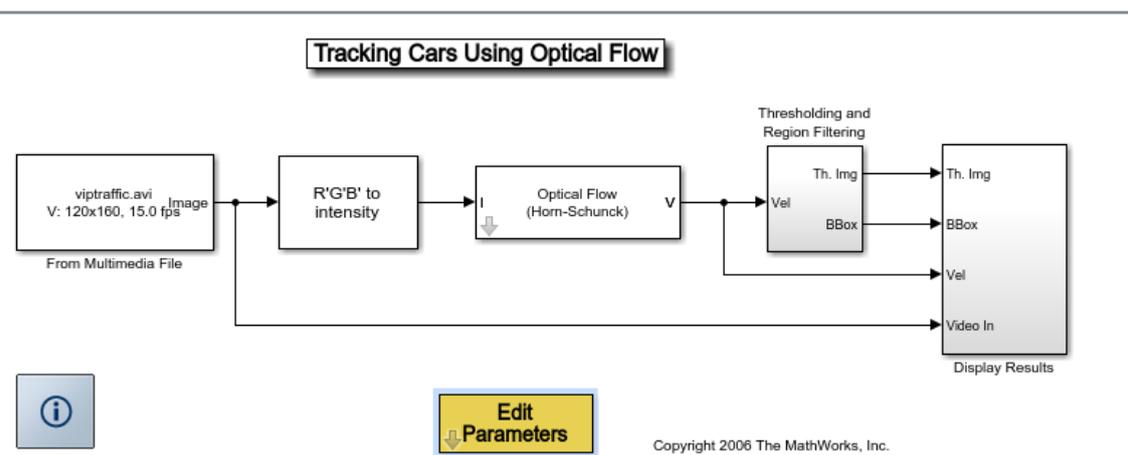


Figura 21: schema a blocchi del modello simulink per il tracciamento delle auto utilizzando il flusso ottico.

4.4 Algoritmo di Horn-Schunck su Matlab

Con Matlab è possibile anche calcolare semplicemente la direzione e la velocità di un oggetto in movimento da un'immagine o da dei frame utilizzando l'algoritmo di *Horn-Schunck*.

Costruzione:

- `opticFlow = opticalFlowHS` utilizzando l'algoritmo di *Horn-Schunck* questo comando restituisce il flusso ottico di un oggetto, il quale può essere utilizzato per trovare velocità e direzione dell'oggetto stesso.
- `opticFlow = opticalFlowHS(Name,Values)` aggiunge ulteriori opzioni, definite da *Names*, in coppia con degli argomenti *Values*.

Input: specificando i vari nomi e valori, separati da una virgola, e scrivendo il nome in questa maniera: '*Name*', è possibile far generare più flussi ottici, che dipenderanno, ovviamente, dalle immagini e dai valori associati ai vari *Names* e *Values*.

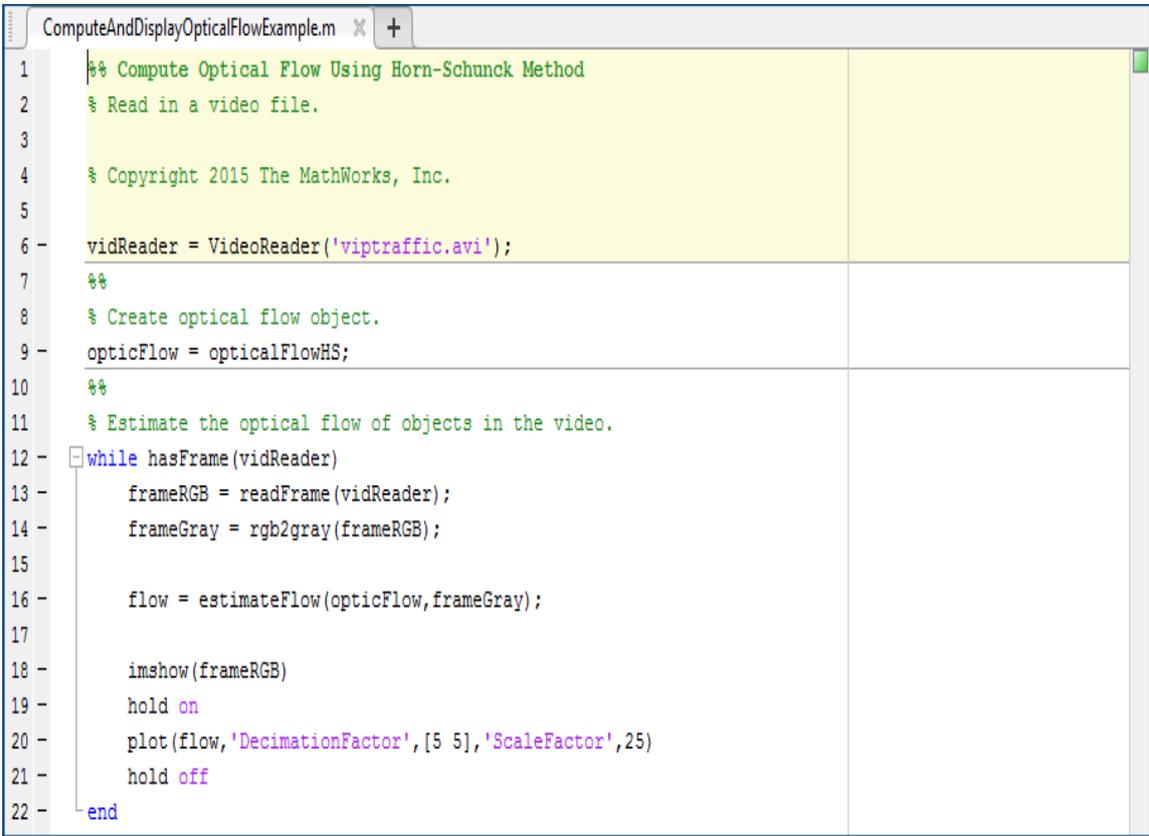
Esempi:

'*MaxIteration*', *Massimo numero di iterazioni*. In questo caso il *Name* fa in modo che le iterazioni necessarie all'algoritmo per ottenere il flusso ottico, siano entro un certo valore massimo. Il *Values* è quel valore massimo (10 è il valore di default, il numero da assegnare deve essere intero e positivo).

'*VelocityDifference*', *Minimo assoluto della differenza di velocità*. In quest'altro caso il flusso ottico verrà calcolato solo per quegli oggetti in movimento che avranno velocità superiore alla velocità di soglia data dal *Values*. Il valore di default è zero, ed anche qui si accettano solo valori positivi, però sono ammessi anche valori non interi. L'algoritmo fermerà le sue iterazioni o quando si è raggiunto il limite massimo di iterazioni, oppure quando l'algoritmo arriva al valore della velocità minima consentita. Nel caso si volesse far finire l'iterazione solo in base al primo criterio, bisognerebbe assegnare al *Values* il valore zero. Data una certa soglia per il *Minimo assoluto della differenza di velocità*, se si volessero ottenere anche i flussi ottici di oggetti che si spostano più lentamente, bisognerebbe semplicemente diminuire il *Values*.

Digitando il comando

“openExample('vision/ComputeAndDisplayOpticalFlowExample')” sulla Command Window di Matlab si aprirà lo script dell’algoritmo (figura 22) che si andrà ad esporre.



```
1 %% Compute Optical Flow Using Horn-Schunck Method
2 % Read in a video file.
3
4 % Copyright 2015 The MathWorks, Inc.
5
6 vidReader = VideoReader('viptraffic.avi');
7 %%
8 % Create optical flow object.
9 opticFlow = opticalFlowHS;
10 %%
11 % Estimate the optical flow of objects in the video.
12 while hasFrame(vidReader)
13     frameRGB = readFrame(vidReader);
14     frameGray = rgb2gray(frameRGB);
15
16     flow = estimateFlow(opticFlow,frameGray);
17
18     imshow(frameRGB)
19     hold on
20     plot(flow, 'DecimationFactor',[5 5], 'ScaleFactor',25)
21     hold off
22 end
```

Figura 22: script per il calcolo del flusso ottico da un video o da un'immagine.

Nella prima parte in verde c'è la spiegazione dello script; questa versione funziona solo per file video. Nella parte subito dopo, la linea di comando seleziona il filmato da analizzare, sempre quello delle auto del modello Simulink discusso prima. Nella linea di codice 8 si ha la spiegazione di ciò che si farà nella linea 9, ovvero si creerà l'oggetto del flusso ottico. L'ultima istanza invece stima il flusso ottico attraverso il video; il ciclo while continua finché (vidReader) ha frame, ovvero finché il video non è finito. Le linee 13 e 14 sono usate per la regolazione dei colori. Lo script infine restituirà l'immagine del calcolo del flusso ottico.

Esiste anche la versione che utilizza l'algoritmo di *Lucas-Kanade*: i tipi di input sono identici allo script appena presentato, quindi saranno sempre una coppia formata da *Names, Values*. In questa versione però è possibile settare altri valori, come ad esempio la soglia di rumore nelle immagini (NoiseThreshold) che viene impostata a 0.0039 di default, ed accetta solo valori positivi come possibili *Values*. In figura 23 si mostra il listato dello script per la versione che implementa l'algoritmo di *Lucas-Kanade*.

```
ComputeOpticalFlowUsingLucasKanadeWithTemporalSmoothingExample.m x +
1 %% Compute Optical Flow Using Lucas-Kanade derivative of Gaussian
2 % Read in a video file.
3
4 % Copyright 2015 The MathWorks, Inc.
5
6 vidReader = VideoReader('viptraffic.avi');
7 %%
8 % Create optical flow object.
9 opticFlow = opticalFlowLKDoG('NumFrames',3);
10 %%
11 % Estimate the optical flow of the objects and display the flow vectors.
12 while hasFrame(vidReader)
13     frameRGB = readFrame(vidReader);
14     frameGray = rgb2gray(frameRGB);
15
16     flow = estimateFlow(opticFlow,frameGray);
17
18     imshow(frameRGB);
19     hold on;
20     plot(flow,'DecimationFactor',[5 5],'ScaleFactor',25);
21     hold off;
22 end
```

Figura 23: versione dello script che ottiene il flusso ottico utilizzando l'algoritmo di Lucas-Kanade.

Bibliografia

- 1) G. Rodriguez, S. Seatzu; “Introduzione alla matematica applicata e computazionale”, Pitagora Editrice Bologna.
- 2) G. Rodriguez; “Algoritmi numerici”, Pitagora Editrice Bologna.
- 3) B.K.P. Horn, B.G. Schunck; “Determining optical flow”, *Artificial Intelligence*, vol. 17, pg. 185-203, 1981
- 4) B.D. Lucas, T. Kanade; “An iterative image registration technique with an application to stereo vision”, pg. 121-130, 1981.
- 5) P. Palladino, “Manuale di Illuminazione”, Tecniche Nuove.
- 6) R. Klette, “Concise Computer Vision. An Introduction into Theory and Algorithms”, Springer.
- 7) R. Szeliski, “Computer Vision: Algorithms and applications”, Springer.
- 8) Yin Li, Heung-Yeung Shum, Chi-Keung Tang, and R. Szeliski, “Stereo reconstruction from multiperspective panoramas”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(1):44-62, January 2004.
- 9) <https://it.mathworks.com/help/vision/examples/tracking-cars-using-optical-flow.html>
- 10) <https://it.mathworks.com/help/vision/ref/opticalflowhs-class.html>
- 11) <https://it.mathworks.com/help/vision/ref/opticalflowlkdog-class.html>