Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Tesi di Laurea Triennale in Ingegneria dell'Informazione

Algoritmo di RANSAC: panoramica, confronti e applicazioni

Candidato:
Alessio Zanchettin

Relatore:
Angelo Cenedese

Anno Accademico 2012/2013

A mia madre,
per essere stata,
e perché la sento ancora,
al mio fianco
in questo viaggio
che è la vita.
A mio padre
e a mia sorella,
per essere solido sostegno
ogni giorno,
ogni passo.

Indice

Abstract

Massa è tutto ciò che non valuta se stesso - nè in bene nè in male - mediante ragioni speciali, ma che si sente come tutto il mondo, e tuttavia non se ne angustia, anzi si sente a suo agio nel riconoscersi identico agli altri.

- Josè Ortega y Gasset -

Masse di dati acquisiti, elaborati, modellizzati e solo infine, dopo numerose operazioni utilizzati.
La tesi si occupa della presentazione del problema della modellizzazione di dati in presenza di outliers, prima facendo una panoramica sulle questioni più spinose, poi presentando l'algoritmo di RANSAC come soluzione, successivamente confrontandolo con l'algoritmo di Support Vector Regression, e infine mostrando una possibile applicazione di RANSAC per migliorare l'algoritmo Scale-Invariant Feature Transform in un semplice problema di visione computazionale.

Capitolo 1

Introduzione

True morality consists not in following the beaten track,
but in finding out the true path for ourselves
and in fearlessly following it.
- Mahatma Mohandas Karamchand Gandhi -

1 Stima dei parametri

Il problema della stima nasce quando si vogliano determinare uno o più parametri incogniti a partire da osservazioni sperimentali, di solito è noto un campione, ossia una registrazione di valori effettuata nel passato: tale campione va considerato come una realizzazione incompleta del processo casuale e va quindi trovato un modello che indichi in maniera corretta l'andamento di tali osservazioni.
L'insieme dei campioni può essere denotato come:

$$d(t), \qquad t = t_1, t_2, ..., t_N \rightarrow \theta(t)$$

dove $\{t_1, t_2, ..., t_N\}$ è l' insieme degli istanti di osservazione.

Partendo quindi dai dati $d(t)$ si vuole calcolare lo stimatore $\theta(t)$.
Uno stimatore è una funzione che associa ad ogni possibile campione un valore del parametro da stimare. È una funzione di un campione di dati che a partire da tali dati osservati in ingresso produce in uscita dei parametri incogniti; tali parametri servono ad individuare il processo casuale atto a

descrivere la grandezza osservata.
Uno stimatore è dunque una variabile casuale funzione del campione, a valori nello spazio parametrico (ossia nell'insieme dei possibili valori del parametro).
Tanto più i dati in ingresso sono rumorosi, tanto più difficoltoso risulta il processo di stima e il design di uno stimatore.
Più esattamente la costruzione di un modello stocastico avviene nel seguente modo:

1. Si sceglie la classe di processi casuali che si ritiene atta a descrivere la grandezza in questione (regressione lineare, parabolica, planare ecc...).
2. Si stimano in base ai campioni i parametri che identificano il processo casuale all'interno della classe scelta. Ciò equivale a selezionare, all'interno della classe prefissata, il modello che corrisponde maggiormente ai dati.
3. Si eseguono test diagnostici per verificare la rispondenza del modello. Se i test hanno esito negativo, si riparte dalla fase 1 con una nuova scelta della classe; se i test hanno esito positivo il modello è pronto per essere usato.

Uno degli ambiti di utilizzo degli stimatori, in presenza di dati rumorosi, si ha nei problemi di data association, come avviene, ad esempio, nel caso delle applicazioni di visione computazionale (computer vision)

2 Panoramica Storica di RANSAC

RANSAC sta per RANdom SAmple Consensus.
È un metodo iterativo per la stima dei parametri di un modello matematico a partire da un insieme di dati di input contenente una grande percentuale di valori outliers.
È un algoritmo non deterministico, cioè produce un risultato corretto solo con una data probabilità, che aumenta al crescere delle iterazioni.
L'algoritmo è stato pubblicato per la prima volta da Fischler e Bolles nel 1981 [1].
A differenza della maggior parte delle tecniche comuni di stima robusta, come

M-stimatori e stima dei minimi quadrati, che sono state adottate dalla comunità di computer vision, RANSAC è stato sviluppato direttamente dalla comunità di computer vision.
Come sottolineato da Fischler e Bolles [1], la differenza SOSTANZIALE rispetto alle convenzionali tecniche di stima è che RANSAC genera delle soluzioni candidate utilizzando il minimo numero di osservazioni (dati di input) necessarie per stimare i parametri del modello.
Infatti, mentre le classiche tecniche utilizzano la maggior quantità di dati possibile per ottenere una soluzione e solo successivamente procedono a eliminare i valori anomali, RANSAC utilizza il più piccolo insieme di dati possibile per determinare il modello e procede nell'ingrandire questo insieme con i punti di dati coerenti con il modello stimato.

3 Definizioni base

Innanzitutto è obbligatorio definire un background di nozioni teoriche e definizioni che verranno poi utilizzate all'interno della tesi. Vengono definiti di seguito i principali concetti relativi al problema della stima matematica e dell'ottimizzazione e successivamente alcuni concetti relativi all'algoritmo di RANSAC.
Tali concetti possono essere ricercati in particolar modo in [2, p. 3].

3.1 Outliers

Una definizione formale del concetto di outlier non esiste, ma una definizione abbastanza generale e sufficientemente formale potrebbe essere la seguente:

> *Un dato è considerato un outlier se non si adatta al vero modello istanziato dal vero insieme di parametri, tenendo conto di una soglia di errore che definisce la deviazione massima attribuibile all'effetto del rumore.*

Cioè un outlier è, in un insieme di dati, un valore anomalo, distante dalle altre osservazioni disponibili che invece si adattano al modello vero, considerando che esse hanno un errore di misura dovuto ad un rumore.
Assumiamo che, se i dati di ingresso non fossero affetti da rumore, si potrebbe

definire il modello vero come quell'insieme di parametri che riesce a generare perfettamente tutti i dati inliers, mentre non riesce a dare origine agli outliers.

3.2 Bias

In statistica, il bias (distorsione o scostamento) di uno stimatore è la differenza tra il valore atteso di questo stimatore e il valore vero del parametro stimato; uno stimatore è affetto da bias quando il valore atteso dello stimatore, per qualche motivo, risulta diverso dalla quantità stimata. Per definire meglio il bias utilizziamo la seguente definizione [3]:

> *Sia $D_I \subset D$ un insieme di inliers e sia $D_{I/O}(m)$ l'insieme precedente dopo che m inliers sono stati sostituiti da valori outliers. Come di consueto $\mathcal{M}$ indica il modello che stiamo considerando e θ il vettore di parametri che lo caratterizzano.*
> *Il bias associato al modello $\mathcal{M}$ e agli insiemi D_I e $D_{I/O}(m)$ definito come:*
>
> $$bias_{\mathcal{M}}(m; D_I) \stackrel{def}{=} \sup_{D_{I/O}(m)} dist_{\mathcal{M}}(\theta(D_I), \theta(D_{I/O}(m))) \quad (1.1)$$

Questa quantità misura la perturbazione massima che potrebbe essere causata ad un vettore di parametri θ stimandolo inizialmente solo usando inliers e poi andando a sostituire m di questi inliers con outliers.
La funzione $dist_{\mathcal{M}}$ misura la distanza tra i vettori di parametri stimati nei due modi differenti.

3.3 Funzione di Costo

In ottimizzazione matematica una funzione di perdita, o funzione di costo, è una funzione che associa un evento, quantificato con un valore reale δ, ad un numero reale che rappresenta intuitivamente il costo associato all'evento $l(\delta)$. Un problema di ottimizzazione cerca di minimizzare una funzione di perdita (o massimizzare una funzione di guadagno).
Gli algoritmi di regressione possono essere considerati come algoritmi di ottimizzazione che devono minimizzare una particolare funzione di costo.

Tecniche che tentano di risolvere questo problema vengono chiamate statistica robusta [5]. In problemi di regressione, la differente scelta delle funzioni di perdita deriva da varie ipotesi riguardo la distribuzione del rumore nella misura [4].
La funzione di perdita più comune è la funzione quadratica(fig. 1.1.a), corrispondente ad un modello di rumore gaussiano a media nulla e ad una deviazione standard che non dipende dagli ingressi.
La funzione di perdita gaussiana viene solitamente utilizzata poichè presenta delle sue notevoli proprietà analitiche. Tuttavia, uno dei potenziali errori della funzione di perdita quadratica standard è che essa riceve grandi contributi dai valori outliers che hanno grandi errori rispetto al resto dei dati: se ci sono lunghe code sulle distribuzioni allora la soluzione può essere dominata da un piccolo numero di valori anomali che cambiano quindi il risultato finale.
Diverse funzioni di perdita non quadratiche sono state introdotte per ridurre la sensibilità ai valori outliers, come ad esempio la funzione di perdita laplaciana, la funzione di perdita di Huber, la funzione di perdita ϵ-insensitive:

- FUNZIONE DI PERDITA LAPLACIANA (fig. 1.1.b): se si assume che l'errore di cui sono affetti i valori è Laplaciano, la funzione di perdita è definita come:

$$l(\delta) = |\delta| \tag{1.2}$$

- FUNZIONE DI PERDITA DI HUBER (fig. 1.1.c): La funzione di perdita di Huber [5] è definita come

$$l(\delta) = \begin{cases} \frac{1}{4\epsilon}\delta^2 & |\delta| \leq 2\epsilon \\ |\delta| - \epsilon & altrove \end{cases} \tag{1.3}$$

  dove $\epsilon > 0$ è un valore di soglia.
- FUNZIONE DI PERDITA ϵ-INSENSITIVE (fig. 1.1.d): la funzione di perdita ϵ-insensitive, proposta da Vapnik nella teoria della Support Vector Machine (SVM) per la regressione (SVR) [6] 3.2, viene definita come:

$$l(\delta) = \begin{cases} 0 & |\delta| \leq \epsilon \\ |\delta| - \epsilon & altrove \end{cases} \tag{1.4}$$

dove $\epsilon > 0$ è un valore di soglia.

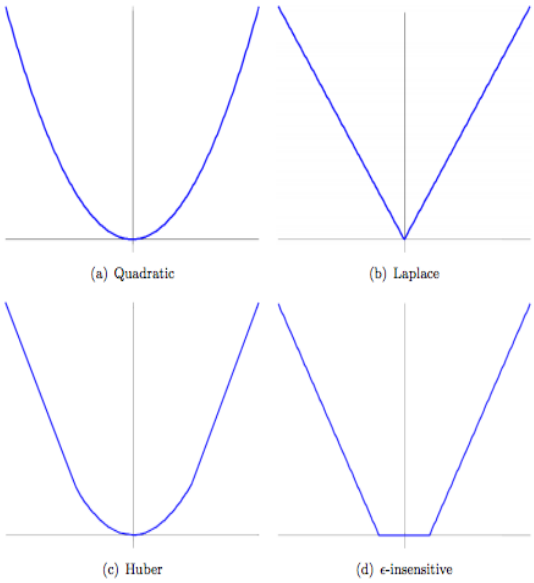

Figura 1.1: Grafici delle varie funzioni di costo descritte

3.4 Dati iniziali e MSS

L'insieme dei dati iniziali composto da N elementi d-dimensionali è indicato da

$$D = \{\bar{d}_1, \ldots, \bar{d}_N\} \qquad con \quad \bar{d}_i \in \mathbb{R}^d \tag{1.5}$$

L'MSS (Minimal Sample Set) è l'insieme minimo s di k campioni iniziali che servono per costruire il modello $\mathcal{M}$.

$$s = \{\bar{d}_{i_1}, \ldots, \bar{d}_{i_k}\} \qquad \bar{d}_{i_i} \in D \tag{1.6}$$

Da questi punti selezionati è possibile dedurre tutti i parametri del modello prescelto e quindi la dimensione dell'MSS deve risultare la minima sufficiente, e necessaria, per stimare questi parametri.
Ad esempio per quanto riguarda la regressione lineare in $\mathbb{R}^2$, il modello stimato sarà un modello lineare e quindi la dimensione dell'MSS sarà

pari a 2. Infatti sono necessari e sufficienti due punti distinti per poter tracciare un unica retta di $\mathbb{R}^2$ che passa per questi due punti.

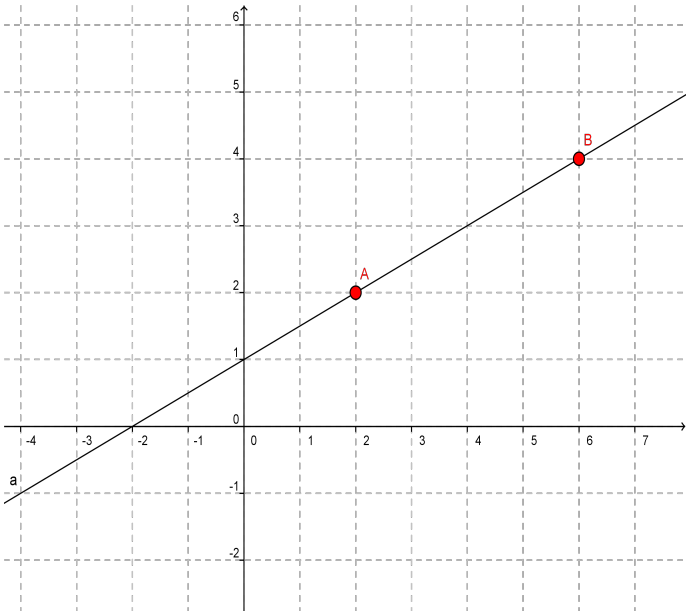

Figura 1.2: Retta passante per due punti

Siano $A = (x_a, y_a)$ e $B = (x_b, y_b)$ allora l'equazione esplicita della retta passante per questi due punti è:

$$y = mx + q \qquad con \qquad m = \frac{y_b - y_a}{x_b - x_a}, \quad q = y_a - mx_a \tag{1.7}$$

se $A, B \in$ MSS allora la retta y risulta il modello lineare relativo a quel MSS e i parametri relativi sono $\theta = (m, q)$.

3.5 Modello

Sia $\bar{\theta}(\{\bar{d}_{i_1}, \ldots, \bar{d}_{i_k}\})$ il vettore di parametri calcolati a partire dai punti appartenenti all'MSS. Il modello $\mathcal{M}$ è definito come:

$$\mathcal{M}(\bar{\theta}) \stackrel{def}{=} \{\bar{d} \in \mathbb{R}^d : f_{\mathcal{M}}(\bar{d}; \bar{\theta}) = 0\} \tag{1.8}$$

dove $\bar{\theta}$ è il vettore dei parametri e $f_{\mathcal{M}}$ è una funzione che si azzera in tutti i punti che verificano il modello $\mathcal{M}$.

3.6 Errore

Definiamo l'errore associato al dato $\bar{d}$ rispetto al modello $\mathcal{M}$ come la distanza tra $\bar{d}$ e $\mathcal{M}(\bar{\theta})$

$$e_{\mathcal{M}}(\bar{d}, \bar{\theta}) \stackrel{def}{=} \min_{\bar{d}^* \in \mathcal{M}(\theta)} dist(\bar{d}, \bar{d}^*) = dist(\bar{d}, \bar{d}^{\perp}) \tag{1.9}$$

dove $\bar{d}^*$ sono tutti i punti appartenenti a $\mathbb{R}^d$ che fanno parte dello spazio del modello $\mathcal{M}(\theta)$, $\bar{d}^{\perp}$ è la proiezione ortogonale di $\bar{d}$ sullo spazio del modello $\mathcal{M}(\theta)$ e $dist(\cdot, \cdot)$ è definita come una appropriata funzione distanza.
Tale valore viene utilizzato per definire quanto è ben approssimato il dato $\bar{d}$ dal modello $\mathcal{M}(\bar{\theta})$, quindi se questo dato può essere considerato inlier oppure va considerato outlier.

3.7 CS

Il CS (Consensus Set) è l'insieme dei dati iniziali che sono consistenti con il modello calcolato grazie all'MSS. Il CS contiene quindi i dati inliers rispetto alla regressione calcolata.
Usando la definizione di errore (3.6) definiamo il CS come:

$$s(\bar{\theta}) \stackrel{def}{=} \{\bar{d} \in D : e_{\mathcal{M}}(\bar{d}; \bar{\theta}) \leq \delta\} \tag{1.10}$$

dove δ è una soglia che dipende dalla natura del problema, o sotto alcune ipotesi può essere calcolata automaticamente (3.8).
Per classificare la bontà di un CS viene calcolata la probabilità di trovare un CS migliore: questa probabilità diminuisce con il numero di prove di ricerca che si compiono e quando questa scende al di sotto di una certa soglia si può considerare di aver trovato un CS con un numero di inliers sufficienti ai nostri scopi (3.9).

3.8 Soglia d'errore

Abbiamo definito il CS (3.7) introducendo δ come variabile che indica la soglia d'errore massima per la quale i dati vengono considerati inliers.
Il valore di tale soglia massima si può relazionare alla distribuzione

statistica dei dati di input e dal rumore che affligge i dati. Possiamo quindi scrivere che:

$$e_{\mathcal{M}}(\bar{d},\theta) = \min_{\bar{d}^* \in \mathcal{M}(\bar{\theta})} \sqrt{\sum_{i=1}^{n}(\bar{d}_i - \bar{d}_i^*)^2} = \sqrt{\sum_{i=1}^{n}(\bar{d}_i - \bar{d}_i^{\perp})^2} \tag{1.11}$$

(dove $\bar{d}^{\perp}$ e $\bar{d}^*$ sono definiti come in 3.6)
Ora supponiamo che il vettore di dati $\bar{d}$ sia affetto da rumore Gaussiano $\bar{\eta} \sim \mathcal{N}(0, \sigma_\eta I)$, e quindi lo sia anche la differenza $\bar{d} - \bar{d}^*$. Il nostro obbiettivo è calcolare il valore di δ che delimiti, con una data probabilità p_{inlier}, l'errore generato da un dato inlier contaminato con un errore Gaussiano.
Più formalmente si vuole trovare δ tale che:

$$P\left[e_{\mathcal{M}}(\bar{d},\theta) \leq \delta\right] = p_{inlier} \tag{1.12}$$

Si può facilmente scoprire, essendo l'errore $e_{\mathcal{M}}$ Gaussiano per ipotesi, che:

$$P\left[e_{\mathcal{M}}(\bar{d},\theta) \leq \delta\right] = P\left[\sum_{i=1}^{n}\eta_i^2 \leq \delta^2\right] = P\left[\sum_{i=1}^{n}\left(\frac{\eta_i}{\sigma_n}\right)^2 \leq \frac{\delta^2}{\sigma_n^2}\right] \tag{1.13}$$

e quindi $\frac{\eta}{\sigma_n} \sim (0,1)$, allora $\sum_{i=1}^{n}\left(\frac{\eta_i}{\sigma_n}\right)^2$ ha distribuzione $\mathcal{X}_n^2$.
Allora

$$\delta = \sigma_n \sqrt{F_{\mathcal{X}_n^2}^{-1}(p_{inlier})} \tag{1.14}$$

dove $F_{\mathcal{X}_n^2}^{-1}$ è la funzione di distribuzione inversa associata alla variabile casuale $\mathcal{X}_n^2$.
Da notare che se p_{inlier} tende ad 1 il valore di $F_{\mathcal{X}_n^2}^{-1}$ diverge all'infinito, quindi più alta è questa probabilità più grande sarà la soglia δ, ma potrebbe esserci il rischio di includere degli outliers nel modello, d'altra parte se si sceglie una soglia di probabilità molto piccola è possibile che alcuni inliers vengano scartati perché la soglia risulta troppo piccola.

3.9 Iterazioni

Il numero di iterazioni deve essere scelto abbastanza alto da garantire che il modello calcolato alla fine delle iterazioni abbia probabilità p di non contenere alcun outlier.
Se u rappresenta la probabilità che un dato sia un inlier, e quindi $v = 1 - u$ sia la probabilità che sia un outlier, allora il numero di iterazioni richieste h per il numero minimo di dati da considerare nel MSS m risolve l'equazione:

$$1 - p = (1 - u^m)^h \tag{1.15}$$

e questa quantità tende a zero se h tende all'infinito.
Se ipotizziamo che la probabilità di avere almeno un outlier $1 - p$ sia minore o uguale ad una soglia ϵ cioè $1 - p \leq \epsilon$ allora il numero di iterazioni deve garantire che, invertendo la (1.15):

$$h \geq \frac{\log(\epsilon)}{\log(1 - (1 - v)^m)} \tag{1.16}$$

4 Algoritmo dei minimi quadrati

4.1 Panoramica

Il metodo dei minimi quadrati (in inglese OLS: Ordinary Least Squares) è una tecnica di ottimizzazione che permette di trovare una funzione, detta curva di regressione, che si avvicini il più possibile ad un insieme di dati (tipicamente punti del piano). In particolare la funzione trovata deve essere quella che minimizza la somma dei quadrati delle distanze tra i dati osservati e la curva che rappresenta la funzione di regressione. L'utilizzo più frequente è l'approssimazione dell'andamento di dati sperimentali con linee di tendenza. Anche altri problemi di ottimizzazione, come la minimizzazione dell'energia o la massimizzazione dell'entropia, possono essere riformulati in una ricerca dei minimi quadrati.

4.2 Il problema

Siano (x_i, y_i) con $i = 1, 2, ..., n$ i punti che rappresentano i dati in ingresso.
Si vuole trovare una funzione f tale che approssimi la successione di punti data. Questa può essere determinata minimizzando la distanza (euclidea) tra le due successioni y_i e $f(x_i)$, ovvero la quantità;

$$S = \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{1.17}$$

da cui il nome minimi quadrati.
Nei casi pratici in genere f(x) è parametrica: in questo modo il problema si riduce a determinare i parametri che minimizzano la distanza dei punti dalla curva.
Naturalmente per ottenere un'unica curva ottimizzata e non un fascio, è necessario un numero di punti sperimentali maggiore del numero di parametri da cui dipende la curva.
In genere dai dati sperimentali ottenuti ci si aspetta una distribuzione regolata da relazioni determinate per via analitica; risulta utile quindi parametrizzare la curva teorica e determinare i parametri in modo da minimizzare S.
Nel caso della regressione lineare i parametri da determinare sono due:

$$f(x) = \beta_0 + \beta_1 x \tag{1.18}$$

in cui β_0 è l'intercetta della regressione e β_1 è la pendenza della regressione.
I parametri β_0 e β_1 da stimare sono quelli che rendono minima S.

4.3 Risoluzione nel caso lineare

Sia $f(x)$ una funzione lineare rispetto ai parametri

$$f(x) = p_1 f_1(x) + p_2 f_2(x) + \cdots + p_k f_k(x) \tag{1.19}$$

dove p_i sono i k parametri, $k \ll n$ e n è il numero di punti noti.

Si può riorganizzare la situazione attraverso il sistema lineare sovradimensionato

$$Ap \approx y \tag{1.20}$$

dove:

$$A = \begin{bmatrix} f_1(x_1) & \dots & f_k(x_1) \\ \vdots & & \vdots \\ f_1(x_n) & \dots & f_k(x_n) \end{bmatrix}, p = \begin{bmatrix} p_1 \\ \vdots \\ p_k \end{bmatrix}, y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}. \tag{1.21}$$

Il problema di minimizzare S si riconduce dunque a minimizzare la norma del residuo

$$\|r\| = \|Ap - y\| \tag{1.22}$$

$$\|r\|^2 = \|Ap - y\|^2 = ([Ap]_1 - y_1)^2 + \dots + ([Ap]_n - y_n)^2 = \sum_{i=1}^{n} (f(x_i) - y_i)^2 = S \tag{1.23}$$

dove con $[Ap]_i$ si intende l' i-esima componente del vettore prodotto fra A e p.
Possiamo minimizzare $\|r\|$ derivando $\|r\|^2$ rispetto a ciascun p_m e ponendo le derivate pari a 0:

$$\frac{d\|r\|^2}{dp_m} = \sum_{i=1}^{n} 2(\sum_{j=1}^{k} a_{ij}p_j - y_i)a_{im} = 0 \tag{1.24}$$

queste equazioni sono equivalenti al sistema:

$$(Ap - y)^T A = 0 \tag{1.25}$$

Quindi il vettore p che minimizza S è la soluzione dell'equazione:

$$A^T Ap = A^T y \tag{1.26}$$

Quest'ultima equazione è detta equazione normale. Se il rango di A è completo allora $A^T A$ è invertibile e dunque:

$$p = (A^T A)^{-1} A^T y \tag{1.27}$$

La matrice $(A^T A)^{-1} A^T$ è detta pseudo-inversa.

Capitolo 2

Implementazioni dell'algoritmo di RANSAC

The beginning of knowledge
is the discovery of something we do not understand.
- Frank Herbert -

Come primo aspetto sono state realizzate due implementazioni dell'algoritmo di RANSAC in *Matlab*, la prima che prevede la generazione di dati casuali di cui stimare il modello e la seconda che prevede l'inserimento di dati da parte dell'utente. Per quanto riguarda questa seconda implementazione sarà poi riportato un confronto con l'algoritmo presentato nella dispensa *Ransac for Dummies* [2, cha. 4].
I codici delle funzioni realizzate si trovano nell'appendice A.

1 Fasi dell'algoritmo di RANSAC

L'algoritmo consiste in cinque fasi successive [7]:

1. **Scelta del MSS** (definizione 1.3.4): l'MSS viene scelto in maniera casuale dall'insieme dei dati iniziali.
   La cardinalità dell'MSS è necessaria e sufficiente per determinare il modello (ad esempio per il modello lineare l'MSS consiste in 2 punti appartenenti a $\mathbb{R}^2$).

2. **Ipotesi del modello**: il modello viene calcolato unicamente con l'utilizzo dei punti dell'MSS.
   Viene tracciata la sottovarietà lineare affine passante per quei punti, appartenente allo spazio a cui appartengono i dati iniziali (ad esempio per il modello lineare si calcola l'equazione della retta passante per i 2 punti dell'MSS)

3. **Stima dell'errore**: controlla per ogni punto appartenente all'insieme dei dati iniziali se sono consistenti con il modello calcolato nel punto 2, cioè viene calcola la distanza di ogni punto dal modello e viene considerato questo valore come errore di modellizzazione (si utilizza una funzione di costo di cui nel 1.3.3) e si tiene conto dell'errore totale per compiere la scelta del punto 5.
   Nella figura 2.1 si vede come vengono calcolate le distanze dei vari punti di input rispetto al modello. In questo algoritmo la funzione di costo adottata è la ϵ-INSENSITIVE.

4. **Selezione del CS** (definizione 1.3.7): i punti che hanno un errore al di sotto di una certa soglia 1.3.8 vengono considerati punti ben approssimati dal modello e quindi inliers, gli altri vengono considerati outliers.
   Se il numero di inliers del CS relativo al modello stimato supera un certo valore e l'errore totale è al di sotto di una certa soglia, tale modello viene considerato buono come approssimazione di tutti i dati iniziali, considerando la presenza di dati outliers.

5. **Ripetizione dell'ipotesi e della verifica**: si ripetono i passi precedenti per un numero di volte sufficienti 3.9 per garantire che la probabilità di non avere outliers nel CS scelto sia maggiore di una certa soglia.
   Si sceglie il modello che nel passo 4 ha i valori migliori per quanto riguarda numero di inliers e grandezza dell'errore.
   Se nessuna ipotesi di modello è sufficientemente accurata l'algoritmo non restituisce alcun modello.

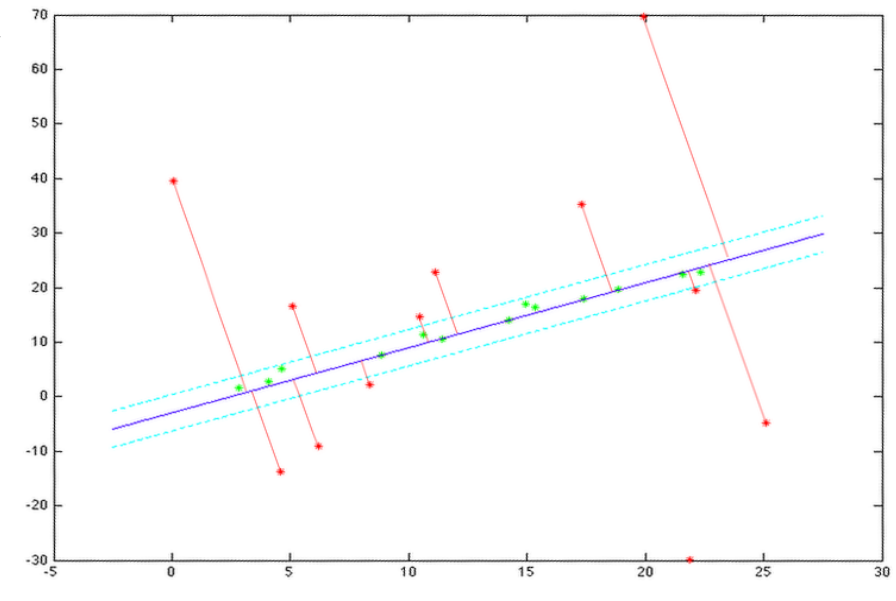

Figura 2.1: Grafico delle distanze degli outliers dal modello calcolato

Nella figura 2.1 sono state evidenziate le distanze dei punti considerati outliers per una certa iterazione: tutti i punti all'interno delle due rette con distanza δ dal modello calcolato avranno peso nullo nella funzione di costo che RANSAC deve minimizzare, mentre tutti i dati al di fuori hanno un peso pari alla loro distanza da modello.

RANSAC quindi nelle sue varie iterazioni costruisce modelli differenti partendo da MSS diversi e controllando se questi modelli sono buoni oppure no per i dati di ingresso inseriti.
Negli algoritmi sviluppati è possibile visualizzare le iterazioni successive che l'algoritmo compie semplicemente impostando il campo *history* del parametro di input *options* al valore TRUE. Se ciò viene fatto possono essere visualizzati i modelli successivi che l'algoritmo calcola evidenziando le soglie entro le quali i dati vengono considerati inliers e al di fuori delle quali i dati sono considerati outliers.

La figura 2.2 mostra le iterazioni successive che l'algoritmo compie per poter trovare il modello migliore in un esempio di applicazione della funzione $RANSAC\_data\_Input$.

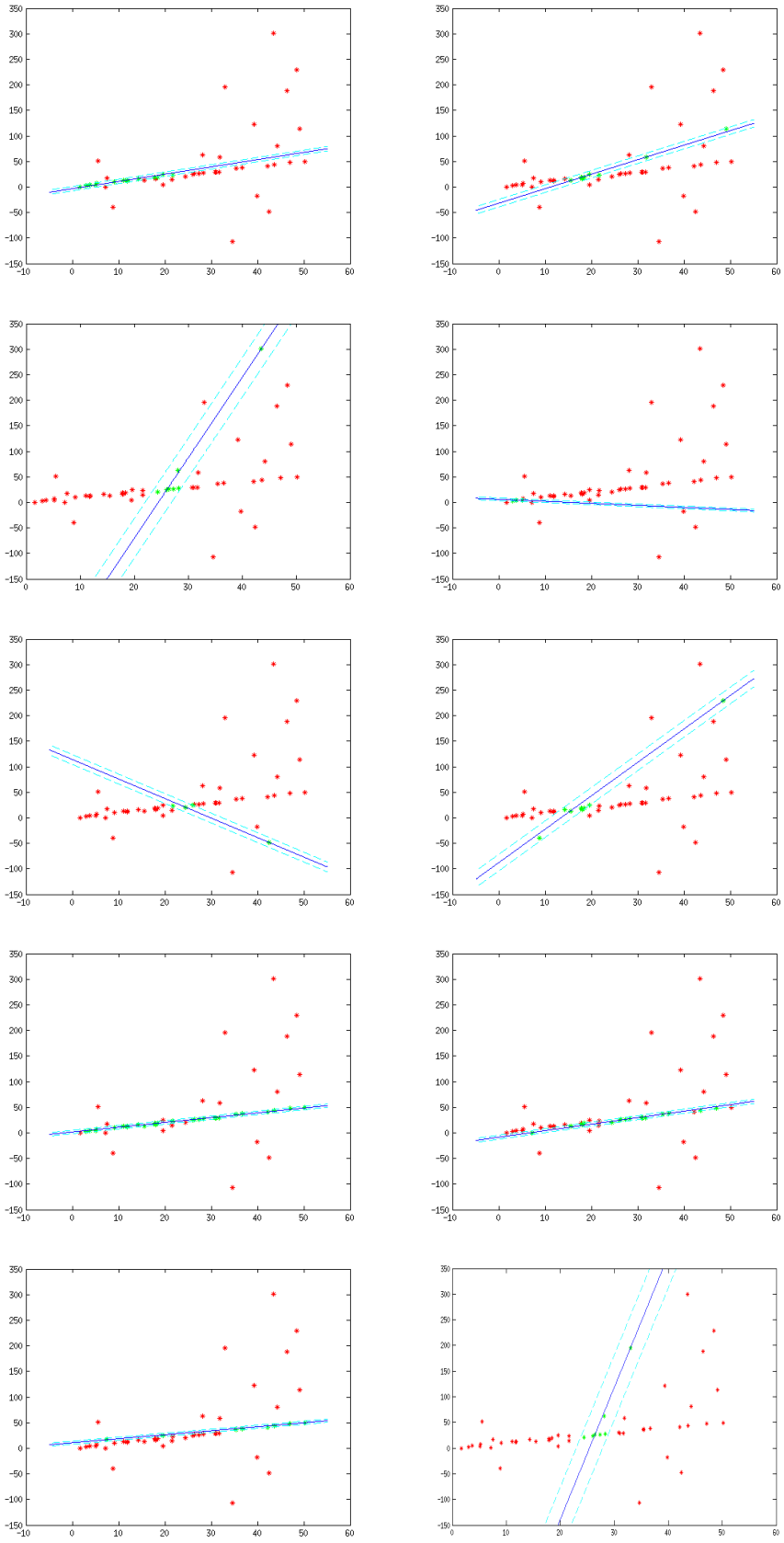

Figura 2.2: Iterazioni dell'algoritmo

2 RANSAC Data Random

2.1 Descrizione della funzione

La prima funzione prevede la generazione di dati casuali che si distribuiscono lungo la retta $y = x$ con un rumore gaussiano $\mathcal{N}(0, 1)$ introdotto in ogni punto.
Successivamente per la metà dei punti generati viene introdotto un errore casuale molto elevato che dovrebbe generare dei dati *outliers*.
Infine sui dati così ottenuti viene eseguito l'algoritmo di RANSAC spiegato in dettaglio nella sezione 1.
Vengono plottati i grafici relativi ai punti creati e al modello stimato dall'algoritmo di RANSAC, confrontato con il modello calcolato con i Minimi Quadrati. [fig. 2.3]
Il codice relativo alla funzione è riportato in A.1.

Di seguito sono riportate le varie caratteristiche della chiamata della funzione *ransac_dataRandom()*.

2.2 Input della funzione

La chiamata alla funzione *Matlab* è la seguente :

$$ransac\_dataRandom(N, Pin, e, options)$$

Questa realizzazione riceve in input i parametri:

- N la dimensione del set di dati che deve realizzare.
- Pin è la probabilità P_{in} che un dato sia considerato inlier, cioè che la distanza tra tra il dato e il modello sia inferiore ad una certa soglia (se $P_{in} \to 1$ tutti i dati potrebbero essere considerati inliers, se $P_{in} \to 0$ alcuni inliers potrebbero non essere considerati tali).
- e è la massima probabilità di trovare un outlier nei dati finali considerati inliers per il modello calcolato alla fine delle iterazioni dell'algoritmo.
- *options* è un oggetto struct (struttura) con i seguenti campi:
  - *history* valore booleano (default false) che indica se si vuole visualizzare graficamente la sequenza di iterazioni dell'algoritmo.

* *compare* valore booleano (default false) che indica se si vuole visualizzare graficamente, nella figura finale, il modello calcolato con i minimi quadrati per confrontarlo con quello ricavato tramite l'algoritmo di RANSAC.

2.3 Output della funzione

L'uscita della funzione è una array chiave-valore:

$$[bestModel, finalInliers, finalOutliers, finalError]$$

Dove:

- *bestModel* è il modello migliore calcolato con l'algoritmo.
- *bestInliers* è l'insieme dei dati inliers dal modello *bestModel*.
- *bestOutliers* è l'insieme dei dati outliers dal modello *bestModel*.
- *bestError* è l'errore totale tra i *bestInliers* e il modello *bestModel*.

Inoltre la figura 2.3 mostra il grafico finale che viene plottato dalla funzione *ransac_dataRandom()*.

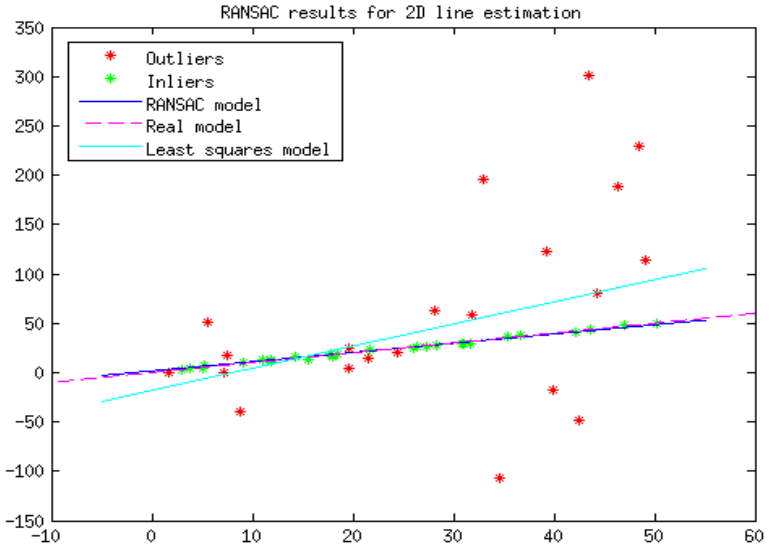

Figura 2.3: Grafico finale di un esempio della chiamata alla funzione *ransac_dataRandom()*

Si può notare già da questo semplice esempio, ripetuto molte volte, che il modello calcolato con l'algoritmo di RANSAC risulta molto più

accurato rispetto a quello calcolato con i minimi quadrati, in presenza di un numero elevato di outliers.

3 RANSAC Data Input

3.1 Descrizione della funzione

La seconda funzione prevede l'inserimento da parte dell'utente di un insieme di dati appartenenti a $\mathbb{R}^2$ di dimensione arbitraria.
Su questi dati viene eseguito l'algoritmo di RANSAC spiegato in dettaglio nella sezione 1.
Vengono plottati i grafici relativi ai punti creati e al modello stimato dall'algoritmo di RANSAC, confrontato con il modello calcolato con i Minimi Quadrati [fig 2.4].
Il codice relativo alla funzione è riportato in A.2.

Di seguito sono riportate le varie caratteristiche della chiamata della funzione *ransac_dataInput()*.

3.2 Input della funzione

La chiamata alla funzione *Matlab* è la seguente:

$$ransac\_dataInput(data, Pin, e, sigma, options)$$

Questa realizzazione riceve in input i parametri:

- *data* è l'insieme dei dati inseriti dall'utente sui quali applicare RANSAC.
- *Pin* è la probabilità P_{in} che un dato sia considerato inlier, cioè che la distanza tra tra il dato e il modello sia inferiore ad una certa soglia (se $P_{in} \to 1$ tutti i dati potrebbero essere considerati inliers, se $P_{in} \to 0$ alcuni inliers potrebbero non essere considerati tali).
- e è la massima probabilità di trovare un outlier nei dati finali considerati inliers per il modello calcolato alla fine delle iterazioni dell'algoritmo.
- *sigma* è lo scarto quadratico medio dei dati in input, (la radice quadrata della varianza dei dati in input).
- *options* è un oggetto struct (struttura) con i seguenti campi:

* *history* valore booleano (default false) che indica se si vuole visualizzare graficamente la sequenza di iterazioni dell'algoritmo.
* *compare* valore booleano (default false) che indica se si vuole visualizzare graficamente, nella figura finale, il modello calcolato con i minimi quadrati per confrontarlo con quello ricavato tramite l'algoritmo di RANSAC.

3.3 Output della funzione

L'uscita della funzione è una array chiave-valore:

$$[bestModel, finalInliers, finalOutliers, finalError]$$

Dove:

- *bestModel* è il modello migliore calcolato con l'algoritmo.
- *bestInliers* è l'insieme dei dati inliers dal modello *bestModel*.
- *bestOutliers* è l'insieme dei dati outliers dal modello *bestModel*.
- *bestError* è l'errore totale tra i *bestInliers* e il modello *bestModel*.

Inoltre la figura 2.4 mostra il grafico finale che viene plottato dalla funzione *ransac_dataInput()*.

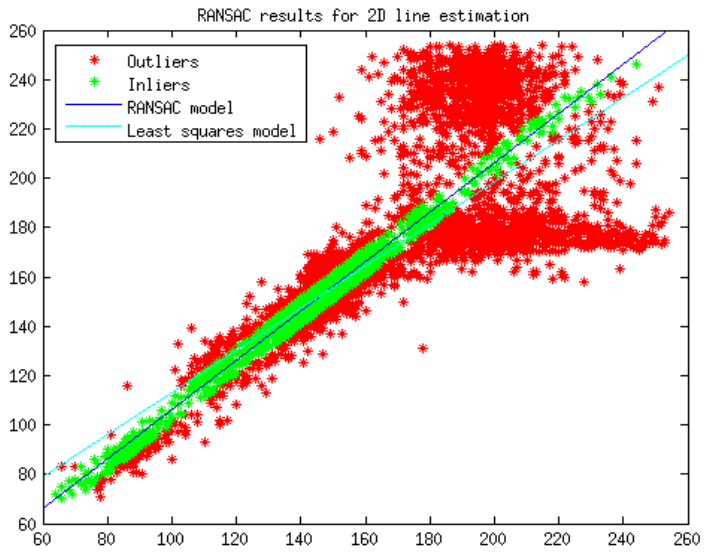

Figura 2.4: Grafico finale di un esempio della chiamata alla funzione *ransac_dataInput()*

Anche in questo caso si può notare come il modello calcolato con RANSAC risulti molto più accurato rispetto a quello calcolato con i Minimi Quadrati. In questo caso però si è notato la peculiarità di RANSAC di essere non deterministico, infatti la grande mole di dati inseriti, per i quali calcolare il modello, in alcuni casi sono stati approssimati con delle rette leggermente differenti da quella indicata nella figura 2.4; tale modello risultava comunque sempre più accurato di quello calcolato con i Minimi Quadrati che risulta essere sempre lo stesso in quanto è un algoritmo deterministico.

3.4 Implementazione dagli autori di RANSAC for Dummies

Nel [2, cha. 4] viene presentato un Toolbox di RANSAC per Matlab & Octave realizzato dall'autore del libro.
Inoltre nella sezione [2, sec. 4.2] sono contenuti alcuni esempi per stimare i parametri di rette e piani, nonché la rotazione e il ridimensionamento.
In particolar modo si vuole confrontare un esempio di stima lineare con il risultato esposto nella sezione 3.3.
È possibile trovare il codice dell'esempio in questione all'indirizzo:

`https://github.com/RANSAC/RANSAC-Toolbox/tree/master/Examples`.

L'esempio è *test_RANSAC_line_02.m* e utilizza il file *LineData.mat* dove sono contenuti i dati appartenenti a $\mathbb{R}^2$ di cui stimare il modello lineare (i dati sono in un numero N=10000).
Anche nell'esempio della sezione 3.3 sono stati utilizzati gli stessi dati e si possono mettere a confronto i due grafici di output che le due implementazioni di RANSAC restituiscono 2.4 e 2.5.

Come si può notare i due algoritmi restituiscono un output molto simile ignorando la casualità della scelta dei punti dell' MSS che sta alla base dell'algoritmo di RANSAC.
Dal punto di vista temporale, invece, questo secondo algoritmo risulta migliore rispetto a quello realizzato e spiegato nella sezione 3.3.

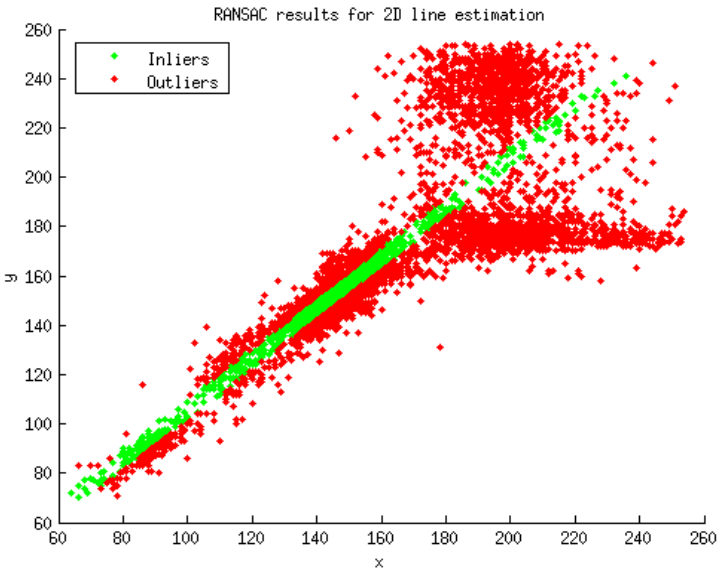

Figura 2.5: Grafico finale algoritmo RANSAC for Dummies

Si riporta di seguito una piccola tabella riassuntiva dei valori più rilevanti relativi al confronto dei due algoritmi.

	Confronto tra i due algoritmi		
Algoritmo	**# Inliers**	**Modello**	**# Iter.**
RANSAC_dataInput	4512	$y = 0.89x + 6.01$	106
RANSAC for Dummies	4508	$y = 0.69x + 5.67$	101

4 Considerazioni finali

Il calcolo sempre preciso del modello di regressione calcolato da RANSAC, anche in presenza di una grande mole di dati con molti outliers, lo rendono un algoritmo estremamente utilizzato in moltissime applicazioni. Si è potuto verificare, con questi semplici esempi, la potenza di tale algoritmo e la sua semplicità di realizzazione. RANSAC, infatti, a differenza di molti altri algoritmi di regressione, si basa su delle nozioni elementari. A mio parere, anche per la sua semplicità di realizzazione, RANSAC è un algoritmo a dir poco geniale.

Capitolo 3

Paragoni tra RANSAC e altri algoritmi

Dietro ogni problema c'è un'opportunità.
- Galileo Galilei -

Il problema della stima di modelli in presenza di outliers è sicuramente uno degli argomenti caldi delle odierne questioni computazionali.
I metodi di regressione robusti sono progettati per non essere eccessivamente colpiti da particolari valori anomali provenienti dal processo dei dati che vengono analizzati e per questo negli anni sono stati sviluppati metodi di regressione sempre più robusti alla presenza di outliers, sempre più veloci e precisi.
In particolar modo nella computer vision questa questione è diventata negli anni di centrale importanza.
Come dice l'autore di [8] il problema comune riscontrato nell'analisi delle scene dinamiche è quello della stima simultanea di modelli e dei loro parametri. Questo problema diventa sempre più pesante più il rumore di misura nei dati aumenta ed i dati sono ulteriormente danneggiati da valori anomali.

1 Panoramica sugli algoritmi utilizzati per l'identificazione degli outliers

Qui di seguito sono riportati alcuni esempi di tecniche per l'identificazione e la cancellazione degli outliers utilizzate in varie applicazioni.

In [9] l'autore, presentando un nuovo metodo per risolvere il problema della segmentazione del moto in dati video, descrive alcuni metodi per identificare ed eliminare gli outliers che nelle elaborazioni verrebbero considerati inliers, ad esempio utilizzando la normale regressione ortogonale (nella quale lo scopo è minimizzare la somma dei quadrati delle distanze tra i vari punti) tra ogni punto e l'iperpiano ottenuto come modello. In particolare presenta due metodologie per il rilevamento dei valori anomali:

- Nelle sezioni [9, sec 2/5] descrive un metodo che consiste nella valutazione dell'influenza che ha la cancellazione di ogni punto sulla soluzione finale. Quando c'è solo un singolo valore errato o un piccolo numero, il metodo funziona abbastanza bene. Tuttavia, il suo funzionamento non risulta buono per trovare valori outliers quando sono in un numero elevato o addirittura superano il numero dei dati inliers.
- Nelle sezioni [9, sec 6/7] invece presenta un metodo che applica la teoria del calcolo della matrice fondamentale [10] per la ricerca e la cancellazione degli outliers. (Nella computer vision, la matrice fondamentale è una matrice 3×3 che mappa i punti corrispondenti di due o più immagini stereo. Essa per ogni punto descrive una linea sull'altra immagine (una linea epipolare) in cui deve giacere il punto corrispondente).

In [11] ci si occupa dell'identificazione di dati anomali nella segmentazione di movimenti nei quali le traiettorie risultanti non sono necessariamente corrette e si propone quindi una tecnica per rimuove gli outliers basata sulla conoscenza che le traiettorie corrette sono considerate un sottospazio del loro dominio, basato quindi sulla separazione dei sottospazi.

In [12] ci si occupa della scomposizione del movimento. La capacità di estrarre sia la forma che il movimento da questa matrice non tiene conto della presenza di valori anomali, presenta quindi uno schema di correzione dei valori outliers che iterativamente aggiorna gli elementi della matrice di misurazione dell'immagine.

Questi approcci si basano su metodi di fattorizzazione o di separazione di sottospazi e hanno quindi un problema dal punto di vista computazionale in quanto devono operare su grandi matrici, senza contare che molti di questi metodi per essere applicati devono imporre limitazioni restrittive che ne limitano l'utilizzo.
Per questo motivo questi approcci molto spesso non risultano buoni e quindi vanno ricercate altre soluzioni che siano maggiormente implementabili e migliori dal punto di vista computazionale.

2 Support Vector Regression (SVR)

2.1 Panoramica generale su SVR

Le Support Vector Machines (SVM), o macchine kernel, sono un insieme di metodi di apprendimento supervisionato per la regressione e la classificazione di pattern, sviluppati negli anni '90 da Vladimir Vapnik ed il suo team presso i laboratori Bell ATT.
In particolare SVR è un algoritmo che serve a calcolare la funzione di regressione di una serie di punti, considerando la presenza di outliers.
Una delle idee più importanti in SVR è che la soluzione viene calcolata utilizzando un piccolo sottoinsieme di punti e ciò dà numerosi vantaggi dal punto di vista computazionale.
L'idea di SVR si basa sul calcolo di una funzione di regressione lineare in un altro spazio rispetto a quello di partenza dei dati, per cui i dati di input vengono mappati attraverso una funzione lineare in questo spazio.

2.2 Panoramica storica

[13] L'algoritmo SV è una generalizzazione non lineare dell'algoritmo di *Generalized Portrait* sviluppato in Russia negli anni Sessanta [Vapnik and Lerner, 1963, Vapnik and Chervonenkis, 1964]. Come tale è saldamente radicato nell'ambito della teoria statistica dell'apprendimento, o teoria VC, sviluppata nel corso degli ultimi tre decenni da Vapnik e Chervonenkis [1974]. In poche parole la teoria VC caratterizza le proprietà delle learning machines che consentono loro di generalizzare bene

i dati sconosciuti.
SVR è stato applicato in vari campi - serie temporali e previsioni finanziarie, approssimazione delle analisi ingegneristiche complesse, programmazione quadratica convessa, scelta di funzioni di perdita, ecc.

2.3 L'idea di base

Supponiamo di avere i dati di ingresso $\{\bar{x}_1, ...\bar{x}_l\} \subset X$, dove X denota lo spazio dei dati di ingresso (ad esempio $X = \mathbb{R}^d$). In SVR, questi dati di ingresso vengono prima mappati in uno spazio di funzioni m-dimensionale utilizzando una mappatura fissa, e quindi il modello viene costruito in questo nuovo spazio.
Ai dati di ingresso quindi viene associato un nuovo valore del tipo $\{(\bar{x}_1, \bar{y}_1)...(\bar{x}_l, \bar{y}_l)\} \subset X \times \mathbb{R}^m$ in quanto i punti $\bar{x}_i$ vengono mappati nel nuovo spazio di dimensione m.
In SVR l'obiettivo è quello di trovare una funzione $f(x)$, che approssimi in maniera ottima l'insieme dei dati di ingresso e che abbia al massimo uno scostamento pari a ϵ dagli obiettivi $\bar{y}_i$ per tutti i dati di ingresso.
Il modello prodotto da SVR dipende solo da un sottoinsieme dei dati di ingresso, perché la funzione di costo per la costruzione al modello ignora i dati di ingresso che sono vicino (entro la soglia ϵ) al modello predetto: viene infatti utilizzata la funzione di perdita ϵ-insensitive (1.3.3).
La formulazione di $f(x)$ nel caso di funzioni lineari è:

$$f(x) = \langle \omega, x \rangle + b \qquad con \quad \omega \in X, \quad b \in \mathbb{R} \tag{3.1}$$

dove $\langle \cdot, \cdot \rangle$ denota il prodotto scalare in X e b indica il termine di bias.

In [14, cha 5] vengono spiegati alcuni passaggi per ottenere le seguenti condizioni che verranno omessi di seguito.

SVR esegue la regressione lineare nella dimensione del nuovo spazio di funzioni in cui sono mappati i punti utilizzando la funzione di perdita ϵ-insensitive e, allo stesso tempo, cerca di ridurre la complessità del modello, quindi per rendere ottima la funzione lineare $f(x)$ dell'equazione (3.1) bisogna minimizzare la norma di ω ($||\omega||^2$).

Formalmente può essere scritto come un problema di ottimizzazione convessa che richiede:

$$di \quad minimizzare \qquad \frac{1}{2}||\omega||^2 \tag{3.2}$$

$$che \quad rispetti \qquad \begin{cases} y_i - \langle \omega, x_i \rangle - b \leq \epsilon \\ \langle \omega, x_i \rangle + b - y_i \leq \epsilon \end{cases} \tag{3.3}$$

L'assunzione che viene fatta in (3.2) e in (3.3) è che la funzione f, che approssima i punti $(\bar{x}_i, \bar{y}_i)$ con precisione ϵ, esiste, cioè che il problema di ottimizzazione convessa sia risolvibile.
Alcune volte, però, potrebbe non esistere f, oppure potrebbe esiste solo se viene considerato un errore che affligge i dati di ingresso.
Questo può essere descritto introducendo delle variabili slack [15] δ_i^- e δ_i^+, che rappresentano i vincoli superiore e inferiore per misurare la deviazione dei campioni al di fuori della zona ϵ-insensitive. Così il problema di ottimizzazione di (3.2) e (3.3) viene riformulato nel seguente modo:

$$minimizzare \qquad \frac{1}{2}||\omega||^2 + C\sum_{i=1}^{l}(\delta_i^+ +_i^-) \tag{3.4}$$

$$che \quad rispetti \qquad \begin{cases} y_i - \langle \omega, x_i \rangle - b \leq \epsilon + \delta_i^+ \\ \langle \omega, x_i \rangle + b - y_i \leq \epsilon + \delta_i^- \\ \delta_i^+, \delta_i^- \geq 0 \end{cases} \tag{3.5}$$

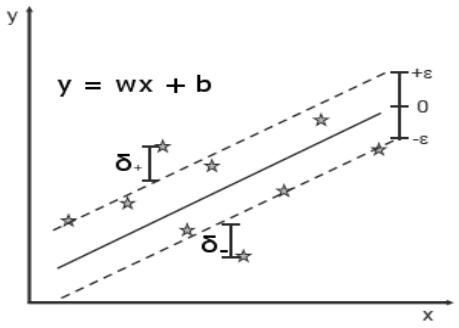

Figura 3.1: Spiegazione grafica del sistema

La costante $C > 0$ determina il compromesso tra la complessità del modello f (linearità) e il valore massimo di tolleranza di deviazioni maggiori di ϵ. Il parametro ϵ controlla la larghezza della zona ϵ-insensitive, utilizzata per adattare i dati di ingresso al modello.

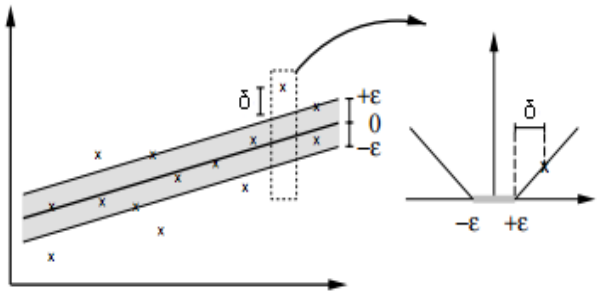

Figura 3.2: Funzione di costo ϵ-insensitive per impostare i margini di perdita

Il problema introdotto in (3.4) e (3.5) può essere risolto più facilmente con la formulazione duale introdotta in [16] . La soluzione finale di questa formulazione duale risulta essere:

$$\omega = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)x_i \qquad 0 \leq \alpha_i, \alpha_i^* \leq C \qquad i = 1, \ldots, l \tag{3.6}$$

dove α_i, α_i^* sono moltiplicatori di Lagrange della funzione di Lagrange costruita a partire da (3.4) e (3.5). Il valore di b nell'equazione 3.1 può essere determinato sostituendo l'equazione 3.6 nell'equazione 3.1, che risulta essere:

$$f(x) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\langle x_i, x\rangle + b \qquad con \quad \omega \in X, \quad b \in \mathbb{R} \tag{3.7}$$

(Per maggiori dettagli su questa soluzione si veda [13] o [16])
È noto che le prestazioni di SVR (accuratezza della stima) dipendono da una buona regolazione dei parametri C, ϵ e quelli legati al kernel. Le esistenti implementazioni software di SVR solitamente trattano questi parametri come ingressi definiti dall'utente. La selezione di un particolare tipo di kernel e i parametri della funzione del kernel sono di solito basati sulla conoscenza del dominio della regressione da effettuare che dovrebbe riflettere la distribuzione dei dati di ingresso.

3 Implementazione dell'algoritmo

Per poter confrontare i due algoritmi RANSAC e SVR è stata realizzata anche un'implementazione dell'algoritmo di SVR. L'implementazione in *Matlab* è stata realizzata basandosi su un algoritmo contenuto in toolbox reperibile a questo indirizzo web

`http://www.isis.ecs.soton.ac.uk/resources/svminfo/`

Abbiamo quindi provato a eseguire l'algoritmo di SVR su dati uguali a quelli utilizzati per l'algoritmo di RANSAC nel capitolo 2.

3.1 In cosa consiste il confronto

È stata realizzata un'applicazione Matlab che, dato un insieme di dati generati casualmente, confronta i risultati dei tre algoritmi (RANSAC, SVR e Minimi Quadrati) nel calcolo del modello lineare che approssima in modo migliore l'andamento di tali dati.
I dati generati sono disposti lungo la retta $y = x$ con rumore gaussiano additivo e per metà di questi è stato generato un errore casuale molto grande che tende a renderli degli outliers rispetto all'andamento, come per i dati generati in 2.
L'applicazione quindi prende questi dati di ingresso e calcola i tre modelli con i diversi algoritmi.
Sono state fatte diverse prove e di seguito vengono riportati i grafici che rappresentano i modelli finali calcolati con i tre algoritmi al variare del numero di dati in input (tenendo costante al 50% il numero di dati outliers) da un minimo di N=0 dati (Fig. 3.3) ad un massimo di N=100 dati (Fig. 3.6).
Sono state fatte numerose prove su dati casuali e di seguito sono riportati i grafici e le considerazioni che si sono potute dedurre nei vari esperimenti.
Sono stati riscontrati dei casi peculiari che non rispettano esattamente le considerazioni seguenti, ma tali casi si sono verificati solo con la generazione casuale di dati particolari, In queste condizioni l'algoritmo di SVR restituiva un modello non del tutto simile a quelli presentati di seguito.

3.2 Output del confronto

Si può notare dalle figure seguenti come SVR restituisca un modello estremamente accurato se i dati sono in numero limitato (confrontato con il modello reale): l'output dell'algoritmo risulta corretto e con tempi di calcolo buoni, inoltre non avendo una base randomica il risultato ha sempre la stessa precisione e risulta estremamente robusto, soprattutto con un numero molto ristretto di dati. RANSAC invece in alcune, poche, occasioni restituisce un risultato non coincidente con il modello reale proprio a causa della sua natura casuale. I minimi quadrati, non essendo robusti alla presenza di outliers, restituiscono un modello molto differente rispetto agli altri due algoritmi.(Fig. 3.3) e (Fig. 3.4).

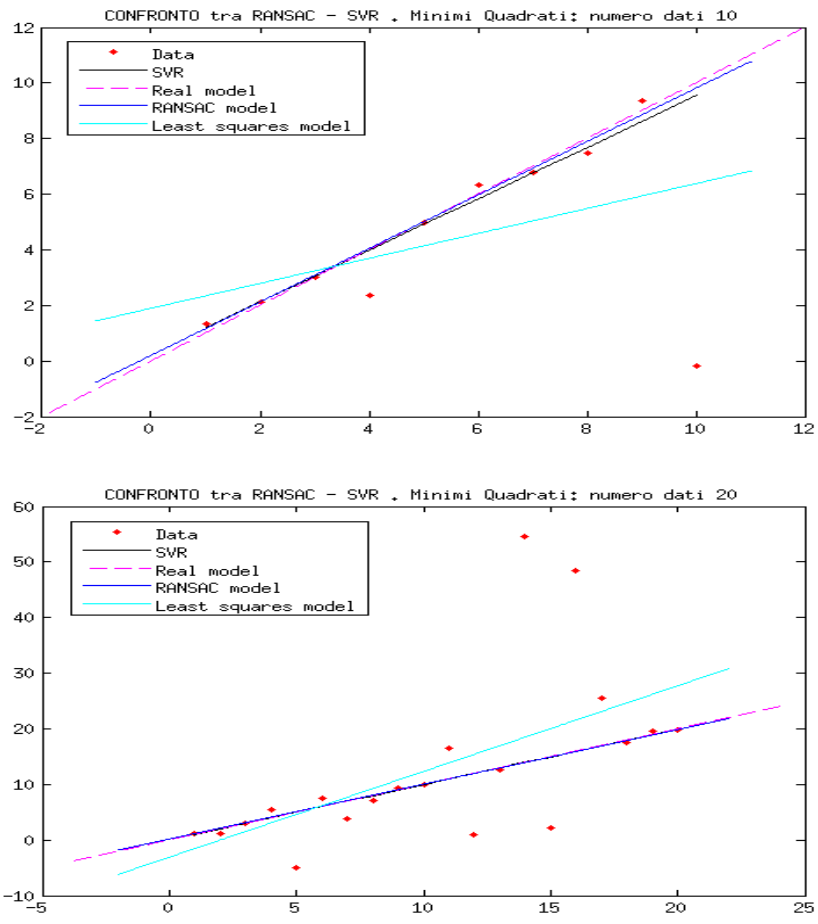

Figura 3.3: Confronto RANSAC - SVR - Minimi Quadrati – N = 10,20

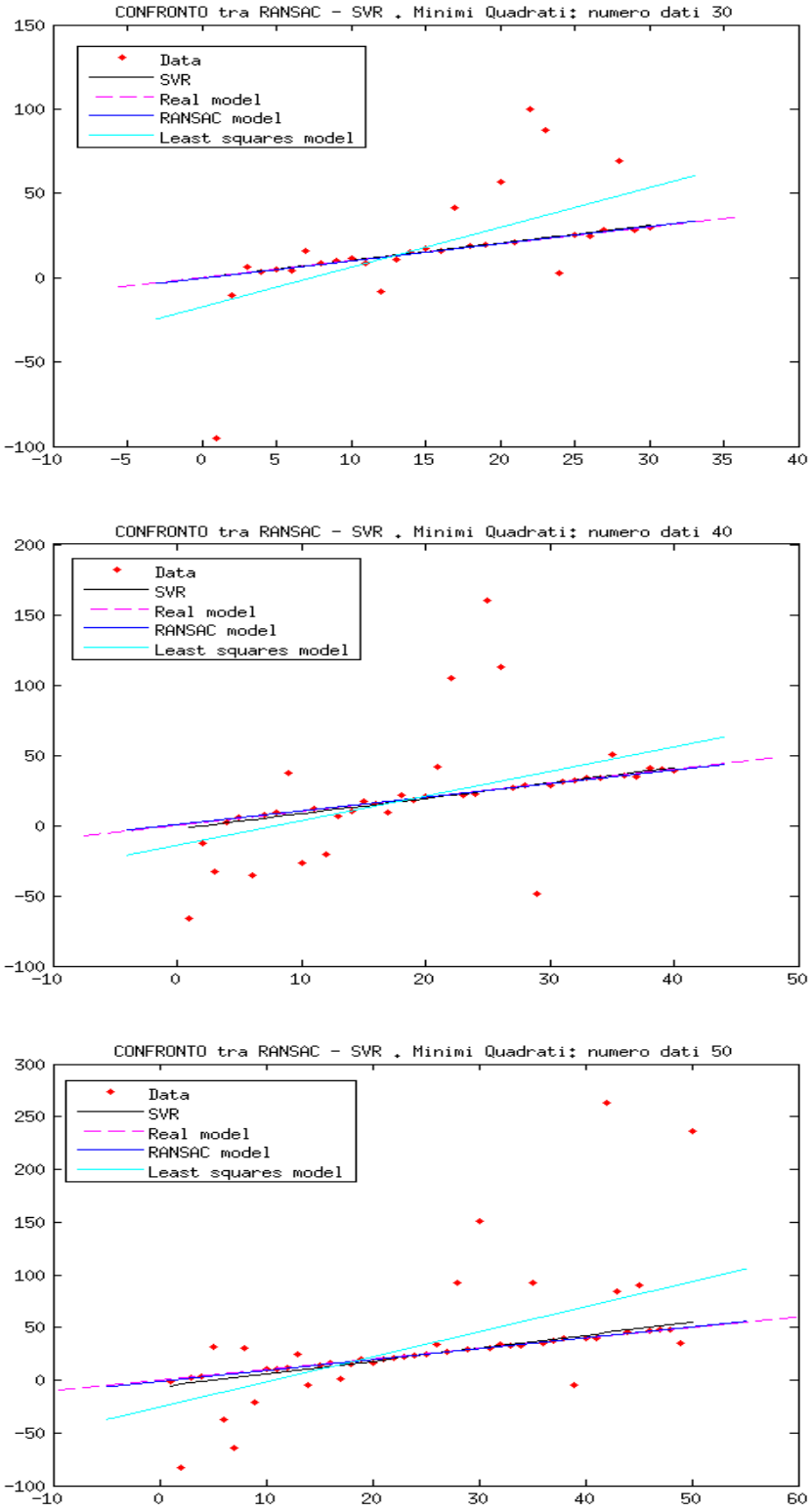

Figura 3.4: Confronto RANSAC - SVR - Minimi Quadrati – N = 30,40,50

Se però i dati cominciano ad aumentare, e anche gli outliers di conseguenza, il modello calcolato da SVR rispetto a quello di RANSAC risulta peggiore, i dati anomali influenzano in modo scorretto l'algoritmo che tende a rispondere con un modello non esatto (Fig. 3.5) e (Fig. 3.6).

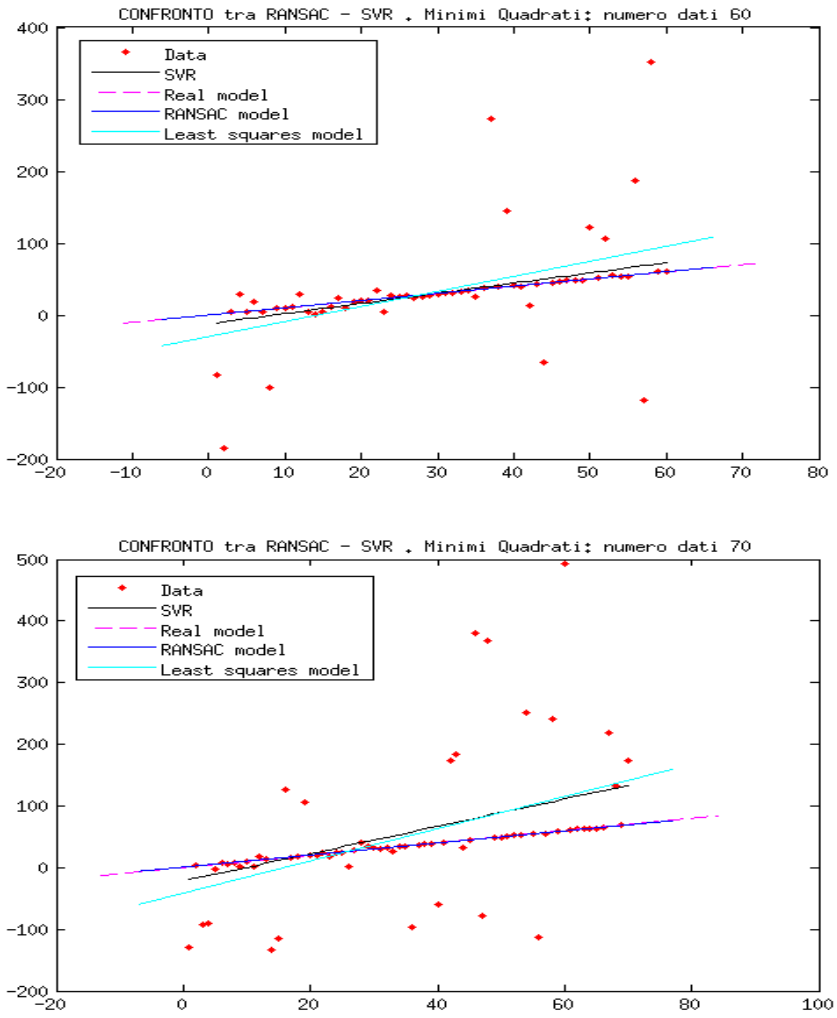

Figura 3.5: Confronto RANSAC - SVR - Minimi Quadrati – N = 60, 70

Si può notare, dalle numerose prove fatte, che il modello calcolato da SVR, da 80 dati in su, tende ad essere peggiore di quello calcolato con i minimi quadrati. RANSAC invece risulta sempre molto robusto anche all'aumentare dei dati di input.

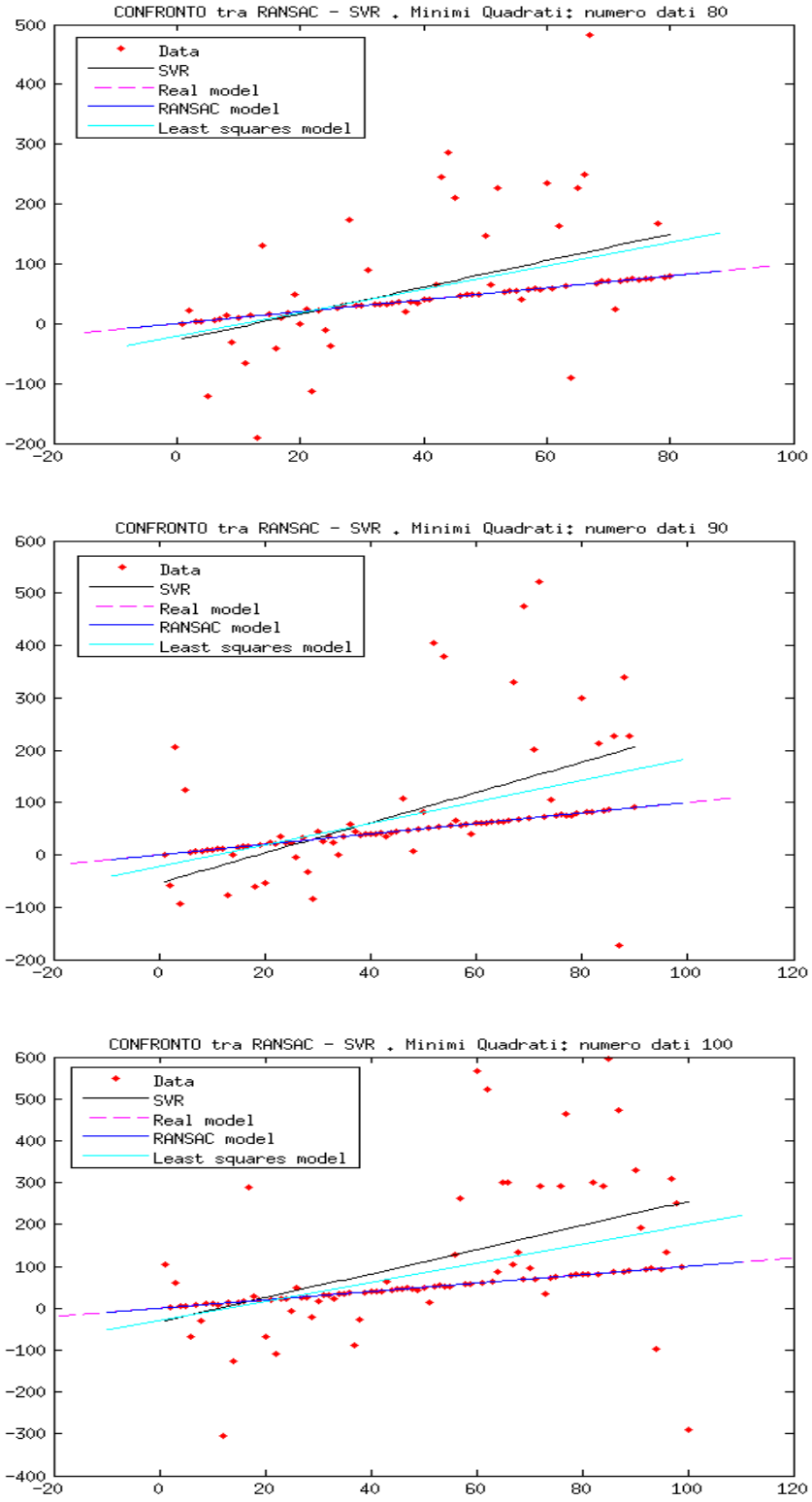

Figura 3.6: Confronto RANSAC - SVR - Minimi Quadrati – N = 80,90,100

3.3 Riflessioni Finali

L'algoritmo di SVR seppur restituendo modelli davvero molto accurati, alcune volte più precisi di RANSAC, il quale ha un aspetto di casualità intrinseco che può far variare il risultato finale, ha tempi di esecuzioni molto superiori rispetto a RANSAC.
Se il numero di dati supera una certa soglia, e comunque sulle grandi moli di dati, SVR richiede un calcolo computazionale e di ottimizzazione che diventa insostenibile all'aumentare dei dati di input.
Ad esempio nella prova esposta nella sotto sezione 3.1 i tempi computazionali dell'algoritmo SVR sono enormemente più lunghi anche dell'algoritmo di RANSAC realizzato (2), che non ha nessuna ottimizzazione dal punto di vista temporale.
In aggiunta si è verificato che la risposta di questa implementazione dell'algoritmo all'aumentare dei dati tende a essere spesso nulla, cioè a non riuscire a trovare il modello.

Capitolo 4

Applicazione di RANSAC alla visione computazionale

Make everything as simple as possible,
but not simpler.
- Albert Einstein -

È stata realizzata un'implementazione dell'algoritmo di SIFT (Scale-Invariant Feature Transform) in Matlab che verrà poi utilizzata in una semplice applicazione di image stitching (cuciture di immagini) e successivamente ottimizzata con l'algoritmo di RANSAC esposto nel primo capitolo.

1 L'algoritmo di SIFT

1.1 Panoramica storica

SIFT è un algoritmo utilizzato in computer vision che permette di rilevare e descrivere caratteristiche, o feature, locali in immagini.
SIFT è utilizzato in molte applicazioni (1.3) che includono: object recognition, robotic mapping e navigation, image stitching, modellazione 3D, riconoscimento dei gesti, video tracking, e match moving. L'algoritmo è stato pubblicato da David G. Lowe nel 1999 ed è stato brevettato negli Stati Uniti; il proprietario è la University of British Columbia.

1.2 Come funziona

SIFT rileva i punti stabili di un oggetto in modo tale che lo stesso oggetto possa essere riconosciuto con varianza di illuminazione, scala, rotazione e trasformazioni affini.
[17] Grazie alla sua invarianza al cambio di illuminazione, scala e rotazione, l'algoritmo di SIFT viene utilizzato per l'acquisizione di immagine in qualsiasi condizione: questo è un grande vantaggio per quanto riguarda l'utente che non ha particolari limitazioni e può svolgere i propri lavori anche in ambienti non cooperativi.

I passi principali dell'algoritmo di SIFT sono i seguenti:

1. **Individuazione degli estremi locali nello scale-space**: si cercano punti interessanti su tutte le scale e posizioni dell'immagine utilizzando una funzione DoG (Difference of Gaussian) [18]. L'approccio utilizzato è quello del filtraggio in cascata (cascade filtering approach), che consente di determinare le posizioni e la scala delle feature candidate ad essere punti chiave e che, in un secondo momento, vengono caratterizzate con maggior dettaglio.
2. **Localizzazione dei keypoints**: per ciascun punto candidato viene costruito un modello dettagliato per determinarne posizione e scala. I punti vengono inoltre selezionati secondo misure di stabilità.
3. **Generazione delle orientazioni**: per ottenere l'invarianza rotazionale, ad ogni punto chiave (keypoint) vengono associate una o più orientazioni calcolate in base ai gradienti locali dell'immagine.
4. **Generazione del descrittore**: a partire dai gradienti locali dell'immagine, dalla scala selezionata e dall'intorno del punto chiave, viene costruito il descrittore.

1.3 Campi di utilizzo

L'algoritmo di SIFT viene utilizzato in moltissimi campi della computer vision:

- Nell'articolo [19] utilizzano l'algoritmo di SIFT per il rilevamento stabile e la rappresentazione dei descrittori locali che sono una componente fondamentale di molti algoritmi di registrazione di

immagini e di riconoscimento visivo in quanto possono essere calcolati in modo efficiente, sono resistenti alla parziale occlusioni e sono relativamente insensibili ai cambiamenti di punto di vista. Questo utilizzo dell'algoritmo di SIFT è comunemente impiegato in molte applicazioni come il riconoscimento di oggetti e il recupero di immagini; SIFT infatti è l'algoritmo più resistente alle deformazioni delle immagini, codificando gli aspetti salienti dell'immagine come il gradiente nelle vicinanze di un punto della funzione.

- Nell'articolo [20] invece si parla di un nuovo utilizzo dell'algoritmo di SIFT leggermente modificato per catalogare immagini che descrivono scene differenti, ad esempio per allineare immagini temporalmente adiacenti, invece che rispetto allo spazio. L'algoritmo si chiama SIFT flow e permette l'allineamento di un'immagine con la sua più vicina in un insieme contenente una grande varietà di scene. Gli autori propongono un framework per un database di analisi e sintesi di immagini che restituisca, fatta una query con un'immagine, l'insieme delle scene con più alta corrispondenza, applicata al riconoscimento facciale, alle immagini satellitari e alla sintesi del movimento.
- Nell'articolo [21] l'algoritmo di SIFT si utilizza per il riconoscimento di oggetti. Il processo consiste nel confrontare un'immagine data, di cui bisogna riconoscere il contenuto, con un database di immagini di oggetti conosciuti utilizzando un algoritmo di nearest-neighbor seguito da una trasformata di Hough [22] per identificare gruppi appartenenti ad un singolo oggetto e infine compiere una verifica. Questo approccio riesce ad identificare in maniera robusta oggetti anche se l'immagini sono distorte o parziali.
- Nell'articolo [23] si presenta un algoritmo di SIFT per il 3D (3 dimensioni) per i video o le immagini 3D come le MRI (3D Magnetic Resonance Imaging). Questo algoritmo rappresenta molto meglio di tanti altri la natura tridimensionale dei video nelle applicazioni di riconoscimento di azioni e movimenti. Per la classificazione di immagini e riconoscimento di oggetti nei video viene utilizzato il paradigma delle bag of keywords [24], e poi viene presentato un

metodo per trovare le relazioni spazio-temporali tra le chiavi in modo da descrivere in maniera migliore il video.

1.4 Problematiche

In alcune applicazioni dell'algoritmo di SIFT possono insorgere alcune problematiche a causa di una valutazione delle caratteristiche dei keypoints solo in un intorno limitato di questi punti.
Infatti è sicuramente possibile analizzare immagini in cui punti differenti assumono descrittori molto simili, se questi vengono analizzati in un intorno molto ristretto.
Questo comporta che in alcuni casi si possono trovare dei risultati non consistenti con la realtà, ed è quindi necessario analizzare ed elaborare i dati provenienti da questo algoritmo prima di poterli utilizzare. Spesso una prima fase di analisi consiste nel filtrare tali valori con algoritmi di regressione che eliminano i dati che non sono buoni lasciando solo quelli utilizzabili.
Un esempio concreto di questa problematica, e una possibile soluzione utilizzando RANSAC, è presentato nella sezione seguente, mentre l'immagine 4.1 mostra come l'algoritmo può identificare due keypoint con descrittori simili anche se questi in realtà non sono punti che combaciano.

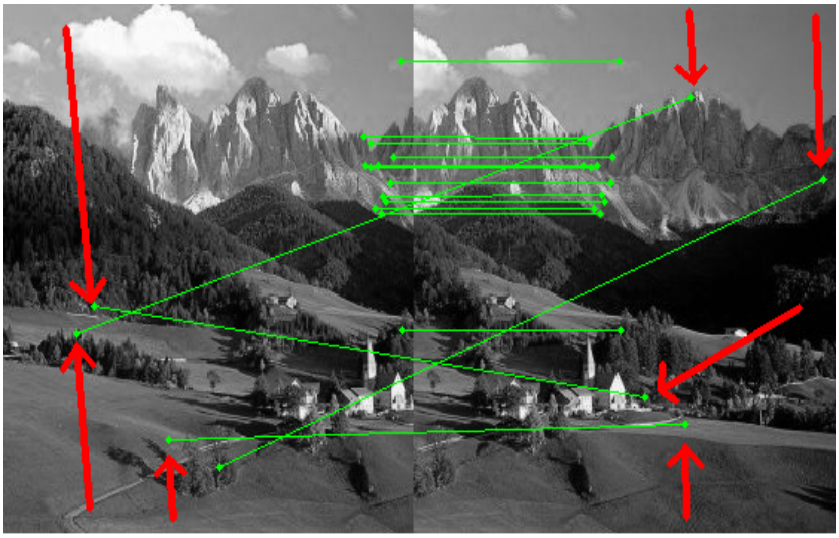

Figura 4.1: Esempio problematiche SIFT

2 Esempio di utilizzo

2.1 Presentazione dell'applicazione di SIFT

Oltre agli utilizzi esposti nella sottosezione 1.3, SIFT può essere utilizzato nell'unione delle immagini per la ricostruzione completamente automatica di panorami partendo da immagini non panoramiche. Le caratteristiche SIFT estratte dalle immagini in ingresso vengono confrontate tra loro per trovare il sottoinsieme di punti che più combaciano per ogni caratteristica. Queste corrispondenze vengono poi utilizzate per trovare m punti candidati per ciascuna immagine. Ora le vere omografie tra coppie di immagini vengono calcolate usando RANSAC e un modello probabilistico viene utilizzato per la verifica.
Utilizzando l'algoritmo di SIFT per la cucitura di panorama, il sistema risultante è insensibile all'ordinamento, l'orientamento, la scala e l'illuminazione delle immagini. Le immagini di input possono contenere inoltre panorami differenti e contenere rumore (possono essere inserite anche immagini completamente estranee), ma le sequenze panoramiche vengono correttamente riconosciute e restituite in uscita.[25]

In questa sezione sarà quindi presentata una semplice implementazione dell'algoritmo di SIFT che consiste nel trovare i punti in comune tra due foto, che presentano per l'appunto una zona condivisa (come potrebbe accadere nello scatto di una foto panoramica in cui le foto in sequenza presentano una fascia uguale che va individuata e sovrapposta). Si mostreranno i limiti dell'algoritmo che propone un risultato con molti punti anomali e successivamente un miglioramento del risultato grazie alla robusta individuazione degli outliers da parte di RANSAC.

Da sottolineare che l'algoritmo di SIFT utilizzato in questo esempio non è stato realizzato personalmente: l'algoritmo è Copyright (c) 2006 The Regents of the University of California ed è stato creato da Andrea Vedaldi presso UCLA Vision Lab Department of Computer Science.

2.2 Preparazione input

Vengono scattate due foto di un paesaggio, spostando di poco il punto di vista, in modo tale che la prima foto abbia la fascia a destra in comune con la fascia sinistra della seconda immagine (in questo esempio sono state scattate due foto del Dipartimento di Ingegneria dell'Informazione con le suddette caratteristiche, figura 4.2).

Figura 4.2: Foto iniziali

Successivamente le foto vengono modificate con un software di elaborazione foto per renderle di dimensioni minori, per rendere più veloce l'applicazione dell'algoritmo di SIFT la cui complessità temporale dipende dal numero di punti per foto. Inoltre vengono rese in bianco e nero perché l'implementazione dell'algoritmo di SIFT utilizzata richiede che venga passato come argomento una matrice di double normalizzati che si riferiscono ad una foto in bianco e nero (nel nostro esempio figura 4.3)

(a) I_1 (b) I_2

Figura 4.3: Elaborazione delle foto iniziali

2.3 Primo output

Viene quindi richiamato l'algoritmo di SIFT per entrambe le immagini I_1 e I_2 elaborate, la chiamata nel toolbox utilizzato ':

$$[frames, descriptors, gss, dogss] = sift(I, varargin)$$

estrae le cornici (frames) SIFT e i loro descrittori (descriptors) dall'immagine I. L'immagine I ($I = I_1$ o I_2), come già detto, deve essere in scala di grigi, di tipo double con un valore tra 0 e 1.
Restituisce:

- frames è una matrice 4xK che memorizza un SIFT frame per colonna.
- descriptor è una matrice DxK che memorizza un descriptor per colonna (solitamente D=128)
- gss è la gaussiana calcolata dall'algoritmo
- dogss è la differenza di gaussiane (DoG) calcolata dall'algoritmo

Utilizzando i descriptors restituiti da entrambe le chiamate a SIFT relative alle due immagini I_1 e I_2 si chiama a questo punto la funzione per trovare i punti che le due immagini hanno in comune; nel toolbox utilizzato la chiamata alla funzione è

$$matches = siftmatches(descr1, descr2, thresh)$$

La funzione usa lo stesso algoritmo suggerito da D. Lowe in [21] per trovare i match che sono troppo ambigui con la specifica soglia thresh. Un descriptor D_1 trova corrispondenza con un descriptor D_2 se e solo se la distanza $d(D_1, D_2)$ moltiplicata per la soglia (thresh) non è più grande della distanza tra D_1 e tutti gli altri descriptors.

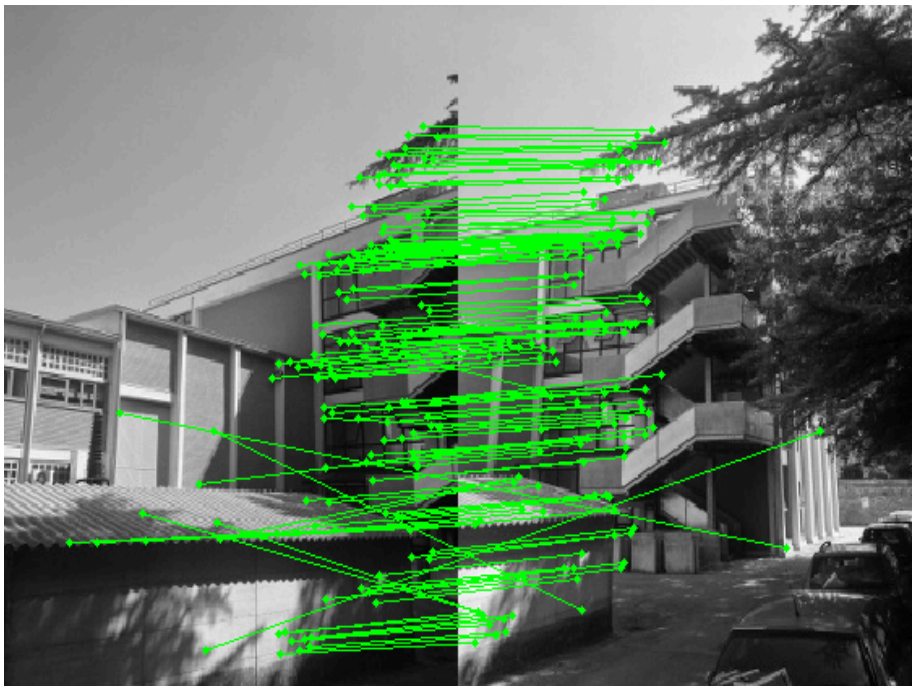

Figura 4.4: Primo output dell'algoritmo di SIFT

Si può osservare che l'algoritmo di SIFT applicato senza alcun accorgimento non restituisce un risultato accurato. I punti individuati come corrispondenze tra le due immagini non sono tutti corretti e vanno scremati.

2.4 Ottimizzazione con RANSAC

Il risultato dell'algoritmo di SIFT 4.4 viene quindi analizzato con RANSAC per poter individuare le corrispondenze corrette.
Innanzitutto si sono analizzati separatamente i due gruppi di keypoints delle due immagini distinte I_1 e I_2; è stato applicato il calcolo della regressione lineare dell'andamento delle loro posizioni con RANSAC per poter individuare i dati in posizione anomala, cioè che non appartengono ad una fascia che dovrebbe corrispondere alla zona in comune tra le due foto. Infatti in una singola immagine, i punti che individuano le corrispondenze si distribuiscono in un'unica fascia dell'immagine, quindi considerando la posizione orizzontale di questi punti abbiamo calcolato il modello di andamento ed eliminato le corrispondenze in posizioni lontane dal resto dei punti. 4.5 Tutti i punti la cui posizione si discosta molto dall'andamento generale vengono eliminati come corrispondenze trovate (e di conseguenza viene eliminato il match nell'altra immagine).

```
points_fig1 = [ frames1(1,matches(1,:)) ; frames1(2,
   matches(1,:))];
points_fig2 = [ frames2(1,matches(2,:)) ;  frames2
   (2,matches(2,:))];
ransac_dataInput(points_fig1,0.95,0.0001,25,struct('
   history',0,'compare',1));
ransac_dataInput(points_fig2,0.95,0.0001,25,struct('
   history',0,'compare',1));
```

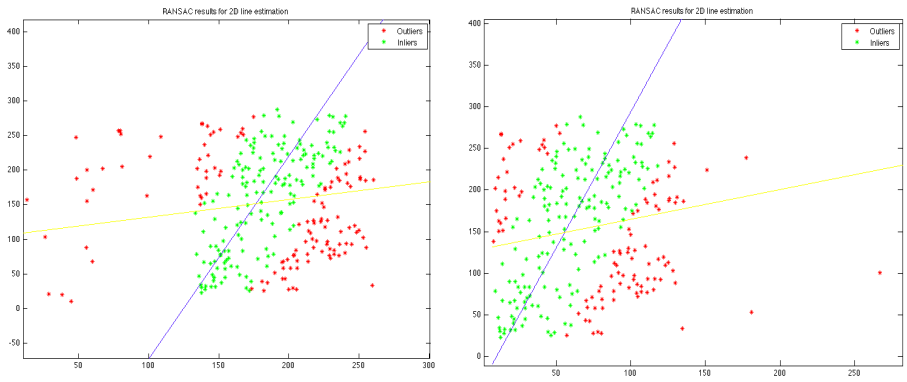

Figura 4.5: Ottimizzazione con Ransac passo 1

Si può notare che però esistono delle coppie di punti che si trovano nella fascia di corrispondenze, ma non costituiscono delle vere corrispondenze in quanto associano due punti nella zona giusta, ma ad altezze diverse (fig. 4.6). Queste non sono state eliminate dalla prima scrematura utilizzando RANSAC in quanto sia in un'immagine, che nell'altra, i punti delle corrispondenze hanno una posizione orizzontale simile a quella di tutti i match veri.

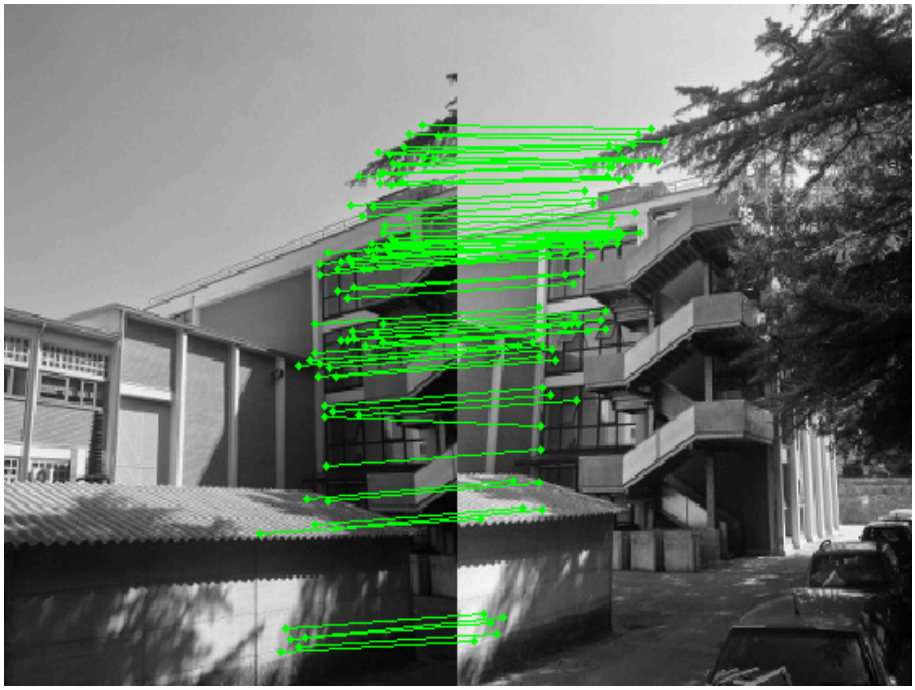

Figura 4.6: Output di SIFT dopo il primo passo dell'ottimizzazione

Si opera quindi una seconda regressione sulla totalità dei dati andando a considerare anche le altezze delle coppie selezionate dalla prima scrematura in modo da riuscire a rilevare le coppie di punti che non rispettano l'andamento generale (Fig. 4.7). Viene infatti calcolata la regressione lineare prendendo i keypoints selezionati dalla prima scrematura, $(match1, match2)$, che vengono plottati utilizzando una formula che associa a tutte le coppie che non sono corrispondenze corrette (che quindi hanno posizioni verticali reciproche, considerando i due punti della corrispondenza, dissimili da tutte le altre coppie) un punto che, rispetto all'andamento generale, risulta essere un outlier.

```
match_temp = (intersect(match1',match2','rows'))';
X_fig = frames1(1,match_temp(1,:)) - frames2(1,
   match_temp(2,:));
Y_fig = frames1(2,match_temp(1,:)) - frames2(2,
   match_temp(2,:));
match_fig = [ X_fig ; Y_fig ];
ransac_dataInput(match_fig,0.95,0.0001,1,struct('
   history',0,'compare',1));
```

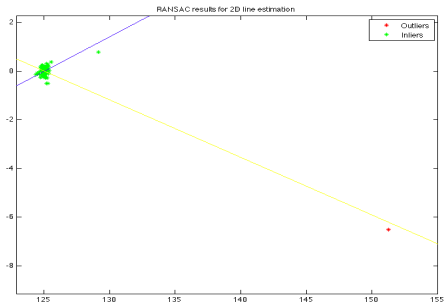

Figura 4.7: Ottimizzazione con Ransac passo 2

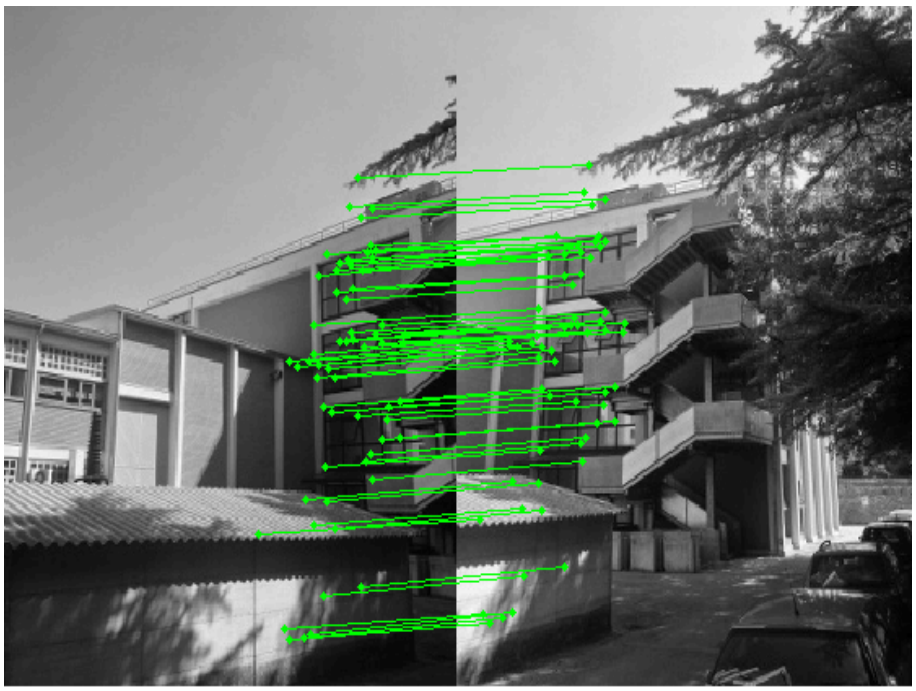

Figura 4.8: Output finale dell'algoritmo di SIFT dopo l'ottimizzazione con RANSAC

Le due scremature fatte con RANSAC eliminano effettivamente in modo corretto tutti i dati anomali, lasciando solo le coppie di dati corrette, che identificano corrispondenze tra le immagini. Vengono eliminate anche delle coppie che erano evidentemente corrette, ma questo fa parte degli algoritmi che computano le regressioni che identificano l'andamento generale, ma non è detto che individuino tutti i dati inliers, infatti lo scopo primo è eliminare i dati anomali in modo da poter ottenere sicuramente dei dati corretti, anche se questi non sono tutti i dati disponibili all'inizio.

Conclusioni

It is not in the stars to hold our destiny
but in ourselves.
- William Shakespeare -

Le problematiche relative alla modellazione di dati in presenza di outliers sono un argomento centrale nei problemi di stima di parametri affrontati oggigiorno dalla comunità scientifica.
La scrittura di questa tesi è stata molto utile per avvicinarmi ad un mondo tanto vasto e farmi rendere conto di quali siano le caratteristiche, le problematiche e i possibili sviluppi di questo settore.

Per quanto riguarda l'algoritmo di RANSAC, dopo questo studio, si può capire perché sta alla base di moltissime applicazioni della computer vision: la sua semplicità di implementazione, la robustezza e la sua velocità lo fanno essere estremamente duttile e utilizzabile in tantissimi campi.
La forza di RANSAC sta proprio nella sua semplicità, unita alla potenza della teoria della probabilità che trasforma un semplice meccanismo per la stima robusta in un vero e proprio algoritmo che può restituire risultati con una precisione elevata a piacere, un algoritmo che non ha nulla da invidiare a tutti gli altri che si possono applicare allo stesso problema, anzi può vantare caratteristiche e prestazioni eccezionali.

Con il confronto fatto nel capitolo 3 si è potuto valutare concretamente la robustezza di RANSAC che a confronto con le stime prodotte dall'algoritmo di SVR e dei minimi quadrati è risultato, nelle prove fatte, sempre migliore.

Analizzare in maniera approfondita l'algoritmo di RANSAC, realizzare un implementazione per capirne il funzionamento, confrontarlo con altri algoritmi paragonabili e poi applicarlo in una realizzazione pratica, sono state tutte fasi davvero interessanti che mi hanno consentito di concretizzare un semplice aspetto della teoria della modellistica.

Questa tesi non voleva essere un lavoro di ricerca: è iniziata come un semplice approfondimento che però ha portato riscontri anche dal punto di vista pratico con confronti e applicazioni funzionanti. I risultati ottenuti, seppur non rivoluzionari, hanno confermato e rimarcato tutti gli aspetti teorici studiati inizialmente, ma soprattutto hanno costituito un valore aggiunto personale dal punto di vista formativo e dell'esperienza fatta.

Giunto alla conclusione di questa tesi posso ritenermi davvero soddisfatto del lavoro portato a termine.

Appendice A

Codice Matlab

1 Ransac_dataRandom

```
function [bestModel,finalInliers,finalOutliers,
   finalError] = ransac_dataRandom(N,Pin,e,options)

%create the vector of data
data = [];
%boolean var
foundModel = 0;

%number of iterations is calculate considering the
   maximum probability to
%   find outlier in the data and the probability
   that the data in input
%   are inliers
iterations = log(e)/log(1-(N/2*Pin*(N/2*Pin-1))/(N*(
   N-1))) + 1;

%threshold error is calculate iverting the gaussian
   function
%   P[distance(data,model)<=thresError] = Pin
errorThreshold = sqrt(chi2inv(Pin,2));

% set the outputs

bestModel = [];
finalInliers = [];
finalOutliers = [];
finalError = inf;

%generate the data with outlier
for i=1:N
    data(1,i)= i+randn(1);
    data(2,i)= i+randn(1);
```

```
28 end
29
30 randomP = randperm(N);
31 for i=1:N/2
32     if(randomP(i)>N/2)
33         data(2,randomP(i))=randomP(i)+randomP(i)*log
              (i)*(randn(1)+0.5);
34     else
35         data(2,randomP(i))=randomP(i)-(N/2 - randomP
              (i))*log(i)*(randn(1)+0.5);
36     end
37 end
38
39 randomP = randperm(N);
40
41 %start RANSAC algorithm
42 for i=1:iterations
43
44     if(iterations < N)
45         p1 = [ data(1,randomP(2*i-1)), data(2,
              randomP(2*i-1)) ];
46         p2 = [ data(1,randomP(2*i)), data(2,randomP
              (2*i)) ];
47     else
48         randomP = randperm(N);
49         p1 = [ data(1,randomP(1)), data(2,randomP(1)
              ) ];
50         p2 = [ data(1,randomP(2)), data(2,randomP(2)
              ) ];
51     end
52
53     inliers = [];
54     outliers = [];
55     ni = 0;
56     no = 0;
57
58     inliers(:,ni+1) = p1;
59     ni = ni+1;
60     inliers(:,ni+1) = p2;
61     ni = ni+1;
62
63     a_model = ( p1(2) - p2(2) )/ ( p1(1) - p2(1) );
64     b_model = p1(2) - a_model*p1(1);
65
66     totalError = 0;
67
68     if(iterations < N)
69         for p=2*i+1:N+2*(i-1)
70             if(p<=N)
71                 point = data(:,randomP(p));
72             else
73                 point = data(:,randomP(p-N));
```

```
74              end
75
76              error = abs( point(1)*a_model + b_model
                   - point(2) )/ sqrt( a_model^2 + 1);
77
78              if( errorThreshold < error )
79                  totalError = totalError +
                       errorThreshold;
80                  outliers(:,no+1) = point;
81                  no = no+1;
82              else
83                  totalError = totalError + error;
84                  inliers(:,ni+1) = point;
85                  ni = ni+1;
86              end
87           end % data iteration end
88      else
89          for p=3:N
90              point = data(:,randomP(p));
91
92              error = abs( point(1)*a_model + b_model
                   - point(2) )/ sqrt( a_model^2 + 1);
93
94              if( errorThreshold < error )
95                  totalError = totalError +
                       errorThreshold;
96                  outliers(:,no+1) = point;
97                  no = no+1;
98              else
99                  totalError = totalError + error;
100                 inliers(:,ni+1) = point;
101                 ni = ni+1;
102             end
103          end % data iteration end
104     end
105
106     %history
107     if(isfield(options, 'history') && options.
           history == true)
108         %m = max(data(:));
109         x = -N/10:1/100:N+N/10;
110         figure
111         p = plot(data(1,:),data(2,:),'*');
112         set(p,'color','r');
113         hold on;
114         in = plot(inliers(1,:),inliers(2,:),'*');
115         set(in,'color','g');
116         hold on;
117         model = plot(x,x*a_model+b_model);
118         set(model,'color','b','LineWidth',1);
119         hold on;
```

```
120         baund1 = plot(x,x*a_model+b_model+
               errorThreshold*sqrt( a_model^2 + 1));
121         set(baund1,'color','c','LineWidth',1, '
               Linestyle', '- -');
122         hold on;
123         baund2 = plot(x,x*a_model+b_model-
               errorThreshold*sqrt( a_model^2 + 1));
124         set(baund2,'color','c','LineWidth',1, '
               Linestyle', '- -');
125         hold on;
126     end
127
128     % check model
129
130     if ( finalError > totalError && N/2*Pin <= ni )
131         bestModel = [a_model,b_model];
132         finalInliers = inliers;
133         finalOutliers = outliers;
134         finalError = totalError;
135         foundModel = 1;
136     end
137
138
139 end % main iteration end
140
141 %calculate the minimum square model
142 pol = polyfit(data(1,:),data(2,:),1);
143
144 %plot
145 if(foundModel == 0)
146     disp('I did not find a model');
147 else
148   figure
149   p = plot(data(1,:),data(2,:),'*');
150   set(p,'color','r');
151   hold on;
152   in = plot(finalInliers(1,:),finalInliers(2,:),'*')
         ;
153   set(in,'color','g');
154   hold on;
155   %m = max(data(:));
156   x = -N/10:1/100:N+N/10;
157   model = plot(x,x*bestModel(1)+bestModel(2));
158   hold on;
159   set(model,'color','b','LineWidth',1);
160   title('RANSAC results for 2D line estimation');
161   legend('Outliers', 'Inliers');
162
163   %compare
164   if(isfield(options, 'compare') && options.compare
         == true)
165       model2 = plot(x,x*pol(1)+pol(2));
```

```
166       set(model2,'color','y');
167       hold on;
168     end
169 end
170 
171 return;
```

2 Ransac_dataInput

```
1  function [bestModel,finalInliers,finalOutliers,
       finalError] = ransac_dataInput(data,Pin,e,sigma,
       options)
2  
3  N = length(data);
4  %boolean var
5  foundModel = 0;
6  
7  %number of iterations is calculate considering the
       maximum probability to
8  %   find outlier in the data and the probability
       that the data in input
9  %   are inliers
10 iterations = log(e)/log(1-(N/2*Pin*(N/2*Pin-1))/(N*(
       N-1))) + 1;
11 
12 %threshold error is calculate iverting the gaussian
       function
13 %   P[distance(data,model)<=thresError] = Pin
14 errorThreshold = sigma*sqrt(chi2inv(Pin,2));
15 
16 % set the outputs
17 bestModel = [];
18 finalInliers = [];
19 finalOutliers = [];
20 finalError = inf;
21 
22 randomP = randperm(N);
23 
24 %start RANSAC algorithm
25 for i=1:iterations
26 
27     if(2*iterations < N)
28         p1 = [ data(1,randomP(2*i-1)), data(2,
               randomP(2*i-1)) ];
29         p2 = [ data(1,randomP(2*i)), data(2,randomP
               (2*i)) ];
30     else
31         randomP = randperm(N);
32         p1 = [ data(1,randomP(1)), data(2,randomP(1)
               ) ];
```

```
33          p2 = [ data(1,randomP(2)), data(2,randomP(2)
                ) ];
34      end
35
36      inliers = [];
37      outliers = [];
38      ni = 0;
39      no = 0;
40
41      inliers(:,ni+1) = p1;
42      ni = ni+1;
43      inliers(:,ni+1) = p2;
44      ni = ni+1;
45
46      a_model = ( p1(2) - p2(2) )/ ( p1(1) - p2(1) );
47      b_model = p1(2) - a_model*p1(1);
48
49      totalError = 0;
50
51      if(2*iterations < N)
52          for p=2*i+1:N+2*(i-1)
53              if(p<=N)
54                  point = data(:,randomP(p));
55              else
56                  point = data(:,randomP(p-N));
57              end
58
59              error = abs( point(1)*a_model + b_model
                    - point(2) )/ sqrt( a_model^2 + 1);
60
61              if( errorThreshold < error )
62                  totalError = totalError +
                        errorThreshold;
63                  outliers(:,no+1) = point;
64                  no = no+1;
65              else
66                  totalError = totalError + error;
67                  inliers(:,ni+1) = point;
68                  ni = ni+1;
69              end
70           end % data iteration end
71      else
72          for p=3:N
73              point = data(:,randomP(p));
74
75              error = abs( point(1)*a_model + b_model
                    - point(2) )/ sqrt( a_model^2 + 1);
76
77              if( errorThreshold < error )
78                  totalError = totalError +
                        errorThreshold;
79                  outliers(:,no+1) = point;
80                  no = no+1;
```

```
81              else
82                  totalError = totalError + error;
83                  inliers(:,ni+1) = point;
84                  ni = ni+1;
85              end
86           end % data iteration end
87      end % data iteration end
88
89      %history
90      if(isfield(options, 'history') && options.
           history == true)
91          m = min(data(:));
92          M = max(data(:));
93          x = m-m/10:1/100:M+M/10;
94          figure
95          plot(x,x*a_model+b_model, inliers(1,:),
               inliers(2,:),'*');
96      end
97
98      % check model
99
100     if ( finalError > totalError && N/2*Pin <= ni )
101         bestModel = [a_model,b_model];
102         finalInliers = inliers;
103         finalOutliers = outliers;
104         finalError = totalError;
105         foundModel = 1;
106     end
107
108
109 end % main iteration end
110
111 %calculate the minimum square model
112 pol = polyfit(data(1,:),data(2,:),1);
113
114 %plot
115 if(foundModel == 0)
116     disp('I␣did␣not␣find␣a␣model');
117 else
118   figure
119   p = plot(data(1,:),data(2,:),'*');
120   set(p,'color','r');
121   hold on;
122   in = plot(finalInliers(1,:),finalInliers(2,:),'*')
        ;
123   set(in,'color','y');
124   hold on;
125   m = min(data(:));
126   M = max(data(:));
127   x = m-m/10:1/100:M+M/10;
128   model = plot(x,x*bestModel(1)+bestModel(2));
129   set(model,'color','b','LineWidth',1);
```

```
130   title('RANSAC␣results␣for␣2D␣line␣estimation');
131   legend('Outliers', 'Inliers');
132
133   %compare
134   if(isfield(options, 'compare') && options.compare
         == true)
135       model2 = plot(x,x*pol(1)+pol(2));
136       set(model2,'color','g');
137   end
138 end
139
140 return;
```

Bibliografia

[1] M.A. Fischler and R.C. Bolles,
Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,
Communications of the ACM, vol 24: pp 381-395, 06/1981

[2] Marco Zuliani,
RANSAC for Dummies
2008-2010

[3] P. J. Rousseeuw, A. M. Leroy,
Robust regression and outlier detection
New York: Wiley, 1987

[4] W. Chu, S. S. Keerthi, C. J. Ong,
A Unified Loss Function in Bayesian Framework for Support Vector Regression,
Proc. 18th International Conf. on Machine Learning, pp 51-58, 2001

[5] P. J. Huber,
Robust statistics,
New York: Wiley, 1981

[6] V. N. Vapnik
The nature of statistical learning theory
Springer, 1995

[7] K. G. Derpanis,
Overview of the RANSAC Algorithm,
York University, 2010

[8] W. Zhang, J. Kosecka,
Nonparametric estimation of multiple structures with outliers,
Workshop on Dynamic Vision, European Conference on Computer Vision 2006, vol 4358: pp 60-74, 2006

[9] P. Torr, D. Murray,
Outlier Detection and Motion Segmentation,
Sensor Fusion VI P.S. Schenker, vol 2059: pp 432-443, 1993

[10] Q. T. Luong, O. D. Faugeras,
The Fundamental MAtrix: Theory, Algorithms, and Stability Analysis,
International Journal of Computer Vision, vol 17(1): pp 43-76, 1994

[11] Y. Sugaya, K. Kanatani,
Outlier Removal for Motion Tracking by Subspace Separation,
IEICE Trans. Information and Systems, vol E86-D-6: pp 1095-1102, 2003

[12] D. Q. Huynh, R. Hartley, A. Heyden,
Outlier Correction in Image Sequences for the Affine Camera,
Proceedings ICCV, vol 1: pp 85-90, 2003

[13] A. J. Smola, B. Scholkopf,
A Tutorial on Support Vector Regression,
Statistics and Computing, 2001

[14] S. R. Gunn,
Support Vector Machines for Classification and Regression,
Technical Report, University of Southampton, 1997

[15] Variabili slack,
`http://en.wikipedia.org/wiki/Slack_variable`

[16] V. N. Vapnik,
Statistical Learning Theory,
New York: Wiley, 1998

[17] F. A. Fernandez,
Iris Recognition Based on SIFT Features,
Universidad Autonoma de Madrid, 2009

[18] *Difference of Gaussians*,
`http://en.wikipedia.org/wiki/Difference_of_Gaussians`

[19] Y. Ke, R. Sukthankar,
PCA-SIFT: A More Distinctive Representation for Local Image Descriptors,
CVPR, Washington DC, pp 66-75, 2004

[20] C. Liu, J. Yuen, A. Torralba,
SIFT Flow: Dense Correspondence across Scenes and its Applications,
European Conference on Computer Vision (ECCV), 2008

[21] D. G. Lowe,
Distinctive image features from scale-invariant keypoints,
International Journal of Computer Vision, vol 60(2): pp 91-110, 2004

[22] D. H. Ballard,
Generalizing the Hough transform to detect arbitrary shapes,
Pattern Recognition, vol 13(2): pp 111-122, 1981

[23] P. Scovanner, S. Ali, M. Shah,
A 3-Dimensional SIFT Descriptor and its Application to Action Recognition,
Proceedings of the 15th international conference on Multimedia, pp 357-360, 09/2007

[24] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, C. Bray,
Visual Categorization with Bags of Keypoints,
ECCV International Workshop on Statistical Learning in Computer Vision, 2004

[25] M. Brown, D. G. Lowe,
Recognising Panorama,
Ninth International Conference on Computer Vision (ICCV'03), pp 1218-1225, 2003

Rigraziamenti

In poche righe è difficile ricordare tutti coloro che in questi anni mi sono stati vicini.

Desidero ringraziare il prof. Angelo Cenedese per avermi aiutato nell'affrontare questa tesi, per la disponibilità sempre dimostrata e i preziosi suggerimenti che ha saputo darmi.

Grazie alla mia famiglia per tutto l'affetto che mi ha dato:
a mio padre perché sei riuscito a darmi sempre sostegno e fiducia, perché senza il tuo aiuto e il tuo amore non sarei riuscito ad arrivare dove sono ora e non riuscirei ad andare dove la vita mi condurrà;
a mia madre perché sei stata solide fondamenta per il mio avvenire, mi piace pensare a quanto saresti fiera di me se fossi ancora qui, grazie ai tuoi insegnamenti sono arrivato fino a questo traguardo... e sono pronto ad affrontare le prossime sfide;
a mia sorella perché sei la mia guida, il mio punto di riferimento, la mia più cara complice e amica e al tempo stesso la migliore insegnante di vita.
a Paolo per i continui spunti che mi dà, per le domande a cui mi fa pensare e per essere stato un punto di riferimento in questi anni;
a Marcello perché sei stato, e perché sei, solida roccia su cui poter contare, padrino, amico e modello da seguire;
a tutti i miei parenti, agli zii e ai miei cugini che mi sono stati accanto.

Grazie ai miei amici per quello che mi hanno fatto vivere e provare:
ad Alessia (Gemellina) perché sei sempre presente da otto anni a questa parte nella mia vita, vera amica, spalla su cui fare sempre affidamento, compagna

di mille avventure passate e sicuramente future;
a David (Pablo) perché sei stato uno dei veri motivi per cui questi tre anni a Padova saranno indelebili nei miei ricordi, grande amico e compagno, all'inizio di studio, ma ora soprattutto di vita, cazzate, avventure e futuro;
a Sabrina (Sabri) perché in tutti questi anni mi sei sempre stata accanto e con vera amicizia abbiamo superato gli ostacoli che si sono presentati dinnanzi, assieme, e lo saremo ancora per molto tempo;
a Tommaso (Tommy-V) per le avventure passate assieme, per le serate piene di iniziative ludiche-sportive-studiose passate con un vero amico, perché mi hai insegnato che nella vita è essenziale fare le cose con passione;
a Marco per il sostegno dimostratomi in ogni occasione anche con poche e semplici parole, per tutti gli insegnamenti che mi hai dato semplicemente mostrandoti per il generoso e disponibile amico che sei;
a Fabio perché basta grattare un po' la superficie per scoprire che sotto la tua scorza da eccentrico, freakettone, saccente, c'è un grande amico un po' eccentrico, spesso saccente e sì, molto freakttone, ma davvero buono e vero;
a Enea perché la tua spontanea briosità è così virale che contagia l'animo, si impadronisce della mente e fa scoppiare un sorriso sempre, ed è così bello ridere con un amico;
a Carlotta (Carloz) perché sei stata compagna di giornate di studio, di chiacchierate seduti sulla panchina fuori da geo, per avermi insegnato a comportarmi da bambino adulto che non fissa le persone quando si sta parlando di loro;
a Cristina (Cri) per la tua spontaneità, perché mi porti sempre un sorriso quando ci incontriamo, a prescindere da come stai tu e da come sto io, perché sei semplicemente sole, mi riscaldi e rallegri;
a Chiara perché sei stata la migliore vicina di casa che esista, per le chiacchiere fumando una cicca parlando di tutto quello che ci veniva in mente, momenti che mi mancheranno tanto;
a Marco (Shaz), Matteo (Pianca) e Alberto (Betta) perché mi avete sopportato come coinquilino in questi anni universitari, per aver compreso il mio non essere mai puntuale, il mio scassare la minchia per tante cavolate e la mia assenza in alcune, forse troppe, giornate, ma soprattutto per essere stati dei buonissimi compagni di mercoledì universitari e serate in compagnia;

a tutti gli amici da San Donà (e dintorni) e Padova che ho incontrato lungo la mia strada che, anche se non vedo spesso, mi hanno fatto passare tante piacevoli serate e saranno nitido ricordo di festa e allegria;

a Margherita (Meggy Mu) che sei arrivata nella mia vita e mi hai fatto svegliare con il sorriso tanti giorni in cui non sentivo nessun altro motivo per gioire, che mi hai fatto riscoprire la luna e le stelle, che mi sei stata accanto paziente, con amore.