POLITECNICO DI MILANO

Facoltà di Ingegneria dell'Informazione

Corso di Studi in Ingegneria delle Telecomunicazioni

Tesi di Laurea Magistrale

STIMA ROBUSTA DI PIANI DA NUVOLE DI PUNTI 3D PER LA NAVIGAZIONE ASSISTITI

Relatore:
Prof. Marco Marcon

Correlatore:
Paolo Pasteris

Tesi di Laurea Magistrale di:
Ciolino Antonino Fabio
Matr. 765931

ANNO ACCADEMICO 2011/2012

Sommario

La tesi è stata sviluppata nell'ambito del Progetto Europeo Astute, il quale mira a portare innovazione tecnologica nel campo del mercato automobilistico. Uno degli ambiti di innovazione, riguarda la realizzazione di un sistema di frenata automatico in situazioni d'emergenza. Indagini statistiche, infatti, hanno messo in luce che la presenza a bordo di tale sistema potrebbe ridurre il numero di incidenti del 27%.

Una possibile soluzione per realizzare tale sistema consiste nella caratterizzazione di una nuvola di punti 3D ottenuta mediante ricostruzione tridimensionale a partire da una sequenza di immagini. Grazie ai miglioramenti delle tecnologie di acquisizione di immagini digitali è possibile lavorare su ricostruzioni tridimensionali molte accurate.

La tesi propone un metodo per la stima robusta di piani da nuvole di punti 3D per la navigazione assistita. È stato scelto di stimare i piani in quanto la gran parte degli oggetti presenti in ambiente urbano (facciate dei palazzi, cartelli stradali, etc) presentano una superficie pressoché planare. Tale metodo permette di associare ciascun punto della nuvola di punti 3D ottenuta mediante ricostruzione tridimensionale ad un piano euclideo mediante il quale si identifica un oggetto presente nell'ambiente attorno l'autovettura.

Le prestazioni di tale metodo sono state valutate prendendo in esame sia dati sintetici che dati sperimentali.

Abstract

The present thesis has been developed in the context of the Astute European Project, which aims at bringing technological innovation in the automotive Market. One of the proposed innovations is the realization of an automatic braking system for emergency situation. Statistical surveys, in fact, have shown that the presence on board of this system could reduce the accidents number by 27%.

A possible solution to realize it consists in the characterization of 3D points cloud obtained by visual reconstruction of images sequence. Through technological development of digital image capture, it's possible to have very accurate visual reconstructions.

The thesis proposes a method for strong estimation of plans from 3D point clouds for assisted navigation. I have chosen planar surface because many urban environment objects (front of buildings, road signs, etc.) have this kind of surface. This method enables to link each point of the 3D point cloud to a Euclidean plane that identifies an object present in the environment around the vehicle.

The method performance was valueted by synthetic and experimental data.

Sommario

Idice Figure

INTRODUZIONE

Lo studio ed il lavoro svolto per realizzare tale elaborato riguardano l'implementazione di un algoritmo il cui fine è quello di ottenere una stima robusta di piani nello spazio partendo da nuvole di punti 3D. Il lavoro da me svolto per raggiungere questo obbiettivo, è stato realizzato in collaborazione con l'azienda "ST Microelectronics", ed in particolare con la divisione interna di quest'ultima Advanced System Technology (AST). Tale elaborato, infatti, è inserito all'interno del progetto europeo *Astute* (astute-project), di cui il Politecnico di Milano e ST Microelectronics sono partners. Scopo di tale progetto è quello di sviluppare una piattaforma per sistemi embedded (W.Fornaciari & Brandolese) al fine di fornire tecnologie avanzate di interazione uomo-macchina di supporto in contesti di sovraccarico di informazioni. Gli scenari applicativi che tale progetto impatta sono svariati, ma l'algoritmo da me realizzato e descritto in tale elaborato è da inserirsi in ambito *auto-motive*. A tale scopo è stato realizzato un algoritmo da inserire in una *pipeline* complessiva, già in parte realizzata da ST Microelectronics. Essa dovrà essere in grado, partendo semplicemente da un insieme di immagini scattate da una fotocamera posizionata sull'autovettura, di individuare la presenza di eventuali oggetti presenti intorno a quest'ultima, valutarne la distanza da essa e quindi eseguire eventuali azioni al fine di evitare una collisione. Un'indagine statistica (Autoblog) effettuata da Euro NCAP[1], infatti, ha messo in luce che la presenza di un sistema di frenata autonoma che agisca in condizioni d'emergenza (*AEB – Autonomous Emergency Breaking*) può ridurre il numero di incidenti del 27%. La scelta di stimare il miglior piano

[1] Euro NCAP fu fondato nel 1997 ed e ormai sostenuto da sette Governi europei, dalla Commissione Europea e dalle associazioni di consumatori e di automobilisti di tutti i paesi dell'Unione Europea. Esso è diventato rapidamente un catalizzatore di iniziative volte a promuovere l'aspetto della sicurezza nella progettazione dei veicoli.

associabile ad una nuvola di punti 3D nasce dalla costatazione che molti elementi presenti in un ambiente urbano (facciate dei palazzi, segnaletica stradale, etc) possono essere approssimati con un piano.

Fatte queste premesse, il punto di partenza del lavoro da me svolto, ha riguardato lo studio e l'apprendimento della *pipeline* realizzata da ST Microelectronics. Tale *pipeline*, permette, tramite l'applicazione della tecnica *Structure from Motion(SfM)*, di ricostruire da una sequenza di immagini scattate da una camera in movimento, sia la struttura tridimensionale dell'ambiente circostante la camera che la posizione della camera stessa. Tale ricostruzione presenta un errore di riproiezione ad essa associato non nullo. Poiché la struttura 3D restituita da tale tecnica costituisce l'argomento in ingresso dell'algoritmo di stima dei piani realizzato, è stato necessario implementare un metodo di ottimizzazione per ridurre l'errore. Una volta appresa la tecnica *SfM*, quindi, mi sono dedicato all'implementazione ed alla successiva validazione rispetto allo stato dell'arte, di un algoritmo, , che permettesse di risolvere il problema del *Bundle Adjustment(BA)*[2], che è descritto nel primo capitolo. Tale implementazione mi ha permesso di minimizzare l'errore di riproiezione restituito dalla tecnica *SfM*, e di ottenere sia la struttura 3D *ottima* che i parametri della camera *ottimi*. Una volta minimizzato l'errore associato ai dati in ingresso, il passo successivo ha riguardato l'implementazione vera e propria dell'algoritmo oggetto di tale elaborato. Tale algoritmo è realizzato mediante l'applicazione di tre tecniche: *Trasformata di Radon*, *Principal Component Analysis(PCA)*, *RANdom SAmple Consensus (RANSAC)*. Nei capitoli successivi verrà fornita una descrizione di tali tecniche e di come esse sono state applicate al caso specifico per raggiungere gli obbiettivi prefissati. Di seguito, invece, riporto una breve descrizione dell'applicazione dell'algoritmo alla singola immagine. Quest'ultimo, infatti, è applicato iterativamente a tutte le immagini nel medesimo modo.

[2] Per Bundle Adjustment si fa riferimento al problema di raffinazione di una ricostruzione 3D da una sequenza di immagine.

Il primo passo, consiste nel valutare la *Trasformata di Radon* dei dati a nostra disposizione, ovvero la nuvola di punti 3D associata all'immagine presa in esame. Mediante tale tecnica è possibile ottenere una suddivisione di quest'ultima in più nuvole di punti 3D. Ciascuna di essa è costituita da tutti i punti 3D associabili ad un oggetto presente nell'ambiente circostante la camera (palazzo, aiuola, cartelli stradali, etc.).

Tale suddivisione, per motivi che verranno descritti nel capitolo 2, potrebbe dar luogo a nuvole di punti 3D che presentino dei punti, denominati *outliers*, non associabili al medesimo oggetto descritto dai restanti punti. Il secondo passo consiste nell'applicare a ciascuna delle singole nuvole di punti 3D la tecnica *Principal Component Analysis*. Tale tecnica, la cui descrizione ed applicazione verrà fornita al capitolo 3, è utilizzata per ottenere il miglior piano[3] associabile a ciascuna delle nuvole di punti 3D ottenute al passo precedente. Essa, realizza una trasformazione lineare[4] su un insieme di variabili (i punti 3D) al fine di ottenere un nuovo insieme di variabili, tra loro ortogonali, chiamate *Principal Components*. Dalla conoscenza di tali componenti è possibile ricavare il piano di nostro interesse, mentre da un'analisi degli autovalori ed autovettori ad esse associate è possibile verificare se la stima del piano ottenuto è robusta. Qualora così fosse, l'esecuzione dell'algoritmo termina e si passa all'applicazione dello stesso per l'immagine successiva. Viceversa viene eseguito il terzo ed ultimo passo dell'algoritmo. La principale causa di un'eventuale stima non corretta è dovuta alla presenza di *outliers* nelle nuvole di punti 3D ottenute a seguito dell'applicazione della *Trasformata di Radon*. Al fine d'individuare tali *outliers*, nel terzo passo viene utilizzato l'algoritmo *RANSAC*. Quest'ultimo, la cui descrizione ed applicazione al caso specifico verrà fornita nel capitolo 3, permette, mediante l'applicazione di un metodo iterativo, di stimare i parametri di un

[3] Per miglior piano, si intende quel piano che minimizza l'errore quadrato

[4] Una trasformazione lineare è un omomorfismo di spazi vettoriali, in quanto conserva la forma di ogni istanza dell'operazione che caratterizza gli spazi vettoriali. E', quindi, funzione tra due spazi vettoriali che preserva la forma delle operazioni di somma di vettori e di moltiplicazione per scalare.

modello matematico (i coefficienti del piano) a partire da un insieme di dati contenente *outliers*. Al termine di tale passo, l'esecuzione dell'algoritmo termina e si passa all'applicazione dello stesso per l'immagine successiva.

Di seguito il diagramma a blocchi dell'algoritmo appena descritto:

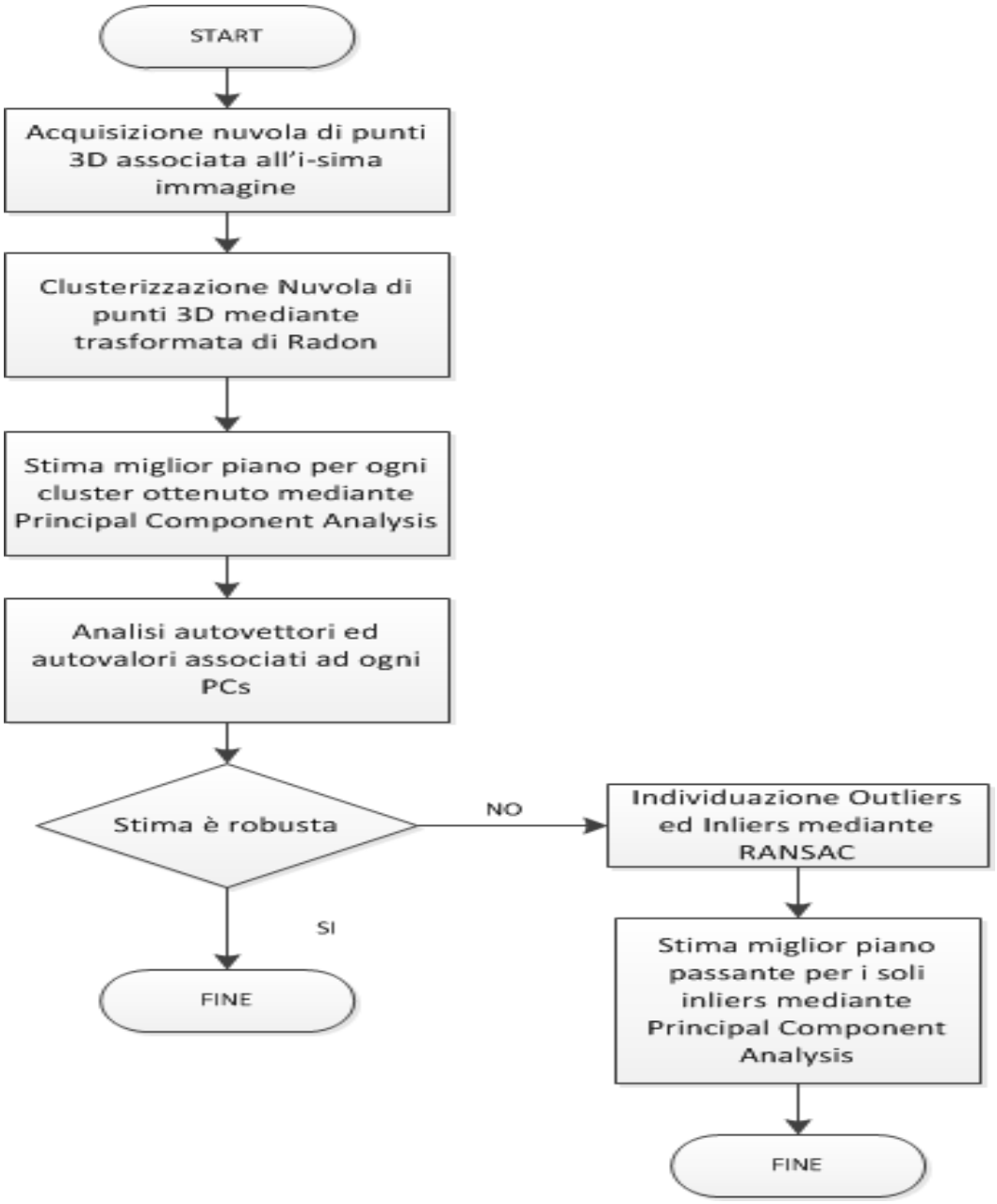


Figura 1: Diagramma di flusso algoritmo

Conclusa la descrizione delle tecniche utilizzate per realizzare l'algoritmo e di come esse sono state applicate, vengono riportati, rispettivamente nei capitoli 4 e 5, i risultati ottenuti e le conclusioni a cui sono giunto sulla base di essi. In particolare i risultati riportano l'andamento di alcuni parametri necessari al fine di validare l'effettivo funzionamento dell'algoritmo.
Da quanto detto, gli obbiettivi di tale tesi sono :

- Minimizzare l' errore di riproiezione dei dati di partenza;
- Realizzare un algoritmo per la stima robusta di piani da nuvole di punti 3D;
- Individuare i singoli oggetti presenti nell'ambiente circostante la camera.

L'ambiente di sviluppo scelto per realizzare quanto finora descritto è MATLAB (*MATrix LABoratory*). MATLAB è un linguaggio di alto livello ed un ambiente interattivo per il calcolo, la visualizzazione e la programmazione numerica. Esso consente di analizzare dati, sviluppare algoritmi, creare modelli e applicazioni (Matlab). MATLAB è usato sia in ambito industriale che accademico, soprattutto per via dei numerosi *tools* a supporto di svariati campi di studio che esso mette a disposizione. Tra i vari *Toolbox*, lo *Statistics Toolbox* contiene svariate funzioni e tecniche, come la *Principal Component Analysis*, già implementate che mi hanno agevolato nel raggiungimento degli obbiettivi prefissati.

CAPITOLO 1
BUNDLE ADJUSTMENT

Con il termine *Bundle Adjustment(BA)* si identifica il problema di raffinazione di una ricostruzione visuale[5] al fine di ottenere sia la struttura 3D *ottima* che i parametri della camera[6] *ottimi* (Bill Triggs Philip McLauchlan). *Ottimo* sta ad indicare che i parametri stimati e la struttura 3D sono ottenuti minimizzando la funzione di costo che quantifica l'errore associato al modello di partenza. Da tale definizione, è evidente come una sua implementazione sia estremamente utile al termine di un algoritmo di ricostruzione al fine di migliorarne l'accuratezza dei parametri stimati.

1.1 Descrizione del Problema

Consideriamo una situazione in cui la scena che si vuole ricostruire è rigida, ovvero non vi sono oggetti in movimento (macchine, pedoni). L'obbiettivo è, dato un insieme di coordinate immagine x_j^i, ottenute ad esempio mediante la tecnica *SfM*, trovare le matrici di proiezioni delle camere P^i e l'insieme dei punti 3D tale per cui $P^i X_j = x_j^i$. Tale ricostruzione è nota come *ricostruzione proiettiva*. Se, però, l'insieme di coordinate x_j^i sono rumorose, l'equazione $P^i X_j = x_j^i$ non è soddisfatta esattamente. Riproiettando, infatti, i punti stimati $\widehat{X_j}$ mediante le matrici stimate $\widehat{P^i}$ si avrà che tali punti si troveranno ad un certa distanza *d(x,y)*[7]

[5] ricostruzione 3D ottenuta da una sequenza d'immagini
[6] posizione della camera o parametri di calibrazione intrinseci
[7] d(x,y) è la distanza geometrica tra i punti omogenei x ed y

dai punti immagine misurati. L'errore che ne segue è noto come errore di riproiezione. La soluzione cercata, supponendo che il rumore sia Gaussiano, è la soluzione a Massima Verosimiglianza (Spagnolini, 2010) ovvero quella soluzione che stimi i punti $\widehat{X_j}$ e le matrici $\widehat{P^i}$ che minimizzino la distanza $d(\widehat{P^i}\widehat{X_j}, x_j^i)^2$:

$$\min_{\widehat{P^i}\widehat{X_j}} \sum_{ij} d(\widehat{P^i}\widehat{X_j}, x_j^i)^2$$

Tale problema di stima, che comporta la minimizzazione dell'errore di riproiezione e quindi della distanza prima introdotta, è il *BA*. I metodi matematici con cui è possibile ottenere la minimizzazione desiderata sono svariati. I più noti sono: il metodo di Newton; l'algoritmo del gradiente e l'algoritmo di Leveneberg-Marquardt. Essi sono tutti metodi che permettono di realizzare una stima ai minimi quadrati non lineare (Hartley & Zisserman, Multiple View Geometry in computer vision, 2003). Per motivi che chiarirò nel prosieguo, il metodo scelto per implementare la risoluzione del *BA* è il metodo di Levenberg-Marquardt(L-M), il quale può essere visto come un ibrido tra l'algoritmo di Newton e l'algoritmo del gradiente (Hartley & Zisserman, Multiple View Geometry in computer vision, 2003). Tale metodo permette di trovare lo zero di una funzione non lineare mediante un processo iterativo. Esso, infatti, data una relazione funzionale del tipo:

$$\mathbf{X}=f(\mathbf{P})$$

ove **X** è il vettore misurato, **P** il vettore dei parametri rispettivamente negli spazi Euclidei R^N e R^M ed f una funzione non lineare, ha come obbiettivo la stima del vettore $\widehat{\boldsymbol{P_{ott}}}$ ovvero di quel vettore tale per cui la relazione $\mathbf{X}=f(\widehat{\boldsymbol{P_{ott}}})$ - ε abbia ε minimo. Poiché f non è lineare, per trovare il vettore $\widehat{\boldsymbol{P}}$ che minimizza ε, è necessario partire da una stima iniziale di **P**, che indico come $\mathbf{P_0}$, e raffinare quest'ultima fino a che f non la si può assumere lineare localmente. A tal fine definisco $\boldsymbol{\varepsilon}_0 = f(\mathbf{P_0}) - \mathbf{X}$ l'errore commesso per $\mathbf{P} = \mathbf{P_0}$ e $J = \frac{\partial f}{\partial P}$ la matrice

Jacobiana[8] assumo, inoltre, che la funzione *f* la si possa approssimare, per **P** ancora pari a $\mathbf{P_0}$, come $f(\mathbf{P_0} + \Delta) = f(\mathbf{P_0}) + \mathbf{J}\Delta$. Il passo successivo dell'algoritmo, allora, è trovare il vettore $\mathbf{P_1} = \mathbf{P_0} + \Delta$ per il quale $f(\mathbf{P_1}) - \mathbf{X} = f(\mathbf{P_0}) + \mathbf{J}\Delta - \mathbf{X} = \boldsymbol{\varepsilon_0} + \mathbf{J}\Delta$ sia minima. Per ottenere ciò, è pertanto necessario valutare il vettore $\Delta = [\delta_1,\delta_2,...,\delta_M]$, ove δ_i è l'incremento di ogni singolo parametro, che permetta di minimizzare $\| \boldsymbol{\varepsilon_0} + \mathbf{J}\Delta \|$, il ché equivale a risolvere un problema ai minimi quadrati lineare.

Nell'algoritmo di L-M, a differenza dell'algoritmo di Newton, tale vettore è ottenuto mediante la risoluzione del seguente sistema noto come *equazioni normali aumentate:*

$$(\mathbf{J}^T + \lambda\mathbf{I})\Delta = -\mathbf{J}^T\boldsymbol{\varepsilon}$$

Il termine *aumentate* è introdotto proprio per distinguerle dalle *equazioni normali* dell'algoritmo di Newton[9]. In particolare rispetto a quest'ultime, l'algoritmo di L-M utilizza un parametro ulteriore, λ, il quale varia ad ogni iterazione a seconda che il valore di Δ ottenuto all'i-sima iterazione comporti una diminuzione o un aumento dell'errore che si vuole minimizzare. Tale parametro è noto come *fattore di damping*. Partendo, infatti, da un valore iniziale solitamente pari a 10^{-3} volte la media degli elementi della matrice $\mathbf{N} = \mathbf{J}^T\mathbf{J}$, se il valore di Δ ottenuto all'i-sima iterazione ha dato luogo ad una diminuzione dell'errore, λ viene diviso per un certo fattore (solitamente 10) e si passa all'iterazione successivo; viceversa λ viene moltiplicato per il medesimo fattore e le equazioni normali aumentate vengono risolte nuovamente fintantoché non viene ottenuto un valore di Δ che dia luogo ad una diminuzione dell'errore. Il procedimento appena descritto per ricavare il vettore $\mathbf{P_1}$ viene reiterato fintantoché non viene ottenuto il vettore $\widehat{\boldsymbol{P}}$ tale per cui l'algoritmo converga.

[8] La matrice Jacobiana è quella matrice i cui elementi sono le derivate parziali prime di una funzione che ha dominio e codominio in uno spazio euclideo

[9] $\mathbf{J}^T\mathbf{J}\Delta = -\mathbf{J}^T\boldsymbol{\varepsilon_0}$

La stima di partenza $\mathbf{P}_0$ è pertanto ad ogni iterazione aggiornata fino a giungere a convergenza secondo la relazione:

$$\boldsymbol{P}_{i+1} = \boldsymbol{P}_i + \boldsymbol{\Delta}_i$$

Da quanto sopra detto, è evidente che la peculiarità di tale metodo iterativo rispetto agli altri, consiste nell'utilizzo del parametro λ. Al fine di comprendere il vantaggio di ciò, riporto di seguito cosa accade a seconda del valore di λ. Nel caso in cui λ assume valori molto piccoli, allora il comportamento dell'algoritmo di L-M è simile a quello di Newton in quanto esso converge molto velocemente se la stima iniziale è nelle vicinanze della soluzione cercata. Viceversa, quando λ è grande allora l'equazione normale aumentata la si può approssimare come $\lambda\boldsymbol{\Delta} = -\mathbf{J}^{\mathrm{T}}\varepsilon$, pertanto il parametro $\boldsymbol{\Delta}$ è incrementato in maniera simile all'algoritmo del gradiente il quale garantisce un decremento della funzione di costo e quindi dell'errore anche nel caso in cui sia difficile una sia minimizzazione. L'algoritmo di L-M, quindi, mediante il parametro λ, permette di sfruttare i vantaggi degli algoritmi di Gauss e del gradiente seguendo l'approccio dell'uno o dell'altro a seconda della funzione che si deve minimizzare e della stima di partenza.

L'applicazione dell'algoritmo di L-M, così come descritto finora, va bene solamente nel caso in cui il numero di parametri che si vuole stimare sono ridotti. Da un punto di vista computazionale, infatti, se il numero di parametri è elevato diventa estremamente oneroso raggiungere la convergenza. Ciò è dovuto soprattutto alla risoluzione delle equazioni normali aumentate la cui complessità è N^3, ove N è il numero dei parametri che si vuole stimare. Applicare quindi tale metodo nella sua versione di base al fine di raffinare una ricostruzione proiettiva non è computazionalmente conveniente, visto che i parametri da stimare sono pari a $12*m + 3*n$ ove m è il numero delle camere ed n il numero di punti 3D. Per tale motivo, il BA di un problema di ricostruzione è risolto mediante una variante dell'algoritmo di L-M nota come "sparsa". Di seguito descriverò tale variante, facendo riferimento proprio al caso in cui essa venga applicata per raffinare la

stima dei punti 3D e dei parametri della camera. Il primo passo, consiste proprio nel partizionare il vettore **P** appartenente a R^M, con cui finora abbiamo indicato i parametri da stimare, in due vettori **a** e **b** contenenti rispettivamente i parametri che descrivono la camera e i parametri che descrivono i punti. Il vettore **P** lo si può pertanto riscrivere come $\mathbf{P} = (\mathbf{a}^T, \mathbf{b}^T)^T$. Conseguenza di ciò, è che il vettore degli incrementi è costituito anch'esso da due vettori $\Delta = (\delta_a, \delta_b)$ e la matrice Jacobiana **J** presenta una struttura a blocchi, ovvero $\mathbf{J} = [\mathbf{A} \mid \mathbf{B}]$ ove A e B sono entrambe definite come sottomatrici Jacobiane e sono ottenute come:

$$\mathbf{A} = \frac{\partial \boldsymbol{X}}{\partial \boldsymbol{a}} \qquad\qquad \mathbf{B} = \frac{\partial \boldsymbol{X}}{\partial \boldsymbol{b}}$$

Le equazioni normali aumentate che bisogna risolvere sono quindi della forma:

$$\left[\begin{array}{c|c} A^T\Sigma_X^{-1}A & A^T\Sigma_X^{-1}B \\ \hline B^T\Sigma_X^{-1}A & B^T\Sigma_X^{-1}B \end{array}\right] \left(\begin{array}{c} \delta_a \\ \hline \delta_b \end{array}\right) = \left(\begin{array}{c} A^T\Sigma_X^{-1}\epsilon \\ \hline B^T\Sigma_X^{-1}\epsilon \end{array}\right)$$

Moltiplicando gli elementi lungo la diagonale principale per $(1 + \lambda)$ e ponendo(inserire corrispondenze), tali equazioni le possiamo riscrivere come:

$$\left[\begin{array}{cc} U^* & W \\ W^T & V^* \end{array}\right] \left(\begin{array}{c} \delta_a \\ \delta_b \end{array}\right) = \left(\begin{array}{c} \epsilon_A \\ \epsilon_B \end{array}\right).$$

Risolvendo tali equazioni si ottengono i valori di δ_a e δ_b e quindi valutato il vettore **P** come $\mathbf{P} = ((\mathbf{a} + \delta_a) + (\mathbf{b} + \delta_b))$. Così come nella versione di base dell'algoritmo, anche in tale variante, il parametro λ viene dimezzato e si passa all'iterazione successiva se il nuovo vettore **P** da luogo ad una diminuzione della funzione d'errore. Viceversa λ viene moltiplicato per il medesimo fattore e le equazioni normali aumentate vengono risolte nuovamente fintantoché non vengono ottenuti degli incrementi δ_a e δ_b tali per cui il vettore **P** dia luogo ad una diminuzione dell'errore. Dalla descrizione di tale variante appena fatta, il vantaggio computazionale di cui parlavo precedentemente non è molto evidente. Il vantaggio è che con questa variante, a differenza della versione base, è possibile

sfruttare l'assunzione che il vettore dei dati misurati **X** é sparso, ovvero ogni stima $\widehat{\boldsymbol{X}_\mathbf{i}}$ di ogni sua componente è dipendente solamente dal vettore dei parametri **a** e da b_i , ma non dalle restante componenti b_j (con j≠i) del vettore **b**. La conseguenza di ciò è che la matrice Jacobiana J, presenterà anch'essa una struttura sparsa, in quanto $\frac{\partial \boldsymbol{X_i}}{\partial \boldsymbol{b_j}}$ è sempre nulla eccetto nel caso in cui j = i. Le equazioni normali aumentate da risolvere sono, quindi:

$$J\boldsymbol{\delta} = \left[\begin{array}{c|c|cc} A_1 & B_1 & & \\ A_2 & & B_2 & \\ \vdots & & & \ddots \\ A_n & & & B_n \end{array}\right] \left(\begin{array}{c} \boldsymbol{\delta}_\mathbf{a} \\ \hline \boldsymbol{\delta}_{b_1} \\ \vdots \\ \boldsymbol{\delta}_{b_n} \end{array}\right) = \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{pmatrix}.$$

In tale forma si ha che ogni iterazione dell'algoritmo richiede una complessità computazionale pari lineare, e non più cubica, all'aumentare del numero di parametri *n*.

1.2 Procedura per Validare l'Implementazione del Bundle Adjustmente Rispetto Stato dell'Arte

Come anticipato nel capitolo introduttivo, in merito al *BA*, il lavoro da me svolto ha riguardato l'implementazione del metodo di Levenberg-Marquadt sparso e poi la successiva validazione rispetto allo stato dell'arte, del codice realizzato. Nel seguito indicherò con la sigla BA_ST tale implementazione. Quest'ultima, come già detto, è stata realizzata al fine di ottimizzare la stima dei parametri di partenza, che ho indicato con il vettore $\mathbf{P_0}$, ottenuta a seguito dell'esecuzione della tecnica *SfM*. Il codice di riferimento scelto, che nel seguito indicherò come SBA, è stato realizzato da Manolis Lourakis presso *Istitute Computer Science of the*

Foundation for Research and Technology (Grecia) (Lourakis & Argyros, March 2009). Anch'esso, ovviamente, implementa il BA mediante la versione sparsa dell'algoritmo di L-M e permette di raffinare sia i parametri della camera che la struttura 3D. La principale differenza tra i due codici, consiste nel diverso linguaggio con cui essi sono implementati. Lo SBA, infatti, è realizzato mediante il linguaggio di programmazione C/C++, mentre BA_ST è stato realizzato mediante MATLAB. Ai fini della validazione, però, tale differenza non è stata rilevante, in quanto non è stato preso in esame il tempo di elaborazione come parametro di confronto, bensì il numero di iterazioni effettuate da ciascuna delle due implementazioni affinché l'algoritmo giunga a convergenza.

Per poter confrontare il comportamento dei due codici, entrambi sono stati applicati come ultimo passo della tecnica *SfM* al fine di raffinare la struttura 3D ed i parametri delle camere da essa restituiti. In particolare la tecnica *SfM* è stata applicata per ricostruire una scacchiera partendo da tre immagini scattate dalla stessa camera posta in tre posizioni differenti. Riporto di seguito le tre immagini utilizzate:

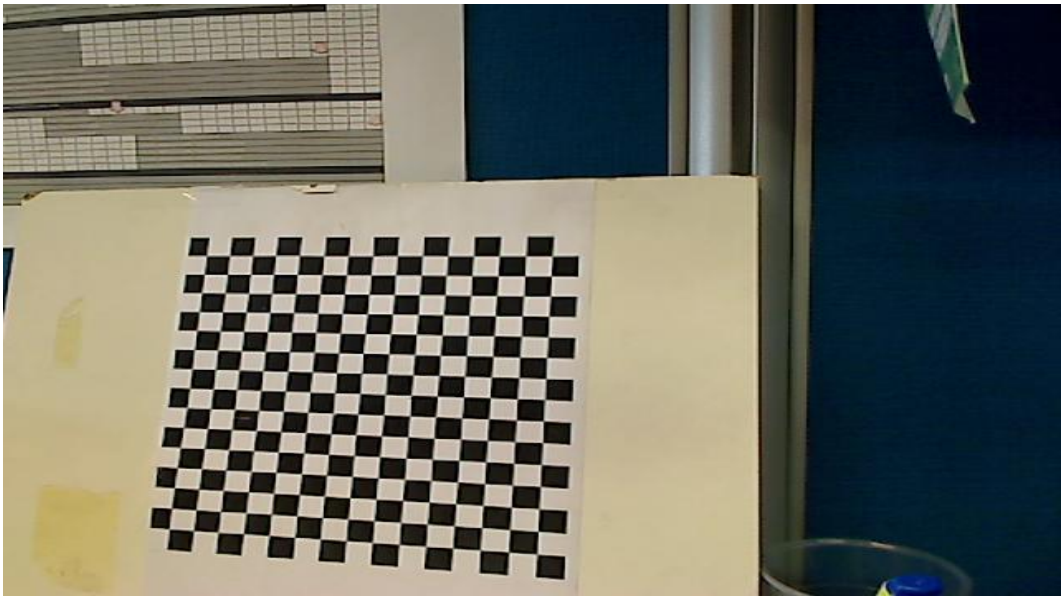


Figura 2: Scacchiera 1° posizione

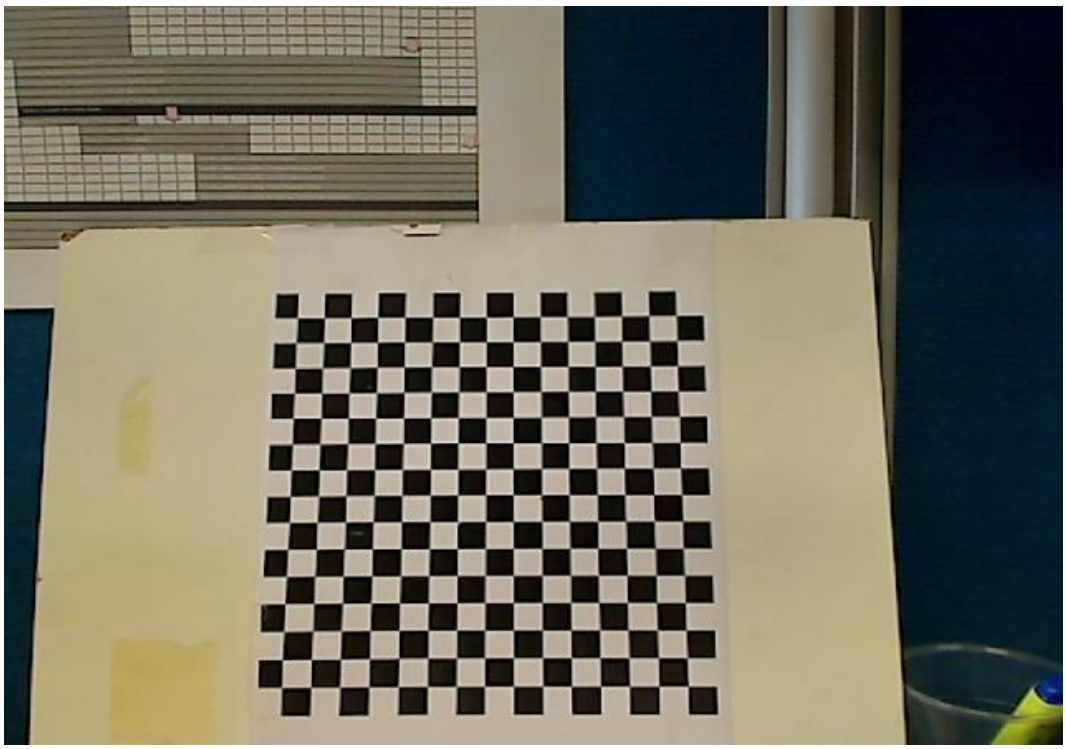


Figura 3: Scacchiera 2° posizione

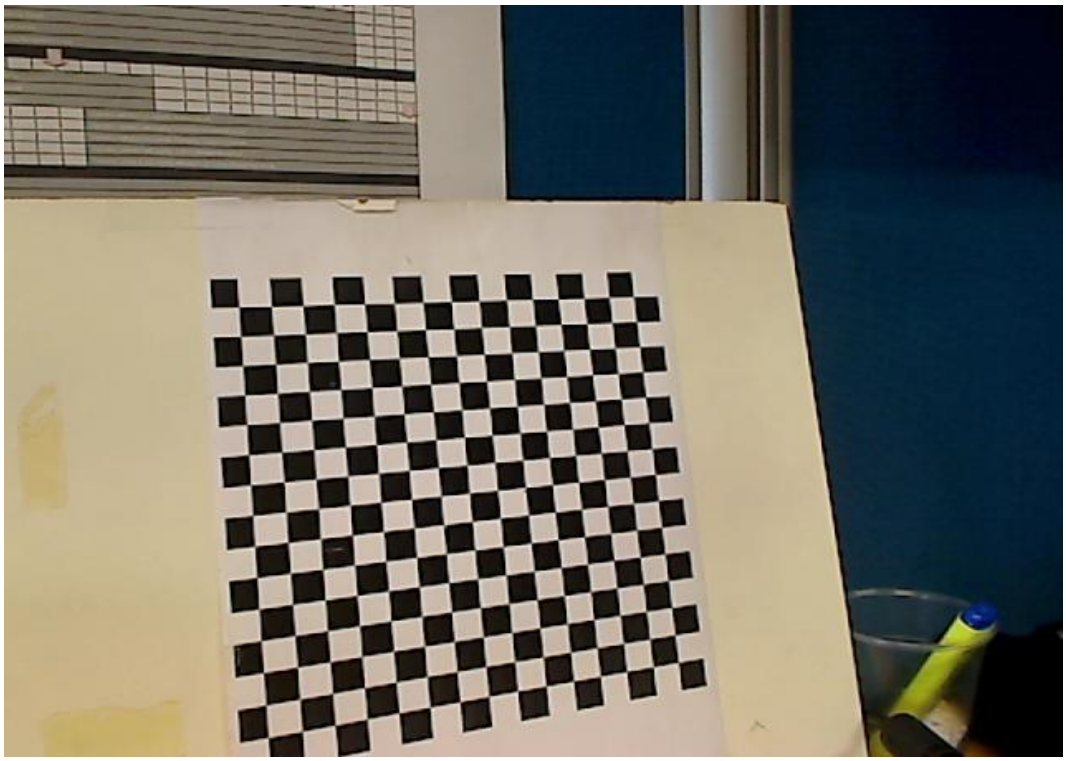


Figura 4: Scacchiera 3° posizione

Il primo problema è stato adattare i dati restituiti dalla tecnica *SfM* al formato accettato dallo SBA al fine di avere in ingresso ad entrambi gli algoritmi i medesimi dati di partenza. Se per il BA_ST non vi sono stati problemi in quanto è stato implementato tenendo conto del formato restituito dalla *SfM* realizzata da STMicroelectronics, è invece stato necessario per lo SBA realizzare uno script apposito che permettesse di realizzare una conversione della matrice di roto-traslazione, restituita dalla tecnica *SfM* per definire la posa delle camere, in un quaternione (JeanPierre, 17th – 20th August, 1993). Oltre ai parametri che descrivono la posa delle camere, entrambe le implementazioni richiedono:

- la stima di partenza delle coordinate dei punti 3D, anch'esse restituite dalla tecnica *SfM*, e delle corrispondenti coordinate 2D dei punti immagini proiettati(se il punto 3D è visto da tutte e tre le immagini allora tale punto avrà tre punti 2D corrispondenti);
- i parametri di calibrazione intrinseci della camera.

Per tali parametri è bastato semplicemente riadattare la struttura come indicato nella documentazione fornita dall'autore dello SBA. Ovviamente le conversioni e le operazioni effettuate sui dati in ingresso, sono state effettuate anche sui dati restituiti dalle due implementazioni al termine della loro esecuzione. Una volta realizzati tali strumenti e quindi ottenuti formati equivalenti per i dati in ingresso ed in uscita delle due implementazioni, ho effettuato l'attività di validazione. Essa è stata realizzata prendendo in esame dodici esecuzioni delle due implementazioni. Ciascuna di essa differisce dalle altre per le differenti stime di partenza dei punti 3D e dei parametri di posa di ciascuna delle tre camere. Il confronto, allora, tra lo SBA ed il BA_ST è stato effettuato valutando i seguenti parametri che le due implementazioni restituiscono:

- Errore di Riproiezione;
- Numero di iterazioni;
- Angoli di Eulero e quindi rotazione su ciascun asse del sistema di riferimento(x,y,z) di ogni camera;

- Traslazione su ciascun asse del sistema di riferimento(x,y,z) di ogni camera;

Da una prima valutazione di tali parametri, è stato evidenziato che entrambe le implementazioni convergono al medesimo valore ma il numero d'iterazioni che impiegano sono differenti. Al fine di colmare tale differenza ho effettuato un opportuna operazione di *tuning* sul fattore di *damping* λ= Tmax[diag(J)] a seguito della quale si è scelto di porre $\lambda = 10^{-6}$. Una volta verificato che errore di riproiezione e numero d'iterazioni medie restituiti dalle due implementazioni coincide, è stato effettuato un confronto anche sui parametri di posa, rotazione e traslazione, di ciascuna delle tre camere e sulle coordinate dei punti 3D restituite dalle due implementazioni. Come mi aspettavo, anche per tali parametri ho ottenuto concordanza tra i valori restituiti dalle due implementazioni. Rimando al capitolo "Risultati" per i dettagli dei risultati ottenuti.

CAPITOLO 2
TRASFORMATA DI RADON

La trasformata di Radon di una generica funzione f è uno strumento solitamente utilizzato per risolvere problemi di ricostruzione tomografica. Essa, infatti, permette di ricavare la struttura interna di un oggetto partendo dall'osservazione delle sue proiezioni. Per i nostri scopi, però, tale strumento è stato utilizzato con finalità differenti. In particolare, con l'applicazione della Trasformata di Radon, si è voluto realizzare un metodo per la caratterizzazione di una nuvola di punti 3D. In tale capitolo pertanto, fornirò prima una descrizione teorica di tale Trasformata, per poi passare a dettagliare in che modo essa è stata applicata.

2.1 Descrizione della Tecnica

Determinare una funzione *f(x,y)* dalla conoscenza del suo integrale di linea (caso 2D), o una funzione *f(x,y,z)* dall'integrale sui piani(caso 3D) costituisce un problema da risolvere in un insieme di scenari sempre maggiore (medical imaging, astronomy, automotive). Ricavare tali funzioni, infatti, equivale ad ottenere la struttura interna di un oggetto semplicemente dall'osservazione delle sue proiezioni. La Trasformata di Radon e la sua forma inversa, permettono di risolvere tale problema. Al fine di comprendere la trasformata, è necessario introdurre due spazi: spazio delle *features* e spazio di Radon. Lo spazio delle *features* è uno spazio Euclideo in 2,3 o *n* dimensioni, in cui la distribuzione spaziale *f* di alcune proprietà fisiche sono definite. Lo spazio di Radon , invece, altro non è che lo spazio comprendente le corrispondenti trasformate. Fatte queste

premesse, dunque, la trasformata di Radon di una funzione 2D, *f(x,y)*, è definita come (Deans):

$$\check{f}(p,\phi) = \int_{-\infty}^{\infty} f(\boldsymbol{r})\, dl^{10} \qquad (2.1)$$

ovvero come l'integrale di linea di *f* per tutte le linee *l* tracciate dalle coordinate *p* e ϕ.

In particolare l'equazione della generica retta *l* è:

$$p = x\cos\phi + y\sin\phi$$

graficamente infatti:

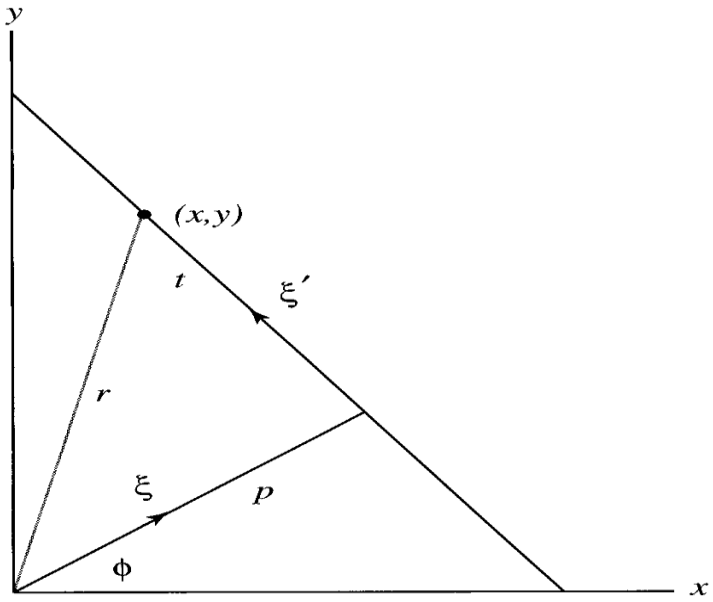


Figura 5: Coordinate nello spazio delle features della trasformata di Radon

Facendo uso della funzione delta di dirac[11], una definizione del tutto equivalente a quella su data è:

$$\check{f}(p,\phi) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\delta(p - x\cos\phi - y\sin\phi)dxdy \qquad (2.2)$$

[10] **r** è un vettore posizione, ovvero **r** = (x,y)

[11] La funzione delta di dirac è definita dalla notazione $\int_{-\infty}^{\infty}\delta(x)\varphi(x)dx = \varphi(0)$. Essa pertanto è una funzione che vale 1 quando x=0 ed è invece nulla quando x≠0.

Da tale definizione, è evidente come la funzione $\check{f}$ non è definita su un sistema di coordinate polari circolari[12]. Lo spazio appropriato, infatti, è la superficie di un semi-cilindro di lunghezza p, angolo di rotazione ϕ rispetto ad una posizione di riferimento arbitraria:

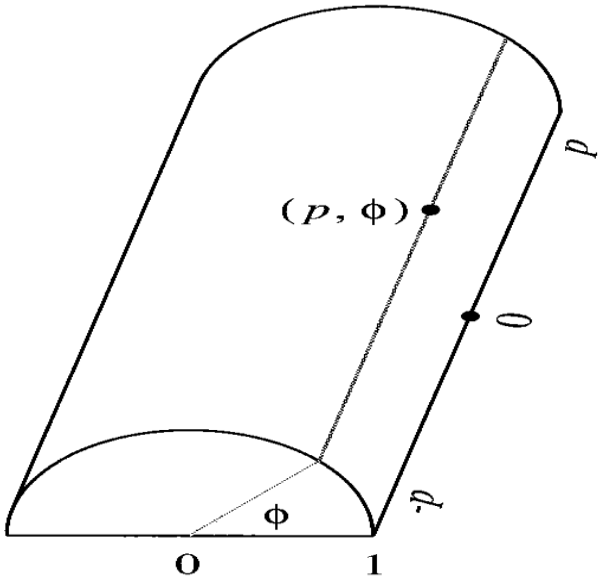


Figura 6: Coordinate nello spazio di Radon sulla superficie di un cilindro

La conseguenza di ciò, è che se $\check{f}$ è nota per $-\infty < p < \infty$, allora al fine di ricostruire l'intera immagine, basterà valutare solamente gli angoli ϕ compresi tra 0 ed π. Poiché, infatti, $\delta(x) = \delta(-x)$, allora le coordinate $(-p, \phi)$ e $(p, \phi + \pi)$ denotano lo stesso punto nello spazio di Radon.

Supponendo di srotolare tale semi-cilindro, quello che si ottiene è un piano i cui punti avranno coordinate (p, ϕ) su una griglia rettangolare. La trasformata di Radon allora altro non è che una superficie in tre dimensioni perpendicolare a tale piano. Un ulteriore definizione ancora equivalente alla (2.1) è:

$$\check{f}(p, \phi) = \int_{\xi r = p} f(x, y)\, ds \qquad (2.3)$$

[12] Un sistema di coordinate polari è un sistema in cui ogni punto del piano è identificato da un angolo ϕ, noto come coordinata angolare e da una distanza r da un punto fisso detto polo, nota come coordinata radiale. Nel caso di coordinate polari circolari

ove $\boldsymbol{\xi} = (cos\phi, sin\phi)$ è il vettore unità. In tal caso allora, la Trasformata di Radon è valutata integrando lungo la linea $\boldsymbol{\xi r} = p$ prendendo in esame ogni segmento infinitesimo ds di ques'ultima.

Ruotando le coordinate affinché la linea su cui integriamo sia perpendicolare all'asse *p*(figura 6), sia la (2.3) che la (2.2) possono essere riscritte come:

$$\check{f}(p,\phi) = \int_{-\infty}^{\infty}(f(pcos\phi - tsin\phi\ ,\ psin\phi + tcos\phi))dt \qquad (2.4)$$

Fissato $\phi = \Phi$, allora la trasformata di Radon di $f_\phi(p,t) = f(pcos\phi - tsin\phi\ ,\ psin\phi + tcos\phi)$ equivale ad una funzione della sola variabile p e la quale integra $f_\phi(p,t)$ rispetto a *t* con un angolo ϕ fissato:

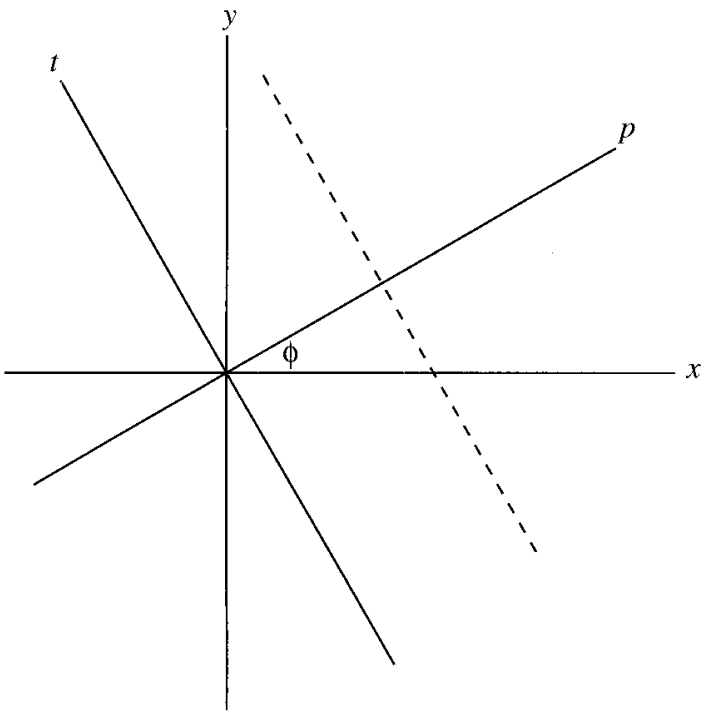


Figura 7: Coordinate ruotate affinché la liena d'integrazione sia perpendicolare all'asse p

Al variare di Φ dunque si avranno varie funzioni di *p*, le quali sono chiamate proiezioni di *f(x,y)* con angolo Φ.

Quanto finora detto per il caso in due dimensioni può essere facilmente esteso anche al caso in cui il numero di dimensioni è *n* con *n* > 2. In tali casi ovviamente,

il vettore di posizione **r** = (x_1,x_2…x_n) ed il vettore unità $\boldsymbol{\xi}$ avranno cardinalità pari ad *n*.

La trasformata di Radon di una funzione ***f(r)***, allora, è ottenuta integrando tale funzione su tutti gli iperpiani di equazione $p = \boldsymbol{\xi r}$, ovvero:

$$\check{f}(p, \boldsymbol{\xi}) = \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} f(\boldsymbol{r}) \delta(p - \boldsymbol{\xi r})\, dx_1 \dots dx_n \qquad (2.5)$$

Le principali proprietà[13] della Trasformata di Radon su *n* dimensioni possono essere ricavate come naturale espansione delle proprietà della stessa applicata al caso 2D. Di seguito riporto la definizione della Trasformata di Radon su tre dimensioni, in quanto, essa è la dimensionalità dell'insieme di dati(i Punti 3D) su cui applico tale strumento:

$$\check{f}(p, \boldsymbol{\xi}) = \iiint_{-\infty}^{\infty} f(\boldsymbol{r}) \delta(p - \boldsymbol{\xi r}) dx dy dz \qquad (2.6)$$

Quanto detto finora è riferito al caso in cui la funzione ***f(r)*** di cui si vuole ricavare la Trasformata di Radon sia continua. L'estensione di quest'ultima al caso discreto, ovvero in cui le coordinate *(x,y)* della funzione *f* sono discrete, venne sviluppata da G. Beylkin nel 1987 (Beylkin, 1987).

2.2 Applicazione della Tecnica

Come anticipato nel capitolo introduttivo, la Trasformata di Radon è una delle tre tecniche utilizzate per realizzare l'algoritmo oggetto di tale tesi. Rispetto al diagramma di flusso in figura 1, la sua applicazione costituisce il primo passo dell'algoritmo. In particolare, essa è applicata al fine di ottenere una suddivisione della nuvola di punti 3D posta in ingresso all'algoritmo in più nuvole di punti 3D ciascuna delle quali può essere ricondotta ad un oggetto presente nella scena 3D. Rispetto, quindi, alla descrizione teorica su fornita, la Trasformata di Radon è stata applicata ad un insieme di dati tridimensionale e discreto.

[13] Tra le proprietà più importanti vi sono: Linearità; Simmetria; Traslazione; Differenziazione; Convoluzione.

L'applicazione della tecnica consta di tre passi:

- Spazzolamento della scena 3D;
- Associazione di ciascun punto appartenente alla nuvola di punti 3D in ingresso all'algoritmo, ad i piani precedentemente ottenuti.
- Restituzione di tutti i piani contenenti almeno un numero minimo di punti *N_min.*

Lo spazzolamento della scena 3D, individuata prendendo i valori di massimo e di minimo di ciascuna coordinata, è ottenuto agendo sui parametri direttori *a*, *b* e *c* e sul vettore normale *d* dell'equazione canonica di un generico piano nello spazio tridimensionale $\boldsymbol{R^3}$:

$$ax + by + cz + d = 0. \qquad (2.7)$$

In particolare, tenendo fissi i valori dei parametri *b*, *c* ed variando i parametri *a* e *d*, da un valore minimo ad un valore massimo rispettivamente con passi *k* e *g*, si ottengono tutte le possibili variazioni lungo l'asse *x*, ovvero:

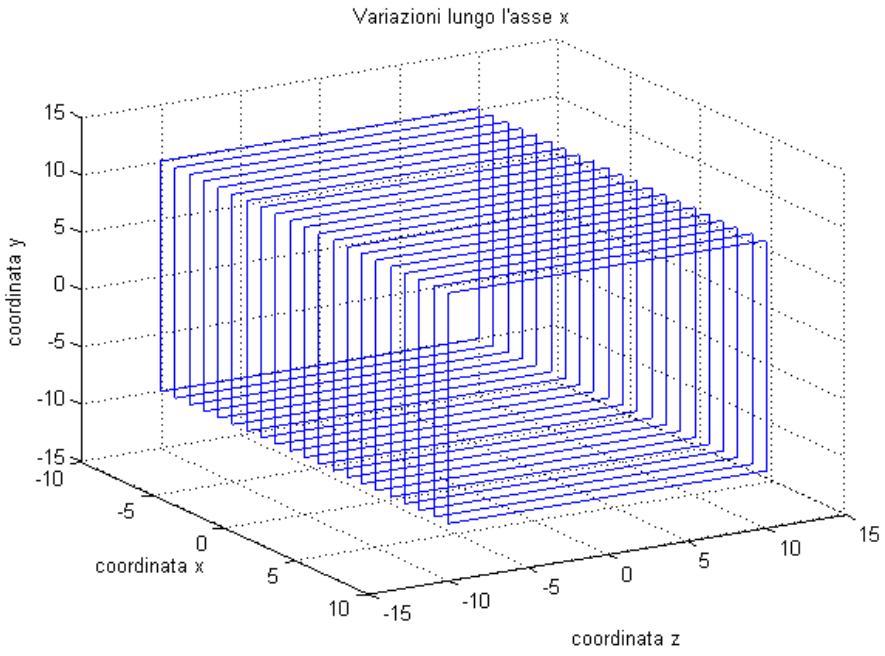


Figura 5: Variazioni lungo l'asse x per valori di -10 < x < 10 e passo di campionamento pari a 1

Analogamente fissati i parametri *a*, *c* ed variando i parametri *b* e *d*, da un valore minimo ad un valore massimo rispettivamente con passi *h* e *g*, si ottengono le possibili variazioni lungo l'asse *y*:

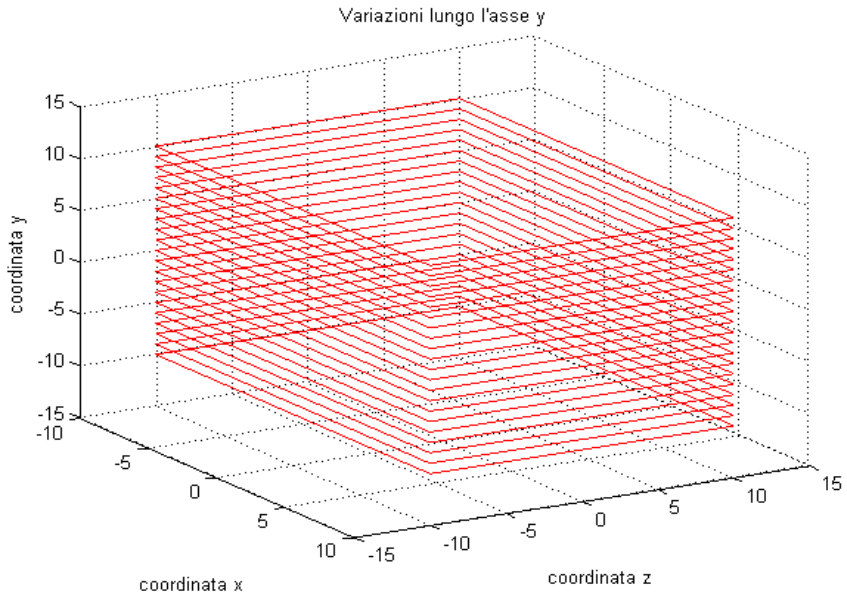


Figura 6: Variazioni lungo l'asse y per valori di -10 < y < 10 e passo di campionamento pari a 1

Infine fissati i parametri *a*, *b* ed variando i parametri *c* e *d*, da un valore minimo ad un valore massimo rispettivamente con passi *f* e *g*, si ottengono le possibili variazioni lungo l'asse *y*:

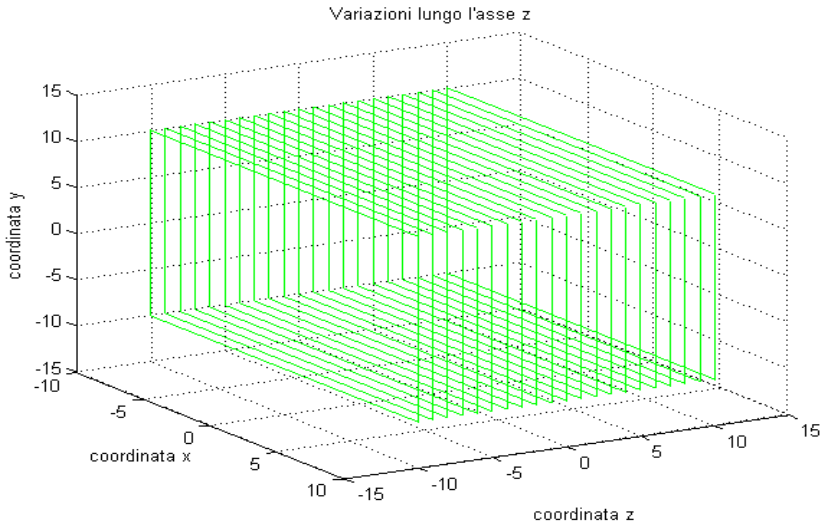


Figura 7: Variazioni lungo l'asse z per valori di -10 < z < 10 e passo di campionamento pari a 1

Lo spazzolamento dell'intera scena 3D è quindi:

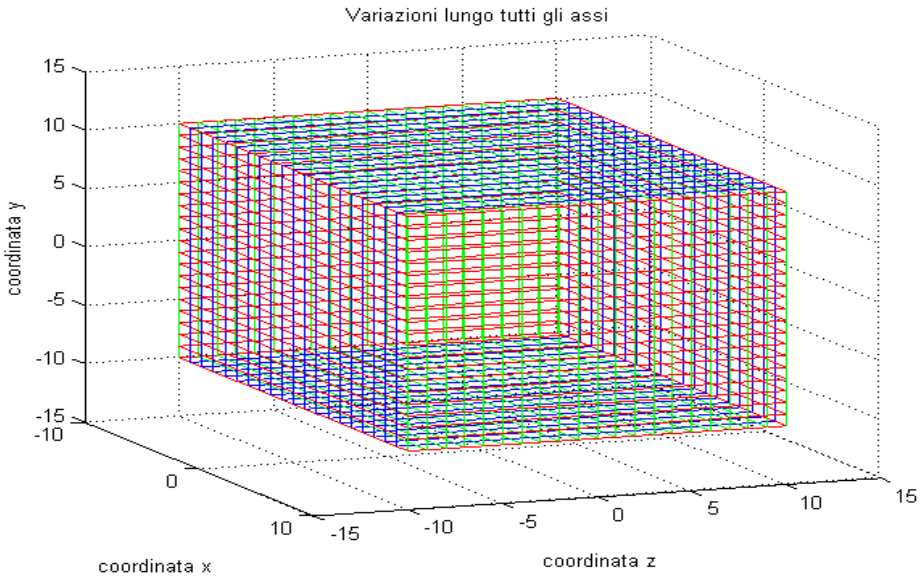


Figura 8: Spazzolamento scena 3D in cui tutti i punti hanno tutte e 3 le coordinate comprese tra -10 ed 10 ed il passo di campionamento è pari ad 1

Ovviamente a seconda dei valori dei passi di campionamento *k*, *h*, *f* e *g*, il numero di piani sarà maggiore o minore e di conseguensa lo spazzolamento sarà più o meno fitto. In particolare più il valore di ognuno dei passi è piccolo più il numero di piani sarà maggiore più lo spazzolamento sarà denso e quindi accurato. Di contro, però, utilizzare passi troppo piccoli comporta un aumento notevole della complessità computazionale. È necessario, quindi, realizzare un compromesso tra accuratezza dello spazzolamento della scena 3D e complessità computazionale.

Una volta ottenuti tutti gli M piani con cui la scena 3D è spazzolata, il passo successivo consiste nell'associare a ciascuno di essi tutti quei punti della nuvola di punti 3D in ingresso all'algoritmo che sono valutati come giacenti sul piano in esame o comunque ad una certa distanza da esso. Per far ciò, viene verificata la seguente disequazione:

$$a_j x_i + b_j y_i + c_j z_i + d_j \leq D \qquad (2.8)$$

Ove: a_j, b_j, c_j *e* d_j sono i coefficienti del *j-simo* piano tra gli M individuati; x_i, y_i *e* z_i sono le coordinate dell'*i-simo* punto tra gli *N* appartenenti alla nuvola di punti; *D* è un parametro di soglia. In particolare il valore del parametro *D* è estremamente importante in quanto esso determina il grado di accuratezza con cui un punto è associato ad un piano. Maggiore è tale valore, maggiore è l'approssimazione che viene fatta e, quindi, l'errore finale commesso. Dall'altra parte, scegliere un *D* troppo piccolo può essere un'imposizione troppo stringente che comporta la non associazione di svariati punti ad uno dei piani individuati. Al fine di trovare il valore di soglia ottima, sono stati effettuati svariate simulazioni su dati sintetici. Rimando al capitolo "Risultati" per i dettagli.

Terminata tale associazione, a ciascuno dei piani individuati è legata una nuvola di punti 3D, ciascuna contenente un numero di punti più piccolo rispetto alla nuvola di punti 3D di partenza. Se tale numero di punti è superiore ad una certa soglia *N_min*, tale nuvola è processata mediante la tecnica *PCA*, descritta nel succesivo capitolo, mentre se è inferiore i punti ad essa appartenente vengono scartati.

2.3 Implementazione della Tecnica

L'implementazione della Trasformata di Radon al fine di ottenere quanto precedentemente detto, è stata effettuata realizzando un'apposita funzione chiamata `radon3D`[14]. L'implementazione presente in MATLAB di tale Trasformata (Radon Matlab),infatti, oltre ad essere applicabile sono per problemi bidimensionale e non tridimensionali, è finalizzata alla risoluzione di problemi tomografici.

[14] Vedi appendice A per il dettaglio della funzione

In particolare, la funzione realizzata è così definita:

```
function [H] = radon_3D (matrice_punti,D)
```

I parametri in ingresso sono:

- `matrice_punti`: Matrice di dimensioni *nxp* ove *n* è il numero di punti della nuvola di punti 3D che si vuole clusterizzare e *p=3*(ovvero le tre coordinate *{x,y,z}* associate ad ogni punto).
- `D`: Valore della soglia *D* prima introdotta. Essa, dunque, costituisce la distanza massima a cui un punto può trovarsi da un generico piano per poter essere associato a quest'ultimo.

Al termine dell'esecuzione la funzione restituisce un vettore `H` di dimensioni *1xn*, ove *n* è pari al numero di cluster individuati. In particolare ciascun elemento di tale vettore è un oggetto della classe `ClusterPunti`[15], la quale comprende tre attributi: *Punti*, *Equazione Piano*, *Autovalori*(tale attributo è impostato dopo l'applicazione della tecnica *PCA*).

Entrando più nel dettaglio delle scelte implementative da me effettuate, le principali riguardano la realizzazione dello spazzolamento della scena 3D. In un primo momento, infatti, esso è stato effettuato fissando il valore del coefficiente *d* dell'equazione del piano e valutando tutte le variazioni lungo i tre assi del sistema di riferimento. In particolare i valori degli altri coefficienti variavano da un valore minimo ottenuto come $d/Coord_{max}$ ad un valore massimo ottenuto invece come $d/Coord_{min}$, ove *Coord = x, y* e *z* a seconda che si stia valutando a_{min}, b_{min} o c_{min}. A seguito di tale implementazione, ho riscontrato i seguenti problemi: lo stesso punto venie associato a più di un piano; valutare valori adeguati per *d* e per la Soglia D è molto difficile.

[15] Vedi Appendice A

Al fine di risolvere tale problema, ho scelto di fissare i valori dei parametri direttori in modo tale che la loro somma in modulo quadro fosse pari ad 1, ovvero:

$$|a|^2 + |b|^2 + |c|^2 = 1 \qquad (2.9)$$

Inoltre, ho scelto di non valutare tutte le variazioni lungo l'asse *y*. Come visto nel precedente paragrafo, variando il coefficiente *b*, si ottengono tutti quei piani paralleli al piano *xz*. Immaginando un ambiente urbano tali piani altro non sono che tutti quelli paralleli alla superficie stradale, ovvero al piano ottico della camera. In tale ambiente non ha pertanto alcun senso valutare tali piani, in quanto gli oggetti presenti nella scena saranno tutti approssimati con piani perpendicolari alla superficie stradali. Ho quindi scelto di porre il coefficiente direttore *b* pari a 0. Al fine di soddisfare la relazione 2.9 le variazioni lungo l'asse *x*, sono state ottenute imponendo che:

$$-1 \leq a \leq 1$$

Da cui ne scaturisce che lo spazzolamento lungo l'asse *z* sarà ottenuto variando il coefficiente *c* nel seguente modo:

$$-\sqrt{1-a^2} \leq c \leq \sqrt{1-a^2}$$

Una volta, ottenuti tutti i piani con cui la scena 3D è stata spazzolata, mediante un semplice ciclo `for` è verificato per ogni piano quali e quanti punti, tra quelli presenti nella nuvola di punti 3D di partenza, soddisfa la relazione 2.8. Tutti i cluster così ottenuti che contengono un numero di punti inferiore a 5 vengono eliminati e non restituiti dalla funzione `radon3D`.

2.3.1 Esempio

Al fine di comprendere meglio in che modo la trasformata di Radon sia stata utilizzata, riporto di seguito un esempio pratico. Per tale esempio, la nuvola di punti 3D presa in esame è:

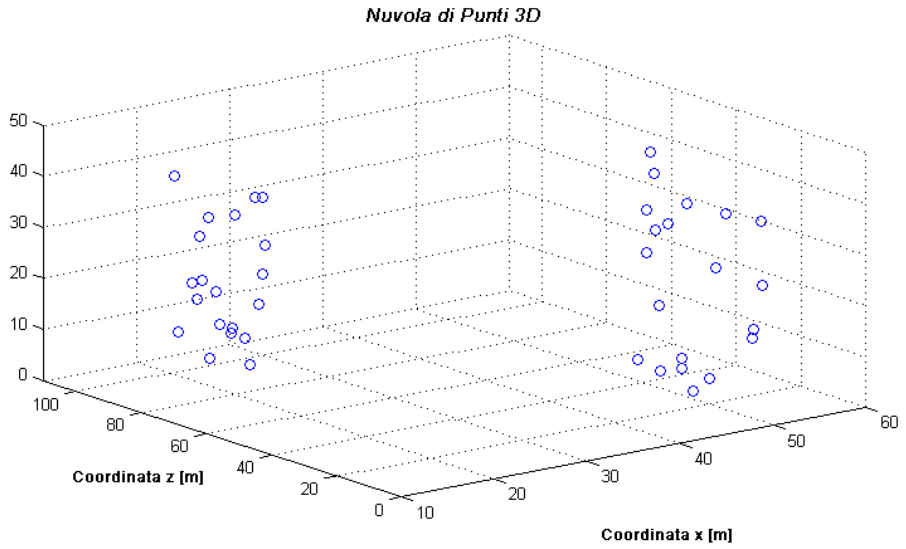


Figura 9:Nuvola punti 3D in esame vista 1

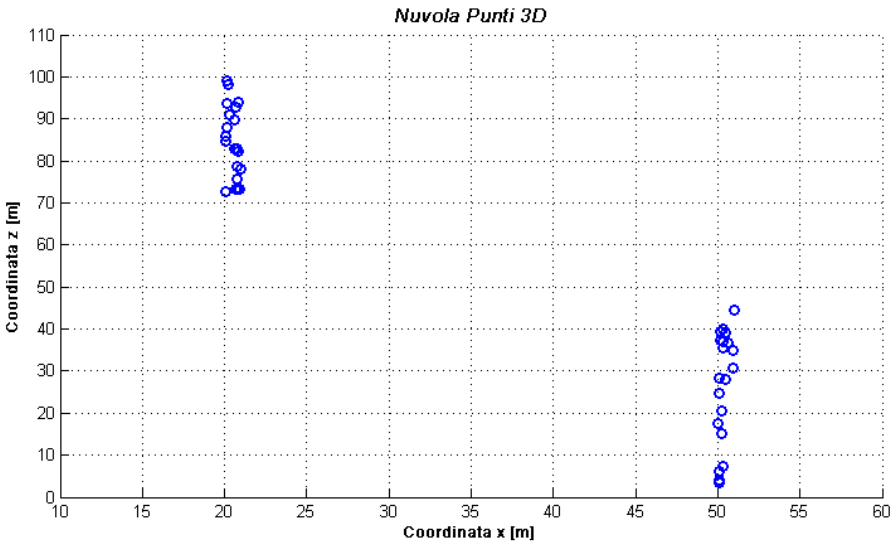


Figura 10: Nuvola punti 3D in esame vista 2

L'obbiettivo che ho voluto raggiungere mediante l'applicazione della trasformata di Radon è, come ampiamento detto precedentemente, suddividere tale nuvola in più sotto nuvole di punti 3D ciascuna comprendente tutti i punti associabili al medesimo oggetto presente nell'ambiente intorno alla camera. Rispetto, dunque, ai punti delle immagini precedenti, le sottonuvole d'interesse sono due e ciascuna composta da 20 punti:

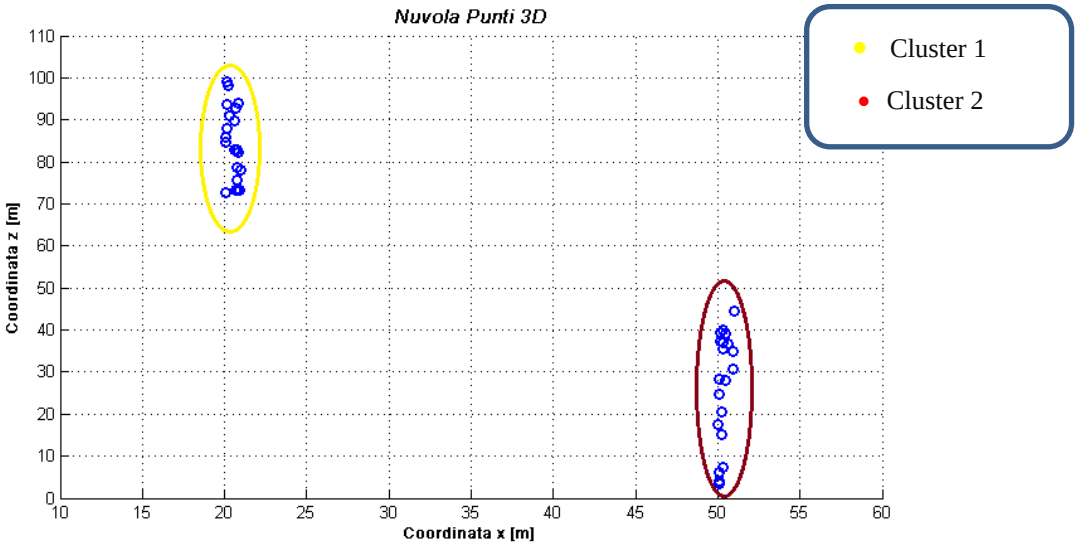


Figura 11: Evidenza Piani D'interesse

Per individuare tali sottonuvole, lo spazzolamento della scena 3D è stato ottenuto variando i coefficiente *a*, *c*, *d* (*b = 0* per i motivi suddetti) con i seguenti passi di campionamento:

- *k = 0.1* passo campionamento associato al coefficiente *a*;
- *f = 0.1* passo campionamento associato al coefficiente *c*;
- *g =0.5* passo campionamento associato al coefficiente *d*;

Mediante tali variazioni la scena 3D i piani individuati al fine di spazzolare la scena 3D sono stati:

```
num_piani =

       22840
```

Figura 12: Numero di piani con cui la scena 3D è stata spazzolata

Con tali passi di campionamento lo spazzolamento ottenuto è abbastanza denso. Al fine di comprendere visualizzare meglio tale spazzolamento in termini visivi, nelle seguenti figure ho stato riportato un piano ogni 20:

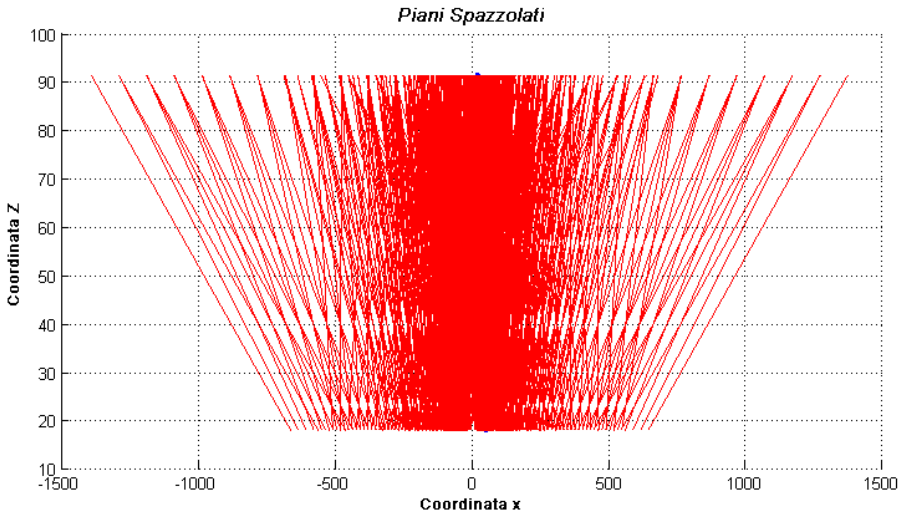


Figura 13: Spazzolamento esempio Radon vista 1

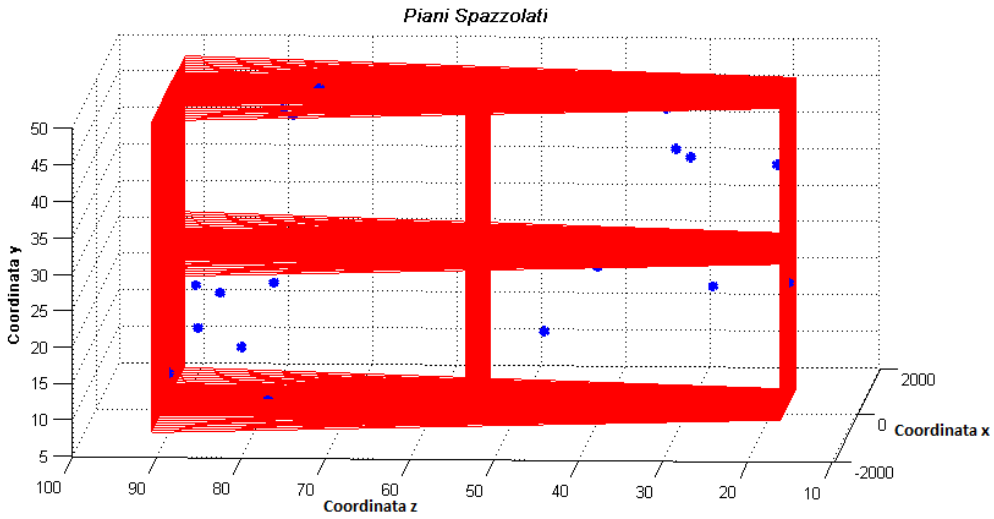


Figura 14: Spazzolamento Esempio Radon vista 2

Una volta spazzolata la scena 3D, ho scelto di associare a ciascuno dei 22840 piani tutti quei punti che soddisfano la relazione 2.8:

$$a_j x_i + b_j y_i + c_j z_i + d_j \leq D$$

ove per tale esempio $D = 0.5$.

A seguito di tale associazione, sono stati scartati tutti quei piani a cui è stato associato un numero di punti inferiore a 20, ovvero tutti quei piani che non comprendono tutti i punti. Di seguito i piani individuati ed i punti ad essi associati:

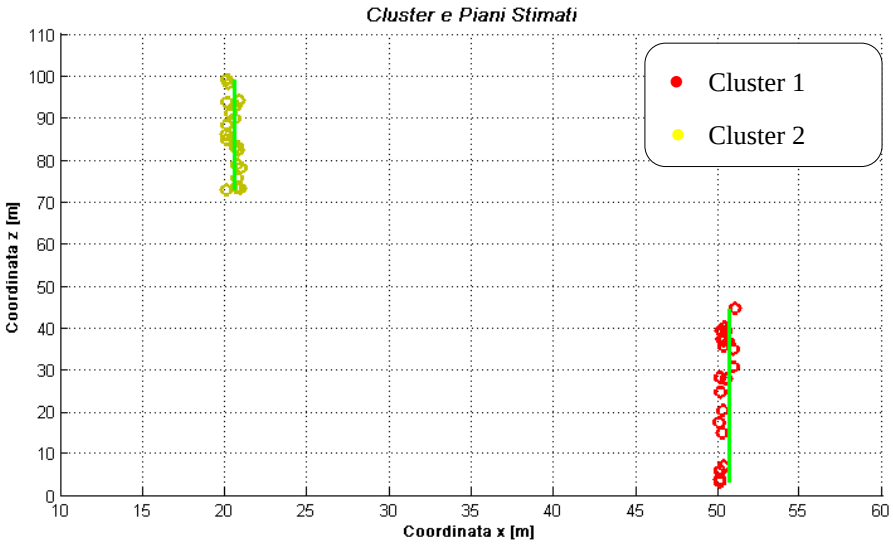


Figura 15: Piani e Cluster Associati Vista 1

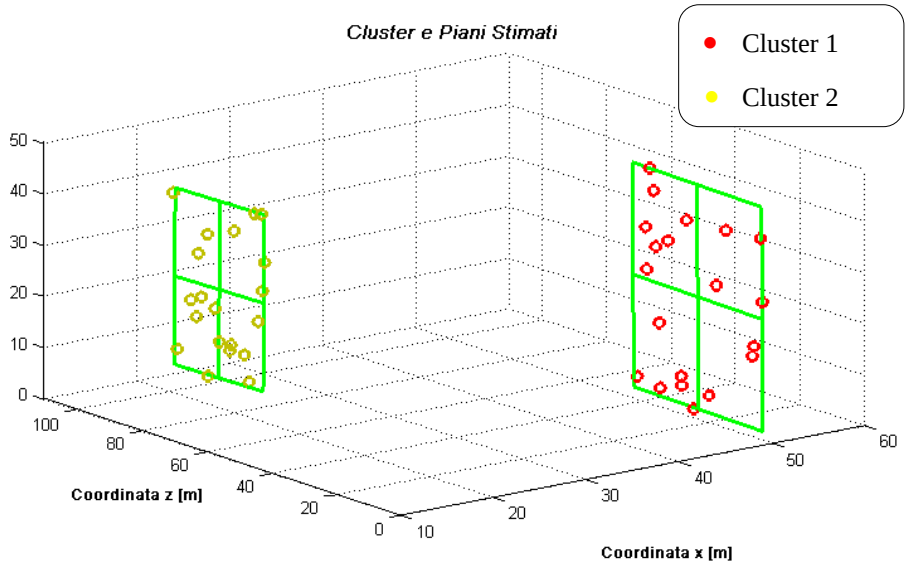


Figura 16: Piani e Cluster Associati Vista 2

CAPITOLO 3 PRICIPAL COMPONENT ANALYSIS

Principal Component Analysis(PCA) è una tecnica di analisi multivariata[16] (Feinstein, 1996) nella quale un insieme di variabili correlate sono trasformate in un insieme di variabili, solitamente più piccolo, di variabili non correlate (Jackson)

3.1 DESCRIZIONE DELLA TECNICA

Come precedentemente definito, la *PCA* è applicata ad un insieme di variabili casuali tra loro correlate (es. più osservazioni del medesimo evento). Definito **x** un vettore costituito da *p* variabili casuali e supponendo di conoscerne la matrice di correlazione[17], il primo passo della PCA consiste nel cercare una funzione lineare, $\boldsymbol{\alpha}_1'\boldsymbol{x}$,degli elementi di **x** che abbia varianza massima, ove $\boldsymbol{\alpha}$ è un vettore costituito da *p* costanti noto come *vettore dei coefficienti*. La funzione cercata è, quindi, del tipo:

$$z_1 = \boldsymbol{\alpha}_1'\boldsymbol{x} = \sum_{j=1}^{p} \alpha_{1j}' x_j$$

Il passo successivo, consiste nel cercare una nuova funzione lineare, , $\boldsymbol{\alpha}_2'\boldsymbol{x}$, non correlata a $\boldsymbol{\alpha}_1'\boldsymbol{x}$ la quale abbia ancora varianza massima.

[16] L'analisi multivariata comprende tutte quelle tecniche che considerano due o più variabili casuali, tra loro relazionate, come un'unica entità e tentano di generare un risultato complessivo sfruttando le relazioni tra di esse

[17] Matrice di dimensione *pxp* in cui gli elementi lungo la diagonale principale, s_i, costituiscono la varianza di ogni singola variabile e gli elementi s_{ij} la covarianza tra l'elemento *i-esimo* e l'elemento *j-esimo*

Quest'ultimo passo viene reiterato *k* volte (*k* al più è pari a *p*), ed in particolare fino a che è possibile trovare una funzione lineare a varianza massima che soddisfi il vincolo di non correlazione con le precedenti funzioni trovate. Ciascuna delle *k* funzioni trovate, prende il nome di *Componente Principale(PC)* ed è statisticamente indipendente[18] dalle altre. Al fine di comprendere meglio tale procedimento, riporto come esempio il caso più semplice, ovvero in cui $p = 2$ (Jolliffe). Per tale esempio, supponiamo di avere 50 osservazioni di un dato evento rispetto alle variabili x_1 e x_2 così distribuite:

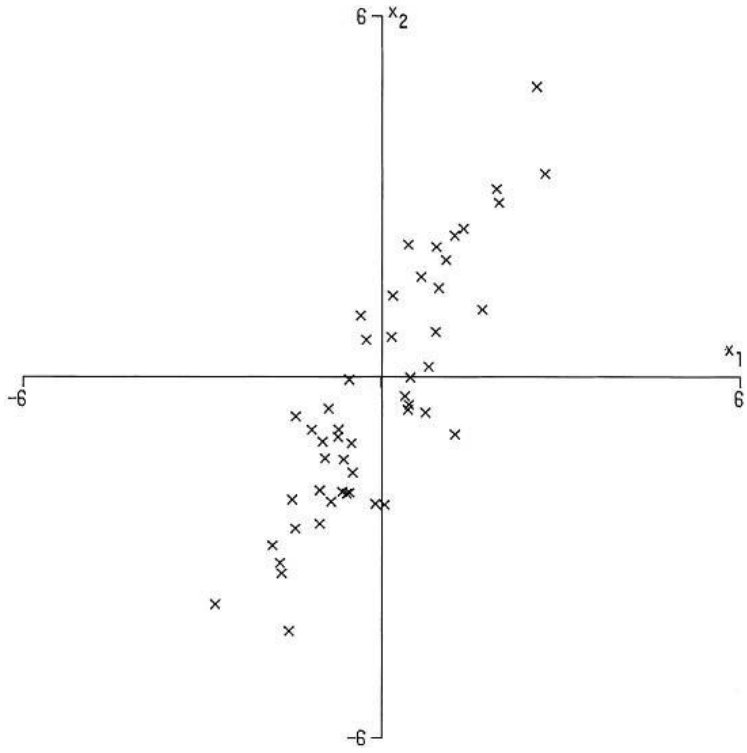


Figura 17: 50 Osservazioni rispetto alle variabili x_1 e x_2

Come si può vedere, le variazioni sono considerevoli rispetto ad entrambe le variabili, ma comunque maggiori rispetto alla variabile x_2.

[18] Due variabili sono statisticamente indipendenti se la loro covarianza è nulla. Questa condizione significa che la conoscenza del valore di una variabile non dà informazioni sul valore che assume l'altra

Graficando ora tali osservazioni rispetto alle PC, z_1 e z_2, si ottiene la seguente distribuzione:

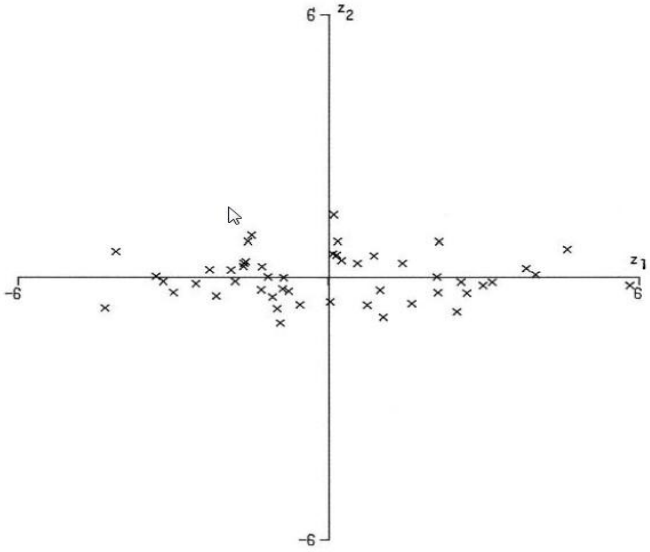


Figura 18: Osservazioni rispetto a z_1 e z_2

In tal caso è evidente come la variazione più grande si abbia lungo la direzione z_1, mentre lungo z_2 essa è abbastanza piccola. Da tale esempio, se ne deduce che la prima PC indica la direzione dove vi è la maggiore variazione delle variabili originali, mentre la seconda indica la direzione ove la variazione è inferiore. Tale esempio può facilmente estendersi al caso in cui *p* abbia una dimensione maggiore di due. In tal caso, infatti, le prime PC identificheranno le direzioni con la maggiore variazione delle variabili originali passando progressivamente a quelle con minor variazione. Tale peculiarità delle PCs ha avuto notevole importanza, come sottolineerò nel prosieguo del capitolo, nella valutazione della robustezza della stima realizzata dall'algortimo.

Dalla definizione di PC appena fornita, è evidente che per ottenere quest'ultime è necessario trovare quel vettore $\boldsymbol{\alpha}_i'$ che massimizzi la $var[\boldsymbol{\alpha}_1'\boldsymbol{x}] = \boldsymbol{\alpha}_i' \sum \boldsymbol{\alpha}_i$, ove $\sum$ è la matrice di correlazione associata al vettore **x**. L'approccio standard utilizzato per valutare il vettore $\boldsymbol{\alpha}_i'$ che massimizzi $var[\boldsymbol{\alpha}_i'\boldsymbol{x}] = \boldsymbol{\alpha}_i' \sum \boldsymbol{\alpha}_i$ è quello che fa uso della tecnica dei moltiplicatori di Lagrange (Vapnyarskii). Tale tecnica è applicata differentemente a seconda della PC che si vuole valutare in quanto la funzione di

costo varia a seconda che si valuti la PC_1 o le successive. Nel caso della PC_1, l'unico vincolo che il vettore $\boldsymbol{\alpha}_1'$ deve rispettare è quello della normalizzazione[19]. Di conseguenza la funzione di costo da massimizzare è:

$$\boldsymbol{\alpha}_1' \sum \boldsymbol{\alpha}_1 - \lambda(\boldsymbol{\alpha}_1' \boldsymbol{\alpha}_1 - 1)$$

ove il parametro λ è noto come moltiplicatore di Lagrange. Tale operazione è effettuata valutando la derivata della funzione di costo e ponendo tale derivata pari a zero, ovvero:

$$(\sum - \lambda \boldsymbol{I}_p)\boldsymbol{\alpha}_1 = 0$$

ove $\boldsymbol{I}_p$ è la matrice identità[20] di dimensione *p x p*. Il moltiplicatore di Lagrange λ, quindi, altro non è che uno dei *p* autovalori della matrice di correlazione $\sum$ e $\boldsymbol{\alpha}_1$ l'autovettore ad esso associato. In particolare in quanto vogliamo trovare il vettore $\boldsymbol{\alpha}_1'$ che massimizzi la funzione di costo sopra definita, la scelta, per quanto detto, ricadrà sull'autovettore associato al maggiore autovalore della matrice $\sum$. Trovata la prima PC, per trovare le successive, qualora esistano, è necessario modificare leggermente la funzione di costo che si vuole massimizzare. Tale modifica è necessaria in quanto le PCs successive alla prima devono, oltre che soddisfare il vincolo di normalizzazione, essere anche non correlate con le precedenti PCs. Applicando ancora la tecnica dei moltiplicatori di Lagrange per trovare la seconda PC, $\boldsymbol{\alpha}_2' \boldsymbol{x}$, si ha che stavolta la funzione di costo che il vettore $\boldsymbol{\alpha}_2'$ deve massimizzare è:

$$\boldsymbol{\alpha}_2' \sum \boldsymbol{\alpha}_2 - \lambda(\boldsymbol{\alpha}_2' \boldsymbol{\alpha}_2 - 1) - \phi \boldsymbol{\alpha}_2' \boldsymbol{\alpha}_1.$$

In tal caso, come si può vedere i moltiplicatori di Lagrange, λ e ϕ, sono due e non più uno soltanto.

[19] $\boldsymbol{\alpha}_1' \boldsymbol{\alpha}_1 = 1$

[20] La **matrice identità** o **matrice identica** o **matrice unità** è una matrice quadrata in cui tutti gli elementi della diagonale principale sono costituiti dal numero 1, mentre i restanti elementi sono costituiti dal numero 0

Derivando tale funzione stavolta rispetto ad $\boldsymbol{\alpha}_2$ e ponendo il risultato pari a zero, si ottiene la seguente relazione:

$$\sum \boldsymbol{\alpha}_2 - \lambda \boldsymbol{\alpha}_2 - \phi \boldsymbol{\alpha}_1 = 0$$

Moltiplicando per $\boldsymbol{\alpha}_1'$ ambo i membri:

$$(\sum - \lambda \boldsymbol{I}_p)\boldsymbol{\alpha}_2 = 0$$

Ancora una volta per massimizzare la funzione di costo si deve prende l'autovalore λ più grande ed il suo autovettore $\boldsymbol{\alpha}_2$. Ovviamente tale λ non può essere uguale al λ trovato per la prima PC in quanto se così fosse non verrebbe soddisfatto il vincolo di non correlazione imposto poiché $\boldsymbol{\alpha}_1'\boldsymbol{\alpha}_2 = 0$. Pertanto λ_2 è il secondo autovalore più grande presente in $\sum$. Il procedimento utilizzato per trovare la seconda PC può essere applicato in maniera eguale per tutte le altre PCs, che, come detto precedentemente, possono essere al più pari a *p*, se con *p* si indica la dimensione del vettore **x**.

3.2 Applicazione della Tecnica

Come anticipato nel capitolo introduttivo, la tecnica appena descritta è stata utilizzata al fine di ricavare il miglior piano associato ai cluster di punti 3D ottenuti a seguito della *Trasformata di Radon*. In particolare, rispetto a quanto detto nel precedente paragrafo, si hanno *n* osservazioni, con *n* pari al numero di punti 3D presenti in ogni cluster, e cardinalità del vettore **x** pari a *p*=3 ovvero le tre coordinate *{x,y,z}*. Poiché in tal caso la matrice di covarianza $\sum$ non è nota a priori, essa viene ottenuta nel seguente modo:

$$\sum = \frac{1}{n}(P-\bar{P})(P-\bar{P})^T$$

Ove, $\boldsymbol{P}$ è la matrice associata ad ogni cluster di punti 3D ed ha pertanto dimensione *3xn*, e $\bar{P}$ è il vettore di dimensioni *1x3* i cui elementi sono le coordinate del centroide dei punti del cluster.

La matrice così ottenuta, è decomposta mediante *decomposizione ai valori singolari (SVD, Single Value Decomposition)* (Spagnolini, 2010) al fine di ottenere gli autovalori e gli autovettori ad essi associati necessari per trovare le PCs. In particolare, se tutti i punti del cluster contenuti nella matrice ***P*** appartengono al medesimo piano *q*, allora la matrice $\sum$ ha rango due e di conseguenza il terzo autovalore è nullo. Poiché, però, i punti sono associati a ciascun piano a meno di una Soglia *D*, nei problemi di stima affrontati si ha una certa varianza lungo tutte e tre le direzioni. Di conseguenza il terzo autovalore è diverso da zero, e quindi le PCs corrispondenti ad **x** sono tre. In particolare le prime due PCs costituiscono una base per il piano *q*, mentre la terza PC costituisce la normale al piano. Il piano così ottenuto costituisce la migliore approssimazione lineare 2D per l'insieme di punti in esame in quanto: le prime due PCs mostrano tutta la varianza possibile dei dati lungo le direzioni *xy*, *xz* e *yz* a seconda di come i punti sono distribuiti; la terza PC, invece, definisce la minima quantità di variazione sui dati, ovvero il termine d'errore nell'approssimazione effettuata.

3.3 Implementazione della Tecnica

La realizzazione di quanto detto finora, è stata abbastanza semplice in quanto nel *Toolbox* di Statistica che MATLAB mette a disposizione è già presente una funzione che implementa la tecnica PCA. In particolare tale funzione è chiamata *princomp* ed a seconda delle informazioni a cui si è interessati esistono tre diverse varianti (mathworks):

```
[COEFF,SCORE] = princomp(X)
[COEFF,SCORE,latent] = princomp(X)
[COEFF,SCORE,latent,tsquare] = princomp(X)
```

Indipendentemente dalla variante utilizzata, tale funzione richiede in ingresso una matrice `X` di dimensione *nxp*, ove *n* è il numero delle osservazioni e *p* indica le coordinate del sistema di riferimento di tali osservazioni. Tale matrice, quindi,

contiene l'insieme delle osservazioni alle quali si vuole applicare la tecnica PCA al fine di conoscerne le PCs. Rispetto al nostro problema, tale matrice altro non è che la matrice ***P*** prima introdotta in cui pertanto *n* dipenderà dalla dimensione di ogni singola sottonuvola mentre *p* è sempre pari a 3. Le informazioni restituite dalla funzione *princomp*, invece, cambiano a seconda della variante scelta. Prendendo in esame la terza variante le informazioni restituite sono:

- `COEFF:` matrice di dimensione *pxp* ove ciascuna colonna altro non è che l'autovettore associato alle PCs. Ciascun elemento della colonna i-esima costituisce quindi uno dei coefficienti del vettore $\boldsymbol{\alpha}_i'$ indicato nella trattazione teorica. In particolare la prima colonna sarà associata alla prima PC, la seconda colonna alla seconda PCs e così via. Nel nostro caso tale matrice avrà dimensione *3x3*.
- `SCORE:` Matrice di dimensione *nxm*, ove *m* è il numero di PCs ottenute(ricordiamo che al più *m=p*). Tale matrice contiene la rappresentazione delle osservazioni rispetto alle PCs e non più rispetto alle variabili di partenza **x**. Nel nostro caso tale matrice sarà ancora pari a *nx3* in quanto *m=p*.
- `latent:` è un vettore contenente gli autovalori della matrice di covarianza associata alla matrice delle osservazioni X. Ciascuno di tali autovalori è associato ad una PCs. In particolare il primo elemento di tale vettore è l'autovalore più grande ed è quindi associato alla prima PCs, il secondo elemento è il secondo autovalore più grande ed è associato alla seconda PCs, e così via per gli altri. Nel nostro caso tale vettore contiene 3 elementi.
- `Tsquare:` Vettore contenente la statistica T2 di Hotelling's[21] per ciascun punto presente nella matrice X. Tale parametro non è stato utilizzato.

[21] Distanza di ciascuna osservazione multivariata dal centro dell'insieme delle osservazioni

Delle tre varianti messe a disposizione da MATLAB, è stata utilizzata la seconda, ovvero:

```
[COEFF, SCORE, latent] = princomp(X)
```

Tale scelta è dovuta al fatto che per l'algoritmo realizzato i soli parametri necessari sono `COEFF` e `latent`. In particolare `COEFF` è utilizzato per ottenere l'equazione del miglior piano associato all'insieme di punti di ogni cluster, `latent`, invece, è utilizzato per valutare se la stima del piano calcolato è robusta o meno. L'equazione del miglior piano adattato ai punti 3D di ogni cluster, è quindi ottenuta come:

```
(X - meanX)*normal = 0
```

ove:

- `X`: è la matrice delle osservazioni prima introdotta;
- `meanX`: è un vettore di dimensione *1x3*, contenente le coordinate *{x,y,z}* del punto 3D medio, ottenuto mediando le singole coordinate *{x,y,z}* di tutti i punti 3D del cluster in esame.
- `normal`: è un vettore di dimensione *3x1*, costituito dagli elementi della terza colonna della matrice `COEFF`.

Il piano in questione passa per il punto medio e la sua normale, ovvero la distanza perpendicolare all'origine è pari al prodotto matriciale `meanX*normal`. Il piano così ricavato garantisce che la somma dell'errore al quadrato sia minima. Tale errore è valutando ricavando la distanza perpendicolare di ogni punto 3D appartenente al cluster dal piano ottenuto. Tale distanza altro non è che il prodotto scalare tra il punto 3D in esame centrato e la normale del piano, ovvero:

```
error = abs((X-repmat(meanX,n,1))*normal);
```

3.3.1 Esempio

Riportiamo di seguito un esempio di piano ricavato per un dato cluster di punti 3D. In particolare, per tale esempio la matrice delle osservazioni X è costituita da **80** punti 3D:

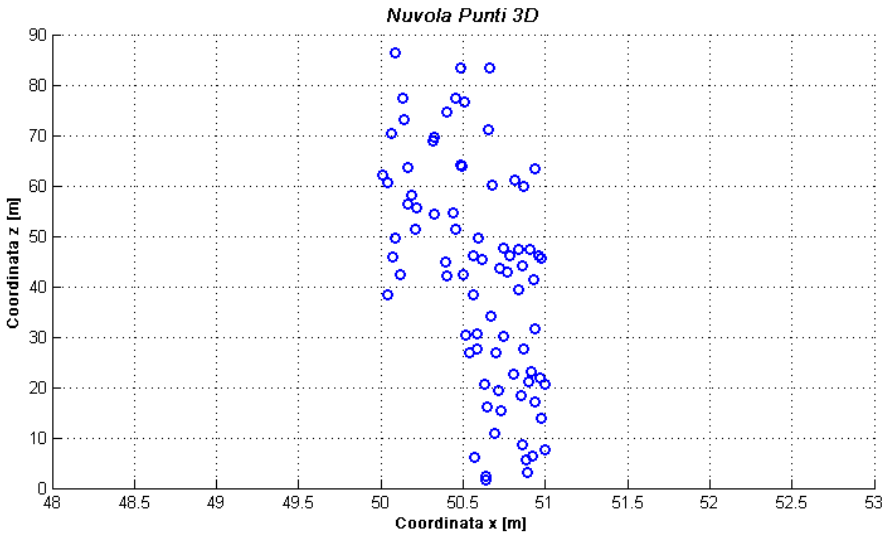


Figura 19: Nuvola Punti 3D

A seguito dell'applicazione della Trasformata di Radon è individuato il seguente piano a cui sono associati tutti e **80** i punti:

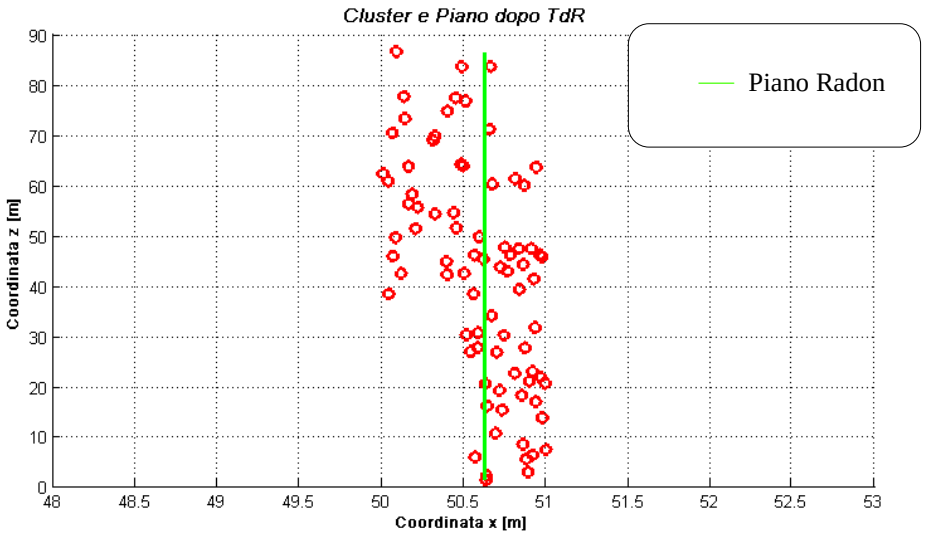


Figura 20: Sottonuvola e Piano Associato dopo l'Applicazione della *TdR*.

Applicando la funzione *princomp* nella variante scelta a tale nuvola di punti, i valori dei parametri prima indicati sono:

```
coeff =

   -0.00713701402799251   -0.00165958110818094    0.999973154050302
     0.06096815976257      0.998137519775087    0.00209167693572546
    0.998114195132967     -0.0609814513420521    0.00702253987435793


normal =

    0.999973154050302
   0.00209167693572546
   0.00702253987435793


meanX =

    50.5907760887275     24.516217508878     42.2353310179853
```

Figura 21: Valori parametri restituiti da *princomp*

Il piano stimato è:

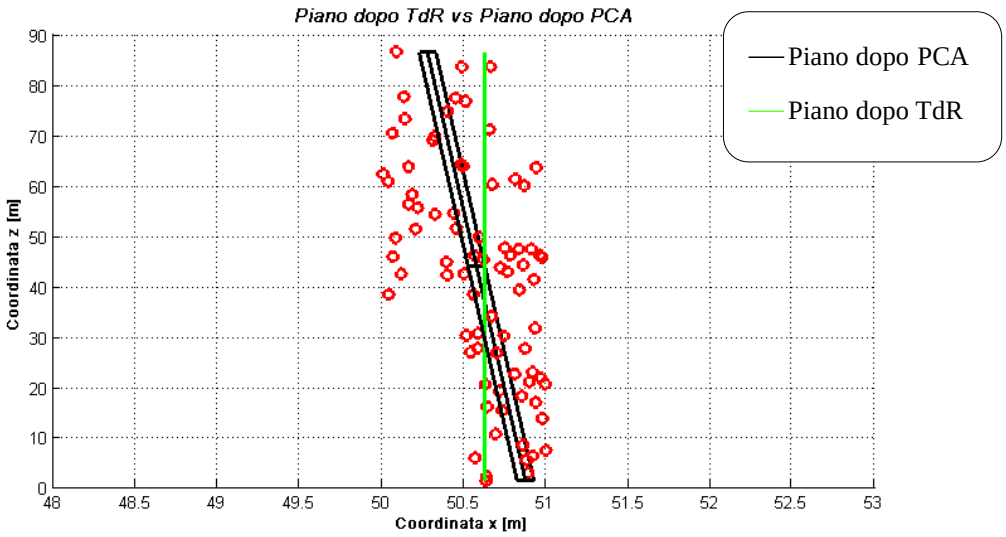


Figura 22:Piano dopo TdR Vs Piano dopo PCA

L'equazione di tale piano è:

$$0.999x + 0.0020y + 0.0070z + 50.93 = 0$$

L'errore, valutato come somma in modulo quadro della distanza di ciascun punto da tale piano è

```
somma_modulo_quadro_errore =

    4.61780715112746
```

Figura 23: somma modulo quadro di tali errori

Per verificare che la stima effettuata sia corretta, come già anticipato nel capitolo introduttivo, è effettuato un controllo sui valori dei tre autovalori associati alle tre PCs. In particolare, il valore del terzo autovalore è confrontato con l'autovalore associato alla seconda PC. Come sottolineato nella trattazione teorica, infatti, se la stima realizzata è corretta, l'intera varianza dei dati è mostrata dalle prime due PCs. Il terzo autovalore di conseguenza ha un valore molto più piccolo rispetto agli autovalori corrispondenti alla prima ed alla seconda PCs. Se così non fosse significherebbe che lungo tutte e tre le direzioni vi è molta varianza e quindi che la stima realizzata non è corretta. Ciò potrebbe essere causato dalla presenza tra i punti presenti nella sottonuvola in esame di *outliers*. Da un punto di vista implementativo, tale confronto è stato effettuato grazie al vettore `latent` restituito dalla funzione *princomp* di MATLAB. Per valutare in che rapporto gli autovalori stanno tra loro, sono state effettuate apposite misurazioni su un insieme di dati sintetici. Da tali misurazioni è emerso come tale rapporto è strettamente dipendente dal valore di soglia *D* scelto. Rimando al capitolo *"Risultati"* per il dettaglio delle misurazioni effettuate. Rispetto all'esempio qui riportato, si ha che:

```
autovalori =

    0.742294963487652    0.257617571872083    8.74646402648811e-005
```

Figura 24: Autovalori restituiti da PCA

Avendo per tale esempio scelto *D = 0.5*, allora per avere una stima corretta il rapporto λ_2/λ_3 deve essere maggiore di 2500. Tale condizione è con i valori su scritti rispettata infatti $\frac{\lambda_2}{\lambda_3} = 2945$. Qualora tale condizione sugli autovalori non viene rispettata, allora viene eseguito l'algoritmo RANSAC al fine di individuare eventuali *outliers*.

CAPITOLO 4
RANDOM SAMPLE CONSENSUS

RANdom SAmple Consensus(RANSAC) è un algoritmo, realizzato da Fischler e Bolles, che permette di adattare un modello cercato ad un insieme di osservazioni al fine di trovare i valori migliori dei parametri liberi di tale modello. L'obbiettivo di tale algoritmo è, quindi, il medesimo di quello della PCA e più in generale di tutte le tecniche di stima di parametri ottimizzata. La grande differenza rispetto alla PCA è, però, che essa non prevede alcun meccanismo che permetta di rilevare la presenza di *outliers* tra le osservazioni, di conseguenza la stima da essa restituita è, come visto, ottenuta tenendo conto anche di quest'ultimi. Il RANSAC, invece, permette di smussare i dati individuando la presenza di *outliers* in essi. La stima ottenuta mediante il RANSAC è pertanto robusta aglii *outliers*,.

4.1 Descrizione Algoritmo

Come precedentemente definito, l'algoritmo è applicato ad un insieme **P** di *p* osservazioni. In particolare, esso è formalmente definito come (Fishler & Bolles, March 1980):

- Dato un modello che richiede minimo *n* osservazioni, con $n < p$, per poterne stimare i parametri liberi, allora in maniera casuale viene scelto un sottoinsieme ***S1***di *n* osservazioni da **P** e viene istanziato il modello *M1*. Tale modello è utilizzato per determinare il sottoinsieme ***S1**** , il quale è costituito da tutti i punti di **P** che danno luogo ad un errore rispetto al modello *M1* inferiore ad una soglia, nota come *errore tollerato*. Tale sottoinsieme ***S1**** è chiamato *consensus set* di ***S1*** ed i punti in esso contenuti sono definiti *inliers*. I restanti punti di **P** che invece generano un errore superiore a quello tollerato sono detti *outliers*.

- Se la dimensione di ***S1**** è più grande di una soglia *t*, la quale è una funzione della stima del numero di *outliers* presenti in **P**, allora ***S1**** è utilizzato per ricavare un nuovo modello *M1** mediante stima ai minimi quadrati.
- Se la dimensione di ***S1**** è inferiore alla soglia t, allora in maniera casuale viene preso un nuovo insieme di *n* punti ***S2*** da **P** e ripetuto il processo suddetto. Se dopo un certo numero di tentativi non è trovato alcun *consensus set* con un numero di punti maggiore di *t* l'algoritmo termina e restituisce fallito.

Da tale descrizione del funzionamento dell'algoritmo è evidente come esso si fondi su tre parametri, ovvero:

- *Errore tollerato*: sulla base di tale parametro si decide se un punto di **P** appartiene o meno al sottoinsieme ***SI***. Solitamente il valore di tale parametro è ottenuto sulla base di evidenze sperimentali, ovvero la deviazione è ottenuta perturbando le osservazioni, valutando il modello ad esse associato e misurando l'errore rispetto al modello non perturbato. L'errore tollerato è posto ad un valore pari ad uno o due punti di deviazione standard dall'errore medio.
- *Tentativi massimi(k):* indica il numero massimo di tentativi che l'algoritmo effettua per ricercare un *consensus set* e quindi un modello corretto. Tale parametro può essere ottenuto valutando il valore atteso di tentativi *E(k)* richiesto per trovare un sottoinsieme di *n* osservazioni buone. Ovviamente tale scelta non assicura né che l'algoritmo trovi un modello corretto, né che venga trovato il modello ottimo. In particolare, detta *w* la probabilità che una data osservazione generi un errore inferiore all'*errore tollerato*, allora:

$$E(k) = w^{-n}.$$

Solitamente, però, il numero di tentativi effettuati dall'algoritmo è superiore a tale valore atteso. Anche in questo caso, infatti, è preferibile porre tale parametro pari ad uno o due punti in più la deviazione standard di *k*, la quale è pari a:

$$SD(k) = \sqrt{(1 - w^{-n})} * ({}^{1}/_{w^{-n}})$$

Qualora, invece, si richieda con probabilità *z* che l'algoritmo restituisca un modello corretto è necessario effettuare un numero di tentativi *k* pari a:

$$k = {}^{\log(1-z)}/_{\log(1-b)}$$

ove $(1 - z) = (1 - b)^k$.

- *Soglia t*: indica il numero di punti minimo che un certo *consensus set* deve avere affinché il corrispettivo modello sia ritenuto corretto. Tale valore deve essere scelto grande abbastanza affinché il modello giudicato corretto lo sia effettivamente per l'insieme di osservazioni a cui l'algoritmo è applicato. A tal fine, detta *y* la probabilità che una data osservazione generi un errore inferiore all'errore tollerato di un modello non corretto, bisogna verificare che la quantità y^{t-n} sia molto piccola. Poiché per valutare *y* non esiste una procedura generale, si è soliti porre *y* ad un valore inferiore a *w*.

Al fine di comprendere meglio l'algoritmo appena descritto, riporto di seguito un diagramma di flusso che ne descrive i singoli passi:

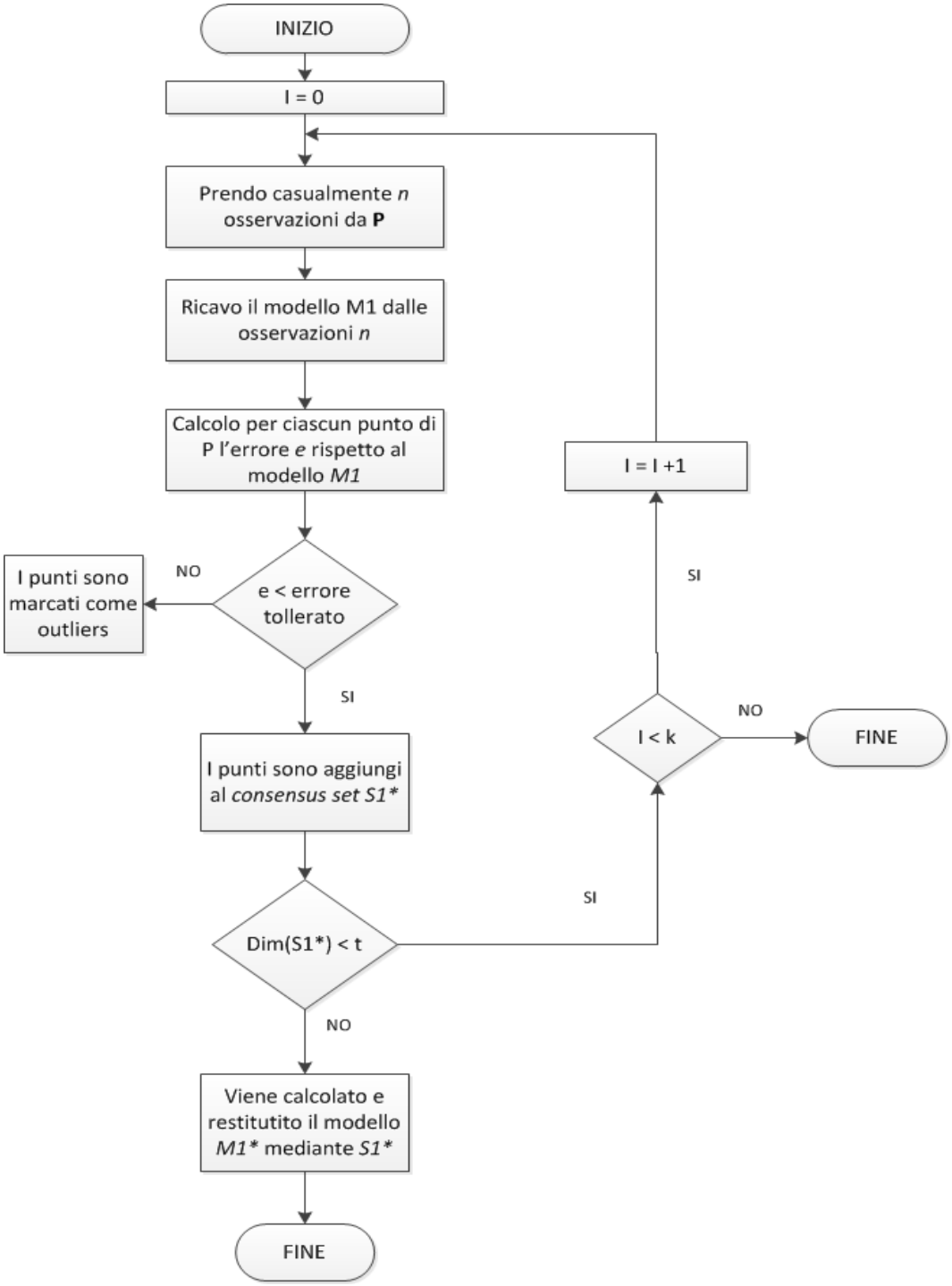


Figura 25: Diagramma di flusso algoritmo RANSAC teorico

4.2 Applicazione dell'Algoritmo

Come anticipato nel capitolo introduttivo, l'algoritmo RANSAC è utilizzato a seguito della tecnica PCA solamente nel caso in cui, i tre autovalori associati alle tre PCs che quest'ultima restituisce, non soddisfano determinate condizioni. Ciò, come già detto, è riconducibile alla presenza di *outliers* nelle sottonuvole di punti 3D restituite dalla tecnica *SfM*. Prima di proseguire con la descrizione di come ho applicato l'algoritmo RANSAC per i miei scopi, ritengo opportuno chiarire perché esso viene applicato solo dopo la tecnica PCA e non in sostituzione ad essa. Se, infatti, è vero che il RANSAC permette di effettuare una stima di un modello cercato robusta alla presenza degli *outliers* (vantaggio rispetto alla PCA), è altrettanto vero che se il numero di osservazioni a cui adattare il modello è elevato, allora il numero d'iterazioni necessarie per essere sicuri di aver stimato un modello ottimo e non solo corretto cresce notevolmente e con esse il tempo di elaborazione. Inoltre, a differenza della PCA, è necessario settare le soglie prima introdotte in maniera specifica per ogni problema che si vuole affrontare. Poiché l'algoritmo oggetto di tale tesi ha come scopo quello di adattarsi ad una ricostruzione tridimensionale di un ambiente sempre diverso e di garantire tempi di elaborazione i più bassi possibili, ho scelto di utilizzare il RANSAC solamente nel caso suddetto. Fatta questa premessa, l'applicazione del RANSAC ha avuto come obbiettivo quello d'individuare eventuali *outliers* presenti nelle sottonuvole di punti 3D. A tal fine, detta **P** la matrice contenente i punti 3D della sottonuvola di cui vogliamo stimare il piano, i passi effettuati dall'algoritmo sono i seguenti:

- Si calcola l'*errore tollerato*, sulla base del piano stimato al passo precedente mediante la tecnica PCA. In particolare, il valore di tale parametro è ottenuto valutando il valor medio delle distanze di ciascuno dei punti di **P** rispetto al piano stimato con la tecnica PCA. Indico con **d** il vettore di tali distanze;

- Vengono scelti in maniera casuale *n*=3 punti 3D tra quelli appartenenti al la matrice **P**. Tali punti costituiscono il sottoinsieme ***S1***;
- Si verifica che i punti 3D scelti non siano in configurazione degenere, ovvero non siano collineari tra loro.
- Si calcola il piano *q* passante per i punti del sottoinsieme ***S1***. Tale piano costituisce il modello *M1*.
- Per ciascun punto della matrice **P** viene valutato l'errore commesso, ovvero la distanza dal piano *q*. Se tale errore è inferiore all'*errore tollerato*, il punto è considerato *inlier* e inserito nel *consensus set*, altrimenti è considerato un *outlier*.
- Gli ultimi quattro passi vengono ripetuti *k* volte, ove *k* ho scelto di porlo come un parametro opzionale. Di default *k* = 100.
- Viene restituito il *consensus set* contenente il maggior numero di punti.

Rispetto alla descrizione teorica sopra riportata, nell'applicazione dell'algoritmo RANSAC ho scelto di:

- non valutare il numero massimo di iterazioni in funzione della probabilità di ottenere un modello corretto.
- non utilizzare il parametro di soglia *t*.
- verificare che i punti scelti casualmente non siano collineari tra di loro.

La seconda scelta è scaturita dal fatto che tale algoritmo è applicato non sulla nuvola di punti 3D di partenza, ma su cluster di punti 3D ottenuti mediante la trasformata di Radon, ovvero su un insieme di punti in cui il numero di *outliers* è decisamente inferiore al numero di *inliers*.

Riporto anche in tal caso il diagramma di flusso con i passi dell'algoritmo:

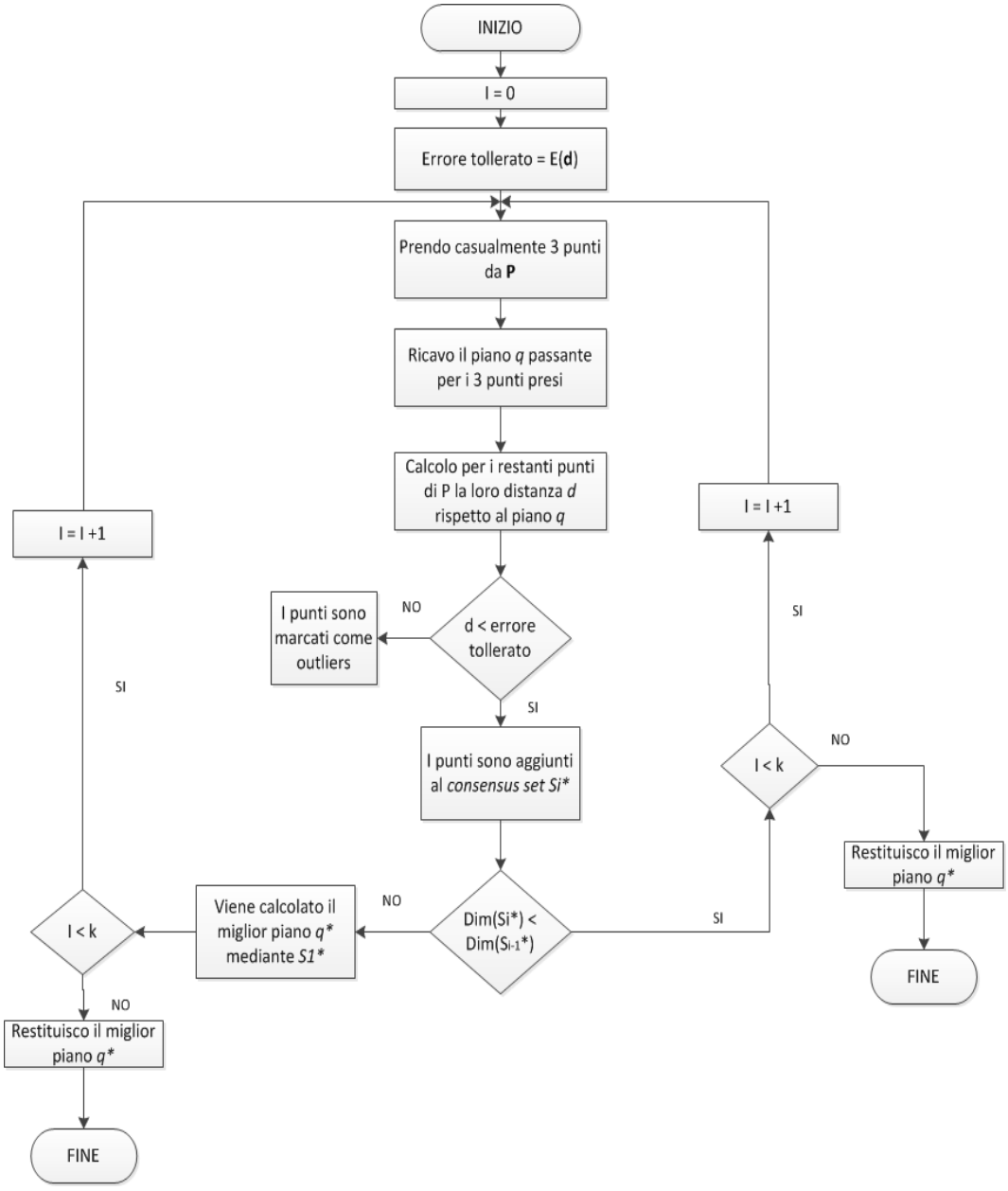


Figura 26: Diagramma di flusso algoritmo RANSAC versione applicata

4.3 Implementazione dell'Algoritmo

L'implementazione dell'algoritmo RANSAC con gli accorgimenti descritti nel precedente paragrafo, è stata effettuata prendendo spunto da un implementazione effettuata da ST Microelectronics. In particolare, poiché quest'ultima è stata realizzata in linguaggio C++, ho dovuto prima riportare tale implementazione in MATLAB, ed in seguito apportare alcune modifiche al fine di ottenere quanto voluto. A tal proposito, ho creato la seguente funzione:

```
[coeff_piano, inliers] = get_plane_ransac(X, err_toll)
```

Tale funzione riceve in ingresso:

- una matrice **x** di dimensione *nxp* ove *n* è il numero di osservazioni a disposizione e *p* è il numero di coordinate del sistema di riferimento di quest'ultime;
- ed un valore di soglia `err_toll` utilizzato per decidere se un dato punto è da classificare come *inlier* o come *outlier*.

Nel nostro caso **x** è la matrice **P** finora utilizzata per identificare l'insieme di punti 3D di ogni sottonuvola. `err_toll`, invece, è ottenuto valutando il valor medio delle distanze di ciascun punto di **P** rispetto al piano stimato dalla PCA, che indichiamo come q_0. Pertanto, detto, `error` il vettore di dimensione *nx1* contenente la distanza di ogni punto di **P** da q_0, si ha che:

```
err_toll = mean(error)
```

I parametri restituiti sono invece: `coeff_piano` ed `inliers`. Il primo è un vettore di dimensione *4x1* contenente i coefficienti del piano stimato. Il secondo invece è un vettore di dimensione *ix1*, ove *i* è un valore variabile che dipende dal numero di *inliers* associati al piano *q* giudicato ottimo, il quale contiene gli indici delle righe di **P** associati ai punti classificati come *inliers*.

All'interno di tale funzione, viene richiamata un'altra funzione da me creata, la quale implementa l'algoritmo RANSAC vero e proprio, ovvero:

```
[inliers] = ransac(X,Piano_x_3_pnt_fn, dst_pnt_piano_fn,…,
        conf_degen_fn, err_toll, num_max_tentativi)
```

Tale funzione richiede in ingresso, oltre alla matrice `X` ed al parametro `err_toll`, un ulteriore parametro e tre riferimenti a tre funzioni[22]. Il parametro ulteriore è denominato `num_max_tentativi`, ed indica il numero massimo di iterazioni compiute dall'algoritmo per trovare un piano corretto. I riferimenti alle tre funzioni sono invece:

- `Piano_x_3_pnt_fn`: Riferimento ad una funzione che permette di valutare il piano passante per i tre punti 3D scelti casualmente dalla matrice X. Dati tre punti 3D esiste uno ed un solo piano passante per questi punti.
- `dst_pnt_piano_fn`: Riferimento ad una funzione che permette di valutare la distanza di un dato punto 3D da un dato piano.
- `conf_degen_fn`: Riferimento ad una funzione che permette di verificare che dati tre punti 3D essi non siano in una configurazione degenere, ovvero non siano collineari tra loro. In tal caso, infatti, il piano passante per questi tre punti non è più uno solo ma infiniti.

Il parametro restituito è invece il vettore *inliers* prima definito. Una volta noti i punti 3D del cluster in esame classificati inliers, il piano ottimo associato a tali punti è stimato mediante la tecnica PCA. In tal modo, infatti, è immediato valutare il vantaggio ottenuto in termini di diminuzione dell'errore grazie all'algoritmo RANSAC.

[22] Il dettaglio di tali funzioni è riportato nell'appendice A

4.3.1 Esempio

Così come fatto nel capitolo precedente, riporto di seguito un esempio di stima del piano ricavato mediante l'applicazione dell'algoritmo RANSAC. In particolare il caso riportato prende in esame una sottonuvola di **102** punti 3D così distribuiti:

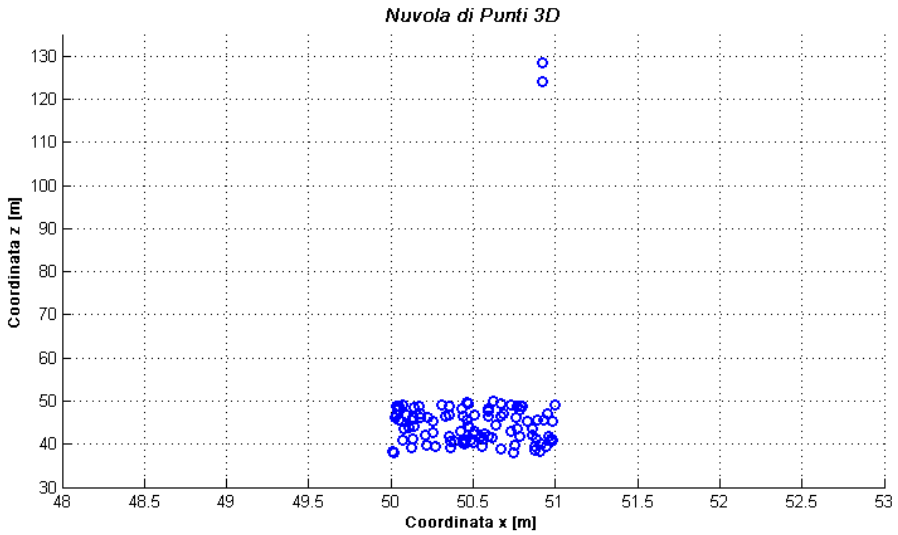


Figura 27: Visualizzazione Cluster di Punti 3D vista 1

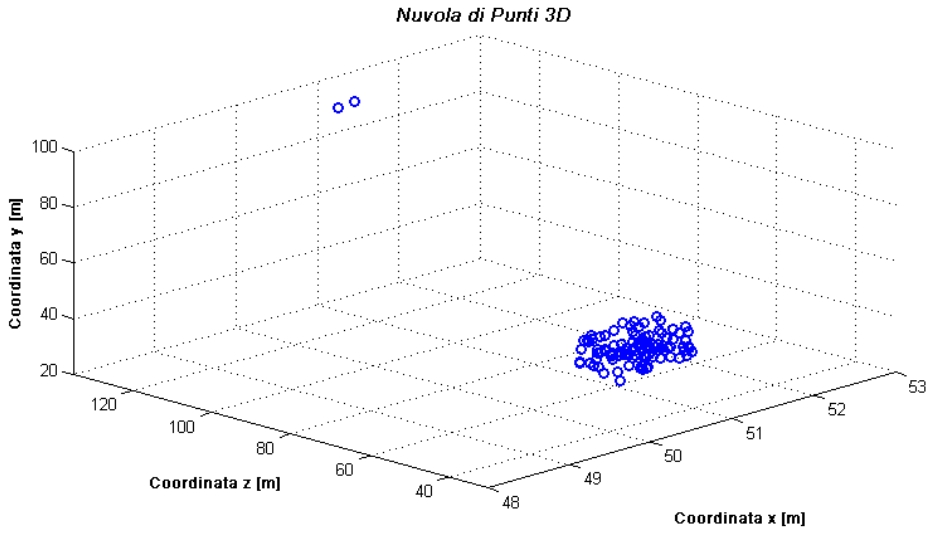


Figura 28: Visualizzazione Cluster di Punti 3D vista 2

Come è abbastanza evidente dalle viste sopra, la sottonuvola presenta 2 **outlier** e 100 **inliers**, ovvero:

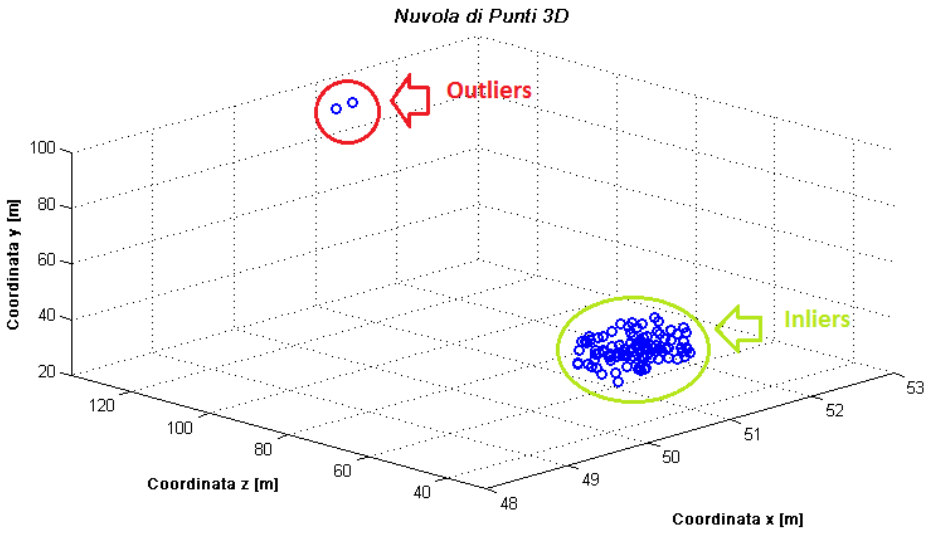


Figura 29:Inliers ed Outliers Attesi

A seguito della tecnica PCA si ha che il piano stimato è:

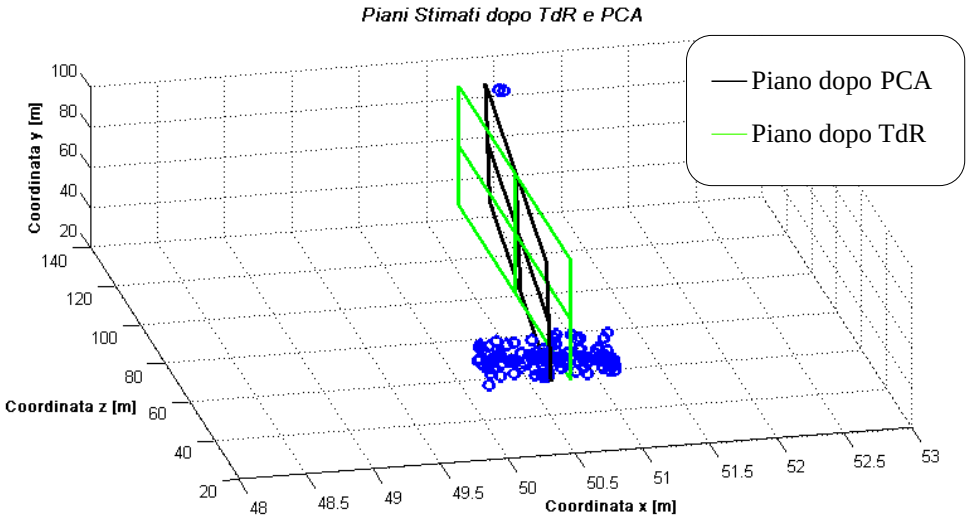


Figura 30: Piano Stimato mediante PCA

Come si può vedere a causa dei due *outliers*, la stima ottenuta con la sola PCA è totalmente errata.

Ciò è confermato andando a verificare il rapporto tra gli autovalori, il quale avendo utilizzato una Soglia *D=0.5*, deve essere maggiore di 2500. Gli autovalori restituiti dalla PCA ed l'errore associato al piano stimato sono:

```
autovalori =

        0.935831093965239        0.0637400068449416       0.000428899189819165


somma_modulo_quadro_errore =

           9.08504139825959  [m]
```

Figura 31: Autovalori e somma modulo quadro dell'errore associati al piano stimato

Con tali valori, si ha che il rapporto $\frac{\lambda_2}{\lambda_3} = 157{,}5$ è molto inferiore a 2500. Applicando a tale sottonuvola la funzione `get_plane_ransac`, essa restituirà gli indici dei punti 3D di X classificati *inliers* e quindi anche di quelli classificati *outliers*:

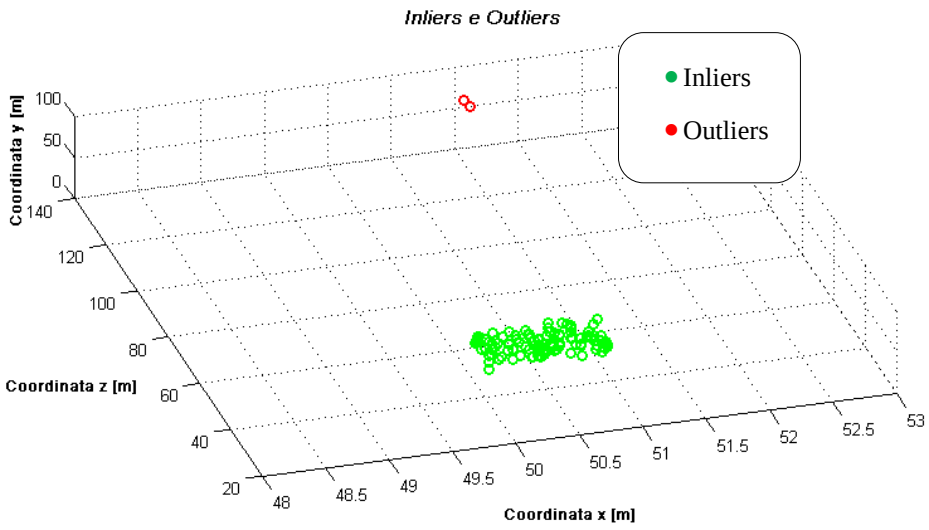


Figura 32: Inliers e Outliers

Che è esattamente quello che mi aspettavo.

Applicando ai soli *inliers* la tecnica PCA si ottiene che il piano ottimo passante per questi punti è:

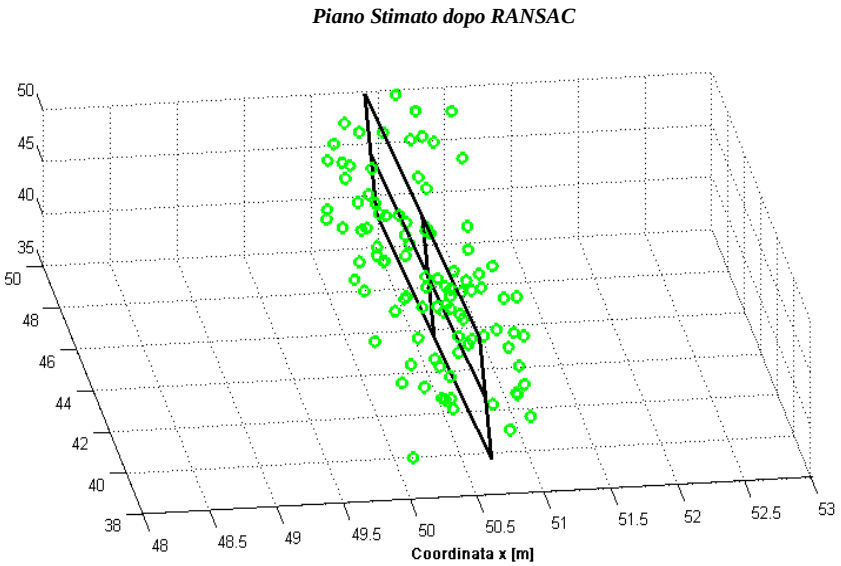


Figura 33: Piano Stimato dopo Ransac

CAPITOLO 5
RISULTATI SPERIMENTALI

Al fine di testare e quindi valutare il corretto comportamento dell'algoritmo realizzato, sono state effettuate una serie di simulazioni. Al termine di esse, è stato monitorato l'andamento di tutti i parametri d'interesse. In tale capitolo, fornirò i risultati ottenuti e gli andamenti dei parametri d'interesse.

5.1 Risultati Bundle Adjustment

Come ampiamente descritto nel capitolo 2, la risoluzione del problema del *BA* è avvenuta mediante l'implementazione di un algoritmo basato sul metodo di minimizzazione di L-M. Al fine di validare tale algoritmo rispetto allo stato dell'arte, i risultati ottenuti sono stati confrontati con quelli dell'algoritmo che ho indicato con la sigla SBA. Entrambe le implementazioni, permettono di minimizzare l'errore di riproiezione agendo sia sulla posa delle camere che sulla struttura 3D. Al fine di realizzare tale confronto, le due implementazioni sono state valutate su un problema di raffinazione di una ricostruzione 3D di una scacchiera, ottenuta partendo da immagini 2D scattate da tre camere(figura 1.2; figura 1.3; figura 1.4) che da ora in poi indicherò come: *camera1*, *camera2 e camera3*. In particolare ho eseguito 12 valutazioni, a ciascuna delle quali corrisponde una ricostruzione della scacchiera 3D ed un posizionamento delle camere differenti l'une dalle altre. Gli output presi in esame e confrontati tra loro sono:

- Errore di Riproiezione;
- Numero di Iterazioni;

- Angoli di Eulero e quindi rotazione su ciascun asse del sistema di riferimento(x,y,z) di ogni camera;
- Traslazione su ciascun asse del sistema di riferimento(x,y,z) di ogni camera;
- Ricostruzione Scacchiera 3D Post BA.

ERRORE DI RIPROIEZIONE FINALE:

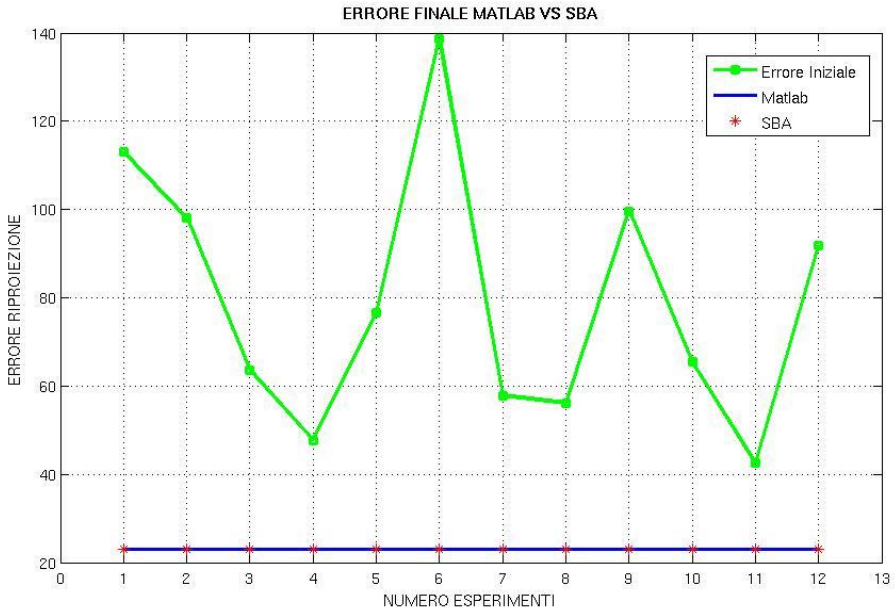


Figura 34: Errore Riproiezione Iniziale e Finale

Come si vede da tale grafico, indipendentemente dall'errore di riproiezione di partenza, entrambe le implementazione dei due BA convergono al medesimo valore pari a 22,98 [mm]. Accostando tale risultato al numero di iterazioni eseguite da ciascun BA per raggiungere il valore di convergenza ottenuto, si hanno, prima dell'operazione di *tuning*, i seguenti andamenti:

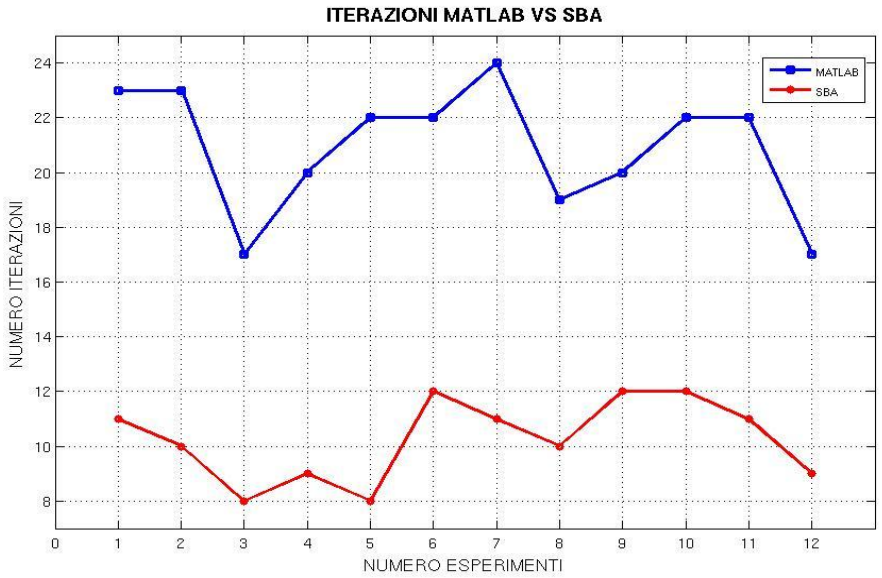


Figura 35: Iterazioni SBA Vs Iterazioni BA_ST Pre-Tuning

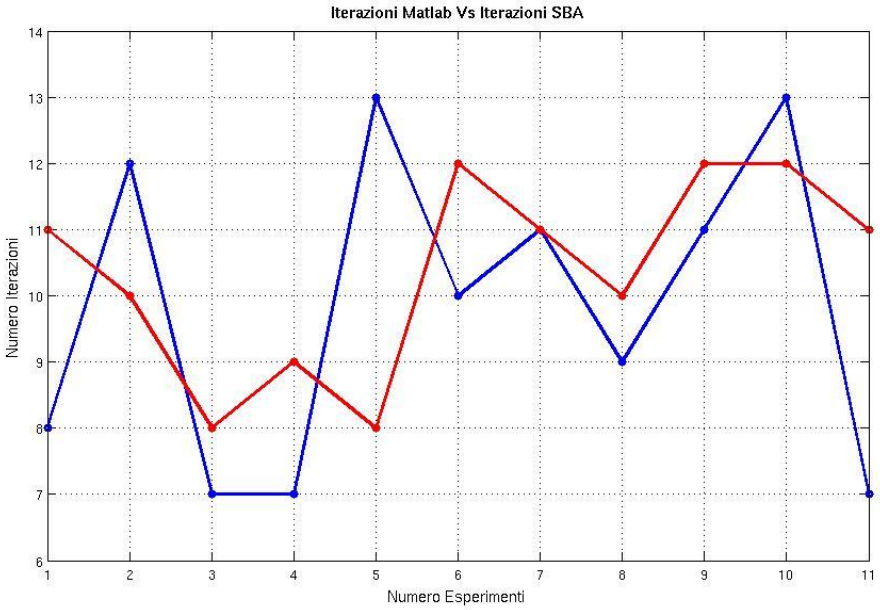


Figura 36:Iterazioni SBA Vs Iterazioni BA_ST Post-Tuning

Come si può vedere dal grafico sopra, a seguito di un'opportuna operazione di *tuning* il numero di iterazioni è lo stesso o differisce di poco a favore dell'una o dell'altra implementazione. Valutando il valor medio si ha che:

- *Numero_Iterazioni_Medio_BA_ST* = 9,81 iterazioni;

- *Numero_Iterazioni_Medio_SBA* = 10,36 iterazioni;

Dopo aver verificato questi due parametri, ho confrontato gli angoli di Eulero e le traslazioni che caratterizzano la posa di ciascuna camera restituiti dalle due implementazioni. Di seguito quanto ottenuto:

CAMERA 1:

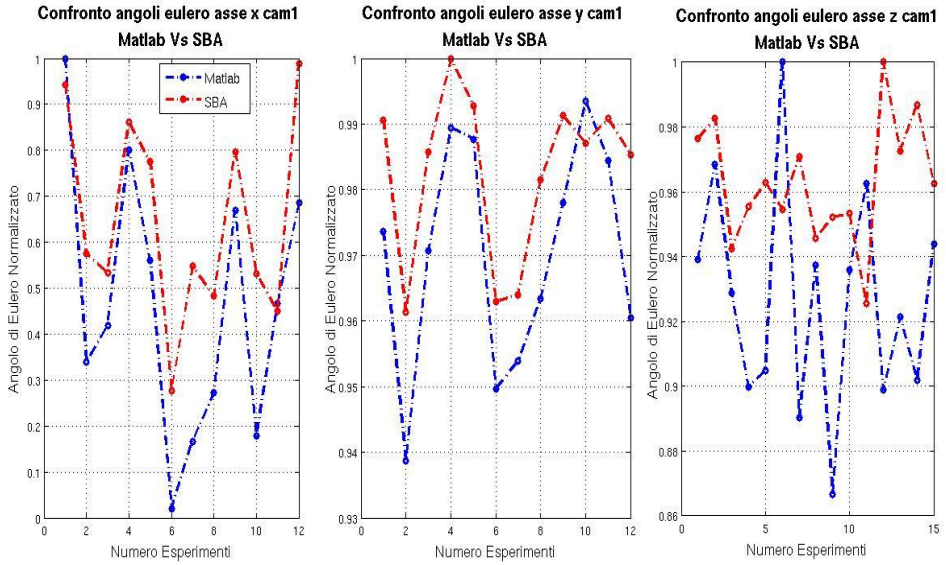


Figura 37: Confronto angoli di Eulero Camera 1

CAMERA 2:

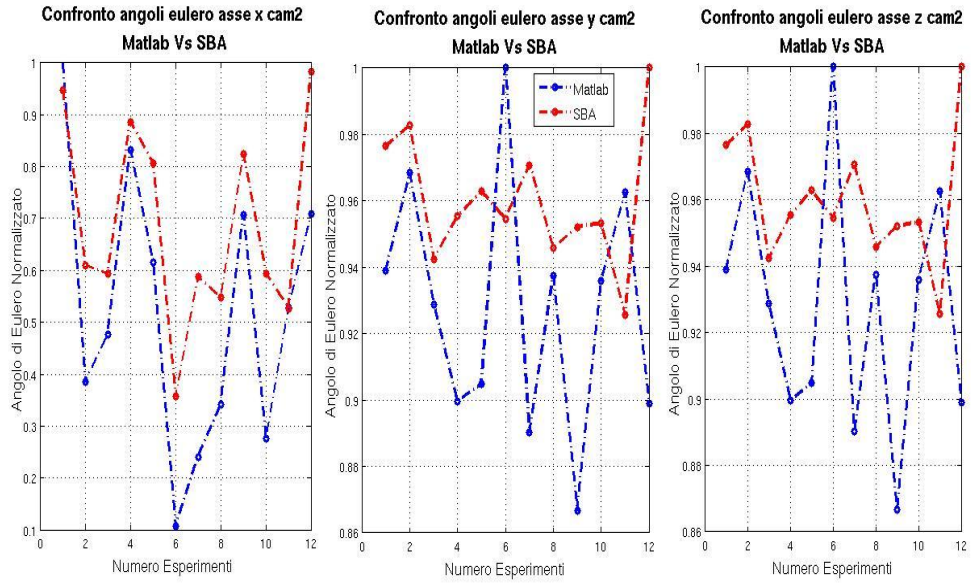


Figura 38: Confronto Angoli di Eulero Camera 2

CAMERA 3:

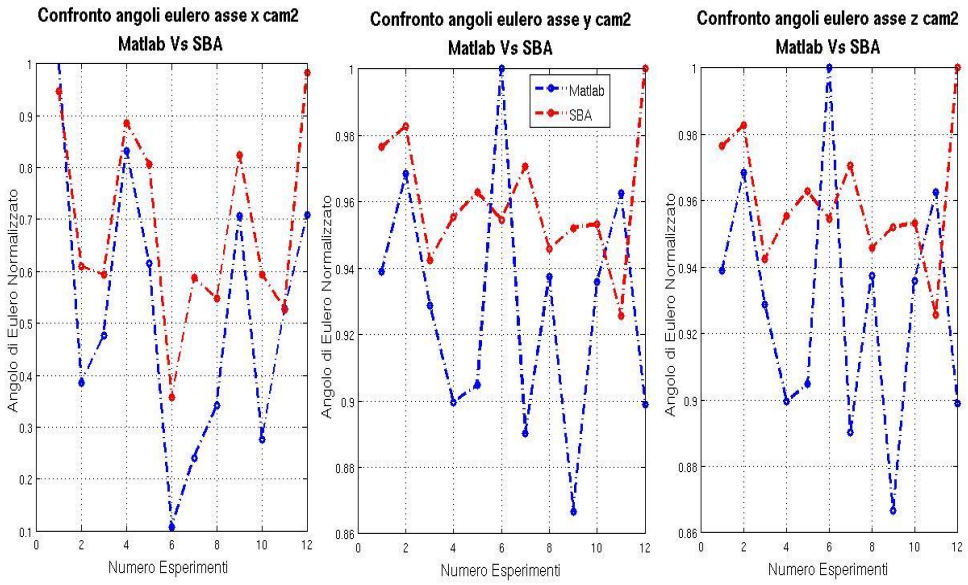


Figura 39: Confronto Angoli di Eulero Camera 3

A conferma di quanto mi aspettavo, tali grafici mostrano come le due implementazioni agiscano in maniera praticamente simile sugli angoli di Eulero delle tre camere a meno di piccolissime differenze. Ciò era abbastanza ovvio in quanto l'errore di riproiezione finale a cui giungono entrambi i BA è, come visto precedentemente, identico. Analogalmente per le Traslazioni abbiamo:

CAMERA 1:

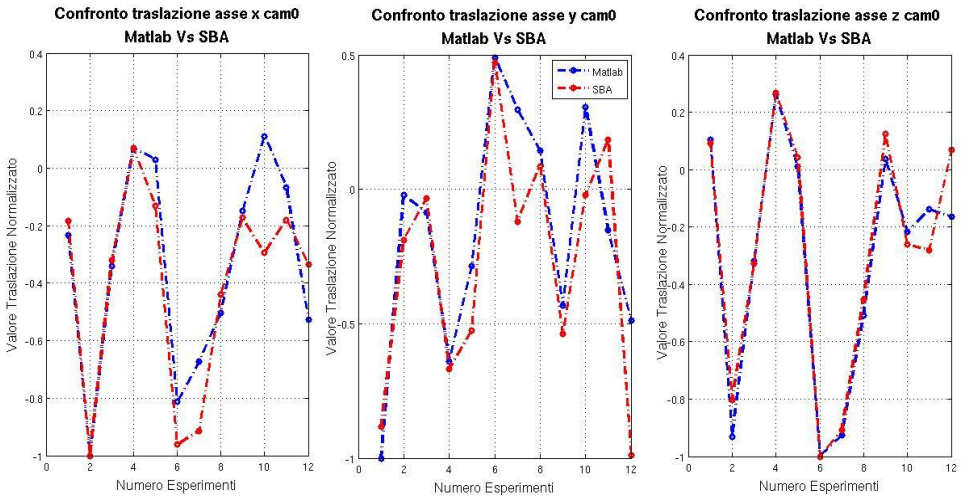


Figura 40: Confronto Traslazioni Camera 1

CAMERA 2:

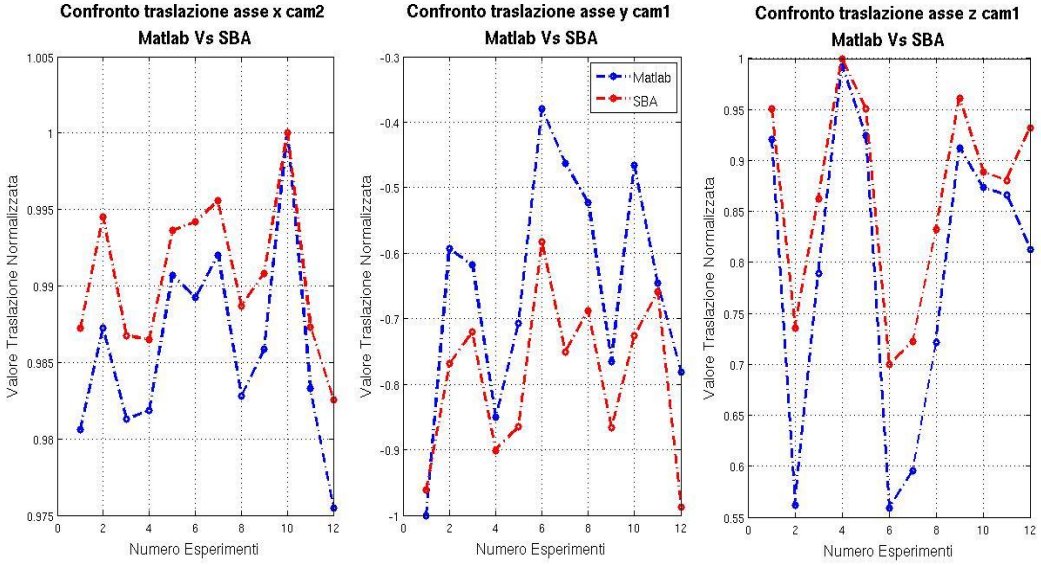


Figura 41: Confronto Traslazioni Camera 2

CAMERA 3:

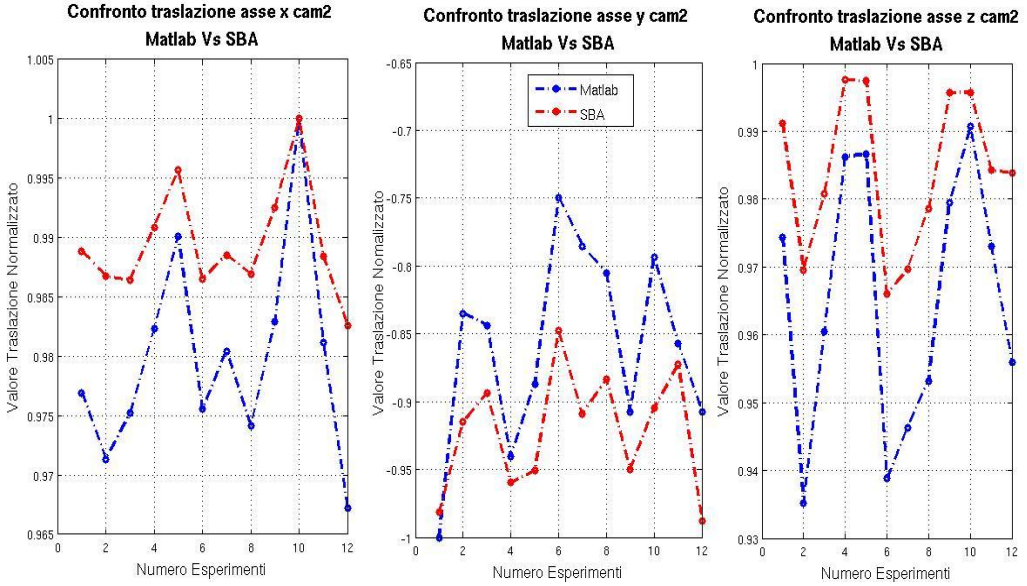


Figura 42: Confronto Traslazioni Camera 3

Così come per gli angoli di Eulero, anche per le traslazioni le due implementazioni si comportano, a meno di piccole differenze, nel medesimo modo. Infine riporto la ricostruzione 3D della scacchiera ottenuta a seguito dell'applicazione del BA_ST, facendo riferimento al sesto esperimento in cui avevamo il maggior errore di riproiezione(e=140; Mean(e)=79,23)

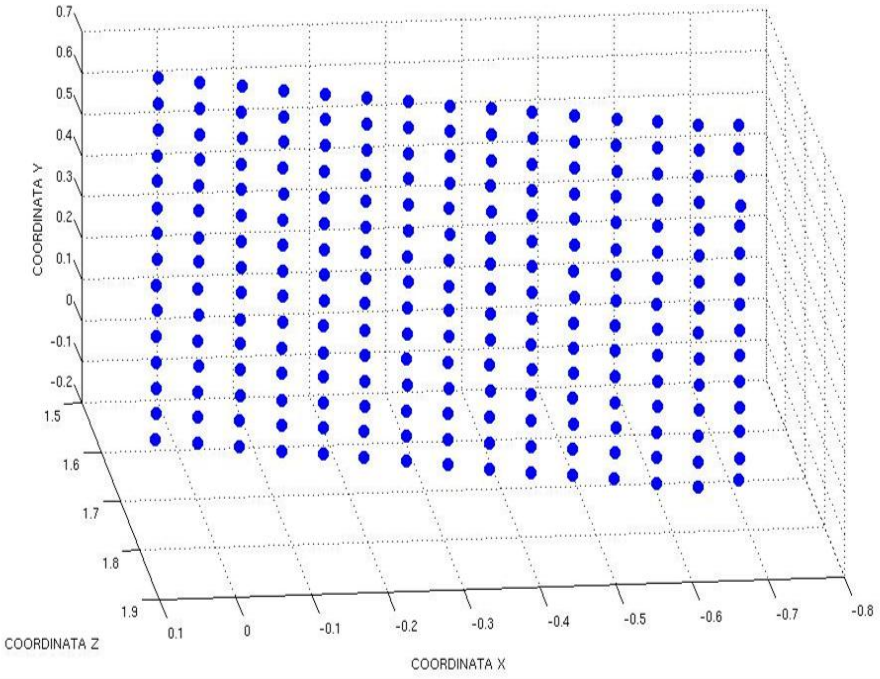


Figura 43: Ricostruzione post BA

5.2 Risultati Algoritmo per la Stima di Piani

Al fine di testare il corretto funzionamento dell'algoritmo di stima di piani da nuvole di punti 3D realizzato, esso è stato applicato prima ad un insieme di dati sintetici e poi ad un insieme di dati reali. In particolare oltre all'errore di stima ho

cercato di valutare in che rapporto i vari parametri di soglia ed i passi di campionamento stanno tra loro.

5.2.1 Applicazione a Dati Sintetici

Al fine di ricreare delle scene 3D sintetiche, è stata utilizzata la funzione rand messa a disposizione da MATLAB. Tale funzione permette di generali valori casuali distribuiti uniformemente in un dato intervallo scelto dall'utilizzatore. In tal modo è stato possibile ricreare un'unica scena composta da più oggetti. I primi esperimenti sono stati effettuati al fine di capire in che misura i parametri utilizzati dall'algoritmo sono tra loro legati. A tal proposito sono state generate due distribuzione uniforme di 100 punti ciascuna ma nella prima $x \in [49{,}50]$, $y \in [1, 50]$ e $z \in [1, 50]$, mentre nella seconda $x \in [48{,}50]$ ed y e z restano immutati

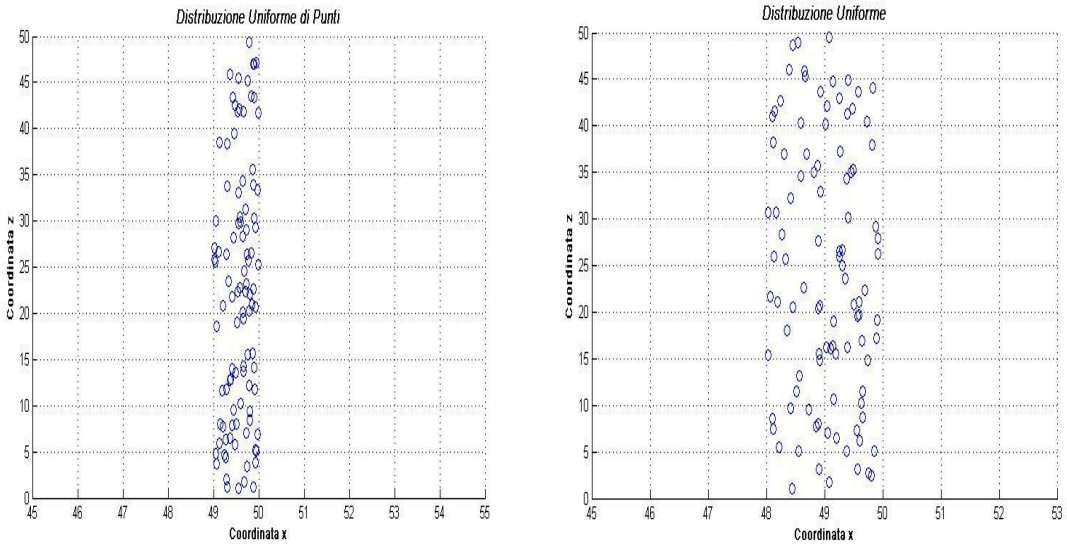


Figura 44: Distribuzioni Uniformi

La prima relazione che ho voluto ottenere è quella tra la scelta del passo di campionamento g del coefficiente del piano d e la Soglia D. L'obbiettivo è ovviamente impostare questi due valori in modo tale che nello spazzolamento

della scena 3D vi sia un piano a cui siano associati tutti i 100 punti. A seguito di svariati esperimenti ho trovato che per individuare un solo piano a cui sono associati tutti i punti è necessario che:

$$g = D = \Delta/2$$

Ove Δ è la larghezza dell'intervallo di appartenenza della *x*. Utilizzare valori di g e d tra loro differenti comporta da luogo o, alla individuazione di più piani per il medesimo insieme di punti(caso in cui $D > g$) o, all'individuazione di un solo piano a cui non sono associati tutti i punti dell'insieme (caso in cui $g > D$). La seconda relazione che ho cercato di ricavare, è quella tra la soglia *D*, ed i valori degli autovalori associati a ciascuna PCs. Se, infatti, la larghezza dell'intervallo di possibili valori della coordinata x (può essere anche la coordinata z), è troppo grande, ponendo $D = \Delta/2$, la varianza della componente associata alla coordinata x aumenta e di conseguenza aumenta anche il valore dell'autovalore associato a tale PC. In particolare si ha che:

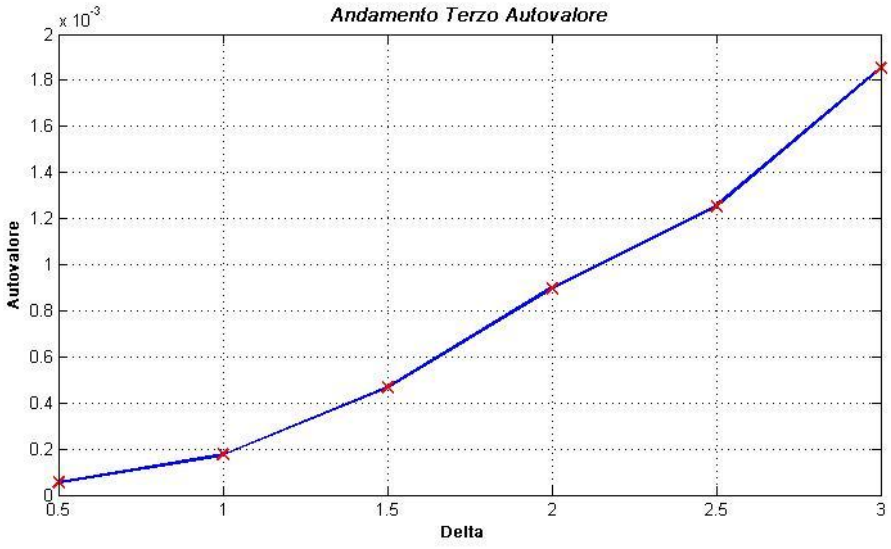


Figura 45: Andamento Terzo Autovalore

Come si può vedere esso è praticamente nullo per $\Delta = 0.5$ e quindi per $D = 0.25$.

La conseguenza di ciò è che il rapporto tra tale autovalore ed quello legato alla seconda PC ha il seguente andamento:

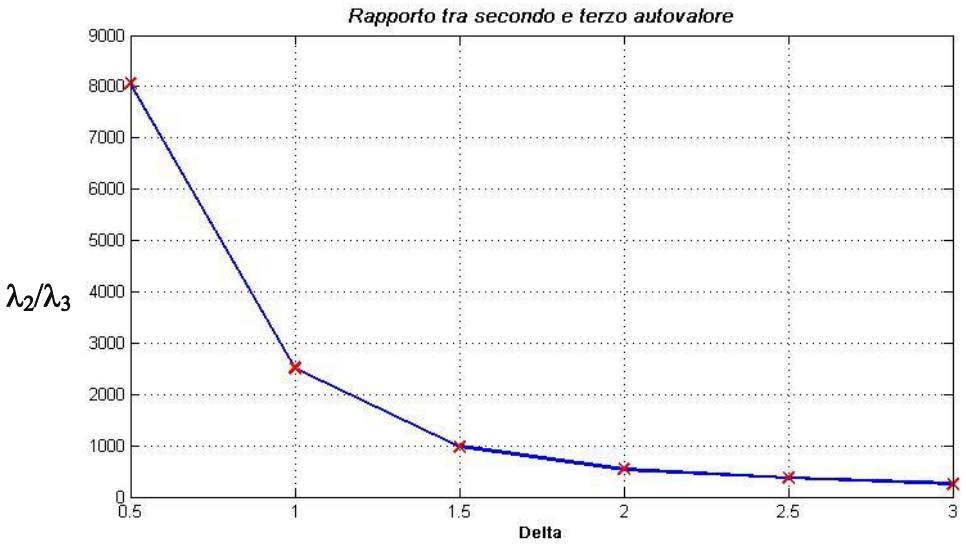


Figura 46: Rapporto tra Secondo e Terzo Autovalore

È evidente come quindi il parametro di soglia utilizzato per valutare se la stima del piano è corretta o meno non può essere fisso, ma dovrà variare a seconda del Δ quindi della *D*. Se così non fosse si eseguirebbe RANSAC anche quando non ve ne è bisogno. Trovate le seguenti relazioni, il passo successivo è è stato quello di verificare se in condizioni ottimali, ovvero in cui ciascuna coordinata dei punti 3D associata ad un oggetto della scena è compreso in un intervallo di misura estremamente piccolo ($x_{max} - x_{min} < 0.5\ m$), l'algoritmo riuscisse ad stimare correttamente un piano per ciascuno degli oggetti presenti nella nuvola di punti di partenza. Tale condizione si verifica a seguito di una ricostruzione dell'ambiente circostante la camera molto accurata. Prendendo come esempio la facciata di un palazzo parallelo alla direzione di spostamento dell'autovettura, infatti, i punti 3D ad essa associata dovrebbero avere tutti la il medesimo valore per la coordinata x. Poiché la ricostruzione è affetta da un certo errore di riproiezione, ciò che accade è che i valori di tale coordinata variano all'interno di un intervallo che è tanto più piccolo quanto più l'errore di riproiezione è piccolo.

Per valutare tale caso, sono stati realizzati 4 cluster sintetici da 40 punti ciascuno:

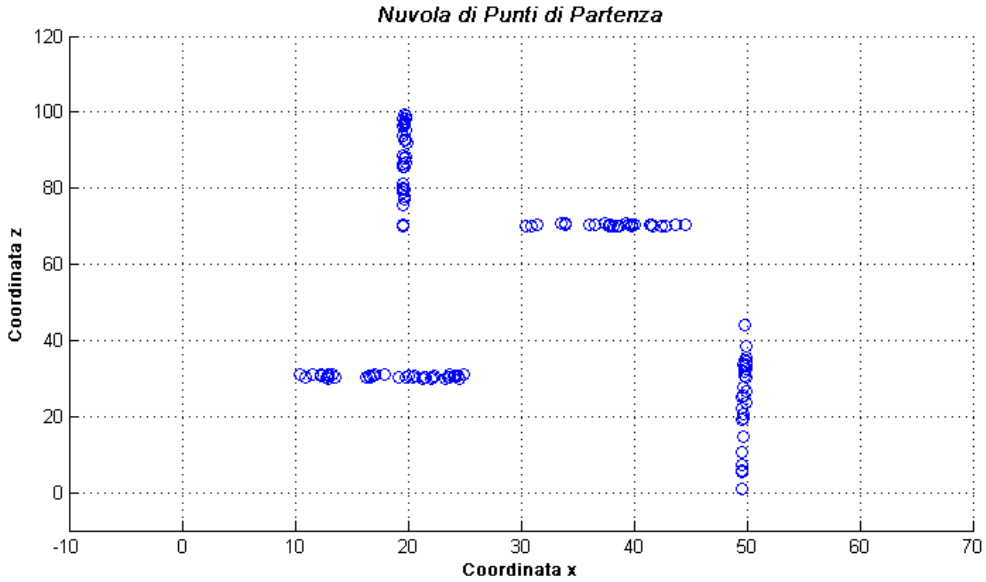


Figura 47: Nuvola di Punti 3D di Partenza

Il risultato ottenuto al termine dell'esecuzione dell'algoritmo è stato:

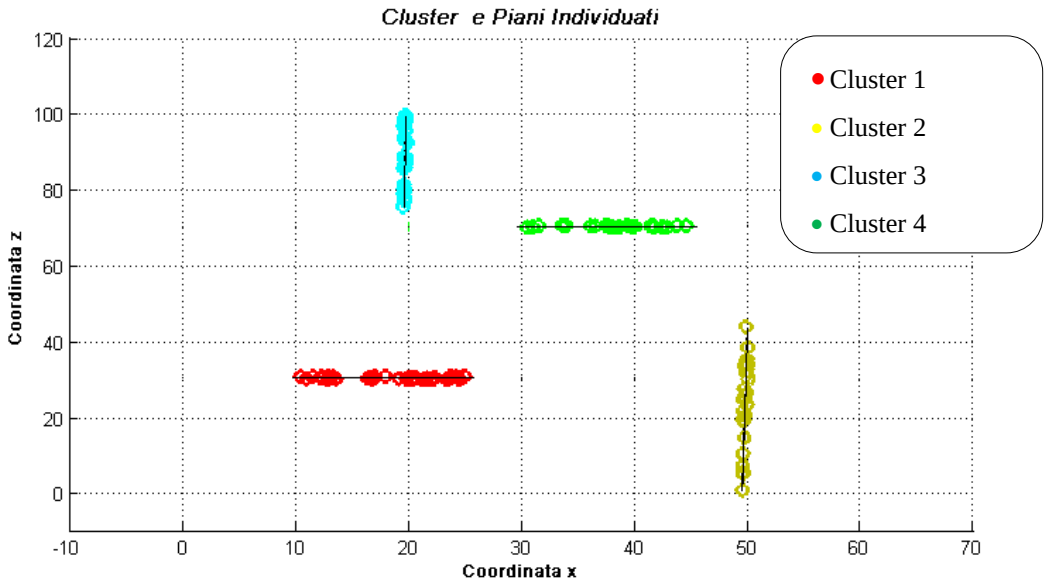


Figura 48: Cluster e Piani Individuati

L'errore associato a ciascuno di tali piani è:

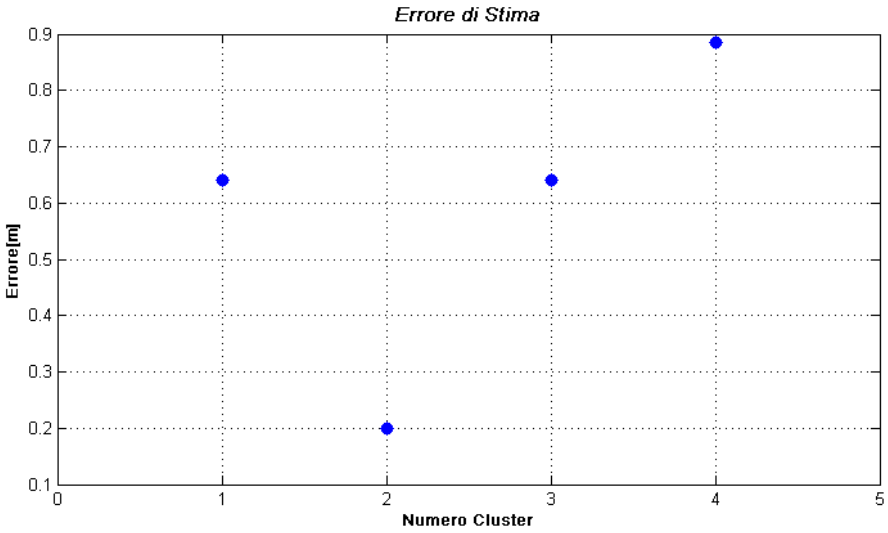


Figura 49: Errore di Stima

I risultati sopra descritti, sono stati ottenuti imponendo i seguenti valori per le soglie utilizzate:

- $D = 0.25$ distanza massima del punto dal piano stimato;
- $k = 0.1$ passo campionamento associato al coefficiente *a*;
- $f = 0.1$ passo campionamento associato al coefficiente *c*;
- $g = 0.25$ passo campionamento associato al coefficiente *d*;
- $N\_Min =$ Numero minimo di punti appartenenti al cluster;
- $\lambda_2/\lambda_3 > 1000$
- $X_{max} - X_{min} < 0.5\ [m]$ per i Cluster 2 e 3
- $Z_{max} - Z_{min} < 0.5\ [m]$ per i Cluster 1 e 4

Una volta verificato il comportamento in tali condizioni, ho provato a valutare la medesima situazione allargando l'intervallo dei possibili valori di ciascuna coordinata, in modo tale da simulare il caso di una ricostruzione affetta da un errore di riproiezione maggiore.

In particolare ho imposto che:

- $X_{max} - X_{min} < 2.5\ [m]$ per i Cluster 2 e 3
- $Z_{max} - Z_{min} < 2.5\ [m]$ per i Cluster 1 e 4

Variando i parametri sulla base delle relazioni precedentemente ricavate, ovvero:ti

- $g = \Delta / 2 = 1{,}25$;
- $D = g = 1{,}25$;
- $\lambda_2/\lambda_3 > 100$

ho ottenuto i seguenti risultati:

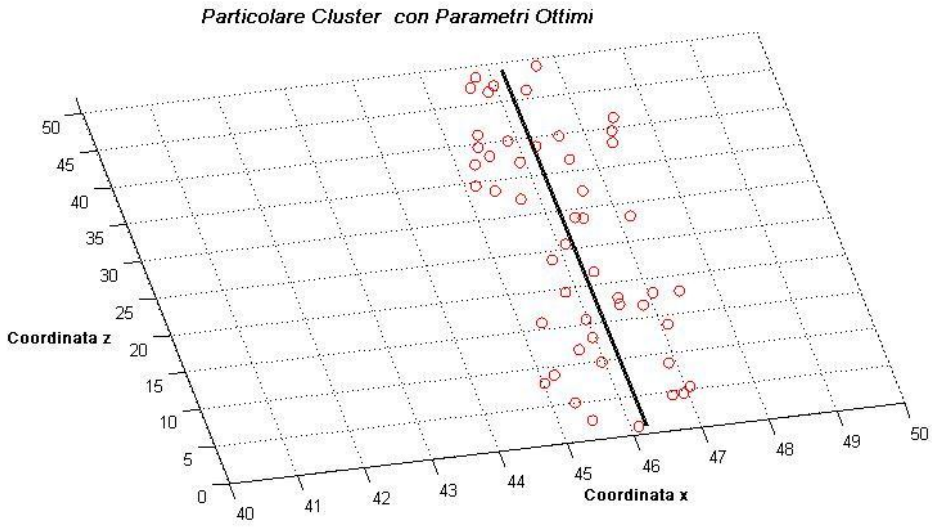


Figura 50: Particolare Cluster con Parametri Ottimi e Variazioni < 2.5 m

5.2.2 Risultati Sperimentali

Oltre che ai dati sintetici precedentemente descritti, l'algoritmo realizzato è stato applicato anche a ricostruzioni di ambienti urbani reali. A tal proposito, sono state utilizzate due sequenze di immagini fornite da ST Microelectronics. In tale fase, è stato valutato l'errore di stima e cercato di impostare in maniera pseudo ottima i parametri dell'algoritmo. A differenza dei dati sintetici, in una sequenza reale

l'escursione massima dell'intervallo dei possibili valori assunti da ogni coordinata non è nota a priori. In tale contesto è fondamentale che la ricostruzione restituita dalla tecnica *SfM* sia la più accurata possibile, ovvero che l'errore di riproiezione sia molto piccolo.

La prima sequenza considerata è ottenuta facendo semplicemente ruotare la camera ma senza effettuare alcuna traslazione. Riporto di seguito le immagini 1, 5, 10 e 15 della sequenza:

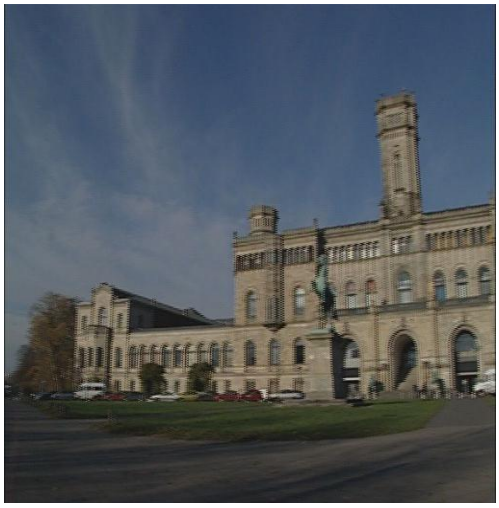


Figura 52: Immagine sequenza n.1

Figura 51a: Immagine sequenza n.5

Figura 54: Immagine sequenza n.10

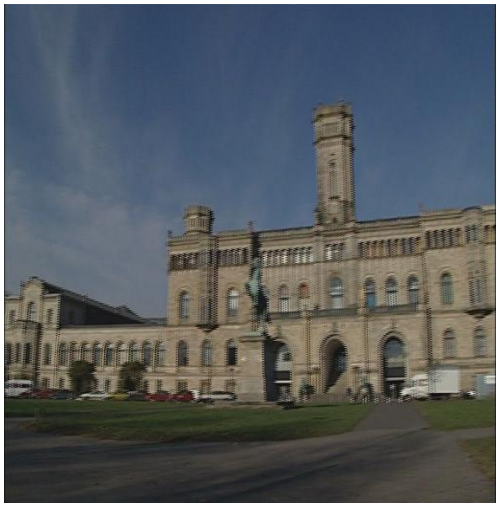


Figura 53b: Immagine sequenza n.15

La ricostruzione riferita all'immagine 10 di tale sequenza restituita dalla tecnica *SfM*, è costituita da 715 punti ed è:

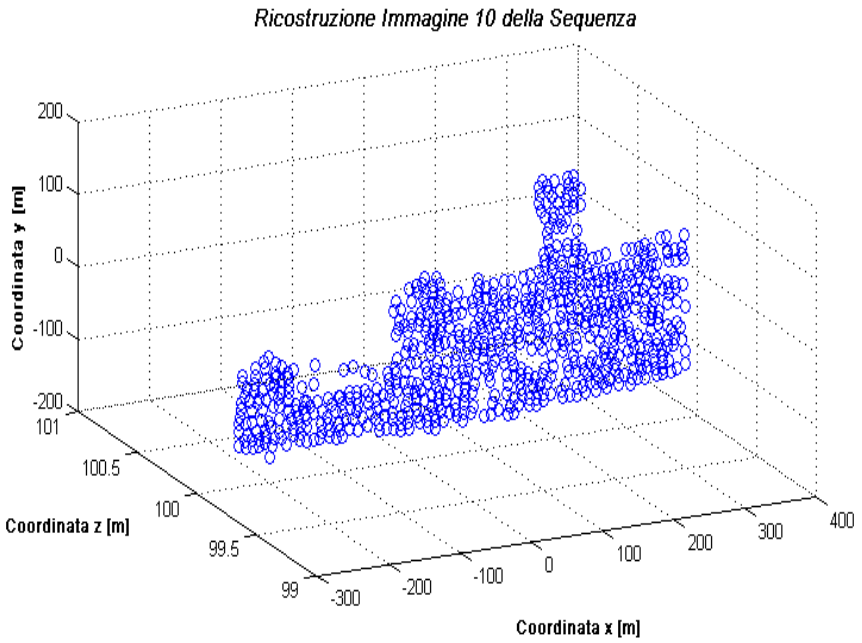


Figura 55: Ricostruzione Immagine 10

In tal caso, come si può vedere, la ricostruzione è estremamente accurata. Le variazioni lungo l'asse z, sono addirittura inferiori a 0.15 m. Imponendo i seguenti valori ai parametri dell'algoritmo di stima realizzato:

- $g = (Z_{max} - Z_{min}) / 2 = 0.06$;
- $N\_min = 700$;
- $D = g = 0.06$;
- $k = 0.1$ passo campionamento associato al coefficiente *a*;
- $f = 0.1$ passo campionamento associato al coefficiente *c*;
- $\lambda_2/\lambda_3 > 1000$

Il risultato ottenuto è stato:

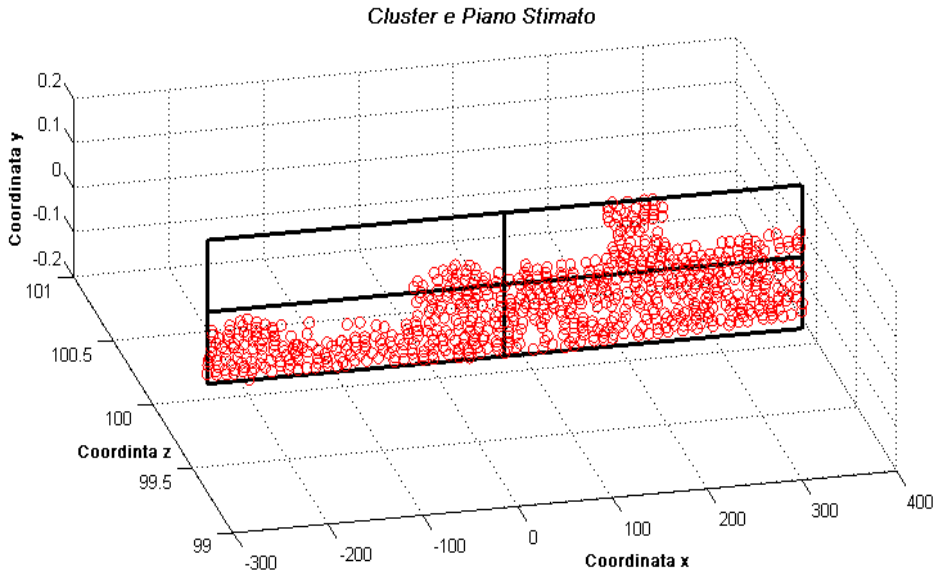


Figura 56: Cluster e Piano Stimato

La seconda sequenza presa in esame, invece, è ottenuta scattando delle foto da un'autovettura in movimento. Al fine di comprendere il movimento seguito dalla vettura, riporto di seguito le immagini 1, 5, 10, 15:

Figura 58: Immagine sequenza n.1

Figura 57a: Immagine sequenza n.5

Figura 60a: Immagine sequenza n.10

Figura 59b: Immagine sequenza n.15

Prendendo come riferimento ancora l'immagine 5, si ha la seguente ricostruzione a seguito della tecnica *SfM*:

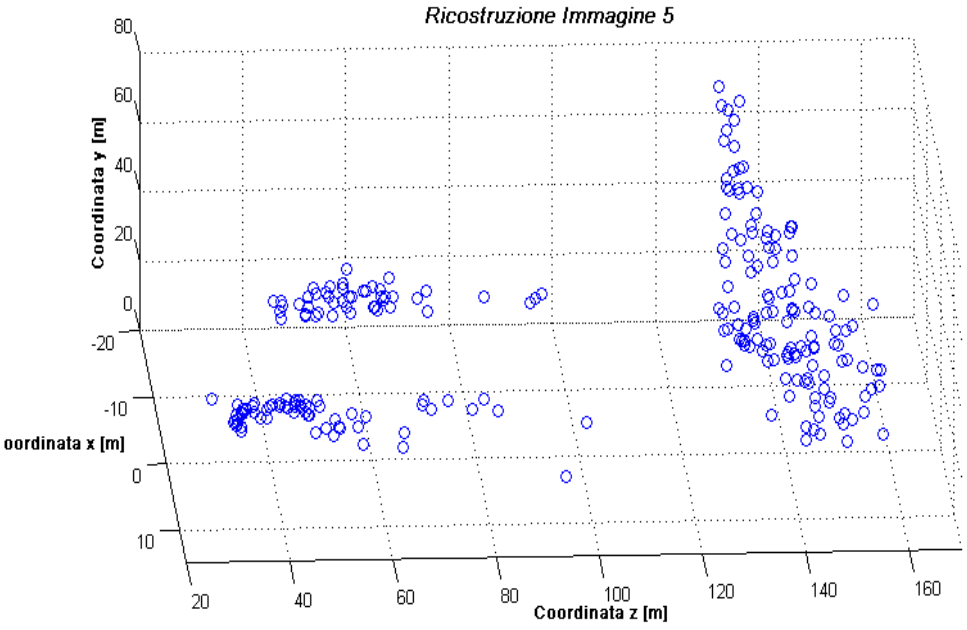


Figura 61:Ricostruzione Immagine 5 vista 1

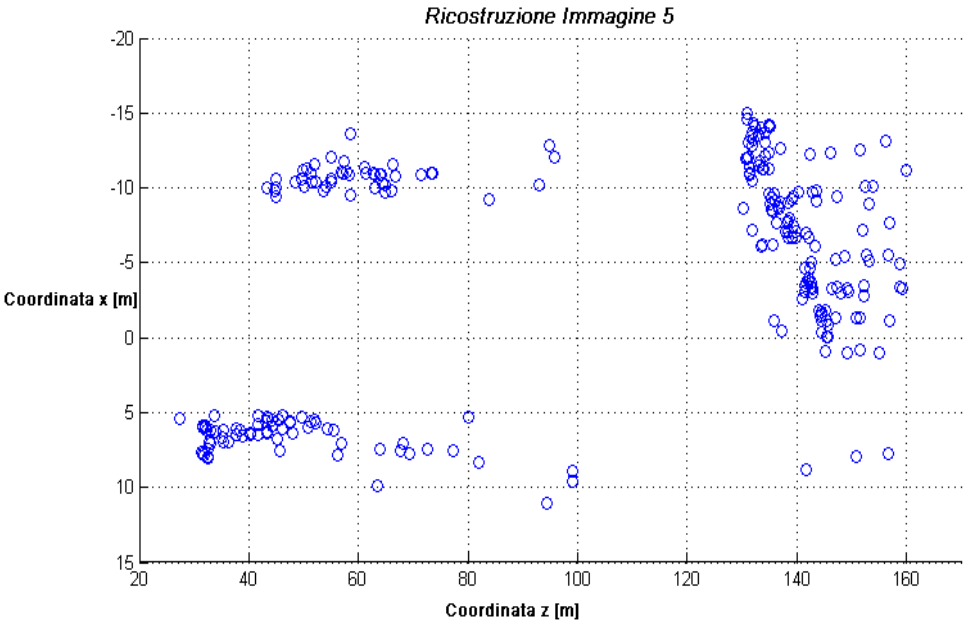


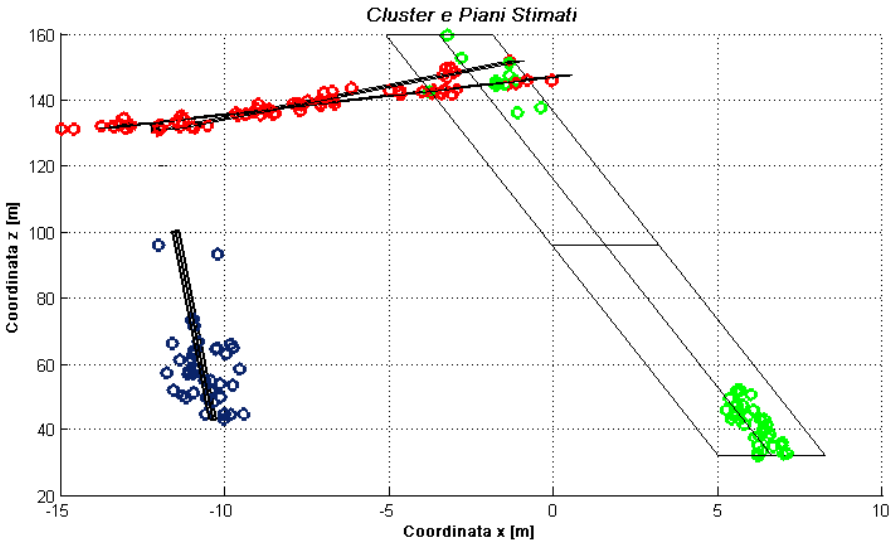
Figura 62: Ricostruzione Immagine 5 vista 2

Come si può vedere, in tal caso la ricostruzione è decisamente meno accurata di quella dell'esempio precedente. Ciò è spiegabile dal fatto che la scena in esame è molto più complessa in quanto oltre al movimento della camera, che non è più solo rotatorio, sono presenti altri oggetti che si muovono di moto proprio. Dalle immagini sopra si può notare come le variazioni lungo l'asse x sono all'incirca pari a 2.5 m, mentre quelli lungo l'asse z sono all'incirca pari a 5 m. Inoltre vi è la presenza di alcuni punti che sono sicuramente da ritenersi outliers. Al fine di ottenere delle stime di piani corrette ho scelto di differenziare il valore di campionamento del coefficiente *d* a seconda che i piani spazzolati fossero paralleli o meno all'asse x. In tal caso, inoltre, essendo presenti più oggetti ai quali il numero di punti associati è molto diverso, non è stato possibile imporre la condizione $N\_min \cong N\_tot$. Al fine dunque di ottenere la stima dei piani desiderata sono stati impostanti i seguenti valori per i soliti parametri:

- $g = 2$ se $a=0$;
- $g=1{,}6$ se $a \neq 0$;

- $N\_min = 80$;
- $D = 1,2$;
- $k = 0.1$ passo campionamento associato al coefficiente a;
- $f = 0.1$ passo campionamento associato al coefficiente c;
- $\lambda_2/\lambda_3 > 100$

Il risultato ottenuto è stato:



Come si può vedere la stima non è stata accuratissima. In particolare il piano associato al cluster verde presenta un errore pari a 10m circa, mentre i punti del cluster rosso sono stati associati a più piani. Questi errori sono chiaramente dovuti al fatto che è stato necessario utilizzare dei parametri differenti da quelli ottimi. Notiamo, però, come l'algoritmo abbia comunque eliminato buona parte degli outliers.

CONCLUSIONI

Il lavoro di tesi da me portato avanti nel corso dell'ultimo anno, ha avuto come scopo quello di realizzare un algoritmo che permettesse di ottenere una stima robusta di piani da nuvole di punti 3D per la navigazione assistita. Nell'ambito del Progetto Europeo *Astute*, l'algoritmo realizzato può essere utilizzato, come in parte mostrato in tale elaborato, al fine di individuare gli oggetti presenti nell'ambiente attorno ad un'autovettura e quindi sfruttare tale informazione per realizzare un sistema di frenata automatica per casi d'emergenza. Da qui la notazione *"per la navigazione assistita"* utilizzata nel titolo stesso della tesi. Una volta implementato l'algoritmo, esso è stato testato sia su dati sintetici che su dati sperimentali. In particolare i primi sono stati utilizzati inizialmente per verificare il funzionamento dell'algoritmo e poi per trovare in che misura i vari parametri di soglia utilizzati sono tra loro legati. Dai risultati ottenuti, i quali sono stati mostrati nel precedente capitolo, possiamo senz'altro confermare che l'algoritmo funzioni correttamente e permetta, mediante stima robusta di piani, di individuare correttamente gli oggetti presenti nella scena almeno fintantoché l'intervallo di variazione dei possibili valori della coordinata x o della coordinata z, rispettivamente se l'oggetto si trovi parallelo o perpendicolare alla direzione di spostamento della vettura, non siano maggiori di 5 metri. Tale valore è da considerarsi più che accettabile in quanto parecchio sovradimensionato rispetto ai valori reali. Al fine di ottenere questi risultati, sono state trovate le seguenti relazioni ottime tra i parametri:

- $-1 \leq a \leq 1$;
- $-\sqrt{(1-a^2)} \leq c \leq \sqrt{(1-a^2)}$
- $0 \leq g \leq X_{max}$ nei casi in cui $a \neq 0$ e $0 \leq j \leq Z_{max}$ nei casi in cui $a = 0$ per il coefficiente *d*
- $k = 0.1$ passo campionamento associato al coefficiente *a*;
- $f = 0.1$ passo campionamento associato al coefficiente *c*;

- $g = \Delta / 2$;
- $N\_min \cong N\_tot$
- $D = g$;
- $\lambda_2/\lambda_3 > 1000$ se D < 0,75;
- $\lambda_2/\lambda_3 > 500$ se $0{,}75 \leq D \leq 1$;
- $\lambda_2/\lambda_3 > 100$ se D > 1

Una volta trovate le relazioni che legano tra loro i parametri utilizzati, l'algoritmo è stato eseguito su due ricostruzioni sperimentali, una estremamente semplice e di conseguenza affetta da un errore di riproiezione quasi nullo, ed un'altra invece molto complessa che presentava un errore di riproiezione elevato. Nel primo caso, l'algoritmo non ha avuto alcun problema nell'individuare i punti associati allo stesso oggetti identificato dal piano stimato; nel secondo caso, invece, l'algoritmo riesce ancora ad individuare gli oggetti presenti nella scena ma alcuni punti ad esso associabili non vengono presi ed inoltre si individuano più piani per lo stesso oggetto. Se ne conclude, quindi, che al fine di ottenere un risultato corretto, è necessario che la ricostruzione dell'ambiente attorno la camere sia la più accurata possibile. È per tale motivo, che è stato realizzato un algoritmo basato sul metodo di Levenberg_Marquadt che permette di risolvere il problema del Bundle Adjustment.

SVILUPPI FUTURI

Un possibile sviluppo futuro di tale lavoro di tesi, consiste nel proseguire il lavoro da me svolto al fine di realizzare il sistema di frenata automatico per le situazioni d'emergenza. In tal caso, infatti, sarebbe necessario sicuramente diminuire la complessità computazionale dell'algoritmo, magari mediante opportune operazioni di parallelizzazione del codice, ed inoltre dovrebbe essere

implementata una nuova parte che, sfruttando le informazioni sulla posa della camera restituite dalla tecnica *SfM*, permetta di valutare la distanza dell'autovettura. Correlando tale informazione con quella della velocità di viaggio dell'auto è possibile valutare lo spazio di frenata necessario per evitare l'urto e quindi attuare una frenata automatica se tale spazio è inferiore ad una certa soglia.

Bibliografia

[1] *astute-project*. (n.d.). Retrieved from http://www.astute-project.eu

[2] *Autoblog*. (n.d.). Retrieved from http://www.autoblog.com/2012/06/15/euro-ncap-will-soon-require-auto-braking-for-five-star-safety-ra/

[3] Bill Triggs Philip McLauchlan, R. H. (n.d.). Bundle Adjustment - A Modern Synthesis.

[4] Deans, S. (n.d.). Radon and Abel Transforms. In S. Deans, *The Transforms and Applications Handbook.*

[5] Feinstein, A. R. (1996). Multivariable Analysis. *Yale University Press* . New Haven.

[6] Fishler, M. A., & Bolles, R. C. (March 1980). *Random Sample Consensus: a Paradigm for Model Fitting with Application to Image Analysis and Automated Cartography.*

[7] Hartley, A., & Zisserman, R. (2003). Multiple View Geometry in computer vision. Cambridge University Press.

[8] Hartley, A., & Zisserman, R. (2003). Multiple View Geometry in computer vision. Cambridge University Press.

[9] Jackson, J. E. (n.d.). *A User's Guide to Principal Components* .

[10] JeanPierre, A. G. (17th – 20th August, 1993). The Physical Heritage of Sir W.R. Hamilton. *The Mathematical Heritage of.* Dublin.

[11] Jolliffe, I. (n.d.). *Principal Component Analysis.*

[12] Lourakis, M. I., & Argyros, A. A. (March 2009). SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Transactions on Mathematical Software*.

[13] *mathworks*. (n.d.). Retrieved from www.mathworks.it/it/help/stats/princomp.html

[14] *Matlab*. (n.d.). Retrieved from http://www.mathworks.it/products/matlab

[15] Spagnolini, U. (2010). Teoria della Stima. In U.Spagnolini, *Elaborazione Statistica dei Segnali* (pp. 67-87).

[16] Spagnolini, U. (2010). Teoria della Stima. In U.Spagnolini, *Elaborazione Statistica dei Segnali* (p. 23).

[17] Vapnyarskii, I. (n.d.). Lagrange multipliers. In *Encyclopaedia of Mathematics.*

[18] W.Fornaciari, & Brandolese, C. (n.d.). *Sistemi embedded. Sviluppo hardware e software per sistemi dedicati.*