Politecnico Di Milano
Dipartimento di Elettronica e Informazione
Facoltà di Ingengeria Informatica
Tesi di Laurea Specialistica

SLAM con camera omnidirezionale all'interno del framework MoonSlam

Bacciocchi Roberto
732761

Relatore: Prof. Matteucci Matteo
Correlatore: Ing. Ceriani Simone

Anno Accademico 2011-2012

Ringraziamenti

Ringrazio essenzialmente i miei genitori per avermi assistito tanto economicamente quanto moralmente durante gli innumerevoli anni di studio. Ringrazio inoltre il mio relatore e il mio correlatore per la competenza e l'entusiasmo nonché la pazienza e la disponibilità mostratami durante lo sviluppo del presente lavoro di tesi. Ringrazio infine la mia amica Anna, che mi ha aiutato nella correzione dell'*abstract*.

Sommario

La capacità di orientamento di un robot è una caratteristica di fondamentale importanza affinché esso possa svolgere i propri compiti in totale autonomia. Nel caso in cui l'ambiente sia sconosciuto, lo SLAM (*Simultaneous Localization and Mapping*), attraverso l'elaborazione delle misure provenienti dai sensori del robot durante la fase di esplorazione, permette di stimare probabilisticamente la traiettoria percorsa dal robot, congiuntamente alla costruzione di una mappa dell'ambiente.

L'utilizzo di una camera omnidirezionale come unico sensore (*Monocular SLAM*), grazie a un campo visivo di 360°, permette di osservare le caratteristiche rilevanti dell'ambiente molto più a lungo di una camera prospettica, ottenendo una mappa e una traiettoria del robot di qualità superiore. Le immagini ottenute con una camera omnidirezionale sono però affette da forte distorsione e questo rende particolarmente problematico il *tracking* dei punti caratteristici, ossia l'acquisizione delle misure.

Nel presente lavoro di tesi sono state sviluppate due nuove tecniche di *tracking* per camere omnidirezionali: la prima trasforma le immagini in panoramiche e su queste applica un metodo di *patch tracking* classico per camere prospettiche; la seconda effettua invece una trasformazione prospettica delle aree nell'intorno dei punti caratteristici, eliminando ogni fattore di distorsione. È stato inoltre applicato al caso omnidirezionale il metodo del *patch warping* omografico, già utilizzato con successo nel caso prospettico. Per quanto riguarda gli aspetti relativi allo SLAM è stato invece introdotto un metodo che inizializza la posizione dei punti caratteristici del mondo sul piano del terreno sul quale il robot si muove, in alternativa all'inizializzazione classica a distanza costante. Tutti i metodi sviluppati sono stati implementati all'interno del *framework MoonSlam*[*], un congiunto di librerie per la realizzazione di applicazioni *Visual SLAM* basate sul filtro di Kalman esteso, realizzato da Simone Ceriani all'interno del Laboratorio di Intelligenza Artificiale e Robotica (AIRLab) del Politecnico di Milano. I risultati ottenuti dimostrano le potenzialità della camera omnidirezionale quando applicata allo SLAM, nonché la validità dei metodi sviluppati.

Parole chiave: Monocular SLAM, EKF, patch tracking, camera omnidirezionale.

[*] http://airlab.elet.polimi.it/index.php/MoonSlam

Abstract

The orientation ability of a robot is a fundamental characteristic that allows the robot to perform its tasks in complete independence. If the environment is unknown, the SLAM (*Simultaneous Localization and Mapping*), by processing the measures of the robot sensors during the exploration phase, allows for the estimate of the trajectory followed by the robot, along with the construction of a map of the environment.

The use of an omnidirectional camera as a unique sensor (*Monocular SLAM*), thanks to a 360° visual field, allows for the observation of the relevant features of the environment for a longer time compared to a perspective camera, thus obtaining a map and a robot trajectory of higher quality. On the other hand, images obtained from an omnidirectional camera have a strong distortion, which makes it difficult to track the characteristic points, therefore the acquisition of the measures.

In this dissertation, two new tracking techniques for omnidirectional cameras are developed: the former transforms the omnidirectional images into panoramic ones, applying to them a classic patch tracking method for perspective cameras; the latter operates a perspective transformation of the areas around tracked features, removing any distortion factor. The homographic *patch warping* method, already used successfully with perspective cameras, is also applied to omnidirectional ones. As for the SLAM, a method to initialize the position of the characteristic points of the world on the ground plane on which the robot moves has been introduced, as an alternative to the classical initialization at a constant distance. All developed methods have been implemented within the *MoonSlam* framework*, a set of libraries for the development of *Visual SLAM* applications based on the extended Kalman filter, developed by Simone Ceriani at the *Laboratorio di Intelligenza Artificiale e Robotica (AIRLab)* of *Politecnico di Milano*. Results achieved by this work demonstrate the potentialities of an omnidirectional camera when applied to SLAM, as well as the validity of the methods developed.

Keywords: Monocular SLAM, EKF, patch tracking, omnidirectional camera.

*http://airlab.elet.polimi.it/index.php/MoonSlam

Indice

Dizionario

Punto caratteristico	Punto dell'immagine che si differenzia nettamente dai punti che lo circondano.
Descrittore	Informazione associata a un punto caratteristico che ne permette l'identificazione univoca. Generalmente basato sull'informazione contenuta nell'area dell'immagine circostante il punto.
Tracking	Inseguimento di punti caratteristici o regioni in una sequenza di immagini.
Patch	Regione dell'immagine nell'intorno di un punto caratteristico, usata come descrittore del punto.
Matching	Operazione che misura la distanza fra due descrittori per verificare se i relativi punti caratteristici siano in realtà lo stesso.
SIFT	*Scale-Invariant Feature Transform*, algoritmo per la rilevazione di punti caratteristici e la loro descrizione basata su istogrammi del gradiente della regione dell'immagine intorno ad essi.
RANSAC	*RANdom SAmple Consensus*, metodo per l'eliminazione degli *outlier* da un insieme di misurazioni contenente errori di classificazione
EKF	*Extended Kalman Filter*, filtro che permette di stimare probabilisticamente lo stato di un sistema dinamico non lineare.
SLAM	Simultanea localizzazione e costruzione della mappa (*Simultaneous Localization And Mapping*), metodo utilizzato da un robot per creare la mappa dell'ambiente in cui si muove e stimare la sua posizione all'interno di essa.
Landmark	Punto del mondo la cui posizione è stimata dallo SLAM. L'insieme dei landmark costituisce la mappa.
Visual SLAM	SLAM facente uso di camere come sensori principali.

Monocular SLAM	SLAM facente uso di una singola camera come sensore.
Punto in coordinate omogenee	Un punto n-dimensionale P_o è espresso in coordinate omogenee quando i parametri che lo definiscono sono $n+1$ e sussiste la seguente relazione con la sua rappresentazione cartesiana P: $$P_o = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ w \end{bmatrix} \iff \begin{bmatrix} x_1/w \\ x_2/w \\ \vdots \\ x_n/w \end{bmatrix} = P$$

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Uno degli obiettivi principali della robotica è quello rendere le macchine sempre più autonome, in modo che possano svolgere compiti sempre più complessi, in ambienti generici e potenzialmente ignoti. Una caratteristica di fondamentale importanza per raggiungere questo obiettivo, è la capacità del robot di determinare la propria posizione all'interno dell'ambiente. Lo SLAM (*Simultaneous Localization And Mapping*) è un algoritmo che, elaborando le misure provenienti dai sensori del robot, permette di stimare la traiettoria percorsa dal robot durante la fase di esplorazione dell'ambiente, inizialmente sconosciuto, congiuntamente alla costruzione di una sua mappa.

Sebbene inizialmente i sensori adoperati per lo SLAM fossero generalmente telemetri laser e sonar, negli ultimi anni ha acquistato molta importanza l'uso di telecamere, a causa della loro economicità e del loro peso e consumo ridotti. Una camera è un dispositivo in grado di acquisire immagini dell'ambiente ricche di dettagli, sulle quali effettuare il *tracking* delle *feature*, le cui posizioni rappresentano le misure necessarie al funzionamento dello SLAM. Una delle tecniche di *tracking* per camere prospettiche più utilizzata ed efficace è il *patch tracking*, che misura lo spostamento della posizione delle *feature* lungo la sequenza di frame, comparando le aree dell'immagine nell'intorno delle *feature*. Le *feature* di un'immagine corrispondono a punti caratteristici del mondo, anche chiamati *landmark*, i quali si assumono essere statici e costituiscono la mappa dell'ambiente stimata dallo SLAM. Purtroppo, la posizione di una *feature* nell'immagine, non fornisce nessuna informazione sulla distanza del relativo *landmark* dalla camera, complicando notevolmente la sua fase di inizializzazione.

Il presente lavoro di tesi si focalizza sull'uso di una camera omnidirezionale come unico sensore (*Monocular SLAM*), la cui caratteristica principale è l'ampio campo visivo orizzontale di 360°. Questo permette di misurare i *landmark* dell'ambiente per periodi più lunghi rispetto a una camera prospettica, e quindi di ottenere una stima più precisa della loro posizione nel mondo. Le immagini acquisite con una camera omnidirezionale sono però affette da forte distorsione, che non permette l'applicazione diretta dei metodi di *tracking* sviluppati negli ultimi decenni per le camere prospettiche.

L'obiettivo della tesi è stato quello di estendere il *framework MoonSlam*, affinché si potesse utilizzare una camera omnidirezionale per effettuare SLAM. *MoonSlam*, realizzato da Simone Ceriani all'interno dell'AIRLab (Laboratorio di Intelligenza Artificiale e Robotica) del Politecnico di Milano, è composto da un insieme di librerie sviluppate in C++, che permettono di realizzare applicazioni *Visual SLAM* basate sul filtro esteso di Kalman.

Sono state realizzate le parametrizzazioni dei *landmark* per camere omnidirezionali *Inverse Scaling*, *Unified Inverse Depth*, *Framed Homogeneous Point* e *Framed Inverse Depth*, necessarie per rappresentare in maniera corretta la distribuzione di probabilità della posizione dei *landmark* all'interno del filtro, durante la loro fase di inizializzazione.

È stata introdotta una nuova strategia di inizializzazione dei landmark, che li dispone sul piano

del terreno sul quale il robot si muove, in alternativa all'inizializzazione classica a distanza costante. Questo metodo è stato pensato per ambienti esterni, dove localmente al robot il terreno sottostante risulta approssimativamente planare.

Sono stati studiati e implementati alcuni nuovi metodi di *tracking*, che permettono di superare le difficoltà derivanti dalla forte distorsione delle immagini omnidirezionali:

- ▷ *tracking panoramico* — Le immagini omnidirezionali vengono trasformate in immagini panoramiche, sulle quali viene applicato il classico metodo di *patch tracking* per camere prospettiche. Le immagini panoramiche hanno una distorsione ridotta rispetto alle immagini di partenza, rendendo possibile il *tracking* delle *patch* anche a fronte di una rotazione del robot intorno al proprio asse.
- ▷ *tracking con patch prospettiche* — Il metodo applica a livello locale delle *feature* una trasformazione prospettica che permette di eliminare la distorsione propria delle immagini omnidirezionali. Una variante di questo metodo, permette inoltre di sfruttare le informazioni prodotte dallo SLAM per tener conto del cambiamento di scala delle *feature*, che si ha quando un landmark viene osservato a lungo, e la sua distanza dalla camera cambia radicalmente.
- ▷ *patch warping omnidirezionale* — Si assume che un generico *landmark* del mondo faccia parte di una superficie planare, la cui normale viene ipotizzata essere la media fra le direzioni dal centro ottico della camera al landmark, in fase di inizializzazione e attuale. Utilizzando una trasformazione omografica, calcolata utilizzando le informazioni sulla posizione del *landmark* e della camera stimate dallo SLAM, il metodo è in grado calcolare l'aspetto attuale della *patch*, elaborando la sua immagine acquisita in fase di inizializzazione. Questo metodo di *tracking* è già stato applicato con successo con immagini prospettiche, e viene qui riproposto e adattato per l'utilizzo con camere omnidirezionali.
- ▷ *aggiornamento delle patch* — In presenza di un metodo in grado di valutare l'affidabilità delle misure effettuate, è possibile aggiornare l'aspetto delle *patch*, sostituendole con le porzioni di immagine attuali, centrate nelle nuove posizioni delle *feature*; Il metodo di aggiornamento delle *patch*, per effettuare il *tracking* omnidirezionale, è applicabile al metodo panoramico e a quello con *patch* prospettiche nella sua versione base (senza la variante che gestisce il cambiamento di scala con le informazioni dello SLAM). Il metodo è inoltre applicabile al *patch tracking* classico per camere prospettiche, permettendone l'utilizzo anche con camere omnidirezionali.

Tutti i metodi sviluppati sono stati implementati all'interno di *MoonSlam* e sono stati opportunamente testati. I risultati ottenuti dimostrano le potenzialità della camera omnidirezionale e la validità degli algoritmi introdotti.

La struttura del presente elaborato è la seguente:

- ▷ *Capitolo 2* — Vengono discussi i modelli geometrici e matematici di camera, a partire da quello *pinhole* per camera prospettica a due possibili modelli per camera omnidirezionale, fra cui il modello a sfera, molto utilizzato dalla comunità scientifica.
- ▷ *Capitolo 3* — Tratta il problema del *tracking*, illustrando le tecniche più conosciute ed efficaci per il modello di camera prospettico, sulle quali si basano anche i metodi di *tracking* per camere omnidirezionali sviluppati.

- *Capitolo 4* Discute la teoria del filtraggio, illustrando nel dettaglio il funzionamento del filtro di Kalman esteso, frequentemente utilizzato per implementare sistemi di SLAM.
- *Capitolo 5* Affronta il problema dello SLAM, illustrandone le caratteristiche, la storia e le soluzioni più utilizzate. Viene spiegato come utilizzare il filtro di Kalman esteso per risolvere il problema dello SLAM e alcuni dettagli implementativi indispensabili per ottenere una soluzione computazionalmente efficiente. Viene analizzato il problema di inizializzazione dei landmark per il *Monocular SLAM* e vengono presentati alcuni algoritmi accessori per migliorare le prestazioni e la qualità dello SLAM, come la suddivisione in sottomappe (CI-SLAM) e il rilevamento di osservazioni non corrette (1-Point RANSAC).
- *Capitolo 6* Entra nel dettaglio delle parametrizzazioni dei *landmark* per il *Monocular SLAM*, indispensabili per una corretta inizializzazione dei *landmark* sin dalla loro prima osservazione.
- *Capitolo 7* Vengono presentati i contributi teorici del lavoro di tesi, illustrando le tecniche di *tracking* per immagini omnidirezionali introdotte e gli altri algoritmi sviluppati.
- *Capitolo 8* Presenta i contributi pratici del lavoro di tesi, implementati all'interno del framework *MoonSlam*.
- *Capitolo 9* Illustra i risultati dei test effettuati con le tecniche sviluppate, mostrando la buona qualità dello SLAM ottenuta con l'utilizzo della camera omnidirezionale.
- *Capitolo 10* Commenta brevemente i risultati ottenuti e introduce nuovi spunti per sviluppi futuri.

Capitolo 2

Modello di camera

Una camera è un sensore che permette di acquisire i raggi luminosi provenienti dal mondo. Gli scopi di questa acquisizione sono i più disparati: dall'archiviazione di ricordi, alla produzione di film, alla videoconferenza e all'interfacciamento fra il mondo e la macchina, quale il nostro caso. Esistono diverse tipologie di camera, ognuna delle quali presenta delle specifiche caratteristiche che la rendono più adatta a un utilizzo piuttosto che a un altro. Un esempio potrebbe essere la fotocamera digitale, che presenta generalmente un'alta risoluzione e permette quindi l'acquisizione di fotografie ad alta definizione. Tale tipo di camera non sarebbe però indicata come mezzo per effettuare videoconferenze, in quanto non in grado di acquisire fotogrammi a una frequenza tale da rendere fluida la comunicazione. Per l'utilizzo della camera che questa tesi si propone di fare, è indispensabile che il flusso di acquisizione sia continuo (videocamera), in quanto è necessario seguire lo spostamento di alcuni punti caratteristici al passare dei frame (*tracking*). Grazie a questi spostamenti è possibile stimare il movimento della camera, ed è logico pensare che se l'acquisizione non fosse continua come per esempio nel caso della fotocamera, oltre a incontrare maggiori difficoltà a effettuare il tracking, la stima della sua posizione nell'intervallo fra due frame sarebbe solo approssimabile per interpolazione. È inoltre importante ricordare che un'alta frequenza di acquisizione facilita l'inseguimento dei punti caratteristici, in quanto la ricerca di un punto nel frame successivo viene fatta in un area limitata nell'intorno della posizione che il punto assumeva nel frame precedente, e tale area può essere ridotta a fronte di un innalzamento della frequenza di campionamento.

Fra le altre caratteristiche di una camera, vi è il campo di visione, o angolo visivo, ossia la porzione di mondo che può essere acquisita con un singolo frame. Generalmente viene definito come angolo, rispetto ai 360 gradi che è la visione completa del mondo. Il nostro occhio umano ha un campo di visione di 155 gradi in orizzontale e di 120 gradi in verticale.

Un'altra caratteristica interessante è la distorsione radiale, introdotta dalle lenti di cui la camera è dotata. Questo tipo di distorsione fa si che linee rette nel mondo si trasformino in linee curve nell'immagine acquisita.

La tipologia di camera più utilizzata è senza dubbio quella prospettica che presenta un campo di visione ridotto, ma una distorsione radiale limitata. Esempi di camera di questo tipo sono le normali fotocamere digitali e videocamere. Le immagini prodotte da queste camere sono facilmente valutabili dall'occhio umano, perché rappresentano la realtà nello stesso modo in cui la rappresenta il nostro occhio. Per l'interfacciamento di un computer con il mondo esterno la distorsione radiale non pone particolari problemi, a patto di conoscere le equazioni che la governano. Al contrario, un campo di visione più ampio, permetterebbe di ottenere maggiori informazioni sulla realtà circostante. Per questo motivo, nei sistemi di robotica vengono spesso usate camere *omnidirezionali*. La loro caratteristica principale è la capacità di avere un campo visivo orizzontale di 360 gradi, al costo di un'elevata distorsione radiale.

Verrà ora presentato il modello geometrico di camera prospettica, che definisce le equazioni che legano la direzione di un raggio luminoso entrante nella camera e il corrispondente pixel sull'immagine

Figura 2.1: *Camera pinhole del XIX secolo*

catturata. A seguire, verrà descritto brevemente il modello di camera ortografica e successivamente due modelli di camera omnidirezionale, uno dei quali viene sviluppato a partire dal modello di camera prospettica.

2.1 La camera prospettica

Il modello utilizzato nella letteratura per descrivere la camera prospettica è il modello "pinhole". La geometria di tale modello prevede che i raggi luminosi provenienti dal mondo che passano attraverso un foro di dimensione infinitesima, vadano a formare l'immagine su un piano disposto dietro il foro, chiamato piano immagine. Non essendoci nessuna lente, non vi è distorsione radiale. Il campo di visione dipende dalla dimensione del piano immagine e dalla lunghezza focale, ossia la sua distanza dal foro.

Descriviamo il modello *pinhole* in termini matematici. Consideriamo un punto nel mondo con coordinate $\mathbf{X}_c = [X_c, Y_c, Z_c]^T$ il cui sistema di riferimento $\mathbf{O}$ è centrato nel foro (sistema di riferimento camera). $\mathbf{O}$ prende il nome di centro ottico. L'asse z è l'asse ottico, ed è la retta perpendicolare al piano dell'immagine passante per il centro ottico. Il raggio luminoso che segue l'asse ottico è chiamato raggio principale. Come detto precedentemente la distanza fra il piano immagine e il centro ottico è la lunghezza focale f. Per la geometria mostrata in Figura 2.2 risulta che il punto sul piano immagine $\mathbf{x} = [x, y]^T$ corrispondente a $\mathbf{X}_c$ è così definito:

$$x = -f\frac{X_c}{Z_c} \qquad y = -f\frac{Y_c}{Z_c}.$$

Si noti il segno meno che precede la lunghezza focale f, che ribalta orizzontalmente e verticalmente l'immagine. Per questioni di praticità l'immagine viene ribaltata a posteriori. Questa operazione dal punto di vista del modello è equivalente a porre il piano immagine di fronte al centro ottico invece che retrostante, chiaramente alla stessa distanza f da esso (Figura 2.3). Si ottiene così:

$$x = f\frac{X_c}{Z_c} \qquad y = f\frac{Y_c}{Z_c}.$$

Essendo il piano immagine limitato, il suo campo visivo sarà limitato a sua volta. Se $2r$ è la larghezza del piano immagine allora il campo visivo orizzontale è:

$$\theta = \arctan(r/f)$$

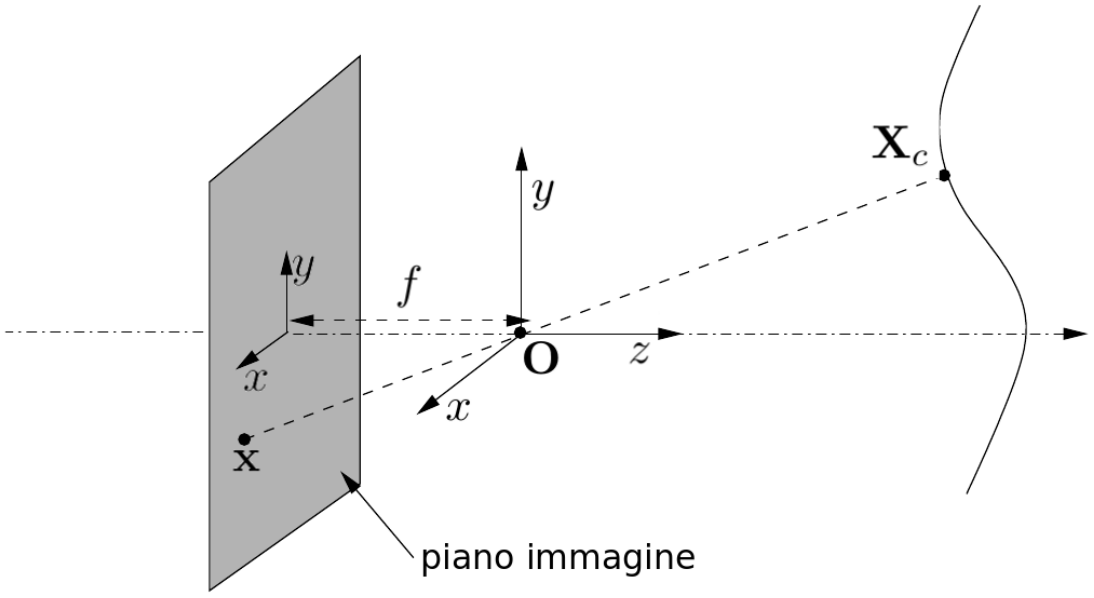


Figura 2.2: *Modello di camera pinhole con piano immagine dietro il centro ottico*

che non può comunque superare i 180 gradi.

2.1.1 Camera prospettica ideale

Consideriamo un generico punto nel mondo con coordinate $\mathbf{X}_w = [X_w, Y_w, Z_w]^T$, nel sistema di riferimento mondo. Lo stesso punto nel sistema di riferimento camera è esprimibile come:

$$\mathbf{X}_c = R\mathbf{X}_w + T \qquad \text{con} \quad R \in \mathbb{R}^{3\times 3} \quad \text{e} \quad T \in \mathbb{R}^{3\times 1}$$

dove $\mathbf{X}_c = [X_c, Y_c, Z_c]^T$ è il punto nel sistema di riferimento camera, R e T le matrici di rotazione e traslazione fra il sistema di riferimento mondo e camera.

Utilizzando il modello pinhole di camera, il punto $\mathbf{X}_c$ è proiettato sul piano immagine nel punto $\mathbf{x}$:

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} = \frac{f}{Z_c} \begin{bmatrix} X_c \\ Y_c \end{bmatrix}$$

che in coordinate omogenee è esprimibile come:

$$Z_c \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}.$$

Il fattore Z_c che precede $\mathbf{x}$ è la profondità (generalmente ignota) del punto nel mondo visto dalla camera e può essere meglio espressa chiamandola $\lambda \in \mathbb{R}^+$.

La matrice 3×4 può essere suddivisa nelle due matrici K_f e Π_0 nel modo seguente:

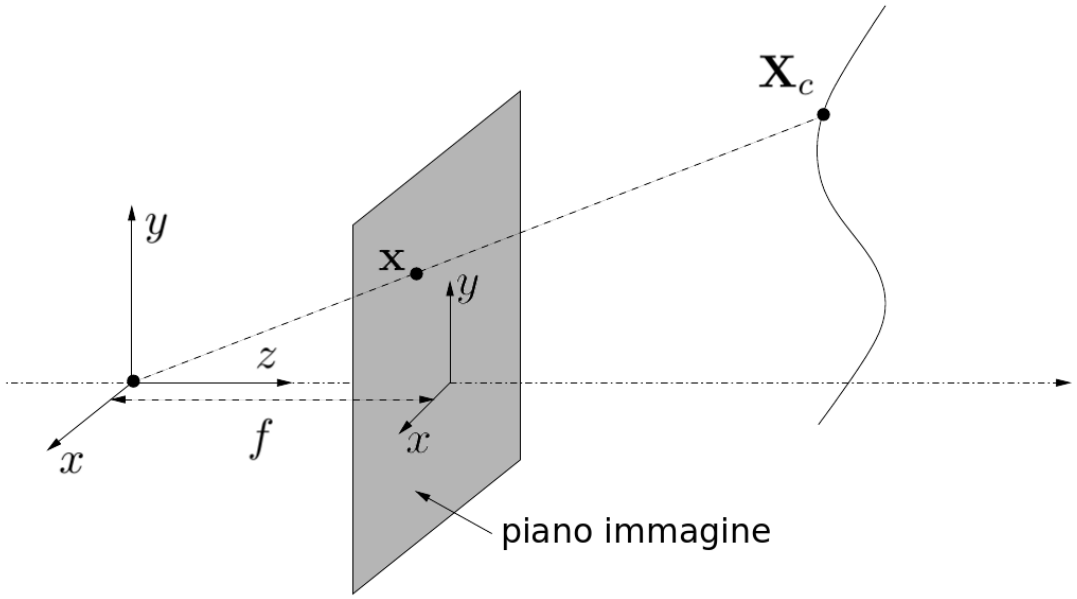


Figura 2.3: *Modello di camera pinhole con piano immagine in fronte al centro ottico*

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = K_f \Pi_0$$

dove la matrice Π_0 è solitamente chiamata matrice di proiezione canonica.

Componendo il tutto, il modello geometrico di una camera ideale è il seguente:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

o in notazione matriciale:

$$\lambda \mathbf{x} = K_f \Pi_0 S_w^c \mathbf{X}_w$$

chiamando S_w^c la matrice di rotazione e traslazione fra sistema di riferimento mondo e camera.

2.1.2 Camera prospettica con parametri intrinseci

Il modello di camera prospettica ideale presuppone che il raggio principale venga impresso nell'immagine alle coordinate $[0, 0]^T$ (sistema di riferimento del piano immagine). Nelle camere reali una immagine è espressa in termini di pixel, e il suo sistema di riferimento è generalmente centrato nell'angolo in alto a sinistra dell'immagine. Dobbiamo quindi definire la relazione che lega il modello reale a quello ideale.

Cominciamo col definire le unità di misura lungo gli assi. Se x e y sono definiti in millimetri ad esempio, x_s e y_s sono il loro corrispondente in pixel. x_s e y_s sono quindi una versione scalata di x e y secondo la relazione

$$\begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

s_x e s_y possono essere a priori diversi fra loro, codificando quindi una forma del pixel rettangolare piuttosto che quadrata. In realtà si potrebbe anche considerare il caso più generale di pixel trapezoidale introducendo il termine s_θ come secondo elemento della prima riga della matrice di scala, ma questa condizione nelle camere moderne e normalmente utilizzate non è praticamente mai verificata. Si continuerà quindi ad assumere $s_\theta = 0$.

Il punto $[x_s, y_s]^T$ ha ancora come sistema di riferimento il punto $[0, 0]^T$. È necessario quindi effettuare la seguente traslazione:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x_s \\ y_s \end{bmatrix} + \begin{bmatrix} x_c \\ y_c \end{bmatrix}$$

dove $[x_c, y_c]^T$ sono le coordinate in pixel del punto principale e $[x', y']^T$ le coordinate in pixel del punto $[x, y]^T$ con il nuovo sistema di riferimento.

Riassumendo, la trasformazione di un punto $\mathbf{x}$ dal modello ideale al punto $\mathbf{x}'$ del modello reale è descritta in coordinate omogenee come:

$$\mathbf{x}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_c \\ 0 & s_y & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

o in forma matriciale

$$\mathbf{x}' = K_s \mathbf{x}$$

Si ottiene così il modello geometrico completo di camera prospettica:

$$\lambda \mathbf{x}' = \lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_c \\ 0 & s_y & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

ossia

$$\lambda \mathbf{x}' = K_s K_f \Pi_0 S_w^c \mathbf{X}_w.$$

I parametri definiti nelle matrici K_s e K_f non sono misurabili con precisione, ma vengono solamente stimati con un processo di calibrazione. Tale processo non permette di determinare le due matrici, ma soltanto il loro prodotto. È per questo motivo che viene definita semplicemente la matrice $K = K_s K_f$ come segue:

$$K = K_s K_f = \begin{bmatrix} f s_x & 0 & x_c \\ 0 & f s_y & y_c \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & x_c \\ 0 & f_y & y_c \\ 0 & 0 & 1 \end{bmatrix}.$$

La matrice K è chiamata matrice dei parametri intrinseci della camera, o semplicemente matrice intrinseca.

(a) *Immagine distorta* **(b)** *Immagine rettificata*

Figura 2.4: *Distorsione nelle camere prospettiche*

2.1.3 Distorsione nelle camere prospettiche

Il modello pinhole è un'astrazione universalmente utilizzata nonostante nessuna camera risponda ai suoi requisiti: la presenza di un foro di dimensione infinitesimale non è realizzabile, in quanto l'energia luminosa che entrerebbe da tale foro e che colpirebbe la retina non sarebbe sufficiente. L'aumento della dimensione del foro porterebbe a uno sfocamento dell'immagine formata sulla retina, in quanto un suo punto non corrisponderebbe più a un unico raggio luminoso proveniente dal mondo, ma alla somma di più raggi contenuti in un angolo solido di dimensione proporzionale alla sezione del foro. Per ovviare a questi problemi, vengono utilizzate una serie di lenti, che convogliano i raggi luminosi in un foro di dimensioni contenute, ma con energia sufficiente. L'introduzione di tali lenti provoca però una distorsione dell'immagine più o meno evidente. La Figura 2.4 mostra un esempio di immagine distorta e la sua rettificazione.

Il modello matematico di Brown-Conrady descrive il fenomeno della distorsione radiale, ma include anche un termine legato alla distorsione tangenziale, di scarsa importanza nelle camere prospettiche.

Chiamiamo x_d e y_d le coordinate di un punto generico $\mathbf{x}_d$ dell'immagine affetto da distorsione, x_u e y_u le coordinate del corrispondente punto $\mathbf{x}_u$ dopo la rettifica (rimozione della distorsione). Le coordinate sono espresse in pixel.

$$\begin{aligned} x_u &= x_d + \underbrace{(x_d - x_c)(K_1 r^2 + K_2 r^4 + \dots)}_{\text{Distorsione radiale}} + \underbrace{(P_1(r^2 + 2(x_d - x_c)^2) + 2P_2(x_d - x_c)(y_d - y_c))(1 + P_3 r^2 + \dots)}_{\text{Distorsione tangenziale}} \\ y_u &= y_d + \underbrace{(y_d - y_c)(K_1 r^2 + K_2 r^4 + \dots)}_{\text{Distorsione radiale}} + \underbrace{(P_2(r^2 + 2(y_d - y_c)^2) + 2P_1(x_d - x_c)(y_d - y_c))(1 + P_3 r^2 + \dots)}_{\text{Distorsione tangenziale}} \end{aligned}$$

Dove:

- x_c e y_c = coordinate del punto principale
- K_i = i-esimo coefficiente della distorsione radiale
- P_i = i-esimo coefficiente della distorsione tangenziale
- $r = \sqrt{(x_d - x_c)^2 + (y_d - y_c)^2}$ = distanza del punto $\mathbf{x}_d$ dal punto principale.

Per le normali camere prospettiche generalmente i primi due coefficienti di distorsione radiale K_1 e K_2 sono sufficienti ad allineare il modello geometrico di camera alla realtà.

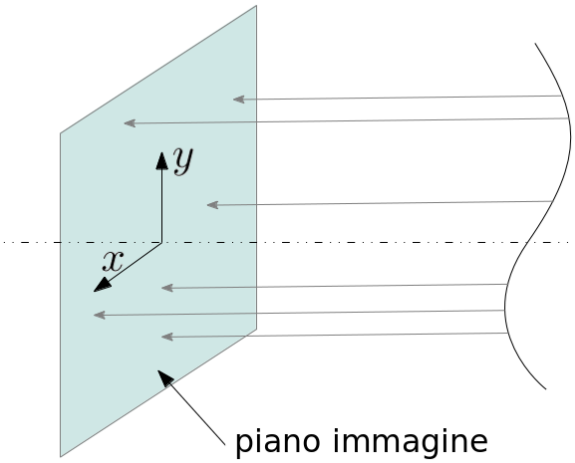


Figura 2.5: *Modello di camera ortografica*

2.2 La camera ortografica

Una camera ortografica risponde al modello di proiezione ortografica, che si differenzia dalla proiezione prospettica descritta in precedenza nel modo in cui i raggi formano l'immagine. Nella camera prospettica tutti i raggi entranti sono costretti a passare per il centro ottico, mentre in quella ortografica viaggiano parallelamente come mostrato in Figura 2.5.

La camera ortografica è un'astrazione, ma è facilmente approssimabile nella realtà dotando una camera prospettica di particolari lenti.

2.3 La camera omnidirezionale

Una camera omnidirezionale ha come principale caratteristica un campo visivo orizzontale di 360 gradi. Vi sono varie tipologie di realizzazioni differenti per raggiungere tale obiettivo:

1. Utilizzando più camere prospettiche, ognuna delle quali cattura una porzione della realtà
2. Utilizzando una camera prospettica con in fronte una serie di specchi (ad esempio a forma di piramide)
3. Applicando particolari lenti a una camera prospettica (camere fisheye)
4. Utilizzando una camera prospettica con in fronte uno specchio opportunamente sagomato a forma di iperbole
5. Utilizzando una camera ortografica con in fronte uno specchio a forma di parabola
6. Utilizzando una camera e una forma dello specchio diversi dai casi 4 e 5

Una caratteristica di fondamentale importanza che una camera deve possedere per alcune applicazioni nel campo della robotica, è l'avere un unico centro di proiezione (centro ottico nelle camere prospettiche). La sua importanza deriva dal fatto che l'obiettivo è quello di riuscire a determinare la direzione del raggio che ha formato un determinato punto dell'immagine, e tale obiettivo sarebbe più difficilmente raggiungibile con camere senza questa proprietà. Inoltre la presenza di un unico punto di vista (centro di proiezione) permette la ricostruzione di immagini prospettiche geometricamente corrette a partire da immagini omnidirezionali.

Le opzioni 1, 2 e 6 non hanno un unico centro di proiezione mentre le altre tre opzioni possiedono tutte questa proprietà. Le camere fisheye hanno un campo visivo non così esteso come le camere catadiottriche (opzioni 4, 5 e 6), d'altra parte nelle camere catadiottriche una porzione del campo

(a) *fisheye* **(b)** *catadiottrica*

Figura 2.6: *Esempio di immagini prodotte con una camera fisheye e catadiottrica*

visivo è ostruita dalla camera stessa che si riflette nello specchio. Nel campo della robotica si utilizzano spesso le camere catadiottriche perché rappresentano una buona soluzione a un costo contenuto.

Nell'articlo di Baker e Nayar [BN99] vengono analizzate tutte le possibili realizzazioni di camere catadiottriche, ed è dimostrato che vi sono soltanto tre tipi di accoppiamenti fra camera e forma dello specchio che mantengono un unico centro di proiezione: i primi due li abbiamo già elencati (opzioni 4 e 5) e il terzo è il caso di camera prospettica e specchio a forma di ellisse. La terza alternativa non è stata da noi considerata perché raramente utilizzata a causa della sua scarsità di campo visivo nel confronto con le altre due tipologie.

Per dare origine a camere omnidirezionali con un unico centro di proiezione, è necessario inoltre che lo specchio sia posizionato con precisione in uno specifico punto rispetto alla camera che lo fotografa: il centro ottico della camera prospettica deve essere posizionato nel secondo fuoco dell'iperbole o ellisse che sagomano lo specchio e il loro asse dev'essere perpendicolare al piano immagine. Il vincolo sulla perpendicolarità dell'asse al piano immagine persiste anche nel caso di camera ortografica e specchio parabolico, ma la posizione dello specchio può essere traslata senza troppi problemi, in quanto i raggi continueranno ad arrivare perpendicolarmente sul piano immagine.

L'opzione 5 ha inoltre un altro paio di utili vantaggi: è calibrabile con maggior facilità e la sua costruzione fisica risulta più semplice in quanto il modello risente meno degli errori di posizionamento dello specchio.

Baker e Nayar hanno inoltre dimostrato che la curvatura dello specchio aumenta lo sfocamento dell'immagine, e quindi le immagini catadiottriche, partendo dal centro vanno via via sfocandosi verso l'esterno. Questa caratteristica è facilmente verificabile anche in Figura 2.6b.

Verranno ora presentati due modelli matematici per la camera catadiottrica, uno molto generico sviluppato da Scaramuzza [SMS06],[Sca08] e l'altro molto più specifico sviluppato prima da Geyer e Barreto e successivamente da Mei [MR07]: il modello a sfera. Questi modelli in realtà sono adattabili anche a camere fisheye.

2.3.1 Il modello proposto da Scaramuzza

La costruzione di camere catadiottriche con un unico centro di proiezione impone i vincoli sopra citati, fra i quali, nel caso di accoppiamento camera prospettica e specchio iperbolico, il fatto che il centro ottico della camera dev'essere posto esattamente nel secondo fuoco dell'iperbole. La

validità del modello è quindi legata fortemente al processo di costruzione della camera e all'errore di allineamento dello specchio.

Per ovviare a questo problema Scaramuzza propone un modello parametrico generico con relativo metodo di calibrazione e toolbox per Matlab. L'assunzione sottostante è che la funzione di formazione dell'immagine possa essere descritta tramite una serie di Taylor.

Definiamo $\mathbf{x} = [x, y]^T$ il punto sul piano immagine ideale e $\mathbf{x}' = [x', y']^T$ il suo corrispondente sul piano immagine reale espresso in pixel. Come osservato nella Sezione 2.1.2 la relazione che sussiste fra i due è una semplice trasformazione affine:

$$\mathbf{x}' = S\mathbf{x}+\mathbf{x}_c = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}\mathbf{x}+\begin{bmatrix} x_c \\ y_c \end{bmatrix} \qquad \Rightarrow \qquad \mathbf{x} = S^{-1}(\mathbf{x}'-\mathbf{x}_c) = \begin{bmatrix} \frac{1}{s_x} & 0 \\ 0 & \frac{1}{s_y} \end{bmatrix}\mathbf{x}'+\begin{bmatrix} \frac{x_c}{s_x} \\ \frac{y_c}{s_y} \end{bmatrix} = A\mathbf{x}'+T.$$

Introduciamo ora una funzione g che mappa un punto $\mathbf{x}$ dal piano immagine ideale nel corrispondente punto $\mathbf{X}_c$ del mondo (nel sistema di riferimento camera):

$$\lambda\mathbf{X}_c = \lambda g(\mathbf{x}) = \lambda g(A\mathbf{x}' + T) \qquad \text{con} \qquad \lambda \in \mathbb{R}^+.$$

Per la geometria delle camere omnidirezionali si può assumere che la funzione g sia della forma:

$$g(\mathbf{x}) = g(x, y) = [x, y, f(\rho)]^T \qquad \text{con} \qquad \rho = \sqrt{x^2 + y^2}$$

con f funzione rotazionalmente simmetrica e dipendente solo dalla distanza del punto $\mathbf{x}$ dall'origine (raggio).

Definiamo la forma della funzione f come espansione in serie di Taylor:

$$f(\rho) = a_0 + a_1\rho + a_2\rho^2 + \cdots + a_N\rho^N.$$

Riassumendo, il modello matematico di camera omnidirezionale proposto da Scaramuzza è:

$$\lambda\mathbf{X}_c = \lambda\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \lambda\mathbf{g}(\mathbf{x}) = \lambda\begin{bmatrix} x \\ y \\ \mathbf{f}(\rho) \end{bmatrix} = \lambda\begin{bmatrix} A\mathbf{x_s} + T \\ a_0 + a_1\rho + a_2\rho^2 + \cdots + a_N\rho^N \end{bmatrix}.$$

La calibrazione del modello è effettuata a partire da immagini acquisite con la camera di un pattern piano a forma di scacchiera. Il processo di calibrazione produce una stima dei parametri A, T, l'ordine dell'espansione di Taylor N e i suoi coefficienti a_i.

Esperimenti hanno provato che un numero di coefficienti pari a 5 o 6 è sufficiente.

2.3.2 Il modello proposto da Mei

La geometria della camera catadiottrica ha un modello di proiezione leggermente più complesso del caso prospettico, composto da due distinte fasi:

1. Un punto nel mondo viene proiettato sulla superficie dello specchio in direzione del suo centro ottico (fuoco della parabola/iperbole che lo specchio disegna)
2. Il punto sulla superficie dello specchio viene proiettato sul piano immagine secondo una proiezione prospettica o ortografica, nel caso lo specchio abbia forma iperboidale o parabolica rispettivamente. Naturalmente nel caso fosse una proiezione prospettica il centro ottico verso cui il punto è proiettato è quello della camera prospettica che fotografa lo specchio.

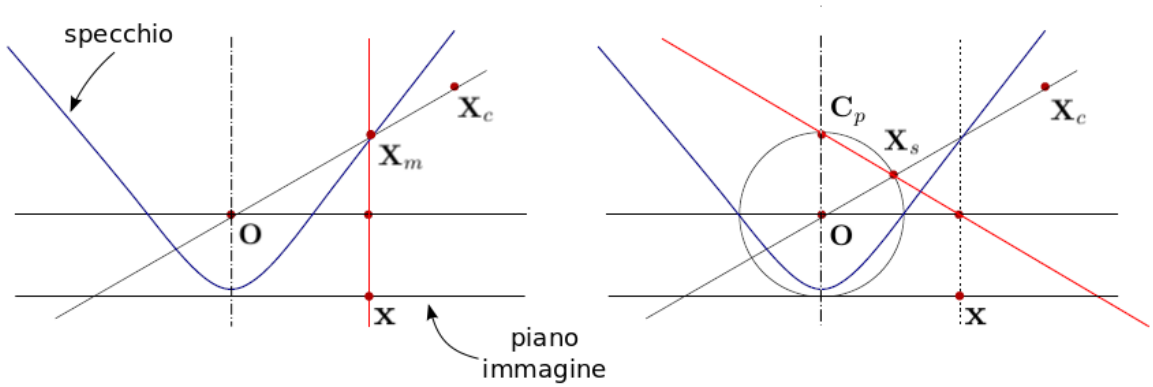


Figura 2.7: *Proiezioni equivalenti sulle quali si basa il modello di camera omnidirezionale proposto da Mei, con specchio parabolico e camera ortografica*

Il modello di Mei si sviluppa sull'equivalenza che sussiste fra la proiezione appena descritta e la seguente serie di passaggi:

1. Un punto del mondo viene mappato su una sfera di raggio unitario centrata nel fuoco dello specchio.
2. Il punto sulla sfera così ottenuto viene proiettato sul piano immagine a partire da un particolare punto $\mathbf{C}_p$ sull'asse della sfera.

La posizione del punto $\mathbf{C}_p$ dipende dalla forma dello specchio, assume un valore intermedio fra il centro della sfera e il polo nord per specchi iperbolici, mentre per il caso di specchi parabolici è esattamente il polo nord.

In Figura 2.7 è mostrata l'equivalenza descritta, nel caso di specchio parabolico, in cui $\mathbf{X}_c$ è il punto nel mondo, $\mathbf{X}_m$ la sua proiezione sulla superficie dello specchio, $\mathbf{X}_s$ la sua proiezione sulla sfera di raggio unitario, $\mathbf{O}$ il fuoco dello specchio, $\mathbf{C}_p$ centro di proiezione che proietta il punto $\mathbf{X}_s$ sul piano immagine e $\mathbf{x}$ il punto sul piano immagine.

Traduciamo in equazioni matematiche il procedimento appena descritto:

Definiamo $\mathbf{X}_c = [X_c, Y_c, Z_c]^T$ un punto nel mondo con sistema di riferimento camera (fuoco dello specchio).

1. Proiettiamo il punto $\mathbf{X}_c$ su una sfera di raggio unitario centrata nel fuoco dello specchio $\mathbf{O}$:

$$\mathbf{X}_s = \frac{\mathbf{X}_c}{||\mathbf{X}_c||} = \begin{bmatrix} X_s \\ Y_s \\ Z_s \end{bmatrix}.$$

2. Operiamo uno spostamento del sistema di riferimento, dal fuoco dello specchio $\mathbf{O}$ al centro di proiezione $\mathbf{C}_p$ sull'asse della sfera (e dello specchio):

$$\mathbf{X}_S = \begin{bmatrix} X_s \\ Y_s \\ Z_s + \xi \end{bmatrix}$$

   dove ξ è la distanza fra $\mathbf{O}$ e $\mathbf{C}_p$.

3. Proiettiamo il punto $\mathbf{X}_S$ sul piano dell'immagine utilizzando il nuovo sistema di riferimento:

$$\mathbf{m}_u = \hbar(\mathbf{X}_S) = \begin{bmatrix} \frac{X_s}{Z_s+\xi} \\ \frac{Y_s}{Z_s+\xi} \\ 1 \end{bmatrix}$$

4. Aggiungiamo un termine di distorsione radiale e tangenziale, che dipende palla posizione del punto sul piano immagine e dai coefficienti di distorsione chiamati genericamente V. Una trattazione più ampia della distorsione è già stata fatta nella Sezione 2.1.3 per la camera prospettica, e rimane identicamente valida per il caso omnidirezionale (si noti però che ora l'operazione di distorsione avviene fra punti sul piano normalizzato con unità di misura metrica, non pixel).
$$\mathbf{m}_d = \mathbf{m}_u + D(\mathbf{m}_u, V).$$

5. Convertiamo infine il punto $\mathbf{m}_d$ dalle coordinate metriche a quelle in pixel tramite la matrice K, come già mostrato nel caso prospettico nella Sezione 2.1.2. Nel caso catadiottrico questa matrice assume un valore più generico, in quanto non si riferisce solamente alla camera prospettica che fotografa lo specchio, ma al sistema nel suo complesso. Per questo motivo Mei ha chiamato questa matrice non più matrice intrinseca ma matrice generalizzata di proiezione, cambiando alcuni nomi ai parametri:
$$\mathbf{K} = \begin{bmatrix} f_1\eta & f_1\eta\alpha & u_0 \\ 0 & f_2\eta & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \gamma_1 & \gamma_\theta & u_0 \\ 0 & \gamma_2 & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$
con:
   - f_1 e f_2: lunghezza focale orizzontale e verticale rispettivamente
   - η: parametro che dipende dalla forma dello specchio
   - α: coefficiente di *skewness* che avevamo approssimato a 0 nel caso prospettico e continueremo a fare anche nel seguito.
   - u_0 e v_0: coordinate del punto principale dell'immagine (nel caso prospettico le avevamo chiamate x_c e y_c)
   - γ_1 e γ_2: lunghezza focale generalizzata orizzontale e verticale

   Come si può verificare, nell'espressione di K è presente il parametro η che dipende dalla forma dello specchio, e che quindi non ha nulla a che vedere con i parametri intrinseci della camera di cui il sistema è composto.

Il modello di Mei è riassunto nella Figura 2.8.

La calibrazione di una camera omnidirezionale con il modello di Mei consiste nello stimare il valore dei seguenti parametri:

- ξ: Distanza fra $\mathbf{O}$ e $\mathbf{C}_p$;
- K_1 e K_2: Coefficienti di distorsione radiale. Un approssimazione al secondo grado è sufficiente;
- P_1 e P_2: Coefficienti di distorsione tangenziale. A differenza della camera prospettica qui sono necessari, e un approssimazione al secondo grado è sufficiente;
- γ_1 e γ_2: Lunghezza focale generalizzata orizzontale e verticale rispettivamente;
- u_0 e v_0: coordinate del punto principale dell'immagine;

Per la calibrazione secondo questo modello, Mei ha realizzato un toolbox per Matlab, che funziona in modo pressoché analogo al toolbox di Scaramuzza e a quello per camere prospettiche. Anche in questo caso il processo di calibrazione prende come input una serie di immagini realizzate con la camera omnidirezionale di un pattern piano a scacchiera.

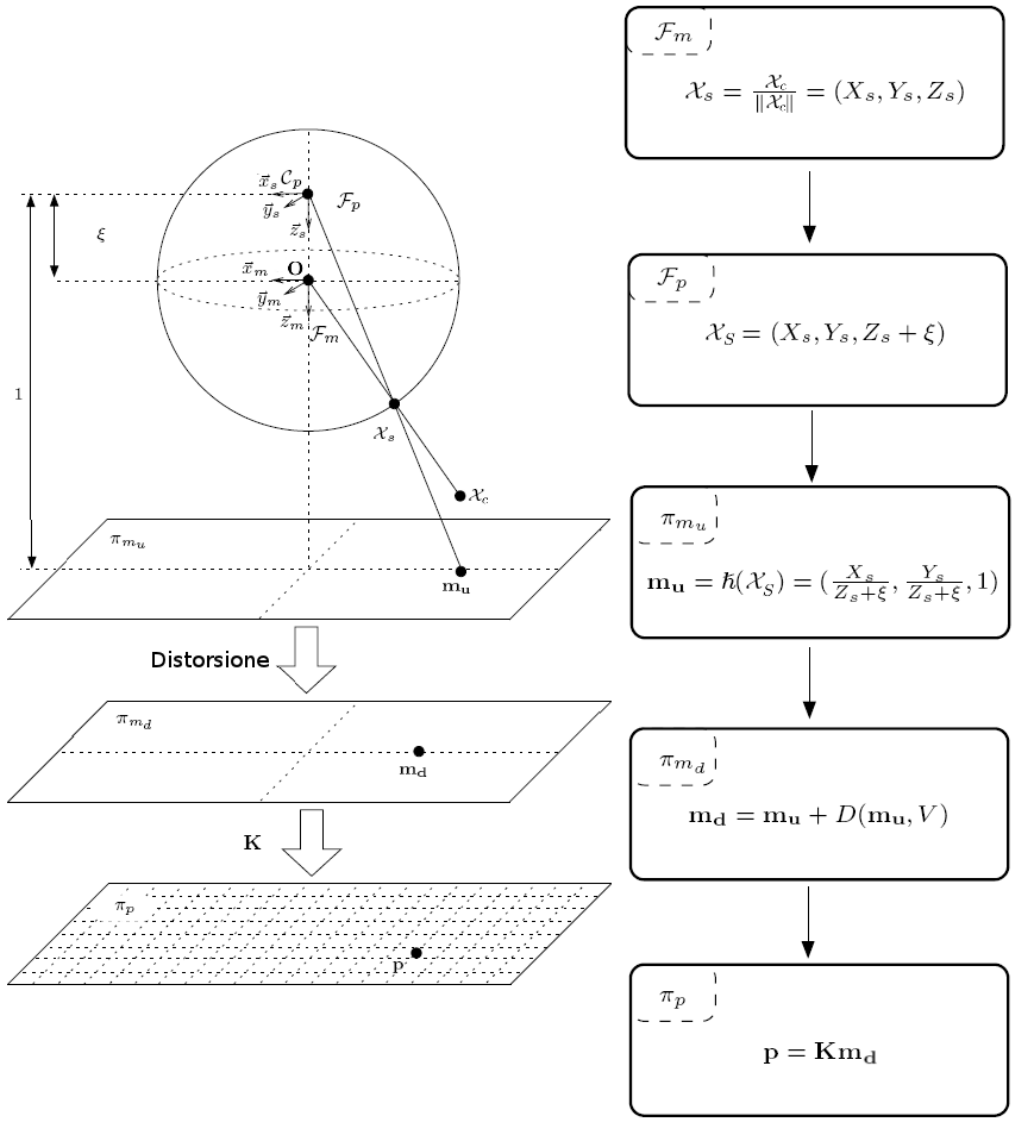


Figura 2.8: *Modello a sfera di camera omnidirezionale proposto da Mei*

2.4 Considerazioni finali sui modelli di camera

All'inizio del capitolo abbiamo spiegato i vantaggi che le camere omnidirezionali presentano rispetto a quelle prospettiche, nell'ambito della robotica. Questo è il motivo per il quale in questa tesi viene utilizzata una camera omnidirezionale.

La domanda ora è: quale dei due modelli di camera omnidirezionale utilizzare? Partiamo da alcune considerazioni. Il modello di Scaramuzza è molto generico e facile da capire mentre quello di Mei è molto più specifico. A differenza del modello di Scaramuzza, quello di Mei è costituito da una serie di passaggi geometrici e matematici direttamente interpretabili nel sistema da modellare.

Da entrambe le due opzioni è possibile ricavare il relativo modello inverso, che porta da un punto sull'immagine al corrispondente raggio di vista che l'ha generato. Il modello inverso insieme a quello diretto è esattamente ciò di cui abbiamo bisogno nel nostro lavoro per effettuare SLAM. Nel modello di Mei la funzione di distorsione D non è invertibile analiticamente, ma solo approssimabile numericamente con metodi iterativi.

Per decidere quale dei due modelli è migliore abbiamo però utilizzato altre due caratteristiche di fondamentale importanza: la velocità di applicazione del modello e la sua precisione.

- *velocità*: Nel nostro lavoro vi saranno due funzioni chiamate "world2cam" e "cam2world": la prima proietta un punto del mondo sull'immagine omnidirezionale, mentre la seconda al contrario recupera il raggio che ha formato un determinato punto nell'immagine. Tali funzioni sono utilizzate assiduamente come routine di base, ed è necessario che siano il più efficienti possibile.
- *precisione*: Per verificare la precisione della calibrazione e del modello, in entrambe i toolbox di Matlab è disponibile una funzione che effettua la riproiezione dei punti della scacchiera sull'immagine (conoscendo le caratteristiche della scacchiera, utilizzando i parametri stimati e la la posizione della scacchiera nel mondo anch'essa stimata). Questa funzione ha come output l'errore medio di riproiezione espresso in pixel, che definisce l'accuratezza del modello.

Nonostante la sua maggiore complessità il modello di Mei è più performante in termini di velocità di applicazione del modello. Per confrontare l'accuratezza abbiamo effettuato due calibrazioni, una con Mei e l'altra con Scaramuzza a partire da un unico dataset di immagini di calibrazione. L'errore medio di riproiezione per il modello di Scaramuzza risulta essere circa il doppio di quello di Mei.

In definitiva, considerando le due caratteristiche più importanti, il modello di Mei è migliore del modello di Scaramuzza. L'inferiorità del modello di Scaramuzza potrebbe essere giustificata dalla sua generalità e semplicità.

Nel nostro progetto si sono implementati entrambi i modelli, ma attualmente per fare SLAM si è portato avanti solo il modello di Mei.

Capitolo 3

Tracking di punti caratteristici

Viene chiamata tracking l'operazione di seguire lo spostamento di alcuni punti caratteristici in una sequenza di frame. Un punto caratteristico, anche chiamato feature, è un punto dell'immagine facilmente riconoscibile e differenziabile dagli altri punti nel suo intorno. Un esempio del processo di tracking è mostrato in Figura 3.1.

Il tracking generalmente non è un operazione fine a sé stessa, ma il suo risultato viene utilizzato da algoritmi a più alto livello. Nel nostro lavoro, il tracking viene utilizzato per stimare la posizione e il movimento della camera, nonché la posizione dei punti caratteristici nel mondo. La qualità del tracking influenza in maniera molto significativa gli algoritmi che lo utilizzano. È quindi essenziale sviluppare un tracking il più preciso e affidabile possibile. Questo argomento è stato studiato e sviluppato per decenni, ottenendo una varietà abbastanza ampia di soluzioni. Nella Sezione 3.1 viene descritto il tracking usando il famoso algoritmo SIFT e nella Sezione 3.2 usando patch (regioni dell'immagine intorno ai punti caratteristici). Nella Sezione 3.3 vengono accennate alcune soluzioni di tracking per camere omnidirezionali. Una trattazione più esaustiva verrà fatta nel Capitolo 7, dove verranno presentate le soluzioni sviluppate nel presente lavoro di tesi. Infine nella Sezione 3.4 vengono espresse alcune considerazioni sui vari metodi descritti.

3.1 Tracking con SIFT

SIFT (*Scale-Invariant Feature Transform*) è un metodo sviluppato da Lowe [Low04], che permette di rilevare i punti caratteristici di una immagine e associargli dei particolari descrittori basati sui gradienti delle aree dell'immagine nell'intorno delle feature. L'utilizzo naturale di questo algoritmo è la ricerca e l'associazione di punti caratteristici fra due o più immagini della stessa scena ma fotografate da punti di vista differenti. Inizialmente non fu utilizzato per fare tracking, a cau-

frame 1 frame 11 frame 21

Figura 3.1: *Il processo di tracking: i punti caratteristici marcati con una croce rossa e numerati vengono seguiti per una sequenza di 20 frame, dei quali ne sono mostrati tre.*

sa del suo alto costo computazionale, ma con l'evoluzione dei computer è ora possibile utilizzarlo efficacemente anche per questo scopo.

Un punto caratteristico di una immagine, chiamato da Lowe *keypoint*, è composto delle seguenti parti:

- *posizione*: coordinate x e y del punto nell'immagine
- *scala*: scala alla quale è stato rilevato il punto caratteristico
- *orientamento*: direzione principale del gradiente della regione nell'intorno del punto
- *descrittore*: vettore di 128 elementi che descrive in maniera pressoché univoca il punto caratteristico.

3.1.1 SIFT Detector

Data una immagine, per trovare i punti caratteristici presenti in essa, viene calcolato in modo efficiente il Laplaciano dell'immagine su tutte le possibili scale. Vengono estratti i punti che identificano i minimi e i massimi locali, i quali saranno i candidati punti caratteristici. La scala associata al punto sarà quindi la scala alla quale è stato trovato il minimo o massimo locale del Laplaciano.

Per definire l'orientamento del *keypoint* viene calcolato il gradiente della ragione nell'intorno del punto. Con l'orientamento dei pixel in questa area viene creato un istogramma di di 36 elementi che coprono l'intero campo dei 360 gradi. L'elemento con frequenza maggiore viene scelto come l'orientamento complessivo del punto caratteristico. Se vi sono più elementi con una frequenza simile alla maggiore vengono creati più *keypoint* ognuno dei quali con stessa posizione e scala, ma orientamento diverso.

3.1.2 SIFT Descriptor

Per calcolare il descrittore SIFT viene ripreso il gradiente dell'immagine in una regione di 16×16 pixel centrata nel *keypoint*. Questa regione viene divisa in una griglia 4×4. Vengono creati 16 istogrammi, uno per ogni cella della griglia. Ogni istogramma è composto da 8 direzioni principali che coprono il campo dei 360 gradi. Le orientazioni di tutti i pixel in una cella della griglia vengono inserite nel relativo istogramma. Il descrittore è dunque composto da un vettore di $4 \times 4 \times 8 = 128$ elementi contenente tutte le informazioni degli istogrammi calcolati. Per rendere il descrittore più efficace, l'attribuzione del gradiente di un pixel a uno specifico elemento dell'istogramma viene soppesata da una funzione Gaussiana bivariata centrata nel *keypoint*, in modo da dare minor importanza ai pixel lontani da esso. Inoltre, per rendere il descrittore invariante alla rotazione, è necessario sommare all'orientamento di ogni pixel l'orientamento generale del *keypoint* prima di inserirlo nell'apposito istogramma.

È stato dimostrato sperimentalmente che tale tipo di descrittore possiede una buona specificità, ossia, descrittori di uno stesso punto caratteristico visto da angolazioni diverse saranno simili, mentre punti caratteristici differenti avranno descrittori molto diversi fra loro. Il descrittore SIFT è invariante a cambiamenti di scala e rotazioni.

3.1.3 Associazione dei *keypoint* (*Matching*)

Per poter ritrovare una serie di punti caratteristici in due immagini della stessa scena, come prima cosa vengono estratti tutti i *keypoint* dalle due immagini, e calcolati i relativi descrittori. Per ogni punto caratteristico p_i della prima immagine, viene confrontato il suo descrittore con tutti i descrittori dei *keypoint* della seconda immagine. Il punto caratteristico con il descrittore che gli assomiglierà di più e che supera una soglia di somiglianza verrà definito come il corrispondente del punto p_i nella seconda immagine. Tale operazione nella letteratura scientifica viene chiamata

Figura 3.2: *Patch tracking: Nell'immagine di sinistra è mostrato un punto caratteristico (punto rosso) e la sua patch delimitata dal quadrato rosso. L'immagine sulla destra è il frame successivo: il quadrato blu identifica l'area di ricerca centrata nella posizione che assumeva il punto caratteristico nella prima immagine, il punto e quadrato rosso identificano il miglior match fra le patch dei punti nell'area di ricerca e la patch del punto caratteristico*

matching. Il confronto fra due descrittori viene generalmente calcolato come la distanza Euclidea fra i due vettori di 128 elementi che li rappresentano.

3.2 Tracking con patch

Una *patch* è una regione dell'immagine nell'intorno di un punto caratteristico. Generalmente la sua forma è quadrata o rettangolare, in quanto più semplice da trattare, muovendoci su una griglia discreta quale un'immagine in pixel.

Per effettuare il tracking usando patch, come prima operazione vengono rilevati i punti caratteristici di un frame, utilizzando un qualsiasi algoritmo di *corner detection*, come il famoso *Harris Corner Detector*. Al descrittore di ognuno dei punti rilevati, viene associata la sua patch. Nel frame successivo, si ipotizza che la posizione di un punto caratteristico non sia cambiata radicalmente, quindi la sua ricerca viene effettuata solamente in un area limitata centrata nella vecchia posizione del punto. Per ognuno dei pixel nell'area di ricerca viene estratta una patch centrata in tale punto e confrontata con la patch del punto caratteristico che si sta cercando. Il pixel con la somiglianza migliore, se al di sopra di una soglia di accettabilità, è definito come il corrispondente del punto caratteristico cercato.

In tutti i frame successivi, il confronto fra patch avviene sempre fra porzioni dell'immagine attuale e la patch estratta in fase di inizializzazione. La patch non viene aggiornata a ogni frame con il match trovato in quanto non vi è garanzia che la stima della sua nuova posizione sia giusta, e un tale errore si propagherebbe in tutti i frame successivi per l'intero tempo di vita del punto caratteristico. Effettuando il confronto sempre con la patch iniziale, se la stima di un punto caratteristico viene errata in un frame, può essere recuperata senza problemi nel frame successivo.

Vi sono vari metodi per effettuare il confronto fra due patch: SSD(*Sum of Squared Differences*), correlazione e le versioni normalizzate di questi due metodi in modo che la cifra di merito calcolata sia sempre compresa fra 0 e 1.

La dimensione delle patch dev'essere scelta con cautela, in quanto patch troppo piccole non sono in grado di catturare sufficienti informazioni dell'area circostante il punto caratteristico e identificare univocamente il punto, mentre patch troppo grandi comprenderebbero anche informazioni non legate all'oggetto a cui il punto appartiene, le quali cambierebbero radicalmente allo scorrere dei frame. La dimensione "giusta" delle patch è strettamente correlata alla risoluzione dell'immagine, ma anche alle caratteristiche della scena che l'immagine fotografa (informazione generalmente ignota). Inoltre alla larghezza e all'altezza delle patch vengono generalmente attribuiti valori dispari, in modo da avere una regione esattamente centrata intorno al suo punto caratteristico.

Figura 3.3: *Esempio di trasformazione affine di un'immagine*

I descrittori a patch non hanno però la proprietà di invarianza né alla rotazione né al cambiamento di scala. Senza queste proprietà un punto caratteristico non riuscirà a essere seguito per molti frame. Per ovviare a questa mancanza sono state studiate varie soluzioni, una delle quali è il *warping* delle patch, ossia la loro trasformazione affine che approssima bene l'effetto del cambiamento di prospettiva.

3.2.1 Patch warping

Un punto caratteristico di una immagine corrisponde a un punto fisico nel mondo in direzione del raggio luminoso che lo ha formato. Si supponga che, nell'istante in cui il punto caratteristico viene inizializzato, il suo corrispondente fisico sia parte di una regione piana perpendicolare al raggio di vista.

L'operazione di *patch warping* consiste nel deformare la patch iniziale secondo una opportuna trasformazione affine. Tale operazione è in grado di approssimare abbastanza bene l'effetto del cambio di prospettiva fra due frame, sempre che l'assunzione di planarità sussista. Un esempio di tale trasformazione è dato in Figura 3.3.

Una trasformazione affine è descritta dalla seguente formula:

$$\mathbf{x}_w = A\mathbf{x} + \mathbf{t} \qquad \text{con} \quad A \in \mathbb{R}^{2\times 2} \quad \text{e} \quad \mathbf{t} \in \mathbb{R}^{2\times 1}$$

dove $\mathbf{x}$ sono le coordinate di un punto della patch originale e $\mathbf{x}_w$ quelle del corrispondente punto della patch trasformata.

Assumiamo che il sistema di riferimnto sia posto nel punto centrale della patch. Il vettore $\mathbf{t}$ corrisponde alla traslazione del punto centrale della patch, ossia del punto caratteristico associato a essa. La matrice A interviene invece sulla forma della patch, alterando angoli e distanze ma preservando le linee rette.

Esempi comuni di trasformazioni affini sono:

- Cambio di scala: $A = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$
- Rotazione: $A = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ con θ angolo di rotazione
- Traslazione
- Riflessione

In generale non esiste nessun vincolo sui valori della matrice A, per cui le trasformazioni di cui sopra si possano comporre fra loro.

Figura 3.4: *Esempio di trasformazione omografica di un'immagine*

Nel famoso articolo *Good features to track* di Shi e Tomasi [ST94] oltre a definire un nuovo metodo per rilevare i punti caratteristici di una immagine[1], viene proposto un algoritmo di tracking che utilizza il patch warping. Viene definita una funzione di dissimilarità fra patch molto più generale delle consuete:

$$\epsilon = \iint_W [I_2(A\mathbf{x} + \mathbf{t}) - I_1(\mathbf{x})]^2 w(\mathbf{x}) \mathrm{d}\mathbf{x}$$

dove $\iint_W$ indica la somma di tutti i contributi dei pixel della patch, $I_1(\mathbf{p})$ e $I_2(\mathbf{p})$ restituiscono l'intensità del pixel $\mathbf{p}$ nella prima e seconda immagine rispettivamente, le matrici $A \in \mathbb{R}^{2\times2}$ e $\mathbf{t} \in \mathbb{R}^{2\times1}$ sono la trasformazione affine applicata ad ogni pixel. $w(\mathbf{x})$ è una funzione che pesa l'importanza del pixel all'interno della patch, generalmente definita come funzione costante 1, ma sostituibile con una funzione Gaussiana bivariata per dare più importanza ai pixel centrali della patch. La funzione di dissimilarità viene minimizzata trovando i valori di A e $\mathbf{t}$, utilizzando un metodo iterativo alla *Newton-Raphson*. Il tracker così descritto è il popolare KLT[2](*Kanade-Lucas-Tomasi*), inizialmente sviluppato e descritto in un articolo di Lucas e Kanade, seguito da un articolo di Tomasi e Kanade e ultimato con l'articolo sopra descritto di Shi e Tomasi.

Consideriamo ora che una trasformazione affine è solamente un'approssimazione della vera trasformazione omografica che si avrebbe osservando un piano da due diverse prospettive. (assunzione iniziale). Per esempio, un'omografia, contrariamente a una trasformazione affine, è in grado di trasformare un rettangolo in un "trapezio"(vedi Figura 3.4).

L'utilizzo di un metodo come quello di Shi e Tomasi con trasformazioni omografiche sarebbe computazionalmente proibitivo dato l'elevato numero di parametri della matrice omografica 3×3 da stimare. Fortunatamente in alcuni casi quale lo SLAM, ciò è fattibile: l'algoritmo di SLAM produce una stima della posizione e dell'orientamento della camera per ogni frame della sequenza video; nella fase di inizializzazione di un punto caratteristico, oltre ad attribuirgli una patch come descrittore, gli vengono associate la posizione e l'orientamento della camera in quell'istante; nei frame successivi, tramite la stima dell'attuale posizione e orientamento della camera e alla stima della posizione della feature nel mondo è possibile ricostruire l'aspetto della patch, dato dal cambio di prospettiva subito. Questo metodo verrà ampiamente discusso nel Capitolo 7, a valle dei capitoli sullo SLAM.

Finché la stima della posizione e dell'orientamento della camera rimane affidabile, questo metodo permette di ottenere un miglioramento del matching fra punti caratteristici visti da prospettive

[1] Il metodo di rilevamento dei punti punti caratteristici descritto da Shi e Tomasi è anche implementato nella funzione `goodFeaturesToTrack` della libreria di Computer Vision *OpenCV* (http://opencv.willowgarage.com).

[2] Una versione del KLT tracker (senza patch warping) è disponibile anche in *OpenCV*, con la funzione `calcOpticalFlowPyrLK`

differenti, a un costo computazionale contenuto, inferiore a quello del metodo di Shi e Tomasi. Questa possibilità è stata utilizzata anche da Pietzsch [Pie08b] sebbene non la usi per il tracking di semplici punti, bensì di segmenti di piano.

3.3 Tracking con camere omnidirezionali

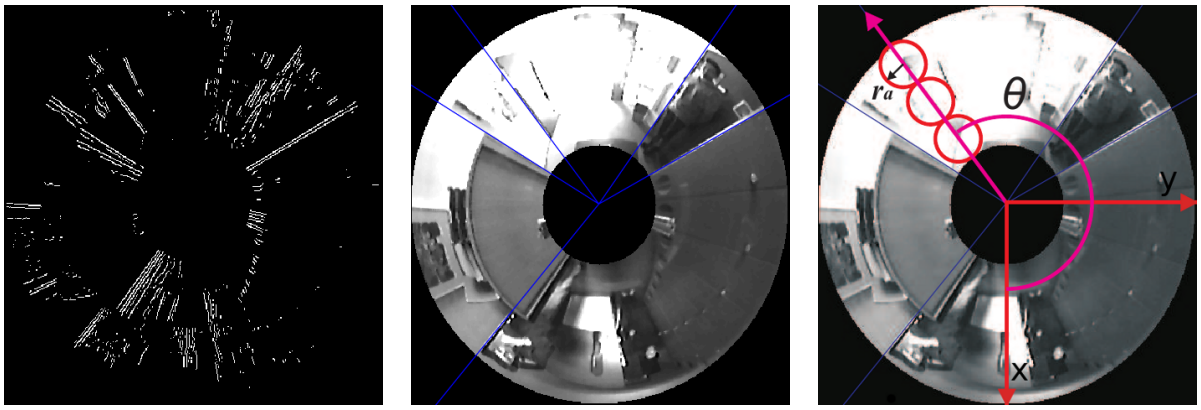
Nei paragrafi precedenti sono stati descritti alcuni metodi per effettuare il tracking dei punti caratteristici in immagini prospettiche. Queste soluzioni fanno uso della regione circostante il punto caratteristico, tanto nel caso delle patch quanto nel caso SIFT (si ricordi che il descrittore SIFT è calcolato sulla base del gradiente dell'area nell'intorno del *keypoint*). Affinché un punto possa essere seguito con successo, la sua area circostante deve mantenere entro certi limiti il proprio aspetto al passare del tempo e al cambio di prospettiva. Questo non accade nelle camere omnidirezionali a causa della loro alta distorsione: la regione circostante un punto subisce infatti forti deformazioni anche a fronte di piccoli cambi di prospettiva. Per tale motivo il tracking con immagini omnidirezionali richiede un riadattamento dei metodi presentati in precedenza per il caso prospettico.

Scaramuzza [SCMS07] propone di fare il tracking di linee verticali, molto presenti in ambienti strutturati, assumendo che la camera si muova su un piano perfettamente orizzontale e che la sua rotazione possa avvenire solo intorno all'asse perpendicolare al piano. In questo caso le linee verticali del mondo vengono mappate in linee radiali nell'immagine omnidirezionale. Queste linee radiali vengono estratte con una successione di calcolo dei gradienti, esclusione dei punti il cui gradiente non punta verso il centro dell'immagine e soppressione dei punti il cui gradiente non è un massimo locale. A questo punto l'immagine viene suddivisa radialmente in 720 settori (0,5 gradi ciascuna) e viene costruito un istogramma, contando il numero di pixel diversi da 0 in ogni settore. I settori con un numero sufficiente di pixel saranno le feature da seguire. Per fare il matching di queste linee radiali è necessario associargli dei descrittori. Scaramuzza sviluppa un descrittore analogo a quello di SIFT: crea tre regioni circolari nell'immagine omnidirezionale di ugual dimensione lungo la direzione della linea radiale; ognuna di queste regioni viene a sua volta suddivisa in una parte sinistra e destra dalla linea radiale stessa; viene calcolato il gradiente su queste aree, e creati i relativi istogrammi (30 orientazioni possibili ciascuno); il descrittore è dunque composto dai valori di tutti gli istogrammi, quindi di dimensione $3\ \textit{aree} \times 2\ \textit{lati} \times 30\ \textit{orientazioni} = 180\ \textit{elementi}$. L'invarianza rotazionale viene raggiunta ruotando tutti i gradienti delle aree del descrittore, dell'angolo dato dalla direzione della linea radiale.

Le operazioni per la rilevazione delle feature radiali e il calcolo del relativo descrittore sono mostrate in Figura 3.5

Hesch [HT05] come Scaramuzza esegue il tracking di linee verticali per fare SLAM. Per la loro estrazione, l'immagine omnidirezionale viene trasformata in panoramica e successivamente applicati gli stessi metodi usati da Scaramuzza. Non viene però creato nessun descrittore, in quanto il matching di una linea viene integrato direttamente nel processo di SLAM.

Hansen [HCBD07] adatta invece l'algoritmo di SIFT, sviluppato da Lowe per il caso prospettico, per funzionare con immagini omnidirezionali. Per trovare i *keypoint* nello spazio delle scale, non ha più senso effettuare la convoluzione dell'immagine con un filtro Gaussiano: la corretta operazione da fare è la convoluzione con la soluzione dell'equazione di diffusione sferica. Hansen effettua questa operazione nello spazio delle frequenze, usando la trasformata di Fourier sferica (DSFT, *Discrete Spherical Fourier Transform*). Il descrittore "SIFT" non è più calcolato sulla base dei gradienti di in area circolare intorno al *keypoint* nell'immagine, bensì su un'area circolare sulla superficie della sfera di raggio unitario sulla quale l'immagine viene proiettata.



(a) *Gradiente dell'immagine dopo la soppressione dei non minimi e massimi locali, nonché dei punti il cui gradiente non punta verso il centro dell'immagine*

(b) *Feature radiali rilevate, corrispondenti a linee verticali nel mondo*

(c) *Calcolo del descrittore a partire dagli istogrammi del gradiente nelle tre aree circolari*

Figura 3.5: *Processo di creazione del descrittore di Scaramuzza per linee radiali in immagini omnidirezionali*

3.4 Considerazioni finali sul tracking di feature

Nel corso del capitolo sono stati presentati alcuni metodi per effettuare tracking, ognuno dei quali presenta vantaggi e svantaggi, dipendendo dal tipo di camera utilizzato e dell'elaborazione ad alto livello che si intende fare con i risultati del tracking.

Il descrittore SIFT proposto da Lowe si applica a immagini prospettiche, e ha il vantaggio di descrivere efficacemente un punto caratteristico, risultando inoltre invariante a scala e rotazione. Il tracking con patch è un metodo molto semplice e intuitivo, che non gode però delle proprietà di invarianza a scala e rotazione. Il suo costo computazionale è nettamente inferiore a quello di SIFT.

Per ovviare alla mancanza della proprietà di invarianza a scala e rotazione del patch tracking, è stato introdotto il *patch warping*, che deforma le patch secondo una trasformazione affine ottenendo non solo l'invarianza a scala e rotazione, ma anche un miglioramento nel cambio di prospettiva. È importante osservare che la trasformazione affine applicata a una patch è frutto di una minimizzazione che premia la somiglianza fra tale trasformazione e una regione dell'immagine attuale: questa somiglianza potrebbe risultare "forzata" dall'ottimizzazione dei parametri e non davvero presente nella realtà. Il *patch warping* come descritto da Shi e Tomasi non è stato preso da noi in considerazione, in quanto nel nostro lavoro di SLAM abbiamo a disposizione una stima di posizione e orientamento della camera e di alcuni punti nel mondo, che ci permette di calcolare l'apparenza di una patch nei frame successivi, sotto l'ipotesi che tale patch sia frutto di una regione del mondo planare e perpendicolare alla camera quando vista per la prima volta. Questo secondo metodo ha sicuramente un costo computazionale molto ridotto rispetto al primo, e non soffre del problema della somiglianza "forzata" accennato in precedenza.

L'assunzione di planarità fatta non è quasi mai soddisfatta, in quanto un punto caratteristico corrisponde generalmente a spigoli del mondo e in tutti i casi la perpendicolarità rispetto alla camera è totalmente ingiustificabile. Per onor di cronaca, altri lavori come quello di Pietzsch [Pie08b] o Molton e Davison[MDR04] hanno tentato di rilassare il vincolo della perpendicolarità enunciato, stimando l'orientamento del piano della patch. Il costo computazionale di una tale soluzione è chiaramente maggiore e i risultati ottenuti non si discostano significativamente dalla versione perpendicolare.

I metodi di tracking sin qui discussi si applicano a immagini catturate per mezzo di camere prospettiche e non sono applicabili direttamente nel caso omnidirezionale. A questo scopo Hansen ha sviluppato il corrispondente di SIFT per immagini omnidirezionali, ma esso continua a soffrire degli

stessi problemi elencati per SIFT nel caso prospettico. Scaramuzza esegue invece tracking di feature radiali. Nel nostro lavoro di tesi sono state invece riadattate le tecniche di patch tracking, a causa della loro maggior efficienza e qualità di tracking prodotto. Queste soluzioni saranno presentate nel Capitolo 7.

Capitolo 4

Extended Kalman Filter

Il filtro di Kalman esteso è un potente oggetto matematico che permette di stimare statisticamente lo stato di un sistema dinamico sotto particolari condizioni di linearità delle equazioni che lo governano. Il filtro di Kalman è stato inventato da Kalman nel 1960, ma a causa della sua complessità solo con l'avanzare della tecnologia e della potenza di calcolo fu possibile sfruttarlo nelle applicazioni di *Computer Vision*. Nella letteratura vi sono innumerevoli presentazioni del filtro di Kalmam. Questo capitolo è stato redatto sulla base di due report: il primo scritto da Welch e Bishop [WB06], descrive in poche pagine il filtro di Kalman e le sue equazioni; il secondo è scritto da Ribeiro [Rib04] e offre una trattazione più ampia e orientata alla statistica, che svela buona parte dei segreti che stanno alla base della scatola nera che è il filtro di Kalman. Nella Sezione 4.1 viene presentato il problema generico del filtraggio. Nella Sezione 4.2 viene descritto il filtro di Kalman, che permette di ottenere la miglior stima possibile dello stato di un sistema, quando questo è lineare nella sua dinamica interna e nelle equazioni che ne permettono la sua osservazione. Nella Sezione 4.3 viene rilassato il vincolo di linearità che impone il filtro di Kalman, presentandone la sua versione estesa che sostanzialmente linearizza il sistema in questione. A causa di questa approssimazione il filtro di Kalman esteso non garantisce l'ottimalità della soluzione. Infine nella Sezione 4.4 vengono presentate brevemente alcune alternative all'uso del filtro esteso di Kalman.

4.1 Il filtraggio

L'ambiente naturale nel quale viene impiegato un generico filtro è composto da un sistema tempovariante il cui stato non è direttamente misurabile. Il sistema riceve in ingresso i segnali di controllo e un "rumore" che descrive l'incertezza sul modello del sistema utilizzato. Il sistema è misurabile indirettamente attraverso sensori che a loro volta introducono un'errore di misurazione. Un generico filtro si occupa di stimare lo stato del sistema, a partire dalle sue imprecise misurazioni, l'incertezza sul modello e il segnale di controllo in ingresso.

D'ora in poi restringeremo il campo ai sistemi a tempo discreto, in quanto l'obiettivo finale sarà definire un metodo codificabile come algoritmo ed eseguibile su un elaboratore. Una trattazione analoga esiste per il caso a tempo continuo. Il sistema in oggetto può essere descritto matematicamente come:

$$\begin{aligned} \mathbf{x}_k &= f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{w}_{k-1}) \\ \mathbf{y}_k &= h(\mathbf{x}_k, \mathbf{v}_k) \end{aligned} \tag{4.1}$$

dove:

- $\mathbf{x}_k$: stato del sistema
- $\mathbf{u}_k$: segnale di controllo in ingresso al sistema
- $\mathbf{w}_k$: rumore in ingresso al sistema che ne definisce l'incertezza sul suo modello

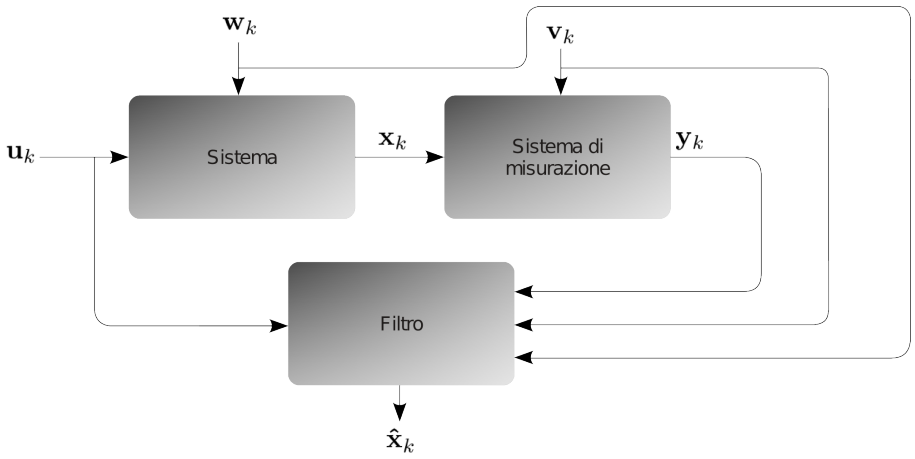


Figura 4.1: *Schema a blocchi dell'applicazione di un generico filtro per la stima dello stato di un sistema dinamico*

- $f(\cdot)$: dinamica del sistema
- $\mathbf{v}_k$: errore di misurazione
- $h(\cdot)$: equazione di misurazione del sistema
- $\mathbf{y}_k$: misura

Tutte le variabili sono espresse in funzione di k per specificare che il loro valore è riferito al tempo k.

Introduciamo la notazione $\mathbf{S}_i^j$ per indicare l'insieme dei valori di $\mathbf{S}$ dal tempo i al tempo j, dove $\mathbf{S}$ è una generica variabile:

$$\mathbf{S}_i^j = \{\mathbf{S}_i, \mathbf{S}_{i+1}, \mathbf{S}_{i+2}, \ldots, \mathbf{S}_{j-1}, \mathbf{S}_j\} \quad \text{con} \quad i \leq j \quad \text{e} \quad i, j \in \mathbb{N}$$

Un generico filtro quindi, data la conoscenza della dinamica del sistema f, dell'equazione di misurazione h, della tipologia di rumore $\mathbf{w}$, degli ingressi $\mathbf{u}_0^{k-1}$, delle misure $\mathbf{y}_1^k$ e della tipologia dell'errore di misurazione $\mathbf{v}$, deve essere in grado di ottenere la miglior stima $\hat{\mathbf{x}}_k$ dello stato del sistema $\mathbf{x}_k$ al tempo k. Lo schema a blocchi dell'applicazione di un generico filtro è mostrata in Figura 4.1.

Da un punto di vista probabilistico un filtro propaga la distribuzione di probabilità dello stato del sistema condizionandola ripetutamente con nuove osservazioni. La stima dello stato del sistema dipenderà dalla statistica che si desidera utilizzare fra le seguenti:

- *media*: centro di massa della probabilità, che corrisponde al alla stima del minimo errore quadratico medio MMSE (*minimum mean square error*)
- *moda*: valore con la probabilità più alta, corrispondente alla stima per massimo a posteriori MAP (maximum a posterior)
- *mediana*: valore che lascia metà dell'intera probabilità a sinistra e l'altra metà a destra di se stesso

4.2 Il filtro di Kalman

Partendo dal modello di filtro generico della sezione precedente, definiamo i seguenti vincoli sul sistema e sulle variabili che interagiscono con esso:

- l'equazione della dinamica del sistema f e la funzione di misurazione h sono lineari;

- lo stato iniziale del sistema $\mathbf{x}_0$ è una variabile Gaussiana;
- il rumore in ingresso al sistema $\mathbf{w}$ e l'errore di misurazione $\mathbf{v}$ sono variabili Gaussiane a media nulla;

Con il sistema vincolato in questo modo, anche la variabile $\mathbf{x}_k$ che rappresenta lo stato del sistema e la misura $\mathbf{y}_k$ sono variabili Gaussiane. Una distribuzione Gaussiana è completamente determinata dalla sua media e matrice di covarianza. In questo caso sarà quindi sufficiente propagare nel tempo attraverso il filtro queste due statistiche e non l'intera distribuzione di probabilità.

Sotto queste ipotesi il filtro viene chiamato filtro di Kalman e la stima dello stato del sistema avviene con l'approccio MMSE, stimatore corretto e a minima varianza. Come enunciato in precedenza, l'uso di MMSE, minimo errore quadratico medio, corrisponderebbe a prendere come stima la media della distribuzione, ma si ricordi che nel caso Gaussiano media, moda e mediana coincidono.

Il le equazioni del sistema generale 4.1 possono essere riscritte per il caso lineare nel seguente modo:

$$\begin{aligned} \mathbf{x}_k &= A_{k-1}\mathbf{x}_{k-1} + B_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \\ \mathbf{y}_k &= H_k\mathbf{x}_k + \mathbf{v}_k \end{aligned} \tag{4.2}$$

dove:

- $\mathbf{x}_k \sim \mathcal{N}(\overline{\mathbf{x}}_k, P_k)$ con $\overline{\mathbf{x}}_k \in \mathbb{R}^n$ e $P_k \in \mathbb{R}^{n\times n}$: stato del sistema descritto dal vettore di stato e dalla matrice di covarianza.
- $A_k \in \mathbb{R}^{n\times n}$: matrice della dinamica del sistema in assenza di rumore e ingressi
- $\mathbf{w}_k \sim \mathcal{N}(0, Q_k)$ con $Q_k \in \mathbb{R}^{n\times n}$: rumore Gaussiano a media nulla applicato al sistema
- $\mathbf{u}_k \in \mathbb{R}^m$: ingresso deterministico del sistema
- $B_k \in \mathbb{R}^{n\times m}$: matrice associata all'ingresso del sistema che specifica come questo modifica lo stato del sistema
- $\mathbf{y}_k \sim \mathcal{N}(\overline{\mathbf{y}}_k, S_k)$ con $\overline{\mathbf{y}}_k \in \mathbb{R}^r$ e $S_k \in \mathbb{R}^{r\times r}$: osservazione del sistema (misura)
- $H_k \in \mathbb{R}^{r\times n}$: matrice che definisce la relazione fra l'osservazione del sistema e il suo stato, in assenza di errore di misurazione
- $\mathbf{v}_k \sim \mathcal{N}(0, R_k)$ con $R_k \in \mathbb{R}^{r\times r}$: errore di misurazione Gaussiano a media nulla

I rumori Gaussiani $\mathbf{w}_k$ e $\mathbf{v}_k$ sono indipendenti e non correlati fra loro. Lo stato iniziale del sistema $\mathbf{x}_0$ sarà un vettore Gaussiano con media $\overline{\mathbf{x}}_0$ e covarianza P_0.

Uno dei grandi vantaggi del filtro di Kalman è che, per stimare lo stato del sistema $\hat{\mathbf{x}}_k$ al tempo k, sono necessarie e sufficienti la stima al passo precedente $\hat{\mathbf{x}}_{k-1}$, le nuove osservazioni $\mathbf{y}_k$ e l'eventuale ingresso $\mathbf{u}_{k-1}$[1]. Ad ogni istante discreto di tempo k il filtro di Kalman esegue essenzialmente due operazioni per ottenere una nuova stima:

1. *Predizione*: sulla base della stima $\hat{\mathbf{x}}_{k-1}$ allo al tempo $k-1$ e della conoscenza della dinamica del sistema, viene calcolata una stima a priori del nuovo stato del sistema, che indicheremo con $\hat{\mathbf{x}}_k^-$. La predizione avviene senza usufruire delle nuove osservazioni.
2. *Aggiornamento*: la predizione ottenuta viene integrata con le nuove osservazioni fatte $\mathbf{y}_k$, ottenendo una stima a posteriori sullo stato del sistema $\hat{\mathbf{x}}_k$

La predizione del nuovo stato del sistema è facilmente calcolabile sulla base delle equazioni del sistema stesso:

$$\hat{\mathbf{x}}_k^- = A_{k-1}\hat{\mathbf{x}}_{k-1} + B_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \quad \Longrightarrow \quad \begin{cases} \hat{\overline{\mathbf{x}}}_k^- &= A_{k-1}\hat{\overline{\mathbf{x}}}_{k-1} + B_{k-1}\mathbf{u}_{k-1} \\ \hat{P}_k^- &= A_{k-1}\hat{P}_{k-1}A_{k-1}^T + Q_{k-1} \end{cases}. \tag{4.3}$$

[1] Si lascia sottintesa la dipendenza della stima $\hat{\mathbf{x}}_k$ dai rumori $\mathbf{w}$ e $\mathbf{v}$.

L'aggiornamento del filtro per prima cosa calcola una stima delle osservazioni sulla base della predizione dello stato:

$$\hat{\mathbf{y}}_k = H_k \hat{\mathbf{x}}_k^- + \mathbf{v}_k \qquad \Longrightarrow \qquad \begin{cases} \hat{\bar{\mathbf{y}}}_k &= H_k \hat{\bar{\mathbf{x}}}_k^- \\ \hat{S}_k &= H_k \hat{P}_k^- H_k^T + R_k \end{cases}.$$

Definiamo il termine $\tilde{\mathbf{y}}_k$ come la differenza fra le osservazioni e la loro stima, chiamato innovazione:

$$\tilde{\mathbf{y}}_k = \mathbf{y}_k - \hat{\mathbf{y}}_k.$$

Ragionamenti probabilistici dimostrano che la stima dello stato del sistema $\hat{\mathbf{x}}_k$ dipende solamente della sua predizione $\hat{\mathbf{x}}_k^-$ e dall'innovazione introdotta $\tilde{\mathbf{y}}_k$ secondo la formula:

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + K_k \tilde{\mathbf{y}}_k \quad \Longrightarrow \quad \begin{cases} \hat{\bar{\mathbf{x}}}_k = \hat{\bar{\mathbf{x}}}_k^- + K_k \tilde{\bar{\mathbf{y}}}_k = \hat{\bar{\mathbf{x}}}_k^- + K_k(\bar{\mathbf{y}}_k - H_k \hat{\bar{\mathbf{x}}}_k^-) \\ \hat{P}_k = \hat{P}_k^- + K_k(-H_k \hat{P}_k^-) = (\mathbf{I} - K_k H_k)\hat{P}_k^- \end{cases} \tag{4.4}$$

dove K_k è chiamato guadagno del filtro ed è calcolato come:

$$K_k = \hat{P}_k^- H_k^T \hat{S}_k^{-1} = \hat{P}_k^- H_k^T [H_k \hat{P}_k^- H_k^T + R_k]^{-1}.$$

Il guadagno così definito permette di ottenere la miglior stima possibile dello stato del sistema. Come si può notare, al diminuire di $\hat{P}_k^-$ (incertezza della predizione), anche il valore di K_k diminuirà, dando più credito alla predizione fatta anziché alle osservazioni. Viceversa, al diminuire di R_k (errore di misurazione) diminuisce $\hat{S}_k$ e quindi aumenterà K_k, dando maggior credito alle osservazioni piuttosto che alla predizione.

4.3 Il filtro di Kalman esteso

Nel caso in cui la dinamica del sistema e le equazioni di misurazione non fossero lineari, la distribuzione di probabilità propagata dal filtro non sarebbe Gaussiana. In questo caso media e varianza non descriverebbero più l'intera distribuzione, e il filtro di Kalman non sarebbe più un opzione valida. Il filtro ottimale, non lineare, che propaga una generica distribuzione in molti casi non è applicabile, in quanto il suo costo computazionale diventa intrattabile. Il filtro esteso di Kalman, sotto alcune ipotesi di linearità locale delle funzioni f e h, offre una valida approssimazione.

Il filtro esteso di Kalman EKF (*Extended Kalman Filter*) sostanzialmente applica il filtro di Kalman del caso lineare a una versione del sistema non lineare che viene linearizzata a ogni instante di tempo discreto, sulla base della stima al passo precedente.

Il sistema non lineare sul quale il filtro di Kalman esteso opera è quello del caso generale 4.1, qui ripetuto per chiarezza:

$$\begin{aligned} \mathbf{x}_k &= f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{w}_{k-1}) \\ \mathbf{y}_k &= h(\mathbf{x}_k, \mathbf{v}_k) \end{aligned}$$

dove:

- $\mathbf{x}_k$: stato del sistema, vettore aleatorio di cardinalità n
- $\mathbf{u}_k \in \mathbb{R}^m$: ingresso deterministico del sistema
- $\mathbf{w}_k \sim \mathcal{N}(0, Q_k)$ con $Q_k \in \mathbb{R}^{n \times n}$: rumore Gaussiano a media nulla applicato al sistema
- $f(\cdot)$: dinamica non lineare del sistema
- $\mathbf{v}_k \sim \mathcal{N}(0, R_k)$ con $R_k \in \mathbb{R}^{r \times r}$: errore di misurazione Gaussiano a media nulla
- $h(\cdot)$: equazione non lineare di misurazione del sistema

- $\mathbf{y}_k$: osservazione del sistema, vettore aleatorio di cardinalità r

Lo stato iniziale del sistema $\mathbf{x}_0$ è una variabile Gaussiana con media $\overline{\mathbf{x}}_0$ e matrice di covarianza P_0.

Si noti che seppur utilizzando il modello di sistema di un generico filtro, le variabili in gioco sono Gaussiane.

Come già enunciato a causa della non linearità delle funzione f e h, sebbene lo stato iniziale del sistema $\mathbf{x}_0$ sia Gaussiano, tutti i suoi stati successivi, insieme alle sue misurazioni non saranno Gaussiani. Il filtro di Kalman esteso si ostina ad approssimare queste generiche distribuzioni come delle Gaussiane: per l'approssimazione della media (stima puntuale), la non linearità non pone nessun problema, mentre per estrapolare le matrici di covarianza è necessario linearizzare il sistema.

$$\begin{aligned} \hat{\overline{\mathbf{x}}}_k &= f(\hat{\overline{\mathbf{x}}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0}) \\ \hat{\overline{\mathbf{y}}}_k &= h(\hat{\overline{\mathbf{x}}}_k, \mathbf{0}) \end{aligned}$$

Il sistema linearizzato sarà:

$$\begin{aligned} \mathbf{x}_k &\approx \hat{\overline{\mathbf{x}}}_k + F_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}) + W_{k-1}\mathbf{w}_{k-1} \\ \mathbf{y}_k &\approx \hat{\overline{\mathbf{y}}}_k + H_k(\mathbf{x}_k - \hat{\overline{\mathbf{x}}}_k) + V_k\mathbf{v}_k \end{aligned}$$

dove:

- $\mathbf{x}_k \sim \mathcal{N}(\overline{\mathbf{x}}_k, P_k)$ con $\overline{\mathbf{x}}_k \in \mathbb{R}^n$ e $P_k \in \mathbb{R}^{n\times n}$: stato del sistema;
- $\hat{\overline{\mathbf{x}}}_k \in \mathbb{R}^n$: stima puntuale del sistema;
- $\hat{\mathbf{x}}_{k-1} \sim \mathcal{N}(\hat{\overline{\mathbf{x}}}_{k-1}, \hat{P}_{k-1})$ con $\hat{\overline{\mathbf{x}}}_{k-1} \in \mathbb{R}^n$ e $\hat{P}_{k-1} \in \mathbb{R}^{n\times n}$: stima dello stato precedente del sistema;
- $F_{k-1} = \frac{\partial f}{\partial \mathbf{x}_{k-1}}\Big|_{\hat{\overline{\mathbf{x}}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0}}$: Jacobiano della funzione f rispetto a $\mathbf{x}_{k-1}$, valutato alla stima puntuale dello stato precedente, all'ingresso ricevuto e in assenza di errore;
- $\mathbf{w}_{k-1} \sim \mathcal{N}(0, Q_{k-1})$ con $Q_{k-1} \in \mathbb{R}^{n\times n}$: rumore Gaussiano a media nulla applicato al sistema;
- $W_{k-1} = \frac{\partial f}{\partial \mathbf{w}_{k-1}}\Big|_{\hat{\overline{\mathbf{x}}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0}}$: Jacobiano della funzione f rispetto al rumore in ingresso w;
- $\mathbf{y}_k \sim \mathcal{N}(\overline{\mathbf{y}}_k, S_k)$ con $\overline{\mathbf{y}}_k \in \mathbb{R}^r$ e $S_k \in \mathbb{R}^{r\times r}$: osservazioni del sistema;
- $\hat{\overline{\mathbf{y}}}_k \in \mathbb{R}^r$: stima puntuale delle osservazioni;
- $H_k = \frac{\partial h}{\partial \mathbf{x}_k}\Big|_{\hat{\overline{\mathbf{x}}}_k, \mathbf{0}}$: Jacobiano della funzione di misurazione h rispetto a $\mathbf{x}_k$, valutato alla stima puntuale del sistema al tempo k e in assenza di errore di misurazione;
- $\mathbf{v}_k \sim \mathcal{N}(0, R_k)$ con $R_k \in \mathbb{R}^{r\times r}$: errore di misurazione Gaussiano a media nulla;
- $V_k = \frac{\partial h}{\partial \mathbf{v}_k}\Big|_{\hat{\overline{\mathbf{x}}}_k, \mathbf{0}}$: Jacobiano della funzione di misurazione h rispetto al suo errore Gaussiano v;

In queste condizioni possiamo riscrivere le equazioni per la predizione e l'aggiornamento del filtro come abbiamo fatto con il filtro di Kalman. La predizione del nuovo stato del sistema $\hat{\mathbf{x}}_k^-$ sarà quindi calcolata come:

$$\hat{\mathbf{x}}_k^- = \begin{cases} \hat{\overline{\mathbf{x}}}_k^- &= f(\hat{\overline{\mathbf{x}}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0}) \\ \hat{P}_k^- &= F_{k-1}\hat{P}_{k-1}F_{k-1}^T + W_{k-1}Q_{k-1}W_{k-1}^T \end{cases}$$

dove la media è stata calcolata sul sistema non lineare originale, mentre la matrice di covarianza sul corrispondente sistema linearizzato.

L'aggiornamento sarà invece calcolato come:

$$\hat{\mathbf{x}} = \begin{cases} \hat{\overline{\mathbf{x}}}_k &= \hat{\overline{\mathbf{x}}}_k^- + K_k(\mathbf{y}_k - \hat{\overline{\mathbf{y}}}_k) \\ \hat{P}_k &= (\mathbf{I} - K_kH_k)\hat{P}_k^- \end{cases}$$

dove K_k sarà calcolata come:

$$K_k = \hat{P}_k^- H_k^T [H_k \hat{P}_k^- H_k^T + V_k R_k V_k^T]^{-1}.$$

A causa della linearizzazione effettuata, il filtro di Kalman esteso non produce la miglior stima dello stato del sistema possibile. La sua qualità dipenderà da quanto bene la versione linearizzata del sistema approssimerà quella originale. Se l'approssimazione non fosse sufficiente buona, il filtro di Kalman esteso potrebbe anche divergere.

Nel calcolo di H_k e V_k, gli Jacobiani vengono valutati nella miglior stima dello stato disponibile al tempo k, ossia la sua predizione $\hat{\hat{\mathbf{x}}}_k^-$. A valle dell'aggiornamento del filtro si ottiene una stima migliore e si potrebbero quindi calcolare nuovamente H_k e V_k ed eseguire un nuovo aggiornamento. Questo processo potrebbe essere ripetuto più volte, finché il miglioramento in termini di covarianza della stima non scende sotto una certa soglia. Questa versione iterativa del filtro esteso di Kalman prende il nome di *Iterated Extended Kalman Filter* (IEKF).

4.4 Alternative all'EKF

Presentiamo brevemente due filtri che prendono spunto dall'EKF e permettono di ottenere una stima migliore dello stato del sistema non lineare, a discapito di un costo computazionale maggiore.

4.4.1 Unscented Kalman Filter

Consideriamo nuovamente il sistema non lineare 4.1. Il filtro di Kalman esteso considera la distribuzione di $\mathbf{x}_{k-1}$ Gaussiana e per poterla propagare attraverso le funzioni non lineari f e h mantenendone la linearità, le approssima con la loro versione linearizzata.

L'idea alla base dell'UKF (*Unscented Kalman Filter*) è quella di approssimare la distribuzione di $\mathbf{x}_{k-1}$ con una combinazione lineare di impulsi pesati, che la descrivano al meglio. Tali impulsi possono essere propagati direttamente attraverso le funzioni non lineari f e h, ottenendo così una migliore approssimazione della distribuzione di $\mathbf{x}_k$ e $\mathbf{y}_k$.

Ipotizzando che la distribuzione di $\mathbf{x}_{k-1}$ sia Gaussiana, abbiamo bisogno di una sua approssimazione con una serie di impulsi. È dimostrabile che una Gaussiana multivariata in m dimensioni può essere approssimata con $2m+1$ impulsi, uno centrato nella media con peso w_0 e i rimanenti disposti lungo gli assi dell'ellisse di covarianza[2] con pesi uguali (vedi Figura 4.2).

I $2m+1$ impulsi vengono trasformati con la funzione f per approssimare la distribuzione di $\mathbf{x}_k^-$. Gli impulsi trasformati vengono reinterpretati come una Gaussiana calcolando media e covarianza. La distribuzione Gaussiana di $\mathbf{x}_k^-$ ottenuta viene nuovamente approssimata come serie di impulsi e successivamente propagati attraverso la funzione h ottenendo una approssimazione della distribuzione di $\mathbf{y}_k$, che verrà anch'essa reinterpretata come Gaussiana. Con le consuete equazioni di aggiornamento del filtro di Kalman (Equazione 4.4) è quindi possibile ottenere la stima di $\mathbf{x}_k$.

L'*Unscented Kalman Filter* permette di ottenere una precisione del secondo ordine senza calcolare Jacobiani o Hessiani. D'altra parte per poter disporre gli impulsi lungo gli assi dell'ellisse di covarianza è necessario diagonalizzare le matrici P_{k-1} e P_k^- a ogni passo. Alcuni lavori come per esempio quello di Holmes [HKM08] dimostrano che l'UKF ha un costo computazionale di circa un ordine di grandezza maggiore di quello dell'EKF.

[2] Una matrice di covarianza può sempre essere interpretata graficamente come un ellisse n-dimensionale che definisce l'intervallo di confidenza della distribuzione.

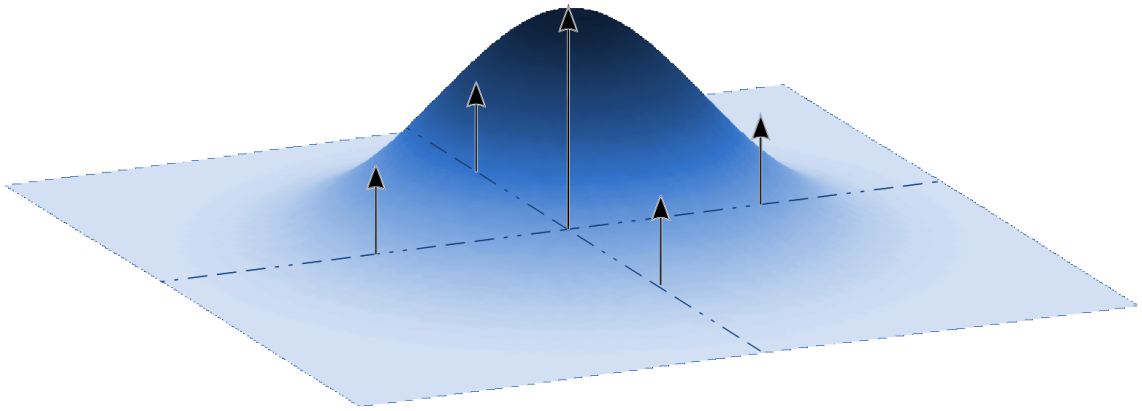


Figura 4.2: *Gaussiana bivariata approssimata da cinque impulsi, sulla quali si basa il funzionamento dell'UKF*

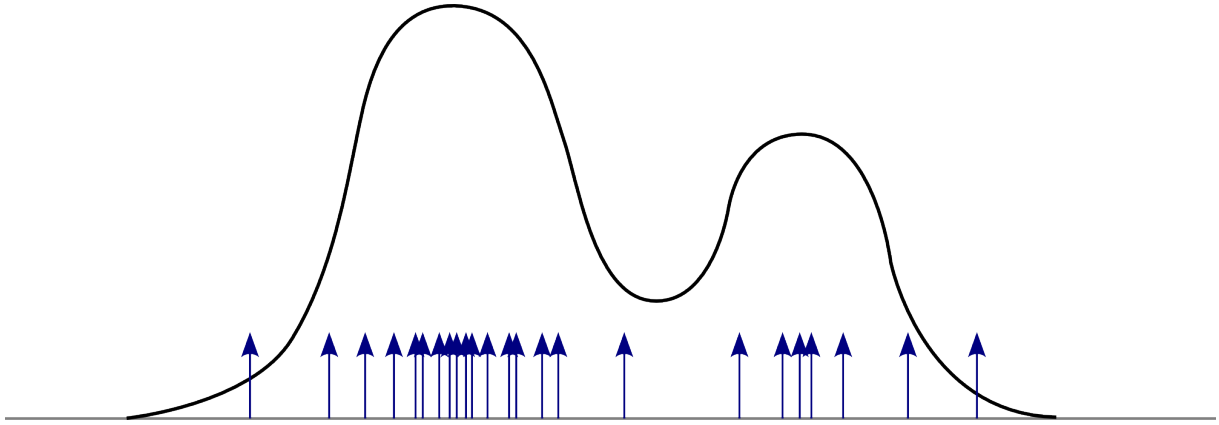


Figura 4.3: *Esempio di campionamento da una distribuzione generica per il filtro particellare*

4.4.2 Filtro particellare

Un filtro particellare estremizza l'idea alla base dell'UKF di approssimare una generica distribuzione con una combinazione lineare di impulsi. Ogni impulso è chiamato particella e viene campionato dalla distribuzione che si vuole approssimare. In questo modo una zona della distribuzione ad alta probabilità avrà più particelle di una a bassa probabilità (Figura 4.3).

Il filtro di particelle, per stimare lo stato del sistema, a ogni istante discreto di tempo k effettua le seguenti operazioni:

1. La generica distribuzione di $\mathbf{x}_{k-1}$ viene campionata probabilisticamente ottenendo un insieme di particelle $\mathbf{x}_{k-1,i}$. Il peso iniziale w_i di ognuna è costante e pari a $1/N$ dove N è il numero di particelle. i è l'indice della particella compreso fra 1 e N.
2. Viene effettuato il passo di predizione, propagando ogni particella attraverso la funzione f, ottenendo $\mathbf{x}^-_{k,i}$.
3. Viene effettuato il passo di aggiornamento, che modifica i pesi delle particelle, fino a qui identici fra loro, sulla base delle osservazioni pervenute:

$$w_i = p(\mathbf{y}_k | \mathbf{x}_k = \mathbf{x}^-_{k,i})$$

Al passo successivo, un nuovo insieme di N particelle con peso uguale fra loro si ottiene attraverso il ricampionamento dell'insieme ottenuto con l'aggiornamento.

Ogni particella può essere interpretata come una ipotesi dello stato del sistema e il suo peso come la sua probabilità di rappresentarne il vero valore.

Il reale vantaggio di un filtro particellare, è la sua applicabilità a qualsiasi sistema dinamico affetto da qualsiasi tipo di rumore. D'altra parte ha come svantaggio un costo computazionale molto alto, spesso proibitivo. Inoltre la fase di ricampionamento introduce un degli errori di approssimazione.

4.5 Considerazioni finali

In presenza di un sistema dinamico lineare affetto da rumore Gaussiano, con equazioni di misurazione lineari, errore di misurazione Gaussiano, distribuzione di probabilità dello stato iniziale del sistema Gaussiana, la scelta obbligata è il filtro di Kalman, che permette di ottenere una stima ottimale dello stato del sistema, secondo il metodo del minimo errore quadratico medio MMSE.

In presenza di non-linearità nelle equazioni della dinamica del sistema o nelle equazioni di misurazione, a parità delle altre condizioni, una soluzione spesso adottata è il filtro di Kalman esteso. La propagazione di una distribuzione Gaussiana attraverso una funzione non lineare, viene fatta in maniera separata per media e covarianza: la media viene calcolata direttamente con l'equazione non lineare, mentre per il calcolo della matrice di covarianza viene utilizzata una sua versione linearizzata. Il filtro di Kalman esteso non è in grado di ottenere la stima ottimale dello stato del sistema e la qualità di tale stima dipende fortemente da quanto bene la versione linearizzata dal sistema approssima quella non lineare.

L'Unscented Kalman Filter adotta un approccio diverso al problema: invece di approssimare le non-linearità del sistema come fa l'EKF, approssima le distribuzioni di probabilità Gaussiane da propagare come combinazione lineare di impulsi, le propaga attraverso le funzioni non lineari e le reinterpreta come Gaussiane. L'UKF presenta una maggior precisione nella stima, al costo di una complessità maggiore.

EKF e UKF possono essere utilizzati con sistemi non lineari affetti da rumori unimodali. Nel caso in cui nel sistema ci fossero delle distribuzioni multi-modali, il loro utilizzo è sconsigliato. Per tale tipo di sistemi si usa invece un filtro particellare, che permette di propagare distribuzioni qualsiasi attraverso funzioni non lineari, al costo di una ancor maggiore complessità.

Questo capitolo è stato introdotto in quanto nel nostro lavoro di SLAM è stato utilizzato il filtro di Kalman esteso per stimare la posizione della camera nel tempo e la posizione di alcuni punti nel mondo, sulla base delle corrispondenze ottenute con il tracking. In un'applicazione di questo tipo, lo stato del sistema conserva al suo interno la rappresentazione dei punti del mondo che costituiscono la mappa. La complessità quadratica dell'operazione di aggiornamento dell'EKF rende questa soluzione poco scalabile: affinché l'applicazione lavori in tempo reale lo stato del sistema può contenere non più alcune centinaia di elementi. Nel Capitolo 5 verrà presentata una versione dell'EKF adattata al particolare problema dello SLAM, che permette di abbattere notevolmente il costo computazionale della soluzione base presentata in questo capitolo.

L'Unscented Kalman Filter e il filtro particellare hanno un costo computazionale molto superiore a quello del filtro esteso di Kalman e per poterli utilizzare nello SLAM è necessario introdurre delle approssimazioni al problema affinché rimanga trattabile.

Capitolo 5

SLAM

Per rendere i robot realmente autonomi, è necessario che essi possano muoversi in ambienti generici senza la supervisione dell'uomo. Lo SLAM (*Simultaneous Localization And Mapping*) si occupa esattamente di questo problema e permette a un robot posto in un ambiente sconosciuto di muoversi ed esplorare tale ambiente, costruendone la mappa e localizzandosi al suo interno. Per svolgere questo compito, il robot viene dotato di sensori che gli permettono di osservare il mondo circostante. Tali sensori possono essere di vario genere: camere, sonar, range laser ecc. Con essi il robot individua alcuni punti o regioni del mondo chiamati landmark naturali, o semplicemente landmark, i quali si suppongono essere statici. Attraverso la loro osservazione nel tempo, il robot deduce il proprio movimento e la propria posizione. L'insieme dei landmark usati forma la mappa. In Figura 5.1 è mostrata una rappresentazione grafica del problema dello SLAM.

Inizialmente i sensori utilizzati per fare SLAM erano più che altro telemetri laser(*laser rangefinder*) e sonar, solo ultimamente si è dato molto credito all'uso di camere, anche come unico sensore. Il loro utilizzo presenta alcuni vantaggi: una camera permette di ottenere una rappresentazione del mondo ricca di dettagli, è molto più economica di sonar o laser e ha un consumo e un peso ridotti. Lo SLAM che fa uso di camere come sensori principali è chiamato *Visual SLAM* e nel qual caso la camera fosse una sola, viene denominato *Monocular SLAM*.

Spesso, ove disponibile, si usa anche l'informazione data dai sensori che misurano il moto del robot: accelerazione, velocità, angolo di sterzo ecc., chiamata generalmente odometria.

Il problema dello SLAM può essere ormai considerato un problema chiuso dal punto di vista teorico; ciò che resta da fare è ideare metodi implementativi più efficienti e robusti, che possano essere utilizzati in ambienti sempre più generici, su larga scala e utilizzando sensori a basso costo.

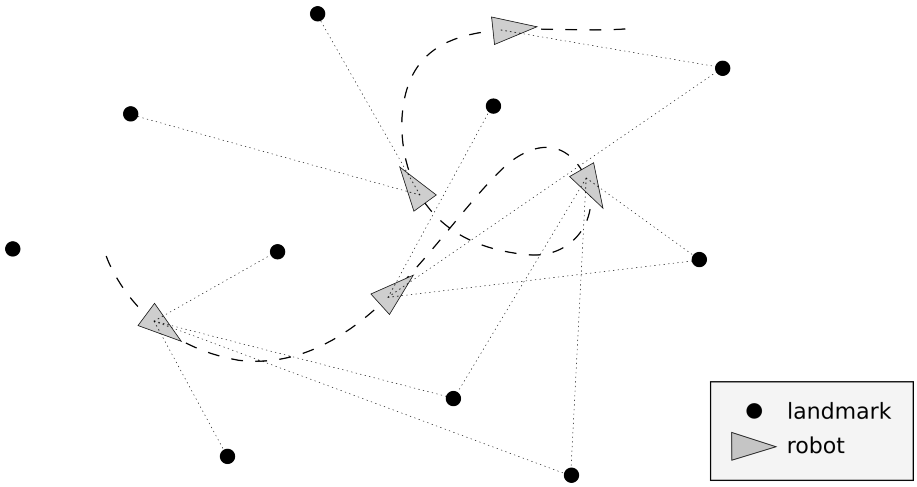


Figura 5.1: *Il problema dello SLAM: stimare simultaneamente la posizione dei landmark e del robot.*

Alcuni di questi sensori non permettono di stimare completamente la posizione di un landmark alla sua prima osservazione: un sonar rileva la sua distanza, ma la sua direzione è molto imprecisa; al contrario una camera rileva con alta precisione la direzione, ma non ha alcuna informazione sulla distanza. A causa di questa osservabilità parziale, quando si utilizza uno di questi sensori come unico strumento di misura, la fase di inizializzazione di nuovi landmark risulta essere molto critica.

Le due tecniche principali utilizzate per fare SLAM sono l'EKF e il filtro particellare. La prima alternativa è quella da noi presa come riferimento e verrà ampiamente discussa nella Sezione 5.2. Una implementazione molto famosa dello SLAM con filtro particellare, chiamata *FastSLAM*, è stata invece introdotta da Montemerlo nel 2002. Nella rappresentazione classica dello SLAM, come utilizzata dell'EKF-SLAM, il problema viene espresso nello spazio di stato. In queste condizioni, l'applicazione diretta del filtro particellare, in sostituzione dell'EKF, non è computazionalmente realizzabile a causa dell'alto numero di dimensioni presenti nel sistema. In FastSLAM, viene utilizzato un filtro particellare *Rao-Blackwellized* nel quale ogni particella rappresenta una traiettoria del robot e non l'intera mappa congiuntamente all'ultima posizione del robot. Questa codifica sfrutta il fatto che i landmark della mappa sono condizionalmente indipendenti fra loro se la traiettoria del robot è conosciuta con precisione. In questo scenario la mappa associata a ogni particella è un semplice insieme di variabili gaussiane indipendenti, con complessità lineare anziché quadratica come nel caso dell'EKF.

Deans [DH00], applica la tecnica del *bundle adjustment*, molto utilizzata nell'ambito SFM (*Structure From Motion*), al problema dello SLAM e la confronta con la soluzione classica dell'EKF. Il *bundle adjustment* è un metodo di minimizzazione non lineare, con un alto costo computazionale, che ottimizza la stima della posizione corrente del robot congiuntamente alla mappa utilizzando tutte le osservazioni accumulate dall'inizio della sessione di SLAM. Per questo motivo l'esecuzione dell'algoritmo a ogni istante discreto di tempo sarebbe impensabile, il che rende la soluzione non applicabile in tempo reale. Anche considerando un tipo di esecuzione *batch*, la dimensione del problema rimane comunque intrattabile in quanto col passare del tempo il numero di landmark aumenta monotonicamente. La soluzione adottata da Deans è quella di dividere la sequenza di immagini in sottosequenze di dimensione fissa e applicare il *bundle adjustment* a ognuna di esse.

Vari autori fra cui Bailey e Deans [Dea05], hanno realizzato versioni di SLAM utilizzando l'Unscented Kalman Filter al posto del classico EKF. L'utilizzo dell'UKF riduce gli errori di linearizzazione commessi dall'EKF ma a un costo computazionale di circa un ordine di grandezza superiore.

Nella Sezione 5.1[1] viene descritta brevemente la storia dello SLAM e gli importanti risultati teorici ottenuti dalla comunità scientifica nel tempo che ci permettono di considerare chiuso il problema dello SLAM. La Sezione 5.2 sviluppa il classico metodo implementativo dello SLAM con il filtro di Kalman esteso, con alcuni dettagli implementativi indispensabili per ottenere una soluzione computazionalmente efficiente. Nella Sezione 5.3 viene analizzato il processo di inizializzazione di nuovi landmark nel caso di utilizzo di una camera come unico sensore. La Sezione 5.4 elenca alcune importanti estensioni dello SLAM per permetterne il suo utilizzo in spazi sempre più ampi e ostici. Nella Sezione 5.5 viene presentato il metodo *1-Point RANSAC*[2], che sfruttando le informazioni contenute nell'EKF permette di rilevare ed escludere in maniera efficiente molti errori di *data association*. Il capitolo si chiude con la Sezione 5.6 nella quale vengono fatte alcune considerazioni finali sulle soluzioni presentate all'interno del capitolo.

[1] Il materiale di questa sezione è stato preso da [HT06a]

[2] Questa sezione è concettualmente separata dal resto del capitolo, ma è stato scelto di esporla a valle della trattazione sull'EKF del Capitolo 4 e della sua particolare implementazione per il problema dello SLAM

5.1 Cenni storici

Alla fine degli anni ottanta, i primi ricercatori che si occupavano dello SLAM, descrivevano il problema con un sistema dinamico e applicavano il filtro di Kalman esteso: il vettore di stato del filtro conteneva posizione e orientamento del robot nonché la posizione di tutti i landmark. In questa configurazione, posizione, orientamento e insieme dei landmark erano tutti correlati fra loro. A quell'epoca, non era ancora stato dimostrato se lo SLAM così costruito convergesse verso una soluzione stabile, si pensava invece che l'errore e l'incertezza sulla stima dello stato del sistema aumentasse monotonicamente. Le prestazioni computazionali degli elaboratori erano inoltre molto inferiori a quelle attuali, quindi la soluzione era di scarsa applicabilità a causa della complessità quadratica della gestione della matrice di covarianza dell'EKF. A causa di questi due motivi si cominciò ad approssimare il problema eliminando la correlazione presente fra i landmark, supponendo che fosse trascurabile e ottenendo un insieme di correlazioni fra singoli landmark e stato attuale del robot. Dopo questa fase lo SLAM ebbe un periodo di pausa, nel quale ci si concentrò sui due problemi della localizzazione e della creazione della mappa in maniera separata. Negli anni a seguire si scoprì che i due problemi considerati congiuntamente garantivano la convergenza del filtro, e che la correlazione fra i landmark della mappa, prima considerata trascurabile, era invece di fondamentale importanza: l'incertezza sulle posizioni assolute dei landmark dipende da quella sulla posizione del robot che le ha stimate, ma l'incertezza delle posizioni relative fra loro può risultare molto bassa anche a fronte di una forte incertezza sulla posizione del robot. Questa forte correlazione fra i landmark fa si che l'osservazione di uno di essi modifichi non solo la stima della propria posizione nella mappa, ma anche quella di tutti gli altri landmark, anche se non attualmente osservati. La correlazione fra i landmark aumenta monotonicamente a ogni nuova osservazione, portando alla convergenza della loro posizione.

Il termine SLAM viene coniato nel 1995 nell'occasione dell'*International Symposium on Robotics Research*.

5.2 EKF SLAM

La realizzazione dello SLAM utilizzando il filtro di Kalman esteso, prevede la descrizione del problema come un sistema dinamico. Come descritto nella sezione precedente, posizione e orientamento del robot e tutti i landmark della mappa sono correlati fra loro. Il vettore di stato del sistema dev'essere quindi così composto:

$$\mathbf{x}_k = [\mathbf{p}_k, \vartheta_k, \mathbf{m}_k]^T = [\mathbf{p}_k, \vartheta_k, \mathbf{l}_{1,k}, \mathbf{l}_{2,k}, \ldots, \mathbf{l}_{N,k}]^T$$

dove $\mathbf{p}_k$ e ϑ_k sono la posizione e l'orientamento del robot al tempo k e $\mathbf{m}_k$ è la mappa, composta da un insieme di N landmark $\mathbf{l}_{i,k}$ con $1 \leq i \leq N$.

Nel passato, lo SLAM era prevalentemente effettuato in ambienti chiusi e strutturati, si assumeva quindi che il robot si muovesse su pavimenti perfettamente orizzontali. In questa condizione la posizione del robot è esprimibile semplicemente attraverso le sue coordinate x e y, non considerando l'altezza da terra z in quanto costante. Anche il suo orientamento risulta notevolmente semplificato ed espresso tramite un unico angolo che descrive la direzione sul piano $x - y$. Uno SLAM di questo genere è chiamato 2D-SLAM. Il caso più generico, è però quello di un robot in grado di muoversi liberamente nel mondo, nei suoi sei gradi di libertà. Questo tipo di SLAM è chiamato 3D-SLAM ed è quello di cui ci occuperemo. In queste condizioni la posizione del robot dev'essere espressa nella sua interezza:

$$\mathbf{p}_k = [x_k, y_k, z_k]^T .$$

Per l'orientamento nel mondo a tre dimensioni, un solo angolo non è più sufficiente, ed è necessario esprimerlo come rotazione rispetto al sistema di riferimento mondo. Le rappresentazioni più comuni

sono le matrici di rotazione, gli angoli di Eulero e i quaternioni. Una matrice di rotazione 3×3 è una rappresentazione molto prolissa e soffre di problemi numerici. Gli angoli di Eulero sono la rappresentazione più compatta, con soli tre angoli per descrivere la rotazione. Sfortunatamente questo metodo soffre del problema chiamato "*gimbal lock*", che in alcune configurazioni perde un grado di libertà. Un quaternione unitario permette infine di definire una rotazione di un angolo α intorno a un asse di rotazione $\vec{\mathbf{u}}$ attraverso un vettore di quattro elementi $[w, x, y, z]^T$. Questa soluzione è sufficientemente compatta ed evita i problemi citati per gli altri due metodi. Date queste considerazioni, l'orientamento del robot nel 3D-SLAM viene generalmente espresso con il quaternione unitario:

$$\vartheta_k = [w_k, x_k, y_k, z_k]^T .$$

Ogni singolo landmark $\mathbf{l}_{i,k}$ rappresenta un punto nel mondo. Nel caso più semplice e intuitivo esso è descrivibile tramite le sue coordinate cartesiane, anche se in base al tipo di sensore utilizzato generalmente si preferisce utilizzare una codifica più complessa che risolve le problematiche di inizializzazione e non linearità delle equazioni di misura (vedi Sezione 5.3). Per ora consideriamo:

$$\mathbf{l}_{i,k} = [l^x_{i,k}, l^y_{i,k}, l^z_{i,k}]^T;$$

si noti l'indice k a pedice di ogni variabile, che esprime la dipendenza del loro valore dall'istante di tempo discreto k. Tale dipendenza nel caso della posizione e dell'orientamento del robot è ovvia, in quanto esso è in grado di muoversi. La staticità della mappa imporrebbe invece che la posizione dei landmark non variasse nel tempo; non dimentichiamo però che queste informazioni sono solamente stimabili e che vengono approssimate sempre meglio col passare del tempo a ogni nuova osservazione.

Riportiamo il sistema dinamico della Sezione 4.3 del capitolo precedente, necessario all'impiego del filtro di Kalman esteso:

$$\begin{aligned} \mathbf{x}_k &= f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_{k-1} \\ \mathbf{y}_k &= h(\mathbf{x}_k, \mathbf{t}_k) \end{aligned}$$

dove abbiamo deciso di rinominare $\mathbf{t}_k$ l'errore di misurazione, $\mathbf{t}_k \sim \mathcal{N}(\mathbf{0}, R_k)$. Abbiamo inoltre portato fuori dalla funzione f il rumore $\mathbf{w}_{k-1}$. Ciò è stato fatto in quanto generalmente non si conosce in che modo tale rumore influenza il modello, e perché nella realtà esso è la somma di più fattori, come l'incertezza sulle intenzioni del robot e l'incertezza sui suoi attuatori che ne sviluppano il moto. Questa semplificazione permette di evitare il calcolo dello Jacobiano della funzione f rispetto a tale rumore.

Nell'applicazione dello SLAM, f è il modello di moto del robot, che data la sua posizione e orientamento al tempo $k-1$, predice quale sarà il loro valore al tempo k. Dal punto di vista dell'EKF, f relaziona il vecchio vettore di stato alla sua nuova stima $\hat{\mathbf{x}}_k^-$. Si noti che il vettore di stato non contiene soltanto le informazioni sul robot, ma anche le posizioni di tutti i landmark, che però rimarranno invariate nella fase di predizione, data l'assunzione di staticità del mondo.

In presenza dei sensori che misurano il moto del robot, il cui insieme è chiamato IMU (*Inertial Measurement Unit*), la predizione del filtro è possibile farla sfruttando queste informazioni. Altrimenti, se disponibili i comandi inviati al robot (es. *avanti, ruota in senso orario* ecc.), è possibile predire grossolanamente il nuovo stato del sistema. In assenza di qualsiasi informazione esterna, il modello di moto deve approssimare al meglio il reale movimento del robot basandosi sulla conoscenza a priori delle sue caratteristiche. Uno dei modelli più usati è quello a velocità costante: la velocità attuale del robot, tanto lineare quanto angolare, viene stimata tramite gli ultimi due stati del sistema; la predizione del prossimo stato del robot viene fatta assumendo che tale velocità sia rimasta costante. In realtà generalmente si preferisce stimare la velocità del robot inserendo anch'essa nel filtro, ottenendo quindi un vettore di stato:

$$\mathbf{x}_k = [\mathbf{p}_k, \vartheta_k, \mathbf{v}_k, \omega_k, \mathbf{m}_k]^T$$

dove $\mathbf{v}_k$ e ω_k sono le velocità lineare e angolare rispettivamente.

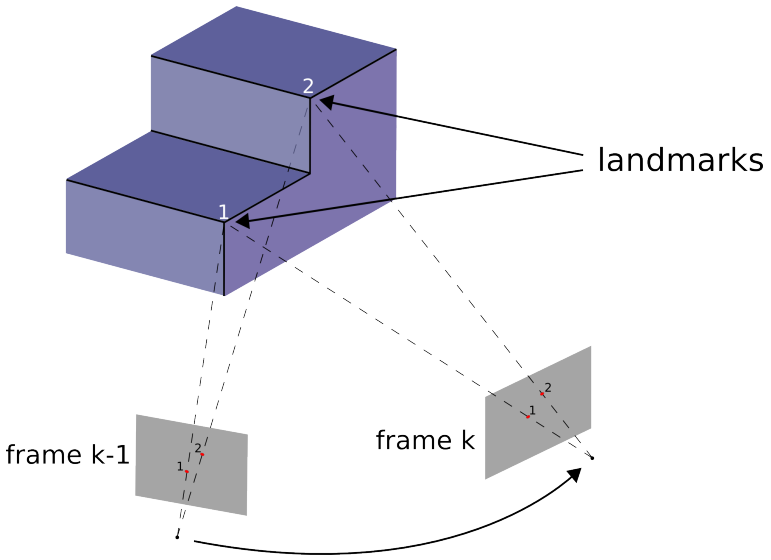


Figura 5.2: *Osservazione dei landmark nel Visual SLAM con camera prospettica*

Ricordiamo che il filtro di Kalman esteso, lavora con variabili Gaussiane e non semplicemente puntuali, quindi oltre a predire quale sarà la nuova posizione e il nuovo orientamento del robot, predirà anche la loro incertezza. Come spiegato nel Capitolo 4, per fare ciò è necessario ottenere gli Jacobiani della funzione f rispetto a posizione e orientamento del robot calcolati nella stima puntuale (media) del vettore di stato $\hat{\mathbf{x}}_{k-1}$.

La funzione h è invece l'equazione di misura, che predice il valore delle osservazioni dei landmark della mappa. Ogni landmark viene elaborato dalla funzione h separatamente. Nel caso Visual SLAM, i landmark sono i punti nel mondo (in 3D) da cui provengono i raggi luminosi che hanno prodotto i punti caratteristici che si stanno seguendo con il tracking. L'osservazione di un landmark consiste quindi nella misura delle coordinate x e y del corrispondente punto caratteristico sul piano immagine (Figura 5.2). La posizione dei landmark nel mondo non è conosciuta a priori, ma solo stimata. Assumendo che tale stima sia sufficientemente accurata, e che la predizione di posizione e orientamento del robot nel nuovo istante di tempo k sia anch'essa buona, è possibile calcolare approssimativamente quale sarà la nuova posizione del punto caratteristico associato al landmark, ossia la sua osservazione. Per fare ciò si utilizza esattamente l'equazione di misura h.

Anche per la funzione h, come fatto per f, affinché il filtro possa calcolare l'incertezza sulla predizione delle osservazioni, è necessario ottenere i suoi Jacobiani rispetto a posizione e orientamento del robot, rispetto al landmark stesso e rispetto all'errore di misura, calcolati nella stima puntuale (media) della predizione del nuovo stato $\hat{\mathbf{x}}_k^-$.

Il discorso fatto per il caso Visual SLAM è pressoché analogo per eventuali altri tipi di sensore, cambierà semplicemente la forma e il significato della funzione h. Non esporremo quindi nessuna possibile forma della funzione h, né il calcolo dei relativi Jacobiani, in quanto variano a seconda del tipo di sensore utilizzato (sonar, range laser, camera ecc.), e dal tipo di codifica utilizzata per descrivere la posizione dei landmark nel mondo. Si noti che anche nel caso in cui si stia utilizzando come sensore una camera, l'equazione di misurazione varierebbe in base alla tipologia di camera: il modello di proiezione di una camera prospettica è infatti differente da quello di una camera omnidirezionale. Questo argomento sarà oggetto del Capitolo 6, in cui verranno approfondite le possibili codifiche dei landmark e le possibili forme della funzione h, per lo specifico caso del Monocular SLAM.

Definiamo ora le condizioni iniziali del filtro, ossia quali valori assume il vettore di stato e la relativa matrice di covarianza in principio. Per quanto riguarda la posizione e l'orientamento del robot, il valore iniziale delle loro medie definisce lo stato del robot rispetto al sistema di riferimento mondo. Siccome sia lo stato iniziale del robot che il sistema di riferimento mondo lo definiamo

noi, l'incertezza iniziale di posizione e orientamento del robot viene considerata nulla. Questa inizializzazione assume una particolare importanza, in quanto l'aggiornamento del filtro tiene conto dell'incertezza su posizione e orientamento del robot, che si ripercuote anche sull'incertezza della stima dei landmark osservati. Nel caso di SLAM senza sensori del moto, come descritto in precedenza, nel filtro ci sono anche la velocità lineare e angolare del robot. Generalmente le loro medie iniziali vengono poste a zero, assumendo che il robot parta da fermo, e attribuendo invece valori di covarianza consoni alla probabilità che questa assunzione sia vera.

Per quanto riguarda i landmark, è necessario rivedere il nostro modello di filtro, in quanto inizialmente non vi sarà nessun landmark, i quali verranno inseriti nel filtro nel momento in cui verranno rilevati dai sensori del robot. Inoltre, la lunghezza del percorso che verrà seguito dal robot è indefinita, quindi, esplorando nuovi ambienti sarà necessario creare nuovi landmark affinché lo SLAM funzioni. Il numero dei landmark nel filtro è quindi destinato ad aumentare monotonicamente e ciò rende la soluzione poco scalabile. Una possibile soluzione al problema è quella di eliminare i vecchi landmark non osservati per un lungo periodo, rimuovendoli dal filtro. Una soluzione migliore è quella di suddividere la sessione di SLAM in sessioni di breve durata, dando origine a sottomappe. In tutti i casi, il filtro di Kalman esteso per lo SLAM è un filtro il cui stato cambia dinamicamente di dimensione.

Definita la struttura del vettore di stato, definite le funzioni f e h e le condizioni iniziali, abbiamo tutti gli ingredienti per applicare il filtro di Kalman esteso al problema dello SLAM.

5.2.1 Implementazione dell'EKF SLAM

In questa sezione faremo riferimento all'implementazione del filtro di Kalman esteso del framework *Moonslam*[3], che permette di ottimizzare alcune operazioni e renderlo applicabile in tempo reale al problema dello SLAM.

Per prima cosa suddividiamo concettualmente il vettore di stato in tre parti distinte:

- dinamica
- parametri
- mappa

La dinamica conterrà posizione e orientamento del robot ed eventualmente la sua velocità lineare e angolare. La dinamica è la parte che viene aggiornata nell'operazione di predizione del filtro. La parte dei parametri, se presente, contiene dei parametri del modello da stimare. La mappa è infine l'insieme dei landmark. Sia i parametri che la mappa non vengono modificati nella fase di predizione del filtro. I parametri sono presenti sin dall'inizio e vengono raffinati con l'evolvere del filtro. I landmark verranno invece inseriti e rimossi dal filtro secondo convenienza. Un esempio di parametri a volte utilizzati nel Visual SLAM, sono i parametri intrinseci della camera. In questo modo se la loro stima iniziale è sufficientemente buona da permettere la convergenza del filtro, essi verranno raffinati in modo da approssimare al meglio le osservazioni.

Definita la suddivisione precedente, la soluzione completa come spiegata finora, possiede un vettore di stato di dimensione:

$$\#(\mathbf{x}_k) = \#(dinamica) + \#(parametri) + N_k \cdot \#(landmark)$$

dove N_k è il numero di landmark presenti nella mappa al tempo k. Facendo un esempio concreto,

[3]Framework per l'implementazione di applicazioni di Visual SLAM sviluppato all'AIRLab del Politecnico di Milano e all'interno del quale verranno implementati i metodi sviluppati in questa tesi. (http://airlab.elet.polimi.it/index.php/MoonSlam)

nel 3D-SLAM senza IMU e senza parametri, il vettore di stato avrebbe una cardinalità:

$$\#(\mathbf{x}_k) = \underbrace{3}_{\text{posizione}} + \underbrace{4}_{\text{orientamento}} + \underbrace{3}_{\text{velocità lineare}} + \underbrace{3}_{\text{velocità angolare}} + N_k \cdot \underbrace{3}_{\text{landmark}} = 13 + N_k \cdot 3$$

dove si ipotizza che i landmark siano codificati in coordinate cartesiane.

La matrice di covarianza associata al vettore di stato avrà una dimensione quadratica rispetto al numero degli elementi presenti in esso. Ricordando le equazioni del filtro di Kalman, è possibile notare che in esse sono implicate moltiplicazioni e inversioni di questa matrice. Per questo motivo la soluzione come descritta finora è computazionalmente proibitiva, o comunque non potrebbe essere utilizzata "*online*". Per risolvere questo problema è possibile sfruttare la conoscenza che abbiamo sulla suddivisione del vettore di stato e sulle proprietà del problema dello SLAM, e implementare le operazioni base del filtro in maniera efficiente.

Inserimento di un nuovo landmark nel filtro

Quando un nuovo landmark viene osservato per la prima volta, esso dev'essere inserito nel filtro. Per questioni di efficienza si sceglie di inserirlo in coda al vettore di stato. Esso avrà quindi una nuovo elemento, e la matrice di covarianza una nuova linea e una nuova colonna:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_d \\ \mathbf{x}_l \\ - \end{bmatrix} = \begin{bmatrix} \mathbf{x}_d \\ \mathbf{x}_l \\ \mathbf{x}_{new} \end{bmatrix}$$

$$P = \begin{bmatrix} P_{dd} & P_{dl} & - \\ P_{dl}^T & P_{ll} & - \\ - & - & - \end{bmatrix} \Leftarrow \begin{bmatrix} P_{dd} & P_{dl} & P_{dd} \cdot J_d^T \\ P_{dl}^T & P_{ll} & P_{dl}^T \cdot J_d^T \\ J_d \cdot P_{dd} & J_d \cdot P_{dl} & J_d \cdot P_{dd} \cdot J_d^T + R \end{bmatrix}.$$

$\mathbf{x}_{new}$ esprime la posizione del nuovo landmark, calcolata con l'equazione di misurazione inversa h^{-1}, J_d è lo Jacobiano di h^{-1} rispetto alle variabili della dinamica e R è la covarianza dell'errore di misurazione, che definisce la precisione del sensore. Si noti che il pedice d si riferisce alla parte delle dinamica ed eventuali parametri, mentre il pedice l alla parte della mappa. Come si può vedere, l'incertezza sulla dinamica del robot P_{dd} si ripercuote sull'incertezza della posizione del nuovo landmark, attraverso l'equazione di misurazione inversa.

Rimozione dei landmark

La rimozione di un landmark i non pone particolari problemi:

$$\mathbf{x} \Leftarrow \begin{bmatrix} \mathbf{x}_d \\ \mathbf{x}_l \\ \cancel{\mathbf{x}_i} \\ \mathbf{x}_L \end{bmatrix} \qquad P \Leftarrow \begin{bmatrix} P_{dd} & P_{dl} & \cancel{P_{di}} & P_{dL} \\ P_{dl}^T & P_{ll} & \cancel{P_{li}} & P_{lL} \\ \cancel{P_{di}^T} & \cancel{P_{li}^T} & \cancel{P_{ii}} & \cancel{P_{iL}} \\ P_{dL}^T & P_{lL}^T & \cancel{P_{iL}^T} & P_{LL} \end{bmatrix}.$$

Il pedice d identifica la parte della dinamica e dei parametri, l la parte di mappa che precede il landmark i da cancellare, e L la parte di mappa che segue.

Predizione del nuovo stato

La predizione del nuovo stato del sistema, utilizza il modello di moto per predire quale sarà la nuova posizione e il nuovo orientamento del robot. Tale operazione non andrà quindi a modificare la parte relativa ai parametri né tantomeno la mappa. Sfruttando questa proprietà la predizione può essere fatta in maniera efficiente sostituendo solo le parti del filtro interessate dalla dinamica.

Per quanto riguarda il vettore di stato, l'operazione da eseguire è la seguente:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_d \\ \mathbf{x}_p \\ \mathbf{x}_l \end{bmatrix} \Leftarrow \begin{bmatrix} f(\mathbf{x}_d) \\ \mathbf{x}_p \\ \mathbf{x}_l \end{bmatrix}$$

dove pedici d, p e l indicano le parti di dinamica, parametri e mappa rispettivamente, mentre f è il modello di moto che proietta lo stato da un istante discreto di tempo a quello successivo.

Per quanto riguarda la matrice di covarianza, cominciamo col definirne la sua forma:

$$P = \begin{bmatrix} P_{dd} & P_{dp} & P_{dl} \\ P_{dp}^T & P_{pp} & P_{pl} \\ P_{dl}^T & P_{pl}^T & P_{ll} \end{bmatrix}.$$

La sua modifica in fase di predizione del filtro passerà attraverso la riassegnazione della prima riga e della prima colonna, i cui blocchi sono coinvolti della dinamica:

$$P_{dd} \Leftarrow F_d \cdot P_{dd} \cdot F_d^T + F_p \cdot P_{dp}^T \cdot F_d^T + F_d \cdot P_{dp} \cdot F_p^T + F_p \cdot P_{pp} \cdot F_p^T + Q$$

$$P_{dp} \Leftarrow F_d \cdot P_{dp} + F_p \cdot P_{pp}$$

$$P_{dl} \Leftarrow F_d \cdot P_{dl} + F_p \cdot P_{pl}.$$

F_d e F_p sono gli Jacobiani della funzione f rispetto alle variabili della dinamica e dei parametri rispettivamente. Q è invece la covarianza del rumore gaussiano che agisce sul sistema.

Calcolo delle nuove osservazioni

Utilizzando la stima della posizione dei landmark nel mondo e la stima dello stato attuale del robot, possiamo calcolare quale dovrebbe essere il valore delle osservazioni. Il calcolo di queste predizioni, avviene singolarmente per ogni landmark i osservato:

$$\hat{\mathbf{y}}_i = h(\mathbf{x}, \mathbf{t}) \qquad \Longrightarrow \qquad \begin{cases} \hat{\bar{\mathbf{y}}}_i &= h(\mathbf{x}, \mathbf{0}) \\ S_i &= H_i \cdot P \cdot H_i^T + T_i \cdot R \cdot T_i^T \end{cases}$$

dove H_i è lo Jacobiano della funzione h rispetto a tutte le variabili del vettore di stato, mentre T_i lo Jacobiano di h rispetto all'errore di misura.

Tutte le misure stimate e le relative osservazioni vengono poi concatenate nel seguente modo:

$$\mathbf{h} = \begin{bmatrix} \hat{\bar{\mathbf{y}}}_1 \\ \hat{\bar{\mathbf{y}}}_2 \\ \vdots \\ \hat{\bar{\mathbf{y}}}_M \end{bmatrix} \mathbf{z} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_M \end{bmatrix} H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_M \end{bmatrix} S = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_M \end{bmatrix}$$

Per rendere efficiente la matrice H e il suo futuro utilizzo, essa viene codificata come matrice sparsa, sfruttando il fatto che le derivate dell'equazione di misura per uno specifico landmark rispetto al resto della mappa sono pari a zero.

Aggiornamento del filtro

Con le informazioni ottenute dal passo di predizione dell'EKF e con le nuove osservazioni e relativa predizione, è possibile aggiornare la stima del nuovo stato del sistema applicando le equazioni

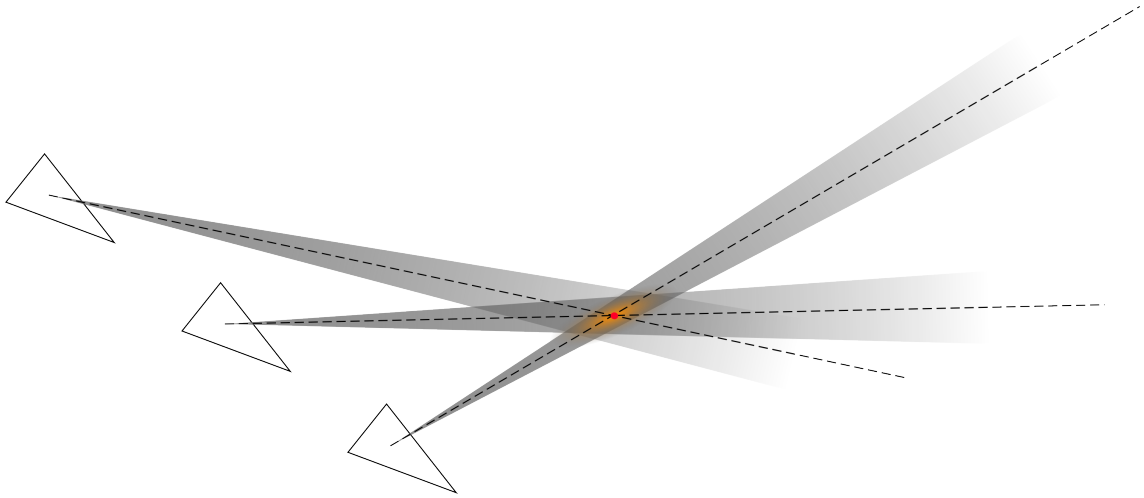


Figura 5.3: *Triangolazione delle osservazioni necessaria per stimare la posizione di un landmark nel Monocular SLAM*

del filtro di Kalman esteso:

$$\begin{aligned} K &\Leftarrow P \cdot H^T \cdot S^{-1} \\ \mathbf{x} &\Leftarrow \mathbf{x} + K(\mathbf{z} - \mathbf{h}) \\ P &\Leftarrow (I - K \cdot H) \cdot P = P - K \cdot S \cdot K^T \end{aligned}$$

con I matrice identità.

Ricordiamo che la matrice di covarianza è simmetrica per definizione. A causa di errori numerici, dopo questi passaggi la proprietà non è più garantita. Un metodo per risolvere il problema è trasformare la matrice P ottenuta come $P \Leftarrow P/2 + P^T/2$.

5.3 Inizializzazione di nuovi landmark

L'inizializzazione di un nuovo landmark per il problema dello SLAM dipende dal tipo di codifica che si intende utilizzare e dal tipo di sensore usato per osservarlo. Inizialmente lo SLAM veniva fatto principalmente usando telemetri laser, che con una singola osservazione erano in grado di determinare in maniera sufficientemente accurata la posizione dei landmark, identificandone direzione e distanza. Negli ultimi anni si è invece investito molto nel Visual SLAM, dove le camere sono i sensori principali, a causa della loro economicità, peso, dimensione e consumi ridotti.

L'uso di un'unica camera come sensore principale (*Monocular SLAM*), ha però il grande svantaggio di non essere in grado di determinare la posizione di un landmark con una singola osservazione. In questo scenario sono necessarie successive osservazioni da prospettive diverse per stimarne la posizione, come illustrato in Figura 5.3. Una soluzione a questo problema sarebbe l'uso di una camera stereo (*Stereo SLAM*), che tramite triangolazione dei raggi uscenti dai suoi obiettivi in direzione del landmark, sarebbe in grado di stimarne la distanza. L'adozione di una camera stereo presenterebbe però due svantaggi non indifferenti: il costo computazionale del tracking aumenterebbe notevolmente e la stima della distanza del landmark rimarrebbe buona solamente nelle vicinanze della camera a causa della ristretta distanza fra i suoi obiettivi (*baseline*).

Utilizzando una singola camera, la soluzione banale è quella dell'inizializzazione "delayed". Alla prima osservazione di un landmark, esso non viene inserito nel filtro, ma posizione e orientamento del robot vengono salvati. Nelle successive osservazioni, si verifica se il cambio di prospettiva fra lo stato attuale del robot e quello salvato è sufficiente per ottenere una buona stima della posizione del landmark attraverso la triangolazione. Se ciò accade il landmark viene inserito nel filtro.

Solà [SML05] propone invece di distribuire diverse ipotesi di distanza lungo il raggio uscente dalla camera verso il landmark, in un range predefinito. Ogni ipotesi è una opportuna gaussiana che viene inserita nel filtro. Nelle successive osservazioni le ipotesi vengono testate con un metodo di verosimiglianza, e via via scartate finché non ne rimane una sola. Questo metodo ha il grande vantaggio di sfruttare le informazioni di un landmark sin dalla sua prima osservazione. Per contro ha un alto costo computazionale. Il metodo viene generalmente chiamato "somma di gaussiane".

Lemaire [LLS05] prende spunto da quest'ultima soluzione e ne costruisce una analoga di tipo *delayed* nella quale le ipotesi di distanza non vengono inserite nel filtro.

Una soluzione simile a quella di Solà viene introdotta da Strasdat [SSBB07], nella quale utilizza un filtro particellare Rao-Blackwellized al posto del classico EKF. Il problema dell'inizializzazione viene risolto distribuendo le particelle lungo il raggio che va dalla camera al landmark, uniformemente, in un range predefinito.

Nelle soluzioni al *Monocular SLAM* descritte finora i landmark all'interno del filtro sono codificati con le classiche coordinate cartesiane 3D nel sistema di riferimento mondo. Tutte queste tecniche si possono ormai considerare superate dal lavoro di Montiel, Civera e Davison del 2006 [CDM08] che introdussero la parametrizzazione *Unified Inverse Depth* (UID), nella quale la posizione del landmark viene codificata tramite la posizione del robot alla sua prima osservazione, la direzione del raggio luminoso proveniente dal landmark e la distanza dalla camera lungo questa direzione espressa tramite il suo inverso. In questo modo un landmark può essere inserito nel filtro alla sua prima osservazione, ponendo un ampia incertezza sul suo valore di distanza. Il metodo verrà discusso esaustivamente nel prossimo capitolo insieme alle sue varianti.

Le discussioni sullo SLAM fatte sin ora, in particolare per il *Monocular SLAM*, prescindono dal tipo di camera utilizzato. La loro validità rimarrà quindi immutata nel caso in cui la camera in uso non sia prospettica ma omnidirezionale. La sostanziale differenza fra le due risiede nella forma dell'equazione di misura e nel relativo Jacobiano, a causa del diverso modello di proiezione (vedi Capitolo 2). Chiaramente uno SLAM effettuato con camera omnidirezionale beneficia dell'ampio campo visivo disponibile in questa tipologia di dispositivo, che permette di osservare i landmark per lunghi periodi e ottenere una stima della loro posizione molto accurata.

Lemaire [LL07] e Gutierrez [GRMG11] svilupparono uno SLAM con camera omnidirezionale utilizzando essenzialmente l'implementazione SLAM classica facente uso dell'EKF. Lemaire inizializza i landmark con il metodo della somma di gaussiane introdotto da Solà mentre Gutierrez utilizza la codifica UID (vedi Capitolo 6).

Scaramuzza[4][Sca08], Burbridge [BSC08] e Hesch [HT05] non utilizzano come landmark i punti 3D del mondo, bensì linee verticali, abbondantemente presenti in ambienti strutturati. Posizionando una camera omnidirezionale in modo che il suo asse sia parallelo alle linee verticali (configurazione tipica), le loro proiezioni sulle immagini acquisite saranno linee radiali. Tramite opportuni descrittori, come quello realizzato da Scaramuzza e descritto nella Sezione 3.3 del Capitolo 3, queste *feature* radiali vengono seguite nel tempo, e la posizione delle relative linee verticali viene stimata tramite SLAM. L'adozione di questa tecnica non permette di fare uno SLAM con sei gradi di libertà, ma solamente il più semplice 2D-SLAM.

5.4 SLAM su larga scala

Le soluzioni fin qui presentate funzionano bene per sessioni di SLAM sufficientemente corte. La loro applicazione diretta a sequenze molto lunghe fa emergere svariati problemi, che possono essere in parte risolti con le tecniche discusse in questa sezione.

Col passare del tempo e la conseguente esplorazione di nuovi ambienti, il numero dei landmark inseriti nel filtro di Kalman esteso aumenta monotonicamente. Data la complessità quadratica del

[4] In realtà Scaramuzza fa Visual Odometry e non SLAM

suo aggiornamento, le prestazioni dell'algoritmo degradano rapidamente non consentendo la sua applicazione in tempo reale. Per risolvere questo problema si ricorre generalmente alla suddivisione in sottomappe, ossia, quando il filtro raggiunge una dimensione considerata limite, la mappa finora creata viene "congelata" e ne viene iniziata una nuova, con un nuovo EKF.

Nell'utilizzo dell'EKF per risolvere il problema dello SLAM generalmente si ipotizza la Markovianità del problema, ossia che il nuovo stato del robot dipenda solamente dal suo stato precedente e dagli ingressi attuali, assumendo che gli stati ancora precedenti non portino nuove informazioni. Questo permette di mantenere all'interno del filtro solo lo stato corrente del robot e non l'intera traiettoria che porterebbe la sua dimensione a crescere monotonicamente. In alcune occasioni questa approssimazione non è valida: si pensi per esempio a un robot che in un determinato momento non riesce a ottenere osservazioni sufficienti a stimare con precisione il proprio stato, e che dopo alcuni frame riesce nuovamente a orientarsi; gli stati del sistema stimati in precedenza rimarranno imprecisi non essendo modificabili in quanto non presenti nel filtro. Il *bundle adjustment* è un algoritmo di ottimizzazione non lineare, che applicato al problema dello SLAM è in grado di ottimizzare l'intera traiettoria del robot congiuntamente alla mappa sfruttando l'intera informazione disponibile. Il suo costo computazionale è molto alto e non ne permette l'applicazione *online*. Tale metodo può essere però applicato su sottosequenze più corte ed eseguito su un thread parallelo con una frequenza di aggiornamento molto inferiore.

Un altro problema dello SLAM, è che essendo un metodo incrementale, l'errore di stima sulla posizione del robot e sulla posizione dei landmark aumenta inevitabilmente col passare del tempo. In alcune occasioni la traiettoria del robot porta a rivisitare più volte la stessa zona, ma a causa degli errori citati la stima dello stato del robot e della mappa potrebbero non essere consistenti. Il *loop closure* si occupa di risolvere questo problema: sfruttando le informazioni derivanti da queste rivisitazioni, registra la traiettoria percorsa in modo da soddisfare i vincoli geometrici.

5.4.1 SLAM gerarchico

Lo SLAM gerarchico, introdotto da Estrada, Neira e Tardós [ENT05], implementa la suddivisione in sottomappe ed effettua la chiusura dei *loop* permettendo di ottenere una soluzione *real time* per SLAM, applicabile in ambienti difficili e su sessioni di lunga durata.

Il problema dello SLAM viene scomposto in sottomappe statisticamente indipendenti, organizzate in una struttura ad albero o a grafo. Ogni sottomappa ha il proprio sistema di riferimento locale. Al raggiungimento della dimensione massima stabilita per una mappa, essa viene congelata, e un nuovo filtro viene inizializzato, relativo alla nuova mappa, dove posizione e orientamento del robot e la relativa incertezza vengono inizializzati a zero. I landmark ancora visibili nella mappa appena abbandonata vengono inseriti anche in quella nuova, ma per mantenere l'indipendenza statistica, essi devono essere nuovamente inizializzati sfruttando solamente le coordinate immagine delle loro ultime osservazioni, come se fossero osservati per la prima volta. Adottando un sistema di riferimento locale a ogni sottomappa, gli errori di linearizzazione che si accumulano durante l'evoluzione dell'EKF rimangono confinati alle singole sottomappe.

Nel grafo a livello globale ogni vertice è associato a una sottomappa, mentre ogni arco identifica la trasformazione fra i sistemi di riferimento associati alle due sottomappe che congiunge. Ogni trasformazione fra mappe adiacenti è frutto dell'evoluzione dell'EKF relativo alla mappa origine dell'arco. Tale informazione è quindi una stima, la cui incertezza è presente nella matrice di covarianza. Il grafo può quindi essere visto come una mappa a livello globale, descritta da un vettore di stato contenente le trasformazioni fra le sottomappe e una matrice di covarianza diagonale a blocchi.

Alla rivisitazione di luoghi già mappati in precedenza (loop closure), si hanno due mappe della stessa zona, quella costruita in passato e quella attuale, che devono essere fuse fra loro. A livello globale dev'essere inserita una nuova trasformazione fra i sistemi di riferimento delle due mappe

fuse, insieme alla relativa correlazione. A causa della chiusura del loop vengono creati dei vincoli fra mappe non adiacenti, espressi tramite blocchi di covarianza fra le mappe in questione, al di fuori della diagonale. La fusione di due mappe corrisponde quindi alla chiusura di un loop a livello globale. A causa della forte incertezza che si accumula in loop molto grandi, la composizione delle trasformazioni lungo il *loop* sarà con buona probabilità inconsistente. Per imporre la consistenza è necessario effettuare una ottimizzazione, basata sul vincolo che tale composizione deve riportare alla posizione di partenza. Questa ottimizzazione è spesso effettuata con un EKF (generalmente IEKF) a livello globale, ma potrebbe essere svolta anche tramite bundle adjustment.

Nel *Monocular SLAM* senza odometria, la scala delle mappe costruite è pressoché casuale. Ciò è vero per ogni sottomappa, in quanto indipendenti fra loro e inizializzate senza usufruire della stima delle posizioni dei landmark della mappa precedente. L'algoritmo così delineato non può quindi essere direttamente applicato a questo caso, ma è necessario ricavare il fattore di scala di ogni mappa relativo a un sistema di riferimento comune. Questa informazione è ricavabile dai landmark in comune fra le sottomappe, applicando appositi algoritmi di *matching*.

Lo SLAM gerarchico è stato utilizzato anche nei lavori di Paz [PTN08] e Clemente [CDR+07].

5.4.2 SLAM Condizionalmente Indipendente

Nel 2008 Piniés e Tardós [PT08] dimostrarono che la struttura intrinseca del problema dello SLAM permetteva di condividere le informazioni fra sottomappe adiacenti mantenendone l'indipendenza condizionale, data la conoscenza dei landmark comuni.

Nel loro lavoro applicarono questa tecnica di suddivisione in sottomappe sia nel caso di sottomappe locali con un proprio sistema di riferimento sia nel caso di sottomappe assolute. L'inizializzazione di una nuova sottomappa con sistema di riferimento assoluto, comporta la copia esatta delle variabili gaussiane di posizione e orientamento del robot, degli eventuali parametri come velocità lineare e angolare del robot, e dei landmark visibili nell'ultimo frame elaborato. Nel caso di sottomappe locali, il sistema di riferimento di una nuova sottomappa è la posizione e orientamento del robot nell'ultimo stato della mappa precedente. Lo stato del robot viene reinizializzato a zero, come la sua incertezza, mentre i parametri verranno semplicemente copiati nel nuovo filtro. I landmark ancora visibili devono essere inseriti nella nuova mappa, ma necessitano di essere ridefiniti rispetto al nuovo sistema di riferimento locale.

Con la duplicazione dei landmark comuni a due sottomappe adiacenti, le nuove copie della seconda mappa potranno essere raffinate con nuove osservazioni, mentre le copie rimaste nella prima mappa manterranno i valori che assumevano al momento della creazione della nuova mappa. L'operazione di *backpropagation* (retropropagazione delle stime) aggiorna i valori di questi landmark propagando nelle mappe precedenti le stime ottenute nelle mappe successive. Si noti che data la stretta correlazione fra i landmark di una mappa, l'aggiornamento dei landmark comuni fra due sottomappe consecutive si propagherà anche sui restanti landmark della prima sottomappa, e quindi anche sugli eventuali landmark comuni ad una sottomappa ancora precedente, che a sua volta dovrà essere aggiornata. L'operazione di *backpropagation* può essere eseguita in un qualsiasi momento. Inoltre, eseguire questa operazione più volte in successione non modifica in alcun modo le stime, in quanto dopo la prima, le esecuzioni successive non porterebbero nessuna nuova informazione. Generalmente si sceglie di posticipare la *backpropagation* alla fine della sessione di SLAM, o finché non viene rilevato un loop, dato che l'algoritmo di *loop closure* necessita di avere una stima consistente dei landmark delle sottomappe coinvolte. Il costo computazionale della retropropagazione è $O(n)$ con n numero sottomappe.

La ricostruzione di un'unica mappa a partire dalle sottomappe costruite richiederebbe $O(n^2)$ operazioni, ma fortunatamente questa operazione non è mai necessaria. Dato che la dimensione di ogni sottomappa è limitata, le operazioni locali dell'EKF della sottomappa corrente hanno un costo $O(1)$. Complessivamente i requisiti di memoria per l'intero algoritmo sono $O(n)$.

A differenza dello SLAM Gerarchico, la possibilità di condividere la velocità lineare e angolare del robot e la stima dei landmark comuni a due sottomappe successive, permette di mantenere lo stesso fattore di scala per tutte le mappe anche nel caso del *Monocular SLAM* senza odometria.

I test sperimentali condotti da Piniés e Tardós hanno inoltre mostrato che l'uso di sottomappe locali, rispetto a quelle con sistema di riferimento assoluto, producono risultati più consistenti, confinando gli errori di linearizzazione dell'EKF solamente alla sottomappa che esso rappresenta.

5.5 1-Point RANSAC con EKF

Supponiamo di avere un insieme di dati ottenuti a partire da un modello parametrico del quale ignoriamo il valore dei suoi parametri. Supponiamo inoltre che il metodo di ottenimento di questi dati non sia sufficientemente preciso, e oltre ad avere il classico rumore gaussiano a media nulla agente su di essi, vi siano anche alcuni errori grossolani chiamati *outliers*, generalmente frutto di errori di classificazione. Si vuole stimare il valore dei parametri del modello a partire dall'insieme di dati.

I classici metodi come *Least Square*, che utilizzano l'intero *dataset* per ottenere la soluzione, falliscono in presenza di errori macroscopici. L'algoritmo di RANSAC, introdotto da Fischler [FB81] più di vent'anni fa, è invece in grado di discernere gli *outliers* e successivamente costruire la soluzione utilizzando solamente i dati validi. Il funzionamento dell'algoritmo è molto semplice: viene formulata un'ipotesi sui parametri del modello, a partire da un insieme di dati minimale scelto casualmente all'interno del *dataset*; i restanti dati vengono testati uno a uno per verificare se concordano con il modello istanziato e l'insieme dei dati concordanti forma quello che viene chiamato "insieme di consenso". Se la cardinalità di tale insieme è sufficientemente alta rispetto al numero di dati presenti nel dataset, il modello può essere considerato corretto, e i suoi parametri possono essere raffinati applicando *Least Square* all'insieme di consenso. Terminata questa operazione, alcuni dei dati che erano stati considerati *outliers* potrebbero ora rientrare nel modello raffinato, e vengono nuovamente testati per ottenere l'insieme di consenso finale. Iterando questo procedimento varie volte, scegliendo casualmente un insieme di dati iniziale, si ottiene con ragionevole certezza almeno una soluzione corretta. Il modello migliore verrà scelto basandosi sulla cardinalità dell'insieme di consenso.

RANSAC si può applicare al problema del tracking per riconoscere quali corrispondenze fra i punti caratteristici di due frame successivi sono valide e quali no. A causa della staticità della scena, le posizioni di tutti i punti di una immagine sono fortemente correlate fra di loro. Tramite l'algoritmo dei 5, 7 o 8 punti è possibile stimare la matrice essenziale E, secondo la quale tutte le corrispondenze fra le due immagini, in linea teorica, dovrebbero rispettare l'equazione $\mathbf{x'}^T E\mathbf{x} = 0$, con $\mathbf{x}$ e $\mathbf{x'}$, rispettivamente, il generico punto caratteristico della prima immagine e il suo corrispondente nella seconda. A causa della rumorosità dei dati, l'equazione per il test di appartenenza all'insieme di consenso diventa $\mathbf{x'}^T E\mathbf{x} < soglia$.

Per istanziare il modello, ossia la matrice E, sono quindi necessarie almeno cinque corrispondenze, utilizzando l'algoritmo dei 5 punti. Il numero di ipotesi da testare affinché vi sia una garanzia sufficiente di ottenere almeno un modello iniziale privo di *outliers*, cresce esponenzialmente rispetto al numero di parametri del modello:

$$k = \frac{log(1-p)}{log(1-(1-p_0)^n)}$$

con k numero di tentativi, p probabilità desiderata di ottenere un insieme di dati iniziale senza *outliers*, p_0 probabilità che un dato sia un *outlier*, n numero di parametri del modello.

L'algoritmo 1-Point RANSAC, introdotto da Civera, Grasa, Davison e Montiel [CGDM10], applicabile al problema dello SLAM con EKF, permette di ridurre notevolmente il numero di iterazioni

k, sfruttando l'informazione presente nel filtro sui parametri del modello da istanziare. Con questa soluzione, è sufficiente un'unica corrispondenza per costruire un'ipotesi del modello. Facendo un esempio pratico, assumiamo $p_0 = 0.5$ e $p = 0.99$: con il RANSAC nella sua versione originale, utilizzando l'algoritmo dei cinque punti, il numero di iterazioni necessarie sarebbe

$$k = \frac{log(1 - 0.99)}{log(1 - (1 - 0.5)^5)} = 145.05$$

mentre con 1-Point RANSAC si ridurrebbe a

$$k = \frac{log(1 - 0.99)}{log(1 - (1 - 0.5)^1)} = 6.64.$$

Per ottenere almeno una soluzione corretta, costruita a partire da dati privi di errori grossolani, con una probabilità del 99%, a partire da un insieme di dati in cui la metà di essi sono *outliers*, sono quindi necessarie 146 iterazioni nel caso del RANSAC classico e solamente 7 nel caso 1-Point RANSAC.

Nell'EKF-SLAM, la procedura di matching dei punti caratteristici fra due frame successivi avviene ricercando i descrittori dei landmark nell'intorno della posizione predetta per essi dall'EKF. Questa operazione produce un'insieme di corrispondenze da verificare con 1-Point RANSAC prima di effettuare l'aggiornamento del filtro.

La formulazione di un'ipotesi si basa sul fatto che esiste una forte correlazione fra tutte le variabili presenti nell'EKF (posizione e orientamento del robot e mappa), quindi, il suo aggiornamento a partire da un'unica osservazione vincola notevolmente la posizione assumibile dalle restanti. La nuova ipotesi dello stato del sistema viene quindi calcolata scegliendo casualmente un'osservazione ed eseguendo un temporaneo aggiornamento del filtro. Solamente il vettore di stato necessita di essere aggiornato, e non la matrice di covarianza che richiederebbe un tempo quadratico rispetto alla dimensione del filtro. Si parla di aggiornamento temporaneo perché il vettore di stato dev'essere successivamente ripristinato, per la generazione di nuove ipotesi e per il vero aggiornamento finale.

A partire dal temporaneo nuovo stato del sistema, tutti i landmark vengono nuovamente proiettati nell'immagine attraverso l'equazione di misura. La posizione dell'osservazione di ogni landmark viene confrontata con la relativa proiezione. L'osservazione viene inserita nell'insieme di consenso se la loro distanza è inferiore a una soglia stabilita. Questo insieme di consenso è chiamato da Civera "insieme a bassa innovazione", in quanto in esso sono presenti solamente osservazioni che si discostano lievemente dalla loro previsione.

L'esecuzione ripetuta delle operazioni fin qui descritte, permette con ragionevole certezza di ottenere un modello valido, con un ampio insieme di consenso. Con il miglior insieme ottenuto, il filtro può essere realmente aggiornato con tutte le osservazioni presenti al suo interno, producendo una miglior stima del nuovo stato del sistema. Si noti che ora l'aggiornamento coinvolge anche la matrice di covarianza.

Le osservazioni rimaste al di fuori dell'insieme di consenso sono state considerate *outliers*, ma potrebbero rientrare nelle ellissi di incertezza dei relativi landmark, fin qui ignorate a causa del mancato aggiornamento della matrice di covarianza. Viene quindi eseguita una operazione di "ripescaggio", inserendo in un nuovo insieme chiamato "insieme ad alta innovazione" le osservazioni precedentemente scartate che rientrano in tali ellissi. Viene infine eseguito un secondo aggiornamento del filtro con le osservazioni ad alta innovazione.

5.6 Considerazioni finali sullo SLAM

Benché dal punto di vista teorico lo SLAM possa essere considerato un problema chiuso, molto ancora rimane da fare per ottenere soluzioni efficienti e precise, realmente applicabili. Il principale

ostacolo che si incontra nella realizzazione di un qualsiasi algoritmo di SLAM, è l'alto numero di dimensioni intrinseco nel problema, dal quale deriva un alto costo computazionale. Una delle tecniche più adottate per realizzare un sistema SLAM è quella basata su filtro di Kalman esteso, che sfruttando alcuni accorgimenti implementativi permette di ottenere una soluzione efficiente e applicabile in tempo reale. A discapito della sua buona efficienza, il funzionamento dell'EKF si basa su alcune ipotesi di linearità locale delle equazioni di moto e di misura, che non sempre rispecchiano la realtà. Alternative all'EKF sono l'UKF e il filtro particellare, che rilassano queste assunzioni di linearità, ma il loro costo computazionale decisamente superiore, non ne permette l'utilizzo in tempo reale, se non effettuando altri tipi di approssimazioni al problema dello SLAM.

Dal punto di vista sensoriale, negli anni si è passati dallo sviluppo di SLAM con range laser e sonar al Visual SLAM, facente uso di camere come sensori principali. Nel caso particolare dei laser, il loro grande vantaggio era quello di poter ottenere una stima precisa della posizione dei landmark fin dalla loro prima osservazione. D'altra parte il costo di questi sensori era molto alto e il loro utilizzo limitato ad ambienti in cui i laser non mettevano a rischio l'incolumità dell'uomo e degli oggetti circostanti. Le camere hanno invece un costo irrisorio e possono essere adoperate i qualsiasi tipo di ambiente, ma possiedono anche il grande svantaggio di non essere in grado di stimare la profondità di un landmark alla sua prima osservazione. Questo problema rende la fase di inizializzazione di nuovi landmark particolarmente problematica, e fa emergere la necessità di nuove tipologie di codifica degli stessi all'interno del filtro (parametrizzazioni), le quali verranno discusse nel prossimo capitolo.

Nel *Visual SLAM*, l'uso di una camera omnidirezionale al posto di una camera prospettica, permette di ottenere un campo di visione molto ampio nel quale i landmark possono essere osservati per lunghi periodi e sono disposti tutt'intorno al robot, ottenendo di conseguenza un miglior condizionamento del problema. D'altra parte, data l'alta distorsione delle immagini di una sequenza omnidirezionale, sono necessarie nuove tecniche di tracking per ottenere osservazioni continuative e affidabili dei landmark. Benché questo si traduca in un ulteriore costo computazionale, esso è pienamente giustificato dall'evidente miglioramento della qualità dello SLAM raggiungibile con questa tipologia di camere.

Come si può evincere da questa discussione, lo SLAM, come molti altri problemi ingegneristici, è una coperta corta ed è necessario bilanciare accuratamente precisione, costo computazionale nonché costi economici della soluzione.

Capitolo 6

Parametrizzazioni

Il *monocular* SLAM ha acquisito popolarità negli ultimi anni, a partire dal lavoro svolto da Davison[Dav03] nel 2003. Il grande vantaggio che deriva dall'uso di una singola camera è la possibilità di effettuare SLAM in tempo reale nei suoi sei gradi di libertà, con buona accuratezza e un costo contenuto.

Il principale problema che si incontra utilizzando una singola camera come unico sensore, è l'impossibilità di determinare la posizione dei landmark in fase di inizializzazione: supponendo di conoscere la posizione e l'orientamento della camera all'acquisizione di una data immagine, l'osservazione di un landmark fornisce una misura accurata della direzione del raggio luminoso fra landmark e centro ottico della camera, ma non porta alcuna informazione sulla distanza alla quale giace il landmark lungo tale raggio. Questo problema provoca inoltre la non determinabilità del fattore di scala della mappa e della traiettoria del robot. A causa di questa limitazione, l'inizializzazione di nuovi landmark nello SLAM monoculare risulta particolarmente problematica, quando i landmark nell'EKF sono espressi tramite una Gaussiana multivariata nelle tre coordinate cartesiane classiche. Tale distribuzione infatti non permette di esprimere la buona qualità della direzione del landmark e la totale ignoranza sulla sua distanza. Varie soluzioni sono state formulate dal 2003 in poi, alcune delle quali descritte nella Sezione 5.3 del capitolo precedente.

Le differenti tipologie di inizializzazione, si possono sostanzialmente dividere in due categorie: *delayed* e *undelayed*. I metodi *delayed* non inseriscono nel filtro le informazioni acquisite su un landmark alla sua prima osservazione, ma attendono finché il cambio di prospettiva diventa sufficiente a effettuare una triangolazione ben condizionata e la distribuzione di probabilità della posizione del landmark diventa pressoché gaussiana. Per quanto riguarda i metodi *undelayed*, inizialmente si utilizzarono particolari implementazioni dello SLAM che permisero di risolvere il problema. Solà [SML05] inizializzava un landmark inserendo nel filtro una molteplicità di gaussiane, ognuna delle quali esprimeva un'ipotesi di distanza fra camera e landmark. Nei frame successivi queste ipotesi venivano testate con un metodo di verosimiglianza e scartate qualora non compatibili con le nuove osservazioni effettuate. Strasdat [SSBB07] implementò invece uno SLAM monoculare con filtro particellare e risolse il problema dell'inizializzazione distribuendo uniformemente una serie di particelle lungo la direzione del landmark, in un range limitato.

La maturità del *monocular* EKF-SLAM si ottenne con la soluzione proposta da Montiel nel 2006, chiamata *Unified Inverse Depth* (UID). L'idea alla base di questa soluzione risiede nel ripensare la codifica dei landmark nel filtro, ossia sostituire la classica parametrizzazione cartesiana con una più complessa che permetta di rappresentare meglio la distribuzione di probabilità della posizione dei landmark, in particolare, nella loro fase di inizializzazione. Da questa intuizione si svilupparono successivamente tutte le altre parametrizzazioni, descritte nel seguito del capitolo. UID e tutte le altre parametrizzazioni rientrano nella categoria delle inizializzazioni *undelayed*. Le diverse parametrizzazioni sostanzialmente aumentano la potenza espressiva della codifica dei landmark, permettendo di rappresentare meglio le distribuzioni di probabilità delle loro posizioni. Questo guadagno in espres-

sività è però controbilanciato da un aumento del numero di parametri necessari alla codifica dei landmark, con un conseguente incremento del costo computazionale dell'algoritmo di SLAM[1].

Tutte le parametrizzazioni che verranno descritte, codificano la posizione dei landmark separando la componente ignota della distanza dalla camera per potergli attribuire un'ampia incertezza. Tutte quante inoltre codificano la distanza tramite il suo inverso, ottenendo una distribuzione sulla componente distanza più vicina alla reale.

I vantaggi derivanti da questa scelta verranno illustrati nella Sezione 6.1, mentre in quella successiva verrà esposta un'alternativa che codifica la componente della distanza con il suo logaritmo negativo. Nella Sezione 6.3 saranno spese alcune parole sulla parametrizzazione classica in coordinate cartesiane, a introduzione di quelle efficacemente applicabili al *monocular* SLAM descritte nelle sezioni successive, in ordine di complessità crescente. Per ognuna di esse verranno definite le equazioni di inizializzazione di un nuovo landmark, della sua misura, e dalla sua trasformazione in coordinate cartesiane. Gli Jacobiani necessari verranno mostrati solo per la prima parametrizzazione (*Inverse Scaling*), demandando al lettore l'analogo calcolo per le altre parametrizzazioni. Allo stesso modo, solo per la parametrizzazione *Inverse Scaling* viene mostrata la variante *FixedFrame*, che permette di specificare la posizione della camera rispetto a un sistema di riferimento robot, non più vincolato a essere il centro ottico della camera.

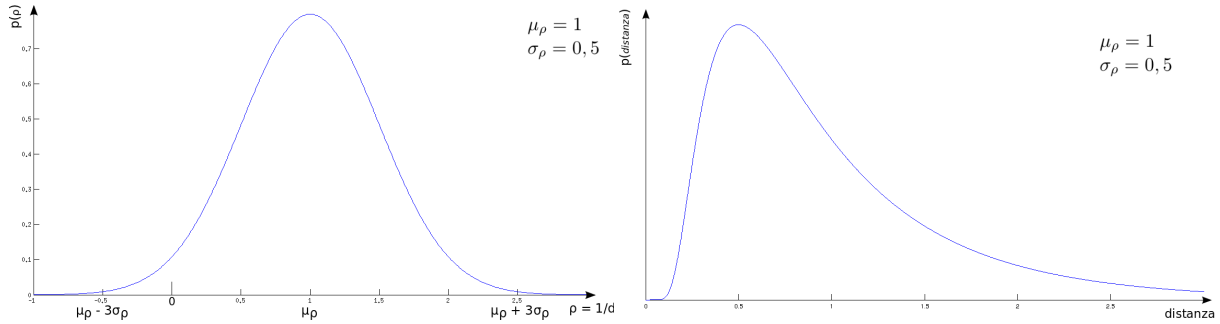
Le equazioni presentate in questo capitolo fanno riferimento al *monocular* SLAM con camera prospettica, ma sono facilmente estendibili al caso omnidirezionale applicando uno dei modelli di camera introdotti nel Capitolo 2. Nella Sezione 7.6 del prossimo capitolo verranno illustrate alcune modifiche alle parametrizzazioni qui introdotte, necessarie per il caso omnidirezionale. Le consuete considerazioni di fine capitolo sono racchiuse nell'ultima sezione, in cui vengono messe a confronto le diverse parametrizzazioni.

6.1 Parametrizzazione della distanza *Inverse Depth*

Nella fase di inizializzazione di un nuovo landmark, la sua distanza d dalla camera non è misurabile e in teoria la sua distribuzione di probabilità dovrebbe essere uniforme fra zero e infinito. Nella realtà la forma di questa distribuzione potrebbe variare a seconda delle conoscenze riguardanti l'ambiente circostante, per esempio, in ambienti interni si potrebbe assumere una distribuzione maggiormente concentrata su valori bassi di distanza.

Né la distribuzione uniforme né gli altri tipi di distribuzione ipotizzabili sono efficacemente approssimabili per mezzo di una Gaussiana. Esprimendo però la componente della distanza tramite il suo inverso $\rho = 1/d$ (*inverse depth*), e assumendo per esso una distribuzione gaussiana $\rho \sim \mathcal{N}(\mu_\rho, \sigma_\rho)$, si ottiene una distribuzione della distanza analoga a quella mostrata in Figura 6.1b. Questo tipo di densità di probabilità presenta il grande vantaggio di poter distribuire la probabilità su un range molto ampio: nello SLAM i landmark molto vicini sono necessari per ottenere una buona stima della componente traslazionale della traiettoria, mentre landmark molto lontani contribuiscono alla stima della parte rotazionale. È quindi importante che la distribuzione della distanza contenga anche valori molto alti, tendenti all'infinito e questa condizione è facilmente raggiungibile imponendo che il valore zero di ρ sia sotteso dalla Gaussiana con cui viene rappresentata. Come risaputo, la densità di probabilità gaussiana non è mai nulla, ossia qualsiasi valore attribuibile alla variabile ha una certa probabilità, benché piccola, di manifestarsi. Si consideri però che il 99,7% della probabilità è racchiusa nel range $(\mu - 3\sigma, \mu + 3\sigma)$. I valori al di fuori di questo range possono quindi considerarsi altamente improbabili, approssimativamente impossibili. Per poter considerare landmark a distanza infinita è quindi necessario che il valore $\rho = 0$ risieda nel range $(\mu_\rho - 3\sigma_\rho, \mu_\rho + 3\sigma_\rho)$.

[1] Per ridurre l'impatto di questo incremento di complessità è possibile adottare la tecnica descritta da Civera [CDM08] che trasforma il landmark dall'attuale parametrizzazione alla classica codifica cartesiana qualora la sua distribuzione fosse sufficientemente gaussiana.



(a) **(b)**

Figura 6.1: **(a)** *Distribuzione gaussiana dell'inverso della distanza* ($\rho = 1/d$). **(b)** *Distribuzione della distanza corrispondente ad (a).*

Uno spiacevole effetto collaterale di questa codifica è che parte della probabilità della Gaussiana di ρ, è distribuita su valori minori di zero, i quali però rappresentano distanze negative che non hanno alcun significato dal punto di vista fisico. Questa caratteristica provoca alcuni problemi al filtro di Kalman esteso, in quanto esso non è in grado di interpretare questo vincolo, e in alcune occasioni il suo aggiornamento provoca una stima negativa della distanza di alcuni landmark. La sezione seguente presenta una parametrizzazione della distanza alternativa a ρ, introdotta da Parsley [PJ08], che risolve questo problema. Purtroppo questa soluzione introduce alcuni aspetti negativi, non presenti nella codifica *Inverse Depth*, tali per cui essa generalmente non viene adottata dalla comunità scientifica.

Un'altra limitazione della codifica *Invese Depth* è la presenza di un limite inferiore sulla distribuzione della distanza. Nella distribuzione gaussiana di ρ, la probabilità che il suo vero valore sia superiore a $\mu_\rho + 3\sigma_\rho$ è $(100\% - 99,73\%)/2 = 0.135\%$, quindi trascurabile. Il limite inferiore della distribuzione della distanza è quindi dato da:

$$d_{min} = \frac{1}{\mu_\rho + 3\sigma_\rho}$$

Riassumendo, è necessario scegliere accuratamente i valori μ_ρ e σ_ρ in modo da soddisfare i vincoli citati:

$$\begin{cases} \mu_\rho > 0 \\ \mu_\rho - 3\sigma_\rho \leq 0 \\ \frac{1}{\mu_\rho + 3\sigma_\rho} \leq d_{min} \end{cases}$$

Una buona scelta di questi parametri, espressi in funzione della minima distanza che si intende rappresentare, è

$$\mu_\rho = \frac{1}{2d_{min}} \qquad \sigma_\rho = \frac{1}{6d_{min}}$$

In questo modo, la distribuzione di probabilità della distanza risulta particolarmente adeguata in fase di inizializzazione di un nuovo landmark. Durante l'evoluzione del filtro e con multiple osservazioni del landmark l'incertezza sulla sua componente ρ diminuisce drasticamente. Come si può vedere dalla Figura 6.2, al diminuire della deviazione standard di ρ la forma della distribuzione della distanza diventa approssimativamente gaussiana.

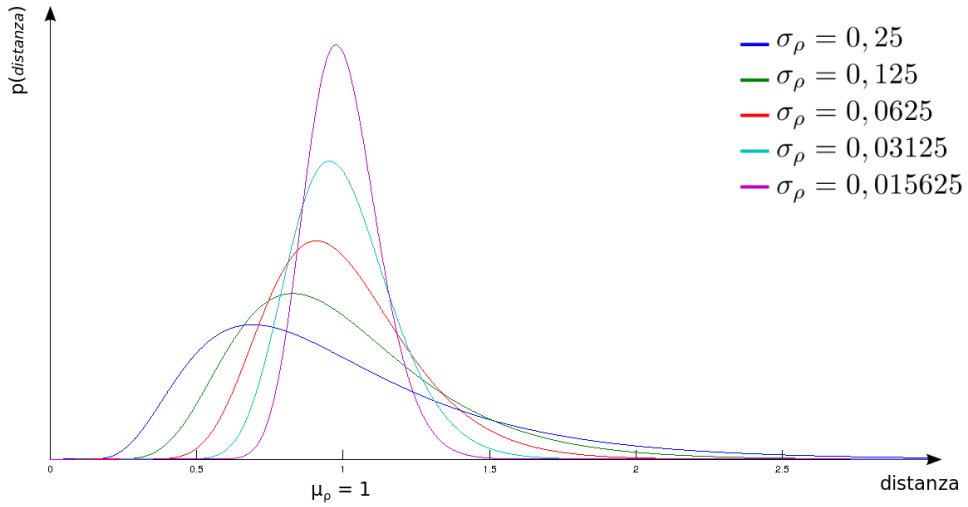


Figura 6.2: *Distribuzione della distanza di un landmark per la codifica Inverse Depth al diminuire della deviazione standard*

6.2 Parametrizzazione della distanza *Negative Log*

Nella sezione precedente è stata presentata la parametrizzazione *Inverse Depth*, applicabile al *monocular* SLAM per descrivere in maniera adeguata la distribuzione di probabilità della distanza di un landmark dalla camera, nella fase di inizializzazione. Questa soluzione ha il grande vantaggio di poter disporre la densità di probabilità su un range infinito di valori di distanza, esprimendo bene il concetto di totale mancanza di conoscenza sul suo reale valore. La soluzione soffre però di due aspetti negativi: esiste un limite inferiore per il range di distanza rappresentabile ed è possibile che si stimino valori di distanza negativi, a causa di errori di linearizzazione dell'EKF. Questi valori negativi non hanno però nessun significato dal punto di vista fisico e dovrebbero essere evitati. Sono disponibili varie soluzioni per risolvere questo problema. La soluzione banale è l'eliminazione dei landmark con ρ negativa. Un altro approccio è quello di fissare la media della Gaussiana che descrive ρ in campo positivo a un valore $\varepsilon = 0^+$. Si sceglie ε perché le distanze negative si ottengono a causa del passaggio di ρ da 0^+ a 0^-, ossia passando da una distanza $+\infty$ a una $-\infty$. Si presuppone quindi che i landmark con distanza inversa negativa siano in realtà molto lontani, a una distanza approssimativamente infinita.

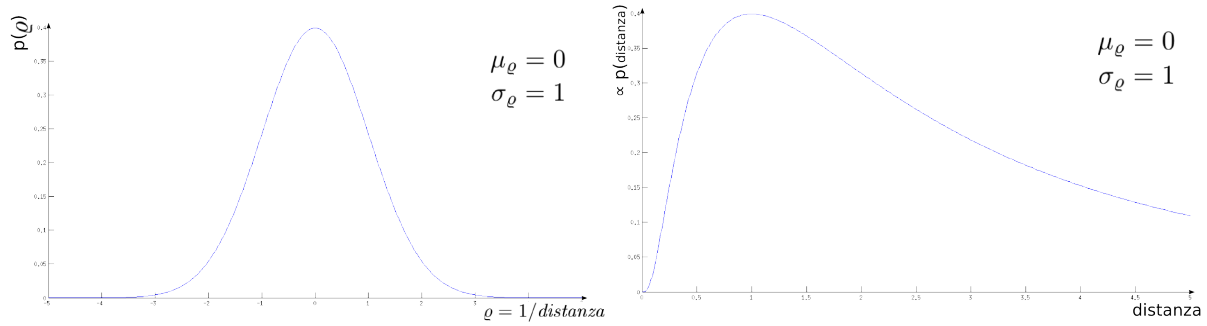
Una soluzione radicalmente diversa è stata presentata da Parsley [PJ08], in cui la distanza viene codificata con il suo logaritmo negativo:

$$\varrho = -\log(d).$$

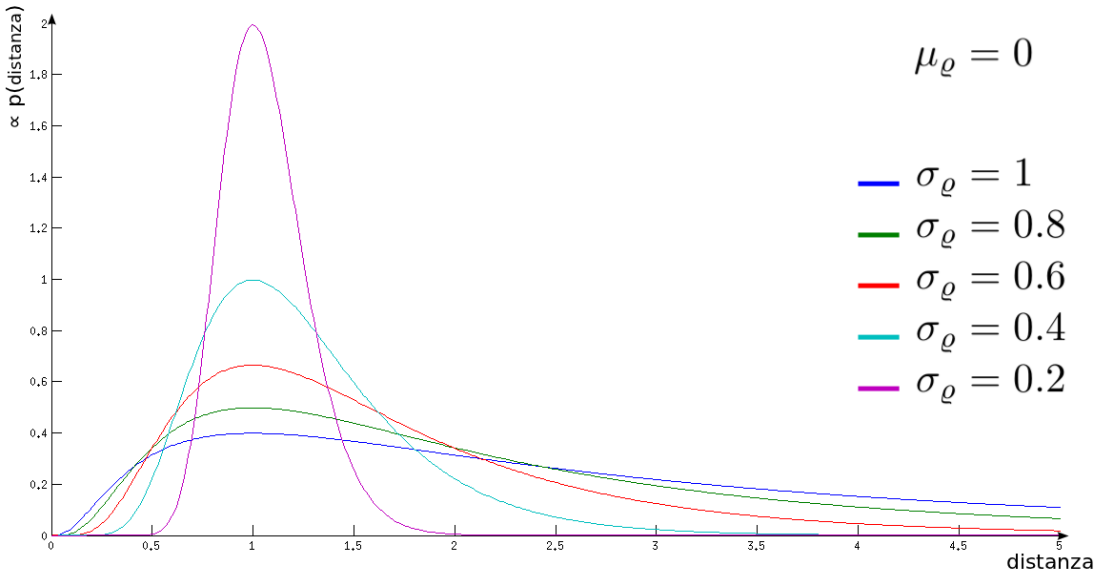
In questo modo tutti i possibili valori di ϱ all'interno del campo reale hanno un'interpretazione fisica: valori negativi codificano distanze maggiori di 1 mentre valori positivi distanze comprese fra 0 e 1. Come si può vedere dalla Figura 6.3b viene risolto anche il problema del limite inferiore sul range di distanze rappresentabile, in quanto la distanza decresce in maniera esponenziale rispetto al crescere di ϱ ed è quindi sufficiente scegliere una deviazione standard sufficientemente ampia.

Per contro in questo caso si ha un limite superiore al range di distanze rappresentabile, dato da:

$$d_{max} = e^{-(\mu_\varrho - 3\sigma_\varrho)}.$$



(a) **(b)**



(c)

Figura 6.3: **(a)** *Distribuzione gaussiana del logaritmo negativo della distanza* ($\varrho = -\log(distanza)$). **(b)** *Distribuzione della distanza corrispondente ad (a).* **(c)** *Convergenza della distribuzione della distanza a una Gaussiana con la riduzione della deviazione standard.*

6.3 Punti Euclidei

Secondo la parametrizzazione classica, un landmark l viene codificato con le sue coordinate cartesiane espresse nel sistema di riferimento mondo:

$$\mathbf{l} = [x \quad y \quad z]^T.$$

La proiezione di un landmark sul piano immagine di una camera con posizione e orientamento conosciute avviene con l'equazione:

$$\mathbf{l_o} = h(\mathbf{l}, \mathbf{T}, R) = KR^T(\mathbf{l} - \mathbf{T}) \tag{6.1}$$

dove $\mathbf{l_o}$ identifica l'osservazione del landmark, ossia le coordinate immagine espresse in pixel della proiezione del landmark, K la matrice dei parametri intrinseci della camera, R e $\mathbf{T}$ la matrice di rotazione e il vettore di traslazione della camera rispetto al sistema di riferimento mondo. Si noti che l'osservazione $\mathbf{l_o}$ è espressa in coordinate omogenee.

Questa parametrizzazione non permette l'inizializzazione *undelayed* dei landmark, in quanto non possiede la sufficiente potenza espressiva per descrivere la reale distribuzione delle loro posizioni, molto lontano dalla gaussianità. Qualora si ottenesse una buona stima della posizione del landmark, per mezzo di successive osservazioni da prospettive differenti, questa codifica rappresenterebbe efficacemente con soli tre parametri la distribuzione di probabilità ormai gaussiana della posizione del landmark.

6.4 Inverse Scaling

L'idea alla base della parametrizzazione introdotta da Marzorati [MMMS09], è l'osservazione che la posizione di un landmark nel mondo è descrivibile scalando opportunamente il triangolo formato dal centro ottico della camera, il suo punto principale e la proiezione del landmark sul piano immagine. Riprendendo l'idea dell'UID (vedi prossima sezione) la scala viene codificata tramite il suo inverso, per beneficiare della forma a coda della distribuzione di probabilità come descritto nella Sezione 6.1. Da queste due considerazioni deriva il nome "Inverse Scaling" (IS).

Nel sistema di riferimento camera, un landmark può quindi essere espresso come:

$$\mathbf{l}_c = [u \quad v \quad w \quad \rho]^T$$

dove u e v sono le coordinate immagine della proiezione del landmark, w la lunghezza focale della camera e ρ un parametro inversamente proporzionale alla distanza del landmark dalla camera (sarebbe esattamente l'inverso se il vettore direzionale $[u\ v\ w]^T$ fosse unitario). u,v e w sono espresse in coordinate metriche e non in pixel.

In questo modo, nella fase di inizializzazione di un nuovo landmark è possibile associare un valore predefinito iniziale al parametro ρ, attribuendogli un'alta incertezza che denunci la pressoché totale mancanza di informazione sulla distanza del landmark (vedi Sezione 6.1).

La codifica adottata può essere vista come la versione omogenea della classica codifica cartesiana, nel sistema di riferimento camera. Nell'EKF-SLAM comunemente adottato i landmark nel filtro vengono espressi nel sistema di riferimento mondo e è quindi necessario applicare la matrice di rototraslazione seguente:

$$H = \begin{bmatrix} R & \mathbf{T} \\ 0 & 1 \end{bmatrix}$$

con $R \in \mathbb{R}^{3\times3}$ e $\mathbf{T} \in \mathbb{R}^3$ rotazione e traslazione della camera rispetto al sistema di riferimento mondo.

Dopo questa operazione si ottiene un nuovo punto omogeneo $\mathbf{l}$, le cui componenti hanno però perso il loro significato originale.

La trasformazione di un landmark dalla parametrizzazione *Inverse Scaling* a quella classica è banalmente:

$$\mathbf{l}_{xyz} = \tau(\mathbf{l}) = \frac{1}{w} \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{6.2}$$

L'inizializzazione di un nuovo landmark viene effettuata con la seguente equazione:

$$\mathbf{l} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = h^{-1}(\mathbf{T}, R, \mathbf{l_o}, \rho_{init}) = H \begin{bmatrix} K^{-1}\mathbf{l_o} \\ \rho_{init} \end{bmatrix} = H \begin{bmatrix} \frac{u-c_x}{fx} \\ \frac{v-c_y}{fy} \\ 1 \\ \rho_{init} \end{bmatrix} \tag{6.3}$$

dove $H \in \mathbb{R}^{4\times4}$ è la matrice di rototraslazione definita in precedenza, $K \in \mathbb{R}^{3\times3}$ la matrice dei parametri intrinseci della camera prospettica, $\mathbf{l_o}$ l'osservazione del landmark sull'immagine in pixel e in coordinate omogenee, ρ_{init} parametro proporzionale all'inverso della distanza iniziale secondo la relazione $\rho_{init} = \|K^{-1}\mathbf{l_o}\|/d_{init}$.

Ricordando l'equazione 6.1 e applicandola a 6.2 si ottiene l'equazione di misura seguente:

$$\mathbf{l_o} = h(\mathbf{l}, \mathbf{T}, R) = KR^T(\mathbf{l}_{xyz} - \mathbf{T}) = KR^T \left(\frac{1}{w} \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \mathbf{T} \right)$$

dove x, y e z sono le prime tre componenti del landmark e w la quarta. $\mathbf{l_o}$ è l'osservazione del landmark. Essendo $\mathbf{l_o}$ espresso in coordinate omogenee, una equazione meglio condizionata, che scongiura la divisione per zero quando $w = 0$, si ottiene moltiplicando il tutto per w ottenendo:

$$\mathbf{l_o} = KR^T \left(\begin{bmatrix} x \\ y \\ z \end{bmatrix} - w\mathbf{T} \right).$$

Nella fase di inizializzazione, la buona approssimazione della distribuzione di probabilità sulla posizione del landmark ottenuta nel sistema di riferimento camera, nel caso in cui posizione e orientamento della camera fossero conosciute con precisione, verrebbe propagata senza distorsione attraverso l'operazione di rototraslazione. In caso contrario, l'incertezza su posizione e orientamento della camera si distribuisce su tutte e quattro le componenti del landmark.

L'operazione di rototraslazione non è lineare, e come tale necessita di essere linearizzata per poterla applicare a una variabile gaussiana preservando tale proprietà. Gli errori di linearizzazione commessi si distribuiranno quindi sulla totalità delle componenti del landmark. A causa di ciò le prestazioni della parametrizzazione *Inverse Scaling*, pur essendo migliori della classica rappresentazione cartesiana, non risultano particolarmente soddisfacenti.

6.4.1 Calcolo degli Jacobiani necessari

Tanto per l'equazione di inizializzazione che per quelle di misura e trasformazione in coordinate cartesiane classiche, è necessario calcolarne gli Jacobiani, affinché oltre al valore medio possa essere calcolata anche la matrice di covarianza delle grandezze calcolate. Il calcolo degli Jacobiani, necessario per tutte le parametrizzazioni, verrà mostrato solo in questa occasione, demandando al lettore lo sviluppo degli analoghi calcoli per le parametrizzazioni che verranno illustrate in seguito.

Ricordando l'operazione di inserimento di un nuovo landmark nell'EKF per SLAM descritto nella Sezione 5.2.1 del Capitolo 5, è necessario calcolare gli Jacobiani dell'Equazione 6.3 rispetto alle variabili della dinamica, nonché lo Jacobiano rispetto alla posizione della feature nell'immagine e dal valore iniziale attribuito a ρ. Gli Jacobiani rispetto alle variabili della dinamica, ossia rispetto a posizione e orientamento della camera, sono necessari per poter correlare correttamente il landmark alle relative variabili nel filtro, mentre l'altro Jacobiano serve per poter calcolare correttamente la covarianza in aggiunta a quella derivante dalla dinamica, per il blocco della matrice di covarianza del filtro relativo al nuovo landmark.

Assumiamo che la variabile del filtro che esprime l'orientamento della camera sia un quaternione $\mathbf{q}$ e che quindi la matrice R finora utilizzata sia il frutto della trasformazione ν di tale quaternione nella relativa matrice di rotazione:

$$R = \nu(\mathbf{q}).$$

Riassumendo, gli Jacobiani da calcolare per l'inizializzazione di un nuovo landmark sono i seguenti:

$$J_T^{init} = \frac{\partial h^{-1}(\mathbf{T}, \mathbf{q}, \mathbf{l_0}, \rho_{init})}{\partial \mathbf{T}} \qquad J_q^{init} = \frac{\partial h^{-1}(\mathbf{T}, \mathbf{q}, \mathbf{l_0}, \rho_{init})}{\partial \mathbf{q}} \qquad J_{noise}^{init} = \frac{\partial h^{-1}(\mathbf{T}, \mathbf{q}, \mathbf{l_0}, \rho_{init})}{\begin{bmatrix} \mathbf{l_o} \\ \rho_{init} \end{bmatrix}}.$$

Ricordando nuovamente la Sezione 5.2.1 del Capitolo 5 relativa all'implementazione dell'EKF per SLAM, nella parte di predizione delle nuove osservazioni dei landmark viene richiesto il calcolo dello Jacobiano H dell'equazione di misura rispetto a tutte le variabili del vettore di stato del filtro. Considerando l'equazione 6.4, notiamo che essa dipende solamente da posizione e orientamento attuale della camera e dal landmark che si sta misurando. Il calcolo dello Jacobiano H può quindi essere scomposto nel calcolo dei tre Jacobiani seguenti:

$$J_T^{meas} = \frac{\partial h(\mathbf{T}, \mathbf{q}, \mathbf{l})}{\partial \mathbf{T}} \qquad J_q^{meas} = \frac{\partial h(\mathbf{T}, \mathbf{q}, \mathbf{l})}{\partial \mathbf{q}} \qquad J_l^{meas} = \frac{\partial h(\mathbf{T}, \mathbf{q}, \mathbf{l})}{\partial \mathbf{l}}.$$

La trasformazione di un landmark dalla parametrizzazione IS a EP operata dall'Equazione 6.2 necessita a sua volta del calcolo dello Jacobiano rispetto alla variabile del landmark, per poter propagare la matrice di covarianza del landmark attraverso questa funzione:

$$J_{IS}^{EP} = \frac{\partial \tau(\mathbf{l})}{\partial \mathbf{l}}.$$

6.4.2 Variante *FixedFrame*

Tutte le parametrizzazioni presentate all'interno di questo capitolo, assumono che il sistema di riferimento robot e il sistema di riferimento camera siano coincidenti, e per questo motivo, all'interno del discorso, abbiamo in più occasioni utilizzato i due termini come sinonimi. La variante *FixedFrame*, applicabile a tutte le parametrizzazioni, permette di rilassare questo vincolo, specificando la relazione di rototraslazione che sussiste fra i sistemi di riferimento camera e robot. Questa soluzione risulta particolarmente utile quando si integrano nello SLAM multipli sensori, ognuno dei quali ha il proprio sistema di riferimento locale che necessita di essere vincolato rispetto a un sistema di riferimento comune. Lo sviluppo delle equazioni per la variante *FixedFrame* viene mostrato solo in questa occasione per la parametrizzazione IS, lasciando al lettore l'eventuale sviluppo delle analoghe equazioni per le altre parametrizzazioni.

Definiamo $\mathbf{T_c}$ e R_c come la traslazione e la rotazione della camera rispetto al sistema di riferimento robot e organizziamoli all'interno della seguente matrice di rototraslazione:

$$H_c = \begin{bmatrix} R_c & \mathbf{T_c} \\ 0 & 1 \end{bmatrix}$$

L'equazione di inizializzazione dei landmark diventa quindi:

$$\mathbf{l} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = H \cdot H_c \cdot \begin{bmatrix} K^{-1}\mathbf{l_o} \\ \rho_{init} \end{bmatrix} = \begin{bmatrix} R(R_c K^{-1}\mathbf{l_o} + T_c\rho_{init}) + T\rho_{init} \\ \rho_{init} \end{bmatrix}.$$

Allo stesso modo l'equazione di misura si modifica in:

$$\mathbf{l_o} = K \cdot \text{homogeneous2cartesian}(H_c^{-1}H^{-1}\mathbf{l}) = KR_c^T \left(R^T \left(\begin{bmatrix} x \\ y \\ z \end{bmatrix} - w\mathbf{T} \right) - wT_c \right).$$

L'equazione della trasformazione dalla parametrizzazione IS a EP non viene invece alterata.

6.5 Unified Inverse Depth

Montiel, Civera e Davison [CDM08] nel 2006 fecero da apripista per le parametrizzazioni nel *monocular* SLAM, presentando la famosa codifica *Unified Inverse Depth* (UID). La loro parametrizzazione è pensata per descrivere al meglio la distribuzione di probabilità della posizione di un landmark nella sua fase di inizializzazione, tramite il vettore:

$$\mathbf{l} \quad = \quad [\mathbf{t}_0 \quad \vartheta \quad \varphi \quad \rho]^T \quad = \quad [x_0 \quad y_0 \quad z_0 \quad \vartheta \quad \varphi \quad \rho]^T$$

dove:

- $\mathbf{t}_0 = [x_0 \quad y_0 \quad z_0]$: posizione della camera alla prima osservazione del landmark.
- ϑ e φ: *azimuth* ed *elevation* della direzione del raggio fra $\mathbf{t}_0$ e il landmark, espressi nel sistema di riferimento mondo.
- $\rho = 1/d$: inverso della distanza del landmark dal centro ottico della camera $\mathbf{t}_0$.

In Figura 6.4 è mostrato lo schema geometrico della parametrizzazione.

All'osservazione $\mathbf{l_o}$ di un nuovo landmark, esso viene inizializzato assegnando i valori iniziali per i sei parametri dell'UID. I valori da attribuire a $\mathbf{t}_0$ sono banalmente le coordinate cartesiane della posizione della camera al tempo attuale:

$$\mathbf{t}_0 = \mathbf{T}.$$

Il calcolo della direzione del landmark è invece più laborioso, in quanto in essa sono raggruppate le informazioni sulla direzione del landmark nel sistema di riferimento camera e l'orientamento della camera stessa. La direzione del landmark nel sistema di riferimento camera è calcolabile come

$$\overrightarrow{\mathbf{h}_c} = K^{-1}\mathbf{l_o} = K^{-1}[u \quad v \quad 1]^T$$

con u e v coordinate cartesiane dell'osservazione $\mathbf{l_o}$ espresse in pixel e K matrice dei parametri intrinseci della camera. È ora necessario applicare la matrice dell'orientamento della camera R alla direzione $\overrightarrow{\mathbf{h}_c}$ ricavata, in modo da esprimerla nel sistema di riferimento mondo. Da questa, come

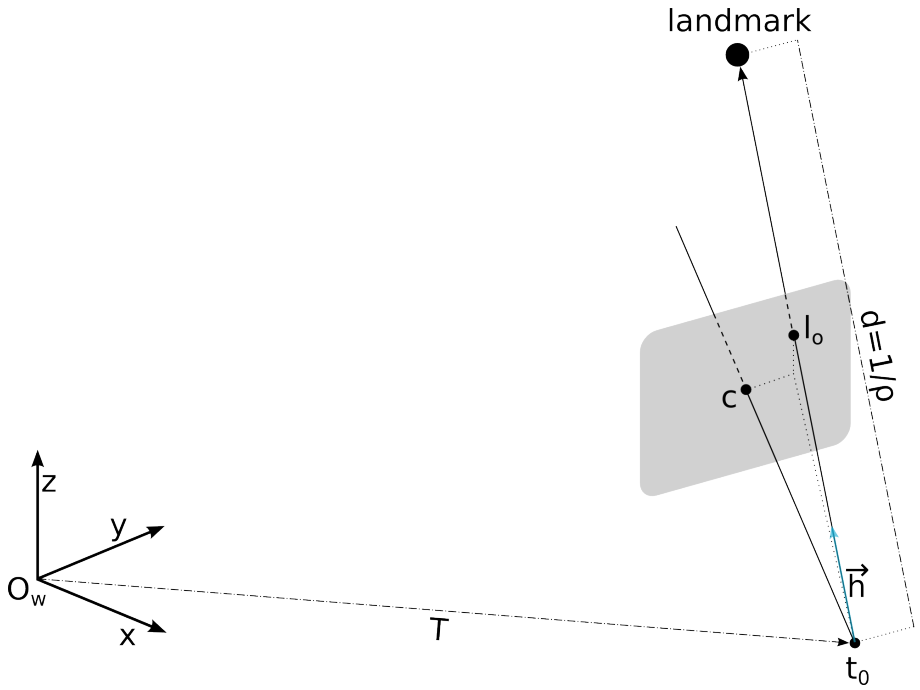


Figura 6.4: *Schema della parametrizzazione UID: O_w è il sistema di riferimento mondo, c il punto principale della camera, $\overrightarrow{\mathbf{h}} = R\,K^{-1}\mathbf{l_o}$ il versore direzione del landmark.*

ultima operazione, è necessario calcolare i valori di *azimuth* ed *elevation* corrispondenti, applicando la funzione η[2]:

$$\begin{bmatrix} \vartheta \\ \varphi \end{bmatrix} = \eta(\overrightarrow{\mathbf{x}}) = \eta(x,y,z) = \begin{bmatrix} \arctan(x/z) \\ \arctan(y/\sqrt{x^2+z^2}) \end{bmatrix}.$$

Per completare l'inizializzazione del landmark è infine necessario assegnare un valore all'inverso della distanza ρ, scelto opportunamente (vedi Sezione 6.1).

Riassumendo, data l'osservazione $\mathbf{l_o} = [u \quad v \quad 1]^T$, data la posizione e orientamento della camera $\mathbf{T}$ e R e dato un valore iniziale ρ_{init} per l'inverso della distanza, l'inizializzazione di un nuovo landmark avviene come:

$$\mathbf{l} = \begin{bmatrix} \mathbf{t}_0 \\ \vartheta \\ \varphi \\ \rho \end{bmatrix} = \begin{bmatrix} \mathbf{T} \\ \eta(R\,K^{-1}\mathbf{l_o}) \\ \\ \rho_{init} \end{bmatrix}.$$

La trasformazione del landmark dalla codifica UID alle classiche coordinate cartesiane è la seguente:

$$\mathbf{l}_{xyz} = \mathbf{t_0} + \frac{\eta^{-1}(\vartheta,\varphi)}{\rho}. \tag{6.4}$$

La funzione η^{-1} applicata è la funzione inversa di η che trasforma *azimuth* ed *elevation* in un vettore direzionale unitario:

$$\overrightarrow{\mathbf{h}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \eta^{-1}(\vartheta,\varphi) = \begin{bmatrix} \cos(\varphi)\sin(\vartheta) \\ -\sin(\varphi) \\ \cos(\varphi)\cos(\vartheta) \end{bmatrix}.$$

[2]Tanto la funzione η quanto la sua funzione inversa η^{-1} utilizzata in seguito, dipendono dalla convenzione del sistema di riferimento utilizzata per il mondo. La definizione delle due funzioni qui mostrata si riferisce a un sistema di riferimento con il piano $x-y$ coplanare al piano immagine della camera e il versore z in direzione frontale.

Applicando l'equazione 6.1 alla 6.4 si ottiene l'equazione di misurazione h:

$$\mathbf{l_o} = h(\mathbf{l}, \mathbf{T}, R) = KR^T(\mathbf{l}_{xyz} - \mathbf{T}) = KR^T\left(\mathbf{t_0} + \frac{\eta^{-1}(\vartheta, \varphi)}{\rho} - \mathbf{T}\right).$$

Essendo $\mathbf{l_o}$ l'osservazione del landmark in coordinate omogenee, l'equazione precedente si può rimodellare in modo da eludere la singolarità data dal caso $\rho = 0$, moltiplicando per ρ:

$$\mathbf{l_o} = KR^T\left(\eta^{-1}(\vartheta, \varphi) + \rho(\mathbf{t_0} - \mathbf{T})\right).$$

Questo tipo di parametrizzazione, oltre a descrivere con buona approssimazione la distribuzione iniziale della posizione di un nuovo landmark, permette di attribuire meglio l'incertezza su parametri specifici: l'incertezza sulla posizione della camera a differenza della codifica *Inverse Scaling*, non influisce sull'intero vettore del landmark, ma solo sulla componente a essa dedicata $\mathbf{t_0}$. In questa soluzione, l'orientamento della camera viene inglobato insieme alla direzione del landmark nei valori di *azimuth* ed *elevation*, quindi una forte incertezza sull'orientamento della camera si ripercuoterà in maniera pesante su questi parametri, anche a fronte di un errore di misurazione del sensore molto basso (generalmente 1 pixel).

Risultati sperimentali ottenuti da Civera [CDM08] e in seguito da Solà [Sol10] dimostrano che soluzioni di SLAM utilizzanti la parametrizzazione UID ottengono buoni risultati in termini di precisione. Dal punto di vista computazione, UID paga una complessità più elevata rispetto a *Inverse Scaling* data dal maggior numero di parametri con cui viene codificato un landmark, sei anziché quattro.

6.6 Anchored Homogeneous Point

La parametrizzazione introdotta da Solà nel 2010 [Sol10] è essenzialmente equivalente alla *Unified Inverse Depth*, tranne per il fatto che la direzione del landmark non viene più espressa tramite *azimuth* ed *elevation*, bensì dal versore direzionale a tre dimensioni:

$$\mathbf{l} \quad = \quad [\mathbf{t}_0 \quad \overrightarrow{\mathbf{h}} \quad \rho]^T \quad = \quad [x_0 \quad y_0 \quad z_0 \quad x_h \quad y_h \quad z_h \quad \rho]^T$$

con $\mathbf{t}_0$ posizione della camera alla prima osservazione del landmark, $\overrightarrow{\mathbf{h}}$ versore unitario della direzione del landmark nel sistema di riferimento mondo traslato in $\mathbf{t}_0$, ρ inverso della distanza del landmark da $\mathbf{t}_0$.

In realtà il significato attribuito a $\overrightarrow{\mathbf{h}}$ e ρ è valido solo per la fase di inizializzazione, in quanto successivi aggiornamenti del filtro non garantiscono di preservare l'unitarietà del versore $\overrightarrow{\mathbf{h}}$: il modulo di $\overrightarrow{\mathbf{h}}$ potrebbe aumentare o diminuire a fronte di una modifica complementare di ρ, lasciando inalterato il loro rapporto. Per mantenere il significato iniziale anche negli step successivi si potrebbe normalizzare $\overrightarrow{\mathbf{h}}$ e spostarne il modulo sul parametro ρ, ma si preferisce lasciare il filtro libero di evolvere senza costrizioni che potrebbero incrementare l'errore di linearizzazione. x_h, y_h, z_h e ρ sono quindi interpretabili come un punto espresso in coordinate omogenee nel sistema di riferimento centrato in $\mathbf{t_0}$ (àncora), da cui deriva il nome *Anchored Homogeneous Point* (AHP).

L'inizializzazione di un nuovo landmark per questa nuova parametrizzazione è pressoché identica al caso UID:

$$\mathbf{l} = \begin{bmatrix} \mathbf{t}_0 \\ \overrightarrow{\mathbf{h}} \\ \rho \end{bmatrix} = \begin{bmatrix} \mathbf{T} \\ R\,K^{-1}\mathbf{l_o} \\ \rho_{init} \end{bmatrix}$$

dove $\mathbf{T}$ è la posizione della camera attuale, R la sua rotazione, K la matrice dei parametri intrinseci, $\mathbf{l_o} = [u \quad v \quad 1]^T$ l'osservazione del landmark in coordinate omogenee espresse in pixel e ρ_{init} l'inverso

della distanza iniziale, scelto opportunamente seguendo le indicazioni della Sezione 6.1.

La trasformazione di un landmark AHP nella classica codifica in coordinate cartesiane avviene con l'equazione:

$$\mathbf{l}_{xyz} = \mathbf{t_0} + \frac{\overrightarrow{\mathbf{h}}}{\rho}; \tag{6.5}$$

l'equazione di misurazione si ottiene componendo le Equazioni 6.1 e 6.5 ottenendo

$$\mathbf{l_o} = h(\mathbf{l}, \mathbf{T}, R) = KR^T \left(\overrightarrow{\mathbf{h}} + \rho(\mathbf{t_0} - \mathbf{T}) \right)$$

L'utilizzo di una codifica a punto omogeneo, in luogo di *azimuth*, *elevation* e ρ del caso UID, evita l'utilizzo delle funzioni goniometriche altamente non lineari presenti in η e η^{-1}, riducendo quindi gli errori di linearizzazione provocati nella fase di propagazione dell'incertezza attraverso l'equazione di misura. Identicamente al caso UID, la direzione del raggio fra $\mathbf{t_0}$ e il landmark incorpora due sorgenti di informazione: l'orientamento della camera e la sua misura del landmark.

6.7 Framed Homogeneous Point

Ceriani nel 2011 propone una nuova parametrizzazione [CMM+11], nella quale mantiene separata l'informazione relativa a posizione e orientamento della camera dall'informazione della sua misurazione del landmark:

$$\mathbf{l} \quad = \quad [\mathbf{t_0} \quad \mathbf{q_0} \quad u \quad v \quad \rho]^T \quad = \quad [x_{t_0} \quad y_{t_0} \quad z_{t_0} \quad w_{q_0} \quad x_{q_0} \quad y_{q_0} \quad z_{q_0} \quad u \quad v \quad \rho]^T$$

dove $\mathbf{t_0} = [x_{t_0}\ y_{t_0}\ z_{t_0}]^T$ e $\mathbf{q_0} = [w_{q_0}\ x_{q_0}\ y_{q_0}\ z_{q_0}]^T$ sono la posizione e l'orientamento della camera alla prima osservazione del landmark, u e v identificano la direzione del landmark nel sistema di riferimento $(\mathbf{t_0}, \mathbf{q_0})$ e ρ un parametro proporzionale all'inverso della distanza fra $\mathbf{t_0}$ e landmark.

L'orientamento della camera è codificato come quaternione unitario perché possiede delle buone caratteristiche di linearità e perché generalmente codificato in questa forma anche nella variabile *attitude* all'interno del filtro di Kalman esteso. Una possibile alternativa sarebbe l'uso degli angoli di Eulero. È da escludere invece la codifica a matrice di rotazione, in quanto computazionalmente molto svantaggiosa (9 parametri) e numericamente instabile.

La direzione del landmark espressa tramite u e v sottintende il vettore direzionale $[u\ v\ 1]^T$ che va dal centro ottico della camera ($\mathbf{t_0}$) alla proiezione del landmark sul piano immagine normalizzato. L'insieme dei parametri u, v e ρ può essere quindi visto come un punto in coordinate omogenee $[u\ v\ 1\ \rho]^T$, in cui la coordinata z rimane costante a uno, essendo essa la distanza del centro ottico dal piano immagine normalizzato.

Si noti che ρ identificherebbe esattamente l'inverso della distanza del landmark, se il vettore direzionale $[u\ v\ 1]^T$ fosse unitario. La relazione che sussiste fra il valore di distanza, il parametro ρ e questo vettore direzionale risulta invece:

$$d = \frac{\|[u\ v\ 1]^T\|}{\rho} = \frac{\sqrt{u^2 + v^2 + 1}}{\rho}.$$

I parametri $\mathbf{t_0}$ e $\mathbf{q_0}$ vengono congiuntamente chiamati "frame" e identificano una particolare prospettiva dal quale i landmark sono osservati. Il nome *Framed Homogeneous Point* (FHP) deriva dalle due considerazioni precedenti.

L'inizializzazione di un nuovo landmark avviene con la seguente equazione:

$$\mathbf{l} = \begin{bmatrix} \mathbf{t_0} \\ \mathbf{q_0} \\ u \\ v \\ \rho \end{bmatrix} = \begin{bmatrix} \mathbf{T} \\ \mathbf{q} \\ u_n \\ v_n \\ \rho_{init} \end{bmatrix}$$

dove $\mathbf{T}$ e $\mathbf{q}$ sono la posizione e l'orientamento attuale della camera, espresso in forma di quaternione unitario. ρ_{init} è proporzionale all'inverso della distanza iniziale del landmark, scelto secondo i criteri descritti nella Sezione 6.1. u_n e v_n sono la trasformazione dell'osservazione $\mathbf{l_o}$ da pixel a coordinate metriche, utilizzando la matrice dei parametri intrinseci della camera K:

$$\begin{bmatrix} u_n \\ v_n \\ 1 \end{bmatrix} = K^{-1} \begin{bmatrix} x_{l_o} \\ y_{l_o} \\ 1 \end{bmatrix}.$$

La trasformazione di un punto FHP nelle classiche coordinate cartesiane è la seguente:

$$\mathbf{l}_{xyz} = \mathbf{t_0} + \nu(\mathbf{q_0}) \frac{[u\ v\ 1]^T}{\rho} \tag{6.6}$$

anche esprimibile in coordinate omogenee come:

$$\mathbf{l}_{xyz} = H_0 \begin{bmatrix} u \\ v \\ 1 \\ \rho \end{bmatrix} = \begin{bmatrix} \nu(\mathbf{q_0}) & \mathbf{t_0} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \\ \rho \end{bmatrix}$$

dove ν è la funzione che trasforma un quaternione nella rispettiva matrice di rotazione.

Considerando la 6.6 e la 6.1 l'equazione di misura di un landmark è:

$$\mathbf{l_o} = h(\mathbf{l}, \mathbf{T}, \mathbf{q}) = K\nu(\mathbf{q})^T(\mathbf{t_0} + \nu(\mathbf{q_0}) \frac{[u\ v\ 1]^T}{\rho} - \mathbf{T})$$

meglio esprimibile come:

$$\mathbf{l_o} = K\nu(\mathbf{q})^T(\nu(\mathbf{q_0})[u\ v\ 1]^T + \rho(\mathbf{t_0} - \mathbf{T}))$$

La parametrizzazione ottenuta, in fase di inizializzazione, permette di distribuire l'incertezza di posizione e orientamento della camera direttamente sui relativi parametri $\mathbf{t_0}$ e $\mathbf{q_0}$. Nelle parametrizzazioni precedenti quest'ultima veniva condensata insieme alle informazioni della misura del landmark, introducendo non linearità.

Test sperimentali hanno dimostrato che le prestazioni qualitative di FHP non si discostano significativamente da quelle di UID e AHP. D'altro canto la parametrizzazione risulta computazionalmente molto più pesante di UID e AHP a causa dell'alto numero di parametri. Notiamo però che FHP ha l'effetto collaterale benevolo di conservare nei landmark del filtro posizione e orientamento della camera relativi ad alcuni istanti di tempo, ossia negli istanti di tempo in cui vengono inizializzati i landmark. Queste informazioni vengono elaborate e raffinate dal filtro con successive osservazioni dei landmark, ottenendo delle stime migliori rispetto a quelle ottenute nel passato dalle variabili *pose* e *attitude*.

6.8 Framed Inverse Depth

Nella parametrizzazione FHP, i due parametri u e v identificano la direzione del landmark nel sistema di riferimento camera. Supponendo di aver calibrato accuratamente la camera e che vi sia una sufficiente risoluzione, la direzione del landmark viene misurata con estrema accuratezza, quindi, non è necessario stimarla ulteriormente mantenendo i due parametri nel filtro di Kalman. u e v possono quindi essere salvati al suo esterno dando origine alla variante di FHP chiamata *Framed Inverse Depth* (FID), sviluppata anch'essa da Ceriani[3].

Analogamente al caso FHP, l'inizializzazione di un nuovo landmark FID avviene nel seguente modo:

$$\mathbf{l} = \begin{bmatrix} \mathbf{t_0} \\ \mathbf{q_0} \\ \rho \end{bmatrix} = \begin{bmatrix} \mathbf{T} \\ \mathbf{q} \\ \rho_{init} \end{bmatrix}$$

ricordandosi però di associare al landmark anche la sua osservazione che ne ha fatto scaturire la creazione $\mathbf{l_o^0} = [x_0\ y_0]^t$.

L'equazione di misurazione diventa:

$$\mathbf{l_o} = K\nu(\mathbf{q})^T(\nu(\mathbf{q_0})\overrightarrow{\delta} + \rho(\mathbf{t_0} - \mathbf{T}))$$

dove $\overrightarrow{\delta}$ è la direzione del landmark calcolata a partire da $\mathbf{l_o^0}$ nel seguente modo:

$$\overrightarrow{\delta} = K^{-1}(\mathbf{l_o^0} + \iota) \qquad \text{con} \qquad \iota \sim \mathcal{N}(0, \sigma_\iota).$$

ι identifica l'incertezza della misura iniziale data dall'accuratezza della camera.

6.9 Considerazioni finali

In questo capitolo sono state introdotte una serie di parametrizzazioni applicabili al *monocular* SLAM, che permettono l'inizializzazione *undelayed* dei landmark alla loro prima osservazione.

Un aspetto particolarmente importante che accomuna tutte queste parametrizzazioni è l'utilizzo di componenti separate per descrivere la direzione del landmark e la sua distanza dalla camera, in modo da potergli attribuire valori di incertezza differenti: bassi per la direzione e alti per la distanza. Questa distanza viene codificata generalmente tramite il suo inverso, o in alternativa con il suo logaritmo negativo, permettendo in entrambe i casi, in fase di inizializzazione di nuovi landmark, di ottenere una descrizione della sua distribuzione più fedele a quella reale, uniforme fra zero e infinito.

Le parametrizzazioni sono state presentate in ordine di complessità crescente. *Inverse Scaling* (IS) descrive un punto 3D in coordinate omogenee con soli 4 parametri. L'incertezza relativa a posizione e orientamento della camera in fase di inizializzazione viene integrata in questi quattro parametri attraverso una rototraslazione, operazione non lineare che comporta una perdita di informazione quando linearizzata per propagare la covarianza. Per questo motivo i risultati ottenuti in letteratura con questa soluzione non sono soddisfacenti. La parametrizzazione più conosciuta e usata è l'*Unified Inverse Depth* (UID), che introduce il concetto di "àncora", ossia il punto 3D dal quale il landmark viene osservato per la prima volta. In questo modo in fase di inizializzazione l'incertezza della posizione della camera viene propagata linearmente sull'ancora del landmark, e non influirà sui restanti parametri. La direzione del landmark viene codificata tramite *azimuth* ed *elevation*, inglobando in essa anche l'orientamento della camera con la relativa incertezza. I risultati ottenuti con questa parametrizzazione sono qualitativamente buoni, a discapito di un costo com-

[3] Una parametrizzazione pressoché identica a FID, chiamata *Inverse Depth Bundle* è stata presentata da Pietzsch [Pie08a].

putazionale ben quattro volte superiore alla classica codifica in coordinate cartesiane (vedi Tabella 6.1). La parametrizzazione *Anchored Homogeneous Point* (AHP) è praticamente identica all'UID, con l'unica differenza che la direzione del landmark non è più espressa tramite *azimuth* ed *elevation*, bensì dal corrispondente vettore direzionale. Il miglioramento introdotto consiste proprio nell'evitare l'utilizzo delle funzioni goniometriche per il calcolo di *azimuth* ed *elevation*, altamente non lineari. I risultati ottenuti con questa soluzione sono in linea con quelli di UID. La codifica *Framed Homogeneous Point* estende l'idea alla base di UID e AHP, parametrizzando non solo la posizione della camera quando un landmark viene osservato per la prima volta, ma anche il suo orientamento. In questo modo, in fase di inizializzazione, posizione e orientamento della camera vengono assegnati direttamente ai corrispettivi parametri, ottenendo una equazione perfettamente lineare. Anche in questo caso i risultati ottenuti non si discostano in maniera significativa da quelli ottenuti con UID e AHP. Il costo computazionale di FHP è invece molto alto, dato dall'utilizzo di ben dieci parametri. *Framed Inverse Depth* (FID) è una parametrizzazione pressoché identica a FHP, in cui i due parametri della direzione del landmark non vengono inseriti nel filtro, in quanto essa viene misurata con buona precisione della camera, quindi, non si ritengono necessarie successive stime. Questo accorgimento porta a un abbattimento del costo computazionale significativo, mantenendo tutte le caratteristiche di linearità della parametrizzazione FHP. L'utilizzo di FHP e FID, benché non si siano dimostrati qualitativamente migliori di UID e AHP, può essere giustificato dalla possibilità di mantenere nel filtro parte della traiettoria percorsa dal robot.

Come risulta evidente dalla Tabella 6.1 tutte le parametrizzazioni introdotte hanno un costo computazionale nettamente superiore alla classica codifica in coordinate cartesiane (EP). Ciò nonostante il loro utilizzo è indispensabile in fase di inizializzazione per descrivere in maniera adeguata la distribuzione di probabilità iniziale non gaussiana della posizione dei landmark. Considerando il periodo di vita di un landmark, nelle fasi successive alla sua creazione la stima della sua posizione viene raffinata e la sua incertezza ridotta fino a ottenere una distribuzione pressoché gaussiana. Sotto queste condizioni è quindi possibile trasformare il landmark dalla parametrizzazione attuale a quella classica, senza degradare la qualità dello SLAM, ma ottenendo un notevole risparmio computazionale.

Un'altra miglioria, da noi chiamata *Shared*, applicabile alle parametrizzazioni che fanno uso di un'*àncora* (UID, AHP) o di un *frame* (FHP, FID) si basa sulla seguente osservazione: se una serie di landmark vengono inizializzati contemporaneamente in un determinato istante di tempo, la loro *àncora*/*frame* sarà identica. È quindi possibile condividere questa informazione fra i vari landmark coinvolti, evitandone la replica all'interno della parametrizzazione di ognuno di essi. Come viene mostrato nella quarta colonna della Tabella 6.1, questa soluzione porta a una notevole riduzione del costo computazionale, nel caso più landmark vengano inizializzati nel medesimo istante di tempo: consideriamo per esempio l'inizializzazione contemporanea di 10 landmark; con la parametrizzazione EP sarebbero necessari un totale di 30 parametri, mentre con FID solamente 17, 10 per il parametro ρ di ogni landmark e 7 per il *frame* condiviso.

parametrizzazione	numero di parametri	fattore di costo computazionale rispetto a EP	costo computazionale della variante *Shared*	qualità del monocular SLAM ottenibile
EP	3	1	3N	pessima
IS	4	1.78	4N	scarsa
UID	6	4	3+3N	buona
AHP	7	5.44	3+4N	buona
FHP	10	11.11	7+3N	buona
FID	8	7.11	7+N	buona

Tabella 6.1: *Comparazione delle diverse parametrizzazioni per il monocular SLAM. La quarta colonna mostra il costo computazionale di* N *landmark inizializzati contemporaneamente quando si utilizza la variante* Shared *delle parametrizzazioni.*

Capitolo 7

Visual SLAM con immagini omnidirezionali

Il contributo teorico di questa tesi consiste nello studio di nuovi metodi per effettuare SLAM utilizzando camere omnidirezionali, in modo da sfruttare al meglio l'ampio campo di visione a disposizione con questa tipologia di dispositivi. Questa soluzione di SLAM, benché molto promettente non risulta ancora molto studiata e diffusa in letteratura.

Le principali differenze fra un sistema SLAM classico con camera prospettica e uno con camera omnidirezionale risiedono nei metodi per effettuare il tracking, in quanto le tecniche classiche sviluppate negli ultimi decenni per il caso prospettico non si adattano bene al caso omnidirezionale, a causa della forte distorsione presente nelle sue immagini. Inoltre, a causa del diverso modello di proiezione della camera, le equazioni per la creazione di nuovi landmark e quelle per la loro misura devono essere modificate.

Per effettuare il tracking delle feature con camere omnidirezionali, ci siamo concentrati sui metodi di tracking con patch e ne abbiamo sviluppati due che tentano di risolvere il problema della distorsione: il primo metodo trasforma l'immagine omnidirezionale nel suo sviluppo panoramico, riducendone notevolmente la distorsione e applicando successivamente i metodi classici; il secondo metodo lavora invece a livello locale, individuando le nuove feature direttamente sull'immagine omnidirezionale e trasformando solamente le zone circostanti le feature nelle corrispettive immagini prospettiche per generare e confrontare le patch.

Nelle sezioni 7.1 e 7.2 vengono dettagliati i due metodi di tracking introdotti. Nella Sezione 7.3 viene spiegato il metodo del patch warping per camere prospettiche e come applicarlo al caso omnidirezionale. Nella Sezione 7.4 stata studiata la possibilità di aggiornare le patch, sostituendole con il loro nuovo aspetto assunto nei frame successivi al primo. Nella Sezione 7.5 viene discusso un metodo alternativo all'inizializzazione costante della distanza inversa dei nuovi landmark, che invece di disporli a una distanza fissa dal centro ottico della camera, li dispone sul piano del terreno sul quale il robot si muove. Nella Sezione 7.6 vengono introdotte alcune modifiche alle parametrizzazioni del capitolo precedente, necessarie per il loro utilizzo con il nuovo modello di camera omnidirezionale. Infine, nell'ultima sezione del capitolo vengono espresse alcune considerazioni sui metodi sviluppati, con un breve confronto fra loro.

7.1 Tracking su sviluppo panoramico

A causa della forte distorsione presente nelle immagini acquisite con una camera omnidirezionale, l'applicazione diretta del patch tracking classico a questa tipologia di camere non produce buoni risultati, in quanto anche a fronte di ridotti spostamenti o cambi di prospettiva, le aree dell'immagine circostanti le feature cambiano radicalmente, rendendo particolarmente imprecisa l'operazione di

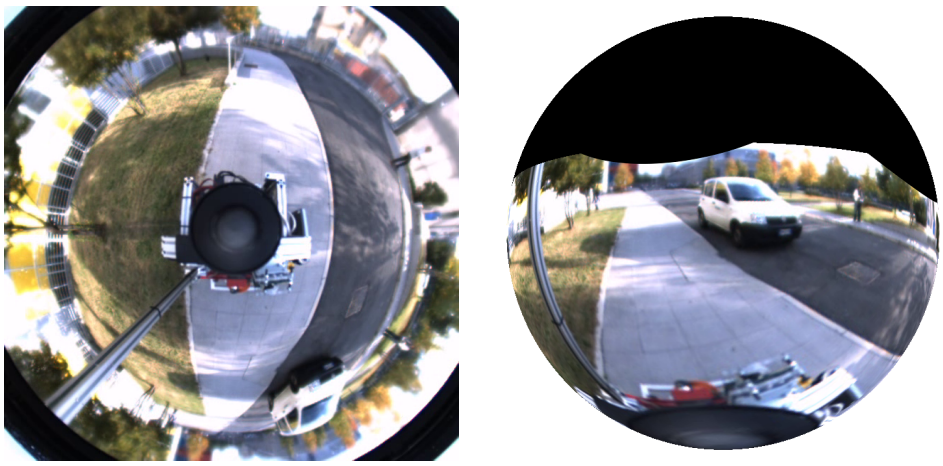


Figura 7.1: *Immagine omnidirezionale e sua interpretazione sferica*

Figura 7.2: *Costruzione di un'immagine panoramica: l'immagine omnidirezionale viene proiettata sulla parete di un cilindro e successivamente la sua superficie viene "srotolata".*

matching. Per ovviare a questo problema, si è realizzato un metodo che trasforma l'immagine omnidirezionale originale nel suo sviluppo panoramico, e applica il patch tracking classico su questa nuova rappresentazione.

Come spiegato nel Capitolo 2, una camera omnidirezionale con un unico centro di proiezione, consente di ottenere la direzione del raggio luminoso corrispondente a un qualsiasi pixel dell'immagine omnidirezionale all'interno del campo di visione. Astraendo, è quindi possibile pensare all'immagine omnidirezionale come alla superficie di una sfera il cui centro e asse di simmetria coincidono nel caso catadiottrico con il fuoco della parabola o iperbole dello specchio e il suo asse. Il caso della camera *fisheye* è analogo. Chiaramente non tutta la superficie della sfera è coperta dal campo visivo della camera: generalmente la porzione visibile risulta essere una banda centrale nel caso di camera catadiottrica oppure un emisfero nel caso *fisheye* (vedi Figura 7.1).

Immaginiamo ora di introdurre questa sfera all'interno di un cilindro, il cui diametro e asse di simmetria coincidano con quelli della sfera. Proiettiamo quindi l'immagine omnidirezionale dalla superficie della sfera sulla parete del cilindro, utilizzando una proiezione prospettica con origine nel centro della sfera. Infine immaginiamo di "srotolare" la parete del cilindro e renderla planare, tagliandola verticalmente in un punto qualsivoglia. Otteniamo così l'immagine panoramica corrispondente all'immagine omnidirezionale originale (Figura 7.2).

Dal punto di vista matematico e implementativo, sarebbe necessario calcolare per ogni pixel dell'immagine omnidirezionale originale, la posizione del corrispondente pixel nell'immagine panoramica. È invece risaputo che in questa tipologia di trasformazioni fra immagini, è necessario effettuare i calcoli nella direzione opposta, in quanto non è garantito che per ogni pixel dell'immagine panoramica ve ne sia uno dell'immagine omnidirezionale che lo generi. Consideriamo inoltre che in generale il risultato del calcolo del pixel destinazione non ha coordinate intere. Calcolando

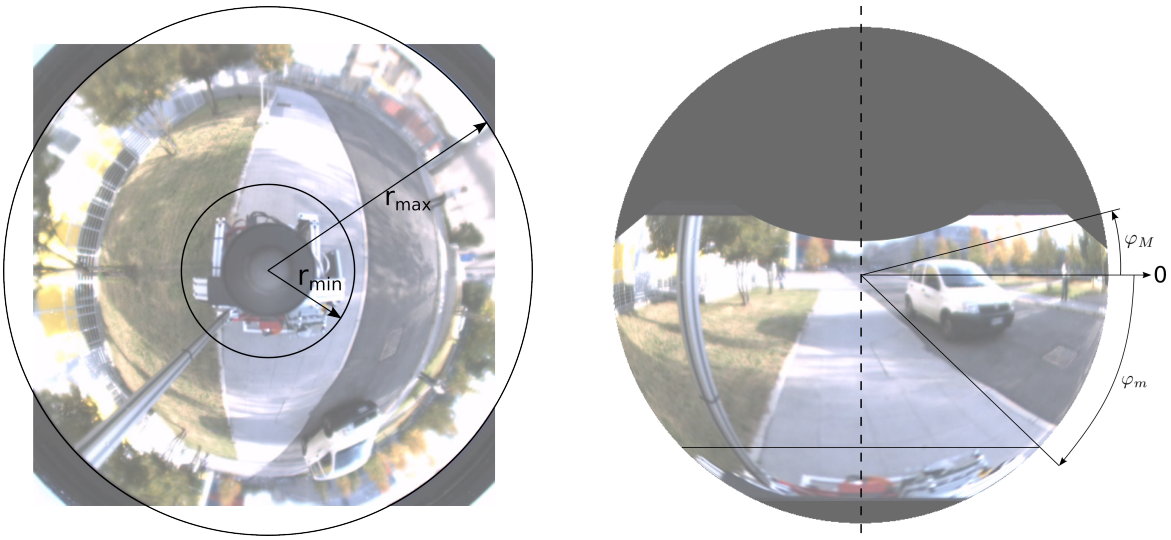


Figura 7.3: *Definizione del campo visivo verticale di una camera omnidirezionale: φ_M e φ_m sono gli angoli di elevation che definiscono gli estremi del campo visivo verticale, corrispondente ai raggi r_{max} e r_{min} sull'immagine omnidirezionale.*

invece la posizione del pixel nell'immagine omnidirezionale corrispondente a ogni pixel di quella panoramica, si ha garanzia della completa copertura dell'intera immagine panoramica ed è inoltre possibile assegnare con maggior precisione la tinta a ogni pixel, interpolando quelli dell'immagine sorgente.

La trasformazione di un generico punto $\mathbf{x_p}$ dell'immagine panoramica nel suo corrispondente $\mathbf{x_o}$ sull'immagine omnidirezionale è svolta nei seguenti passaggi:

- Il punto $\mathbf{x_p}$ viene proiettato sulla sfera di raggio unitario ottenendo il punto $\mathbf{x_s} \in \mathbb{R}^3$
- Il punto $\mathbf{x_s}$ viene trasformato nel corrispondente $\mathbf{x_o}$ usando la funzione `world2cam` definita dal modello di camera (vedi Capitolo 2)

Per semplicità definiamo a raggio unitario la sfera che rappresenta l'immagine omnidirezionale, senza perdita di generalità. Definiamo gli angoli ϑ e φ come *azimuth* ed *elevation* di $\mathbf{x_s}$. Definiamo φ_M e φ_m come gli estremi superiore e inferiore del campo di visione verticale (Figura 7.3): un raggio luminoso uscente dalla camera con un φ superiore a φ_M uscirebbe dal campo di visione, ad esempio nel caso catadiottrico non colpirebbe lo specchio. Considerando nuovamente il caso catadiottrico, i raggi con φ inferiore a φ_m formano l'area centrale dell'immagine omnidirezionale che riflette la camera stessa, la quale non ha nessuna utilità pratica.

Definiamo w e h la larghezza e l'altezza dell'immagine panoramica in pixel. I valori di questi parametri sono legati fra loro da una proporzione che dev'essere rispettata, ma entrambi possono essere alterati da un fattore di scala che definisce la risoluzione dell'immagine. La proporzione che lega w e h fra loro è (vedi anche Figura 7.4):

$$h = \frac{\tan\varphi_M - \tan\varphi_m}{2\pi} w. \tag{7.1}$$

Definiamo γ come il rapporto fra unità di misura metrica e pixel, facilmente calcolabile osservando che la larghezza dell'immagine panoramica w in pixel corrisponde alla lunghezza della circonferenza sulla superficie della sfera corrispondente a φ uguale a zero, che a causa dell'unitarietà del raggio risulta essere 2π:

$$\gamma = \frac{2\pi}{w}.$$

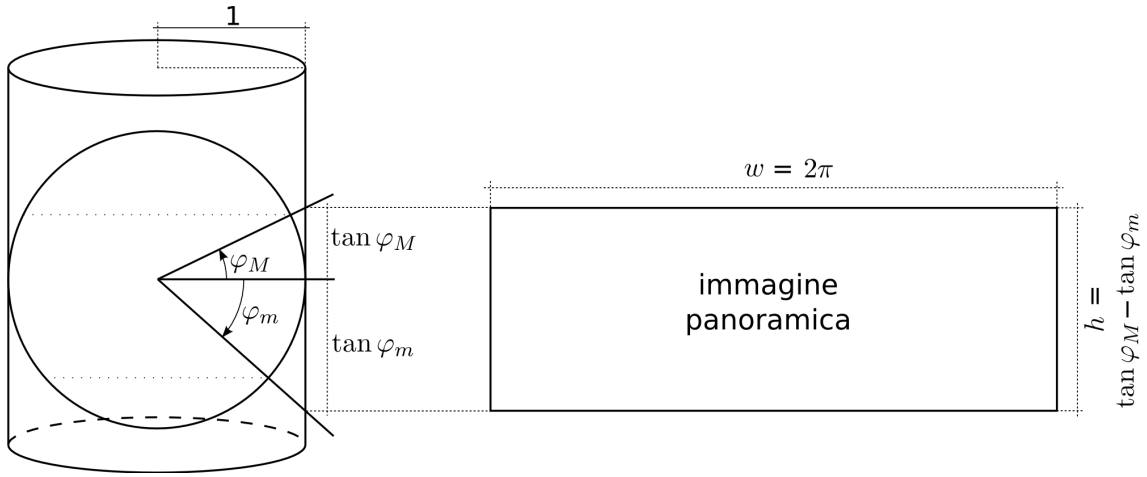


Figura 7.4: *Calcolo della proporzione fra altezza e larghezza di un'immagine panoramica*

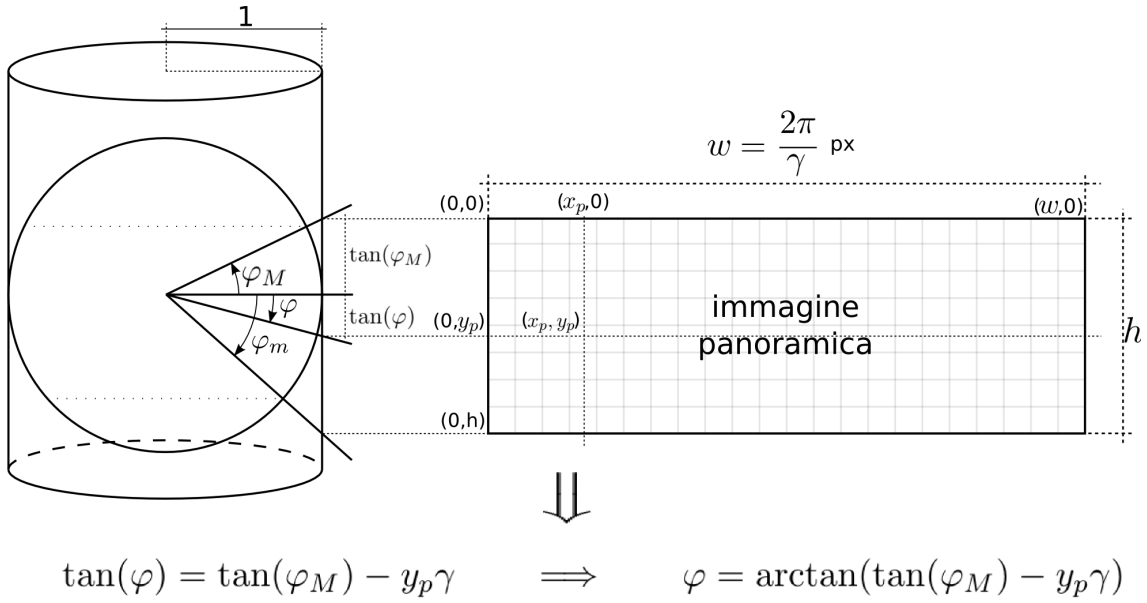


Figura 7.5: *Calcolo dell'angolo di elevation φ corrispondente alla coordinata y_p di un pixel dell'immagine panoramica*

Proiettiamo il generico punto $\mathbf{x_p} = [x_p, y_p]^T$ dell'immagine panoramica sulla sfera di raggio unitario, ottenendo il punto $\mathbf{x_s}$ come *azimuth* (ϑ) ed *elevation* (φ):

$$\begin{aligned} \vartheta &= x_p\gamma \\ \varphi &= \arctan(\tan(\varphi_M) - y_p\gamma). \end{aligned} \tag{7.2}$$

Il calcolo di ϑ è immediato, trasformando la coordinata x_p da pixel in unità di misura metrica usando γ e ricordando che l'immagine omnidirezionale è larga 2π e che ricopre l'intero angolo giro. Il calcolo di φ viene invece illustrato in Figura 7.5.

Da ϑ e φ calcoliamo il punto $\mathbf{x_s}$ in coordinate cartesiane:

$$\mathbf{x_s} = \begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix} = \begin{bmatrix} \cos\varphi \sin\vartheta \\ \cos\varphi \cos\vartheta \\ \sin\varphi \end{bmatrix}.$$

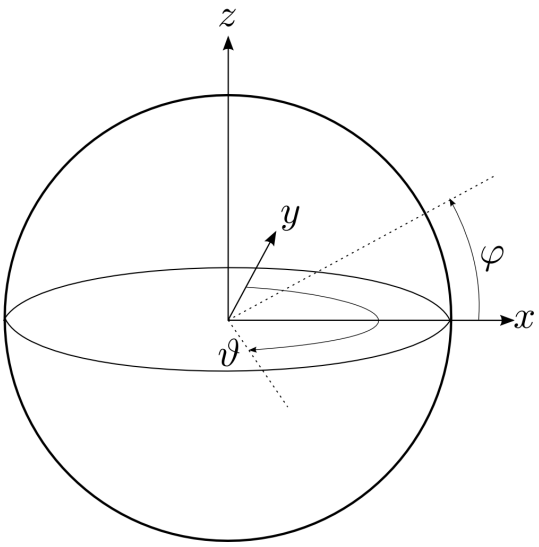


Figura 7.6: *Sistema di riferimento adottato per la camera omnidirezionale*

Si noti che questa trasformazione dipende dal sistema di riferimento adottato: nel nostro caso si è scelto un sistema di riferimento destrorso, con l'asse z rivolto verso l'alto, l'asse x verso destra e l'asse y frontalmente (Figura 7.6).

Infine calcoliamo $\mathbf{x_o}$, trasformando il punto $\mathbf{x_s}$ della sfera nel corrispondente punto $\mathbf{x_o}$ dell'immagine omnidirezionale, con la funzione `world2cam` corrispondente al modello di camera in uso (vedi Capitolo 2):

$$\mathbf{x_o} = \texttt{world2cam}(\mathbf{x_s}).$$

La trasformazione cilindrica descritta, considerando la direzione verticale dell'immagine panoramica, applica una proiezione prospettica su una superficie la cui normale in coordinate sferiche ha sempre *elevation* nulla. Se il campo visivo verticale della camera è piuttosto ampio, questa proiezione provoca una cambiamento di scala delle regioni che si trovano ad angoli di *elevation* molto distanti dallo zero. Dato che l'obiettivo della trasformazione panoramica introdotta è il tracking delle patch, questo cambiamento di scala rende particolarmente difficile l'operazione di matching delle feature. Si può quindi pensare di sostituire la superficie del cilindro con la superficie della sfera di raggio unitario, risolvendo questo problema di scala, ma introducendo ulteriore distorsione.

Per effettuare questa diversa trasformazione è sufficiente sostituire le Equazioni 7.1 e 7.2 con le Equazioni 7.3 e 7.4 seguenti:

$$h = \frac{\varphi_M - \varphi_m}{2\pi} w \tag{7.3}$$

$$\begin{aligned} \vartheta &= x_p \gamma \\ \varphi &= \arctan(\varphi_M - y_p \gamma). \end{aligned} \tag{7.4}$$

La Figura 7.7 mostra l'applicazione delle due trasformazioni a un'immagine omnidirezionale. Osservando nella parte destra dell'immagine le linee del marciapiede, che nel mondo tridimensionale sono rette, è possibile notare come la distorsione di questa seconda trasformazione introdotta sia più marcata.

È facilmente verificabile che un immagine panoramica è più simile a una prospettica, rispetto alla sua versione omnidirezionale, anche se la distorsione rimane evidente: linee rette del mondo reale sarebbero preservate in immagini prospettiche, ma ciò non accade in quelle panoramiche, nelle quali vengono trasformate in linee curve.

(a) *Immagine omnidirezionale*

(b) *Immagine panoramica da sviluppo cilindrico*

(c) *Immagine panoramica da sviluppo sferico*

Figura 7.7: *Immagine omnidirezionale e rispettiva immagine panoramica, con sviluppo cilindrico o sferico*

Data la "somiglianza" fra immagini panoramiche e prospettiche, è possibile tentar di utilizzare le tecniche di tracking sviluppate per il caso prospettico su questa nuova rappresentazione, con un piccolo accorgimento: dato che l'immagine panoramica è lo "srotolamento" del cilindro sul quale l'immagine sferica è stata proiettata, è chiaro che i suoi lati destro e sinistro sono in realtà contigui, quindi un punto caratteristico potrà scomparire da un lato e rientrare dall'altro.

Assumendo ora di essere in grado di effettuare il tracking di feature su immagini panoramiche, e ricordando che il nostro obiettivo iniziale era il tracking omnidirezionale, le posizioni delle feature panoramiche possono essere in qualsiasi momento convertite nelle corrispondenti posizioni sull'immagine omnidirezionale, applicando le equazioni descritte in precedenza.

È lecito chiedersi se lo sviluppo di una immagine panoramica da una omnidirezionale non abbia un costo computazionale troppo elevato, essendo necessario farlo per ogni frame della sequenza di tracking. Per rispondere a questa domanda si tenga presente che per ogni pixel dell'immagine panoramica, è necessario effettuare i calcoli sopra descritti. Fortunatamente questo calcolo oneroso necessita di essere effettuato solo per il primo sviluppo, in quanto nei frame successivi, sebbene l'immagine da trasformare cambi, la mappatura che lega le coordinate di un punto sull'immagine

panoramica a quelle sull'immagine omnidirezionale rimane invariata. È quindi importante archiviare questa mappa, anche chiamata LUT (*Lookup Table*), dopo la sua costruzione, e sfruttarla in tutti i frame successivi al primo. La *Lookup Table* è formata da due matrici `LUTx` e `LUTy` di valori *float* della stessa dimensione dell'immagine panoramica: il valore del generico elemento (x,y) di `LUTx` identifica il valore di ascissa sull'immagine omnidirezionale del punto corrispondente al pixel (x,y) dell'immagine panoramica; allo stesso modo `LUTy` conserva i valori delle ordinate. La dimensione della memoria occupata dalla *Lookup Table* è quindi calcolabile come:

$$\texttt{dim}(LUT) = 2[\text{matrici}] \times w[\text{px}] \times h[\text{px}] \times 4 \left[\frac{\text{bytes}}{\text{float}}\right] = 8 \cdot h \cdot w \text{ [bytes]}$$

In definitiva, con l'utilizzo di questa soluzione, la trasformazione di una immagine omnidirezionale in una panoramica viene fatta molto efficientemente, a un costo computazionale contenuto.

7.2 Tracking con patch prospettiche

Una camera omnidirezionale con un unico centro di proiezione, è in grado di ricostruire immagini prospettiche geometricamente corrette di una porzione del suo campo visivo. Un esempio di tale trasformazione è mostrata in Figura 7.8. Sfruttando questa proprietà si è sviluppata una tecnica di tracking su immagini omnidirezionali composta dai seguenti passi:

- I punti caratteristici vengono estratti direttamente dall'immagine omnidirezionale con uno dei metodi noti (es. Harris corner detector)
- L'area circostante ogni punto caratteristico viene trasformata in immagine prospettica, e da questa estratta la patch che descrive la feature.
- Nei frame successivi, le aree dell'immagine nell'intorno delle posizioni dei punti caratteristici rilevate al frame precedente, vengono a loro volta trasformate in immagini prospettiche, e su queste viene effettuato il matching delle patch.
- Dalle nuove posizioni delle feature all'interno delle trasformazioni prospettiche prodotte al punto precedente, si risale alle corrispondenti posizioni all'interno dell'immagine omnidirezionale.

In questo modo il tracking su immagini omnidirezionali viene riportato a un tracking su immagini prospettiche svolto a livello locale singolarmente per ogni punto caratteristico, beneficiando dei vantaggi in termini di distorsione che le immagini prospettiche posseggono.

Occupiamoci ora di dettagliare l'operazione che trasforma una porzione di immagine omnidirezionale nella corrispondente prospettica. Interpretiamo nuovamente l'immagine omnidirezionale come la superficie di una sfera di raggio unitario (Figura 7.1) e immaginiamo di applicare un piano su tale superficie in modo che il suo punto centrale sia tangente alla sfera nel centro dell'area che si desidera sviluppare. Proiettiamo infine l'immagine sferica sul piano, ottenendo l'immagine prospettica (vedi Figura 7.9).

Anche in questo caso, come per la trasformazione panoramica, il processo viene svolto calcolando per ogni pixel dell'immagine destinazione la corrispondente posizione nell'immagine omnidirezionale sorgente, creando una mappa di posizioni corrispondenti. Consideriamo quindi la proiezione inversa di un generico punto $\mathbf{x}_p$ dell'immagine prospettica e sviluppiamo i passaggi matematici che ci permettono di trovare il corrispondente punto $\mathbf{x}_o$ sull'immagine omnidirezionale.

Definiamo w e h la larghezza e l'altezza in pixel dell'immagine prospettica. È necessario scegliere questi parametri come numeri dispari, in modo da avere un unico pixel centrale $\mathbf{x}_c$ che corrisponda al pixel centrale dell'area dell'immagine omnidirezionale che si desidera trasformare. Questi due parametri definiranno solamente la risoluzione dell'immagine, non l'ampiezza della corrispondente regione dell'immagine omnidirezionale (a parità di rapporto h/w). Il parametro che definisce la

Figura 7.8: *Ricostruzione di immagini prospettiche da un'immagine omnidirezionale*

dimensione di tale regione è il campo visivo orizzontale fov (*field of view*) espresso in radianti. Il campo visivo verticale sarà proporzionale a quello orizzontale secondo il rapporto h/w.

Le coordinate in pixel del punto centrale dell'immagine prospettica saranno quindi:

$$x_c = \frac{w-1}{2} \qquad y_c = \frac{h-1}{2}.$$

Supponiamo ora di voler trasformare una regione dell'immagine omnidirezionale nell'intorno di un generico punto $\mathbf{x}_{oc}$. Applichiamo la funzione `cam2world` ottenendo il corrispondente punto sulla sfera di raggio unitario che identifica la direzione del raggio luminoso che ha creato il punto $\mathbf{x}_{oc}$ (come spiegato nel Capitolo 2). Trasformiamo il punto così ottenuto in coordinate sferiche ϑ_c e φ_c, *azimuth* ed *elevation* rispettivamente.

Tralasciando momentaneamente l'informazione appena ottenuta, supponiamo che l'area che vogliamo trasformare abbia come punto principale quello corrispondente a $\vartheta = \varphi = 0$, come mostrato in Figura 7.10.

Calcoliamo il rapporto che lega pixel e unità di misura metrica, considerando che il corrispondente di x_c (espresso in pixel) è $\tan(fov/2)$, come mostrato in Figura 7.11. Il fattore di scala che lega le

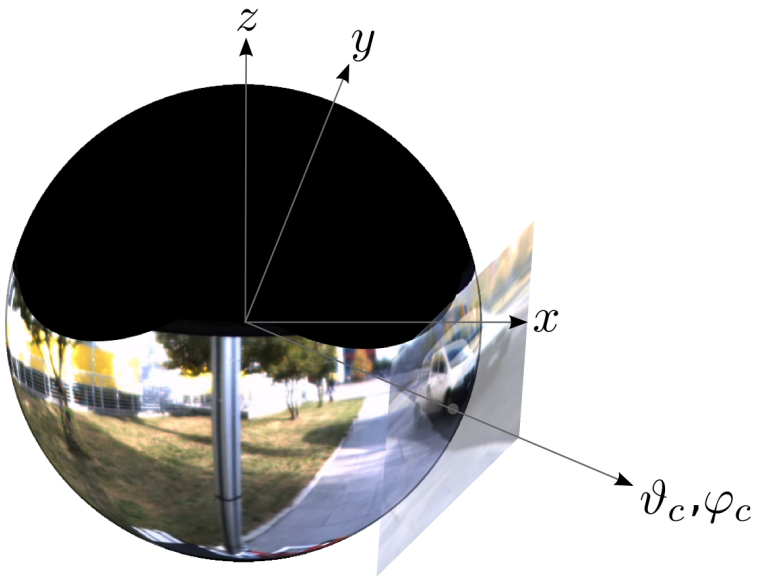


Figura 7.9: *Sviluppo di un immagine prospettica a partire da un'immagine omnidirezionale su sfera*

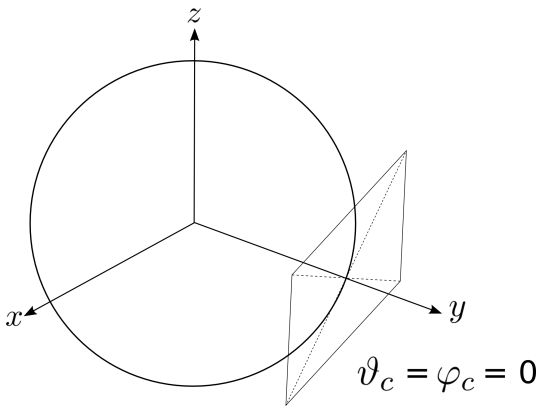


Figura 7.10: *Schema per la creazione di un immagine prospettica in direzione* $\vartheta_c = \varphi_c = 0$

due grandezze è quindi:

$$\gamma = \frac{\tan(fov/2)}{x_c}. \tag{7.5}$$

Calcoliamo ora le coordinate metriche 3D del generico punto $\mathbf{x}_p$ dell'immagine prospettica nella configurazione di Figura 7.10, nel nostro sistema di riferimento (Figura 7.6), ottenendo il punto $\mathbf{x}_{pm0}$:

$$\mathbf{x}_{pm0} = \begin{bmatrix} x_{pm0} \\ y_{pm0} \\ z_{pm0} \end{bmatrix} = \begin{bmatrix} (x_p - x_c)\gamma \\ 1 \\ (-y_p + y_c)\gamma \end{bmatrix}.$$

La coordinata y è fissa a 1 in quanto il piano dell'immagine prospettica è perpendicolare all'asse y e lo incrocia a distanza 1 (raggio della sfera). Il calcolo delle altre due coordinate è illustrato in Figura 7.12. Il pedice $pm0$ sta a indicare che il punto dell'immagine prospettica è in coordinate metriche nella configurazione $\vartheta = \varphi = 0$.

A questo punto, notiamo che la differenza delle posizioni metriche dei punti dell'immagine prospettica fra il caso particolare $\vartheta_c = \varphi_c = 0$ e il caso generale, risiede sostanzialmente in una operazione di rotazione che porta il raggio principale nella direzione stabilita in precedenza (ϑ_c e φ_c). Applicando quindi questa operazione al generico punto $\mathbf{x}_{pm0}$, si ottiene il punto $\mathbf{x}_{pm}$:

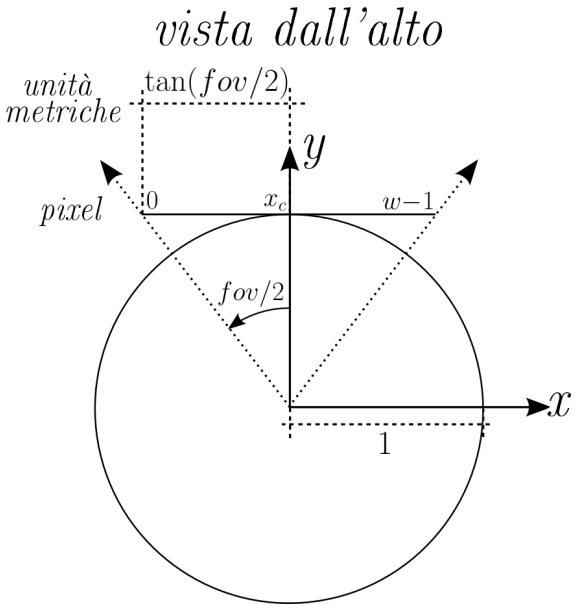


Figura 7.11: *Schema per il calcolo del rapporto fra unità metrica e pixel nella creazione di un'immagine prospettica*

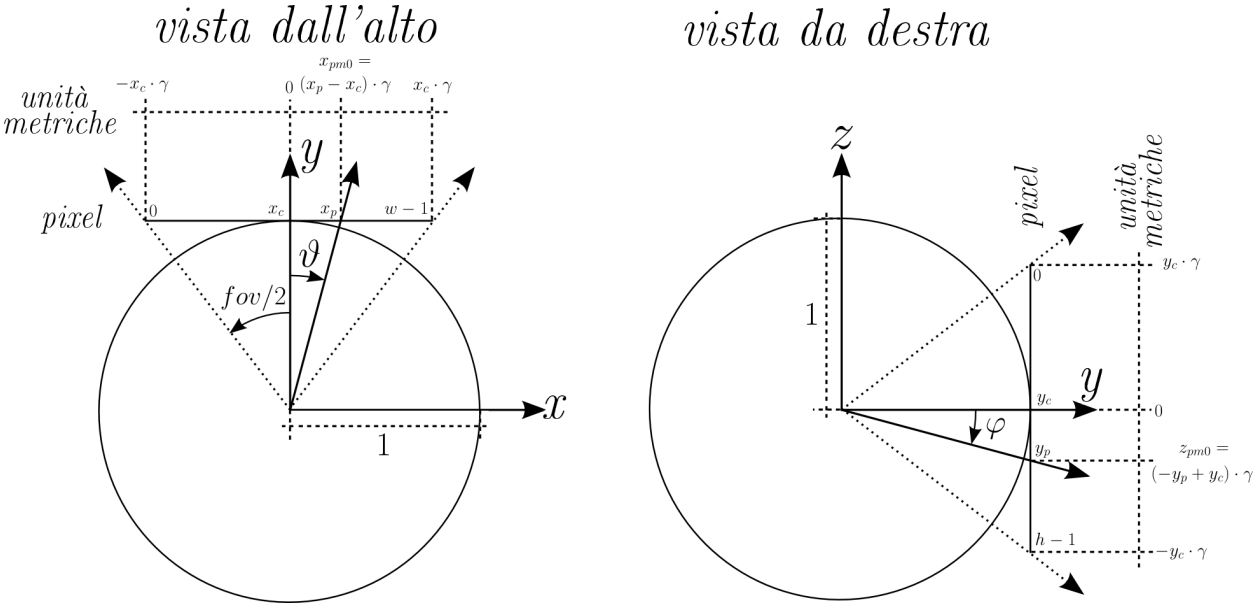


Figura 7.12: *Schema per il calcolo delle coordinate metriche di un punto dell'immagine prospettica, con* $\vartheta_c = \varphi_c = 0$ *e nel sistema di riferimento di Figura 7.6*

$$\mathbf{x}_{pm} = \begin{bmatrix} x_{pm} \\ y_{pm} \\ z_{pm} \end{bmatrix} = R\mathbf{x}_{pm0} = \begin{bmatrix} \cos\vartheta_c & \cos\varphi_c \sin\vartheta_c & -\sin\varphi_c \sin\vartheta_c \\ -\sin\vartheta_c & \cos\varphi_c \cos\vartheta_c & -\sin\varphi_c \cos\vartheta_c \\ 0 & \sin\varphi_c & \cos\varphi_c \end{bmatrix} \mathbf{x}_{pm0}.$$

Infine, per ottenere il corrispondente punto $\mathbf{x}_o$ sull'immagine omnidirezionale è sufficiente applicare la funzione `world2cam` al punto $\mathbf{x}_{pm}$ (vedi Capitolo 2).

Applicando le equazioni appena descritte a tutti i pixel dell'immagine prospettica, si ottiene una *Lookup Table* che applicata all'immagine omnidirezionale produce la corrispondente prospettica.

Ricordando che il nostro obiettivo è quello di effettuare il tracking omnidirezionale, una volta che la patch prospettica è stata associata all'interno dell'area di ricerca (anch'essa prospettica), è necessario tradurre la nuova posizione della feature dalle coordinate dell'area di ricerca prospettica alle corrispondenti coordinate nell'immagine omnidirezionale. Questo può essere fatto semplicemente applicando nuovamente le equazioni precedenti.

Sfortunatamente il guadagno in efficienza che si otteneva nella costruzione di immagini panoramiche, sfruttando molteplici volte la LUT (*Lookup Table*), in questo caso non sussiste, in quanto la variazione della direzione del raggio principale (ϑ_c e φ_c) modifica completamente la LUT da applicare.

Mauthner nel suo articolo [MFB06] adotta essenzialmente questo metodo, con la differenza che il tracking che effettua non è di semplici feature, bensì di intere regioni estratte con MSER (*Maximally Stable Extremal Regions*). Tali regioni vengono trasformate in immagini prospettiche e successivamente, la fase di matching viene effettuata usando il *warping* e i descrittori SIFT.

Guitierrez [GRMG11] approssima invece il metodo da noi descritto lavorando direttamente sull'immagine omnidirezionale, a discapito di passaggi matematici ben più complicati. Nel suo metodo Gutierrez si trova inoltre a dover prendere in considerazione il diverso fattore di scala intrinseco nelle camere omnidirezionali, dipendente dalla forma dello specchio. Nel nostro metodo questo problema non sussiste, perché ragioniamo sull'astrazione della sfera che non presenta variazioni di scala. A riprova di questo fatto si noti in Figura 7.8 che le regioni dell'immagine omnidirezionale hanno dimensioni diverse, sebbene tutte abbiano un campo visivo orizzontale di 40 gradi. Un ulteriore problema con il quale ha a che fare Gutierrez è la rotazione delle patch, in quanto una rotazione della camera su se stessa intorno all'asse z, evento tutt'altro che raro, provoca una conseguente rotazione della regione dell'immagine intorno alla feature, oltre che una traslazione. Anche in questo caso, nella nostra soluzione questo problema non compare, in quanto una variazione dell'angolo ϑ_c provoca solamente una rotazione della LUT da applicare, ma l'immagine prospettica risultante rimane coerente con la patch inizialmente estratta. Sempre in Figura 7.8 si può verificare che le tre immagini prospettiche sono orientate normalmente, sebbene le corrispondenti aree dell'immagine omnidirezionale abbiano orientazioni differenti.

7.2.1 Tracking con patch prospettiche, con fattore di scala stimato da SLAM

La procedura di tracking con patch prospettiche descritta finora è applicabile indipendentemente dall'uso che si intende fare delle corrispondenze fra feature prodotte dall'algoritmo. Nel particolare caso dello SLAM, è possibile sfruttare le informazioni sulla posizione della camera e dei landmark per migliorare ulteriormente il tracking. Quello che si intende fare è risolvere il problema del cambiamento di scala che si ha nell'osservare uno stesso landmark da distanze differenti: si pensi ad esempio a un robot che si muove frontalmente e inizializza un nuovo landmark molto distante; nei frame successivi il robot si avvicinerà progressivamente al landmark, e la regione dell'immagine corrispondente alla patch associata al landmark diventerà maggiore.

Grazie allo SLAM, è possibile stimare le coordinate tridimensionali del landmark e quelle della camera nell'istante attuale e in quello in cui il landmark è stato inizializzato ed è stata estratta la

patch. Con queste informazioni è possibile calcolare il cambiamento di scala e con esso ingrandire o ridurre l'area di ricerca prospettica nella quale ricercare la patch, in modo che entrambe abbiano lo stesso fattore di scala. In realtà, la dimensione in termini metrici dell'area di ricerca non viene modifica da questo cambiamento di scala, bensì la sua risoluzione.

Definiamo $\mathbf{c_1}$ la posizione che assumeva la camera quando ha inizializzato il landmark, $\mathbf{c_2}$ la sua attuale posizione e $\mathbf{l}$ le coordinate del landmark. $\mathbf{c_1}$, $\mathbf{c_2}$ e $\mathbf{l}$ devono essere espressi rispetto a uno stesso sistema di riferimento, generalmente il sistema di riferimento mondo. Le distanze della camera dal landmark nei due istanti di tempo sono calcolabili come

$$d_{c_1-l} = ||\mathbf{c_1} - \mathbf{l}|| \qquad d_{c_2-l} = ||\mathbf{c_2} - \mathbf{l}||$$

e il fattore di cambiamento di scala diventa

$$d_{fact} = \frac{d_{c_1-l}}{d_{c_2-l}}.$$

Nel tracking con patch prospettiche nella sua versione base, il fattore γ (Eq. 7.5 e Figura 7.11) che definisce il rapporto fra pixel e unità metriche, cioè la risoluzione di un'immagine prospettica, è lo stesso sia per la creazione delle patch che per la trasformazione delle aree di ricerca. Il suo valore è determinato esclusivamente dal campo visivo orizzontale della patch e dalla sua larghezza in pixel ($patchFOV$ e $patchWidth$). Volendo prendere in considerazione il cambiamento del fattore di scala fra la patch e l'attuale area di ricerca, nella trasformazione dell'area di ricerca in immagine prospettica è necessario calcolare un nuovo fattore di risoluzione:

$$\gamma_{sa} = \gamma \cdot d_{fact} = \frac{2\tan(patchFOV/2)}{patchWidth} \cdot d_{fact}$$

dove γ_{sa} è il fattore di risoluzione dell'area di ricerca e γ è il fattore di risoluzione della patch.

Con il nuovo fattore di risoluzione è possibile calcolare la dimensione in pixel dell'area di ricerca:

$$searchAreaWidth = \frac{2 \cdot \tan(searchAreaFOVHorizontal/2)}{\gamma_{sa}}$$

$$searchAreaHeight = \frac{2 \cdot \tan(searchAreaFOVVertical/2)}{\gamma_{sa}}$$

dove $searchAreaFOVHorizontal$ e $searchAreaFOVVertical$ definiscono l'ampiezza dell'area di ricerca in termini di angolo visivo orizzontale e verticale rispettivamente.

7.3 Patch Warping

Nei due metodi di *patch tracking* illustrati nelle sezioni precedenti, come d'altra parte anche nei classici metodi in ambito prospettico, la patch associata a una feature viene creata al primo rilevamento della feature stessa e utilizzata nei frame successivi per monitorarne lo spostamento. Senza ulteriori elaborazioni, questo metodo funziona bene soltanto in caso di piccoli cambi di prospettiva e rotazione della camera intorno al proprio asse verticale e orizzontale[1]. Ciò accade perché l'osservazione della stessa scena da prospettive diverse ne altera notevolmente l'immagine percepita, anche a livello locale in piccole porzioni dell'immagine come nel caso delle patch.

[1] Considerando la camera con un orientamento standard, la sua rotazione intorno all'asse verticale provoca approssimativamente una traslazione orizzontale dell'immagine, mentre una rotazione intorno all'asse orizzontale ne provoca una traslazione verticale. È da escludere invece il caso di rotazione intorno al terzo asse (frontale), che provocherebbe invece una rotazione dell'immagine.

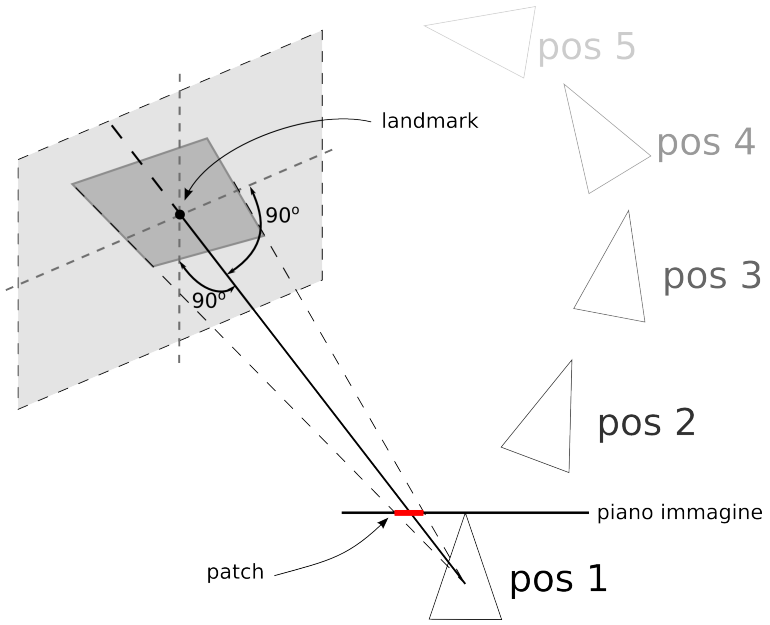


Figura 7.13: *Schema dell'ipotesi di planarità della porzione di mondo rappresentata da una patch nel metodo del Patch Warping*

Questo problema provoca un tracking di scarsa qualità, nel quale le feature vengono seguite per brevi tratti e poi perse, a causa dell'incompatibilità fra patch e immagine attuale. Questo inconveniente si ripercuote a cascata sulla qualità dello SLAM che si intende sviluppare, in quanto il tempo di vita delle feature è di fondamentale importanza affinché i landmark associati alle patch si stabilizzino e la loro posizione venga stimata con precisione.

Immaginiamo ora di conoscere l'esatta conformazione del mondo e la posizione e l'orientamento della camera al suo interno, in ogni istante. In queste condizioni, sarebbe possibile calcolare l'aspetto dell'immagine catturata dalla camera (esattamente ciò che accade nei videogiochi 3D). Purtroppo la nostra informazione sul mondo deriva solamente dalle immagini catturate dalla camera, le quali non ci danno nessuna informazione di profondità e quindi nessuna informazione sulla struttura tridimensionale del mondo. Se decidessimo invece di integrare questa idea nel processo di SLAM, alcune informazioni diverrebbero disponibili: la posizione della camera e la posizione di alcuni punti del mondo (landmark). Chiaramente queste informazioni non sono sufficienti per realizzare il nostro obiettivo, ma con alcune ipotesi e approssimazioni è possibile realizzare il processo di "patch warping", il quale porta a ricostruire il nuovo aspetto di una patch al cambiare del punto di vista, effettuando opportune operazioni sulla patch originale catturata in fase di inizializzazione della corrispondente feature.

L'ipotesi che viene fatta è quella di planarità della porzione di mondo catturata da una patch, il cui vettore normale ha la stessa direzione del raggio che va dal centro ottico della camera al landmark associato a tale patch[2], considerando l'istante in cui il landmark viene creato, come illustrato in Figura 7.13. Si suppone inoltre che le informazioni sulla posizione del landmark e della camera in tutti gli istanti di tempo siano conosciuti con precisione.

L'assunzione di planarità fatta è molto forte e quasi mai verificata nella realtà, ma in assenza di ulteriore informazione è l'unica ipotesi plausibile da fare, la quale ci permette di calcolare il nuovo aspetto delle patch nel tempo. L'ipotesi di precisa conoscenza della posizione del landmark risulta invece assolutamente violata nei suoi primi frame di vita, in quanto si ha una informazione di scarsissima qualità sulla sua distanza dalla camera, ma in queste condizioni, essendoci un trascurabile cambio di prospettiva, il warping della patch produrrà essenzialmente lo stesso aspetto iniziale,

[2]Esistono dei lavori che tentano di stimare la normale del piano delle patch, come quello presentato nell'articolo di Molton, Davison e Reid [MDR04].

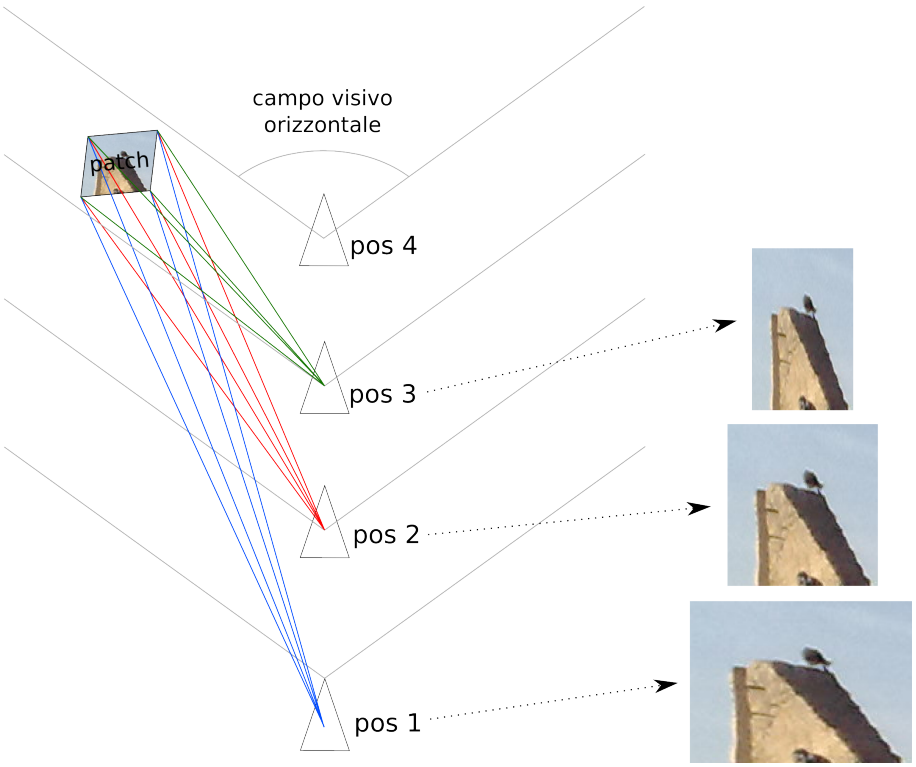


Figura 7.14: *Patch warping con camera prospettica: la patch viene inizializzata quando la camera è in posizione 1; Nelle posizioni 2 e 3 la patch non viene più osservata frontalmente e il suo aspetto ne risulta alterato. Nella posizione 4 la patch è uscita dal campo visivo*

a patto di inizializzare il landmark sufficientemente lontano. Quando il cambio di prospettiva si fa rilevante, la stima della posizione del landmark diventa più precisa e quindi l'ipotesi iniziale è verificata.

Considerando il caso di camera prospettica, nel quale il robot si muove frontalmente, come si può vedere in Figura 7.14 l'assunzione di planarità è ancora accettabile, in quanto quando il cambio di prospettiva diventa tale da rendere il piano della patch osservabile da un punto di vista molto critico, in realtà il landmark è già uscito dal campo di visione della camera. Chiaramente se la patch fosse stata inizializzata come in Figura 7.14 ma il movimento del robot non fosse prettamente frontale e il landmark non uscisse dal campo visivo, il warping della patch fallirebbe e il punto verrebbe perso. Per risolvere almeno in parte questo problema, si sceglie di cambiare l'orientamento del piano all'ampliarsi del cambio di prospettiva, considerando la normale al piano come la media fra il raggio uscente dalla camera in fase di inizializzazione e quello corrente, ottenuto grazie alla stima della posizione e orientamento della camera e del landmark (vedi Figura 7.15).

Le informazioni in nostro possesso sono la posizione e orientamento della camera rispetto al mondo nella fase di inizializzazione al tempo t_i ($\mathbf{t}^i_{wc}$ e $\mathbf{r}^i_{wc}$) e corrente al tempo t_c ($\mathbf{t}^c_{wc}$ e $\mathbf{r}^c_{wc}$), la posizione in coordinate mondo del landmark al tempo t_c ($\mathbf{l}_w$), le coordinate in pixel del punto dell'immagine al tempo t_i che ha dato origine al landmark ($\mathbf{p}^i$) e le coordinate del punto in cui stimo di poterlo osservare attualmente ($\mathbf{p}^c$).

Organizziamo le informazioni su posizione e orientamento della camera ai tempi t_i e t_c in due matrici 4×4:

$$H^i_{wc} = \begin{bmatrix} \mathbf{r}^i_{wc} & \mathbf{t}^i_{wc} \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} \qquad H^c_{wc} = \begin{bmatrix} \mathbf{r}^c_{wc} & \mathbf{t}^c_{wc} \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times4}$$

Le matrici H^i_{wc} e H^c_{wc} applicate a un punto 3D in coordinate omogenee, nel sistema di riferimento camera al tempo t_i e t_c rispettivamente, lo trasformano in coordinate mondo.

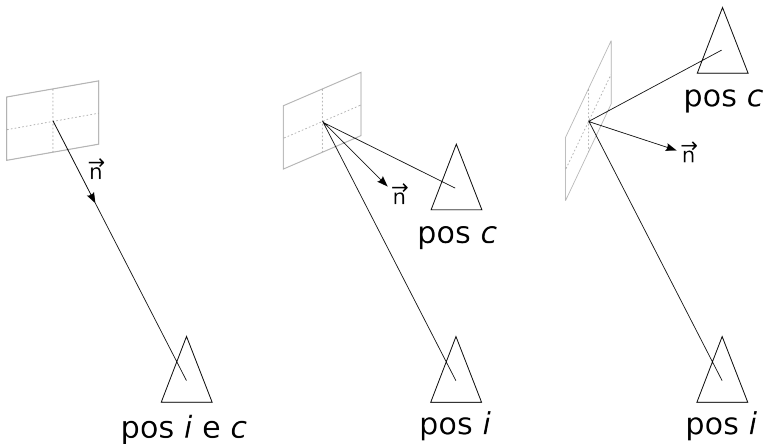


Figura 7.15: *Schema per il calcolo della normale del piano della patch come media fra il raggio iniziale e attuale*

Definiamo la seguente matrice:

$$H = (H^i_{wc})^{-1} \cdot H^c_{wc} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \qquad \in \mathbb{R}^{4\times4}$$

La matrice H così definita permette di trasformare un punto 3D dal sistema di riferimento camera al tempo t_c, al corrispondente punto nel sistema di riferimento camera al tempo t_i.

Per poter calcolare la normale del piano corrispondente alla patch, come media fra i due raggi tra camera e landmark al tempo t_i e t_c, cominciamo con il recupero della direzione di questi raggi:

$$\overrightarrow{\mathbf{n}^i} = -\texttt{cam2world}(\mathbf{p}^i) \quad \in \mathbb{R}^{3\times1} \qquad\qquad \overrightarrow{\mathbf{n}^c_t} = -\texttt{cam2world}(\mathbf{p}^c) \quad \in \mathbb{R}^{3\times1}$$

Abbiamo messo il segno negativo davanti alle due formule in modo da considerare le direzioni dei raggi come uscenti dal piano della patch in direzione della camera. Il primo raggio $\overrightarrow{\mathbf{n}^i}$ è in coordinate camera al tempo t_i, mentre $\overrightarrow{\mathbf{n}^c_t}$ è in coordinate camera al tempo t_c e necessita di essere trasformato utilizzando la matrice H (il pedice t significa temporaneo):

$$\overrightarrow{\mathbf{n}^c} = R \cdot \overrightarrow{\mathbf{n}^c_t} + T \qquad \in \mathbb{R}^{3\times1}$$

La normale al piano sarà quindi:

$$\overrightarrow{\mathbf{n}} = \frac{\overrightarrow{\mathbf{n}^i} + \overrightarrow{\mathbf{n}^c}}{\|\overrightarrow{\mathbf{n}^i} + \overrightarrow{\mathbf{n}^c}\|} \qquad \in \mathbb{R}^{3\times1}$$

Continuando a lavorare nel sistema di riferimento camera al tempo t_i, calcoliamo la posizione del landmark $\mathbf{l}_w$ in tale sistema di riferimento:

$$\mathbf{l}_c = \texttt{homogeneus2cartesian}\left((H^i_{wc})^{-1} \begin{bmatrix} \mathbf{l}_w \\ 1 \end{bmatrix}\right) \qquad \in \mathbb{R}^{3\times1}$$

dove `homogeneus2cartesian` è una funzione che trasforma un punto da coordinate omogenee a coordinate cartesiane.

Conosciamo ora il punto $\mathbf{l}_c$ che appartiene al piano e conosciamo la sua normale. Per ottenere una rappresentazione standard del piano calcoliamo ora la distanza dall'origine (centro ottico della camera al tempo t_i) lungo la normale:

$$d = -\vec{\mathbf{n}} \cdot \mathbf{l}_c$$

Abbiamo ora tutti gli ingredienti necessari per definire la matrice omografica che ci permette di mappare un punto percepito dalla camera al tempo t_i nel corrispondente punto percepito al tempo t_c:

$$H_{ic} = R - \frac{T \cdot \vec{\mathbf{n}}^T}{d} \qquad \in \mathbb{R}^{3\times 3}$$

Più specificamente la matrice H_{ic} lega le direzioni dei punti percepiti ed è quindi necessario utilizzare le funzioni `word2cam` e `cam2world` per mappare i pixel da un'immagine all'altra secondo le formule:

$$\begin{aligned} \mathbf{p}^c_{generic} &= \texttt{world2cam}(H_{ic} \cdot \texttt{cam2world}(\mathbf{p}^i_{generic})) \\ \mathbf{p}^i_{generic} &= \texttt{world2cam}(H_{ci} \cdot \texttt{cam2world}(\mathbf{p}^c_{generic})) \end{aligned} \qquad \text{dove} \quad H_{ci} = (H_{ic})^{-1}$$

Con queste equazioni è quindi ora possibile calcolare la *lookup table* che mappa i punti della nuova patch in quelli della patch originale.

Consideriamo che generalmente si preferisce avere una patch di dimensione costante in tutti gli istanti di tempo, quindi quello che realmente si fa è in fase di inizializzazione di archiviare un'area dell'immagine intorno al punto caratteristico maggiore della patch, in modo che la distorsione che interverrà nei frame successivi ci consentirà comunque di ricavare una patch con la dimensione costante scelta, e coperta in ogni sua parte. La copertura della patch viene meno quando il cambio di prospettiva è notevole e tale che l'informazione necessaria non rientra nemmeno nell'area di immagine iniziale scelta appositamente maggiore della patch. Si potrebbe quindi pensare di ampliare ulteriormente tale area, ma consideriamo che più ci si allontana dalla feature centrale meno valgono le ipotesi di planarità fatte in precedenza.

Il procedimento matematico appena descritto vale indipendentemente dal tipo di camera utilizzato e l'unica differenza fra il caso prospettico e omnidirezionale risiede nelle due funzioni `world2cam` e `cam2world` da applicare.

Ricordando il grande vantaggio delle camere omnidirezionali di possedere un campo visivo orizzontale di 360°, in linea teorica dovrebbe permetterci di seguire un punto caratteristico molto a lungo. Consideriamo nuovamente il caso di Figura 7.14, sostituendo la camera prospettica con una omnidirezionale: in posizione 4 la feature continuerebbe a rimanere nel campo visivo della camera e ci rimarrebbe anche nelle successive posizioni che si avrebbero continuando il moto frontale. Questo fenomeno purtroppo dà luogo a un notevole cambio di prospettiva, il quale causa una trasformazione della patch molto scadente e il punto viene perso. Un esempio di questo fenomeno è mostrato nella sequenza di immagini di Figura 7.16.

Riassumendo, il metodo del patch warping integrato nel processo di SLAM permette di seguire più agevolmente le feature dell'immagine rispetto al patch tracking di base classico, sia con camera prospettica che omnidirezionale. Nel caso omnidirezionale si ha il vantaggio di operare direttamente sull'immagine originale, a differenza dei due metodi dei paragrafi precedenti che richiedevano una rappresentazione intermedia.

Dal punto di vista computazionale sebbene si lavori direttamente sull'immagine omnidirezionale, è necessario anche in questo caso effettuare l'oneroso calcolo di una *lookup table* per ogni landmark, a ogni frame. Risulta comunque meno pesante rispetto al tracking con patch prospettiche, in quanto il calcolo della LUT e la relativa applicazione avviene per la patch, di dimensione contenuta, invece che per l'area di ricerca.

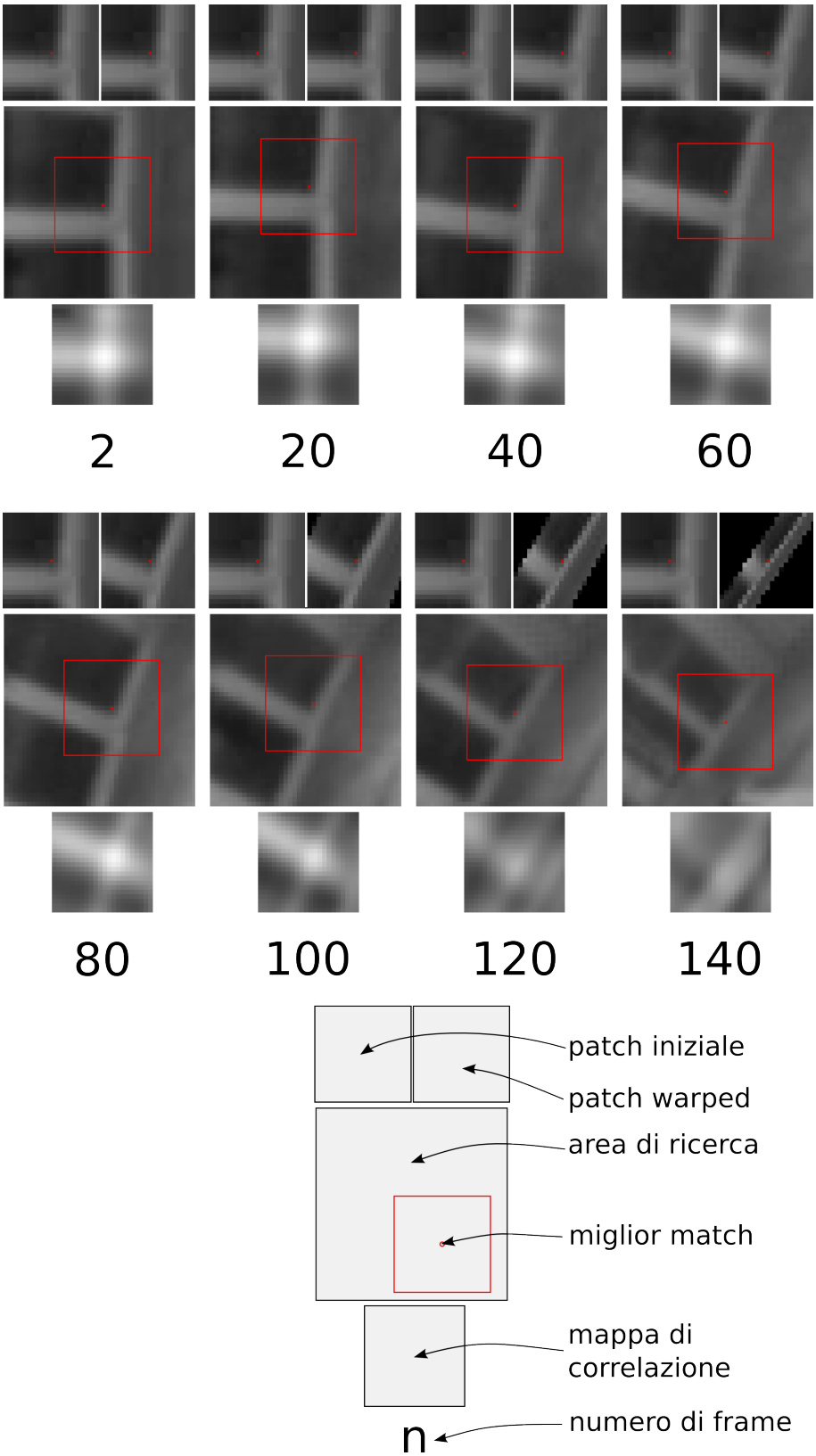


Figura 7.16: *Esempio di evoluzione di una patch: la sequenza ci mostra che la patch viene seguita ragionevolmente bene per i primi 80, 100 frame, dopodiché il cambio di prospettiva diventa tale da rendere il warping inefficace. La mappa di correlazione ci mostra il grado di adattamento della patch trasformata, all'interno dell'area di ricerca. Il bianco indica un buon adattamento mentre il nero ne indica uno cattivo. Le prime immagini della sequenza mostrano delle zone ad alta intensità di bianco nella mappa di correlazione, esprimendo un basso grado di incertezza sulla nuova osservazione del landmark. Nelle ultime immagini vi sono invece multiple zone a bassa intensità di bianco, indicando un'alta incertezza.*

7.4 Patch Update

I metodi di patch tracking spiegati in precedenza creano le patch nella fase di inizializzazione di nuove feature. Dipendendo dal metodo utilizzato, il processo di creazione di una nuova patch elabora l'area dell'immagine intorno alla feature in maniera diversa: nel patch tracking classico viene semplicemente salvata un'area rettangolare dell'immagine; nel patch tracking panoramico viene effettuata la stessa operazione, ma sulla trasformazione panoramica dell'immagine omnidirezionale; nel patch tracking omnidirezionale con patch prospettiche l'area intorno alla feature viene convertita nella corrispondente prospettica e infine nel patch warping un'area intorno alla feature di dimensione superiore alla patch viene salvata ed elaborata nella fase di matching di tutti i frame successivi. Tutti questi metodi partono dall'informazione fotometrica locale nell'inforno della feature nel frame di inizializzazione e tale informazione non viene aggiornata per tutta la durata del tracking di questa feature. Questa condizione impedisce che le feature possano essere seguite a lungo, in quanto un progressivo cambio di prospettiva cambia radicalmente la percezione della scena. Ciò che banalmente si potrebbe pensare è di aggiornare questa informazione a ogni frame: nel frame di inizializzazione viene creata la patch, nel frame successivo viene effettuata la ricerca con questa patch nell'area intorno alla posizione nella quale ci si aspetta si sia spostata la feature; se la patch risulta positivamente associata possiamo supporre di conoscerne il suo nuovo aspetto e quindi sostituirlo all'informazione precedente, in caso contrario non facciamo nessun tipo di operazione. Se il *matching* delle patch fosse sempre corretto, una feature potrebbe essere seguita per intero lungo tutta la sua traiettoria, finché non uscirebbe dal campo visivo o verrebbe occlusa. Purtroppo, essendo l'algoritmo di *matching* non così affidabile, capita che una patch venga associata in una posizione non corretta. In questo caso, se non aggiornassimo la patch, in alcuni frame il processo di matching restituirebbe una informazione scorretta, ma nei frame immediatamente successivi avrebbe modo di recuperare la posizione corretta. Se aggiornassimo le patch questo non potrebbe accadere in quanto un errore di matching sostituirebbe la patch con una sua versione scorretta che verrebbe utilizzata per il matching nel frame successivo.

Riassumendo, generalmente nelle tecniche di patch tracking conosciute non viene effettuato l'aggiornamento delle patch, in quanto gli errori e le imprecisioni del matching si propagano in tutti i frame successivi in maniera cumulativa.

Nel caso in cui si abbia un meccanismo per verificare la concordanza delle informazioni restituite dal processo di matching, come RANSAC o nel nostro caso 1PointRansac(vedi Sezione 5.5 del Capitolo 5), è possibile riconoscere ed escludere molti degli errori di matching. È quindi possibile pensare di effettuare l'aggiornamento delle patch solamente per quelle feature che passano il test di concordanza e che possono quindi essere considerate affidabili. Questa tecnica è quella che abbiamo chiamato *Patch Update*, nella quale abbiamo anche introdotto una soglia di correlazione apposita diversa da quella utilizzata per il matching, in modo che la patch venga sostituita soltanto nel caso in cui la sua nuova versione abbia un valore di correlazione superiore a tale soglia.

7.5 Inizializzazione dei landmark sul piano

Tutte le parametrizzazioni illustrate nel capitolo precedente, hanno la componente ρ che codifica la distanza inversa dei landmark rispetto alla posizione assunta della camera nella loro fase di inizializzazione. Dato che la camera non è in grado misurare nessuna informazione di profondità, alla creazione di un nuovo landmark tale componente viene inizializzata a un valore costante ρ^{init} alla quale viene associata un'alta incertezza σ_ρ^{init} anch'essa costante. Questo tipo di inizializzazione dispone i nuovi landmark nella direzione corretta, ma a una distanza fissa, confidando sul fatto che misurazioni successive facciano convergere la sua stima al valore corretto. Affinché questo avvenga è necessario che per ogni nuovo landmark ce ne siano molti altri già stabilizzati che permettano al processo di SLAM di stimare correttamente posizione e orientamento della camera, fintantoché il nuovo landmark si sia anch'esso stabilizzato. In caso contrario, in presenza di molti landmark recen-

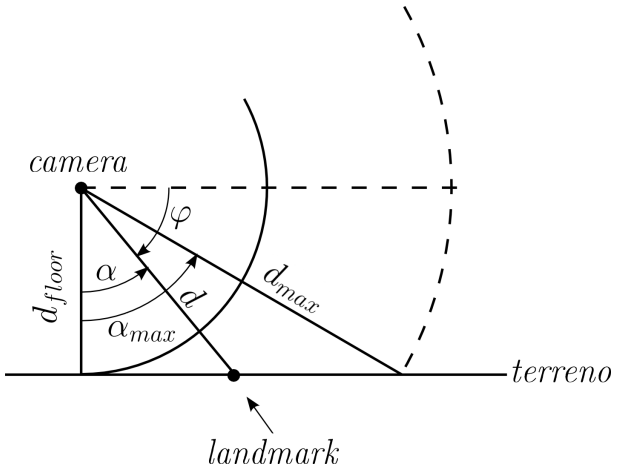


Figura 7.17: *Schema per l'inizializzazione dei landmark sul piano del terreno*

temente inizializzati e di una quantità di landmark stabilizzati non sufficiente, le stime di posizione e orientamento della camera risulterebbero sbagliate e a cascata anche quelle della posizione dei landmark. Ciò che si è riscontrato nelle applicazioni pratiche è che in questa condizione si ha una variazione del fattore di scala (*scale drift*).

Nel caso in cui il robot si muova localmente su un piano, sia esso un pavimento, una strada o altro, se si è a conoscenza dell'orientamento della camera rispetto a tale piano, è possibile attribuire un valore iniziale al parametro ρ in modo che i landmark vengano inizialmente disposti su di esso. Chiaramente non tutti i landmark possono essere inizializzati in questo modo, in quanto ve ne saranno alcuni sopra la linea dell'orizzonte la cui direzione non incrocia mai il piano. Inoltre anche landmark con direzione prossima all'orizzonte, che incrocerebbero il piano a distanza molto elevata, quasi infinita, difficilmente nella realtà risiedono sul piano, a meno che si stia svolgendo una sessione di SLAM in spazi molto aperti. Per tutti questi casi l'inizializzazione della distanza inversa continuerà a essere costante. La Figura 7.17 schematizza questo tipo di inizializzazione.

Definiamo d_{floor} la distanza del centro ottico della camera dal terreno sottostante. Definiamo α come l'angolo fra la normale al piano passante per il centro ottico della camera e la direzione del landmark. Definiamo α_{max} come l'estremo superiore di α dopo il quale il landmark non viene più inizializzato sul piano, ma a una distanza costante pari a d_{max}. d_{max} è la distanza corrispondente ad α_{max} come mostrato in Figura 7.17. d è invece la distanza alla quale inizializzare il landmark.

Considerando il caso particolare di camera omnidirezionale con orientamento standard, ossia con l'asse di rotazione perpendicolare al piano del terreno. Nelle fasi di inizializzazione di un nuovo landmark, abbiamo a disposizione le coordinate del pixel della feature associata e quindi la direzione del raggio che va dalla camera al landmark. Trasformiamo tale direzione in coordinate sferiche ϑ e φ, *azimuth* ed *elevation* rispettivamente. L'angolo α di Figura 7.17 può essere direttamente ricavato da φ come:

$$\alpha = \frac{\pi}{2} + \varphi.$$

Se avessimo una camera qualsiasi con un generico orientamento il ragionamento sarebbe analogo.

Nel caso in cui α sia minore di α_{max}, con semplici considerazioni geometriche ricaviamo che:

$$d_{floor} = d \cdot \cos\alpha \qquad \Longrightarrow \qquad d = \frac{d_{floor}}{\cos\alpha}.$$

Nel caso in cui α fosse maggiore o uguale ad α_{max}, la distanza del landmark diverrebbe semplicemente:

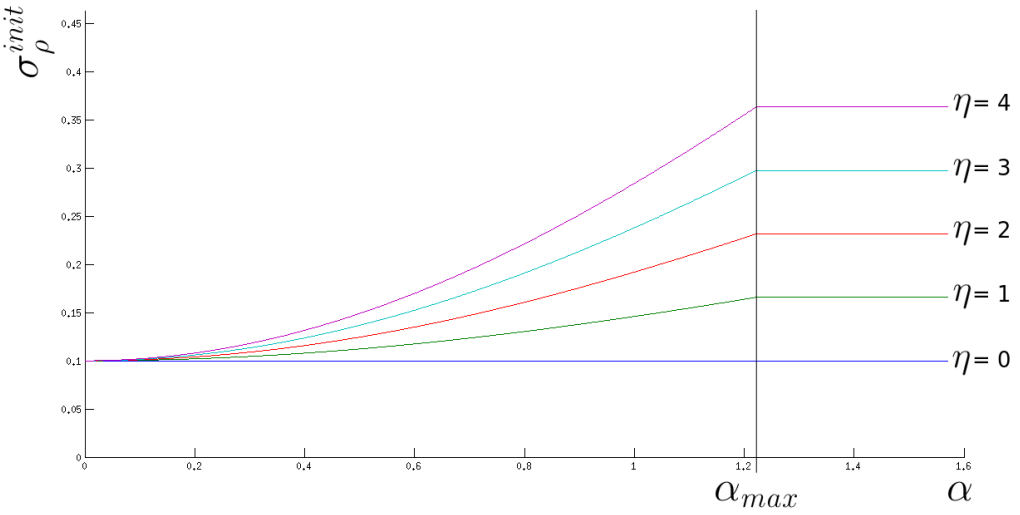


Figura 7.18: *Inizializzazione della deviazione standard della distanza inversa σ_ρ^{init} in funzione dell'angolo α. La funzione del grafico ha come parametri $\alpha_{max} = 70^o = 1.22rad$, $\sigma_{\rho_{floor}} = 0.1$ e alcuni valori di η*

$$d_{max} = \frac{d_{floor}}{\cos \alpha_{max}}.$$

Dato che le parametrizzazioni utilizzate per lo SLAM utilizzano la distanza inversa, è sufficiente porre il parametro $\rho_{init} = 1/d$.

L'assunzione che i landmark si trovino sul piano del terreno, è ragionevolmente verificata per valori di α bassi e perde progressivamente di validità al crescere dell'angolo. L'incertezza σ_ρ^{init} può quindi essere inizializzata in funzione dell'angolo α. La formula da noi adottata per il suo calcolo è la seguente:

$$\sigma_\rho^{init} = \begin{cases} \text{se} \quad \alpha < \alpha_{max} & [1 + \eta(1 - \cos\alpha)] \cdot \sigma_{\rho_{floor}} \\ \text{se} \quad \alpha \geq \alpha_{max} & [1 + \eta(1 - \cos\alpha_{max})] \cdot \sigma_{\rho_{floor}} \end{cases}$$

dove $\sigma_{\rho_{floor}}$ è il valore di deviazione standard minima che la funzione può assumere, in corrispondenza di $\alpha = 0$, mentre il parametro η è un fattore di scala. In Figura 7.18 è mostrato l'andamento della funzione per alcuni valori di η.

7.6 Riadattamento delle parametrizzazioni

Nel capitolo precedente sono state presentate una serie di parametrizzazioni per il *monocular* SLAM con camera prospettica. Tutte queste parametrizzazioni sono direttamente applicabili al caso omnidirezionale con opportune modifiche, a eccezione di FHP, la quale necessità di modifiche più profonde.

Osserviamo che in tutte le parametrizzazioni, nelle equazioni di creazione di un nuovo landmark e in quelle di misurazione, viene utilizzata la matrice dei parametri intrinseci della camera K e la sua inversa K^{-1}, per convertire l'informazione della posizione di un pixel nella direzione del raggio corrispondente e viceversa. Come spiegato nel Capitolo 2, nel caso omnidirezionale questa conversione è più complessa e non esprimibile con una semplice moltiplicazione per una matrice. Notiamo inoltre che nella spiegazione delle parametrizzazioni è stata completamente ignorata la distorsione radiale, la quale interviene in maniera significativa in questo processo di conversione. In definitiva, è necessario revisionare le equazioni del capitolo precedente, sostituendo ogni qual

volta si incontra una forma del tipo $K \cdot (\dots)$ con la corrispondente funzione `world2cam`$(\dots)$ e viceversa sostituire ogni occorrenza di una forma del tipo $K^{-1} \cdot (\dots)$ con la funzione `cam2world`$(\dots)$. Ricordiamo che queste funzioni trattano opportunamente anche la distorsione.

Considerando ora la parametrizzazione FHP, ricordiamo che fra i suoi dieci parametri vi sono u e v che esprimo la direzione del landmark nel sistema di riferimento camera. Questi due parametri riescono a esprimere una direzione in tre dimensioni grazie al fatto che la terza componente (z) assume implicitamente il valore costante 1. Questo avviene perché nel modello di camera prospettica la direzione dei punti visibili è esprimibile tramite il punto stesso convertito in coordinate metriche sul piano immagine normalizzato, che si trova a distanza 1 lungo la sua normale, che coincide con l'asse z del sistema di riferimento camera. Quello che è il piano normalizzato di una camera prospettica, nel caso omnidirezionale è la superficie di una sfera con raggio unitario. Per esprimere la direzione di un punto di questa superficie sembrerebbe necessario utilizzare tre componenti, ma sarebbe possibile anche usarne semplicemente due dato che la terza è vincolata dal fatto che il modulo del vettore direzione è costante e pari a 1. In questa seconda codifica si avrebbe però un'ambiguità di segno della componente implicita, a causa dell'operazione di estrazione della radice quadrata necessaria per la sua ricostruzione. Anche per tale motivo, attualmente si è scelto di realizzare la prima alternativa, che complessivamente richiede ben 11 parametri:

$$\mathbf{l} \quad = \quad [\mathbf{t_0} \quad \mathbf{q_0} \quad \overrightarrow{\mathbf{h}_c} \quad \rho]^T \quad = \quad [x_{t_0} \quad y_{t_0} \quad z_{t_0} \quad w_{q_0} \quad x_{q_0} \quad y_{q_0} \quad z_{q_0} \quad h_x \quad h_y \quad h_z \quad \rho]^T$$

L'equazione per la creazione di un nuovo landmark diventa:

$$\mathbf{l} = \begin{bmatrix} \mathbf{t_0} \\ \mathbf{q_0} \\ \overrightarrow{\mathbf{h}_c} \\ \rho \end{bmatrix} = \begin{bmatrix} \mathbf{T} \\ \mathbf{q} \\ \texttt{cam2world}(\mathbf{l_o}) \\ \rho_{init} \end{bmatrix}$$

con $\mathbf{T}$ e $\mathbf{q}$ posizione e orientamento della camera (in forma di quaternione) e $\mathbf{l_o}$ l'osservazione del landmark, ossia le coordinate x e y del pixel a esso associato.

La trasformazione di un landmark dalle coordinate FHP alle classiche coordinate cartesiane diventa:

$$\mathbf{l}_{xyz} = \mathbf{t_0} + \nu(\mathbf{q_0}) \frac{\overrightarrow{\mathbf{h}_c}}{\rho}$$

dove la funzione ν trasforma un quaternione nella corrispondente matrice di rotazione.

Infine l'equazione di misurazione diventa:

$$\mathbf{l_o} = \texttt{world2cam}(\nu(\mathbf{q})^T [\nu(\mathbf{q_0}) \cdot \overrightarrow{\mathbf{h}_c} + \rho(\mathbf{t_0} - \mathbf{T})])$$

7.7 Considerazioni finali

All'interno del presente capitolo sono stati presentati i contributi teorici del lavoro di tesi. Sono stati introdotti due nuovi metodi di patch tracking per camere omnidirezionali: quello sullo sviluppo panoramico e quello sulla conversione prospettica delle patch. È stato inoltre studiato e adattato al caso omnidirezionale il metodo del *patch warping*.

Per la costruzione di immagini panoramiche a partire da immagini omnidirezionali, sono state considerate due opzioni: lo "srotolamento" della superficie di un cilindro sul quale viene proiet-

tata l'immagine sferica, astrazione dell'immagine omnidirezionale, oppure lo "srotolamento" della superficie della sfera stessa. Una conversione del primo tipo rispetto a una del secondo, riduce maggiormente la distorsione dell'immagine, ma il fattore di scala non risulta omogeneo.

Per l'algoritmo di tracking con patch prospettiche, sono state presentate due versioni: la versione base applicabile a sequenze omnidirezionali indipendentemente dagli algoritmi a più alto livello che fanno uso del tracking e una versione per SLAM, che sfrutta le conoscenze sulla posizione dei landmark e sulle posizioni della camera nel tempo per risolvere i problemi di scala che si incontrano inseguendo le patch di landmark osservati progressivamente a distanze differenti.

È stato inoltre studiato il metodo da noi chiamato *Patch Update*, che aggiorna l'aspetto delle patch qualora siano correttamente associate, la loro correlazione superi una soglia stabilita e superino il test di concordanza di *1PointRansac*. Questo metodo può essere applicato al tracking con patch prospettiche di base e alle due versioni del tracking panoramico. L'utilizzo di questo metodo permette inoltre di utilizzare il patch tracking classico per camere prospettiche anche con camere omnidirezionali, ottenendo un notevole risparmio in termini di costo computazionale rispetto agli algoritmi presentati.

Per quanto riguarda gli algoritmi di SLAM, sono state revisionate le parametrizzazioni del capitolo precedente per il loro utilizzo con camere omnidirezionali ed è stato introdotto un nuovo metodo di inizializzazione della distanza inversa delle feature, in modo che esse vangano inizialmente disposte sul piano del terreno sul quale il robot si muove.

Capitolo 8

Implementazione

Le tecniche presentate nel capitolo precedente sono state implementate all'interno del framework *MoonSlam*[1]. Questo framework è composto da una serie di librerie che permettono all'utente di sviluppare e personalizzare la propria sessione di SLAM. MoonSlam viene descritto nella Sezione 8.1, mentre Le componenti del framework sviluppate nell'ambito del presente lavoro di tesi vengono dettagliate nella Sezione 8.2, suddivise per libreria.

8.1 MoonSlam

MoonSlam è un framework C++ per lo sviluppo di applicazioni /*Visual* SLAM realizzato da Simone Ceriani all'interno del Laboratorio di Intelligenza Artificiale e Robotica (AIRLab) del Politecnico di Milano. La sua organizzazione modulare permette all'utente di personalizzare la propria sessione di SLAM, decidendo quali tecniche applicare, con quali parametri e con quale tipologia di camera. La scelta del C++ come linguaggio implementativo, a differenza di altri linguaggi a più alto livello come *Matlab*, permette di ottenere prestazioni elevate e un miglior utilizzo della memoria.

Essendo un progetto molto ampio nel quale vengono utilizzate tecniche provenienti da svariate aree scientifiche (visual computing, statistica, algebra lineare ecc.), il suo sviluppo si appoggia su librerie di terze parti, per lo svolgimento di particolari compiti:

- **OpenCV**[2]: libreria di *computer vision* molto nota, sempre in evoluzione, che permette di gestire ed elaborare in maniera semplice ed efficiente le immagini, e implementa molti degli algoritmi più noti come SIFT, *Harris Corner Detector* ecc.
- **Eigen**[3]: libreria di algebra lineare che permette di effettuare operazioni con martici in maniera particolarmente efficiente.
- **MRPT**[4]: *Mobile Robot Programming Toolkit*, un insieme di librerie contenente algoritmi e applicazioni di *Computer Vision*, SLAM, *motion planning* ecc., dalla quale viene presa in prestito la parte di visualizzazione 3D.
- **LibConfig**[5]: per la gestione dei file di configurazione.
- **FFMPEG**[6]: per la lettura e scrittura di file video. In alternativa alla versione interna di OpenCV, che a seconda del Sistema Operativo e della versione della libreria si è riscontrato avere alcuni problemi.

[1] http://airlab.elet.polimi.it/index.php/MoonSlam
[2] http://opencv.willowgarage.com
[3] http://eigen.tuxfamily.org
[4] http://www.mrpt.org
[5] http://www.hyperrealm.com/libconfig
[6] http://ffmpeg.org

- **wxWidgets**[7]: libreria per la realizzazione di interfacce grafiche, utilizzata per la realizzazione della nostra interfaccia 2D.
- **wxMathPlot**[8]: estensione della libreria *wxWidgets* per creazione di grafici bidimensionali.

Specifichiamo inoltre che attualmente MoonSlam non lavora in tempo reale, ma su dataset preacquisiti e specificatamente quelli di Rawseeds[9]. Ciò significa che alcune componenti per l'acquisizione dei dati sono compatibili solamente con il formato di dati Rawseeds. Nel caso si volesse utilizzare MoonSlam con altre fonti è possibile svilupparsi le proprie componenti.

Il progetto MoonSlam è suddiviso in una serie di librerie, ognuna delle quali si occupa di gestire un particolare ambito dello SLAM:

- **AIRGeneric**: implementa una serie di funzioni di utilità generale, per la manipolazione di stringhe, per la gestione del tempo, ecc.
- **AIRMath**: sono qui sviluppate alcune funzioni di algebra lineare, come la composizione di rototraslazioni, la conversione di quaternioni in angoli di Eulero, ecc.
- **AIRImage**: si occupa della parte visuale dello SLAM, implementando il meccanismo di caricamento delle immagini sorgente, i modelli di camera e le tecniche di tracking. Attualmente MoonSlam non opera in temporeale, ma carica un dataset di immagini sorgente, ossia una sequenza di acquisizioni di una camera. Tale dataset può essere un filmato in uno dei qualsiasi formati comunemente utilizzati (mpeg, mp4, flv, ecc.) oppure una vera e propria sequenza di immagini (jpeg, png, ecc.), una per ogni frame. Le tecniche di tracking attualmente sviluppate rientrano nella categoria a patch. Vi è il patch tracking classico per camera prospettica, quello panoramico e a sviluppo prospettico per camere omnidirezionali, e il patch warping per entrambe le tipologie di camera. I modelli di camera implementati sono il classico modello pinhole per camere prospettiche, il modello proposto da Scaramuzza e il modello a sfera di Mei per le camere omnidirezionali, ampiamente discussi nel Capitolo 2. Viene qui definita anche la classe base *Landmark*, sulla quale si sviluppano altre classi molto importanti come *LandmarksAdder*, *LandmarksDeleter* e *LandmarksMatcher* che permettono di aggiungere, eliminare ed effettuare il matching delle feature dell'immagine. Nella sua implementazione base di questa libreria, la classe *Landmark* ha soltanto il riferimento al proprio descrittore e alla sua posizione all'interno dell'immagine. Nella libreria *AIRVisualSlam* essa viene estesa associandole una o più variabili del filtro di Kamlan per poter essere utilizzata nell'ambito dello SLAM. Infine, per la visualizzazione è qui definita anche la classe *LandmarksDrawing* che modifica l'immagine sorgente attuale marcando la posizione dei landmark, la loro area di ricerca e il loro stato.
- **AIREkfSlam**: implementa la versione del filtro di Kalman esteso per SLAM presentata nella Sezione 5.2 del Capitolo 5. L'EKF sviluppato è molto flessibile e di semplice utilizzo, in quanto predispone una serie di metodi che permettono di creare, manipolare ed eliminare le variabili nel filtro mascherandone la reale posizione al suo interno. All'interno di questa libreria sono definite inoltre le classi base per l'implementazione dei modelli di moto, per la creazione di nuovi landmark da inserire nel filtro e per la loro misurazione. Infine, è qui implementato anche l'algoritmo di *1PointRansac* (vedi 5.5), che effettua una selezione delle misure dei landmark ottenute escludendo gli *outliers*, prima che esse vengano utilizzate per l'update del filtro.
- **AIRVisualSlam**: in questa libreria vengono estese le classi base del modello di moto e della gestione dei landmark della libreria *AIREkfSlam*. I modelli di moto attualmente disponibili sono a velocità costante o con odometria. Sono qui implementate tutte le parametrizzazioni per lo SLAM introdotte nel Capitolo 6, tanto per il modello di camera prospettico, quanto per quello omnidirezionale. Per ognuna di queste sono disponibili inoltre le varianti *FixedFrame* e *Shared*: le prime permettono di specificare posizione e orientamento della camera rispetto a un

[7] http://www.wxwidgets.org
[8] http://wxmathplot.sourceforge.net
[9] http://www.rawseeds.org

sistema di riferimento robot, le quali facilitano l'uso dell'odometria; le seconde gestiscono la parte di àncora o frame delle parametrizzazioni in maniera separata, creando per essa un unica variabile condivisa nel filtro per i landmark che vengono inizializzati nel medesimo istante di tempo.

- **AIRSlamSubmaps**: implementa l'algoritmo *Conditional Indipendent SLAM* descritto nella Sezione 5.4.2 del Capitolo 5 per la suddivisione in sottomappe.
- **AIRConfig**: permette di rendere l'applicazione SLAM completamente configurabile, specificando in un file di configurazione esterno tutti gli oggetti necessari al funzionamento dell'applicazione e i relativi parametri.
- **AIRLog**: permette di scrivere su log lo stato dei componenti dello SLAM, utilizzato tipicamente per effettuare *debug* o per analisi successive.
- **AIRGps**: permette di reperire le informazioni del GPS per la sessione di SLAM. Tali informazioni vengono lette da un file, la cui codifica necessita di una complessa elaborazione per poter convertire l'informazione in esso contenuta nelle coordinate geostazionarie classiche: latitudine, longitudine e altitudine. L'utilizzo del segnale GPS viene utilizzato solamente a scopo di verifica del buon andamento dello SLAM e non viene in alcun modo sfruttato per migliorarne il risultato.
- **AIRUserInterface**: Interfaccia grafica 3D che permette di visualizzare in temporeale i risultati della sessione di SLAM in corso.
- **AIRGui**: Interfaccia grafica 2D che permette di visualizzare grafici, statistiche e risultati della sessione di SLAM corrente.

8.2 Componenti di MoonSlam realizzate nel lavoro di tesi

Nel presente lavoro di tesi sono state sviluppate le componenti necessarie per effettuare SLAM con camera omnidirezionale, insieme ad altre componenti di utilità generale. Presentiamo le varie parti sviluppate suddivise per libreria.

8.2.1 AIRImage

All'interno della libreria *AIRImage* sono stati implementati i due modelli di camera omnidirezionale presentati nel Capitolo 2. Tutti i modelli di camera sono stati organizzati nella gerarchia di Figura 8.1.

Le due funzioni `cam2world` e `world2cam` della classe base *CameraInterface* sono metodi puramente virtuali, sviluppati concretamente dai vari modelli di camera che estendono la classe. Ricordiamo che `cam2world` trasforma un punto 2D dell'immagine espresso in coordinate pixel, nella direzione del raggio di vista che l'ha generato, espressa in coordinate metriche 3D nel sistema di riferimento camera. Viceversa `world2cam` proietta un punto 3D nel sistema di riferimento camera, nel corrispondente pixel dell'immagine.

Le classi del livello intermedio della gerarchia *CameraParametersGeneric* e *OmnidirectionalCamera* distinguono le due tipologie di camera, prospettica e omnidirezionale rispettivamente. La classe *CameraParametersTopLeft* implementa il modello *pinhole* di camera prospettica. Le classi *OmniCameraScaramuzza* e *OmniCameraMei* realizzano invece i modelli di camera omnidirezionale.

La classe *OmnidirectionalCamera* è stata dotata di una ampia serie di metodi che permettono di elaborare un'immagine omnidirezionale, indipendentemente dal modello di camera concretamente adottato. I metodi più significativi sono i seguenti:

Figura 8.1: *Gerarchia dei modelli di camera implementati in MoonSlam*

- `getPanoramicSize`: utilizzata per la creazione di immagini panoramiche, riceve in ingresso la larghezza desiderata per il panorama e in base ai parametri della camera, in particolare al campo visivo verticale, restituisce la corretta dimensione per tale immagine.
- `createPanoramicImage`: trasforma l'immagine omnidirezionale in ingresso nel corrispondente panorama, come spiegato nella Sezione 7.1 del precedente capitolo. Alla prima invocazione del metodo viene creata la *Lookup Table*, che verrà riutilizzata nelle chiamate successive. Con la *Lookup Table* il panorama viene creato interpolando l'immagine omnidirezionale con uno dei metodi resi disponibili da OpenCV (interpolazione bilineare, bicubica, nearest neighbor).
- `createPerspectiveImage`: trasforma una porzione dell'immagine omnidirezionale nel corrispondente sviluppo prospettico, come spiegato nella Sezione 7.2 del capitolo precedente. Il punto centrale dell'area da trasformare può essere espresso in tre modi diversi: tramite le coordinate 2D del punto sull'immagine omnidirezionale, tramite *azimuth* ed *elevation* del corrispondente punto 3D sulla rappresentazione sferica oppure con le coordinate 3D del vettore direzione del raggio corrispondente al punto. La dimensione dell'area da trasformare è definita dal parametro fov, ossia l'ampiezza espressa in gradi del campo visivo orizzontale. A differenza del caso panoramico, la *Lookup Table* viene calcolata a ogni chiamata e successivamente interpolata con uno dei metodi disponibili. Il metodo restituisce in uscita un vettore di due matrici, che è la *Lookup Table* calcolata.
- `convertOmniPoint2ThetaPhi`, `convertThetaPhi2OmniPoint`, `convertPoint3d2ThetaPhi`, `convertThetaPhi2Point3d`: permettono la conversione bidirezionale fra punto 2D dell'immagine omnidirezionale, direzione del raggio corrispondente in 3D, *azimuth* ed *elevation* del punto sulla rappresentazione sferica.
- `pers2omni`, `pan2omni`, `panoramic2omniUsingLUT`: date le coordinate di un pixel di un'immagine panoramica o prospettica, risale alle coordinate del corrispondente pixel dell'immagine omnidirezionale. La differenza fra il secondo e il terzo metodo è che uno effettua

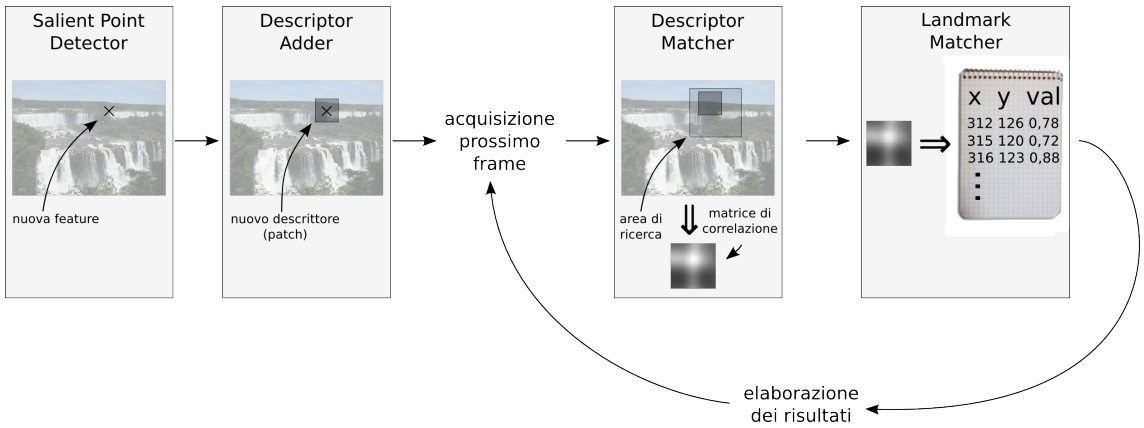


Figura 8.2: *Ruolo degli oggetti* Salient Point Detector, Descriptor Adder, Descriptor Matcher *e* Landmark Matcher *per il* patch tracking *classico su immagini prospettiche.*

- Salient Point Detector*: rileva nuovi corner all'interno dell'immagine;*
- Descriptor Adder*: crea e associa un descrittore (patch) alla feature;*
- Descriptor Matcher*: nei frame successivi all'inizializzazione della feature, esamina l'area di ricerca in cui si suppone essere la nuova posizione della feature, producendo una matrice di correlazione che indica per ogni pixel dell'area di ricerca il valore di correlazione fra patch e immagine attuale in tale posizione. La matrice di correlazione può essere rappresentata come immagine, dove pixel chiari indicano una buona correlazione mentre pixel scuri una scarsa;*
- Landmark Matcher*: esamina la matrice di correlazione prodotta dal* Descriptor Matcher *e crea una lista con le coordinate e il valore di correlazione di tutti gli elementi della matrice che superano una determinata soglia. Altri oggetti in seguito esamineranno questa lista ed eleggeranno uno di questi punti come la nuova posizione della feature (attualmente in* MoonSlam *viene semplicemente scelto il punto con il miglior valore di correlazione).*

l'intero calcolo, mentre l'altro recupera le informazioni dalla *Lookup Table*.

Nella sezione *Omni* della libreria *AIRImage*, sono state implementate le classi per effettuare il tracking panoramico, quello con patch prospettiche e quello con il *patch warping* omnidirezionale. Ognuno di questi metodi si compone di un *Salient Point Detector*, di un *Descriptor Adder*, di un *Descriptor Matcher* e di un *Landmark Matcher*. Il ruolo generale di questi oggetti è descritto in Figura 8.2, a titolo di esempio per il classico *patch tracking* su immagini prospettiche.

Il *Salient Point Detector* è il medesimo per tutti i metodi, sebbene il tracking panoramico lo applichi alla trasformazione panoramica dell'immagine omnidirezionale e non direttamente su essa come gli altri.

Nel tracking panoramico, il *Descriptor Adder* è banalmente un oggetto che assegna come descrittore a un nuovo landmark una porzione rettangolare dell'immagine panoramica nell'intorno della feature. Nel tracking con *patch warping* omnidirezionale, il *Descriptor Adder* associa al landmark un descrittore di tipo *DescriptorPatchForWarping*, che al suo interno conserva una porzione di immagine più ampia della dimensione delle patch e le informazioni su posizione e orientamento della camera, necessarie per la fase di matching. Nel tracking con patch prospettiche, la tipologia di descrittore è la stessa del caso panoramico, ma la patch che viene memorizzata è la trasformazione prospettica della regione dell'immagine omnidirezionale nell'intorno della feature.

Il *Descriptor Matcher* del tracking panoramico, ricerca la patch associata a un landmark all'interno di un'area di ricerca sull'immagine panoramica. Se la posizione del landmark rimane molto vicino al margine sinistro o destro del panorama, tale che l'area di ricerca non sia contenuta per intero al suo interno, essa viene assemblata prendendo la porzione mancante dal lato opposto, ricordando che l'immagine panoramica è orizzontalmente contigua. Nel caso del tracking con *patch warping*, il *Descriptor Matcher* si occupa di trasformare la porzione di immagine iniziale archiviata all'interno del descrittore, in una patch il cui aspetto dovrebbe assomigliare a quello assunto

attualmente dalla regione dell'immagine omnidirezionale nell'intorno della posizione del landmark (vedi Sezione 7.3 del capitolo precedente). Infine, nel tracking con patch prospettiche, il *Descriptor Matcher* trasforma l'area di ricerca dell'immagine omnidirezionale in un'immagine prospettica, sulla quale effettua il matching della patch prospettica contenuta nel descrittore.

Il compito del *Landmark Matcher* è quello di esaminare il risultato prodotto dal *Descriptor Matcher* e stilare una lista di nuove posizioni candidate, affiancate da una cifra di merito, demandando a un momento successivo la scelta effettiva della nuova osservazione. Per ognuno dei tre metodi vi è un *Landmark Matcher* differente che gestisce le peculiarità di ognuno di essi: nel coso panoramico la contiguità dell'immagine, nel caso di patch prospettiche la conversione della posizione del *match* all'interno dell'area di ricerca prospettica nella corrispondente posizione nell'immagine omnidirezionale.

Sempre all'interno della sezione *Omni* della libreria *AIRImage* abbiamo realizzato un *Landmarks Adder* che si occupa di lanciare la procedura di creazione di un nuovo landmark qualora una zona dell'immagine ne rimanga sprovvista. Tale oggetto era già presente nella libreria, per l'utilizzo con camera prospettica, che dividendo l'immagine in ingresso in una griglia come quella in Figura 8.3a e mantenendo un istogramma delle sue celle era in grado di rilevare la necessità di nuovi landmark. La forma di questa griglia non ha però molto senso in un'immagine omnidirezionale ed è per questo motivo che si è realizzato un *Landmarks Adder* apposito. La forma della griglia adottata nel caso omnidirezionale è mostrata in Figura 8.3b. In realtà, pur mantenendo essenzialmente questa forma, la suddivisione radiale della griglia può essere scelta secondo tre diversi principi (Figura 8.4):

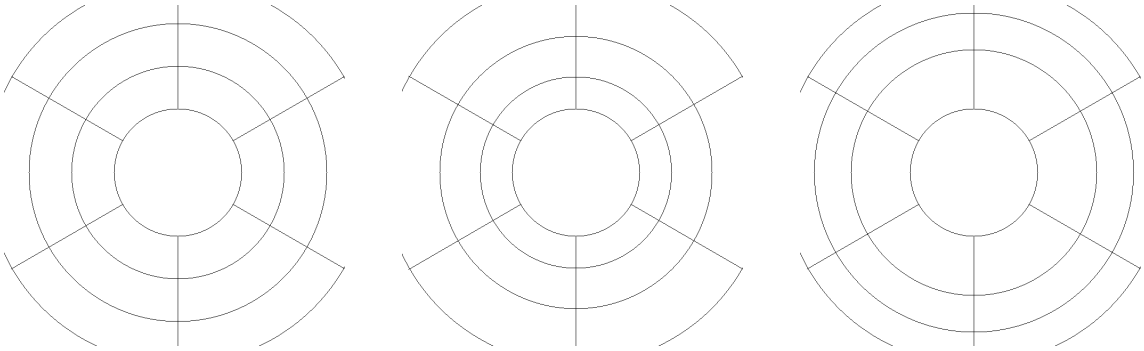
- *uguale raggio*: il raggio di ogni circonferenza che suddivide radialmente l'immagine omnidirezionale cresce linearmente, ossia ogni raggio differisce da quello precedente per una quantità costante.
- *uguale azimuth*: il campo visivo verticale della camera viene suddiviso in angoli uguali e ognuno di essi viene trasformato nel corrispondente raggio di circonferenza. L'effetto visivo di questa scelta è che i raggi delle circonferenze crescono sovralinearmente dal centro verso l'esterno. Questo metodo permette di dividere equamente la porzione di mondo visibile, sebbene rappresentata nell'immagine omnidirezionale da regioni di dimensione diversa.
- *azimuth inverso*: ha lo scopo di far crescere sublinearmente i raggi delle circonferenze, secondo la formula $r_i = r_{min} + (r_{max} - r_{min}) \cdot (i/n)^{\alpha}$, dove r_{min} e r_{max} sono i raggi delle circonferenze che definiscono il campo di visione verticale, n è il numero di suddivisioni radiali, i è l'indice dell'i-esima circonferenza ($0 \leq i \leq n$) e α un parametro fra 0 e 1 che amplifica o diminuisce l'effetto sublineare della funzione. Questa tipologia di griglia non ha nessun significato fisico ma dipendendo dal tipo di dataset in uso potrebbe comportarsi meglio delle altre.

All'interno della stessa sezione di *AIRImage* abbiamo definito inoltre alcuni nuovi *Landmarks Deleter*, che quando invocati eliminano i landmark che non rispettano alcune caratteristiche:

- *LandmarksDeleterOmniOutOfMirror*: elimina i landmark la cui posizione non rientra all'interno dell'area circolare dell'immagine omnidirezionale definita dal limite superiore del campo visivo verticale della camera.
- *LandmarksDeleterOmniNotMatchedInFieldOfView*: elimina i landmark la cui differenza fra il l'istante discreto di tempo dell'ultima volta che avrebbe potuto essere osservato (la predizione dell'osservazione fatta con lo SLAM cade all'interno dell'immagine) e l'istante di tempo dell'ultima volta che è stato realmente osservato supera una determinata soglia. In sostanza elimina quei landmark che secondo i calcoli dovrebbero essere attualmente visibili (potenzialmente osservabili), ma non lo sono, probabilmente a causa di un'occlusione o un cambio radicale dell'aspetto della patch.
- *LandmarksDeleterBroken*: elimina i landmark marcati come "broken", ossia coloro i quali hanno la componente ρ (distanza inversa) negativa.
- *LandmarksDeleterEqualDisposition*: considerando la suddivisione in griglia del *Landmarks Ad-*

(a) *Griglia per immagini prospettiche* **(b)** *Griglia per immagini omnidirezionali*

Figura 8.3: *Tipologie di griglie adottate dal* LandmarksAdder *per camera prospettica e omnidirezionale*



(a) *uguale raggio* **(b)** *uguale azimuth* **(c)** *azimuth inverso con* $\alpha = 0.7$

Figura 8.4: *Tipologie di suddivisione radiale della griglia per la ricerca di nuovi landmark in immagini omnidirezionali*

der, viene definito un limite superiore al numero di landmark presenti all'interno di ogni singola cella. Se il numero di landmark è superiore a tale soglia, viene fatta una selezione dei landmark da eliminare, conservando quelli recentemente osservati.

Si è creato infine un *Landmarks Deleter* particolare, chiamato *LandmarksDeleterCascade* che permette di comporre i *Landmarks Deleter* precedenti in un unico oggetto, tramite il metodo `appendDeleter`. Alla chiamata del metodo `deleteLandmarks` di questo oggetto, vengono invocati in cascata i deleter di cui è composto.

Un ultimo contributo a questa libreria è dato dall'oggetto *DescriptorPatchReplaceableUpdater1PR* che realizza il metodo del *Patch Update* della Sezione 7.4 del precedente capitolo. Il metodo `updatePatches` esamina semplicemente tutti i landmark, e per coloro che risultano validi, che sono stati "metchati" con un grado di correlazione superiore a una soglia stabilita e che non sono stati scartati dal *1PointRansac*, sostituisce la patch con la nuova porzione di immagine.

Citiamo soltanto la presenza all'interno di *AIRImage* anche delle classi *FFMPEGVideoReader* e *FFMPEGVideoWriter*, che si pongono come alternativa agli oggetti *VideoCapture* e *VideoWriter* di OpenCV, che in alcuni casi presenta dei bachi che non permettono di recuperare le informazioni su un filmato in maniera corretta e non permettono di effettuarne il *seek*.

8.2.2 AIRVisualSlam

In questa libreria sono state aggiunte tutte le parametrizzazioni omnidirezionali, già presenti in precedenza nella versione prospettica. Dal punto di vista teorico per ma maggioranza delle parametrizzazioni, la differenza fra quelle prospettiche e omnidirezionali risiede nelle operazioni di `cam2world` e `world2cam`. Si sarebbe potuto pensare quindi di implementare un modulo differente per queste due tipologie di camera e interfacciarlo con il modulo caratteristico di una particolare parametrizzazione, comunemente utilizzato nel caso prospettico e omnidirezionale. Data la frequenza con cui vengono invocate le operazioni di creazione di un nuovo landmark e ancora di più quelle per la sua misurazione, ed essendo di per sé piuttosto pesanti computazionalmente, la scelta che si è fatta in MoonSlam è di creare per ogni parametrizzazione i propri algoritmi di creazione e misurazione dei landmark, in maniera compatta ed efficiente. A questo scopo è stata utilizzata l'applicazione *Maxima*[10], un risolutore matematico molto potente che permette di scrivere le equazioni base della parametrizzazione, e automaticamente è in grado di calcolare gli Jacobiani necessari e infine ottimizzare l'intero algoritmo con l'utilizzo di variabili temporanee. L'algoritmo così ottimizzato viene poi tradotto direttamente in codice C++.

Nella sezione *Omni* della libreria *AIRVisualSlam* sono state implementate tutte le parametrizzazioni[11] del Capitolo 6 per camere omnidirezionali, nelle quattro versioni *Normal*, *FixedFrame*, *Shared* e *SharedFixedFrame*(insieme delle varianti *Shared* e *FixedFrame*).

8.2.3 AIRGui

La libreria *AIRGui* rappresenta una interfaccia grafica realizzata con *wxWidgets*, che permette di visualizzare in temporeale i risultati della sessione di SLAM in svolgimento. Dato che MoonSlam è altamente modulare e permette di personalizzare la sessione di SLAM, anche l'interfaccia grafica, per quanto possibile, dev'essere adattabile e personalizzabile secondo le esigenze dell'utente.

Cominciamo col descrivere le componenti di base dell'interfaccia:

- *Window*: è la classe base da cui ereditano tutte le tipologie di finestre visibili nell'interfaccia. Al suo interno sono stati definiti alcuni metodi che gestiscono i comportamenti comuni a qualsiasi finestra, come la possibilità di creare una barra di stato o nascondere e visualizzare nuovamente una finestra.
- *Plot*: *wrapper* della classe *mpWindow* della libreria *wxMathPlot* che realizza un generico grafico 2D. Il metodo `createSerie` permette di di creare dinamicamente nuove serie di dati da visualizzare al suo interno. Il metodo `getScreenshot` permette di catturare il grafico attualmente visibile e restituirlo come immagine OpenCV al chiamante.
- *Serie*: il metodo `createSerie` di *Plot* restituisce un oggetto di questo tipo, che rappresenta una serie di dati visualizzati nel grafico. Al suo interno vi è un vettore di coppie di coordinate $x-y$ che definiscono l'aspetto della funzione/serie da visualizzare. Alla sua creazione, la serie non contiene dati, i quali possono essere inseriti uno per volta con il metodo `append`. Il metodo `replaceVectors` permette invece di sostituire l'intero vettore di dati. Il metodo `setHistorySize` permette di definire un limite superiore alla lunghezza del vettore della serie: ogni dato che viene aggiunto dopo il raggiungimento di questa dimensione provoca l'eliminazione del dato più vecchio, ossia il primo elemento del vettore. Infine, con `setPen` è possibile definire il colore e lo spessore della linea continua che visualizza la serie.
- *PlotPanel*: *wrapper* della classe *Plot*, definito specificatamente per visualizzare funzioni di grandezze gaussiane. Un oggetto di questo tipo contiene al suo interno tre *Serie*, che rappre-

[10] http://maxima.sourceforge.net/

[11] In realtà la parametrizzazione AHP non è stata implementata, in quanto è stato testato che nel caso di camera prospettica non portava significativi vantaggi rispetto a UID e si presume quindi che ciò avvenga anche nel caso omnidirezionale.

sentano la media e la deviazione standard $(\mu, \mu - \sigma, \mu + \sigma)$. I dati vengono inseriti nel grafico con il metodo `append` che riceve in ingresso le coordinate x e y ed eventualmente come terzo parametro la deviazione standard.

- *PlotWindow*: realizza una finestra con al suo interno un grafico.
- *ImagePanel*: realizza un oggetto che permette di visualizzare un'immagine. L'immagine può essere inserita e sostituita in qualsiasi momento con il metodo `setImage`.
- *ImageWindow*: realizza una finestra con al suo interno un'immagine.
- *Map*: *wrapper* della classe *Plot* che permette di gestire il grafico come una mappa. Nel suo costruttore riceve un riferimento all'oggetto *LandmarksContainer*, il quale contiene la lista di tutti i landmark seguiti. Con questa informazione, alla chiamata del metodo `Update`, la mappa viene aggiornata rispecchiando l'attuale stato del *LandmarksContainer*.
- *Landmark*: rappresenta visivamente un singolo landmark nella mappa. Esso può essere attualmente rappresentato come una stella o come un quadrato. Con esso viene facoltativamente visualizzato anche l'ellisse di covarianza che esprime il grado di incertezza sulla sua posizione. L'aggiornamento del landmark avviene con il metodo `update`, che opzionalmente riceve come parametro una rototraslazione da applicare alla posizione contenuta nel landmark del *LandmarksContainer*, da utilizzare quando il sistema di riferimento dei landmark non è l'origine (sottomappe).
- *SlamStatistics*: produce una serie di statistiche sulla sessione di SLAM corrente. Nel suo costruttore riceve il riferimento agli oggetti *core* del processo di SLAM, come il *LandmarksContainer*, le variabili *pose* e *attitude* del robot contenute nel filtro di Kalman ecc. Sfruttando un meccanismo a eventi, che verrà descritto in seguito, a ogni ciclo del processo di SLAM, esamina lo stato degli oggetti *core* e ne calcola le statistiche. Per cronometrare la durata di ogni macro-operazione dello SLAM (ricerca e aggiunta di nuovi landmark, *update* del filtro, predizione del filtro, matching delle patch ecc.), viene sfruttato nuovamente il meccanismo a eventi, agganciando dei *listeners* prima e dopo ogni macro-operazione, i quali gestiscono una serie di *timer*. Questa classe non ha nessun componente visuale, ma soltanto un'ampia serie di metodi *get* per recuperare le statistiche.
- *SlamStatisticsWindow*: implementa una finestra nella quale vengono visualizzate tutte le statistiche recuperabili dall'oggetto *SlamStatistics*. Si è reso disponibile inoltre il metodo `createStatistic`, che permette di aggiungere dinamicamente una nuova statistica alla lista. Un'istanza di questo oggetto si può vedere nella Figura 8.5.

Tornando a una visione globale dell'applicazione, è importante specificare che la sua struttura differisce significativamente da una classica applicazione C++, ad esempio non esiste più una funzione *main* che viene eseguita al lancio dell'applicazione[12]. Abbiamo creato una classe *AIRGuiApplication*, che implementa le operazioni base per la creazione di una applicazione con interfaccia *wxWidgets* specifica per lo SLAM. Al suo interno viene inoltre creato un *thread* indipendente, che svolgerà il processo automatico e computazionalmente intensivo dello SLAM, demandando al *thread* padre semplicemente la parte visualizzativa. Per la realizzazione di una istanza dell'applicazione, è necessario estendere la classe *AIRGuiApplication* e implementare i due metodi virtuali seguenti:

- *OnInit*: definisce il preambolo dell'applicazione ed è il luogo adatto per la creazione di finestre o il caricamento di file di configurazione, e in generale per la creazione di oggetti che vengono usati nel *thread* figlio. Il metodo viene chiamato automaticamente all'interno del costruttore di *AIRGuiApplication*. Nella ridefinizione di questo metodo è necessario collocare come sua ultima istruzione la chiamata al metodo base della classe *AIRGuiApplication*, il quale fra le altre cose lancia l'esecuzione del *thread* figlio.
- *Entry*: il codice inserito all'interno di questo metodo viene eseguito dal *thread* figlio. Qui deve

[12] In realtà la funzione *main* viene mascherata all'interno della macro `IMPLEMENT_APP` di *wxWidgets*, sempre presente in una applicazione che utilizza un'interfaccia grafica realizzata con essa.

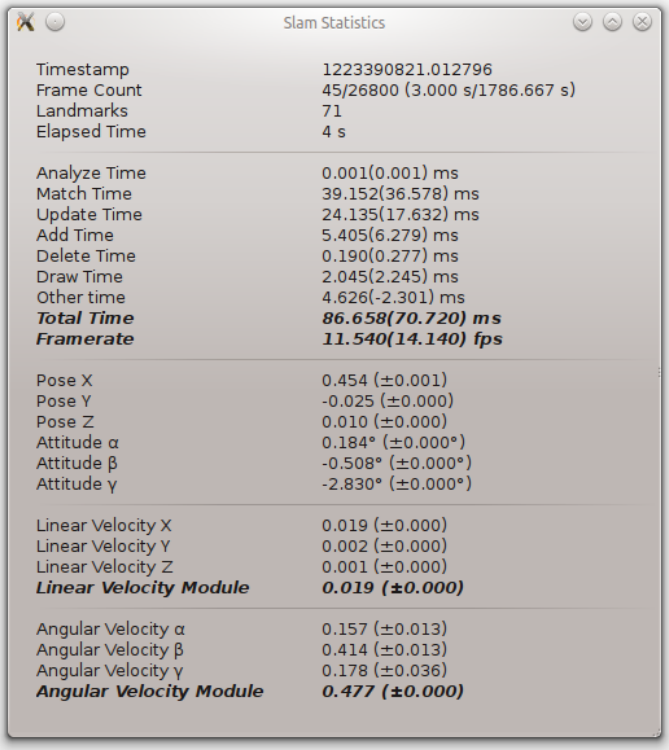


Figura 8.5: *Snapshot della finestra delle statistiche di AIRGui*

essere definito il processo di SLAM, e fra le sue operazioni devono essere chiamati i metodi per l'aggiornamento dell'interfaccia grafica.

Le applicazioni *wxWidgets* impongono una gerarchia fra le finestre, ossia la presenza di una finestra principale (root). Si è scelto di realizzare una versione base per questa finestra, che in *GUIInterface* implementa una serie di componenti e comportamenti necessari a una applicazione SLAM, indipendentemente dall'aspetto della sua interfaccia: menu con alcune opzioni base come *Exit*, *Pause*, *StepByStep*, la possibilità di creare e aggiornare in maniera veloce la barra di stato ecc. *GUIInterface* deve essere estesa per creare modelli concreti della finestra principale. Attualmente se ne sono sviluppati due:

- *GUIImageWithMaps*: il suo aspetto è mostrato in Figura 8.6. L'immagine nella parte superiore a sinistra mostra l'ultimo frame elaborato dallo SLAM, sul quale sono marcati i landmark, con colori diversi a seconda del loro stato e con il relativo ellisse di incertezza. Alla destra dell'immagine c'è la mappa con vista dall'alto (coordinate $x-y$), mentre nella parte inferiore della schermata ci solo le mappe frontale ($x-z$) e laterale ($y-z$).
- *GUIImageWithPoseAttitudeGraphs*: in Figura 8.7 è mostrato il suo aspetto. Come nella finestra *GUIImageWithMaps*, in alto a sinistra troviamo l'immagine del frame elaborato. Alla sua destra vi sono tre grafici che mostrano l'andamento nel tempo delle tre componenti x,y e z della posizione del robot stimata dallo SLAM. Nella parte inferiore troviamo invece i grafici del suo orientamento, con i tre angoli di Eulero α, β e γ.

La finestra principale deve essere creata all'interno del metodo `OnInit`, assegnando il suo riferimento all'attributo `mainWin` di *AIRGuiApplication*.

Nell'implementazione di questa interfaccia grafica si è dovuto fronteggiare la difficoltà della sincronizzazione fra *therad* padre e figlio. L'aggiornamento delle componenti grafiche, di competenza del *thread* padre, deve essere lanciato dal *thread* figlio, ma ciò viola uno dei principi di *wxWidgets*,

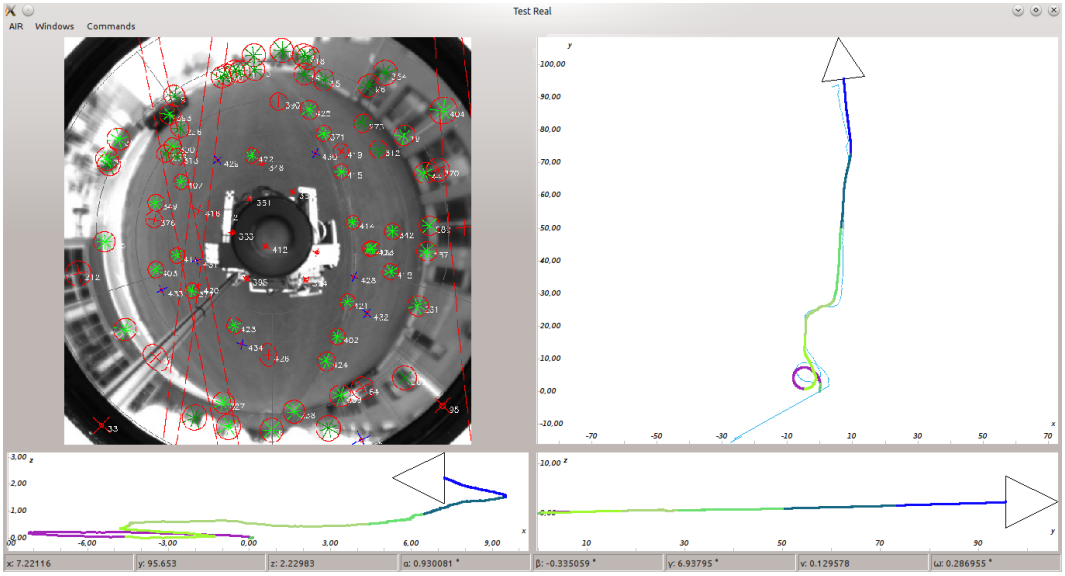


Figura 8.6: *Snapshot della finestra* GUIImageWithMaps *di AIRGui*

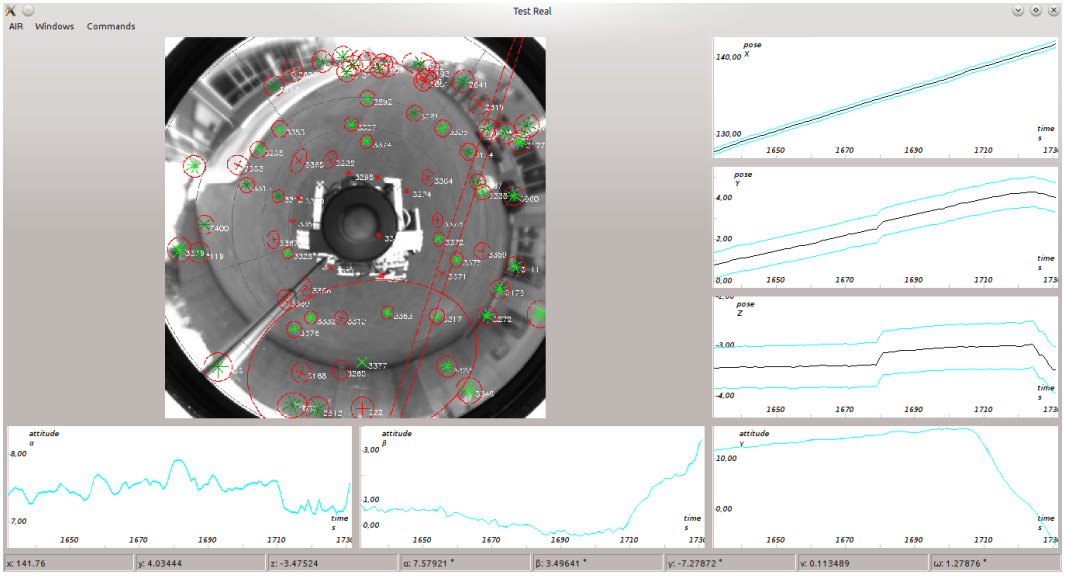


Figura 8.7: *Snapshot della finestra* GUIImageWithPoseAttitudeGraphs *di AIRGui*

che ne impedisce la chiamata diretta dei metodi. Per questo motivo, ogni operazione sull'interfaccia che si vuol rendere chiamabile dal *thread* figlio, necessita di essere elaborata nel seguente modo: creare un nuovo tipo di evento (*wxEventType*) da associare all'operazione, con i relativi parametri, connettere il metodo che effettua fisicamente l'operazione come routine di risposta all'evento e predisporre un ulteriore metodo chiamabile dal *thread* figlio, il quale non fa altro che creare una nuova istanza di evento, assegnargli i parametri dell'operazione e scatenare l'evento. Con questa laboriosa procedura è possibile spostare la responsabilità dell'esecuzione dell' operazione dal *thread* figlio al *thread* padre.

La creazione dinamica di nuove finestre deve anch'essa rispettare la regola precedente. Per questo motivo, per rendere disponibile la loro creazione dal codice del *thread* figlio, sono stati creati tre metodi appositi all'interno della classe *AIRGuiApplication*: `CreateImageWindow`, `CreatePlotWindow` e `CreateStatisticsWindow`, che creano e restituiscono al chiamante oggetti di tipo *ImageWindow*, *PlotWindow* e *SlamStatisticsWindow* rispettivamente. È quindi consigliabile l'uso di queste funzioni per la creazione di nuove finestre, evitandone l'istanziazione diretta.

8.2.4 AIRGeneric

Nella libreria *AIRGeneric* è stata creata la classe *EventHandler* che implementa un meccanismo a eventi utilizzando un *observer pattern*. In questa configurazione c'è una classe "soggetto" che definisce gli eventi in grado di scatenare e permette a oggetti terzi di sottoscriversi a essi. La classe "soggetto" deve estendere con l'ereditarietà la classe base *EventHandler* e definire i propri eventi con una `typedef enum` (es. `typedef enum {MyEvent1, MyEvent2, ...}MyEventType;`). Nel codice dei metodi della classe "soggetto", quando si realizza un'evento si deve utilizzare il metodo `fireEvent` per il suo scatenamento. Gli oggetti che intendono sottoscriversi agli eventi, anche chiamati *listeners*, lo possono fare chiamando il metodo `addEventListener` della classe "soggetto", che riceve come parametri il tipo di evento al quale ci si vuole sottoscrivere e il metodo che deve essere eseguito al suo scatenamento. Con il metodo `removeEventListener` è invece possibile disdire la notifica di un evento a cui ci si era registrati in precedenza.

Il meccanismo implementato essenzialmente conserva all'interno della classe "soggetto" un vettore di routine/metodi per ogni tipologia di evento. Inizialmente questi vettori sono vuoti e vengono successivamente riempiti o svuotati con le chiamate ai metodi `addEventListener` e `removeEventListener`. Lo scatenamento di un evento tramite il metodo `fireEvent` provoca l'esecuzione sequenziale di tutte le routine all'interno del vettore corrispondente a tale evento.

Il sistema a eventi realizzato è stato utilizzato in alcune parti di MoonSlam, ad esempio nella classe *SlamStatistics* di *AIRGui*, come descritto in precedenza.

8.2.5 AIRSim

Nel progetto MoonSlam, oltre alla realizzazione di sessioni di SLAM utilizzando *dataset* reali, è disponibile un ambiente simulato che permette di testare gli algoritmi di SLAM implementati. La struttura di questo ambiente si compone semplicemente di una serie di punti con una precisa posizione nel mondo tridimensionale. In alternativa agli oggetti della classe *AIRImage* per effettuare il tracking su immagini reali, all'interno di *AIRVisualSlam* ve ne è una versione che lavora in ambiente simulato. A differenza dei metodi di tracking reali, nell'ambiente simulato non vi sono mai errori di classificazione, ossia non può mai accadere di effettuare una osservazione completamente errata. D'altra parte osservazioni perfette rappresenterebbero una condizione troppo forte che non metterebbe sufficientemente alla prova gli algoritmi di SLAM. Per questo motivo, le osservazioni vengono alterate aggiungendo alle loro posizioni un rumore gaussiano a media nulla. In questo ambiente simulato è possibile testare la bontà di nuove parametrizzazioni, senza che essa venga influenzata dalla qualità del tracking sottostante.

Il contributo all'ambiente simulato di questa tesi voleva essere la creazione di diversi modelli 3D del mondo e di altrettanti modelli di moto che permettessero di esplorarli. La libreria *AIRSim* è attualmente ancora allo stato embrionale e al suo interno sono stati realizzati solamente due modelli di moto: *SquareMotion* e *DolphinMotion*. Il primo percorre essenzialmente una traiettoria quadrata, eventualmente arrotondando gli angoli. Il moto è altamente personalizzabile, nella dimensione del quadrato, nel grado di arrotondamento degli angoli e nella velocità di percorrenza. Il secondo modello di moto agisce invece sulla componente verticale della traiettoria (coordinata z) dandogli un andamento sinusoidale. Questo modello è stato pensato come estensione di un modello di moto che si muova sul piano $x - y$ come ad esempio *SquareMotion*.

8.2.6 rawseeds2video

Il progetto MoonSlam è attualmente legato al formato dei dataset di *Rawseeds* per quanto riguarda gli oggetti che si occupano di caricare i dati dall'esterno (sequenza video, GPS, ground truth, timestamps ecc.). Le sequenze video dei dataset di *Rawseeds* sono delle vere e proprie sequenze

di immagini in formato *png*, una per ogni frame. L'uso diretto di questo formato non è molto efficiente, in quanto al caricamento di ogni frame dev'essere effettuata un'apertura di file. Inoltre la dimensione di un dataset in questo formato supera spesso i 10 *GBytes* di memoria.

rawseeds2video è una piccola applicazione indipendente da MoonSlam, inserita al suo interno come *utility* che permette di trasformare un dataset Rawseeds in un file video. Dipendendo dal formato di codifica scelto, la dimensione del video potrebbe rimanere comunque molto alta. Si è scelto quindi di dare la possibilità di suddividere la sequenza in più filmati o convertirne solamente una sua parte. Per ogni file video prodotto, viene creato anche un file ".lst" associato, con la lista dei *timestamps* dei frame contenuti nel corrispondente file video.

L'applicazione, eseguibile da linea di comando, è corredata dalle seguenti opzioni:

- `-h --help`: visualizza una mini guida che specifica come utilizzare l'applicazione e il significato dei parametri.
- `-i --dataset_folder`: specifica il percorso della cartella dove sono contenute tutte le immagini dei frame della sequenza *Rawseeds*.
- `-o --output_folder`: specifica il percorso della cartella di destinazione, dove verranno creati i file video.
- `-c --chunk_length`: parametro opzionale, se specificato indica la durata di ogni sottosequenza espressa in numero di secondi. Se l'opzione non viene utilizzata verrà creato un unico file video.
- `-s --start_time`: permette di ignorare i frame iniziali e di considerare il dataset partendo dal numero di secondi specificato dal parametro.
- `-e --end_time`: termina la creazione del filmato al raggiungimento del tempo (in secondi) indicato dal parametro.
- `-S --start_frame`: come $-s$ ma il frame iniziale viene indicato con il nome del proprio file.
- `-E --end_frame`: come $-e$ ma il frame finale viene indicato con il nome del proprio file.
- `-b --basename`: nome di base per i file video da creare.
- `-C --codec`: specifica quale codifica utilizzare. Attualmente le opzioni possibili sono *FFV1*, *MPEG* e *FLV*.

8.3 Considerazioni finali

In questo capitolo è stato presentato il framework *MoonSlam* all'interno del quale sono state sviluppate le tecniche e le componenti necessarie per effettuare SLAM con camere omnidirezionali. Dopo una breve introduzione al framework sono state presentate nello specifico le componenti sviluppate dal presente lavoro di tesi, con alcuni dettagli implementativi. I risultati dei test effettuati saranno presentati nel prossimo capitolo.

Capitolo 9

Risultati

I metodi presentati nel Capitolo 7 e implementati all'interno del framework *MoonSlam* come spiegato nel capitolo precedente, sono stati testati per valutarne la qualità. I risultati di questi test vengono mostrati e discussi nel presente capitolo.

I *dataset* utilizzati per valutare le tecniche implementate fanno parte del progetto *Rawseeds*[1]. *Rawseeds* fornisce una serie di dataset multisensoriali, nonché applicazioni e soluzioni per alcuni problemi inerenti la robotica, fra cui lo SLAM. Attualmente i dataset disponibili sono stati acquisiti presso l'Università di Milano-Bicocca e la sede di Bovisa del Politecnico di Milano. Le riprese di Bicocca sono ti tipo *indoor*, mentre quelle di Bovisa sono *outdoor*. Per ognuno dei due ambienti sono disponibili vari dataset con percorsi e caratteristiche diverse, come la presenza o meno di oggetti e persone in movimento. Il robot con cui sono state effettuate le acquisizioni è dotato dei seguenti sensori:

- camera frontale;
- camera stereo;
- camera omnidirezionale catadiottrica;
- sistema a ultrasuoni con 12 trasduttori (per ambienti *indoor*);
- due rilevatori laser;
- IMU (*Inertial Measurement Unit*);
- odometria differenziale;
- ricevitore GPS (per ambienti *outdoor*);

Le misurazioni effettuate con il GPS in ambienti *outdoor* vengono utilizzate come *ground truth* per valutare la traiettoria del robot stimata dallo SLAM. Negli ambienti interni di Bicocca la *ground truth* è stata invece ottenuta con metodi di *scan matching* e attraverso una rete di telecamere.

I test effettuati nel presente lavoro di tesi utilizzano come *input* la sola camera omnidirezionale e, in alcuni casi, l'odometria. La camera omnidirezionale è di tipo catadiottrico, composta da una camera ortografica e uno specchio paraboidale, ha una risoluzione di 640×640 pixel e una frequenza di acquisizione di 15 frame al secondo. I dataset utilizzati sono i seguenti tre:

- *Bicocca_2009-02-27a*, ambiente statico indoor;
- *Bovisa_2008-10-07*, ambiente outdoor dinamico (auto e persone in movimento);
- *Bovisa_2008-10-11b*, ambiente outdoor statico;

Nel seguito del capitolo ci riferiremo ai tre *dataset* con i nomi *Bicocca*, *Bovisa1* e *Bovisa2* rispettivamente. In Figura 9.1 sono mostrate tre immagini omnidirezionali prese dai tre *dataset*.

[1] http://www.rawseeds.org

(a) *Bicocca_2009-02-27a (Bicocca)* **(b)** *Bovisa_2008-10-07 (Bovisa1)* **(c)** *Bovisa_2008-10-07 (Bovisa2)*

Figura 9.1: *Immagini omnidirezionali di esempio dei dataset Rawseeds utilizzati nei test.*

Negli esperimenti effettuati si è fatto uso della tecnica di *1-Point RANSAC* (Sezione 5.5 del Capitolo 5) per escludere osservazioni non corrette, e del *submapping* condizionalmente indipendente (Sezione 5.4.2 del Capitolo 5). Tutti i risultati ottenuti non fanno uso di tecniche di *loop closure*, le quali sono ancora in fase di sviluppo nel framework *MoonSlam*.

Per valutare le prestazioni degli algoritmi sviluppati, non è stato possibile testare l'intero spazio di ricerca dei parametri dello SLAM, a causa della sua ampiezza e della durata di ogni singolo test, generalmente superiore alla mezz'ora. Inoltre, i risultati dei test in ambiente *outdoor*, a causa della rumorosità del segnale GPS, non sono valutabili con una procedura automatica, ma solo visivamente, sovrapponendo alla mappa satellitare dell'ambiente la traiettoria del robot stimata, opportunamente scalata e ruotata. Si consideri infine che al risultato di un test contribuiscono anche diversi fattori di casualità, derivanti dalla scelta dei punti caratteristici da seguire, dal loro tracking impreciso, dall'inizializzazione dei landmark e dagli errori di linearizzazione dell'EKF. Questi fattori migliorano o peggiorano aleatoriamente il risultato del test, non rendendolo completamente attendibile. I risultati mostrati nel presente capitolo sono stati ottenuti selezionando i migliori fra centinaia di test effettuati, allo scopo di dimostrare sperimentalmente l'*upper bound* della qualità dello SLAM ottenibile con i metodi sviluppati. I test sono stati condotti ottimizzando i parametri dell'intero algoritmo di SLAM con la sensibilità sviluppata dall'autore sull'argomento.

Per ogni risultato mostrato vengono elencati i parametri dello SLAM con cui è stato ottenuto, il cui significato è il seguente:

- *Metodo di tracking* — algoritmo di tracking applicato; può assumere i seguenti valori: `Classico` (per immagini prospettiche), `Panoramico`, `Warping` (patch warping per camere omnidirezionali), `Patch prospettiche`;
- *Dimensione patch* — dimensione in pixel delle patch utilizzate dal tracking (es. 15×15 px);
- *Campo visivo patch* — ampiezza del campo visivo orizzontale per il metodo di tracking con patch prospettiche;
- *Larghezza panorama* — larghezza in pixel delle immagini panoramiche per il metodo di tracking panoramico;
- *Stile panorama* — tipologia di trasformazione panoramica, con proiezione su superficie cilindrica o superficie sferica, per il metodo di tracking panoramico;

- *Aggiornamento patch* — parametro booleano che indica l'applicazione dell'algoritmo di *Patch Update*;
- *Agg. patch threshold* — soglia di correlazione delle patch oltre la quale possono essere sostituite, per il metodo del *Patch Update*;
- *Correzione scala con SLAM* — parametro booleano per il metodo di tracking con patch prospettiche, che indica se viene applicata la correzione della scala delle patch con le informazioni dello SLAM;
- *Cov. rumore trasl.* — valori di covarianza del rumore gaussiano agente sulla componente traslativa del modello di moto;
- *Cov. rumore rot.* — valori di covarianza del rumore gaussiano agente sulla componente rotativa del modello di moto;
- *Parametrizzazione* — parametrizzazione dei landmark adottata (`UID`, `FHP` o `FID`), con eventuale indicazione della variante applicata (`Shared`, `FixedFrame` o `SharedFixedFrame`);
- *Iniz. distanza inversa* — metodo di inizializzazione della distanza inversa dei landmark, sul piano o costante;
- d_{floor} — distanza del centro ottico della camera dal piano del terreno sul quale il robot si muove, per l'inizializzazione dei landmark sul piano;
- α_{max} — angolazione della direzione dei landmark entro la quale vengono inizializzati sul piano, dopo la quale vengono inizializzati a distanza costante;
- $\varsigma_{d_{floor}}$ — deviazione standard minima della componente della distanza inversa dei landmark (ρ) attribuita in fase di inizializzazione sul piano;
- η — fattore di amplificazione della deviazione standard della distanza inversa del landmark all'aumentare dell'angolo α della sua direzione, in fase di inizializzazione sul piano;
- *Rumore sulla misura* — errore di misurazione della camera, espresso in pixel;
- *Tipologia di griglia* — forma della griglia utilizzata per la ricerca dei nuovi punti caratteristici di un'immagine; può assumere i seguenti valori: `classica` (griglia rettangolare per immagini prospettiche), `uguale raggio`, `uguale azimuth` e `azimuth inverso`. Vedi Figura 8.4 del Capitolo 8;
- *Numero di colonne/settori* — numero di colonne o settori della griglia;
- *Numero di righe/tracce* — numero di righe o traccie della griglia;
- *Min landmark per cella* — numero minimo di landmark osservabili in ogni cella della griglia. Qualora in un determinato momento una cella avesse un numero inferiore di landmark, ne vengono creati di nuovi;
- *Qualità Harris detector* — soglia minima di qualità dei punti caratteristici rilevati dall'*Harris corner detector* per la creazione di nuovi landmark;
- *Match threshold* — soglia minima di correlazione fra una patch e la regione dell'area di ricerca a maggior correlazione, affinché la patch venga considerata correttamente associata;

I test sono stati elaborati con un computer portatile *Hewlett-Packard G62A16SL*, dotato di processore *Intel i3-350M* (2.26 Ghz, 64 bit, dualcore, 3MB di cache L3) e 4 GB di memoria RAM DDR3. Durante i test effettuati sono state acquisite alcune statistiche, anch'esse elencate insieme ai risultati mostrati nel presente capitolo. Le statistiche elaborate e il loro significato è il seguente:

- *Numero medio di landmark nel filtro* — nella sessione di SLAM, utilizzando il *submapping* condizionalmente indipendente, vi sono più EKF, uno per ogni sottomappa. La statistica è stata ottenuta accumulando il numero di landmark presenti ad ogni istante di tempo nel filtro attivo, corrispondente alla mappa attuale, e dividendo successivamente per il numero di istanti di tempo elaborati;
- *Tempo medio tracking* — durata media dell'operazione di tracking delle feature di un frame;
- *Tempo medio aggiornamento filtro* — durata media dell'operazione di aggiornamento dell'EKF;
- *Tempo medio di aggiunta nuovi landmark* — durata media della procedura di ricerca di nuovi landmark e della loro inizializzazione;
- *Tempo medio di cancellazione landmark* — durata media dell'operazione di eliminazione dei landmark dal filtro. Le politiche di eliminazione adottate sono mirate a evitare l'eccessivo accumulo dei landmark in una determinata regione dell'immagine, nonché a eliminare i landmark che ottengono una stima negativa del parametro della distanza inversa;
- *Framerate medio* — numero medio di frame elaborati al secondo. La statistica si riferisce all'intero ciclo di SLAM, comprendendo anche fattori come il caricamento del nuovo frame (durata $\approx$ 5ms), il *drawing* del frame per evidenziale la posizione dei landmark e la loro area di ricerca, l'elaborazione del segnale GPS o *ground truth* in modo che risulti allineato con la traiettoria stimata, il passaggio da una sottomappa alla successiva, l'aggiornamento dell'interfaccia grafica con lo stato attuale delle stime.

Nella Sezione 9.1 viene effettuato un confronto fra i metodi di tracking per camere omnidirezionali sviluppati. Nella Sezione 9.2 vengono valutate le prestazioni delle parametrizzazioni dei landmark per camere omnidirezionali, eseguendo test in ambiente simulato. Nella Sezione 9.3 vengono confrontati i migliori risultati di SLAM ottenuti con i metodi di tracking sviluppati, con il *dataset* di Bovisa2 senza l'uso dell'odometria. Nella Sezione 9.4 vengono mostrati e commentati i migliori risultati ottenuti in ambienti *outdoor* (Bovisa1 e Bovisa2), con e senza l'uso dell'odometria. La stessa tipologia di test viene applicata al *dataset indoor* di Bicocca nella Sezione 9.5. Per terminare, nella Sezione 9.6 vengono tratte le conclusioni del capitolo.

9.1 Tracking

La qualità del tracking, seppur valutabile visivamente, non è facilmente dimostrabile attraverso delle statistiche o delle immagini, se non marcando manualmente per ogni feature la corretta corrispondenza in tutti i frame successivi della sequenza, ma questo richiederebbe un tempo molto lungo, oltre le nostre disponibilità. Le dichiarazioni che presenteremo nel seguito della sezione, in merito ai test effettuati con i diversi metodi di tracking sviluppati, si basano quindi sulla qualità dello SLAM ottenuto con essi, sull'interpretazione dei sui risultati e sulla semplice valutazione personale della sequenza visiva di tracking.

Fra i metodi di tracking illustrati nel Capitolo 7, solo il metodo panoramico e quello con patch prospettiche nella sua versione base (senza l'adattamento della scala con SLAM) sono testabili a prescindere dall'algoritmo di SLAM con il quale vengono utilizzati. Questo perché sia il metodo del *patch warping*, sia quello con patch prospettiche con la modifica della scala, utilizzano le informazioni provenienti dallo SLAM per effettuare le opportune trasformazioni delle patch o delle rispettive aree di ricerca. Anche il metodo dell'aggiornamento delle patch dipende dallo SLAM, in quanto utilizza le informazioni prodotte dall'algoritmo di *1-Point RANSAC*. In questi ultimi tre casi, la qualità del tracking è quindi influenzata dalla qualità dello SLAM.

La trasformazione panoramica di immagini omnidirezionali, sebbene riduca notevolmente la distorsione rispetto alle immagini omnidirezionali di partenza, non presenta ancora le caratteristiche di linearità ideali affinché si possano utilizzare efficacemente i classici metodi di tracking del caso prospettico. I test effettuati mostrano che la distorsione rimanente nelle immagini panoramiche, provoca la frequente manifestazione di un fenomeno di slittamento delle feature, in particolare di quelle originate da *corner* di scarsa qualità. In Figura 9.2 viene mostrato il fenomeno in questione. I test dimostrano inoltre che utilizzando la trasformazione panoramica su cilindro, le feature nella parte inferiore dell'immagine panoramica non vengono seguite in maniera soddisfacente a causa del forte cambio di scala che subisce questa parte di immagine. L'utilizzo della trasformazione che "srotola" direttamente la superficie sferica dell'immagine omnidirezionale ha permesso di ottenere risultati migliori.

Il tracking con patch prospettiche riduce notevolmente il problema dello slittamento, effettuando un tracking di discreta qualità. I due parametri *patch size* e *fov* (field of view) da attribuire al metodo determinano rispettivamente la risoluzione delle patch e l'ampiezza dell'area corrispondente sull'immagine omnidirezionale. Il loro valore ottimale è chiaramente legato alla risoluzione delle immagini della camera omnidirezionale e in parte alla tipologia di ambiente che esse catturano. Per la tipologia di camera da noi adottata (risoluzione 640×640), un buon compromesso fra prestazioni computazionali e qualità del tracking si è riscontrato con l'utilizzo di patch 31×31 e un *fov* di 5^o. L'utilizzo delle informazioni provenienti dallo SLAM per rendere l'algoritmo di tracking robusto ai cambi di scala derivanti dall'osservazione dei landmark a distanze differenti, permette di innalzare leggermente la vita media dei landmark.

I test effettuati con il metodo del *patch warping*, illustrato nella Sezione 7.3 del Capitolo 7, dimostrano la sua efficacia anche nel caso omnidirezionale nel caso in cui le le informazioni prodotte dallo SLAM siano sufficientemente precise.

Per quanto riguarda il metodo dell'aggiornamento delle patch[2], i test dimostrano che il suo utilizzo in combinazione con il metodo panoramico amplifica notevolmente l'effetto di slittamento delle feature. Anche nel caso del metodo delle patch prospettiche, l'aggiornamento delle patch non produce risultati soddisfacenti. Risulta invece di fondamentale importanza nel caso in cui si voglia utilizzare il patch tracking classico per camere prospettiche, in quanto, seppur in maniera non precisa, risolve il problema della distorsione delle immagini e della rotazione della camera omnidirezionale intorno al proprio asse. La qualità del tracking ottenuta risulta comunque inferiore

[2]Ricordiamo che l'aggiornamento delle patch non può essere applicato al metodo del patch warping e a quello con patch prospettiche con l'adattamento della scala

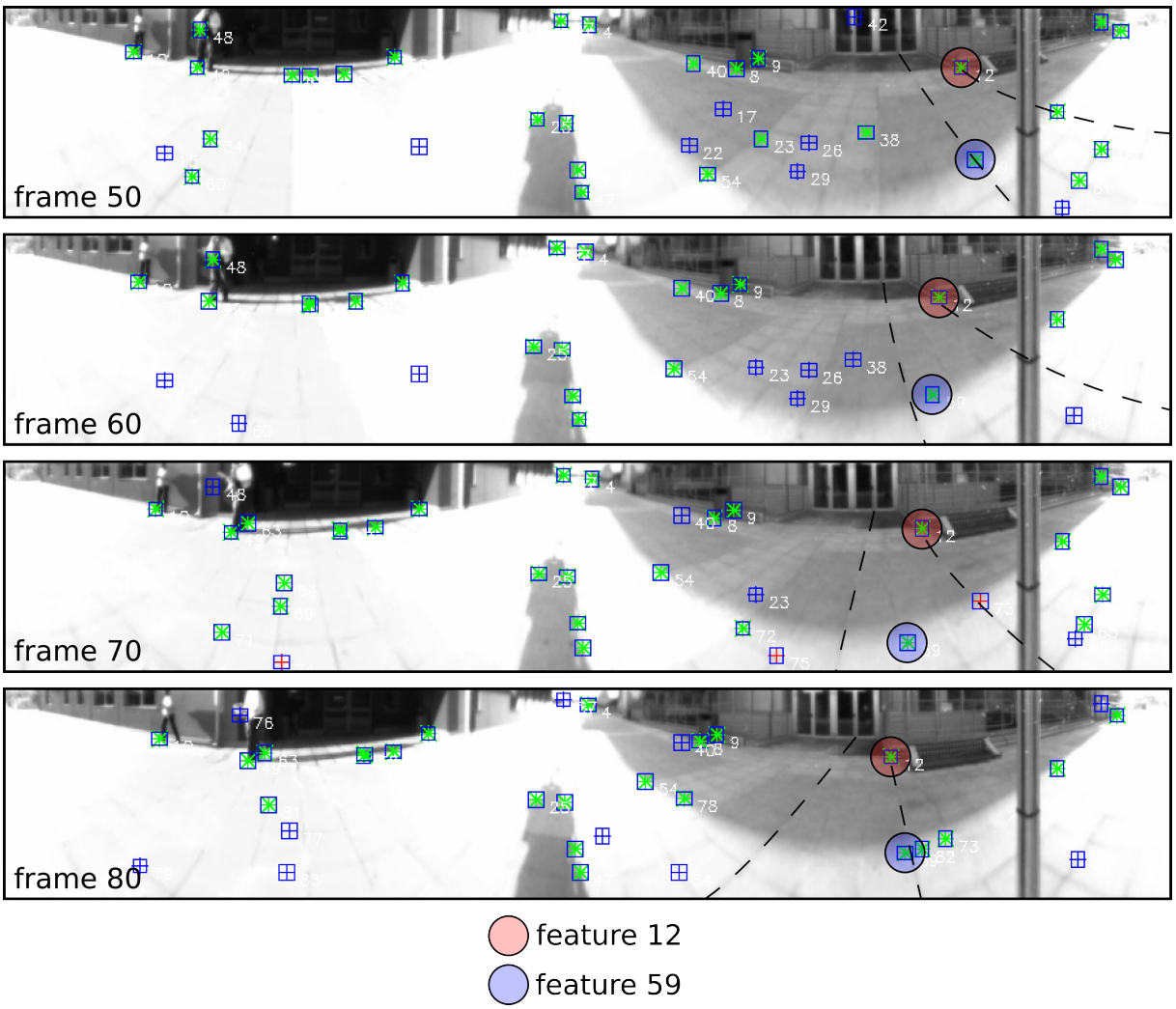


Figura 9.2: *Slittamento delle feature con il tracking panoramico: le linee tratteggiate evidenziano due linee statiche parallele della superficie piastrellata sulla quale il robot si muove; seguiamo lo spostamento reciproco delle feature 12 e 59 per 40 frame, dei quali ne mostriamo solamente quattro selezionati ogni dieci frame. Il corner della feature 12 è particolarmente buono e viene seguito in maniera corretta per tutti i 40 frame. Come mostrano chiaramente le immagini, la feature 59 non viene invece seguita correttamente e subisce uno slittamento verso destra, causato della distorsione dell'immagine panoramica.*

rispetto a quella ottenibile con il metodo panoramico o a patch prospettiche; Il reale vantaggio del patch tracking classico con l'aggiornamento delle patch è dato dal suo costo computazionale che è nettamente inferiore a quello degli altri metodi.

In conclusione, fra i metodi sviluppati quello che sembra ottenere i risultati migliori è il tracking con patch prospettiche, sia nella sua versione base che in quella con l'adattamento della scala.

9.2 Test in ambiente simulato

Per poter testare il corretto funzionamento delle parametrizzazioni realizzate, sono stati eseguiti dei test in ambiente simulato. Il tracking delle feature realizzato in questo ambiente esclude la presenza di errori macroscopici o di classificazione come potrebbe accadere in un tracking reale, permettendo una valutazione delle parametrizzazioni indipendente dalla qualità dell'algoritmo di tracking sottostante. Sebbene in questa condizione la posizione delle feature nelle immagini sia conosciuta con estrema precisione, a essa viene sempre aggiunto un rumore gaussiano a media nulla, per simulare gli errori di misurazione che commetterebbe una qualsiasi camera in ambiente reale.

La mappa dell'ambiente simulato è composta da una serie di punti posizionati a forma di chiostro. Nella simulazione il robot percorre due giri del chiostro seguendo una traiettoria quadrata smussata agli angoli, alzandosi e abbassandosi seguendo un andamento sinusoidale (vedi Sezione 8.2.5 del capitolo precedente). La struttura del chiostro e la traiettoria del robot sono visibili in Figura 9.4, dove sono stimate con estrema precisione dal test con la parametrizzazione FID.

Sono state testate le parametrizzazioni IS, UID, FHP, FID per camera omnidirezionale con differenti valori dei parametri e in tutte le loro varianti (parametrizzazioni originali, *Shared*, *FixedFrame* e *SharedFixedFrame*). I risultati ottenuti per la parametrizzazione *Inverse Scaling*, mostrati in Figura 9.3, sono di scarsa qualità, e confermano la sua inadeguatezza per il *monocular SLAM*. In Figura 9.4 è mostrato invece il risultato del test sulla parametrizzazione FID. I risultati dei test sulle restanti parametrizzazioni, in tutte le loro varianti, sono pressoché identici a quello mostrato per FID e verranno omessi. Come si evince chiaramente dalla figura, esclusa IS, le altre parametrizzazioni risultano adeguate per il *Monocular* SLAM: in assenza di errori grossolani di tracking e con una buona distribuzione delle feature sulle immagini, tutti quanti producono una traiettoria e una mappa pressoché perfette.

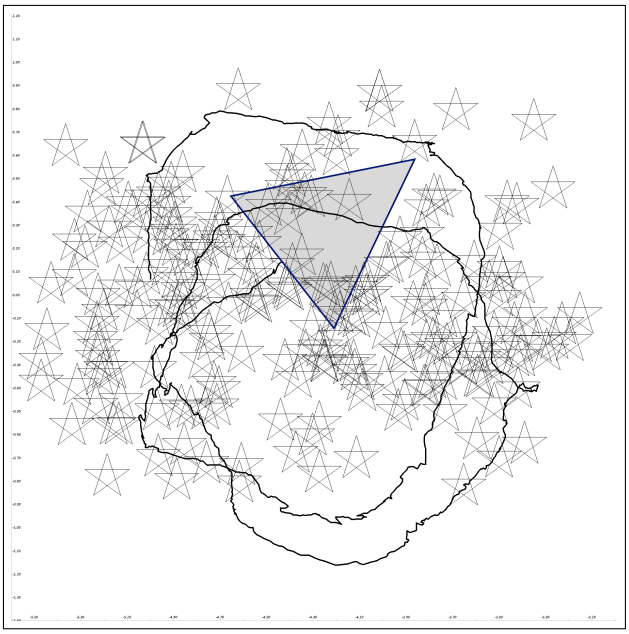
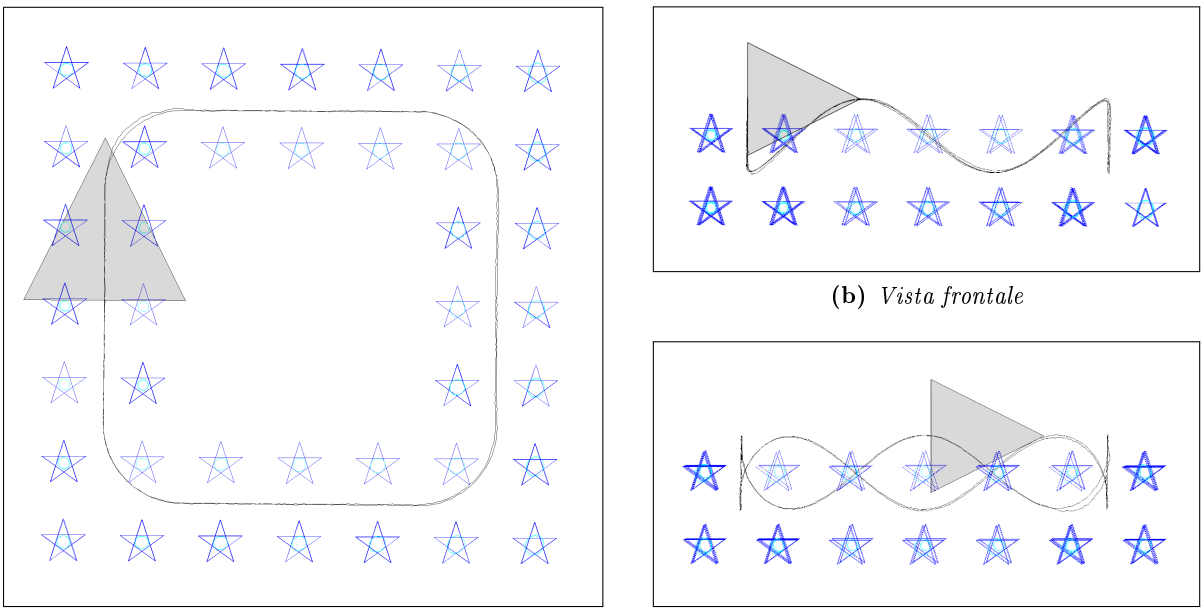


Figura 9.3: *Risultato dello SLAM in ambiente simulato con la parametrizzazione* Inverse Scaling



(a) *Vista dall'alto*

(b) *Vista frontale*

(c) *Vista laterale*

Figura 9.4: *Risultato dello SLAM in ambiente simulato con la parametrizzazione* FID

9.3 SLAM con i diversi metodi di tracking

L'applicazione allo SLAM dei metodi di patch tracking sviluppati, non permette di testarli in maniera sistematica variando solamente i parametri riguardanti il tracking e lasciando inalterati i rimanenti. Questo perché ogni metodo di tracking ha delle proprie caratteristiche che richiedono un adeguato settaggio anche dei restanti parametri: prendiamo per esempio il patch warping, la precisione del suo tracking è buona, ma i landmark non riescono ad essere seguiti per lunghi tratti, ed è quindi necessario seguire un numero di landmark molto alto. Per valutare le prestazioni dello SLAM con le varie tecniche di tracking, abbiamo effettuato per ognuna di esse decine di test, variando i parametri dell'intero algoritmo di SLAM con l'obiettivo di ottimizzarne il loro valore. Lo spazio di ricerca dei parametri è estremamente ampio e considerando che ogni test effettuato non dura meno di mezz'ora, una ricerca esaustiva della miglior combinazione di parametri non è stata possibile. L'ottimizzazione dei parametri è stata guidata della sensibilità al problema dello SLAM sviluppata dall'autore. I test di questa sezione sono stati effettuati sul *dataset* di Bovisa2, senza utilizzare l'odometria e con la parametrizzazione FID nella variante *Shared*. I risultati sono stati valutati visivamente sovrapponendo alla mappa satellitare dell'ambiente la traiettoria del robot stimata, dopo averla opportunamente scalata e ruotata.

Delle decine di test effettuati, abbiamo estratto il miglior risultato ottenuto con ogni metodo di tracking, allo scopo di mostrare la massima qualità dello SLAM raggiungibile con ognuno di essi. I parametri dei test rappresentativi di ogni metodo vengono elencati in Tabella 9.1, mentre in Tabella 9.2 vengono mostrate alcune statistiche raccolte durante le quattro sessioni di SLAM. In Figura 9.5 sono mostrate le quattro traiettorie ottenute.

Per l'utilizzo del patch tracking classico, è stato necessario adoperare la tecnica dell'aggiornamento delle patch, altrimenti anche una semplice rotazione di pochi gradi del robot intorno al proprio asse avrebbe provocato il fallimento del *matching* di tutti i landmark. L'utilizzo dell'aggiornamento delle patch provoca però errori di tracking che si propagano nel tempo. Per controbilanciare questo fenomeno è stato necessario utilizzare un numero di landmark molto alto. Anche per il metodo panoramico e il patch warping sono necessari molti landmark, nel primo caso per controbilanciare il fenomeno di slittamento delle feature, nel secondo caso a causa della breve vita media dei landmark. In tutti i casi si è utilizzata l'inizializzazione della distanza inversa delle feature sul piano, che per lo specifico *dataset* di Bovisa2, riduce lo *scale drift* della traiettoria del robot e permette di ottenere

un fattore di scala prossimo a quello reale a condizione di conoscere l'altezza da terra della camera. Si tenga presente che questi parametri sono stati tarati per lo specifico *dataset* ed è presumibile che un diverso ambiente, con una camera differente necessiti di parametri significativamente differenti.

Osservando le quattro traiettorie di Figura 9.5 è facilmente riconoscibile come il risultato migliore sia quello ottenuto con il tracking con patch prospettiche, benché il risultato del *patch warping* sia di poco inferiore.

Analizzando le statistiche di Tabella 9.2, osserviamo che il metodo con patch prospettiche mantiene nel filtro mediamente un numero inferiore di landmark e nonostante ciò ottiene il risultato migliore. Questo minor numero di landmark si riflette anche sui tempi medi delle operazioni di aggiornamento del filtro, aggiunta dei nuovi landmark e cancellazione di quelli ritenuti non necessari o addirittura dannosi (con stima della distanza inversa negativa). In realtà, i bassi valori delle operazioni di aggiunta e cancellazione, è presumibile che siano dovuti anche a una durata superiore della vita media dei landmark. D'altra parte il tempo medio del passo di *tracking* mostra invece come il metodo delle patch prospettiche sia computazionalmente più oneroso degli altri. Il *framerate* medio sembrerebbe dimostrare come il metodo del *patch warping* permetta di ottenere un risultato simile al metodo con patch prospettiche con un costo computazionale dimezzato.

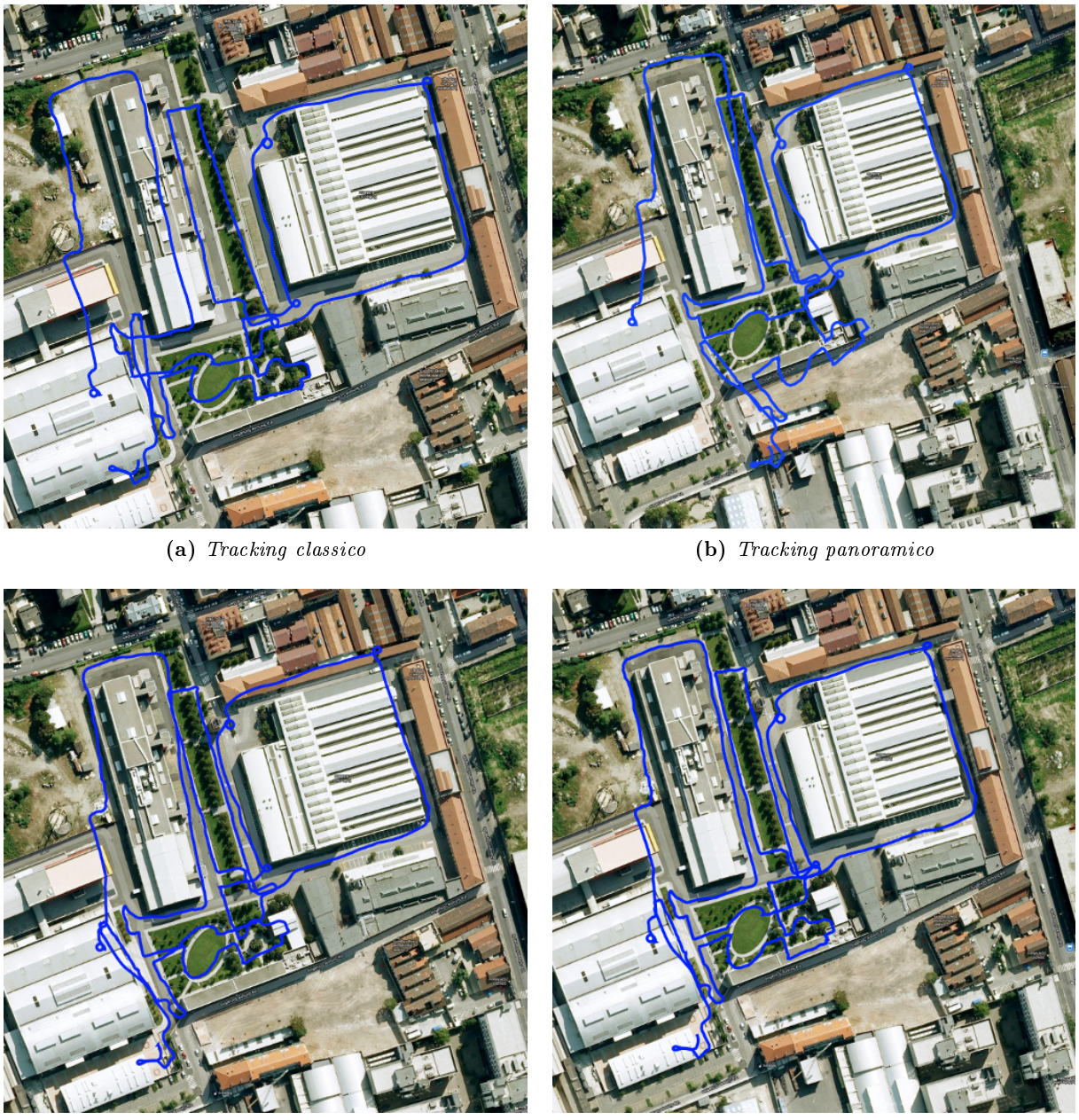
Nel confronto non abbiamo considerato la mappa, formata da tutti i landmark del filtro, in quanto difficilmente interpretabile e comparabile. Si tenga però presente che la sua qualità dipende molto dalla qualità del tracking di ogni singolo landmark, oltre che dalla traiettoria del robot, quindi il tracking con patch prospettiche otterrà presumibilmente anche la mappa migliore, a causa della miglior traiettoria e della durata superiore della vita media dei landmark.

Metodo di tracking	Classico	Panoramico	Warping	Patch prospettiche
Dimensione patch	15×15	15×15	21×21	31×31
Campo visivo patch	-	-	-	5^o
Larghezza panorama	-	1500 px	-	-
Stile panorama	-	superficie sfera	-	-
Aggiornamento patch	si	no	-	no
Agg. patch threshold	0.9	-	-	-
Correzione scala con SLAM	-	-	-	si
Cov. rumore trasl. ($\times 10^{-3}$)	25 25 25	25 25 25	25 25 25	25 25 25
Cov. rumore rot. ($\times 10^{-3}$)	25 25 25	25 25 35	25 25 35	25 25 25
Iniz. distanza inversa	sul piano	sul piano	sul piano	sul piano
d_{floor}	1	1.083	1	1
$\varsigma_{d_{floor}}$	0.001	0.01	0.01	0.01
α_{max}	80^o	80^o	80^o	80^o
η	10	10	10	2
Rumore sulla misura	1 px	1 px	1 px	1 px
Tipologia di griglia	classica	classica	uguale azimuth	uguale azimuth
Numero di colonne/settori	5	5	5	5
Numero di righe/tracce	3	4	4	4
Min landmark per cella	6	5	5	3
Qualità Harris detector	10^{-7}	10^{-7}	10^{-6}	10^8
Match threshold	0.9	0.85	0.85	0.9

Tabella 9.1: *Parametri di configurazione delle migliori sessioni di SLAM senza odometria per il dataset di Bovisa2, ottenute con i metodi di tracking sviluppati*

Metodo di tracking	Classico	Panoramico	Warping	Patch prosp.
Numero medio di landmark nel filtro	104.5	104.0	95.1	72.5
Tempo medio tracking	59.5 ms	85.6 ms	54.6 ms	150.3 ms
Tempo medio aggiornamento filtro	56.9 ms	55.9 ms	23.2 ms	20.1 ms
Tempo medio di aggiunta nuovi landmark	11.6 ms	8.5 ms	8.1 ms	9.2 ms
Tempo medio di cancellazione landmark	4.5 ms	4.8 ms	2.8 ms	2.2 ms
Framerate medio	6.5 fps	5.5 fps	10.3 fps	4.8 fps

Tabella 9.2: *Statistiche delle migliori sessioni di SLAM senza odometria per il dataset di Bovisa2, ottenute con i quattro metodi di tracking sviluppati, con i parametri di Tabella 9.1*



(a) *Tracking classico*

(b) *Tracking panoramico*

(c) *Tracking con patch warping*

(d) *Tracking con patch prospettiche*

Figura 9.5: *Traiettorie delle migliori sessioni di SLAM senza odometria per il dataset di Bovisa2, ottenute con i metodi di tracking sviluppati, con i parametri della Tabella 9.1*

9.4 Ambiente *outdoor*

I due *dataset* di Bovisa, e in generale gli ambienti *outdoor* possiedono delle specifiche caratteristiche delle quali è indispensabile tener conto nell'impostazione dei parametri dello SLAM. Negli ambienti *outdoor*, a differenza di quelli *indoor*, il robot si muove generalmente su un terreno più grezzo e sconnesso, che provoca vibrazioni e sfocamenti della telecamera, e un andamento molto rumoroso della variabile dell'orientamento del robot. Né il modello di moto dato dall'odometria, né tanto meno quello a velocità costante sono in grado di catturare quest'ultimo fenomeno, e per questo motivo è opportuno impostare valori relativamente alti di covarianza del rumore che insiste sulla componente dell'orientamento del modello di moto. In questo modo l'EKF farà meno affidamento sulla predizione fatta dal modello di moto e darà maggior peso alle osservazioni. L'innalzamento del rumore sul modello di moto provoca un ampliamento dell'area di ricerca dei landmark sull'immagine, in modo che i loro descrittori possano essere correttamente associati nella fase di *matching* anche a fronte di predizioni delle osservazioni molto imprecise.

Sebbene nei *dataset outdoor* di Rawseeds sia disponibile il segnale GPS, questo in alcuni tratti risulta molto disturbato. Per valutare la qualità della traiettoria compiuta dal robot, essa è stata opportunamente scalata e ruotata, e sovrapposta alla mappa satellitare dell'ambiente, estratta da *Google Maps*.

9.4.1 Test in ambiente *outdoor* senza odometria

Le migliori sessioni di SLAM ottenute per i *dataset* di *Bovisa1* e *Bovisa2* senza l'uso dell'odometria sono state ottenute con i parametri di Tabella 9.3, e le statistiche rilevate sono mostrate in Tabella 9.4. La Figura 9.6 mostra le traiettorie ottenute mentre nelle Figure 9.7 e 9.8 vengono mostrate anche le mappe dei landmark, viste da tre prospettive differenti.

Osservando i parametri delle due sessioni di SLAM notiamo come essi siano molto simili fra loro. Nel *dataset* di Bovisa1, per la parte traslazionale del rumore sul modello di moto, abbiamo dei valori inferiori rispetto a Bovisa2: è possibile che in Bovisa2 il robot acceleri e freni più bruscamente che in Bovisa1, in quanto attribuendo un valore di 0.01 anche in Bovisa2 si rilevava un leggero scale drift. Notiamo inoltre la notevole differenza del parametro η, che determina la variazione di incertezza fra le distanze dei landmark inizializzati sul piano nelle vicinanze del robot e quelli lontani; in Bovisa1 il robot percorre dei tratti in cui si trova molto vicino a dei muri, quindi alcuni landmark che vengono inizializzati molto lontani saranno in realtà molto vicini, e affinché il loro reale valore di distanza sia all'interno del range probabilistico di inizializzazione, è necessario attribuire un'alta incertezza. In Bovisa2 ciò non accade, ed è quindi possibile ridurre tale incertezza.

Nel *dataset* di Bovisa2, in molte parti della sequenza di immagini, le feature nelle vicinanze del robot sono di scarsa qualità e per rilevarle abbiamo ridotto la soglia di accettabilità del *corner detector*. Osservando le statistiche, notiamo come questo si traduca in un maggior numero medio di landmark nel filtro, con un conseguente innalzamento dei tempi di tutte le altre statistiche. Ricordiamo che i landmark in prossimità del robot sono di fondamentale importanza per stimare correttamente la parte traslazionale del moto del robot.

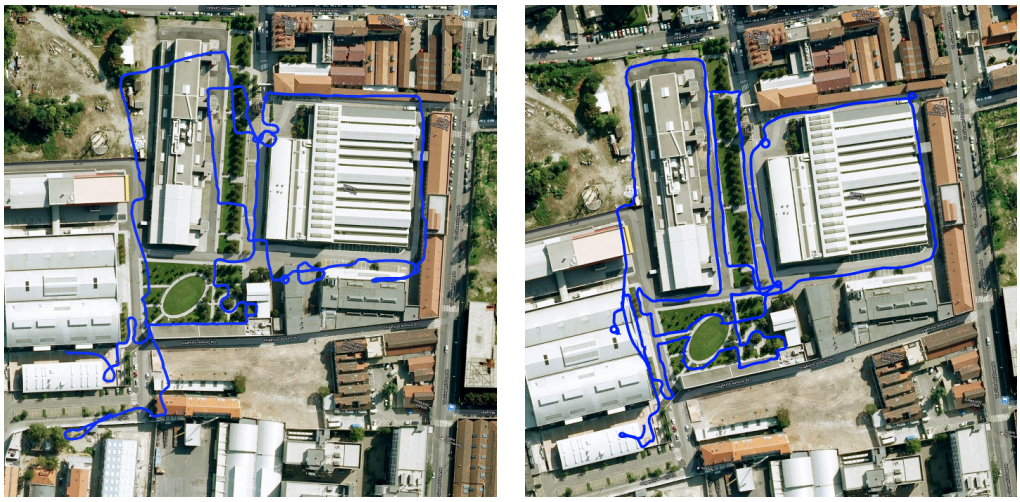
Osservando il risultato ottenuto per Bovisa1 notiamo come traiettoria del robot sia pressoché perfetta sin quasi a metà *dataset*. Il punto critico in cui commette un piccolo errore di orientamento avviene quando il robot percorre un tratto rettilineo parallelamente a un muro molto vicino, e dall'altra parte vi sono persone in movimento. Nel *dataset* di Bovisa2 invece viene commesso un piccolo errore di orientamento percorrendo un tratto di strada con poche feature e di scarsa qualità, facenti parte del bordo di un marciapiede sul quale si ha l'effetto di slittamento.

	Bovisa1	Bovisa2
Tipologia di tracking	patch prospettiche	patch prospettiche
Correzione scala con SLAM	si	si
Dimensione patch	31×31	31×31
Campo visivo patch	5^o	5^o
Cov. rumore traslazione	0.01 0.01 0.01	0.025 0.025 0.025
Cov. rumore rotazione	0.025 0.025 0.025	0.025 0.025 0.025
Parametrizzazione	FID *SharedFixedFrame*	FID *Shared*
Iniz. distanza inversa	sul piano	sul piano
d_{floor}	1	1
$\varsigma_{d_{floor}}$	0.01	0.01
α_{max}	80^o	80^o
η	10	2
Rumore sulla misura	1 px	1 px
Tipologia di griglia	uguale azimuth	uguale azimuth
Numero di settori	5	5
Numero di tracce	4	4
Num. landmark per cella	3	3
Qualità Harris detector	$5 \cdot 10^{-6}$	10^{-8}
Match threshold	0.85	0.9

Tabella 9.3: *Parametri delle migliori sessioni di SLAM senza odometria per i dataset di Bovisa1 e Bovisa2*

	Bovisa1	Bovisa2
Numero medio di landmark nel filtro	61.0	72.5
Tempo medio tracking	96.0 ms	150.3 ms
Tempo medio aggiornamento filtro	12.1 ms	20.1 ms
Tempo medio di aggiunta nuovi landmark	6.2 ms	9.2 ms
Tempo medio di cancellazione landmark	1.1 ms	2.2 ms
Framerate medio	8.1 fps	4.8 fps

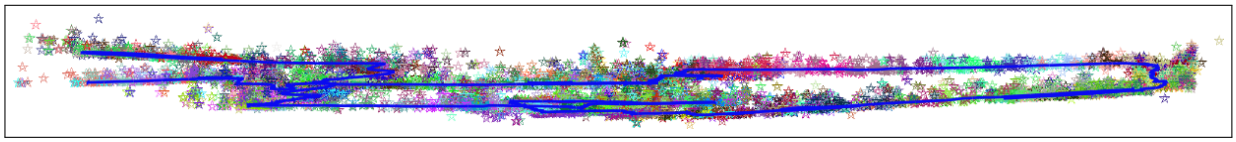
Tabella 9.4: *Statistiche delle migliori sessioni di SLAM senza odometria per i dataset di Bovisa1 e Bovisa2*



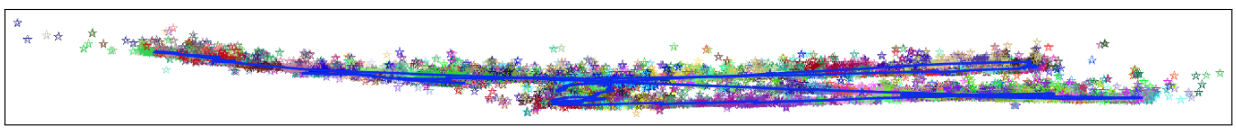
(a) *Bovisa1* **(b)** *Bovisa2*

Figura 9.6: *Traiettoria della migliore sessione di SLAM senza odometria per i dataset di Bovisa1 e Bovisa2*

(a) *Vista dall'alto*

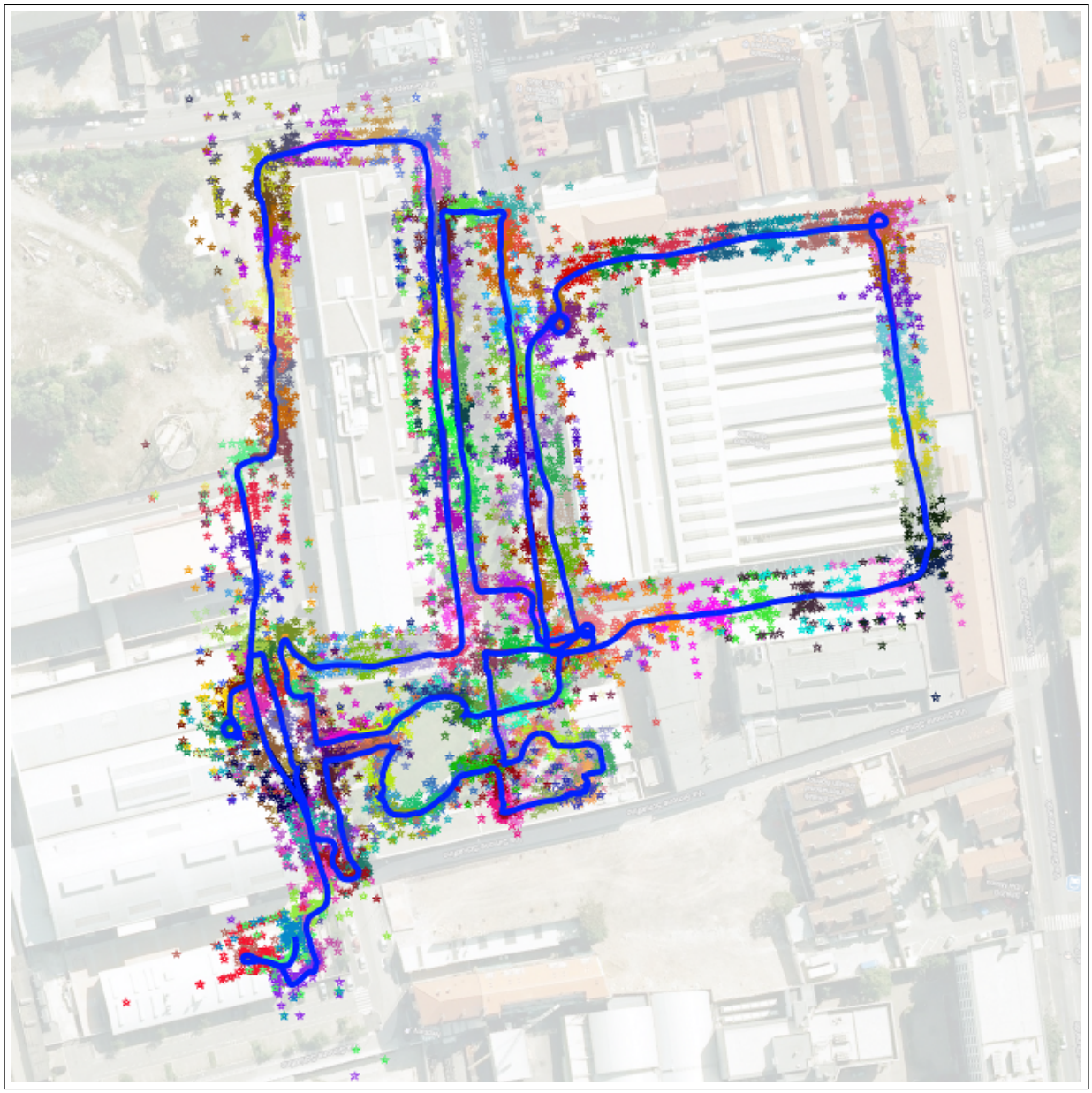


(b) *Vista frontale*

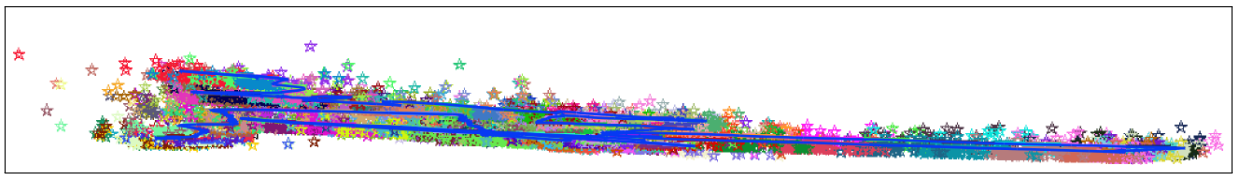


(c) *Vista laterale*

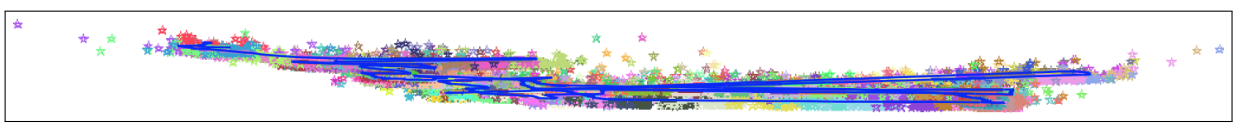
Figura 9.7: *Mappa della migliore sessione di SLAM senza odometria per il dataset di Bovisa1*



(a) *Vista dall'alto*



(b) *Vista frontale*



(c) *Vista laterale*

Figura 9.8: *Mappa della migliore sessione di SLAM senza odometria per il dataset di Bovisa2*

9.4.2 Test in ambiente *outdoor* con odometria

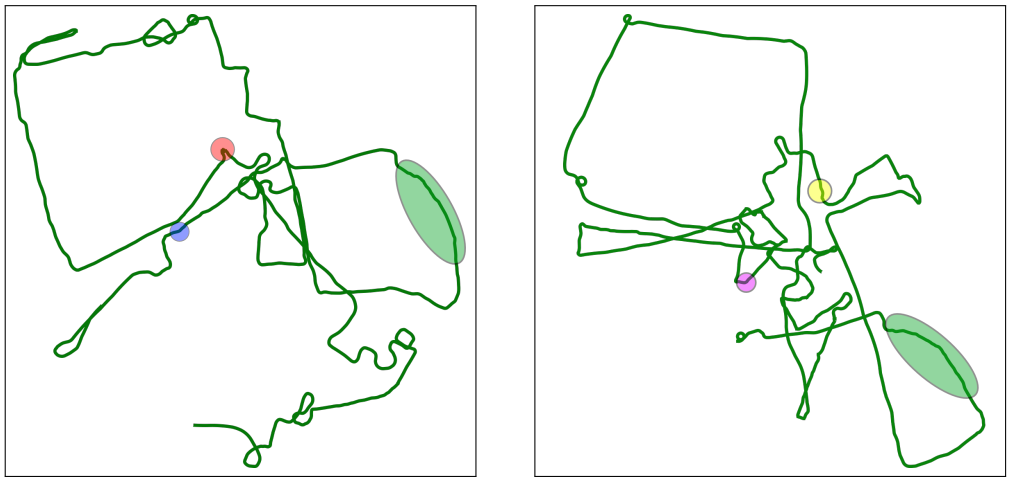
Nell'ambiente *outdoor* dei due *dataset* di Bovisa1 e Bovisa2 il robot percorre tratti di strada sterrata, sale e scende dai marciapiedi, passa sopra a tombini e avvallamenti della strada. Tutti questi eventi, provocano una bassa qualità dell'odometria per quanto riguarda le rotazioni. L'odometria dei due *dataset* è mostrata in Figura 9.9. Dato che gli errori nella componente rotazionale del moto sono in alcuni punti molto marcati, affinché lo SLAM funzioni e riesca ad associare correttamente i descrittori dei landmark anche a fronte di predizioni delle osservazioni molto errate, è necessario aumentare il valore di covarianza del rumore agente sul modello di moto per questa componente.

I parametri delle due migliori sessioni di SLAM sono elencati in Tabella 9.5 e le relative statistiche in Tabella 9.6. Le traiettorie sono mostrate in Figura 9.10, mentre le mappe sono apprezzabili nelle Figure 9.11 e 9.12.

I parametri delle due sessioni di SLAM sono molto simili fra loro, e si discostano notevolmente da quelli utilizzati nel caso senza odometria per il rumore sulla componente rotativa del moto del robot intorno al proprio asse. Si noti inoltre fra i parametri di Bovisa2, come i valori di rumore sulla componente traslativa del modello di moto, siano minori rispetto al caso senza odometria, in quanto la predizione effettuata dall'odometria è molto buona. La soglia di qualità dei punti caratteristici rilevati dal *corner detector* è invece più stingente, in quanto quelli vicini al robot, che con tale soglia vengono in buona parte scartati a causa della loro scarsa qualità per lo specifico *dataset* di Bovisa2, non sono più di fondamentale importanza per la componente traslativa del moto del robot.

Le statistiche mostrano come l'aumento del rumore sulla componente rotazionale del modello di moto innalzi il tempo necessario a effettuare il tracking delle feature, dovendo trasformare in prospettiche aree di ricerca molto più grandi ed effettuare la correlazione delle patch su di esse.

I risultati delle sessioni di SLAM sono molto simili a quelli ottenuti senza odometria. Dato che gli unici errori commessi nelle sessioni di SLAM senza odometria erano di tipo orientamentale, l'odometria potrebbe addirittura peggiorare i risultati, a causa della scarsissima qualità delle rotazioni stimate.



(a) *Bovisa1* **(b)** *Bovisa2*

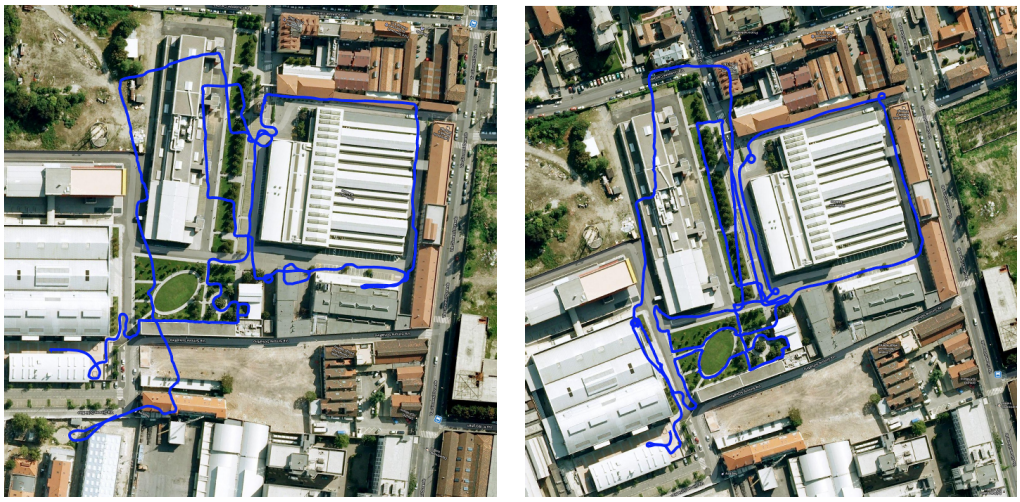
Figura 9.9: *Odometria dei dataset di Bovias1 e Bovisa2. Le parti evidenziate mostrano gli errori più grandi, facilmente rilevabili confrontando le traiettorie con le buone soluzioni mostrate in precedenza nella Sezione 9.4.1. In entrambe le traiettorie, in colore verde è mostrato l'errore che viene commesso quando il robot percorre un tratto di sterrato lungo circa 80 metri. In colore rosso è indicato l'errore commesso dal robot quando rimane letteralmente incastrato nel salire uno scivolo, e dev'essere spinto manualmente dall'operatore per disincagliarsi. L'errore di orientamento commesso è di almeno 180°. In colore giallo viene invece mostrato l'errore commesso dal robot quando esce dal marciapiede sul quale si muove e finisce per incastrarsi in una buca dell'aiuola vicina. In questo caso l'errore commesso è di circa 90°. Gli altri errori non sono invece facilmente spiegabili dall'osservazione del filmato.*

	Bovisa1	Bovisa2
Tipologia di tracking	patch prospettiche	patch prospettiche
Correzione scala con SLAM	si	si
Dimensione patch	31×31	31×31
Campo visivo patch	5^o	5^o
Cov. rumore traslazione	0.01 0.01 0.01	0.01 0.01 0.025
Cov. rumore rotazione	0.025 0.025 0.055	0.025 0.025 0.075
Parametrizzazione	FID *SharedFixedFrame*	FID *SharedFixedFrame*
Iniz. distanza inversa	sul piano	sul piano
d_{floor}	1	1
$\varsigma_{d_{floor}}$	0.01	0.01
α_{max}	80^o	80^o
η	10	10
Rumore sulla misura	1 px	1 px
Tipologia di griglia	uguale azimuth	uguale azimuth
Numero di settori	5	3
Numero di tracce	4	4
Num. landmark per cella	5	5
Qualità Harris detector	$5 \cdot 10^{-6}$	10^{-6}
Match threshold	0.85	0.85

Tabella 9.5: *Parametri delle migliori sessioni di SLAM con odometria per i dataset di Bovisa1 e Bovisa2*

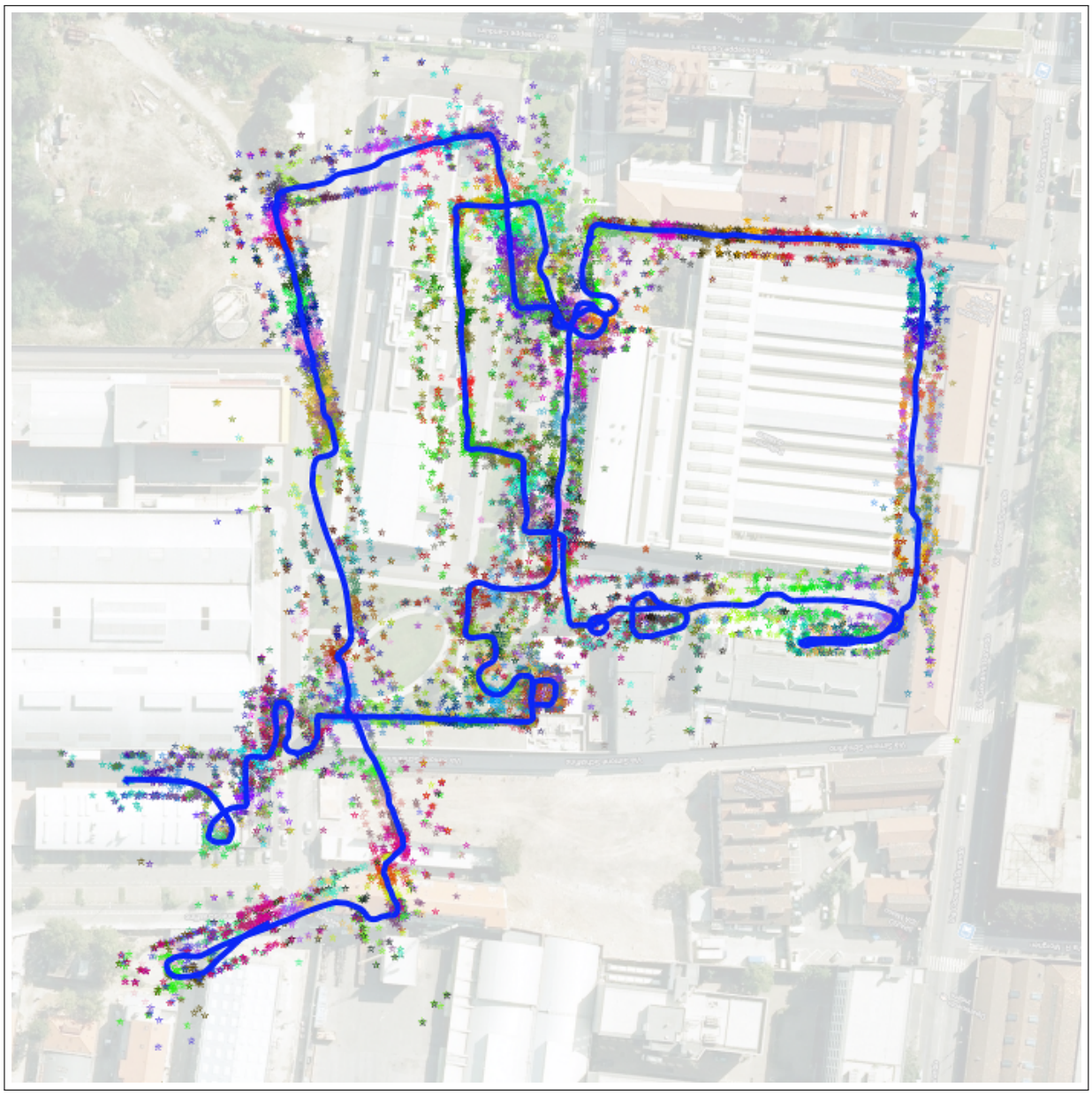
	Bovisa1	Bovisa2
Numero medio di landmark nel filtro	88.0	65.1
Tempo medio tracking	295.0 ms	223.3 ms
Tempo medio aggiornamento filtro	24.3 ms	11.0 ms
Tempo medio di aggiunta nuovi landmark	9.0 ms	5.7 ms
Tempo medio di cancellazione landmark	2.3 ms	1.0 ms
Framerate medio	2.9 fps	4.0 fps

Tabella 9.6: *Statistiche delle migliori sessioni di SLAM con odometria per i dataset di Bovisa1 e Bovisa2*

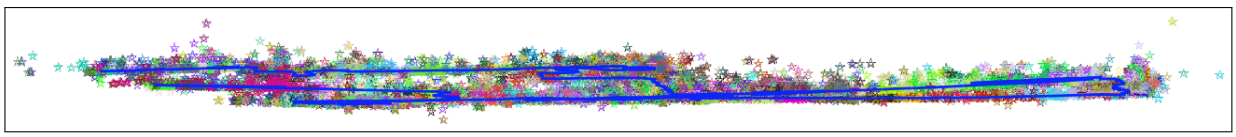


(a) *Bovisa1* **(b)** *Bovisa2*

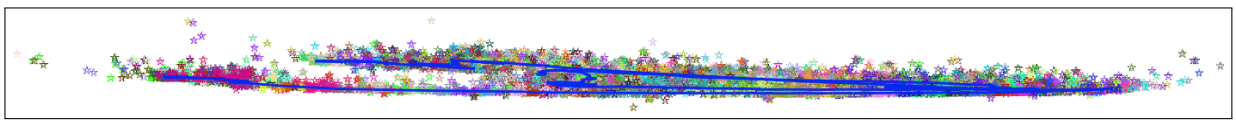
Figura 9.10: *Traiettoria della migliore sessione di SLAM con odometria per i dataset di Bovisa1 e Bovisa2*



(a) *Vista dall'alto*

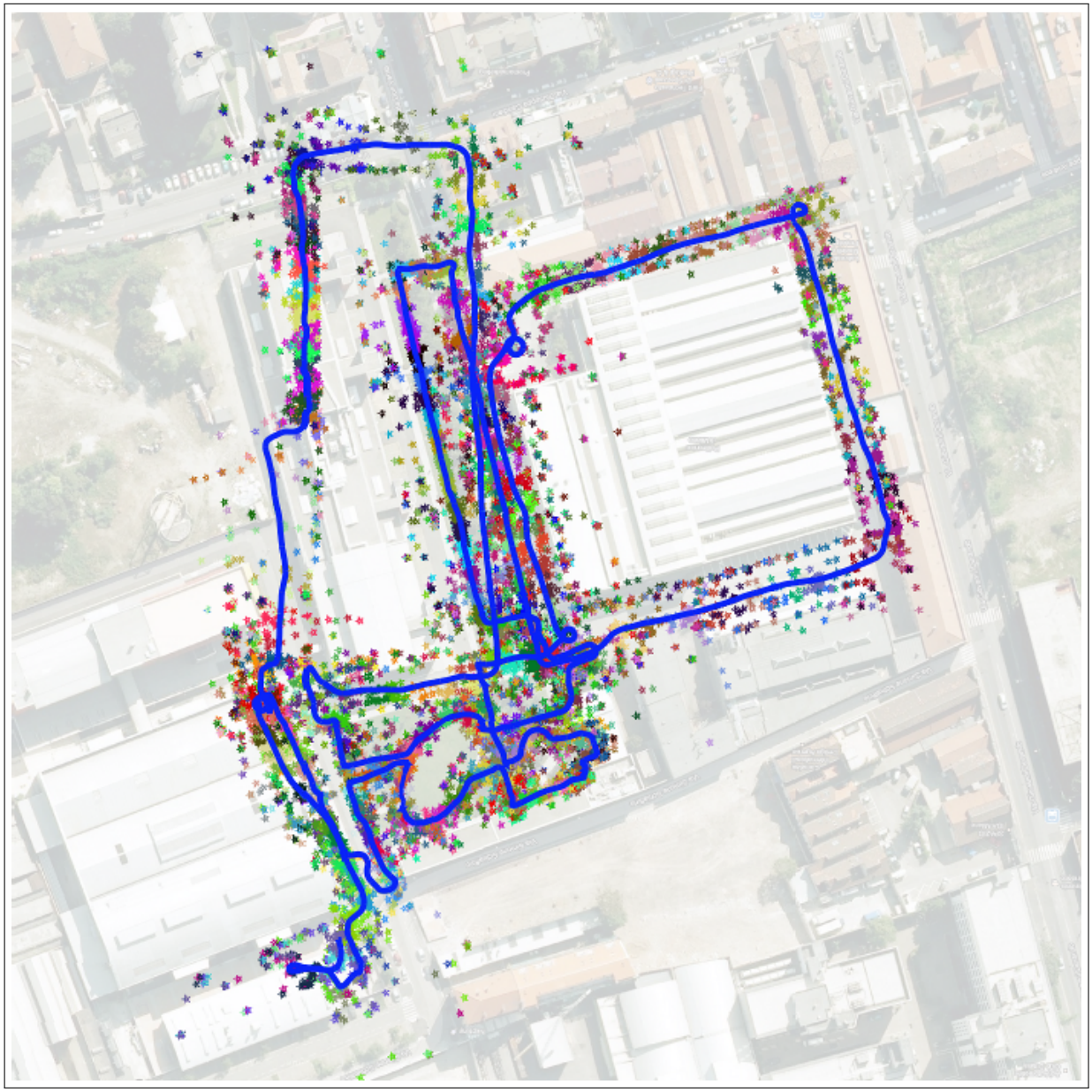


(b) *Vista frontale*

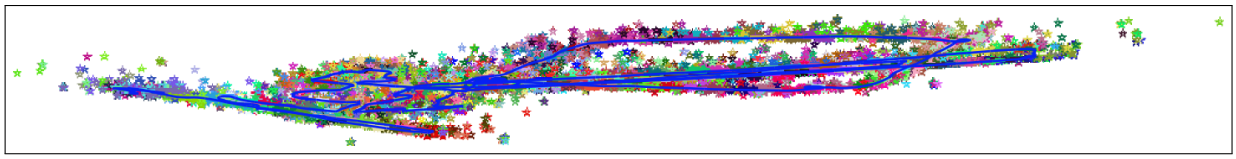


(c) *Vista laterale*

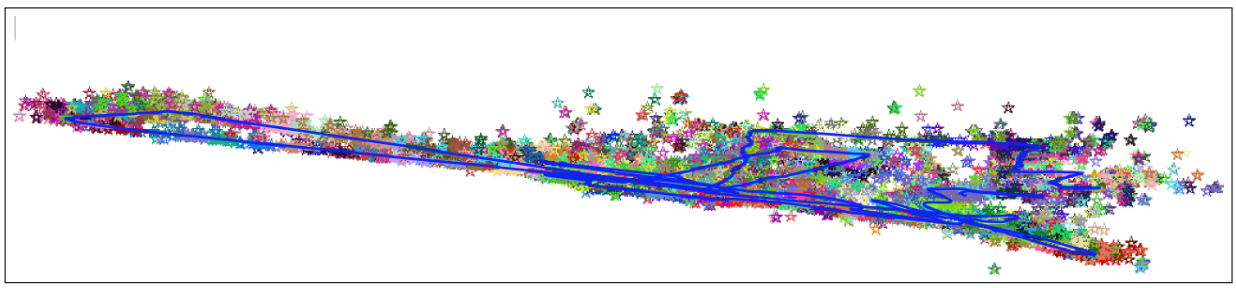
Figura 9.11: *Mappa della migliore sessione di SLAM con odometria per il dataset di Bovisa1*



(a) *Vista dall'alto*



(b) *Vista frontale*



(c) *Vista laterale*

Figura 9.12: *Mappa della migliore sessione di SLAM con odometria per il dataset di Bovisa2*

9.5 Ambiente *indoor*

Negli ambienti *indoor*, generalmente la superficie sulla quale si muove il robot è di miglior qualità, e non presenta buche o avvallamenti come nel caso *outdoor*. Grazie a queste caratteristiche il moto del robot è molto meno rumoroso e la covarianza sul rumore del modello di moto può essere ridotta notevolmente. Nel *dataset* di Bicocca a nostra disposizione, il robot segue un percorso privo di salite o discese e senza effettuare manovre brusche. Queste caratteristiche ci permettono di vincolare ancor più il modello di moto nelle sue componenti verticale e rotazionali α e β (non γ che è la rotazione del robot intorno al proprio asse). D'altra parte il *dataset* in questione presenta anche delle caratteristiche fotometriche che mettono a dura prova il *Visual SLAM*: il robot percorre corridoi estremamente stretti e bui, sulle cui pareti vi sono poche feature per il tracking; sul pavimento lucido si riflettono le luci del soffitto, le cui feature non sono quindi statiche; dai corridoi passa poi a spazi molto luminosi e ampi. L'inizializzazione delle feature sul piano, negli ambienti *indoor*, non è in grado di stabilizzare il fattore di scala della traiettoria del robot, in quanto la maggior parte dei landmark inizializzati non si trova sul pavimento, ma sulle pareti.

Per il *dataset* di Bicocca è disponibile la *ground truth*, mostrata in Figura 9.13. A differenza del segnale GPS per i *dataset outdoor* la sua qualità è molto buona, ed è stata utilizzata per valutare la bontà delle sessioni di SLAM effettuate.

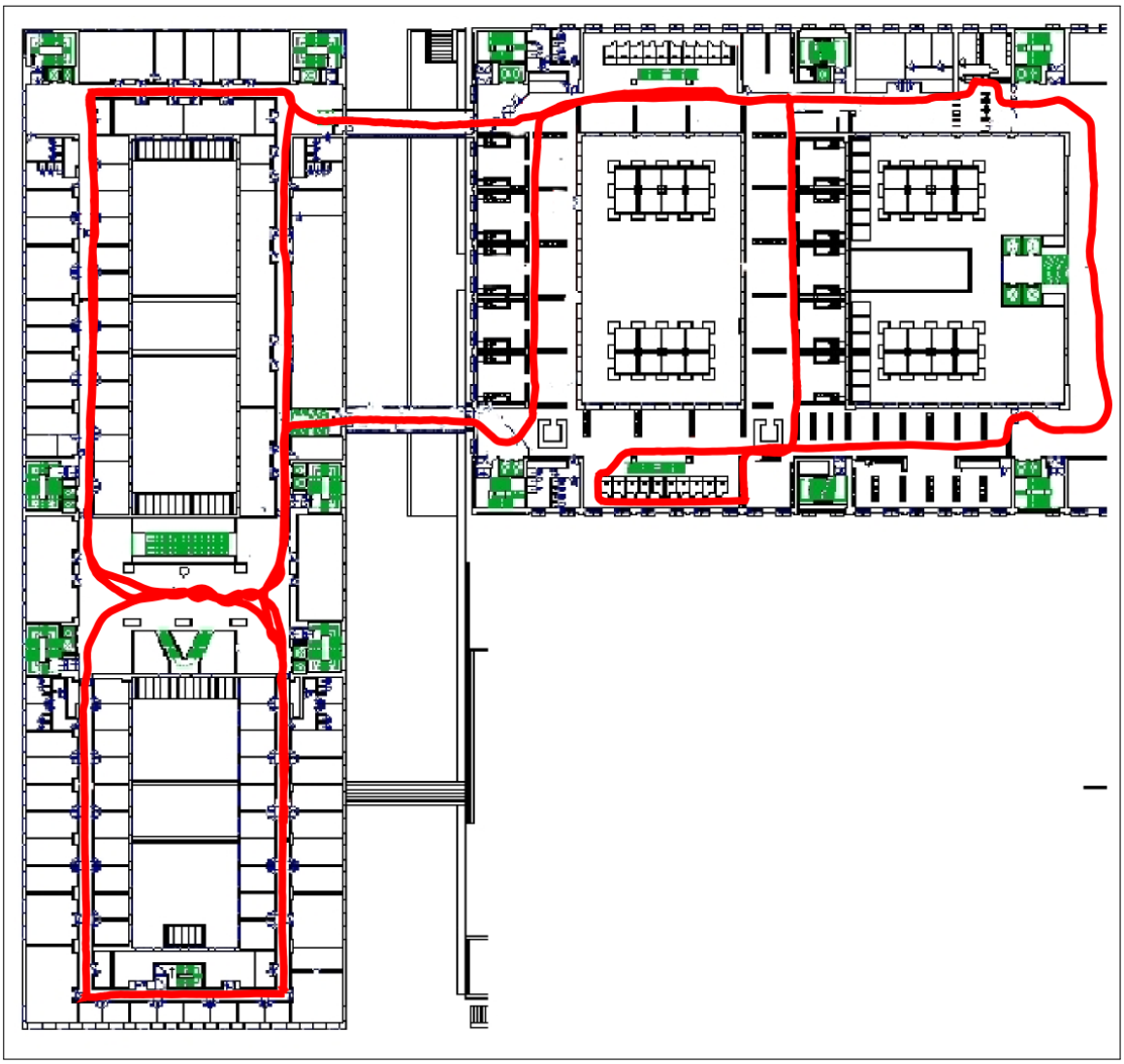


Figura 9.13: *Ground truth per il dataset di Bicocca*

9.5.1 Test in ambiente *indoor* senza odometria

La miglior sessione di SLAM senza l'uso dell'odometria per il *dataset* di Bicocca è mostrata in Figura 9.14, ed è stata ottenuta con i parametri di Tabella 9.7. Le statistiche della sessione sono invece riportate in Tabella 9.8.

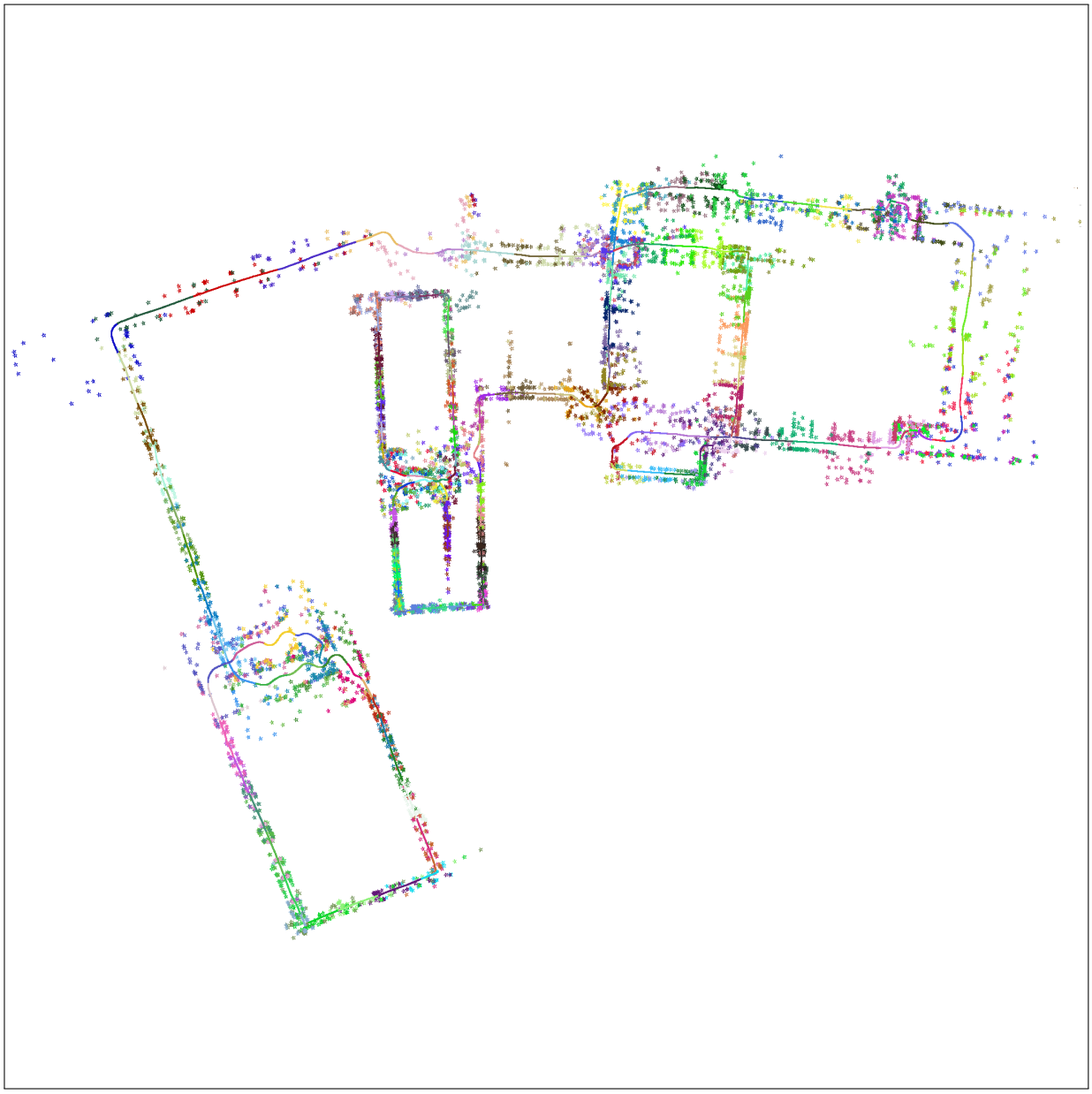
Analizzando i parametri utilizzati, sembrerebbe che anche per gli ambienti *indoor* il metodo di tracking con patch prospettiche si comporti meglio dei suoi concorrenti. Negli ambienti *indoor*, i corner per il tracking sono generalmente di qualità migliore rispetto a quelli *outdoor*: negli ambienti esterni vengono seguite feature derivanti da imperfezioni della strada o feature della chioma degli alberi mentre in ambienti interni si hanno generalmente feature derivanti da spigoli. Grazie a questa proprietà, è possibile considerare patch di dimensione ridotta, con campo visivo di 4^o e una risoluzione di 21×21 pixel. In questo modo si ottiene un notevole risparmio computazionale nella fase di tracking, come verificabile in Tabella 9.8. Sebbene la qualità delle feature sia migliore, l'aspetto delle loro patch cambia molto più rapidamente che in ambienti *outdoor*, in quanto i landmark a esse associati sono molto più vicini al robot. È quindi necessario seguire una gran quantità di punti e impostare una soglia di matching non troppo stringente. Data la povertà di feature lungo i corridoi del *dataset*, abbiamo attribuito un notevole peso al modello di moto a velocità costante, tranne che sulla componente della rotazione del robot intorno al proprio asse. Abbiamo utilizzato l'inizializzazione dei landmark sul piano, sebbene quella costante si comporti pressoché allo stesso modo. Il risultato ottenuto non è comunque soddisfacente, mostrando un fortissimo *scale drift*.

Tipologia di tracking	patch prospettiche
Correzione scala con SLAM	si
Dimensione patch	21×21
Campo visivo patch	4^o
Cov. rumore traslazione	0.001 0.001 0.0
Cov. rumore rotazione	0.0 0.0 0.01
Parametrizzazione	FID *SharedFixedFrame*
Iniz. distanza inversa	sul piano
d_{floor}	1.083
$\varsigma_{d_{floor}}$	0.5
α_{max}	80^o
η	10
Rumore sulla misura	1 px
Tipologia di griglia	uguale raggio
Numero di settori	5
Numero di tracce	4
Numero di landmark per cella	5
Qualità Harris detector	$7 \cdot 10^{-6}$
Match threshold	0.8

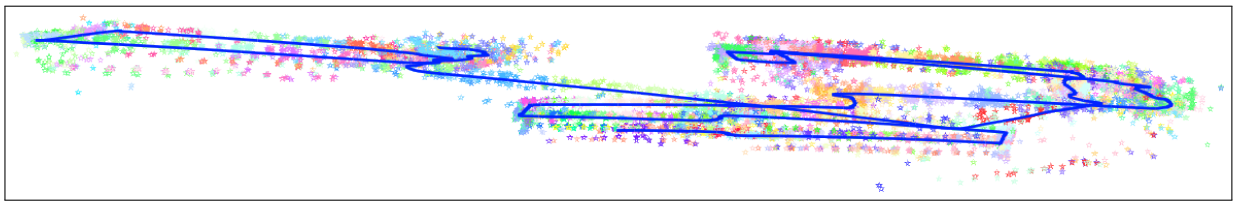
Tabella 9.7: *Parametri della migliore sessione di SLAM senza odometria per il dataset di Bicocca*

Numero medio di landmark nel filtro	73.7
Tempo medio tracking	36.9 ms
Tempo medio aggiornamento filtro	35.6 ms
Tempo medio di aggiunta nuovi landmark	13.5 ms
Tempo medio di cancellazione landmark	1.6 ms
Framerate medio	9.9 fps

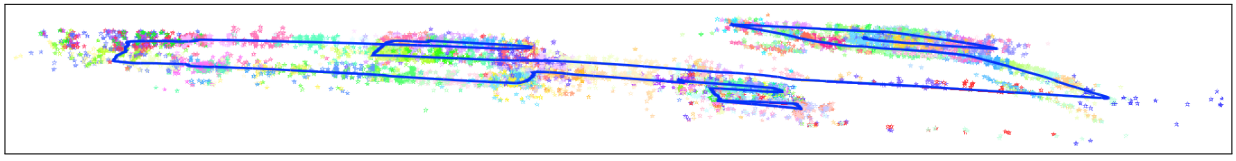
Tabella 9.8: *Statistiche della migliore sessione di SLAM senza odometria per il dataset di Bicocca*



(a) *Vista dall'alto*



(b) *Vista frontale*



(c) *Vista laterale*

Figura 9.14: *Risultato della migliore sessione di SLAM senza odometria per il dataset di Bicocca*

9.5.2 Test in ambiente *indoor* con odometria

In ambienti interni, grazie alla buona qualità della superficie sulla quale generalmente il robot si muove, l'odometria non presenta errori macroscopici. In Figura 9.15 è mostrata l'odometria per il *dataset* di Bicocca. È facilmente verificabile come la componente traslativa sia di ottima qualità mentre quella rotazionale possieda delle imperfezioni.

Nella Tabella 9.9 sono elencati i parametri per la migliore sessione di SLAM ottenuta, mentre la Tabella 9.10 mostra le relative statistiche. La buona qualità della traiettoria ottenuta è visibile in Figura 9.16. Per completezza in Figura 9.17 viene mostrata anche la mappa da tre punti di vista differenti.

Per quanto riguarda i parametri del tracking, le impostazioni rimangono le stesse del caso senza odometria. Anche la covarianza del rumore sul modello di moto è simile, con l'unica differenza che ora è possibile rilassare il vincolo che il moto del robot debba risiedere sul piano, grazie alla buona qualità dell'odometria. L'ottima qualità della componente traslativa ci permette inoltre di attribuire un'incertezza maggiore alla distanza inversa dei landmark in fase di inizializzazione, che ricordiamo essere sconosciuta. L'utilizzo di questi livelli di incertezza senza l'uso dell'odometria porta generalmente al fallimento dell'algoritmo di SLAM. Rispetto al caso senza odometria, abbiamo alzato la soglia di *matching* per ottenere una precisione maggiore, al costo di una durata della vita media dei landmark inferiore che abbiamo controbilanciato con un alto numero di landmark nel filtro. Nel caso senza odometria era invece più importante la durata media della vita dei landmark, necessaria per ridurre lo *scale drift*. Come si può verificare dalle statistiche della Tabella 9.10, la scelta di un alto numero di landmark e di una soglia di matching relativamente alta si paga in termini di costo computazionale.

Le Figure 9.16 e 9.17 mostrano come in questo caso il *Visual SLAM* sia in grado di correggere l'odometria di Figura 9.15, ottenendo una traiettoria di ottima qualità.

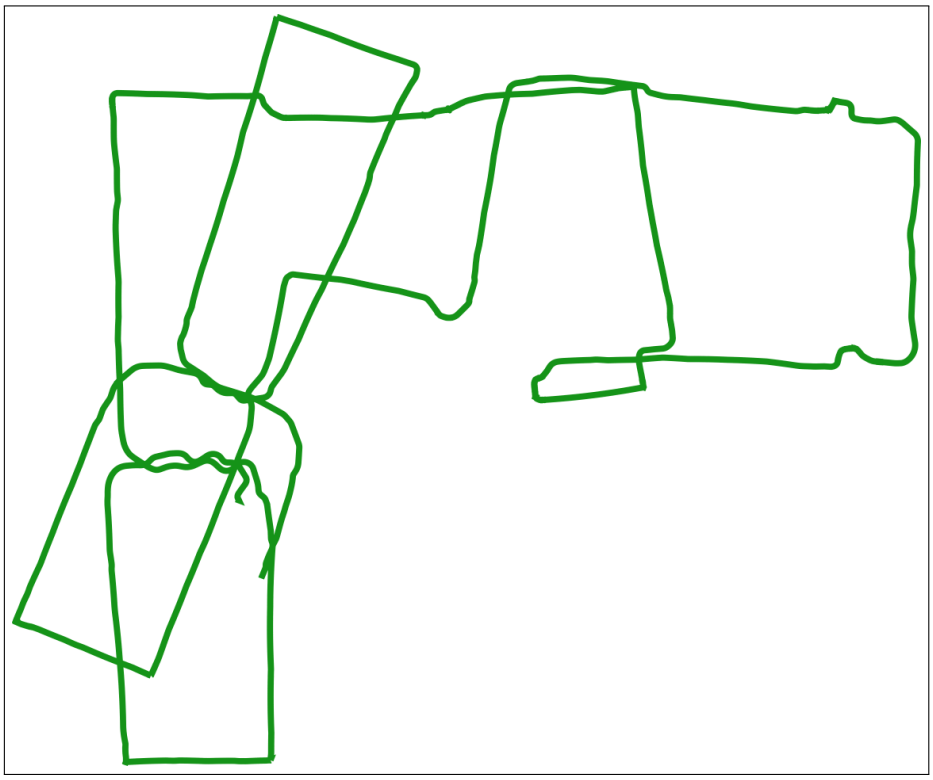


Figura 9.15: *Odometria del dataset di Bicocca*

Tipologia di tracking	patch prospettiche
Correzione scala con SLAM	si
Dimensione patch	21×21
Campo visivo patch	4^o
Cov. rumore traslazione	0.001 0.001 0.01
Cov. rumore rotazione	0.01 0.01 0.01
Parametrizzazione	FID *SharedFixedFrame*
Iniz. distanza inversa	sul piano
d_{floor}	1.083
$\varsigma_{d_{floor}}$	1
α_{max}	80^o
η	20
Rumore sulla misura	1 px
Tipologia di griglia	uguale raggio
Numero di settori	5
Numero di tracce	4
Numero di landmark per cella	10
Qualità Harris detector	10^{-6}
Match threshold	0.9

Tabella 9.9: *Parametri della migliore sessione di SLAM con odometria per il dataset di Bicocca*

Numero medio di landmark nel filtro	146.4
Tempo medio tracking	106.3 ms
Tempo medio aggiornamento filtro	96.4 ms
Tempo medio di aggiunta nuovi landmark	18.4 ms
Tempo medio di cancellazione landmark	6.8 ms
Framerate medio	4.1 fps

Tabella 9.10: *Statistiche della migliore sessione di SLAM con odometria per il dataset di Bicocca*

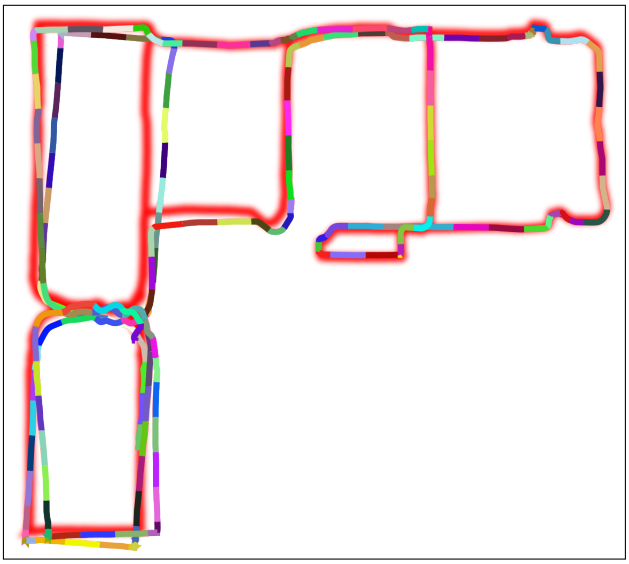
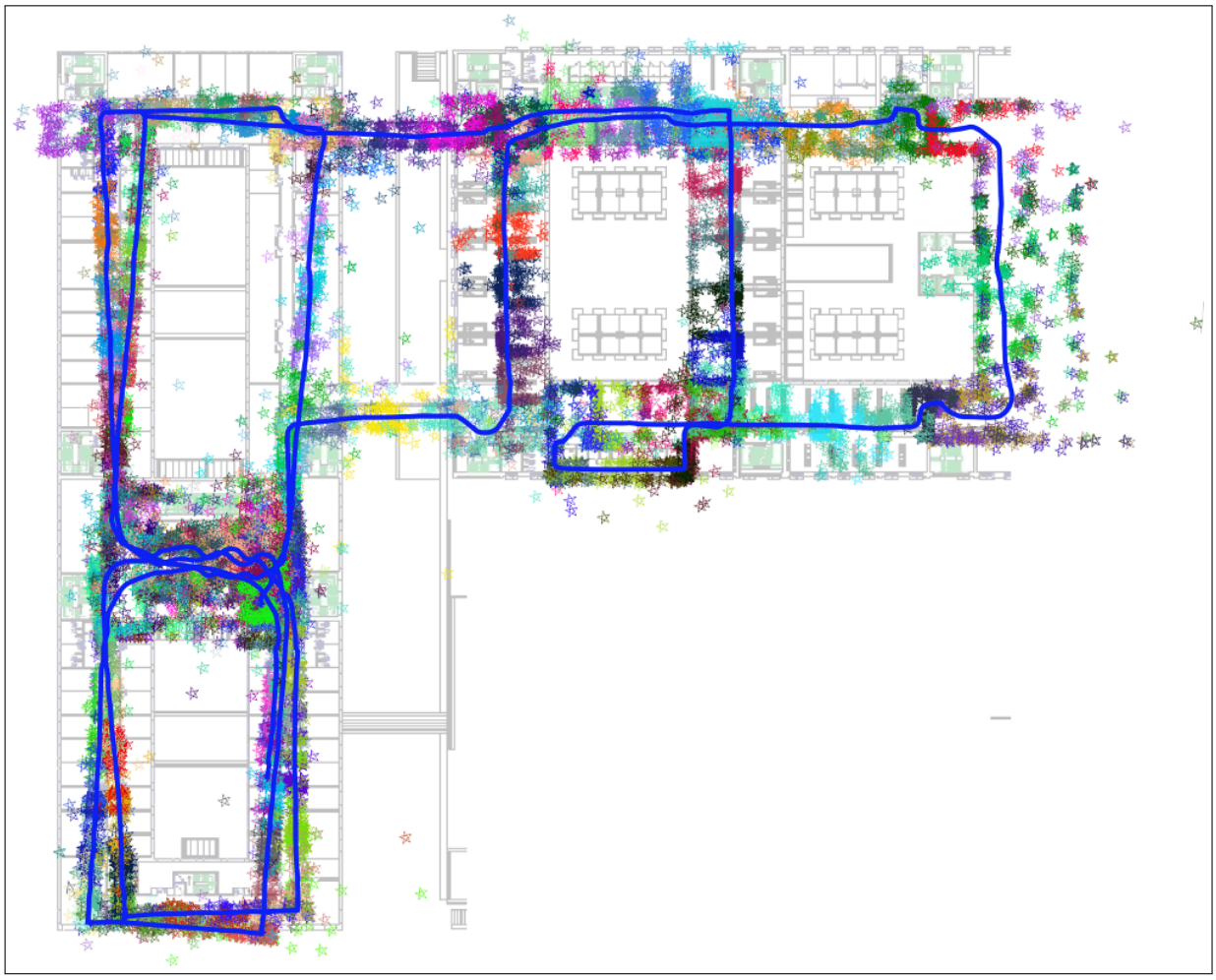
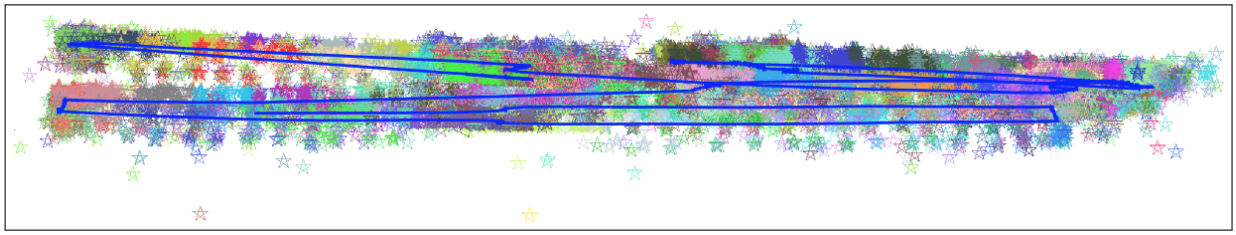


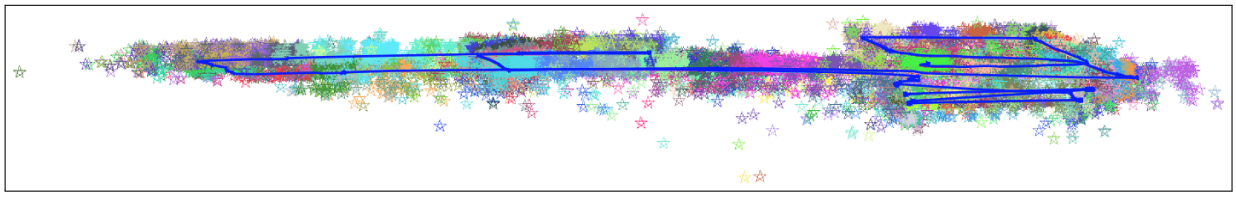
Figura 9.16: *Traiettoria della migliore sessione di SLAM con odometria per il dataset di Bicocca. La traiettoria color rosso in sottofondo è la ground truth. La traiettoria stimata ha diversi colori a indicare le diverse sottomappe ottenute con lo SLAM condizionalmente indipendente.*



(a) *Vista dall'alto*



(b) *Vista frontale*



(c) *Vista laterale*

Figura 9.17: *Mappa della migliore sessione di SLAM con odometria per il dataset di Bicocca*

9.6 Considerazioni finali

Nelle sessioni di SLAM effettuate si è verificato come la qualità dello SLAM sia molto buona e stabile rispetto a piccole variazioni dei parametri, qualora il problema risulti ben condizionato, ossia in presenza di un buon numero di feature da seguire, la cui qualità sia sufficientemente alta. Al contrario, quando vi sono poche feature e di scarsa qualità, i risultati dello SLAM diventano molto sensibili ai parametri, in quanto la componente casuale degli errori di tracking aumenta notevolmente di peso. I risultati presentati in questo capitolo sono le migliori sessioni di SLAM ottenute su centinaia di test effettuati con parametri e metodi diversi. Per questo motivo i risultati mostrati sono in realtà un *upper bound* della qualità dello SLAM ottenibile con gli algoritmi sviluppati. Si consideri inoltre come i parametri siano stati tarati opportunamente per ogni specifico *dataset*, quindi i risultati ottenuti non hanno validità generale. Molti parametri dello SLAM sono fortemente correlati fra di loro, e non è stato possibile tararli in maniera indipendente e separata. La scelta dei parametri è stata fatta basandosi sulle caratteristiche dell'ambiente del *dataset* e della traiettoria del robot.

Sebbene i risultati mostrati non siano rappresentativi della vera efficacia dei metodi sviluppati, si può affermare che mediamente il metodo di tracking con patch prospettiche permetta di ottenere i risultati migliori rispetto ai metodi studiati. Le prestazioni della sua versione con adattamento della scala sono leggermente superiori a quelle ottenibili con la versione base, benché abbiano un costo computazionale leggermente superiore. Per quanto il punto debole di questo metodo sia il suo alto costo computazionale, dovuto alla trasformazione prospettica di tutte le aree di ricerca delle feature a ogni frame, con il suo utilizzo i landmark riescono a essere osservati più a lungo, e ciò permette di ottenere buoni risultati anche con un numero ridotto di landmark. Si è verificato invece come il metodo dell'aggiornamento delle patch porti generalmente a un degrado della qualità dello SLAM, benché rimanga di fondamentale importanza quando si utilizza il metodo di patch tracking classico, in modo che le feature possano essere seguite anche a fronte di rotazioni del robot intorno al proprio asse.

Non è stato mostrato il confronto fra le parametrizzazioni utilizzando *dataset* reali, ma come nel caso simulato, la qualità dello SLAM ottenuto è pressoché identica nel caso delle tre parametrizzazioni UID, FHP e FID, in tutte le loro varianti. A parità di prestazioni qualitative è quindi preferibile l'utilizzo della parametrizzazione FID nelle varianti *Shared* o *SharedFixedFrame*, le quali permettono di condividere sette parametri degli otto di cui si compongono, fra i landmark inizializzati nello stesso istante discreto di tempo, ottenendo un notevole risparmio computazionale nella fase di aggiornamento del filtro di Kalman esteso.

Nei test in ambiente *outdoor* è stata dimostrata la validità del metodo di inizializzazione dei landmark sul piano, che permettono di ridurre notevolmente lo *scale drift* rispetto all'uso dell'inizializzazione costante classica. Anche in questo caso, le prestazioni del metodo sono fortemente legate alle caratteristiche del *dataset*, e in particolare alla validità dell'assunzione di planarità locale del terreno sul quale il robot si muove. In ambienti *indoor* questa proprietà non è quasi mai verificata e le prestazioni ottenibili con i due metodi sono simili.

L'odometria che si ottiene nei *dataset outdoor*, a causa delle imperfezioni del terreno sul quale si muove il robot, presenta errori molto marcati nella componente rotazionale. Nelle sessioni di SLAM effettuate su questi *dataset* si è riuscito a ottenere risultati privi di *scale drift* anche senza l'uso dell'odometria, quindi il suo utilizzo potrebbe addirittura portare a un degrado della qualità dello SLAM. In tutti i casi, la scarsa qualità della componente rotazionale dell'odometria, ha richiesto un notevole ampliamento delle aree di ricerca delle feature, con un conseguente aumento del costo computazionale del tracking. Contrariamente al caso *outdoor*, in ambienti *indoor*, la superficie sulla quale si muove il robot è generalmente di ottima qualità e di conseguenza anche l'odometria. Nello specifico *dataset indoor* da noi utilizzato, l'odometria è di fondamentale importanza per non avere *scale drift*, in quanto in molti tratti percorsi dal robot vi sono poche feature seguibili e di scarsa qualità. Il risultato mostrato dimostra come lo SLAM sia in grado di correggere la componente

rotazionale dell'odometria, ottenendo un risultato quasi perfetto.

Ricordiamo che tutte le sessioni di SLAM effettuate, sebbene utilizzino il metodo del *submapping* condizionalmente indipendente, non applichino nessun meccanismo di *loop closure*. È facile supporre che in sua presenza i risultati ottenibili sarebbero di qualità ancora superiore, sia in ambiente interni che in ambienti esterni.

Capitolo 10

Conclusioni e sviluppi futuri

Nel presente lavoro di tesi è stato esteso il *framework MoonSlam*, introducendo la possibilità di utilizzare una camera omnidirezionale per realizzare applicazioni di *Monocular SLAM*. Il nuovo modello di camera introdotto, ha determinato la necessità di rielaborare le parametrizzazioni dei landmark per camere prospettiche già presenti in *MoonSlam*. I test effettuati con le nuove parametrizzazioni, sia in ambiente simulato, sia con l'utilizzo di *dataset* reali, hanno dimostrato la loro efficacia, permettendo di ottenere delle traiettorie del robot e delle mappe di ottima qualità.

Sono stati sviluppati alcuni nuovi metodi per effettuare il *tracking* su immagini prodotte da camere omnidirezionali, in quanto non è stato possibile applicare direttamente i metodi di *tracking* per camere prospettiche, a causa della forte distorsione delle immagini omnidirezionali. Le tecniche di *tracking* per camere omnidirezionali introdotte sono: il *patch tracking* classico per camere prospettiche con l'aggiornamento delle patch, il *tracking* panoramico, il *patch warping* per camere omnidirezionali e il *tracking* con patch prospettiche. I test condotti applicando i quatto metodi di *tracking* allo SLAM, dimostrano l'adeguatezza dei metodi sviluppati, ottenendo dei risultati di buona qualità. Ognuno di essi ha però delle caratteristiche differenti, delle quali è necessario tener conto nella scelta del metodo da applicare e nell'impostazione dei parametri dell'intero algoritmo di SLAM:

- L'utilizzo del *patch tracking* classico per camere prospettiche, in concomitanza con l'aggiornamento delle patch, permette di ottenere delle prestazioni computazionali elevate, ma il *tracking* effettuato risulta alquanto impreciso. Per questo motivo, per poter ottenere dei risultati di SLAM soddisfacenti è necessario seguire molti punti caratteristici.
- Il metodo di *tracking* panoramico permette di ridurre notevolmente la distorsione delle immagini omnidirezionali di partenza, e realizza un tracking di discreta qualità senza l'applicazione del metodo di aggiornamento delle patch, il quale provoca generalmente un sensibile degrado dei risultati. La distorsione rimanente nelle immagini panoramiche, provoca però un frequente fenomeno di slittamento delle feature, che dev'essere controbilanciato effettuando il tracking di un alto numero di *feature*.
- Il *patch warping* omnidirezionale è in grado di effettuare il *tracking* delle *feature* a fronte di un movimento della camera qualsiasi e permette di ottenere una buona precisione. Il suo costo computazionale è relativamente basso, sebbene superiore a quello del *patch tracking* classico, dovendo effettuare a ogni frame una trasformazione omografica delle patch. D'altra parte, le ipotesi di planarità delle patch fanno sì che i landmark non riescano ad essere seguiti per lunghi periodi, rendendo nuovamente necessario l'utilizzo di un alto numero di landmark.
- Il *tracking* con patch prospettiche, e in particolar modo la versione che utilizza le informazioni dello SLAM per adattare il fattore di scala delle *patch*, permette generalmente di ottenere i risultati migliori, ma il suo costo computazionale è sensibilmente superiore a quello degli altri metodi.

È stata infine introdotta una nuova strategia di inizializzazione della distanza inversa dei landmark, che li dispone sul piano del terreno sul quale il robot si muove, in alternativa alla classica inizializzazione a distanza costante. I test realizzati in ambienti *outdoor*, dove l'ipotesi di planarità locale del terreno nei pressi del robot è generalmente verificata, dimostrano che l'applicazione del metodo permette di ottenere risultati privi di *scale drift*. L'adozione di questa tecnica insieme all'algoritmo di *tracking* con patch prospettiche, permette di ottenere delle mappe e delle traiettorie del robot per ambienti *outdoor* di ottima qualità, anche senza l'utilizzo dell'odometria come modello di moto. In ambienti *outdoor*, l'odometria presenta errori grossolani nella componente rotazionale, che impone valori di incertezza molto alti sulla relativa componente del modello di moto, provocando un aumento del costo computazionale del *tracking* a causa dell'ampliamento delle aree di ricerca nelle quali correlare le patch, nonché possibili nuovi errori di *matching*. In ambienti *indoor*, l'utilizzo del modello di moto a velocità costante porta facilmente ad avere un marcato *scale drift* della traiettoria percorsa dal robot, a causa della difficile inizializzazione dei landmark. In tale occasione l'uso dell'odometria, che generalmente non presenta i problemi del caso *outdoor*, permette di risolvere il problema e ottenere ottimi risultati.

In conclusione, i risultati ottenuti con il presente lavoro di tesi dimostrano le potenzialità delle camere omnidirezionali e degli algoritmi sviluppati per effettuare SLAM. Si consideri inoltre che sebbene utilizzando l'algoritmo di *submapping* condizionalmente indipendente, non è stato applicato nessun meccanismo di *loop closure*, il quale permetterebbe di migliorare ulteriormente i risultati ottenibili.

Sebbene il metodo di *tracking* con patch prospettiche permetta di ottenere i risultati migliori, in termini di qualità di traiettoria del robot e mappa, il suo alto costo computazionale ne rende difficile l'applicazione *real time*. L'algoritmo è però intrinsecamente parallelizzabile, elaborando ogni singola *feature* in maniera separata. Una tale implementazione permetterebbe a una macchina multi-core di abbattere notevolmente i tempi di esecuzione. Consideriamo inoltre che buona parte del tempo di esecuzione del metodo, viene impiegata per la trasformazione di porzioni di immagini omnidirezionali in prospettiche, operazione che potrebbe essere eseguita sulla GPU di una qualsiasi scheda grafica moderna, sfruttando le centinaia di core a loro disposizione.

Bibliografia

[BN99] Simon Baker and Shree K. Nayar. A theory of single-viewpoint catadioptric image formation. *International Journal of Computer Vision*, 35(1):1 – 22, 1999. 12

[BSC08] C. Burbridge, L. Spacek, and J. Condell. Monocular omnidirectional vision based robot localisation and mapping. *Proceedings of TAROS*, 2008. 44

[CDM08] Javier Civera, Andrew J. Davison, and J. M. M. Montiel. Inverse depth parametrization for monocular slam. *IEEE Transactions on Robotics*, 24(5):932–945, 2008. 44, 52, 59, 61

[CDR+07] Laura A. Clemente, Andrew J. Davison, Ian D. Reid, José Neira, and Juan D. Tardós. Mapping large loops with a single hand-held camera. In Wolfram Burgard, Oliver Brock, and Cyrill Stachniss, editors, *Robotics: Science and Systems*. The MIT Press, 2007. 46

[CGDM10] Javier Civera, Oscar G. Grasa, Andrew J. Davison, and J. M. M. Montiel. 1-point ransac for extended kalman filtering: Application to real-time structure from motion and visual odometry. *J. Field Robotics*, 27(5):609–631, 2010. 47

[CMM+11] Simone Ceriani, Daniele Marzorati, Matteo Matteucci, Davide Migliore, and Domenico G. Sorrenti. On feature parameterization for ekf-based monocular slam. In *In proceedings of 18th World Congress of the International Federation of Automatic Control (IFAC)*, pages 6829–6834, August 2011. 62

[Dav03] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *ICCV*, pages 1403–1410, 2003. 51

[Dea05] Matthew Deans. *Bearings-Only Localization and Mapping*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, September 2005. 36

[DH00] Matthew Deans and Martial Hebert. Experimental comparison of techniques for localization and mapping using a bearing-only sensor. In *ISER*, pages 395–404, 2000. 36

[ENT05] C. Estrada, J. Neira, and J. D. Tardós. Hierarchical slam: real-time accurate mapping of large environments. *IEEE Transactions on Robotics*, 21(4):588–596, August 2005. 45

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981. 47

[GMC06] Andrew P. Gee and Walterio W. Mayol-Cuevas. Real-time model-based SLAM using line segments. In *Advances in Visual Computing, Second International Symposium, ISVC 2006 Lake Tahoe, NV, USA, November 6-8, 2006. Proceedings, Part II*, volume 4292 of *Lecture Notes in Computer Science*, pages 354–363. Springer, 2006.

[GRMG11] Daniel Gutiérrez, Alejandro Rituerto, J. M. M. Montiel, and J. J. Guerrero. Adapting a real-time monocular visual slam from conventional to omnidirectional cameras. 2011. 44, 77

[HCBD07] Peter Hansen, Peter Corke, Wageeh W. Boles, and Kostas Daniilidis. Scale invariant feature matching with wide angle images. In *IROS*, pages 1689–1694. IEEE, 2007. 24

[HKM08] Steven A. Holmes, Georg Klein, and David W. Murray. A square root unscented kalman filter for visual monoSLAM. In *ICRA*, pages 3710–3716. IEEE, 2008. 32

[HT05] Joel A. Hesch and Nikolas Trawny. Simultaneous localization and mapping using an omni-directional camera. (2005-001), May 2005. 24, 44

[HT06a] Durrant-Whyte H. and Bailey T. Simultaneous localization and mapping: part 1. In *Robotics and Automation Magazine, IEEE*, pages 99–110, June 2006. 36

[HT06b] Durrant-Whyte H. and Bailey T. Simultaneous localization and mapping: part 2. In *Robotics and Automation Magazine, IEEE*, pages 108–117, June 2006.

[LL07] Thomas Lemaire and Simon Lacroix. Slam with panoramic vision. *J. Field Robotics*, 24(1-2):91–111, 2007. 44

[LLS05] Thomas Lemaire, Simon Lacroix, and Joan Solà. A practical 3D bearing only SLAM algorithm. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Edmonton, Canada, August 2005. 44

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. 19

[MDR04] N. Molton, A. Davison, and I. Reid. Locally planar patch features for real-time structure from motion. In *BMVC*, 2004. 25, 79

[MFB06] Thomas Mauthner, Friedrich Fraundorfer, and Horst Bischof. Region matching for omnidirectional images using virtual camera planes. 2006. 77

[MMMS09] Daniele Marzorati, Matteo Matteucci, Davide Migliore, and Domenico G. Sorrenti. On the use of inverse scaling in monocular slam. In *ICRA*, pages 2030–2036, 2009. 56

[MP03] Branislav Micusík and Tomás Pajdla. Omnidirectional camera model and epipolar geometry estimation by ransac with bucketing. In *SCIA*, pages 83–90, 2003.

[MR07] C. Mei and P. Rives. Single view point omnidirectional camera calibration from planar grids. In *IEEE International Conference on Robotics and Automation*, April 2007. 12

[NHS07] Viet Nguyen, Ahad Harati, and Roland Siegwart. A lightweight SLAM algorithm using orthogonal planes for indoor mobile robotics. In *IROS*, pages 658–663. IEEE, 2007.

[Pie08a] Tobias Pietzsch. Efficient feature parameterisation for visual SLAM using inverse depth bundles. In Mark Everingham, Chris J. Needham, and Roberto Fraile, editors, *Proceedings of the British Machine Vision Conference 2008, Leeds, September 2008*. British Machine Vision Association, 2008. 64

[Pie08b] Tobias Pietzsch. Planar features for visual slam. In *KI*, pages 119–126, 2008. 24, 25

[PJ08] Martin P. Parsley and Simon J. Julier. Avoiding negative depth in inverse depth bearing-only SLAM. In *IROS*, pages 2066–2071. IEEE, 2008. 53, 54

[PT08] Pedro Piniés and J. D. Tardós. Large scale slam building conditionally independent local maps: Application to monocular vision. *IEEE Transactions on Robotics*, 24(no. 5):1094–1106, October 2008. 46

[PTN08] Lina María Paz, Juan D. Tardós, and José Neira. Divide and conquer: EKF SLAM in O(n). *IEEE Transactions on Robotics*, 24(5):1107–1120, 2008. 46

[Rib04] M. Isabel Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties, February 2004. 27

[RPG10] Alejandro Rituerto, L. Puig, and J. J. Guerrero. Comparison of omnidirectional and conventional monocular systems for visual slam. 2010.

[Sca08] Davide Scaramuzza. Omnidirectional vision: from calibration to robot motion estimation, February 2008. 12, 44

[SCMS07] Davide Scaramuzza, Nicolas Criblez, Agostino Martinelli, and Roland Siegwart. Robust feature extraction and matching for omnidirectional images. In Christian Laugier and Roland Siegwart, editors, *FSR*, volume 42 of *Springer Tracts in Advanced Robotics*, pages 71–81. Springer, 2007. 24

[SML05] Joan Solà, A. Monin, and T. Lemaire. Undelayed initialization in bearing only slam. In *IROS*. IEEE, 2005. 44, 51

[SMS06] Davide Scaramuzza, Agostino Martinelli, and Roland Siegwart. A flexible technique for accurate omnidirectional camera calibration and structure from motion. In *ICVS*, page 45, 2006. 12

[Sol10] Joan Solà. Consistency of the monocular EKF-SLAM algorithm for three different landmark parametrizations. In *ICRA*, pages 3513–3518. IEEE, 2010. 61

[SSBB07] Hauke Strasdat, Cyrill Stachniss, Maren Bennewitz, and Wolfram Burgard. Visual bearing-only simultaneous localization and mapping with improved feature matching. In Karsten Berns and Tobias Luksch, editors, *AMS*, Informatik Aktuell, pages 15–21. Springer, 2007. 44, 51

[ST94] Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, Seattle, June 1994. 23

[WB06] Greg Welch and Gary Bishop. An introduction to the kalman filter, 2006. 27