UNIVERSITÀ DI PARMA

UNIVERSITA' DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXXI Ciclo

Applicazioni di visione stereoscopica per sistemi embedded in agenti autonomi.

Coordinatore:

Chiar.mo Prof. Marco Locatelli

Tutor:

Chiar.mo Prof. Massimo Bertozzi

Dottorando: *Domenico Giaquinto*

Anni 2015/2018

a mia madre
"asciuga le tue lacrime e non piangere, se mi ami:
il tuo sorriso è la mia pace"
S. Agostino

Sommario

La capacità di navigare autonomamente, interessa non solo il campo automobilistico ma anche il settore dei droni. Nel campo dei veicoli autonomi due dei principali problemi sono rappresentati dalla localizzazione e dal rilevamento degli ostacoli. Il sistema autonomo, percepisce l'ambiente circostante mediante l'impiego di sensori ed utilizza queste informazioni per determinare la sua posa nel mondo e per riconoscere possibili ostacoli lungo il percorso. Tra i sensori più utilizzati in ambito automotive e droni, vi sono le telecamere: queste sono, generalmente, più economiche di soluzioni come lidar e laser scanner, e hanno il vantaggio di restituire una maggiore qualità di informazione.
Tuttavia l'elaborazione dei dati ottenuti con questo tipo di sensore risulta molto complessa e le soluzioni proposte allo stato dell'arte richiedono hardware specializzato e caratterizzato da consumi energetici elevati. Questo è oltremodo vero se le camere vengono utilizzate in coppia, formando cosi un sistema stereoscopico in grado di generare con opportuni algoritmi mappe di disparità, che al loro interno contengo l'informazione relativa alla posizione tridimensionale per ogni punto del mondo catturato da entrambe le camere.
Questa tesi ha come obiettivo la presentazione di due algoritmi basati su mappe di disparità: un algoritmo di localizzazione basato su keyframe e un rilevatore di ostacoli. In particolare gli approcci sviluppati sono stati realizzati per essere eseguiti su chip caratterizzati da un ridotto consumo energetico, realizzati dal'azienda Ambarella, per la quale è stato svolto il dottorato. L'algoritmo basato sull'approccio di keyframe localization realizzato, utilizza nello specifico una coppia di immagini con una risoluzione di 3840x1728 *px* a una frequenza di 30 frame per secondo e una disparità sparsa per dare supporto ad un filtro di localizzazione e consentire l'atterraggio in determinate basi con un errore inferiore ai 15 *cm*. L'algoritmo di rilevazione degli ostacoli utilizza una mappa di disparità densa con una risoluzione di 3840x1728 *px* per riuscire a individuare oggetti piccoli (dai 7 *cm* ai 40 *cm*) quali ad esempio coni stradali alti 30 *cm*, rilevabili ad una distanza di 92 *m*, e oggetti di grandi dimensioni come pedoni a una distanza di 150 *m*.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Negli ultimi anni è stato esteso anche al mondo dei droni il concetto assodato di *guida autonoma* legato al settore automobilistico, in cui era previsto un veicolo in grado di rilevare l'ambiente e la navigazione senza intervento umano.
Oggigiorno tuttavia è sempre più comune trovare droni in grado di effettuare alcuni azioni in modo autonomo quali ad esempio il decollo, l'atterraggio e non solo. Come già accennato la guida autonoma trova le fondamenta all'interno dell'industria automobilistica che ha contribuito all'innovazione e alla crescita economica per più di un secolo. Nell'ultimo decennio case automobilistiche come BMW, Volkswagen, Audi, Nissan, Toyota, Honda, Ford, e Tesla, insieme a colossi come Google e Apple, hanno iniziato a realizzare i primi prototipi di "veicoli a guida autonoma".
Alcune di queste aziende, e.g. Google e Tesla, hanno già iniziato il collaudo di questi veicoli lungo le strade della California. L'obbiettivo di questa tecnologia è quello di ridurre, fino all'eliminazione, l'interazione dell'essere umano con il veicolo, durante l'utilizzo quotidiano. Il raggiungimento di tale obbiettivo consentirebbe la riduzione degli incidenti stradali causati da distrazione durante la guida. Ad oggi, è possibile trovare un esempio di adozioni di apparati autonomi in campo industriale per la movimentazione di prodotti all'interno di stabilimenti, come descritto in [4], dove, il contesto fortemente strutturato, ne ha favorito la diffusione.
Uno dei requisiti basilare che accomuna questi sistemi è il monitoraggio costante dell'ambiente circostante, in cui l'eliminazione dei punti ciechi gioca un ruolo fondamentale. Per risolvere il problema vengono utilizzati sempre più spesso differenti

sensori come: Radar, LiDAR[1], telecamere e ultrasuoni. In tabella 1 vengono messi a confronto i pro e i contro dei diversi tipi di sensori.

Si notano dalla tabella 1 come i difetti dei vari sensori disponibili rendano necessaria la cooperazione in un sistema autonomo per renderlo robusto ed efficiente. Tuttavia come descritto in [5] l'insieme di più apparati rende complesso la comunicazione, incrementando di conseguenza anche il costo del sistema. L'utilizzo massivo della sensoristica all'interno del campo automotive ha comportato l'esigenza da parte di alcuni organismi, tra cui la SAE International Automotive Engineers, di stabilire attualmente 6 livelli per identificare la guida autonoma dei veicoli. Si tratta di uno standard per verificare il grado di autonomia delle auto. Si parte dal livello 0 in cui si trovano tutti i veicoli privi di sensoristica fino ad arrivare all'ultimo livello dove ricadono gli autoveicoli interamente autonomi. I livelli intermedi vengono classificati in base al grado di automazione del veicolo e all'interazione con la persona che è posta alla guida. Tra i vari sistemi di supporto alla guida ci sono gli **ADAS**[2] che si differenziano in base al tipo di assistenza alla guida che offrono.
Tra i compiti in grado di essere svolti dagli ADAS ci sono:

- **ACC** - Adaptive Cruise Control - Mantenimento della velocità di crociera e della distanza.
- **AEB** - Autonomous Emergency Braking - Frenata automatica di emergenza.
- **ALC** - Adaptive Light Control - Controllo attivo delle luci.
- **APS** - Automatic Parking System.
- **CMS** - Camera Monitor System - Monitoraggio con videocamere.
- **CTA** - Cross Traffic Alert - Monitoraggio del traffico agli incroci.
- **DAS** - Driving Attention Assist - Monitoraggio dell'attenzione del conducente.
- **EDA** - Emergency Driver Assistant - Assistente di emergenza per il guidatore.

[1]LiDAR: Laser Imaging Detection and Ranging.
[2]ADAS - Advanced Driver Assistance System

Sensore	Pro	Contro
Radar	• Oggetti in movimento. • Veloce e accurato nella misura. • Adatto per ostacoli vicini e lontani.	• Adatto solo per il settore automotive e non per il campo dei droni. • Non adatto per individuare oggetti di piccole dimensioni. • Non adatto per oggetti fermi. • Sensore di tipo attivo. • Sensibile al fumo.
Lidar/Laser	• Veloce e accurato nella misura. • Precisione invariante alla distanza.	• Adatto solo per il settore automotive e non per il campo dei droni. • Parti meccaniche in movimento. • Sensore di tipo attivo. • Sensibile al fumo. • Alto costo.
Sonar	• Rilevamento precisa per ostacoli vicini. • Ampio FoV. • Basso costo.	• Distorsione dovuta alle riflessioni sull'asfalto. • Misura monodimensionale. • Sensore di tipo attivo. • Sensibile al fumo. • Echo. • Lenti.
Visione	• Mostrano le immagini della realtà. • Grande quantità di informazioni. • Flessibilità. • Sensore di tipo passivo. • Basso consumo. • Basso costo.	• Grande mole di dati da processare, richiedono hardware specifico. • Sensibile agli agenti atmosferici. • Per ottenere l'informazione tridimensionale del mondo le camere devono essere utilizzate in coppia formando un sistema stereoscopico.

Tabella 1: Pro e contro di alcuni sensori disponibili sul mercato.

- **HDC** - Hill Descent Control - Assistente alla discesa.
- **LCA** - Lane Change Assist - Assistente al cambio di corsia.
- **LKA** - Lane Keeping Assist - Assistente al mantenimento di corsia.
- **NVA** - Night View Assit - Assistente alla visione notturna.
- **RTA** - Rear Traffic Alert - Allarme per il traffico posteriore.
- **TJA** - Traffic Jam Assistant - Assistente alla guida in coda.
- **TSR** - Traffic Sign Recognition - Riconoscimento dei segnali stradali.
- **Blind Spot Monitor** - Monitoraggio dell'angolo cieco.

Le telecamere trovano un'applicazione massiva anche nel campo dei droni. L'utilizzo degli UAV[3] è vario, sia in ambienti chiusi che all'aperto, dall'applicazione in campo militare, alla videosorveglianza, fino ad arrivare all'agricoltura e all'esplorazione di ambienti ostili.
Anche nel settore degli aeromobili come per il campo automobilistico l'impiego di sistemi per l'aiuto alla guida fino alla stessa guida autonoma sono di grande importanza, poiché aiutano nell'eseguire le operazione che richiedono un costante stato di allerta da parte del guidatore. Uno dei vantaggi dell'utilizzo di telecamere è rappresentato dal fatto che, oltre a permettere di elaborare l'apparenza del mondo circostante, se usate in coppia permettono di ricostruire tridimensionalmente il mondo stesso.
Nel campo della visione artificiale la ricostruzione 3D dell'ambiente circostante attraverso la visione stereoscopica è un argomento ampiamente studiato in letteratura [si veda [6], [7] [8] [9]]. I vari algoritmi in letteratura offrono diversi compromessi in termini di complessità computazionale, qualità della ricostruzione e robustezza al rumore delle immagini in input. Nel corso degli anni sono stati studiati diversi algoritmi che permettono di ottenere la mappa di disparità partendo dalle immagini catturate. Come descritto in [10], [11], alcuni algoritmi sono stati portati su chip e

[3]UAV: Unmanned Aerial Vehicle

sono diventati dei prodotti commercializzati come ad esempio il 3DV fig. I.1 descritto in [12], prodotto dall'azienda VisLab[4].

Figura I.1: 3DV-E realizzato da Vislab.

Questo sistema embedded che permette di ottenere in *real-time* la nuvola di punti della scena non è il solo commercializzato, ci sono infatti altri prodotti simili che consentono di lavorare sulla disparità come:

- **Kinect** - conosciuto con il nome di Project Natal, è un accessorio sviluppato da Microsoft per la console Xbox 360, sensibile al movimento in particolare del corpo umano, la produzione è stata cessata da fine 2017.
  Il sistema era dotato di una camera RGB[5] e di una telecamera a radiazione infrarossa. Per il calcolo della lontananza degli oggetti, invece si avvaleva di uno scanner 3D a luce strutturata, in combinazione con gli infrarossi. Nelle diverse versione sviluppate sono state migliorate la risoluzione adottata e aumentata le capacità di lettura del movimento.

- **ZEED** - sfrutta la visione stereoscopica ed è dotato di un sistema che legge le immagini in alta definizioni su due canali sincronizzati in formato side-by-side e le rende disponibili su una USB 3.0. Per il calcolo della mappa di disparità utilizza un'architettura di calcolo parallelo CUDA[6] sviluppato da NVIDIA[7].

[4] Vislab: an Ambarella Inc. company
[5]RGB - è un modello di colori le cui specifiche sono state descritte nel 1936 dalla CIE
[6]CUDA -Compute Unified Device Architecture.
[7]NVIDIA - www.nvidia.com

- **Bumblebee** - a differenza delle precedente ne esistono diversi modelli tra cui la Bumblebee XB3 costituita da tre camere che permettono una ricostruzione 3D più dettagliata sfruttando la multi-baseline.

Oltre a questi prodotti embedded che restituiscono come unico dato la nuvola di punti, vi sono in particolare nel campo automotive sistemi come Nvidia Drive PX che sono una serie di computer studiati e realizzati per eseguire algoritmi come l'assistenza alla guida fino ad eseguire algoritmi per la guida autonoma. Questo tipo di sistema è realizzato dalla combinazione di una o più schede Nvidia TegraX2 in base alla potenza di calcolo richiesta dagli algoritmi. Queste configurazioni vengono montate su alcuni modelli di veicolo come la Tesla[8], infine come dichiarato da Nvidia il consumo di energia può variare da 10W fino ad arrivare a 500W in base alla complessità del sistema richiesto.

La ricostruzione dell'ambiente circostante è di grande importanza per l'esecuzione di due aspetti fondamentali di un agente autonomo: localizzazione e la rilevazione degli ostacoli. Per questo motivo durante il mio dottorato mi sono occupato di questi due aspetti, sviluppando in particolare due algoritmi:

- Keyframe localization - un'applicazione di supporto alla localizzazione senza GPS per droni in grado di correggere l'errore di deriva di cui è in genere affetta.
- Obstacle detector - un algoritmo per l'individuazione di ostacoli di piccole dimensioni e di oggetti lontani davanti al veicolo.

Gli approcci sviluppati al contrario di quelli presenti nello stato dell'arte possono essere eseguiti su chip a basso consumo, sfruttando la nuvola di punti 3D in maniera densa o sparsa. Il porting è stato fatto per il CV1[9] sviluppato da Ambarella[10], azienda nella quale ho realizzato il dottorato. Questo chip al contrario di quelli presenti in commercio permette di acquisire le immagine stereo, calcolare la disparità ed eseguire

[8] Tesla, Inc. - www.tesla.com

[9] CV1 - 4K Computer Vision Processor

[10] Ambarella - azienda esperta nella produzione di chip low power utilizzati nell'elaborazione e compressione video.

gli algoritmi sviluppati. Per ogni applicazione verrà riservata una sezione che descrive l'ambiente di sviluppo, l'algoritmo, il porting e i test. Infine dato l'uso sempre più ampio di coppie stereo fisheye verrà riservata una sezione su come riadattare ogni algoritmo per essere esteso al modello sferico.

Capitolo 1

Stato dell'Arte

Come descritto nel capitolo precedente, l'utilizzo di una coppia di camere stereoscopiche è di grande importanza per due dei più importanti ambiti della guida autonoma: il rilevamento della posizione attuale del veicolo, assieme ai suoi dati di velocità e assetto e la percezione degli ostacoli circostanti. In questo capitolo verrà analizzato lo stato dell'arte dei due ambienti separatamente e i tipi di sistemi *embeded* disponibili per l'elaborazione dei dati.

1.1 Localizzazione

Per localizzazione si intende la stima di posa e orientamento di un robot, rispetto a un sistema di riferimento solidale con l'ambiente che circonda il robot. Questa stima avviene come descritto in [13], utilizzando misure provenienti da sensori eterogenei, come odometri ed unità inerziali, unitamente a tecniche di ricostruzione del moto.
In letteratura, come proposto da Thrun et al. [14] [15], sono descritte diverse tecniche che permettono, dato il rumore da cui sono afflitte le misure dei sensori, l'utilizzo di strumenti matematici basati sulla teoria della probabilità e dalla teoria del controllo per ottenere misure affidabili. Le tecniche appena citate consentono di mitigare due dei principali problemi di questo tipo di sistemi:

- Filtraggio del rumore.

- Drift, cioè l'aumento dell'errore della misura di localizzazione dovuta al continuo accumulo di misure relative.

Per attenuare questi problemi in letteratura vengono proposti diversi approcci per la fusione dei dati [13] [15], che, come dice il termine, permettono di fondere insieme dati provenienti da sorgenti eterogenee. I dati utilizzati in questo contesto possono fornire informazioni di localizzazione relative, cioè riferite ad un precedente istante temporale, o informazioni di localizzazione globali, cioè globalmente valide nel sistema di riferimento in questione indipendentemente dalla sequenza di movimento che ha portato il drone in quel punto. In letteratura vengono proposte diverse soluzioni per la localizzazione a seguito del tipo di contesto in cui il robot deve compiere le missioni, ai sensori e alla potenza computazionale di cui è dotato, etc. É quindi possibile distinguere tra le seguenti tecniche di localizzazione:

- **Localizzazione relativa** - la posizione iniziale è conosciuta a priori, e la localizzazione si occupa di mantenere traccia della posizione durante gli spostamenti nell'ambiente sulla base di misure relative del moto del robot.

- **Localizzazione assoluta** - la posizione e l'orientamento del robot sono derivate misurando la posizione di alcuni *"landmark"*, ossia di alcune feature solidali al sistema di riferimento in cui si intende esprimere la posizione del robot stesso, in questo tipo di approccio è richiesta la conoscenza della mappa a priori.

- **SLAM** (Simultaneous Localization and Mapping) - viene stimata nello stesso momento la posizione del robot e la mappa dell'ambiente in cui il robot si trova. Questo processo è normalmente difficoltoso poiché per la stima della posizione si ha bisogno di una buona mappa e per la stima di una mappa si ha bisogno di un buon posizionamento. Inoltre queste tecniche sono computazionalmente pesanti e la mappa tende a essere molto pesante, per questi motivi, lo SLAM non è normalmente implementabile su chip low power.

1.1.1 Localizzazione relativa

I sistemi di localizzazione relativa sono i più utilizzati e diffusi. Questi sistemi basano la stima della posizione sulla base delle misure relative del moto del robot. Poiché ogni misura si basa sulle precedenti, come mostrato in fig. 1.1 nei sistemi di localizzazione relativa gli errori crescono con il tempo durante la fase di movimento del robot. Per ovviare al problema viene utilizzata la conoscenza a priori del modello cinematico della meccanica del robot, attraverso approcci probabilistici di correzione e previsione dello stato del sistema; un esempio è il filtro di Kalman come descritto in [16]. Tuttavia data la semplicità questi approcci sono preferibili ad altri più complessi e accurati, spesso si trovano accoppiati a sistemi di localizzazione assoluta.
Le posizioni relative del robot possono essere acquisite generalmente tramite odometri, sia meccanica che visuale, e navigazione inerziale o come descritto in [17].

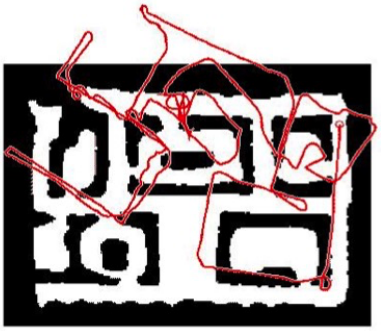


Figura 1.1: La figura mostra l'errore commesso dal robot durante la navigazione, sovrapponendo il percorso effettuato (rosso) con la mappa.

1.1.2 Localizzazione Assoluta

Questo approccio sfrutta la conoscenza a priori dell'ambiente ed ha quindi a disposizione una mappa.
A differenza degli algoritmi citati precedentemente in questo metodo i dati ottenuti dalle informazioni odometriche e inerziali sono integrate con un sistema di percezione che viene montato sul robot, fornendo informazioni che possono essere combinate con tecniche di filtraggio come il filtro di Kalman e le sue estensioni.
All'interno della mappa conosciuta dal robot, oltre alle informazioni dimensionali dell'ambiente vi sono raccolte una serie di informazioni sulla posizione dei landmarks rispetto a un sistema di riferimento noto a priori.
I landmark rappresentano dei punti distinti dell'ambiente circostante con particolari caratteristiche di riconoscibilità, essi possono essere percepiti da sistemi diversi come laser, ultrasuoni o sistemi di visione artificiale.
In letteratura questo tipo di approccio alla localizzazione viene utilizzato sia per situazioni in cui è noto lo stato iniziale (orientamento posizione, velocità) sia in altri tipi di problemi:

- **Lost robot problem** - sono le situazioni in cui il sistema perde la localizzazione e deve riacquistarla, attraverso la fusione dei dati provenienti dai sensori e le informazioni contenute nella mappa.

- **Kidnapped robot problem** - ossia situazioni in cui un robot autonomo viene trasportato in una posizione arbitraria senza che il sistema di controllo possa intuire questo movimento. Queste prove in cui il robot viene "rapito" sono comunemente usate per testare l'abilità del sistema di localizzazione dopo un possibile guasto critico.

1.1.3 SLAM

Tra le tecniche più studiate in letteratura troviamo lo SLAM[1], questo approccio prevede due importanti fasi che vengono effettuate simultaneamente ovvero:

- Creazione della mappa.
- Localizzazione.

Grazie a questa strategia è possibile esplorare ambienti sconosciuti tenendo traccia degli spostamenti del robot rispetto a un sistema di riferimento ad ogni nuova attivazione del sistema.
Al contrario degli algoritmi precedenti in cui al robot veniva data una mappa dettagliata dello spazio circostante, nello SLAM, data la totale assenza a priori di informazioni, è lo stesso robot, che esplorando l'ambiente, genera una mappa dello spazio in cui deve operare. Il robot deve quindi esplorare l'ambiente e trovare dei landmark per potersi localizzare rispetto ad essi. Con questa tipologia di algoritmi il sistema di riferimento può essere espresso in modo egocentrico, ovvero il sistema di riferimento è solidale al robot, oppure relativo, in cui la posa è riferita rispetto alla posizione iniziale del sistema.
A questo punto, la stima della posizione avviene utilizzando tecniche di fusione dei dati provenienti dai sensori e dei dati odometrici raccolti. Tra i dati raccolti dai sensori ci sono le informazioni ottenute durante l'esplorazione dell'ambiente che prevedono il riconoscimento di feature ovvero caratteristiche ambientali o landmark artificiali che sono stati preventivamente inseriti, la cui posizione esatta nell'ambiente è sconosciuta al sistema durante la navigazione.
Nello SLAM, lo stato del sistema non è descritto solo dalla posa del robot, ma anche dalla mappa. In base alle tecniche scelte, la posizione può essere costituita da un insieme di feature ambientali che si sommano ogni volta che nuovi punti di riferimento vengono aggiunti in mappa e corretta grazie alle osservazioni multiple dei landmark presenti in memoria: questo processo è chiamato *"state augmentation"*.
Esistono diverse tecniche di SLAM poiché i dati di input provengono da informazioni

[1]SLAM - simultaneous localization and mapping

relative e sono affette da incertezza. Infatti la posa viene ricostruita partendo dal punto di partenza del sistema, concatenando un certo numero di osservazioni durante il compimento della missione del robot.
Lo SLAM si basa su un'intuizione secondo la quale al contrario dell'errore accumulato dall'informazione spaziale ad ogni iterazione, le relazioni tra landmark derivate dalle singole osservazioni locali tramite dei sensori, non sono correlate tra loro. Pur essendo i sensori utilizzati affetti da un errore e quindi intrinsecamente imprecisi, questa imprecisione è indipendente dalla posizione con cui il robot valuta i landmark. Quindi date due osservazioni provenienti da un sistema di visione che inquadrano lo stesso landmark da punti di vista differenti, le caratteristiche ambientali non potranno cambiare le misure relative all'interno di ciascuna osservazione, nota la relazione spaziale tra il landmark e la camera utilizzata, seppure i sensori utilizzati siano affetti da rumore. Infatti le informazioni di posizione relativa ottenute dalle osservazioni del landmark in entrambe le immagini risulteranno comunque imprecise data la natura del sensore utilizzato, ma non avranno una deriva nel tempo come i dati odometrici durante il compimento della missione del robot, infatti come descritto in [18], si può parlare di "certezza nelle relazioni nonostante l'incertezza sulla misura". La data associazione, ovvero il riconoscimento e l'associazione di caratteristiche dell'ambiente anche in posizioni e situazioni diverse, è un argomento ampiamente descritto in letteratura. Da notare che, dipendentemente dall'approccio utilizzato, i landmark possono essere usati direttamente come punto di riferimento globali, oppure possono essere accumulati unitamente alle misure relative e ottimizzate durante il processo di localizzazione.
Questo è possibile tramite il riconoscimento di anelli, ovvero riconoscendo di trovarsi in zone della mappa già esplorate in precedenza attraverso l'utilizzo dei landmark salvati.Ciò consente di diminuire l'incertezza accumulata o l'errore complessivo.
La letteratura divide questi approcci in due categorie in base allo scopo, alla modalità di soluzione e alle possibili applicazioni.

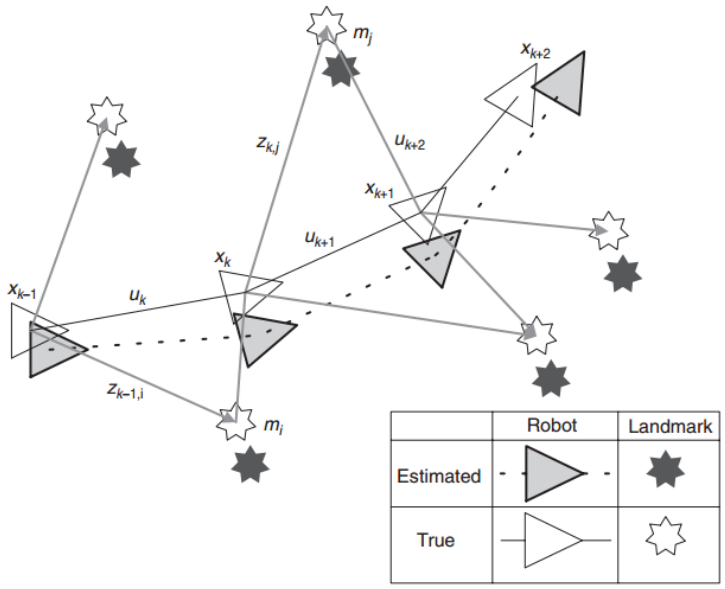


Figura 1.2: Mostra il problema dello SLAM nel quale il robot cerca di stimare contemporaneamente sia la propria posa che quella dei landmark nella mappa. Le posizioni reali (in bianco) non sono mai osservate direttamente, nè conosciute a priori. L'unica informazione a disposizione è la stima delle posizioni relative tra robot e landmark, derivante dalle osservazioni compiute attraverso i sensori.

1.1.3.1 Online SLAM

Questa categoria comprende tutti i metodi di localizzazione e mappatura che attraverso il filtraggio dei dati in *real time* direttamente a bordo del sistema, cercano di stimare la posa attuale del robot e a ogni nuovo landmark tentano di completare la mappa dell'ambiente in cui si compie la missione. Se non gestito in modo adeguato, questo aggiornamento della mappa può portare tante volte ad un aumento critico delle dimensioni occupate in memoria.
In termini probabilistici, questo corrisponde alla stima *bayesiana* della probabilità a posteriori p di trovarsi all'istante T in una determinata posa x_T e di considerare valida una specifica mappa m, data la conoscenza di tutte le misure odometriche derivanti da tutti i comandi dati agli attuatori $u_{1:t}$ e di tutte le osservazioni sensoriali $z_{1:t}$ raccolte durante la missione.

$$p(x_{\mathrm{T}}, m | u_{1:\mathrm{T}}, z_{1:\mathrm{T}}) \tag{1.1}$$

Con il filtraggio, il problema di SLAM viene visto come la stima dello stato di un sistema dinamico, dove lo stato è composto dall'ultima posa e dalla mappa, costituita dall'elenco delle posizioni dei landmark accumulati nell'ambiente esplorato. In questo approccio, la mappa ha come solo obbiettivo di correggere i dati di posizione ricavati dall'odometria e non fa parte dei dati prodotti in output dalla soluzione del problema. Il sistema può essere ulteriormente migliorato se ad ogni iterazione durante la raccolta di nuove osservazioni si cercano eventuali anelli ossia zone già mappate in precedenza.
Nello SLAM online si utilizzano spesso il filtro di Kalman e la sua versione estesa (EKF-SLAM) in cui si considera il rumore presente sulle misure ottenute (sia odometriche che sensoriale) come rumore gaussiano. Approcci più complessi prevedono altri tipi di filtri, come nel FastSLAM in cui si utilizza un filtro particellare Rao-Blackwellized per modellare il rumore visto in maniera non-gaussiana. Infine in letteratura ci sono anche altre tecniche di SLAM che si basano su *informaton filter*.

1.1.3.2 Offline SLAM

L'approccio Offline SLAM è anche detto full SLAM, attraverso queste tecniche si cerca di ricostruire a posteriori sia la mappa dell'ambiente, sia l'intera traiettoria compiuta dal robot, a differenza dello SLAM Online il cui unico obbiettivo era la localizzazione del robot durante la missione. L'ottimizzazione della mappa a posteriori consente una migliore localizzazione nelle missioni future. Questo processo avviene attraverso inferenze *bayesiane* che utilizzano tutte le pose del robot raccolte durante il funzionamento.
Un modo per affrontare il full SLAM è attraverso l'utilizzo della formulazione basata sulla teoria matematica dei grafi chiamata anche Graph SLAM. Questa tecnica consente di risolvere il problema di *mapping* attraverso l'utilizzo di grafi, i cui nodi presenti nella mappa possono rappresentare i landmark inseriti o le pose che il robot assume; mentre gli archi possono essere di tipi differenti:

- **archi di movimento** - rappresentano la distanza tra due cose del robot.
- **archi di misurazione** - rappresentano l'osservazione in una determinata posa di un landmark.

L'utilizzo dei metodi del Graph SLAM è possibile poiché la distribuzione delle osservazioni e delle pose formano un grafo sparso. Con l'utilizzo di questi metodi risulta invariata la struttura topologica del grafo e quindi le relative informazioni metriche. Un modo per modellare lo SLAM con i landmark è attraverso l'utilizzo di un *factor graph*, ovvero: un grafico bipartito che rappresenta la fattorizzazione di una funzione in sottoinsiemi ridotti, o fattori. Come mostrato in 1.2 attraverso un prodotto di singoli fattori è possibile esprimere la distribuzione di probabilità congiunta.

$$p(x_t, M|u_t, z_t) = \prod_i p(x_i|u_i, x_{i-1}) \prod_{k,i,j} p(l_k|x_i, z_i) \tag{1.2}$$

dove l_k è la posizione dei landmark che costituiscono la mappa M. Si ha quindi una prima fase in cui la mappa viene costruita attraverso un grafo, il quale viene arricchito inserendo successivamente i nodi e gli archi che corrispondono ai vincoli e alle variabili. Questo metodo però è notevolmente sensibile agli errori di associazione, infatti

la costruzione di un grafo errato può essere provocato da una chiusura erronea di un loop, che presenta quindi vincoli errati. Gli errori che portano a un'errata topologia del grafo portano a non rispecchiare più il reale ambiente in cui il robot ha compiuto la missione.
Per questo è importante inserire una ulteriore fase di ottimizzazione. In questa fase si cerca di massimizzare la funzione (1.2) relativa alle pose dei landmark inseriti nella mappa, e correggere i valori dei vincoli. In questo processo di ottimizzazione può essere modificata se necessario la struttura del grafo eliminando eventuali inconsistenze dovute a osservazioni in contraddizione. Tuttavia l'alterazione della struttura topologica del grafo durante questa fase non può essere radicale o casuale. L'ottimizzazione consiste nella ricerca della configurazione ottima delle pose, dei landmark e nel trovare la topologia ottima priva di vincoli non validi.

1.1.4 Feature

Come descritto in [19], l'individuazione di punti caratteristici ovvero di feature, la loro caratterizzazione (feature descriptor) ed infine il *matching* sono tematiche importanti nella *computer vision*.
Il concetto alla base delle feature è che solo alcuni punti che costituiscono l'immagine hanno una probabilità alta di essere individuati senza ambiguità durante un confronto. Le features all'interno di un immagine possono essere identificate attraverso uno specifico pattern che è unico rispetto ai pixel immediatamente vicini, a cui sono associate una o più proprietà come descritto in [20] [3]. Queste proprietà possono essere diverse, bordi, angoli, regioni, etc. Infine i punti successivamente alla loro individuazione vengono convertiti in descrittori numerici unici e compatti.
Poiché, questi algoritmi sono in genere onerosi dal punto di vista computazionale, esistono molti metodi per estrarre le feature, che differiscono in base ai dati da elaborare e al tipo di applicazione da sviluppare. Tuttavia, idealmente gli estrattori di feature si basano su delle importati proprietà:

- **Caratteristiche distintive** - i modelli di pattern utilizzati dovrebbero essere ricchi di variazioni che possono essere utilizzate per distinguere e confrontare

le feature.

- **Località** - le caratteristiche estratte dovrebbero essere locali ovvero in grado di ridurre la possibilità di essere occluse tra due viste consecutive.
- **Quantità** - il numero di feature estratto dovrebbe essere sufficientemente grande da riuscire a descrivere il contenuto dei frame in una forma compatta.
- **Precisione** - le feature devono essere rilevate in modo accurato.
- **Efficienza** - devono essere identificate velocemente in modo da essere adatti ad applicazioni real-time.
- **Ripetibilità** - dati due fotogrammi dello stesso oggetto con punti di vista differenti, le feature estratte devono essere il più possibile sovrapposte.
- **Invarianza** - in presenza di una grande deformazione della scena dati due frame (rotazione, scala, etc.), l'algoritmo di estrazione dovrebbe modellare matematicamente queste deformazioni il più precisamente possibile.
- **Robustezza** - l'algoritmo deve essere il più possibile robusto rispetto ad immagini che presentano sfocature, artefatti, etc.

Dato il grande numero di esigenze richieste dalle applicazioni, in letteratura esistono molti *survey* come [20] [3], che descrivono le performance dei vari algoritmi (come mostrato in tabella 1.1) di *feature extraction*.

Algoritmi	**Invarianza**			**Qualità**			
	Rotazione	**Scala**	**Affine**	**Ripetibilità**	**Localizzazione**	**Robustezza**	**Efficienza**
Harris	✓	-	-	+++	+++	+++	++
Hessian	✓	-	-	++	++	++	+
Susan	✓	-	-	++	++	++	+++
Harris-Laplace	✓	✓	-	+++	+++	++	+
Hessian-Laplace	✓	✓	-	+++	+++	+++	+
DoG	✓	✓	-	++	++	++	++
Salient Regions	✓	✓	✓	+	+	++	+
SURF	✓	✓	-	++	+++	++	+++
SIFT	✓	✓	-	++	+++	+++	++
MSER	✓	✓	✓	+++	+++	++	+++

Tabella 1.1: Mostra le performance degli algoritmi di estrazione di feature descritte in [3].

In generale gli algoritmi come mostrato in [3] differiscono per le caratteristiche estratte, tra le quali ci sono:

- **Bordi** - si riferiscono a uno specifico pattern in cui si identificano i pixel che cambiano bruscamente intensità.
- **Angoli** - ovvero quei punti in cui uno o più spigoli si intersecano.
- **Regioni** - ossia l'insieme di pixel connessi attraverso un criterio di omogeneità.

Si può notare come esista una forte relazione tra queste caratteristiche locali, ad esempio, più spigoli a volte circondano una regione, una regione è delineata da dei bordi che a loro volta definiscono degli angoli [21].

1.1.5 Localizzazione con feature per Droni

I velivoli a pilotaggio remoto (UAVs [2]) stanno diventando sempre più popolari grazie al loro basso costo e ai loro molteplici utilizzi. Ad oggi sono usati per soddisfare vari compiti come già detto nel capitolo precedente, questi includono la sorveglianza, fotografia aerea, ispezione, ricerca e salvataggio, etc.
In molti di questi casi però la navigazione da remoto anche per piloti esperti è difficile, come ad esempio in edifici collassati o siti di grandi dimensioni. In queste situazioni gli UAV autonomi con elaborazione a bordo possono essere una valida soluzione, usando questa tecnologia gli operatori devono solo inizializzare la missione e raccogliere i dati memorizzati dal drone durante la missione. Oltre al riconoscimento degli ostacoli, la localizzazione gioca un ruolo fondamentale sopratutto in ambienti privi di GPS.
Solo di recente l'esplorazione autonoma da parte di droni ha attirato l'attenzione nella letteratura come descritto in [22] [23] [24]. La maggior parte degli approcci allo stato dell'arte sono basati su SLAM o tecniche di *Visual Odometry*.
In [25] viene utilizzato un algoritmo SLAM con telecamera basato su keyframe e FAST feature, il quale riesce a stabilizzare i 6 gradi di libertà (6DOF [3]) del drone, permettendo di correggere la deriva ed eliminando la dipendenza del GPS nella localizzazione. Sempre in letteratura Von Stumberg et al. [26] hanno sviluppato un sistema per l'esplorazione autonoma con droni basato su LSD-SLAM [27], questo risulta uno degli attuali algoritmi SLAM allo stato dell'arte. In [28] viene utilizzato un approccio di visual odometry basato su visione stereo e funzionalità SIFT [29] per stimare la posizione del velivolo. Tutti gli algoritmi studiati in letteratura presentano però delle limitazioni, poiché molto spesso come in [26] l'elaborazione viene effettuata da un pc remoto in quanto gli algoritmi risultano molto complessi e per garantire una esecuzione real time l'hardware risulta inadeguato. Altri come Nuske et al. [24] utilizzano dei laser scanner 3D che ruotando danno una ricostruzione dell'ambiente, questi tipi di sensori oltre ad essere molto costosi, risultano sensibili agli urti e infine richiedono molta energia per funzionare, caratteristiche che non possono essere sottovalutare al fine di potere commercializzare dei droni.

[2] UAVs -Unmanned Aerial Vehicles
[3] 6DOF - Six Degrees Of Freedom

L'approccio che verrà presentato di seguito e descritto in Giaquinto et al. [30], utilizza per la localizzazione entrambe le tecniche viste in letteratura, fondendo le informazioni provenienti da due coppie stereo poste davanti e sotto il drone. Il sistema utilizza per la localizzazione tre moduli asincroni: visual odometry, keyframe detector (presentato nel capitolo 3) e infine un modulo chiamato *Visual Height Estimation* che attraverso la disparità ottenuta dalla camera posta sotto il drone restituisce la quota a cui si trova il drone. Tutti questi moduli, in aggiunta ad altri tra cui: *Obstacle Detection* e *Path Planner*, sono stati eseguiti in real time su un DJI Matrice100 con a bordo un chip chiamato CV1 sviluppato dall'azienda per cui ho svolto il dottorato.

1.1.6 Matching and place recognition

Uno dei requisiti più importanti è il riconoscimento di una particolare zona dell'ambiente circostante al robot già esplorata in precedenza.
Come descritto in [31] il riconoscimento dei luoghi ha ottenuto nella comunità robotica una grande attenzione dati i grandi risultati ottenuti e discussi in [32] [33]. Un esempio è il sistema FAB-MAP [34] che è in grado di identificare dei loop con una telecamera omnidirezionale. Questo tipo di algoritmo rappresenta le immagini attraverso delle parole, successivamente viene generato *offline* un albero di Chow-Liu [35] per apprendere la probabilità di condividere delle parole. Allo stato dell'arte l'approccio FAB-MAP rappresenta uno standard per quanto riguarda l'individuazione di loop, ma la sua robustezza cala quando le immagini risultano molto simili per un periodo prolungato, ad esempio quando si usano delle camere frontali [33]. Nel lavoro di Angeli et al. [32] si trova un altro approccio in cui vengono creati due vocabolari visivi (aspetto e colore) online in modo incrementale. Le due rappresentazioni chiamate *"bag-of-word"* sono usate insieme come input di un filtro Bayesiano, che stima la probabilità di corrispondenza tra due immagini tenendo conto della probabilità di corrispondenza dei casi precedenti. In contrasto con questi approcci in [31] si basano su un controllo di coerenza temporale per considerare le corrispondenze precedenti e migliorare l'affidabilità dei rilevamenti. Questo tipo di approccio differisce dai precedenti poiché sono i primi a utilizzare dei dizionari di parole binarie, oltre ad avere sviluppato delle tecniche per evitare che le immagini nel tempo raffigurino sempre lo

stesso luogo.
Con il termine *"loop-closing"* si intende il problema nel riconoscere posti visitati precedentemente. Tipicamente questo problema viene affrontato tramite dei controlli geometrici, applicando un vincolo epipolare sul miglior candidato come descritto in [33], in altri casi come in [36] viene utilizzata la visual odometry attraverso una coppia di camere stereo per creare in real time la mappa dell'ambiente circostante, rilevando i loop attraverso un approccio *"bag of word"*. Il loro controllo geometrico consiste nel calcolare una trasformazione spaziale tra le immagini, tuttavia, non tengono traccia dei match precedenti, portando ad applicare il controllo geometrico a diversi candidati.
Nella maggior parte dei lavori in letteratura per il controllo di *loop-closure* le feature utilizzate sono SIFT [29] o SURF [37], poiché sono invarianti sia alla scala che alla rotazione, e mostrano un buon comportamento in vista di cambiamenti di prospettiva. Alcuni lavori tra cui [36] offrono soluzioni per ridurre i tempi computazionali, calcolando *offline* la similarità tra *patch* di immagini differenti e riducendo il vettore dei descrittori attraverso orto-proiezioni casuali, generando cosi descrittori molto veloci utilizzabili in applicazioni real-time. Sempre in [31] vengono utilizzati altri descrittori tra cui BRIEF o ORB [38], che oltre ad essere molto veloci occupano poco spazio in memoria. L'approccio [31] come verrà descritto nel paragrafo 3.3 è stato testato per verificarne i vantaggi nella ri-localizzazione.

1.2 Rilevamento degli ostacoli

Qualsiasi entità in movimento richiede la conoscenza di ciò che lo circonda per evitare collisioni. La determinazione delle aree in cui ci si può muovere senza causare incidenti è conosciuto come rilevamento degli ostacoli.
Il rilevamento degli ostacoli può essere considerato come la classificazione dello spazio che circonda un sistema in movimento in:

- libero - si indicano delle aree in cui il sistema potrebbe muoversi liberamente.
- non libero - si considerano delle aree parte di un ostacolo.

Nel contesto automotive in letteratura vengono proposti diversi sensori per descrivere l'ambiente 3D circostante al veicolo, tra cui il più utilizzato è il LIDAR. Questo tipo di sensore risulta però molto costoso e difficilmente integrabile all'interno di un veicolo, basti pensare all'aspetto dell'automobile progettata da Google (fig. 1.3), infine come descritto in [39] rispetto alle coppie stereo, la nuvola di punti ottenuta ha una risoluzione nettamente inferiore.

Figura 1.3: Veicolo autonomo di Google.).

In letteratura vengono descritti diversi approcci per il rilevamento degli ostacoli attraverso la visione artificiale. Un possibile approccio per calcolare la distanza degli ostacoli contenuti in una scena è attraverso l'uso di sistemi monoculari. Questo tipo

di approccio richiede però un modo per calcolare il fattore di scala che permette di proiettare gli ostacoli nel mondo reale ovvero conoscere le dimensioni dei possibili ostacoli. Per raggiungere questo obbiettivo, alcuni approcci utilizzano informazioni provenienti da un sensore attivo come gli ultrasuoni [40], oppure fanno assunzioni sulle camere [41], sul mondo [42] o addirittura usano i dati odometrici del veicolo per trovare la scala.
Per la guida autonoma, ricostruire l'ambiente circostante al veicolo per identificare gli ostacoli in modo preciso è fondamentale. Per la ricostruzione tuttavia la fusione con altri sensori come l'odometro molto spesso non garantisce la possibilità di ottenere la scala, siccome a velocità ridotte il sensore risulta molto rumoroso e anche l'utilizzo di altre tecniche come ad esempio lo *structure for motion* (SFM) che permette di ricostruire la nuvola di punti 3D partendo da camere mono non funziona in casi in cui il veicolo è fermo, ad esempio in fase di parcheggio. Tutto questo limita le applicazione che si avvalgono della visione monoculare. Nel capitolo 2 verrà mostrato come è possibile calcolare la mappa di disparità, che insieme ad una triangolazione può essere utilizzata per stimare la distanza degli oggetti nell'immagine. La triangolazione dei punti che formano la disparità permette di ottenere *point cloud* 3D molto dense, queste nuvole sono simili a quelle ottenute dai classici LIDAR o approcci basati sui laser. Questa analogia permette di riutilizzare un ampio set di algoritmi sviluppati per rilevare ostacoli con i LIDAR. Tuttavia la nuvola di punti densa sebbene descriva in maniera dettagliata l'ambiente circostante, comporta una grande quantità di informazioni che si traduce in elaborazioni dispendiose in termini di risorse hardware.
Per ridurre questo fabbisogno di risorse in letteratura sono state proposte diverse alternative per rappresentare l'ambiente circostante e individuare dunque gli ostacoli, tra cui:

- Griglie occupazionali (occupation map).
- Mappe altimetriche (elevation map).
- Stixel.

Poichè la triangolazione viene eseguita prima del rilevamento degli ostacoli, qualsiasi errore in questa fase si ripercuote sulla posizione degli ostacoli nel mondo 3D. Per

evitare questa propagazione, alcuni metodi tra cui l'approccio sviluppato e presentato nel capitolo 4, lavorano direttamente sull'immagine di disparità. Questi algoritmi eseguono la triangolazione solo dopo che i pixel sono stati identificati come possibili ostacoli. Gli approcci che verranno presentati di seguito al contrario di quello sviluppato hanno bisogno per il funzionamento di pc con CPU preformanti o addirittura come descritto dagli stessi autori da GPU, l'approccio presentato prossimamente è stato pensato per girare su un sistema integrato che consuma circa 5 watt ed utilizza una disparità densa.

1.2.1 Mappa di Disparità

Data una coppia di telecamere stereo poste alla stessa altezza dal suolo, e scelto un punto su un oggetto 3D, la sua proiezione nelle due immagini 2D sarà differente: si definisce disparità di quel punto, la differenza tra le sue coordinate nelle proiezioni sul piano immagine. Questa definizione può essere generalizzata applicandola a ogni punto della scena e l'insieme delle disparità costituisce come mostrato in fig. 1.4 un'immagine di disparità.

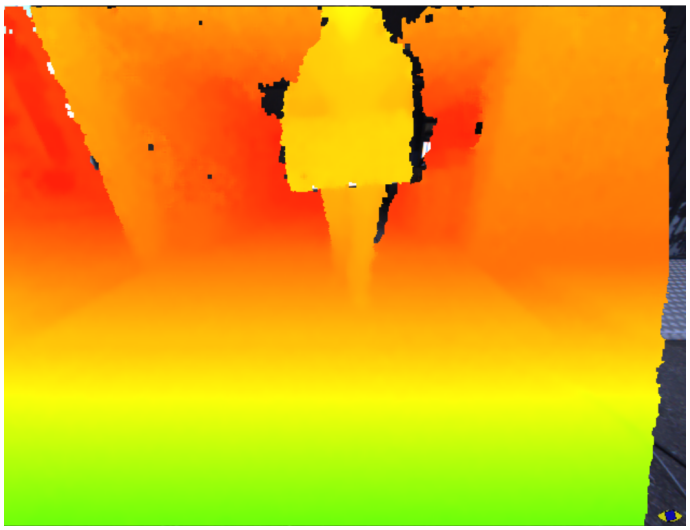


Figura 1.4: Disparità su camere pinhole.

La mappa di disparità (DSI) contiene sufficienti informazioni per identificare gli oggetti nel mondo 3D senza eseguire la triangolazione dei punti. La proiezione dei punti immagini 2D in punti mondo 3D è soggetta a errori causati dall'incertezza nella fase di calibrazione in particolare nella fase di rettifica. Gli algoritmi basati sulla DSI evitano il requisito di rappresentare gli oggetti nel mondo 3D attraverso dei modelli, voxel o griglie. In oltre evitando la triangolazione , viene anche ridotto il numero di calcoli richiesti.
Come descritto in letteratura [43] [44] accumulando i valori di disparità lungo le colonne o le righe è possibile come mostrato in fig. 1.5 creare degli istogrammi che prendono il nome di u-disparity e v-disparity e che risultano utili per identificare degli ostacoli o per stimare il terreno. In particolare la v-disparity, proposta da Labayrade, Aubert e Tarel [43], permette di ottenere un'immagine accumulando i pixel con la stessa disparità lungo lasse v dell'immagine (orizzontale). Al contrario la u-disparity è il complemento della precedente, l'immagine viene ottenuta accumulando i pixel con la stessa disparità lungo l'asse u dell'immagine (verticale), questo approccio venne presentato da Hu and Uchimura [44]. Come descritto in [43] la v-disparity può essere usata per stimare il profilo del terreno e per rilevare gli ostacoli. In particolare nell'articolo gli autori suppongono che gli angoli di rollio (roll) e di imbardata (yaw) della camera siano cosi piccoli da poter essere ignorati e attraverso la trasformata di Hough stimano la retta rappresentante la pendenza del terreno. Questa retta viene utilizzata per calcolare il profilo, l'altezza della camera e il pitch della stessa. Infine le linee verticali sulla v-disparity identificano i punti in cui iniziano gli ostacoli sulla strada. Il metodo così proposto per individuare gli ostacoli tende a unire ostacoli molto vicini tra loro, per questo motivo la v-disparity è diventato un metodo popolare per l'estrazione del profilo del terreno.
Hu and Uchimura in [44], utilizzano la v-disparity per stimare il terreno, e propongono la u-disparity per identificare gli ostacoli. Gli autori dimostrano nell'articolo che gli ostacoli all'interno della u-disparity possono essere identificati come delle linee orizzontali, tali linee sono realizzate poiché gli ostacoli sono formati da punti con lo stesso valore di disparità. Questo approccio è stato dimostrato essere ampiamente utilizzato per il rilevamento degli ostacoli, anche se il problema principale rimane

la scelta della soglia che determina quando una lina orizzontale dovrebbe essere considerato come un ostacolo o parte della strada.
Queste tecniche sono utilizzati in alcuni algoritmi come in [45] [46] per il calcolo della griglia occupazionale (*occupancy grid*) che verrà spiegata nella prossima sezione, in [47] per stimare *egomotion*, infine [48] Iloie, Giosan, and Nedevschi utilizzano la uv-disparity e la segmentazione per rilevare e raggruppare i pixel associati ai pedoni. In questa tesi come verrà spiegato successivamente verrà proposto una nuova variante sul modo con cui vengono accumulati i pixel della u-disparity.

Figura 1.5: Sono mostrate in figura come descritto in [1] l'immagine acquisita (in alto a sinistra) con la corrispettiva mappa di disparità (in alto al centro). La V-Disparity sulla destra, mostra il profilo stradale mentre la U-Disparity in basso evidenzia gli ostacoli.

1.2.2 Griglie Occupazionali

Le griglie occupazionali descrivono lo spazio attorno al veicolo attraverso delle matrici in cui vengono accumulati seguendo approcci probabilistici. Sono state introdotte per la prima volta da Elfes [49], per la rappresentazione di dati sonar utilizzati durante la navigazione. Da allora sono state estese per essere utilizzate con ogni tipo di dato, tra cui LIDAR in [50] e stereo camere in [51], inoltre sempre in [51] la stessa griglia viene usata per fondere dati provenienti da sensori differenti. Le griglie occupazioni

richiedono inoltre di modellare l'incertezza del dato proveniente dal sensore, nel caso delle camere stereoscopiche questa informazione viene ottenuta modellando l'errore di triangolazione del punto immagine come mostrato in [51], integrando il costo del volume [52], oppure usando una confidenza sulle corrispondenze ottenute [53].
Quando si passa da punti immagine 2D a punti mondo 3D, gli oggetti lontani vengono descritti da pochi punti, questo effetto non aiuta l'identificazione degli ostacoli. Per risolvere questo problema sono state introdotte nel tempo differenti griglie, tra cui quella polare che rappresenta lo spazio della griglia (u,v), in cui u rappresenta la posizione orizzontale del punto in coordinate camere, mentre v rappresenta la disparità del pixel [54]. Un vantaggio nell'utilizzo della griglia polare consiste nel non dovere utilizzare la triangolazione del punto e quindi tutti i calcoli possono essere eseguiti in coordinate immagine. Badino, Franke e Mester [54] hanno descritto come utilizzare questo tipo di griglie per individuare i confini degli oggetti rispetto al piano stradale attraverso l'utilizzo della programmazione dinamica, ottenendo risultati rapidi e accurati. Tuttavia questo tipo di approccio seppure dia una buona rappresentazione del mondo e permette la fusione delle informazioni provenienti da sensori diversi, non permette di conservare le informazioni riguardanti l'altezza e la geometria dell'ostacolo stesso.

1.2.3 Digital Elevation Maps

Le *digital elevation maps* (DEM) forniscono una rappresentazione a griglia del mondo 3D dall'alto, in cui il valore associato ad ogni cella corrisponde all'altezza di un oggetto o del terreno. Questo tipo di approccio fu proposto da Zhang [55] per la creazione di mappe 3D utilizzate da rover nel progetto VAP (Autonomous Planetary Rover program). Le DEM hanno il vantaggio di fornire una stima dell'altezza degli oggetti nelle vicinanze, al contrario delle griglie occupazionali che indicano solo lo spazio libero o non libero. Questo tipo di algoritmo come anche il precedente ha il vantaggio di poter essere calcolato partendo dalla nuvola di punti indipendentemente dal tipo di sensore utilizzato.
In [55] vengono usati piani e quadriche per modellare i punti ottenuti dallo *stereo-matching*. Iterativamente vengono eliminati i punti troppo lontani dalla superficie

trovata (attraverso una soglia scelta a priori), e le superfici vengono ricalcolate. Gli ostacoli sono identificati attraverso una soglia sulla DEM. Altri approcci, come quello di Oniga and Nedevschi [56], classificano la *point cloud* in punti appartenenti alla strada o agli ostacoli in base alla densità dei punti contenuti nelle celle della DEM. Oniga and Nedevschi presuppongono che le celle contenenti un maggiore numero di punti identificano degli ostacoli rispetto alle celle che contengono il terreno ad una data distanza, poiché gli oggetti sono identificabili come delle superfici verticali costituiti da più punti con la stessa disparità rispetto alle altre. Infine attraverso strumenti matematici come RANSAC[4] viene generato un modello rappresentante la strada, permettendo cosi di separare in maniera più precisa i punti appartenenti agli ostacoli da quelli del terreno.
L'utilizzo dei punti 3D durante la fase di accumulazione comporta la propagazione dell'errore ottenuto durante la fase di triangolazione dei punti 2D per ottenere il punto mondo desiderato, questo errore si ripercuote nella precisione con cui gli ostacoli vengono identificati nella DEM. Infine questi algoritmi richiedono l'elaborazione di ogni punto della *point cloud* (*dense point cloud*), questo può essere un problema se si utilizzano le moderne camere stereoscopiche poiché hanno un'alta risoluzione. Rimane ancora aperto in letteratura l'utilizzo sparso delle *point cloud* per la generazione delle DEM.

[4]RANSAC - RANdom SAmple Consensus

1.2.4 Stixel

Come mostrato in figura 1.6 gli stixel rappresentano gli ostacoli nel mondo attraverso dei rettangoli verticali e ad ognuno di essi è associato un valore di disparità. Risulta implicito che ogni pixel che non appartiene ad uno stixel appartiene alla strada. Ogni stixel può essere descritto attraverso una terna di punti che corrispondono a due vertici e alla disparità, attraverso questa forma compatta le risorse richieste per memorizzare le informazioni sugli ostacoli è notevolmente bassa.
Questo tipo di approccio è stato proposto da Badino, Franke, e Pfeiffer [57], per analizzare in modo efficiente le immagini di disparità ottenute da immagini ad alta risoluzione. Altre rappresentazioni simili sono state proposte in precedenza da Rebut, Toulminet e Bensrhair [58]. Anche Similarly Kubota, Nakano, e Okamoto [59] utilizzano delle colonne verticali per rappresentare gli ostacoli nella disparità, sebbene la loro ricerca sia focalizzata sulla rilevazione dei bordi tra ostacoli e strada utilizzando la programmazione dinamica e ignorando l'altezza degli ostacoli.
L'algoritmo proposto da Badino, Franke, e Pfeiffer [57], identifica lo spazio libero attraverso una griglia polare e utilizzano una FPGA per calcolare la disparità attraverso l'algoritmo SGM. All'interno dell'algoritmo viene individuato il *foreground* sottraendo le informazioni ottenute durante il calcolo della griglia, e attraverso la programmazione dinamica ottengono una segmentazione degli ostacoli in maniera precisa. Infine utilizzando le informazioni ricavate generano gli stixel. Questo tipo di rappresentazione oltre ad essere robusta e compatta fornisce informazioni sufficienti per identificare l'ambiente circostante a un veicolo.
Altri utilizzi degli stixel sono descritti in [60], in questo articolo Benenson, Timofte, e Van Gool hanno usato questo approccio per identificare i pedoni, ma invece di utilizzare la disparità hanno usato un volume dei costi in cui si ha un immagine 3D in cui l'intensità rappresenta il costo corrispondente dei pixel (u,v) a una determinata disparità. Ciò comporta una riduzione degli errori possibili sulla triangolazione e permette di incrementare la velocità di elaborazione.
Allo stesso modo in [60], Benenson, Mathias, Timofte, et al. [61], hanno proposto una formulazione alternativa al mondo degli stixel per accelerare la rilevazione degli ostacoli. Il piano stradale viene rilevato utilizzando la v-disparity ottenuta attraverso

il volume dei costi come nell'articolo precedente, di questa però vengono elaborate solo N righe al disotto dell'orizzonte (dato fornito in input). La stima della distanza degli stixel è ottenuta suddividendo l'immagine di input in più colonne e per ognuna di essa viene selezionato il gradiente orizzontale massimo. Questo approccio però suppone che gli ostacoli siano più larghi delle colonne che suddividono l'immagine, inoltre l'algoritmo richiede prestazioni hardware elevate, infatti gli autori affermano di ottenere delle performance elevate, di circa 260 fps per il calcolo degli stixel ma su un pc di fascia alta combinando l'elaborazione multi core alla GPU.

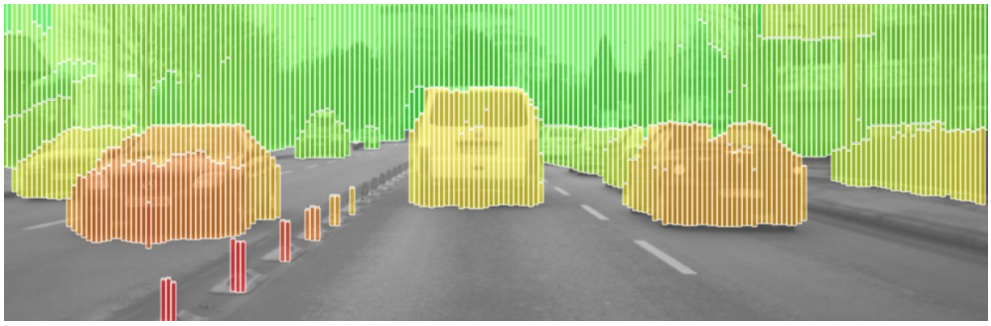


Figura 1.6: Descrive gli Stixel calcolati come descritto in [2], il colore rappresenta la distanza dell'ostacolo (rosso oggetti vicini e verde oggetti lontani).

1.3 Sistemi embedded

Come detto in precedenza il calcolo della mappa di disparità e il suo successivo utilizzo sono un compito molto oneroso per un sistema di elaborazione, sopratutto con le immagini ad alta definizione degli ultimi giorni. Per questo motivo, la maggiore parte dei sistemi analizzati come accennato nell'introduzione permettono oltre all'acquisizione il solo calcolo della mappa di disparità, tra questi è stato utilizzato per la prototipazione di un algoritmo il 3DV-E realizzato da Vislab come verrà descritto nei prossimi capitoli. Questo sistema permette di acquisire e calcolare la DSI con una risoluzione di 640x480 px a una frequenza massima di 25fps.
Tra i sistemi in commercio c'è il CV1 (fig. 1.7) realizzato dall'azienda Ambarella per la quale ho svolto il dottorato di ricerca, questo chip è il primo di una famiglia di processori per la computer vision. Il CV1 supporta l'elaborazione fino a una risoluzione di 3040x1728 *px* o 8 Mega pixel, l'architettura del chip è completamente programmabile, offrendo alte prestazioni a un bassissimo consumo (5W). Tale chip come descritto prossimamente è stato utilizzato per testare gli algoritmi che sono stati sviluppati.

Figura 1.7: Chip CV1 realizzato da Ambarella.

Capitolo 2

Strumenti di visione artificiale

Con il termine "visione artificiale" ci si riferisce a quella branca dell'informatica che si occupa dell'analisi delle immagini con lo scopo di determinare le proprietà geometriche, fisiche e dinamiche delle informazioni contenute nell'immagine stessa. Alcuni obbiettivi di questa disciplina sono ad esempio creare partendo da immagini un modello del mondo tridimensionale, riconoscere e classificare oggetti che ne fanno parte oppure estrarre le relazioni spaziali e logiche presenti nel mondo reale. Lo scopo di questa sezione è fornire alcuni strumenti che descrivano meglio i dati utilizzati negli algoritmi sviluppati e descritti nei prossimi capitoli. Verrà quindi presentato il modello pinhole e un modello sferico per la gestione di immagini ottenute con ottiche *fisheye*.
In particolare il modello sferico studiato e proposto è stato implementato e successivamente portato su un chip dall'azienda per cui ho svolto il dottorato.

2.1 Modello Pinhole Camera

Il modello Pinhole Camera [62], o anche definita camera stenopeica, è uno schema semplice ed efficace del funzionamento del sistema di visione artificiale di una fotocamera a lunghezza focale fissa. Questo modello risulta di gran lunga il più diffuso tra i ricercatori e nelle applicazioni reali. Il grande successo del pinhole è dovuto sia alla sua semplicità che alla grande disponibilità di strumenti matematici che possono esservi applicati, infatti grazie alla sua formulazione possono essere applicati ad esempio i teoremi e le formule della geometria epipolare, dell'invarianza proiettiva di segmenti conici e linee rette sottoposti alla trasformazione prospettica.
Innanzitutto, nel modello pinhole, si suppone che tutti i raggi luminosi che entrano nella fotocamera attraversando la lente convergano in un unico punto *C*, chiamato *centro ottico*, prima di colpire il piano immagine, ovvero il sensore della fotocamera come mostrato in fig. 2.1. Con questa ipotesi è possibile modellare dal punto di vista geometrico l'acquisizione di un'immagine come la trasformazione prospettica di una serie di punti tridimensionali in coordinate mondo ad una loro proiezione bidimensionale su un piano chiamato *piano immagine*, in coordinate immagini (*u*,*v*).

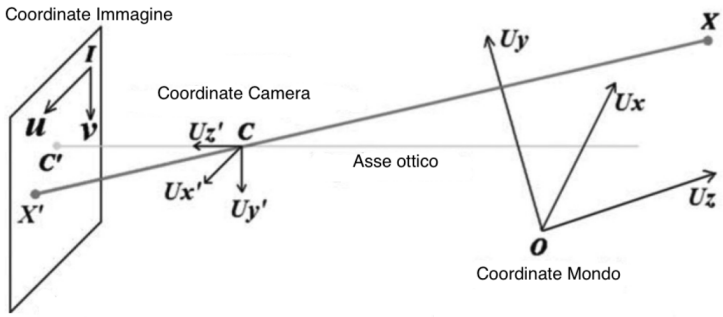


Figura 2.1: Rappresentazione del modello Pinhole.

Come mostrato in fig. 2.1, il piano immagine è parallelo al piano x'y' della camera e tutti i punti che vi giacciono sono descritti dal sistema di coordinate *I* definito attraverso i versori $\vec{u}$ e $\vec{v}$. La proiezione del *centro ottico* della camera *C'*,

è chiamato *punto principale* ed è posto idealmente al centro del piano immagine, nel punto in cui l'asse $\vec{z'}$ della camera interseca il piano. Tuttavia nei casi reali, a causa del disassamento tra macchina e ottica nel processo di fabbricazione, il punto principale non può essere considerato al centro dell'immagine, ma più genericamente alle coordinate $C' = (c_x , c_y)$.
L'asse ottico è ortogonale ed è identificato da $\vec{z'}$, mentre la distanza $\overline{CC'}$ tra la lente e il sensore è chiamato lunghezza focale f che è una caratteristica di ciascuna camera a focale fissa.
Elencati i sistemi di riferimento (immagine, camera e mondo) e le grandezze ad esse associate, si può osservare come il modello pinhole implementi una proiezione centrale in cui un generico punto mondo $X = (x, y, z)$ viene proiettato sul piano immagine $X' = (u', v')$, attraverso una retta passante sia per il punto X che per il centro ottico C.
Questo punto può essere descritto attraverso le *coordinate omogenee*, uno strumento matematico usato nella geometria proiettiva per descrivere i punti. Attraverso questa convenzione è possibile formulare i calcoli da effettuare come moltiplicazioni tra matrici e vettori. Si può quindi definire la *matrice di proiezione prospettica P* che descrive i parametri fisici della fotocamera e permette di modellare la proiezione prospettica. Attraverso tale matrice è possibile ottenere la matrice dei parametri intrinseci K e quella dei parametri estrinseci $[R|t]$.
Prendendo un generico punto mondo $X \in \mathbb{R}^3$ è possibile definirlo in *coordinate omogenee* $\overline{X} = (x, y, z, 1) \in \mathbb{R}^4$, analogamente un punto immagine $X' \in \mathbb{R}^2$ può essere scritto come $\overline{X'} = (u', v', 1) \in \mathbb{R}^3$, la relazione che lega X alla sua proiezione X' è descritta dall'equazione (2.1).

$$\overline{X'} = P\overline{X} = K\,[R|t]\,\overline{X} \tag{2.1}$$

Come osservabile dall'equazione (2.1) la matrice P può essere vista come la moltiplicazione di due matrici:

- **K** - la matrice 3x3 (2.2) con 5 gradi di libertà che descrive i parametri intrinseci della camera come la lunghezza focale f, il fattore di *skew* s solitamente

approssimato a 0 e la posizione del punto principale C' =(c_x , c_y).

$$K = \begin{bmatrix} f_u & s & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

- $[R|t]$ - la matrice 3x4 dei parametri estrinseci identifica la posizione del sistema di riferimento della camera nello spazio tridimensionale attraverso 6 gradi, ovvero la rotazione 3D e le rotazioni attorno agli angoli euclidei: *Roll*, *Pitch* e *Yaw*.

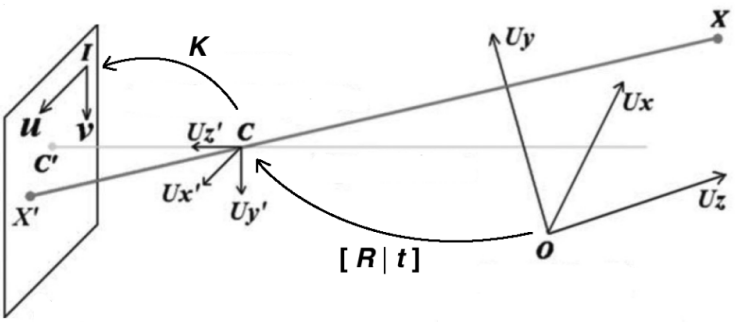


Figura 2.2: Rappresentazione delle trasformazioni applicate tramite le matrici dei parametri estrinseci ed intrinseci nel modello Pinhole.

Come mostrato in fig. 2.2, l'applicazione in successione delle matrici permette di descrivere il punto mondo, prima nel suo omologo in coordinate camera e successivamente di proiettare il punto attraverso la matrice dei parametri intrinseci sul piano immagine. Il risultato descritto dall'equazione (2.1) è definito a meno di un fattore di scala, ossia la terza coordinata del vettore $\overline{X'}$. Per ottenere le coordinate immagini partendo quindi da questo risultato è possibile dividere tutti gli elementi per il terzo elemento come descritto nell'equazione (2.3).

$$\overline{X'} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \equiv \begin{bmatrix} u/w \\ v/w \\ w/w \end{bmatrix} \equiv \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} \equiv \begin{bmatrix} u' \\ v' \end{bmatrix} = X' \tag{2.3}$$

Si nota che la trasformazione ottenuta attraverso il modello pinhole non è invertibile, ovvero non è possibile ricostruire il punto tridimensionale X a partire dalla sua proiezione X'. L'utilizzo delle coordinate omogenee (2.3) nasconde la non linearità delle operazioni svolte sul punto mondo. Questa riformulazione permette di osservare come si perda l'informazione contenuta nella terza dimensione z. Invertendo le operazioni si può ottenere la rappresentazione matematica della retta su cui giace il punto X in coordinate camera, si nota quindi che il punto originario può trovarsi in una qualsiasi posizione lungo la retta e per trovare il punto originario, il dato deve essere integrato con altre informazioni.
A causa delle caratteristiche delle ottiche della fotocamera, la proiezione pura implementata dal modello pinhole non è sufficiente per descrivere il processo di formazione dell'immagine, specialmente per sistemi a basso costo, poiché la distorsione introdotta dalle ottiche sposta i punti proiettati sul piano immagine deformando di conseguenza l'immagine catturata. Questo effetto è tanto più grande quanto più ci si allontana dal centro della lente, mentre per i raggi che attraversano il centro dell'ottica questo effetto si riduce fino ad annullarsi nel punto chiamato centro di proiezione, generalmente vicino al punto principale del modello stenopeico.
Secondo il modello ricavato dai lavori di Brown-Conrady [63] [64], la distorsione può essere comunque suddivisa in due componenti:

- **Radiale** - spostamenti diretti lungo la retta che unisce un punto al centro della distorsione

- **Tangenziale** - sono normalmente molto piccoli e vengono spesso trascurati.

La si può descrivere attraverso la funzione (2.4), dove x_c e y_c rappresentano le coordinate del centro ottico, K_i P_i sono corrispettivamente i-esimo coefficiente della distorsione radiale e tangenziale ed r è la distanza del punto x_d dal punto principale

e può essere calcolato come $\sqrt{(x_d - x_c)^2 + (y_d - y_c)^2}$, da questa ultima equazione è possibile notare la forte non linearità.

$$\begin{aligned} x_u &= x_d + (x_d - x_c)(K_1 r^2 + K_2 r^4 + ...) + (P_1(r^2 + 2(x_d - x_c)^2) + 2P_2(x_d - x_c)(y_d - y_c))(1 + P_3 r^2 + ...) \\ y_u &= y_d + \underbrace{(y_d - y_c)(K_1 r^2 + K_2 r^4 + ...)}_{\text{distorsione radiale}} + \underbrace{(P_2(r^2 + 2(y_d - y_c)^2) + 2P_1(x_d - x_c)(y_d - y_c))(1 + P_3 r^2 + ...)}_{\text{distorsione tangenziale}} \end{aligned} \tag{2.4}$$

Tuttavia i primi due coefficienti della distorsione radiale K_1 e K_2 sono sufficienti ad allineare il modello geometrico di camere alla realtà, almeno per le camere che non utilizzano ottiche di tipo *fisheye* (verranno introdotte nel prossimo paragrafo).
Per questo tipo di ottiche il modello pinhole risulta inefficiente.
Per stimare questi parametri si usa solitamente un modello chiamato R3D1P1, completamente determinato da un vettore d di 7 parametri reali: 3 termini radiali (k_1, k_2, k_3), 1 termine di decentramento (p_1, p_2) e 1 grado di distorsione prismatica (s_1, s_2). É possibile comunque utilizzare altri modelli più semplici, come ad esempio R3, che include solo i termini radiali. Questi parametri vengono solitamente stimati in modo automatico attraverso diverse tecniche disponibili e ampiamente descritte in letteratura che fanno uso di geometrie dalle dimensioni note, chiamate griglie di calibrazione.

2.2 Modello Sferico

Il modello pinhole camera affrontato nel paragrafo 2.1 non è in grado di descrivere delle camere che utilizzano ottiche con FoV maggiori di 120°. Questo tipo di ottiche offre il vantaggio di riuscire ad osservare oggetti posti anche a 180° rispetto, alla camera in base al tipo di lenti scelte.
Sono stati presentati in letteratura diversi approcci che potessero descrivere questo tipo di problema e il più utilizzato è il modello sferico trattato in [65] da Shigang Li. Questo modello suppone di avere una sfera unitaria e un punto P nello spazio. Come mostrato in fig. 2.3, l'intersezione della superficie della sfera con la linea che collega il punto P e il centro della sfera è la proiezione del punto sulla sfera. L'intera

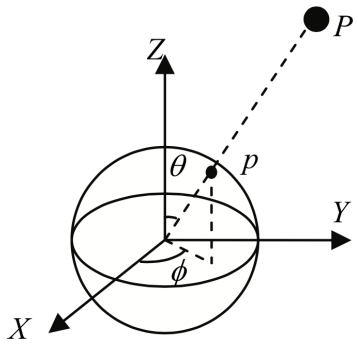


Figura 2.3: Modello sferico con relativa proiezione

immagine digitale viene quindi ottenuta dalla proiezione di tutti i punti attorno alla sfera. Le caratteristiche di questo modello sono:

- Full FOV: il modello idealmente riesce a descrivere i 360° in cui è immerso.
- Singolo *"View Point"*: tutti i punti sono descritti rispetto al centro della sfera.
- Cattura simultanea dell'ambiente circostante: i punti che compongono la scena sono proiettati tutti insieme sulla sfera.

Il modello sferico può essere quindi rappresentato come mostrato in fig. 2.3, in cui si osserva che il punto p corrisponde all'intersezione del raggio passante per il punto P della scena al centro ottico, e rappresenta la sua proiezione sul piano immagine.
Anche per il modello sferico cosi come per il modello $pinhole$ è possibile applicare la **geometria epipolare**, che serve per descrivere le relazioni e i vincoli geometrici che legano un'immagine 2D ad una scena 3D che l'ha generata.

2.2.1 Modello di Mei

Il modello pinhole generalizzato proposto nell'articolo di Mei [66] è basato su un lavoro di Geyer e Barreto insieme a una funzione per la distorsione radiale.
Questi tipi di strumenti utilizzano il modello sferico visto in precedenza.
Come mostrato in fig. 2.4, i punti mondo X catturati dal sistema ottico vengono proiettati all'interno della sfera unitaria X_s:

$$(X)_{Fm} \longrightarrow (Xs)_{Fm} = \frac{X}{||X||} = (X_s, Y_s, Z_s) \tag{2.5}$$

I punti vengono quindi descritti rispetto ad un altro sistema di riferimento centrato C_p, traducendosi in uno sfasamento lungo l'asse z con verso uscente della camera, vedi fig. 2.4.

$$(Xs)_{Fm} \longrightarrow (Xs)_{Fp} = (X_s, Y_s, Z_s + \xi) \tag{2.6}$$

Questi sono a loro volta proiettati mediante il centro prospettico C_p sul piano normalizzato π_{m_u}

$$m_u = (\frac{X_s}{Z_s + \xi}, \frac{Y_s}{Z_s + \xi}, 1) = h(X_s) \tag{2.7}$$

Dal piano normalizzato si passa al piano π_{m_d} attraverso l'aggiunta della distorsione radiale e tangenziale

$$m_d = m_u + D(m_u, V) \tag{2.8}$$

Infine la proiezione viene trasformata in coordinate camera tramite la matrice K, permettendo quindi il passaggio tra distanze metriche in pixel ed introducendo lo scostamento del *principal point* dal centro dell'immagine. Dove:

1. $[f_1, f_2]^T$ è la lunghezza focale in mm.
2. α è lo *skew*.
3. η è un parametro che tiene conto della distorsione della lente.
4. (u_0, v_0) sono il *principal point* in px.

$$p = Km = \begin{bmatrix} f_1\eta & f_1\eta\alpha & u_0 \\ 0 & f_2\eta & v_0 \\ 0 & 0 & 1 \end{bmatrix} m \tag{2.9}$$

È importante ricordare che nel modello non si considera il sensore come una camera con un catadiottro (si veda fig. 2.4), ma piuttosto come un unico dispositivo, non permettendo di scindere la lunghezza focale f da η. Possiamo quindi riscrivere 2.9, dove:

1. γ è una generalizzazione della lunghezza focale;

2. s è lo *skew* ed r è *l'aspect ratio*.

$$p = K(m) = \begin{bmatrix} \gamma & \gamma_s & u_0 \\ 0 & \gamma_r & v_0 \\ 0 & 0 & 1 \end{bmatrix} m = k(m) \tag{2.10}$$

A questo punto essendo h invertibile permette di calcolare il punto X_s corrispondente ad un punto m dato.

$$h^{-1}(m_u) = \begin{bmatrix} \frac{\xi+\sqrt{1+(1-\xi^2)(x^2+y^2)}}{x^2+y^2+1}x \\ \frac{\xi+\sqrt{1+(1-\xi^2)(x^2+y^2)}}{x^2+y^2+1}y \\ \frac{\xi+\sqrt{1+(1-\xi^2)(x^2+y^2)}}{x^2+y^2+1} - \xi \end{bmatrix} \tag{2.11}$$

Il modello descritto si basa su un caso ideale. L'applicazione dello stesso nel caso reale può essere definito attraverso quattro fasi:

1. Applicazione dei parametri estrinseci.
   Definiscono la posizione e l'orientamento del sistema di riferimento della camera, rispetto al mondo. Questa trasformazione viene indicata in [66] come W descritta attraverso l'uso del vettore $V^1 = [q_1, q_2, q_3, q_4, t_1, t_2, t_3]$ che contiene i parametri ad essa associati.

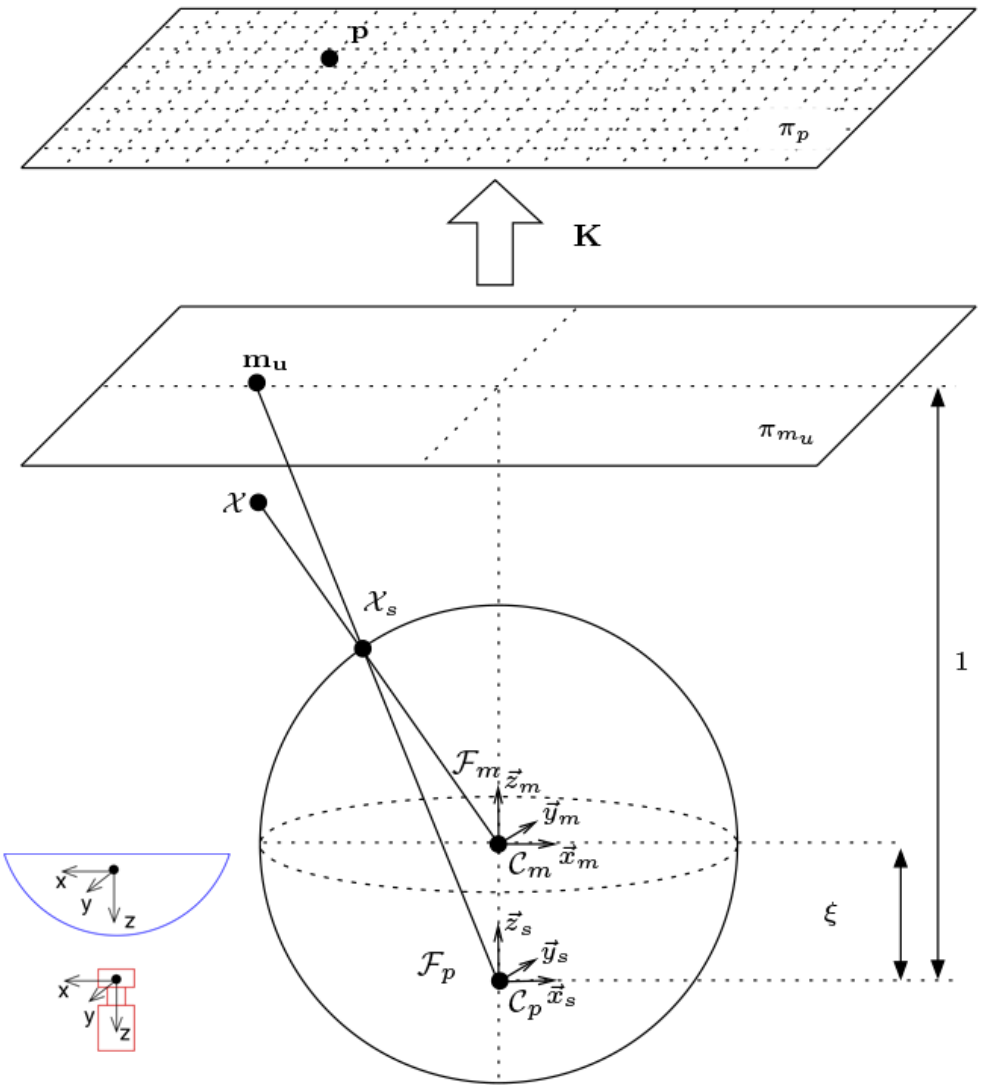


Figura 2.4: Dispositivo utilizzato con convenzione degli assi annesso a sinistra e modello *pinhole* generalizzato a destra

2. Ottica e proiezione sul piano normalizzato.
   Questa fase che in [66] viene indicata con H e si riduce, come abbiamo visto nel caso ideale all'applicazione della matrice h che dipende sostanzialmente da $V^2 = [\xi]$.

3. Distorsione.
   In [66] vengono trattati, nel caso reale, due tipi di distorsione assenti nel modello ideale.

   - Imperfezione della forma della lente, dovuta ad esempio ad un'imperfezione di fabbrica. Ciò è modellabile come una distorsione radiale, attraverso un modello con tre parametri:

   $$\mathcal{L}(\rho) = 1 + k_1\rho^2 + k_2\rho^4 + k_5\rho^6 \tag{2.12}$$

   con

   $$\rho = \sqrt{x^2 + y^2} \tag{2.13}$$

   - Assemblaggio non ottimo di lenti e camera che generano distorsioni tangenziali, modellabile attraverso due parametri:

   $$dx = \begin{bmatrix} 2k_3xy + k_4(\rho^2 + 2x^2) \\ k_3(\rho^2 + 2y^2) + 2k_4xy \end{bmatrix} \tag{2.14}$$

   Viene quindi indicata con D in [66] la funzione di distorsione e con $V^3 = [k_1], k_2, k_3, k_4, k_5]$ il vettore contenente i parametri.

4. Modello della camera.
   Indicata con P in [66] è la fase che descrive l'ultima trasformazione che sfrutta il modello *pinhole* per la proiezioni sul piano della camera. I parametri della trasformazione sono contenuti in $V^4 = [\alpha, \gamma_1, \gamma_2, u_0, v_0]^T$ che risultano essere:

   - α l'apertura focale.
   - γ_1, γ_2 le distanze focali.

- (u_0, v_0) il *principal point*.

La composizione delle quattro trasformazione in [66] forma l'equazione finale:

$$G = P \circ D \circ H \circ W \tag{2.15}$$

$$V = [V^1, V^2, V^3, V^4] \tag{2.16}$$

dove V è un vettore composto da 18 elementi rappresentanti i parametri delle varie trasformazioni.

Come per il modello pinhole camera anche per questo esistono diversi metodi di calibrazione, il più utilizzato risulta quello descritto in [67] chiamato Camodocal.

2.3 Visione stereoscopica

Nei paragrafi precedenti sono stati introdotti due modelli che permettono di descrivere il processo di formazione dell'immagine e delle relative trasformazioni prospettiche. Entrambi i modelli presentati mostrano come avvenga la perdita dell'informazione legata alla scala degli oggetti acquisiti, per risolvere questo problema può essere usato un sistema stereoscopico, ossia una coppia di camere che acquisiscono la stessa scena allo stesso istante da due punti di vista differenti.
In questa sezione si presenterà il problema della ricostruzione dell'informazione tridimensionale attraverso la triangolazione dei punti appartenenti alle immagini catturate. In particolare verranno presentati i concetti di geometria epipolare per i due modelli. Questo consente di ricostruire una scena osservata, se prima viene risolto il problema della corrispondenza o *stereo matching*.

2.3.1 Triangolazione con due camere pinhole

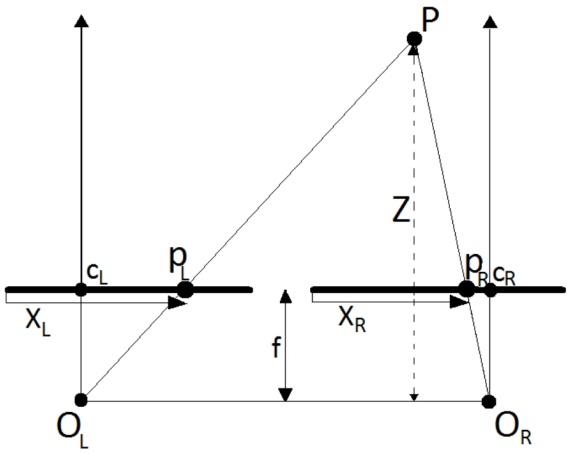


Figura 2.5: Disparità nel modello pinhole.

Come mostrato in fig. 2.5, un punto generico p_l sull'immagine acquisita dalla camera sinistra può essere proiettato nel mondo in una qualsiasi posizione, purché

appartenga alla retta passante per O_L e p_L. Si può quindi osservare che il punto lungo l'asse può essere posizionato in infiniti modi poiché, attraverso la trasformazione prospettica, l'informazione che identifica la posizione precisa è stata persa.
Si può però identificare la corretta collocazione del punto in coordinate mondo, se si riesce a individuare anche nell'immagine destra il punto da localizzare. Si può tracciare una retta passante per O_R e p_r che intersechi la retta precedente. In visione artificiale, l'individuazione del punto omologo di p_l ovvero p_r, prende il nome di stereo matching ossia problema dell'accoppiamento. Nella visione stereoscopica, si fa riferimento a due modelli: il sistema stereo laterale e quello a geometria epipolare.

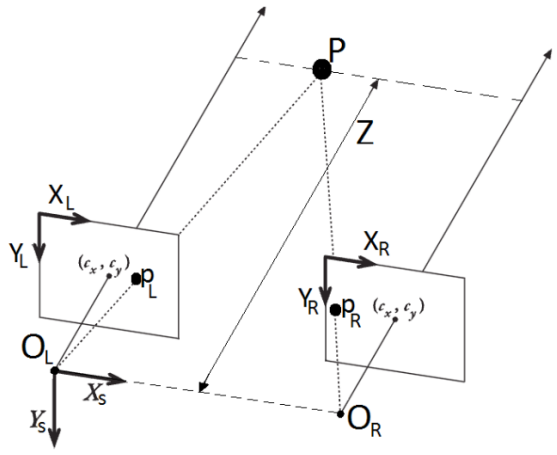


Figura 2.6: Assonometria un sistema stereo.

Il sistema stereo laterale è il modello standard e prevede come mostrato in fig. 2.6 due telecamere con stessa focale, con gli assi (x, y, z) paralleli e i piani immagine complanari. I due sistemi di riferimento sono quindi solidali a meno di una traslazione orizzontale, che viene definita dalla retta che congiunge i centri ottici O_L, O_r delle telecamere ed è chiamata *baseline b* del sistema stereoscopico.
Per ogni punto P nello spazio individuato dalle telecamere, la disparità è definita come $d = b * \frac{f}{z}$, che rappresenta la differenza tra le ascisse X_L e X_R dei punti situati sul piano immagine. Tuttavia le ipotesi del modello standard non sono applicabili in una

configurazione reale, quindi bisogna considerare il modello della *geometria epipolare* 2.7.

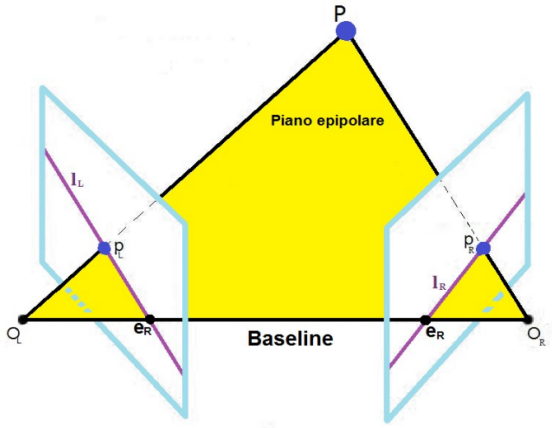


Figura 2.7: Geometria epipolare nel modello pinhole.

Per ottenere le informazioni sulla profondità data una coppia di immagini è necessaria una stereopsi computazione che individua i punti corrispondenti in entrambe le immagini ovvero che sono proiezione di uno stesso punto 3D. Questi punti, detti punti coniugati, possono essere computazionalmente onerosi da calcolare se ci si basa esclusivamente sui vincoli di similarità. Per ogni punto di una immagine sarebbe necessario verificare la similarità su tutte le righe e colonne dell'altra immagine generando innumerevoli falsi accoppiamenti. Ciò nonostante, il punto coniugato sull'altra immagine può trovarsi soltanto lungo una retta, detta retta epipolare. Questo vincolo è ottenuto dalla geometria dell'intersezione dei piani immagine con il fascio di piani passanti per il *baseline*.
Questa geometria, detta anche geometria epipolare, dipende solo dai parametri intrinseci delle due camere e non dalla geometria tridimensionale della scena. Risulta perciò che una retta nel piano ha come equazione in forma implicita $ax + by + c = 0$, quindi una retta epipolare in immagine può essere rappresentata dal vettore $l_i = \begin{bmatrix} a_i \\ b_i \\ c_i \end{bmatrix}$.

Analizzando l'immagine 2.7 si può quindi considerare tra tutti i piani passanti per la baseline quello passante per il punto P, chiamato piano epipolare. L'intersezione dei due piani immagine con il piano epipolare, genera due rette l_L e l_R dette linee epipolari passanti per le rispettive proiezioni p_L e p_R di P sui due piani immagine. Si può quindi dedurre che il punto corrispondente di uno dei due punti dovrà giacere sulla linea epipolare corrispondente sull'altro piano immagine. Da questa geometria si individuano anche i punti e_L e e_R ottenuti dalla intersezione dei piani immagine con la baseline, questi punti vengono detti *epipoli*.

2.3.2 Triangolazione con due camere fisheye

Il concetto per la triangolazione dei punti descritto precedentemente nel caso pinhole è espandibile come mostrato in fig. 2.8 anche al caso del modello sferico.

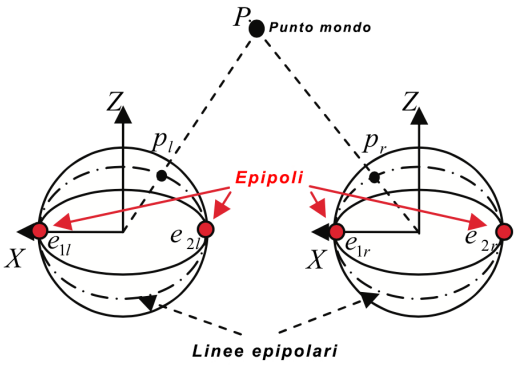


Figura 2.8: Modello sferico con elementi di geometria epipolare.

Tuttavia la ricerca dei punti omologhi come affrontato precedentemente non è più lungo una retta ma lungo un arco di semicirconferenza, questa ricerca risulta essere quindi lunga e inefficiente. Perciò dato il costo elevato della ricerca è stato adottato, con le opportune modifiche, l'algoritmo proposto in [68] che effettua una trasformazione degli archi in rette, la trasformazione viene chiamata *"latitude - longitude"*. In questo modo si ritorna, come nel modello classico, alla ricerca dei punti omologhi

lungo la stessa riga, riutilizzando di fatto gli eventuali algoritmi proposti per le camere non fisheye.

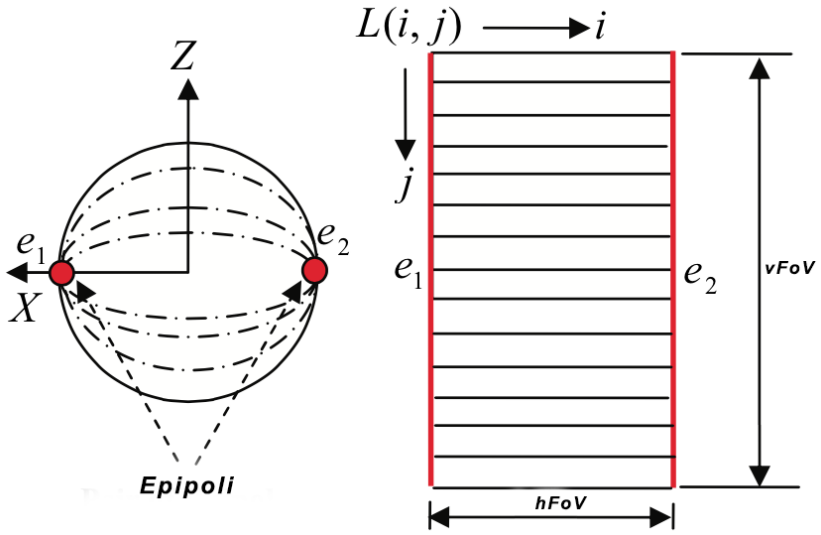


Figura 2.9: Trasformazione *latitude-longitude*.

Come mostrato in fig. 2.9 e spiegato in 2.2, le linee epipolari nel modello sferico sono dei grandi cerchi che si intersecano con gli epipoli. Questi vengono considerati nell'algoritmo come i due poli di un "pianeta" e si effettua quindi una rettificazione dei meridiani.
Con questa trasformazione, come si vede in fig. 2.9, i cerchi intersecanti gli epipoli diventano delle rette parallele. Nello specifico un punto mappato nell'immagine $L(i, j)$, $(i = 0, 1, ..., m; j = 0, 1, .., n)$ può essere rimappato nella sfera normalizzata $p(u, v, s)$

con le seguenti equazioni:

$$u = cos(\frac{i}{m}\pi) \tag{2.17}$$

$$v = sin(\frac{i}{m}\pi)cos(\frac{j}{n}2\pi) \tag{2.18}$$

$$s = sin(\frac{i}{m}\pi)sin(\frac{j}{n}2\pi) \tag{2.19}$$

dove m e n corrispondono rispettivamente alla distanza tra gli epipoli e alla circonferenza più grande nel modello sferico. Queste equazioni per i test effettuati sono state modificate in:

$$x = cos(\frac{zenit}{m}\pi) \tag{2.20}$$

$$y = sin(\frac{zenit}{m}\pi)cos(\frac{azimuth}{n}m_{FOV}) \tag{2.21}$$

$$z = sin(\frac{zenit}{m}\pi)sin(\frac{azimuth}{n}m_{FOV}) \tag{2.22}$$

dove x, y, z rappresentano le coordinate sferiche nella sfera unitaria, $zenit$ e $azimuth$ vengono inizializzati rispettivamente a 0 e $-\frac{n}{2}$ e fatti scorrere lungo tutta la dimensione $m \times n$ dell'immagine. Infine la variabile m_{FOV} contiene il *horizontal field of view* (hFoV) e π corrisponde al *vertical field of view* (vFoV). Nelle fig. 2.10, 2.11 viene mostrata una coppia di immagini stereo fisheye prima e dopo l'applicazione dell'algoritmo.

La definizione di disparità che è stata data per il pinhole non è più valida per il modello sferico. Shigang Li in [69] ridefinisce il concetto di disparità per una coppia di camere sferiche.

Supponiamo di avere una coppia di telecamere sferiche, come mostrato in fig. 2.8, che utilizza per la proiezione delle immagini il modello sferico discusso in 2.2. Si assuma anche (fig. 2.8) che i sistemi di coordinate siano paralleli e che gli assi x siano collineari, si può quindi definire che la *baseline* (fig. 2.12) sia la retta che collega i due fuochi e che attraversa gli epipoli delle due sfere, gli epipoli trovandosi sulla superficie della sfera unitaria avranno come coordinate $(\pm 1, 0, 0)$. Per definire la

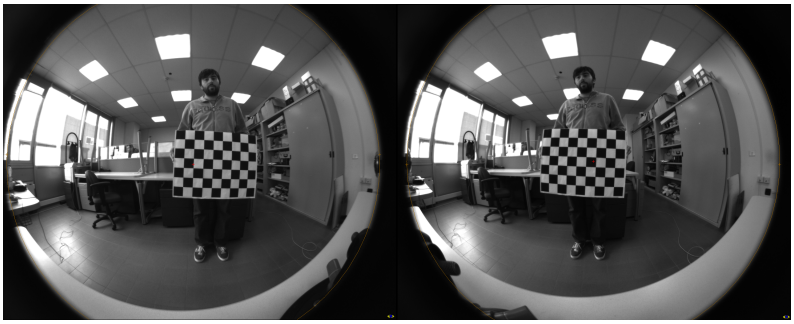


Figura 2.10: Frame prima dell'applicazione dell'algoritmo.

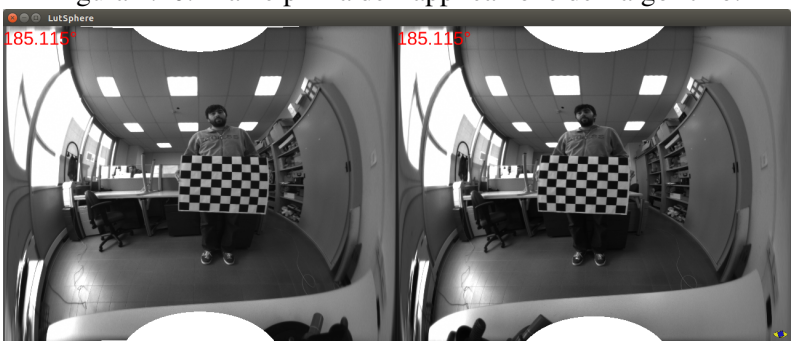


Figura 2.11: Risultato dell'algoritmo rivisitato *latitude-longitude*

disparità in [69] (fig. 2.12) si prende in considerazione il piano epipolare individuato dai centri ottici e dal punto P e vengono definiti θ_l e θ_r come gli angoli relativi alla proiezione del punto della scena nell'immagine sferica rispetto all'asse x.
Si definisce la disparità d_s del punto della scena P come la differenza degli angoli θ_l e θ_r.

$$d_s = f_s(\theta_l - \theta_r) \tag{2.23}$$

dove f_s è il raggio della circonferenza, ovvero la lunghezza focale dell'immagine sferica.
Essendo la sfera normalizzata si ha $f_s = 1$ e quindi si ottiene la disparità normalizzata

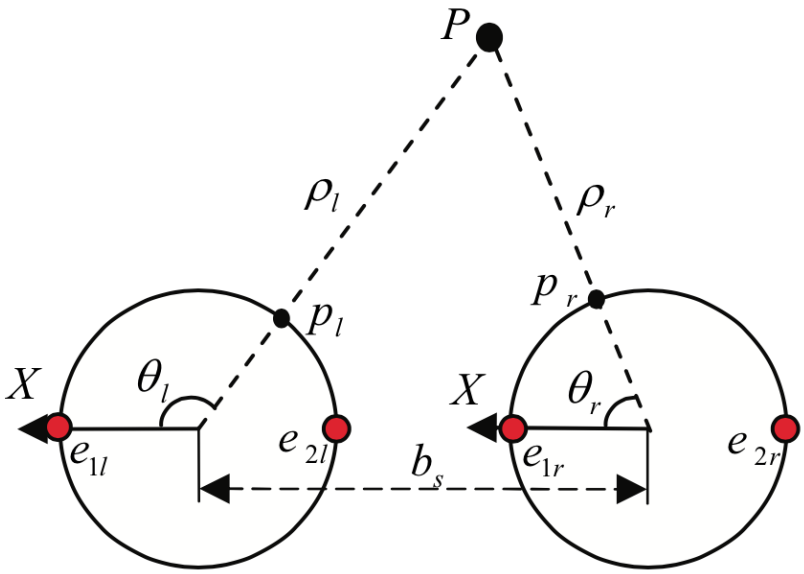


Figura 2.12: Disparità nel modello sferico

come:

$$d_n = \theta_l - \theta_r \tag{2.24}$$

L'equazione così ottenuta restituisce un angolo che risulta indipendente dai parametri intrinseci della camera. Rimane infine il calcolo della distanza dei punti data la disparità espressa come differenza di angoli, dato che i punti della scena sono distribuiti sulle immagini sferiche, la posizione dei punti nel mondo coincide con la distanza dal punto al centro della sfera.

Dalla fig. 2.12, possiamo riscrivere θ_r e θ_l rispettivamente come:

$$\theta_r = \frac{a_r}{f_s} \tag{2.25}$$

$$\theta_l = \frac{a_l}{f_s} \tag{2.26}$$

Possiamo quindi calcolare la distanza dal centro delle sfere al punto P come:

$$\rho_l = b_s \frac{sin(\theta_r)}{sin(\theta_l - \theta_r)} \tag{2.27}$$

$$= b_s \frac{sin(\frac{a_r}{f_s})}{sin(\frac{a_l}{f_s} - \frac{a_r}{f_s})} \tag{2.28}$$

$$= b_s \frac{sin(\frac{a_r}{f_s})}{sin(\frac{d_s}{f_s})} \tag{2.29}$$

$$\rho_r = b_s \frac{sin(\pi - \theta_l)}{sin(\theta_l - \theta_r)} \tag{2.30}$$

$$= b_s \frac{sin(\pi - \frac{a_l}{f_s})}{sin(\frac{a_l}{f_s} - \frac{a_r}{f_s})} \tag{2.31}$$

$$= b_s \frac{sin(\pi - \frac{a_l}{f_s})}{sin(\frac{d_s}{f_s})} \tag{2.32}$$

$$\tag{2.33}$$

Dato che la sfera è normalizzata possiamo riscriverle in:

$$\rho_l = b_s \frac{sin(\theta_r)}{sin(d_n)} = b_s \frac{sin(\theta_l - d_n)}{sin(d_n)} \tag{2.34}$$

$$\rho_r = b_s \frac{sin(\pi - \theta_l)}{sin(d_n)} = b_s \frac{cos(\theta_r + d_n)}{sin(d_n)} \tag{2.35}$$

Nei test effettuati per testare questo modello che è stato portato su chip si è calcolata la distanza del punto sfruttando solo la proiezione ottenuta della sfera destra, adattandola in questo modo:

$$\rho_l = b_s \frac{sin((u * m\_lat\_resolution) - (d_n * m\_lat\_resolution))}{sin(d_n * m\_lat\_resolution)} \tag{2.36}$$

dove b_s è la distanza tra le camere, d_n è il valore della disparità e $m\_lat\_resolution = \frac{\pi}{m}$ con m pari alla larghezza dell'immagine rettificata col metodo *"latitude-longitude"* 2.9.

2.4 Algoritmi per il calcolo della disparità

Nei paragrafi precedenti è stato mostrato come a prescindere dal significato che viene attribuito alla disparità che nel modello pinhole corrisponde a una differenze tra pixel mentre nel modello sferico è riferito ad una differenza di angoli, il modo con cui i vengono cercati i pixel omologhi si riconduce in entrambi i modelli con una ricerca lungo una retta.
Questi algoritmi sono tuttora uno dei filoni di ricerca in letteratura più attivi nel campo della visione artificiale.
Gli approcci proposti in letteratura si dividono in due macro-categorie:

- **feature-based** - il *matching* viene effettuato tra determinate *feature* estratte dall'immagine. L'immagine di disparità ricavata risulta essere sparsa poichè vengono analizzati solo certi punti dell'immagine. Il vantaggio di questi algoritmi è che sono robusti e computazionalmente veloci.

- **area-based** - il *matching* è effettuato tra tutti i pixel dell'immagine. L'immagine di disparità risulta quindi densa e per ogni pixel viene associato un determinato valore, escludendo le aree occluse o ambigue per la mancanza di *texture*. Questo approccio risulta molto oneroso poiché vengono presi in considerazione tutti i pixel dell'immagine di riferimento e confrontati con tutti i punti omologhi dell'immagine di confronto.

Per quanto riguarda la seconda categoria, un riassunto e una valutazione di tali algoritmi è stata presentata da Scharstein e Szeliski [70]. Si suddividono sostanzialmente in tre categorie:

- **algoritmi locali** - la disparità assegnata ad un pixel viene calcolata da pixel spazialmente vicini a quello considerato. Il criterio utilizzato per la ricerca è costituito dalla similarità tra finestre di dimensioni prefissate o adattive centrate nei pixel considerati. Questi approcci sono generalmente veloci, ma meno accurati.

- **algoritmi globali** - la disparità assegnata ad un pixel dipende dalle informazioni dedotte da tutta l'immagine. Questo problema viene rappresentato generalmente

attraverso la minimizzazione di una funzione energia. Questi approcci sono computazionalmente onerosi, ma molto accurati.

- **algoritmi semi-globali** - l'approccio è lo stesso dei globali, ma viene utilizzato un sotto-insieme dell'intera immagine, generalmente analizzando la funzione energia solo lungo certi percorsi. Questa soluzione è un compromesso tra velocità e accuratezza.

Nel lavoro presentato da Scharstein e Szeliski, inoltre vengono identificati quattro blocchi ricorrenti negli algoritmi area-based densi:

- Calcolo dei costi delle corrispondenze.
- Strategia di aggregazione dei costi.
- Calcolo della disparità.
- Raffinamento della disparità.

Per gli algoritmi che verranno presentati successivamente sono stati usati entrambi i metodi per il calcolo della disparità, in particolare per il landmark detector è stato realizzato un algoritmo *feature-based* con disparità sparsa mentre per l'algoritmo di rilevazione degli ostacoli è stato utilizzato l'algoritmo SGM[1] poiché già presente sul chip utilizzato, questo algoritmo genera una disparità densa.

2.4.1 SGM - Semi Global Matching

Questo tipo di algoritmo appartiene alla categoria dei semi-globali, che effettua il calcolo dell'energia lungo 8 percorsi come mostrato in 2.13, chiamati *scanline*, a 45° l'uno dall'altro, inoltre le relative funzioni di costo associate alle direzioni sono indipendenti fra di loro.

La funzione di costo è espressa dalla funzione $E(d) = E_{data}(d) + E_{smooth}(d)$, dove $E_{data}(d)$ rappresenta la funzione di costo che misura la corrispondenze diretta

[1]SGM - Semi Global Matching

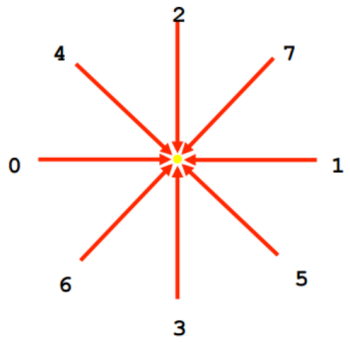


Figura 2.13: Le 8 scanline in SGM.

tra i punti presi in esame e $E_{smooth}(d)$ è un termine che penalizza le variazioni di disparità. Sulla funzione di penalizzazione $E_{smooth}(d)$ si possono ricavare delle considerazioni:

- In regioni uniformi il termine $E_{smooth}(d)$ è sufficientemente alto da impedire variazioni considerevoli di disparità, consentendo di ridurre il rumore.
- Lungo i bordi ovvero nelle regioni di discontinuità ci sarà una fase transitoria in cui inizialmente $E_{smooth}(d)$ impedirà la variazione di disparità. Nel momento in cui avviene la variazione entrambi i termini vengono ridotti e consentono di effettuare il cambio di disparità.

La funzione di costo è calcolata lungo le 8 direzioni all'interno dell'intervallo di disparità ovvero $d \in [Disparity_{MIN}, Disparity_{MAX}]$.
Indicando con x e y le coordinate del punto analizzato, la funzione di costo per la scanline 0 può essere descritta dalla funzione 2.37.

$$L(x,y,d) = C(x,y,d) + \min L(x-1,y,d), L(x-1,y,d-1) + P1, L(x-1,y,d+1), L(x-1,y,i) + P2 \tag{2.37}$$

Si può notare dalla funzione 2.37 che le due componenti $E_{data}(d)$ e $E_{smooth}(d)$ sono ben suddivise:

- $E_{data} = C(x, y, d)$ indica il risultato di una funzione di costo locale applicata ai punti presi in esame che è indipendente dal resto dell'algoritmo.

- E_{smooth} è calcolata in maniera ricorsiva, consente quindi di tenere traccia della storia passata lungo uno specifico percorso e tende a penalizzare i cambi di disparità attraverso l'utilizzo dei due pesi $P1$ e $P2$ con $P1 < P2$. Trattandosi di una somma di termini sempre positivi, il valore del costo di un cammino può crescere senza limite, per questo motivo si sottrae il termine $minL(x - 1, y, i)$

Lo scopo dei due pesi $P1$ e $P2$ è quello di penalizzare i cambi di disparità bruschi tra punti vicini, garantendo allo stesso tempo la continuità dell'immagine di disparità e permettendo all'algoritmo di adattarsi alle superfici curve o inclinate grazie al peso $P1$, ma anche di preservare eventuali discontinuità presenti nella scena con il peso $P2$. Una volta calcolate le 8 funzioni di costo associate alle *scanline*, queste vengono aggregate e ne viene calcolato il minimo. L'indice del minimo d rappresenta la disparità associato al pixel analizzato.

Capitolo 3

Keyframe Localization

Il problema della localizzazione per droni in situazioni senza GPS come descritto nel capitolo 1.1 è ben documentato in letteratura, tuttavia le soluzioni proposte non si adattano ad un sistema l *ow power*.
In questo capitolo verrà descritto l'algoritmo sviluppato e il porting su chip: l'obbiettivo è di realizzare un algoritmo che sfruttando una disparità sparsa e utilizzando un chip *low power*, si possa ridurre il *drift* da cui la localizzazione esistente era affetta e permetta al drone durante una missione di atterrare su determinate basi di dimensioni 1x1m. In particolare il modulo esistente di localizzazione attraverso un filtro di Kalman esteso (EKF) permetteva di ottenere posizione e assetto del drone, utilizzando come dati di input la visual odometry e uno stimatore di altezza (Visual Height Estimation). Come verrà mostrato nella sezione dei test l'utilizzo di questi due soli dati non era sufficiente al compimento della missione.
Il capitolo è suddiviso in cinque sezioni, nella prima viene descritto il sistema in cui è stato testato l'algoritmo durante la fase di sviluppo. La seconda sezione descrive l'algoritmo nelle sue diverse parti, nella terza viene accennato come modificare tale algoritmo per sostituire il modello pinhole con il modello sferico descritto in 2.2 e il quarto paragrafo si pone come obbiettivo di descrivere come sia stato possibile portare il lavoro sul chip sviluppato dall'azienda. Infine nella sezione cinque sono mostrati i risultati ottenuti.

3.1 Ambiente di sviluppo

Il drone utilizzato è il DJI Matrice 100. Come mostrato in fig. 3.1 il drone è un quadricottero realizzato da DJI (Dà-Jiāng Innovations Science and Technology Co. Ltd), un'azienda tecnologica cinese con sede a Shenzhen, nel Guangdong, con stabilimenti in tutto il mondo.

Figura 3.1: Dji Matrice 100.

Le specifiche su questo quadricottero sono riportate in tabella 3.1.

L'elaborazione degli algoritmi che permettono al drone di muoversi autonomamente è svolta attraverso una scheda Pico-ITX fig. 3.2 montata sul drone stesso. Sulla scheda è presente la distribuzione linux LUbuntu 14.0 (Trusty Tahr) e le specifiche tecniche sono riportate nella tabella 3.2. Come osservabile dalle specifiche questa scheda si presta bene alla prototipazione, poiché offre una buona CPU, grafica, prestazioni multimediali e flessibilità. Inoltre avendo un peso ridotto e consumi accettabili permette di essere montata facilmente sul DJI Matrice 100.

Per la percezione sono state montate due coppie di telecamere stereo, come mostrato in fig. 3.3, poste una frontalmente al drone e una al di sotto del drone. In

Peso	2355g
Peso massimo al decollo	3400g
Massimo angolo di inclinazione	30°
Velocità in discesa	5m/s
Massima resistenza al vento	10m/s
Velocità massima	22m/s
Tempo in volo (senza carico)	22min
Tempo in volo (500g di carico)	17min
Tempo in volo (1Kg di carico)	13min

Tabella 3.1: DJI Matrice 100 specifiche

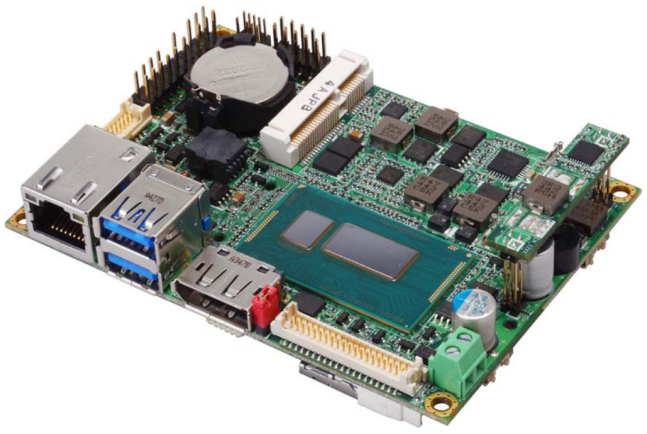


Figura 3.2: Pico-ITX motherboard.

CPU	Intel® Broadwell I7 5650U CoreTM U-series Processor
Memory	One DDR3L (support 1.35V) 1333/1600 SO-DIMM up to 8GB
Integrated Graphics	Intel® 5th/4th Gen integrated HD Graphics
LVDS interface	Onboard 24-bit dual channel LVDS connector with +3.3V / +5V supply
Serial ATA	Support 1 x SATA3 with 600MB/s (6Gb/s) transfer rate
Audio	Realtek ALC888 HD Audio
LAN Interface	Intel® I218-LM Gigabit LAN
Extended interface	One PCIE mini card socket or mSATA.
Internal I/O	1 x SATA3, 1 x DVI-D, 1 x LVDS, 1 x LCD inverter, 1 x PS/2, 2 x RS232, 1 x LPC, 1 x SMBUS, 1 x Audio, 2 x USB2.0, 1 x DC Out
Rear I/O	1 x Display Port, 1 x RJ45, 2 x USB 3.0
Power	9-24V input or DC 12V(Optional)

Tabella 3.2: Pico-ITX specifiche

particolare le coppie stereo utilizzate sono dei 3DV-E realizzati dal Vislab con una struttura opportunamente alleggerita per essere montato su un quadricottero. Ogni 3DV-E monta due camere con lenti pinhole e una FPGA che permette di ottenere a 10fps le immagini sincrone catturate con una risoluzione 640x480px, insieme a una mappa di disparità calcolata attraverso l'algoritmo SGM (2.4.1).

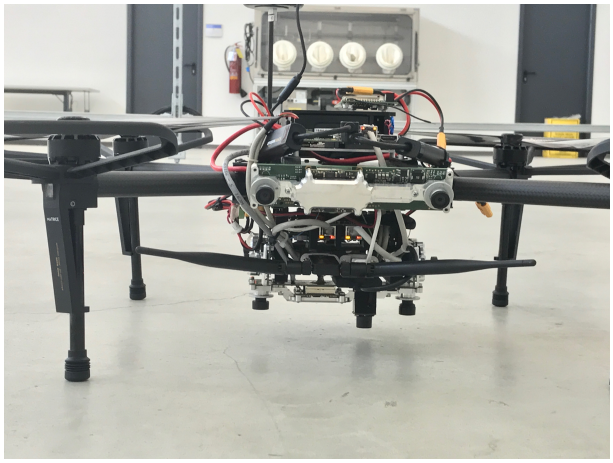


Figura 3.3: Vista frontale del drone allestito con i due 3DV-E.

3.2 Algoritmo

L'obbiettivo dell'algoritmo è quello di determinare la posa della camera rispetto ad alcuni punti noti, ovvero rispetto dei frame selezionati attraverso delle euristiche e chiamati keyframe; in questo modo il filtro di localizzazione può applicare una correzione nel determinare la posa del drone. L'algoritmo realizzato sfrutta la coppia stereo posizionata sotto il drone e monta lenti pinhole. In fase di sviluppo dell'algoritmo è stato deciso di non calcolare la disparità ma di utilizzare la mappa densa già fornita dai 3DV-E in input, ma come verrà mostrato nei prossimi paragrafi sono state apportate delle modifiche per il porting dell'algoritmo sul chip.
I passi eseguiti da tale approccio in fase di sviluppo sono i seguenti:

- Salvataggio della posa attuale del drone ottenuta dal filtro di localizzazione (INS).
- Estrazione dei keypoint e dei descrittori dalle immagini stereo ottenute in input.
- Calcolo dei punti 3D e validazione dei keypoint attraverso l'uso della DSI ottenuta in input.

- Selezione del keyframe più vicino alla posizione corrente (i keyframe vengono salvati attraverso un'euristica).
- Associazione tra i descrittori (*brute-force*).
- Calcolo della matrice Fondamentale per il filtraggio di eventuali *outlier* (con RANSAC).
- Ricerca della rotazione e della traslazione tra la camera e il keyframe, attraverso il riallineamento delle point cloud 3D.
- Ottimizzazione della mappa e ri-localizzazione.

3.2.1 Estrazione dei keypoint

Nell'algoritmo sviluppato l'estrazione delle feature è uno dei punti principali, poiché questi punti vengono utilizzati per localizzarsi. Come descritto nel capitolo (1.1.4), esistono diversi tipi di approcci che permettono di estrarre delle caratteristiche salienti da un'immagine. Dato che le immagini da utilizzare sono catturate da un drone in volo e le telecamere sono poste sotto di esso, le immagini acquisite risultano spesso ruotate o scalate, per questo motivo si è deciso di utilizzare dei detector che fossero invarianti alla rotazione e alla scala, in modo da ridurre le associazioni errate ed ridurre il numero di keyframe necessari per la localizzazione. Sono stati quindi scelti ORB [1] [71] e AKAZE [72] rispettivamente.

3.2.2 Validazione dei punti 3D

Per ogni frame le feature che non soddisfano alcuni criteri vengono eliminate. I criteri utilizzati per la validazione delle feature estratte sono:

- Avere una disparità valida.
- Il valore di disparità deve essere contenuto in un intervallo prestabilito.

[1] ORB - Oriented FAST and Rotated BRIEF

In fase di sviluppo per la validazione dei punti si è utilizzata una mappa di disparità densa poiché il 3DV-E utilizzato permetteva di avere in uscita il dato già pronto all'uso, come si vedrà nel prossimo capitolo per motivi di efficienza si è passati ad un approccio differente.

3.2.3 Selezione o salvataggio del Keyframe e confronto tra i descrittori.

Ad ogni frame, viene ricevuta la posizione del drone proveniente dal filtro di localizzazione.
Questo dato viene utilizzato dall'algoritmo per cercare in memoria il keyframe più vicino spazialmente alla posa attuale. Sono state adottate delle euristiche per il salvataggio dei keyframe. In particolare, se un frame dista più di due metri dal frame preso in esame allora il frame corrente viene salvato in memoria come keyframe, queste euristiche sono state decise in base al tipo di ottiche utilizzate in quanto si è cercato di non avere keyframe che contenessero porzioni di scene in comune, in modo da ottimizzare lo spazio occupato dai keyframe nel sistema.
Successivamente, a tale euristica ne sono state aggiunte altre, come la selezione del keyframe quando il numero di feature rispetto al frame precedente è inferiore a una determinata soglia e la selezione del primo keyframe a 40cm dal terreno. Questi controlli sono stati inseriti in fase di *porting*, come vedremo più avanti. Selezionato il keyframe più vicino si prosegue con il *matching* dei descrittori. Avendo dei descrittori binari è possibile utilizzare la distanza di Hamming per il confronto. Date due stringhe di ugual lunghezza la distanza tra di esse è il numero di posizioni nelle quali i simboli corrispondenti sono diversi. Come mostrato in fig. 3.4 la distanza di Hamming misura il numero di sostituzioni necessarie per convertire una stringa nell'altra.

3.2.4 Validazione dei punti attraverso la Matrice Fondamentale

Date le varie corrispondenze dei match ottenute al passo precedente tra i descrittori del frame corrente con quelli del keyframe, è possibile eliminare gli *outlier* attraverso il calcolo della matrice fondamentale con RANSAC.
RANSAC, è un algoritmo non deterministico pubblicato da Fischler [73] ed è utilizzato

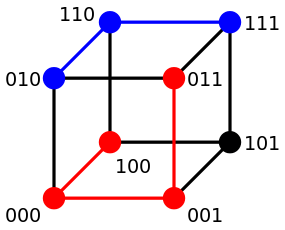


Figura 3.4: Due esempi di distanza: 100->011 ha distanza 3 (percorso rosso); 010->111 ha distanza 2 (percorso blu)

per la stima di parametri appartenenti ad un modello in cui l'insieme dei dati è rumoroso. Tale algoritmo iterativamente si alterna tra due fasi: la fase di generazione delle ipotesi e la fase di valutazione delle ipotesi.
All'interno di RANSAC la matrice Fondamentale è calcolata attraverso l'utilizzo di un elenco di punti omologhi, nello specifico è stato utilizzato l'algoritmo a 8 punti. La matrice Fondamentale è definita (Faugeras e Hartley, 1992) come:

$$p_2{}^T F p_1 = 0 \tag{3.1}$$

dove $p_1 e p_1$ sono, coordinate omogenee, dei punti omologhi rispettivamente sulla prima e sulla seconda immagine, nel nostro caso tali punti sono identificati dai match nel frame corrente e in quello precedente ovvero del keyframe. Nel caso in cui i match descrivono lo stesso punto nel mondo e quindi sono corretti allora l'equazione 3.1 deve essere soddisfatta.

3.2.5 Allineamento di point cloud 3D

Trovare la migliore rotazione e traslazione tra due nuvole di punti 3D come mostrato in fig. 3.5, in modo che risultino allineate, è un problema comune in letteratura. Si è deciso di utilizzare come approccio un metodo presentato da Besl and McKay in [74]; tale approccio è utilizzato per trattare il riallineamento di *point cloud* 3D provenienti da sensori come laser scanner o kinect, ma può essere riutilizzato anche nel caso di camere stereo. L'utilizzo di questa tecnica è stata preferita ad altre per

motivi di efficienza. I dati utilizzati in questa fase sono poveri di *outlier* e dai test effettuati le iterazioni massime necessarie in RANSAC per ottenere il riallineamento delle *point cloud* è massimo di dieci. Infine, altre tecniche come la ri-proiezione dei punti dell'immagine corrente sull'immagine contenente il keyframe avrebbe richiesto un minimizzatore per ridurre la distanza tra i punti, e questo processo richiederebbe prestazioni hardware elevate.

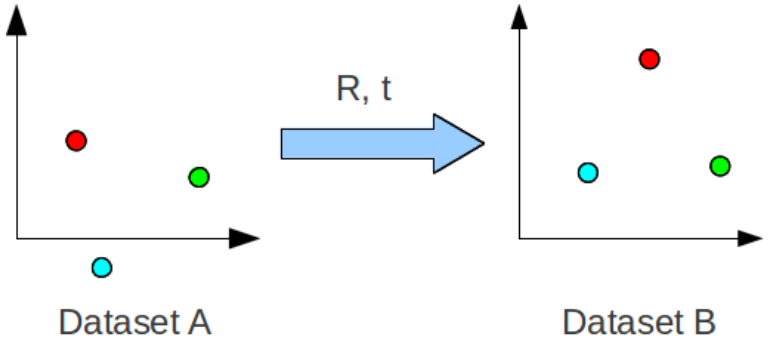


Figura 3.5: Ricerca della rotazione R e traslazione t ottime per il riallineamento di *point cloud*.

Questa trasformazione può essere anche chiamata euclidea o rigida, poiché conserva la forma e la dimensione. Questo è in contrasto con una trasformazione affine, che include lo *scaling* e lo *shearing*.
La soluzione proposta, consiste nel trovare la rotazione R e la traslazione t nell'equazione :

$$B = R * A + t \tag{3.2}$$

Dove R e t sono le trasformazioni applicate al dataset A per allinearlo nel migliore dei modi rispetto al dataset B. La trasformazione rigida può essere individuata attraverso tre *step* che sono:

- Cercare i centroidi in entrambi i data set.

- Portare i dataset nell'origine come mostrato in fig. 3.2.5 e trovare la matrice di rotazione ottima R.
- Trovare la traslazione t.

I centroidi possono essere calcolati come segue:

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.3}$$

$$centroid_A = \frac{1}{N} \sum_{i=1}^{N} {P_A}^i \tag{3.4}$$

$$centroid_B = \frac{1}{N} \sum_{i=1}^{N} {P_B}^i \tag{3.5}$$

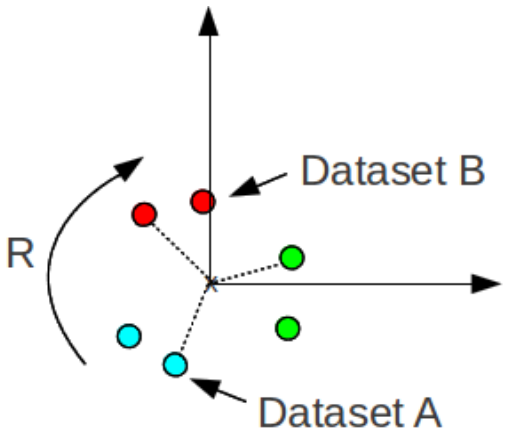
Dove, P_A e P_B sono rispettivamente i punti contenuti nel dataset A e B.
Per ottenere la rotazione ottimale è stato implementato SVD[2] (la decomposizione ai valori singoli è utilizzata in algebra lineare, è una particolare fattorizzazione di una matrice basata sull'uso di autovalori e autovettori). Attraverso questo strumento è possibile scomporre una matrice reale o complessa E di dimensioni mxn, in altre tre matrici, in modo tale che:

$$\Big[U, S, D\Big] = SVD(E) \tag{3.6}$$

$$E = USV^T \tag{3.7}$$

In particolare se E è una matrice quadrata, allora anche U, S e V hanno la stessa dimensione. Per trovare quindi la rotazione ottima si riallineano entrambe le nuvole di punti rispetto all'origine come mostrato in fig. 3.2.5, questo permette di rimuovere la traslazione lasciando quindi solo la rotazione incognita.

[2]Singular Value Decomposition



É possibile quindi trovare la matrice H da decomporre con SVD come segue:

$$H = \sum_{i=1}^{N} ({P_A}^i - centroid_A)({P_B}^i - centroid_B)^T \left[U, S, D\right] = SVD(H) \tag{3.8}$$

$$R = VU^T \tag{3.9}$$

Tuttavia è possibile notare un caso singolare in cui SVD può restituire una matrice corretta ma che non può accadere nella realtà, questo può essere risolto controllando il determinante della matrice R e, se negativo, la terza colonna di V può essere moltiplicata per -1.
Rimane adesso da calcolare la traslazione che può essere descritta come :

$$t = -R \times centroid_A + centroid_B \tag{3.10}$$

dove t è un vettore di dimensione $3x1$. Il segno meno indica che abbiamo spostato i punti di A nell'origine come mostrato in fig. 3.2.5 e la matrice R è la rotazione calcolata al passo precedente che permette di ruotare i punti. Possiamo infine traslare i punti del dataset A nell'origine del dataset B.
Questo tipo di soluzione è possibile se i dataset contengono almeno tre punti. Quando questo vincolo è soddisfatto allora la soluzione ai minimi quadrati può essere trovata,

tale da minimizzare l'errore:

$$err = \sum_{i=1}^{N} \|RP_A{}^i + t - P_B{}^i\|^2 \tag{3.11}$$

Dove con ||| si indica lo scalare ottenuto come distanza euclidea tra due vettori, *err* indica l'errore quadrato della distanza tra i punti nei due dataset.

3.3 Ottimizzazione della mappa e ri-localizzazione

É stato testato, in fase di sviluppo, la possibilità di ottimizzare offline la mappa e ri-localizzarsi su di essa in un secondo momento, facendo partire il drone da una posizione casuale.
In particolare, per l'ottimizzazione offline della mappa si è utilizzato g^2o[3] [75], un *ramework* C++ *open-source* sviluppato presso l'università di Friburgo da Rainer Kummerle per eseguire l'ottimizzazione di problemi non lineari ai minimi quadrati che possono essere descritti attraverso un grafo o un iper-grafo (un'estensione del concetto di grafo in cui un arco può collegare più nodi e non solo due). Poiché i test con il drone sono stati effettuati in ambienti differenti, l'utilizzo di questo software risulta poco pratico per i nostri scopi e quindi tale approccio è stato abbandonato.
Siccome la missione può partire da un punto a caso della mappa, e gli algoritmi classici che impiegano il brute force per determinare il keyframe più vicino al frame corrente risultano poco efficienti è stato deciso di testare l'algoritmo DBoW2. Tale approccio è stato realizzato da Dorian Gàlvez-Lòpez e Juan D. Tardòs descritto in [31] è una libreria C++ *open source* per l'indicizzazione e la conversione di immagini in una rappresentazione *bag-of-word* sfruttando gli alberi di Chow–Liu, prevede per l'utilizzo una fase offline per la creazione del dizionario partendo dalle feature associate a ogni keyframe.
In fig. 3.6 è riportato il test che confronta questo algoritmo (linea verde) con un classico *brute force* (linea rossa). É possibile notare come l'algoritmo DBoW2 riesca

[3] g^2o - General Graph Optimization

a trovare il frame in meno di 20ms al contrario di un *brute force* che impiega non meno di 80ms.

Figura 3.6: Tempi di elaborazione dell'algoritmo DBoW2 (verde) confrontato con un generico *brute force* (rosso).

3.4 Adattamento a un modello sferico

L'algoritmo presentato può essere adattato anche per modelli differenti dal pinhole, ad esempio usando il modello sferico descritto in 2.2. L'utilizzo di tale modello permetterebbe l'uso di ottiche con un FoV molto più ampio con la possibilità di inquadrare più porzioni dell'ambiente circostante, permettendo quindi di ridurre il numero di keyframe da salvare.
La parte che richiede modifiche è il calcolo della matrice fondamentale poiché l'algoritmo implementato sfrutta ottimizzazioni per il modello pinhole. Come descritto da Sarthak Pathak et al. in [76] è possibile stimare la matrice fondamentale utilizzando una minimizzazione ai minimi quadrati. In particolare, la quantità da minimizzare è fortemente dipendente dalla geometria sferica. Col modello pinhole, e in particolare

nell'algoritmo la quantità minimizzata è la distanza punto-retta epipolare sul piano immagine. Con la geometria sferica questa distanza si traduce in un angolo tra il punto in coordinate omogenee sulla sfera e la circonferenza individuata tra l'intersezione del piano generato dal punto omologo e la sfera stessa. Minimizzando questa quantità è possibile calcolare una rototraslazione a meno di un fattore di scala con cui è possibile selezionare gli *inlier* tra le possibili corrispondenze generate dal feature matcher. Per la gestione degli *outlier* nella minimizzazione si può utilizzare un least square pesato [77] che adatta i residui a seconda del modello stimato.

3.5 Porting su chip

L'algoritmo presentato è stato infine portato sul chip descritto in 1.3 sviluppato dall'azienda per la quale ho svolto il dottorato industriale. All'interno del chip vi sono sostanzialmente due parti per effettuare i calcoli: c'è una zona ARM estremamente flessibile ma non velocissima e una parte molto veloce ma non flessibile adibita ad effettuare innumerevoli volte la stessa operazione su grandi moli di dati omogenei. Le applicazioni possono essere individuate attraverso dei grafi aciclici orientati DAG[4], è quindi importante individuare l'ordine con cui le applicazioni rendono disponibili i dati poiché un applicazione in mezzo al grafo potrà inviare un messaggio alla prima applicazione solo al ciclo successivo quando tutte le altre applicazioni dopo di lui saranno state eseguite. In fig. 3.8 è mostrato come è stato spezzato l'algoritmo, in particolare la parte di feature extractor, di rotazione delle feature e del *matching* delle stesse non è stato effettuato su ARM ma sulla parte dedicata del chip.
Per garantire l'esecuzione a 30fps dell'algoritmo (in fase di sviluppo la velocità massima raggiungibile con i 3DV-E era 10fps) e per adattare l'algoritmo alla struttura del chip sono state effettuate delle modifiche. In particolare, sono state cambiate le feature utilizzate e il metodo con cui viene calcolata la disparità. Si è scelto di utilizzare come estrattore di feature Harris Corner Detector e come descrittore binario BRIEF. Harris Corner Detector è un corner detector ampiamente utilizzato nella visione artificiale per estrarre feature è stato introdotto da Chris Harris e Mike Stephens nel

[4]DAG - Directed acyclic graph

1988 come miglioramento dell'algoritmo di Moravec's [78]. BRIEF come descritto in [71] è un descrittore binario per feature, ha il vantaggio di essere particolarmente veloce ed entrambi si prestano per essere eseguiti su chip. Questo tipo di feature non è invariante alla scale e alla rotazione come le feature ORB e AKAZE testate in fase di sviluppo. Per attenuare questo problema si è discretizzato in sedici parti lo *yaw* del drone come mostrato in fig. 3.7, e si è individuato lo spicchio in cui il drone si trova effettuando una differenza tra lo *yaw* ottenuto dal filtro di localizzazione nel frame corrente, con lo *yaw* associato nel keyframe selezionato. Si è deciso di suddividere l'angolo in sedici porzioni poiché nei test effettuati risultava essere un buon compromesso tra efficienza e velocità di calcolo.

Figura 3.7: Mostra come l'angolo imbardata (yaw) del drone sia stato quantizzato 16 sezioni.

Ottenuto l'angolo si effettua quindi uno *shift*, dei descrittori associati alle feature del frame corrente in modo da potere riprendere l'algoritmo nelle sue fasi.
Anche l'euristica per il salvataggio dei keyframe è stata modificata, aggiungendo i seguenti controlli:

- Il primo frame viene memorizzato a 40cm.
- I frame vengono memorizzati quando il numero di feature tra un frame e quello successivo è inferiore a una determinata soglia.
- La distanza tra un frame e l'altro è maggiore di 2m.

Come descritto nel paragrafo precedente, per la validazione dei punti in fase di sviluppo veniva utilizzata una disparità densa poiché già calcolata dallo stesso 3DV-E che acquisiva le immagini stereo.
Nel chip, per ridurre il numero di operazioni e incrementare l'efficienza si è utilizzata una disparità sparsa, calcolandola solo per le feature che vengono estratte al passo precedente. La disparità associata alle feature viene quindi salvata nel caso di un keyframe in modo da non effettuare conti multipli. Infine per la ri-localizzazione è stato portato DBoW2 sul chip, ma i tempi impiegati per la localizzazione risultavano elevati e quindi si è deciso di fare partire il drone da un punto noto della mappa e testare tutte e sedici gli orientamenti possibili, poiché il sistema girando a 30fps impiega meno di un secondo per validare tutte le possibilità.

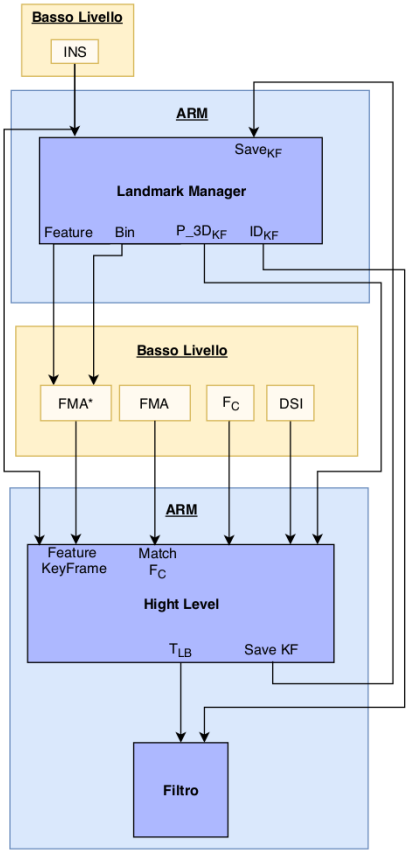


Figura 3.8: Mostra la suddivisione dell'algoritmo per il porting nel chip.

3.6 Risultati

Per valutare l'accuratezza del sistema sviluppato, sono stati effettuati dei test *outdoor* in modo da utilizzare il GPS come ground truth. Durante la fase dei test il drone è stato incaricato di decollare da una determinata base e portarsi ad una quota predefinita, che nello specifico era di due metri e attraversare quattro *waypoint* in un ordine specifico per poi terminare la missione atterrando nel punto di partenza. Un esempio di tale test è mostrato in fig. 3.9, in questo caso il GPS che rappresenta il *ground truth* è rappresentato dal colore rosso e in nero la posizione calcolata dal filtro di localizzazione. Riguardo a quest'ultimo, è possibile osservare un numero di quadrati rossi che corrispondono ai keyframe acquisiti dall'algoritmo sviluppato e utilizzati per inviare le correzioni al filtro, rappresentate in giallo.
Un dettaglio del risultato di localizzazione, preso nella corrispondenza della posizione di decollo e di atterraggio è mostrato in fig. 3.11. In questo caso viene mostrato che l'errore descrivibile come la distanza tra il punto di decollo e di atterraggio è inferiore ai 15cm. Alcuni test comparativi sono stati effettuati tra alcuni input del filtro di localizzazione e il risultato complessivo della stima. Ad esempio in fig. 3.11 viene mostrato come la visual odometry da sola possa portare a risultati errati a causa di una deriva significativa nella stima della posa. Al contrario, sfruttando altre informazioni visive il filtro è in grado di eseguire una stima corretta, come mostrato dalla linea nera. Altri test come illustrato nella fig. 3.12 mostrano come il drone riesce a mantenere l'altitudine richiesta e come i picchi del VO vengono compensati dal filtro attraverso l'uso degli altri dati tra cui i keyframe. Notiamo anche come il sistema è più affidabile del GPS nel fornire la posizione durante la fase di atterraggio.

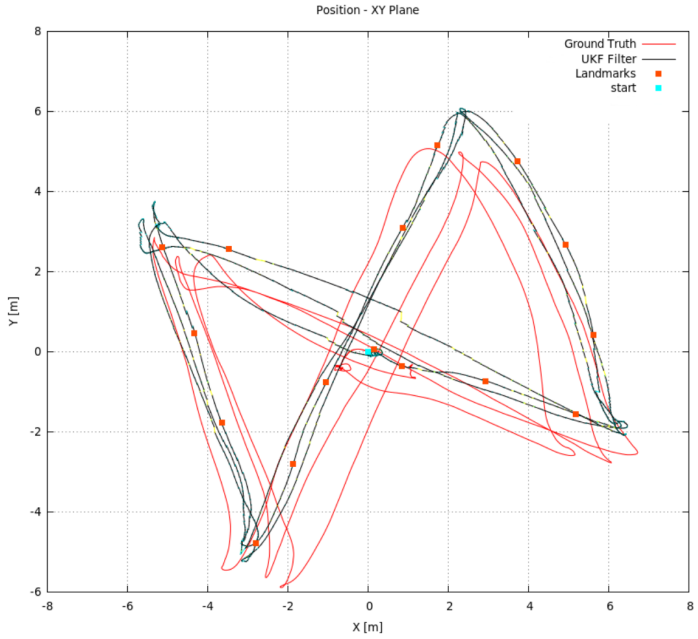


Figura 3.9: Mostra un test sulla localizzazione, in particolare il punto di partenza è rappresentato dal quadrato azzurro, in rosso è rappresentato il GPS utilizzato come ground truth e in nero il filtro di localizzazione. I quadrati rossi rappresentano i keyframe mentre in giallo sono rappresentate le correzioni.

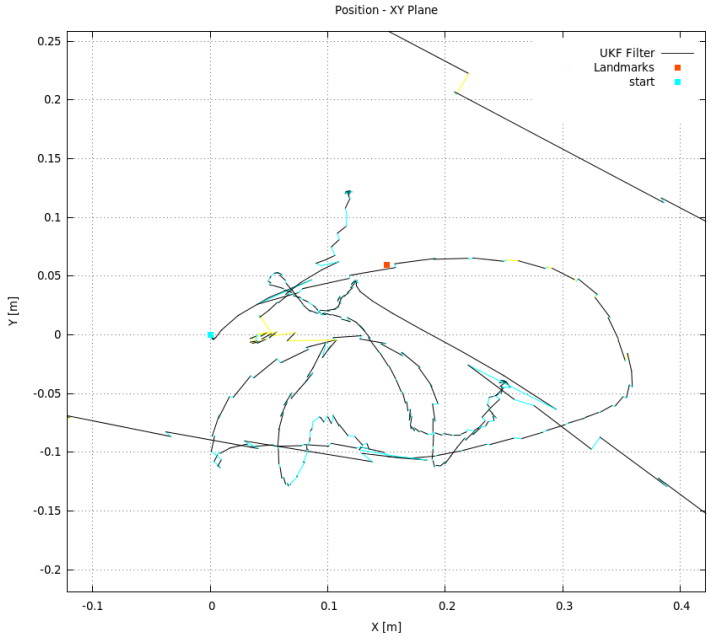


Figura 3.10: Dettaglio del test rappresentante la fase di decollo (quadrato azzurro) e atterraggio.

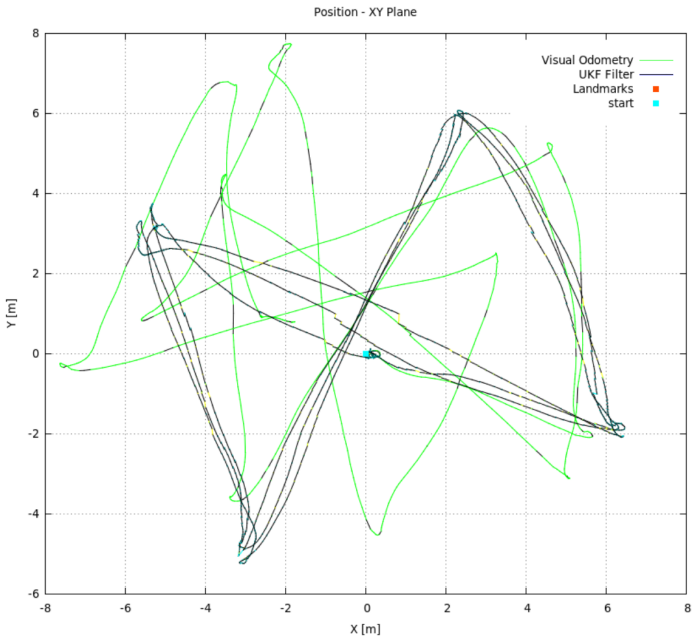


Figura 3.11: Confronto tra visual odometry (verde) e filtro di localizzazione (nero).

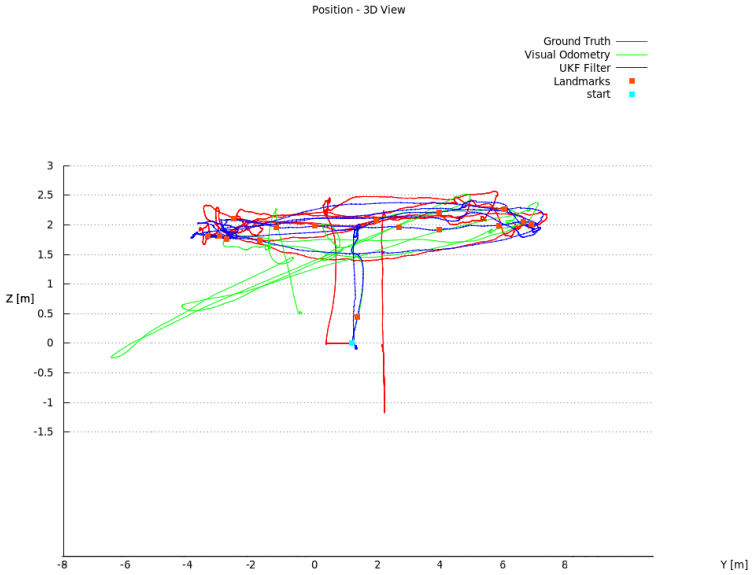


Figura 3.12: Dettaglio laterale, in verde è mostrata la traccia compiuta dalla visual odeometry, in rosso il GPS usato come *ground truth* e in blu la traccia del filtro. I quadrati rossi rappresentano la posizione dei keyframe utilizzati dal filtro per correggersi.

Capitolo 4

Obstacle Detector

L'identificazione degli ostacoli nel settore *automotive* risulta ancora oggi un problema aperto anche se in letteratura si trovano diverse proposte che garantiscono una accurata percezione del mondo esterno, incrementando il livello di sicurezza del veicolo. Come descritto nello stato dell'arte gli approcci proposti non sono utilizzabili su chip low power e sempre più spesso la soluzione adottata per ridurre i calcoli è l'utilizzo di mappe di disparità sparsa, poiché attraverso questo approccio si riduce il numero di punti da elaborare, a scapito di una ricostruzione 3D più accurata.
In questo capitolo viene descritto un algoritmo per chip a basso consumo per l'identificazione di ostacoli fissi e mobili che sfrutta una disparità 4K densa. Il capitolo è suddiviso in tre sezioni, nella prima viene descritto il sistema in cui è stato testato l'algoritmo durante la fase di sviluppo.
La seconda sezione descrive l'approccio nelle sue diverse parti e nel terzo sono mostrati i risultati ottenuti.

4.1 Ambiente di sviluppo

L'algoritmo durante la fase di sviluppo è stato testato su delle sequenze registrate con delle camere adibite alla visione di medio-lungo raggio poste sul tetto del veicolo autonomo EVA[1] fig. 4.1. Questo veicolo è stato realizzato dall'azienda per la quale si volge il dottorato di ricerca, la tecnologia integrata nel mezzo garantisce un'automazione di livello 5 dello standard introdotto dalla SAE International Automotive, ovvero prevede la piena gestione autonoma di accelerazione, frenata, direzione e controllo traffico. Nello specifico la coppia stereo utilizzata ha un frame rate di 30fps e una risoluzione di 4K, e permette di ricostruire la scena 3D dinanzi al veicolo, l'obbiettivo dell'algoritmo sviluppato è quindi identificare tutti gli ostacoli in tempo reale sfruttando la mappa di disparità densa.

Figura 4.1: Eva veicolo autonomo realizzato da Vislab.

Nella fase di sviluppo si è utilizzato il framework GOLD[2], sistema proprietario di Vislab, il quale mette a disposizione strumenti per la realizzazione di applicazioni

[1]EVA - Embedded Vehicle Autonomous
[2]GOLD - General Obstacle and Lane Detection

di Visione Artificiale. Infine dopo lo sviluppo è stato testato su alcuni dataset di immagini acquisiti dalla flotta di veicoli del Vislab, le cui caratteristiche sono riportate nel paragrafo 4.3.

4.2 Algoritmo

L'algoritmo sviluppato restituisce in output tutti gli ostacoli individuati, partendo da una immagine catturata dalla coppia stereo e dalla mappa di disparità densa associata. I passi svolti dall'algoritmo sono:

- Calcolo della V-Disparity.
- Segmentazione della strada utilizzando VLDH[3].
- Calcolo della Advanced U-Disparity (AUD).
- Clustering e Tracking degli ostacoli con feature.

4.2.1 V-Disparity

Questo approccio introdotto da Labayrade [43] la v-disparity, viene calcolata accumulando i pixel aventi lo stesso valore di disparità sull'asse "u" dell'immagine lungo le righe.
Come mostrato in fig. 1.5, dato un pixel P di coordinate (v_p, δ_p), l'intensità, o valore, del pixel P corrisponde al numero di pixel della riga v_p della mappa di disparità δ_p. Grazie all'accumulo dei valori, la v-disparity è robusta al rumore che è presente nella normale mappa di disparità. Questo risultato è molto utile per potere facilmente identificare il piano stradale e gli ostacoli posti sullo stesso. Gli ostacoli presenti sul piano stradale sono rappresentati da linee verticali. Utilizzando le tecniche per la rilevazione di linee nelle immagini, come ad esempio la trasformata di Hough [79], è possibile determinare le equazioni della retta che rappresenta il piano stradale.
All'interno dell'algoritmo si è utilizzato questo approccio poiché adatto a essere eseguito su chip; in particolare l'informazione estratta è l'andamento della strada che

[3]VLDH - Vertically Local Disparity Histogram.

può essere facilmente dedotto prendendo per ogni riga della V-Disparity il massimo e cercando su di essi una retta.

4.2.2 Segmentazione della strada utilizzando VLDH

Per separare ostacoli dal terreno è stato implementato un algoritmo, partendo dal concetto di VLDH descritto da Shinji Kakegawa et al. [80].
L'algoritmo descritto in [80] prevede la realizzazione di un istogramma partendo dalla mappa di disparità, basandosi sull'equazione 4.1.

$$VLDH(u, v, \bar{d}) = \sum_{j=0}^{N-1} \delta(D(u, v + j), \bar{d}) \tag{4.1}$$

VLDH è definito per ogni punto dell'immagine (u, v) e per ciascun valore di disparità quantizzato d. N indica il numero di pixel del "vicinato" preso in esame, questo dato è un parametro del sistema e generalmente determina la minima dimensione degli ostacoli presi in esame. Infine δ rappresenta la delta di Kronecker e vale 1 se la disparità $D(u, v + j)$ è contenuta in $\bar{d}$, altrimenti ritorna 0.
Il valore ottenuto quindi per ogni pixel è massimo quando ci si trova in prossimità di un ostacolo poiché i pixel nella finestra avranno lo stesso valore. Rispetto alla classica U-Disparity il punto di forza di questo algoritmo riguarda la conservazione dell'informazione che identifica la posizione in coordinate immagine del confine tra ostacolo e strada. Infine l'algoritmo termina con l'individuazione per ogni colonna del valore massimo, consentendo di eliminare tutti gli altri valori nella colonna e ricavare un'immagine di output che evidenzia i confini tra terreno e il primo ostacolo per ogni coordinata dell'immagine.
Partendo da questo concetto, si è creato un algoritmo con le seguenti ottimizzazioni:

- L'algoritmo è stato eseguito solo su una porzione dell'immagine individuata attraverso la V-Disparity calcolata precedentemente.
- Tutti i punti individuati dall'algoritmo ma che sono identificati dalla V-Disparity come terreno sono stati eliminati, questo ha permesso di eliminare un buon numero di falsi.

- Sono stati conservati per ogni colonna tutti i massimi in modo da evidenziare non solo il confine col terreno ma anche la forma dell'ostacolo.

Applicando queste ottimizzazioni è stata prodotta in uscita dall'algoritmo una maschera binaria che identifica la posizione in coordinate immagine degli ostacoli rilevati come mostrato in fig. 4.5.

4.2.3 Calcolo della Advanced U-Disparity

L'ultimo step dell'estrazione degli ostacoli è il calcolo della AUD[4].
In questo passaggio, le maschere calcolate negli step precedenti vengono utilizzate unitamente ad una versione modificata di U-Disparity, per la classificazione di ogni pixel di un immagine come appartenente ad un ostacolo o al terreno.
L'approccio della U-Disparity è la versione duale della V-Disparity, in cui però l'accumulazione della disparità con valori simili è ottenuta lungo le colonne: l'istogramma ottenuto avrà quindi la stessa larghezza dell'immagine di input, e altezza equivalente al massimo valore di disparità.
La versione sviluppata, data la mappa di disparità, genera in output una griglia di dimensioni $width$ e $height$ arbitrarie. Le ricorrenze vengono accumulate analizzando la mappa di disparità per colonne come nella classica U-Disparity. Mentre i corrispondenti limiti sull'asse delle ascisse sono determinati utilizzando la funzione 4.2.

$$v_{grid} = \frac{\left(log^{\frac{d}{dsi_{min}}}\right)^{k}}{\frac{\left(log^{\frac{dsi_{max}}{dsi_{min}}}\right)^{k}}{N_{row}}} \tag{4.2}$$

In questa funzione, d indica la disparità relativa al pixel in esame con coordinate (u, v) nella mappa di disparità, dsi_{max} e dsi_{min} sono rispettivamente il valore massimo e minimo che la disparità può assumere, N_{row} è il numero massimo delle righe della griglia. Infine k è un parametro del sistema scelto a priori che permette di modificare

[4]AUD - Advanced U-Disparity

l'andamento della funzione 4.2, privilegiando o penalizzando alcuni parti del dominio di disparità, come mostrata in fig. 4.2, in cui nelle ordinate ci sono i valori di disparità e nelle ascisse la relativa riga nella griglia di accumulazione.

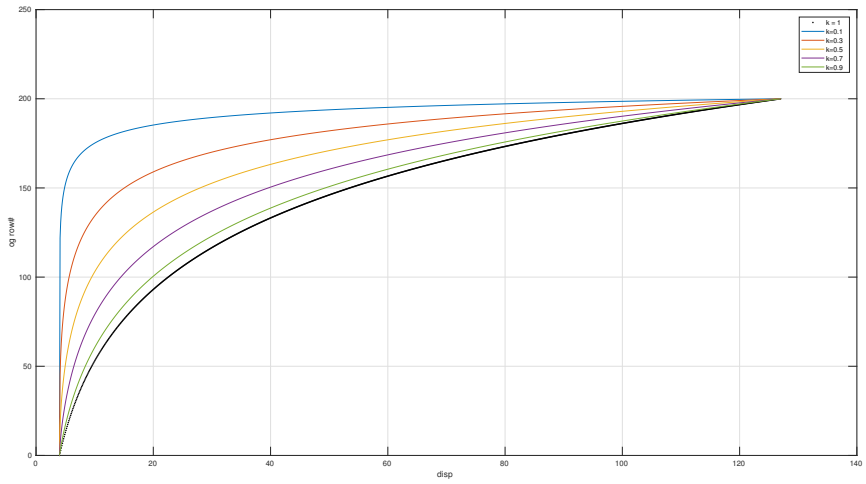


Figura 4.2: Descrive come varia la curva al variare del parametro k, dove le ordinate rappresentano la disparità e le ascisse le righe della griglia di accumulazione.

Durante la fase di riempimento della griglia la maschera generata dallo step precedente viene utilizzata per pesare le celle attuali, in particolare, i punti classificati come spazio libero vengono penalizzati in modo da ridurre i falsi positivi evidenziati dal sistema.
Come ultimo passo, ogni cella viene classificata come ostacolo/non ostacolo valutando il valore al suo interno rispetto alla funzione 4.3.

$$n_{min} = \frac{Y_{3dmin}}{B} * d \tag{4.3}$$

In 4.3, Y_{3dmin} è l'altezza minima in metri degli ostacoli che si vogliono individuare, B è la baseline del sistema stereoscopico e d è il valore medio di disparità associato

alla cella.
Da notare che per il paramento Y_{3dmin} è necessario trovare un compromesso tra precisione e stabilità del sistema. Un'altezza minima troppo piccola provoca a grandi distanze molti falsi positivi essendo la disparità affetta da rumore. Per questo motivo si è deciso di utilizzare due parametri differenti per Y_{3dmin}, in modo da cercare vicino al veicolo oggetti piccoli (3cm), mentre con una soglia più alta (30cm) è possibile eliminare il numero di falsi positivi a grandi distanze.

4.2.4 Clustering e Tracking

Ottenuta la griglia si procede al raggruppamento degli elementi omogenei su di essa, vengono cosi creati dei gruppi partendo dal pixel in alto a sinistra e valutando il vicinato lungo le otto direzioni, si procede quindi muovendosi a macchia d'olio, un esempio di griglia clusterizzata è raffigurata in fig. 4.9.
Vengono, poi, estratte dall'immagine di input sulla quale è stata calcolata la disparità, le feature ottenute attraverso HARRIS e BRIEF e queste vengono utilizzate per validare i cluster della griglia e proiettarli sull'immagine fig. 4.8.
Temporalmente vengono associati i cluster tra di loro valutando:

- Il numero di pixel contenuti in ognuno di essi.
- La posizione del centroide di ogni cluster.

4.3 Test

Per valutare l'accuratezza del sistema è stata acquisita una sequenza con il veicolo EVA di Vislab fig. 4.1, in cui oltre alle immagini registrate con la coppia stereo montata sopra al veicolo con una risoluzione di 4K e una baseline di 30cm, sono stati acquisiti anche i dati relativi alla posizione del veicolo con il GPS Aplanix.
Questo GPS differenziale permette post processando i dati, attraverso dei loro sistemi, di ottenere un'accuratezza del cm sulla posizione.
In particolare, come mostrato in fig. 4.3, sono stati posti alcuni oggetti di altezza nota uno accanto all'altro e allontanandosi con il veicolo sono stati raccolti i dati. La misura del GPS è servita offline per determinare la distanza massima a cui gli oggetti sono visibili dall'algoritmo sviluppato.
Il test si è svolto registrando come punto iniziale la posizione del GPS in cui l'auto era appoggiata agli ostacoli, si è poi proceduto ad allontanarsi dagli ostacoli in retromarcia fino a quando gli ostacoli non erano più identificati dall'algoritmo e quindi si è confrontata la distanza ottenuta con il GPS con la misura ottenuto dall'approccio sviluppato.
Gli oggetti inseriti nel test sono:

- Tre coni stradali arancioni alti 30cm.
- Una bomboletta spray di 6cm di diametro e 19cm di altezza.
- Un metro alto 3cm e largo 7cm.
- Una scatola alta 43cm e larga 19cm.
- Una scatola di scotch alta 7cm e larga 22cm.
- Un cestino della carta alto 110cm 58cm.

Come mostrato dalla fig. 4.4 i coni sono visibili fino a una distanza max di 92m, mentre gli altri oggetti "più piccoli" iniziano ad essere rilevati ad una distanza di 40m fig. 4.5. In fig. 4.6 è mostrata la distanza a cui viene rilevato un pedone e il cestino della carta alti rispettivamente 1.70m e 1.10m, la distanza rilevata è di 130m.

In fig. 4.7 è mostrata un esempio dell'immagine calcolata attraverso l'algoritmo VLDH opportunamente modificato e descritto in 4.2.2 in bianco sono rappresentati gli ostacoli che questo approccio riesce ad identificare. Si può notare come gli ostacoli molto piccoli ad esempio "il marciapiede" vengano solo descritti da pochi punti e invece come mostrato in fig. 4.6 l'utilizzo di AUD 4.2.3 riesca a identificare in maniera precisa tutto il marciapiede.
In fig. 4.8 è mostrato la validazione dei cluster attraverso le feature (punti rossi): è possibile osservare come i cluster neri che sono i falsi generati dall'algoritmo vengano filtrati attraverso l'utilizzo delle features. Tuttavia questa fase di *clustering* mostra come gli oggetti tendano ad essere uniti e difficilmente separabili.
Infine, in fig. 4.9 è mostrata la griglia generata attraverso la AUD, in cui sono visibili i diversi cluster degli ostacoli individuati.

Figura 4.3: É raffigurata la configurazione utilizzata nei test per valutare l'individuazione degli ostacoli.

Figura 4.4: É raffigurata la distanza massima a cui sono visibili i coni stradali alti 30cm.

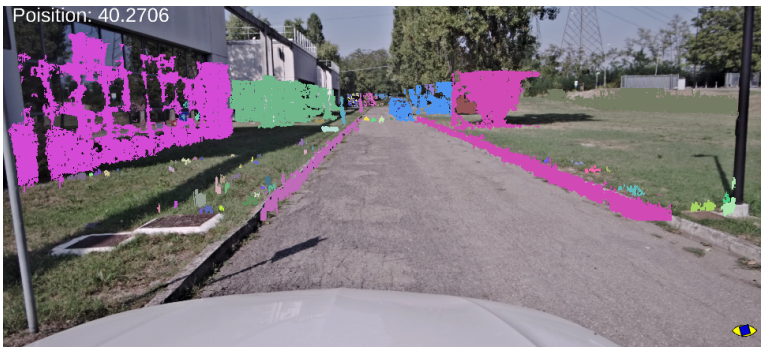


Figura 4.5: É raffigurata la distanza massima a cui sono visibili gli oggetti con altezza compresa tra i 7cm e 22cm.

Figura 4.6: É raffigurata rilevazione di un pedone alto 1,70m e un cestino della carta alto 1,10m.

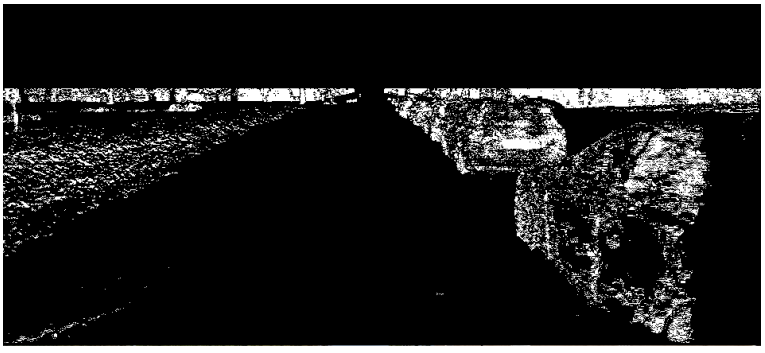


Figura 4.7: Output dell'algoritmo VLDH opportunamente modificato come descritto in 4.2.2.

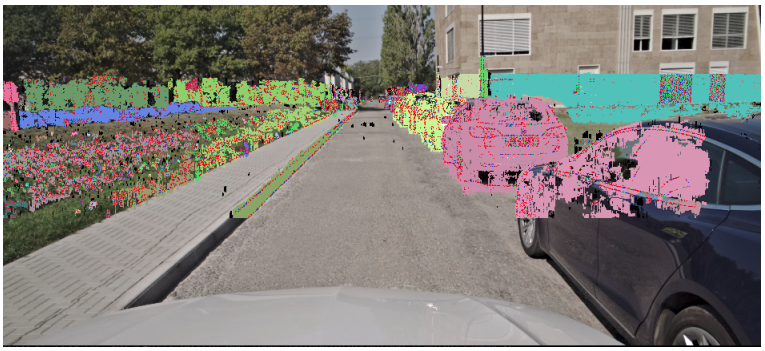


Figura 4.8: Sono raffigurati i cluster validati con le feature (punti rossi), è possibile notare come i cluster neri vengano eliminati, poiché privi di feature.

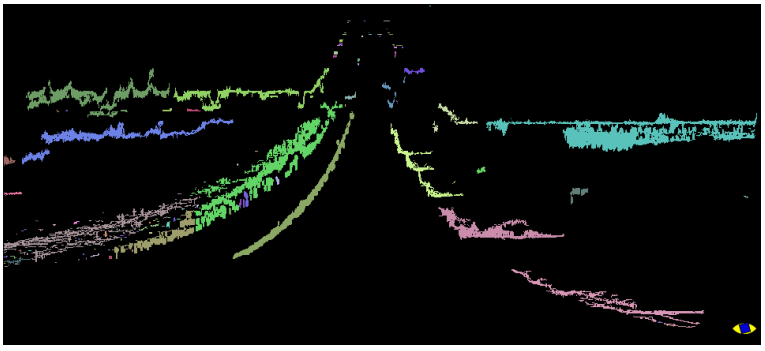


Figura 4.9: Griglia occupazionale realizzata attraverso l'algoritmo AUD 4.2.3.

Capitolo 5

Conclusioni

In questa tesi sono stati affrontati lo studio e la realizzazione di algoritmi per agenti autonomi che, sfruttando le informazioni provenienti da un'immagine di profondità, possano garantire le prestazioni necessarie alla guida autonoma pur essendo atte all'esecuzione su un processore a basso consumo, come richiesto dalle applicazioni in cui potrà essere utilizzato.
Sono state in particolare affrontate due tematiche: la localizzazione di un drone autonomo in ambiente non strutturato e assenza di GPS, con requisiti di precisione che permettessero l'atterraggio e il decollo da alcune basi di un metro quadrato e portando particolare attenzione alla correzione della deriva tipico dell'integrazione di informazioni di velocità e la rilevazione di ostacoli per veicoli, avendo come requisito un'altezza minima di 10 cm e un distanza di rilevamento paragonabile a quella posseduta da altri sensori commerciali, come ad esempio un radar. Entrambi i problemi in letteratura vengono affrontati e risolti attraverso algoritmi computazionalmente pesanti che necessitano di hardware caratterizzati da consumi energetici elevati per poter soddisfare i requisiti di precisione e accuratezza richiesti dalle applicazioni in oggetto.
Gli algoritmi proposti e sviluppati nel corso del dottorato sono stati successivamente implementati su un chip a basso consumo sviluppato dall'azienda Ambarella, evidenziando come le loro prestazioni rimangano paragonabili a quelle prodotte dai loro

alter-ego su piattaforma PC standard.
Nello specifico gli algoritmi sviluppati:

- Keyframe Localization
- Rilevazione degli Ostacoli

Entrambi gli approcci utilizzano in modo denso o sparso la disparità di immagini a 3840x1728 px a una frequenza di 30fps.
I test effettuati sull'algoritmo di Keyframe Localization mostrano, come descritto in 3.6, come il drone riesca ad effettuare una manovra di *homing*, ovvero a ritornare e atterrare nel punto di decollo in maniera autonoma, con un errore di posizionamento inferiore ai 10cm, e in particolare come la soluzione adottata in partenza, che prevedeva il solo utilizzo del'integrazione di sensori odometrici, sia visuali che inerziali, risultasse insufficiente al compimento della missione.
L'implementazione su chip del'algoritmo Keyframe Localization ha richiesto alcuni adattamenti: in primo luogo, data la mancanza, per ragioni di complessità computazionale, di *feature descriptor* invarianti alla rotazione, si è reso necessaria l'implementazione di un metodo che permettesse il confronto fra i descrittori di frame inquadranti le stesse feature ma con rotazioni diverse. L'invarianza a posteriori è stata ottenuta, come descritto nel paragrafo 3.5, mediante la rotazione dei descrittori secondo un angolo canonico, estratto dal giroscopio/magnetometro di cui il drone era dotato.
In secondo luogo, al fine di ottimizzare l'uso della memoria all'interno del chip, si è passati da una disparità densa a una sparsa, calcolando la disparità solo per punti ritenuti interessati durante la fase di *matching* delle feature. Per quanto riguarda l'algoritmo di identificazione degli ostacoli, l'approccio presentato mostra come sia possibile utilizzare mappe di disparità dense su chip a basso consumo. L'utilizzo di mappe dense è fondamentale per garantire una ricostruzione dettagliata e precisa per la rilevazione degli ostacoli nella guida autonoma. L'impiego di una V-Disparity e del'algoritmo VLDH 4.2.2 opportunamente modificato vengono combinati per creare una maschera che viene utilizzata nella fase di creazione di AUD. I test sono stati effettuati impiegando un sistema stereo con una risoluzione 3840x1728 a 30fps, il quale restituisce una nuvola di punti in grado di garantire una ricostruzione accurata della

scena di fronte al veicolo. I risultati in 4.3 mostrano come le distanze massime a cui si possono rilevare ostacoli piccoli siano elevate, il che rende l'approccio utilizzabile nella guida autonoma.
Concludendo, nel corso di questa tesi si è studiata la mappa di disparità sparsa o densa generata da un sistema stereoscopico, formato da una coppia di telecamere in grado di restituire la posizione tridimensionale dei punti mondo osservati. Sono stati quindi sviluppati due algoritmi differenti che utilizzano la disparità in modo diverso per compiere due compiti quali la localizzazione di un drone in ambiente non strutturato e l'identificazione di ostacoli per un veicolo. Si è dimostrato infine come attraverso una implementazione ad-hoc di algoritmi per chip a basso consumo abiliti il loro utilizzo nella guida autonoma soddisfacendo tutti i requisiti richiesti da questo tipo di ambiente. Si noti come le difficoltà maggiori durante l'implementazione sono state causate da:

- operatori/feature mancanti per limiti computazionali
- memoria limitata o generalmente lenta
- requisiti di hard real time

In particolare questi ultimi due punti sono da tenere in particolare considerazione, in quanto rappresentano il maggiore ostacolo per l'adozione di algoritmi *general purpose* in fase di produzione di un veicolo autonomo.

Bibliografia

[1] Florin Oniga, Ervin Sarkozi, and Sergiu Nedevschi. Fast obstacle detection using u-disparity maps with stereo vision. In *Intelligent Computer Communication and Processing (ICCP), 2015 IEEE International Conference on*, pages 203–207. IEEE, 2015.

[2] David Pfeiffer and Uwe Franke. Towards a global optimal multi-layer stixel representation of dense 3d data. In *BMVC*, volume 11, pages 51–1, 2011.

[3] Yali Li, Shengjin Wang, Qi Tian, and Xiaoqing Ding. A survey of recent advances in visual feature detection. *Neurocomputing*, 149:736–751, 2015.

[4] Marius Drulea, Istvan Szakats, Andrei Vatavu, and Sergiu Nedevschi. Omnidirectional stereo vision using fisheye lenses. In *Intelligent Computer Communication and Processing (ICCP), 2014 IEEE International Conference on*, pages 251–258. IEEE, 2014.

[5] A Rathore. State-of-the-art self driving cars. *International Journal of Conceptions on Computing and Information Technology*, 4:1–5, 2016.

[6] Heiko Hirschmüller, Peter R Innocent, and Jon Garibaldi. Real-time correlation-based stereo vision with reduced border errors. *International Journal of Computer Vision*, 47(1-3):229–246, 2002.

[7] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2008.

[8] Mirko Felisa and Paolo Zani. Incremental disparity space image computation for automotive applications. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 345–350. IEEE, 2009.

[9] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. In *Computer Vision–ACCV 2010*, pages 25–38. Springer, 2010.

[10] Deepayan Bhowmik and Kofi Appiah. Embedded vision systems: A review of the literature. 2018.

[11] Stefania Perri, Fabio Frustaci, Fanny Spagnolo, and Pasquale Corsonello. Design of real-time fpga-based embedded system for stereo vision. In *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*, pages 1–5. IEEE, 2018.

[12] Gabriele Camellini, Mirko Felisa, Paolo Medici, Paolo Zani, Francesco Gregoretti, Claudio Passerone, and Roberto Passerone. 3dv—an embedded, dense stereovision-based depth mapping system. In *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, pages 1435–1440. IEEE, 2014.

[13] Johann Borenstein, Hobart R Everett, Liqiang Feng, and David Wehe. Mobile robot positioning: Sensors and techniques. *Journal of robotic systems*, 14(4):231–249, 1997.

[14] Sebastian Thrun, Michael Beetz, Maren Bennewitz, Wolfram Burgard, Armin B Cremers, Frank Dellaert, Dieter Fox, Dirk Haehnel, Chuck Rosenberg, Nicholas Roy, et al. Probabilistic algorithms and the interactive museum tour-guide robot minerva. *The International Journal of Robotics Research*, 19(11):972–999, 2000.

[15] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[16] Simon J Julier and Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition*

VI, volume 3068, pages 182–194. International Society for Optics and Photonics, 1997.

[17] Johann Borenstein and Liqiang Feng. Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on robotics and automation*, 12(6):869–880, 1996.

[18] Udo Frese. A discussion of simultaneous localization and mapping. *Autonomous Robots*, 20(1):25–42, 2006.

[19] Ehab Salahat and Murad Qasaimeh. Recent advances in features extraction and description algorithms: A comprehensive survey. In *Industrial Technology (ICIT), 2017 IEEE International Conference on*, pages 1059–1063. IEEE, 2017.

[20] Tinne Tuytelaars, Krystian Mikolajczyk, et al. Local invariant feature detectors: a survey. *Foundations and trends® in computer graphics and vision*, 3(3):177–280, 2008.

[21] Qiang Liu, Ruihao Li, Huosheng Hu, and Dongbing Gu. Extracting semantic information from visual data: A survey. *Robotics*, 5(1):8, 2016.

[22] Shaojie Shen, Nathan Michael, and Vijay Kumar. Autonomous indoor 3d exploration with a micro-aerial vehicle. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 9–15. IEEE, 2012.

[23] Luke Yoder and Sebastian Scherer. Autonomous exploration for infrastructure modeling with a micro aerial vehicle. In *Field and service robotics*, pages 427–440. Springer, 2016.

[24] Stephen Nuske, Sanjiban Choudhury, Sezal Jain, Andrew Chambers, Luke Yoder, Sebastian Scherer, Lyle Chamberlain, Hugh Cover, and Sanjiv Singh. Autonomous exploration and motion planning for an unmanned aerial vehicle navigating rivers. *Journal of Field Robotics*, 32(8):1141–1162, 2015.

[25] Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. Monocular-slam–based navigation for autonomous micro helicopters in gps-denied environments. *Journal of Field Robotics*, 28(6):854–874, 2011.

[26] Lukas von Stumberg, Vladyslav Usenko, Jakob Engel, Jörg Stückler, and Daniel Cremers. From monocular slam to autonomous drone exploration. In *Mobile Robots (ECMR), 2017 European Conference on*, pages 1–8. IEEE, 2017.

[27] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[28] Syaril Azrad, Mohammad Fadhil, Farid Kendoul, and Kenzo Nonami. Quadrotor uav indoor localization using embedded stereo camera. In *Applied Mechanics and Materials*, volume 629, pages 270–277. Trans Tech Publ, 2014.

[29] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[30] Luigi Musto Andrea Zinelli Alberto Broggi in press Francesco Valenti, Domenico Giaquinto. Enabling computer vision-based autonomous navigation for unmanned aerial vehicles in cluttered gps-denied environments. *IEEE Intelligent Transportation Systems Society Conference Management System*, 2018.

[31] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

[32] Adrien Angeli, David Filliat, Stéphane Doncieux, and Jean-Arcady Meyer. Fast and incremental method for loop-closure detection using bags of visual words. *IEEE Transactions on Robotics*, 24(5):1027–1037, 2008.

[33] P Pinies, LM Paz, D Galvez-Lopez, and JD Tardos. Ci-graph slam for 3d reconstruction of large and complex environments using a multicamera system. *J. of Field Robotics*, 27(5):561–586, 2010.

[34] Mark Cummins and Paul Newman. Appearance-only slam at large scale with fab-map 2.0. *The International Journal of Robotics Research*, 30(9):1100–1123, 2011.

[35] C Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

[36] Michael Calonder, Vincent Lepetit, Pascal Fua, Kurt Konolige, James Bowman, and Patrick Mihelich. Compact signatures for high-speed interest point description and matching. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 357–364. IEEE, 2009.

[37] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[38] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[39] Alberto Broggi, Paolo Grisleri, and Paolo Zani. Sensors technologies for intelligent vehicles perception systems: A comparison between vision and 3d-lidar. In *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*, pages 887–892. IEEE, 2013.

[40] André Gonçalves, André Godinho, and Joao Sequeira. Low cost sensing for autonomous car driving in highways. In *ICINCO-RA (2)*, pages 370–377, 2007.

[41] Thomas Pollard and Joseph L Mundy. Change detection in a 3-d world. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–6. Ieee, 2007.

[42] Kenneth Sebesta and John Baillieul. Animal-inspired agile flight using optical flow sensing. *arXiv preprint arXiv:1203.2816*, 2012.

[43] Raphael Labayrade, Didier Aubert, and J-P Tarel. Real time obstacle detection in stereovision on non flat road geometry through" v-disparity" representation. In *Intelligent Vehicle Symposium, 2002. IEEE*, volume 2, pages 646–651. IEEE, 2002.

[44] Zhencheng Hu and Keiichi Uchimura. Uv-disparity: an efficient algorithm for stereovision based scene analysis. In *Intelligent Vehicles Symposium, 2005. Proceedings. IEEE*, pages 48–54. IEEE, 2005.

[45] Mathias Perrollaz, John-David Yoder, Anne Spalanzani, and Christian Laugier. Using the disparity space to compute occupancy grids from stereo-vision. In *International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[46] Mathias Perrollaz, John-David Yoder, Amaury Nègre, Anne Spalanzani, and Christian Laugier. A visibility-based approach for occupancy grid computation in disparity space. *IEEE Transactions on Intelligent Transportation Systems*, 13(3):1383–1393, 2012.

[47] Basam Musleh, David Martin, Arturo de la Escalera, and Jose Maria Armingol. Visual ego motion estimation in urban environments based on uv disparity. In *Intelligent Vehicles Symposium*, pages 444–449, 2012.

[48] Alexandru Iloie, Ion Giosan, and Sergiu Nedevschi. Uv disparity based obstacle detection and pedestrian classification in urban traffic scenarios. In *2014 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 119–125, 2014.

[49] Alberto Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, 3(3):249–265, 1987.

[50] Manuel Yguel, Olivier Aycard, and Christian Laugier. Efficient gpu-based construction of occupancy girds using several laser range-finders. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 105–110. IEEE, 2006.

[51] Larry Matthies and Alberto Elfes. Integration of sonar and stereo range data using a grid-based representation. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, pages 727–733. IEEE, 1988.

[52] Martim Brandao, Ricardo Ferreira, Kenji Hashimoto, José Santos-Victor, and Atsuo Takanishi. Integrating the whole cost-curve of stereo into occupancy grids. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4681–4686. IEEE, 2013.

[53] Mathias Perrollaz, Anne Spalanzani, and Didier Aubert. Probabilistic representation of the uncertainty of stereo-vision and application to obstacle detection. In *Intelligent vehicles symposium (IV), 2010 IEEE*, pages 313–318. IEEE, 2010.

[54] Hernán Badino, Uwe Franke, and Rudolf Mester. Free space computation using stochastic occupancy grids and dynamic programming. In *Workshop on Dynamical Vision, ICCV, Rio de Janeiro, Brazil*, volume 20. Citeseer, 2007.

[55] Zhengyou Zhang. A stereovision system for a planetary rover: Calibration, correlation, registration, and fusion. *Machine Vision and Applications*, 10(1):27–34, 1997.

[56] Florin Oniga and Sergiu Nedevschi. Processing dense stereo data using elevation maps: Road surface, traffic isle, and obstacle detection. *IEEE Transactions on Vehicular Technology*, 59(3):1172–1182, 2010.

[57] Hernán Badino, Uwe Franke, and David Pfeiffer. The stixel world-a compact medium level representation of the 3d-world. In *Joint Pattern Recognition Symposium*, pages 51–60. Springer, 2009.

[58] J Rebut, G Toulminet, and A Bensrhair. Road obstacles detection using a self-adaptive stereo vision sensor: a contribution to the arcos french project. In *Intelligent Vehicles Symposium, 2004 IEEE*, pages 738–743. IEEE, 2004.

[59] Susumu Kubota, Tsuyoshi Nakano, and Yasukazu Okamoto. A global optimization algorithm for real-time on-board stereo obstacle detection systems. In *Intelligent Vehicles Symposium, 2007 IEEE*, pages 7–12. IEEE, 2007.

[60] Rodrigo Benenson, Radu Timofte, and Luc Van Gool. Stixels estimation without depth map computation. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 2010–2017. IEEE, 2011.

[61] Rodrigo Benenson, Markus Mathias, Radu Timofte, and Luc Van Gool. Fast stixel computation for fast pedestrian detection. In *European Conference on Computer Vision*, pages 11–20. Springer, 2012.

[62] Peter Sturm, Srikumar Ramalingam, Jean-Philippe Tardif, Simone Gasparini, Joao Barreto, et al. Camera models and fundamental concepts used in geometric computer vision. *Foundations and Trends® in Computer Graphics and Vision*, 6(1–2):1–183, 2011.

[63] D.C. Brown. Decentered lens systems. in 29th annual meeting of the american society of photogrammetric engineering. 1965.

[64] A. Conrady. Decentered lens systems. 1919.

[65] Shigang Li. Full-view spherical image camera. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 4, pages 386–390. IEEE, 2006.

[66] Christopher Mei and Patrick Rives. Single view point omnidirectional camera calibration from planar grids. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3945–3950. IEEE, 2007.

[67] Lionel Heng, Bo Li, and Marc Pollefeys. Camodocal: Automatic intrinsic and extrinsic calibration of a rig with multiple generic cameras and odometry. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1793–1800. IEEE, 2013.

[68] Shigang Li. Binocular spherical stereo. *IEEE Transactions on intelligent transportation systems*, 9(4):589–600, 2008.

[69] Shigang Li. Real-time spherical stereo. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 3, pages 1046–1049. IEEE, 2006.

[70] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.

[71] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[72] Pablo F Alcantarilla and T Solutions. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *IEEE Trans. Patt. Anal. Mach. Intell*, 34(7):1281–1298, 2011.

[73] Martin A Fischler and Robert C Bolles. A paradigm for model fitting with applications to image analysis and automated cartography (reprinted in readings in computer vision, ed. ma fischler,". *Comm. ACM*, 24(6):381–395, 1981.

[74] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 586–607. International Society for Optics and Photonics, 1992.

[75] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g 2 o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613. IEEE, 2011.

[76] Sarthak Pathak, Alessandro Moro, Hiromitsu Fujii, Atsushi Yamashita, and Hajime Asama. 3d reconstruction of structures using spherical cameras with small

motion. In *Control, Automation and Systems (ICCAS), 2016 16th International Conference on*, pages 117–122. IEEE, 2016.

[77] Donald B Rubin. Iteratively reweighted least squares. *Wiley StatsRef: Statistics Reference Online*, 2014.

[78] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[79] Tinku Acharya and Ajoy K Ray. *Image processing: principles and applications*. John Wiley & Sons, 2005.

[80] Shinji Kakegawa, Haruki Matono, Hideaki Kido, and Takeshi Shima. Road surface segmentation based on vertically local disparity histogram for stereo camera. *International Journal of Intelligent Transportation Systems Research*, 16(2):90–97, 2018.

Ringraziamenti

Dedico quest'ultimo capitolo della mia istruzione a mia madre che ci ha lasciato troppo presto, ringraziandola per avermi insegnato il rispetto, la determinazione e la dedizione.
Ringrazio mio padre per avermi mostrato il significato della parola "Uomo", mia sorella, mio cognato e al mio nipotino per i sorrisi e le domeniche in famiglia.
Vorrei ringraziare la mia compagna per essermi stata accanto e per avermi spronato in questi anni difficili, con il suo incrollabile sostegno morale.
Un grazie speciale al mio capo squadra e mentore, Stefano D., per avermi aiutato a concludere questo percorso difficile e ai miei colleghi, nonché amici, Matteo, Alessandro Giacomazzo e Alessandro Coati per avere preservato l'equilibrio della mia mente, attraverso l'allenamento quotidiano.
Infine, ringrazio Alberto Broggi e Massimo Bertozzi per avermi concesso l'opportunità di vivere questa esperienza.