Università degli Studi di Milano Bicocca

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di Laurea Magistrale in Informatica

Localizzazione di un veicolo con reti neurali mediante associazione di immagini e mappe LiDAR

Relatore: *Prof.* Domenico G. SORRENTI

Co-relatore: *Dott.* Daniele CATTANEO

Tesi di Laurea Magistrale di:

Matteo VAGHI

Matricola 780915

Anno Accademico 2017-2018

Indice

Capitolo 1

Introduzione

Nel campo della robotica mobile, la localizzazione di un robot all'interno del proprio ambiente operativo costituisce uno dei principali problemi. In ambito automotive, dove per robot si intende un veicolo, la corretta stima della pose (posizione e orientamento) costituisce un problema tutt'oggi ancora aperto e costantemente soggetto a ricerca. Ad esempio, sensori come il GPS non risultano affidabili in ambito urbano ai fini della localizzazione dato il degrado del segnale dovuto alla presenza degli edifici. Un possibile approccio per la stima della pose del veicolo è quello che si basa sull'odometria, cioè la posizione e l'orientamento vengono stimati osservando le misurazioni del movimento delle ruote. Tuttavia la precisione di tale sistema degrada rapidamente a causa dell'accumulo degli errori di misurazione. Ad esempio, al verificarsi di uno slittamento delle ruote rispetto al suolo si ottengono delle misurazioni errate il cui effetto sulla stima della pose si amplifica con il passare del tempo.

Rispetto alla robotica industriale, dove le imprevedibilità dell'ambiente operativo sono limitate, la robotica mobile *outdoor* deve affrontare i problemi legati all'elevata dinamicità dell'ambiente di navigazione e alla presenza di molteplici attori. Ad esempio, i pedoni ed i veicoli costituiscono una delle principali fonti di imprevedibilità che un sistema a guida autonoma deve essere in grado di gestire. Di conseguenza in ambito urbano, conoscere la corretta pose del robot costituisce un problema critico, poiché eventuali errori di localizzazione, durante l'operazione di navigazione, potrebbero mettere in pericolo la vita degli altri utenti della strada e dei passeggeri a bordo del veicolo.

L'obiettivo del lavoro di tesi svolto riguarda lo sviluppo di un approccio innovativo per affrontare il problema di localizzazione di un veicolo a guida autonoma in ambito

urbano, andando ad utilizzare tecniche basate su reti neurali convoluzionali per integrare informazioni di mappe 3D e immagini RGB. In particolare, lo scopo è quello di correggere le informazioni di posizione e orientamento (*pose*) fornite in input al sistema da un elemento esterno come, ad esempio, un sensore GPS. Tale input fornisce una stima della pose del veicolo all'interno del suo spazio operativo, ma ad essa è legata una componente di incertezza che rende la localizzazione poco precisa. La rete neurale implementata ha il compito di stimare l'errore sulla pose iniziale, basandosi sull'informazione immagine ottenuta da una camera montata sul veicolo e dalla porzione di mappa osservata secondo il punto vista ottenuto dalla pose rumorosa in input. Per mappa si intende un insieme di scansioni ottenute mediante un sensore laser come, ad esempio, un LiDAR (Light Detection and Ranging), che forniscono una rappresentazione 3D dell'ambiente operativo del robot. In particolare la scena tridimensionale ottenuta è rappresentata da un insieme di punti, che formano una struttura chiamata point-cloud, a cui sono associate le coordinate 3D delle rilevazioni rispetto a un sistema di riferimento e l'informazione di riflettanza degli elementi scansionati della scena. La riflettanza costituisce il quantitativo di luce riflessa da un oggetto intersecato dal raggio luminoso del sensore laser.

Dato il punto di vista ottenuto dalla pose errata di input, l'informazione di profondità e riflettanza della scena viene proiettata all'interno di un piano immagine, con l'obiettivo di simulare il funzionamento di un sensore camera. L'immagine così ottenuta prende il nome di immagine LiDAR ed è costituita da due canali: uno rappresentante la profondità della scena ed uno rappresentante la riflettanza degli oggetti.

L'immagine RGB ripresa dalla camera e l'immagine LiDAR appena descritta costituiscono l'input della rete neurale convoluzionale sviluppata, che ha l'obiettivo di stimare l'errore sulla pose iniziale mettendo a confronto i due punti di vista forniti dalle due immagini.

Negli ultimi anni, all'interno del campo della computer vision applicata alla robotica mobile, la ricerca si è spostata verso tecniche basate su reti neurali convoluzionali per affrontare una moltitudine di task come, ad esempio, rilevamento di pedoni e veicoli, ricostruzione tridimensionale della scena e localizzazione. Un trend che sta prendendo piede nella comunità scientifica è quello di utilizzare, oltre che alle immagini, anche il dato tridimensionale proveniente da sensori laser come input di una rete neurale convoluzionale. Uno dei metodi che permette di convertire l'informazione 3D in un dato processabile da reti neurali convoluzionali è quello di proiettare all'interno di un piano

immagine le scansioni, in maniera tale da simulare il funzionamento di un sensore camera. Questo processo, però, richiede che vengano affrontate diverse problematiche tra cui, principalmente, il fenomeno della sparsità delle proiezioni dei punti delle scansioni, da cui ne consegue una rappresentazione poco riconoscibile della scena osservata. In letteratura esistono approcci che utilizzano tecniche di filtraggio [32, 33] per aumentare la densità dell'immagine ottenuta. Un'altra possibile tecnica è quella che utilizza la ricostruzione delle superfici di una scena [30] prima di svolgere l'operazione di proiezione.

Per quanto riguarda il problema della localizzazione mediante reti neurali convoluzionali uno dei lavori più influenti è quello di Kendall et al. [20], che offre un approccio utilizzabile solamente all'interno di uno specifico ambiente che viene appreso da una rete. Altri lavori hanno seguito questo approccio, proponendo reti neurali con errori di localizzazione molto bassi, ma limitati dall'utilizzo della rete al solo luogo in cui è stata addestrata. Il presente lavoro di tesi, al contrario, ha come obiettivo quello di generalizzare il concetto di mappa mediante una rete neurale convoluzionale e rendere possibile la localizzazione all'interno di ambienti che non sono mai stati mostrati alla rete durante le fasi di apprendimento.

Il lavoro di tesi svolto si è sviluppato in diversi passi che di seguito vengono descritti in maniera sintetica:

1. Inizialmente si sono ricercati all'interno della letteratura i lavori che sfruttano tecniche basate su reti neurali convoluzionali per affrontare problemi legati al mondo automotive [1, 8, 20, 39]. In particolare, si è posta l'attenzione su approcci che utilizzano sia immagini RGB sia immagini LiDAR, andando a comprendere i metodi utilizzati per rappresentare a livello immagine l'informazione ottenuta dalle scansioni laser di una scena. Inoltre, si sono confrontate le modalità con cui è possibile rappresentare le mappe 3D e si sono ricercati gli approcci esistenti per ovviare al problema della sparsità delle immagini LiDAR.

2. Successivamente, si è effettuata la ricerca e l'analisi dei dataset utilizzati all'interno del mondo automotive e messi a disposizione dalla comunità scientifica. In particolare si sono ricercati dataset che contengano al loro interno acquisizioni di scansioni laser, immagini RGB e pose camera, ottenute durante le fasi di navigazione di un veicolo in scenari urbani. L'obiettivo è stato quello di utilizzare un dataset di partenza per la generazione di mappe, da cui ottenere le immagini LiDAR. La scelta finale è ricaduta sul Oxford Robotcar dataset [25].

3. Partendo dal dataset di Robotcar, si è proceduto a sviluppare una pipeline automatizzata per la creazione di un dataset differente, contenente al suo interno: immagini LiDAR, immagini RGB e le groundtruth di disallineamento tra le due. La prima parte della pipeline sviluppata si occupa di generare la point-cloud rappresentante la mappa 3D, andando ad allineare diverse scansioni laser, con il fine di rappresentare un tratto di strada percorso dal veicolo ad un certo istante temporale. Questa fase del processo è stata sviluppata con l'ausilio di una libreria messa a disposizione dai creatori di Robotcar, effettuando una re-implementazione di alcune delle sue funzionalità.

4. Successivamente è stato sviluppato un metodo per effettuare il passaggio dalla rappresentazione a point-cloud alla rappresentazione a voxel della mappa, utilizzando il framework OctoMap sviluppato da Hornung et al. [15] che permette di generare delle mappe a voxel basandosi su particolari strutture dati ad albero chiamate *octree*. In particolare, è stata re-implementata la classe rappresentante la mappa rendendo così possibile tenere traccia del dato di riflettanza per ogni singolo voxel. Il valore rappresentante viene scelto effettuando dapprima una ricerca dei 5 punti più vicini al centro del voxel all'interno della point-cloud di partenza, di cui viene calcolata la media. Per la manipolazione delle point-cloud è stata utilizzata una libreria molto popolare chiamata PCL [37] (Point Cloud Library). Inoltre, si è sperimentato l'utilizzo di mappe a voxel con diversa risoluzione per creare dataset differenti.

5. Data la mappa voxelizzata, si è utilizzata la tecnica di raytracing per rappresentare a livello immagine le informazioni di profondità e riflettanza degli elementi osservati. Questa tecnica è stata applicata a partire da una pose camera le cui componenti di rotazione e traslazione vengono perturbate secondo un errore casuale all'interno di un certo range, andando a simulare la rumorosità della pose fornita in input al veicolo. L'obiettivo è stato quello di ottenere la rappresentazione della scena osservata da una punto di vista che è differente rispetto a quello del sensore camera a causa dell'errore di localizzazione. Le direzioni dei raggi emessi tengono in considerazione i parametri intrinseci della camera montata sul veicolo, con lo scopo di simularne il funzionamento. Le immagini LiDAR ottenute possiedono la medesima risoluzione di quelle RGB.

6. Per ridurre i tempi di computazione della creazione del dataset e per fornire una maggiore variabilità di esempi, si è implementato un metodo per applicare la tecnica di *data augmentation*. In particolare per un stessa immagine RGB e mappa a voxel 3D, sono state applicate iterativamente operazioni di mirroring e rotazioni in-place secondo diverse combinazioni.

7. A seguito delle ricerche svolte all'interno dello stato dell'arte, è stata implementata una rete neurale di nome RegNet, frutto del lavoro di Schneider et al. [39], andandone a modificare parte dell'architettura per adattarla al formato delle immagini LiDAR in input. In particolare è stato utilizzato un framework python di nome PyTorch per implementare la rete neurale, per gestire i dataset creati in precedenza e per effettuare le operazioni di training e testing.

8. Infine, si è proceduto a testare la rete neurale convoluzionale, andando ad osservarne le differenze in termini di performance in base alle variazioni dei suoi iperparametri utilizzando, ad esempio, diverse funzioni di loss [18] e diversi dataset tra quelli generati.

Per comprendere la validità dell'approccio sviluppato, si sono andate ad osservare le performance della rete neurale rispetto alle componenti di traslazione e rotazione della pose predetta. Tuttavia fornire una metrica unificata, per le due tipologie di componenti, risulta un problema ancora aperto. Di conseguenza i risultati, dei test effettuati con la rete neurale, sono stati mostrati evidenziando separatamente le migliori performance in termini di traslazione e le migliori in termini di rotazione.

Dai risultati ottenuti si può concludere che l'approccio sviluppato costituisce un buon di punto di partenza per affrontare il problema della localizzazione mediante mappe LiDAR e immagini RGB.

Di seguito viene fornita la struttura del presente elaborato, descrivendo brevemente il contenuto dei diversi capitoli.

Nel secondo capitolo verrà fornita un visione dei lavori presenti all'interno dello stato dell'arte, mettendo in evidenza gli approcci basati su reti neurali convoluzionali che sono utilizzati per affrontare le problematiche riguardanti la robotica mobile. Inoltre, verranno mostrate le modalità con cui è possibile rappresentare una mappa 3D ottenuta mediante scansioni laser e verrà effettuato un confronto tra i diversi approcci esistenti per la generazione di immagini LiDAR dense.

Nel terzo capitolo verranno mostrate le motivazioni che hanno portato allo svolgimento del presente lavoro, fornendo anche una visione ad alto livello del lavoro di tesi svolto.

Nel quarto capitolo si parlerà della pipeline implementata per la generazione dei dataset contenenti immagini LiDAR, immagini RGB e le groundtruth di disallineamento tra le due. Inizialmente verrà descritto il dataset di partenza utilizzato per l'ottenimento delle immagini e si motiverà la scelta di quest'ultimo rispetto ad altri dataset esistenti. Successivamente, verranno descritti i passi sviluppati per la generazione dei dataset utilizzati per la fase di apprendimento della rete neurale: descrivendo le modalità di rappresentazione delle mappe LiDAR, la tecnica di proiezione utilizzata e le tecniche di data augmentation applicate.

Nel quinto capitolo verrà descritta la rete neurale convoluzionale implementata e verranno mostrati i diversi test effettuati modificando gli iper-parametri della rete e provando ad utilizzare dataset differenti. Dati i risultati ottenuti si effettueranno delle considerazioni riguardo la qualità delle performance ottenute, andando ad evidenziare le possibili soluzioni per risolvere le problematiche emerse.

All'interno del sesto capitolo si parlerà delle conclusioni rispetto al lavoro svolto e ne verranno mostrati i possibili sviluppi futuri.

Capitolo 2

Stato dell'arte

In questo capitolo verrà trattato lo stato dell'arte inerente al lavoro di tesi svolto. In particolare si parlerà di come negli ultimi anni il mondo della computer vision si sia orientato verso tecniche di deep learning per risolvere problemi noti nel mondo della robotica mobile.

Per robot si intende un sistema formato da componenti hardware e software che è in grado di interagire con l'ambiente in cui opera per mezzo di sensori ed attuatori. Tale ambiente rappresenta lo spazio operativo all'interno del quale il robot ha il compito di svolgere delle attività che vanno dalla movimentazione alla manipolazione di oggetti. Lo svolgimento di queste attività implica che il robot acquisisca le informazioni necessarie dall'ambiente circostante per mezzo di sensori di diversa natura come camere e laser. Infine gli attuatori sono le componenti del sistema che ne permettono di eseguire delle azioni come, ad esempio, un motore che agisce per regolare lo sterzo di un veicolo.

Nella robotica mobile l'esecuzione di operazioni che permettono la navigazione di un robot risultano essere un problema complesso. L'imprevedibilità dell'ambiente circostante causata dalla presenza di ostacoli statici e dinamici è una delle possibili fonti che generano difficoltà nello svolgere queste operazioni. Inoltre vi è una forte componente di incertezza legata sia agli errori generati dai sensori, dovuta alla loro capacità limitata di misurazione, sia alle semplificazioni dei modelli di moto implementate a livello software. Tali equazioni risultano essere un'approssimazione del funzionamento reale di questi sistemi in maniera tale da garantire ad un robot di lavorare ad alte frequenze. In particolare in ambito outdoor queste problematiche risultano essere più difficili da affrontare a causa della natura più complessa dell'ambiente.

Data la rivoluzione sperimentata nel campo della computer vision dovuta all'utilizzo delle reti neurali convoluzionali, inizialmente verrà fornita un'introduzione del funzionamento generale di queste tecniche di deep learning. Successivamente si parlerà di come esse abbiano preso piede negli ultimi anni all'interno della robotica mobile.

Essendo stati utilizzati dati provenienti da sensori laser nel lavoro di tesi svolto, verrà spiegato il loro funzionamento. I sensori laser permettono di acquisire le distanze degli oggetti di una scena mediante l'emissione di raggi luminosi, andando a formare una scansione di un ambiente. Di tali sensori esiste una tipologia, chiamata LiDAR (Light Detection and Ranging), che permette di acquisire anche l'informazione di riflettanza, che costituisce il quantitativo di luce ritornata al sensore da un oggetto intersecato dal raggio emesso. In particolare verranno citati alcuni lavori presenti in letteratura per la rappresentazione delle scansioni e si parlerà di come utilizzare tali informazioni per la creazione di mappe.

Verranno inoltre fornite le nozioni base del modello di proiezione di una camera, poiché esso costituisce la base di parte del lavoro svolto.

Infine si mostreranno le tecniche per l'ottenimento di immagini LiDAR dense e si metteranno a confronto i diversi approcci. Notare, che una immagine LiDAR rappresenta le proiezioni delle scansioni 3D ottenute da un sensore LiDAR all'interno di un piano immagine.

2.1 Introduzione alle reti neurali convoluzionali

Le reti neurali convoluzionali, anche conosciute come CNN (*convolutional neural networks*), sono una specifica classe di reti neurali applicate nel campo del deep learning per l'elaborazione di immagini.

Questa tipologia di reti ha riscosso molto successo nell'ultima decade nel campo della *computer vision*, poiché hanno superato in termini di performance le tecniche classiche adottate nelle operazioni, ad esempio, di classificazione e segmentazione di immagini.

Nonostante la teorizzazione delle reti neurali risalga all'incirca agli anni '80 [12], l'impiego delle CNN ha avuto largo utilizzo solo in tempi recenti per i seguenti motivi:

- Mancanza di un gran numero di dati per generalizzare correttamente i concetti da apprendere
- Mancanza della potenza di calcolo necessaria per permettere una computazione in tempi ragionevoli

Rispetto agli approcci standard, che si basano su feature create ad hoc [6, 24, 29], le reti neurali convoluzionali hanno la peculiarità di estrarre in autonomia, dalle immagini in input, l'insieme di features che permette di minimizzare l'errore sulla predizione finale.

Alcune delle tecniche sviluppate in passato, ad esempio nel campo della classificazione, richiedono che venga eseguita la ricerca delle regioni di interesse (*ROI: regions of interest*) all'interno dell'immagine. Lo *sliding window*, come proposto nel lavoro di Viola et al. [45], è un esempio di tecnica che opera a tale scopo. In particolare permette di campionare le features all'interno delle sotto-regioni di un'immagine traslando su di essa delle maschere a diverse risoluzioni. Questo processo possiede il grande svantaggio di essere computazionalmente oneroso, non rendendolo applicabile in sistemi che necessitano alte frequenze di elaborazione come nell'ambito automotive. Una possibile soluzione per ridurre i tempi di calcolo consiste nel processare immagini a bassa risoluzione, comportando però una riduzione delle performance in termini, ad esempio, di accuratezza della classificazione.

Le reti neurali che si focalizzano sull'estrazione di regioni d'interesse con il fine, ad esempio, del rilevamento di oggetti prendono il nome di *region-based* CNN. Un lavoro presente in letteratura che tratta questa tipologia di reti è quello di Girshick [11]. All'interno di tale lavoro la rete mostrata possiede la caratteristica di estrarre le ROI impiegando solamente 0.15 secondi.

Il vantaggio, che ha permesso alle CNN di prendere piede all'interno del campo della computer vision, risiede nella velocità di computazione in fase di predizione e nelle performance ottenute, ad esempio, nel campo della classificazioni di immagini.

Un esempio è fornito da YOLO (*You Only Look Once*), frutto del lavoro di Redmon et al. [35]. Questa rete neurale è utilizzata per il task di object detection ed è in grado di processare immagini in real-time a $45 fps$ (frame per secondo) classificando 1000 categorie diverse di oggetti. Di tale rete esistono diverse versioni che sono in grado di processare

immagini circa tre volte più velocemente [40] o che classificano fino a 9000 classi diverse [34].

2.2 Applicazione delle reti neurali nella robotica mobile

Le reti neurali convoluzionali hanno avuto un impatto decisivo anche nell'ambito della della robotica mobile. Negli ultimi anni si è iniziato ad utilizzare tecniche di deep learning per affrontare problematiche quali rilevamento di ostacoli, ricostruzione 3D e localizzazione. Recentemente, all'interno della comunità scientifica, si è iniziato ad integrare l'informazione proveniente da sensori laser come il LiDAR (Light Detection and Ragning) per fornire informazioni di profondità (*depth*) della scena e riflettanza degli oggetti in essa contenuti. Di conseguenza hanno fatta la comparsa in letteratura CNN che sfruttano, oltre che il dato pittorico RGB, anche immagini LiDAR.

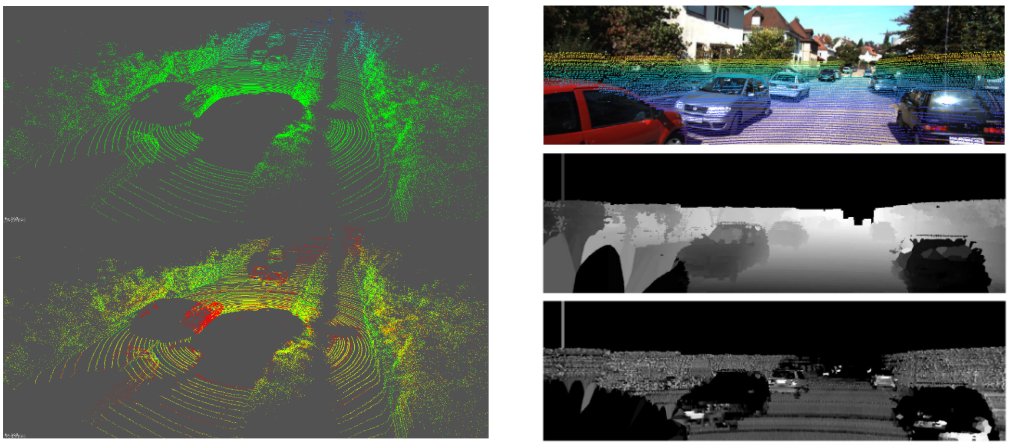
Il sensore LiDAR permette di ricostruire in maniera puntuale la tridimensionalità di una scena ed è in grado, dipendentemente dalla tipologia, di reperire il quantitativo di luce riflessa dagli oggetti colpiti dal raggio di luce emesso. Due esempi di scansioni vengono riportati in figura 2.1a.

Per ottenere un'immagine basandosi su questa tipologia di sensore è necessario proiettare su un piano immagine i punti della scena, ottenendo generalmente un'immagine a due canali: un canale rappresentante la profondità e uno rappresentante la riflettanza. Questo passaggio si rende necessario, poiché una rete neurale convoluzionale in grado di processare dati 3D risulta essere dispendiosa in termini di risorse computazionali. In figura 2.1b è riportato un esempio di proiezione di una scansione su un piano immagine e le immagini di profondità e riflettanza ricostruite.

I dettagli riguardanti questa tipologia di sensore e le modalità su come integrare a livello immagine le informazioni da esso ottenute, verranno spiegate successivamente (sezione 2.5).

Per quanto riguarda i lavori presenti in letteratura che applicano reti neurali all'interno della robotica mobile ci si focalizzerà sulle seguenti problematiche:

- Rilevamento di ostacoli
- Visione tridimensionale
- Calibrazione e localizzazione



(a) (b)

Figura 2.1: In figura (a) vengono mostrati due esempi di scansioni LiDAR di una strada: quello più in basso mostra l'informazione di riflettanza dei punti rilevati. In figura (b) vengono mostrate dall'alto verso il basso: un'immagine RGB con la proiezione dei punti di una scansione LiDAR, un'immagine di profondità e una di riflettanza

Lo scopo è quello di fornire al lettore una visione d'insieme dello stato dell'arte su queste tematiche e far trasparire la caratteristica comune della gran parte dei lavori citati nell'utilizzare immagini LiDAR.

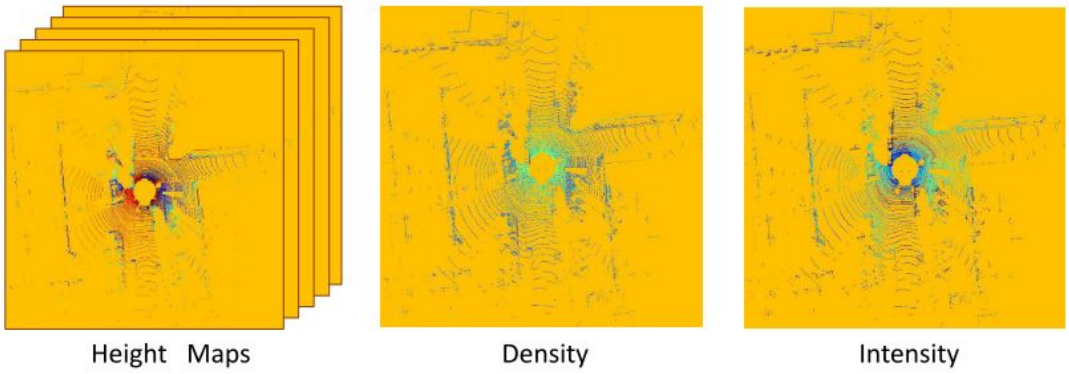
2.2.1 Rilevamento di ostacoli

Nella robotica automotive uno dei principali problemi risiede nella corretta identificazione di ostacoli statici e dinamici. Per ostacoli statici si intendono gli elementi della scena la cui posizione e orientamento non cambia al variare del tempo (e.g. edifici, alberi, marciapiedi), mentre viceversa si definiscono gli ostacoli dinamici (e.g. veicoli, ciclisti, pedoni). Il corretto rilevamento di ostacoli è un'operazione critica in robotica mobile, poiché nel caso di incidente un veicolo potrebbe ledere l'incolumità di un essere umano.

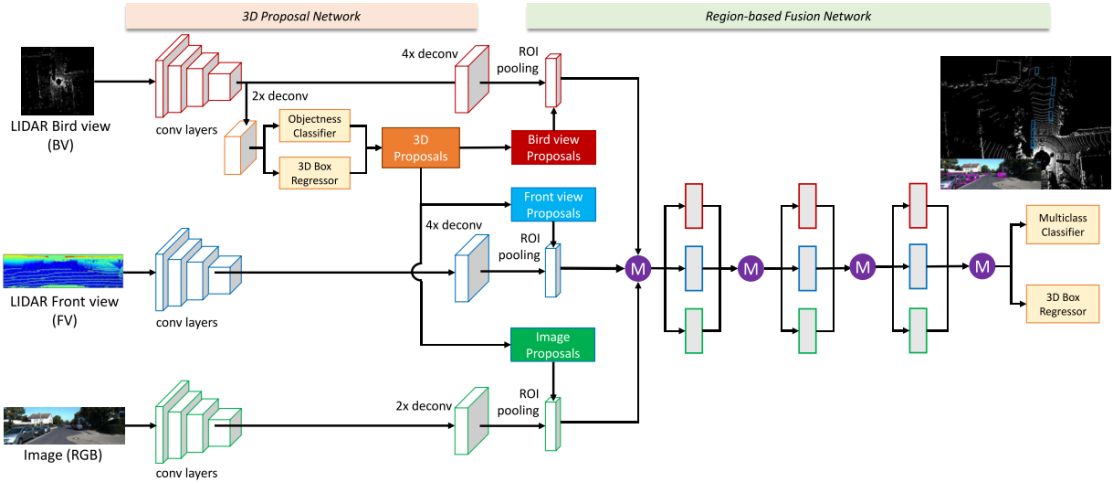
Normalmente è solito trovare in letteratura approcci che sfruttano solo il dato camera per effettuare la classificazione degli oggetti, come mostrato nel lavoro citato in precedenza Redmon et al. [35]. Tuttavia negli ultimi anni si è iniziato a utilizzare sensori laser per sfruttare il dato di profondità e riflettanza oltre al dato camera RGB. Un esempio è fornito da Asvadi et al. [1] dove il dato laser viene integrato per effettuare il riconoscimento di veicoli.

Un lavoro interessante nell'ambito del rilevamento di ostacoli che integra il dato LiDAR è quello di Chen et al. [5]. In particolare nell'approccio mostrato viene sviluppata una CNN per il rilevamento di veicoli effettuando una classificazione sia a livello immagine sia a livello di scansione laser 3D. La peculiarità del lavoro è che vengono sfruttate due tipologie di immagini LiDAR: una chiamata *front view* ed una chiamata *birds's eye view*. La rappresentazione front view costituisce la proiezione dei punti su un piano immagine di una camera che tiene traccia dell'informazione di profondità, riflettanza e altezza dei punti di una scansione. Notare che tali proiezioni vengono effettuate considerando la posizione e l'orientamento (*pose*) della camera da cui si vuole ottenere l'immagine RGB. La rappresentazione bird's eye view fornisce un'immagine multicanale di una scansione laser nel 3D, fissando il punto di osservazione ad una certa elevazione. Il numero di canali viene ottenuto andando a discretizzare la scansione LiDAR in termini di altitudine, con la conseguenza di rappresentare per ogni canale i punti giacenti all'altezza che il singolo canale rappresenta. A tale rappresentazione viene inoltre associata l'informazione di densità dei punti della scansione e della loro riflettanza. Un esempio esplicativo di questa rappresentazione è fornita in figura 2.2a. La rete sviluppata processa inizialmente le immagini bird's eye view, front view e RGB separatamente ed utilizza la prima tipologia per estrarre delle regioni di interesse a livello tridimensionale. Successivamente, data la relazione spaziale tra le immagini di input, le regioni 3D proposte vengono utilizzate per estrarre regioni di interesse nel 2D per le altre due tipologie di immagini. Il ramo finale della rete fonde le features ottenute dalle tre diramazioni ed infine predice le bounding box 3D rappresentanti i veicoli rilevati all'interno della scansione iniziale. L'architettura descritta è riportata in figura 2.2b.

Nel lavoro di Vaquero et al. [44] viene presentata una rete neurale convoluzionale che si occupa di detection di veicoli prendendo in input esclusivamente immagini di profondità e riflettanza. Queste immagini vengono ottenute partendo da scansioni laser provenienti da un LiDAR 64 piani. Una volta processate le immagini mediante la CNN ed estratte le regioni di interesse corrispondenti ai veicoli, la classificazione della rete viene estesa alla scansione tridimensionale grazie al dato di profondità. Tale lavoro si differenzia dal precedente poiché la rete non effettua una predizione nello spazio 3D, ma estende la classificazione 2D sfruttando il dato di profondità di partenza. Inoltre viene implementato un sistema di *tracking* dei veicoli basato su *Kalman filtering*.



(a) Immagini ottenute mediante Bird's eye view



(b) Architettura della rete riportata in Chen et al. [5].

Figura 2.2: In figura (a) viene mostrata la rappresentazione bird's eye view di una scansione LiDAR. In (b) è riportata l'architettura della rete neurale convoluzionale proposta da Chen et al. Le immagini mostrate provengono dal lavoro dei medesimi autori [5]

.

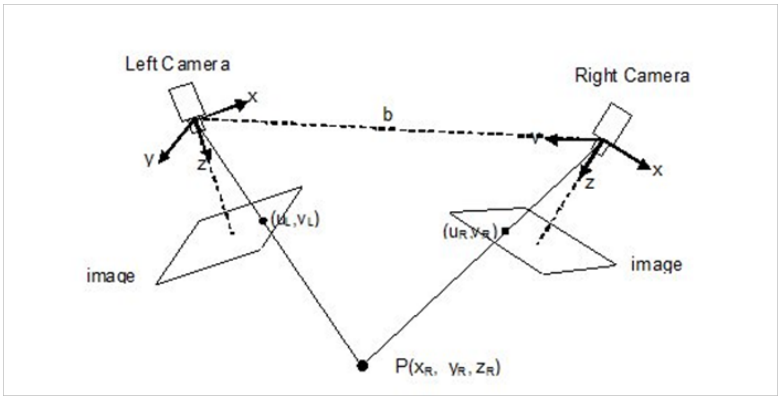
2.2.2 Reti neurali applicate a problemi di visione 3D

Un altro problema comunemente trattato all'interno del campo della computer vision è quello di ottenere informazione tridimensionale partendo dal dato immagine. La stereoscopia costituisce l'insieme di tecniche adottate per ottenere informazione tridimensionale di un ambiente partendo da due immagini che osservano una stessa scena da due punti di vista differenti [13, 38]. I sistemi che sfruttano questa tecnica si basano su una coppia di camere stereo (figura 2.3a). L'ottenimento di informazione 3D della scena data una coppia di immagini prende il nome di *ricostruzione* e viene effettuata partendo da una mappa di disparità. Tale mappa è ottenuta mediante l'applicazione delle tecnica di *ricerca delle corrispondenze* che permette di identificare lo stesso punto della scena rappresentato nelle due immagini e di conseguenza ricavare l'informazione tridimensionale tramite triangolazione. Gli approcci che operano a livello pixel (definiti *pixel-level*) vengono chiamati densi, ma esistono anche tecniche definite *sparse* che mettono a confronto features estratte dalle due immagini di partenza utilizzando, ad esempio, SIFT (Scale Invariant Feature Transform).

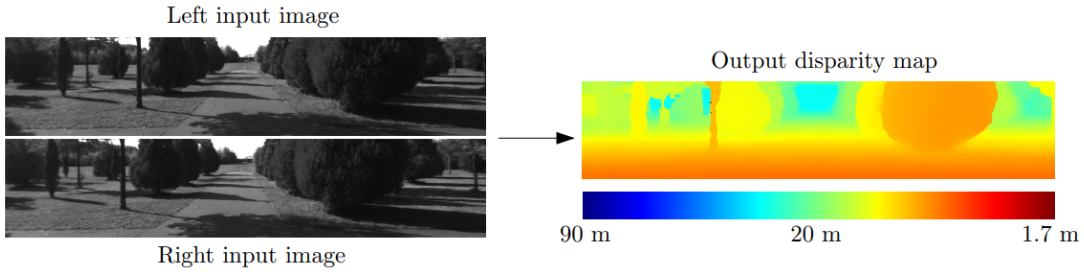
Le reti neurali che affrontano questa problematica non si posizionano in maniera centrale all'interno del lavoro svolto, di conseguenza verranno menzionate in maniera sintetica. L'obiettivo è quello di far trasparire quanto l'utilizzo delle tecniche di deep learning si sia allargato anche a questi ambiti applicativi della robotica mobile. Inoltre, sebbene non si integri il dato LiDAR, tali reti forniscono un approccio per ricavare il dato di profondità di una scena.

Due esempi di lavori trovati in letteratura che utilizzano una rete neurale per affrontare il problema della ricostruzione sono quelli di Zbontar and LeCun [47] e di Chang and Chen [4]. Questi lavori si pongono l'obiettivo di effettuare la regressione di una mappa di disparità data una coppia di immagini RGB in input. L'architettura delle reti presentate processa le due immagini di input in due rami separati della rete che condividono tra di loro i pesi dei layer convolutivi. Questa tipologia di reti rientra nella categoria di *siamese networks*.

Nel lavoro di Chang et al. viene utilizzata come funzione di loss una variante della distanza euclidea avente la caratteristica, rispetto alla distanza L_2, di essere più robusta agli outlier. Tale funzione di loss viene riportata in precedenza nel lavoro di Girshick [11] ed è stata utilizzata all'interno di questo lavoro di tesi. Questa distanza prende il nome di $smooth_{L1}$ loss e di seguito vengono fornite la formula della sua applicazione su



(a) Sistema di camera stereo



(b) Input (a destra) e output (a sinistra) della rete sviluppata da Zbontar and LeCun [47]

Figura 2.3: Nell'immagine (a) viene riportata una versione semplificata di un sistema binoculare per la ricostruzione 3D, mentre in figura (b) viene mostrato l'input e l'output di una rete utilizzata per la creazione di una mappa di disparità. L'immagine (b) è stata presa da Zbontar and LeCun [47].

una mappa di disparità e la sua formula generica:

$$L(d, \hat{d}) = \frac{1}{N} \sum_{i=1}^{N} smooth_{L1}(d - \hat{d}) \tag{2.2.2.1}$$

dove $\hat{d}, d$ rappresentano rispettivamente la groundtruth e la predizione dell'immagine di disparità, mentre N identifica il numero di pixel.

La formula della funzione di loss $smooth_{L1}$ è la seguente:

$$smooth_{L1}(x) = \begin{cases} 0.5x^2, & \text{se } |x| < 1 \\ |x| - 0.5, & \text{altrimenti} \end{cases} \tag{2.2.2.2}$$

All'interno della letteratura oggigiorno è possibile trovare lavori che sfruttano reti neurali convoluzionali per affrontare anche il problema della stima del flusso ottico (*optical flow*). Tale problematica viene affrontata con il fine di effettuare la stima del moto degli oggetti presenti in una scena. Il problema della stima del flusso ottico viene trattato sin dagli anni '80 [14] e ha sempre riscosso molto interesse nel campo della computer vision [27, 46].

FlowNet è una rete neurale sviluppata da Dosovitskiy et al. [8] che stima il flusso ottico che intercorre tra due frame immagine in input. Esiste una sua evoluzione chiamata FlowNet2.0 sviluppata da Ilg et al. [17] che ne migliora le performance in termini di rilevazione di piccoli movimenti degli elementi della scena. Questo miglioramento è dovuto anche dai differenti dataset utilizzati per allenare le due reti: nel primo caso viene utilizzato il dataset Sintel [3] di immagine sintetiche, mentre nel secondo viene utilizzato anche il dataset UFC101 [41] contenente una moltitudine di video reali all'interno dei quali vengono portate a compimento diverse tipologie di azioni umane.

Notare come in certi ambiti applicativi possedere un dataset di esempi reali correttamente etichettati non è sempre possibile oppure le dimensioni di tali dataset impediscono un approccio basato su reti neurali convoluzionali, poiché queste tecniche richiedono grandi quantità di dati. Come mostrato nei lavori appena citati a volte si rende necessario utilizzare immagini sintetiche.

2.2.3 Calibrazione e localizzazione

Altre operazioni critiche all'interno del mondo della robotica e computer vision sono rispettivamente quelli della localizzazione e calibrazione del sensore camera.

Per localizzazione si intende l'operazione che permette di stimare la pose (*posizione e orientamento*) di un robot all'interno dell'ambiente in cui esso opera.

La calibrazione dei parametri camera, invece, consiste nell'insieme di tecniche utilizzate per la stima dei parametri chiamati intrinseci ed estrinseci. Per parametri intrinseci si intendono l'insieme di valori che caratterizzano l'hardware del sensore camera utilizzato come, ad esempio, lunghezza focale e posizione del punto principale del piano immagine. Per parametri estrinseci si intende l'insieme di valori che identificano la pose del sensore rispetto ad un sistema di riferimento principale. Ad esempio, in ambito automotive tali parametri descrivono la trasformazione geometrica che intercorre tra la pose della camera e la pose del veicolo su cui essa è montata. Per quanto riguarda il funzionamento di un modello camera tradizionale verrà presentata una introduzione nelle sezioni successive (2.4).

Negli ultimi anni hanno fatto la comparsa reti neurali convoluzionali che affrontano queste tipologie di problemi e che in particolare effettuano regressione di una pose. Di seguito vengono mostrati alcuni esempi trovati all'interno della letteratura.

Un lavoro che tratta il problema della calibrazione, in particolare della stima dei parametri estrinseci della camera, è quello di Schneider et al. [39]. La peculiarità dell'approccio fornito è che rispetto ai metodi tradizionali per la calibrazione di sensori [2, 9, 22, 28], il problema della calibrazione viene affrontato mediante una CNN che prende il nome di RegNet. In particolare si integra il dato della scansioni di un sensore LIDAR all'interno del sistema. Di questo lavoro viene di seguito fornita una descrizione dettagliata poiché l'architettura di rete è stata ripresa nel lavoro svolto. La rete neurale prende in input un'immagine RGB ed una di profondità ottenuta dalla proiezione di una scansione LiDAR. Le immagini RGB e le scansioni LIDAR utilizzate provengono dal dataset di KITTI [10], molto popolare all'interno della robotica automotive. Per ottenere un quantitativo di immagini sufficiente ad allenare la rete neurale sviluppata, viene perturbata la pose iniziale del sensore camera in maniera casuale e secondo un range di decalibrazione fissato. La decalibrazione viene successivamente utilizzata per proiettare i punti della scansione LiDAR nel piano immagine ottenuto da questo processo. Questa tecnica prende il nome di *data augmentation* e grazie ad essa è possibile ottenere un dataset contenente un numero di immagini adatto per il training della rete. Un esempio di decalibrazione viene fornito in figura 2.4. RegNet inizialmente processa in maniera separata l'immagine RGB e l'immagine di depth. Le feature estratte vengono poi conca-

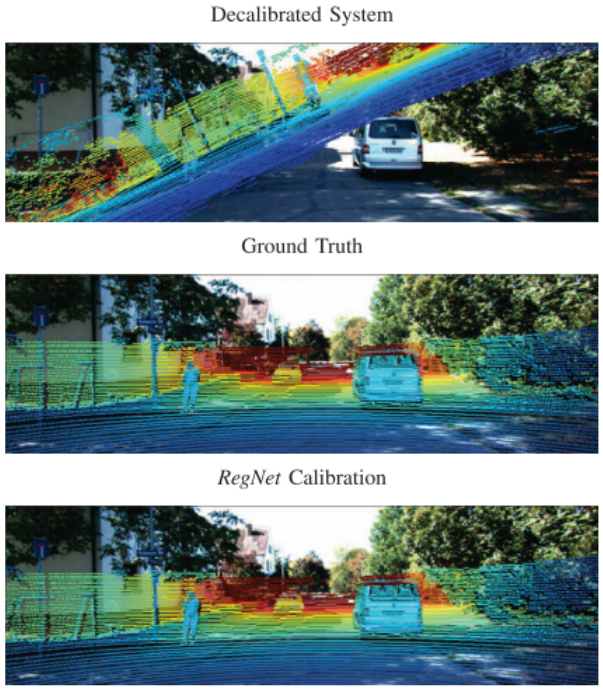


Figura 2.4: Immagine presa da [39] rappresentante la correzione dell'errore sulla calibrazione del sensore camera effettuata da RegNet.

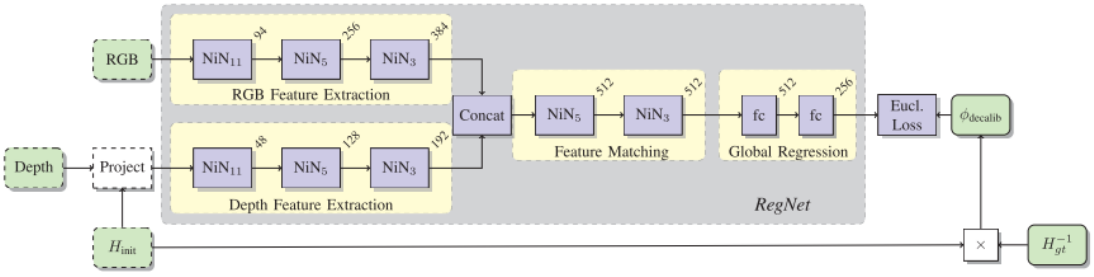


Figura 2.5: Architettura di RegNet. Immagine presa da [39].

tenate e processate all'interno di un unico ramo finale di layer convolutivi. Particolarità della rete è che i layer convolutivi vengono strutturati secondo il lavoro di Lin et al. [23], cioè utilizzando dei blocchi chiamati *Network in Network* (NiN). Un blocco NiN è composto da una convoluzione *n x n* seguita da una serie di *1 x 1* convoluzioni. La regressione dell'errore di calibrazione viene effettuata dai layer fully-connected posti al termine dell'architettura. Questi si diramano in due filoni distinti per predirre in maniera indipendente i parametri di traslazione e rotazione. L'architettura di RegNet è mostrata in figura 2.5. Uno dei problemi osservati dagli autori all'interno del lavoro, è che di fronte a forti decalibrazioni della pose camera la rete difficilmente riesce a trovare un numero di corrispondenze esaustivo tra l'immagine RGB e l'immagine di depth. La conseguenza è quella di non riuscire a correggere di molto l'errore di decalibrazione. La soluzione adottata per risolvere il problema riguarda l'utilizzo di un approccio di *iterative refiniment*: ad ogni correzione della decalibrazione predetta dalla rete viene riproiettata l'immagine di depth secondo la nuova pose, creando così un nuovo input per la rete. Iterando questo approccio, su reti specializzate in diversi range di errore, è possibile andare a migliorare ad ogni passo la decalibrazione predetta dalla rete.

La funzione di loss applicata dalla rete calcola separatamente la distanza L_1 sulla traslazione e sulla rotazione. Successivamente i valori ottenuti vengono sommati andando a riscalare la loss sulla rotazione. Questa operazione viene effettuata poiché i paramatri di rotazione vengono rappresentati su scala diversa rispetto alla traslazione. Identificare correttamente una funzione di loss applicabile a reti neurali di questo tipo richiede che vengano effettuate diverse considerazioni, infatti nella sotto-sezione successiva ne verrà fornita una spiegazione dettagliata.

Sempre all'interno dello stesso lavoro viene mostrato come il tipo di rappresentazione della pose influisce sulle performance della rete. Ad esempio la rappresentazione dell'orientamento mediante angoli di Eulero porta a dei risultati peggiori rispetto alla rappresentazione mediante quaternioni. Gli angoli di Eulero è noto come soffrano del fenomeno del blocco cardanico, conosciuto anche come *Gimbal-lock*, di conseguenza è preferibile adottare un altro tipo di rappresentazione.

Per quanto riguarda la problematica della localizzazione mediante reti neurali convoluzionali, esiste all'interno dello stato dell'arte il lavoro di Kendall et al. [20]. L'obiettivo di tale lavoro consiste nello sviluppo di una CNN che sia in grado di effettuare regressione della pose di un sistema camera monoculare. La rete sviluppata prende il nome

di *PoseNet*. Il dataset utilizzato per allenare la rete è stato creato e rilasciato dagli stessi autori dell'articolo e comprende un insieme di immagini acquisite da un sensore camera in scenari urbani e all'aperto. Il metodo di acquisizione si basa su una tecnica chiamata *structure from motion*, tecnica che rientra nel campo della ricostruzione 3D di una scena data una sequenza di immagini. Per quanto riguarda l'architettura della rete utilizzata, nell'articolo si fa riferimento a GoogLeNet, frutto del lavoro di Szegedy et al. [42]. L'architettura di tale rete viene opportunamente modificata per effettuare il task di localizzazione e la funzione di loss utilizzata è la medesima applicata a RegNet [39].

2.2.4 Funzioni di loss applicate alla regressione di una pose

Utilizzare una rete neurale convoluzionale per effettuare la regressione di una pose, pone delle domande su come mettere a confronto gli errori di traslazione e rotazione mediante l'utilizzo di una funzione di loss. La problematica risiede nel fatto che le componenti della traslazione rispetto a quelle della rotazione vengono rappresentate su una scala di valori differente. Basti pensare che in ambito automotive l'errore sulle componenti di traslazione può variare di diversi metri, mentre sulle rotazioni l'utilizzo, ad esempio, della rappresentazione a quaternioni porta ad avere valori nell'intervallo $[0, 1]$ sulle diverse componenti, che per vincolo di rappresentazione devono formare un vettore di norma unitaria.

Per affrontare la problematica appena descritta, in letteratura esistono diversi approcci:

- fissare un parametro per riscalare la rotazione rispetto alla traslazione
- utilizzare un funzione di loss che permette alla rete di apprendere tale parametro di rescaling
- calcolare una distanza tra le proiezioni dei punti di una scena nella pose camera predetta dalla rete rispetto a quella obiettivo

Nell'articolo di PoseNet [20] e nel lavoro di Schneider et al. [39] viene utilizzato il primo approccio. La funzione di loss apllicata all'interno dei due lavori è la seguente:

$$loss(I) = \|\hat{x} - x\|_2 + \beta \cdot \left\| \hat{q} - \frac{q}{\|q\|} \right\|_2 \tag{2.2.4.1}$$

dove I rappresenta l'input della rete, $\hat{x}$ e x identificano rispettivamente i parametri di traslazione attesi e predetti ed infine $\hat{q}$ e q rappresentano i parametri di rotazione attesi e predetti sotto forma di quaternione. Notare che β identifica il parametro di rescaling e la loss sulle componenti di traslazione e rotazione viene definita come descritto in PoseNet, cioè utilizzando la formula di distanza L_2.

Nell'articolo di PoseNet vengono sperimentati diversi valori da assegnare al parametro β e vengono fissati degli intervalli consigliati entro i quali definirlo a seconda che l'ambito applicativo sia indoor o outdoor. Viene consigliato di utilizzare $\beta \in [120; 750]$ per scenari indoor, mentre per scenari all'aperto si suggerisce $\beta \in [250; 2000]$.

La problematica principale di questo tipo di loss risiede nel fissare il parametro beta in maniera ottimale, poiché richiede di svolgere un grande quantitativo di esperimenti. Dati i lunghi tempi richiesti per l'apprendimento di una rete neurale questa loss può risultare poco efficiente. Altro difetto riguarda l'utilizzo di una distanza euclidea sui quaternioni, che non sono definiti in uno spazio euclideo. Nonostante questa imperfezione la funzione di loss si dimostra funzionale per piccoli errori sulle rotazioni.

Le altre due tipologie di approcci elencati in precedenza fanno riferimento al lavoro di Kendall and Cipolla [18].

La seconda funzione di loss prende spunto da un precedente articolo degli stessi autori [19], andando a modellare quella che viene chiamata *incertezza omoschedastica*. L'incertezza omoschedastica rappresenta una misura di incertezza che è indipendente dai dati in input e può essere appressa mediante tecniche di deep learning. In particolare è possibile effettuare l'operazione di regressione di una pose mediante la *likelihood di Laplace*:

$$\mathcal{L}_\sigma(I) = \mathcal{L}_x(I)\hat{\sigma}_x^{-2} + \log(\hat{\sigma}_x^2) + \mathcal{L}_q(I)\hat{\sigma}_q^{-2} + \log(\hat{\sigma}_q^2) \tag{2.2.4.2}$$

Dove $\hat{\sigma}_x^2, \hat{\sigma}_q^2$ rappresentano le incertezze omoschedastice da apprendere tramite la back-propagation di una rete neurale. Notare che x e q fanno riferimento alla funzione di loss applicata alla traslazione e rotazione rispettivamente.

Per un problema di stabilità numerica viene ottimizzato il parametro $s = \log(\hat{\sigma}^2)$, da cui ne consegue la riscrittura della loss nella maniera seguente:

$$\mathcal{L}_\sigma(I) = \mathcal{L}_x(I)\exp(-s_x) + s_x + \mathcal{L}_q(I)\exp(-s_q) + s_q \tag{2.2.4.3}$$

I parametri s_x ed s_q sono degli scalari scelti in base ad una stima iniziale su cui la CNN effettuerà l'ottimizzazione. Gli autori dell'articolo definiscono dei valori iniziali $s_x = 0$ e $s_q = -3$.

La terzo approccio descritto si pone l'obiettivo di evitare la riscalatura delle loss rispetto ai parametri di traslazione e rotazione. In particolare, conoscendo un insieme di punti 3D della scena, viene creata una loss che agisce sull'errore di riproiezione degli stessi. Ad esempio, dato un insieme di punti $\mathcal{G}$ appartenenti allo spazio 3D e date due pose differenti di una camera, rappresentanti rispettivamente la groundtruth e la predizione, è possibile proiettare i punti della scena sui due piani immagine per mezzo di una funzione π e calcolare la media delle distanze tra le coppie di punti che hanno medesime coordinate nel 3D. La formula di tale loss è definita come segue:

$$\mathcal{L}_g(I) = \frac{1}{|\mathcal{G}'|} \sum_{g_i \in \mathcal{G}'} \|\pi(x, q, g_i) - \pi(\hat{x}, \hat{q}, g_i)\|_\gamma \tag{2.2.4.4}$$

dove x, q rappresentano la traslazione e rotazione della groundtruth della pose della camera, $\hat{x}, \hat{q}$ rappresentano la pose predetta e γ è il tipo di distanza utilizzata.

2.3 Creazione di mappe

L'utilizzo di una mappa in ambito automotive a volte si rende necessario per fare in modo che il robot, date le acquisizioni dei sensori, riesca a localizzarsi al suo interno. Esistono diverse tipologie di mappe:

- Mappe 2D e 2.5D
- Mappe 3D

Le prime due rappresentazioni verranno spiegate in sintesi per fornire una visione più completa, mentre l'utilizzo di mappe 3D verrà spiegato in maniera più dettagliata poiché inerente a parte del lavoro svolto.

2.3.1 Mappe 2D e 2.5D

Le mappe 2D forniscono la rappresentazione più semplice della spazio operativo di un robot. Esse si basano sul concetto di *griglie di occupazione* 2D che forniscono una visione della mappa da un punto di vista elevato dove all'interno di ogni cella viene espressa la

probabilità che essa sia occupata o meno. Il punto di vista "dall'alto" prende il nome di *bird's eye view*, ma è necessario specificare che questo tipo di rappresentazione è diverso da quello riportato nel lavoro di Chen et al. [5].

Le mappe 2D sono adatte in contesti in cui un robot deve operare in un ambiente piano e data la loro semplicità risultano vantaggiose in termini computazionali. Di contro esse risultano inefficaci nel descrivere ambienti operativi più complessi.

Le mappe 2.5D possiedono una maggiore espressività rispetto alle precedenti. Questo tipo di rappresentazione si basa anch'esso sulle griglie di occupazione, ma diversamente dalle mappe 2D, per ogni cella viene effettuata una stima di elevazione della mappa.

2.3.2 Mappe 3D e Octomap

Le mappe 3D rappresentano il tipo di mappa più esplicativo in termini di informazione e si suddividono in due categorie:

- mappe che utilizzano una rappresentazione *puntuale*
- mappe che utilizzano una rappresentazione a *voxel*

La prima categoria permette di rappresentare le scansioni laser mediante una struttura dati chiamata *point-cloud* nella quale vengono memorizzate le coordinate di tutti i punti 3D e permettono di associare altre informazioni come, ad esempio, la riflettanza.

Le *point-cloud* permettono di descrivere la scena con un alto grado di risoluzione data la natura della rappresentazione mediante punti. Inoltre non richiedono che venga fissata una dimensione massima della mappa ed è possibile integrare più scansioni all'interno di una stessa struttura dati con semplicità.

Il problema principale delle point-cloud risiede nella difficoltà della loro manipolazione in termini computazionali all'aumentare del numero di punti. Inoltre data la sparsità della rappresentazione può risultare difficile ricostruire in maniera adeguata la geometria della scena. In figura 2.6 viene mostrato un esempio di point-cloud.

La seconda categoria di mappe 3D sfrutta invece l'approccio a *voxel*. Per *voxel* (il nome deriva da *volumetric pixel*) si intende il singolo elemento di forma cubica appartenente ad una griglia tridimensionale che prende il nome di *voxel grid*. Per sfruttare una struttura dati di questo tipo è necessario partire da una rappresentazione puntuale della scena e successivamente discretizzarla fissando la risoluzione della griglia. Un voxel risulta occupato se almeno un certo numero di punti appartenenti alla *point-cloud*

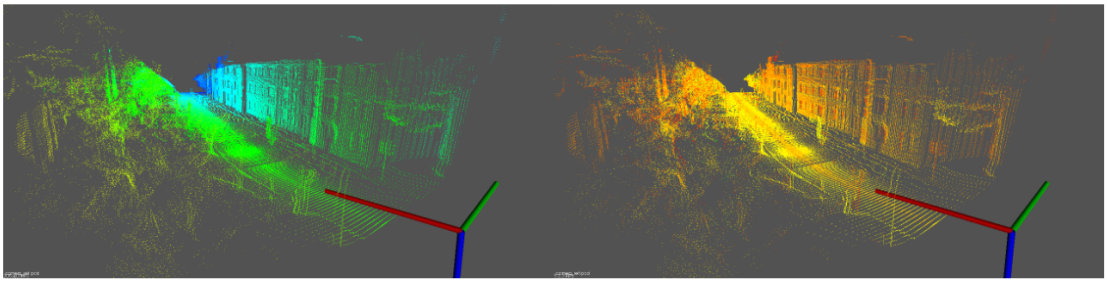


Figura 2.6: Immagine raffigurante due pointcloud di una stessa mappa e il relativo sistema di riferimento camera: a sinistra la scala di colori rappresenta le distanze dei punti rispetto al frame di origine, a destra i colori mostrano la riflettanza dei punti rilevati

ricade al suo interno. Nel caso contrario non è conosciuta la natura del voxel, cioè la rappresentazione fornita non effettua una distinzione tra la configurazione di *libero* e *sconosciuto*. Tale struttura dati facilita l'accesso in termini computazionali (tempo costante) agli elementi della griglia 3D, ma ha il forte svantaggio di vincolare la *point-cloud* di partenza ad una estensione massima se si vuole mantenere sempre il medesimo grado di risoluzione della mappa.

All'interno della letteratura il lavoro di Hornung et al. [15] fornisce un'estensione delle *voxel map* mediante un framework open-source chiamato *octomap* scritto in C++.

Le octomap sono basate sugli *octree*: strutture gerarchiche per la suddivisione spaziale del 3D [26]. Ogni nodo dell'albero rappresenta un volume della griglia che viene suddiviso ricorsivamente in otto sotto-voxel fino a che non viene raggiunta la risoluzione minima della stessa. Andando ad escludere i nodi dell'albero da un suo determinato livello di profondità è possibile ottenere una rappresentazione della mappa con una minore risoluzione. Un esempio delle funzionalità appena descritte è mostrato in figura 2.7. Ogni nodo viene rappresentato mediante un valore *Booleano* che indica lo stato di occupazione del voxel. Nel caso il voxel risulti *occupato*, allora il relativo nodo dell'albero viene inizializzato, garantendo che la struttura dati venga memorizzata in maniera efficiente e compatta. In questa configurazione è resa possibile la rappresentazione esplicita dello spazio *libero*. Grande vantaggio delle octomap risiede nel tempo di computazione richiesto per effettuare query su un albero. Dato un albero di profondità d il tempo di esecuzione è $O(d) = O(n)$ cioè lineare. Altra caratteristica di octomap è quella di permettere la creazione di alberi che tengano traccia di informazione aggiuntive oltre al

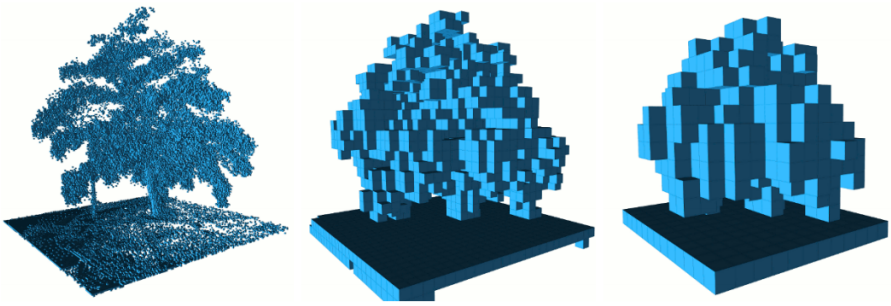


Figura 2.7: Passaggio da pointcloud a rappresentazione a voxel mediante octomap con due tipi di risoluzione diversi. Immagine presa da [15].

valore di occupazione di un *voxel*. Ad esempio, viene fornita un'estensione della classe dell'albero base in grado di assegnare ad un voxel il dato colore (figura 2.8).

2.4 Modello camera pinhole

Di seguito verrà fornita una spiegazione del modello di proiezione utilizzato nella gran parte dei sensori camera oggigiorno esistenti. Dapprima verrà fornita un'introduzione teorica del modello per capirne appieno i concetti di base e successivamente ci si addentrerà in una spiegazione più realistica col fine di darne una descrizione più vicina al funzionamento reale dei sensori camera. Notare che le nozioni descritte di seguito stanno alla base di parte del lavoro di tesi svolto e che hanno permesso di applicare la tecnica di *raytracing* che verrà spiegata nella sezione 2.4.1.

Il modello che rappresenta il funzionamento dei sensori camera tradizionali è quello comunemente noto come *pinhole* e fornisce le equazioni matematiche che definiscono la *proiezione prospettica* dei punti di una scena.

L'idea che sta alla base del modello è quella di proiettare i punti visibili di una scena 3D su un piano immagine, facendoli passare attraverso un punto chiamato *centro di proiezione*. Conoscendo le coordinate del centro di proiezione e di un punto della scena è possibile ottenere l'equazione della retta passante per i due punti e di conseguenza l'intersezione di quest'ultima con il piano immagine. Tale piano è definito ad una distanza predefinita dal centro di proiezione e prende il nome di *lunghezza focale*.

Il problema di questa definizione è che fornisce un modello ideale e non verosimile rispetto al funzionamento reale di una camera. Ad esempio, il centro di proiezione è rap-

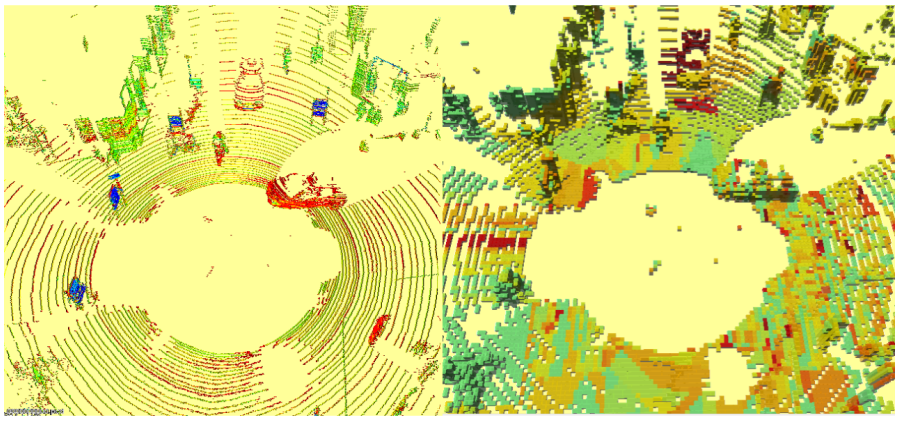
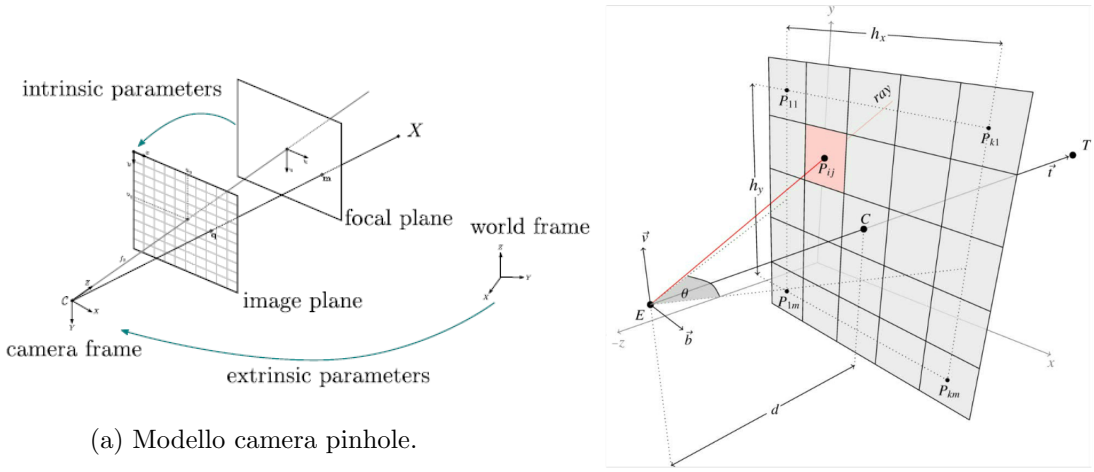


Figura 2.8: A sinistra viene rappresentata una pointcloud di un scansione LiDAR mettendo in risalto la riflettanza dei punti. A destra viene fornita la rappresentazione della stessa scansione effettuando la voxelizzazione tramite octomap. Alla mappa a voxel viene associata una scala colore che varia in base al volore di riflettanza.

presentato secondo un punto infinitesimo che non è riproducibile nella realtà. Da questa problematica deriva la necessità di utilizzare un'ottica in grado di risolvere il problema dei *cerchi di sfocamento*, ma che allo stesso tempo introduce il problema delle distorsioni. Le distorsioni possono essere di due tipi: *radiali* o *tangenziali*. Le distorsioni radiali sono quelle più intense e si dividono a loro volta in *cuscinetto* e *barilotto* e modificano le distanze percepite dei punti della scena. Le distorsioni tangenziali sono solitamente meno intense e modificano la posizione del punto immagine lungo una retta ortogonale rispetto alla retta di proiezione. Tali distorsioni possono essere stimate mediante la calibrazione della camera e ridotte applicando del post-processing all'immagine risultante tramite l'operazione di *undistort*.

Nel discreto il piano immagine di una camera viene definito come una griglia di *pixel* (*picture element*) e il numero di questi elementi ne definisce la risoluzione.

In figura 2.9a viene mostrato il modello pinhole di una camera tradizionale mettendo in evidenza quelli che sono i parametri intrinseci ed estrinseci.



(a) Modello camera pinhole.

(b) Tecnica di raytracing.

Figura 2.9: Nell'immagine (a) viene mostrato schematicamente il modello camera pinhole e in figura (b) viene raffigurata graficamente il funzionamento della tecnica di raytracing.

2.4.1 Tecnica di raytracing

Lavorando con mappe tridimensionali è possibile applicare il modello di proiezione appena descritto per ottenere delle immagini delle scena rispetto ad un punto di osservazione. Una tecnica nata dal campo computer grafica che permette di applicare un metodo differente di proiezione prende il nome di *raytracing*.

La tecnica di *raytracing* inverte il paradigma di proiezione: la scena non viene ricostruita sul piano immagine basandosi sulle proiezioni dei punti 3D, ma partendo dal punto di vista camera e considerando i parametri intrinseci di quest'ultima è possibile emettere delle rette di proiezione (*raggi*) che attraversano la scena cercando le intersezioni con gli oggetti. Le equazioni di tali rette vengono definite considerando le coordinate 3D del punto di proiezione e il punto centrale dei singoli pixel. Notare che per poter applicare una tecnica di questo tipo è necessario descrivere la geometria della scena mediante poligoni (*mesh*) o solidi come i *voxel*, poiché intersecare esattamente un punto nello spazio mediante una retta di proiezione risulterebbe estremamente improbabile.

Questa tecnica si è affermata da molti anni nel campo della computer grafica e viene utilizzata, ad esempio, per effettuare la stima della luce locale all'interno di scene 3D sintetiche [36], ma può trovare applicazioni anche in contesti differenti come verrà spiegato all'interno del lavoro svolto (capitolo 4).

2.5 Creazione e processing di immagini LiDAR

Per LiDAR (*Light Detection and Ranging*) si intende un sensore laser in grado di reperire informazione di distanza e riflettanza degli oggetti all'interno di un ambiente. Mentre la distanza banalmente descrive le coordinate spaziali di un punto della scena di un oggetto rilevato, la riflettanza rappresenta una grandezza adimensionale che fornisce il quantitativo di luce riflessa da un oggetto. Essa dipende dai seguenti fattori:

- Proprietà del materiale
- Distanza dell'oggetto colpito dal raggio
- Angolo di intersezione raggio-oggetto
- Fattori ambientali esterni che modificano proprietà di riflessione del materiale (e.g. asfalto asciutto, asfalto bagnato)

I sensori LiDAR risultano essere molto affidabili dato il loro basso tasso di errore, ma la loro utilità viene meno in presenza di condizioni climatiche avverse come, ad esempio, pioggia, neve o presenza di nebbia.

Un altro problema legato alla tipologia del sensore è quella del costo, che aumenta sensibilmente all'aumentare della risoluzione richiesta. Questa caratteristica può risultare un forte svantaggio perché ne limita il suo utilizzo in contesti dove è possibile affrontare costi elevati. Notare che tale sensore può essere utilizzato non solo per ottenere singole scansioni, ma anche per effettuare operazione di *mapping* allineando più scansioni consecutive.

Come mostrato in precedenza (2.2) esistono lavori in letteratura che utilizzano i dati LiDAR per la creazione di immagini, tuttavia applicare il modello di proiezione camera descritto porterebbe all'ottenimento di immagini non realistiche a causa di due principali problematiche:

- Sparsità dei dati all'interno dell'immagine
- Rumore introdotto da punti non visibili dall'osservatore, ma comunque proiettati sul piano immagine

La prima problematica deriva dal fatto che la rappresentazione fornita dal LiDAR è di tipo puntale (figura 2.10), mentre la seconda casistica si verifica quando si lavora con una mappa precedentemente creata a partire da molteplici scansioni.

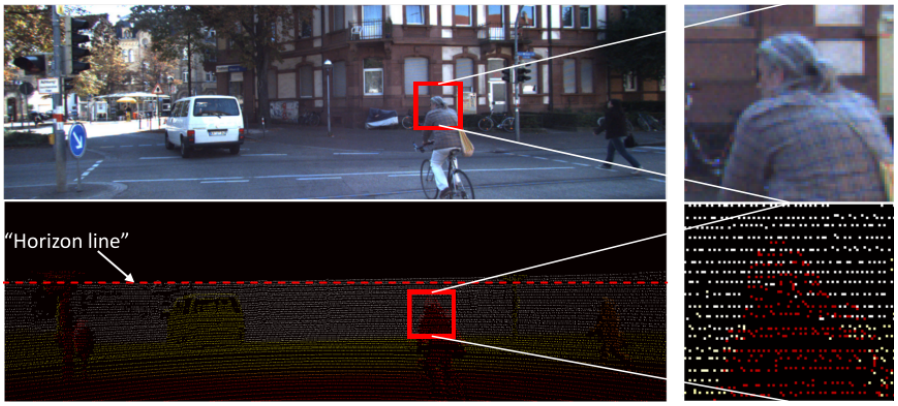


Figura 2.10: Esempio di immagine di una scena reale (in alto) e della sua rappresentazione proiettando i punti di una scansione LiDAR su un piano immagine. Notare come si verifichi il fenomeno della sparsità delle proiezioni. L'immagine è presa da [33].

2.5.1 Tecniche di upsampling

Per quanto concerne la problematica delle immagini sparse esiste, ad esempio, il lavoro di Premebida et al. [32] che tratta il task di detection di pedoni andando a fondere dato camera RGB e dato LiDAR. All'interno dell'articolo ci si approccia a livello immagine per risolvere la sparsità dei dati andando ad applicare tecniche di filtraggio. In particolare viene traslata una maschera lungo l'immagine a cui si applica una versione particolare di *Bilateral Filter*. Per *Bilateral Filtering* si intende una tecnica di filtraggio utilizzata su immagini che viene adottata per rimuovere del rumore mediante *smoothing*, cioè il valore del pixel centrale della maschera applicata all'immagine viene mediato in base al valore dei propri vicini. Rispetto ad altri filtri di *smoothing* la sua caratteristica principale è quella di preservare i contorni degli oggetti all'interno dell'immagine. Di contro possiede lo svantaggio di essere un filtro non lineare da cui ne consegue una maggiore richiesta computazionale. Il criterio con cui viene assegnato il valore del pixel centrale della maschera tiene in considerazione la distanza tra quest'ultimo e i pixel del vicinato. In particolare nel lavoro citato, dove questo tipo di filtraggio viene applicato all'immagine LiDAR, viene pesata anche la distanza tra i valori di profondità dei pixel come definito nella formula seguente.

$$D_p = \frac{1}{W_p} \sum_{q \in N} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_q|) I_q \tag{2.5.1.1}$$

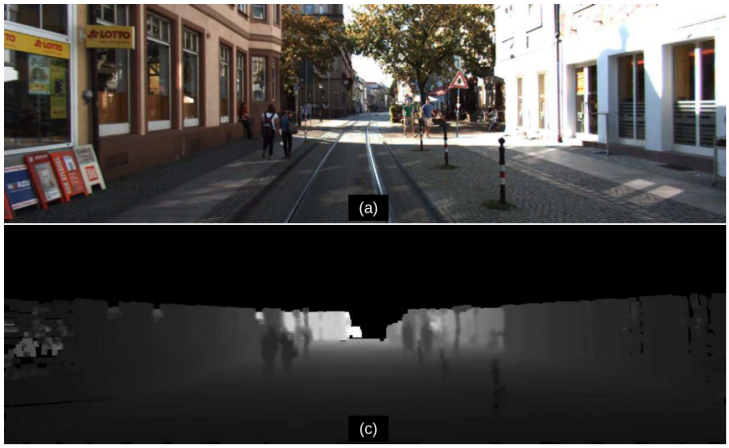


Figura 2.11: Immagine RGB (in alto) e immagine LiDAR (in basso) ottenuta mediante bilateral filtering. Immagine presa da [32]

G_{σ_s} rappresenta la funzione che pesa la distanza dei pixel all'interno della maschera, mentre G_{σ_r} pesa la distanza tra pixel in base al valore di profondità ed infine W_p rappresenta un coefficiente di normalizzazione per far sì che i pesi abbiano somma uno.

Un altro lavoro che utilizza Bilateral Filtering per risolvere il problema della sparsità delle immagini LiDAR è quello di Premebida et al. [33]. Un esempio di risultati ottenuti utilizzando questo approccio è mostrato in figura 2.11.

Nel lavoro precedentemente citato di Asvadi et al. [1], il problema della sparsità delle immagini viene affrontato sia per il dato di profondità che per quello di riflettanza. Per la generazione di immagini LiDAR ad alta risoluzione viene applicata le tecnica di *triangolazione di Delunay* col fine di creare una mesh triangolare lungo tutta l'immagine partendo dai punti proiettati (2.1a). Questa tecnica ha la caratteristica di evitare la generazione di triangoli aventi angoli molto acuti, assicurando in questa maniera una migliore ricostruzione. Un esempio di mesh ottenuta mediante triangolazione di Delunay è fornita in figura 2.12. Nel lavoro di Asvadi, una volta applicata questa tecnica, si effettua una interpolazione del valori dei punti immagine, che ricadono all'interno di un triangolo, basandosi sui valori dei suoi vertici. La tecnica utilizzata all'interno dell'articolo è quella di *nearest-neighbors*.

Le tecniche appena elencate vengono definite di *upsampling*. Nonostante riescano ad affrontare in maniera funzionale il problema dalla sparsità delle proiezioni LiDAR, hanno il difetto di fornire una rappresentazione verosimile e non veritiera della scena mettendo

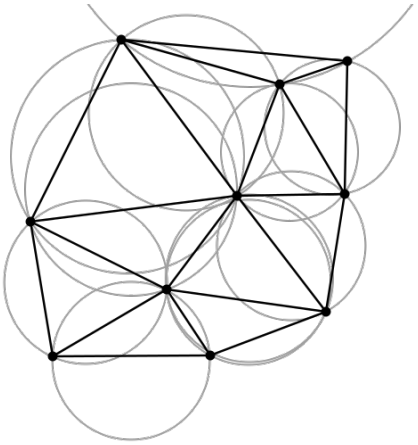


Figura 2.12: Esempio di triangolazione di Delunay nel 2D.

in relazione i punti sul piano immagine che nel 3D potrebbero distare anche di parecchi metri. Inoltre vi è un difetto intrinseco nella rappresentazione puntuale che da un lato permette di avere una rappresentazione più fine, ma dall'altro non fornisce conoscenza riguardo a gran parte dello spazio scansionato dal sensore. Data questa peculiarità della rappresentazione 3D adottata è impossibile recuperare l'informazione esatta delle regioni di cui non si ha informazione.

Ai fini del lavoro svolto queste tecniche sono state scartate data la forte assunzione di relazione che viene accettata per i punti proiettati sul piano immagine. In particolare in ambito automotive è necessario che il singolo dato rappresenti un'informazione veritiera viste le criticità costanti che un mezzo a guida autonoma deve affrontare durante le operazioni di movimento.

2.5.2 Tecniche per immagini a bassa risoluzione

Altra tecnica esistente in letteratura è quella mostrata nel lavoro di Vaquero et al. [44] precedentemente citato. Per l'ottenimento di immagini dense non viene applicata nessuna tecnica di upsampling, ma i punti della scansioni 3D vengono proiettati su una superficie cilindrica tenendo in considerazione il numero di piani di scansione del sensore laser, cioè un LiDAR a 64 piani. Questa tecnica permette di ottenere immagini dense, ma a bassa risoluzione. Per effettuare questo tipo di proiezione viene prima effettuato un passaggio da *coordinate cartesiane* a *coordinate sferiche* dei punti 3D. Il vantaggio di questa rappresentazione sta nella semplicità del processo di generazione dell'immagine,

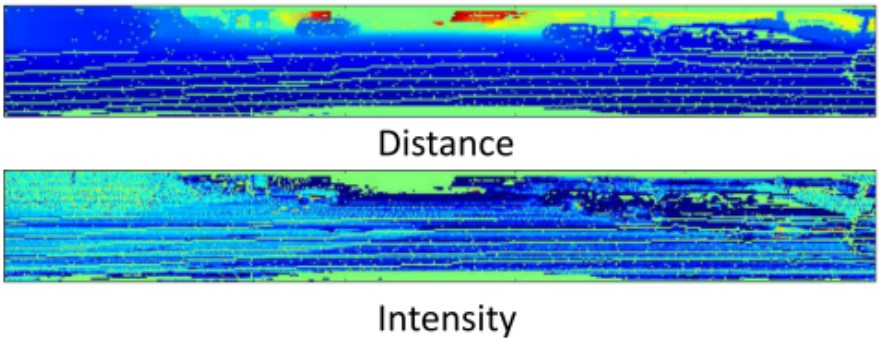


Figura 2.13: Esempio di immagini LiDAR a bassa risoluzione ottenute mediante proiezione di una scansione su una superficie cilindrica.

ma di contro fornisce immagini a bassa risoluzione e con un forte sbilanciamento tra le dimensioni di altezza e larghezza di quest'ultima.

Dato un punto definito in coordinate cartesiane nel 3D $P(x, y, z)$, la sua proiezione P(r, c) su una superficie cilindrica viene definita nella maniera seguente.

$$c = \lfloor \frac{atan2(x, y)}{\Delta\theta} \rfloor$$
$$r = \lfloor \frac{atan2(z, \sqrt{x^2 + y^2})}{\Delta\phi} \rfloor$$

dove $\Delta\theta$ e $\Delta\phi$ rappresentano rispettivamente la variazione dell'angolo di *elevazione* e di *azimuth* del LiDAR.

2.5.3 Approcci basati su mesh

Le tecniche fino ad ora mostrate affrontano il problema della sparsità delle immagini LiDAR focalizzandosi in un contesto dove la proiezione dei punti riguarda una singola scansione. In questa casistica risulta semplice discriminare i punti che non sono visibili da un osservatore (e.g. punti che risiedono ad una profondità minore rispetto al piano immagine). Se invece si considera che le scansioni possano essere molteplici, come nel caso dell'utilizzo di mappe, allora le tecniche mostrate fino ad ora risultano inefficaci. Il motivo di tale inefficacia deriva dai limiti della rappresentazione puntuale della scena che non fornisce informazione riguardo a elementi geometrici più complessi come, ad esempio, le superfici degli oggetti. Di conseguenza comprendere se un punto è visibile dall'osservatore oppure è occluso da un altro elemento della scena non risulta un lavoro

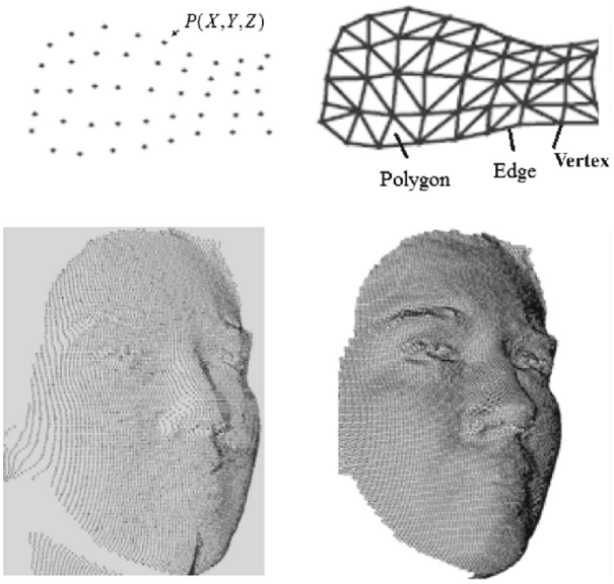


Figura 2.14: In figura viene fornita a sinistra la rappresentazione puntuale di un volto, mentre a destra la sua ricostruzione solida mediante mesh.

semplice. Ad esempio, in uno scenario urbano di cui è fornita una mappa LiDAR, non è desiderabile proiettare punti che risiedono al di là di una parete di un edificio osservato, poiché non sarebbero visibili dal punto di vista dell'osservatore.

Per far fronte a questa problematica è necessario ricostruire la geometria della scena a partire dalla rappresentazione puntuale della mappa utilizzando ad esempio un approccio basato su mesh. Per mesh si intende un insieme di vertici (punti) che definiscono i poligoni costituenti gli oggetti di un scena permettendo una ricostruzione solida della stessa. In figura 2.14 viene mostrato il passaggio della rappresentazione puntuale 3D a quella mediante mesh.

Il lavoro di Pascoe et al. [30] mostra un approccio di questo tipo andando a sfruttare immagini LiDAR ottenute da mappe costitute da mesh triangolari congiuntamente ad immagini RGB. Il fine del lavoro è quello di affrontare il problema di localizzazione di un veicolo e la calibrazione di una camera. In particolare viene sfruttato un LiDAR 2D per la generazione delle scansioni. Per la creazione della mappa vengono accumulate le scansioni secondo tempi di acquisizione crescenti e ad ogni scansione inclusa si effettua una triangolazione con quelle precedenti, con il fine di ottenere una mesh triangolare i cui poligoni sono il più possibile consistenti con la realtà della scena. Per garantire una buona

Figura 2.15: A sinistra l'immagine RGB rappresentante una strada. A destra la corrispondente immagine LiDAR ottenuta mediante l'approccio a mesh.

qualità della mappa generata, un triangolo viene incluso nella mesh se e solo se tutti i suoi lati hanno lunghezza inferiore ad un certa soglia. In particolare nell'articolo viene specificato che tale soglia è fissata ad un metro di lunghezza. Conoscendo i parametri intrinseci ed estrinseci del sensore camera, i triangoli generati vengono proiettati sul piano immagine per ottenere immagini di depth ad alta risoluzione.

Un altro lavoro che fornisce un metodo di ricostruzione di mesh in *real-time* è quello di Piazza et al. [31]

Il vantaggio degli approcci tramite mesh è che risultano particolarmente efficaci per ricostruire la geometria della scena. Tuttavia per garantire una buona qualità di ricostruzione, la *point-cloud* di partenza deve essere costituita da un insieme di punti denso in rapporto alle dimensioni della mesh che si desidera generare. Come affermato in precedenza (2.3.2), all'aumentare dei punti contenuti nella *point-cloud* aumenta la richiesta computazionale per gestire queste strutture dati. Inoltre la densità dell'insieme di scansioni è dipendente dal dataset di partenza e non sempre è possibile utilizzare un approccio di questo tipo.

A causa di queste ultime problematiche questa tecnica non è stata considerata ai fini del lavoro svolto, ma si è proceduto ad utilizzare una rappresentazione basata su *voxel* per la ricostruzione solida della scena. Successivamente si è applicata la tecnica di raytracing per ottenere immagini LiDAR dense. I dettagli di questo processo verranno forniti successivamente quando si parlerà del lavoro svolto (capitolo 4).

2.5.4 Trattare immagini sparse mediante CNN

Nei lavori mostrati fino ad ora risulta importante fornire ad una rete neurale convoluzionale delle immagini LiDAR dense, poiché processare immagini sparse generalmente porta ad un decremento delle performance (e.g. difficoltà nel trovare corrispondenze tra immagini RGB e di depth) oltre che ad uno spreco di risorse computazionali, cioè si processano immagini la cui maggioranza di valori è nullo a causa della mancanza di informazione. Inoltre esiste la problematica su come fornire una rappresentazione che effettui una distinzione tra elementi osservati ed elementi non conosciuti.

Il lavoro di Uhrig et al. [43] si focalizza sulla creazione di una CNN che sia invariante rispetto la sparsità di un'immagine, senza utilizzare alcun tipo di tecnica di *preprocessing* come quelle elencate in precedenza. Il problema viene formulato nella seguente maniera: Sia f una funzione che mappa un'immagine in input appartenente ad X ad un output appartenente a Y rappresentante anch'esso un'immagine, dove gli elementi $x(u, v) \in X$ rappresentano i pixel che sono parzialmente osservati. Nel contesto in cui si utilizzano immagini con valori sparsi si pone il problema su come rappresentare gli elementi di cui non si ha informazione. Dato che una possibile inizializzazione $x_{u,v} = 0$ di tali elementi porta all'impossibilità di distinguere i valori osservati da quelli sconosciuti (alcuni valori osservati potrebbero essere uguali a zero), è preferibile definire una maschera che tenga traccia di questa distinzione. Di conseguenza è necessario definire una funzione $o(x_{u,v})$ che restituisca valori booleani in base al fatto che gli elementi immagine di input abbiano un valore conosciuto o meno:

$$o(x_{u,v}) = \begin{cases} 1, & x_{u,v} \text{ è conosciuto.} \\ 0, & \text{altrimenti.} \end{cases} \tag{2.5.4.1}$$

Utilizzando operazioni di convoluzione standard, avendo come input l'immagine sparsa originale e la relativa maschera, vi è la possibilità che la rete apprenda i pesi in maniera tale da considerare solo gli elementi osservati. Tuttavia all'interno del lavoro viene creato un nuovo operatore di convoluzione per rendere questa associazione immediata ed esplicita. La formula dell'operatore di convoluzione è definita come segue:

$$f_{u,v}(x, o) = \frac{\sum_{i,j=-k}^{k} o_{u+i,v+j} x_{u+i,v+j} w_{i,j}}{\sum_{i,j=-k}^{k} o_{u+i,v+j} + \epsilon} + b \tag{2.5.4.2}$$

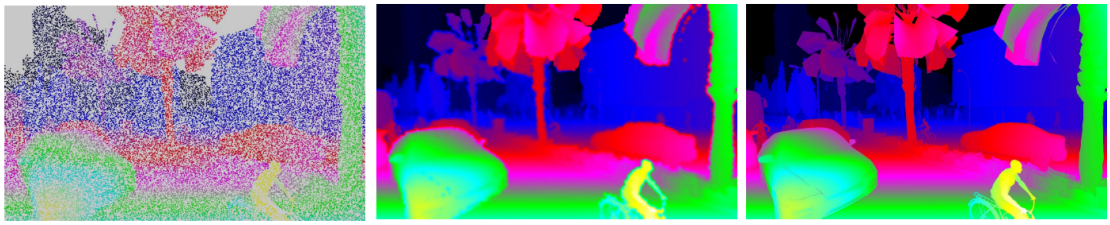


Figura 2.16: A sinistra viene mostrata l'immagine di profondità sparsa, al centro l'immagine densa ottenuta col metodo proposto e a destra l'immagine di groundtruth

.

con dimensione del kernel fissata a $2k+1$, bias b e pesi w. Notare che il denominatore ha lo scopo di normalizzare il risultato e per evitare divisioni per zero viene sommato un valore infinitesimo ϵ.

Per tenere traccia della distinzione tra elementi osservati e non conosciuti tra le successioni di layer convolutivi, si utilizzano delle maschere ottenute mediante una particolare operazione di *max-pooling* definito come segue:

$$f^{o}_{u,v}(o) = \max_{i,j=-k,\dots,k} o_{o+i,o+j} \tag{2.5.4.3}$$

La formula restituisce 1 se almeno uno degli elementi della sotto-regione definita dal kernel di *max-pooling* contiene almeno un valore osservato, 0 altrimenti.

L'obiettivo finale del lavoro è quello di predirre delle mappe di profondità dense mediante l'utilizzo di una rete neurale. Un esempio è riportato in figura 2.16.

Anche nel lavoro precedentemente citato di Schneider et al. [39] si affronta il problema della sparsità delle proiezioni LiDAR a livello di rete neurale. In particolare prima di processare le immagini di profondità mediante i layer convolutivi si applica l'operazione di *max-pooling* sull'immagine LiDAR in maniera tale da dilatare l'informazione di profondità dell'immagine di partenza.

Questi approcci, rientrando anch'essi nelle tecniche di upsampling, non sono stati considerati ai fini del lavoro svolto, ma comunque forniti per una maggiore chiarezza sulle possibili tecniche.

Capitolo 3

Descrizione del lavoro svolto e motivazioni

Successivamente alla trattazione dei lavori presenti in letteratura, inerenti alle tematiche affrontate durante il periodo di tesi, in questo capitolo verranno descritte le motivazioni che hanno portato allo svolgimento del presente lavoro. Inoltre verrà fornita una visione ad alto livello del processo sviluppato per affrontare il problema della localizzazione: partendo dalla modellazione del problema, fino ad arrivare all'implementazione della rete neurale convoluzionale utilizzata.

3.1 Problema della localizzazione

Nell'ambito della robotica mobile, dove per robot si intende un veicolo, il problema della localizzazione risulta essere uno dei temi di maggiore interesse all'interno della comunità scientifica. Per localizzazione si intende la stima della posizione e orientamento (*pose*) di un robot all'interno dell'ambiente in cui esso deve operare. La localizzazione *outdoor*, nel campo *automotive*, costituisce un problema complesso, data la dinamicità dell'ambiente di navigazione. Ad esempio, in uno scenario urbano sono presenti ostacoli che possono essere sia statici (e.g. edifici) sia dinamici (e.g. veicoli) ed inoltre, la presenza di molteplici attori (e.g. gli altri utenti della strada) rende elevata l'imprevedibilità dell'ambiente. In particolare, la localizzazione in determinati contesti costituisce un problema critico, poiché eventuali errori nella stima della pose del veicolo potrebbero mettere a repentaglio la vita dei passeggeri a bordo e degli altri utenti della strada.

In ambito urbano l'utilizzo di sensori come, ad esempio, il GPS non risultano adeguati per fornire una corretta pose del robot, poiché la presenza degli edifici interferisce con il segnale del sensore. Un altro possibile approccio si basa su una tecnica chiamata *odometria*, andando ad utilizzare le misurazioni del movimento rotazionale delle ruote per stimare la pose del veicolo durante la navigazione. Tuttavia, tali misurazioni risultano errate a causa, ad esempio, dello slittamento delle ruote in presenza di condizioni climatiche avverse o delle diverse conformazioni dei pneumatici.

L'utilizzo di una conoscenza pregressa dell'ambiente, come una mappa, permette di facilitare il compito della localizzazione, poiché non rende necessario preoccuparsi di definire a *run-time* una struttura del luogo in cui il veicolo sta navigando. L'assunzione che sta alla base di questo approccio è che la configurazione della scena sia statica e che non possa variare nel corso del tempo. Di conseguenza, all'interno della rappresentazione fornita non si considerano gli elementi dinamici della scena (e.g. veicoli e pedoni).

3.2 Modellazione del problema

La localizzazione, in ambito *automotive*, ha come scopo quello di stimare la posizione e l'orientamento (*pose*) di un veicolo, generalmente espressi rispetto ad un sistema di riferimento principale. La relazione, che associa la pose del veicolo rispetto al sistema di riferimento, ad esempio, di una mappa, è descritta mediante una matrice di *rototraslazione*.

All'interno del lavoro svolto, il robot viene considerato come un veicolo dotato di un sensore camera che gli permette di osservare costantemente la scena durante le sue fasi di guida. In questa configurazione, la posizione e l'orientamento del sensore camera vengono descritti mediante un pose p_{camera}, che è definita rispetto al sistema di riferimento del veicolo $p_{vehicle}$. La relazione geometrica che intercorre tra queste due pose è ottenuta mediante l'operazione di calibrazione, da cui vengono stimati i parametri estrinseci della camera. All'interno del lavoro di tesi tali parametri camera sono conosciuti, così come sono ben definiti i parametri intrinseci del sensore stesso. In figura 3.1 vengono mostrate in maniera schematica le relazioni dei sistemi di riferimento associati ai diversi elementi appena descritti.

Nel lavoro di tesi svolto, si è deciso di utilizzare una mappa per rappresentare l'ambiente di navigazione del veicolo. Essa è costituita da un insieme di scansioni laser acquisite mediante un sensore LiDAR (Light Detection and Ranging), formando una struttura

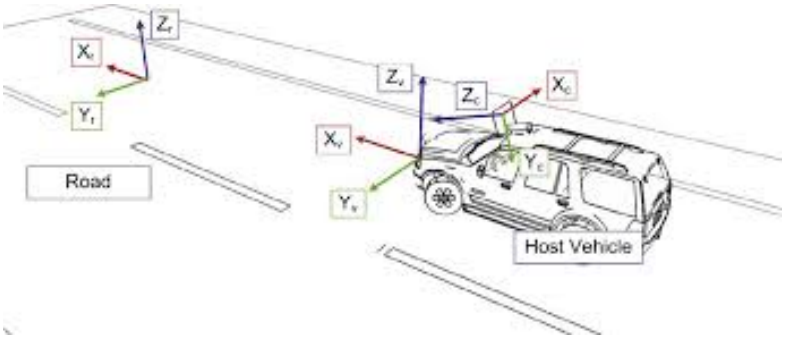


Figura 3.1: Schema di un veicolo dotato di un sensore camera con evidenziati i sistemi di riferimento. Si può notare come il sistema di riferimento associato al veicolo sia ruotato rispetto a quella della camera.

chiamta point-cloud. Il sistema sviluppato ha come input una pose p_{init}, ottenuta mediante un agente terzo come, ad esempio, un sensore GPS, che fornisce la posizione e l'orientamento del veicolo all'interno della mappa. Tuttavia, alla pose ottenuta da tale sensore è associata dell'incertezza causata dall'imprecisione delle sue misurazioni. Dati i parametri estrinseci della camera e data la pose del veicolo in input p_{init}, è possibile calcolare la pose p_{camera} del sensore camera, a cui, a causa dell'incertezza legata alla pose p_{init}, viene propagato l'errore di localizzazione iniziale.

Conoscendo i parametri intrinseci del sensore camera e la sua pose p_{camera} soggetta ad errore, è possibile proiettare gli elementi della mappa all'interno di un piano immagine utilizzando il modello di proiezione *pin-hole*. L'obiettivo è quello di simulare il funzionamento del sensore camera montato sul veicolo ed in particolare, osservare la porzione di mappa secondo il punto di vista fornito dalla pose rumorosa di input. Dalle proiezioni degli elementi della mappa, all'interno del piano immagine, si ottengono le informazioni di profondità e riflettanza degli oggetti osservati, andando così a creare un'immagine LiDAR.

L'immagine LiDAR ottenuta e l'immagine RGB definiscono due punti di osservazione differenti e rappresentano l'input di una rete neurale convoluzionale. Tale rete neurale ha il compito di stimare l'errore di localizzazione iniziale sulla pose p_{init}, andando a confrontare i punti di vista forniti dalle due immagini.

L'approccio appena descritto, riporta il problema della localizzazione di un veicolo rispetto al sistema di riferimento del sensore camera. Tuttavia, avendo conoscenza dei parametri estrinseci di quest'ultimo è possibile ricondursi dalla pose p_{camera} alla pose

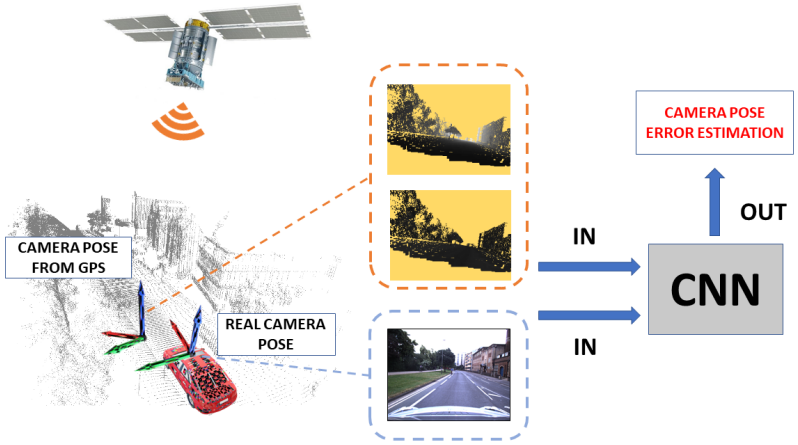


Figura 3.2: In figura viene mostrato uno schema del problema affrontato durante il lavoro di tesi. Data la pose GPS in input, si ricava la pose p_{camera} che, come raffigurato, si dimostra differente rispetto a quella corretta. Tale pose costituisce un punto di vista differente rispetto al sensore camera reale e da esso si ricava la porzione di mappa osservata. La CNN prende in input le immagini ottenute dai due punti di vista ed effettua la predizione sull'errore iniziale.

$p_{vehicle}$. In figura 3.2 viene descritto graficamente il problema appena modellato.

3.3 Descrizione del lavoro svolto

Nella fase iniziale del lavoro di tesi ci si è concentrati nella ricerca di un dataset tra quelli presenti nella comunità scientifica. In particolare si è mostrato necessario che tale dataset contenesse al suo interno le acquisizioni di sensori laser, camera e GPS, registrate durante la navigazione di un veicolo in uno scenario urbano. Tali informazioni si sono richieste ai fini dell'ottenimento di mappe 3D e di immagini RGB dell'ambiente di navigazione del veicolo. Tra i dataset inzialmente considerati vi sono il *KITTI dataset* e l'*Oxford Robotcar dataset*. Tuttavia all'interno del dataset di KITTI si sono riscontrate delle problematiche legate al corretto allineamento delle scansioni LiDAR e di conseguenza si è preferito l'utilizzo del dataset di Robotcar, poiché le mappe generate hanno mostrato una maggiore consistenza rispetto la scena osservata dal sensore camera.

Una volta scelto il dataset di partenza, si è sviluppata una pipeline automatizzata

per la creazione di un nuovo dataset. L'obiettivo è stato quello di generare immagini LiDAR, immagini RGB e le *groundtruth* delle pose camera soggette ad errore.

Inizialmente per la generazione delle mappe, partendo dal Robotcar dataset, si è utilizzata una libreria python chiamata *robotcar-sdk*, messa a disposizione dai creatori del dataset. Di tale libreria è stata effettua la re-implementazione di alcune sue funzionalità per ottimizzare i tempi di generazione delle mappe andando ad effettuare un sottocampionamento delle scansioni laser. Questo processo si è rivelato necessario, poiché in fase, ad esempio, di frenata del veicolo, vengono accumulate innumerevoli scansioni che non sono necessarie ai fini della corretta ricostruzione 3D della scena. Al termine di questa fase viene ottenuta una point-cloud che si sviluppa per $100m$ rispetto alla direzione di movimento del veicolo. Notare, che la point-cloud fornisce una rappresentazione puntuale della scena e le coordinate dei punti sono espresse mediante il sistema di riferimento camera.

Data la point-cloud generata, si è proceduto a sviluppare un metodo per la ricostruzione solida della scena. In particolare, la rappresentazione mediante voxel è stata scelta come tipologia di ricostruzione. Gli approcci basati su mesh non sono stati considerati, poiché computazionalmente onerosi rispetto all'hardware a disposizione, inoltre essi garantiscono una buona ricostruzione della scena solamente in presenza di point-cloud con elevate densità di punti. Per il processo di voxelizzazione si è utilizzato un framework C++ di nome Octomap, che permette la creazione di mappe a voxel basandosi su particolari strutture ad albero chiamate *octree*. Della classe rappresentante l'albero si è dovuto effettuare una sua re-implementazione per permettere di tenere traccia del dato di riflettanza della mappa originale. Inoltre, è stato sviluppato un processo per associare ai voxel della mappa un valore di riflettanza partendo dai punti appartenenti alla point-cloud. In particolare, si effettua la ricerca dei 5 punti più vicini rispetto al centro del singolo voxel andando a calcolare una media dei valori di riflettanza. La ricerca viene effettuata andando ad utilizzare delle particolare strutture dati chiamate *kd-tree* utilizzate per un partizionamento efficiente dello spazio 3D. La ricerca dei punti all'interno della point-cloud viene effettuata mediante *nearest neighbor search*. La libreria utilizzata per la manipolazione delle point-cloud prende il nome di PCL (Point Cloud Library). Inoltre, sono state sperimentate diverse risoluzioni della mappe con il fine di creare dataset differenti.

Una volta ottenuta la mappa a voxel si è proceduto a perturbare la pose camera

iniziale secondo un rumore casuale e ad applicare la tecnica di raytracing. Il raytracing è una tecnica di proiezione che, date delle rette definite dai parametri intrinseci di una camera e le coordinate dei centri dei pixel, permette di ricercare le intersezioni raggio-voxel della mappa creando in questo modo un'immagine. Questa tecnica permette di acquisire un'immagine LiDAR avente due canali: uno di profondità e uno di riflettanza. Dato il rumore sulla pose camera, che simula l'errore di localizzazione iniziale, si ottiene un'immagine che rappresenta un punto di vista differente rispetto a quello del sensore camera montato sul veicolo.

Per diminuire i tempi di computazione riguardanti la creazione del dataset e fornire una maggiore variabilità di esempi, si è applicata la tecnica di *data augmentation*. Data la mappa a voxel 3D e l'immagine RGB, vengono applicate operazioni di mirroring per creare una rappresentazione speculare di una scena e rotazioni casuali dell'immagine camera rispetto al suo centro per simulare, ad esempio, diverse compressioni delle sospensioni del veicolo. Quest'ultima operazione viene applicata nel 3D effettuando una rotazione rispetto l'asse del sistema di riferimento camera che identifica la direzione di visione del sensore. Le operazioni appena descritte vengono applicate secondo diverse combinazioni.

A seguito delle ricerche svolte all'interno della letteratura, si è ricercata una rete neurale convoluzionale pensata per l'elaborazione di immagini RGB e LiDAR, avente lo scopo finale di effettuare la regressione di una pose. La rete neurale infine scelta prende il nome di RegNet. L'architettura delle rete è stata inizialmente modificata per permettere l'elaborazione del formato delle immagini di input provenienti dai dataset in precedenza creati. Successivamente ci si è concentrati nel ricercare e definire delle funzioni di loss applicabili alle predizioni delle traslazione e rotazioni effettuate dalla rete neurale. Mettere a confronto gli errori di traslazione e rotazione a livello di rete richiede che vengano effettuate alcune considerazioni, poiché essi sono definiti su scale di valori diverse ed inoltre, rotazione e traslazione definiscono due concetti spaziali differenti. Per l'implementazione della CNN è stato utilizzato un framework python di nome PyTorch, molto popolare nel campo dell'apprendimento automatico, inoltre sono stati definiti degli script per gestire il dataset utilizzato ed effettuare le operazioni di training e testing della rete.

Nella fase finale del lavoro si sono definite le modalità di test con cui osservare le performance della rete neurale. In particolare sono stati messi in evidenza i diversi

dataset utilizzati, le tipologie di loss applicate e le informazioni di input fornite alla rete. Per ogni test i risultati vengono mostrati mettendo in evidenza separatamente le configurazioni di pesi della rete che portano ad un errore minore sulle rotazione e sulle traslazioni. Definire una metrica unificata per mettere a confronto l'errore di predizione sulla posizione e orientamento predetti costituisce un problema ancora aperto.

Nei capitoli successivi si entrerà maggiormente nel dettaglio dei passi appena descritti.

Capitolo 4

Dataset utilizzato e preprocessing

In questa sezione verrà mostrato il processo implementato per la creazione del dataset, che è stato successivamente utilizzato durante le fasi di apprendimento della rete neurale convoluzionale. Inizialmente verrà fornita una descrizione del dataset di Robotcar, grazie al quale si sono potute costruire le mappe LiDAR utilizzate per ottenere le immagini di profondità e riflettanza, inoltre verrà motivata la scelta di quest'ultimo rispetto ad altri dataset esistenti. Successivamente verranno mostrati in dettaglio i diversi step della pipeline di creazione. Inizialmente si procederà a descrivere il processo di costruzione della point-cloud rappresentante la mappa e del processo di voxelizzazione effettuato. Dopodiché verrà descritto il metodo di ottenimento delle immagini LiDAR e delle operazioni di data augmentation svolte, che hanno permesso di diminuire i tempi di creazione del dataset e di aumentare la varietà di esempi generati. Infine si parlerà delle informazioni di cui si è tenuta traccia per la creazione del dataset finale.

4.1 Oxford Robotcar dataset

Nel mondo automotive esistono diversi dataset messi a disposizione liberamente alla comunità scientifica, grazie ai quali si permette ai diversi gruppi di ricerca di sviluppare tecniche innovative per affrontare i problemi inerenti alla robotica mobile. Generalmente questi dataset contengono al loro interno dati provenienti da diversi sensori come, ad esempio, immagini camera, scansioni LiDAR e pose GPS che vengono acquisite durante la navigazione di un veicolo che opera in scenari urbani.

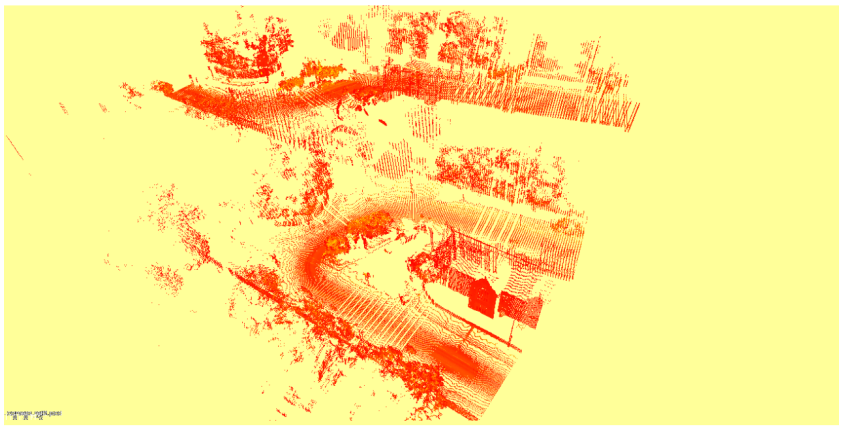


Figura 4.1: Esempio di una porzione di mappa che presenta il problema del disallineamento delle scansioni. Notare come le strade acquisite giacciano ad altitudini differenti, mentre dovrebbero essere rappresentate sullo stesso piano.

Inizialmente il lavoro di tesi si è focalizzato sulla scelta di un dataset adatto alla creazione di mappe 3D. La scelta inizialmente considerata è ricaduta su KITTI, dataset molto popolare nel campo della robotica mobile. Effettuando un'analisi preliminare di tale dataset, si è riscontrato un problema di allineamento delle scansioni facenti parte delle mappe 3D generate, a causa delle imprecisioni delle pose GPS fornite da KITTI. Di conseguenza si è deciso di non utilizzare tale dataset ai fini del lavoro svolto.

Per costruire una mappa LiDAR è necessario conoscere con precisione la pose di acquisizione delle scansioni. Se tali pose non sono precise, allora la mappa, creata andando a concatenare le diverse scansioni consecutive, non rispecchierà correttamente l'ambiente acquisito. In figura 4.1 viene mostrato il fenomeno di disallineamento delle scansioni, da cui si ottiene una mappa che non rappresenta correttamente la scena.

Una possibile soluzione per risolvere la problematica precedente è quella di provare ad allineare le diverse scansioni in maniera tale da fornire una rappresentazione della scena più coerente. Il laboratorio di robotica *IRALab* dell'Università degli studi Milano-Bicocca ha sviluppato un software chiamato *laserodo*, che permette di effettuare l'operazione di allineamento delle scansioni mediante una tecnica chiamata SLAM (*Simultaneous Localization and Mapping*). Questo processo si dimostra essere dispendioso in termini computazionali, poiché per l'allineamento delle scansioni di una mappa, ad esempio di qualche centinaio di metri, sono richieste diverse ore di computazione. Di

Figura 4.2: A sinistra vengono mostrati i diversi percorsi effettuati durante le fasi di acquisizione del dataset, a destra viene raffigurato il veicolo utilizzato insieme al sistema di sensoristica.

conseguenza, data la necessità di sviluppare un processo in grado di creare un grande quantitativo di mappe, l'utilizzo del dataset di KITTI è stato abbandonato in favore di Robotcar.

L'Oxford Robotcar dataset[1] è frutto del lavoro di Maddern et al. [25] e consiste in un insieme di immagini, scansioni LiDAR e informazioni GPS con integrazione di dati inerziali. Le fasi di acquisizione del dataset sono state svolte percorrendo più volte uno stesso tracciato all'interno della città di Oxford con un veicolo dotato di sensori. Una caratteristica del dataset riguarda la distanza totale percorsa dal veicolo durante le fasi di acquisizione, che si attesta oltre i $1000km$. Inoltre, Robotcar comprende al suo interno diverse *run* che costituiscono diverse giornate di acquisizione caratterizzate da condizioni ambientali differenti. Di seguito vengono descritti i diversi scenari ambientali rappresentati all'interno del dataset:

- Tempo soleggiato, nuvoloso o con presenza di precipitazioni
- Scenari notturni o in presenza di luce naturale
- Presenza di lavori stradali che modificano le conformazioni della scena

In figura 4.2 vengono mostrati i percorsi effettuati all'interno della città di Oxford e il sistema di sensori associato al veicolo.

Problematica simile al datset di KITTI riguarda i dati provenienti dal sensore GPS, che non garantendo pose accurate, portano all'ottenimento di mappe rumorose. Tuttavia

[1] https://robotcar-dataset.robots.ox.ac.uk, data consultazione: *gennaio 2019*

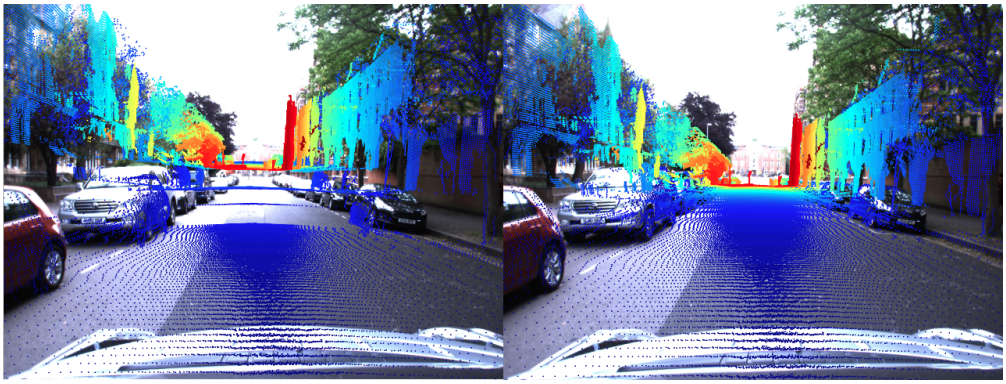


Figura 4.3: A sinistra viene mostrata la proiezione della mappa ottenuta mediante il dato GPS, mentre a destra viene mostrata la casistica in cui vengono utilizzate le pose ricavate dalla *visual odometry*.

i creatori di Robotcar mettono a disposizione le pose ottenute mediante la tecnica di *visual odometry*, che forniscono una rappresentazione della scena più coerente prendendo in considerazione brevi tratti stradali. Con tale approccio le mappe create posseggono un maggior grado di precisione e sono rese consistenti rispetto alla scena osservata dal sensore camera montato sul veicolo. In figura 4.3 vengono sovrapposte ad una immagine camera le proiezioni dei punti della mappe ottenute mediante le due tipologie di pose. Notare come l'approccio mediante *visual odometry* porti a dei risultati migliori.

Il dataset viene suddiviso in base alle giornate nelle quali sono state svolte le acquisizioni. Di conseguenza è possibile andare ad utilizzare delle run che rispondo a specifiche caratteristiche (e.g. selezione in base al tempo atmosferico). In figura 4.4 vengono mostrati degli esempi di immagini del dataset e di una point-cloud generata dall'allineamento delle scansioni.

4.1.1 Robotcar Software Development Kit

Insieme al dataset di Robtocar viene messa a disposizione dai suoi creatori una libreria *python* e *matlab* che ne permette la manipolazione[2].

[2]https://github.com/ori-drs/robotcar-dataset-sdk, data consultazione: *febbraio 2019*

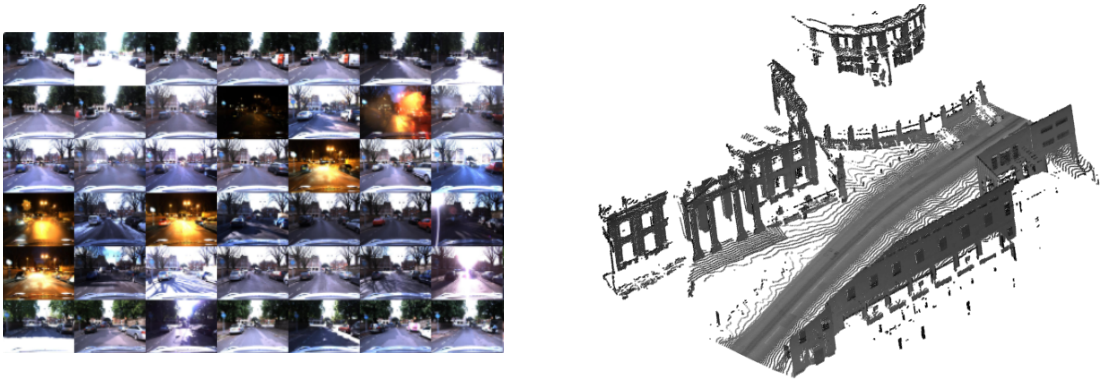


Figura 4.4: A sinistra vine mostrato un insieme di immagini acquisite durante le diverse run, mentre a destra viene mostrato un esempio di point-cloud ottenibile dal dataset.

Tale libreria contiene un insieme di script che permettono di eseguire principalmente le seguenti operazioni:

- Visualizzazione delle immagini del dataset con eventuale rimozione delle distorsioni dell'ottica della camera
- Creazione di una point-cloud combinando multiple scansioni LiDAR
- Proiezione della point-cloud su un immagine utilizzando il modello camera pinhole

Insieme alla libreria vengono forniti i modelli dei sensori camera, i quali contengono le stime dei parametri intrinseci, dei parametri estrinseci e dei coefficienti di distorsione introdotti dall'ottica.

4.2 Creazione del dataset

In questa sezione verrà descritto il framework sviluppato per generare il dataset, utilizzato successivamente per addestrare e validare la CNN implementata. Il software è stato scritto nel linguaggio C++, ma richiama l'esecuzione di script python appartenenti alla libreria associata al dataset di Robotcar. La scelta del linguaggio deriva dal bisogno di sviluppare un framework computazionalmente veloce e dalla necessità di interfacciarsi con librerie implementate in C++ per la gestione di point-cloud e mappe.

Le librerie utilizzate vengono mostrate nell'elenco qui di seguito:

- *robotcar-sdk* per il processamento del dataset di partenza (python)
- *Point Cloud Library* [3] *(PCL)* per la gestione e manipolazione di point-cloud (C++)
- *Octomap*[4] per il passaggio alle mappe a voxel e applicazione dell'operazione di raytracing (C++)

PCL è una libreria open-source scritta in C++ che permette la gestione e manipolazione di point-cloud 2D e 3D [37]. Al suo interno contiene algoritmi che permettono, ad esempio, il filtraggio, la segmentazione, la classificazione e la ricostruzione di superfici a partire da una point-cloud.

Il funzionamento della libreria Octomap è descritto all'interno della sezione 2.3.2 facente parte dello stato dell'arte.

Come verrà mostrato nelle sotto-sezioni successive, il funzionamento generale del processo si sviluppa nei seguenti passi:

1. creazione della point-cloud rappresentante la mappa partendo dal dataset di Robotcar
2. passaggio da point-cloud a mappa voxelizzata
3. aggiunta di rumore alla pose camera e applicazione della tecnica di raytracing per l'ottenimento di immagini LiDAR
4. applicazione di tecniche di data augmentation
5. salvataggio delle immagini ottenute e delle informazioni di creazione

In figura 4.5 viene mostrato un esempio schematico del processo di creazione appena descritto.

Dai difetti emersi dalle tecniche presenti in letteratura (sezione 2.5) per l'ottenimento di immagini LiDAR dense, è necessario sottolineare che si è utilizzato un approccio differente rispetto a quelli mostrati. Per ricostruire la geometria solida della scena si è deciso di sfruttare la rappresentazione a voxel. Le mappe a voxel possiedono il grande

[3] http://www.pointclouds.org, data consultazione: *febbraio 2019*
[4] https://github.com/OctoMap/octomap, data consultazione: *febbraio 2019*

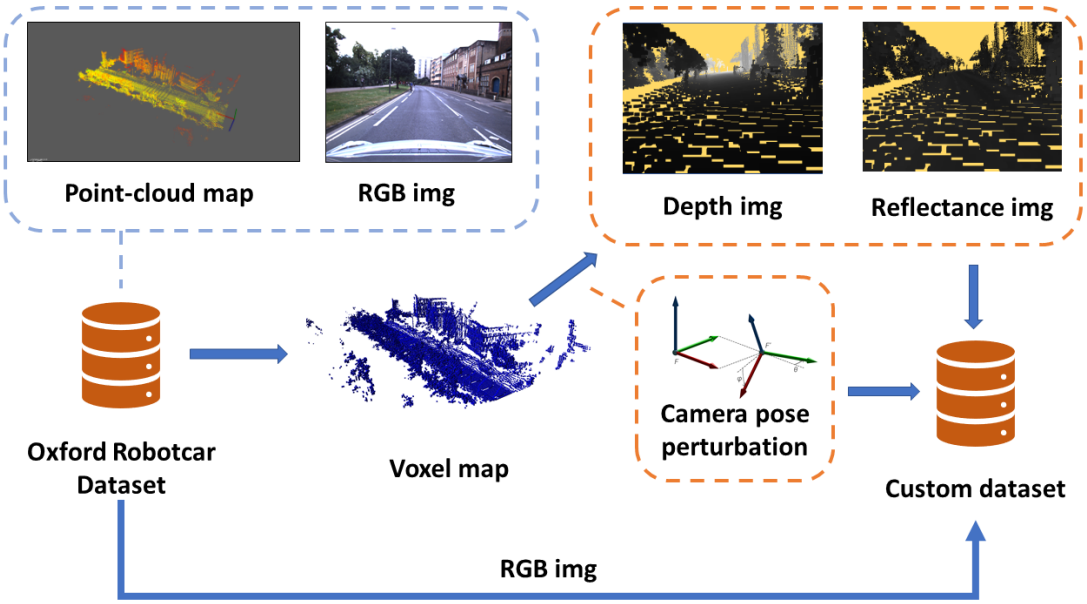


Figura 4.5: In figura viene mostrato il processo di creazione. Partendo dal dataset di Robotcar si costruisce una point-cloud rappresentante la mappa LiDAR e si reperisce l'immagine RGB associata al sistema di riferimento della mappa. Successivamente si trasforma la point-cloud generata in una mappa a voxel e andando a perturbare la pose della camera si ottengono le immagini LiDAR mediante raytracing. A tale processo si aggiunge l'applicazione di tecniche di data augmentation ed infine le immagini LiDAR, RGB e le informazioni della pose perturbata vengono salvate all'interno di un nuovo dataset.

vantaggio, rispetto alle mesh, di fornire una buona ricostruzione della scena senza che vi sia la necessità di partire da point-cloud dense. L'utilizzo di questa rappresentazione permette di ottenere, mediante l'operazione di raytracing, immagini LiDAR ad alta densità, senza che vengano applicate strategie di upsampling. Inoltre, grazie alla ricostruzione solida della scena, si riesce a mitigare il problema della proiezione di elementi nascosti dal punto di vista dell'osservatore.

4.2.1 Creazione della mappa

All'interno del dataset di robotcar ad ogni tipologia di dato è associato un *timestamp*, che definisce il suo tempo di acquisizione. In particolare tale informazione temporale è sincronizzata rispetto all'intero sistema di sensori del veicolo. Di conseguenza risulta semplice risalire, ad esempio, alle scansioni laser che intercorrono tra due frame camera acquisiti. Inoltre, ad ogni frame camera viene associata una pose, che viene calcolata mediante la tecnica di visual odometry, permettendo l'associazione tra pose del veicolo e acquisizione camera. Le scansioni consecutive vengono accumulate osservando i loro tempi di acquisizione e, basandosi sulle pose del veicolo fornite ai diversi timestamp, viene effettuata un'interpolazione delle pose dei loro punti 3D. Tale processo viene svolto da un script fornito dalla libreria di Robotcar.

Un difetto riscontrato all'interno del robotcar-sdk riguarda i lunghi tempi computazionali necessari per la creazione di una mappa che si sviluppa per decine di metri. Il problema risiede nell'integrazioni dei punti di scena all'interno di un array secondo un processo iterativo. La conseguenza è quella di rendere necessaria la copiatura della struttura dati, in una di dimensioni più grande, ogni volta che si include una nuova scansione. Con questo tipo di processo, creare una point-cloud che si sviluppa per cento metri e che possiede una grande quantità di scansioni, richiede un tempo computazionale di circa 3-5 minuti. Inoltre data la frequenza di campionamento elevata del sensore LiDAR ($12.5Hz$), in fase di frenata o partenza del veicolo, vengono registrate un gran numero di scansioni che però descrivono solo un breve tratto del percorso. La conseguenza è quella di creare point-cloud che sono molto dense in alcuni punti della mappa, comportando un grande utilizzo di memoria e lunghi tempi di creazione.

Queste problematiche vengono affrontate andando a modificare la libreria originale di robotcar, aggiungendo una funzionalità che permette di effettuare un sotto-

campionamento delle scansioni del dataset. Questo processo viene svolto con l'ausilio dalle pose registrate grazie alla visual odometry e si sviluppa nella maniera seguente:

1. Inizialmente si definisce un passo di sotto-campionamento e l'estensione massima della mappa in metri, corrispondente alla lunghezza percorsa dal veicolo in un tratto della strada (e.g. 100 m)

2. Viene calcolata la distanza percorsa tra la pose fornita dalla visual odometry al tempo t e quella a tempo $t + 1$

3. Se la distanza calcolata è minore di una certa soglia (e.g. 0.2 m) si procede al sotto-campionamento di tutte le scansioni, secondo il passo definito in (1), il cui tempo di acquisizione $t_{scan} \in [t, t + 1]$.

Notare che all'intervallo $[t, t + 1]$ possono appartenere molteplici scansioni LiDAR a causa della frequenza di campionamento del sensore laser che risulta maggiore rispetto a quella del sensore camera. Il processo appena descritto permette di passare da un tempo di computazione di circa 3-5 minuti, per la creazione di una mappa di 100 metri, ad un tempo di circa 2-10 secondi.

La mappa così ottenuta viene salvata in un file tenendo traccia delle coordinate cartesiane dei suoi punti e del dato di riflettanza ad essi associato. Notare che le coordinate salvate vengono espresse rispetto al sistema di riferimento camera. In figura 4.6 viene fornito un esempio di mappa generata dal processo appena definito.

4.2.2 Passaggio alla rappresentazione a voxel

Una volta ottenuta la point-cloud rappresentante la mappa, viene applicato il processo di voxelizzazione. Octomap permette di effettuare questo tipo di operazione, tuttavia la classe base dell'octree, utilizzata per rappresentare la mappa a voxel, non consente di tenere traccia di informazioni aggiuntive oltre allo stato di occupazione di una cella. Di conseguenza si è resa necessaria una sua re-implementazione con lo scopo di associare alla struttura dati ad albero anche il dato di riflettanza. La classe dell'octree implementato prende il nome di *ColorReflOctree* e permette, oltre che di tenere traccia dell'informazione di riflettanza mediante una variabile *float*, associare una terna colore RGB per visualizzare in maniera grafica tale informazione. Utilizzare solamente una terna colore per identificare il dato di riflettanza comporterebbe ad una perdita di informazione,

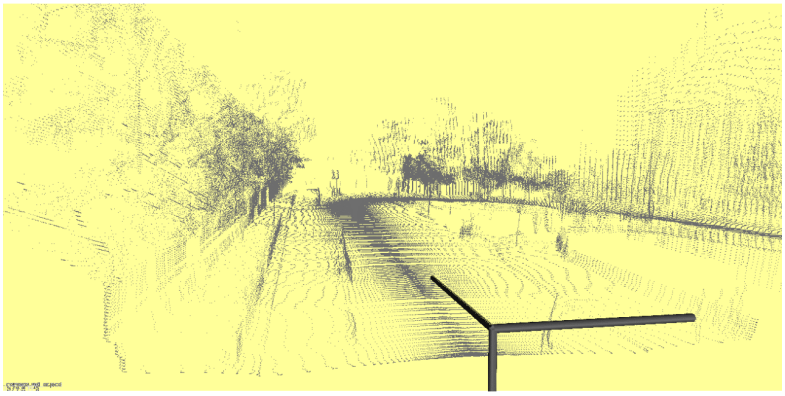


Figura 4.6: In figura viene mostrata la point-cloud generata grazie al processo di sotto-campionamento.

poiché il tipo di variabile associato ad ogni canale colore è di tipo *uint8* ed inoltre, non si fornirebbe una rappresentazione diretta del valore ottenuto dal sensore LiDAR.

Una problematica che ne consegue riguarda la creazione una procedura che permetta, a partire dalla point-cloud, di assegnare un valore di riflettanza ad uno specifico voxel, poiché la libreria Octomap non possiede alcuna funzionalità che permetta questo tipo di processo. La soluzione implementata prevede di effettuare un'iterazione sui i nodi dell'albero con lo scopo di ricercare le celle di dimensione minima occupate. Successivamente, da queste vengono ricavate le coordinate tridimensionali del centro del voxel che rappresentano. Per ogni centro così ottenuto, si effettua una ricerca all'interno della point-cloud di partenza con lo scopo di ricavare i punti ricadenti in un intorno. I punti vengono ricercati utilizzando una particolare struttura dati chiamata *kd-tree*. Tali strutture dati permettono di effettuare il partizionamento di uno spazio tridimensionale e sono molto utilizzate per la manipolazione di point-cloud, poiché permettono di applicare ricerche di punti in maniera efficiente utilizzando, ad esempio, *nearest neighbor search*. Nel processo sviluppato si effettua la ricerca dei cinque punti più vicini al centro del voxel da cui si ricava la media dei valori di riflettanza. In particolare PCL permette di fissare un raggio massimo entro il quale ricercare i punti vicini, in maniera tale da restringere la ricerca all'interno del volume rappresentato dal voxel. La media delle riflettanze ottenute costituisce il valore rappresentante.

Il numero di vicini che si osserva per calcolare la media è fissato in base alle speri-

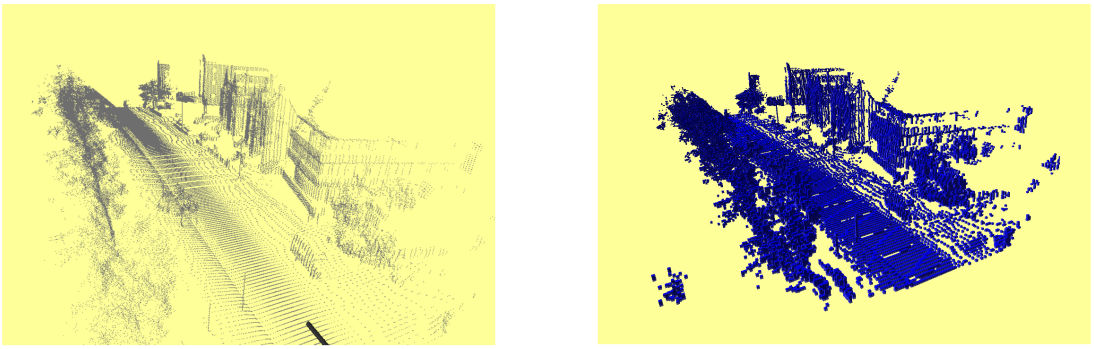


Figura 4.7: A sinistra viene mostrata la point-cloud ottenuta da Robotcar. A destra viene fornito un esempio rappresentazione mediante voxel della stessa mappa.

mentazioni effettuate. Notare che aumentare tale numero porti ad ottenere dei valori di riflettanza troppo mediati, impedendo di discriminare dal resto della scena elementi che dovrebbero essere molto riflettenti (e.g. cartelli stradali catarifrangenti). Al contrario, fissare un valore troppo basso porta all'ottenimento di valori estremamente variabili per uno stesso oggetto voxelizzato della scena. Un esempio di passaggio da point-cloud a mappa a voxel è mostrato in figura 4.7.

Una volta ottenuta la mappa a voxel si è proceduto a perturbare la pose camera e ad applicare la tecnica di raytracing.

4.2.3 Perturbazione della pose camera e raytracing

Come descritto nelle motivazioni del lavoro svolto (capitolo 3), lo scopo della tesi è quello di permettere ad una rete neurale convoluzionale di predire l'errore di localizzazione di un veicolo. Come affermato in precedenza, conoscendo i parametri estrinseci di una camera, il problema viene riportato alla corretta predizione dell'errore rispetto alla pose di quest'ultima. Di conseguenza durante la generazione delle immagini del dataset è necessario creare degli esempi non corretti, affinché la rete neurale apprenda dal dato immagine LiDAR lo scostamento della pose rispetto alla scena rappresentata nell'immagine camera. Questo processo viene svolto andando ad aggiungere del rumore ai parametri di pose della camera, in maniera tale da osservare una porzione della mappa che non combacia perfettamente con quella realmente visibile e rappresentata dall'immagine RGB. In particolare la posizione della camera viene espressa mediante le traslazioni sulle sue componenti (x, y, z), mentre il suo orientamento è rappresentato secondo la convenzione

degli angoli di Eulero $(Roll, Pitch, Yaw)$.

L'operazione di applicazione del rumore sui parametri di pose viene effettuata nella seguente modalità:

1. Si definisce un range di errore da applicare ai parametri di traslazione (e.g. $[-1.5m, +1,5m]$) e un range di errore da applicare alle rotazioni (e.g. $[-20degs, +20degs]$)

2. Dati gli intervalli definiti in (1), per ogni parametro della pose viene estratto un valore secondo una distribuzione di probabilità uniforme

3. La pose perturbata viene utilizzata per rototraslare la mappa a voxel secondo la convenzione di Tait-Bryan

Notare che fissare degli intervalli di errore troppo ampi porterebbe alla proiezione di parti della mappa che non hanno alcuna corrispondenza con il dato immagine RGB, che invece viene acquisito dal sensore camera. In questa casistica sarebbe impossibile per una rete neurale convoluzionale riuscire a trovare un numero di corrispondenze sufficienti tra mappa osservata e immagine RGB, con la conseguenza di non riuscire a predire correttamente l'errore di localizzazione iniziale.

Per rappresentare la parte di mappa osservata mediante un'immagine, viene applicata l'operazione di raytracing ed Octomap contiene al suo interno i metodi che rendono possibile utilizzare questa tecnica di proiezione. La definizione della direzioni dei raggi emessi tiene in considerazione i parametri intrinseci camera definiti all'interno del Robotcar dataset e dalle specifiche del datasheet del sensore[5].

In figura 4.8 viene mostrata l'immagine con sovrapposto il canale di riflettanza dell'immagine LiDAR ottenuta mediante raytracing. In questo caso la pose camera utilizzata per l'emissione dei raggi non è soggetta ad un rumore iniziale.

L'operazione di raytracing è una tecnica che si basa sul tracciamento di raggi, le cui direzioni sono identificate dalle rette passanti dal centro di proiezione di una camera e dai diversi centri dei pixel appartenenti al piano immagine. Una volta definiti i raggi di emissione, per ognuno di essi si ricavano i punti di intersezione più vicini rispetto agli oggetti presenti nella scena. Conoscendo il punto d'intersezione con un oggetto è possibile ricavare informazioni come, ad esempio, il dato colore. Per quanto riguarda il lavoro svolto, vengono ricavate le intersezioni dei raggi rispetto ai voxel della mappa.

[5]https://eu.ptgrey.com/support/downloads/10132, data consultazione: *febbraio 2019*

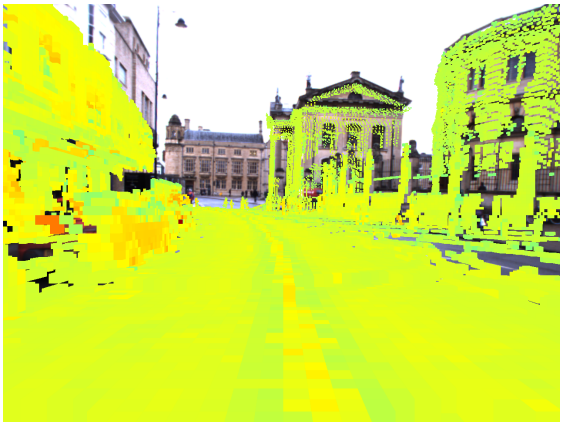


Figura 4.8: Nell'immagine viene mostrata la sovrapposizione tra immagine di riflettanza e immagine RGB. Per quanto riguarda la riflettanza i colori freddi rappresentano basse intensità e i colori caldi valori di riflettanza più alti. Notare che tale rappresentazione viene utilizzata puramente per scopi di visualizzazione.

Una volta intersecato il voxel occupato più vicino al punto di vista camera, si procede all'acquisizione dei dati di profondità e riflettanza ad esso associati. Una problematica, che sorge durante questa fase del processo, riguarda la comprensione di come rappresentare a livello immagine le due tipologie di informazioni. Generalmente, un pixel di un'immagine a livelli di grigio viene rappresentato mediante un dato di tipo *uint8* (unsigned int a 8 bit), ma utilizzare 256 valori per rappresentare la profondità e riflettanza di una scena porterebbe ad una grande perdita di informazione. Inoltre vi è la necessità di riscalare correttamente i valori di riflettanza all'interno dell'intervallo di valori dell'immagine. Tuttavia conoscere il range dinamico del sensore LiDAR non è possibile, poiché esso non è specificato all'interno del datasheet fornito dal produttore. Notare che questa problematica non si presenta per quanto riguarda i valori di profondità, poiché nel processo di creazione della mappa viene fissata la sua estensione massima (e.g. $100m$), producendo in questo modo un intervallo di valori di profondità ben definito.

Per risolvere le problematiche appena descritte vengono proposte le seguenti soluzioni:

- utilizzare delle immagini di tipo *uint16* (unsigned int a 16 bit) per aumentare il range dinamico a disposizione per rappresentare le immagini di riflettanza e profondità

- creare la mappa rappresentante un'intera run di robotcar ed estrarre il valore massimo di riflettanza. Successivamente assegnare il valore estratto come massimo del range dinamico di riflettanza della run considerata

Notare che per riportare i range dinamici di profondità e riflettanza rispetto all'intervallo di valori delle immagini di tipo uint16, viene applicata una semplice proporzione con l'arrotondamento di eventuali valori decimali.

Le immagini LiDAR, ottenute mediante questo processo, hanno una risoluzione 960x1280 pixel pari alle immagini RGB del sensore camera.

4.2.4 Data augmentation

Il tempo computazionale necessario per creare la point-cloud di una mappa e per il passaggio alla rappresentazione a voxel richiede circa $13 secondi$, tuttavia il collo di bottiglia del processo di creazione del dataset risiede nell'applicazione della tecnica di raytracing. Il tempo di generazione di un'immagine LiDAR varia a seconda del grado di risoluzione della mappa voxelizzata, poiché al diminuire della grandezza del singolo voxel aumentano il numero di intersezioni raggio-mappa che bisogna computare. Ad esempio, l'utilizzo di una grandezza del voxel di $30cm$ porta ad un tempo medio di computazione di circa $17 secondi$ per immagine, invece dimezzandone la sua dimensione a $15cm$ si percepisce un aumento del tempo computazionale a circa $30 secondi$ per immagine. Di conseguenza se l'obiettivo è quello, ad esempio, di generare un dataset di 10000 campioni, il processo per come è descritto risulta molto dispendioso in termini di tempo. All'interno del lavoro di tesi non ci si è concentrati sull'ottimizzazione dell'operazione di raytracing, anche a causa dei limiti dell'hardware a disposizione. Tuttavia questa tecnica è largamente utilizzata all'interno del mondo della computer grafica, dove le tempistiche di renderizzazione di una scena avvengono a *real-time*. Di conseguenza, con le opportune ottimizzazioni, tale processo potrebbe essere reso computazionalmente più veloce.

Di seguito viene fornita una stima del tempo di calcolo necessario per generare 10000 campioni del dataset:

$$\begin{aligned} \hat{t}_{tot} &= (\hat{t}_{pcl} + \hat{t}_{vox} + \hat{t}_{ray}) \cdot n = \\ &= (3\ sec + 10\ sec + 17\ sec) \cdot 10000 = \\ &= 300000\ sec \approx 83\ h \end{aligned} \tag{4.2.4.1}$$

dove $\hat{t}_{pcl}$, $\hat{t}_{vox}$ e $\hat{t}_{ray}$ rappresentano la media stimata del tempo computazionale necessario per la creazione della point-cloud, per il passaggio alla voxel map e per l'applicazione della tecnica di raytracing. Infine n identifica il numero di campioni del dataset che si vuole creare. Notare che i tempi di calcolo necessari per l'assegnazione dei valori di riflettanza ai voxel e alla rototraslazione della mappa secondo la perturbazione della pose camera non sono stati riportati, essendo questi trascurabili.

Come appena descritto, la generazione di immagini a partire da frame camera sempre differenti porta ad un tempo computazionale elevato nella fase di creazione del dataset. Non potendo applicare la parallelizzazione del processo di raytracing, a causa dei limiti dell'hardware a disposizione, si è proceduto ad applicare tecniche di *data augmentation*. Lo scopo è quello di migliorare i tempi di computazione necessari per la creazione del dataset, andando a diminuire il numero di iterazioni per la creazione di mappe, e riuscire, allo stesso tempo, ottenere un gran numero di esempi da mostrare alla rete neurale.

Le tecnica utilizzata prende ispirazione dal lavoro di Schneider et al. [39] e anziché considerare pose camera differenti ad ogni iterazione del processo, si ripete più volte per uno stesso frame camera la perturbazione della pose dell'osservatore. Ad ogni pose perturbata successivamente è applicata l'operazione di raytracing. Notare che il rumore della pose camera viene aggiunto ogni volta a partire dalla pose corretta.

Rispetto a quanto implementato, per ogni frame e pose camera considerati, si effettuano 10 iterazioni del processo appena descritto. Di conseguenza, rispetto alle stime precedenti, il tempo computazionale ottenuto con 10000 campioni è il seguente:

$$\begin{aligned} \hat{t}_{tot} &= (\hat{t}_{pcl} + \hat{t}_{vox}) \cdot n/10 + \hat{t}_{ray} \cdot n = \\ &= (3\ sec + 10\ sec) \cdot 1000 + 17\ sec \cdot 10000 = \\ &= 190000\ sec \approx 51\ h \end{aligned} \tag{4.2.4.2}$$

Notare che con l'utilizzo di questa tecnica si è ridotto sensibilmente il tempo di computazione. Inoltre eseguendo in parallelo diverse istanze del programma di creazione del dataset e processando diverse locazioni geografiche delle mappe di Robotcar, è possibile diminuire ulteriormente il tempo computazionale.

L'approccio appena descritto possiede il difetto di creare uno sbilanciamento tra immagini LiDAR e immagini RGB, cioè per ogni 10 immagini LiDAR viene associata una stessa immagine camera. Per risolvere questa problematica vengono applicate le operazioni di *mirroring* e di rotazione *in-place* sull'immagine RGB.

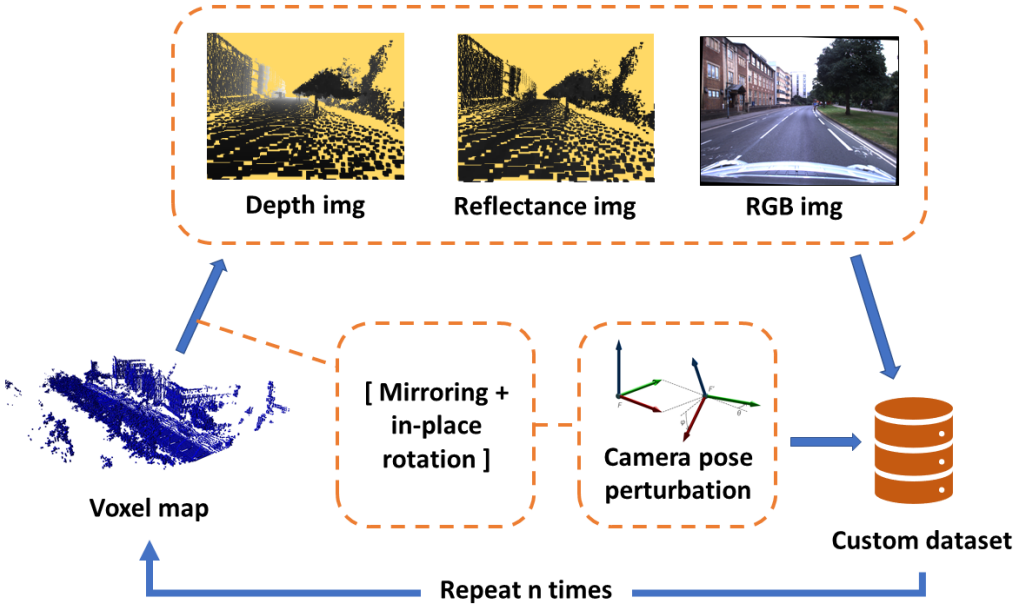


Figura 4.9: Esempio schematico dell'operazione di *data augmentation*. Partendo da una mappa a voxel ed una immagine RGB si applicano un certo numero di volte le operazioni di mirroring, rotazione in-place e perturbazione della pose camera secondo diverse combinazioni.

Per mirroring si intende l'operazione che effettua una riflessione di un'immagine rispetto ad un asse (e.g. orizzontale), mentre per rotazione in-place si intende l'operazione di rotazione planare dell'immagine rispetto al centro. Queste operazioni vengono estese alla rappresentazione 3D della mappa in maniera tale da renderla consistente con la scena osservata dalla camera.

L'obiettivo è quello di fornire, sia a livello immagine sia a livello di mappa 3D, un insieme di esempi più variabile ed in particolare, con la rotazione in-place dell'immagine, si vuole simulare possibili inclinazioni del veicolo dovute, ad esempio, a diversi livelli di compressione delle sospensioni. Le rotazioni utilizzate per applicare la rotazione in-place vengono estratte casualmente nell'intervallo $[-3degs; +3degs]$ secondo una distribuzione di probabilità uniforme. Andare a considerare delle rotazioni maggiori porterebbe all'ottenimento di immagini che rappresentano scenari inverosimili. Dato che l'obiettivo del lavoro è quello di affrontare il problema della localizzazione, intervalli maggiori non vengono considerati.

Per ogni iterazione del processo vengono applicate le operazioni appena descritte secondo diverse combinazioni. Di seguito viene fornito l'elenco delle terne di immagini ottenute (profondità, riflettanza, RGB) su un totale di 10 iterazioni per ogni frame camera considerato:

- 1 terna applicando: perturbazione pose camera
- 4 terne applicando: rotazione in-place, perturbazione pose camera
- 1 terna applicando: mirroring, perturbazione pose camera
- 4 terne applicando: mirroring, rotazione in-place, perturbazione pose camera

notare che l'ordine delle operazioni deve essere applicato così come descritto per non interferire con le rototraslazioni della perturbazione della pose camera.

In figura 4.10 vengono mostrati degli esempi ottenuti mediante il processo di creazione del dataset. Notare che dell'immagine LiDAR vengono mostrati separatamente i canali di profondità e riflettanza.

Il colore di sfondo, utilizzato nelle immagini LiDAR mostrate, ha lo scopo di migliorarne la visualizzazione. Nelle immagini LiDAR originali si utilizza il colore nero per rappresentare lo sfondo dei diversi canali.

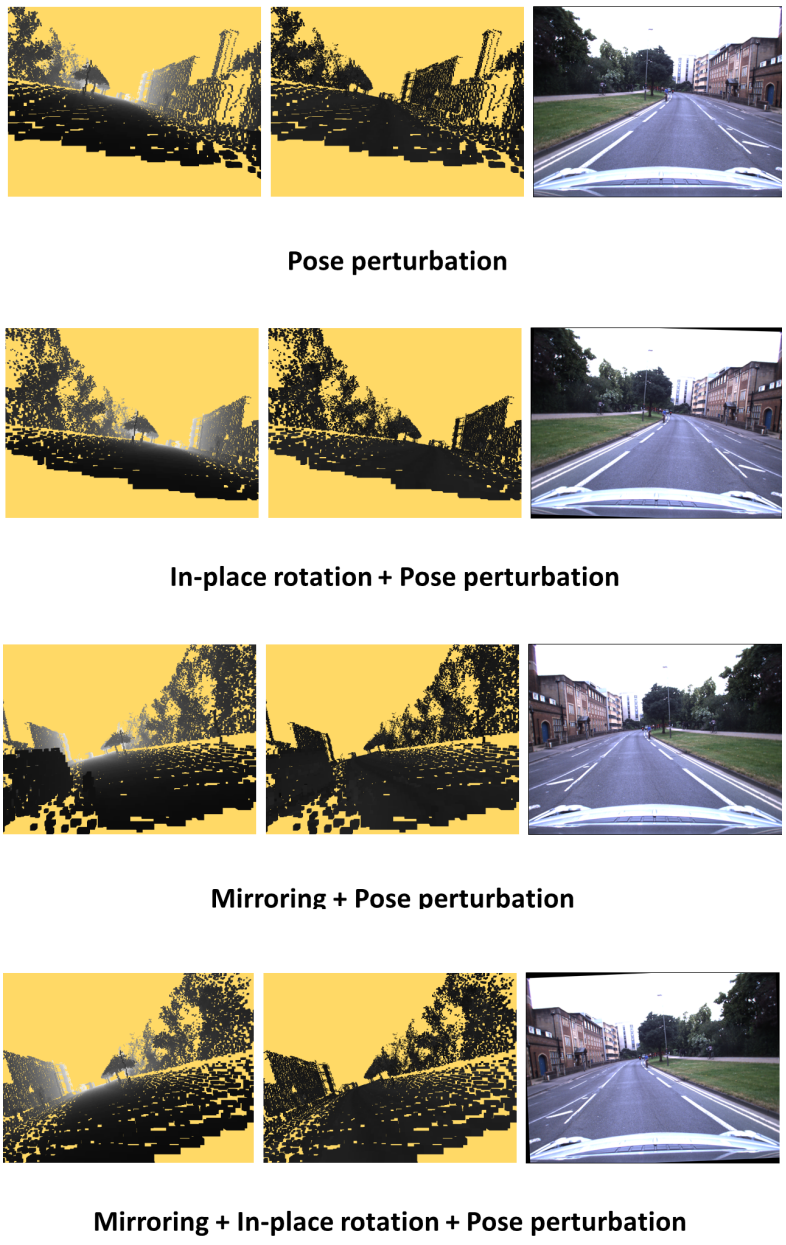


Figura 4.10: Immagini ottenute dal processo di data augmentation. Da sinistra vengono rappresentate: l'immagine di profondità, l'immagine di riflettanza e l'immagine RGB della camera.

Al termine di ogni iterazione, che porta all'ottenimento di un'immagine LiDAR e RGB, si tiene traccia delle informazioni di creazione mediante un file csv. In particolare si tiene traccia dei seguenti dati:

- percorsi delle immagini del dataset generate
- rotazioni in-place (in radianti)
- componenti di traslazione camera (in radianti)
- componenti di rotazione camera espresse secondo angoli di Eulero (in radianti)
- componenti di rotazione camera espresse mediante la rappresentazione dei quaternioni

Le rotazioni espresse mediante quaternioni sono molto utilizzate nel campo della fisica e computer grafica e, rispetto agli angoli di Eulero, possiedono il vantaggio di non soffrire del problema del blocco cardanico. L'insieme di rotazioni nel 3D descritto da un quaternione q è rappresentato nella seguente maniera:

$$q = a + bi + cj + dk \tag{4.2.4.3}$$

con a, b, c, d numeri reali ed i, j, k simboli simili all'unità immaginaria dei numeri complessi.

I quaternioni che descrivono pure rotazioni fanno parte del *gruppo di Lie* descritto dall'insieme S^3 così definito:

$$S^3 = \{(a, b, c, d) \in R^4 | a^2 + b^2 + c^2 + d^2 = 1\} \tag{4.2.4.4}$$

Per maggiori dettagli riguardo tale gruppo e le proprietà dei quaternioni, si rimanda a Zhang [48].

Dati i parametri di rotazione nel 3D di Eulero $(Roll, Pitch, Yaw)$, di seguito viene mostrato il passaggio alla rappresentazione a quaternione:

$$\begin{aligned} a =& cos(Yaw/2) \cdot cos(Pitch/2) \cdot cos(Roll/2) + \\ &+ sin(Yaw/2) \cdot sin(Pitch/2) \cdot sin(Roll/2) \\ b =& cos(Yaw/2) \cdot cos(Pitch/2) \cdot cos(Roll/2) + \\ &- sin(Yaw/2) \cdot sin(Pitch/2) \cdot sin(Roll/2) \\ c =& sin(Yaw/2) \cdot cos(Pitch/2) \cdot sin(Roll/2) + \\ &+ cos(Yaw/2) \cdot sin(Pitch/2) \cdot cos(Roll/2) \\ d =& sin(Yaw/2) \cdot cos(Pitch/2) \cdot sin(Roll/2) + \\ &- cos(Yaw/2) \cdot sin(Pitch/2) \cdot cos(Roll/2) \end{aligned} \tag{4.2.4.5}$$

Lo scopo di tenere anche traccia di un altro tipo di rappresentazione è quello di osservare il comportamento della CNN nel predire le rotazioni e capire se il tipo di rappresentazione può influire sui suoi risultati.

4.3 Dataset ottenuti

Ogni dataset creato durante il lavoro di tesi svolto corrisponde ad una specifica run di Robotcar. In particolare le giornate di acquisizione del dataset di Oxford sono quelle del 19 maggio 2014 delle 13:20 e del 26 giugno 2014 delle 9:31.

La run del 19 maggio 2014 contiene al suo interno strade alternate e presenta un tempo atmosferico che varia dal nuvoloso al soleggiato. Invece la run del 26 maggio 2014 presenta un tempo atmosferico che varia dal nuvoloso al parzialmente nuvoloso.

Partendo dalle run di robotcar appena descritte, vengono creati tre diversi dataset. Il primo fa riferimento alla run del 19 maggio 2014 e contiene un totale di 10000 campioni di immagini LiDAR e RGB, utilizzando per la ricostruzione della mappa una dimensione del lato del voxel pari a $30cm$. I restanti vengono creati a partire dal dataset del 26 giugno 2014 e contengono rispettivamente 18000 e 10000 campioni di immagini LiDAR ed RGB. Di questi due la dimensione del lato del voxel è di $30cm$ per il primo e di

15cm per il secondo. Per ognuno dei dataset così creati, viene preso in considerazione un frame camera ogni dieci del dataset di Robotcar. Questo approccio permette di mitigare la presenza di immagini camera che ritraggono la medesima scena quando, ad esempio, il veicolo non è in movimento. Di seguito vengono mostrati degli esempi di immagini ottenute per ognuno dei dataset creati.

Figura 4.11: In figura vengono mostrati degli esempi di immagini ottenute dal dataset acquisito il 19 maggio. In particolare, da sinistra vengono mostrate le immagini di: profondità, riflettanza e RGB. L'errore aggiunto sulla pose iniziale è compreso nell'intervallo $[-1.5m, 1.5m]$ per le traslazioni e $[-20degs, +20degs]$ per le rotazioni. Il lato del voxel delle mappe generate ha dimensione che si aggira intorno ai $30cm$.

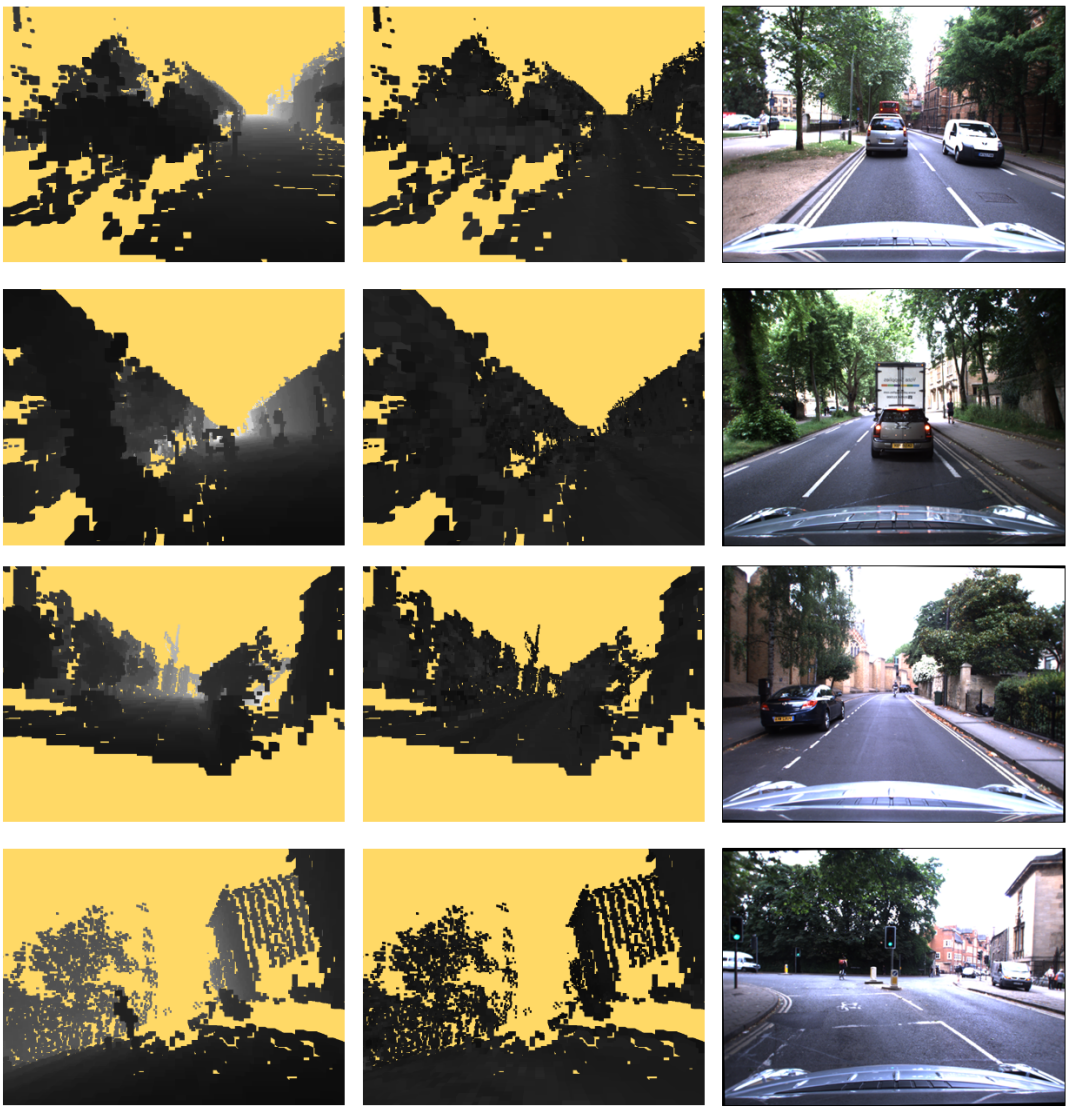


Figura 4.12: In figura vengono mostrati degli esempi delle immagini ottenute a partire dal dataset del 26 giugno. I parametri di creazione sono gli stessi riportati nella descrizione della figura 4.11. Allo stesso modo viene riportato l'ordine delle figure.

Figura 4.13: In figura vengono mostrati degli di immagini generate partendo dal dataset del 26 giugno. L'errore iniziale sulla pose camera è compreso nell'intervallo $[-1.5m, +1.5m]$ per le traslazioni e $[-15degs, +15degs]$ sulle rotazioni.

Capitolo 5

CNN utilizzata ed esperimenti svolti

In questa sezione verrà mostrata la rete neurale convoluzionale implementata e le modalità con cui sono state svolte le fasi di training e testing.

Inizialmente verranno mostrate le modifiche apportate all'architettura di RegNet, frutto del lavoro di Schneider et al. [39], motivando il suo utilizzo rispetto ad altre CNNs. Verrà introdotta la libreria di *PyTorch* utilizzata per implementare e testare la rete neurale. Successivamente si fornirà l'insieme di test effettuati ed infine verrà svolta un'analisi delle performance ottenute.

5.1 RegNet e modifiche applicate

Partendo dai lavori presenti in letteratura, si è ricercata una CNN pensata per il processamento sia di immagini RGB sia di immagini LiDAR. In particolare, si è posta molta attenzione su reti neurali concepite per svolgere il compito di regressione di una pose. La scelta finale è ricaduta su una rete neurale convoluzionale di nome RegNet, poiché congrua rispetto le caratteristiche appena descritte.

Notare che l'implementazione di RegNet, mostrata all'interno del lavoro di Schneider, ha uno scopo differente rispetto a quello su cui si basa il lavoro di tesi. In particolare nel lavoro di Schneider, la rete neurale viene utilizzata per affrontare il problema della calibrazione dei parametri estrinseci di un sensore camera, mentre lo scopo del presente lavoro riguarda il problema della localizzazione andando a stimare l'errore di una

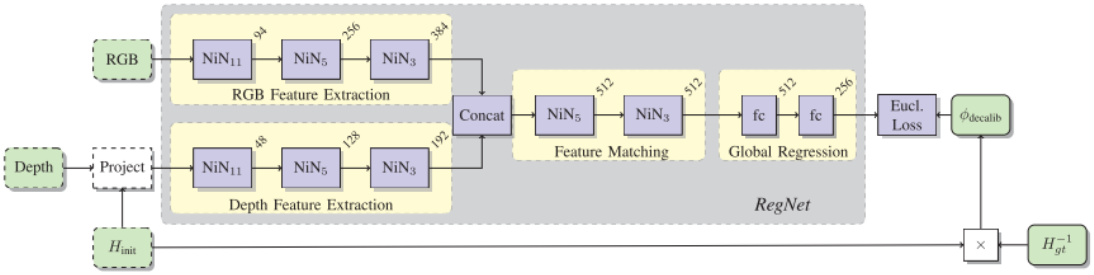


Figura 5.1: Architettura di RegNet presentata nel lavoro di Schneider et al. [39].

pose camera in input. Inoltre nel lavoro di Schneider, le immagini LiDAR utilizzate rappresentano solamente il dato di profondità ottenuto mediante le proiezioni di una singola scansione LiDAR, mentre nel lavoro di tesi viene anche trattata l'informazione di riflettanza e viene utilizzata una mappa, ottenuta dall'allineamento di molteplici scansioni. L'utilizzo di mappe rispetto a singole scansioni porta ad affrontare un problema di difficoltà maggiore. L'immagine LiDAR risultante dalla proiezione di una scansione, acquisita allo stesso istante di un frame camera, rappresenta esclusivamente gli oggetti che sono realmente presenti all'interno della scena. L'utilizzo di una mappa, contrariamente, porta a rappresentare all'interno delle immagini LiDAR elementi dinamici della scena che non sono presenti nell'immagine RGB del sensore camera. Questo fenomeno si verifica a causa dei differenti tempi di registrazione delle scansioni laser che vengono successivamente allineate per formare la mappa. In figura 5.1 viene riproposta l'architettura originale di RegNet, a cui vengono apportate le seguenti modifiche:

- vengono rimossi i layer di max-pooling iniziali, utilizzati nel lavoro di Schneider per aumentare la densità delle immagini LiDAR
- viene adattata l'architettura per processare le immagini dei dataset creati, poiché il formato di queste ultime risulta differente rispetto a quello adottato nel lavoro di Schneider

L'architettura di RegNet, come mostrato in figura 5.1, è composta da tre componenti distinte: la prima si occupa dell'estrazione delle features dalle due immagini di input, la seconda si occupa di effettuare il *matching* tra le features estratte dall'immagine RGB e le features estratte dall'immagine LiDAR, mentre l'ultima componente effettua l'operazione di regressione dell'errore sulla pose iniziale.

La prima componente dell'architettura comprende due rami separati: il primo prende in input un'immagine RGB ottenuta dalla camera posizionata sul veicolo ed il secondo prende in input un'immagine LiDAR ottenuta partendo dalla pose camera perturbata. Nello specifico, ogni ramo è composto da diversi layer convolutivi, strutturati secondo diversi blocchi *Network in Network* (NiN) [23]. Ogni blocco NiN è composto da una convoluzione k x k seguita da diverse convoluzioni *1* x *1*. Il vantaggio di questa struttura è quello di permettere la diminuzione dei tempi di apprendimento richiesti da una rete neurale convoluzionale, in quanto utilizza un numero di parametri inferiori rispetto ad una rete che utilizza solamente convoluzioni, ad esempio, 3 x 3. Inoltre, a seguito di ogni convoluzione viene applicata la funzione di attivazione *ReLu* (*Rectified Linear Unit*) e al termine di ogni blocco NiN l'output viene sotto-campionato mediante l'operazione di *max pooling*.

Una volta estratte le features dalle due diversi sorgenti immagine, esse vengono concatenate per fornirne una rappresentazione congiunta. La seconda componente dell'architettura estrae ulteriori features avendo tale rappresentazione come input. La componente della rete appena descritta prende spunto dal lavoro di Dosovitskiy et al. [8] e viene ristrutturata applicando diverse successioni di blocchi *NiN*. Inoltre, viene applicata la funzione di attivazione *ReLu* e l'operazione di *max pooling* nelle medesime modalità della prima componente.

L'ultima componente effettua l'operazione di regressione di una pose partendo dalle feature estratte in precedenza ed è composta da 2 layer *fully-connected*, che nella parte finale si ramificano per effettuare separatamente la predizione dell'errore di traslazione e rotazione. Rispetto al lavoro svolto, la pose predetta dalla rete rappresenta la stima dell'errore di localizzazione iniziale.

Per quanto riguarda la descrizione delle funzioni di loss testate verrà dedicata la sezione successiva, poiché mettere a confronto gli errori di predizione della rete neurale sulle componenti di traslazione e rotazione non è un processo immediato e richiede che vengano effettuate alcune considerazioni.

5.2 Funzioni di loss utilizzate

Parte della fase di implementazione della rete neurale si è concentrata sulla scelta di una funzione di loss che sia applicabile alla pose fornita in output.

Come spiegato nella sezione precedente, la regressione dell'errore viene effettuata per mezzo di due rami separati di layer *fully-connected* posti al termine dell'architettura, i quali restituiscono rispettivamente la stima dell'errore di traslazione e rotazione. Inizialmente l'errore della predizione viene calcolato separatamente sui due output e successivamente i risultati vengono aggregati. Notare che al variare del tipo di rappresentazione degli angoli scelta, varia anche l'output del layer *fully-connected* che effettua la predizione sugli errori di rotazione. In particolare, se vengono utilizzati gli angoli di eulero, la rete neurale restituisce le tre componenti RPY (roll, pitch e yaw), mentre con l'utilizzo dei quaternioni il numero di parametri predetti è pari a quattro. Inoltre per quanto riguarda i quaternioni, la predizione viene normalizzata, poiché un quaternione esprime una pura rotazione solamente quando le sue componenti formano un vettore di lunghezza unitaria.

Come spiegato all'interno dello stato dell'arte (sezione 2.2.4), mettere in relazione gli errori di traslazione e rotazione non è un'operazione immediata. Il problema risiede nelle diverse scale di valori con cui essi sono espressi ed inoltre, traslazione e rotazione esprimono due concetti spaziali differenti (posizione e orientamento). La problematica descritta è particolarmente accentuata quando si utilizza la rappresentazione delle rotazioni mediante quaternioni, poiché per esprimere una pura rotazione è necessario che gli elementi di un quaternione q formino un vettore di lunghezza unitaria.

Rifacendosi anche ai lavori analizzati in letteratura, le funzioni di loss considerate all'interno del lavoro di tesi sono le seguenti:

1. somma delle distanze L_1 calcolate separatamente su rotazione e traslazione, utilizzando un parametro di rescaling β per l'errore di rotazione [18].

2. somma delle distanze L_1 calcolate su rotazione e traslazione, riscalando gli errori ottenuti secondo dei pesi s_x, s_q appresi dalla rete [19]

3. somma della distanza L_1 calcolata sulla traslazione e della distanza angolare D_q calcolata sulle rotazioni espresse in quaternioni con applicazione di un parametro di rescaling β per l'errore di rotazione

Data la pose obiettivo p_{target} e la pose predetta p_{pred}, di segutito vengono fornite le formule di loss rispettivamente ai casi appena descritti:

1. $\mathcal{L}(p_{target}, p_{pred}) = L_1(t_{target}, t_{pred}) + \beta \cdot L_1(r_{target}, r_{pred})$

2. $\mathcal{L}(p_{target}, p_{pred}) = L_1(t_{target}, t_{pred}) \exp(-s_x) + s_x + L_1(r_{target}, r_{pred}) \exp(-s_q) + s_q$
3. $\mathcal{L}(p_{target}, p_{pred}) = L_1(t_{target}, t_{pred}) + \beta \cdot D_q(r_{target}, r_{pred})$

dove t_{target}, t_{pred} rappresentano le traslazioni, mentre r_{target}, r_{pred} le rotazioni.

Dati due vettori p, q, di seguito viene riportata la formula di distanza L_1 applicata su di essi:

$$L_1(p, q) = \|p - q\|_1 = \sum_{i=1}^{n} |p_i - q_i| \tag{5.2.0.1}$$

dove p_i e q_i rappresentano gli elementi i-esimi dei rispettivi vettori. Di questa distanza, utilizzata come formula di loss, è stata infine applicata una sue versione, che comparata alla distanza L_2, si dimostra più robusta rispetto agli outlier e che prende il nome di *smooth L_1 loss*. Tale formula è stata concepita all'interno del lavoro di Girshick [11]. La formula della funzione di loss $smooth_{L1}$ è la seguente:

$$smooth_{L1}(x) = \begin{cases} 0.5x^2, & \text{se } |x| < 1 \\ |x| - 0.5, & \text{altrimenti} \end{cases} \tag{5.2.0.2}$$

Nella prima e terza formula mostrate, fissare in maniera ottimale un parametro di rescaling dell'errore sulle rotazioni richiede che vengano effettuate innumerevoli test con la CNN. Tale processo si dimostra oneroso in termini di tempo, date le ore che generalmente si dimostrano necessarie per effettuare il training di una rete neurale.

Per ovviare alla problematica appena descritta il lavoro di Kendall and Cipolla [18] propone una funzione di loss che permette alla CNN di apprendere i parametri di rescaling (seconda formula). Con questo approccio non viene riscalato solamente l'errore sulla rotazione, ma l'operazione di rescaling viene applicata anche all'errore di traslazione. Il valore di questi parametri, come verrà mostrato nella fase di test, devono essere definiti in fase iniziale dall'utente. Notare che i parametri appena descritti rappresentano dei pesi della rete e non un suo output.

Per quanto riguarda la loss sulla rotazione, una funzione largamente utilizzata in letteratura è la distanza L_1 applicata ai quaternioni. Tuttavia, dato che i quaternioni non sono definiti all'interno di uno spazio euclideo, applicare una formula di questo tipo non è corretto, poiché le operazioni svolte non possiedono il medesimo significato.

Questa problematica è nota all'interno dei lavori esistenti, ma viene comunque trascurata, poiché la formula di distanza applicata si dimostra efficace in fase di predizione

dell'errore sulle rotazioni. Tuttavia tale funzione di loss si dimostra affidabile solamente quando l'errore iniziale da correggere non è particolarmente elevato (e.g. errore in un range di $20degs$ sulle rotazioni).

Facendo riferimento al lavoro svolto, il problema di definire una funzione di distanza da applicare ai quaternioni è stato affrontato, andando ad individuare ed analizzare funzioni di distanza con un significato geometrico corretto. Come mostrato nella terza formula, tale distanza deve essere comunque riscalata rispetto all'errore di traslazione.

5.2.1 Formule di distanza angolare sui quaternioni

Di seguito vengono mostrate le formule di distanza considerate per calcolare l'errore di rotazione rappresentato mediante quaternioni.

La prima formula presentata prende il nome di *geodesic distance* [16] e di seguito vengono forniti i passi necessari per il calcolo di tale distanza. Inizialmente, dati due quaternioni q, r è necessario dapprima ricavare le rispettive rotazioni espresse mediante una matrice di rotazione. Dato il quaternione q tale che $q = a + bi + cj + dk$, si ha che la sua matrice rotazione viene definita nella maniera seguente:

$$R = \begin{bmatrix} a^2 + b^2 - c^2 - d^2 & 2bc - 2ad & 2bd + 2ac \\ 2bc + 2ad & a^2 - b^2 + c^2 - d^2 & 2cd - 2ab \\ 2bd - 2ac & 2cd + 2ab & a^2 - b^2 - c^2 + d^2 \end{bmatrix}$$

Dalla matrice appena descritta si ricavano le matrici di rotazione R_q, R_r associate ai quaternioni q ed r. Date le matrici così ottenute, è possibile ottenere la matrice di rotazione *differenza* R_d:

$$R_d = R_q \cdot R_r^T \tag{5.2.1.1}$$

dove R_r^T rappresenta la matrice trasposta di R_r. Una volta ottenuta R_d, la distanza viene calcolata nella maniera seguente:

$$\theta = ||\log(R_d)|| \tag{5.2.1.2}$$

dove $\log(R_d)$ fornisce la matrice antisimmetrica che incorpora l'asse di rotazione e l'angolo della rotazione. L'operatore $||.||$ fornisce la magnitudine dell'angolo di rotazione.

Lo svantaggio di questa formula, nonostante fornisca una distanza angolare precisa, risiede nel gran numero di operazioni svolte, rendendola poco efficiente in termini computazionali. Di conseguenza non si è considerata tale distanza per definire una funzione di loss da applicare alle rotazioni espresse in quaternioni.

Di seguito, vengono mostrate altre due formule di distanza applicabili ai quaternioni, che rispetto alla precedente risultano più efficienti in termini computazionali. Le differenze riscontrate mostrano variazioni di pochi millesimi di grado.

La seconda formula presentata si basa sul prodotto scalare di quaternioni unitari [16]. Dati due quaternioni q e r la loro distanza angolare θ è definita nella maniera seguente:

$$\begin{aligned} m =& q \cdot r^T \\ \theta =& arccos(|m|) \end{aligned} \tag{5.2.1.3}$$

dove l'operazione $q \cdot r^T$ rappresenta il prodotto scalare tra i due e non l'operazione di moltiplicazione tra quaternioni. Il problema di questa formula di distanza riguarda la non differenziabilità della funzione *arcoseno* in -1 e $+1$, essendo la sua derivata $\frac{d}{dx}arccos(x) = \frac{-1}{\sqrt{1-x^2}}$. Questo fenomeno si presenta, ad esempio, quando i due quaternioni rappresentano la medesima rotazione.

La terza possibile formula di distanza è quella che utilizza l'operatore *arcotangente2*. Dati due quaternioni q, r, la loro distanza angolare θ viene così calcolata:

$$\begin{aligned} m =& mul(q, inv(r)) \\ \theta =& atan2(\sqrt{b_m^2 + c_m^2 + d_m^2}, |a_m|) \end{aligned} \tag{5.2.1.4}$$

dove le operazioni inv e mul sono rispettivamente l'operazione di inversione e di moltiplicazione secondo l'algebra dei quaternioni e i termini a_m, b_m, c_m, d_m rappresentano le componenti del quaternione m. L'inversione di un quaternione è definita come segue:

$$inv(r) = a_r - b_r i - c_r j - d_r k \tag{5.2.1.5}$$

dove $r = a_r + b_r i + c_r j + d_r k$.

Gli elementi del quaternione m ottenuti mediante l'operazione di moltiplicazione *mul* sono così definiti:

$$\begin{aligned} a_m =& a_{inv(r)} \cdot a_q - b_{inv(r)} \cdot b_q - c_{inv(r)} \cdot c_q - d_{inv(r)} \cdot d_q \\ b_m =& a_{inv(r)} \cdot b_q + b_{inv(r)} \cdot a_q - c_{inv(r)} \cdot d_q + d_{inv(r)} \cdot c_q \\ c_m =& a_{inv(r)} \cdot c_q + b_{inv(r)} \cdot d_q + c_{inv(r)} \cdot a_q - d_{inv(r)} \cdot b_q \\ d_m =& a_{inv(r)} \cdot d_q - b_{inv(r)} \cdot c_q + c_{inv(r)} \cdot b_q + d_{inv(r)} \cdot a_q \end{aligned} \tag{5.2.1.6}$$

La funzione di *arcotangente2* ha il vantaggio di essere sempre differenziabile con quaternioni che esprimono pure rotazioni, cioè aventi norma pari a uno. Di conseguenza tale formula è stata infine utilizzata come funzione di distanza.

5.3 Implementazione della rete in PyTorch

Il framework utilizzato per l'implementazione e il testing della rete neurale convoluzionale prende il nome di PyTorch.

PyTorch è una libreria python open-source utilizzata nel campo dell'apprendimento automatico[1] che supporta il calcolo mediante GPU (*Graphic Processing Unit*). In particolare la libreria fornisce l'insieme di classi e metodi che permettono l'implementazione di reti neurali e il processing di dataset.

La libreria di Pytorch è costituita da cinque principali componenti:

- **torch**: che permette la creazione di *Tensori* supportando il calcolo mediante GPU
- **torch.autograd**: che fornisce i metodi per il calcolo differenziale mediante una tecnica chiamata *tape-based*
- **torch.nn**: che fornisce le classi per l'implementazione di reti neurali
- **torch.multiprocessing**: che permette di eseguire il calcolo parallelo in maniera ottimizzata
- **torch.utils**: che fornisce metodi e classi di supporto per la gestione del dataset

Per tape-based si intende una tecnica che permette di creare in maniera dinamica, a runtime, il grafo rappresentante il modello di rete neurale implementato, permetten-

[1] https://github.com/pytorch/pytorch, data consultazione: *febbraio 2019*

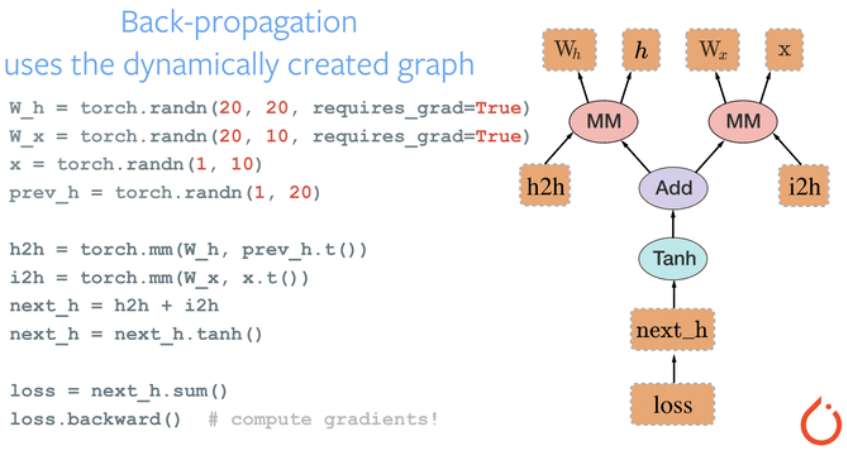


Figura 5.2: Grafo generato a runtime mediante approccio tape-based.

do di effettuare in maniera efficiente l'operazione di *backpropagation*. Un esempio di generazione del grafo viene mostrato in figura 5.2.

Altra caratteristica di PyTorch è quella di integrare al suo interno una libreria, chiamata *torchvision*, che permette il processamento di immagini.

Per quanto riguarda implementazione della rete neurale convoluzionale, sono stati sviluppati tre script python: *regnet.py*, *regnetDataset.py* e *main.py*. Lo script regnet.py descrive al suo interno l'architettura della rete implementata, mentre regnetDataset.py estende l'implemetazione della classe *Dataset* di PyTorch, in maniera tale da processare il dataset di input secondo dei criteri personalizzati. Infine lo script main.py si occupa di effettuare le operazioni di training e test della rete neurale sviluppata. In particolare, quest'ultimo permette di definire, durante la fase di avvio dello script, alcuni parametri di test come, ad esempio, funzione di loss utilizzata, tipologia di rappresentazione delle rotazioni, numero di epoche di training della rete ed il dataset che si desidera utilizzare.

5.4 Descrizione delle modalità di test

Di seguito viene fornita una descrizione dell'approccio utilizzato per effettuare il training e testing della rete neurale convoluzionale implementata.

Una volta scelto il dataset con cui osservare le performance della rete neurale, esso viene partizionato in due insiemi:

- un insieme di training costituito dall'80% dei campioni del dataset
- un insieme di test per la parte restante

Il partizionamento viene effettuato in maniera tale da garantire che l'insieme di training e l'insieme di test costituiscano due porzioni geograficamente differenti della mappa globale. Così facendo è possibile, durante la fase di test, osservare il comportamento della rete neurale di fronte a parti della mappa che non le sono state mostrate durante la fase di training.

Gli esempi facenti parte del training set vengono forniti in input alla CNN secondo un ordine casuale, in maniera tale da evitare di processare locazioni consecutive della mappa e di conseguenza prevenire il fenomeno dell'overfitting. Inoltre durante questa fase, viene effettuata un'ulteriore operazione di data augmentation, andando a variare casualmente la luminosità delle immagini RGB del training set ad ogni epoca con una probabilità del 50%. Notare che la variazione della luminosità di una scena non interferisce con il dato di riflettanza del LiDAR, data la natura attiva del sensore. Il preprocessing applicato sui dati di input, viene effettuato dalla classe python implementata all'interno di *regnetDataset.py*.

Durante la fase di training lo script python, che si occupa della gestione del dataset, fornisce in input alla rete neurale le seguenti informazioni:

- immagine LiDAR avente un canale di profondità e uno di riflettanza
- immagine RGB della camera posizionata sul veicolo

Per ogni canale delle immagini RGB in input viene effettuata l'operazione di normalizzazione secondo la media e la deviazione standard calcolata sul dataset di imageNet [7].

Per quanto riguarda le immagini LiDAR, la normalizzazione viene svolta andando a riscalere i valori dei due canali in un intervallo *float* $[0, 1]$, tenendo in considerazione gli estremi del range dinamico del tipo *uint16* associato alle immagini.

I test effettuati hanno lo scopo di comprendere il comportamento della CNN di fronte ai diversi dataset creati e alle variazioni dei suoi iper-parametri. In particolare, in tabella 5.1 e 5.2 vengono mostrati i test significativi effettuati, mettendo in risalto: la rappresentazione degli angoli utilizzata, le funzioni di loss applicate sulle traslazioni

Runs	Angles repr.	Transl. loss	Rot. loss	Rot rescaling	Transl rescaling	Depth	Refl	Robotcar run	Voxel size
1	quaternions	SmoothL1	QuatDistance	-	-	✓	✓	May 19	30 cm
2	euler	SmoothL1	QuatDistance	-	-	✓	✓	May 19	30 cm
3	euler	SmoothL1	QuatDistance	100	-	✓	✓	May 19	30 cm
4	quaternions	SmoothL1	SmoothL1	100	-	✓	✓	May 19	30 cm
5	quaternions	SmoothL1	SmoothL1	learned	learned	✓	✓	May 19	30 cm
6	euler	SmoothL1	SmoothL1	learned	learned	✓	✓	May 19	30 cm
7	quaternions	SmoothL1	QuatDistance	-	-	✓	✓	June 26	30 cm
8	euler	SmoothL1	SmoothL1	-	-	✓	✓	June 26	30 cm
9	quaternions	SmoothL1	QuatDistance	100	-	✓	✓	June 26	30 cm

Tabella 5.1: Elenco dei test effettuati sul dataset creato a partire dalla run del 19 maggio e 26 giugno di Robotcar. Gli errori iniziali sono appartenenti all'intervallo $[-1.5m; +1.5m]$ sulle componenti di traslazione e $[-20degs; +20degs]$ sulle componenti di rotazione. La grandezza del voxel delle mappe utilizzate si aggira intorno ai $30cm$ di lato. Il dataset utilizzato è composto da 10000 esempi per il dataset ottenuto a partire dalla run del 19 maggio, mentre si attesta a 18000 esempi per il dataset ottenuto a partire dalla run 26 giugno. Notare che per *QuatDistance* si intende la distanza angolare calcolata sui quaternioni con l'operatore di *atan2*.

e rotazioni, eventuali rescaling delle loss, i canali dell'immagine LiDAR in input e la dimensione del voxel della mappa di partenza.

Notare che la tabella 5.1 si riferisce ai test effettuati sul dataset ottenuto a partire dalle run del 19 maggio e 26 giungo di Robotcar adottando una risoluzione del voxel della mappa di $30cm$. Invece in tabella 5.2 si fa riferimento al dataset ottenuto dalla run del 26 giugno adottando una risoluzione del voxel di $15cm$.

Durante la fase di training la rete neurale viene allenata per 50 epoche, utilizzando $batchsize = 10$ e applicando l'ottimizzatore *Adam* [21] per l'apprendimento dei pesi. Il learning rate viene fissato a $lr = 1e^{-4}$ con parametri di ottimizzazione $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-8}$ e *weight decay*$= 5e^{-6}$. Notare che il numero di iterazioni per epoca varia a seconda del dataset di partenza, poiché il dataset del 26 giugno, con dimensione del voxel fissata a $30cm$, contiene 8000 campioni in più rispetto agli altri due dataset creati. Inoltre viene applicato del dropout sui layer fully-connected finali secondo una probabilità di 0.4 per evitare il fenomeno dell'overfitting.

Alla conclusione di ogni epoca si processa il testset in base ai pesi calcolati dalla rete neurale e si osserva l'andamento dell'errore sulle rotazioni e traslazioni predette.

Runs	Angles repr.	Transl. loss	Rot. loss	Rot rescaling	Transl rescaling	Depth	Refl	Robotcar run	Voxel size
1	quaternions	SmoothL1	QuatDistance	-	250	✓	✓	June 26	15 cm
2	quaternions	SmoothL1	QuatDistance	-	-	✓	-	June 26	15 cm
3	quaternions	SmoothL1	QuatDistance	-	-	-	✓	June 26	15 cm
4	quaternions	SmoothL1	SmoothL1	750	-	✓	✓	June 26	15 cm
5	quaternions	SmoothL1	SmoothL1	100	-	✓	✓	June 26	15 cm
6	quaternions	SmoothL1	SmoothL1	-	-	✓	✓	June 26	15 cm
7	euler	SmoothL1	SmoothL1	750	-	✓	✓	June 26	15 cm
8	euler	SmoothL1	SmoothL1	100	-	✓	✓	June 26	15 cm
9	euler	SmoothL1	SmoothL1	-	-	✓	✓	June 26	15 cm
10	quaternions	SmoothL1	QuatDistance	-	50	✓	✓	June 26	15 cm
11	quaternions	SmoothL1	QuatDistance	0.2	-	✓	✓	June 26	15 cm
12	quaternions	SmoothL1	QuatDistance	0.75	-	✓	✓	June 26	15 cm
13	quaternions	SmoothL1	QuatDistance	-	5	✓	✓	June 26	15 cm
14	quaternions	SmoothL1	QuatDistance	learned	learned	✓	✓	June 26	15 cm
15	quaternions	SmoothL1	SmoothL1	learned	learned	✓	✓	June 26	15 cm
16	euler	SmoothL1	SmoothL1	learned	learned	✓	✓	June 26	15 cm
17	quaternions	SmoothL1	QuatDistance	-	-	✓	✓	June 26	15 cm

Tabella 5.2: Elenco dei test effettuati sul dataset creato a partire dalla run del 26 giugno di Robotcar. Gli errori iniziali sono appartenenti all'intervallo $[-1.5m; +1.5m]$ sulle componenti di traslazione e $[-15degs; +15degs]$ sulle componenti di rotazione. La grandezza del voxel delle mappe utilizzate si aggira intorno ai $15cm$ di lato. Il dataset utilizzato è composto da 10000 esempi. Notare che per *QuatDistance* si intende la distanza angolare calcolata sui quaternioni con l'operatore di *atan2*

5.5 Analisi dei risultati

Dai test effettuati sulla rete neurale convoluzionale vengono ottenuti i risultati mostrati in tabella 5.3 e 5.4. In particolare, per ogni test vengono riportati i risultati migliori ottenuti dalla CNN sia per l'errore di traslazione sia per l'errore di rotazione. L'errore che intercorre tra predizione e groundtruth è calcolato nella maniera seguente:

- Distanza euclidea tra le componenti di traslazione della groundtruth e componenti di traslazione dell'errore stimato.
- Distanza angolare tra quaternioni calcolata tra la rotazione della groundtruth e l'errore di rotazione stimato.

Gli errori mostrati all'interno delle tabelle costituiscono l'errore medio calcolato sugli esempi appartenenti al test set. Notare che nel caso venga utilizzata la rappresentazione degli angoli di eulero durante la fase di test, la distanza angolare viene calcolata dapprima effettuando la trasformazione a quaternioni delle rotazioni.

Dati i risultati ottenuti da un test, un problema che risulta tutt'ora aperto riguarda la scelta tra la configurazione della rete che minimizza l'errore di traslazione o che minimizza l'errore di rotazione. Il criterio di selezione dei risultati mostrato possiede il difetto di non mettere a confronto i due tipi di errori, poiché la configurazione di pesi viene scelta andando ad osservare le performance solamente sulle traslazioni oppure sulle rotazioni. Nel caso il criterio di selezione si basi sull'errore, ad esempio, di rotazione, questo approccio comporta all'esclusione dei risultati che mostrano un errore leggermente maggiore per le rotazioni ed un errore nettamente minore sulle traslazioni. Di conseguenza sarebbe necessario definire una formula di distanza che unifichi l'errore di traslazione e rotazione, adottando, ad esempio, la funzione di loss proposta da Kendall et al. [20]. Tale formula di distanza si basa sull'errore di riproiezione di un insieme di punti, andando di fatto a evitare di confrontare le componenti di traslazione e rotazione di due pose. Dato un insieme di punti nello spazio 3D, si effettua la loro proiezione all'interno di due piani immagine: uno definito dalla pose camera corretta secondo la predizione della rete e uno definito dalla pose camera reale. L'errore viene successivamente computato andando a calcolare la media delle distanze euclidee sulle coppie di punti omologhi proiettati.

Considerando i range iniziali di errore della pose camera in input, si evince la difficoltà della rete neurale nel correggere lo scostamento iniziale in termini di traslazione, mentre

	Min translation error		**Min rotation error**	
Runs	Mean translation error (cm)	Mean rotation error (degs)	Mean translation error (cm)	Mean rotation error (degs)
1	112.2	5.824	116.0	5.182
2	**105.3**	9.539	110.3	6.639
3	111.0	5.182	116.6	5.031
4	123.5	5.161	124.4	**4.932**
5	110.6	5.689	114.7	5.213
6	106.7	7.19	110.9	6.283
7	167.8	5.973	169.5	5.942
8	169.2	7.549	173.8	7.451
9	194.8	5.393	197.6	5.218

Tabella 5.3: Risultati dei test rappresentati in tabella 5.1. Per ogni test vengono messi in evidenza i risultati migliori ottenuti dalla rete in termini di traslazione (a sinistra) e rotazione (a destra). La configurazione di iper-parametri della rete utilizzata per la seconda run porta i risultati migliori in termini di traslazione sul dataset utilizzato, mentre la quarta run porta i risultati migliori in termini di rotazione.

	Min translation error		**Min rotation error**	
Runs	Mean translation error (cm)	Mean rotation error (degs)	Mean translation error (cm)	Mean rotation error (degs)
1	72.39	6.891	74.98	6.52
2	72.83	4.086	77.27	**4.013**
3	74.88	4.598	78.94	4.416
4	78.42	4.749	128	4.337
5	76.68	4.212	80.94	4.146
6	76.68	6.604	71.92	5.401
7	70.79	4.337	107.6	4.101
8	81.1	4.687	80.74	4.059
9	74.44	7.058	74.41	4.871
10	**69.1**	5.314	70.17	5.303
11	69.43	5.174	81.55	4.632
12	74.07	4.405	78.0	4.35
13	75.54	6.467	78.0	5.508
14	76.16	4.609	81.7	4.421
15	71.41	7.067	72.42	5.296
16	72.14	4.869	74.79	4.729
17	76.49	4.639	78.0	4.192

Tabella 5.4: Risultati dei test descritti in tabella 5.2. Per ogni test vengono messi in evidenza i risultati migliori ottenuti dalla rete in termini di traslazione (a sinistra) e rotazione (a destra). La configurazione di iper-parametri della rete utilizzata per la decima run porta i risultati migliori in termini di traslazione sul dataset utilizzato, mentre la seconda run porta i risultati migliori in termini di rotazione.

essa si dimostra più incisiva in termini di correzione delle rotazioni. Inoltre, un altro fenomeno che emerge, riguarda la similarità delle performance della rete neurale al variare degli iper-parametri.

Durante la fase di testing, si è sperimentato un generale overfitting della rete dopo la metà delle epoche fissate all'avvio del test. La variazione dei parametri di dropout e dimensione del batch non portano a una variazione sostanziale dei risultati, così come la variazione del learning rate applicato alla rete neurale. Dati i risultati ottenuti dai test mediante queste variazioni, essi non sono stati considerati significativi e di conseguenza non sono stati riportati. Le uniche differenze sostanziali, che emergono in termini di performance, riguardano i differenti dataset utilizzati. In particolare, dimezzando la grandezza del lato del voxel, vengono ottenute delle performance decisamente migliori soprattutto in termini di traslazioni. Notare, che la *run2* in tabella 5.4 è quella che ha ottenuto le performance migliori in termini di rotazioni. La particolarità di questo test risiede nel non utilizzo dell'informazione di riflettanza dell'immagine LiDAR. Di conseguenza si potrebbe pensare che il dato di riflettanza non sia discriminante rispetto la correzione dell'errore sulla pose iniziale. Tuttavia, data la similarità dei risultati sullo stesso dataset, tale ipotesi andrebbe verificata mediante ulteriori test. La *run10* in tabella 5.4 mostra le performance migliori sulle traslazioni.

Da quanto emerge dalle diverse sperimentazioni svolte, è possibile affermare che vi sia la necessità di rivedere le modalità con cui il dataset viene creato. In particolare, bisognerebbe osservare il comportamento della rete utilizzando mappe aventi dimensioni del voxel minori. Inoltre, sarebbe necessario sperimentare l'utilizzo di dataset contenenti un maggiore quantitativo di esempi, con lo scopo di riuscire a mitigare il fenomeno dell'overfitting. Tuttavia, data la natura sperimentale del lavoro di tesi e i risultati ottenuti, l'approccio proposto risulta essere un buon punto di partenza.

Se anche con un aumento della risoluzione iniziale delle mappe non si riuscisse a migliorare le perfomance della rete neurale, allora sarebbe necessario considerare altri tipi di approcci per la rappresentazione delle scansioni 3D. Ad esempio, si potrebbe effettuare una ricostruzione delle superfici andando ad applicare degli algoritmi di meshing a partire dalle point-cloud e successivamente effettuare un confronto con la rappresentazione mediante voxel.

Un'altra possibile strategia attuabile, per il miglioramento delle performance, consisterebbe nell'applicazione dell'operazione di *iterative refinement*. Sarebbe necessario

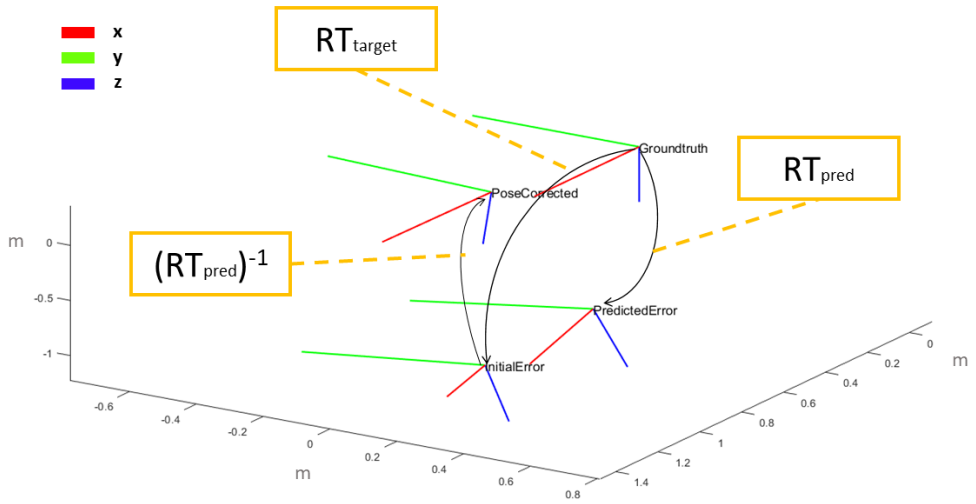


Figura 5.3: Esempio schematico di rototraslazioni applicate per correggere la pose di input in base all'errore predetto. Notare che la direzioni di osservazione della camera è definita dall'asse x.

addestrare diverse reti specializzate nella correzione di pose rispetto a range di errore differenti. Ad esempio, facendo riferimento ai risultati ottenuti in tabella 5.4, andrebbe addestrata una CNN specializzata nella correzione di errori in un intervallo di $[-75cm; +75cm]$ sulle traslazioni e di $[-4.5degs; +4.5degs]$ sulle rotazioni. Ad ogni errore predetto dalla rete neurale si potrebbe correggere la pose iniziale e ottenere, secondo la nuova pose, nuove immagini LiDAR da utilizzare con la rete successiva, specializzata in correzioni di errori che ricadono in range minori.

Per osservare la bontà della stima dell'errore predetto dalla rete neurale, si è proceduto a ripetere l'operazione di raytracing partendo dalla pose camera a cui è applicata la correzione dell'errore. L'obiettivo è quello di verificare, a livello percettivo, l'effettivo miglioramento apportato dalla CNN allenata.

Per effettuare l'operazione appena descritta, partendo dalla pose obiettivo p_{target} e predetta p_{pred} vengono ricavate le matrici di rototraslazione RT_{target} e RT_{pred}. Successivamente a RT_{target} viene pre-moltiplicata RT_{pred}^{-1}, che identifica la trasformazione geometrica inversa di RT_{pred}. L'obiettivo è quello di rototraslare la pose errata di partenza

nella posizione e orientamento corretti, casistica che si verifica quando $RT_{pred} = RT_{target}$. In figura 5.3 viene mostrato un esempio schematico del processo appena descritto.

Una volta ottenuta la pose con applicata la correzione, viene applicata l'operazione di raytracing sulla mappa. Nelle figure 5.4, 5.5, 5.6 e 5.7 vengono mostrati degli esempi del canale di profondità di alcune immagini LiDAR ottenute mediante le diverse pose camera. Partendo da sinistra l'ordine di rappresentazione è il seguente: pose iniziale con errore, pose con applicata la correzione e groundtruth. A livello percettivo si può notare come la rete neurale riesca a correggere meglio gli errori di rotazione rispetto agli errori di traslazione. A livello di proiezione eventuali rotazioni della pose camera sono maggiormente percettibili e da questo fenomeno è possibile comprendere come la CNN riesca a correggere meglio l'orientamento della camera.

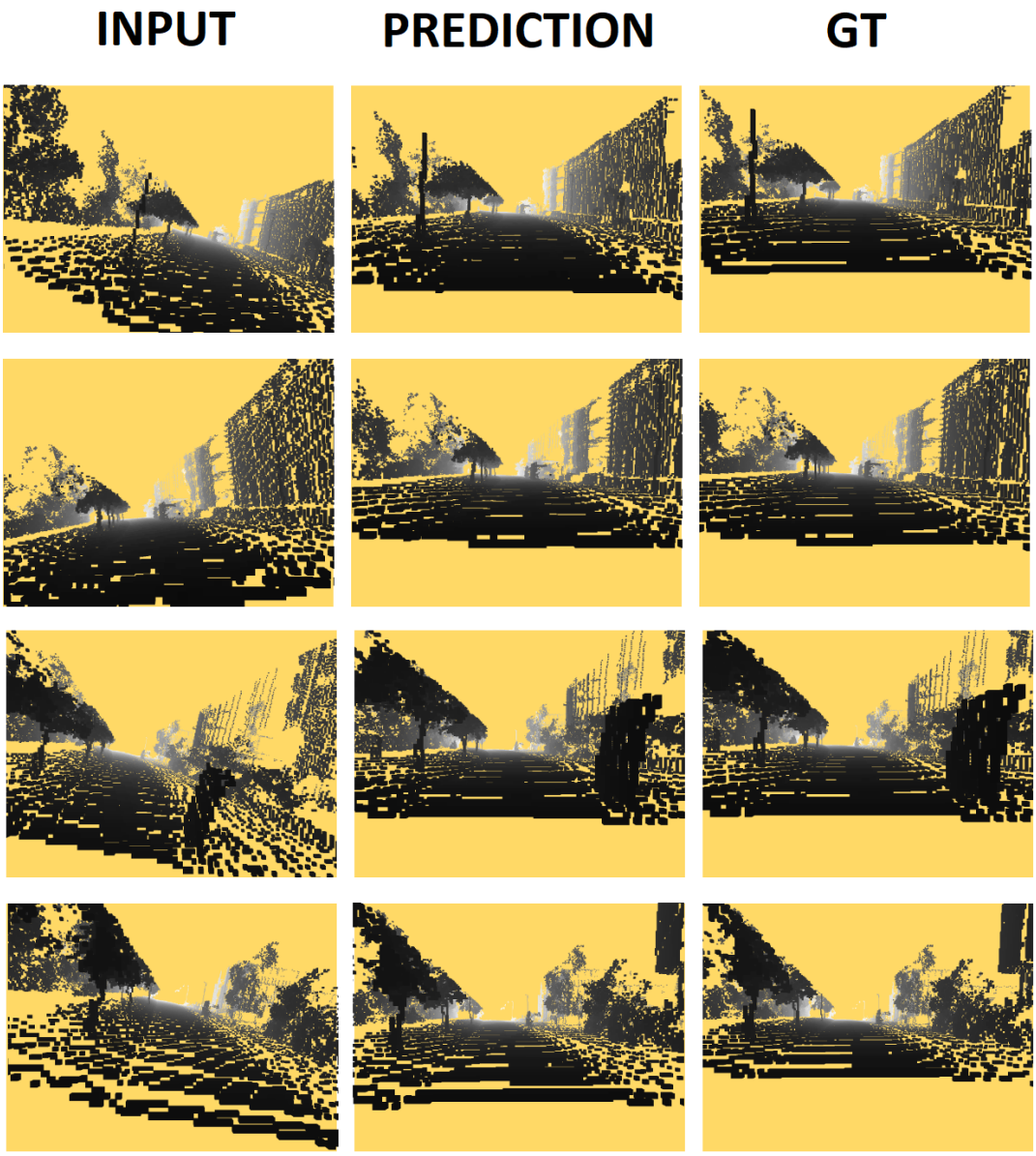


Figura 5.4: Confronto delle immagini LiDAR ottenute a partire da pose differenti. Per ogni tripla le pose di partenza utilizzate per la proiezione della mappa sono le seguenti: a sinistra l'immagine viene ottenuta a partire dalla pose errata in input, al centro l'immagine viene ottenuta a partire dalla pose corretta in base all'errore stimato dalla rete e a destra viene rappresentata l'immagine ottenuta dalla pose di groundtruth.

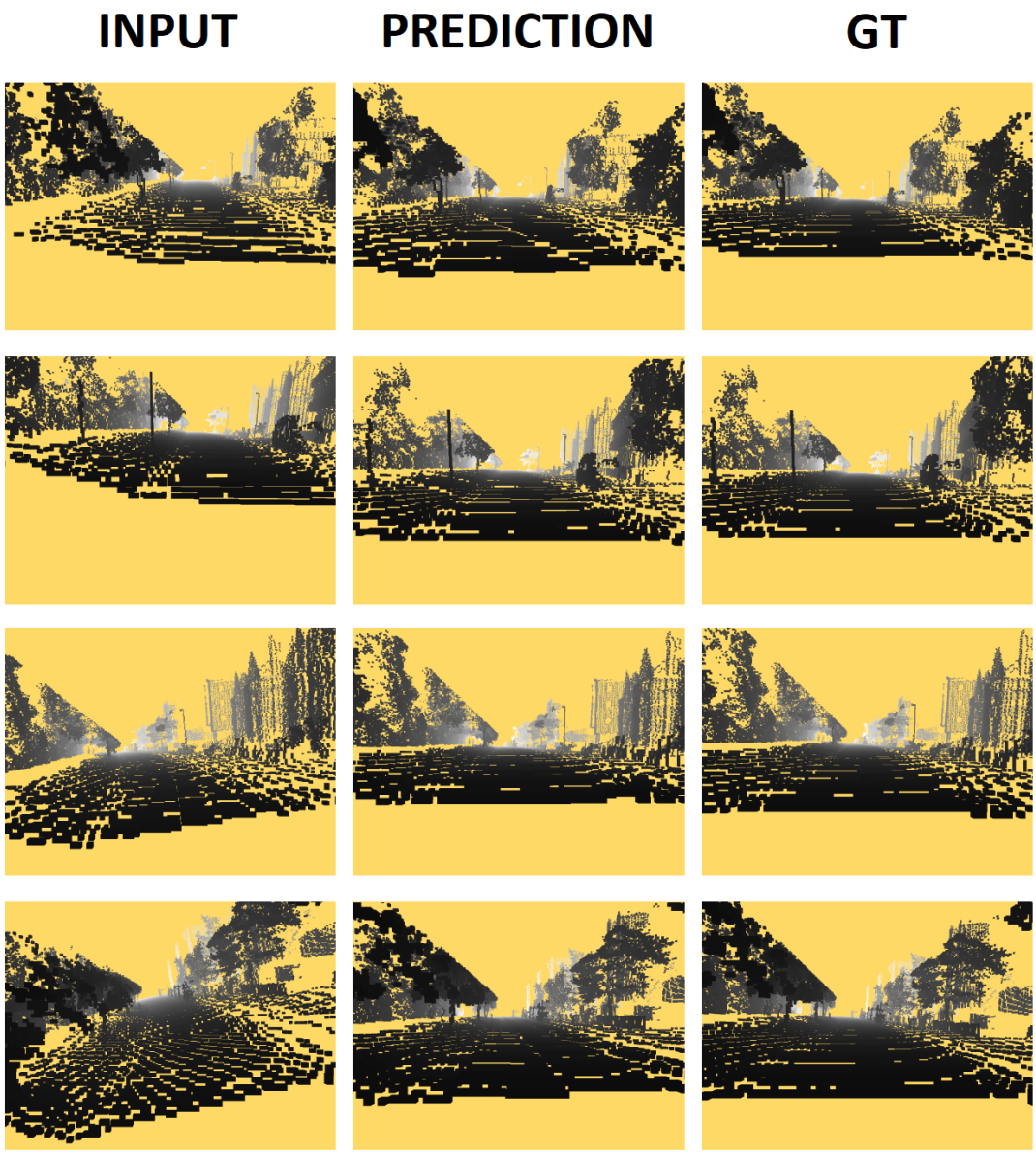


Figura 5.5: Confronto delle immagini LiDAR ottenute a partire da pose differenti. Per ogni tripla le pose di partenza utilizzate per la proiezione della mappa sono le seguenti: a sinistra l'immagine viene ottenuta a partire dalla pose errata in input, al centro l'immagine viene ottenuta a partire dalla pose corretta in base all'errore stimato dalla rete e a destra viene rappresentata l'immagine ottenuta dalla pose di groundtruth.

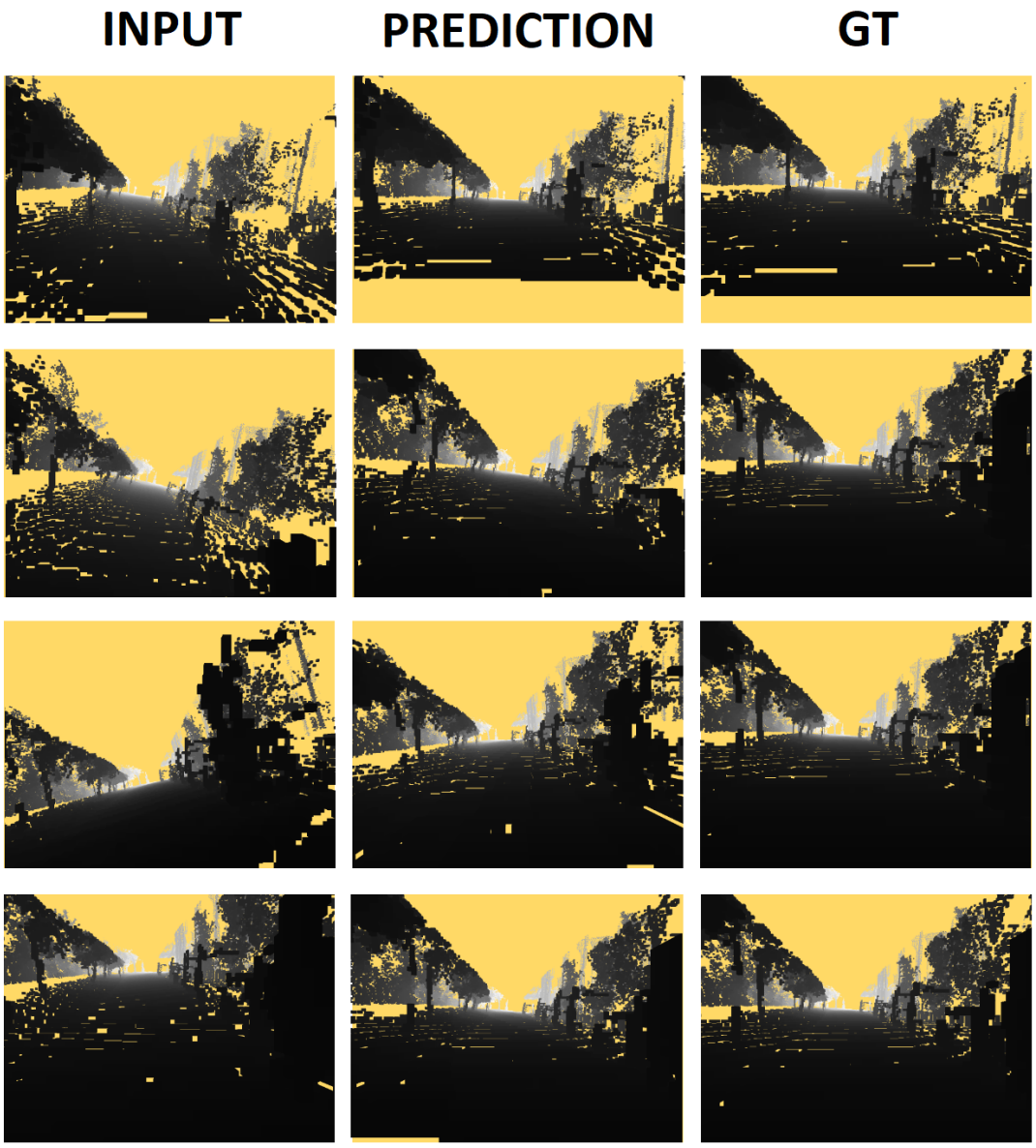


Figura 5.6: Confronto delle immagini LiDAR ottenute a partire da pose differenti. Per ogni tripla le pose di partenza utilizzate per la proiezione della mappa sono le seguenti: a sinistra l'immagine viene ottenuta a partire dalla pose errata in input, al centro l'immagine viene ottenuta a partire dalla pose corretta in base all'errore stimato dalla rete e a destra viene rappresentata l'immagine ottenuta dalla pose di groundtruth.

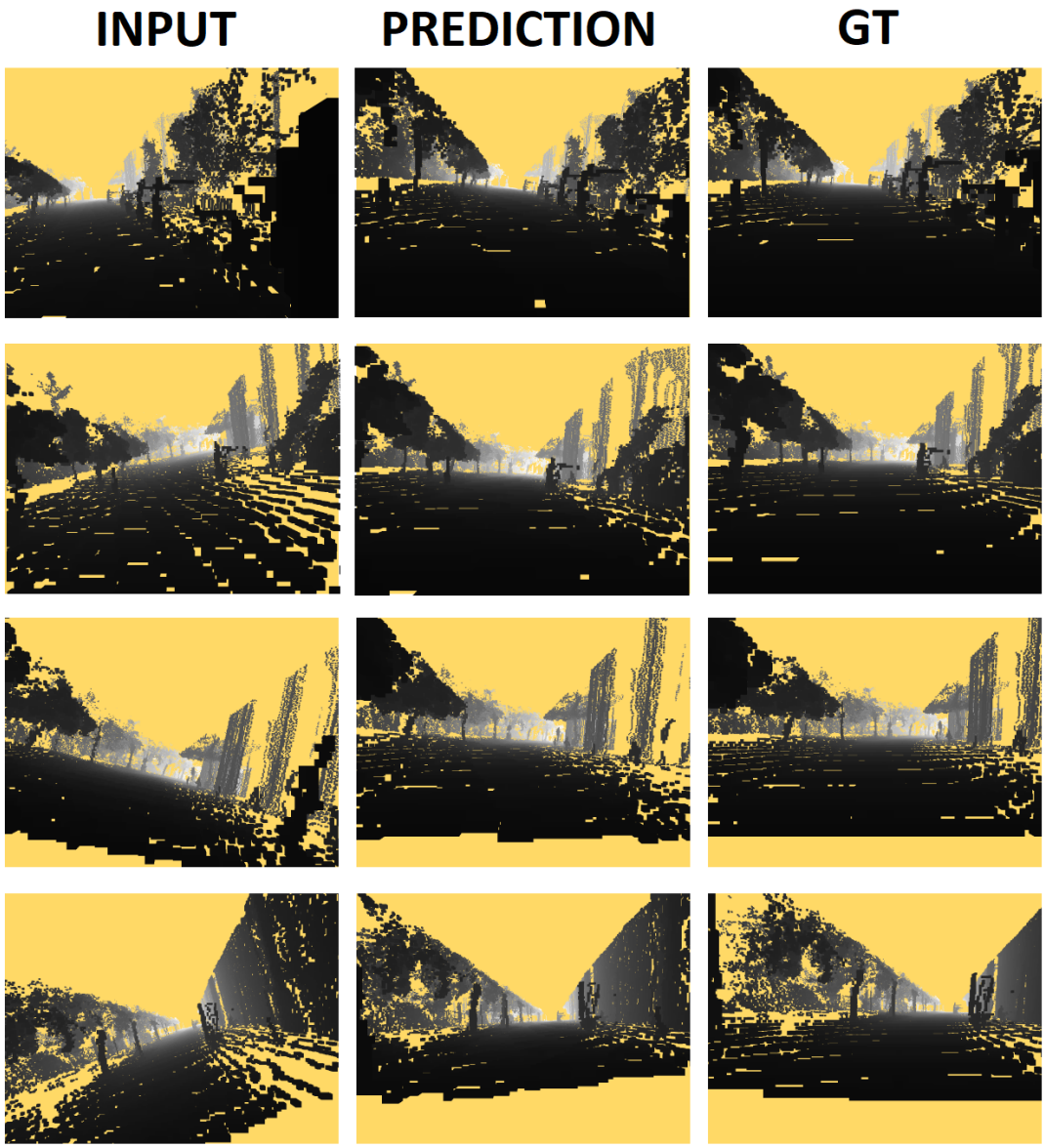


Figura 5.7: Confronto delle immagini LiDAR ottenute a partire da pose differenti. Per ogni tripla le pose di partenza utilizzate per la proiezione della mappa sono le seguenti: a sinistra l'immagine viene ottenuta a partire dalla pose errata in input, al centro l'immagine viene ottenuta a partire dalla pose corretta in base all'errore stimato dalla rete e a destra viene rappresentata l'immagine ottenuta dalla pose di groundtruth.

Capitolo 6

Conclusioni e sviluppi futuri

In questo lavoro di tesi si è affrontato il problema della localizzazione di un veicolo, andando ad implementare una rete neurale convoluzionale per effettuare la stima dell'errore su una pose (posizione e orientamento) di input. La particolarità del sistema implementato risiede nell'utilizzo congiunto di immagini RGB e immagini LiDAR, queste ultime ottenute andando ad integrare a livello immagine il dato di profondità e riflettanza proveniente da una mappa.

L'approccio sperimentale sviluppato costituisce un metodo innovativo e si basa su tecniche oggigiorno molto diffuse all'interno del campo della computer vision, rendendolo di fatto di massima attualità.

I risultati ottenuti mostrano come il lavoro svolto possa costituire un buon punto di partenza per affrontare il problema della localizzazione. Tuttavia, è necessario migliorare le performance del sistema sviluppato ed è importante andare a considerare range di errore iniziale maggiore, in maniera tale da rendere tale approccio effettivamente applicabile nell'ambito automotive.

Durante il lavoro di tesi svolto non ci si è concentrati sull'ottimizzazione dell'operazione di raytracing applicata, anche a causa dei limiti dell'hardware a disposizione. Di conseguenza, al momento l'approccio sviluppato non è applicabile direttamente in un contesto reale, tuttavia con le dovute ottimizzazioni si potrebbe portare il processo di proiezione delle mappe LiDAR a *real-time*. Inoltre, una maggiore velocità della creazione degli esempi si tradurrebbe in una maggiore facilità nella creazione di dataset di dimensioni maggiori. In alternativa, con tempi di computazione nettamente inferiori, si potrebbe pensare di applicare tecniche di data augmentation per la generazione di

immagini LiDAR a *runtime* direttamente durante la fase di training della rete neurale.

Altro elemento da considerare riguarda la tipologia di rappresentazione della mappa 3D. In particolare, è necessario non solo analizzare la relazione che intercorre tra le performance della rete ed ulteriori variazioni della dimensione dei voxel della mappa, ma bisognerebbe anche sperimentare l'utilizzo, ad esempio, di mesh per la ricostruzione della scena, in maniera tale da comprendere qual è il tipo di ricostruzione più efficace.

Il lavoro portato avanti si dimostra un'ottima base predisposta ad ulteriori sviluppi futuri. A seguito dell'analisi dei risultati ottenuti, si è dedotto che i principali step da applicare per migliorare l'approccio sviluppato siano i seguenti:

- Si ritiene necessario creare un dataset con un numero di esempi maggiori, ottenuti da mappe di Robotcar diverse e che siano caratterizzate da condizioni ambientali differenti. Lo scopo è capire se all'aumentare della variabilità degli input e del numero di esempi forniti, si percepisca un miglioramento della capacità di generalizzazione della rete neurale.

- Occorre allenare diverse reti neurali convoluzionali con il fine di specializzarle su diversi range di errore, così da rendere applicabile un processo iterativo per il miglioramento dell'errore finale di predizione.

- Bisognerebbe effettuare ulteriori sperimentazioni utilizzando diverse reti neurali convoluzionali presenti all'interno dei lavori esistenti in letteratura, in maniera tale da permettere un confronto tra i diversi tipi di architetture.

In conclusione, per rendere la rete utilizzabile in uno scenario reale, migliorandone le sue prestazioni e garantendone una validazione esaustiva, si ritiene necessaria l'applicazione dei passi appena descritti.

Capitolo 7

Ringraziamenti

Innanzitutto voglio ringraziare la mia famiglia per avermi sempre sostenuto durante questi anni. Ringrazio soprattutto i miei genitori che hanno fatto di tutto per permettermi di realizzare i miei progetti e spero che questo risultato possa ripagarvi dei sacrifici che in silenzio avete sempre fatto per me.

Ringrazio il Prof. Sorrenti per avermi dato la possibilità di svolgere un lavoro di tesi molto attuale e Daniele per avermi aiutato nel momento del bisogno nonostante le distanze. Ringrazio Augusto, Simone ed i ragazzi del team IRAlab per avermi accompagnato in questa esperienza e soprattutto per aver reso divertente il periodo di permanenza in laboratorio. Grazie a tutti voi ho avuto la possibilità di capire cosa significa far parte di un gruppo di ricerca e di dare un senso più profondo alla mia esperienza universitaria.

Ringrazio Sandro, Claudio, Victor, Ilaria e gli altri compagni di Università per tutte le ore passate assieme a studiare e a fare progetti. In questi anni siete stati un punto di riferimento e grazie a voi sono riuscito a migliorarmi costantemente. Non dimenticherò mai gli episodi divertenti, le prese in giro, le litigate ai tavoloni ed i momenti imbarazzanti creati dal caratteraccio di Sandro e Claudio (ma ci piacete così).

Ringrazio i miei amici della Brianza (non me ne vogliate siete troppi da nominare) per essermi sempre vicino. Ringrazio gli amici storici che, nonostante i cambiamenti della vita e le strade diverse intraprese, non hanno mai perso la voglia di continuare a sentirci. Ringrazio gli amici di più recente data per avermi ispirato a dare il massimo e per avermi supportato nei bei momenti della mia vita, ma soprattutto in quelli più difficili.

Se sono riuscito ad arrivare fin qui è anche merito di tutti voi, perciò GRAZIE.

Bibliografia

[1] Alireza Asvadi, Luis Garrote, Cristiano Premebida, Paulo Peixoto, and Urbano J. Nunes. Multimodal vehicle detection: Fusing 3D-LIDAR and color camera data. *Pattern Recognition Letters*, 0:1–10, 2017. ISSN 01678655. doi: 10.1016/j.patrec.2017.09.038.

[2] Stanley M. Bileschi. Fully automatic calibration of lidar and video streams from a vehicle. *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1457–1464, 2009.

[3] Daniel J Butler, Jonas Wulff, Garrett B Stanley, and Michael J Black. for Optical Flow Evaluation. *Eccv*, pages 611–625, 2012. doi: 10.1007/978-3-642-33783-3_44.

[4] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5410–5418, 2018.

[5] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-View 3D Object Detection Network for Autonomous Driving. *Cvpr2017*, pages 1907–1915, 2016. doi: 10.1109/CVPR.2017.691.

[6] Navneet Dalal and William Triggs. Histograms of Oriented Gradients for Human Detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR05*, 1(3):886–893, 2004. ISSN 1063-6919. doi: 10.1109/CVPR.2005.177.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[8] Alexey Dosovitskiy, Philipp Fischery, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. FlowNet: Learning optical flow with convolutional networks. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:2758–2766, 2015. ISSN 15505499. doi: 10.1109/ICCV.2015.316.

[9] Andreas Geiger, Frank Moosmann, Omer Car, and Bernhard Schuster. Automatic camera and range sensor calibration using a single shot. *2012 IEEE International Conference on Robotics and Automation*, pages 3936–3943, 2012.

[10] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[11] Ross Girshick. Fast R-CNN. *IEEE International Conference on Computer Vision (ICCV 2015)*, pages 1440–1448, 2015. ISSN 15505499. doi: 10.1109/iccv.2015.169.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[13] Hirschmuller H. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, 2008. ISSN 0162-8828.

[14] Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1):185 – 203, 1981. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(81)90024-2.

[15] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. doi: 10.1007/s10514-012-9321-0.

[16] Du Q. Huynh. Metrics for 3D rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009. ISSN 09249907. doi: 10.1007/s10851-009-0161-2.

[17] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[18] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:6555–6564, 2017. ISSN 1063-6919. doi: 10.1109/CVPR.2017.694.

[19] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems 30*, pages 5574–5584. Curran Associates, Inc., 2017.

[20] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[22] Jesse Levinson and Sebastian Thrun. Automatic online calibration of cameras and lasers. In *Robotics: Science and Systems IX, Technische Universität Berlin, Berlin, Germany, June 24 - June 28, 2013*, 2013. doi: 10.15607/RSS.2013.IX.029.

[23] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[24] D.G. Lowe. Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, pages 1150–1157 vol.2, 1999. ISSN 0-7695-0164-8. doi: 10.1109/ICCV.1999.790410.

[25] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The Oxford RobotCar dataset. *International Journal of Robotics Research*, 36 (1):3–15, 2017. ISSN 17413176. doi: 10.1177/0278364916679498.

[26] Donald Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129 – 147, 1982. ISSN 0146-664X. doi: https://doi.org/10.1016/0146-664X(82)90104-6.

[27] Etienne Memin and Patrick Perez. Dense estimation and object-based segmentation of the optical flow with robust techniques. *IEEE Transactions on Image Processing*, 7(5):703–719, 1998. ISSN 10577149. doi: 10.1109/83.668027.

[28] Faraz M. Mirzaei, Dimitrios G. Kottas, and Stergios I. Roumeliotis. 3D LIDAR-camera intrinsic and extrinsic calibration: Identifiability and analytical least-squares-based initialization. *International Journal of Robotics Research*, 31(4):452–467, 2012. ISSN 02783649. doi: 10.1177/0278364911435689.

[29] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987, 2002. ISSN 01628828. doi: 10.1109/TPAMI.2002.1017623.

[30] Geoffrey Pascoe, William Maddern, and Paul Newman. Direct visual localisation and calibration for road vehicles in changing city environments. In *The IEEE International Conference on Computer Vision (ICCV) Workshops*, December 2015.

[31] Enrico Piazza, Andrea Romanoni, and Matteo Matteucci. Real-time cpu-based large-scale 3d mesh reconstruction. *CoRR*, abs/1801.05230, 2018.

[32] Cristiano Premebida, J Carreira, Jorge Batista, and Urbano Nunes. Pedestrian detection combining rgb and dense lidar data. In *IEEE International Conference on Intelligent Robots and Systems*, pages 4112–4117, 09 2014. doi: 10.1109/IROS.2014.6943141.

[33] Cristiano Premebida, Luis Garrote, Alireza Asvadi, A. Pedro Ribeiro, and Urbano Nunes. High-resolution LIDAR-based depth mapping using bilateral filter. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, pages 2469–2474, 2016. doi: 10.1109/ITSC.2016.7795953.

[34] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[35] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[36] Carl Rosendahl. Computer graphics in commercial and broadcast production (Panel). *ACM SIGGRAPH Computer Graphics*, 18(3):205, 1984. ISSN 00978930. doi: 10.1145/964965.808599.

[37] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[38] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47:7–42, 04 2002. doi: 10.1023/A:1014573219977.

[39] Nick Schneider, Florian Piewak, Christoph Stiller, and Uwe Franke. RegNet: Multimodal Sensor Registration Using Deep Neural Networks. *Proc. IEEE Intelligent Vehicles Symposium (IV)*, pages 1803–1810, 2017. doi: 10.1109/IVS.2017.7995968.

[40] Mohammad Javad Shafiee, Brendan Chywl, Francis Li, and Alexander Wong. Fast YOLO: A fast you only look once system for real-time embedded object detection in video. *CoRR*, abs/1709.05943, 2017.

[41] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012.

[42] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[43] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity invariant cnns. *CoRR*, abs/1708.06500, 2017.

[44] Victor Vaquero, Ivan Del Pino, Francese Moreno-Noguer, Joan Sola, Alberto Sanfeliu, and Juan Andrade-Cetto. Deconvolutional networks for point-cloud vehicle detection and tracking in driving scenarios. *2017 European Conference on Mobile Robots, ECMR 2017*, 2017. doi: 10.1109/ECMR.2017.8098657.

[45] Paul Viola, Michael J. Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161, 2005. ISSN 09205691. doi: 10.1007/s11263-005-6644-8.

[46] Andreas Wedel, Daniel Cremers, Thomas Pock, and Horst Bischof. Structure- and motion-adaptive regularization for high accuracy optic flow. *Proceedings of the*

IEEE International Conference on Computer Vision, pages 1663–1668, 2009. ISSN 1550-5499. doi: 10.1109/ICCV.2009.5459375.

[47] Jure Zbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, 17:1–32, 2016.

[48] Fuzhen Zhang. Quaternions and matrices of quaternions. *Linear Algebra and its Applications*, 251:21 – 57, 1997. ISSN 0024-3795. doi: https://doi.org/10.1016/0024-3795(95)00543-9.