UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria Industriale DII

Corso di Laurea Magistrale in Ingegneria Aerospaziale

SVILUPPO DI UN METODO PER LA MAPPATURA TERMICA MEDIANTE LA SENSOR FUSION TRA IL LIDAR E LA TERMOCAMERA INSTALLATI A BORDO DEL ROVER MORPHEUS

Relatore: Stefano Debei

Correlatore: Sebastiano Chiodini

Davide De Pazzi
1204908

Anno Accademico 2021/2022

Sommario

In questo elaborato viene descritto un metodo per eseguire la mappatura termica di un ambiente mediante la *sensor fusion* tra le *point cloud* tridimensionali prodotte da un LiDAR e le immagini termiche radiometriche prodotte da una termocamera.
Gli strumenti sono stati installati a bordo del rover MORPHEUS, progettato e costruito dagli studenti dell'Università di Padova. Prima di procedere all'acquisizione dei dati, la termocamera e il sistema termocamera–LiDAR sono stati opportunamente calibrati.
L'obbiettivo è quello di creare una rappresentazione tridimensionale discreta della distribuzione della temperatura apparente all'interno dell'ambiente. Per ottenere questo risultato le *point cloud* vengono integrate con le corrispondenti immagini termiche, sfruttando le relazioni di proiezione prospettica, per poi essere registrate grazie alle informazioni ottenute ricostruendo la traiettoria del rover. L'insieme delle *point cloud* termiche registrate ed unite viene quindi elaborato da una opportuna funzione di decomposizione spaziale, suddividendo l'ambiente in elementi discreti che possono essere associati alla temperatura media dei punti contenuti nel loro volume.
L'algoritmo che implementa tale processo in ambiente MATLAB è stato testato utilizzando vari dataset, raccolti sia all'interno che all'esterno di un edificio. Alcune delle mappe di distribuzione di temperatura ottenute sono state poi analizzate, per verificare che la posizione degli elementi termicamente significativi individuati al loro interno (potenzialmente sfruttabili come *landmark* per la navigazione) fosse compatibile con le misurazioni effettuate sul posto.

Indice

Figure e tabelle

Capitolo 1

Introduzione

1.1 I rover per l'esplorazione planetaria

I mezzi robotici hanno ricoperto da sempre un ruolo di primo piano nel campo dell'esplorazione spaziale. Sono stati sufficienti pochi decenni per passare dalla messa in orbita dei primi satelliti artificiali all'invio di sonde automatiche verso i pianeti interni ed esterni del Sistema Solare. Da questi *orbiter* si è poi passati ai *lander*, capaci di posarsi sulla superficie dei corpi planetari e di interagire direttamente con il loro ambiente, e quindi ai *rover* veri e propri: veicoli indipendenti più o meno autonomi, in grado di muoversi attivamente sul terreno.

Attualmente, con la presenza umana nello spazio limitata all'orbita terrestre bassa, l'attività di esplorazione è strettamente legata all'utilizzo di sistemi automatici.
Il loro design, i loro componenti (dai sistemi di locomozione ai manipolatori ai sensori) e le loro capacità hanno quindi subito una costante evoluzione. Basti pensare che il rover *Sojourner*, il primo a muoversi sul suolo marziano nel Luglio del 1997, pesava appena 11.5 kg ed è riuscito a percorrere poco più di 100 m prima di smettere di funzionare. Il rover *Curiosity*, invece, pesa più di 900 kg e attualmente si trova ad oltre 23 km dal punto in cui ha toccato il suolo di Marte nell'Agosto del 2012.

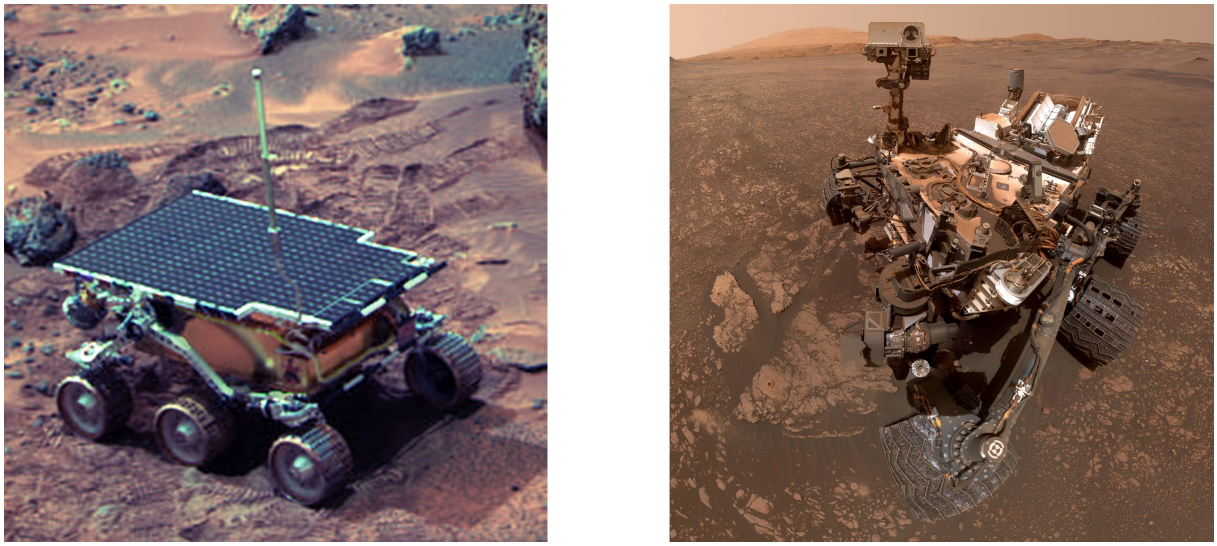

Figura 1-1: Rover *Sojourner* (a sinistra) e rover *Curiosity* (a destra). Fonte: NASA/JPL.

Nel tempo le stesse tecnologie impiegate da questi rover hanno trovato sempre più applicazioni anche in ambito terrestre, sia nel campo industriale che in quello della ricerca scientifica o delle attività di soccorso.

Il ricorso ai rover autonomi in ambito spaziale non è dovuto solo alla loro capacità di operare in una varietà di ambienti ostili, ma anche dalla durata estremamente lunga che caratterizza tutte le missioni dirette oltre l'orbita lunare.
La distanza a cui si trovano corpi esplorati, in realtà, rappresenta una sfida tecnica tanto quanto le condizioni incontrate sulla loro superficie. Il ritardo nelle comunicazioni, che già su Marte può superare i 15 minuti, esclude infatti qualsiasi possibilità di un controllo diretto da remoto. Per questa ragione, i moderni rover per l'esplorazione planetaria devono necessariamente presentare elevatissimi livelli di autonomia.

1.2 Ruolo dei sistemi di percezione

I sistemi di percezione rappresentano un elemento fondamentale nel funzionamento di qualsiasi rover realmente autonomo, sia in ambito spaziale che in ambito terrestre.
Questi sistemi non si limitano a raccogliere i dati previsti dagli obbiettivi di missione: sono anche alla base della capacità di interpretare l'ambiente circostante che conferisce al rover la possibilità di operare in modo indipendente.

Un tipico esempio è fornito dal problema della navigazione e della guida autonome. Un rover dovrebbe infatti essere in grado di determinare la propria posizione all'interno dell'ambiente che deve attraversare, individuare la propria destinazione e pianificare un percorso privo di ostacoli per raggiungerla. A questo risultato spesso contribuiscono anche dispositivi come accelerometri e giroscopi, che forniscono informazioni sullo stato in cui si trova il rover, ma il ruolo principale è ricoperto da quei sensori in grado di stabilire la morfologia del terreno e rilevare gli eventuali ostacoli con un anticipo sufficiente.

I sistemi di percezione basati su sensori come i LiDAR e i RADAR, che effettuano misurazioni di distanza, possono stabilire con una precisione la posizione dei vari elementi dell'ambiente. La loro incapacità di percepire dettagli prettamente visuali (come il colore o la *texture* di una data superficie) però rende i dati che acquisiscono più difficili da interpretare.
Sensori di questo tipo operano in modo attivo, emettendo energia e valutando le modalità della sua interazione con la materia circostante. Ciò significa che non richiedono una sorgente esterna per illuminare gli oggetti osservati, ma risentono comunque di tutti quei fenomeni che possono interferire con la propagazione della radiazione elettromagnetica.

Le telecamere e le termocamere, al contrario, producono vere e proprie immagini. I singoli oggetti presenti sulla scena possono quindi essere riconosciuti più facilmente, ma le loro dimensioni e la loro posizione non vengono determinate direttamente. Le tecniche di stereovisione (monoculare oppure binoculare) consentono di superare questa limitazione, confrontando più immagini della stessa scena per ricavare informazioni di profondità, ma l'accuratezza dei risultati ottenuti può essere molto variabile.

In generale l'acquisizione delle immagini nel campo della luce visibile può avvenire solo quando le condizioni di illuminazione nell'ambiente sono adeguate. Nel campo dell'infrarosso, invece, si possono verificare due casi diversi: in base all'intervallo di lunghezze d'onda a cui è sensibile, la termocamera potrebbe rilevare l'emissione termica di un corpo oppure la componente infrarossa della radiazione che riflette.
In linea teorica misurare la quantità di radiazione che viene emessa da una superficie consente di determinarne la temperatura apparente, conoscendo a priori le proprietà del materiale di cui è composta. Il livello di illuminazione naturale non ha quindi un effetto diretto sulla formazione delle immagini acquisite nell'infrarosso termico, anche se può influenzare pesantemente i calcoli radiometrici e quindi i valori di temperatura ottenuti.

In questo contesto, la scelta di sfruttare contemporaneamente diverse tipologie di sensori appare particolarmente vantaggiosa. Si parla quindi di *sensor fusion* (combinazione di sensori) come di un processo in cui i dati provenienti da sistemi di percezione con caratteristiche differenti sono integrati tra loro, in modo da fornire più informazioni di quelle ottenibili analizzando gli stessi dati separatamente.

Tenendo conto delle considerazioni precedenti, desta particolare interesse la *sensor fusion* di un sistema di percezione che misura delle distanze (come un LiDAR) con uno in grado di produrre direttamente delle immagini (come una telecamera RGB o una termocamera).
In particolare, le *point cloud* prodotte da un LiDAR 3–D e le immagini termiche radiometriche prodotte da una termocamera possono essere combinate per creare una rappresentazione in tre dimensioni delle caratteristiche termiche di un particolare ambiente: nello specifico, una mappa discreta della distribuzione spaziale della temperatura apparente.

Una mappa di questo tipo può avere diversi utilizzi, sia nell'ambito dei rover per l'esplorazione planetaria che in quello della robotica per applicazioni terrestri:

- Evidenziare la geometria globale di un ambiente, indicando gli elementi che si discostano da tale geometria come potenziali ostacoli al passaggio del rover.
- Individuare elementi termicamente distintivi da utilizzare come punti di riferimento per la navigazione autonoma e la mappatura dell'ambiente.
- Fornire informazioni sulle caratteristiche del suolo, ad esempio sfruttando le relazioni che legano la traversabilità dei terreni granulari (come la regolite, tipica degli ambienti lunari e marziani) con la loro inerzia termica.
- Analizzare lo stato termico di particolari oggetti presenti nell'ambiente (tubazioni, quadri elettrici sotto tensione, materiali infiammabili) per valutarne la pericolosità.

1.3 Il rover MORPHEUS

Il LiDAR 3–D e la termocamera utilizzati per creare la mappa di distribuzione di temperatura sono quelli presenti a bordo del rover MORPHEUS, costruito nell'ambito di un progetto gestito dal Centro di Ateneo di Studi e Attività Spaziali (CISAS) dell'Università di Padova.

Questo progetto è finalizzato allo sviluppo di un rover per l'esplorazione planetaria, con il quale simulare missioni di interesse scientifico in ambiente lunare o marziano.
Le specifiche originali del rover MORPHEUS sono state fissate sulla base dei requisiti imposti dalla European Rover Challenge, una competizione studentesca internazionale che si svolge ogni anno in Polonia. Si tratta quindi di un veicolo semi-autonomo, che può essere guidato a distanza da un operatore oppure muoversi in autonomia sfruttando i dati forniti dai vari sensori presenti a bordo. La sua struttura generale è simile a quella dei rover recentemente inviati sulla Luna e su Marte, anche se in questo caso sono state volutamente adottate delle soluzioni tecniche meno sofisticate per privilegiare la semplicità costruttiva e l'affidabilità.

Il telaio del rover è costituito da un assemblaggio di profilati a sezione quadrata di 20×20 mm realizzati in Acciaio spesso 2 mm. Un certo numero di piastre in Acciaio o Alluminio completano la struttura, formando il vano in cui sono alloggiati i componenti elettrici ed elettronici.
Il rover si muove su tre coppie di ruote, montate su un sistema di sospensione composto da tre bilancieri indipendenti (due laterali e uno posteriore). Il centro di ogni bilanciere, a sua volta, è collegato alle piastre del telaio tramite un cuscinetto flangiato. Questa configurazione (che non

prevede elementi elastici o ammortizzanti) dovrebbe massimizzare il numero di ruote mantenute a contatto con il terreno quando il rover si trova a dover superare un ostacolo.
L'ingombro complessivo del rover MORPHEUS è di circa 1300×850×350 mm, comprendendo le ruote, con il telaio mantenuto a circa 200 mm dal terreno. Il suo peso varia tra i 40 e i 50 kg, in base alla quantità di apparecchiature installate in un dato momento.

Figura 1-2: Struttura e sistema di locomozione del rover MORPHEUS.

Ognuna delle ruote è collegata ad un motoriduttore *brushless* da una cinghia dentata (che funge anche da giunto elastico tra i due assi) tenuta in posizione da un tensionatore. In origine questi motoriduttori sono stati dimensionati assumendo di dover raggiungere una velocità massima di 1 m/s (su una pendenza massima di 15°) con un tempo di accelerazione pari a 5 s.
Si è anche scelto di adottare una configurazione di sterzo a strisciamento (*skid steering*) simile a quella comunemente utilizzata nei veicoli cingolati: la rotazione intorno all'asse verticale non avviene modificando la direzione di alcune delle ruote, ma facendo avanzare i due lati del rover a velocità differenti (differenziando la coppia erogata dai singoli motoriduttori).

Il computer di bordo del rover MORPHEUS è costituito da una scheda NVIDIA JETSON TX2 che si interfaccia con tutti i componenti principali del veicolo, compresi i motori delle ruote (con i relativi *encoder*) e i sensori utilizzati per la navigazione autonoma.
Su questa scheda sono installati il sistema operativo Ubuntu ed un *framework* per la robotica denominato ROS (*Robot Operating System*). La struttura computazionale di un sistema basato su ROS incorpora i seguenti elementi fondamentali:

- *Nodes*: Ogni nodo rappresenta un processo che compie una particolare serie di operazioni utilizzando le risorse disponibili.

- *Messages*: I singoli nodi interagiscono tra loro scambiandosi dei messaggi, la cui struttura è predeterminata in base alla tipologia dei dati contenuti.

- *Topics*: Tutti i messaggi vengono pubblicati all'interno di categorie che identificano i loro contenuti, quindi per accedere ad una certa tipologia di dati i nodi devono semplicemente sottoscrivere il *topic* corrispondente.

– *Services*: In generale i nodi che pubblicano dei messaggi e i nodi che li ricevono non sono consapevoli della reciproca esistenza, quindi sono previsti appositi servizi per gestire tutte le comunicazioni che non possono avvenire seguendo questa logica.

– *Master*: Senza questo elemento a gestire l'organizzazione del sistema, i nodi non sarebbero in grado di pubblicare messaggi, sottoscrivere *topic* o usufruire dei servizi.

– *Bag*: Tutti i dati acquisiti o elaborati da uno specifico nodo possono essere memorizzati in un file *rosbag* con lo stesso formato dei relativi messaggi, che essendo predeterminato può essere interpretato anche in un ambiente diverso da quello ROS.

Al momento dell'avvio il computer di bordo genera un ponte WiFi per consentire la connessione con un computer portatile che funge da *base station* remota, attraverso il quale l'operatore può gestire il rover, comandare i suoi movimenti, monitorare lo svolgimento dei processi e verificare lo stato dei vari componenti.
L'alimentazione del sistema di locomozione è assicurata da una batteria al Litio ricaricabile ad alta capacità. Una batteria simile, ma di dimensioni minori, alimenta il computer di bordo e le sue periferiche primarie. Il funzionamento del rover, di conseguenza, non richiede connessioni cablate di alcun tipo.

Al momento attuale gli strumenti installati a bordo del rover MORPHEUS sono:

– Un LiDAR 3–D Livox Horizon (descritto nella Sezione 2.1).

– Una termocamera FLIR Vue Pro R (descritta nella Sezione 2.2).

– Una stereocamera binoculare StereoLabs ZED.

– Un ricevitore GNSS (*Global Navigation Satellite System*) della Swift Navigation.

Figura 1-3: Il rover MORPHEUS con la sua *base station* durante una raccolta di dati.

Capitolo 2

Descrizione degli strumenti

2.1 LiDAR Livox Horizon

Il LiDAR 3–D presente a bordo del rover MORPHEUS appartiene alla linea "Horizon" prodotta dalla Livox Technology Company. La Tabella 2-1 riassume le sue specifiche tecniche.

Livox Horizon	
Lunghezza d'onda del laser	905 nm
Distanza di rilevamento massima	90 m @ riflettività 10% 130 m @ riflettività 20% 260 m @ riflettività 80%
Distanza di rilevamento minima	0.5 m
Campo di vista	$81.7° \times 25.1°$
Errore massimo (1σ @ 20 m)	0.02 m (lineare) 0.05° (angolare)
Frequenza di acquisizione	240 000 punti/s
Copertura del campo di vista	60% @ 0.1 s 98% @ 0.5 s
Alimentazione	$10 - 15$ V DC
Potenza assorbita	12 W (30 W all'avvio)
Dimensioni e peso	77×115×84 mm, 1180 g

Table 2-1: Specifiche tecniche del LiDAR.

Questo strumento sostituisce il LiDAR 2–D di tipo industriale utilizzato in precedenza, che non era in grado di produrre *point cloud* tridimensionali e che, per il suo ingombro ed il suo peso, era risultato poco adatto all'installazione su una piattaforma mobile. Al contrario, il Livox Horizon è stato originariamente sviluppato per applicazioni legate alla guida autonoma in campo automobilistico: è quindi più compatto e leggero, ma possiede anche un campo di vista più ampio e una velocità di scansione più elevata.

2.1.1 Caratteristiche e funzionamento

Il termine LiDAR (ovvero *Light Detection And Ranging*) identifica tutti quei dispositivi capaci di eseguire misurazioni di distanza mediante l'emissione e la ricezione di impulsi laser, ma anche le tecniche di telerilevamento (*remote sensing*) che si basano sul loro utilizzo.

Figura 2-2: Livox Horizon (a sinistra) e unità Livox Converter (a destra).

Il funzionamento dei LiDAR si basa sulla possibilità di calcolare la distanza d a cui si trova una superficie misurando il tempo Δt impiegato dalla radiazione elettromagnetica per raggiungerla e ritornare alla sorgente dopo essere stata riflessa. Se l'emissione e la ricezione avvengono nello stesso punto dello spazio, indicando con c la velocità della luce si ha:

$$2\,d = c\,\Delta t \tag{2.1}$$

Nella sua forma più semplice, un dispositivo di questo tipo è in grado di misurare la distanza in un'unica direzione. Per produrre delle *point cloud*, tuttavia, è necessario eseguire la misurazione in più direzioni diverse ed associare ad ognuna di esse il relativo valore di distanza. All'emettitore e al ricevitore di radiazione (costituiti, rispettivamente, da una sorgente laser e da un rilevatore di fotoni a stato solido) deve quindi essere affiancato un adeguato sistema di scansione.

In quasi tutti LiDAR 2–D la radiazione viene direzionata grazie ad uno specchio rotante mosso da un apposito attuatore: la rotazione θ dello specchio identifica la direzione, mentre la misura del tempo di volo Δt fornisce la distanza. Si produce così una *point cloud* bidimensionale, i cui elementi (d, θ) si trovano all'intersezione tra il piano di scansione (che è perpendicolare all'asse di rotazione dello specchio) e l'ambiente.
Nei LiDAR 3–D si utilizzano invece sistemi di scansione più sofisticati, costituiti da vari specchi mobili (in rotazione su assi diversi) oppure da sorgenti di radiazione multiple che colpiscono un solo specchio da varie direzioni. Spesso questi sistemi utilizzano elementi microelettromeccanici (MEMS) per minimizzare il numero di parti in movimento, in modo da aumentare l'affidabilità e diminuire gli ingombri. Le *point cloud* prodotte sono quindi tridimensionali, poiché la distanza può essere misurata in qualsiasi direzione all'interno del campo di vista. In genere le coordinate sferiche (d, θ, φ) dei punti acquisiti sono convertite in coordinate cartesiane (X, Y, Z) espresse in un sistema di riferimento dall'orientamento facilmente identificabile.

La qualità dei dati raccolti da un LiDAR 2–D o 3–D non dipende solo dalle prestazioni del suo sistema di scansione, ma anche da quelle dei rilevatori e dell'elettronica di gestione:

- Per valutare il tempo di volo di un singolo impulso laser è sufficiente misurare le variazioni nella quantità di radiazione che raggiunge lo strumento.
- Misurare l'effettiva intensità della radiazione incidente, confrontandola poi con quella della radiazione emessa, permette di associare ad ogni punto una stima di riflettività.
- Valutare la correlazione di fase tra gli impulsi emessi e quelli ricevuti consente di raffinare la misurazione di distanza. In questo modo è anche possibile eseguire più misurazioni allo stesso tempo, distinguendo i segnali di ritorno dei singoli emettitori.

L'elevata velocità di acquisizione che caratterizza il Livox Horizon è dovuta proprio all'impiego contemporaneo di emettitori e ricevitori multipli. La possibilità di controllare con precisione la direzione di emissione della radiazione permette di adottare un particolare schema di scansione non ripetitivo, che raggiunge rapidamente un'elevata copertura del campo di vista. Il dispositivo può anche essere impostato per raddoppiare il numero di impulsi di ritorno: in questa modalità *dual return* la frequenza di acquisizione può raggiungere i 480 000 punti al secondo.

All'interno del Livox Horizon è presente anche una piattaforma inerziale (*Inertial Measurement Unit*, o IMU) integrata, in grado di fornire delle misurazioni di accelerazione lineare e angolare con una frequenza che può raggiungere i 200 Hz.

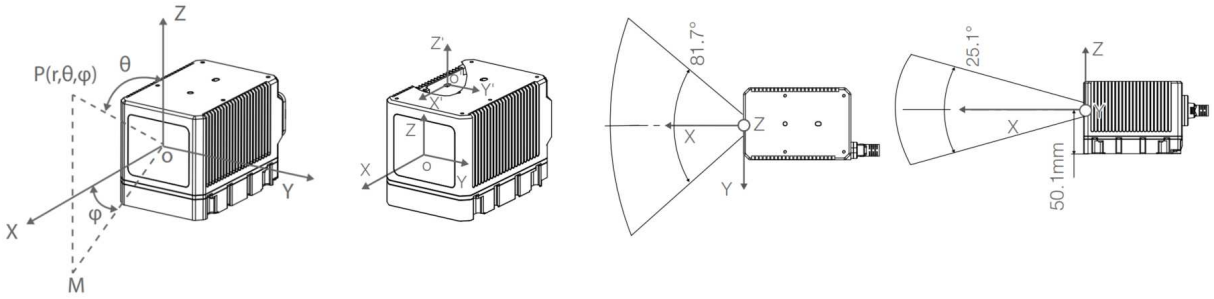

Figura 2-3: Sistemi di riferimento del LiDAR e dalla piattaforma inerziale integrata.

Nella Figura 2-3 sono evidenziati il sistema di riferimento $(\mathrm{O} - XYZ)$ del LiDAR e il sistema di riferimento $(\mathrm{O}' - X'Y'Z')$ della piattaforma inerziale. La conversione tra coordinate sferiche e coordinate cartesiane avviene ponendo

$$\begin{aligned} X &= d\,\sin(\theta)\,\cos(\varphi) \\ Y &= d\,\sin(\theta)\,\sin(\varphi) \\ Z &= d\,\cos(\varphi) \end{aligned} \tag{2.2}$$

compatibilmente con i limiti posti dall'estensione del campo di vista. L'origine O' del sistema di riferimento della piattaforma inerziale ha coordinate $(-0.05512$ m, -0.02226 m, 0.02970 m) nel sistema di riferimento del LiDAR.

Come si può notare dalla Figura 2-2, la finestra ottica attraverso cui viene emessa e ricevuta la radiazione occupa la maggior parte della superficie frontale del LiDAR. La superficie inferiore è invece occupata dal modulo di dissipazione termica, che mantiene i componenti dello strumento alla loro temperatura operativa ottimale.
Un connettore nella parte posteriore permette di collegare il LiDAR all'unità Livox Converter che gli fornisce potenza elettrica. In un primo momento lo strumento è stato alimentato da una singola batteria agli ioni di Litio da 14.8 V e 1 300 mAh, sufficiente a garantire circa 40 minuti di funzionamento continuato. In seguito è stato utilizzato un accumulatore da 12 V e 20 400 mAh già presente a bordo del rover. L'unità Livox Converter consente anche la trasmissione dei dati e dei comandi, che avviene con protocollo IP attraverso un cavo Ethernet RJ45.

Il Livox Horizon può essere controllato in due modi:

- Usando un software dedicato (*Livox Viewer* per Windows) per visualizzare in tempo reale le *point cloud* acquisite e settare i parametri di funzionamento del dispositivo.
- Usando un apposito pacchetto `livox_ros_driver` per ROS, che può essere installato sul computer di bordo del rover e successivamente gestito attraverso il computer portatile che funge da *base station* remota.

2.2 Termocamera FLIR Vue Pro R

La termocamera con capacità radiometriche presente sul rover MORPHEUS insieme al LiDAR 3–D appartiene alla linea "Vue Pro R" prodotta dalla FLIR Systems. La Tabella 2-4 riassume le sue specifiche tecniche.

FLIR Vue Pro R 336	
Tecnologia del rilevatore	microbolometri VOx non raffreddati
Dimensioni del rilevatore	336×256 pixel
Regione spettrale	$7.5 - 13.5$ µm (LWIR)
Campo di vista	$35° \times 27°$ (con ottica da 9 mm)
Temperatura di esercizio	da $-20°$ a $+50°$
Alimentazione	$4.8 - 6.0$ V DC
Potenza assorbita	2.1 W (3.9 W al picco)
Dimensioni e peso	57.4×44.45×44.45 mm, 114 g

Table 2-4: Specifiche tecniche della termocamera.

Lo strumento era già stato acquistato quando si è deciso di utilizzarlo per realizzare il processo di mappatura termica, quindi non è stato selezionato specificamente per un'applicazione di questo tipo. Inoltre, differentemente da quanto è avvenuto nel caso del LiDAR, non deve sostituire uno dei sensori già presenti a bordo del rover.

Figura 2-5: Termocamera FLIR Vue Pro R con ottica da 9 mm.

2.2.1 Caratteristiche e funzionamento

Una termocamera è un dispositivo in grado di convertire la radiazione infrarossa in un'immagine termica, che evidenzia le diverse modalità con cui la materia presente nell'area osservata emette o assorbe potenza radiante in quella particolare regione dello spettro elettromagnetico.
Nelle termocamere con capacità radiometriche ad ogni immagine termica è associata anche una mappa di temperatura, creata convertendo i valori di intensità dei singoli pixel con un apposito software. Quelle fornite dagli algoritmi di radiometria sono però temperature apparenti, poiché alcuni dei parametri coinvolti nel calcolo devono essere già noti a priori.

Le FLIR Vue Pro R sono termocamere progettate per essere installate a bordo di velivoli senza pilota (*Unmanned Aerial Vehicle*, o UAV) di piccole e medie dimensioni. Sono quindi utilizzate più per la ricognizione aerea che per la navigazione robotica o per la guida autonoma. Risultano comunque perfettamente adatte all'installazione su un rover autonomo, per via delle dimensioni ridotte e del peso contenuto.

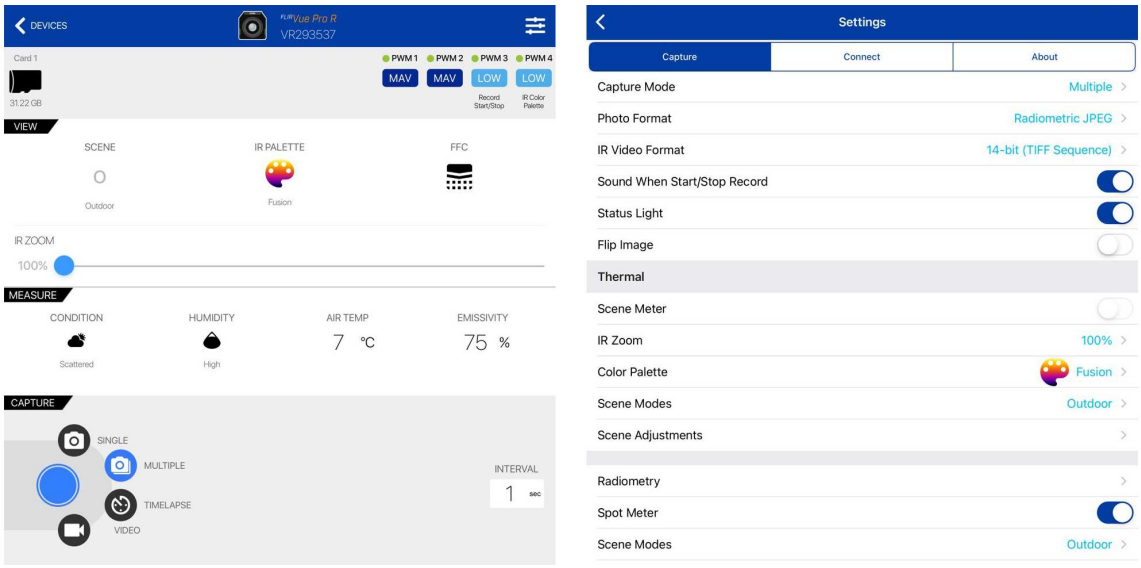

Figura 2-6: Schermate dell'applicazione per la gestione della termocamera.

Il funzionamento di queste termocamere, inoltre, è sostanzialmente indipendente da quello della piattaforma che le trasporta. La potenza elettrica viene fornita da un comune cavo USB, mentre i dati acquisiti sono immagazzinati su una scheda di memoria rimovibile.
Un'apposita applicazione per dispositivi mobili permette di controllare lo strumento a distanza e gestirne le impostazioni grazie ad un collegamento Bluetooth. Oltre ad avviare o interrompere l'acquisizione dei dati, cosa che può avvenire anche utilizzando un pulsante sul lato destro della termocamera, questa applicazione consente di impostare i seguenti parametri:

- Caratteristiche della scena o dell'ambiente.
- Modalità e frequenza con cui vengono acquisiti i dati.
- Formato dei dati acquisiti.
- Gamma di colori per la visualizzazione delle immagini termiche.
- Livello di illuminazione naturale.
- Livello di umidità relativa.
- Temperatura dell'aria.
- Emissività stimata degli oggetti osservati.
- Distanza stimata degli oggetti osservati.

L'algoritmo di radiometria contenuto nel software della termocamera tiene conto dei parametri impostati dall'utilizzatore per calcolare i coefficienti di conversione che consentono di ricavare i valori di temperatura apparente associati ai diversi punti dell'immagine. Con questa particolare termocamera, tuttavia, l'impostazione dei parametri precedentemente elencati può avvenire una sola volta per immagine. I valori dell'emissività e della distanza di osservazione dovranno quindi rappresentare delle condizioni di riferimento valide per la maggior parte della scena, altrimenti la stima delle temperature potrebbe non avvenire in modo coerente.
I coefficienti radiometrici determinati dall'algoritmo vengono inseriti tra i metadati associati ad ognuno dei file prodotti dalla termocamera, ma non è possibile accedervi direttamente. L'unico modo per elaborare le immagini termiche è quello di utilizzare il software *FLIR Tools*, oppure il pacchetto *FLIR Atlas SDK* in ambiente MATLAB.

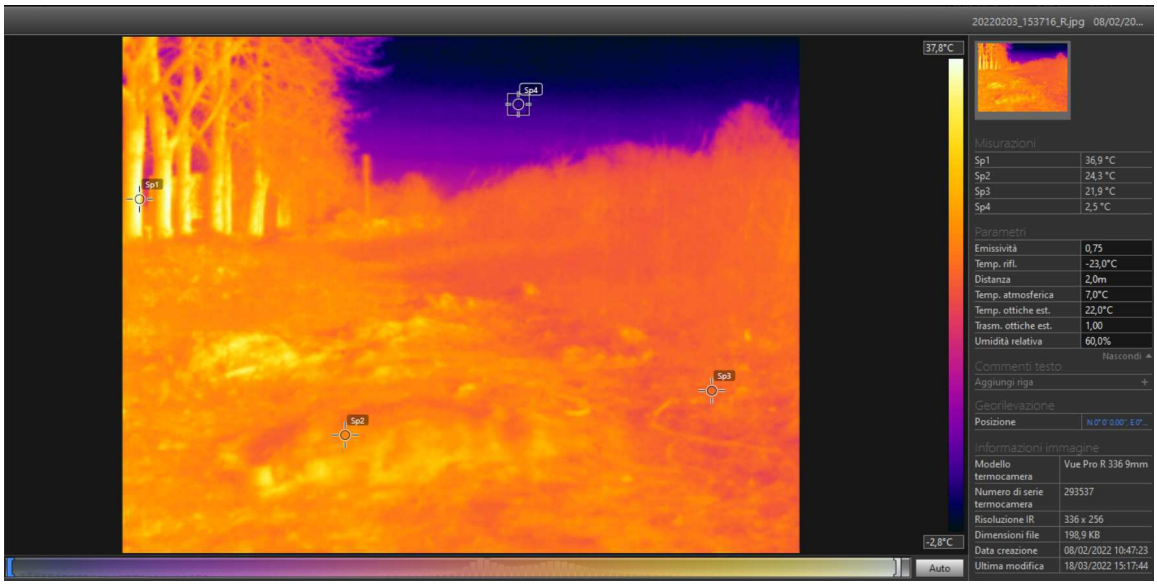

Figura 2-7: Immagine termica radiometrica da una termocamera FLIR Vue Pro R.

2.2.2 Modello termico

Le tecniche di telerilevamento che utilizzano le immagini termiche per studiare le caratteristiche di un oggetto o di un ambiente rientrano nel campo della termografia a distanza. Queste tecniche riguardano una parte dello spettro elettromagnetico che è strettamente correlata con i fenomeni in grado di influenzare lo stato termico della materia.

La principale fonte della radiazione rilevata da una termocamera è rappresentata dall'emissione termica prodotta dalla materia circostante, la cui distribuzione spettrale varia in funzione della temperatura della sorgente. Al di sotto dei 400 K (126.85°C) l'emissione termica è concentrata nell'intervallo di lunghezze d'onda compreso tra 7 e 14 µm, cioè nella cosiddetta regione spettrale dell'infrarosso ad onde lunghe (*Long Wave InfraRed*, o LWIR).

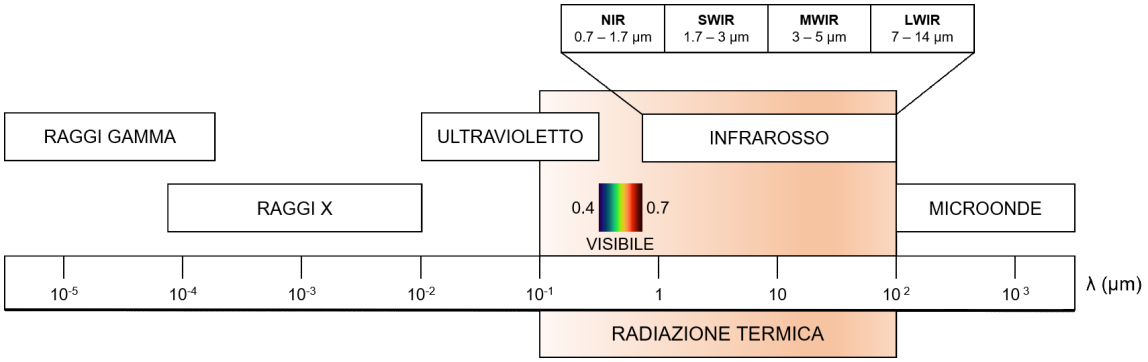

Figura 2-8: Spettro elettromagnetico e regioni spettrali della banda infrarossa.

Per rilevare questa radiazione, la FLIR Vue Pro R sfrutta dei trasduttori di tipo termico: nello specifico, una matrice di microbolometri in Ossido di Vanadio (VOx) non raffreddati.
In questi trasduttori l'assorbimento della potenza radiante incidente modifica la temperatura di un elemento attivo, cambiando la sua resistenza elettrica complessiva. Le variazioni nell'intensità della radiazione che colpisce il trasduttore sono quindi convertite, sia pure indirettamente, in un segnale elettrico che il software dello strumento può analizzare ed interpretare. Un rilevatore di questo tipo non ha un comportamento selettivo, per cui dei filtri a trasmissione sono impiegati per escludere le lunghezze d'onda al di fuori dell'intervallo di sensibilità previsto.

Nei moderni microbolometri l'elemento attivo consiste in una sottile lamina piana di materiale semiconduttore, predisposta in modo da massimizzare l'assorbimento della radiazione incidente e minimizzare tutte le altre forme di scambio termico. Una coppia di elettrodi altrettanto sottili sostiene questa lamina, isolandola termicamente ma collegandola elettricamente al substrato che ospita il circuito integrato di lettura.
Una capacità termica contenuta ed un coefficiente di scambio termico conduttivo molto ridotto permettono al microbolometro di ottenere la massima variazione di resistenza elettrica per una data variazione di temperatura, a parità di potenza radiante incidente.

In generale i rilevatori termici tendono ad essere più lenti nella risposta rispetto ai rilevatori di fotoni, cioè quei trasduttori in grado di convertire direttamente la radiazione elettromagnetica in un segnale elettrico. A differenza di questi ultimi, tuttavia, i rilevatori termici non richiedono un raffreddamento attivo e continuo per mantenere la propria temperatura operativa: i pesi, gli ingombri ed i costi del sistema sono quindi molto più contenuti.

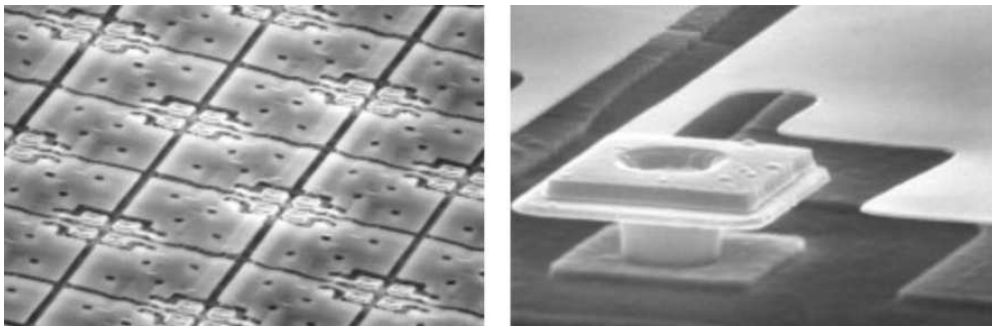

Figura 2-9: Matrice di microbolometri (a sinistra) e dettaglio degli elettrodi (a destra).

Per formare un'immagine termica radiometrica non è sufficiente misurare e registrare l'intensità della radiazione che colpisce ogni elemento del rilevatore. Le superfici presenti nell'area osservata infatti non si limitano ad emettere radiazione termica, poiché assorbono e riemettono una parte della radiazione proveniente dall'ambiente (quella che non viene riflessa o diffusa). La radiazione subisce anche una certa attenuazione da parte dell'atmosfera, che oltretutto contribuisce con la propria emissione termica diretta sia verso la scena che verso il rilevatore.

Il calcolo della temperatura apparente avviene prendendo in considerazione i vari elementi della catena radiometrica, che in generale inizia con l'emissione di una certa quantità di radiazione q_e da parte di un corpo e finisce con l'assorbimento di una certa quantità di radiazione q_s da parte di un rilevatore. Si può considerare la potenza radiante q_s raccolta dalla termocamera come una combinazione di due contributi: la componente q_o che proviene dalla materia presente nell'area osservata e la componente q_a che proviene dall'atmosfera. Si ha quindi:

$$q_s = q_o + q_a \tag{2.3}$$

Nell'assunzione di un mezzo scarsamente diffusivo (riflettività $\rho_a = 0$) risulta

$$q_o = \tau_a \left(q_e + q_r\right) \tag{2.4}$$

dove la trasmissività τ_a viene fornita da un opportuno modello dell'atmosfera, tenendo conto di temperatura, umidità relativa e distanza. In generale il vapore acqueo è il maggior responsabile dei fenomeni di attenuazione atmosferica nella regione spettrale LWIR. Risulta inoltre

$$q_a = \varepsilon_a \, \Phi_a^{BB}\left(T_a\right) = \left(1 - \tau_a\right) \, \Phi_a^{BB}\left(T_a\right) \tag{2.5}$$

dove $\Phi_a^{BB}\left(T_a\right)$ rappresenta la modalità di emissione caratteristica di un corpo nero che si trova alla stessa temperatura T_a dell'atmosfera.

Assumendo che tutte le superfici considerate siano grigie (emissività ε_o costante al variare della lunghezza d'onda) e opache (trasmissività τ_o nulla) la potenza radiante effettivamente dovuta all'emissione termica di un corpo alla temperatura T_o è data da:

$$q_e = \varepsilon_o \, \Phi_o^{BB} \left(T_o \right) \tag{2.6}$$

La potenza radiante che la scena riflette verso lo strumento invece è data da:

$$q_r = \rho_o \, \Phi_r \left(T_r \right) = \left(1 - \varepsilon_o \right) \, \Phi_r \left(T_r \right) \tag{2.7}$$

dove $\Phi_r \left(T_r \right)$ è funzione dalla temperatura riflessa apparente dello sfondo. In questo contesto lo sfondo è costituito da tutto ciò che non rientra nel campo di vista della termocamera, quindi la temperatura T_r viene stimata valutando l'intensità complessiva della radiazione incidente. Nella maggior parte dei casi il principale contributo a Φ_r è costituito dalla radiazione solare.
Inserendo nella (2.3) tutte le relazioni successive alla fine si ottiene

$$q_s = \tau_a \left[\varepsilon_o \, \Phi_o^{BB} \left(T_o \right) + \left(1 - \varepsilon_o \right) \, \Phi_r \left(T_r \right) \right] + \left(1 - \tau_a \right) \, \Phi_a^{BB} \left(T_a \right) \tag{2.8}$$

dove q_s è correlata all'intensità del pixel considerato e la funzione $\Phi_o^{BB} \left(T_o \right)$ introduce nel calcolo l'incognita T_o da determinare. Le funzioni $\Phi_o^{BB} \left(T_o \right)$ e $\Phi_a^{BB} \left(T_a \right)$ vengono continuamente corrette tramite un processo di ricalibrazione che la termocamera esegue in modo automatico. Lo stesso vale per l'algoritmo che calcola la temperatura apparente dello sfondo. L'emissività ε_0 dei corpi osservati, così come la loro distanza, deve invece essere stimata a priori.

2.2.3 Modello ottico

L'impiego di un rilevatore costituito da una matrice di microbolometri implica che le immagini termiche vengano prodotte acquisendo tutti i pixel che le compongono praticamente nello stesso momento. Si può quindi ricorrere al modello di *pinhole camera* per descrivere il comportamento della termocamera dal punto di vista ottico.

Inizialmente si può considerare un sistema ottico formato da un'unica lente biconvessa, con uno spessore molto minore rispetto ai raggi di curvatura delle due superfici. In un sistema di questo tipo, tutti i raggi ottici paralleli all'asse della lente vengono deviati in modo da incontrare l'asse stesso ad una distanza f dal centro della lente (detta distanza focale). I raggi ottici passanti per il centro della lente, invece, non subiscono deviazioni. Di conseguenza vale la relazione

$$\frac{1}{Z} + \frac{1}{z} = \frac{1}{f} \tag{2.9}$$

dove Z e z sono, rispettivamente le distanze di un oggetto e della sua immagine dal centro della lente lungo il suo asse. Se il diametro D di questa lente sottile tende a zero, gli unici raggi ottici in grado di attraversarla sono quelli che passano per il suo centro. In queste ipotesi, il processo di formazione delle immagini può essere descritto come la proiezione di alcuni punti 3–D su un piano, attraverso un centro Ω di proiezione: si definisce così il modello di proiezione prospettica per rilevatori a matrice sul piano focale (*Focal Plane Array*, o FPA).

Sia quindi $(\Omega - xyz)$ il sistema di riferimento per la termocamera, tale che:

- L'origine Ω coincide con il centro del campo di vista.
- L'asse z è diretto parallelamente all'asse del sistema ottico.
- L'asse y è diretto verticalmente verso il basso.
- L'asse x completa la terna levogira.

Sia $(\mathrm{C}-uv)$ il sistema di riferimento dell'immagine nel piano focale $z=f$, tale che:

- L'origine C è individuata dall'intersezione tra l'asse z ed il piano focale.
- Gli assi u e v sono, rispettivamente, paralleli agli assi x e y già definiti.

Incorporando nel modello precedente anche la discretizzazione introdotta dalla matrice di pixel del rilevatore, la trasformazione tra il sistema di riferimento $(\Omega-xyz)$ e il sistema di riferimento $(\mathrm{C}-uv)$ è descritta dalle coppia di equazioni:

$$\begin{aligned} u &= f_u\,(x/z) + u_0 \\ v &= f_v\,(y/z) + v_0 \end{aligned} \tag{2.10}$$

Più in generale, indicando con $\mathbf{u}=\{u,\,v\}^{\mathsf{T}}$ la proiezione sul piano immagine di un determinato punto $\mathbf{x}=\{x,\,y,\,z\}^{\mathbf{T}}$ tridimensionale, dalle (2.10) si ottiene

$$\alpha\,\mathbf{u}^{\mathsf{H}} = \alpha \begin{Bmatrix} \mathbf{u} \\ 1 \end{Bmatrix} = \mathbf{A}\,\mathbf{P}_0 \begin{Bmatrix} \mathbf{x} \\ 1 \end{Bmatrix} = \mathbf{A}\,\mathbf{P}_0\,\mathbf{x}^{\mathsf{H}} \tag{2.11}$$

dove la matrice di proiezione prospettica standard $\mathbf{P}_0=[\mathbf{I}_3\,|\,\mathbf{0}]$ è tale che:

$$\alpha\,\mathbf{u}^{\mathsf{H}} = \mathbf{A}\,\mathbf{P}_0\,\mathbf{x}^{\mathsf{H}} = \mathbf{A}\,\mathbf{x} \tag{2.12}$$

La matrice dei parametri intrinseci $\mathbf{A}$ che compare nella (2.11) invece è definita come

$$\mathbf{A} = \begin{bmatrix} f_u & s_\theta & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

dove f_u e f_v sono le distanze focali (in pixel) nelle direzioni dei due assi, u_0 e v_0 individuano la posizione del punto C rispetto all'angolo superiore sinistro della matrice del rilevatore e s_θ è un coefficiente (*skew factor*) che esprime lo scostamento degli assi dalla perpendicolarità.

La posizione dei punti tridimensionali, tuttavia, non viene necessariamente espressa nel sistema di riferimento della termocamera. In particolare, le coordinate degli elementi di una *point cloud* sono sempre espresse nel sistema di riferimento dello strumento che li acquisisce.
Si deve quindi definire una trasformazione dal sistema di riferimento $(\mathrm{O}-XYZ)$ del LiDAR al sistema di riferimento $(\Omega-xyz)$ della termocamera, data da

$$\mathbf{x} = \mathbf{R}\,\mathbf{w} + \mathbf{t} = [\mathbf{R}\,|\,\mathbf{t}] \begin{Bmatrix} \mathbf{w} \\ 1 \end{Bmatrix} = [\mathbf{R}\,|\,\mathbf{t}]\ \mathbf{w}^{\mathsf{H}} \tag{2.14}$$

dove $\mathbf{w}=\{X,\,Y,\,Z\}^{\mathsf{T}}$ è un generico punto dello spazio, mentre $\mathbf{R}$ e $\mathbf{t}$ sono rispettivamente una matrice di rotazione e un vettore di traslazione. Questa rototraslazione in tre dimensioni indica la posizione e l'orientamento della termocamera nel sistema di riferimento del LiDAR.

Alla fine, sostituendo l'espressione di $\mathbf{x}$ dalla (2.14) nella (2.12) si ottiene:

$$\alpha\,\mathbf{u}^{\mathsf{H}} = \mathbf{A}\,[\mathbf{R}\,|\,\mathbf{t}]\ \mathbf{w}^{\mathsf{H}} = \mathbf{P}\,\mathbf{w}^{\mathsf{H}} \tag{2.15}$$

La matrice di proiezione prospettica $\mathbf{P}$ incorpora sia la matrice dei parametri intrinseci $\mathbf{A}$ della termocamera che i parametri estrinseci $\mathbf{R}$ e $\mathbf{t}$ del sistema termocamera-LiDAR.

Nella realtà una termocamera come la FLIR Vue Pro R possiede un sistema ottico costituito da varie lenti di forme e dimensioni diverse, il cui comportamento è molto più complicato di quello descritto dal modello di *pinhole camera* e dalle relazioni per la proiezione prospettica.

I disturbi indotti da questo sistema ottico (coma, curvatura di campo, astigmatismo) in genere non vengono presi in considerazione, poiché si assume che siano stati resi trascurabili attraverso un'accurata progettazione dei vari componenti. Le distorsioni radiali e tangenziali, provocate dal comportamento non ideale delle lenti, non possono invece essere trascurate. Si dovranno quindi modificare i modelli inizialmente adottati.

Gli effetti di queste distorsioni possono essere rappresentati come alterazioni nella forma di una griglia piana. Sono state ricavate alcune relazioni non lineari che legano le coordinate u e v dei punti della griglia (non affette dalle distorsioni) con le coordinate $\breve{u}$ e $\breve{v}$ dei corrispondenti punti della relativa immagine (affette dalle distorsioni). Essendo $r^2 = u^2 + v^2$ si può porre

$$\begin{aligned} \breve{u} &= u\left[1 + k_1\, r^2 + k_2\, r^4\right] + \left[2\, k_3\, u\, v + k_4\left(3\, u^2 + v^2\right)\right] \\ \breve{v} &= v\left[1 + k_1\, r^2 + k_2\, r^4\right] + \left[k_3\left(u^2 + 3\, v^2\right) + 2\, k_4\, u\, v\right] \end{aligned} \tag{2.16}$$

dove i coefficienti k_1 e k_2 sono riferiti alla distorsione radiale, mentre i coefficienti k_3 e k_4 sono riferiti alla distorsione tangenziale.

2.3 Modalità di installazione

Per permettere l'installazione del LiDAR e della termocamera a bordo del rover MORPHEUS è stato progettato e realizzato un apposito supporto.

La scelta di collocare i due strumenti su una struttura rimovibile, invece di fissarli direttamente al telaio del rover, è stata dettata da alcune considerazioni di ordine pratico:

- È necessario che il LiDAR e la termocamera abbiano sempre la stessa posizione e lo stesso orientamento l'uno rispetto all'altra, poiché qualsiasi alterazione della loro configurazione modificherebbe anche i parametri estrinseci $\mathbf{R}$ e $\mathbf{t}$ del sistema.
- È necessario che entrambi gli strumenti possano essere installati o rimossi con facilità, per collocarli in un'altra parte della piattaforma o per utilizzarli indipendentemente dal rover (come è avvenuto durante il processo di calibrazione).

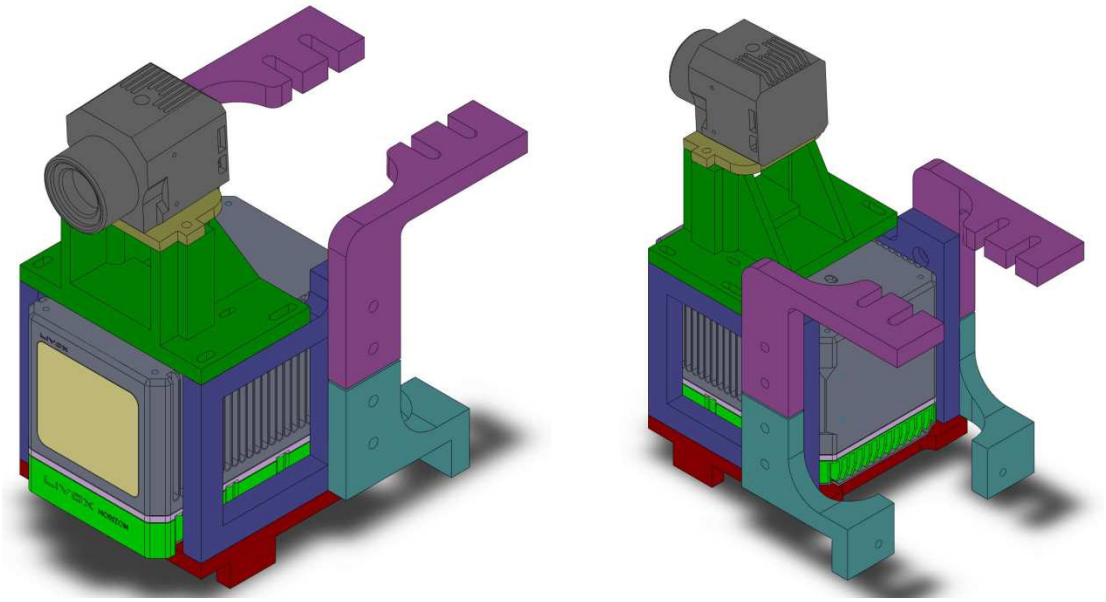

Figura 2-10: Modello in SOLIDWORKS del supporto per gli strumenti.

Il supporto è costituito da dieci elementi separati, realizzati mediante stampa 3–D in materiale plastico. Come si può notare dalla Figura 2-10, ognuno di essi si sviluppa in un'unica direzione a partire da una superficie completamente piatta e non presenta parti aggettanti. Nessun elemento supera le dimensioni di 100×105 mm imposte dai limiti della piattaforma di stampa.

La parte centrale del supporto forma una struttura a scatola che avvolge il LiDAR. Una serie di viti M4, inserite in fori appositamente dimensionati, uniscono le due pareti alla sezione di base e al sostegno della termocamera. Per conferire maggiore solidità al tutto, l'assemblaggio è stato rinforzato con l'applicazione di un adesivo epossidico. Il LiDAR è tenuto in posizione da alcune viti M3 inserite nei fori di montaggio posti nella parte inferiore e superiore.
Il sostegno della termocamera, che si trova al di sopra del LiDAR, è formato da una base piatta collegata a due strutture allungate, con le estremità inclinate di 3° rispetto all'orizzontale. Una basetta sagomata, fissata alla termocamera da due viti M2, permette di collocare lo strumento in posizione. Come evidenzia la Figura 2-11, l'inclinazione della termocamera rispetto al LiDAR determina l'entità della sovrapposizione tra i relativi campi di vista, nonché le distanze dal rover X_g^L e X_g^C della loro intersezione con il suolo.

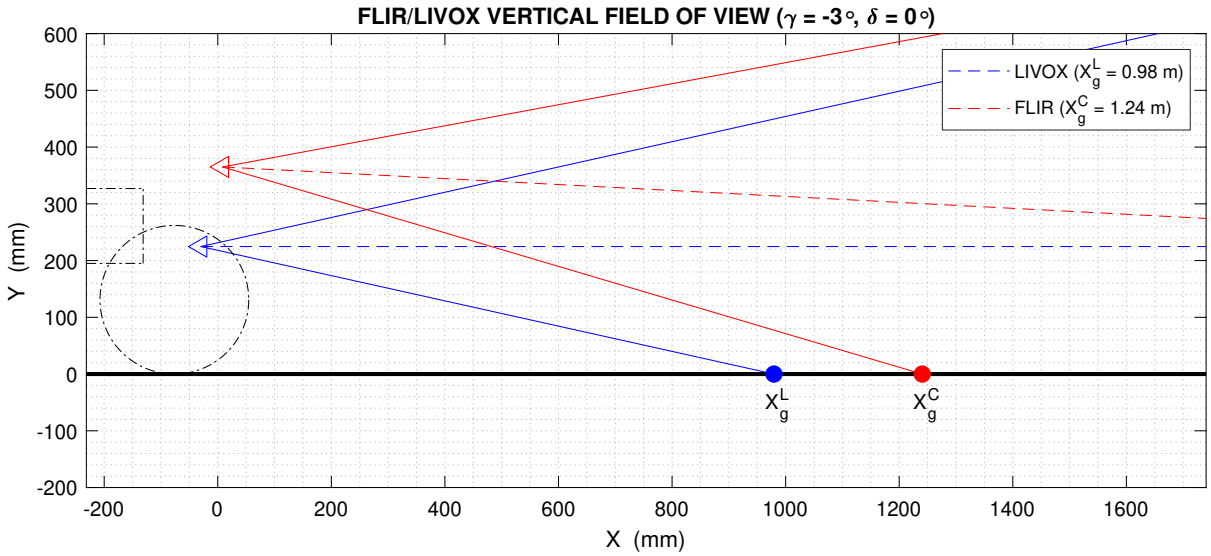

Figura 2-11: Estensione e inclinazione dei campi di vista del LiDAR e della termocamera.

Due elementi ad angolo su ciascun lato della struttura permettono di fissare l'intero supporto al rover. Ognuno di questi elementi è mantenuto in posizione con due viti passanti strette da dadi autobloccanti, per cui può essere sostituito senza bisogno di modificare il resto del supporto: se necessario possono essere usati elementi di fissaggio dimensionati per posizionare il supporto in un altro punto del rover, oppure per dare agli strumenti una particolare inclinazione. Inoltre, le estremità degli elementi di supporto inferiori sono sagomate in modo da essere allo stesso livello delle sporgenze sulla sezione di base, in modo da formare un appoggio stabile quando il supporto viene collocato su una superficie piana.

La Figura 2-12 mostra l'attuale collocazione degli strumenti a bordo del rover MORPHEUS. La sezione frontale del telaio (un profilato in Acciaio a sezione quadrata di 20×20×2 mm) è stretta tra la sezione di base e le estremità degli elementi di sostegno inferiori da una vite passante con due dadi. Gli elementi di fissaggio superiori sono invece avvitati alla piastra in Acciaio che funge da copertura per la sezione anteriore del rover.
Nella Figura 2-13 è visibile anche l'unità Livox Converter che collega il LiDAR con il computer di bordo del rover e con l'accumulatore che alimenta gli strumenti. Lo stesso accumulatore alimenta anche la termocamera, attraverso il relativo cavo USB.

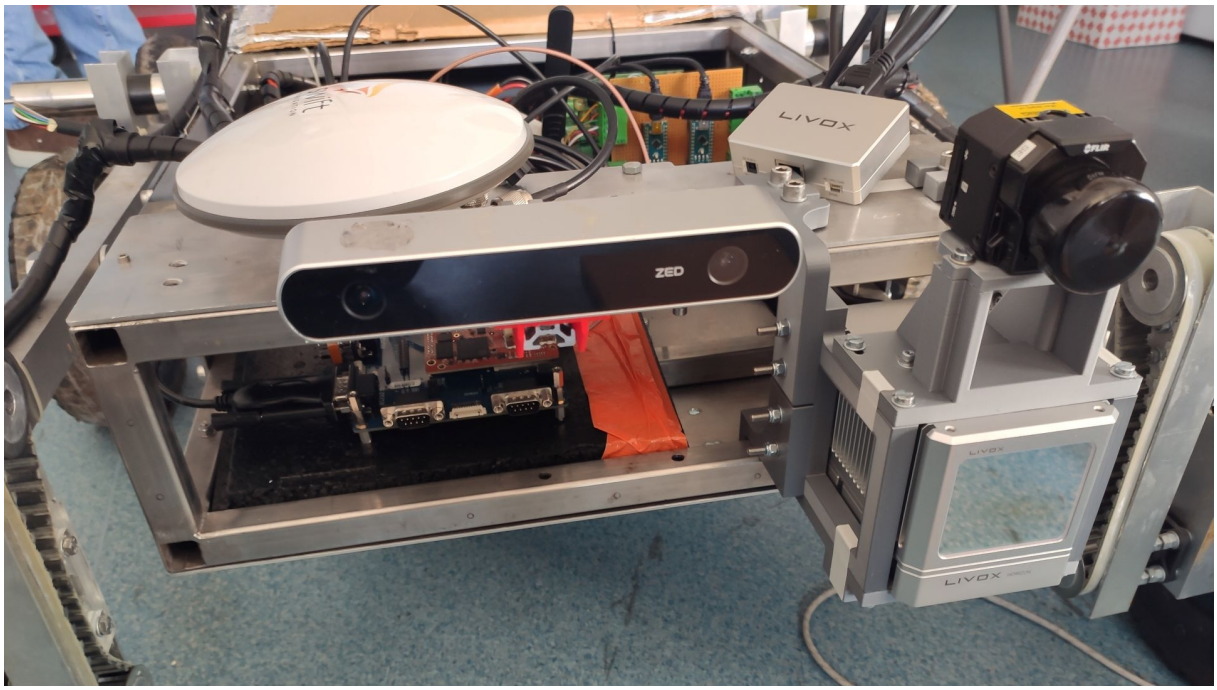

Figura 2-12: Posizionamento degli strumenti a bordo del rover MORPHEUS.

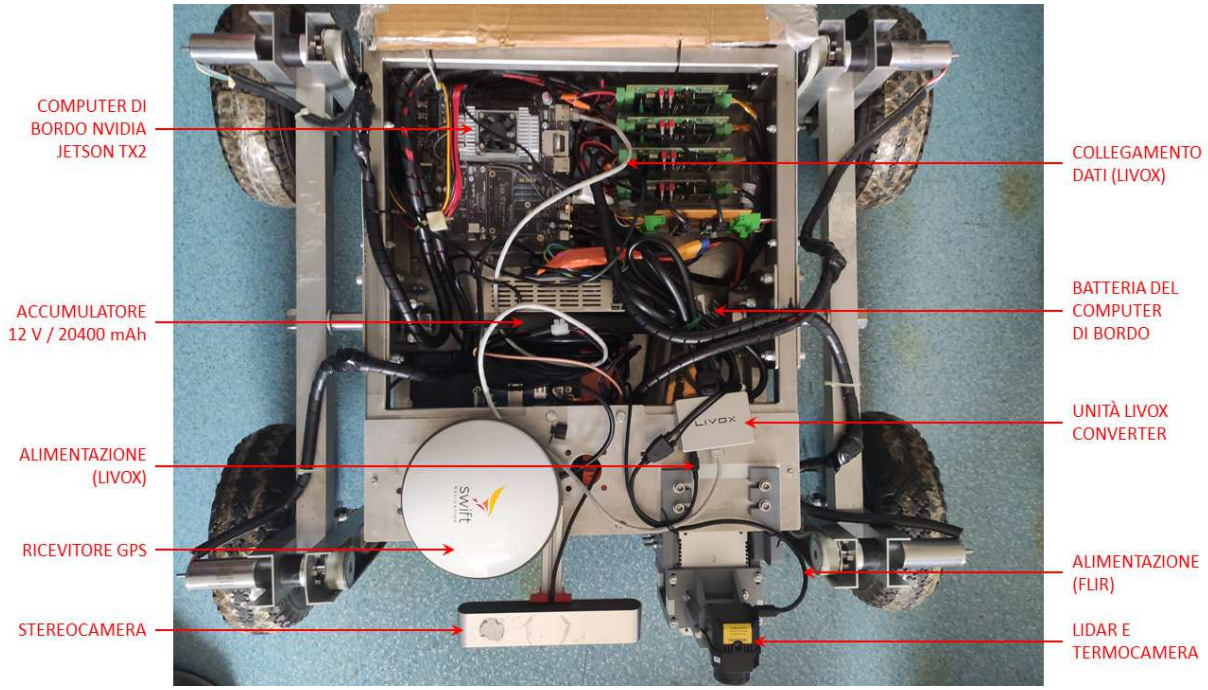

Figura 2-13: Panoramica della strumentazione installata sul rover MORPHEUS

Capitolo 3

Calibrazione degli strumenti

3.1 Calibrazione intrinseca

In questa sezione viene descritta la strategia adottata per eseguire la calibrazione intrinseca della termocamera. Il metodo è quello originariamente proposto da Z. Zhang nel 1998.

La matrice di proiezione prospettica $\mathbf{P} = \mathbf{A}\,[\mathbf{R}\,|\,\mathbf{t}]$ rappresenta il modello analitico che descrive il processo di formazione delle immagini. In particolare, la matrice dei parametri intrinseci $\mathbf{A}$ è quella che definisce la trasformazione dal sistema di riferimento $(\Omega - xyz)$ della termocamera al sistema di riferimento $(\mathrm{C} - uv)$ dell'immagine. Lo scopo della calibrazione è quello di determinare i parametri del modello per lo strumento in questione, ricavando anche i coefficienti necessari a rimuovere dalle immagini le distorsioni introdotte dal suo sistema ottico.
Questo risultato viene raggiunto sfruttando la relazione che lega le coordinate di specifici punti tridimensionali a quelle delle corrispondenti proiezioni bidimensionali. Tali corrispondenze sono definite acquisendo diverse immagini di uno stesso schema di calibrazione, formato da una serie di elementi geometrici regolari di dimensioni note.

Nel caso degli schemi a scacchiera, che sono i più diffusi, i punti di interesse sono rappresentati dai vertici dei riquadri interni. Possono quindi essere individuati utilizzando un rilevatore come quello di Harris-Stephens (che sarà descritto più in dettaglio nella sezione successiva).

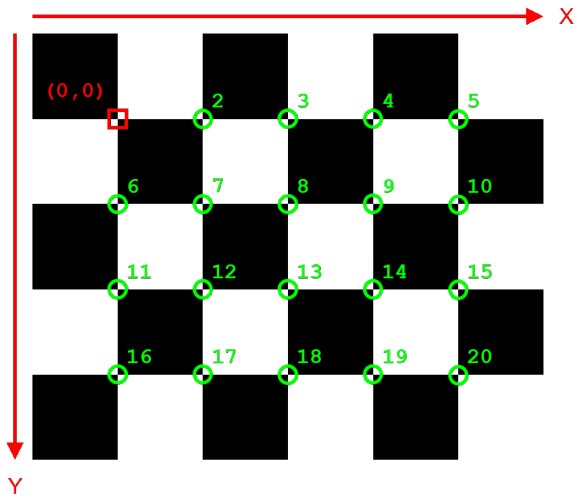

Figura 3-1: Schema di calibrazione a scacchiera e relativo sistema di riferimento.

I punti presenti all'interno di ogni immagine consentono di riconoscere il sistema di riferimento dello schema in quella particolare configurazione. Convenzionalmente la sua origine $(0, 0)$ viene

fissata nell'angolo inferiore destro del riquadro in alto a sinistra, con l'asse X nella direzione del lato più lungo e l'asse Y nella direzione del lato più corto. In questo modo le coordinate 3-D dei singoli punti potranno essere determinate conoscendo le dimensioni della scacchiera.
La collocazione dell'origine e l'orientamento degli assi possono essere stabiliti univocamente solo se lo schema non presenta simmetrie rotazionali, cioè se non riprende lo stesso aspetto dopo una rotazione di 90° o 180° intorno ad un asse perpendicolare alla sua superficie.

Si suppone quindi di avere a disposizione n immagini di uno schema che contiene N vertici. Per ogni immagine dovrebbe esistere una particolare omografia $\mathbf{H}_i$ tale che

$$\alpha \begin{Bmatrix} \mathbf{u}_j \\ 1 \end{Bmatrix} = \mathbf{A} \left[\mathbf{R}_i \,|\, \mathbf{t}_i \right] \begin{Bmatrix} \mathbf{w}_j \\ 1 \end{Bmatrix} = \mathbf{H}_i \begin{Bmatrix} \mathbf{w}_j \\ 1 \end{Bmatrix} \qquad (j = 1, \dots, N) \tag{3.1}$$

indicando con $\mathbf{u}_j = \{u_j,\, v_j\}^\mathsf{T}$ il corrispettivo sul piano immagine del punto $\mathbf{w}_j = \{X_j,\, Y_j,\, Z_j\}^\mathsf{T}$ dello schema. Assumendo (senza perdere generalità) che il bersaglio sia posto in corrispondenza del piano $Z = 0$ e ponendo $\mathbf{m}_j = \{X_j,\, Y_j\}^\mathsf{T}$ si ottiene:

$$\alpha \begin{Bmatrix} \mathbf{u}_j \\ 1 \end{Bmatrix} = \mathbf{A} \left[\mathbf{r}_{i1} \; \mathbf{r}_{i2} \; \mathbf{t}_i \right] \begin{Bmatrix} \mathbf{m}_j \\ 1 \end{Bmatrix} = \left[\mathbf{h}_{i1} \; \mathbf{h}_{i2} \; \mathbf{h}_{i3} \right] \begin{Bmatrix} \mathbf{m}_j \\ 1 \end{Bmatrix} \qquad (j = 1, \dots, N) \tag{3.2}$$

La corrispondenza tra punti tridimensionali e punti bidimensionali però non è perfetta, a causa delle distorsioni presenti nelle immagini e del rumore prodotto dal sensore. Per ricavare una stima dell'omografia associata alla i-esima immagine si deve quindi ricorrere ad un procedimento numerico (*maximum likelihood estimation*). Si cerca allora di minimizzare la funzione

$$f(\mathbf{h}_{i1}, \mathbf{h}_{i2}, \mathbf{h}_{i3}) = \sum_{j=1}^{N} \left\| \mathbf{u}_j - \mathbf{v}_j \right\|^2 = \sum_{j=1}^{N} \left\| \mathbf{u}_j - \frac{1}{\mathbf{h}_{i3}^\mathsf{T} \mathbf{m}_j} \begin{bmatrix} \mathbf{h}_{i1}^\mathsf{T} \mathbf{m}_j \\ \mathbf{h}_{i2}^\mathsf{T} \mathbf{m}_j \end{bmatrix} \right\|^2 \tag{3.3}$$

dove i punti $\mathbf{u}_j$ e $\mathbf{m}_j$ sono solo quelli relativi all'immagine considerata. Si tratta di problema ai minimi quadrati non lineare, risolvibile utilizzando l'algoritmo di Levenberg-Marquardt.

Il legame tra l'omografia stimata $\mathbf{H}_i = [\mathbf{h}_{i1} \; \mathbf{h}_{i2} \; \mathbf{h}_{i3}]$ e la matrice intrinseca $\mathbf{A}$ può essere derivato ricordando che i vettori $\mathbf{r}_{i1}$ e $\mathbf{r}_{i2}$ nella (3.2) rappresentano le prime due colonne di una matrice di rotazione. Devono quindi soddisfare la condizione:

$$\mathbf{r}_{i1}^\mathsf{T} \mathbf{r}_{i2} = \mathbf{r}_{i1}^\mathsf{T} \mathbf{r}_{i1} - \mathbf{r}_{i2}^\mathsf{T} \mathbf{r}_{i2} = 0 \tag{3.4}$$

Poiché la (3.2) implica che $[\mathbf{r}_{i1} \; \mathbf{r}_{i2}] = \mathbf{A}^{-1} [\mathbf{h}_{i1} \; \mathbf{h}_{i2}]$ dalla relazione precedente si ha

$$\mathbf{h}_{i1}^\mathsf{T} \mathbf{B} \, \mathbf{h}_{i2} = \mathbf{h}_{i1}^\mathsf{T} \mathbf{B} \, \mathbf{h}_{i1} - \mathbf{h}_{i2}^\mathsf{T} \mathbf{B} \, \mathbf{h}_{i2} = 0 \tag{3.5}$$

dove $\mathbf{B} = (\mathbf{A}^{-1})^\mathsf{T} = (\mathbf{A}^\mathsf{T})^{-1}$ è una matrice 3×3 simmetrica. Indicando con $\mathbf{b}$ il vettore formato dai 6 elementi di $\mathbf{B}$ (a loro volta funzione dei parametri intrinseci) la (3.5) permette di definire due equazioni lineari nell'incognita $\mathbf{b}$ per ogni omografia $\mathbf{H}_i$ stimata.
Componendo le equazioni relative a tutte le immagini si ottiene un sistema lineare $\mathbf{W}\mathbf{b} = \mathbf{0}$ di $2n$ equazioni in 6 incognite. In generale, per $n \geq 3$ si può determinare $\mathbf{b}$ a meno di un fattore di scala α estraendo l'autovettore di $\mathbf{W}^\mathsf{T} \mathbf{W}$ associato al minimo autovalore.

Conoscendo $\mathbf{b}$ si può ricostruire la matrice $\mathbf{A}$ e ricavare i parametri intrinseci. Il confronto con la stima iniziale dell'omografia fornisce invece i parametri estrinseci $\mathbf{R}_i$ e $\mathbf{t}_i$ per ogni configurazione del bersaglio. Questi risultati, tuttavia, devono essere ulteriormente raffinati. Si procede quindi a minimizzare la somma dei quadrati degli errori di riproiezione

$$\varepsilon_j = \left\| \mathbf{u}_j - \mathbf{u}_j^\star \right\| = \left\| \mathbf{u}_j - \mathbf{u} \left(\mathbf{A},\, \mathbf{R}_i,\, \mathbf{t}_i,\, \mathbf{w}_j \right) \right\| \tag{3.6}$$

dove $\mathbf{u}_j^\star = \mathbf{u}\,(\mathbf{A},\, \mathbf{R}_i,\, \mathbf{t}_i,\, \mathbf{w}_j)$ è dato dalla (3.1) per ogni punto tridimensionale $\mathbf{w}_j$ associato alla i-esima immagine. Questo problema ai minimi quadrati non lineare viene affrontato utilizzando ancora una volta l'algoritmo di Levenberg-Marquardt, sfruttando l'omografia stimata $\mathbf{H}_i$ come valore di avvio da fornire al solutore.
Poiché le matrici $\mathbf{R}_i$ ottenute tramite un procedimento numerico non avranno necessariamente tutte le caratteristiche di una matrice di rotazione, ognuna di esse viene sostituita dalla matrice $\mathbf{S}_i$ con $\mathbf{S}_i^\mathsf{T}\,\mathbf{S}_i = \mathbf{I}$ che minimizza la norma di Frobenius della loro differenza:

$$\|\mathbf{S}_i - \mathbf{R}_i\|_F^2 = \operatorname{tr}\Big((\mathbf{S}_i - \mathbf{R}_i)^\mathsf{T}(\mathbf{S}_i - \mathbf{R}_i)\Big) \tag{3.7}$$

L'effetto delle distorsioni generalmente è piccolo, quindi può essere trascurato quando si ricava la stima iniziale dell'omografia. Deve invece essere tenuto in considerazione quando si determinano i parametri intrinseci ed estrinseci. Il raffinamento dei risultati viene quindi ripetuto, definendo il nuovo errore di riproiezione come

$$\varepsilon_j = \|\mathbf{u}_j - \mathbf{u}\,(\mathbf{A},\, \mathbf{R}_i,\, \mathbf{t}_i,\, k_1,\, k_2,\, k_3,\, k_4,\, \mathbf{w}_j)\| \tag{3.8}$$

dove k_1, k_2 sono i coefficienti del modello che descrive la distorsione radiale, mentre k_3, k_4 sono quelli del modello che descrive la distorsione tangenziale.

3.2 Calibrazione estrinseca

In questa sezione viene descritto il procedimento utilizzato per calibrare la termocamera rispetto al LiDAR. L'operazione permette di ottenere i parametri estrinseci del sistema formato dai due strumenti, rendendo possibile la *sensor fusion* tra *point cloud* e immagini termiche.

Si assume che la termocamera sia già stata calibrata, cioè che i suoi parametri intrinseci (insieme ai coefficienti per la distorsione radiale e tangenziale) siano stati determinati seguendo il metodo descritto nella sezione precedente.
Restano allora da determinare la matrice di rotazione $\mathbf{R}$ ed il vettore di traslazione $\mathbf{t}$ in grado di descrivere la trasformazione dal sistema di riferimento dell'ambiente (che coincide con quello definito dal LiDAR) al sistema di riferimento della termocamera. La calibrazione estrinseca consiste proprio nell'individuare questi parametri confrontando tra loro le *point cloud* e le immagini termiche relative ad uno stesso bersaglio.

La strategia adottata è quella originariamente proposta da Z. Pusztai e L. Hajder nel 2017 per la calibrazione di una telecamera RGB rispetto ad un LiDAR 3–D, opportunamente modificata per poter essere impiegata anche nel caso di una termocamera. Il bersaglio di calibrazione sarà quindi costituito da una semplice scatola, di forma e dimensioni note.

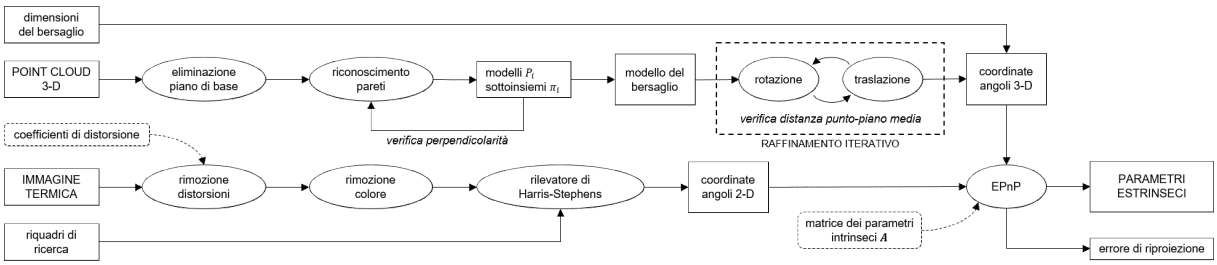

Figura 3-2: Panoramica del processo di calibrazione estrinseca.

I parametri estrinseci si otterranno risolvendo il problema PnP (o *Perspective-n-Point*) che può essere impostato conoscendo le coordinate 3–D e le corrispondenti proiezioni 2–D dei sette angoli visibili della scatola. Le prime possono essere ricavate adattando alla *point cloud* un opportuno

modello del bersaglio, mentre le seconde possono essere estratte dalla relativa immagine termica utilizzando il rilevatore di Harris-Stephens.

3.2.1 Elaborazione della Point Cloud

La prima fase del processo di calibrazione consiste nel riconoscere ed estrarre dalla *point cloud* le superfici piane che rappresentano le pareti del bersaglio.

Ogni angolo della scatola, infatti, è individuato dall'intersezione di tre spigoli rettilinei (uno dei quali può non essere visibile). Ogni spigolo, a sua volta, può essere pensato come l'intersezione tra due piani di estensione indefinita con una certa orientazione nello spazio. Individuare questi piani consente di risalire alla direzione degli spigoli e da questa alla posizione degli angoli.
Si vogliono quindi estrarre tre sottoinsiemi di punti π_i dalla *point cloud* di partenza, ognuno dei quali è associato ad un modello parametrico $P_i = (A_i,\, B_i,\, C_i,\, D_i)$ dove

$$A_i\, X + B_i\, Y + C_i\, Z + D_i = 0 \tag{3.9}$$

è l'equazione del piano nel sistema di riferimento del LiDAR, essendo $\mathbf{n}_i = \{A_i,\, B_i, C_i\}^\mathsf{T}$ il suo vettore normale. Se il piano passa per un certo punto $\mathbf{q}_i = \{q_{i1},\, q_{i2},\, q_{i3}\}^\mathsf{T}$ dello spazio, che non deve appartenere necessariamente al sottoinsieme π_i associato al piano stesso, risulta:

$$D_i = -A_i\, q_{i1} - B_1\, q_{i2} - C_i\, q_{i3} \tag{3.10}$$

Tra i piani estratti dalla *point cloud* non è incluso quello di base, ovvero la superficie che sostiene sia il bersaglio che gli strumenti. I relativi punti sono infatti identificati e rimossi associandoli al modello di piano definito dal versore $\hat{\mathbf{n}}_b = \{0,\, 0,\, 1\}^\mathsf{T}$ perpendicolare alla superficie stessa. Si assume comunque che la *point cloud* includa solo il bersaglio e le sue immediate vicinanze.
Una volta rimosso il piano di base, si cerca il modello parametrico che può essere adattato alla *point cloud* ottenendo il numero di *inlier* più alto. Un dato punto è considerato un *inlier* quando la sua distanza dalla superficie descritta dal modello è minore di una soglia fissata. Questi punti vengono estratti dalla *point cloud* e associati ai parametri del modello che li rappresenta, poi lo stesso procedimento viene ripetuto con i punti rimanenti. Si evita così che lo stesso piano possa essere individuato più volte da gruppi diversi di punti. Dopo aver estratto tre piani distinti tutti i punti rimasti vengono scartati.

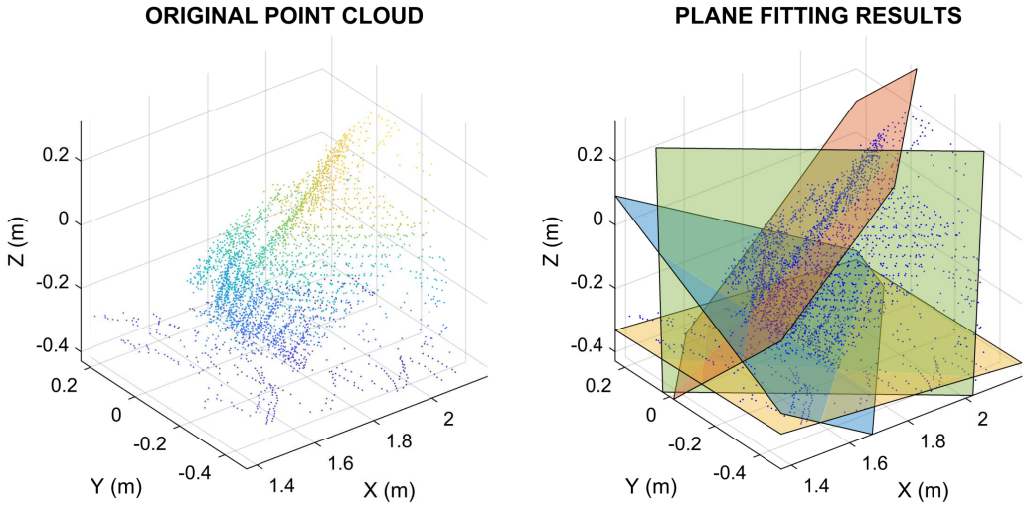

Figura 3-3: Adattamento dei modelli di piano alle pareti del bersaglio.

L'adattamento di un piano ai punti (*plane fitting*) consiste sostanzialmente nella determinazione dei parametri in grado di minimizzare un'opportuna funzione delle distanze tra gli elementi della

point cloud ed il piano stesso. Le principali complicazioni derivano dal fatto che non tutti i punti considerati sono effettivamente riconducibili alla superficie che si vuole modellare: alcuni di essi sono associabili a superfici diverse, mentre altri sono semplicemente affetti da rumore e quindi non rappresentano alcuna superficie reale. La precisione dell'adattamento può essere migliorata specificando il vettore $\hat{\mathbf{n}}_i^\star$ normale al piano che viene cercato.
In questo caso la funzione che effettua il *plane fitting* sfrutta una variante del metodo RANSAC nota come MSAC (o *M-Estimator Sample Consensus*). I risultati ottenuti possono quindi variare leggermente da un'esecuzione all'altra, anche quando i dati iniziali sono gli stessi. I modelli di piano indicati, inoltre, non sono necessariamente perpendicolari tra loro, come invece le pareti della scatola dovrebbero essere. Per questa ragione, i piani individuati vengono scartati se i loro versori normali non soddisfano la condizione

$$|\hat{\mathbf{n}}_1 \cdot \hat{\mathbf{n}}_2| + |\hat{\mathbf{n}}_1 \cdot \hat{\mathbf{n}}_3| + |\hat{\mathbf{n}}_2 \cdot \hat{\mathbf{n}}_3| \leq E_{max} \tag{3.11}$$

dove E_{max} è una soglia stabilita sperimentalmente. Ripetere questa fase del procedimento fino a quando non si ottengono tre piani quasi perpendicolari tra loro garantisce che i dati forniti alle sezioni successive dell'algoritmo siano effettivamente utilizzabili.

I piani appena individuati possono essere associati ad una delle pareti del bersaglio considerando la loro collocazione nello spazio. Sono quindi ordinati in base alla posizione $\mathbf{c}_i = \{X_i^c,\, Y_i^c,\, Z_i^c\}^\mathsf{T}$ del rispettivo centroide, che si ottiene ponendo

$$\mathbf{c}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} \mathbf{p}_j^i \tag{3.12}$$

dove n_i è il numero di punti $\mathbf{p}_j^i = \{X_j^i,\, Y_j^i,\, Z_j^i\}^\mathsf{T}$ nel sottoinsieme π_i corrispondente. In ordine di elevazione Z_i^c crescente si incontrano la parete frontale, quella laterale e quella superiore.

Si è scelto di utilizzare la parete frontale, ovvero quella più vicina sia agli strumenti che al piano di base, come riferimento per descrivere l'orientamento del bersaglio: l'osservazione avviene dal lato "sinistro" se la coordinata Y_i^c della parete laterale è maggiore di quella della parete frontale e dal lato "destro" se accade il contrario. Nel primo caso l'ordinamento convenzionale dei piani è frontale–laterale–superiore, mentre nel secondo è frontale–superiore–laterale. Questa distinzione resta valida indipendentemente dalle dimensioni delle pareti. La Figura 3-4 mostra i due risultati possibili, evidenziando l'ordinamento con la sequenza di colori rosso–verde–blu.

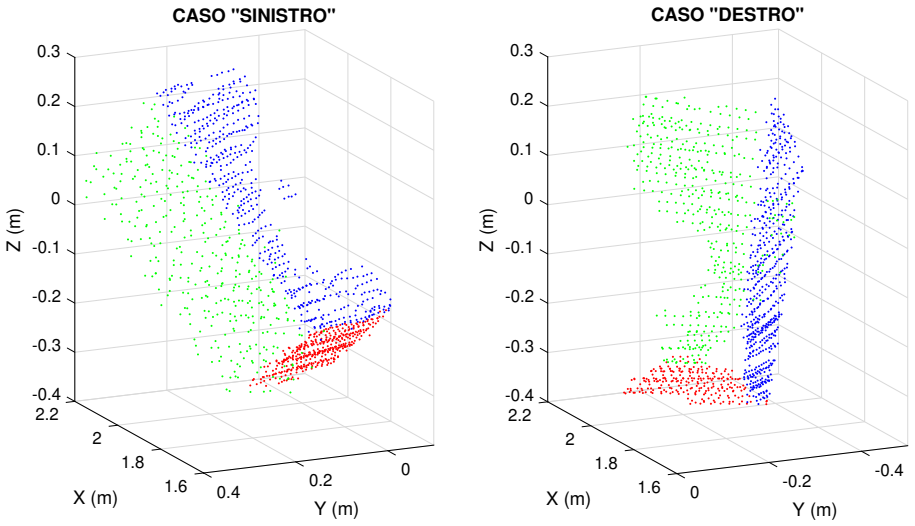

Figura 3-4: Ordinamento dei piani per i due possibili orientamenti del bersaglio.

3.2.2 Modellazione del bersaglio

La seconda fase del processo di calibrazione consiste nell'adattare iterativamente un opportuno modello dell'oggetto osservato ai dati disponibili. In questo modo, conoscendo le reali dimensioni del bersaglio, si potranno ricavare le coordinate 3–D dei suoi angoli visibili.

La creazione del modello di bersaglio inizia definendo il relativo sistema di riferimento, costituito dai tre vettori $\mathbf{e}_i$ e dal punto tridimensionale $\mathbf{q}_1$ ottenuti dall'intersezione dei piani P_i individuati in precedenza. I vettori di bordo $\mathbf{e}_i$ indicano la direzione degli spigoli, che rappresentano le linee di intersezione tra le coppie di piani, quindi sono dati da

$$\begin{aligned} \mathbf{e}_1 &= \hat{\mathbf{n}}_2 \wedge \hat{\mathbf{n}}_3 \\ \mathbf{e}_2 &= \hat{\mathbf{n}}_1 \wedge \hat{\mathbf{n}}_3 \\ \mathbf{e}_3 &= \hat{\mathbf{n}}_1 \wedge \hat{\mathbf{n}}_2 \end{aligned} \tag{3.13}$$

dove $\hat{\mathbf{n}}_i$ sono i versori normali ai piani stessi. Il punto di riferimento $\mathbf{q}_1$ è invece l'unico elemento comune a tutti i piani, per cui imponendo la condizione (3.10) nei tre casi e risolvendo il sistema così ottenuto in forma matriciale si ottiene:

$$\mathbf{q}_1 = - \begin{bmatrix} A_1 & B_1 & C_1 \\ A_2 & B_2 & C_2 \\ A_3 & B_3 & C_3 \end{bmatrix}^{-1} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} \tag{3.14}$$

Si può notare come $\mathbf{q}_1$ coincida con l'angolo del bersaglio più vicino agli strumenti. Ogni vettore di bordo $\mathbf{e}_i$ può allora essere pensato come normale ad un piano passante per questo punto, ma i piani così ottenuti non sarebbero ancora perpendicolari tra loro. Si pone quindi

$$\begin{aligned} \mathbf{m}_1 &= \mathbf{e}_1 \\ \mathbf{m}_2 &= -\left(\mathbf{e}_1 \wedge \mathbf{e}_3\right) \\ \mathbf{m}_3 &= \mathbf{m}_1 \wedge \mathbf{m}_2 \end{aligned} \tag{3.15}$$

assicurandosi che il verso di ogni vettore sia coerente con l'orientamento del bersaglio. I rispettivi versori $\hat{\mathbf{m}}_i$ e il punto $\mathbf{q}_1$ definiscono il modello cercato: i parametri iniziali dovranno però essere raffinati iterativamente per adattarli ai dati forniti.

Ogni iterazione applica al modello tre rotazioni rigide seguite da tre traslazioni rigide, allo scopo di minimizzare la somma dei quadrati delle distanze dei punti di ogni sottoinsieme π_i dal piano corrispondente (cioè quello passante per il punto $\mathbf{q}_1$ e normale al versore $\hat{\mathbf{m}}_i$). Sia quindi

$$d\left(\mathbf{p},\, \mathcal{P}\right) = \mathbf{r}^{\mathsf{T}}\, \hat{\mathbf{n}} = \left(\mathbf{p} - \mathbf{q}\right)^{\mathsf{T}}\, \hat{\mathbf{n}} \tag{3.16}$$

la distanza di un punto $\mathbf{p}$ dal piano $\mathcal{P}$ passante per $\mathbf{q}$ che ha $\hat{\mathbf{n}}$ come versore normale.

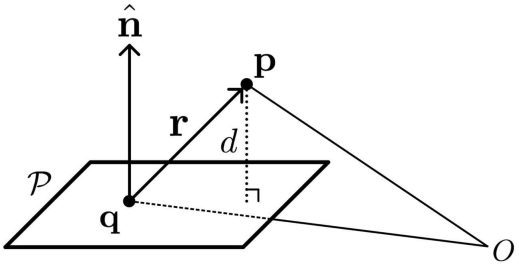

Figura 3-5: Calcolo della distanza tra un punto e un piano nel caso più generale.

Nella fase di rotazione l'intero modello è ruotato, in sequenza, intorno ad ognuno dei versori $\hat{\mathbf{m}}_k$ che ne individuano gli spigoli. Si vuole quindi minimizzare la funzione

$$f_k(\theta) = \sum_{\substack{i=1 \\ i \neq k}}^{3} \sum_{j=1}^{N_i} \left| \left(\mathbf{p}_j^i - \mathbf{q}_1\right)^\mathsf{T} \mathbf{R}_k(\theta)\, \hat{\mathbf{m}}_i \right|^2 \qquad (k = 1, 2, 3) \tag{3.17}$$

dove N_i è il numero di punti $\mathbf{p}_j^i$ nel sottoinsieme π_i e $\mathbf{R}_k(\theta)$ è la matrice di rotazione *axis–angle* per la direzione $\hat{\mathbf{m}}_k$ e l'angolo θ. Si può notare la come rotazione intorno a $\hat{\mathbf{m}}_k$ non modifichi le distanze tra i punti del sottoinsieme π_k ed il k-esimo piano.
Nella fase di traslazione l'intero modello viene invece traslato, in sequenza, lungo la direzione dei versori $\hat{\mathbf{m}}_k$ che ne individuano gli spigoli. La funzione da minimizzare allora diventa

$$g_k(\delta) = \sum_{j=1}^{N_k} \left| \left(\mathbf{p}_j^k - \mathbf{q}_1 - \delta\, \hat{\mathbf{m}}_k\right)^\mathsf{T} \hat{\mathbf{m}}_k \right|^2 \qquad (k = 1, 2, 3) \tag{3.18}$$

poiché, in questo caso, le distanze tra i punti del sottoinsieme π_k ed il k-esimo piano sono le sole ad essere modificate.

La sequenza di rotazioni e traslazioni necessarie al raffinamento del modello si riduce quindi ad una serie di problemi ai minimi quadrati non lineari, che possono essere risolti per via numerica utilizzando l'algoritmo di Levenberg-Marquardt.
Questo algoritmo combina il metodo della discesa del gradiente (*gradient descent*) con il metodo di Gauss-Newton, rispetto al quale risulta più affidabile ma generalmente più lento. In ogni fase il solutore riceve un vettore di distanze punto-piano e restituisce il valore di θ o δ che minimizza la somma dei quadrati degli elementi di quel vettore. Per giungere a convergenza in un numero ragionevole di iterazioni è necessario fornire al solutore un opportuno valore di avvio, indicando anche l'estensione dell'intervallo in cui la soluzione potrà essere cercata.

Una rotazione $\mathbf{R}_k(\theta)$ non modifica la posizione del punto di riferimento $\mathbf{q}_1$ del modello, per cui riguarda solo i versori che identificano gli spigoli. Si sostituisce quindi ogni $\hat{\mathbf{m}}_i$ con:

$$\hat{\mathbf{m}}_i^\star = \frac{\mathbf{R}_k(\theta)\, \hat{\mathbf{m}}_i}{\|\mathbf{R}_k(\theta)\, \hat{\mathbf{m}}_i\|} \qquad (i = 1, 2, 3) \tag{3.19}$$

Al contrario, una traslazione $\delta\hat{\mathbf{m}}_k$ modifica la posizione di $\mathbf{q}_1$ ma lascia inalterato l'orientamento dei vettori che identificano gli spigoli. Il punto di riferimento diventa quindi:

$$\mathbf{q}_1^\star = \mathbf{q_1} + \delta\hat{\mathbf{m}}_k \tag{3.20}$$

Il modello smette di essere aggiornato solo quando la distanza media punto-piano (calcolata su tutti gli elementi di ogni sottoinsieme π_i) scende sotto una soglia fissata.

Una volta terminato il raffinamento iterativo del modello, le coordinate degli altri angoli visibili del bersaglio (il primo angolo è semplicemente $\mathbf{q}_1$) sono date da:

$$\begin{aligned} \mathbf{q}_2 &= \mathbf{q}_1 + l_2\, \hat{\mathbf{m}}_2 & \mathbf{q}_5 &= \mathbf{q}_2 + l_1\, \hat{\mathbf{m}}_1 \\ \mathbf{q}_3 &= \mathbf{q}_1 + l_3\, \hat{\mathbf{m}}_3 & \mathbf{q}_6 &= \mathbf{q}_3 + l_2\, \hat{\mathbf{m}}_2 \\ \mathbf{q}_4 &= \mathbf{q}_1 + l_1\, \hat{\mathbf{m}}_1 & \mathbf{q}_7 &= \mathbf{q}_4 + l_3\, \hat{\mathbf{m}}_3 \end{aligned} \tag{3.21}$$

Le dimensioni del bersaglio sono note, poiché possono essere misurate direttamente, ma il versore che identifica un particolare spigolo varia in base all'orientamento con cui il bersaglio stesso viene osservato. In particolare, se nel caso precedentemente definito "sinistro" risultasse

$$l_1 = d, \qquad l_2 = b, \qquad l_3 = h \tag{3.22}$$

per lo stesso bersaglio visto dal lato "destro" si avrebbe:

$$l_1 = d, \qquad l_2 = h, \qquad l_3 = b \tag{3.23}$$

Le lunghezze misurate d, b e h sono quindi inserite nell'algoritmo supponendo di trovarsi sempre nel primo caso, per poi essere opportunamente associate ai diversi spigoli una volta determinato l'effettivo orientamento del bersaglio. La Figura 3-6 mostra l'ordinamento dei piani, dei punti e degli spigoli nel modello finale del bersaglio.

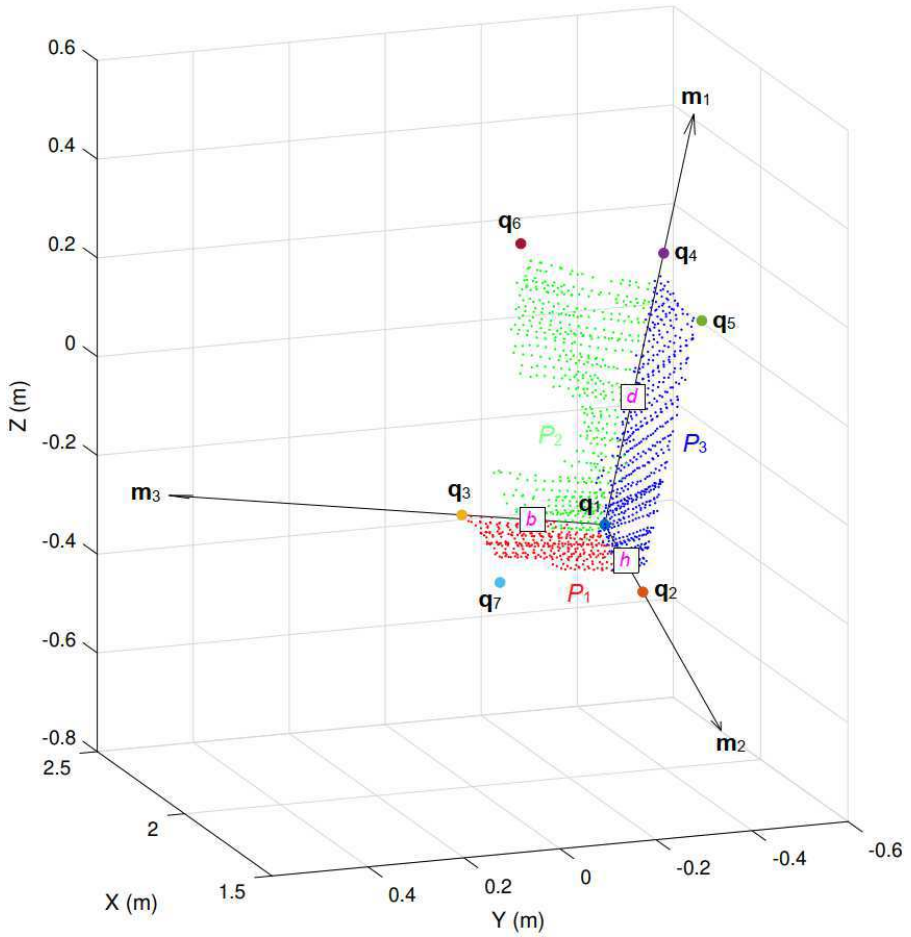

Figura 3-6: Ordinamento di piani, punti e spigoli nel modello finale del bersaglio.

3.2.3 Elaborazione dell'immagine termica

La terza fase del processo di calibrazione consiste nell'estrarre dall'immagine termica associata alla *point cloud* una sequenza di coordinate bidimensionali, che forniscono la posizione dei punti corrispondenti agli angoli visibili del bersaglio.

Prima di essere elaborata l'immagine viene privata del colore, così da poter essere rappresentata tramite una matrice i cui elementi sono i valori di intensità dei singoli pixel. Vengono poi rimossi gli effetti della distorsione radiale e tangenziale, utilizzando i coefficienti determinati durante la calibrazione della termocamera. I punti cercati sono quindi individuati ricorrendo al rilevatore di Harris-Stephens, essendo elementi dell'immagine geometricamente assimilabili ad angoli.

Un angolo all'interno di un'immagine bidimensionale può essere pensato come l'intersezione tra due linee, per cui ci si aspetta che in un intorno di tale punto il gradiente dell'intensità dei pixel presenti due direzioni dominanti ben distinte.

Si può quindi considerare una porzione di immagine $\mathcal{W}$ centrata in $\mathbf{u}$ e supporre che subisca una traslazione $\mathbf{r}$ rispetto alla sua posizione originale. Calcolando la somma dei quadrati degli scarti per l'intensità I dei pixel si ottiene

$$E(\mathbf{r}) = \sum_{\mathbf{u}\in\mathcal{W}} w(\mathbf{u}) \left[I(\mathbf{u}+\mathbf{r}) - I(\mathbf{u})\right]^2 \tag{3.24}$$

dove la funzione $w(\mathbf{u})$ determina la forma della finestra $\mathcal{W}$ e ne filtra il contenuto (generalmente applicando un filtro di tipo Gaussiano). Assumendo che la traslazione $\mathbf{r}$ sia piccola, espandendo in serie I in un intorno di $\mathbf{u}$ si ha $I(\mathbf{u}+\mathbf{r}) \approx I(\mathbf{u}) + \nabla I(\mathbf{u})\ \mathbf{r}$ e quindi:

$$E(\mathbf{r}) \approx \sum_{\mathbf{u}\in\mathcal{W}} w(\mathbf{u}) \left[\nabla I(\mathbf{u})\ \mathbf{r}\right]^2 = \sum_{\mathbf{u}\in\mathcal{W}} \mathbf{r}^\mathsf{T} \left[w(\mathbf{u})\ \nabla I(\mathbf{u})^\mathsf{T}\ \nabla I(\mathbf{u})\right] \mathbf{r} = \mathbf{r}^\mathsf{T}\, \mathbf{M}\, \mathbf{r} \tag{3.25}$$

Indicando con I_u e I_v le derivate parziali di I in direzione orizzontale e verticale (generalmente ottenute approssimando ∇I con un opportuno operatore, come quello di Sobel) risulta:

$$\mathbf{M} = \sum_{\mathbf{u}\in\mathcal{W}} w(\mathbf{u})\ \nabla I(\mathbf{u})^\mathsf{T}\ \nabla I(\mathbf{u}) = \sum_{\mathbf{u}\in\mathcal{W}} \begin{bmatrix} w(\mathbf{u})\, {I_u}^2 & w(\mathbf{u})\, I_u\, I_v \\ w(\mathbf{u})\, I_u\, I_v & w(\mathbf{u})\, {I_v}^2 \end{bmatrix} \tag{3.26}$$

Gli autovettori di $\mathbf{M}$ determinano le direzioni in cui la variazione della funzione $E(\mathbf{r})$ è massima o minima. Se gli autovalori corrispondenti sono entrambi piccoli la regione intorno a $\mathbf{u}$ apparirà uniforme, mentre se uno dei due è notevolmente maggiore dell'altro molto probabilmente conterrà un bordo. Solo quando i due autovalori considerati sono entrambi grandi si può supporre di aver individuato un angolo.

Ogni angolo del bersaglio viene cercato all'interno di una specifica porzione dell'immagine, cioè un riquadro di 15×15 pixel la cui posizione viene specificata indicando le coordinate (sempre in pixel) del suo vertice superiore sinistro. In questo modo i punti bidimensionali potranno essere elaborati nello stesso ordine dei punti tridimensionali corrispondenti.
Il rilevatore potrebbe comunque individuare più di un punto classificabile come angolo all'interno dello stesso riquadro. Ogni punto trovato viene quindi associato ad un parametro (*strength*) che rappresenta l'intensità e la qualità del rilevamento, come il valore

$$R_H = \det(\mathbf{M}) - k \cdot \mathrm{tr}^2(\mathbf{M}) \qquad (0.04 < k < 0.06) \tag{3.27}$$

della funzione di risposta di Harris-Stephens nel pixel considerato. Scegliere il punto in cui tale parametro ha il valore più elevato consente di selezionare l'angolo che ha prodotto una risposta più significativa da parte del rilevatore utilizzato.

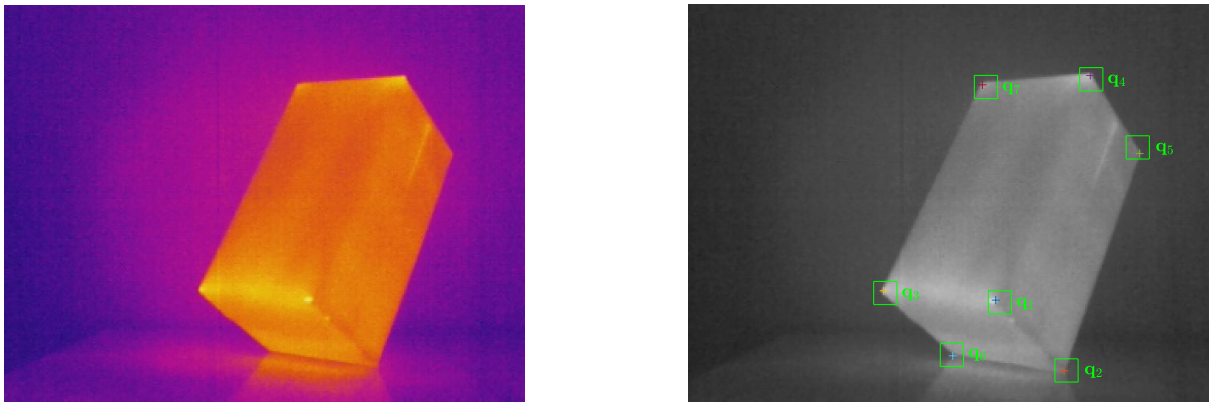

Figura 3-7: Individuazione degli angoli del bersaglio con il rilevatore di Harris-Stephens.

3.2.4 Determinazione dei parametri estrinseci

L'ultima fase del processo di calibrazione consiste nel ricavare i parametri estrinseci del sistema termocamera–LiDAR impostando un problema PnP con $n \leq 7$ coppie di punti.

In generale risolvere un problema PnP significa determinare la posizione e l'orientamento di una telecamera rispetto ad un opportuno sistema di riferimento esterno, conoscendo i suoi parametri intrinseci e avendo a disposizione n corrispondenze tra particolari punti 3–D dell'ambiente ed i relativi punti 2–D di un'immagine.
Il metodo EPnP (o *Efficient Perspective-n-Point*) consente di effettuare questa operazione senza ricorrere ad un procedimento iterativo. Non è quindi necessario impiegare un solutore numerico e cercare una stima iniziale della soluzione in grado di assicurare la convergenza.

Il problema PnP nel caso $n \geq 4$ può infatti essere affrontato esprimendo la posizione di ciascun punto 3–D come la somma pesata delle coordinate $\mathbf{v}_j^w$ di quattro punti di controllo virtuali, non complanari tra di loro. In altri termini, si pone

$$\mathbf{q}_i^w = \sum_{j=1}^{4} a_{ij}\, \mathbf{v}_j^w \qquad (i = 1, \dots, n) \tag{3.28}$$

dove $\mathbf{q}_i^w$ sono i punti tridimensionali estratti dal modello del bersaglio. La stessa relazione vale anche nel sistema di riferimento della termocamera, per cui risulta

$$\mathbf{q}_i^c = \sum_{j=1}^{4} a_{ij}\, \mathbf{v}_j^c \qquad (i = 1, \dots, n) \tag{3.29}$$

utilizzando gli stessi a_{ij} del caso precedente. Inoltre, per ogni $i = 1, \dots, n$ si ha:

$$\sum_{j=1}^{4} a_{ij} = 1 \tag{3.30}$$

Dopo aver scelto arbitrariamente i punti di controllo $\mathbf{v}_j^w$ si può utilizzare la (3.28) per calcolare i vari pesi. Sostituendo la (3.29) nella relazione di proiezione (2.12) si ottiene invece

$$\alpha_i \begin{Bmatrix} \mathbf{u}_i \\ 1 \end{Bmatrix} = \mathbf{A}\, \mathbf{q}_i^c = \mathbf{A} \sum_{j=1}^{4} a_{ij}\, \mathbf{v}_j^c \tag{3.31}$$

Ogni punto 3–D è quindi associato a due equazioni lineari (la terza permette di ricavare α_i per sostituirlo nelle altre due). Considerando tutti gli n punti si ottiene un sistema di $2n$ equazioni in 12 incognite, che sono rappresentate dalle coordinate $\mathbf{v}_j^c$ dei punti di controllo nel sistema di riferimento della termocamera. Queste incognite sono sottoposte a precisi vincoli, derivanti dal fatto che le distanze tra i punti di controllo teoricamente non dovrebbero cambiare passando dal sistema di riferimento della termocamera a quello del LiDAR.

Una volta risolto il sistema lineare come un problema agli autovalori, i parametri estrinseci $\mathbf{R}$ e $\mathbf{t}$ si ricavano dal confronto tra le coordinate $\mathbf{v}_j^c$ e $\mathbf{v}_j^w$ dei punti di controllo nei due diversi sistemi di riferimento, scegliendo la soluzione che minimizza l'errore di riproiezione medio sulle n coppie di punti. Una successiva ottimizzazione, effettuata utilizzando il metodo di Gauss-Newton, può consentire di raffinare ulteriormente i risultati ottenuti.

Capitolo 4

Processo di mappatura

4.1 Considerazioni generali

In questo capitolo viene descritto un processo di mappatura termica basato sulla *sensor fusion* tra il LiDAR 3–D e la termocamera, che produce una rappresentazione tridimensionale discreta della distribuzione di temperatura nell'ambiente attraversato dal rover.

Non è previsto che tale processo debba avvenire in tempo reale, quindi si assume di avere già a disposizione sia una sequenza di *point cloud* prodotte dal LiDAR che una sequenza di immagini termiche prodotte dalla termocamera. Le immagini termiche e le *point cloud* devono essere state acquisite simultaneamente, con gli strumenti nella stessa configurazione che avevano durante il processo di calibrazione. Si suppone anche che i parametri intrinseci della termocamera, insieme ai parametri estrinseci del sistema termocamera–LiDAR, siano stati ricavati con i procedimenti descritti nel capitolo precedente.

Il processo di mappatura può essere suddiviso in quattro fasi distinte. Nella prima, le mappe di colore e le mappe di temperatura sono estratte dalle immagini termiche e proiettate sulle *point cloud* corrispondenti. Nella seconda, le *point cloud* termiche così ottenute sono registrate e unite utilizzando le informazioni ricavate ricostruendo la traiettoria del rover. Nella terza, lo scenario termico formato dall'unione *point cloud* termiche registrate viene discretizzato da una funzione di decomposizione spaziale. Infine, nella quarta, l'insieme dei dati elaborati viene rappresentato graficamente per produrre la mappa di distribuzione di temperatura.

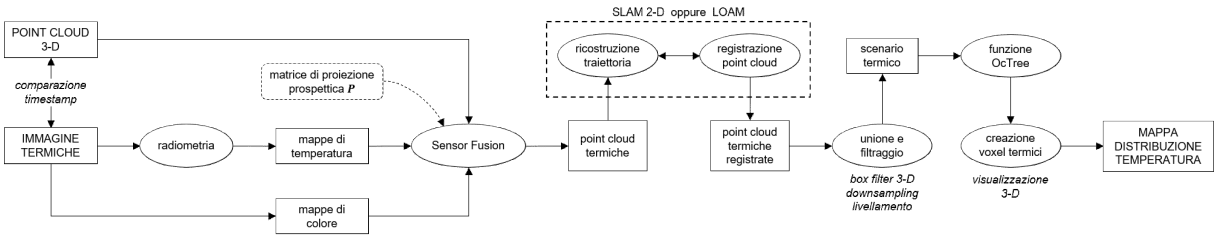

Figura 4-1: Panoramica del processo di mappatura termica.

Sono state valutate due possibili strategie per quanto riguarda la ricostruzione della traiettoria del rover e la registrazione delle *point cloud* termiche:

- Eseguire un'operazione di SLAM 2–D (ovvero *Simultaneous Localization And Mapping* in due dimensioni) per ricavare le informazioni necessarie a registrare correttamente le *point cloud* con una funzione basata sul metodo ICP (*Iterative Closest Point*).
- Ricorrere al metodo LOAM (*LiDAR Odometry And Mapping*) per ricostruire la traiettoria del rover e registrare le *point cloud* allo stesso tempo.

4.2 Sensor Fusion

La *sensor fusion* tra un LiDAR 3-D ed una termocamera consiste essenzialmente nell'associare ad ogni elemento di una *point cloud* le informazioni radiometriche relative ad uno specifico pixel di un'immagine termica.

Se la *point cloud* e l'immagine termica sono state acquisite nello stesso momento, si può ricorrere ancora alla (2.15) per passare dalle coordinate 3–D del riferimento dell'ambiente a quelle 2–D del riferimento dell'immagine. L'algoritmo di mappatura seleziona il pixel p_i che si trova più vicino al punto $\mathbf{u}_i = \{u_i,\, v_i\}^\mathsf{T}$ associato all'elemento $\mathbf{w}_i = \{X_i,\, Y_i,\, Z_i\}^\mathsf{T}$ della *point cloud* considerata. Poiché le immagini vengono elaborate sotto forma di matrici, si pone

$$\mathsf{p}_i = (r_i,\, c_i) = (\lfloor v_i \rceil,\, \lfloor u_i \rceil) \tag{4.1}$$

arrotondando u_i e v_i al numero intero più vicino. Si può notare come l'indice di riga r_i e l'indice di colonna c_i del pixel siano dati, rispettivamente, dalla coordinata verticale e dalla coordinata orizzontale del punto individuato.

La *sensor fusion* avviene associando ad ogni elemento della *point cloud* i valori che la posizione del pixel p_i individua nella mappa di colore e nella mappa di temperatura estratte dalla relativa immagine termica. Ripetendo l'operazione con tutte le *point cloud* e le corrispondenti immagini si ottiene una sequenza di *point cloud* termiche.

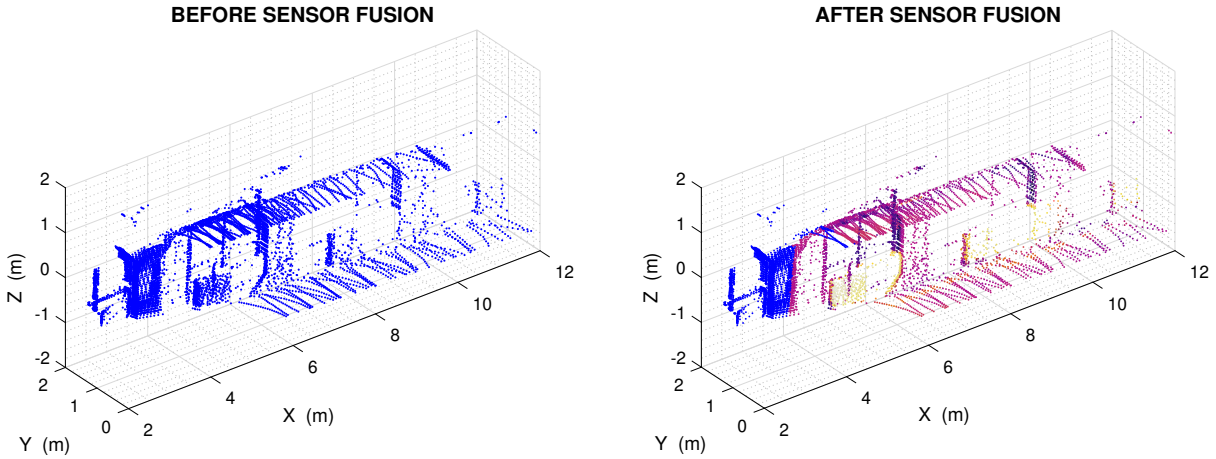

Figura 4-2: Esempio di *sensor fusion* effettuata dall'algoritmo di mappatura.

I punti tridimensionali che non sono compresi all'interno del campo di vista della termocamera corrispondono a punti bidimensionali posti fuori dell'immagine: le loro coordinate nel sistema di riferimento $(\mathrm{C} - uv)$ saranno negative o superiori alle dimensioni dell'immagine stessa. A questi punti viene associato un colore specifico, scelto in modo da non rientrare tra quelli utilizzati per visualizzare le immagini termiche. Ricevono anche `NaN` come valore di temperatura, in modo da essere facilmente identificabili sia nelle figure che all'interno delle variabili. La Figura 4-2 mostra l'aspetto di una *point cloud* prima e dopo questa operazione.

4.3 Registrazione e unione delle Point Cloud

Le coordinate dei punti tridimensionali acquisiti dal LiDAR sono sempre espresse nel sistema di riferimento dello strumento, che però cambia posizione ed orientamento rispetto all'ambiente durante il moto del rover. Le *point cloud* termiche, di conseguenza, non possono essere unite tra loro senza essere prima ricondotte ad un opportuno sistema di riferimento comune.

Registrare una *point cloud* rispetto ad un'altra, assumendo che esistano aree di sovrapposizione tra le due, significa trovare ed applicare la trasformazione spaziale che permette di allineare la prima con la seconda: si devono quindi individuare la rotazione, la traslazione e il cambiamento di scala che minimizzano un'opportuna funzione della distanza tra i due insiemi di punti.
Poiché la velocità con cui il LiDAR acquisisce nuovi punti è elevata rispetto alla velocità con cui si sposta, la distorsione delle *point cloud* dovuta al moto si può considerare trascurabile. Si può inoltre assumere che tutte le trasformazioni comprendano solamente componenti di rotazione e traslazione, senza alcun cambiamento di scala apprezzabile.

In questo caso ogni *point cloud* viene registrata rispetto a quella precedentemente acquisita, per cui componendo le relative trasformazioni l'intera sequenza potrà essere espressa nel sistema di riferimento associato alla *point cloud* iniziale. La Figura 4-3 mostra una sequenza di *point cloud* registrate (rappresentate con colori diversi, per poterle distinguere) che sono state espresse nello stesso sistema di riferimento.

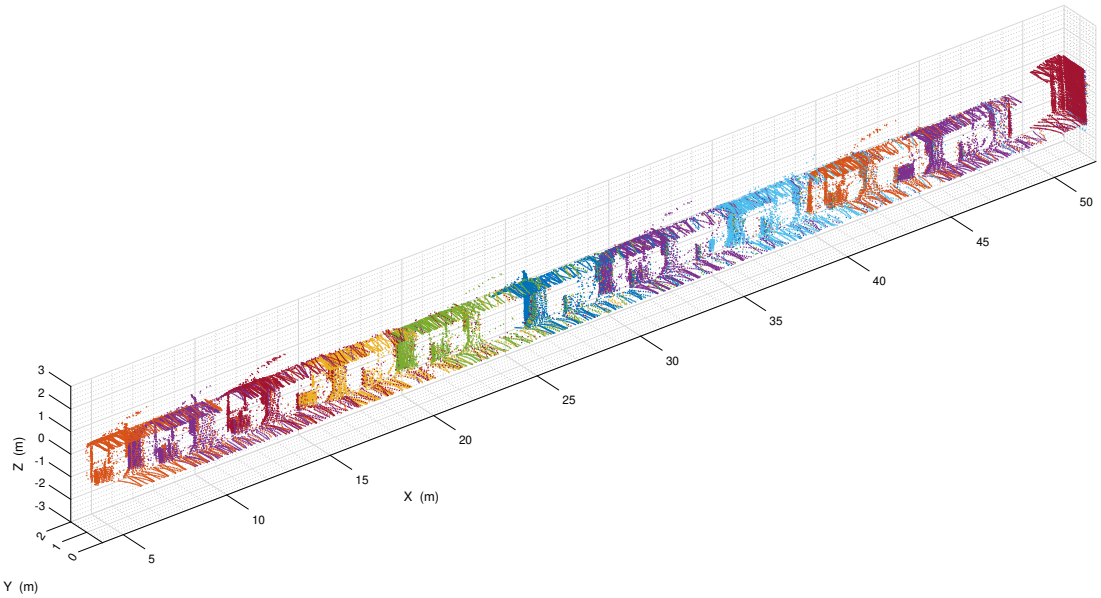

Figura 4-3: Sequenza di *point cloud* registrate rispetto allo stesso sistema di riferimento.

La registrazione consente di collegare le *point cloud* termiche in modo coerente, preservando la conformazione e la continuità delle strutture primitive (come rette, curve o superfici piane) che si trovano al loro interno. Questa unione, tuttavia, non può avvenire limitandosi a raggruppare in un unico insieme gli elementi delle *point cloud* disponibili. Così facendo si produrrebbero infatti eccessive concentrazioni di punti nelle aree di sovrapposizione.
L'algoritmo implementa quindi una forma di filtraggio, che è assimilabile all'applicazione di una *box filter* tridimensionale: dopo aver suddiviso il volume occupato dall'unione delle *point cloud* in cubi di dimensione costante, tutti i punti all'interno di uno stesso cubo vengono sostituiti con il punto individuato dalla media delle loro coordinate. Lo scopo dell'operazione è solo quello di uniformare la densità della *point cloud* ottenuta, quindi la griglia di filtraggio può anche avere una risoluzione (cioè un numero di suddivisioni per unità di lunghezza) molto maggiore di quella che verrà successivamente impiegata dalla funzione di decomposizione spaziale. L'applicazione della *box filter* riguarda tutte le proprietà ai punti, compresa la temperatura.

L'insieme di punti viene poi ricampionato, selezionando in modo casuale solo una certa frazione dei suoi elementi (*random downsampling*). Si procede quindi a livellarlo, adattando un modello di piano ai punti e applicando la trasformazione rigida che rende tale piano orizzontale. Questa operazione però può essere eseguita solo se il dataset utilizzato è stato acquisito in un ambiente

dove il suolo ha un'inclinazione nota e costante: in tutti gli altri casi per eseguire il livellamento si devono avere a disposizione delle informazioni d'assetto aggiuntive, come quelle fornite da una piattaforma inerziale. Successivamente si utilizzerà il termine "scenario termico" per riferirsi alla *point cloud* termica così ottenuta, poiché costituisce una rappresentazione dell'intero ambiente attraversato dal rover nel corso del processo di mappatura.

4.3.1 Registrazione basata su SLAM 2–D

La prima versione dell'algoritmo utilizza una funzione basata sul metodo ICP (*Iterative Closest Point*) per registrare le *point cloud* termiche. La formulazione adottata è quella originariamente proposta da Z. Zhang nel 1993.

Il metodo ICP consente di registrare due *point cloud* senza bisogno di individuare al loro interno delle caratteristiche (cioè delle strutture primitive) da confrontare. I punti sono infatti trattati come gli elementi di due superfici generiche, la cui distanza reciproca dovrà essere minimizzata. Se tale distanza è sin dall'inizio sufficientemente ridotta, si può assumere che un qualsiasi punto della *point cloud* da registrare si trovi poco distante dal suo corrispettivo nella *point cloud* di riferimento. Una volta associato ad ogni elemento della prima il più vicino elemento della seconda è quindi possibile determinare la trasformazione rigida che avvicina maggiormente i due insiemi di punti. Ripetere iterativamente questa procedura, fino a raggiungere un'opportuna condizione di arresto, porta a registrare le due *point cloud* con la precisione richiesta.

L'aspetto più problematico di questo approccio non è l'individuazione della trasformazione (che si ottiene risolvendo numericamente un problema ai minimi quadrati non lineare) ma la corretta associazione tra gli elementi delle due *point cloud* considerate. Siano infatti $\mathcal{S}$ la *point cloud* da registrare e $\mathcal{R}$ quella presa come riferimento:

$$\begin{aligned} \mathcal{S} &= \{\mathbf{x}_i \in \mathbb{R}^3 \mid i = 1, \dots, m\} \\ \mathcal{R} &= \{\mathbf{y}_j \in \mathbb{R}^3 \mid j = 1, \dots, n\} \end{aligned} \tag{4.2}$$

La registrazione di $\mathcal{S}$ rispetto a $\mathcal{R}$ avviene determinando la matrice di rotazione $\mathbf{R}$ ed il vettore di traslazione $\mathbf{t}$ che minimizzano la funzione

$$\mathcal{F}(\mathbf{R},\, \mathbf{t}) = \frac{1}{\sum_{i=1}^{m} p_i} \sum_{i=1}^{m} p_i \, d^{\,2}\left(\mathbf{R}\,\mathbf{x}_i + \mathbf{t},\, \mathcal{R}\right) \tag{4.3}$$

sapendo che la distanza di un punto $\mathbf{x} \in \mathcal{S}$ dalla superficie rappresentata da $\mathcal{R}$ è:

$$d\left(\mathbf{x},\, \mathcal{R}\right) = \min_{\mathbf{y} \in \mathcal{R}} \|\mathbf{x} - \mathbf{y}\| = \|\mathbf{x} - \mathbf{y}^\star\| \tag{4.4}$$

Per ogni punto di $\mathcal{S}$ è infatti sempre possibile indicare il punto $\mathbf{y}^\star \in \mathcal{R}$ ad esso più vicino, ovvero quello individuato dalla condizione:

$$\|\mathbf{x} - \mathbf{y}^\star\| \leq \|\mathbf{x} - \mathbf{z}\| \quad \forall \mathbf{z} \in \mathcal{R} \tag{4.5}$$

Nulla però assicura che $\mathbf{x}$ e $\mathbf{y}^\star$ rappresentino lo stesso punto tridimensionale, vista l'inevitabile presenza di elementi spuri (*outliers*) in entrambe le *point cloud* da registrare. Questa particolare versione del metodo ICP individua le corrispondenze ricorrendo a strutture di tipo *k–d tree* e le filtra analizzando la distribuzione statistica delle distanze tra i punti.
Supponendo di trovare N corrispondenze valide, si può porre $p_i = 1$ quando $\mathbf{x}_i$ è correttamente associato con il relativo $\mathbf{y}_i^\star$ e $p_i = 0$ in caso contrario. La (4.3) allora si riduce a:

$$\mathcal{F}(\mathbf{R},\, \mathbf{t}) = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{R}\,\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i^\star\|^2 \tag{4.6}$$

Si deve comunque avere a disposizione una stima sufficientemente accurata della trasformazione di registrazione, per fornirla come valore di avvio al solutore che minimizza la (4.6) utilizzando l'algoritmo di Levenberg–Marquardt.
Tale stima può essere ottenuta ricostruendo la traiettoria del rover: la trasformazione cercata è infatti quella che descrive lo spostamento degli strumenti nell'intervallo di tempo intercorso tra l'acquisizione delle due *point cloud* da registrare.

In questa versione dell'algoritmo la ricostruzione della traiettoria avviene attraverso un'operazione di SLAM 2–D (*Simulaneous Localization And Mapping* in due dimensioni) effettuata con le *point cloud* acquisite dal LiDAR.
Le informazioni necessarie a creare una mappa dell'ambiente in cui localizzare il rover possono essere rappresentate sinteticamente attraverso un *pose graph* bidimensionale. Ogni osservazione effettuata dagli strumenti viene collocata su un nodo del *pose graph* e correlata alle altre da una funzione di associazione (*scan matching*). In questo caso le osservazioni sono costituite da sezioni bidimensionali delle *point cloud* tridimensionali, estratte selezionando tutti i punti compresi in un ristretto intervallo di elevazione ΔZ e proiettandoli sul piano orizzontale. Si ottiene così una struttura a rete, costituita da una successione di nodi identificati dalla loro posizione e dal loro orientamento. Ognuno dei nodi è soggetto ad uno o più vincoli (*edge constraint*) che lo collegano agli altri nodi o a dei punti di riferimento esterni (*landmarks*).

La funzione di associazione implementa una forma di registrazione delle *point cloud* basata sulle caratteristiche: ogni nuova osservazione viene confrontata con quella precedente, comparando la forma e la posizione delle strutture primitive presenti nelle aree di sovrapposizione. L'operazione è agevolata dal fatto che, dovendo elaborare delle *point cloud* bidimensionali, le caratteristiche da riconoscere sono costituite solo da linee ed angoli.
Una nuova osservazione viene effettivamente inserita nel *pose graph*, producendo un nuovo nodo ed un nuovo vincolo, solo quando il cambiamento di posizione e di orientamento associato alla sua registrazione supera una soglia fissata. Ogni osservazione accettata viene quindi confrontata con tutte le osservazioni situate entro una certa distanza dal nodo in cui è collocata: in questo modo si possono introdurre vincoli aggiuntivi tra nodi non consecutivi, che rappresentano delle condizioni di *loop closure* per la traiettoria.
Incontrare queste condizioni con sufficiente frequenza permette di ottimizzare il *pose graph* per compensare l'accumularsi degli errori di localizzazione, individuando la configurazione dei nodi che rispetta il maggior numero possibile di vincoli contemporaneamente.

Le informazioni sulla posizione e sulla direzione di movimento del rover, contenute nei nodi del *pose graph*, consentono di ricavare una sequenza di trasformazioni piane da fornire alla funzione che implementa il metodo ICP. Le stesse informazioni possono essere utilizzate anche per creare una *occupancy map* a partire dalle *point cloud* 3–D o dalle loro sezioni bidimensionali.
Il termine *occupancy map* si riferisce ad una struttura di dati a griglia che rappresenta un certo ambiente come una successione di celle della stessa dimensione, disposte in modo regolare. Nelle *occupancy map* binarie ogni cella viene considerata "libera" se non contiene nulla e "occupata" in caso contrario. Nelle *occupancy map* probabilistiche, invece, ad ogni cella viene associata una variabile numerica: valori vicini a 1 indicano che la cella ha un'elevata probabilità di contenere ostacoli, mentre valori vicini a 0 indicano che molto probabilmente non ne contiene.

La costruzione di una *occupancy map* probabilistica avviene aggiornando in modo progressivo i valori associati alle celle in cui ricadono gli elementi di ogni *point cloud* aggiunta. Per agevolare l'operazione, la probabilità p è convertita in formato *log-odds* ponendo:

$$r(p) = \log\left(\frac{p}{1-p}\right) \tag{4.7}$$

In questo modo, integrare una nuova osservazione p_n con il valore di probabilità p_c di una data cella equivale a sostituire p_c con:

$$p_c^{\star} = r^{-1}\big(\, r_c + r_n\big) = r^{-1}\big(\, r\,(p_c) + r\,(p_n)\,\big) \tag{4.8}$$

Dato che la funzione $r(p)$ assume valori compresi tra $-\infty$ e $+\infty$ (per $0 \leq p \leq 1$) generalmente si fissano delle soglie di saturazione per ogni cella, superate le quali il valore *log-odds* assegnato non può aumentare o diminuire ulteriormente. In caso contrario, l'accumularsi delle osservazioni precedenti impedirebbe l'aggiornamento della *occupancy map* qualora la situazione di una data cella dovesse cambiare (ad esempio perché è stato rimosso un oggetto dalla scena).
La Figura 4-4 mostra un esempio di *occupancy map* bidimensionale, nella quale i valori assunti dalle variabili assegnate alle celle sono rappresentate con toni diversi di grigio. I valori di soglia che determinano se una cella appare "occupata" (nero) oppure "libera" (bianco) variano in base all'applicazione considerata. Alle celle non influenzate da alcuna osservazione, come quelle oltre le pareti, viene invece assegnato un valore di p_i predefinito.

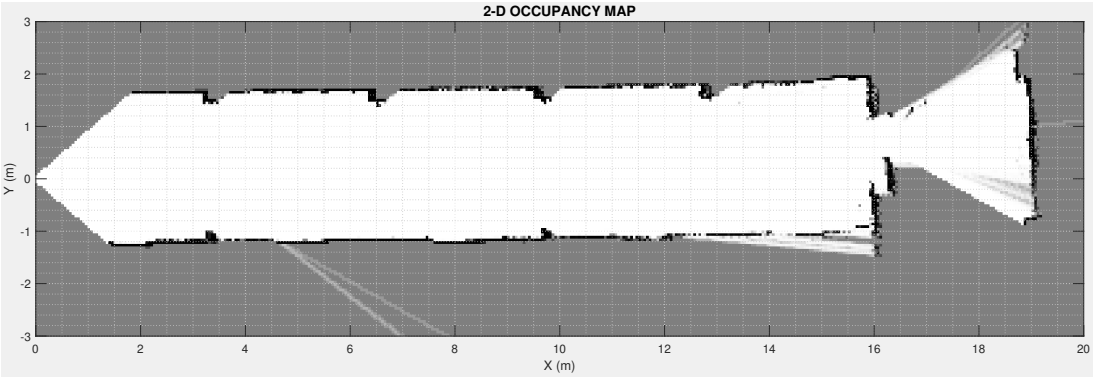

Figura 4-4: Esempio di *occupancy map* bidimensionale.

4.3.2 Registrazione basata su LOAM

La seconda versione dell'algoritmo utilizza il metodo LOAM (*LiDAR Odometry And Mapping*) per ricostruire la traiettoria del rover e registrare le *point cloud* termiche attraverso un processo di odometria e mappatura in tempo reale.
A differenza di quanto avviene nel caso della SLAM 2–D, questo metodo consente di ridurre al minimo la deriva della traiettoria stimata (cioè la sua deviazione da quella reale, provocata dal progressivo accumularsi degli errori di localizzazione) indipendentemente dalla frequenza con cui si possono incontrare condizioni di *loop closure* lungo il percorso.

L'approccio adottato prevede di suddividere un generico problema di SLAM 3–D, che coinvolge un elevato numero di variabili, in due sequenze distinte di operazioni:

- Una prima *funzione di odometria* opera a frequenze elevate ($\sim$10 Hz) per determinare la velocità di movimento del LiDAR, fornendo così una stima iniziale della trasformazione di registrazione e rimuovendo dalle *point cloud* le eventuali distorsioni dovute al moto.

- Una seconda *funzione di mappatura* opera a frequenze minori ($\sim$1 Hz) per registrare con la precisione richiesta le *point cloud* filtrate, costruendo la mappa dell'ambiente.

Le informazioni provenienti dalle due funzioni vengono poi integrate, ottenendo la trasformazione che definisce la posizione e l'orientamento del rover all'interno della mappa creata: è il fatto che le due sequenze di operazioni siano eseguite in parallelo, anche se con frequenze diverse, a consentire di eseguire l'intera procedura in tempo reale.

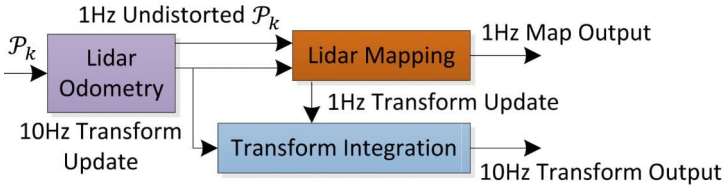

Figura 4-5: Panoramica del metodo LOAM (J. Zhang e S. Singh, 2014).

Entrambe le funzioni estraggono dalle *point cloud* gli stessi punti di interesse, cioè quelli che si trovano sulle sezioni piane o sugli spigoli dove tali sezioni si incontrano. Nella fase di odometria si valuta solo la corrispondenza dei punti individuati nelle diverse *point cloud*, privilegiando la velocità di esecuzione. Nella fase di mappatura la registrazione viene perfezionata considerando anche la distribuzione spaziale dei gruppi di punti caratteristici.
Per essere significativi i punti di interesse devono trovarsi sufficientemente lontani da altri punti dello stesso tipo, non devono trovarsi in prossimità di discontinuità o di regioni occultate (dove la frazione di *outlier* nella *point cloud* tende ad essere più elevata) e non devono appartenere a superfici quasi parallele alla linea di vista del LiDAR (poiché in quei casi la precisione con cui lo strumento misura le distanze si riduce).

In origine la formulazione del metodo LOAM prevedeva di elaborare *point cloud* tridimensionali formate dall'unione di numerose scansioni bidimensionali, effettuate su piani diversi e acquisite progressivamente durante il moto del LiDAR. Questa assunzione permetteva di imporre vincoli di complanarità utili per l'individuazione e l'associazione dei punti caratteristici, ma in seguito è stata superata con l'introduzione di algoritmi e solutori più versatili.

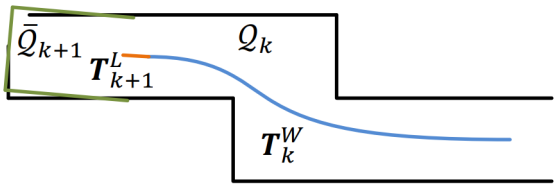

Figura 4-6: Integrazione delle trasformazioni nel metodo LOAM (J. Zhang e S. Singh, 2014).

La Figura 4-6 mostra come la stima del moto $\mathbf{T}^L_k$ fornita dalla funzione di odometria (la curva arancione) e il posizionamento $\mathbf{T}^W_k$ del rover fornito dalla funzione di mappatura (la curva blu) siano combinate per registrare una *point cloud* $\overline{\mathcal{Q}}_{k+1}$ (in verde) sulla mappa $\mathcal{Q}_k$ attuale. Questa osservazione viene poi correlata con quelle precedenti, espandendo la mappa e determinando la trasformazione raffinata $\mathbf{T}^W_{k+1}$ che aggiunge alla traiettoria un nuovo segmento.

Il processo di registrazione risulta quindi notevolmente semplificato, poiché non è più necessario ricorrere ad una funzione basata sul metodo ICP: ad ogni *point cloud*, precedentemente fusa con l'immagine termica associata, viene applicata direttamente la relativa trasformazione.

4.4 Decomposizione OcTree

La terza fase del processo di mappatura consiste nel decomporre lo scenario termico, costituito dall'unione delle *point cloud* termiche registrate, mediante una griglia tridimensionale.
Per ottenere questo risultato i punti vengono raggruppati all'interno di entità che rappresentano specifiche porzioni dello spazio, ognuna delle quali è associata alla temperatura media dei punti che contiene. Si ottiene così una rappresentazione discreta della distribuzione di temperatura in tutto l'ambiente mappato, che avrà la stessa risoluzione della griglia utilizzata.

La funzione per la decomposizione spaziale è basata sul modello *OcTree*, cioè sulla suddivisione ricorsiva di un volume in otto sottovolumi identici. Un'operazione di questo tipo costruisce una struttura di dati ad albero, su più livelli, nella quale ogni nodo rappresenta un cubo con i lati di una determinata lunghezza. Questi elementi di volume sono detti *voxel*, poiché possono essere interpretati come una versione tridimensionale dei pixel bidimensionali.
I rami della struttura collegano tra loro solo nodi che si trovano su livelli diversi. Se le proprietà assegnate ad un particolare nodo non sono sufficienti a descrivere in modo completo il relativo volume, da quel nodo partono otto rami che conducono ad altrettanti discendenti (*child node*) posti al livello inferiore. In questo modo si arriva a rappresentare l'intero contenuto del volume di partenza con una risoluzione pari alla dimensione dei *voxel* terminali, cioè quelli posti all'ultimo livello della suddivisione.

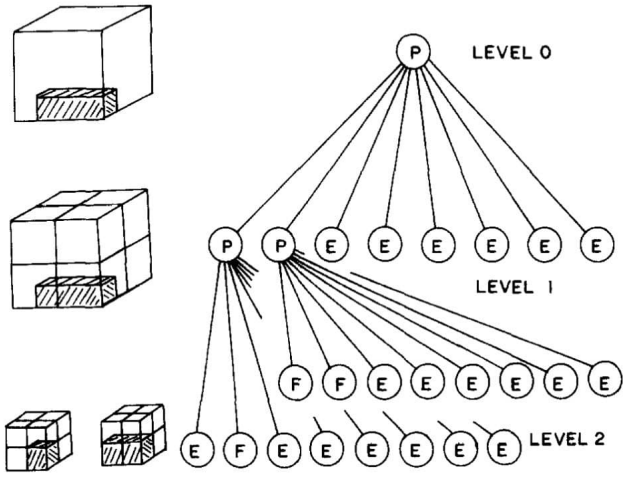

Figura 4-7: Organizzazione di una struttura *OcTree* (D. Meagher, 1981).

Dopo aver creato la struttura è possibile diminuire il numero di livelli o eliminare alcuni elementi senza dover effettuare un'altra decomposizione. Questo approccio consente di accedere in modo rapido ed efficiente alle proprietà dei singoli nodi, semplificando notevolmente l'elaborazione dei dati contenuti nella struttura stessa.

Quando viene applicata allo scenario termico, la funzione di decomposizione spaziale produce una struttura *OcTree* nella quale i nodi terminali sono dei "contenitori" di punti tridimensionali. Inizialmente tutti i punti dello scenario termico sono associati al solo nodo iniziale, per poi essere opportunamente ripartiti tra i suoi discendenti quando il volume iniziale viene suddiviso. Ognuno dei sottovolumi ottenuti potrà quindi essere suddiviso ulteriormente, espandendo la struttura su un nuovo livello.
La decomposizione lungo un determinato ramo si interrompe solo quando viene soddisfatta una particolare condizione, ad indicare che la risoluzione desiderata per la mappa è stata raggiunta o superata. In questo caso l'unica condizione di arresto ammessa dalla funzione di decomposizione è rappresentata dalla dimensione minima che possono avere i *voxel* generati: in altri termini, un nodo non può avere discendenti se il lato del corrispondente *voxel* ha una lunghezza inferiore ad una soglia D fissata. I nodi a cui non risultano associati dei punti non contengono informazioni utili, per cui sono eliminati automaticamente.
Imporre una simile condizione di arresto, tuttavia, non consente di stabilire quale sarà l'effettiva risoluzione della mappa di distribuzione di temperatura. Questo perché la dimensione di un *voxel* appartenente al livello di suddivisione n-esimo può essere solo

$$d = \frac{L}{2^{n-1}} \tag{4.9}$$

dove L è il lato del più piccolo volume cubico in grado di racchiudere completamente lo scenario termico. Si può però prevedere il numero N di suddivisioni necessarie per raggiungere o superare la risoluzione voluta, poiché ponendo $d \leq D$ per $n = N$ nella (4.9) si ottiene:

$$N = \lceil 1 + \log_2 (L/D) \rceil \tag{4.10}$$

4.5 Creazione della mappa di distribuzione di temperatura

Le precedenti sezioni dell'algoritmo di mappatura hanno permesso di trasformare una sequenza di *point cloud* e immagini termiche radiometriche in uno scenario termico

$$\mathsf{S} = \{\mathbf{p}_k,\, (r, g, b)_k,\, T_k \mid k = 1, \dots, N_p\} \tag{4.11}$$

nel quale l'elemento che occupa la posizione $\mathbf{p}_k = \{X_k,\, Y_k,\, Z_k\}^\mathsf{T}$ è associato anche ad un certo colore $(r, g, b)_k$ e ad un valore T_k di temperatura. La relativa struttura *OcTree* descrive in modo sintetico l'organizzazione spaziale di questi elementi, distribuendoli tra i diversi nodi presenti in ogni livello di suddivisione.

Per produrre una mappa di distribuzione di temperatura che sia rappresentabile graficamente è necessario estrarre dai nodi terminali di questa struttura alcune informazioni specifiche, ovvero: la dimensione dei *voxel* corrispondenti, la loro collocazione nello spazio e il valore di temperatura a cui possono essere associati.

La funzione di decomposizione spaziale fornisce direttamente la dimensione d del lato dei *voxel* terminali, oltre alle coordinate $\mathbf{w}_i^c = \{X_i^c,\, Y_i^c,\, Z_i^c\}^\mathsf{T}$ dei loro centroidi. La funzione che consente di visualizzarli, tuttavia, localizza ogni *voxel* attraverso le coordinate

$$\mathbf{w}_i^v = \{X_i^v,\, Y_i^v,\, Z_i^v\}^\mathsf{T} = \left\{ X_i^c - \frac{1}{2}\, d,\, Y_i^c - \frac{1}{2}\, d,\, Z_i^c - \frac{1}{2}\, d \right\}^\mathsf{T} \tag{4.12}$$

dell'angolo più vicino all'origine degli assi (cioè il vertice inferiore destro della faccia posteriore del volume cubico). Se necessario, sommare a tutti i $\mathbf{w}_i^c$ uno stesso vettore $\mathbf{t}_0$ consente di traslare il sistema di riferimento della mappa rispetto a quello dell'ambiente.

Indicando con T_j^i la temperatura del j-esimo punto nel nodo terminale i-esimo, la temperatura di riferimento per il *voxel* associato sarà

$$T_i^v = \frac{1}{N_i} \sum_{j=1}^{N_i} T_j^i \qquad (i = 1, \dots, N_v) \tag{4.13}$$

dove N_i è il numero di punti contenuti nel volume considerato. Si può quindi porre

$$\tau_i = \left\lfloor 1 + (n_c - 1)\, \frac{T_i^v - T_{min}^v}{T_{max}^v - T_{min}^v} \right\rfloor \tag{4.14}$$

dove T_{max}^v e T_{min}^v sono, rispettivamente, il massimo ed il minimo valore di temperatura media associato ai *voxel* in cui è stato suddiviso lo scenario termico. Il parametro τ_i così ottenuto potrà essere usato per selezionare univocamente uno degli n_c colori scelti per visualizzare la mappa di distribuzione di temperatura. Si ottiene così una sequenza di *voxel* termici

$$\mathsf{V} = \{\mathbf{w}_i^v,\, T_i^v,\, \tau_i \mid i = 1, \dots, N_v\} \tag{4.15}$$

rappresentabili come dei cubi del colore appropriato.

Capitolo 5

Implementazione e prove sperimentali

5.1 Calibrazione intrinseca

In questa sezione viene descritta l'implementazione del processo di calibrazione intrinseca della termocamera, effettuata seguendo la strategia descritta nella Sezione 3.1 e utilizzando l'applicazione `cameraCalibrator` inclusa nel *Computer Vision Toolbox* di MATLAB.

La Figura 5-1 mostra le modalità con cui si è svolta l'operazione. Il bersaglio di calibrazione (a sinistra) è posto di fronte ad uno sfondo opaco, ad una certa distanza dalla termocamera e dal computer portatile a cui è collegata (a destra).

Figura 5-1: Setup per la calibrazione intrinseca della termocamera.

Si è scelto di utilizzare come bersaglio una scacchiera di 300×250 mm, costituita da 30 riquadri di 50×50 mm disposti su 5 righe e 6 colonne. Lo schema è stato realizzato applicando del nastro adesivo alluminato su un pannello di legno non verniciato. Una staffa nella parte posteriore del pannello, collegata ad un supporto orientabile, mantiene in posizione il bersaglio.
L'impiego di due materiali con caratteristiche termiche diverse assicura che lo schema sia visibile anche nell'infrarosso, ma se necessario si può aumentare il contrasto tra i riquadri esponendo il bersaglio alla luce solare subito prima di inquadrarlo con la termocamera: le sezioni coperte dal nastro tendono infatti a riscaldarsi più lentamente di quelle scoperte.

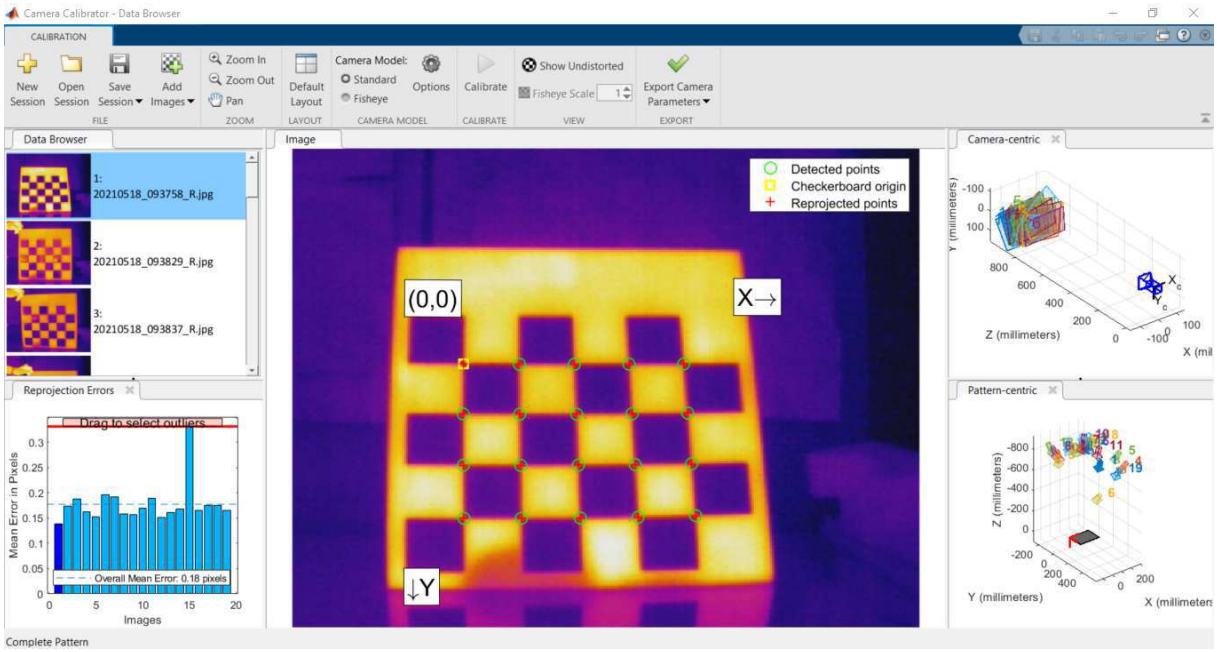

Figura 5-2: Applicazione utilizzata per la calibrazione intrinseca della termocamera.

La Figura 5-2 mostra la schermata dell'applicazione `cameraCalibrator` una volta inserite tutte le immagini termiche contenute nel dataset di calibrazione. Per produrre risultati accettabili, le immagini utilizzate devono rispettare i seguenti requisiti:

- Il bersaglio deve essere visibile da almeno 3 angolazioni diverse.
- Il bersaglio non deve essere inclinato a più di 45° rispetto al piano dell'immagine.
- La distanza di osservazione non deve variare eccessivamente tra un'immagine e l'altra.
- Lo schema a scacchiera deve occupare almeno il 20% di ogni immagine.
- Tutti i punti caratteristici dello schema devono essere identificabili.

I grafici nella parte destra della schermata mostrano la posizione e l'orientamento del bersaglio rispetto alla termocamera (o viceversa) in ognuna delle immagini analizzate. L'istogramma nella parte sinistra mostra invece l'errore di riproiezione medio nei vari casi.

I risultati ottenuti per la termocamera FLIR Vue Pro R utilizzando un dataset di calibrazione composto da 22 immagini (3 delle quali scartate automaticamente) sono:

- Lunghezza focale in direzione orizzontale: $f_u = 576.75$ pixel
- Lunghezza focale in direzione verticale: $f_v = 550.82$ pixel
- Coordinata orizzontale del centro ottico: $u_0 = 166.69$ pixel
- Coordinata verticale del centro ottico: $v_0 = 108.05$ pixel
- Fattore di inclinazione (*skew factor*): $s_\theta = 0.2855$
- Coefficienti per la distorsione radiale: $k_1 = -0.3868$, $k_2 = -0.6829$
- Coefficienti per la distorsione tangenziale: $k_3 = -0.0020$, $k_4 = 0.0028$
- Errore di riproiezione medio su tutti i punti: $\varepsilon_{avg} = 0.1771$ pixel

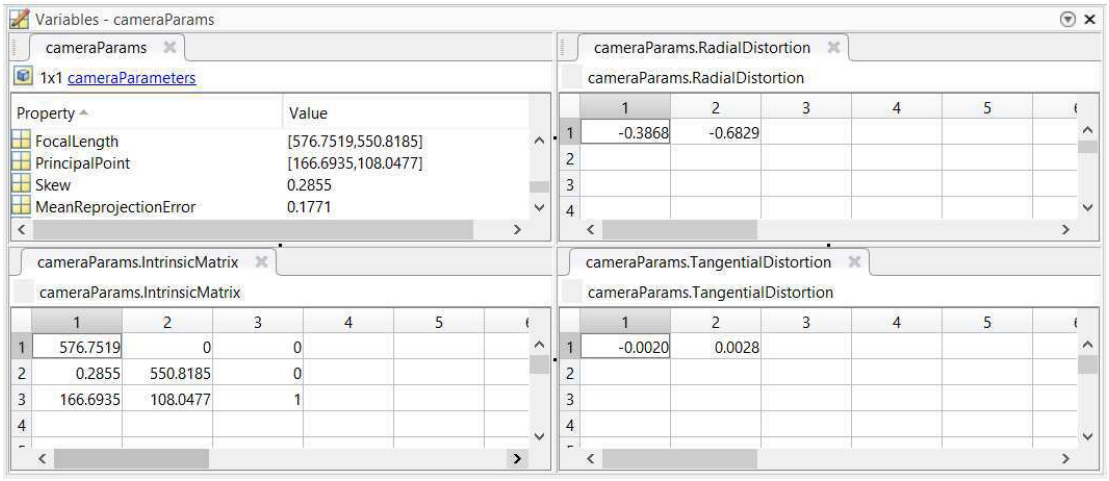

Figura 5-3: Variabile contenente i parametri intrinseci della termocamera.

5.2 Calibrazione estrinseca

In questa sezione viene descritta l'implementazione del processo di calibrazione estrinseca per il sistema termocamera–LiDAR. Teoricamente questa operazione sarebbe potuta avvenire usando lo stesso bersaglio impiegato per la calibrazione intrinseca della termocamera, cioè uno schema a scacchiera piano visibile anche nell'infrarosso. Nella pratica, tuttavia, una soluzione di questo tipo non avrebbe prodotto risultati accettabili.

Individuare il contorno di una superficie piana all'interno di una *point cloud*, determinando con precisione la sua forma e la sua estensione, è infatti estremamente complesso. La presenza dello schema a scacchiera sulla superficie considerata, inoltre, degrada sensibilmente la qualità della *point cloud* ottenuta: il bersaglio potrebbe non essere più riconoscibile come un piano, poiché le differenze di riflettività impediscono al LiDAR di misurare le distanze in modo coerente. L'effetto risulta particolarmente evidente quando, come in questo caso, lo schema è ottenuto applicando uno strato di materiale metallico su una superficie opaca.

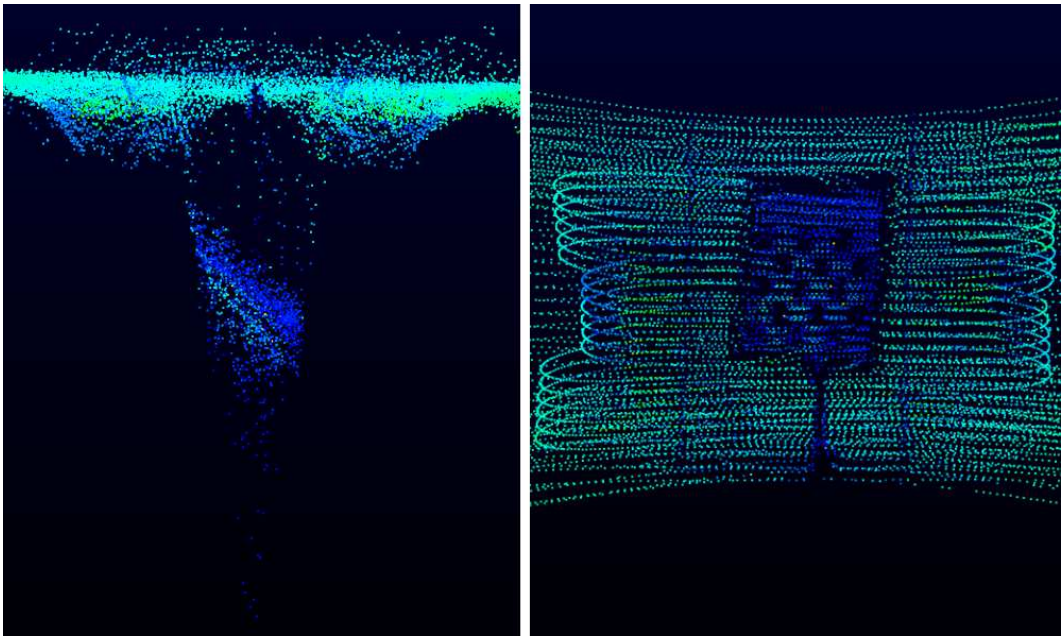

Figura 5-4: Aspetto dello schema a scacchiera nelle *point cloud* acquisite dal LiDAR.

La Figura 5-4 mostra come apparirebbe lo schema a scacchiera in nastro alluminato nelle *point cloud* acquisite dal LiDAR. Osservando la scena dal punto di vista degli strumenti (a destra) si nota chiaramente la differente riflettività del materiale nei diversi riquadri. Osservando la stessa scena dall'alto (a sinistra) viene invece evidenziata la dispersione dei punti nella direzione in cui il LiDAR effettua le misurazioni di distanza.

Si è quindi scelto di utilizzare la strategia sviluppata da Pusztai e Hajder, descritta nella Sezione 3.2, utilizzando come bersaglio una comune scatola in cartone ondulato.

La superficie della scatola, opaca e uniforme, non è stata sottoposta a trattamenti o lavorazioni particolari. La sua visibilità nell'infrarosso, di conseguenza, è assicurata solamente dal contrasto termico con uno sfondo a temperatura inferiore (o sufficientemente lontano). L'unica accortezza adottata è stata quella di riscaldare il materiale con una pistola termica subito prima di acquisire ogni dataset di calibrazione.
Un bersaglio di questo tipo può essere approntato rapidamente, utilizzando materiali facilmente reperibili, il che costituisce un vantaggio nel caso in cui sia necessario effettuare la calibrazione più volte (ad esempio, perché il supporto degli strumenti è stato modificato).

Figura 5-5: Bersaglio per la calibrazione estrinseca del sistema termocamera-LiDAR.

La Figura 5-5 mostra la forma e le dimensioni ($b = h = 0.295$ m, $d = 0.595$ m) della scatola. Le sue pareti sono abbastanza estese da produrre dei piani riconoscibili nelle *point cloud* acquisite da circa 2 m di distanza. L'inevitabile rumore di bordo (*edge noise*) non consente di riconoscere direttamente i contorni di questi piani, ma non impedisce di adattare ai punti il modello da cui saranno ricavate le coordinate tridimensionali degli angoli del bersaglio.
È preferibile che il bersaglio mostri agli strumenti il massimo numero possibile di angoli, così da poter impostare il problema PnP utilizzando fino a sette coppie di coordinate. Tale condizione non si rivela particolarmente restrittiva, poiché viene meno solo quando una parete della scatola è esattamente perpendicolare ad uno degli assi del sistema di riferimento del LiDAR. Inoltre, se il piano di base è orizzontale, la necessità di inclinare opportunamente il bersaglio garantisce di poter distinguere tale superficie dalle pareti della scatola.

La Figura 5-6 mostra il risultato delle operazioni di *plane fitting* che permettono di eliminare il piano di base ed estrarre dalla *point cloud* le tre pareti visibili del bersaglio.

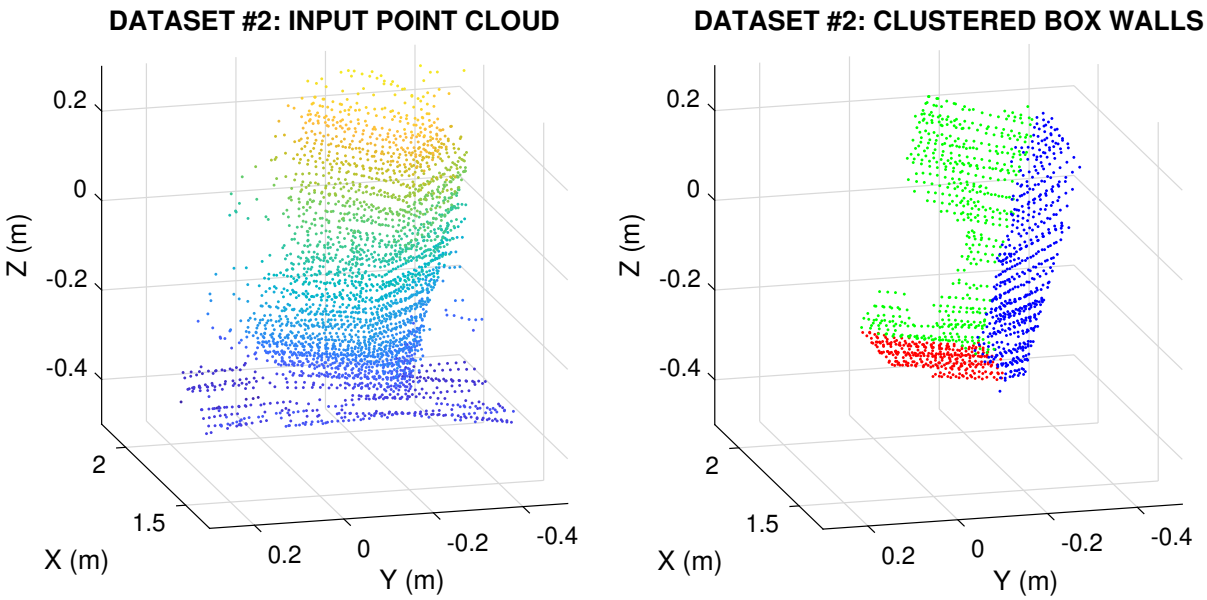

Figura 5-6: Riconoscimento ed estrazione delle pareti del bersaglio.

La Figura 5-7 mostra invece come appare il modello del bersaglio prima e dopo il raffinamento iterativo, evidenziando anche la posizione dei punti tridimensionali estratti. Nel modello iniziale i versori degli spigoli $\hat{\mathbf{m}}_i$ sono rappresentati come frecce tratteggiate, mentre nel modello finale come frecce continue. In entrambi i casi il punto $\mathbf{q}_1$ è individuato dalla loro intersezione.

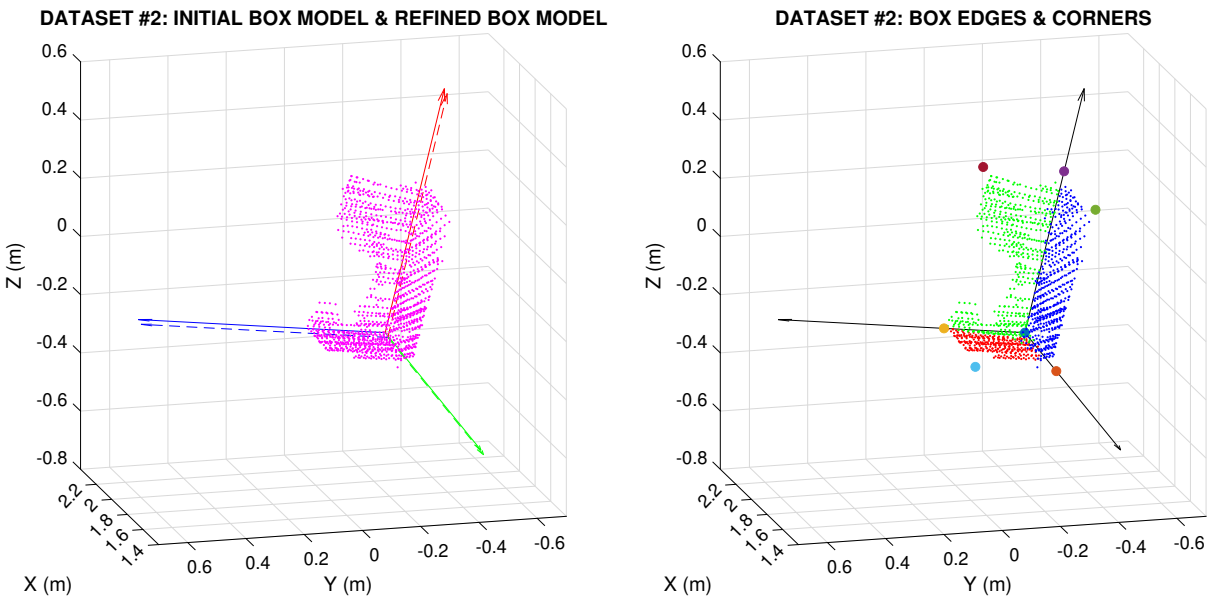

Figura 5-7: Raffinamento del modello del bersaglio e individuazione degli angoli.

Dato che il raffinamento è avvenuto in modo graduale e progressivo, le modifiche apportate agli elementi del modello con ogni data iterazione sono state molto piccole. Il valore di avvio fornito al solutore era infatti $\theta_0 = 0°$ nella fase di rotazione e $\delta_0 = 0.001$ m in quella di traslazione. La massima distanza ammissibile tra valore di avvio e soluzione era pari a $\pm 0.05°$ nel primo caso e a ± 0.001 m nel secondo. Questi parametri sono stati scelti dopo aver verificato il comportamento dell'algoritmo in ognuna delle fasi, utilizzando diversi dataset.

La Figura 5-8 mostra il risultato che si ottiene utilizzando la (2.15) per proiettare sull'immagine termica i punti tridimensionali $\mathbf{q}_i$ estratti dal modello del bersaglio. La posizione di questi punti è indicata da un cerchio, mentre quella dei punti individuati dal rilevatore di Harris-Stephens è indicata da una croce del colore corrispondente. Il minimo errore di riproiezione medio ottenuto risolvendo un problema PnP con 7 punti validi è pari a 1.1734 pixel.
Il fatto che ogni angolo sia stato cercato solo all'interno di una specifica porzione dell'immagine ha impedito al rilevatore di interpretare erroneamente altri elementi del bersaglio o dello sfondo (tra cui i riflessi termici della scatola sulle superfici circostanti) come punti di interesse.

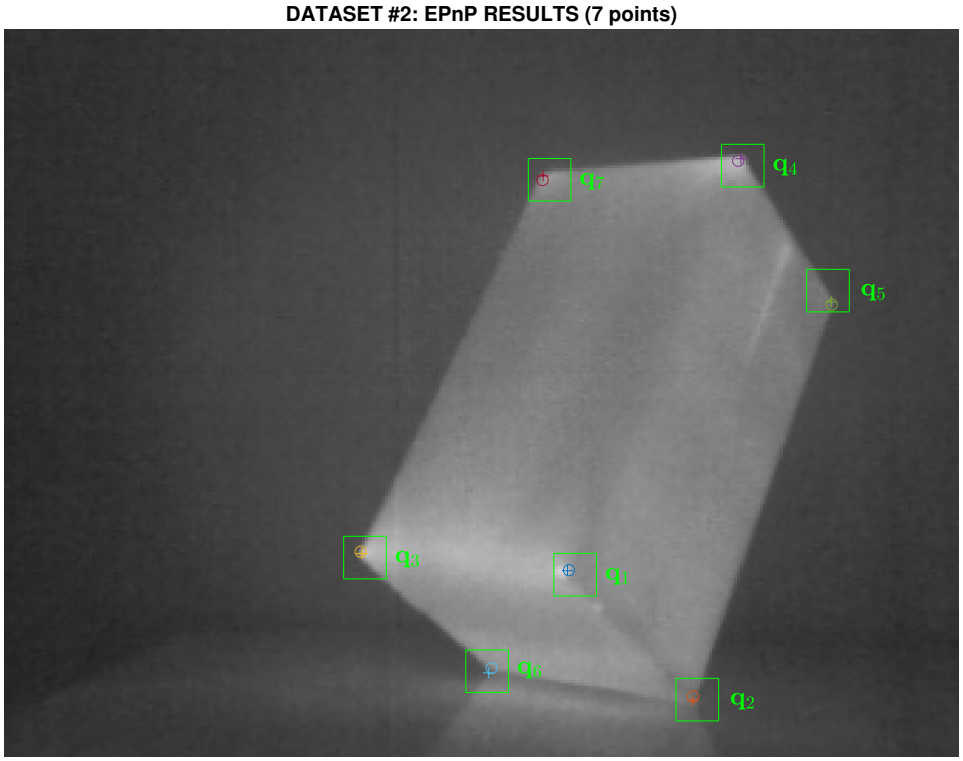

Figura 5-8: Riproiezione dei punti 3–D utilizzando i risultati forniti dall'algoritmo EPnP.

I risultati ottenuti sono stati testati invertendo la (2.15) per proiettare un'immagine termica sulla *point cloud* a cui è associata, in modo da verificare la corrispondenza tra la forma ed il colore dei vari elementi presenti sulla scena. Questa operazione di *sensor fusion* rappresenta anche la base del processo di mappatura termica, quindi la validità dei parametri ricavati in questa fase è stata ulteriormente confermata in seguito. La Figura 5-9 mostra i risultati ottenuti.

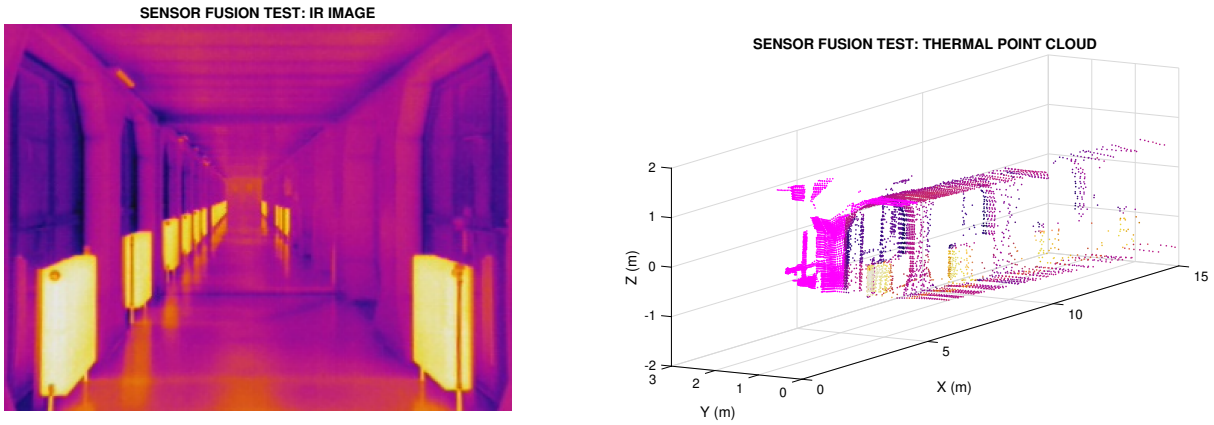

Figura 5-9: Operazione di *sensor fusion* per testare i risultati del processo di calibrazione.

5.3 Mappatura termica

In questa sezione viene descritta l'implementazione delle diverse fasi del processo di mappatura termica, evidenziando i test effettuati ed i risultati ottenuti in ognuna di esse. Come è avvenuto nel caso della calibrazione intrinseca e della calibrazione intrinseca, l'algoritmo utilizzato è stato sviluppato ed eseguito in ambiente MATLAB.

5.3.1 Sensor Fusion

Prima di eseguire la *sensor fusion* è necessario sincronizzare il dataset prodotto dal LiDAR con il dataset prodotto dalla termocamera, per fare in modo che le *point cloud* e le immagini termiche acquisite nello stesso istante siano correttamente associate tra loro.

Mentre le immagini termiche sono disponibili come file singoli, la sequenza di *point cloud* deve essere estratta da un file *rosbag* selezionando il *topic* appropriato. In origine ogni elemento della sequenza è rappresentato da un singolo messaggio ROS di tipo `livox_ros_driver/CustomMsg` prodotto dal LiDAR. La configurazione del pacchetto `livox_ros_driver` stabilisce la frequenza con cui vengono pubblicati questi messaggi e quindi anche la quantità di *point cloud* acquisite lungo la traiettoria. Se nell'intervallo di tempo in cui il LiDAR è attivo la velocità di movimento del rover è costante, imporre una certa separazione temporale tra i messaggi estratti equivale ad imporre una minima separazione spaziale due *point cloud* successive.
I dati prodotti dal LiDAR potrebbero anche essere stati preventivamente elaborati da pacchetti aggiuntivi, ad esempio per effettuare operazioni di localizzazione e mappatura in tempo reale. In questo caso i messaggi prodotti saranno di tipo `sensor_msgs/PointCloud2` e il numero di *point cloud* a disposizione dipenderà sia dalla frequenza di acquisizione che dalla velocità di esecuzione degli algoritmi utilizzati.
Le sequenze di coordinate tridimensionali contenute in questi messaggi sono convertite in oggetti di tipo `pointCloud` (ognuno dei quali rappresenta una singola *point cloud* acquisita dal LiDAR) per consentirne l'elaborazione in ambiente MATLAB invece che in ambiente ROS.

La sequenza di *point cloud* viene sincronizzata con la sequenza di immagini termiche confrontando i marcatori di tempo, o *timestamp*, associati ai relativi elementi. Nel caso delle *point cloud* le informazioni temporali sono espresse in formato numerico *Unix Time* o *Posix Time* (numero di secondi non intercalari trascorsi dalle 00:00:00 del 01/01/1970) e vengono estratte dal file *rosbag* insieme ai relativi messaggi. Nel caso delle immagini termiche, invece, le informazioni temporali in formato `yyyyMMdd_HHmmss` (dove `y` indica l'anno, `M` il mese, `d` il giorno, `H` le ore, `m` i minuti e `s` i secondi) costituiscono il nome stesso del file.
L'elaborazione da parte di pacchetti aggiuntivi in ambiente ROS, tuttavia, può sovrascrivere le informazioni temporali relative alle *point cloud*, impedendo così di associarle alle corrispondenti immagini termiche. In questo caso è necessario ridefinire tutti gli N_L marcatori temporali t_i^L nel dataset del LiDAR, ponendo

$$\tilde{t}_i^L = t_i^L + \Delta t = t_i^L - t_1^L + t_0^C \qquad (i = 1, \dots, N_L) \tag{5.1}$$

dove t_1^L si riferisce alla prima *point cloud* disponibile e t_0^C si riferisce all'immagine termica con la quale è associata. Generalmente per effettuare questa associazione si prende come riferimento l'istante in cui il rover inizia a percorrere la sua traiettoria.

Quando il dataset acquisito dal LiDAR contiene più elementi di quello acquisito dalla termocamera, per ogni immagine termica viene indicata la *point cloud* corrispondente. Quando invece è il dataset acquisito dalla termocamera a contenere più elementi, per ogni *point cloud* viene indicata l'immagine termica corrispondente. In entrambi i casi, la corrispondenza viene individuata minimizzando la differenza assoluta tra i rispettivi *timestamp* in formato numerico.

La mappa di colore è fornita direttamente dalla rappresentazione dell'immagine sotto forma di una matrice 256×336 di triplette RGB. La mappa di temperatura, invece, si ottiene utilizzando il software contenuto nel pacchetto *FLIR Atlas SDK* per estrarre i valori di intensità dei singoli pixel e convertirli in valori di temperatura apparente.
Poiché gli oggetti di tipo `pointCloud` permettono di attribuire ai propri elementi un colore ma non una temperatura, questo parametro viene assegnato alla proprietà `Intensity` di ogni dato punto. In questo modo, qualsiasi operazione eseguita sulla *point cloud* termica coinvolgerà anche i relativi valori di temperatura.

5.3.2 Registrazione e unione delle Point Cloud

Le parti dell'algoritmo relative alla ricostruzione della traiettoria e alla registrazione delle *point cloud* sono state testate separatamente prima di sviluppare quelle successive. I dataset utilizzati per eseguire i test (denominati `Interno1` e `Interno2`) sono stati acquisiti nei corridoi interni di un edificio, cioè in un ambiente parzialmente strutturato: il luogo presenta infatti una geometria globalmente riconoscibile, ma può contenere anche un certo numero di ostacoli dei quali non si conoscono forma e dimensione. La Figura 5-10 mostra l'aspetto di uno degli ambienti di test se osservato nel campo della luce visibile (a sinistra) e in quello dell'infrarosso (a destra).

Figura 5-10: Aspetto dell'ambiente di test nel visibile e nell'infrarosso (dataset `Interno2`).

Per quanto riguarda la prima versione dell'algoritmo, questi test hanno avuto esito positivo solo in alcune sezioni ristrette dell'ambiente attraversato, evidenziando le principali limitazioni delle strategie utilizzate. In particolare, è emerso come i risultati dell'operazione di SLAM 2–D siano parzialmente determinati dalla possibilità di incontrare con sufficiente frequenza le condizioni di *loop closure* necessarie per ottimizzare il *pose graph* e ridurre la deriva della traiettoria stimata. Ciò significa che, quando il percorso seguito dal rover supera una certa lunghezza ma non prevede di attraversare più volte una stessa area, le possibilità di ricostruire la traiettoria con precisione sufficiente sono fortemente limitate. La Figura 5-11 mostra il miglior risultato ottenuto, nel caso di un corridoio di soli 14 metri di lunghezza.

La funzione che implementa il metodo ICP, inoltre, si è rivelata molto sensibile alla disposizione regolare di alcuni elementi (come porte e finestre) presenti nell'ambiente di test.
Un criterio di registrazione basato unicamente su una minimizzazione della distanza tra punti è adatto all'elaborazione di dati raccolti in ambienti scarsamente strutturati, mentre può portare a risultati errati se nelle *point cloud* sono presenti schemi ripetitivi che risultano dominanti nel calcolo delle distanze da minimizzare. Questo comportamento, unito all'impossibilità di ricavare un'adeguata stima iniziale della trasformazione, ha impedito alla funzione di registrare in modo corretto molte delle *point cloud* fornite. Se fossero stati utilizzati dei dataset raccolti in ambienti

meno strutturati questa problematica probabilmente non si sarebbe presentata, ma la difficoltà di ricavare dalle *point cloud* sezioni bidimensionali sufficientemente distintive avrebbe ostacolato ulteriormente la ricostruzione della traiettoria.

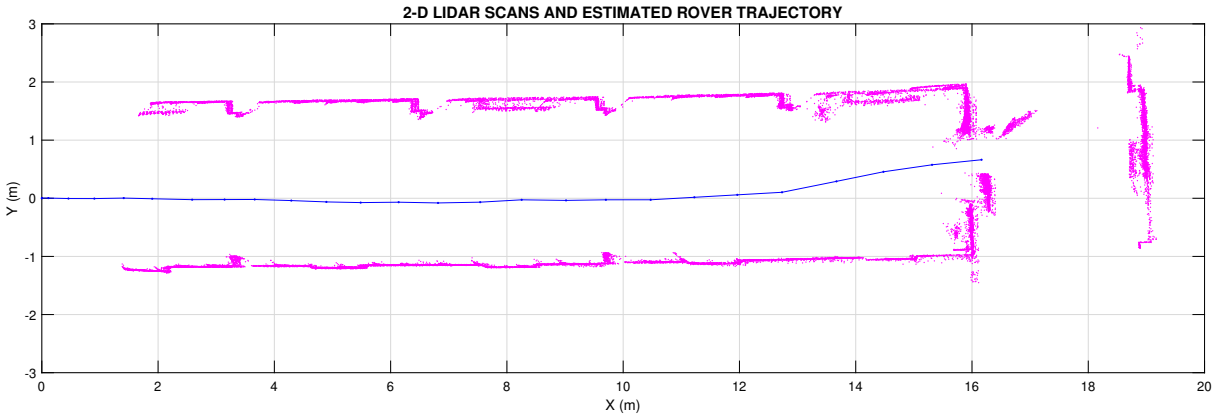

Figura 5-11: Risultato della registrazione basata su SLAM 2–D (dataset `Interno1`).

Poiché le strategie inizialmente adottate non hanno prodotto risultati accettabili, nella seconda versione dell'algoritmo è stato utilizzato il pacchetto `livox_horizon_loam` per implementare il metodo LOAM in ambiente ROS. Questo software sfrutta il solutore *Ceres-Solver* per arrivare a registrare le *point cloud* senza dover imporre condizioni sulla loro geometria, semplificando in modo sensibile la struttura delle funzioni di odometria e mappatura.

Nonostante il metodo LOAM venga implementato in tempo reale in ambiente ROS, la creazione della mappa di distribuzione di temperatura avviene sempre a posteriori in ambiente MATLAB. I dati sono quindi forniti all'algoritmo di mappatura termica all'interno di un file *rosbag*, insieme a quelli prodotti da tutti gli altri sensori eventualmente presenti a bordo del rover.
Il pacchetto `livox_horizon_loam` riceve i messaggi di tipo `livox_ros_driver/CustomMsg` che contengono le singole *point cloud* generate dal LiDAR. I messaggi che pubblica contengono invece le *point cloud* elaborate, prive delle eventuali distorsioni, insieme alle trasformazioni rigide 3–D che consentono di registrarle rispetto al sistema di riferimento iniziale della traiettoria.
Questo pacchetto utilizza il formato `sensor_msgs/PointCloud2` per le *point cloud* e il formato `nav_msgs/Odometry` per le trasformazioni. I relativi *topic* sono denominati `/velodyne_cloud_2` e `/aft_mapped_to_init_high_frec` rispettivamente.

Variables - bag.AvailableTopics

bag.AvailableTopics

bag.AvailableTopics

	1 NumMessages	2 MessageType	3 MessageDefinition	4
1 /aft_mapped_to_init_high_frec	3861	nav_msgs/Odometry	uint32 Seq Time Stamp char...	
2 /livox/imu	81654	sensor_msgs/Imu	uint32 Seq Time Stamp char...	
3 /piksi_multi_base_station/navsatfix_best_fix	4066	sensor_msgs/NavSatFix	uint32 Seq Time Stamp char...	
4 /velodyne_cloud_2	3861	sensor_msgs/PointCloud2	uint32 Seq Time Stamp char...	
5 /velodyne_cloud_registered	1931	sensor_msgs/PointCloud2	uint32 Seq Time Stamp char...	
6				

Figura 5-12: File *rosbag* prodotto dal pacchetto che implementa il metodo LOAM.

Anche se non costituisce un requisito per un'efficace implementazione del metodo LOAM, avere a disposizione una stima indipendente del moto (come quella fornita dalla piattaforma inerziale presente nel LiDAR) può essere utile per gestire gli effetti di spostamenti particolarmente rapidi

o irregolari. Le informazioni relative all'orientamento possono essere utilizzate per agevolare la registrazione delle *point cloud* acquisite, mentre le misure di accelerazione contribuiscono alla compensazione delle distorsioni aggiuntive introdotte dalle brusche variazioni di velocità.

Inizialmente questa seconda versione dell'algoritmo è stata testata utilizzando gli stessi dataset impiegati per verificare il funzionamento di quella precedente. In seguito i test sono stati ripetuti utilizzando dei dataset (denominati `Esterno1` ed `Esterno2`) acquisiti all'aperto, in un ambiente scarsamente strutturato e dalla morfologia più irregolare. In entrambi i casi il ricorso al metodo LOAM ha prodotto risultati positivi, poiché le *point cloud* sono state registrate con precisione soddisfacente. La Figura 5-13 mostra i risultati ottenuti, evidenziando la corrispondenza tra la traiettoria dal rover e la traccia del sentiero visibile nelle *point cloud* termiche.

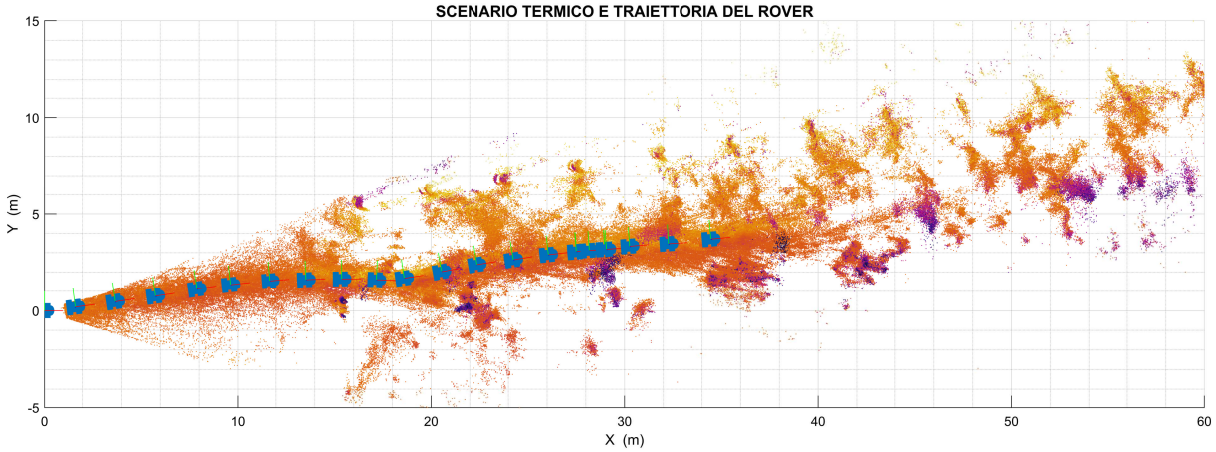

Figura 5-13: Risultato della registrazione basata su LOAM (dataset `Esterno2`).

5.3.3 Creazione della mappa di distribuzione di temperatura

I punti tridimensionali erano stati associati ai nodi della struttura *OcTree* utilizzando gli stessi indici che li identificavano nella *point cloud* di provenienza, ovvero lo scenario termico. Tutte le proprietà dei punti considerati, compresa la loro temperatura apparente, possono quindi essere recuperate utilizzando tali indici. La Figura 5-14 mostra come è organizzata la struttura *OcTree* creata dalla funzione di decomposizione spaziale: all'ultimo livello di suddivisione, i discendenti di ogni nodo sono gli indici dei punti contenuti nel *voxel* corrispondente.

Variables - otObj(8).group

otObj(8).group

otObj(8).group

Fi...	child	groupcenter	cubelength	grouptouch
1	[161738;161739;161741;161743;161746;161747]	[10.1282,0.0425,-1.3741]	0.2736	[2;129;130;776;777;891;892]
2	[160371;160374;160380;160381;160386;160389;160390]	[10.4017,0.0425,-1.3741]	0.2736	*11x1 double*
3	[160370;160376;160378;160383;160385;160388;160391]	[10.6753,0.0425,-1.3741]	0.2736	*11x1 double*
4	*16x1 double*	[10.9489,0.0425,-1.3741]	0.2736	*11x1 double*
5	[156170;156265;156269;156280;156284;159004;159007]	[11.2224,0.0425,-1.3741]	0.2736	*11x1 double*
6	*21x1 double*	[11.4960,0.0425,-1.3741]	0.2736	*11x1 double*
7	*21x1 double*	[11.7696,0.0425,-1.3741]	0.2736	*11x1 double*
8	*24x1 double*	[12.0431,0.0425,-1.3741]	0.2736	*11x1 double*
9	*32x1 double*	[12.3167,0.0425,-1.3741]	0.2736	*11x1 double*
10	*24x1 double*	[12.5903,0.0425,-1.3741]	0.2736	*11x1 double*
11	*29x1 double*	[12.8639,0.0425,-1.3741]	0.2736	*11x1 double*
12	*33x1 double*	[13.1374,0.0425,-1.3741]	0.2736	*11x1 double*

Figura 5-14: Struttura *OcTree* creata dalla funzione di decomposizione spaziale.

Per accelerare la creazione dei *voxel* termici e rendere la mappa di distribuzione di temperatura meno sensibile al rumore presente nelle *point cloud* termiche, si prendono in considerazione solo i nodi terminali a cui sono associati almeno 10 punti. I punti che presentano `NaN` come valore di temperatura (cioè quelli relativi a porzioni dell'ambiente che non ricadevano nel campo di vista della termocamera quando una *point cloud* è stata acquisita) vengono scartati a priori. Le figure seguenti mostrano i risultati ottenuti per alcuni dei dataset disponibili.

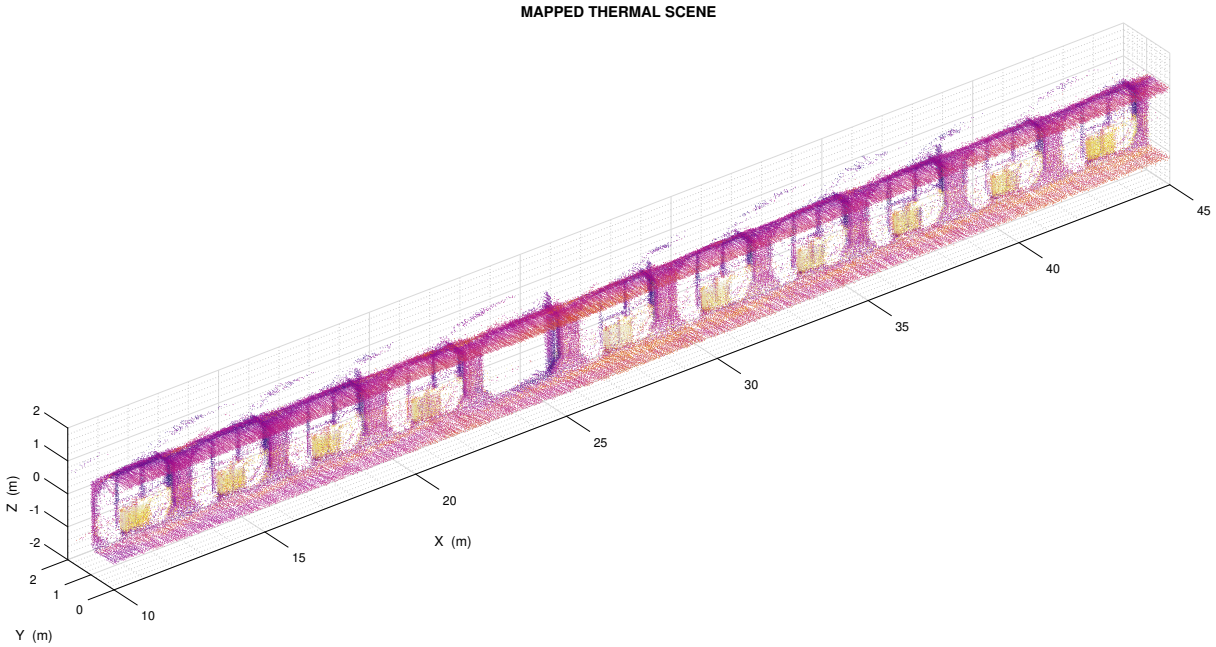

Figura 5-15: Scenario termico mappato (dataset `Interno2`).

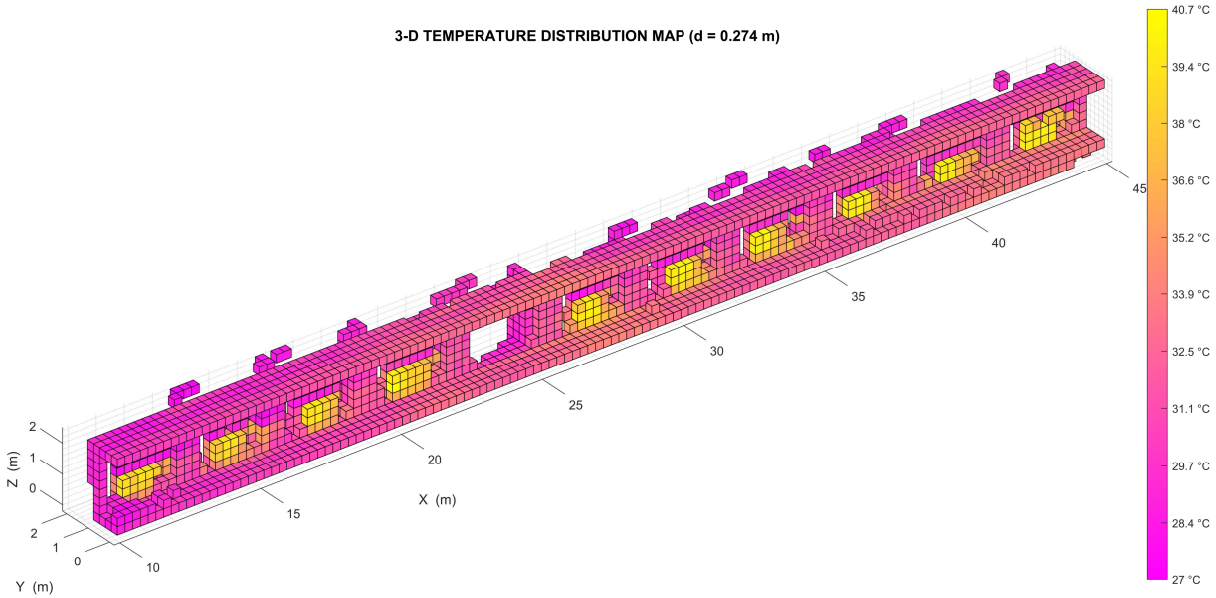

Figura 5-16: Mappa di distribuzione di temperatura (dataset `Interno2`).

Il parametro τ_i associato ad un particolare *voxel* termico identifica una tripletta RGB all'interno della *colormap* usata per rappresentare la mappa di distribuzione di temperatura, che in questo caso è stata scelta per assomigliare il più possibile a quella delle immagini termiche originali. La *colorbar* generata automaticamente insieme alla mappa permette di associare rapidamente ogni sfumatura di colore al relativo valore di temperatura apparente.

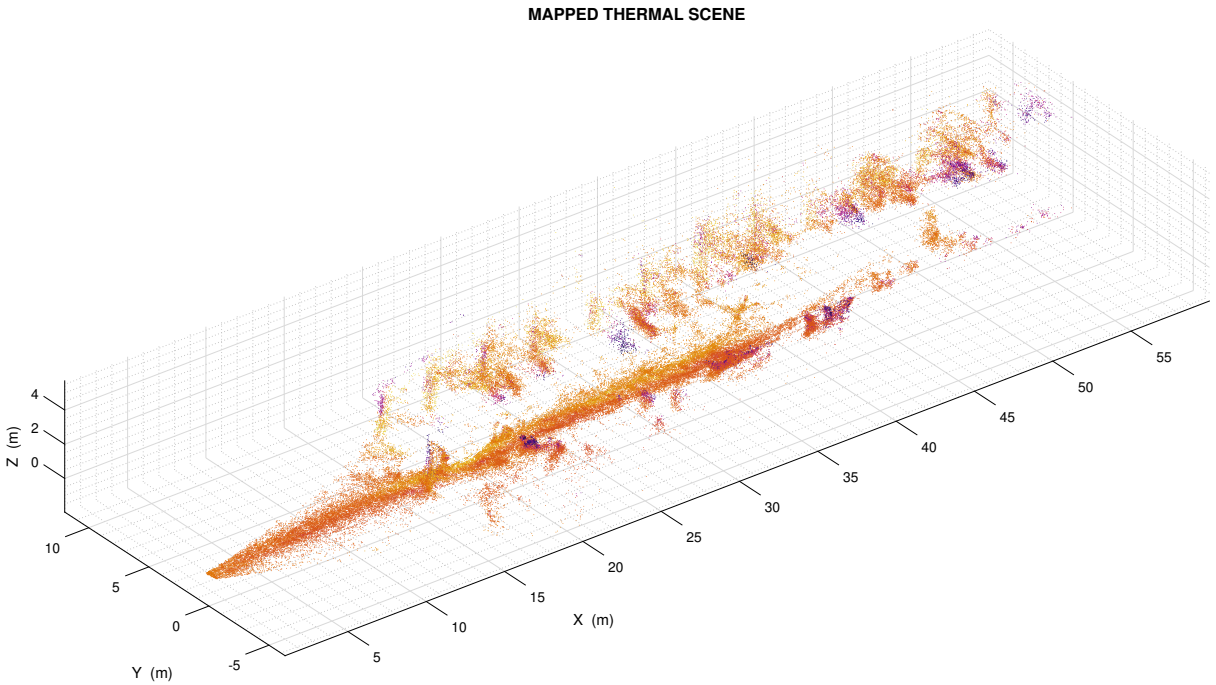

Figura 5-17: Scenario termico mappato (dataset `Esterno2`).

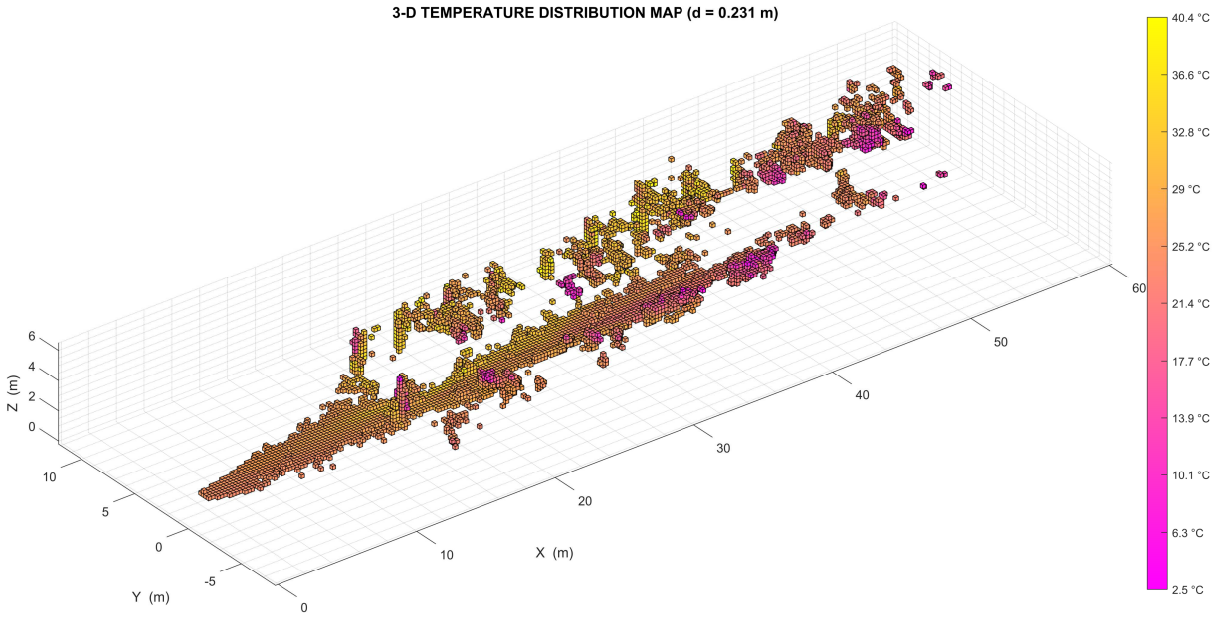

Figura 5-18: Mappa di distribuzione di temperatura (dataset `Esterno2`).

5.4 Analisi dei risultati

I dati acquisiti nell'ambiente più strutturato (dataset `Interno2`) possono anche essere utilizzati per determinare quanto sono accurate le informazioni contenute nella mappa di distribuzione di temperatura, evidenziando l'influenza dei vari parametri (come la dimensione minima dei *voxel* o il minimo numero di punti che devono contenere) che ne governano la creazione.

Confrontando la Figura 5-15 con la Figura 5-16 si nota come la geometria globale dell'ambiente considerato rimanga chiaramente distinguibile tanto nello scenario termico quanto nella mappa di distribuzione di temperatura. Inoltre, viene evidenziata la presenza di elementi termicamente

distintivi (i caloriferi visibili nella Figura 5-10) caratterizzati da una disposizione regolare lungo l'asse X del sistema di riferimento dell'ambiente.
Questi elementi possono essere riconosciuti in base alla loro temperatura, definendo dei punti di riferimento che permettono di stabilire se la mappa creata fornisce una rappresentazione valida dell'ambiente attraversato dal rover. Più in generale, gli stessi riferimenti possono costituire dei *landmark* da utilizzare per applicazioni legate alla navigazione autonoma.

Lo scopo dell'analisi sarà quindi quello di isolare i *voxel* termici che rientrano in un determinato intervallo di temperatura, riunirli in gruppi distinti (*cluster*) e confrontare la loro posizione con le misurazioni effettuate sul posto. Avere a disposizione una mappa di distribuzione di temperatura permette di operare su un numero relativamente basso di *voxel* termici, invece che su un numero molto maggiore di punti disposti in modo irregolare attraverso l'intero volume.

L'analisi avverrà prendendo in considerazione due casi:

- Mappa "a bassa risoluzione" ($\sim$3.7 *voxel*/m) con $D = 0.30$ m e $d = 0.274$ m.
- Mappa "ad alta risoluzione" ($\sim$7.3 *voxel*/m) con $D = 0.15$ m e $d = 0.137$ m.

In entrambi i casi d è la dimensione effettiva dei *voxel* termici, stabilita dalla (4.9) e dalla (4.10) nota la condizione di arresto D per la funzione di decomposizione spaziale.
In generale la risoluzione ottimale per la mappa di distribuzione di temperatura è determinata dalla dimensione delle strutture e degli oggetti presenti nell'ambiente: con una risoluzione troppo alta il numero di *voxel* ottenuti sarà paragonabile a quello dei punti, mentre con una risoluzione troppo bassa la discretizzazione comporterebbe una perdita di informazioni eccessiva.

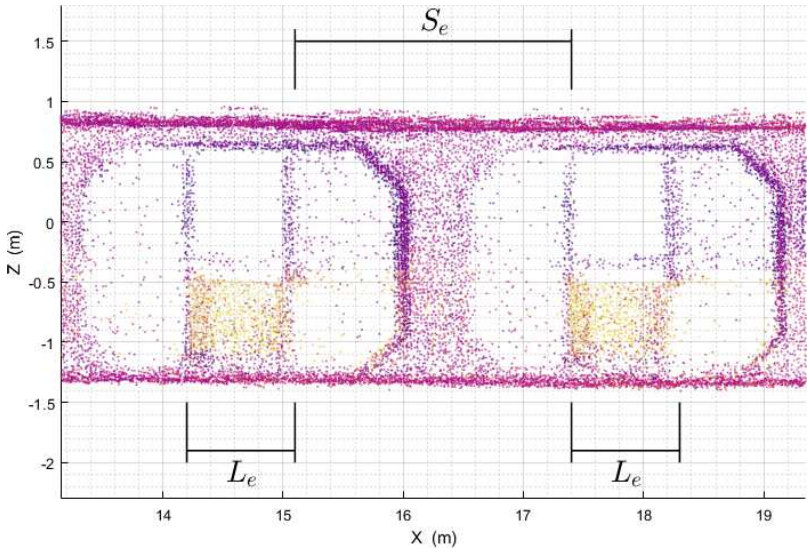

Figura 5-19: Dimensione e distanziamento dei riferimenti termici.

Siano quindi $T_{th} = 37.5°$C la temperatura di soglia per il riconoscimento degli elementi termici di interesse, $L_e = 0.9$ m la loro dimensione e $S_e = 2.3$ m la loro distanza reciproca. L'algoritmo che esegue l'analisi svolge le seguenti operazioni sui *voxel* termici (4.15) appartenenti alla mappa di distribuzione di temperatura:

1. Estrarre tutti i *voxel* con $T_i^v \geq T_{th}$ presenti nella mappa.
2. Eliminare i *voxel* estratti che non rientrano nella regione da analizzare.
3. Definire un intervallo di analisi $[X_0, X_F]$ con $X_0 \leq \min(X_i^v)$ e $X_F \geq \max(X_i^v)$.

4. Definire una finestra di ricerca $\mathcal{F} = [X_1, X_2]$ centrata su X_0 tale che $X_2 - X_1 > L_e$.
5. Se i *voxel* estratti contenuti nella finestra $\mathcal{F}$ sono meno di 4:
   - Usare $X_3 = \frac{1}{2}(X_2 + X_1) + L_e + S_e$ come centro di una nuova finestra di ricerca.
6. Se i *voxel* estratti presenti nella finestra $\mathcal{F}$ sono almeno 4:
   - Recuperare le coordinate $\mathbf{w}_i^c = \{X_i^c, Y_i^c, Z_i^c\}^\mathsf{T}$ dei loro centroidi.
   - Mediarle per ricavare la posizione $\mathbf{w}_k^g = \{X_k^g, Y_k^g, Z_k^g\}^\mathsf{T}$ del *cluster* individuato.
   - Usare $X_3 = X_k^g + L_e + S_e$ come centro di una nuova finestra di ricerca.

Le operazioni da 4 a 6 sono ripetute fino a coprire l'intero intervallo di analisi precedentemente definito. A questo punto, le distanze tra i vari elementi possono essere ricavate confrontando le coordinate X_k^g dei centroidi dei *cluster* consecutivi.

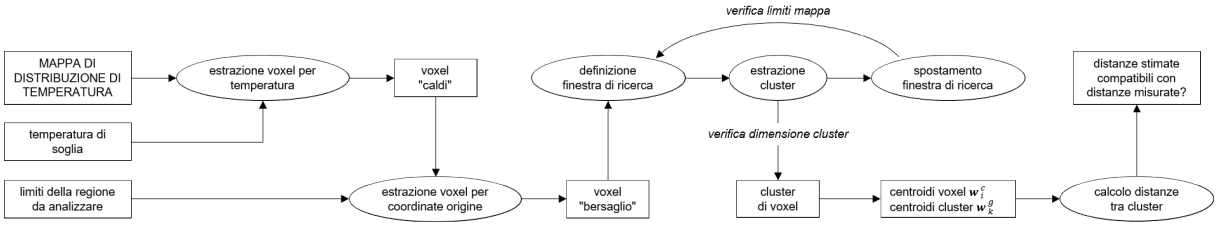

Figura 5-20: Modalità di analisi della mappa di distribuzione di temperatura.

Di seguito sono riportati i risultati dell'analisi effettuata. Le figure mostrano i *voxel* estratti ed i *cluster* individuati, mentre le variabili prodotte dall'algoritmo evidenziano in quali casi è stata riscontrata una compatibilità tra le distanze stimate e quelle misurate. Si ammetterà comunque uno scarto massimo pari a $\pm d$, per tenere conto della discretizzazione effettuata nel passare dallo scenario termico alla mappa di distribuzione di temperatura.

Le variabili prodotte ad ogni esecuzione dall'algoritmo che analizza la mappa di distribuzione di temperatura indicano, nell'ordine:

- `From`: indice del *cluster* considerato.
- `To`: indice del *cluster* precedente.
- `MinimumDistance`: minima distanza possibile $S_{min} = \Delta X^g - d$ tra i *cluster* considerati.
- `EstimatedDistance`: distanza stimata ΔX^g tra i *cluster* considerati.
- `MaximumDistance`: massima distanza possibile $S_{max} = \Delta X^g + d$ tra i *cluster* considerati.
- `ExpectedDistance`: distanza attesa $S^\star$ tra i *cluster* considerati.
- `IsCompatible`: contiene 1 se $S_{min} \leq S^\star \leq S_{max}$ e contiene 0 in caso contrario.

Si noti, ad esempio dalla Figura 5-23, come la distanza attesa non sia sempre $S^\star = S_e + L_e$ a causa delle finestre di ricerca in cui non sono stati individuati *cluster* validi.

I risultati ottenuti nel caso "a bassa risoluzione" mostrano come le distanze stimate siano tutte compatibili con quelle attese. Nel caso "ad alta risoluzione" si può invece notare come la stima per eccesso dell'estensione di un singolo *cluster* (ovvero il secondo sul lato destro) sia sufficiente per influenzare in modo sensibile il calcolo delle distanze: in tutti gli altri casi, infatti, la condizione di compatibilità non viene soddisfatta a causa di scarti inferiori a 0.01 m.

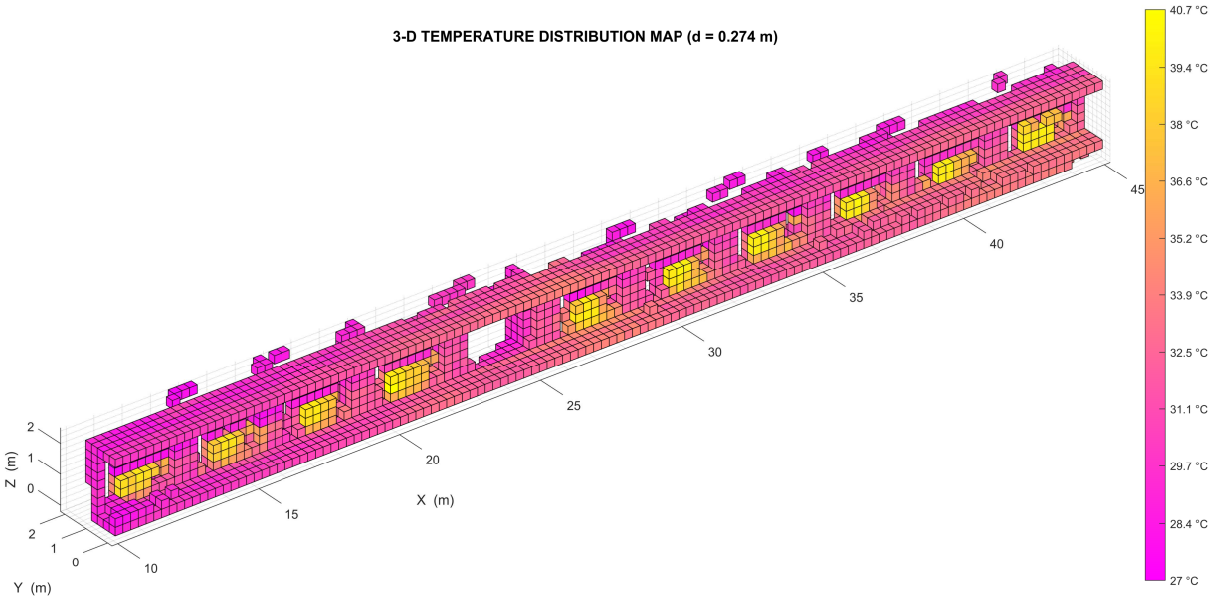

Figura 5-21: Mappa di distribuzione di temperatura (bassa risoluzione, lato sinistro).

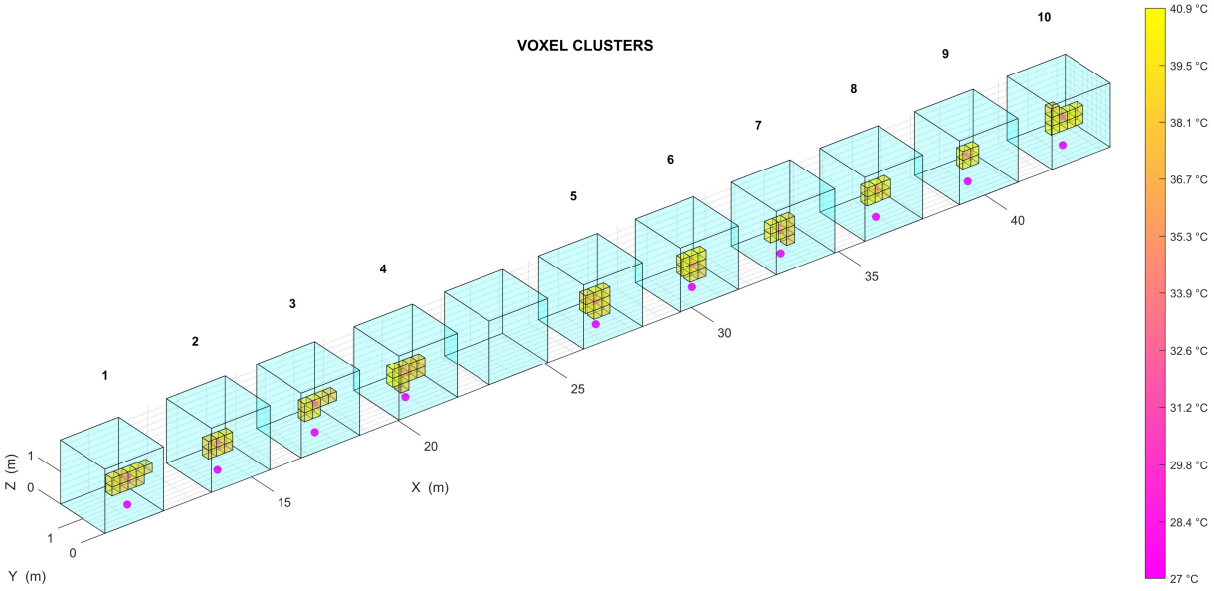

Figura 5-22: Individuazione dei *cluster* di *voxel* (bassa risoluzione, lato sinistro).

Fields	From	To	MinimumDistance	EstimatedDistance	MaximumDistance	ExpectedDistance	IsCompatible
1	2	1	2.7965	3.0701	3.3436	3.2000	1
2	3	2	3.0549	3.3284	3.6020	3.2000	1
3	4	3	2.8117	3.0853	3.3588	3.2000	1
4	5	4	6.2047	6.4783	6.7519	6.4000	1
5	6	5	3.0093	3.2828	3.5564	3.2000	1
6	7	6	2.7406	3.0142	3.2877	3.2000	1
7	8	7	2.9702	3.2438	3.5173	3.2000	1
8	9	8	2.8725	3.1461	3.4196	3.2000	1
9	10	9	2.9637	3.2372	3.5108	3.2000	1

Figura 5-23: Risultati dell'analisi (bassa risoluzione, lato sinistro).

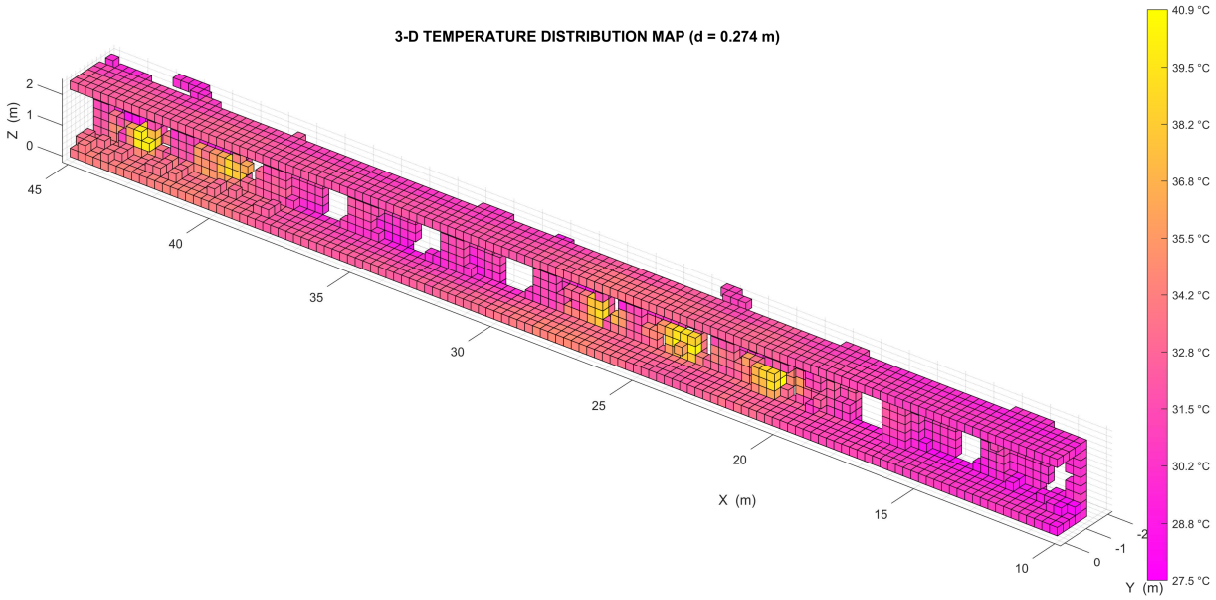

Figura 5-24: Mappa di distribuzione di temperatura (bassa risoluzione, lato destro).

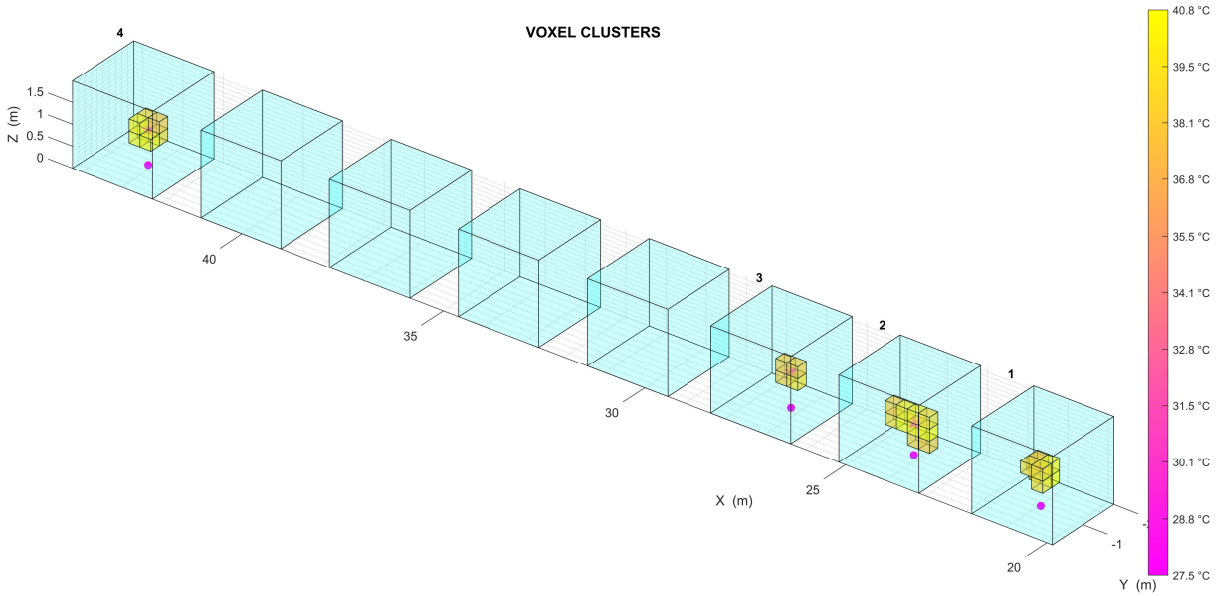

Figura 5-25: Individuazione dei *cluster* di *voxel* (bassa risoluzione, lato destro).

Fields	From	To	MinimumDistance	EstimatedDistance	MaximumDistance	ExpectedDistance	IsCompatible
1	2	1	2.9204	3.1940	3.4675	3.2000	1
2	3	2	2.7904	3.0640	3.3376	3.2000	1
3	4	3	15.6132	15.8867	16.1603	16.0000	1

Figura 5-26: Risultati dell'analisi (bassa risoluzione, lato destro).

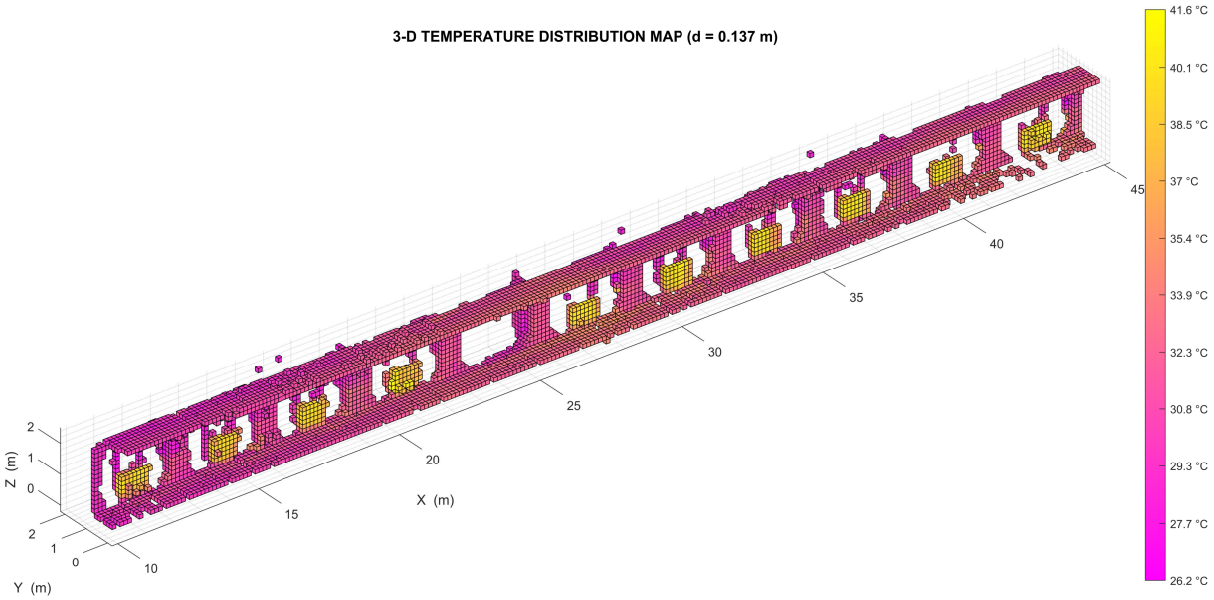

Figura 5-27: Mappa di distribuzione di temperatura (alta risoluzione, lato sinistro).

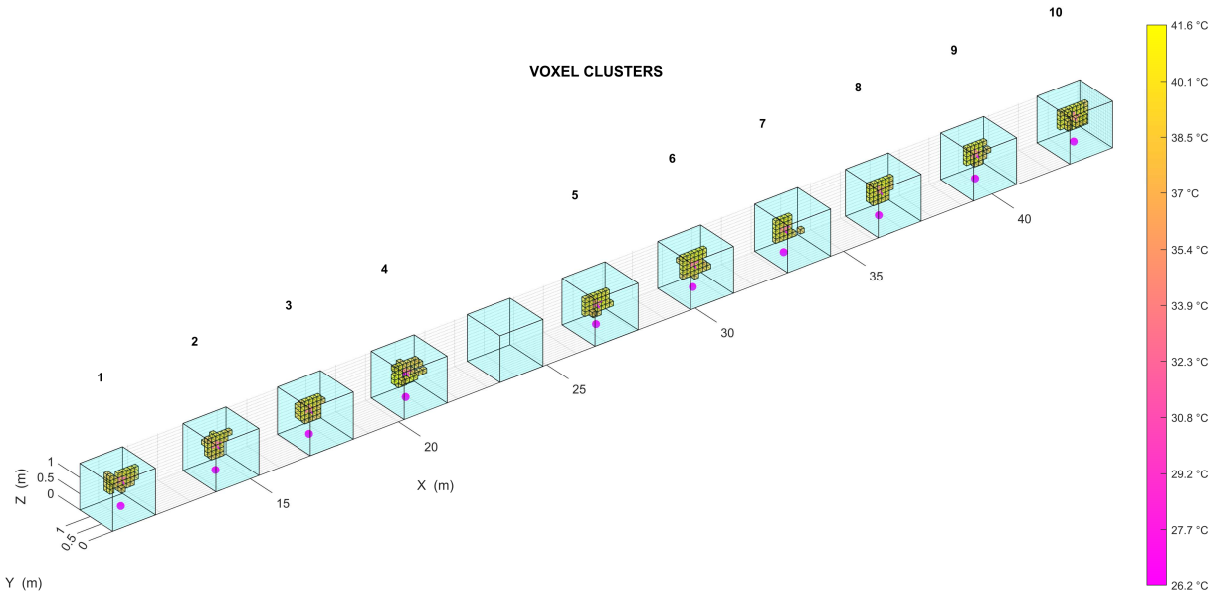

Figura 5-28: Individuazione dei *cluster* di *voxel* (alta risoluzione, lato sinistro).

Fields	From	To	MinimumDistance	EstimatedDistance	MaximumDistance	ExpectedDistance	IsCompatible
1	2	1	3.0469	3.1837	3.3205	3.2000	1
2	3	2	3.0215	3.1583	3.2951	3.2000	1
3	4	3	3.1008	3.2376	3.3744	3.2000	1
4	5	4	6.2836	6.4204	6.5572	6.4000	1
5	6	5	3.1320	3.2688	3.4056	3.2000	1
6	7	6	2.9203	3.0571	3.1938	3.2000	0
7	8	7	3.0766	3.2134	3.3502	3.2000	1
8	9	8	3.1036	3.2404	3.3772	3.2000	1
9	10	9	3.1623	3.2991	3.4359	3.2000	1

Figura 5-29: Risultati dell'analisi (alta risoluzione, lato sinistro).

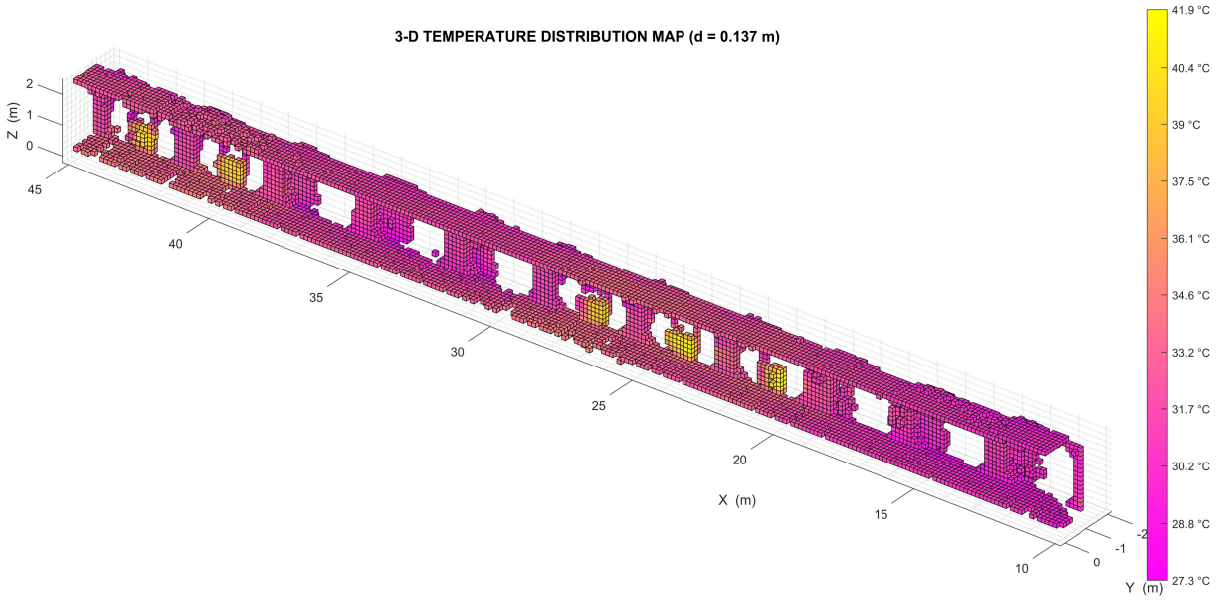

Figura 5-30: Mappa di distribuzione di temperatura (alta risoluzione, lato destro).

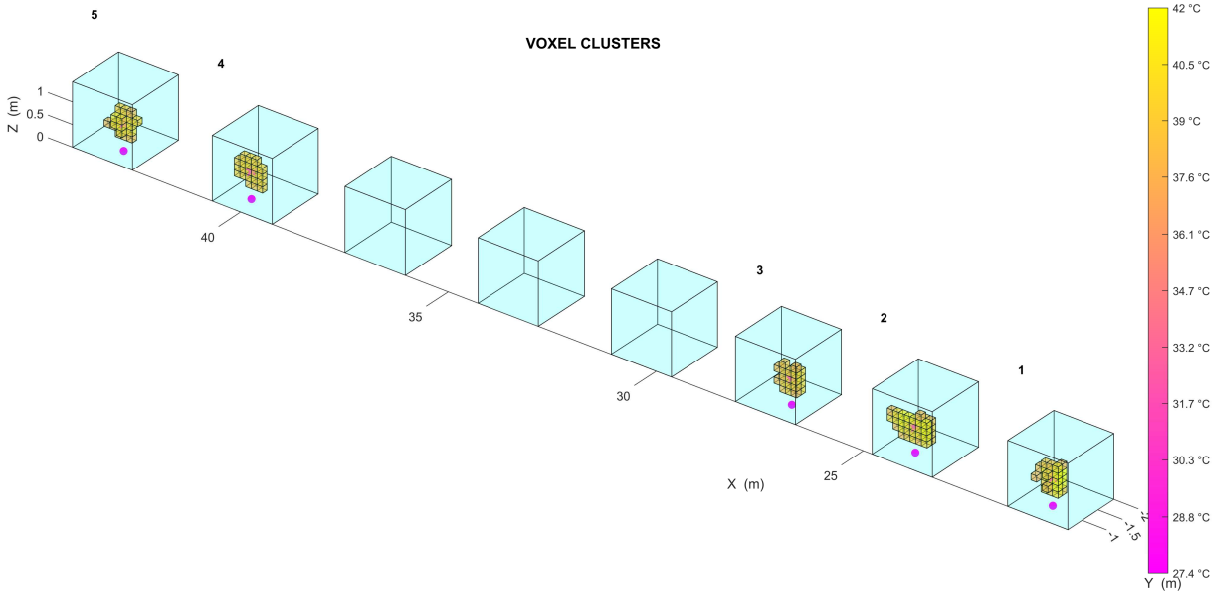

Figura 5-31: Individuazione dei *cluster* di *voxel* (alta risoluzione, lato destro).

Fields	From	To	MinimumDistance	EstimatedDistance	MaximumDistance	ExpectedDistance	IsCompatible
1	2	1	3.1672	3.3040	3.4408	3.2000	1
2	3	2	2.8564	2.9932	3.1300	3.2000	0
3	4	3	12.8384	12.9752	13.1119	12.8000	0
4	5	4	2.9218	3.0586	3.1954	3.2000	0

Figura 5-32: Risultati dell'analisi (alta risoluzione, lato destro).

Capitolo 6

Conclusioni

Nei capitolo precedenti è stato descritto un metodo per la mappatura termica basato sulla *sensor fusion* tra un LiDAR 3–D e una termocamera con capacità radiometriche. I due strumenti sono stati calibrati e quindi installati a bordo del rover MORPHEUS.
L'obiettivo era quello di creare una rappresentazione tridimensionale discreta della distribuzione di temperatura apparente nell'ambiente attraversato dal rover, nonché di stabilire se tale mappa rappresenti in modo adeguato l'ambiente stesso. L'algoritmo che esegue le necessarie operazioni è stato sviluppato in ambiente MATLAB e successivamente testato con vari dataset.

I test preliminari hanno spinto a scegliere il metodo LOAM come strategia per la ricostruzione della traiettoria e la registrazione delle *point cloud* termiche, poiché il ricorso alla SLAM 2–D e al metodo ICP non aveva prodotto risultati soddisfacenti.

I test svolti all'interno di un edificio hanno mostrato come sia possibile individuare nella mappa di distribuzione di temperatura tutti gli elementi termicamente distintivi presenti nell'ambiente considerato, determinandone correttamente la posizione e la dimensione compatibilmente con la risoluzione adottata. È stato inoltre evidenziato come tale risoluzione non debba essere troppo elevata, per limitare il numero dei *voxel* creati e quindi consentire alle temperature dei punti di essere significativamente mediate al loro interno.

I test svolti all'esterno mostrano invece come l'algoritmo sia in grado di elaborare grandi quantità di punti, producendo mappe molto estese. Sono però emersi anche alcuni limiti degli strumenti utilizzati per acquisire i dati.
La termocamera FLIR Vue Pro R è infatti perfettamente in grado di evidenziare le differenze di temperatura tra i vari punti della scena inquadrata, ma la sua capacità di fornire valori coerenti di temperatura apparente non è ottimale. Ad esempio, dalla Figura 2-7 e poi dalla Figura 5-18 appare chiaro come all'aperto la radiazione solare domini l'ambiente termico, portando a rilevare temperature evidentemente eccessive nei punti più esposti.

Queste limitazioni potrebbero essere superate adottando una differente funzione di conversione tra l'intensità dei pixel e la temperatura apparente. Questa funzione, intervenendo nella fase di *sensor fusion*, dovrebbe tenere conto dei coefficienti radiometrici forniti dalla termocamera ma anche delle misure di distanza e di riflettività (e quindi emissività) fornite dal LiDAR.
In futuro il processo di mappatura potrebbe essere modificato per consentirne l'implementazione in tempo reale, creando progressivamente la mappa di distribuzione di temperatura nello stesso modo in cui vengono aggiornate le *occupancy map* probabilistiche. L'elaborazione dei dati però dovrebbe avvenire interamente in ambiente ROS.

Bibliografia

[1] D. Forsyth e J. Ponce. *"Computer vision: a modern approach"*. Prentice Hall, 2011.

[2] R. Hartley e A. Zisserman. *"Multiple view geometry in computer vision"*. Cambrige University Press, 2003.

[3] M. Vollmer e K.-P. Möllmann. *"Infrared thermal imaging: fundamentals, research and applications"*. WILEY-VCH, 2010.

[4] C. Cunningham. "Improved Prediction of Traversability for Planetary Rovers Using Thermal Imaging". *Carnegie Mellon University*, Dissertations 955, 2017.

[5] P. Fritsche, B. Zeise, P. Hemme e B. Wagner. "Fusion of Radar, LiDAR and Thermal Information for Hazard Detection in Low Visibility Environments". *IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pp. 96–101, 2017.

[6] Z. Zhang. "A Flexible New Technique for Camera Calibration". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, n. 11, pp. 1330–1334, 2000.

[7] Z. Pusztai e L. Hajder. "Accurate Calibration of LiDAR-Camera Systems using Ordinary Boxes". *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 394–402, 2017.

[8] C. Harris e M. Stephens. "A combined corner and edge detector". *Alvey Vision Conference*, vol. 15, n. 50, 1988.

[9] V. Lepetit, F. Moreno-Noguer e P. Fua. "EPnP: An accurate O(n) solution to the PnP problem". *International Journal Of Computer Vision*, vol. 81, n. 2, p. 155, 2009.

[10] Z. Zhang. "Iterative Point Matching for Registration of Free-Form Curves and Surfaces". *International Journal of Computer Vision*, vol. 13, n. 2, pp. 119–152, 1994.

[11] J. Zhang e S. Singh. "LOAM: Lidar Odometry and Mapping in Real-time". *Robotics: Science and Systems*, pp. 1–9, 2014.

[12] D. Meagher. "Geometric Modeling Using Octree Encoding". *Computer Graphics and Image Processing*, vol. 19, n. 2, pp. 129–147, 1982.

[13] J.-Y. Bouguet. "Camera Calibration Toolbox for Matlab". `http://www.vision.caltech.edu/bouguetj/calib_doc/index.html`, 2004.

Appendice A

Codice MATLAB

A.1 Calibrazione estrinseca

Script: ***runBoxCalibration.m***

```
% THERMAL MAPPING BY MEANS OF LIDAR AND THERMAL CAMERA SENSOR FUSION
% EXTRINSIC CALIBRATION OF A 3-D LIDAR AND A THERMAL CAMERA
%
% Davide De Pazzi - davide.depazzi{@studenti.unipd.it, @gmail.com}

% DOWNLOAD "EPnP" FOLDER FROM:
% github.com/cvlab-epfl/EPnP/tree/master/matlab/EPnP

clc, clearvars, close all

disp('EXTRINSIC CALIBRATION OF A 3-D LIDAR AND A THERMAL CAMERA');
disp(' ');
disp('Created by:     Davide De Pazzi');
disp('Last modified:  20/03/2022');
disp(' ');
disp(' ');

%% SCRIPT SETUP

% Set DATA folder path:
dataPath = ('C:\Users\Davide\Documents\Tesi\Dati\Calibrazione');

% Select calibration dataset:
disp(dataPath);
dir(dataPath);
datasetNumber = input('Select dataset number: ', 's');
disp(' ');
disp(' ');
dataFolder = [dataPath, '\dataset', datasetNumber];

% Add other required folders to path:
addpath('BoxCalibration');
addpath('EPnP');
addpath('HelperFunctions');
addpath('Intrinsics');

% (LOAD) Load intrinsic calibration results:
load('cameraParams.mat');
```

```
% Set conventional color sequence for vectors and planes:
rgb = [1 0 0; 0 1 0; 0 0 1];

% Set conventional color sequence for corners:
csq = [0 0.4470 0.7410; 0.8500 0.3250 0.0980;...
    0.9290 0.6940 0.1250; 0.4940 0.1840 0.5560; ...
    0.4660 0.6740 0.1880; 0.3010 0.7450 0.9330; ...
    0.6350 0.0780 0.1840];

%% SCRIPT PARAMETERS

% Point-to-plane distance threshold for plane fitting (m):
params.MaxDistance = 0.01;

% Perpendicularity threshold for plane fitting:
params.PerpThreshold = 0.5;

% Maximum number of iterations for plane fitting:
params.MaxItFit = 200;

% Measured dimensions of target box edges (m):
params.b = 0.295;
params.h = 0.295;
params.d = 0.595;

% Average point-to-plane distance threshold for model refinement (m):
params.DistanceThreshold = 0.0045;

% Maximum number of iterations for model refinement:
params.MaxItRef = 200;

% Solver settings for rotation phase of model refinement (deg):
params.SolverSettings.Rotation.x0 = 0;
params.SolverSettings.Rotation.lb = -0.05;
params.SolverSettings.Rotation.ub = 0.05;

% Solver settings for translation phase of model refinement (m):
params.SolverSettings.Translation.x0 = 0.001;
params.SolverSettings.Translation.lb = -0.001;
params.SolverSettings.Translation.ub = 0.002;

% Window size for IR corner detection (px):
params.dPx = 15;

% (SAVE) Save script parameters as .mat file:
save('Extrinsics\calibrationParameters.mat', 'params');

%% OUTPUT OPTIONS

% Enable/disable input point cloud visualization:
viewInputPointCloud = 1;

% Enable/disable initial and refined box models visualization:
viewBoxModels = 1;

% Enable/disable input IR image visualization:
viewInputImage = 1;
```

```
% Enable/disable IR corners visualization:
viewDetectedCorners = 1;

% Enable/disable reprojection results visualization:
viewReprojectionResults = 1;

%% 1 - LOAD LIDAR DATA

% Find input point cloud file in DATA folder:
pcName = dir([dataFolder, '\*.las']).name;
pcName = [dataFolder, '\', pcName];

% Extract point cloud data:
lasFile = lasFileReader(pcName);
ptCloudIn = readPointCloud(lasFile);
ptCloudIn = pointCloud(ptCloudIn.Location, 'Color', [ ]);
clear pcName lasFile

% (FIGURE) View input point cloud:
nF = 1;
if viewInputPointCloud == 1
    figure(nF)
    subplot(1, 2, 1)
    pcshow(ptCloudIn, 'MarkerSize', 8);
    set(gcf, 'color', 'w');
    set(gca, 'color', 'w');
    set(gca, 'XColor','k');
    set(gca, 'YColor','k');
    set(gca, 'ZColor','k');
    xlabel('X (m)');
    ylabel('Y (m)');
    zlabel('Z (m)');
    title(['DATASET #', datasetNumber, ': INPUT POINT CLOUD'], ...
        'Color', 'k');
end

%% 2 - POINT CLOUD CLUSTERING

% Identify and cluster box walls from input point cloud:
[boxPlanes, boxPointClouds, boxSide] = ...
    pointCloudClustering(ptCloudIn, params.MaxDistance, ...
    params.PerpThreshold, params.MaxItFit);

% Pre-process clustered point cloud planes:
boxPlanePoints = cell(3, 1);
for i = 1 : 3
    % Remove noise:
    boxPointClouds{i, 1} = pcdenoise(boxPointClouds{i, 1}, ...
        'NumNeighbors', 12, 'Threshold', 0.9);
    % Extract 3-D point coordinates:
    boxPlanePoints{i, 1} = boxPointClouds{i, 1}.Location;
end
clear i

% (FIGURE) View clustered point cloud planes:
if viewInputPointCloud == 1
    figure(nF)
    subplot(1, 2, 2);
```

```
    for i = 1 : 3
        pcshow(boxPointClouds{i, 1}, 'MarkerSize', 8);
        hold on
    end
    set(gcf, 'color', 'w');
    set(gca, 'color', 'w');
    set(gca, 'XColor','k');
    set(gca, 'YColor','k');
    set(gca, 'ZColor','k');
    xlabel('X (m)');
    ylabel('Y (m)');
    zlabel('Z (m)');
    title(['DATASET #', datasetNumber, ': CLUSTERED BOX WALLS'], ...
        'Color', 'k');
    clear i
    nF = nF + 1;
end

%% 3 - ITERATIVE MODEL REFINEMENT

% Create and refine box model from clustered planes:
boxEdges = [params.b, params.h, params.d];
[boxModel, oldBoxModel, boxCorners, dAvg] = ...
    iterativeModelRefinement(boxPlanes, boxPlanePoints, boxSide, ...
    params.DistanceThreshold, params.MaxItRef, ...
    params.SolverSettings, boxEdges);

% (FIGURE) View initial and refined box model:
if viewBoxModels == 1
    figure(nF)
    % Box model comparison:
    subplot(1, 2, 1)
    for i = 1 : 3
        % Point cloud without ground plane:
        pcshow(pointCloud(boxPointClouds{i, 1}.Location, 'Color', ...
            ones(size(boxPointClouds{i, 1}.Location, 1), 3) ...
            .* [1 0 1]), 'MarkerSize', 8);
        hold on
    end
    for i = 1 : 3
        % Initial box model:
        quiver3(oldBoxModel.refPoint(1), oldBoxModel.refPoint(2), ...
            oldBoxModel.refPoint(3), oldBoxModel.normals(i, 1), ...
            oldBoxModel.normals(i, 2), oldBoxModel.normals(i, 3), ...
            '--', 'Color', rgb(i, :));
    end
    for i = 1 : 3
        % Refined box model:
        quiver3(boxModel.refPoint(1), boxModel.refPoint(2), ...
            boxModel.refPoint(3), boxModel.normals(i, 1), ...
            boxModel.normals(i, 2), boxModel.normals(i, 3), ...
            'Color', rgb(i, :));
    end
    set(gcf, 'color', 'w');
    set(gca, 'color', 'w');
    set(gca, 'XColor','k');
    set(gca, 'YColor','k');
    set(gca, 'ZColor','k');
```

```
    xlabel('X (m)');
    ylabel('Y (m)');
    zlabel('Z (m)');
    title(['DATASET #', datasetNumber, ': INITIAL BOX MODEL & ', ...
        'REFINED BOX MODEL'], 'Color', 'k');
    % Box edges and corners:
    subplot(1, 2, 2)
    for i = 1 : 3
        % Clustered planes:
        pcshow(boxPointClouds{i, 1}, 'MarkerSize', 8);
        hold on
        % Refined box model:
        quiver3(boxModel.refPoint(1), boxModel.refPoint(2), ...
            boxModel.refPoint(3), boxModel.normals(i, 1), ...
            boxModel.normals(i, 2), boxModel.normals(i, 3), 'k');
    end
    for i = 1 : size(boxCorners, 1)
        % Box corners:
        plot3(boxCorners(i, 1), boxCorners(i, 2), ...
            boxCorners(i, 3), 'o', 'Color', csq(i, :), ...
            'MarkerFaceColor', csq(i, :), 'MarkerSize', 5);
    end
    set(gcf, 'color', 'w');
    set(gca, 'color', 'w');
    set(gca, 'XColor','k');
    set(gca, 'YColor','k');
    set(gca, 'ZColor','k');
    xlabel('X (m)');
    ylabel('Y (m)');
    zlabel('Z (m)');
    title(['DATASET #', datasetNumber, ': BOX EDGES & CORNERS'], ...
        'Color', 'k');
    clear i
    nF = nF + 1;
end

%% 4 - LOAD THERMAL DATA

% Find input thermal image file in DATA folder:
imName = dir([dataFolder, '\*.jpg']).name;
imName = [dataFolder, '\', imName];

% Remove color and distortion from input thermal image:
irImageIn = rgb2gray(imread(imName));
[irImageIn, ~] = undistortImage(irImageIn, cameraParams);

% (FIGURE) View input thermal image:
if viewInputImage == 1
    figure(nF)
    imshow(irImageIn);
    hold on
    % Detection ROI example:
    drawSquareHelperFunction(1, 1, params.dPx, 'r');
    text(20, 8, 'ROI', 'Color', 'r');
    set(gcf, 'color', 'w');
    set(gca, 'color', 'w');
    title(['DATASET #', datasetNumber, ': INPUT IR IMAGE']);
    nF = nF + 1;
```

```
end

%% 5 - IR CORNER DETECTION

% (LOAD) Load detection ROIs from .mat file:
load([dataFolder, '\ROI.mat']);

% Corner detection with Harris-Stephens algorithm:
[irCorners, cornersFound] = irCornerDetection(irImageIn, ROI);

% (FIGURE) View corners detected on IR images:
if viewDetectedCorners == 1
    figure(nF)
    % Input IR image:
    imshow(irImageIn);
    hold on
    for c = 1 : numel(cornersFound)
        % Detected corners:
        plot(irCorners(c, 1), irCorners(c, 2), '+', ...
            'Color', csq(c, :));
        % Detection ROIs:
        drawSquareHelperFunction(ROI(c, 1), ROI(c, 2), ...
            params.dPx, 'g');
    end
    set(gcf, 'color', 'w');
    set(gca, 'color', 'w');
    title(['DATASET #', datasetNumber, ': TARGET BOX CORNERS']);
    clear c
    nF = nF + 1;
end

%% 6 - DETERMINATION OF EXTRINSIC PARAMETERS

% Intrinsic matrix of thermal camera:
A = (cameraParams.Intrinsics.IntrinsicMatrix)';

% Generate world points:
worldPoints = double(boxCorners(cornersFound, :));

% Generate image points:
imagePoints = irCorners(cornersFound, :);
imgPts = imagePoints;

% Convert to homogeneous coordinates:
x3d_h = [worldPoints, ones(size(worldPoints, 1), 1)];
x2d_h = [imagePoints, ones(size(imagePoints, 1), 1)];

% Solve PnP problem with EPnP algorithm:
[R, T, ~, ~] = efficient_pnp(x3d_h, x2d_h, A);
P = A * [R, T];

% Reproject world points onto IR image:
y2d_h = zeros(size(x3d_h, 1), 3);
for i = 1 : size(x3d_h, 1)
    y2d_h(i, :) = (P * x3d_h(i, :)')';
end
clear i
y2d_h = y2d_h ./ y2d_h(:, 3);
```

```
repPts = y2d_h(:, 1 : 2);

% Compute average reprojection error:
repErr = sqrt((x2d_h(:, 1) - y2d_h(:, 1)).^2 + (x2d_h(:, 2) - ...
    y2d_h(:, 2)).^2);
avgRepErr = sum(repErr) / numel(repErr);
disp('Efficient Perspective-n-Point (EPnP) algorithm:');
disp(['Average reprojection error = ', num2str(avgRepErr), ...
    ' pixels (', num2str(numel(cornersFound)), ' points)']);
disp(' ');
disp(' ');

% (FIGURE) View image points and reprojected world points:
if viewReprojectionResults == 1
    figure(nF)
    % Input IR image:
    imshow(irImageIn);
    hold on
    for c = 1 : numel(cornersFound)
        cc = cornersFound(c);
        % Image points:
        plot(imgPts(c, 1), imgPts(c, 2), '+', 'Color', csq(c, :));
        % Reprojected world points:
        plot(repPts(c, 1), repPts(c, 2), 'o', 'Color', csq(c, :));
        % Detection ROIs:
        drawSquareHelperFunction(ROI(cc, 1), ROI(cc, 2), ...
            params.dPx, 'g');
    end
    set(gcf, 'color', 'w');
    set(gca, 'color', 'w');
    title(['DATASET #', datasetNumber, ': EPnP RESULTS (', ...
        num2str(numel(cornersFound)), ' points)']);
    clear c
    nF = nF + 1;
end

%% 7 - SENSOR FUSION TEST

% Use sensor fusion to test extrinsic calibration results:
testResults = input('Test results (y/n)? ', 's');
disp(' ');
if isequal(testResults, 'y')
    % Load and pre-process additional LIDAR data:
    load([dataPath, '\ptCloudTest.mat']);
    ptCloudTestDN = pcdenoise(ptCloudTest, 'NumNeighbors', 12, ...
        'Threshold', 0.9);
    % Load and pre-process additional thermal data:
    load([dataPath, '\irImageTest.mat']);
    [irImageTestUD, ~] = undistortImage(irImageTest, cameraParams);
    % Project IR image on corresponding point cloud:
    ptCloudTestOut = camToLidar(ptCloudTestDN, irImageTestUD, P);
    % (FIGURE) View sensor fusion results:
    figure(nF)
    subplot(1, 2, 1)
    imshow(irImageTestUD);
    title('SENSOR FUSION TEST: IR IMAGE', 'Color', 'k');
    subplot(1, 2, 2)
    pcshow(ptCloudTestOut, 'MarkerSize', 8);
```

```
    xlim([0, 15]);
    ylim([0, 3]);
    zlim([-2 2]);
    set(gcf, 'color', 'w');
    set(gca, 'color', 'w');
    set(gca, 'XColor','k');
    set(gca, 'YColor','k');
    set(gca, 'ZColor','k');
    xlabel('X (m)');
    ylabel('Y (m)');
    zlabel('Z (m)');
    title('SENSOR FUSION TEST: THERMAL POINT CLOUD', 'Color', 'k');
end

% (SAVE) Save extrinsic parameters as .mat files:
saveResults = input('Save extrinsic parameters (y/n)? ', 's');
if isequal(saveResults, 'y')
    resultsInfo.DatasetUsed = dataFolder;
    resultsInfo.CornersFound = numel(cornersFound);
    resultsInfo.AverageReprojectionError = avgRepErr;
    save('Extrinsics\rotmR.mat', 'R');
    save('Extrinsics\tvecT.mat', 'T');
    save('Extrinsics\wtcCameraMatrix.mat', 'P')
    save('Extrinsics\calibrationInfo.mat', 'resultsInfo')
    disp('-- Parameters saved.')
end
```

Function: *pointCloudClustering.m*

```
function [boxPlanes, boxPointClouds, boxSide] = ...
    pointCloudClustering(ptCloudIn, maxDistance, pToll, iMax)

% POINTCLOUDCLUSTERING Extracts three almost-perpendicular planes
%   of a box-like object from the input point cloud.
%
%   boxPlanes           detected planes as "planeModel" objects
%   boxPointClouds      detected planes as "pointCloud" objects
%   boxSide             either 'dx' or 'sx' depending on orientation
%   ptCloudIn           input data as a "pointCloud" object
%   maxDistance         maximum inlier distance for plane fitting
%   pToll               plane perpendicularity threshold value
%   iMax                maximum number of attempts allowed
%
% DAVIDE DE PAZZI - Last modified: 20/03/2022

% Identify and remove ground plane:
grndMaxDist = 0.05;
grndRefVec = [0, 0, 1];
[~, ~, outliers] = pcfitplane(ptCloudIn, grndMaxDist, grndRefVec, ...
    'Confidence', 90);
ptCloudBox = select(ptCloudIn, outliers);

% Identify and cluster wall planes:
disp('Box walls detection and clustering:');
planeModels = cell(3, 1);
planePoints = cell(3, 1);
pErr = 2 * pToll;
iNum = 0;
while (pErr > pToll) && (iNum < iMax)
```

```
    iNum = iNum + 1;
    % First plane:
    [planeModels{1}, inliers, outliers] = pcfitplane(ptCloudBox, ...
        maxDistance);
    planePoints{1} = select(ptCloudBox, inliers).Location;
    ptCloudRem = select(ptCloudBox, outliers);
    % Second plane:
    [planeModels{2}, inliers, outliers] = pcfitplane(ptCloudRem, ...
        maxDistance);
    planePoints{2} = select(ptCloudRem, inliers).Location;
    ptCloudRem = select(ptCloudRem, outliers);
    % Third plane:
    [planeModels{3}, inliers, ~] = pcfitplane(ptCloudRem, ...
        maxDistance);
    planePoints{3} = select(ptCloudRem, inliers).Location;

    % Check plane perpendicularity:
    p1 = abs(dot(planeModels{1}.Normal, planeModels{2}.Normal));
    p2 = abs(dot(planeModels{1}.Normal, planeModels{3}.Normal));
    p3 = abs(dot(planeModels{2}.Normal, planeModels{3}.Normal));
    pErr = p1 + p2 + p3;
    disp(['-- Attempt ', num2str(iNum), ': pErr = ', num2str(pErr)]);
end
disp(' ');
disp('Target planes found:');
disp(['pErr = |n1 * n2''| + |n1 * n3''| + |n2 * n3''| = ', ...
    num2str(pErr), ' in ', num2str(iNum), ' attempts']);
disp(' ');
disp(' ');

% Find centroids of clustered planes:
centroidY = zeros(1, 3);
centroidZ = zeros(1, 3);
for i = 1 : 3
    planeCentroid = sum(planePoints{i, 1}) / ...
        size(planePoints{i, 1}, 1);
    centroidY(1, i) = planeCentroid(2);
    centroidZ(1, i) = planeCentroid(3);
end

% Determine box orientation:
[~, I] = sort(centroidZ);
centroidY = centroidY(I);
if centroidY(2) > centroidY(1)
    % Target box seen form "left" side:
    boxSide = 'sx';
else
    % Target box seen from "right" side:
    boxSide = 'dx';
    I(2 : 3) = I(3 : - 1 : 2);
end

% Sort clustered planes:
boxPlanes = planeModels(I);
boxPlanePoints = planePoints(I);

% Create separate point cloud subsets:
boxPointClouds{1, 1} = pointCloud(boxPlanePoints{1, 1}, ...
```

```
    'Color', ones(size(boxPlanePoints{1, 1}, 1), 3) .* [1 0 0]);
boxPointClouds{2, 1} = pointCloud(boxPlanePoints{2, 1}, ...
    'Color', ones(size(boxPlanePoints{2, 1}, 1), 3) .* [0 1 0]);
boxPointClouds{3, 1} = pointCloud(boxPlanePoints{3, 1}, ...
    'Color', ones(size(boxPlanePoints{3, 1}, 1), 3) .* [0 0 1]);

end
```

Function: *iterativeModelRefinement.m*

```
function [boxModel, oldBoxModel, boxCorners, dAvg] = ...
    iterativeModelRefinement(boxPlanes, boxPlanePoints, boxSide, ...
    dToll, iMax, SolverSettings, edgeLengths)

% ITERATIVEMODELREFINEMENT Fits and iteratively refine a model of
%   three perpendicular planes to the points of three
%   nearly-perpendicular planes.
%
%   boxModel            refined box model reference point and normals
%   oldBoxModel         initial box model reference point and normals
%   boxCorners          box corners from the refined box model
%   dAvg                average point-to-plane distance in the model
%
%   boxPlanes           detected planes as "planeModel" objects
%   boxPlanePoints      3-D point coordinates for each detected plane
%   boxSide             either 'dx' or 'sx' depending on orientation
%   dToll               average point-to-plane distance threshold
%   iMax                maximum number of iterations allowed
%   solverSettings      settings of the Levenberg-Marquardt algorithm
%   edgeLengths         width, height, and depth of the target box
%
% DAVIDE DE PAZZI - Last modified: 20/03/2022

% Find first edge vector (intersection of planes II and III):
eA = cross(boxPlanes{2}.Normal, boxPlanes{3}.Normal);

% Find second edge vector (intersection of planes I and III):
eB = cross(boxPlanes{1}.Normal, boxPlanes{3}.Normal);

% Find third edge vector (intersection of planes I and II):
eC = cross(boxPlanes{1}.Normal, boxPlanes{2}.Normal);

% Save found edges in correct order:
boxEdges = [eA; eB; eC];

% Find reference point (intersection of all planes):
M = [boxPlanes{1}.Parameters; boxPlanes{2}.Parameters; ...
    boxPlanes{3}.Parameters];
boxModel.refPoint = - (M(:, 1 : 3) \ M(:, 4))';

% Redefine edge vectors:
m1 = boxEdges(1, :);
m2 = - cross(boxEdges(1, :), boxEdges(3, :));
m3 = cross(m1, m2);

% Edge vectors direction fot target box seen from "left" side:
if isequal(boxSide, 'sx')
    if m1(1) < 0
        m1 = - m1;
```

```
    end
    if m2(2) > 0
        m2 = - m2;
    end
    if m3(3) > 0
        m3 = - m3;
    end
end

% Edge vectors direction fot target box seen from "right" side:
if isequal(boxSide, 'dx')
    if m1(1) < 0
        m1 = - m1;
    end
    if m2(3) > 0
        m2 = - m2;
    end
    if m3(2) < 0
        m3 = - m3;
    end
end

% Normalize edge vectors:
boxModel.normals(1, :) = m1 / norm(m1);
boxModel.normals(2, :) = m2 / norm(m2);
boxModel.normals(3, :) = m3 / norm(m3);

% Iterative model refinement:
disp('Iterative model refinement:');
canContinue = 1;
dAvg = 2 * dToll;
iNum = 0;
oldBoxModel = boxModel;
while (canContinue == 1) && (dAvg > dToll) && (iNum < iMax)
    iNum = iNum + 1;
    % Model rotation step:
    for k = 1 : size(boxPlanePoints, 1)
        % Cost function for rotation phase of model refinement:
        F = @(theta) cFuncRotm(boxPlanePoints, boxModel.refPoint, ...
            boxModel.normals, k, theta);
        % Cost function minimization:
        options = optimset('Algorithm', 'levenberg-marquardt', ...
            'TolX', 1e-5, 'MaxIter', 1500, 'Display', 'off');
        problem.objective = F;
        problem.solver = 'lsqnonlin';
        problem.x0 = SolverSettings.Rotation.x0;
        problem.lb = SolverSettings.Rotation.lb;
        problem.ub = SolverSettings.Rotation.ub;
        problem.options = options;
        theta = lsqnonlin(problem);
        % Rotation matrix for given axis and found angle:
        R = axangrotm(boxModel.normals(k, :)', theta);
        % Apply rotation to normal vectors to update model:
        n1 = (R * boxModel.normals(1, :)')';
        n2 = (R * boxModel.normals(2, :)')';
        n3 = (R * boxModel.normals(3, :)')';
        boxModel.normals(1, :) = n1 / norm(n1);
        boxModel.normals(2, :) = n2 / norm(n2);
```

```
        boxModel.normals(3, :) = n3 / norm(n3);
    end
    % Model translation step:
    for i = 1 : size(boxPlanePoints, 1)
        % Cost function for translation phase of model refinement:
        G = @(delta) cFuncTvec(boxPlanePoints, boxModel.refPoint, ...
            boxModel.normals, i, delta);
        % Cost function minimization:
        options = optimset('Algorithm', 'levenberg-marquardt', ...
            'TolX', 1e-5, 'MaxIter', 1500, 'Display', 'off');
        problem.objective = G;
        problem.solver = 'lsqnonlin';
        problem.x0 = SolverSettings.Translation.x0;
        problem.lb = SolverSettings.Translation.lb;
        problem.ub = SolverSettings.Translation.ub;
        problem.options = options;
        delta = lsqnonlin(problem);
        % Apply translation to reference point to update model:
        boxModel.refPoint = boxModel.refPoint + ...
            delta * boxModel.normals(i, :);
    end
    % Compute average point-to-plane distance:
    D = []; %#ok<*AGROW>
    for i = 1 : size(boxPlanePoints, 1)
        for j = 1 : size(boxPlanePoints{i}, 1)
            D = [D; abs((boxPlanePoints{i}(j, :)' - ...
                boxModel.refPoint')' * boxModel.normals(i, :)')];
        end
    end
    dAvgNew = sum(D) / numel(D);
    disp(['-- Iteration #', num2str(iNum), ': dAvg = ', ...
        num2str(dAvgNew), ' m']);
    % Verify that average point-to-plane distance is decreasing:
    if abs(dAvg - dAvgNew) >= 1e-7
        % Refinement continues:
        dAvg = dAvgNew;
    else
        % Refinement stops:
        canContinue = 0;
    end
end
disp(' ');
disp('Model refinement completed:');
disp(['dAvg = ', num2str(dAvg), ' m in ', num2str(iNum), ...
    ' iterations.']);
disp(' ');
disp(' ');

% Edge lengths with box seen from "left" side:
l1 = edgeLengths(3);
if isequal(boxSide, 'sx')
    l2 = edgeLengths(1);
    l3 = edgeLengths(2);
end

% Edge lengths with box seen from "right" side:
if isequal(boxSide, 'dx')
    l2 = edgeLengths(2);
```

```
    l3 = edgeLengths(1);
end

% Primary corner points:
boxCorners(1, :) = boxModel.refPoint;
boxCorners(2, :) = boxModel.refPoint + l2 * boxModel.normals(2, :);
boxCorners(3, :) = boxModel.refPoint + l3 * boxModel.normals(3, :);
boxCorners(4, :) = boxModel.refPoint + l1 * boxModel.normals(1, :);

% Secondary corner points:
boxCorners(5, :) = boxCorners(2, :) + l1 * boxModel.normals(1, :);
boxCorners(6, :) = boxCorners(3, :) + l2 * boxModel.normals(2, :);
boxCorners(7, :) = boxCorners(4, :) + l3 * boxModel.normals(3, :);

end
```

Function: ***axangrotm.m***

```
function R = axangrotm(v, a)

% AXANGROTM Axis-angle rotation matrix of an unit vector and an angle
%   in degrees (Multiple View Geometry in Computer Vision, p. 584).
%
% DAVIDE DE PAZZI - Last modified: 17/02/2022

u = [v(1); v(2); v(3)] / norm(v);
cpmat = [0, -u(3), u(2); u(3), 0, -u(1); -u(2), u(1), 0];
R = (cosd(a) * eye(3)) + (sind(a) * cpmat) + ((1 - cosd(a)) * ...
    (u * u'));

end
```

Function: ***cFuncRotm.m***

```
function F = cFuncRotm(planePoints, refPoint, normals, k, angle)

% CFUNCROTM Cost function for rotation phase of model refinement.
%
% DAVIDE DE PAZZI - Last modified: 17/01/2022

%#ok<*AGROW>
F = [];
for i = 1 : size(planePoints, 1)
    if i == k
        continue
    else
        for j = 1 : size(planePoints{i}, 1)
            F = [F; abs((planePoints{i}(j, :)' - refPoint')' * ...
                axangrotm(normals(k, :)', angle) * normals(i, :)')];
        end
    end
end
F = double(F);

end
```

Function: ***cFuncTvec.m***

```
function G = cFuncTvec(planePoints, refPoint, normals, i, delta)
```

```
% CFUNCTVEC Cost function for translation phase of model refinement.
%
% DAVIDE DE PAZZI - Last modified: 17/01/2022

%#ok<*AGROW>
G = [];
for j = 1 : size(planePoints{i}, 1)
    G = [G; abs((planePoints{i}(j, :)' - refPoint' - ...
        delta * normals(i, :)')' * normals(i, :)')];
end
G = double(G);

end
```

Function: *irCornerDetection.m*

```
function [irCorners, cornersFound] = irCornerDetection(irImage, ROI)

% IRCORNERDETECTION Detects the corners of the target box in the IR
%   image using Harrisâ€\Stephens algorithm.
%
%   irCorners           2-D coordinates of found corners
%   cornersFound        ordering indexes of found corners
%   irImage             input IR image as a 2-D uint8 matrix
%   ROI                 regions of interest for corner detection (*)
%
%   (*) each row is [ULcornerX, ULcornerY, dX, dY] in pixels
%
% DAVIDE DE PAZZI - Last modified: 16/02/2022

% Detect target box corners:
disp('Corner detection:')
irCorners = zeros(size(ROI, 1), 2);
cornersFound = zeros(size(ROI, 1), 1);
for c = 1 : size(ROI, 1)
    cornerPoints = detectHarrisFeatures(irImage, 'ROI', ...
        ROI(c, :), 'MinQuality', 0.01);
    % Check if any corners are found:
    if cornerPoints.Count > 0
        irCorners(c, :) = cornerPoints.selectStrongest(1).Location;
        cornersFound(c, 1) = c;
        disp(['-- Corner #', num2str(c), ' detected.']);
    end
    cornersFound = nonzeros(cornersFound);
end
disp(' ');
disp(' ');

end
```

Function: *camToLidar.m*

```
function ptCloudOut = camToLidar(ptCloudIn, irImageIn, camMatrix)

% CAMTOLIDAR Projects on a 3-D point cloud an image from a camera
%   that has been calibrated with respect to the LIDAR.
%
% DAVIDE DE PAZZI - Last modified: 16/02/2022
```

```
colorMap = ones(ptCloudIn.Count, 3, 'uint8');
for i = 1 : ptCloudIn.Count
    worldPointH = [ptCloudIn.Location(i, :), 1];
    imagePointH = (camMatrix * worldPointH')';
    u = round(imagePointH(2) / imagePointH(3));
    v = round(imagePointH(1) / imagePointH(3));
    if (u >= 1) && (u <= size(irImageIn, 1)) && ...
            (v >= 1) && (v <= size(irImageIn, 2))
        for j = 1 : 3
            colorMap(i, j) = irImageIn(u, v, j);
        end
    else
        colorMap(i, :) = [0 255 0];
    end
end
ptCloudOut = pointCloud(ptCloudIn.Location, 'Color', colorMap);

end
```

A.2 Mappatura termica

Script: *runLidarCameraSensorFusion.m*

```
% THERMAL MAPPING BY MEANS OF LIDAR AND THERMAL CAMERA SENSOR FUSION
% PART 1: LIDAR AND THERMAL CAMERA SENSOR FUSION
%
% Davide De Pazzi - davide.depazzi{@studenti.unipd.it, @gmail.com}

clc, clearvars, close all

disp('LIDAR AND THERMAL CAMERA SENSOR FUSION');
disp(' ');
disp('Created by:     Davide De Pazzi');
disp('Last modified:  20/03/2022');
disp(' ');
disp(' ');

%% SCRIPT SETUP

run params1.m

% Select output folder:
resultsFolder = ['Results\', dataFolder];
if ~isfolder(resultsFolder)
    mkdir('Results\', dataFolder);
end

% Add required folders to MATLAB path:
addpath([dataPath, dataFolder]);
addpath(resultsFolder);
addpath('Extrinsics');
addpath('HelperFunctions');
addpath('Intrinsics');
addpath('LidarCameraSensorFusion');

% (LOAD) Load intrinsic calibration results:
load('cameraParams.mat');

% (LOAD) Load extrinsic calibration results:
load('wtcCameraMatrix.mat');

%% INPUT OPTIONS

% Enable/disable loading of saved LIDAR data:
loadLidarData = 1;

% Enable/disable loading of saved thermal data:
loadThermalData = 1;

% Enable/disable loading of saved sensor fusion results:
loadSensorFusionResults = 1;

%% OUTPUT OPTIONS

% Enable/disable input point cloud sequence visualization:
viewInputPointClouds = 0;
```

```
% Enable/disable thermal point cloud sequence visualization:
viewThermalPointClouds = 0;

%% 1.1 - LOAD LIDAR DATA

if loadLidarData == 1
    % (LOAD) Load saved variables from .mat file:
    disp('Loading point clouds and rigid transformations...');
    load([resultsFolder, '\ptCloudSeq.mat']);
    load([resultsFolder, '\tformSeq.mat']);
    disp('-- Complete.');
    disp(' ');
else
    % Open and parse rosbag file:
    bag = rosbag([dataPath, dataFolder, '\', datasetName]);
    % Extract input point clouds from rosbag file:
    disp('Extracting input point clouds from rosbag...');
    topicName1 = ('/velodyne_cloud_2');
    bag1 = select(bag, 'Topic', topicName1);
    ptCloudSeq = pcSeqFromRosbag(bag1, params.ReferenceTime);
    clear topicName1 bag1
    disp('-- Completed.');
    disp(' ');
    % Extract rigid transformations from rosbag file:
    disp('Extracting rigid transformations from rosbag...');
    topicName2 = ('/aft_mapped_to_init_high_frec');
    bag2 = select(bag, 'Topic', topicName2);
    tformSeq = tformSeqFromRosbag(bag2);
    clear topicName2 bag2
    disp('-- Completed.');
    disp(' ');
    % Keep only valid point clouds and transformations:
    if params.FirstScan > 0
        ptCloudSeq = ptCloudSeq(params.FirstScan : end, :);
        tformSeq = tformSeq(params.FirstScan : end, :);
    end
    if params.LastScan > 0
        ptCloudSeq = ptCloudSeq(1 : params.LastScan, :);
        tformSeq = tformSeq(1 : params.LastScan, :);
    end
    % (SAVE) Save variables as .mat files:
    disp('Saving point clouds and rigid transformations...');
    save([resultsFolder, '\ptCloudSeq.mat'], 'ptCloudSeq');
    save([resultsFolder, '\tformSeq.mat'], 'tformSeq');
    disp('-- Completed.');
    disp(' ');
end

% (PLAYER) View input point cloud sequence:
if viewInputPointClouds == 1
    viewPointCloudSequenceHelperFunction(ptCloudSeq, ...
        params.WorldReferenceFrame.xRange, ...
        params.WorldReferenceFrame.yRange, ...
        params.WorldReferenceFrame.zRange);
end

%% 1.2 - LOAD THERMAL DATA
```

```
if loadThermalData == 1
    % (LOAD) Load saved variables from .mat files:
    disp('Loading thermal data...');
    load([resultsFolder, '\temperatureMaps.mat']);
    disp('-- Completed.');
    disp(' ');
else
    % Initialize FLIR Atlas SDK:
    atPath = getenv('FLIR_Atlas_MATLAB');
    atImage = strcat(atPath, 'Flir.Atlas.Image.dll');
    asmInfo = NET.addAssembly(atImage);
    % Match input point clouds with thermal images:
    imageFolder = [dataPath, dataFolder];
    temperatureMaps = syncThermalData(ptCloudSeq, ...
        imageFolder, cameraParams);
    clear atPath atImage imageFolder
    % (SAVE) Save variables as .mat files:
    disp('Saving thermal data...');
    save([resultsFolder, '\temperatureMaps.mat'], 'temperatureMaps');
    disp('-- Completed.');
    disp(' ');
end

%% 1.3 - PERFORM LIDAR-CAMERA SENSOR FUSION

if loadSensorFusionResults == 1
    % (LOAD) Load saved variables from .mat files:
    disp('Loading 3-D thermal point clouds...');
    load([resultsFolder, '\thermalSeq.mat']);
    disp('-- Completed.');
    disp(' ');
else
    % Create 3-D thermal point cloud from each temperature map:
    disp('Creating 3-D thermal point clouds...')
    thermalSeq = cell(size(temperatureMaps, 1), 1);
    for i = 1 : size(temperatureMaps, 1)
        % Recover index of matching point cloud:
        j = temperatureMaps{i, 3};
        % Fuse point cloud with temperature map:
        thermalSeq{i, 1} = ...
            fuseThermalCameraToLidar(ptCloudSeq{j, 1}, ...
            imread(temperatureMaps{i, 2}), ...
            temperatureMaps{i, 1}, P);
        disp(['-- Frame ', num2str(i), '/', ...
            num2str(size(temperatureMaps, 1)), ' processed.']);
    end
    clear i j
    disp(' ');
    % (SAVE) Save variables as .mat files:
    disp('Saving 3-D thermal point clouds...');
    save([resultsFolder, '\thermalSeq.mat'], 'thermalSeq');
    disp('-- Completed.');
    disp(' ');
end

% (PLAYER) View thermal point cloud sequence:
if viewThermalPointClouds == 1
    viewPointCloudSequenceHelperFunction(thermalSeq, ...
```

```
        params.WorldReferenceFrame.xRange, ...
        params.WorldReferenceFrame.yRange, ...
        params.WorldReferenceFrame.zRange);
end
```

Function: ***pcSeqFromRosbag.m***

```
function ptCloudSeq = pcSeqFromRosbag(bag, timeVector)

% PCSEQFROMROSBAG Extracts "sensor_msgs/PointCloud2" messages from a
%   rosbag file and converts them into "pointCloud" objects.
%
% DAVIDE DE PAZZI - Last modified: 20/03/2022

% Set reference time:
if (size(timeVector, 2) == 6)
    refTime = datetime(timeVector);
end

% Extract and convert point cloud data:
msgType = ('sensor_msgs/PointCloud2');
n = select(bag, 'MessageType', msgType).NumMessages;
if n > 0
    ptCloudSeq = cell(n, 3);
    % Find "sensor_msgs/PointCloud2" messages:
    bagSel = select(bag, 'MessageType', msgType);
    bagMsgs = readMessages(bagSel);
    for i = 1 : n
        % Point cloud data:
        ptCloudSeq{i, 1} = pointCloud(readXYZ(bagMsgs{i}));
        % Timestamp:
        t = bagSel.MessageList.Time(i);
        if size(timeVector, 2) == 6
            % Adjustment required:
            if i == 1
                tAdj = posixtime(refTime) - t;
            end
        else
            % Adjustment not required:
            tAdj = 0;
        end
        ptCloudSeq{i, 2} = t + tAdj;
        ptCloudSeq{i, 3} = datestr(datetime(t + tAdj, ...
            'ConvertFrom', 'posixtime'), 'yyyymmdd_HHMMSS.FFF');
    end
else
    error('No valid messages found.');
end

end
```

Function: ***tformSeqFromRosbag.m***

```
function tformSeq = tformSeqFromRosbag(bag)

% TFORMSEQFROMROSBAG Extracts "nav_msgs/Odometry" messages from a
%   rosbag file and converts them into "rigid3d" objects.
%
% DAVIDE DE PAZZI - Last modified: 20/03/2022
```

```
% Extract and convert pose information:
msgType = ('nav_msgs/Odometry');
n = select(bag, 'MessageType', msgType).NumMessages;
if n > 0
    tformSeq = cell(n, 1);
    % Find "nav_msgs/Odometry" messages:
    bagSel = select(bag, 'MessageType', msgType);
    bagMessages = readMessages(bagSel);
    for i = 1 : n
        % Convert pose into rigid transformation:
        poseX = bagMessages{i}.Pose.Pose.Position.X;
        poseY = bagMessages{i}.Pose.Pose.Position.Y;
        poseZ = bagMessages{i}.Pose.Pose.Position.Z;
        poseQW =  bagMessages{i}.Pose.Pose.Orientation.W;
        poseQX =  bagMessages{i}.Pose.Pose.Orientation.X;
        poseQY =  bagMessages{i}.Pose.Pose.Orientation.Y;
        poseQZ =  bagMessages{i}.Pose.Pose.Orientation.Z;
        tra = [poseX, poseY, poseZ];
        rot = quat2rotm([poseQW, poseQX, poseQY, poseQZ]);
        tformSeq{i, 1} = rigid3d(rot', tra);
    end
else
    error('No valid messages found.');
end

end
```

Function: *syncThermalData.m*

```
function temperatureMaps = syncThermalData(pointClouds, ...
    imageFolder, cameraParams)

% SYNCTHERMALDATA Compares timestamps to associate each 3-D point
%   cloud with a thermal image and a temperature map.
%
% DAVIDE DE PAZZI - Last modified: 20/03/2022

% Create thermal image list:
imageList = dir([imageFolder, '\*.jpg']);
nIR = size(imageList, 1);

% Convert image file names to timestamps:
imageTimestamps = zeros(nIR, 1);
for i = 1 : nIR
    timeString = imageList(i).name(1 : end - 6);
    imageTimestamps(i) = posixtime(datetime(timeString, ...
        'InputFormat', 'yyyyMMdd_HHmmss'));
end

% Associate a temperature map with each 3-D point cloud:
disp('Generating temperature maps...');
nPC = size(pointClouds, 1);
c = 0;
if nPC < nIR
    % Match each point cloud with correct IR image:
    temperatureMaps = cell(nPC, 2);
    for i = 1 : nPC
        % Compare timestamps:
```

```
        tDiff = abs(pointClouds{i, 2} - imageTimestamps);
        [~, j] = min(tDiff);
        % Specify point cloud index:
        temperatureMaps{i, 3} = i;
        % Specify corresponding thermal image:
        imageName = [imageFolder, '\', imageList(j).name];
        if (i > 1) && isequal(imageName, temperatureMaps{i - 1, 2})
            break
        else
            c = c + 1;
            temperatureMaps{i, 2} = imageName;
        end
        % Generate "ThermalImageFile" object:
        imageFile = Flir.Atlas.Image.ThermalImageFile(imageName);
        % Extract temperature map from thermal image:
        [tMap, ~] = tempMap2D(imageFile, cameraParams);
        temperatureMaps{i, 1} = tMap;
        disp(['-- Image ', num2str(i), '/', num2str(nPC), ...
            ' converted.']);
    end
else
    % Match each IR image with correct point cloud:
    temperatureMaps = cell(nIR, 2);
    for i = 1 : nIR
        % Compare timestamps:
        tDiff = abs(imageTimestamps(i) - ...
            cell2mat(pointClouds(:, 2)));
        [~, j] = min(tDiff);
        % Specify point cloud index:
        if (i > 1) && (j == temperatureMaps{i - 1, 3})
            break
        else
            c = c + 1;
            temperatureMaps{i, 3} = j;
        end

        % Specify corresponding thermal image:
        imageName = [imageFolder, '\', imageList(i).name];
        temperatureMaps{i, 2} = imageName;
        % Generate FLIR ThermalImageFile object:
        imageFile = Flir.Atlas.Image.ThermalImageFile(imageName);
        % Extract temperature map from thermal image:
        tMap = tempMap2D(imageFile, cameraParams);
        temperatureMaps{i, 1} = tMap;
        disp(['-- Image ', num2str(i), '/', num2str(nIR), ...
            ' converted.']);
    end
end
temperatureMaps = temperatureMaps(1 : c, :);
disp(' ');

end
```

Function: ***tempMap2D.m***

```
function tMap = tempMap2D(thermalImageFile, cameraParams)

% TEMPMAP2D Extracts a 2-D temperature map from a FLIR radiometric
%   thermal image after removing lens distortion.
```

```
%
% DAVIDE DE PAZZI - Last modified: 20/03/2022

% Extract pixel intensity:
seq = thermalImageFile.ThermalSequencePlayer;
rawImg = double(seq.ThermalImage.ImageProcessing.GetPixelsArray);

% Remove lens distortion:
[rawImgUD, ~] = undistortImage(rawImg, cameraParams);

% Convert pixel intensity into temperature values:
tMap = NaN * ones(thermalImageFile.Height, thermalImageFile.Width);
for i = 1 : thermalImageFile.Height
    for j = 1 : thermalImageFile.Width
        px = rawImgUD(i, j);
        tMap(i, j) = seq.ThermalImage.GetValueFromSignal(px);
    end
end

end
```

Function: ***fuseThermalCameraToLidar.m***

```
function ptCloudOut = fuseThermalCameraToLidar(ptCloudIn, ...
    irImageIn, tempMapIn, cameraMatrix, varargin)

% FUSETHERMALCAMERATOLIDAR Projects a thermal image on a 3-D point
%   cloud and associates the corresponding temperature value and
%   color with each point.
%
% DAVIDE DE PAZZI - Last modified: 20/03/2022

% (OPTIONAL) Denoise 3-D point cloud:
if (size(varargin, 2) == 3) && isequal(varargin{1}, 'denoise')
    ptCloudIn = pcdenoise(ptCloudIn, 'NumNeighbors', varargin{2}, ...
        'Threshold', varargin{3});
end

% Iteratively process 3-D points:
colorMap = ones(ptCloudIn.Count, 3, 'uint8');
intMap = ones(ptCloudIn.Count, 1);
for i = 1 : ptCloudIn.Count
    % Convert from world coordinates to image coordinates:
    worldPointH = [ptCloudIn.Location(i, :), 1];
    imagePointH = (cameraMatrix * worldPointH')';
    u = round(imagePointH(2) / imagePointH(3));
    v = round(imagePointH(1) / imagePointH(3));
    % Extract pixel data:
    uMax = size(irImageIn, 1);
    vMax = size(irImageIn, 2);
    if (u >= 1) && (u <= uMax) && (v >= 1) && (v <= vMax)
        % Extract color data from IR image:
        for j = 1 : 3
            colorMap(i, j) = irImageIn(u, v, j);
        end
        % Extra temperature data from temperature map:
        intMap(i) = tempMapIn(u, v);
    else
        % Set out-of-palette color:
```

```
        colorMap(i, :) = [0 0 255];
        % Set temperature as NaN:
        intMap(i) = NaN;
    end
end

% Create new point cloud:
ptCloudOut = pointCloud(ptCloudIn.Location, 'Color', colorMap, ...
    'Intensity', intMap);

end
```

Script: *runRegistrationMerging.m*

```
% THERMAL MAPPING BY MEANS OF LIDAR AND THERMAL CAMERA SENSOR FUSION
% PART 2: THERMAL POINT CLOUDS REGISTRATION AND MERGING
%
% Davide De Pazzi - davide.depazzi{@studenti.unipd.it, @gmail.com}

clc, clearvars, close all

disp('THERMAL POINT CLOUD REGISTRATION AND MERGING');
disp(' ');
disp('Created by:     Davide De Pazzi');
disp('Last modified:  20/03/2022');
disp(' ');
disp(' ');

%% SCRIPT SETUP

run params1.m

% Select results folder:
resultsFolder = ['Results\', dataFolder];

% Add required folders to MATLAB path:
addpath(resultsFolder);
addpath('HelperFunctions');
addpath('RegistrationMerging');

%% INPUT OPTIONS

% Enable/disable loading of saved thermal scene:
loadThermalScene = 0;

% Enable/disable loading of saved 3-D occupancy map:
loadOccupancyMap = 0;

%% OUTPUT OPTIONS

% Enable/disable registered thermal point clouds visualization:
viewRegisteredPointClouds = 0;

% Enable/disable merged thermal point clouds visualization:
viewMergedPointClouds = 1;

%% 2.1 - REGISTER AND MERGE THERMAL POINT CLOUDS

if loadThermalScene == 1
```

```
    % (LOAD) Load saved variables from .mat files:
    disp('Loading registration and merging results...');
    load([resultsFolder, '\thermalSeqFilt.mat']);
    load([resultsFolder, '\thermalSeqReg.mat']);
    load([resultsFolder, '\thermalScene.mat']);
    disp('-- Completed.');
    disp(' ');
else
    % (LOAD) Load results of previous steps:
    load([resultsFolder, '\thermalSeq.mat']);
    load([resultsFolder, '\tformSeq.mat']);
    load([resultsFolder, '\temperatureMaps.mat']);
    % Filter thermal point clouds:
    disp('Pre-processing thermal point clouds...');
    thermalSeqFilt = filterThermalPointClouds(thermalSeq, ...
        'crop', params.CroppingWindow);
    disp('-- Completed.');
    disp(' ');
    % Register thermal point clouds:
    disp('Registering thermal point clouds...');
    thermalSeqReg = cell(size(thermalSeqFilt, 1), 1);
    for i = 1 : size(thermalSeqFilt, 1)
        thermalSeqReg{i, 1} = pctransform(thermalSeqFilt{i, 1}, ...
            tformSeq{temperatureMaps{i, 3}, 1});
    end
    clear i
    disp('-- Completed.');
    disp(' ');
    % Merge thermal point clouds:
    disp('Merging thermal point clouds...');
    [thermalScene, tformCorr] = ...
        mergeThermalPointClouds(thermalSeqReg, ...
        params.BoxFilterGridSize, params.MergingOptions{1, :});
    disp('-- Completed.');
    disp(' ');
    % Update rigid transform sequence:
    tformSeqNew = cell(size(tformSeq, 1), 1);
    for i = 1 : size(tformSeq, 1)
        tformSeqNew{i, 1} = rigid3d(tformSeq{i, 1}.T * tformCorr.T);
    end
    % (SAVE) Save variables as .mat files:
    disp('Saving registration and merging results...');
    save([resultsFolder, '\thermalSeqFilt.mat'], 'thermalSeqFilt');
    save([resultsFolder, '\thermalSeqReg.mat'], 'thermalSeqReg');
    save([resultsFolder, '\thermalScene.mat'], 'thermalScene');
    save([resultsFolder, '\tformSeqNew.mat'], 'tformSeqNew');
    disp('-- Completed.');
    disp(' ');
end

% (PLAYER) View registered thermal point cloud sequence:
if viewRegisteredPointClouds == 1
    viewPointCloudSequenceHelperFunction(thermalSeqReg, ...
        params.CroppingWindow(1 : 2), ...
        params.CroppingWindow(3 : 4), ...
        params.CroppingWindow(5 : 6));
end
```

```
% (FIGURE) View thermal scene and estimate rover path:
nF = 1;
if viewMergedPointClouds == 1
    figure(nF)
    pcshow(thermalScene);
    roverPathHelperFunction(nF, 4, params.LidarHeight, ...
        tformSeqNew, temperatureMaps);
    set(gcf, 'color', 'w');
    set(gca, 'color', 'w');
    set(gca, 'XColor','k');
    set(gca, 'YColor','k');
    set(gca, 'ZColor','k');
    xlim(params.CroppingWindow(1 : 2));
    ylim(params.CroppingWindow(3 : 4));
    zlim(params.CroppingWindow(5 : 6));
    grid minor
    xlabel('X  (m)');
    ylabel('Y  (m)');
    zlabel('Z  (m)');
    title('THERMAL SCENE AND ESTIMATED ROVER PATH', 'FontSize', ...
        12, 'Color', 'k');
    nF = nF + 1;
end

%% 2.2 - CREATE 3-D OCCUPANCY MAP

if params.OccMapGridResolution > 0
    if loadOccupancyMap == 1
        % (LOAD) Load saved variables from .mat files:
        disp('Loading 3-D occupancy map...');
        load([resultsFolder, '\occMap3D.mat']);
        disp('-- Completed.');
        disp(' ');
    else
        % Build occupancy map from thermal point clouds:
        disp('Creating 3-D occupancy map...');
        occMap3D = occupancyMap3D(params.OccMapGridResolution);
        occMap3D = buildOccMap3D(occMap3D, thermalSeqFilt, ...
            tformSeq(cell2mat(temperatureMaps(:, 3))), ...
            params.MaxLidarRange, 'downsample', 0.4);
        disp(' ');
        % (SAVE) Save variables as .mat files:
        disp('Saving 3-D occupancy map...');
        save([resultsFolder, '\occMap3D.mat'], 'occMap3D');
        disp('-- Completed.');
        disp(' ');
    end
    % (FIGURE) View 3-D occupancy map:
    figure(nF)
    show(occMap3D);
    axis equal
    grid minor
    xlim(params.CroppingWindow(1 : 2));
    ylim(params.CroppingWindow(3 : 4));
    zlim(params.CroppingWindow(5 : 6));
    set(gcf, 'color', 'w');
    xlabel('X (m)');
    ylabel('Y (m)');
```

```
    zlabel('Z (m)');
    title('3-D ELEVATION-BASED OCCUPANCY MAP', 'FontSize', 12);
    nF = nF + 1;
end
```

Function: ***filterThermalPointClouds.m***

```
function thermalSeqOut = filterThermalPointClouds(thermalSeqIn, ...
    varargin)

% FILTERTHERMALPOINTCLOUDS Filters a sequence of thermal point clouds
%   by excluding points associated with invalid temperature values.
%
% DAVIDE DE PAZZI - Last modified: 20/03/2022

% Iteratively process thermal point clouds:
thermalSeqOut = cell(size(thermalSeqIn, 1), 1);
for i = 1 : size(thermalSeqIn, 1)
    % Select only points associated with valid temperature values:
    ind1 = ~isnan(thermalSeqIn{i, 1}.Intensity);
    tpcFilt = select(thermalSeqIn{i, 1}, ind1);
    % (OPTIONAL) Crop filtered thermal point cloud:
    if (size(varargin, 2) == 2) && isequal(varargin{1}, 'crop')
        ROI = varargin{2};
        ind2 = (tpcFilt.Location(:, 1) > ROI(1)) & ...
            (tpcFilt.Location(:, 1) < ROI(2)) & ...
            (tpcFilt.Location(:, 2) > ROI(3)) & ...
            (tpcFilt.Location(:, 2) < ROI(4)) & ...
            (tpcFilt.Location(:, 3) > ROI(5)) & ...
            (tpcFilt.Location(:, 3) < ROI(6));
        thermalSeqOut{i, 1} = select(tpcFilt, ind2);
    else
        thermalSeqOut{i, 1} = tpcFilt;
    end
end

end
```

Function: ***mergeThermalPointClouds.m***

```
function [thermalScene, tform] = mergeThermalPointClouds(tpcIn, ...
    boxGridSize, varargin)

% MERGETHERMALSCANS Merges pre-registered 3-D thermal point clouds
%   into a single thermal scene, then denoise and downsamples the
%   resulting point cloud to improve the efficiency of the spatial
%   decomposition algorithm.
%
% DAVIDE DE PAZZI - Last modified: 20/03/2022

% Iteratively merge new 3-D thermal point clouds:
tpcNew = tpcIn{1, 1};
for i = 2 : size(tpcIn, 1)
    tpcNew = pcmerge(tpcNew, tpcIn{i, 1}, boxGridSize);
end

% (OPTIONAL) Process merged 3-D thermal point clouds:
if (size(varargin, 2) == 2) && isequal(varargin{1}, 'downsample')
    % Thermal scene random downsampling:
```

```
    samplingRatio = varargin{2};
    thermalScene = pcdownsample(tpcNew, 'random', samplingRatio);
    tform = rigid3d(eye(4));
elseif (size(varargin, 2) == 2) && isequal(varargin{1}, 'level')
    % Thermal scene leveling:
    maxDist = varargin{2};
    refVect = [0 0 1];
    ground = pcfitplane(tpcNew, maxDist, refVect);
    tform = normalRotation(ground, refVect);
    thermalScene = pctransform(tpcNew, tform);
elseif (size(varargin, 2) == 3) && isequal(varargin{1}, 'both')
    % Thermal scene random downsampling:
    samplingRatio = varargin{2};
    tpcNew = pcdownsample(tpcNew, 'random', samplingRatio);
    % Thermal scene leveling:
    maxDist = varargin{3};
    refVect = [0 0 1];
    ground = pcfitplane(tpcNew, maxDist, refVect);
    tform = normalRotation(ground, refVect);
    thermalScene = pctransform(tpcNew, tform);
else
    thermalScene = tpcNew;
    tform = rigid3d(eye(4));
end

end
```

Function: ***buildOccMap3D.m***

```
function occMap3D = buildOccMap3D(occMap3D, scans, poses, ...
    maxRange, varargin)

% BUILDOCCMAP3D Creates a 3-D elevation-based occupancy map using a
%   set of unregistered point clouds and rigid transformations.
%
% DAVIDE DE PAZZI - Last modified: 20/03/2022

% Iteratively process 3-D point clouds:
for i = 1 : size(scans, 1)
    % Convert rigid transformation into quaternion form:
    q = [poses{i, 1}.Translation, rotm2quat(poses{i, 1}.Rotation)];
    % (OPTIONAL) Further downsample point cloud:
    if (size(varargin, 2) == 2) && isequal(varargin{1}, 'downsample')
        pc = pcdownsample(scans{i, 1}, 'random', varargin{2});
    else
        pc = scans{i, 1};
    end
    % Insert point cloud into map:
    insertPointCloud(occMap3D, q, pc.Location, maxRange);
    disp(['-- Point cloud ', num2str(i), '/', ...
        num2str(size(scans, 1)), ' inserted.']);
end

end
```

Script: ***runThermalMapping.m***

```
% THERMAL MAPPING BY MEANS OF LIDAR AND THERMAL CAMERA SENSOR FUSION
% PART 3: TEMPERATURE DISTRIBUTION MAP CREATION
```

```
%
% Davide De Pazzi - davide.depazzi{@studenti.unipd.it, @gmail.com}

% DOWNLOAD "BuildOctree" FUNCTION IN "OcTree" FOLDER FROM:
% it.mathworks.com/matlabcentral/fileexchange/36782-buildoctree

clc, clearvars, close all

disp('TEMPERATURE DISTRIBUTION MAP CREATION');
disp(' ');
disp('Created by:     Davide De Pazzi');
disp('Last modified:  20/03/2022');
disp(' ');
disp(' ');

%% SCRIPT SETUP

run params1.m

% Select results folder:
resultsFolder = ['Results\', dataFolder];

% Add required folders to MATLAB path:
addpath(resultsFolder);
addpath('HelperFunctions');
addpath('OcTree');
addpath('ThermalMapping');

%% INPUT OPTIONS

% Enable/disable loading of saved OcTree object:
loadOctreeObject = 0;

% Enable/disable loading of saved temperature distribution map:
loadTemperatureDistributionMap = 0;

%% OUTPUT OPTIONS

% Enable/disable merged thermal scene visualization:
viewMappedThermalScene = 1;

% Enable/disable temperature distribution map visualization:
viewTemperatureDistributionMap = 0;

%% 3.1 - PERFORM OCTREE SPATIAL DECOMPOSITION

if loadOctreeObject == 1
    % (LOAD) Load variables from .mat files:
    disp('Loading mapped thermal scene and OcTree object...');
    load([resultsFolder, '\mappedScene.mat']);
    load([resultsFolder, '\otObj.mat']);
    disp('-- Completed.');
    disp(' ');
else
    % (LOAD) Load results of previous steps:
    load([resultsFolder, '\thermalScene.mat']);
    % Exclude points outside map limits:
    disp('Pre-processing thermal scene...');
```

```
    ind = (thermalScene.Location(:, 1) > params.MapLimits(1)) & ...
        (thermalScene.Location(:, 1) < params.MapLimits(2)) & ...
        (thermalScene.Location(:, 2) > params.MapLimits(3)) & ...
        (thermalScene.Location(:, 2) < params.MapLimits(4)) & ...
        (thermalScene.Location(:, 3) > params.MapLimits(5)) & ...
        (thermalScene.Location(:, 3) < params.MapLimits(6));
    mappedScene = select(thermalScene, ind);
    clear ind
    disp('-- Completed.');
    disp(' ');
    % Generate OcTree object:
    disp('Generating OcTree object...');
    otObj = BuildOctree(mappedScene.Location(:, 1), ...
        mappedScene.Location(:, 2), mappedScene.Location(:, 3), ...
        params.MinVoxelSize);
    disp('-- Completed.');
    disp(' ');
    % (SAVE) Save variables as .mat files:
    disp('Saving mapped thermal scene and OcTree object...');
    save([resultsFolder, '\mappedScene.mat'], 'mappedScene');
    save([resultsFolder, '\otObj.mat'], 'otObj');
    disp('-- Completed.');
    disp(' ');
end

% (FIGURE) View mapped thermal scene:
nF = 1;
if viewMappedThermalScene == 1
    figure(nF)
    pcshow(mappedScene, 'MarkerSize', 8);
    set(gcf, 'color', 'w');
    set(gca, 'color', 'w');
    set(gca, 'XColor','k');
    set(gca, 'YColor','k');
    set(gca, 'ZColor','k');
    xlim(params.MapLimits(1 : 2));
    ylim(params.MapLimits(3 : 4));
    zlim(params.MapLimits(5 : 6));
    grid minor
    xlabel('X  (m)');
    ylabel('Y  (m)');
    zlabel('Z  (m)');
    title('MAPPED THERMAL SCENE', 'FontSize', 12, 'Color', 'k');
    nF = nF + 1;
end

%% 3.2 - CREATE 3-D TEMPERATURE DISTRIBUTION MAP

if loadTemperatureDistributionMap == 1
    disp('Loading 3-D temperature distribution map...');
    load([resultsFolder, '\thermalVoxels.mat']);
    load([resultsFolder, '\mapInfo.mat']);
    disp('-- Completed.');
    disp(' ');
else
    % Convert OcTree object into thermal voxels:
    disp('Converting OcTree object into thermal voxels...');
    thermalVoxels = octreeToVoxels(mappedScene, otObj, ...
```

```
        params.MinBinSize, 'groundRef');
    % Convert voxel temperature values to colormap indices:
    minTemp = min(cell2mat(thermalVoxels(:, 6)));
    maxTemp = max(cell2mat(thermalVoxels(:, 6)));
    for i = 1 : size(thermalVoxels, 1)
        thermalVoxels{i, 7} = floor(1 + 255 * ...
            (thermalVoxels{i, 6} - minTemp) / (maxTemp - minTemp));
    end
    clear i
    disp('-- Completed.');
    disp(' ');
    % Thermal voxels edge length:
    d = thermalVoxels{1, 3};
    % (SAVE) Save variables as .mat files:
    disp('Saving 3-D temperature distribution map...');
    save([resultsFolder, '\thermalVoxels.mat'], 'thermalVoxels');
    save([resultsFolder, '\mapInfo.mat'], 'minTemp', 'maxTemp');
    disp('-- Completed.');
    disp(' ');
end

% (FIGURE) View temperature distribution map:
if viewTemperatureDistributionMap == 1
    % Plot colored thermal voxels:
    plotTempDistMap3D(thermalVoxels, minTemp, maxTemp, 1, nF, ...
        ['3-D TEMPERATURE DISTRIBUTION MAP (d = ', ...
        num2str(round(d, 3)), ' m)'], 'progress');
    xlim(params.MapLimits(1 : 2) + [-0.2, 0.2]);
    ylim(params.MapLimits(3 : 4) + [-0.2, 0.2]);
    zlim([0, params.MapLimits(6)] + + [-0.2, 0.5]);
    nF = nF + 1;
end
```

Function: *octreeToVoxels.m*

```
function thermalVoxels = octreeToVoxels(thermalScene, ...
    octreeObject, minBinSize, varargin)

% OCTREETOVOXEL Uses the information stored in the OcTree object to
%   turn the thermal scene into a set of thermal voxels.
%
% DAVIDE DE PAZZI - Last modified: 20/03/2022

% Number of subdivisions:
maxDepth = size(octreeObject, 2);

% Extract all non-empty max-depth bins from OcTree object:
maxDepthBins = octreeObject(maxDepth).group;
numBins = size(maxDepthBins, 2);

% Iteratively process max-depth bins into thermal voxels:
thermalVoxels = cell(numBins, 6);
j = 0;
for i = 1 : numBins
    % Exclude points with invalid temperature values:
    pointsIn = maxDepthBins(i).child;
    pointsIn = pointsIn(~isnan(thermalScene.Intensity(pointsIn)));
    % Select only bins containing enough points:
    if numel(pointsIn) > minBinSize
```

```
        % Update voxel counter:
        j = j + 1;
        % Voxel identifier:
        thermalVoxels{j, 1} = i;
        % Voxel origin (lower-right-back corner):
        thermalVoxels{j, 2} = maxDepthBins(i).groupcenter - ...
            0.5 * maxDepthBins(i).cubelength;
        % Voxel edge size:
        thermalVoxels{j, 3} = maxDepthBins(i).cubelength;
        % Contained point indexes:
        thermalVoxels{j, 4} = pointsIn;
        % Contained point temperatures:
        thermalVoxels{j, 5} = thermalScene.Intensity(pointsIn);
        % Voxel average temperature:
        thermalVoxels{j, 6} = sum(thermalVoxels{j, 5}) / ...
            numel(thermalVoxels{j, 5});
    end
end

% Keep only non-empty voxels:
thermalVoxels = thermalVoxels(1 : j, :);

% (OPTIONAL) Set zero of world vertical axis at ground level:
if (size(varargin, 2) == 1) && isequal(varargin{1}, 'groundRef')
    voxelOrigins = cell2mat(thermalVoxels(:, 2));
    voxelOriginsNew = [voxelOrigins(:, 1 : 2), ...
        voxelOrigins(:, 3) + abs(min(voxelOrigins(:, 3)))];
    thermalVoxels(:, 2) = mat2cell(voxelOriginsNew, ones(1, ...
        size(thermalVoxels, 1)), 3);
end

end
```

Function: ***plotTempDistMap3D.m***

```
function plotTempDistMap3D(thermalVoxels, minT, maxT, voxelAlpha, ...
    nF, mapTitle, details)

% PLOTTEMPDISTMAP Plots the thermal voxels and the colorbar of a 3-D
%   temperature distribution map.
%
% DAVIDE DE PAZZI - Last modified: 20/03/2022

if isequal(details, 'status') || isequal(details, 'progress')
    disp('Visualizing 3-D temperature distribution map...');
end
figure(nF)

% Setup temperature distribution map colorbar:
colormap spring % hot
cMap = colormap;
colormapTicks = 0 : 0.1 : 1;
colormapLabels = string(round(minT + colormapTicks * ...
    (maxT - minT), 1)) + " Â°C";

% Plot thermal voxels:
nV = size(thermalVoxels, 1);
for i = 1 : nV
    plotVoxelHelperFunction(thermalVoxels{i, 2}, ...
```

```
        thermalVoxels{i, 3}, cMap(thermalVoxels{i, 7}, :), ...
        voxelAlpha);
    if isequal(details, 'progress')
        disp(['-- Voxel ', num2str(i), '/', num2str(nV), ' added.']);
    end
end
if isequal(details', 'status')
    disp('-- Completed.');
end
if isequal(details, 'status') || isequal(details, 'progress')
    disp(' ');
end
grid minor
axis equal
colorbar('Ticks', colormapTicks, 'TickLabels', colormapLabels);
set(gcf, 'color', 'w');
set(gca, 'color', 'w');
xlabel('X  (m)');
ylabel('Y  (m)');
zlabel('Z  (m)');
title(mapTitle, 'FontSize', 12);

end
```

Script: *testResults.m*

```
% THERMAL MAPPING BY MEANS OF LIDAR AND THERMAL CAMERA SENSOR FUSION
% TEST: TEMPERATURE DISTRIBUTION MAP ANALYSIS
%
% Davide De Pazzi - davide.depazzi{@studenti.unipd.it, @gmail.com}

clc, clearvars, close all

%#ok<*SAGROW>

disp('TEMPERATURE DISTRIBUTION MAP ANALYSIS');
disp(' ');
disp('Created by:     Davide De Pazzi');
disp('Last modified:  20/03/2022');
disp(' ');
disp(' ');

%% SCRIPT SETUP

run params1.m

% Select results folder:
resultsFolder = ['Results\', dataFolder];

% Add required folders to MATLAB path:
addpath(resultsFolder);
addpath('HelperFunctions');
addpath('ThermalMapping');

%% T.1 - PRE-PROCESS TEMPERATURE DISTRIBUTION MAP

% (LOAD) Load thermal voxels:
disp('Loading 3-D temperature distribution map...');
load([resultsFolder, '\thermalVoxels.mat']);
```

```
load([resultsFolder, '\mapInfo.mat']);
disp('-- Completed.');
disp(' ');

% Extract thermal voxels over threshold temperature:
disp('Extracting thermal voxels over threshold temperature...');
thermalVoxelsHot = filterVoxelsHelperFunction(thermalVoxels, ...
    'minTemp', params.ThresholdTemperature);
disp('-- Completed.');
disp(' ');

% Extract thermal voxels inside region of interest:
disp('Extracting thermal voxels inside region of interest...');
thermalVoxelsROI = thermalVoxelsHot;
for i = 1 : 2 : 6
    j = i + 1;
    k = ceil(i / 2);
    ROI(i) = max(params.MapLimits(i), params.AnalysisLimits(i));
    ROI(j) = min(params.MapLimits(j), params.AnalysisLimits(j));
    thermalVoxelsROI = filterVoxelsHelperFunction(...
        thermalVoxelsROI, 'minDist', k, ROI(i));
    thermalVoxelsROI = filterVoxelsHelperFunction(...
        thermalVoxelsROI, 'maxDist', k, ROI(j));
end
clear i j k
disp('-- Completed.');
disp(' ');

% (FIGURE) View thermal voxels inside region of interest:
nF = 1;
plotTempDistMap3D(thermalVoxelsROI, minTemp, maxTemp, 0.5, nF, ...
    'VOXELS OVER THRESHOLD TEMPERATURE IN REGION OF INTEREST', ...
    'none');
xlim(ROI(1 : 2));
ylim(ROI(3 : 4));
zlim(ROI(5 : 6));
nF = nF + 1;

%% T.2 - CLUSTER PRE-PROCESSED TEMPERATURE DISTRIBUTION MAP

% Thermal voxels edge length:
d = thermalVoxels{1, 3};

% Analysis starting point:
X_first = min(cell2mat(thermalVoxelsROI(:, 2)));
X_first = X_first(1);

% Analysis ending point:
X_last = max(cell2mat(thermalVoxelsROI(:, 2)));
X_last = X_last(1) + 0.75 * params.ExpectedClusterSpacing;

% Initial search window:
dX = params.ExpectedClusterLength + 4 * d;
X_min = X_first - 0.4 * dX;
X_max = X_first + 0.6 * dX;

% Expected distance bewteen centroids of voxel clusters:
D_exp = params.ExpectedClusterSpacing + params.ExpectedClusterLength;
```

```
D_jmp = 1.0 * D_exp;

% Group extracted thermal voxels into clusters:
disp('Clustering extracted thermal voxels...');
figure(nF)
hold on
nC = 0;
nA = 0;
while (X_max <= X_last) && (nA <= 50)
    % Update attempts counter:
    nA = nA + 1;
    % Extract voxels in front of X_min:
    voxelGroup = filterVoxelsHelperFunction(thermalVoxelsROI, ...
        'minDist', 1, X_min);
    % Verify if enough voxels are found:
    if size(voxelGroup, 1) >= params.MinClusterSize
        % Extract voxels behind X_max:
        voxelGroup = filterVoxelsHelperFunction(voxelGroup, ...
            'maxDist', 1, X_max);
        % Verify if enough voxels are found:
        if size(voxelGroup, 1) >= params.MinClusterSize
            % Update cluster counter:
            nC = nC + 1;
            % (FIGURE) Plot search window:
            plotVoxelHelperFunction([X_min, ...
                params.AnalysisLimits(3), 0], dX, 'c', 0.1);
            % Find centroid of all voxels in cluster:
            for i = 1 : size(voxelGroup, 1)
                voxelGroup{i, 8} = voxelGroup{i, 2} + 0.5 * d;
            end
            clear i
            voxelClusters{nC, 1} = voxelGroup;
            % (FIGURE) Plot voxel cluster:
            plotTempDistMap3D(voxelClusters{nC}, minTemp, ...
                maxTemp, 0.5, nF, 'VOXEL CLUSTERS', 'none');
            % Find centroid of cluster:
            groupCentroid = sum(cell2mat(voxelGroup(:, 8)), 1) / ...
                size(cell2mat(voxelGroup(:, 8)), 1);
            clusterLocation(nC, :) = [groupCentroid, D_exp];
            % (FIGURE) Plot cluster centroid:
            plot3(clusterLocation(nC, 1), clusterLocation(nC, 2), ...
                clusterLocation(nC, 3), 'pm', 'MarkerFaceColor', ...
                'm', 'MarkerSize', 10);
            plot3(clusterLocation(nC, 1), clusterLocation(nC, 2), ...
                0, 'om', 'MarkerFaceColor', 'm', 'MarkerSize', 6);
            text(clusterLocation(nC, 1), ...
                params.AnalysisLimits(4), 3.5, num2str(nC), ...
                'FontWeight', 'bold');
            % Move to next cluster:
            X_new = groupCentroid(1);
            X_new = X_new + D_exp;
        else
            % (FIGURE) Plot search window:
            plotVoxelHelperFunction([X_min, ...
                params.AnalysisLimits(3), 0], dX, 'c', 0.1);
            % Move to next cluster:
            X_new = 0.5 * (X_min + X_max);
            X_new = X_new + D_jmp;
```

```
            clusterLocation(nC, 4) = X_new - clusterLocation(nC, 1);
        end
    else
        % (FIGURE) Plot search window:
        plotVoxelHelperFunction([X_min, ...
                params.AnalysisLimits(3), 0], dX, 'c', 0.1);
        % Move to next cluster:
        X_new = 0.5 * (X_min + X_max);
        X_new = X_new + D_jmp;
        clusterLocation(nC, 4) = X_new - clusterLocation(nC, 1);
    end
    % Define new search window:
    X_min = X_new - 0.5 * dX;
    X_max = X_new + 0.5 * dX;
end
grid minor
disp('-- Completed.');
disp(' ');

%% T.3 - ANALYSIS RESULTS

% Distance between cluster centroids:
clusterDistance = diff(clusterLocation(:, 1));

% Analyze cluster distribution;
disp('Analyze voxel clusters distribution...');
for i = 1 : size(clusterDistance, 1)
    % Number of current clusters:
    analysisResults(i).From = i + 1;
    analysisResults(i).To = i;
    % Estimated distance between cluster centroids:
    analysisResults(i).MinimumDistance = clusterDistance(i) - d;
    analysisResults(i).EstimatedDistance = clusterDistance(i);
    analysisResults(i).MaximumDistance = clusterDistance(i) + d;
    % Expected distance between cluster centroids:
    analysisResults(i).ExpectedDistance = clusterLocation(i, 4);
    % Compatibility check:
    analysisResults(i).IsCompatible = ...
        (analysisResults(i).ExpectedDistance >= ...
        analysisResults(i).MinimumDistance) & ...
        (analysisResults(i).ExpectedDistance <= ...
        analysisResults(i).MaximumDistance);
end
clear i
disp('-- Complete.');
disp(' ');
openvar('analysisResults');
```

A.3 Script e funzioni ausiliarie

Script: *params1.m*

```
% THERMAL MAPPING BY MEANS OF LIDAR AND THERMAL CAMERA SENSOR FUSION
% PARAMETERS FOR DATASET: "Interno2" (bassa risoluzione, lato sinistro)
%
% Davide De Pazzi - Last modified: 20/03/2022

%% INPUT SETTINGS

% Data folder path:
dataPath = ('C:\Users\Davide\Documents\Tesi\Dati\Test\');

% Data folder name:
dataFolder = ('Interno');

% LIDAR dataset name:
datasetName = ('LOAM.bag');

%% PART 1: runLidarCameraSensorFusion.m

% LIDAR dataset reference time (set to 0 to ignore):
params.ReferenceTime = [2021 12 03 16 42 21];

% First valid point cloud in dataset (set to 0 to ignore):
params.FirstScan = 0;

% Last valid point cloud in dataset (set to 0 to ignore):
params.LastScan = 1065;

% Maximum LIDAR range (m):
params.MaxLidarRange = 60;

% World reference frame limits (m):
params.WorldReferenceFrame.xRange = [0 60];
params.WorldReferenceFrame.yRange = [-4 4];
params.WorldReferenceFrame.zRange = [-2 2];

%% PART 2: runRegistrationMerging.m

% Window size for point cloud cropping (m):
params.CroppingWindow = [0, 52, -2, 2, -2, 2];

% Grid size for point cloud filtering (m):
params.BoxFilterGridSize = 0.03;

% Options for point cloud merging (set to { } to ignore):
params.MergingOptions = {'both', 0.75, 0.05};

% Ground-to-LiDAR distance for rover path visualization:
params.LidarHeight = 0.7;

% Occupancy map cells/meter (set to 0 to ignore):
params.OccMapGridResolution = 0;

%% PART 3: runThermalMapping.m
```

```
% Temperature distribution map limits (m):
params.MapLimits = [10, 45, 0, 2, -2, 2];

% Minimum voxel size (m):
params.MinVoxelSize = 0.3;

% Minimum bin size:
params.MinBinSize = 10;

%% TEST: testResults.m

% Threshold temperature for object detection (Â°C):
params.ThresholdTemperature = 37.5;

% Region of interest for temperature distribution map analysis (m):
params.AnalysisLimits = [10, 45, 0, 2, 0, 2];

% Expected length of voxel clusters (m):
params.ExpectedClusterLength = 0.9;

% Expected spacing between voxel clusters (m):
params.ExpectedClusterSpacing = 2.3;

% Minimum number of elements per cluster:
params.MinClusterSize = 4;
```

Script: *params2.m*

```
% THERMAL MAPPING BY MEANS OF LIDAR AND THERMAL CAMERA SENSOR FUSION
% PARAMETERS FOR DATASET: "Esterno 2"
%
% Davide De Pazzi - Last modified: 20/03/2022

%% INPUT SETTINGS

% Data folder path:
dataPath = ('C:\Users\Davide\Documents\Tesi\Dati\Test\');

% Data folder name:
dataFolder = ('Argine1');

% LIDAR dataset name:
datasetName = ('lungoArgine_parte1_noIMU.bag');

%% PART 1: runLidarCameraSensorFusion.m

% LIDAR dataset reference time (set to 0 to ignore):
params.ReferenceTime = [2022 02 03 15 36 14];

% First valid point cloud in dataset (set to 0 to ignore):
params.FirstScan = 3;

% Last valid point cloud in dataset (set to 0 to ignore):
params.LastScan = 0;

% Maximum LIDAR range (m):
params.MaxLidarRange = 60;

% World reference frame limits (m):
```

```
params.WorldReferenceFrame.xRange = [0 60];
params.WorldReferenceFrame.yRange = [-30 30];
params.WorldReferenceFrame.zRange = [-2 12];

%% PART 2: runRegistrationMerging.m

% Window size for point cloud cropping (m):
params.CroppingWindow = [0, 65, -10, 15, -2, 10];

% Grid size for point cloud filtering (m):
params.BoxFilterGridSize = 0.03;

% Options for point cloud merging (set to { } to ignore):
params.MergingOptions = {'downsample', 0.75};

% Ground-to-LiDAR distance for rover path visualization:
params.LidarHeight = 0;

% Occupancy map cells/meter (set to 0 to ignore):
params.OccMapGridResolution = 0;

%% PART 3: runThermalMapping.m

% Temperature distribution map limits (m):
params.MapLimits = [0, 60, -8, 12, -2, 6];

% Minimum voxel size (m):
params.MinVoxelSize = 0.8;

% Minimum bin size:
params.MinBinSize = 5;
```

Function: *viewPointCloudSequenceHelperFunction.m*

```
function viewPointCloudSequenceHelperFunction(pcSeq, xRange, ...
    yRange, zRange)

% DAVIDE DE PAZZI - Last modified: 20/03/2022

pcplr = pcplayer(xRange, yRange, zRange, 'MarkerSize', 12);
N = size(pcSeq, 1);
f = figure(1);
f.Name = ['Point Cloud Player (frame 1/', num2str(N), ')'];
    view(pcplr, pcSeq{1, 1});
    f.WindowState = 'maximized';
    pause(0.01);
    for i = 1 : size(pcSeq, 1)
        view(pcplr, pcSeq{i, 1});
        f.Name = ['Point Cloud Player (frame ', num2str(i), ...
            '/', num2str(N), ')'];
        pause(0.1);
    end

end
```

Function: *drawSquareHelperFunction.m*

```
function drawSquareHelperFunction(xULC, yULC, L, color)
```

```
% DAVIDE DE PAZZI - Last modified: 20/03/2022

plot([xULC, xULC + L], [yULC, yULC], color);
plot([xULC + L, xULC + L], [yULC, yULC + L], color);
plot([xULC + L, xULC], [yULC + L, yULC + L], color);
plot([xULC, xULC], [yULC, yULC + L], color);

% NOTE: xULC and yULC are the coordinates of the upper left corner

end
```

Function: *roverPathHelperFunction.m*

```
function roverPathHelperFunction(figNum, skipFrames, h0, ...
    tformSeq, temperatureMaps)

% Davide De Pazzi - Last modified: 20/03/2022

figure(figNum)
hold on
t = zeros(size(temperatureMaps, 1), 3);
R = zeros(size(temperatureMaps, 1), 4);
for i = 1 : size(temperatureMaps, 1)
    t(i, :) = tformSeq{temperatureMaps{i, 3}, 1}.Translation;
    R(i, :) = rotm2quat(tformSeq{temperatureMaps{i, 3}, 1}.Rotation');
end
tra = t(1 : skipFrames : end, :) - [0, 0, h0];
rot = R(1 : skipFrames : end, :);
plotTransforms(tra, rot, 'MeshFilePath', 'groundvehicle.stl', ...
    'MeshColor', [0 0.4470 0.7410], 'FrameSize', 1);

end
```

Function: *plotVoxelHelperFunction.m*

```
function plotVoxelHelperFunction(origin, edge, color, alpha)

% DAVIDE DE PAZZI - Last modified: 20/02/2022

u = edge * [1 0 0];
v = edge * [0 1 0];
w = edge * [0 0 1];
corners = [origin; origin + u; origin + v; origin + u + v; ...
    origin + w; origin + u + w; origin + v + w; origin + u + v + w];
faces = [1 2 4 3; 5 6 8 7; 1 2 6 5; 3 4 8 7; 1 3 7 5; 2 4 8 6];
patch('Faces', faces, 'Vertices', corners, 'FaceColor', color, ...
    'FaceAlpha', alpha)
view(3)
axis equal

end
```

Function: *filterVoxelsHelperFunction.m*

```
function voxelsOut = filterVoxelsHelperFunction(voxelsIn, varargin)

% DAVIDE DE PAZZI - Last modified: 20/03/2022

if (size(varargin, 2) == 2) && isequal(varargin{1}, 'minTemp')
```

```
    ind = (cell2mat(voxelsIn(:, 6)) >= varargin{2});
    voxelsOut = voxelsIn(ind, :);
elseif (size(varargin, 2) == 3) && isequal (varargin{1}, 'minDist')
    lim = zeros(1, 3);
    lim(varargin{2}) = varargin{3};
    ind = (cell2mat(voxelsIn(:, 2)) > lim);
    ind = ind(:, varargin{2});
    voxelsOut = voxelsIn(ind, :);
elseif (size(varargin, 2) == 3) && isequal(varargin{1}, 'maxDist')
    lim = zeros(1, 3);
    lim(varargin{2}) = varargin{3};
    ind = (cell2mat(voxelsIn(:, 2)) < lim);
    ind = ind(:, varargin{2});
    voxelsOut = voxelsIn(ind, :);
else
    voxelsOut = voxelsIn;
end

end
```