



UNIVERSITÀ DEGLI STUDI DI PARMA

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

**RICOSTRUZIONE 3D DI OGGETTI
CON SENSORE LASER IN
CONFIGURAZIONE EYE-IN-HAND**

**3D OBJECT RECONSTRUCTION WITH
EYE-IN-HAND LASER SENSOR**

Relatore:

Chiar.mo Prof. STEFANO CASELLI

Correlatori:

Ing. DARIO LODI RIZZINI

Ing. JACOPO ALEOTTI

Tesi di Laurea di:
ANDREA DE PASQUALE

ANNO ACCADEMICO 2010–2011

Indice

1	Introduzione	1
2	Stato dell'arte	5
2.1	Tipologie di sensori	6
2.1.1	Immagini di prossimità	9
2.1.2	Acquisizioni 2D multiple	11
2.2	Tecniche di ricostruzione	12
2.2.1	Vicinanza dei punti in una scansione	13
3	Descrizione del problema	16
3.1	Configurazione sperimentale	16
3.1.1	Robot Comau Smart SiX	17
3.1.2	Laser Sick LMS 400	18
3.1.3	Pinza Schunk PG 70	19
3.2	Sistemi di riferimento	20
3.2.1	Trasformazione di coordinate	21
3.2.2	Calibrazione automatica	23
3.3	Software e librerie utilizzate	24
3.3.1	Driver dei sensori e degli attuatori	25
3.3.2	Librerie per l'elaborazione di dati	27
3.4	Acquisizione e raggruppamento dei dati del sensore laser	29
3.4.1	Procedure di acquisizione	29
3.4.2	Elaborazione dei punti	31
3.5	Identificazione e manipolazione	37

4	Elaborazione di misure di prossimità	40
4.1	Costruzione incrementale di triangolazioni su superfici	41
4.1.1	Triangolazione di Delaunay	41
4.1.2	Costruzione della triangolazione	44
4.1.3	Problemi incontrati	56
4.2	Relazioni di prossimità nelle scansioni	59
4.2.1	Utilizzo della struttura dei dati	60
4.2.2	Costruzione di <i>mesh</i>	69
5	Risultati	75
5.1	Costruzione di <i>mesh</i>	75
5.2	Tempi di ricerca dei vicini	77
5.3	Esperimenti di manipolazione	79
6	Conclusioni	81
	Bibliografia	83

Capitolo 1

Introduzione

La ricostruzione tridimensionale di oggetti è un problema con numerose applicazioni pratiche in differenti campi dell'industria, della medicina, dell'architettura. Si pensi per esempio ad un sistema di controllo qualità capace di determinare se un certo manufatto sia conforme alle specifiche di produzione, oppure ancora ad un apparecchio per la tomografia assiale computerizzata in grado di mostrare al medico un modello 3D del paziente.

Per effettuare la ricostruzione di un oggetto è necessario un sistema di acquisizione, ossia un apparato per trasformare informazioni sul mondo reale in un insieme di valori numerici. Tale sistema si basa sull'uso di uno o più sensori, dispositivi in grado di trasformare una grandezza fisica in un segnale elettronico misurabile. I sistemi di acquisizione maggiormente impiegati dalla comunità di ricerca ed ampiamente presi in rassegna nella letteratura scientifica possono essere suddivisi in due categorie. Alcuni fanno uso di una telecamera di prossimità, sensore intrinsecamente tridimensionale; altri invece adoperano un sensore laser planare, ed acquisiscono l'oggetto avvalendosi di attuatori per variare il punto di osservazione della scena.

In questa tesi è stata impiegata una configurazione del secondo tipo, altresì nota come configurazione *eye-in-hand*, termine metaforico in lingua inglese per identificare rispettivamente il sensore (occhio, *eye*) e la pinza (mano, *hand*). In particolare è stato utilizzato il manipolatore robotico Comau

Smart SiX a sei gradi di libertà, equipaggiandone l'organo terminale con una pinza Schunk PG 70 e con un sensore laser bidimensionale ad alta risoluzione Sick LMS 400.

Il sistema di acquisizione è stato impiegato per immagazzinare informazioni relative ad un insieme di oggetti. Una volta in possesso dei dati numerici relativi all'oggetto esaminato, si possono compiere su di essi varie operazioni preliminari. Per esempio, si rimuove il rumore presente mediante tecniche di analisi statistica e si filtrano eventuali errori di misurazione minori. Quindi, adoperando un modello parametrico, si determina la posizione del piano su cui per ipotesi giacciono gli oggetti. Le misurazioni situate sopra al piano individuato vengono infine suddivise in *cluster*, ossia in gruppi omogenei rispetto alla distanza euclidea.

Ciascuno di questi *cluster* viene quindi sottoposto ad una fase di ricostruzione della superficie originale. In questo lavoro sono state sviluppate e studiate alcune tecniche per effettuare questa procedura. Il primo algoritmo consente la costruzione incrementale della superficie di un oggetto sfruttando l'esistenza di un punto di osservazione (il centro del sensore laser) e la particolare conformazione delle scansioni. Per costruire una triangolazione della superficie si impiegano le suddivisioni, ovvero partizionamenti di varietà 2D nello spazio euclideo 3D in insiemi di vertici, archi e facce. La triangolazione è stata rappresentata mediante il *quad-edge*, una struttura dati precedentemente adoperata da Guibas e Stolfi [18] limitatamente allo spazio bidimensionale.

Un secondo algoritmo fa ricorso invece alle relazioni di prossimità implicite in un insieme di punti acquisiti tramite sensore laser. Per ogni scansione si può determinare un piano, identificato dal centro del laser e dalle misurazioni stesse, nel quale i punti sono ordinati secondo l'angolo formato dal raggio con il sistema di riferimento del sensore. Inoltre, i punti appartenenti a due scansioni acquisite in istanti temporali consecutivi sono con buona probabilità vicini tra loro. L'algoritmo è incentrato pertanto sulla costruzione di una struttura chiamata "grafo di prossimità", in grado di memorizzare le

relazioni di vicinato di ogni punto in maniera incrementale. La superficie dell'oggetto viene ricostruita facendo uso solamente delle informazioni relative alla prossimità dei punti, procedura formalmente dimostrata da Dumitriu [13].

Una volta in possesso dell'oggetto ricostruito, sono stati portati a termine compiti di più alto livello. In primo luogo l'oggetto è stato suddiviso in parti, rappresentandone la topologia mediante grafi di Reeb, ed utilizzando tali informazioni per classificare l'oggetto. Successivamente sono state studiate le varie prese (*grasp*) possibili, esaminando ogni parte indipendentemente dalle altre seguendo un approccio tipico degli esseri umani. Infine sono state effettuate prove sperimentali di manipolazione, sia in ambiente simulato che utilizzando il robot Comau Smart SiX e la pinza Schunk PG 70.

Nel capitolo 2 si descrive lo stato dell'arte, sia per quanto riguarda le varie tipologie di sensori, sia per le tecniche di ricostruzione tridimensionale attualmente esistenti. I sensori vengono distinti in due macrocategorie, attivi e passivi, e per ciascuna categoria si analizzano le configurazioni più comuni. In seguito si presentano gli ultimi sviluppi riguardanti l'uso di telecamere di prossimità e le tecniche di fusione di più acquisizioni laser bidimensionali. Dopo una breve panoramica sugli algoritmi di ricostruzione (*power crust*, *ball pivoting*), si esamina una recente pubblicazione scientifica che introduce alcuni aspetti di cui ci si è avvalsi in questa tesi.

Le specifiche del sistema di acquisizione adoperato vengono fornite all'interno del capitolo 3. Alla descrizione del manipolatore robotico, del sensore laser e dell'organo utensile, segue un'illustrazione dei vari sistemi di riferimento e delle trasformazioni tra i medesimi. Si discute inoltre della calibrazione del sistema e delle possibili procedure per automatizzarla. Nel prosieguo vengono elencati i programmi e le librerie informatiche delle quali si è a vario scopo usufruito. Si riportano quindi il funzionamento dettagliato della procedura di acquisizione ed i principali meccanismi per l'elaborazione dei punti, introducendo poi il problema della ricostruzione della superficie di un

oggetto. Infine sono mostrate le operazioni successive che fanno uso del modello ricostruito: la suddivisione in parti, lo studio delle prese (*grasp*), la manipolazione in ambiente sia virtuale che reale.

Nel capitolo 4 si elencano gli algoritmi di ricostruzione tridimensionale sviluppati in questa tesi, aventi in comune il fatto di ricorrere ad informazioni note sul sistema di acquisizione. In primo luogo si descrive un approccio basato sulla costruzione incrementale di una triangolazione di Delaunay per una varietà bidimensionale nello spazio. Si delineano quindi alcuni problemi emersi a causa dei criteri e predicati geometrici adottati. L'attenzione si sposta pertanto su altre tecniche, facenti impiego delle relazioni di prossimità tra i punti, sia della medesima scansione, sia tra più scansioni temporalmente consecutive. Viene introdotto il concetto di "grafo di prossimità" e ne viene esposto il funzionamento. Per finire, si mostra come è possibile trarre vantaggio dal suddetto grafo al fine di generare una superficie.

I risultati relativi agli algoritmi trattati vengono mostrati nel capitolo 5, mentre nel capitolo 6 si delineano le conclusioni e gli sviluppi futuri di questo lavoro di tesi.

Capitolo 2

Stato dell'arte

La creazione di una rappresentazione virtuale di un oggetto reale è un problema molto sentito nell'ambito della robotica. Lo dimostrano la presenza di numerosi algoritmi e strumenti efficaci ed affidabili, nonché il continuo sviluppo di nuove tecniche per l'acquisizione e l'elaborazione di dati sensoriali.

L'acquisizione viene di norma effettuata mediante un sistema hardware appositamente predisposto. Nuvole di punti, ovvero insiemi di più punti, sono prelevate tramite interazione diretta o indiretta con la superficie dell'oggetto. Le tecniche di acquisizione fanno uso di sensori basati su principi fisici differenti (ottici, acustici, etc), ciascuno dei quali presenta particolari vantaggi e svantaggi. La progettazione di un tale sistema deve tenere in considerazione molti fattori, spesso tra loro in contrasto, ed alcune significative limitazioni. In generale, non esiste un sistema di acquisizione valido in ogni situazione ambientale e per ogni possibile compito. Trovare il giusto compromesso dipende in maggior misura dalle caratteristiche richieste dalla particolare applicazione.

L'elaborazione consiste invece nella modifica dei punti precedentemente acquisiti, ovvero l'esecuzione di svariati filtraggi, estrazioni, integrazioni e trasformazioni di altro tipo. La maggior parte delle tecniche ha come obiettivo la ricostruzione virtuale della superficie dell'oggetto esaminato, o in alternativa l'estrazione di informazioni e caratteristiche utili. A titolo d'esempio si

citano la rimozione del rumore e dei dati non validi, l'estrazione di regioni di interesse, la fusione di due o più nuvole acquisite da differenti punti di vista, la semplificazione delle successive elaborazioni mediante riduzione del numero totale di punti.

2.1 Tipologie di sensori

Il componente chiave di un sistema di acquisizione 3D è sicuramente il sensore, in base al quale si impiegano infatti metodi e strategie di acquisizione differenti. Ogni sensore utilizza una particolare tecnologia per effettuare una o più misurazioni del mondo reale, osservando l'oggetto oppure interagendo con la sua superficie. Una prima distinzione può essere incentrata proprio sull'interazione del sensore con gli oggetti da esaminare (figura 2.1).

I sistemi passivi funzionano senza interferire con il mondo reale, ossia si limitano a ricevere l'energia proveniente dall'ambiente senza emetterne di propria. L'esempio più calzante a cui si può pensare è una telecamera, che necessita soltanto della luce naturale o di un qualche tipo di illuminazione artificiale (non strettamente prodotta dal sensore stesso).

I sistemi attivi trasmettono invece una parte di energia direttamente all'oggetto da acquisire, e per tale motivo richiedono solitamente più potenza di quelli passivi. Alla fase di emissione di energia controllata, segue una procedura di misurazione della reazione dell'ambiente. Alcune tecniche prevedono l'impiego di un fascio laser, un impulso sonar, un raggio infrarosso, etc.

Tra i sistemi passivi in figura 2.1, troviamo due o più telecamere in coppia stereo. In questa configurazione, la terza dimensione viene ricostruita sfruttando la triangolazione tra le varie immagini. Il carico computazionale può essere piuttosto pesante, siccome per risalire alla distanza bisogna estrarre numerose caratteristiche visive ed abbinarle correttamente tra loro. Normalmente si utilizza un *setup* di questo tipo quando si necessita delle immagini anche per altri scopi.

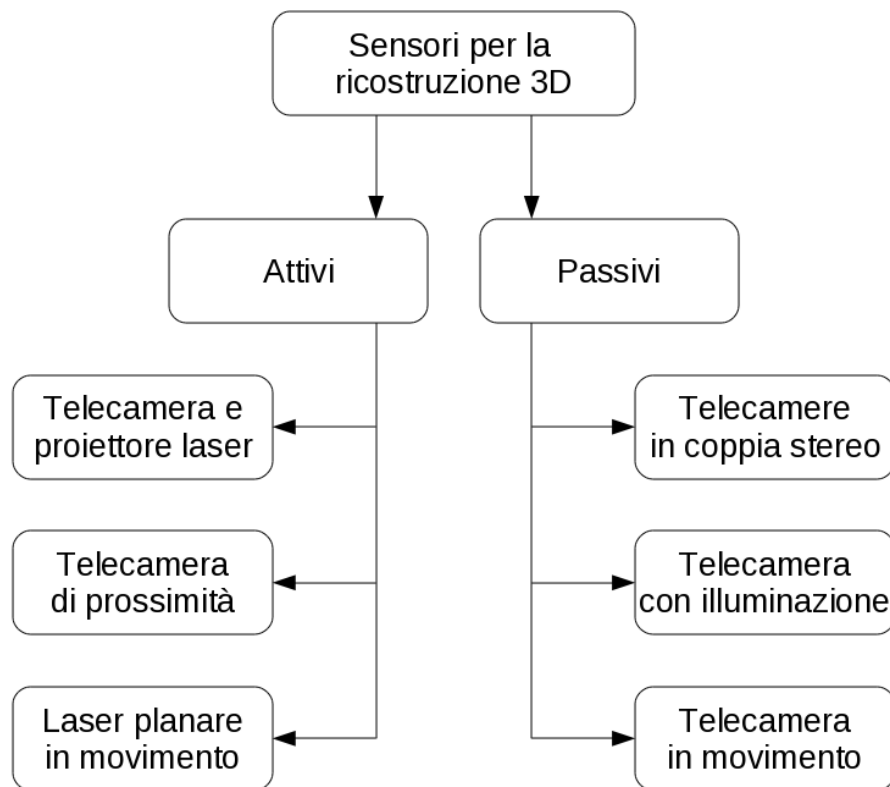


Figura 2.1: Categorie di sensori di acquisizione attivi e passivi, finalizzati ad ottenere dati per effettuare la ricostruzione tridimensionale di oggetti.

Per ridurre ulteriormente i costi, si può pensare di utilizzare una singola telecamera. La profondità viene calcolata variando le condizioni di luminosità dell'ambiente di acquisizione, grazie ad alcune ipotesi sulle normali alla superficie; oppure, variando la posa del sensore o dell'oggetto in esame. Questa soluzione ha molti difetti: *in primis* è molto sensibile a cambiamenti ambientali, inoltre non è adatta a superfici complesse o dall'aspetto non uniforme.

La combinazione di una telecamera (passiva) con un proiettore laser è invece considerata un sensore attivo, ove le due parti di emissione e misurazione sono tra loro separate. Conoscendo le pose di entrambi gli elementi del sensore, si può ricostruire un profilo rettilineo dell'oggetto grazie alla triangolazione. Tramite il movimento del fascio proiettato, oppure tramite lo scorrere di un nastro trasportatore, i profili vengono composti tra loro per dare luogo ad una rappresentazione d'insieme.

Gli altri sensori attivi presi in analisi hanno in comune il fatto di essere sensori di prossimità. Simili apparecchi funzionano misurando la distanza, in termini di tempo di volo, che intercorre tra loro stessi e l'ostacolo intercettato dal segnale emesso.

I sensori laser possono funzionare in due modi: emettendo un impulso e misurando in quanto tempo si riceve una risposta; oppure emettendo un segnale modulato in ampiezza e misurando la variazione di fase tra l'originale trasmesso e quello ricevuto.

Le telecamere di prossimità misurano il tempo di volo mediante una matrice bidimensionale di sensori. I tempi vengono determinati impostando la chiusura di un otturatore ad intervalli regolari. In alcuni modelli si può associare ad ogni pixel anche informazioni di luminosità o colore, in aggiunta al valore corrispondente alla distanza dall'oggetto osservato.

Per loro natura le telecamere di prossimità sono sensori tridimensionali, mentre i *laser scanner* possono essere sia 3D che 2D. Ciononostante, questi ultimi sono in grado di acquisire dati tridimensionali, avvalendosi di uno o più attuatori per variare il punto di osservazione della scena. Tramite gli

attuatori si controlla il processo di scansione, al fine di acquisire per esempio oggetti complessi. Pianificare i punti di vista ottimali per conseguire una buona ricostruzione dell'oggetto è un problema complesso.

2.1.1 Immagini di prossimità

Maldonado ed altri [25] utilizzano una telecamera a tempo di volo nel loro lavoro sulla presa di oggetti precedentemente sconosciuti. In un caso del genere non si possono sfruttare modelli geometrici dell'oggetto, siccome non se ne conosce a priori la forma. Il sistema di percezione deve quindi essere in grado di fornire al manipolatore almeno un'informazione parziale sulla posizione dell'oggetto. La presa verrà poi perfezionata utilizzando i sensori di coppia sulle dita della mano.

La telecamera, installata su un'unità *pan-tilt* a bordo del robot, produce dati tridimensionali molto rumorosi. Dati che sono inoltre incompleti, poiché relativi alla sola faccia anteriore degli oggetti. Bisogna pertanto ipotizzare che gli oggetti giacciono su un piano, come quello di un tavolo da cucina. L'algoritmo sviluppato dagli autori ottimizza le pose delle prese basandosi sulla distribuzione gaussiana dei punti acquisiti, e sopperisce alla rumorosità del sensore.

Un problema simile era già stato affrontato da Klank [21], impiegando sempre una telecamera a tempo di volo, appoggiandosi però ad un *database* di modelli di oggetti precedentemente acquisiti. Limitandosi ad adattare la percezione della realtà ad un modello virtuale, si eliminano gli sforzi computazionali necessari per riconoscere e ricostruire l'oggetto stesso.

In questo caso l'immagine di prossimità serve soltanto a ricostruire la terza dimensione. La ricerca viene poi effettuata utilizzando l'immagine RGB arricchita con questa ulteriore informazione. Sebbene con le limitazioni sopra riportate, anche in questo caso si è in grado di manipolare l'oggetto.

In alternativa, Marton ed altri [26] hanno raggiunto il medesimo risultato, irrobustendolo grazie ad ulteriori assunzioni. Siccome osservando la scena con una *range camera* si vede la parte frontale dell'oggetto, sono necessarie

alcune ipotesi sul lato nascosto. Per esempio, numerosi oggetti di uso comune possiedono un asse di simmetria. L'algoritmo sviluppato però non è sufficientemente rapido per essere adoperato nella pratica.

Steder, Rusu ed altri [37] affrontano invece il problema della segmentazione di immagini di prossimità. In letteratura sono presenti molti lavori per l'estrazione di caratteristiche da un'immagine nel visibile, ma lo stesso non può essere detto per quanto riguarda le immagini con informazioni sulla profondità. Essi espongono un metodo valido anche con dati incompleti e dipendenti fortemente dal punto di vista: l'algoritmo NARF (*normal aligned radial feature*).

Tale algoritmo si basa sull'estrazione di punti di interesse, che devono sia avere una normale ben stabile sia essere in prossimità dei bordi di un oggetto. Per ogni punto estratto si calcola un descrittore, che tiene in considerazione i due parametri sopra esposti ed al tempo stesso renda facili ed efficienti le operazioni di confronto. I risultati ottenuti dimostrano che NARF è attualmente uno dei migliori algoritmi per l'estrazione di caratteristiche da immagini di prossimità.

Foix ed altri [16] si preoccupano del problema dell'integrazione di più immagini di prossimità. Le telecamere a tempo di volo sono rumorose ed hanno una bassa risoluzione, ma riescono a fornire fino a 25 *frame* al secondo. Collocando una telecamera sul braccio di un manipolatore, si possono ottenere numerose scansioni facendo muovere quest'ultimo intorno all'oggetto con traiettoria circolare.

Le immagini così acquisite devono essere combinate tra loro. Tra i metodi di combinazione validi, gli autori citano il ricorso alla cinematica inversa del manipolatore, insieme ad una calibrazione precisa della telecamera rispetto all'organo terminale. Di maggiore interesse però è una variante dell'algoritmo di registrazione ICP (*iterative closest point*) ancora non utilizzata nello spazio 3D, insieme a metodi di SLAM derivati dalla visione artificiale.

2.1.2 Acquisizioni 2D multiple

Klimentjew [23] svolge un interessante confronto dei sensori tridimensionali basati sul movimento di un *laser scanner* 2D. In particolare vengono esaminati un'unità *pan-tilt* ed un robot manipolatore, grazie ai quali l'ambiente può essere ricostruito per mezzo di trasformazioni geometriche. Questo approccio è piuttosto nuovo e soltanto recentemente taluni gruppi di ricerca hanno incominciato esperimenti simili.

Le nuvole di punti ottenute in entrambi i casi sono a detta dell'autore molto soddisfacenti, e risultano un'alternativa valida ad un più costoso sensore laser tridimensionale per compiti come la prevenzione di collisioni, l'approccio alla presa o la manipolazione di oggetti. Le prestazioni non sono però al livello dei laser 3D o delle telecamere di prossimità, a causa ovviamente del maggior numero di scansioni da effettuare per l'acquisizione dello stesso ambiente.

Il problema della segmentazione viene ripreso da Strom e colleghi [38], combinando l'uso di un laser 2D su attuatore *pan-tilt* con una telecamera a colori. Per essere precisi, la segmentazione in base al solo colore viene irrobustita mediante informazioni sulla profondità. Questo è un altro esempio di come un laser planare possa essere adoperato in configurazione 3D per compiti di alto livello (riconoscimento di oggetti, classificazione della scena).

Torabi e Gupta [40] presentano un lavoro sulla ricostruzione di un modello 3D di un generico oggetto. Un sensore laser planare, montato su un manipolatore robotico a sei gradi di libertà, viene attuato con una determinata strategia. Nessuna supposizione viene fatta, né per quanto riguarda la forma dell'oggetto, né sulla struttura dello spazio nel quale esso è collocato.

Per tale motivo si utilizza un particolare algoritmo, chiamato *next best view* (NBV), in grado di determinare quale sia la posa migliore per acquisire nuove informazioni sull'oggetto. Un pianificatore determina quindi una serie di movimenti, priva di collisioni, da far compiere al manipolatore al fine di raggiungere la configurazione desiderata. Mediante un numero relativamente

basso di iterazioni, si è in grado di formare una nuvola di punti sufficiente per ricostruire l'oggetto in esame.

2.2 Tecniche di ricostruzione

Come già evidenziato, in molte applicazioni la ricostruzione della superficie a partire da una nuvola di punti è un requisito fondamentale per svolgere compiti di più alto livello. I metodi di ricostruzione attualmente presenti in letteratura, e considerati in questo breve riassunto, hanno in comune un approccio basato sulla geometria computazionale. In particolare, utilizzando ed interpolando un insieme di punti, sono in grado di determinare una superficie come triangolazione di Delaunay nello spazio tridimensionale. Infatti, la generazione di una triangolazione in modo efficiente è spesso determinante per ridurre il costo computazionale di tali algoritmi.

Una panoramica delle tecniche di ricostruzione basate sulla triangolazione di Delaunay è stata redatta da Cazals e Giesen [11], prestando particolare attenzione alle ipotesi ed ai requisiti geometrici ed algoritmici di ciascun metodo. Da tale analisi sono emerse principalmente tre macro-categorie, elencate nel seguito assieme ad alcuni popolari algoritmi.

Le tecniche basate sul piano tangente alla superficie funzionano ipotizzando che per ogni punto campionato passi un piano con una determinata normale. La ricostruzione di ciascuna faccia viene effettuata sul relativo piano tangente, ed infine le facce vengono unite tra loro. Gran parte di questi metodi derivano dall'algoritmo sviluppato da Boissonnat [8].

Altri metodi invece limitano la triangolazione di Delaunay ad un sottoinsieme di \mathbb{R}^3 in grado di approssimare l'oggetto, come nel caso degli algoritmi *crust* e *cocone* [1, 2]. La ricostruzione mediante etichettatura dei volumi interni od esterni, suddividendo per esempio lo spazio con la tetraedrizazione di Delaunay, può essere considerata come un'evoluzione delle due tecniche fin qui introdotte.

Proseguendo con l'analisi, l'algoritmo *power crust* di Amenta [3, 4] utilizza la trasformata dell'asse mediale per approssimare la superficie. Questa trasformata rappresenta un qualsiasi oggetto come lo scheletro delle sfere di raggio massimale completamente contenute nella superficie del medesimo. Il *power crust*, a partire dai punti che campionano la superficie, costruisce un'approssimazione dello scheletro. Da quest'ultimo, con un'ulteriore approssimazione, utilizza le sfere per ricostruire la superficie.

Per finire, si menziona l'algoritmo di *ball pivoting* ideato da Bernardini ed altri [7]. Tale metodo si basa sul principio che tre punti formano un triangolo se una circonferenza di raggio α li attraversa senza includere altri punti al suo interno. Il processo viene ripetuto ruotando la circonferenza intorno ad un lato del triangolo finché non tocca un nuovo punto.

2.2.1 Vicinanza dei punti in una scansione

Nel caso di acquisizioni bidimensionali multiple, non v'è nessuno tra i metodi sopra introdotti che faccia uso di informazioni aggiuntive, quali ad esempio la presenza di un punto di osservazione. Fa eccezione l'algoritmo sviluppato da Klasing ed altri per segmentare in tempo reale letture di prossimità disomogenee come appartenenti ad oggetti distinti [22].

In contrasto con la maggior parte dei lavori esistenti, esso non richiede l'impiego di immagini di prossimità. È sufficiente infatti disporre di un sensore laser planare con posizione ed orientazione variabili. In passato, molti sviluppatori di piattaforme robotiche hanno adottato soluzioni di questo tipo, anche semplicemente collegando sensori laser ad attuatori *pan-tilt*.

Nonostante negli ultimi tempi l'impiego di telecamere di prossimità sia in continuo aumento, anche grazie a soluzioni commerciali a basso costo come il Microsoft Kinect, presentano comunque un campo visivo, una precisione ed una risoluzione ancora modeste. Per alcune applicazioni, inoltre, elaborare incrementalmente i nuovi punti man mano che vengono aggiunti può risultare una strategia migliore.

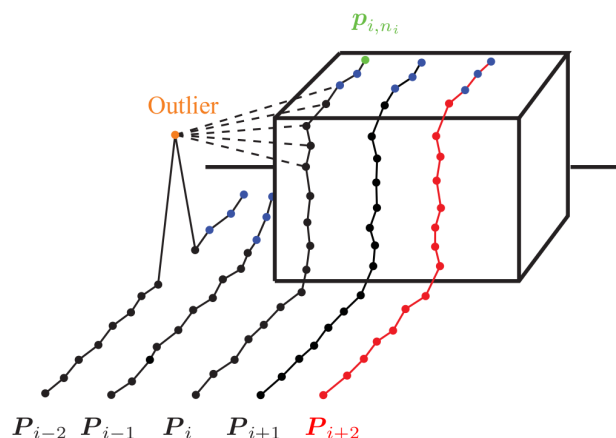


Figura 2.2: Sequenza continua di scansioni bidimensionali.

Il problema affrontato è il recupero dei punti più vicini ad un determinato punto, di particolare interesse in molti campi tra cui la ricostruzione 3D. Il *kd-tree* [6] è una delle strutture dati più efficienti per la ricerca dei vicini nello spazio tridimensionale. Il suo principale svantaggio consiste nella staticità: aggiungere nuovi punti (o rimuoverne di vecchi) infatti non è efficiente, poiché invalida la condizione di bilanciamento dell'albero.

Gli autori dell'algoritmo, chiamato *continuous nearest neighbours*, hanno deciso di sfruttare alcune euristiche sul sensore di prossimità. Si pensi ad un manipolatore robotico equipaggiato con un *laser scanner* solidale allo strumento di lavoro. Al muoversi dell'intero robot, una tale configurazione produce un flusso di dati variabili (figura 2.2).

È interessante osservare che, per ogni punto, possiamo determinare un certo numero di vicini senza interrogare nessun algoritmo. Infatti, ciascuno possiede sia punti adiacenti all'interno della medesima scansione a cui esso appartiene, sia punti vicini appartenenti alle precedenti scansioni. In altre parole, all'interno di una singola scansione laser le letture sono limitrofe per costruzione del sensore, mentre scansioni temporalmente consecutive saranno vicine tra loro.

Klasing definisce quindi una finestra quadrata di dimensioni

$$w_k = 2 \max \left(\left\lfloor \frac{k}{4} \right\rfloor, 1 \right) + 1,$$

all'interno della quale vanno cercati i k -vicini di ogni punto. A tale proposito, si calcola la distanza euclidea tra il punto ed i candidati vicini nella finestra; i k punti a distanza minore vengono considerati essere i k -vicini del punto.

Non è garantito che la finestra contenga i punti in assoluto più vicini, come accade in presenza di rumore (*outliers*). Ciononostante, se l'oggetto è stato scandito in maniera continua, nella maggior parte dei casi i vicini reali sono perfettamente coincidenti con quelli trovati grazie a questo metodo.

Capitolo 3

Descrizione del problema

In questo capitolo si descrive il sistema di acquisizione utilizzato per raccogliere dati sperimentali. Nel dettaglio, si analizzano le varie componenti hardware e le relazioni tra i sistemi di riferimento, e si discutono alcune procedure di calibrazione. In seguito, si trattano i principali programmi e librerie esterne utilizzate per l'elaborazione dei punti.

Si descrive quindi passo dopo passo la procedura di acquisizione, e si illustrano le fasi di elaborazione dei punti antecedenti alla ricostruzione. Successivamente viene brevemente introdotto il lavoro svolto, che consiste nello sviluppo di algoritmi di ricostruzione tridimensionale dell'oggetto a partire dai punti ottenuti mediante acquisizione.

Si accennano infine le fasi consecutive, ovvero la suddivisione dell'oggetto ricostruito in parti elementari, l'analisi del grafo di connettività tra gli elementi, lo studio delle prese (*grasp*). Per concludere, viene mostrato un tipico esperimento di manipolazione in ambiente sia virtuale che reale.

3.1 Configurazione sperimentale

L'hardware adoperato consiste in un manipolatore robotico antropomorfo a 6 gradi di libertà, un sensore laser per la misurazione delle distanze ed una pinza per la presa di oggetti. Nelle varie sottosezioni si trovano informazioni aggiuntive per ciascun componente.



(a) Comau Smart SiX



(b) Comau C4G

Figura 3.1: Il robot manipolatore e la relativa unità di controllo.

3.1.1 Robot Comau Smart SiX

Il Comau Smart SiX (figura 3.1a) è un robot destinato ad applicazioni industriali di manipolazione leggera e saldatura ad arco. La struttura del robot è del tipo antropomorfo, con 6 gradi di libertà. Esso dispone di sei giunti rotoidali, tre per la spalla e tre per il polso, comandati da motori *brushless* e misurati da *encoder* di posizione ad alta risoluzione.

Il robot è predisposto al montaggio di numerosi dispositivi opzionali. In corrispondenza del polso si trova infatti una flangia standard per l'attacco di attrezzi. La capacità di carico non è elevata e corrisponde ad un massimo di 6 kg sul polso e 10 kg sull'avambraccio. Viste le particolari applicazioni per cui è progettato, il robot dispone però di una grande ripetibilità, pari a ± 0.05 mm. Lo spazio operativo è all'incirca una sfera di raggio 1.5 m, con sbraccio massimo orizzontale di 1.4 m.

Il controllo avviene grazie all'unità Comau C4G (figura 3.1b), in grado di comandare motori fino a 600 V e di gestire robot configurati con 6, 8 o 10 assi interpolati. Essa dispone di terminale di programmazione con interfac-

cia utente grafica e comuni porte di comunicazione (seriale, USB, ethernet). L'unità stessa può diventare un nodo di rete ethernet per la programmazione, la gestione e la diagnostica remota.

La programmazione dell'apparecchiatura può avvenire tramite terminale grafico oppure mediante il linguaggio PDL2. Tale linguaggio, simile al Pascal, possiede speciali istruzioni per le applicazioni robotiche, come: il movimento dei bracci del robot; l'invio e la ricezione di informazioni; il rispetto dei vincoli temporali imposti; il controllo di errori o altre condizioni speciali. Il controllore C4G può anche eseguire parallelamente più programmi, per gestire in contemporanea differenti aspetti della stessa applicazione.

3.1.2 Laser Sick LMS 400

Il Sick LMS 400 è un sensore LIDAR (*laser imaging detection and ranging*), ovvero un rilevatore della distanza facente uso di un impulso laser. Per questo motivo, è catalogabile tra gli strumenti di acquisizione attivi e senza contatto diretto con gli oggetti.

Il principio fisico alla base del suo funzionamento è lo sfasamento tra il raggio inviato ed il raggio ricevuto, misurabile grazie al tempo di propagazione del fascio luminoso ed alla lunghezza d'onda usata (650 nm). La differenza di fase è convertita in frequenza, e da questa frequenza il sensore è in grado di risalire alla distanza a cui si trova l'oggetto. Maggiori dettagli riguardanti il funzionamento del sensore possono essere trovati in [27].

Il sensore permette di misurare distanze all'interno di un'area bidimensionale (figura 3.2). Queste distanze vanno da un minimo di 0.7 m ad un massimo di 3 m dal centro di emissione dei raggi, marcato sull'involucro con un apposito simbolo. Il campo visivo copre un intervallo angolare massimo di 70°, con risoluzione compresa tra 0.1° e 1°, ed angoli iniziale e finale configurabili dall'utente.

L'errore di misurazione sistematico è ± 4 mm, mentre quello statistico varia tra ± 3 e ± 10 mm in base alla distanza dell'oggetto. L'errore angolare invece è compreso tra $\pm 0.1^\circ$. La frequenza di scansione è selezionabile tra 150

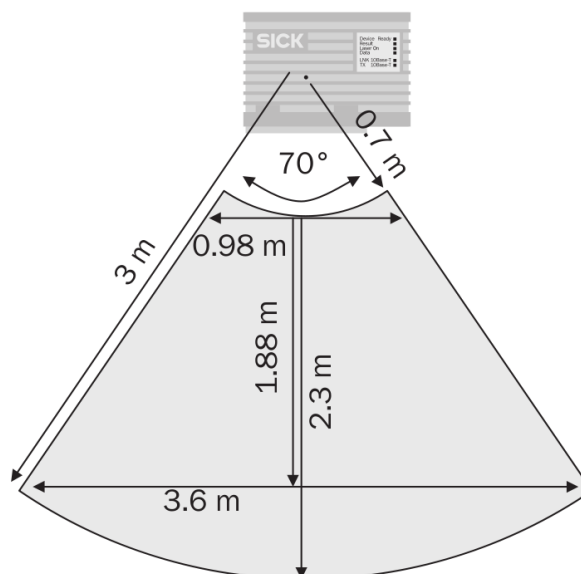


Figura 3.2: Area di lavoro del laser Sick LMS 400.

e 500 Hz. Le interfacce disponibili per il trasferimento dei dati di prossimità ad un calcolatore sono RS-232, RS-422 ed Ethernet 10Base-T. Per ulteriori dettagli, si rimanda al manuale tecnico [36] fornito dal costruttore.

Durante gli esperimenti, il laser Sick LMS 400 è stato impostato su una frequenza di scansione di 190 Hz, una risoluzione angolare di 0.5° , un angolo iniziale di 55° ed uno finale di 125° . L'involucro contenente il sensore è stato fissato all'organo terminale del manipolatore mediante una piastra metallica ad esso solidale.

3.1.3 Pinza Schunk PG 70

Il *gripper* Schunk PG 70, schematizzato in figura 3.3, viene adoperato nelle fasi di manipolazione successive alla ricostruzione di un modello dell'oggetto. Si tratta di una pinza servo-elettrica a due dita parallele, piuttosto versatile per quanto riguarda le aree di applicazione. Il peso contenuto, circa 1.4 kg, ne consente il montaggio sul manipolatore Comau Smart SiX.

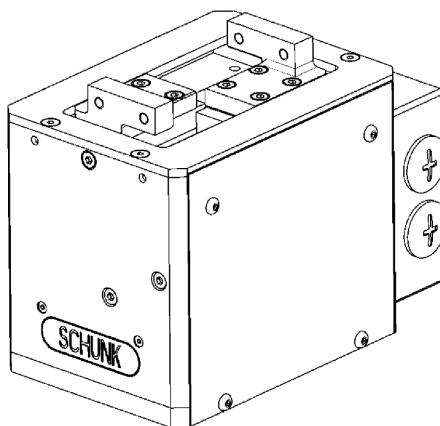


Figura 3.3: Disegno della pinza Schunk PG 70 (dita non presenti).

Questa pinza possiede un controllo molto preciso per quanto riguarda la forza di chiusura, variabile da 30 a 200 N. Ciascuna delle due dita può aprirsi e chiudersi percorrendo una corsa di 35 mm in circa 1.1 s alla velocità massima. Il carico sollevabile è pari ad 1 kg, ed il produttore raccomanda di non eccedere troppo oltre questo valore (si consulti [34]).

Per comunicare con lo strumento, ed inviare comandi di apertura, chiusura e posizione, si possono utilizzare i protocolli Profibus-DP, CAN bus oppure RS-232. Quest'ultimo è proprio quello impiegato nel *setup* del laboratorio, grazie all'uso di un singolo cavo ibrido trasportante anche la tensione di alimentazione di 24 V. La pinza è stata inoltre agganciata al robot manipolatore mediante un elemento connettivo cilindrico.

3.2 Sistemi di riferimento

Quando l'organo terminale di un manipolatore è equipaggiato con un sensore visivo, si parla di configurazione *eye-in-hand*, letteralmente "occhio nella mano". Infatti, le due parti assumono rispettivamente il nome di "mano" ed "occhio" in base alla funzione svolta.

Spesso il sensore è costituito da una o più telecamere, ma esistono anche impianti che fanno uso di un misuratore di prossimità con raggi laser. La

configurazione hardware adoperata in questa tesi prevede proprio l'uso di un sensore laser planare, il Sick LMS 400 precedentemente introdotto. Al variare della posizione dell'organo terminale del manipolatore, al quale il sensore è solidale, varia anche il piano di scansione al quale si riferiscono le misurazioni.

3.2.1 Trasformazione di coordinate

Si pone innanzitutto il problema di trasformare le misure stesse da coordinate polari a coordinate cartesiane sul piano di scansione. Conoscendo le impostazioni del sensore, questa operazione è piuttosto semplice. Avendo utilizzato angoli iniziale $\theta_i = 55^\circ$ e finale $\theta_f = 125^\circ$, con due misurazioni ogni grado, si ottengono $N = 2(\theta_f - \theta_i) + 1 = 141$ letture $\{\rho_0, \dots, \rho_{N-1}\}$. Per risalire alle coordinate cartesiane (x_k, y_k) è sufficiente calcolare $x_k = \rho_k \sin(\theta_k \frac{\pi}{180})$ ed $y_k = \rho_k \cos(\theta_k \frac{\pi}{180})$, ove $\theta_k = \theta_i + \frac{k}{2}$.

In secondo luogo, bisogna far in modo che le coordinate così trovate siano riferite non al centro del sensore laser, che è in continuo movimento, ma ad un punto stabile quale per esempio la base del robot stesso. Si deve tener conto dunque di due trasformazioni: una tra il punto alla base del manipolatore e l'organo terminale, l'altra tra l'organo terminale ed il centro del sensore.

Essendo il laser solidale con l'attrezzo, tra i due esiste una rototraslazione costante, esprimibile mediante una matrice di dimensione 4×4 . Le misure sperimentali hanno determinato l'esistenza delle seguenti traslazioni e rotazioni:

- traslazione tra il centro del sensore ed il centro dell'organo terminale (espressa in millimetri)

$$d_l^t = \begin{bmatrix} 12.0 \\ -105.3 \\ 107.6 \end{bmatrix};$$

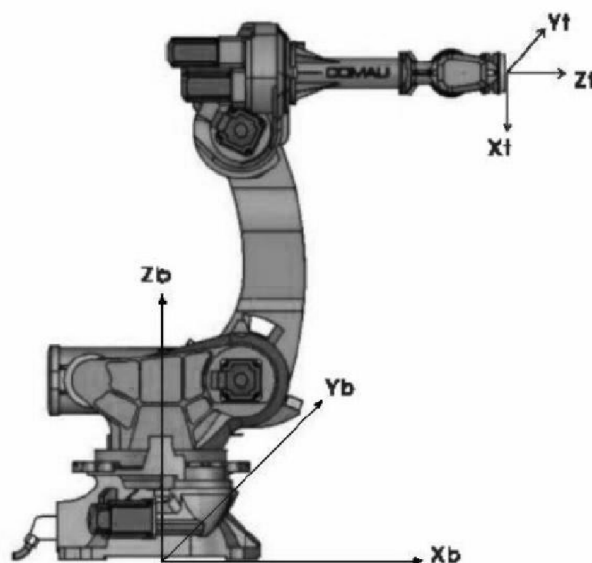


Figura 3.4: Relazione tra la terna di base b e la terna dello strumento t (*tool*).

- rotazione di 90° intorno all'asse x , dal laser all'organo terminale

$$R_l^t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\frac{\pi}{2}) & -\sin(\frac{\pi}{2}) \\ 0 & \sin(\frac{\pi}{2}) & \cos(\frac{\pi}{2}) \end{bmatrix};$$

- infine ulteriore rotazione del laser per $\phi = 1.03914^\circ$ sull'asse y

$$R_l = \begin{bmatrix} \cos(\phi \frac{\pi}{180}) & 0 & \sin(\phi \frac{\pi}{180}) \\ 0 & 1 & 0 \\ -\sin(\phi \frac{\pi}{180}) & 0 & \cos(\phi \frac{\pi}{180}) \end{bmatrix}.$$

Combinando questi elementi, si ottiene quindi una trasformazione costante tra lo strumento ed il sensore:

$$T_l^t = \begin{bmatrix} R_l R_l^t & d_l^t \\ 0 & 1 \end{bmatrix}.$$

Ovviamente, quando il manipolatore è in movimento, la posa dell'organo terminale rispetto alla base del robot cambia nel tempo (figura 3.4). La

matrice relativa a questa trasformazione sarà quindi variabile, e composta da un vettore distanza

$$d_t^b(x, y, z) = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

e da una matrice di rotazione

$$R_t^b(\alpha, \beta, \gamma) = \begin{bmatrix} \cos \alpha \cos \beta \cos \gamma - \sin \alpha \sin \gamma & -\cos \alpha \cos \beta \sin \gamma - \sin \alpha \cos \gamma & \cos \alpha \cos \beta \\ \sin \alpha \cos \beta \cos \gamma + \cos \alpha \sin \gamma & -\sin \alpha \cos \beta \sin \gamma + \cos \alpha \cos \gamma & \sin \alpha \sin \beta \\ -\sin \beta \cos \gamma & \sin \beta \sin \gamma & \cos \alpha \end{bmatrix}$$

secondo la notazione ZYZ, poiché gestita in questo modo dall'unità di controllo Comau C4G. Per maggiori informazioni sulle notazioni minime di orientazione si invita a consultare [17]. Nel complesso, possiamo riassumere la trasformazione variabile come

$$T_t^b(x, y, z, \alpha, \beta, \gamma) = \begin{bmatrix} R_t^b(\alpha, \beta, \gamma) & d_t^b(x, y, z) \\ 0 & 1 \end{bmatrix}.$$

In sostanza, si può quindi scrivere in forma piuttosto compatta la trasformazione che fa corrispondere ogni lettura metrica del sensore ad un punto nello spazio 3D, avente come origine la terna di base del robot manipolatore:

$$T_l^b(x, y, z, \alpha, \beta, \gamma) = \begin{bmatrix} R_t^b(\alpha, \beta, \gamma) & d_t^b(x, y, z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_l R_l^t & d_l^t \\ 0 & 1 \end{bmatrix}.$$

3.2.2 Calibrazione automatica

Date due scansioni del medesimo oggetto relative a due punti di osservazione differenti, è lecito chiedersi se sia possibile stabilire le relazioni di rotazione e traslazione tra di esse. In questo modo si potrebbe determinare se le trasformazioni precedentemente introdotte siano accurate, ed in caso contrario correggerle provvedendo alla loro calibrazione.

Supponendo di avere un certo numero di punti, misurati prima da una posizione, poi da un'altra differente. Scoprire la trasformazione che lega i due punti di osservazione è noto come il problema dell'orientazione assoluta.

Questo problema è di grande importanza in un sistema *eye-in-hand*, per determinare molto accuratamente la trasformazione che intercorre tra l'occhio e la mano.

Una tale trasformazione può essere rappresentata mediante un movimento rigido, e quindi decomposta nella combinazione di una traslazione ed una rotazione. Per determinare questi sei gradi di libertà servono necessariamente almeno due punti, poiché è necessario risolvere 6 equazioni in 6 incognite.

Poiché spesso le misurazioni sono imperfette, si può conseguire una maggiore precisione utilizzando più punti. Il problema diventa quindi trovare un movimento rigido in grado di rendere minima la somma dei quadrati degli errori compiuti su ogni singolo punto.

Esistono numerosi metodi empirici, grafici e numerici per ottenere una soluzione approssimata, migliorabile mediante l'esecuzione di più iterazioni. Berthold Horn, in un articolo del 1988 [20], ha però determinato una soluzione in forma chiusa per il problema dei minimi quadrati per la stima tra due pose.

In questa tesi il problema dell'orientazione assoluta è purtroppo rimasto aperto, in maggior ragione a causa della difficoltà di acquisizione dei medesimi punti da due pose differenti. Disponendo infatti del solo laser planare, e non di una telecamera o di un sensore di prossimità bidimensionale, non si ha la certezza che i punti acquisiti siano esattamente gli stessi, invalidando così ogni possibile risultato ottenuto.

3.3 Software e librerie utilizzate

Per la scrittura dei programmi necessari si è fatto uso di diverse librerie, alcune disponibili pubblicamente, altre sviluppate all'interno del laboratorio di robotica e macchine intelligenti (RIMLab) del Dipartimento di Ingegneria dell'Informazione dell'Università degli Studi di Parma.

3.3.1 Driver dei sensori e degli attuatori

In primo luogo, va sicuramente citato il lavoro di Davide Valeriani [41]. Nella sua tesi, egli ha realizzato una libreria C++ per la programmazione del robot Comau Smart SiX, più semplice da utilizzare della libreria proprietaria PDL2 supportata dall'unità di controllo C4G. L'architettura di sistema si basa su un client C++ ed un server PDL2, come illustrato nel diagramma di figura 3.5.

Tramite semplici chiamate ai metodi della classe `RobotComau`, è possibile inviare direttive al manipolatore, e ricevere al tempo stesso informazioni di controllo. La traduzione delle direttive in comandi PDL2 viene effettuata dal server, in attesa su una determinata *socket*.

La classe `Gripper` integra invece il lavoro svolto da Marco Tarasconi nella sua tesi [39], che riguarda la comunicazione tramite protocollo seriale RS-232 con la pinza Schunk PG 70. Grazie a tale classe, è possibile controllare la presa di oggetti mediante valori di posizione, velocità, accelerazione, jerk e corrente.

Fondamentale poi è il driver per il laser Sick LMS 400, sviluppato da Michele Pattera [30] durante il corso di Robotica. Dopo aver messo in funzione il dispositivo, ed effettuato alcuni primi test con il programma SOPAS fornito dal produttore Sick, Pattera ha realizzato le primitive di invio e ricezione dei dati. In aggiunta, ha provveduto a compiere una caratterizzazione statistica del sensore, confrontando i dati ottenuti con quelli dichiarati nel manuale.

Antonio Ferrazzano, nel suo progetto [14] per il corso di Robotica, ha integrato le varie parti fin qui citate, riuscendo a realizzare un programma in grado di salvare su file le scansioni acquisite. Inoltre con un'apposita procedura ha calibrato la posa del sensore laser, determinando empiricamente l'angolo di sfasamento ϕ tra piano di scansione del laser ed il piano di osservazione.

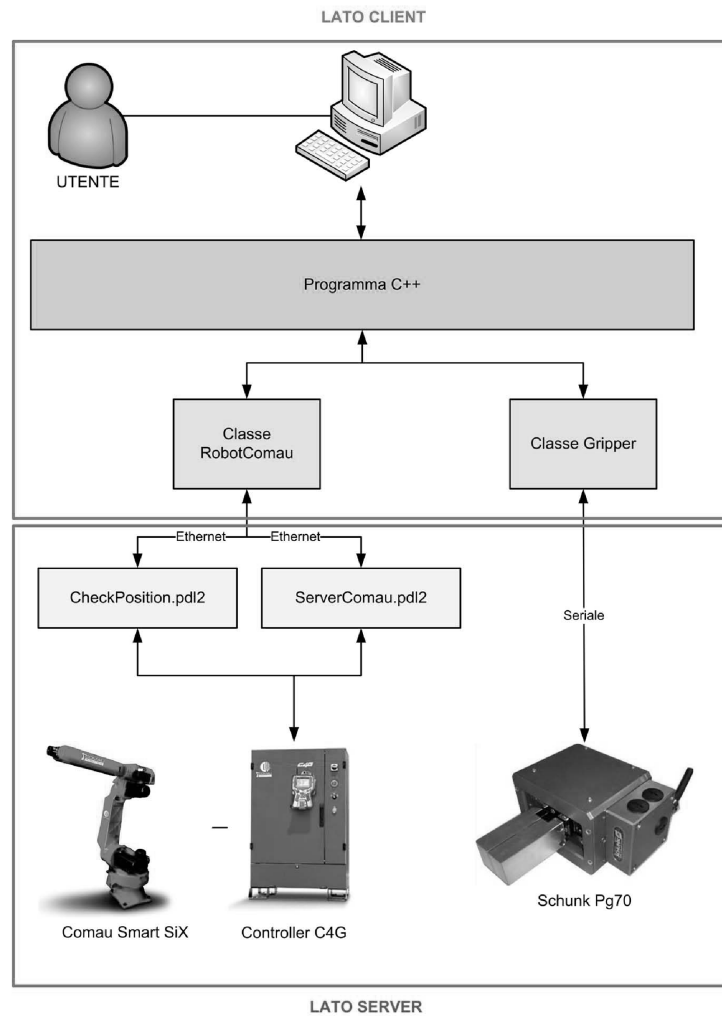


Figura 3.5: Architettura della libreria sviluppata da Davide Valeriani.

3.3.2 Librerie per l'elaborazione di dati

Eigen

Eigen è una libreria per l'algebra lineare ed il calcolo matriciale. È scritta in linguaggio C++ e fa largo uso di *template*. Attualmente supporta matrici di ogni dimensione, sia dense che sparse, con elementi reali o complessi. Utilizza tecniche di ottimizzazione per sfruttare le istruzioni SIMD (*single instruction, multiple data*) e per quanto possibile fa largo uso della memoria *cache*.

Nell'ambito di questa tesi, è stata adoperata per effettuare le trasformazioni da letture laser a coordinate riferite alla base del manipolatore. In particolare, dopo aver trasformato N misure di prossimità in punti (x_k, y_k) sul piano di scansione, è possibile effettuare una singola moltiplicazione tra una matrice di trasformazione affine T_l^b ed una matrice

$$P_l = \begin{bmatrix} x_0 & x_1 & & x_{N-2} & x_{N-1} \\ y_0 & y_1 & \cdots & y_{N-2} & y_{N-1} \\ 0 & 0 & & 0 & 0 \end{bmatrix}$$

per ottenere le coordinate dei medesimi punti rispetto alla terna di base.

Point Cloud Library (PCL)

Una nuvola di punti (*point cloud*) è una struttura dati per la rappresentazione di collezioni di punti. I punti possono essere acquisiti in vario modo: telecamere stereo, scanner tridimensionali, immagini di prossimità, etc.

La Point Cloud Library, in breve PCL, è una libreria per l'elaborazione di nuvole di punti [32]. Include un gran numero di algoritmi utili per diversi scopi, tra cui filtraggio, segmentazione, ricostruzione di superfici, *fitting* di modelli. È una libreria piuttosto recente, quindi in continuo sviluppo ed evoluzione.

La PCL è stata utilizzata per memorizzare ed elaborare i punti acquisiti. In particolare, per effettuare un filtraggio sull'area di lavoro, rimuovere il rumore, trovare i parametri del piano relativo al tavolo su cui giacciono gli

oggetti, estrarre i punti che stanno al di sopra di esso, raggrupparli in insiemi omogenei in base alla distanza.

MeshLab

MeshLab è un visualizzatore di modelli tridimensionali di oggetti, in grado anche di effettuare alcuni tipi di modifiche agli stessi. Nello specifico è pensato proprio per elaborare una gran mole di dati non strutturati provenienti da una qualche scansione, e fornisce strumenti per ispezionare, pulire, riparare ed esportare sottoinsiemi degli stessi.

MeshLab fornisce un ottimo supporto alle nuvole di punti, ovvero ai modelli 3D composti da soli vertici, oltre che a numerosissimi formati grafici. È stato impiegato in primo luogo per controllare visivamente la qualità delle scansioni acquisite, quindi per pulire e migliorare i punti stessi. Tra le operazioni più utilizzate vi sono la semplificazione e lo *smoothing* delle superfici.

OpenRAVE

OpenRAVE è un ambiente di sviluppo e test di algoritmi per la pianificazione del moto in applicazioni robotiche [12]. Fornisce strumenti di simulazione ed analisi della cinematica e delle informazioni geometriche relative alla pianificazione del movimento. Può essere integrato all'interno di sistemi o controllori robotici esistenti, mediante le API in linguaggio C++, Python, Octave o MATLAB.

Per il lavoro descritto in questa tesi, è stato creato un ambiente simulato che replica piuttosto fedelmente il *setup* del laboratorio, anche grazie all'esistenza di un database di modelli di robot e pinze dal quale attingere. OpenRAVE è stato dunque utilizzato per studiare possibili prese sugli oggetti ricostruiti, e per simulare una procedura di raccolta e spostamento dello stesso oggetto.

3.4 Acquisizione e raggruppamento dei dati del sensore laser

3.4.1 Procedure di acquisizione

La procedura di scansione che è stata scritta fa uso del robot manipolatore e del sensore laser, nonché di un calcolatore per l'elaborazione dei dati ed il relativo output. Tale procedura comanda il robot, muovendolo attraverso una serie di pose date in ingresso, mentre al tempo stesso il laser acquisisce letture metriche. Per ogni lettura, il programma memorizza posizione ed orientazione dell'organo terminale, in modo da poter in seguito trasformare le coordinate relativamente alla base del manipolatore.

Come ingresso, il software sviluppato accetta un file in formato testuale avente la seguente struttura.

File 1 Il formato del file di ingresso `.path`

```
x1 y1 z1 α1 β1 γ1
e1,2 k1,2 s1,2
x2 y2 z2 α2 β2 γ2
e2,3 k2,3 s2,3
x3 y3 z3 α3 β3 γ3
⋮
```

Ogni riga dispari corrisponde ad una posa del robot, ove x_i, y_i, z_i sono in millimetri ed $\alpha_i, \beta_i, \gamma_i$ sono in gradi (notazione ZYZ). Le righe pari specificano invece i parametri tra la posa precedente e la posa successiva:

- $e_{i,i+1}$ è un valore booleano che stabilisce se attivare o meno la ricezione di letture da parte del sensore laser;
- $k_{i,i+1}$ sono il numero di acquisizioni da effettuare mentre la traiettoria viene interpolata tra le due pose, ovvero il numero di passi intermedi;
- $s_{i,i+1} \in [0, 100]$ è la velocità alla quale il robot deve muoversi, sempre nel tratto tra le due pose i ed $i + 1$.

In uscita, il programma produce due file differenti: uno “grezzo”, contenente un elenco di letture assieme alla posa dell’organo terminale alla quale esse sono riferite, ed uno sottoposto alla trasformazione in coordinate relative rispetto alla base del robot.

Il primo, avente estensione `.carmen`, contiene alternativamente una riga con prefisso `FLASER` ed una con prefisso `ODOM`. La riga `FLASER` è seguita dal numero N di letture presenti, e dalle relative misure ρ_k in metri, con $k \in [0, N-1]$. La riga `ODOM` riporta invece la posizione x, y, z dell’organo terminale in metri, e l’orientazione α, β, γ dell’organo terminale in radianti (notazione ZYZ).

File 2 Il formato del file di uscita `.carmen`

`FLASER` N_0 $\rho_{0,0}$ $\rho_{0,1}$ \dots $\rho_{0,N-1}$

`ODOM` x_0 y_0 z_0 α_0 β_0 γ_0

`FLASER` N_1 $\rho_{1,0}$ $\rho_{1,1}$ \dots $\rho_{1,N-1}$

`ODOM` x_1 y_1 z_1 α_1 β_1 γ_1

\vdots

Il secondo è un file `.obj`, che consiste in una lista di punti rappresentati dalle loro tre coordinate. Essendo un formato di visualizzazione grafica universalmente accettato, esso viene utilizzato per mostrare i punti in tre dimensioni sotto forma di vertici (per questo motivo il prefisso `v`). Uno dei numerosi programmi in grado di visualizzare file `.obj` è il precedentemente citato MeshLab, grazie al quale è possibile fornire un rapido *feedback* grafico delle scansioni effettuate.

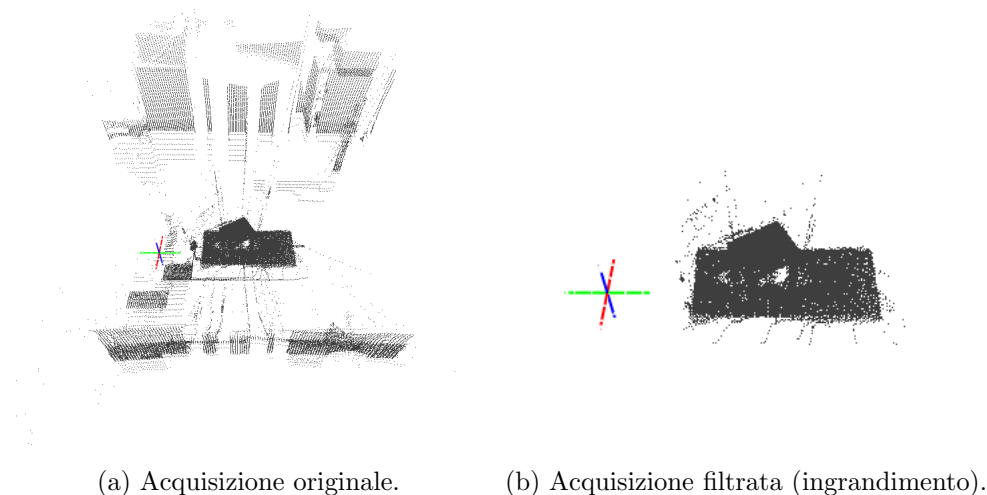
File 3 Il formato del file di uscita `.obj`

`v` x_0 y_0 z_0

`v` x_1 y_1 z_1

`v` x_2 y_2 z_2

\vdots



(a) Acquisizione originale. (b) Acquisizione filtrata (ingrandimento).

Figura 3.6: Filtraggio dei punti esterni al volume di lavoro.

3.4.2 Elaborazione dei punti

Dopo aver scandito l'area di interesse, ed ottenuto una nuvola di punti, è necessario elaborare ulteriormente i dati. Ci si prefigge di estrarre dall'insieme di punti soltanto quelli appartenenti agli oggetti posti sopra un tavolo all'interno dell'area di lavoro del manipolatore.

Innanzitutto, è possibile attuare un primo filtraggio sui valori delle coordinate x, y, z di ciascun punto (figura 3.6). In questo modo si ottiene un volume valido avente la forma di un parallelepipedo, i punti al di fuori del quale vengono scartati. Possiamo così facilmente escludere punti che non possono appartenere a nessun oggetto, in quanto non situati al di sopra del tavolo, oppure aventi una coordinata z troppo alta o troppo bassa.

In una scansione laser, gli errori di misurazione possono diventare punti non corrispondenti alla realtà. Tali punti, chiamati *outliers*, sono valori anomali chiaramente distanti dalle altre osservazioni disponibili. In alcuni casi, grazie a tecniche di analisi statistica sul vicinato di ciascun punto, è possibile individuare questi punti aberranti e rimuoverli (figura 3.7).

Il filtraggio di rimozione del rumore presente nella PCL [33] è basato sul

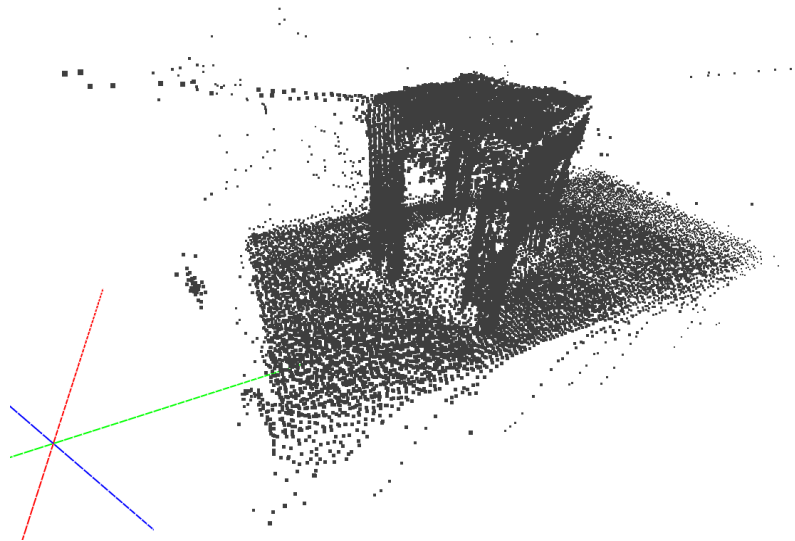
computo della distanza media di ogni punto dai suoi k vicini. Considerando l'intera nuvola, si assume che la distribuzione della misura calcolata abbia andamento gaussiano, con valor medio η e deviazione standard σ ben determinati. Tutti i punti aventi distanza media dai vicini non compresa in $\eta \pm m\sigma$, con m selezionabile a piacere, vengono marcati come *outliers* ed eliminati.

Le restanti irregolarità, ovvero gli errori di misurazione minori, possono essere invece filtrate con altri metodi. In particolare, è possibile effettuare un sottocampionamento o un sovracampionamento della superficie, creando nuovi punti interpolando polinomialmente quelli esistenti con i relativi vicini. In questo modo le normali dei punti risultano meno discordanti tra loro, poiché sistemate utilizzando l'algoritmo *moving least squares*.

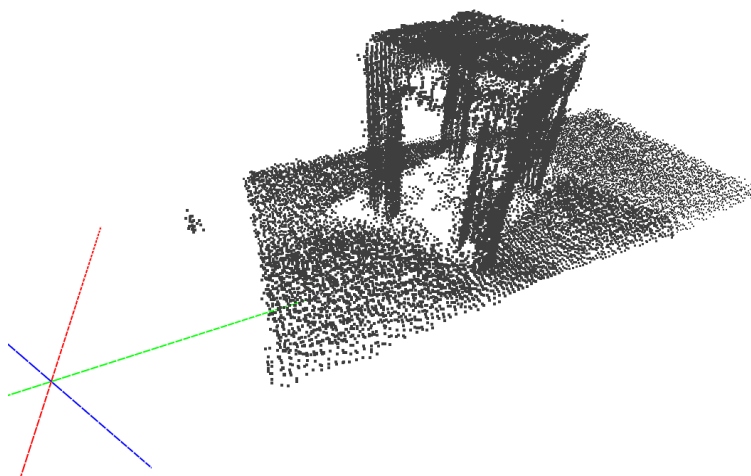
In aggiunta, è lecito operare un qualche tipo di semplificazione per ridurre il carico computazionale, ovviamente a patto che questo non comporti la perdita di troppe informazioni. La libreria PCL fornisce un modo rapido e veloce per ridurre il numero di punti in una nuvola: il sottocampionamento a griglia. Stabilite tre grandezze $\Delta x, \Delta y, \Delta z$, si suddivide l'intero spazio in partizioni di tali dimensioni. Per ogni partizione, esistono quindi due approcci: sostituire tutti i punti esistenti con il centro della partizione, oppure con un singolo punto avente le coordinate del centroide dei punti. Quest'ultima soluzione, nonostante sia più computazionalmente onerosa, porta a risultati molto più accurati.

Ottenuta una nuvola di punti "pulita", bisogna procedere all'estrazione degli oggetti sopra il tavolo. Ovviamente, prima è necessario individuare dove sia posizionato il suddetto tavolo. Ancora una volta viene in aiuto la PCL, che offre una classe per l'estrazione di svariati modelli parametrici, tra cui il piano in notazione normale hessiana. Tramite l'algoritmo *random sample consensus* (RANSAC, [15]) si adatta il modello ai punti della nuvola, poi si distinguono i punti appartenenti al piano (figura 3.8a) dai rimanenti.

I punti inclusi nel piano vengono dunque proiettati sul modello che lo rappresenta, ottenendo un involuppo convesso, ovvero l'intersezione di tutti

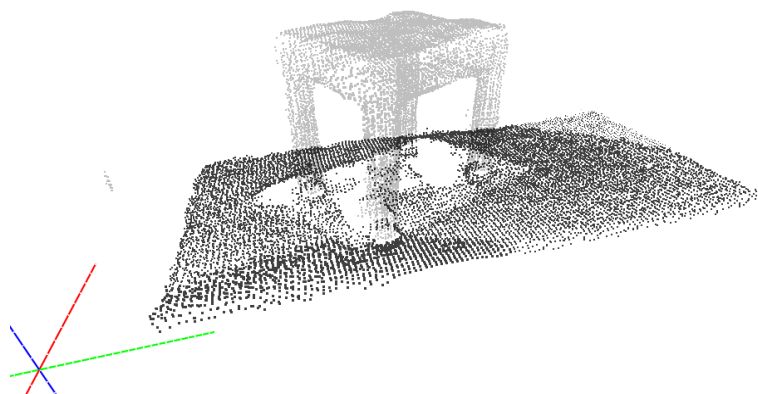


(a) Nuvola di punti prima della rimozione.

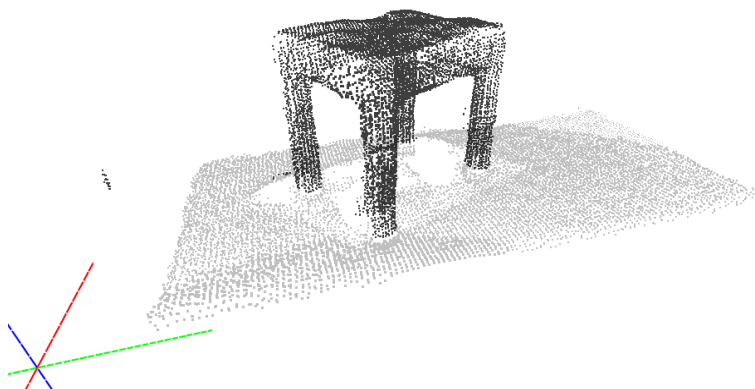


(b) Nuvola di punti dopo la rimozione.

Figura 3.7: Rimozione dei punti aberranti (*outliers*).



(a) Punti appartenenti al piano.



(b) Punti appartenenti agli oggetti.

Figura 3.8: Estrazione dei punti appartenenti al piano ed agli oggetti.

gli insiemi convessi che contengono le proiezioni dei punti sul modello. Finalmente, si estraggono i punti della nuvola che ricadono entro una certa soglia dall'involuppo convesso (figura 3.8b).

In ultimo luogo, tali punti devono essere raggruppati tra loro poiché potrebbero essere relativi a più di un oggetto. A questo scopo esiste un insieme di tecniche di analisi dei gruppi, ovvero per il *clustering*. La somiglianza, in termini di distanza in uno spazio multidimensionale, tra gli elementi di un insieme porta al raggruppamento di quelli tra loro omogenei. Questa tecnica dipende molto dalla scelta della metrica, che nel caso della PCL è la

distanza euclidea, siccome si tratta dello spazio tridimensionale. L'algoritmo 3.1, tratto dalla dissertazione di Radu Bogdan Rusu [31], illustra le fasi che portano al raggruppamento dei punti.

Algoritmo 3.1 Raggruppamento dei punti in base alla distanza euclidea.

Input: nuvola di punti P

Output: lista di cluster C

crea un KD-tree per la nuvola di punti in ingresso P

inizializza una lista di cluster C vuota

inizializza una coda di punti da controllare Q vuota

for all $p_i \in P$ **do**

aggiungi p_i a Q

for all $p_i \in Q$ **do**

costruisci l'insieme $N_i = \{p \in P : \|p - p_i\| < r\}$

for all $n_i \in N_i$ **do**

if il punto n_i non è stato processato **then**

inserisci il punto n_i nella coda Q

end if

end for

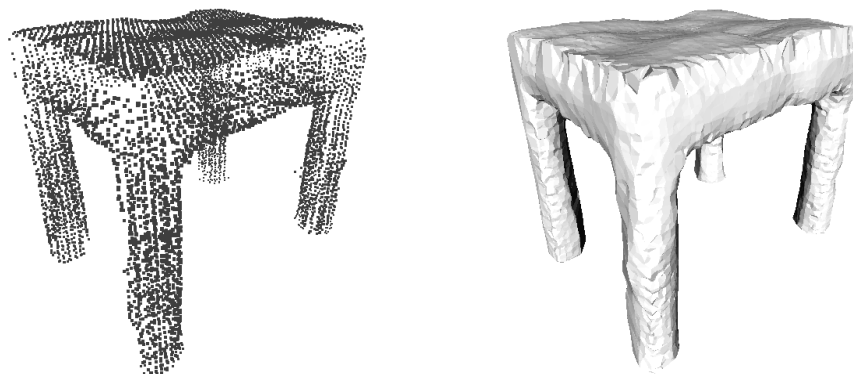
end for

aggiungi a C il cluster formato dagli elementi di Q

svuota la coda Q

end for

Dopo questa fase, ciascuno dei gruppi di punti individuato (figura 3.9a) è pronto per essere sottoposto ad una procedura di ricostruzione 3D. Tra le più famose e robuste troviamo sicuramente il *power crust* [3, 4], in grado di generare un asse mediale approssimato dell'oggetto dal quale poi si deriva la superficie (figura 3.9b). Questo metodo ha diversi pregi, tra i quali il fatto di funzionare bene anche nel caso in cui l'oggetto non sia campionato in maniera uniforme. È inoltre in grado di generare superfici perfettamente *watertight*, ovvero senza buchi o facce mancanti.



(a) *Cluster* dei punti di un oggetto. (b) Superficie dell'oggetto ricostruita mediante l'algoritmo *power crust*.

Figura 3.9: Gruppo di punti appartenenti ad un singolo oggetto e ricostruzione della superficie del medesimo.

Classe PCL	Parametri significativi
PassThrough	Asse a sul quale effettuare il filtraggio; intervallo (b_0, b_1) da considerare valido.
Statistical- OutlierRemoval	Numero k di vicini da analizzare; moltiplicatore m per $\eta \pm m\sigma$.
MovingLeastSquares	Raggio di ricerca r .
VoxelGrid	Dimensioni $\Delta x, \Delta y, \Delta z$ del <i>voxel</i> .
SACSegmentation	Tipo di modello (SACMODEL PLANE); metodo di ricerca (SAC_RANSAC); soglia d per la distanza dal modello.
EuclideanCluster- Extraction	Tolleranza t per punti nello stesso <i>cluster</i> ; dimensioni c_{min}, c_{max} per un <i>cluster</i> .

Tabella 3.1: Riassunto delle operazioni per l'elaborazione dei punti disponibili nella *Point Cloud Library*.

In questa tesi si è tentato di sviluppare alcuni algoritmi innovativi per generare la superficie di un oggetto. Essi sfruttano tutti la presenza di un punto di osservazione, dovuto al tipo di sensore utilizzato ed alla particolare configurazione adottata, e non dovrebbero presentare problemi anche nel caso in cui i punti non siano uniformemente distribuiti. Nel capitolo 4 le varie procedure sono state descritte con maggiore dettaglio.

3.5 Identificazione e manipolazione delle parti degli oggetti

Nel campo della robotica, l'obiettivo di molte applicazioni è la comprensione del significato della scena osservata. Il riconoscimento degli oggetti e la loro corretta interpretazione sono requisiti fondamentali per compiti di più alto livello, come ad esempio la manipolazione. A loro volta, il riconoscimento e l'interpretazione sono dipendenti dalla suddivisione dei dati in porzioni significative. Per questo motivo, una volta ottenuta la superficie di un oggetto, si cerca di risalire alle parti che lo compongono.

Esistono algoritmi e strumenti per caratterizzare una superficie rispetto ad una funzione reale. Lavorando sugli insiemi di livello, questi sono in grado di calcolare massimi, minimi e selle. Una delle funzioni di misurazione migliori allo scopo di segmentare l'oggetto è la funzione integrale geodesica [19], che possiede l'importante pregio di essere invariante alla rotazione.

A partire da tali caratteristiche, si è poi in grado di creare il corrispondente grafo di Reeb. Un grafo di questo tipo descrive infatti la connettività degli insiemi di livello di un oggetto. I grafi di Reeb sono molto utilizzati per il riconoscimento di forme basato sulla topologia, che avviene mediante interrogazione di un database di grafi ed estrazione della classe corrispondente al grafo più simile.

Rita Beltrami [5], durante il corso di Robotica, ha effettuato numerose procedure di acquisizione. Sono state fatte molte prove impiegando oggetti reali e costruiti *ad-hoc*, usando svariati materiali e rivestimenti. È stata

studiata la qualità delle scansioni al variare di molti parametri, ed i campioni migliori sono stati sottoposti alla ricostruzione della superficie ed alla segmentazione (figura 3.10).

Dopo aver ottenuto un modello dell'oggetto, suddiviso in parti, si può pensare di studiare come questo possa essere preso dal manipolatore. Lo studio viene compiuto esaminando ogni parte indipendentemente dalle altre, seguendo l'approccio tipico degli esseri umani. Per ciascuna parte, si analizzano le prese stabili e si memorizzano le stesse all'interno di una lista.

La pianificazione del moto necessario per prendere l'oggetto è affidata al motore dell'ambiente OpenRAVE, che utilizza l'algoritmo *Rapidly-exploring Random Tree* (RRT). Dopo questa fase si dispone di una lista di coordinate nello spazio dei giunti del manipolatore, corrispondenti ad una serie di pose da attraversare. Tale lista è suddivisa in cinque parti, ognuna relativa ad una particolare fase della manipolazione:

- partendo dalla posizione iniziale, ci si avvicina all'oggetto muovendosi a velocità elevata;
- si inizia la fase di *pre-grasp*, ovvero quella antecedente alla presa, posizionando con cura la pinza in corrispondenza della parte dell'oggetto da afferrare, e concludendo con la chiusura della stessa;
- muovendosi a velocità moderata si solleva dunque l'oggetto;
- poi, aumentando la velocità, lo si sposta poco al di sopra del punto dove dev'essere rilasciato;
- infine, si abbassa lentamente l'oggetto e si apre la pinza.

Sono state effettuate prove sperimentali, sia nell'ambiente simulato OpenRAVE, sia nell'ambiente reale utilizzando il robot Comau Smart SiX insieme alla pinza Schunk PG 70. Per l'esattezza, il locale del laboratorio e la configurazione hardware sono state replicate in simulazione. In questo modo è stato possibile eseguire realmente le traiettorie pianificate da OpenRAVE, nonché collaudare alcune delle prese.

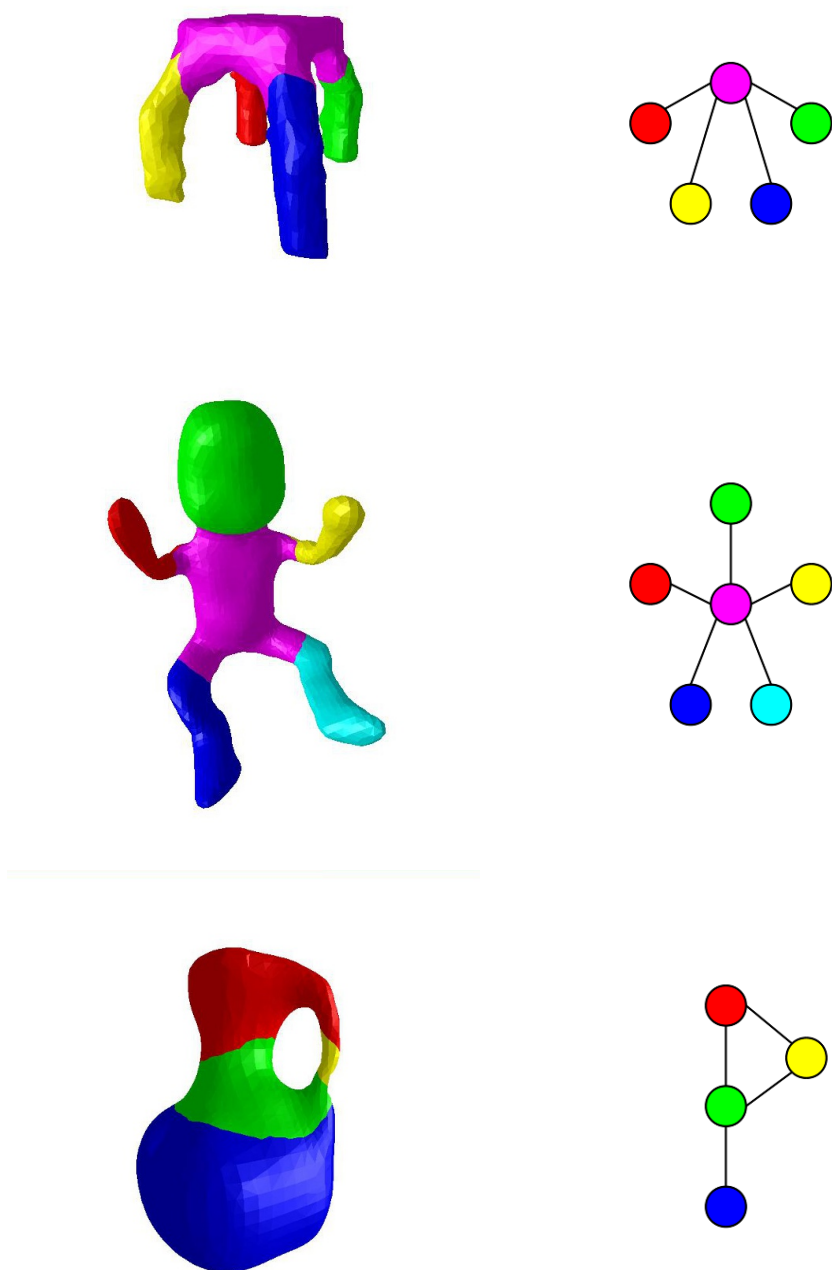


Figura 3.10: Alcuni oggetti segmentati ed i relativi grafi di Reeb.

Capitolo 4

Elaborazione di misure di prossimità

In questo capitolo si elencano le varie operazioni utilizzate al fine di realizzare progressivamente una triangolazione su una superficie 2D. Viene inoltre illustrato come sfruttare le relazioni di vicinanza tra scansioni e come costruire una *mesh* basata sui vincoli di prossimità.

A partire da una nuvola di punti, si effettuano tali elaborazioni con l'obiettivo di ottenere una rappresentazione della superficie del suddetto oggetto. La superficie può essere descritta mediante una varietà bidimensionale nello spazio euclideo tridimensionale, composta da più triangoli (triangolazione) o in generale da più poligoni.

Il sistema di acquisizione utilizzato garantisce il rispetto di alcune ipotesi, fondamentali per il funzionamento degli algoritmi esposti nel seguito. In primo luogo, ogni punto è acquisito da un sensore avente una posizione nello spazio. Nel nostro caso questo punto è rappresentato del centro del sensore laser.

I punti appartenenti alla medesima scansione giacciono inoltre sullo stesso piano, identificato dal punto di osservazione e da due o più punti della scansione. All'interno della scansione, inoltre, i punti sono ordinati secondo l'angolo formato dal raggio con il sistema di riferimento del sensore, e spesso

punti adiacenti sono anche vicini tra loro. L'affermazione si rivela inesatta soltanto in presenza di discontinuità, come accade ad esempio vicino ai contorni di un oggetto. L'indice dei punti è comunque un criterio da utilizzare come prima traccia di prossimità.

In ultimo luogo, si presuppone che i punti appartenenti a due scansioni acquisite in istanti temporali consecutivi siano con buona probabilità vicini tra loro. Questa coerenza temporale in alcuni casi non è rispettata, ad esempio se la posa del sensore varia quando lo stesso è spento, oppure nel caso vengano concatenate più acquisizioni differenti. Una condizione del genere tuttavia può essere rilevata, attivando meccanismi più complessi per l'identificazione del vicinato.

Nel seguito sono illustrati alcuni metodi per la ricostruzione di superfici a partire da dati sensoriali.

4.1 Costruzione incrementale di triangolazioni su superfici

Un obiettivo del lavoro è la costruzione incrementale della superficie di un oggetto a partire dai punti acquisiti con un sensore di prossimità per cui valgono le ipotesi illustrate in precedenza. In particolare, le ipotesi sull'esistenza di un punto di osservazione e l'organizzazione delle scansioni suggeriscono la possibilità di elaborare l'insieme di punti dello spazio sfruttando l'appartenenza ad una varietà bidimensionale. Per ogni nuovo punto misurato dal sensore, la superficie iniziale viene modificata in maniera tale da includerlo all'interno di essa.

4.1.1 Triangolazione di Delaunay

Utilizzando le informazioni provenienti dal sensore, ovvero l'orientamento del punto sulla superficie che racchiude l'oggetto, l'obiettivo è ricostruire con buona approssimazione la sua superficie. A questo scopo ci si avvale delle

triangolazioni, strumenti tradizionali della grafica computazionale per la rappresentazione numerica delle superfici. Numerosi algoritmi di triangolazione presenti in letteratura permettono di effettuare la tassellatura di superfici. Per un piano, ad esempio, una triangolazione corrisponde ad una suddivisione dello stesso in triangoli. Una definizione più formale di triangolazione e complesso simpliciale può essere trovata in [28].

Prima di trattare l'algoritmo impiegato in questa tesi si rende necessario introdurre alcuni concetti geometrici, quali ad esempio il diagramma di Voronoi, la triangolazione di Delaunay, le varietà bidimensionali.

Dato un insieme di punti, un diagramma di Voronoi bidimensionale è una decomposizione del piano formata associando ad ogni punto una determinata regione di influenza (figura 4.1a). Chiamato $P = \{p_1, p_2, \dots, p_n\} \subseteq \mathbb{R}^2$ questo insieme, la regione di Voronoi per $p_i \in P$ è l'insieme di punti $x \in \mathbb{R}^2$ che sono più vicini al sito p_i che ad ogni altro sito $p_j \in P$. In altre parole,

$$V_{p_i} = \{x \in \mathbb{R}^2 \mid \|x - p_i\| \leq \|x - p_j\|, \forall p_j \in P, j \neq i\}.$$

Poiché ogni punto deve possedere almeno un vicino in P , le regioni di Voronoi coprono l'intero piano. I punti che appartengono a più di una regione, ovvero i punti equidistanti da due o più siti, formano il diagramma di Voronoi dell'insieme P . Questo tipo di diagrammi risulta utile per determinare, per ogni punto, quale sia il sito più vicino.

La triangolazione di Delaunay, come mostrato in figura 4.1b, è il duale di un diagramma di Voronoi. Essa si ottiene connettendo i punti $p_i \in P$ tra loro se e soltanto se le regioni di Voronoi hanno un segmento in comune. Il risultato generale è il conseguimento di una suddivisione del piano in regioni triangolari (figura 4.2a). All'interno di tale triangolazione, vale la seguente proprietà: la circonferenza circoscritta ad ogni triangolo non contiene alcun vertice di altri triangoli (figura 4.2b).

La descrizione intuitiva del concetto di "varietà" è quella di seguito illustrata. Per disegnare una mappa della superficie terrestre, è necessario proiettare la superficie sferica convessa su di un piano. Non essendo possibile effettuare questa operazione per l'intera sfera, bisogna suddividerla in

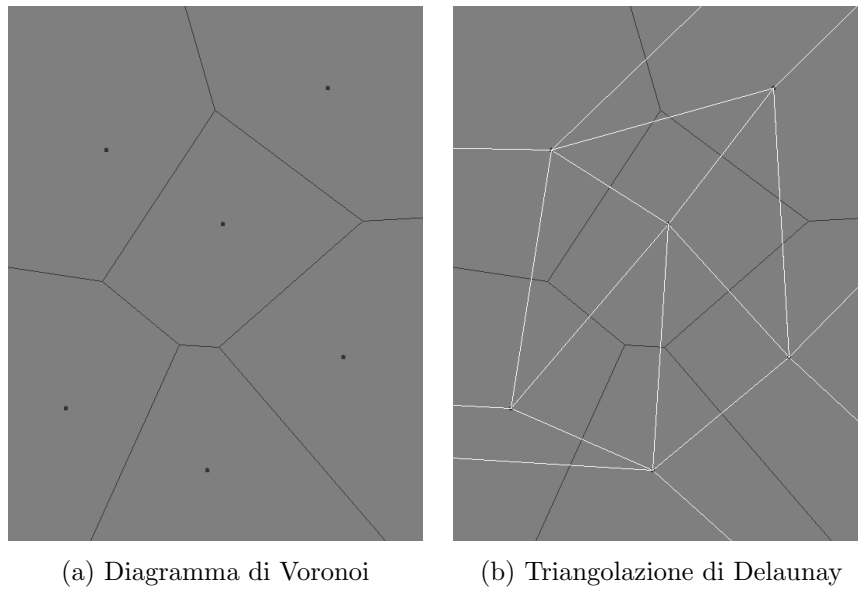


Figura 4.1: Diagramma di Voronoi con sovrapposizione della relativa triangolazione di Delaunay.

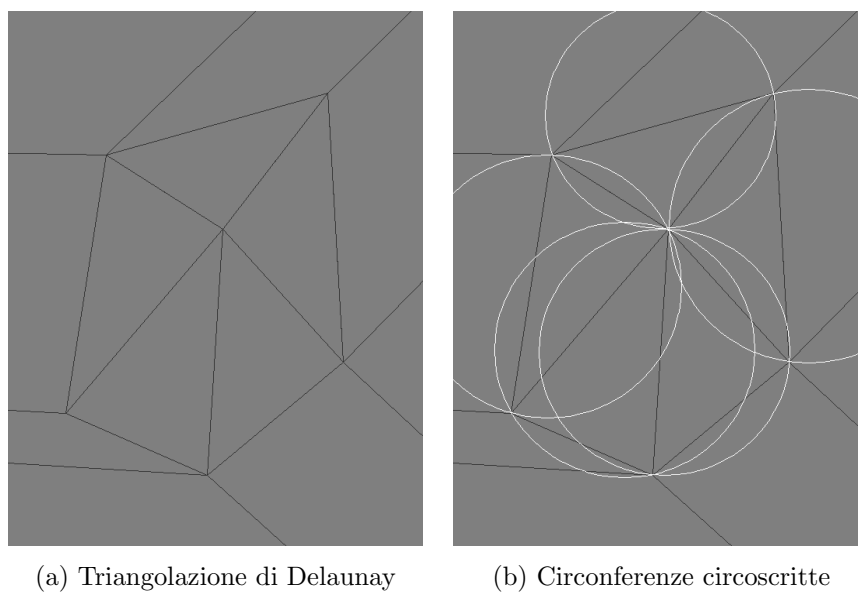


Figura 4.2: Triangolazione di Delaunay con visualizzazione delle circonferenze circoscritte ad ogni triangolo.

porzioni minori e proiettare ognuna di esse in una mappa separata. Si può ricostruire la sfera originale con l'operazione inversa: incollare tra loro le diverse proiezioni secondo criteri di adiacenza. Le parti sovrapposte sono quelle presenti in più di una mappa.

L'esempio più semplice di varietà sono le superfici bidimensionali all'interno dello spazio euclideo tridimensionale. In questa tesi sono state impiegate varietà bidimensionali per costruire una triangolazione della superficie dell'oggetto. In particolare sono state utilizzate le suddivisioni, ovvero il partizionamento di varietà in insiemi di vertici, archi e facce.

Il lavoro svolto si basa sui risultati ottenuti da Guibas e Stolfi [18] per quanto riguarda la sfera ed il piano esteso $\hat{\mathbb{R}}^2 = \mathbb{R}^2 \cup \{\infty\}$. Ci si pone l'obiettivo di adoperare la soluzione proposta dagli autori per triangolare una varietà all'interno dello spazio tridimensionale, adattando in maniera opportuna le primitive geometriche.

4.1.2 Costruzione della triangolazione di Delaunay con *quad-edge*

Ogni suddivisione, come la triangolazione di Delaunay, può essere rappresentata da strutture dati in grado di mettere in relazione tra loro gli elementi della suddivisione, ossia i vertici, i lati e le facce che la costituiscono. Le relazioni tra tali elementi, per esempio l'incidenza dei lati o l'adiacenza delle facce) possono essere convenientemente rappresentata da strutture basate su grafo.

Un esempio di struttura dati nota in letteratura ed ideata per trattare le triangolazioni è il *quad-edge* [18]. In questo modo si rappresenta ovviamente anche la suddivisione duale, ovvero il diagramma di Voronoi. Il *quad-edge* è costituito da tre tipi di elementi: i vertici, i lati e le facce (figura 4.3). Ogni lato possiede due puntatori agli altrettanti vertici connessi dal lato, e due puntatori per le relative facce che condividono il lato. Ciascun lato comprende in se stesso, quindi, due archi orientati che connettono vertici e

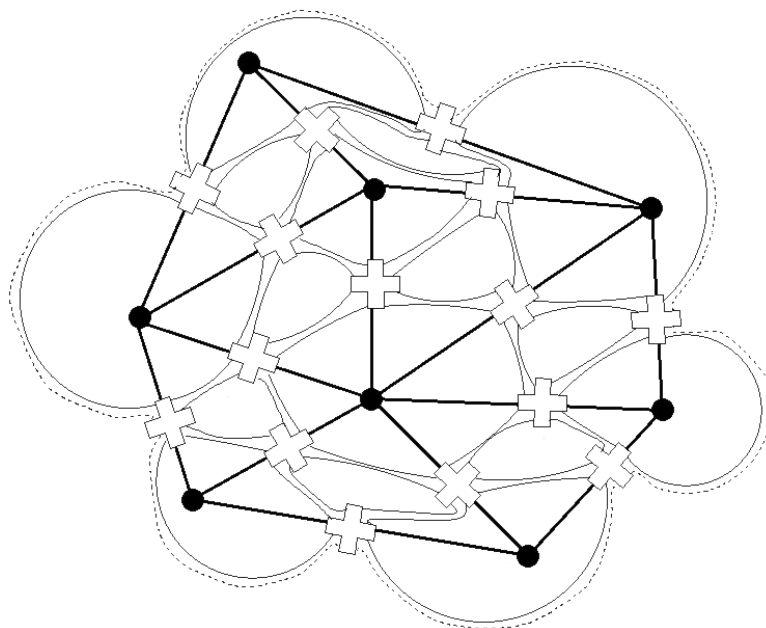


Figura 4.3: Elementi della struttura dati *quad-edge*.

facce. Inoltre è possibile passare dal grafo vertici-lati al suo duale facce-lati senza ulteriore elaborazione.

Le operazioni topologiche di base su una tale struttura sono la creazione di un nuovo arco e la cosiddetta operazione di “giunzione” tra due archi (dal termine inglese *splice*). L’effetto dell’esecuzione quest’ultima sulla struttura dati è il seguente:

- se gli archi condividono lo stesso vertice di origine, ma non hanno la medesima faccia sulla sinistra, allora i vertici verranno separati e le facce saranno unite;
- se gli archi non condividono il medesimo vertice di origine, ma sono connessi sulla sinistra alla stessa faccia, allora le facce verranno separate ed i vertici saranno uniti.

Nel caso in esame, in cui gli archi fanno parte di un diagramma di Voronoi o di una triangolazione di Delaunay, alcuni semplici interventi quali la

connessione o la rimozione di un nuovo arco possono essere ricondotti ad una combinazione delle operazioni topologiche evidenziate precedentemente.

Vi è poi una particolare procedura, utile nel caso in cui qualche triangolo non soddisfi la proprietà di Delaunay: lo scambio di archi (dal termine inglese *swap*). Dato un arco avente una faccia triangolare a destra ed un'altra a sinistra, effettuare lo scambio dell'arco significa eliminare il medesimo e sostituirlo connettendo gli altri due vertici del quadrilatero che si è venuto a formare.

Maggiori dettagli sulle strutture dati e sulle procedure elementari fin qui esposte sono presentati in [24]. Servendosi di questi meccanismi è stato sviluppato un algoritmo per la costruzione incrementale della triangolazione di Delaunay relativa alla superficie di un oggetto.

Inizializzazione della triangolazione

Poiché ogni punto viene inserito singolarmente all'interno della struttura attualmente esistente, è necessario inizializzare la triangolazione. In altre parole, bisogna definire una superficie entro la quale saranno inclusi tutti i punti. Un'inizializzazione conveniente consiste in una singola faccia triangolare, ovvero composta da tre archi e tre vertici, di dimensioni sufficientemente grandi da contenere le proiezioni dei punti della scansione.

L'area di lavoro del manipolatore adoperato è sicuramente limitata. Grazie a questa condizione, si possono facilmente trovare dei limiti per le coordinate x , y , z . Dati i limiti dell'area di lavoro $x_{min} = -2$ m, $x_{max} = 2$ m, $y_{min} = 0$ m ed $y_{max} = 2$ m, sono stati utilizzati i seguenti valori per i primi tre vertici: $A = (5, -1, 0)$ m, $B = (0, 10, 0)$ m, $C = (-5, -1, 0)$ m. La scelta di un valore nullo per la coordinata z è motivata dal fatto che il punto di osservazione nel *setup* utilizzato è sicuramente collocato al di sopra della quota zero.

I tre vertici A, B, C vengono connessi con altrettanti archi (figura 4.4). È importante sottolineare ancora una volta che la faccia ABC deve essere in grado di contenere tutti i punti inseriti nel seguito, pena il malfunzionamento

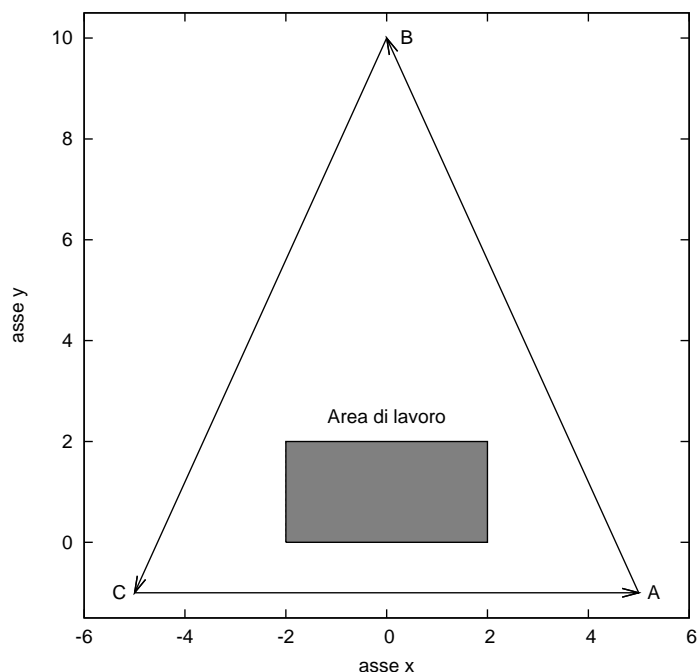


Figura 4.4: La triangolazione iniziale e l'area di lavoro del manipolatore.

dell'algoritmo. Tale faccia deve essere memorizzata al fine di essere adoperata durante le fasi successive della ricerca.

Ricerca nella suddivisione

Dopo aver effettuato una scansione, l'algoritmo prevede l'inserimento di ogni punto separatamente dagli altri. Il punto di osservazione, coincidente con il centro del sensore laser, è il medesimo per ciascuno dei punti appartenenti alla scansione. Prima di inserire un singolo punto, si procede alla ricerca della faccia che contiene la proiezione del punto se osservato dal centro del sensore. Siccome il punto di vista è conosciuto, è possibile adottare alcune semplificazioni illustrate nel seguito.

Per poter effettuare una ricerca sulla superficie orientata rappresentata dalla triangolazione, occorrono delle primitive geometriche che consentano di determinare la posizione relativa degli elementi della suddivisione, per

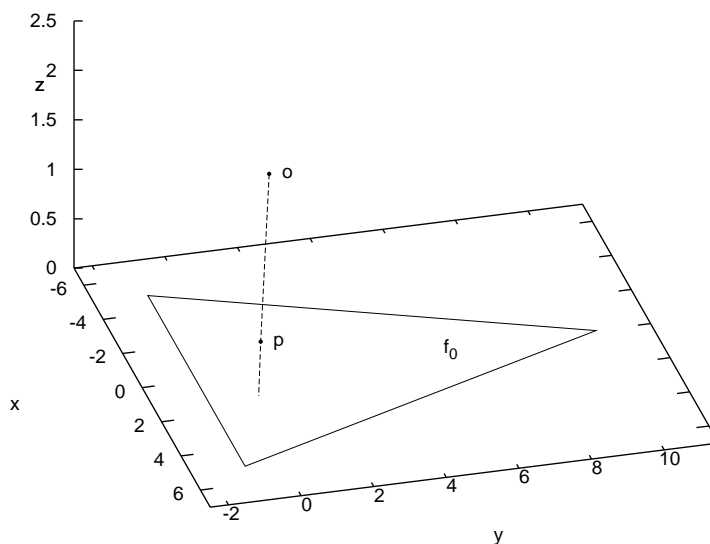


Figura 4.5: Ricerca della faccia contenente la proiezione del punto.

esempio se un punto giace “a destra” oppure “a sinistra” del lato di una faccia. Nel seguito verranno illustrate le primitive applicate al problema in esame.

La procedura di ricerca è illustrata nell’algoritmo 4.1. Dato un nuovo punto p da inserire, e calcolata la semiretta uscente dall’osservatore o passante per p , si vuole trovare una faccia f che intersechi tale semiretta (figura 4.5). Perché la ricerca abbia inizio, è necessario mantenere un puntatore alla faccia f_0 che sarà la prima ad essere esaminata dalla procedura. Un’opportuna scelta di f_0 può accelerare significativamente la ricerca.

In primo luogo si controlla se la faccia corrente risulta essere visibile. Il piano passante per i tre vertici della faccia suddivide lo spazio in due parti. A seconda dell’orientamento degli archi, il punto di osservazione può essere dalla parte visibile (superiore) oppure dalla parte non visibile (inferiore) della superficie.

Si procede con l’esaminare gli archi ed i vertici della faccia corrente. L’o-

rigine e la destinazione di ognuno dei tre archi formano con p ed o altrettanti tetraedri. Grazie al segno del volume di questi tetraedri si è in grado di determinare se il punto p si trova a destra (negativo) oppure a sinistra (positivo) dell'arco e_i quando visto dall'osservatore.

Inoltre, prendendo gli archi in senso antiorario, ciascun arco avrà alla sua sinistra la medesima faccia, ed alla sua destra una faccia differente. Anche per tali facce f_i si effettua un controllo di visibilità. Questo serve a determinare in che direzione spostarsi in funzione della visibilità della faccia di partenza e della faccia di destinazione.

Se la faccia di partenza o la faccia di destinazione sono visibili ed il volume del tetraedro indica che il punto p si trova alla destra dell'arco, ci si muove lungo quella direzione; lo stesso movimento verso destra accade nel caso in cui né la faccia di partenza né la faccia di destinazione sono visibili, ma il volume del tetraedro indica che p è a sinistra dell'arco. Nel caso in cui nessuna di queste condizioni di spostamento sia soddisfatta, significa che il punto è all'interno della faccia corrente, e la ricerca ha termine.

Stabilito che una faccia è per definizione visibile se i tre archi che la delimitano formano un circuito antiorario quando visti dal punto o , esaminiamo con maggiore dettaglio la condizione di visibilità (algoritmo 4.2). Questa condizione equivale ancora una volta a studiare il segno del volume del tetraedro formato dai vertici della faccia e dall'osservatore.

Il volume di un generico tetraedro avente vertici a, b, c, o può essere calcolato con la seguente formula, come suggerito da [29] in forma generale per un semplice di dimensione arbitraria:

$$V = \frac{1}{6} \begin{vmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ o_x & o_y & o_z & 1 \end{vmatrix}$$

Il predicato geometrico `VolumeTetraedro` è dunque equivalente al computo del determinante di una matrice, ovvero ad effettuare semplici somme di

Algoritmo 4.1 Ricerca della faccia più vicina alla proiezione del punto

Input: nuovo punto p , osservatore o , faccia iniziale f_0 **Output:** faccia contenente la proiezione del punto

```
loop
   $v_0 \leftarrow \text{Visibile}(f_0, o)$ 
  for  $i = 1 \rightarrow 3$  do
     $e_i \leftarrow \text{ArcoFaccia}(f_0, i)$ 
     $s_i \leftarrow \text{OrigineArco}(e_i)$ 
     $d_i \leftarrow \text{DestinazioneArco}(e_i)$ 
     $t_i \leftarrow \text{VolumeTetraedro}(s_i, d_i, p, o)$ 
     $f_i \leftarrow \text{FacciaDestra}(e_i)$ 
     $v_i \leftarrow \text{Visibile}(f_i, o)$ 
  end for
  if  $(v_0 \vee v_1) \text{ XOR } (t_1 < 0)$  then
     $f_0 \leftarrow f_1$ 
  else if  $(v_0 \vee v_2) \text{ XOR } (t_2 < 0)$  then
     $f_0 \leftarrow f_2$ 
  else if  $(v_0 \vee v_3) \text{ XOR } (t_3 < 0)$  then
     $f_0 \leftarrow f_3$ 
  else
    return  $f_0$ 
  end if
end loop
```

Algoritmo 4.2 La condizione booleana *Visibile*

Input: faccia da esaminare f , punto di osservazione o

Output: *vero* se la parte superiore della faccia è visibile, *falso* altrimenti

for $i = 1 \rightarrow 3$ **do**

$e_i \leftarrow \text{ArcoFaccia}(f, i)$

$d_i \leftarrow \text{DestinazioneArco}(e_i)$

end for

return $\text{VolumeTetraedro}(d_1, d_2, d_3, o) < 0$

prodotti. Per semplicità, si definiscono i valori di appoggio $\alpha = a - o$, $\beta = b - o$, $\gamma = c - o$ e poi si procede a calcolare il volume

$$V = \frac{1}{6} (-\alpha_z \beta_y \gamma_x + \alpha_y \beta_z \gamma_x + \alpha_z \beta_x \gamma_y - \alpha_x \beta_z \gamma_y - \alpha_y \beta_x \gamma_z + \alpha_x \beta_y \gamma_z).$$

Tale volume può avere segno sia positivo che negativo, in base all'ordinamento dei vertici a, b, c osservati dal restante vertice o : positivo se sono in senso orario, negativo se sono in senso antiorario.

Avendo imposto come condizione iniziale la creazione forzata della prima faccia, sappiamo con certezza che la proiezione del primo punto inserito ricadrà entro di essa. Per il punto successivo, il risultato della ricerca sarà una delle tre facce formatesi durante l'inserimento del primo punto. Questo ragionamento si può estendere a tutti i punti che verranno inseriti in seguito, sempre a patto che le ipotesi sull'area di lavoro siano rispettate.

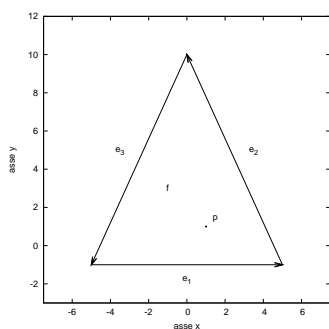
Dopodiché si effettua un controllo per evitare che il nuovo punto inserito coincida con uno già esistente oppure ricada a poca distanza da un arco attualmente presente (algoritmo 4.3). A questo scopo, per ciascun arco della faccia identificata mediante la ricerca, si esaminano: la distanza tra il punto e l'origine, la distanza tra il punto e la destinazione, la lunghezza dell'arco, la distanza del punto dall'arco.

Qualora il punto risultasse troppo vicino ad un vertice, oppure troppo vicino all'arco tra due vertici della faccia, esso viene scartato in quanto considerato non importante al fine di aggiungere informazioni sulla superficie.

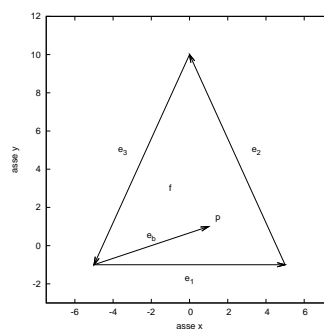
Algoritmo 4.3 Controllo dei casi degeneri

Input: nuovo punto p , faccia f contenente la proiezione di p **Output:** *vero* se il nuovo punto rappresenta un caso degenero

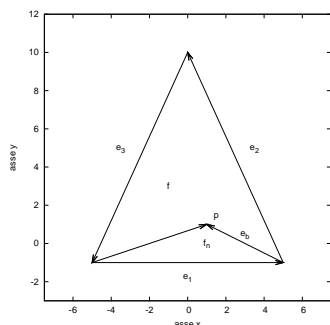
```
for  $i = 1 \rightarrow 3$  do
   $e_i \leftarrow \text{ArcoFaccia}(f_0, i)$ 
   $s_i \leftarrow \text{OrigineArco}(e_i)$ 
   $d_i \leftarrow \text{DestinazioneArco}(e_i)$ 
   $ps_i \leftarrow \text{DistanzaPuntoPunto}(p, s_i)$ 
   $pd_i \leftarrow \text{DistanzaPuntoPunto}(p, d_i)$ 
   $sd_i \leftarrow \text{DistanzaPuntoPunto}(s_i, d_i)$ 
  if  $ps_i < \varepsilon \vee pd_i < \varepsilon$  then
     $r_i \leftarrow \textit{vero}$ 
  else if  $ps_i > sd_i \vee pd_i > sd_i$  then
     $r_i \leftarrow \textit{falso}$ 
  else
     $r_i \leftarrow \text{DistanzaPuntoSegmento}(p, s_i, d_i) < \varepsilon$ 
  end if
end for
return  $r_1 \vee r_2 \vee r_3$ 
```



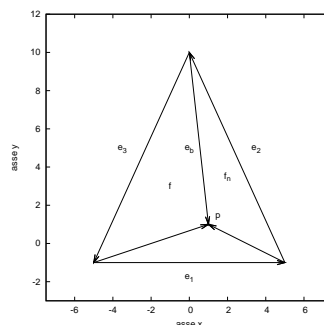
(a) Inserimento del punto



(b) Inserimento del primo arco



(c) Inserimento del secondo arco



(d) Inserimento del terzo arco

Figura 4.6: Inserimento di un nuovo punto p nella faccia f .

In tutti gli altri casi esso è invece conservato ed inserito nella triangolazione, come illustrato nel seguito.

Inserimento ed aggiornamento della struttura

In questa fase, il nuovo punto viene inserito nello spazio determinato dalla faccia f che ne contiene la proiezione (algoritmo 4.4). Si identificano in primo luogo gli archi e_1, e_2, e_3 presenti sul bordo della faccia, e si procede creando un nuovo arco e_b tra il punto p ed il vertice origine di e_1 . Tale arco avrà inizialmente la stessa faccia f sia sulla sinistra che sulla destra. Inoltre, effettuando l'operazione di giunzione `CongiungiArchi` si sistemano i collegamenti di e_b con l'arco e_1 .

Per ognuno dei restanti vertici della faccia originaria f , si provvede a con-

nettere i medesimi al nuovo punto. L'operazione $\text{ConnettiArchi}(x, y)$ crea un nuovo arco avente come origine la destinazione di x e come destinazione l'origine di y . Contestualmente, si crea una nuova faccia f_n e la si associa agli archi preesistenti. Al termine di queste operazioni le facce saranno diventate tre, ed il punto p sarà completamente connesso ai vertici che lo circondano (figura 4.6).

Algoritmo 4.4 Inserimento del punto nel grafo

Input: nuovo punto p , faccia f contenente la proiezione di p

```

for  $i = 1 \rightarrow 3$  do
     $e_i \leftarrow \text{ArcoFaccia}(f, i)$ 
end for
 $e_b \leftarrow \text{NuovoArco}(\text{OrigineArco}(e_1), p)$ 
 $\text{FacciaSinistra}(e_b) \leftarrow f$ 
 $\text{FacciaDestra}(e_b) \leftarrow f$ 
 $\text{CongiungiArchi}(e_b, e_1)$ 
for  $i = 1 \rightarrow 2$  do
     $f_n \leftarrow \text{NuovaFaccia}$ 
     $\text{FacciaSinistra}(e_i) \leftarrow f_n$ 
     $\text{FacciaDestra}(e_b) \leftarrow f_n$ 
     $e_b \leftarrow \text{ConnettiArchi}(e_i, \text{ArcoSimmetrico}(e_b))$ 
     $\text{FacciaSinistra}(e_b) \leftarrow f_n$ 
end for
 $\text{FacciaDestra}(e_b) \leftarrow f$ 

```

A questo punto bisogna assicurarsi che la triangolazione così modificata sia ancora una triangolazione di Delaunay. Il criterio utilizzato nel caso bidimensionale, ovvero l'assenza di altri punti all'interno della circonferenza circoscritta ad ogni triangolo, non è valido in uno spazio a tre dimensioni.

Si analizzano pertanto tutti gli archi che potrebbero dover essere modificati dopo l'inserimento del punto p (algoritmo 4.5). In particolare, per ogni arco e_i si memorizzano l'origine s e la destinazione d ; si prende nota poi

della destinazione t dell'arco precedente rispetto all'origine, ovvero dell'arco più vicino muovendosi ruotando in senso orario intorno all'origine dell'arco.

Qualora osservando t dal punto o esso si trovi dal lato destro dell'arco e_i , si analizza il triangolo s, d, t per controllare se il nuovo punto p si trova all'interno della sfera circoscritta a tale triangolo. Se anche questa condizione è soddisfatta, si procede a scambiare l'arco e_i con un nuovo arco il quale connette i punti t, p , avendo cura di sistemare correttamente le facce di tutti gli archi del quadrilatero di vertici s, t, d, p .

Algoritmo 4.5 Capovolgimento di alcuni archi

Input: nuovo punto p , osservatore o , insieme degli archi da esaminare E

```

for all  $e_i \in E$  do
   $s \leftarrow \text{OrigineArco}(e_i)$ 
   $d \leftarrow \text{DestinazioneArco}(e_i)$ 
   $t \leftarrow \text{DestinazioneArco}(\text{ArcoPrecedenteOrigine}(e_i))$ 
  if  $\text{LatoArco}(s, d, t, o) = \text{destra} \wedge \text{InternoCircosfera}(p, s, d, t)$  then
     $\text{ScambiaArco}(e_i)$  {origine, destinazione, faccia sinistra, faccia destra}
  end if
end for

```

Il capovolgimento degli archi per soddisfare il requisito di Delaunay avviene quindi secondo una determinata proposizione logica. Essa è formata combinando due parti: la prima serve ad assicurarsi che il punto t si trovi rigorosamente a destra dell'arco e_i ; la seconda è un'approssimazione del criterio di ottimalità di Delaunay nel caso tridimensionale.

Il predicato geometrico LatoArco , illustrato nell'algoritmo 4.6, effettua il calcolo del volume del tetraedro (s, d, t, o) . Per quanto visto precedentemente, il segno del volume del tetraedro varia in base all'ordinamento dei vertici s, d, t osservati da o . Se esso è positivo i vertici sono in senso orario, pertanto il punto t si trova a destra dell'arco (s, d) . Viceversa, un volume negativo significa che i vertici sono in senso antiorario, e che quindi il punto t si trova a sinistra dell'arco (s, d) .

Per semplicità, quando il volume V è quasi nullo, ovvero $V < |\varepsilon|$, si ipotizza che t giaccia sulla retta passante per s e d . Questa situazione viene gestita come caso separato, in altre parole si afferma che il punto non sta né a sinistra né a destra dell'arco.

Algoritmo 4.6 Il predicato geometrico `LatoArco`

Input: estremi dell'arco (s, d) , punto da esaminare t , osservatore o

Output: il lato in cui si trova t rispetto all'arco (s, d)

$V \leftarrow \text{VolumeTetraedro}(s, d, t, o)$

if $V < -\varepsilon$ **then**

return *sinistra*

else if $V > \varepsilon$ **then**

return *destra*

else

return *centro*

end if

Il criterio di pseudo-ottimalità di Delaunay `InternoCircosfera` viene verificato calcolando dapprima il circocentro della faccia triangolare (s, d, t) sul piano passante per questi tre punti, ovvero il centro della circonferenza circoscritta al triangolo.

Per sicurezza, nello spazio tridimensionale si verificano le distanze tra il circocentro ed i vertici del triangolo, prendendo la distanza massima come raggio della sfera. Il punto p risulta essere contenuto nella sfera circoscritta alla faccia (s, d, t) semplicemente se la distanza tra punto e circocentro è minore del raggio della sfera.

4.1.3 Problemi incontrati

Vi sono casi in cui l'algoritmo finora esposto è perfettamente funzionante, altri in cui il risultato ottenuto non è affatto soddisfacente. I problemi riscontrati sono riconducibili ad alcuni limiti che non è stato possibile superare. Nel seguito vengono forniti maggiori dettagli per ognuna di esse.

Algoritmo 4.7 Il criterio di pseudo-ottimalità `InternoCircosfera`

Input: punto da esaminare p , estremi del triangolo a, b, c **Output:** *vero* nel caso il punto sia contenuto nella sfera circoscritta

```
 $d \leftarrow \text{Circocentro}(a, b, c)$   
 $r_a \leftarrow \text{DistanzaPuntoPunto}(d, a)$   
 $r_b \leftarrow \text{DistanzaPuntoPunto}(d, b)$   
 $r_c \leftarrow \text{DistanzaPuntoPunto}(d, c)$   
 $r \leftarrow \max\{r_a, r_b, r_c\}$   
 $r_p \leftarrow \text{DistanzaPuntoPunto}(d, p)$   
return  $r_p < r$ 
```

Mancanza di informazioni volumetriche

La triangolazione di Delaunay è definita per un piano e la sua diretta estensione è la tetraedrizzazione, basata su relazioni tra entità dello spazio. Pur essendo il problema in esame sostanzialmente bidimensionale, quando la posizione relativa del punto tridimensionale rispetto alla superficie è determinabile con un predicato geometrico, vi sono casi in cui tale informazione risulta ambigua o compromessa.

In primo luogo, ci sono casi in cui vengono meno le informazioni sul volume del tetraedro, utilizzate come visto precedentemente in pressoché tutti i passi della procedura: per stabilire entro quale faccia vada inserito un nuovo punto, determinare la visibilità di una faccia, individuare l'orientazione reciproca tra un punto ed un segmento.

A titolo d'esempio, si pensi ad una lettura di una superficie "quasi" parallela alla direzione dei raggi laser. L'osservatore in tal caso giace praticamente sullo stesso piano degli altri tre punti del tetraedro. Considerate le distanze in gioco, circa 10^{-3} m, si ottiene un volume piuttosto piccolo, talvolta causa di problemi numerici durante l'elaborazione.

Sono state infatti osservate le seguenti situazioni: notevole perdita di precisione delle cifre decimali, senza conseguenze per la determinazione del

segno del volume; cambiamento di segno rispetto al valore teorico, con successiva inversione del valore dei predicati; azzeramento del volume calcolato ed impossibilità di individuare se esso sia positivo o negativo.

Per tentare di risolvere questo problema, in un primo momento sono stati adottati metodi di aritmetica esatta per ottenere risultati numerici più affidabili. In particolare ci si è avvalsi delle procedure disponibili in [35] per i test di orientamento, appositamente concepiti per gli algoritmi di triangolazione.

Al tempo stesso, ove possibile, è stata limitata la validità del volume soltanto per valori $V > |\varepsilon|$. Volumi troppo piccoli in valore assoluto identificano una riduzione del problema da tridimensionale a bidimensionale, per esempio a causa della complanarità dell'osservatore rispetto agli altri punti. In questi casi l'orientazione è stata determinata mediante criteri basati sull'area e non sul volume.

Difetti nei criteri di pseudo-ottimalità di Delaunay

Un'altra possibile causa di ambiguità è il criterio di pseudo-ottimalità di Delaunay utilizzato. Inizialmente è stato impegnato il metodo `InternoCilindro`, che accetta i parametri (p, a, b, c, n) ed assume valore *vero* quando il punto p è incluso nel più piccolo cilindro contenente a, b, c diretto verso la direzione identificata dalla normale n . Il predicato `InternoCilindro` è l'equivalente locale del test sulla circonferenza circoscritta impiegato nella definizione della triangolazione di Delaunay. Tale criterio non è risultato sufficientemente generale per essere utilizzato.

È stato quindi ipotizzato che la sfera circoscritta fosse una valida alternativa al test sulla circonferenza circoscritta nel caso bidimensionale. Durante la fase sperimentale si è verificato che questa supposizione è valida nella maggior parte dei casi. Non è stato possibile però dimostrare la validità del criterio per ogni configurazione dei punti nello spazio.

Ambiguità dei predicati geometrici

I predicati geometrici come `LatoArco` soffrono dei problemi legati alla mancanza di informazioni volumetriche. A causa di essi, alcune relazioni spaziali per l'orientamento sulla superficie non rispettano le proprietà di simmetria/anti-simmetria e triangolarità.

L'inverarsi di entrambe le proposizioni “ a è a sinistra del lato e ” e “ a è a destra del lato e ” dà luogo a fastidiosi cicli infiniti, difficili sia da identificare che da evitare. Questo accade sia durante la fase di ricerca della faccia che include la proiezione del nuovo punto, sia quando si capovolgono gli archi che non rispettano la condizione di Delaunay. Diversamente da quanto effettuato da Brown e Faigle [10] per l'eliminazione di cicli presenti nell'algoritmo di Guibas e Stolfi [18], non è stato possibile trovare una soluzione a questo problema.

4.2 Relazioni di prossimità nelle scansioni

In questa sezione viene trattato un algoritmo che permette di utilizzare le relazioni di prossimità implicite in un insieme di punti acquisiti tramite particolari sensori. Queste relazioni potrebbero essere utili per effettuare ad esempio una segmentazione, oppure per raggruppare tra loro punti che soddisfano un particolare criterio di distanza, etc.

In un'immagine di prossimità, per esempio, ogni pixel possiede dei vicini. Ogni vicino può avere valori che sono distanti meno di una certa soglia dal valore del pixel in esame, oppure valori superiori. Nel primo caso è probabile che il pixel ed il suo vicino corrispondano a punti dello stesso oggetto, nel secondo caso invece ad oggetti differenti.

Se esaminiamo la struttura di una singola scansione laser, notiamo che i punti giacciono su un piano passante attraverso di essi ed il centro del sensore. Associando un indice i ad ogni raggio, avremo per ogni punto p_i un certo insieme di vicini $\{p_{i-1}, p_{i+1}, p_{i-2}, p_{i+2}, \dots\}$ corrispondenti ai raggi precedenti e successivi.

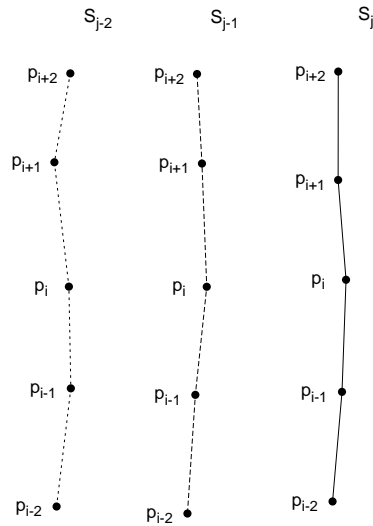


Figura 4.7: Relazioni di prossimità tra scansioni laser consecutive.

Qualora un sensore laser sia montato su un braccio robotico in grado di muoversi lungo le tre dimensioni, l'insieme delle scansioni costituisce una nuvola di punti. Sfruttando la coerenza temporale tra scansioni consecutive, ovvero ipotizzando di effettuare traiettorie di acquisizione continue, possiamo determinare altre relazioni di vicinato. Un punto p_i della scansione S_j corrente avrà come probabili vicini i punti

$$\{ \{p_i, p_{i-1}, p_{i+1}, \dots\}_{j-1}, \{p_i, p_{i-1}, p_{i+1}, \dots\}_{j-2}, \dots \}$$

appartenenti alle scansioni S_{j-1}, S_{j-2}, \dots precedenti (figura 4.7).

4.2.1 Utilizzo della struttura dei dati

Si potrebbe quindi pensare di costruire un insieme ordinato di punti, in modo da preservare le relazioni di vicinato. Disponendo di una nuvola siffatta, tutte le elaborazioni successive sarebbero enormemente avvantaggiate, poiché ottenere i k -vicini di un punto diverrebbe un'operazione indipendente sia

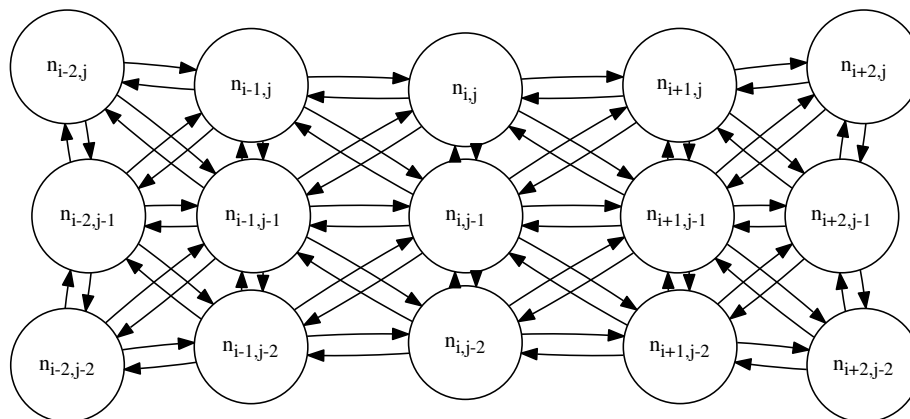


Figura 4.8: Esempio di grafo di prossimità per $v = 8$.

da k che dalla dimensione della nuvola stessa. Le relazioni di prossimità precedentemente evidenziate possono essere quindi sfruttate per ricercare in maniera molto efficiente i vicini di ogni punto.

In questa tesi è stata realizzata una struttura in grado di memorizzare, incrementalmente e per ogni punto, un numero prestabilito v di vicini. Questa struttura, chiamata **ProximityGraph**, è un grafo orientato in cui ad ogni nodo corrisponde un punto ed ad ogni arco una relazione di prossimità. Per questo motivo viene denominato “grafo di prossimità”. In figura 4.8 si trova un esempio di **ProximityGraph**, del quale non sono stati riportati tutti gli archi per motivi di praticità.

L’aggiunta di un nuovo nodo, ossia di un nuovo punto, ad un grafo di prossimità esistente si basa sulla presenza di uno o più nodi seme. Si effettua una visita in ampiezza del grafo, a partire dai semi, limitandosi a considerare i primi m nodi attraversati. Durante questa visita, si tiene traccia dei nodi attraversati, ordinandoli secondo una distanza specificabile. Così facendo, si determina quali siano i nodi candidati ad essere vicini al nuovo nodo. La procedura viene discussa con maggior dettaglio a pagina 63.

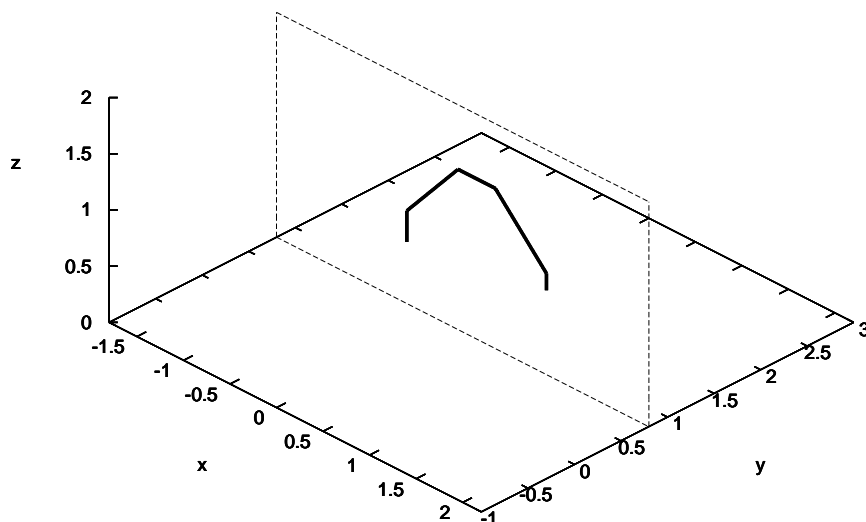


Figura 4.9: Limitazione dei gradi di libertà del manipolatore (piano tratteggiato) ed esempio di traiettoria (in grassetto).

L'ipotesi di continuità delle letture metriche non può essere trascurata. I movimenti congiunti del sensore e del braccio robotico devono essere tali da impedire bruschi cambiamenti del piano di scansione. Non sono stati impiegati particolari algoritmi per la pianificazione del punto di vista ottimale per rispettare i vincoli di continuità. Si è preferito procedere in maniera empirica, utilizzando traiettorie impostate manualmente.

Per comodità, i gradi di libertà del manipolatore sono stati ridotti da 6 a soltanto 3: posizioni x e z , orientazione γ . Quest'ultima è stata inoltre vincolata in funzione delle prime due, in modo da garantire l'osservazione del volume di interesse (figura 4.9). Per le classi di oggetti esaminate in questa tesi, suddette limitazioni non hanno comportato problemi di alcun tipo.

Data in ingresso una serie di scansioni soddisfacenti tutti i requisiti sopra elencati, è necessario provvedere all'inserimento delle medesime nel grafo di prossimità. Volendo costruire una nuvola di punti parallelamente ad un grafo

contenente informazioni sul vicinato, bisogna eseguire ciclicamente alcuni passaggi.

- Acquisizione della scansione, con relativa trasformazione dei punti da coordinate polari a coordinate cartesiane sul piano di scansione; trasformazione dei punti rispetto all'origine del manipolatore, determinabile mediante le matrici di rototraslazione della terna base del manipolatore, della terna dell'organo utensile, e del sensore stesso.
- Filtraggio dei punti, sia per quanto riguarda l'appartenenza o meno ad un preciso volume di lavoro, sia per l'eventuale presenza di *outliers*; nonostante il filtraggio possa anche essere effettuato in fase finale (su tutto l'insieme di scansioni), è più vantaggioso metterlo in pratica incrementalmente evitando così un inutile appesantimento del grafo e la pericolosa introduzione di falsi vicini.
- Inserimento della scansione nella nuvola di punti complessiva ed aggiornamento della struttura `ProximityGraph`, ossia il grafo di prossimità, con ciascuno dei nuovi punti inseriti; essendo piuttosto complessa, questa fase viene descritta nel seguito.

Inserimento di una scansione nel grafo di prossimità

Innanzitutto è necessario specificare una funzione distanza $\mathcal{D}(\cdot, \cdot)$ che verrà usata durante la costruzione del vicinato. Nel nostro caso è stata usata la funzione distanza euclidea nello spazio tridimensionale. Tale funzione fa uso della notazione $\mathcal{P}(n)$ per risalire al punto p corrispondente al nodo n . Questo è ovviamente indispensabile, siccome si vuole calcolare la distanza tra punti, non tra nodi. Si noti infatti che non è stata definita alcuna misura di distanza tra due nodi.

Come precedentemente accennato, la procedura di inserimento (algoritmo 4.8) fa uso di uno o più nodi seme. Per praticità questi semi verranno inseriti in una lista L , inizialmente vuota. Dopodiché, si comincia con l'inserimento di un punto p_i alla volta creando un nodo n_i .

Per tutti i nodi a parte il primo, se una determinata condizione viene rispettata allora si inserisce il nodo precedente nella lista dei semi L . La condizione è una soglia d sulla distanza tra i punti relativi ai due nodi: se è troppo eccessiva probabilmente il nodo precedente non è un buon seme, quindi non lo si considera.

Si procede poi integrando la medesima lista L con i semi t_1, t_2, \dots ricercati all'interno delle precedenti scansioni. A tale scopo, è necessario definire una finestra di dimensioni $w_s \times 2w_p$. L'analisi di eventuali semi viene infatti compiuta nelle w_s scansioni precedenti, considerando per ogni scansione w_p punti con indice inferiore ed altrettanti con indice maggiore.

Nel caso piuttosto sfavorevole in cui la lista sia ancora vuota, si procede con una ricerca di semi g_1, g_2, \dots all'interno dell'intera nuvola. Essendo un'operazione molto dispendiosa, si cerca di limitarne l'uso. Infatti, se si ha cura di effettuare letture adiacenti e non discontinue, questa procedura non viene praticamente mai chiamata.

In ultimo luogo ci si occupa di connettere il nodo n_i con i suoi vicini, adoperando la lista di semi L come inizializzazione per la ricerca. Lista che per finire viene svuotata, al fine di non interferire con le successive iterazioni dell'algoritmo di inserimento.

Nel dettaglio, la creazione degli archi di prossimità per un generico nodo n avviene come riportato nell'algoritmo 4.9. Il funzionamento è basato sull'impiego di code con priorità, ovvero strutture FIFO nelle quali gli elementi vengono mantenuti ordinati secondo un determinato criterio. Grazie a siffatte strutture, l'estrazione di un elemento dalla coda restituisce sempre il contenuto ottimo per il criterio stabilito.

Per questa procedura vengono create due code con priorità Q e C , utilizzando come criterio la minima distanza euclidea dal punto $\mathcal{P}(n)$. Inizialmente, i semi contenuti nella lista L sono inseriti nella coda Q , ovvero ordinati secondo $\mathcal{D}(\mathcal{P}(n), \cdot)$.

Stabilito un numero massimo m di nodi da considerare, si estrae da Q l'elemento più promettente q . Siccome q è il nodo attualmente più prossimo

Algoritmo 4.8 Inserimento di una scansione nel grafo di prossimità.

Input: nuova scansione S_j
 crea una lista di semi $L = \emptyset$
for all $p_i \in S_j$ **do**
 crea un nuovo nodo n_i
 if $i > 0 \wedge \mathcal{D}(\mathcal{P}(n_i), \mathcal{P}(n_{i-1})) < d$ **then**
 aggiorna $L \leftarrow L \cup \{n_{i-1}\}$
 end if
 ricerca i semi t_1, t_2, \dots nelle scansioni precedenti
 aggiorna $L \leftarrow L \cup \{t_1, t_2, \dots\}$
if $L = \emptyset$ **then**
 ricerca i vicini g_1, g_2, \dots nell'intera nuvola
 aggiorna $L \leftarrow L \cup \{g_1, g_2, \dots\}$
end if
 crea gli archi per n_i usando i semi $s \in L$
 svuota la lista $L \leftarrow \emptyset$
end for

ad n , lo si inserisce nella coda dei candidati C . Il nodo q possiede però dei vicini a_1, a_2, \dots che potrebbero essere ancora più prossimi ad n . Per tale motivo, la procedura viene ripetuta inserendo a_1, a_2, \dots nella coda da esaminare Q .

Infine, si prelevano i primi v nodi dalla coda dei candidati C . Ognuno di questi nodi c viene connesso ad n con un arco (n, c) . Così facendo si ottiene che c è un vicino per n , ma non viceversa. Bisogna quindi controllare se sia opportuno aggiornare il vicinato di c per includervi n .

Tra tutti i nodi nell'insieme U adiacenti a c , si calcola quale sia il nodo \bar{u} a distanza massima da c . Nel caso questa distanza sia superiore alla distanza che intercorre tra $\mathcal{P}(c)$ e $\mathcal{P}(n)$, ed il numero di vicini abbia già raggiunto il valore massimo consentito v , si sostituisce l'arco (c, \bar{u}) con l'arco (c, n) . Se invece il vicinato non è ancora saturo ($|U| < v$), ci si limita ad aggiungere

Algoritmo 4.9 Creazione degli archi di prossimità per un nuovo nodo.

Input: nuovo nodo n , lista di semi L

crea due code Q, C ordinate in base a $\mathcal{D}(\mathcal{P}(n), \cdot)$

inserisci tutti i semi contenuti in L nella coda Q

for $1 \rightarrow m$ **do**

estrai il primo nodo q dalla coda Q

inserisci i nodi a_1, a_2, \dots adiacenti a q nella coda Q

inserisci q nella coda C

end for

for $1 \rightarrow v$ **do**

estrai il primo nodo c dalla coda C

connetti n e c con un arco

ricerca i nodi $u \in U$ adiacenti a c

calcola $\bar{u} = \arg \max_{u \in U} \mathcal{D}(\mathcal{P}(c), \mathcal{P}(u))$

if $|U| = v \wedge \mathcal{D}(\mathcal{P}(c), \mathcal{P}(\bar{u})) > \mathcal{D}(\mathcal{P}(c), \mathcal{P}(n))$ **then**

elimina l'arco tra c ed \bar{u}

end if

if $|U| < v$ **then**

connetti c ed n con un arco

end if

end for

l'arco (c, n) relativo al nuovo nodo.

Come si è potuto osservare, il grafo di prossimità è una struttura altamente dinamica. Ad ogni nuovo inserimento, le connessioni esistenti tra i nodi (e quindi tra i punti) sono mutevoli: possono essere aggiunte e rimosse. Il vantaggio rispetto ad altre strutture per la ricerca dei vicini è che queste procedure sono poco onerose.

Ad esempio, nonostante il *kd-tree* preveda la possibilità di aggiungere o rimuovere punti, questo viene spesso evitato poiché causa lo sbilanciamento dell'albero. Per conservare l'efficienza nella ricerca dei vicini, basata su un

Algoritmo 4.10 Ricerca dei k -vicini più prossimi ad un punto.

Input: nodo da interrogare n , numero di vicini k

Output: lista di vicini

crea due code Q, C ordinate in base a $\mathcal{D}(\mathcal{P}(n), \cdot)$

inserisci n nella coda Q

while $|C| < 2k$ **do**

 estrai il primo nodo q dalla coda Q

 inserisci i nodi a_1, a_2, \dots adiacenti a q nella coda Q

if $q \neq n$ **then**

 inserisci q nella coda C

end if

end while

return $\{c_1, \dots, c_k\}$

buon bilanciamento dell'albero, è necessario procedere al ribilanciamento. Una tale operazione è computazionalmente onerosa, soprattutto per gli alberi riferiti ad uno spazio tridimensionale.

Ricerca dei k -vicini di un punto

L'operazione di ricerca dei k -vicini di un punto è piuttosto comune in numerosi algoritmi successivi che agiscono sulla nuvola di punti. Si pensi ad esempio alla rimozione degli *outlier* in base alla distanza media dei vicini, alla suddivisione della nuvola in gruppi contigui di punti, al calcolo delle normali di ogni punto, etc.

Le procedure elencate traggono enorme vantaggio dall'uso del **Proximity-Graph** per trovare il vicinato, poiché talvolta la ricerca è l'operazione che determina la complessità dell'intero algoritmo considerato. Nella sezione 4.2.2 si descrive infatti una tecnica per costruire una triangolazione della superficie dell'oggetto facente uso del grafo di prossimità.

Disponendo di un grafo già costruito, possiamo interrogare la struttura per conoscere quali siano i k punti più prossimi ad un dato punto p . Per far

questo, è necessario prima ottenere il nodo n mediante la funzione inversa $n = \mathcal{P}^{-1}(p)$. Dopodiché si creano due code con priorità Q e C , utilizzando il consueto criterio di ordinamento degli elementi in base alla minima distanza $\mathcal{D}(\mathcal{P}(n), \cdot)$. La coda Q viene inizialmente riempita con il nodo n , mentre C rimane vuota.

Si procede con l'estrarre un nodo dalla coda Q , che nel primo caso sarà pari ad n . Quindi, i nodi a_1, a_2, \dots adiacenti ad n sono prelevati ed inseriti nella coda Q . Dopodiché, si ripete l'estrazione di un nodo q e si inseriscono i relativi vicini nella coda Q . Se inoltre q risulta essere diverso da n , lo si aggiunge alla coda dei candidati C .

Il ciclo viene effettuato fin quando C non possiede $2k$ elementi. Questa fase è necessaria per assicurarsi di visitare nodi a sufficienza, poiché k potrebbe essere superiore rispetto a v . Per concludere, i k -vicini del nodo n sono prelevati estraendo k elementi dalla coda C dei candidati.

Rimozione di singoli punti dal grafo di prossimità

Così come è possibile inserire un punto nel grafo di prossimità, è ovviamente concesso anche rimuoverlo. In generale la rimozione di punti è diretta conseguenza di una catena di elaborazioni, ciascuna delle quali ha come ingresso la nuvola in uscita dallo stadio precedente. Soprattutto dopo filtraggi ed estrazioni di gruppi di punti, infatti, è d'obbligo eliminare dal `ProximityGraph` gli stessi punti tolti dalla nuvola associata.

Di per sé, la rimozione di un punto è un'operazione molto semplice. L'aspetto problematico è la gestione degli archi precedentemente connessi al nodo che rappresentava tale punto. Per tale motivo, si costruisce in primo luogo l'insieme di nodi $r \in R$ per cui l'arco (r, n) esiste. Solo a questo punto si procede con l'eliminazione del nodo n , congiuntamente a tutti gli archi entranti o uscenti da esso.

In seguito, per ogni nodo interessato dalla rimozione si cercano i relativi v -vicini, con v numero massimo di collegamenti uscenti per ogni nodo. Se i nodi $c \in C$ trovati non sono già connessi ad r , allora li si connette con un

Algoritmo 4.11 Rimozione di un singolo punto dal grafo di prossimità.

Input: nodo n da rimuoverericerca i nodi $r \in R$ aventi un arco entrante in n elimina gli archi uscenti o entranti da n rimuovi il nodo n dal grafo**for all** $r \in R$ **do**ricerca i nodi $c \in C$ che sono v -vicini del nodo r **for all** $c \in C$ **do****if** non esiste l'arco (r, c) **then**connetti r e c con un arco**end if****end for****end for**

nuovo arco. In altre parole, si cerca di ripristinare le connessioni perse utilizzando nodi adiacenti.

4.2.2 Costruzione di *mesh*

Un interessante articolo di Dumitriu ed altri [13] illustra una particolare tecnica per ricostruire la superficie di un oggetto. Partendo dall'osservazione che tutti gli algoritmi fanno uso di un qualche tipo di predicato geometrico, e che realizzare in maniera robusta ed affidabile tali predicati è spesso piuttosto complesso, gli autori propongono un approccio nel quale la geometria è abbandonata prima possibile in favore di una struttura a grafo.

In particolare, hanno dimostrato che è possibile ricostruire una superficie facendo uso solamente delle informazioni relative alla prossimità dei punti, utilizzandole intelligentemente per costruire un grafo di adiacenze. La topologia della superficie originale viene fedelmente riprodotta sotto l'ipotesi che l'oggetto sia stato campionato, e quindi acquisito, in maniera sufficientemente densa. Ciononostante, piccole zone non uniformi all'interno della nuvola di punti non comportano il degradarsi dei risultati.

Nel nostro caso, disponendo delle informazioni di prossimità grazie alla struttura `ProximityGraph`, si può pensare di effettuare svariate modifiche all'algoritmo originale come descritte in seguito. La superficie può essere ricostruita in maniera ancora più vantaggiosa, adoperando per l'appunto le informazioni sul vicinato derivanti dalla natura del sensore laser e dalla configurazione del manipolatore.

Insieme indipendente massimale

L'algoritmo originale prevede *in primis* la creazione di un grafo di vicinato G che connetta simbolicamente ogni punto ai suoi k -vicini. Fortunatamente questa fase può essere saltata, poiché la struttura a grafo di prossimità prima illustrata, sebbene non esattamente corrispondente, fa le veci di un tale grafo G . Si ricorda infatti che nel `ProximityGraph` ogni nodo possiede v vicini, e che mediante una semplice procedura di ricerca (algoritmo 4.10) è possibile risalire ad un vicinato di dimensione k qualsiasi.

Il passo successivo consiste nel calcolare un sottoinsieme indipendente massimale ad h passi del grafo G , ovvero trovare un insieme massimale S tale per cui due vertici qualsiasi appartenenti ad S sono tra loro distanti almeno h passi. In questo caso, per distanza si intende il numero di nodi da attraversare (ultimo compreso) per passare da un nodo ad un altro.

La procedura per il calcolo è illustrata nell'algoritmo 4.12. Si creano inizialmente due insiemi vuoti T ed S , che serviranno rispettivamente a contenere i nodi visitati ed i nodi selezionati come sottoinsieme indipendente massimale, e si entra poi in un ciclo che avrà termine soltanto quando tutti i nodi del grafo G saranno stati visitati.

Alla prima iterazione si procede ad impostare s , il nodo dal quale avrà inizio l'operazione di espansione mostrata a seguire, selezionandolo a piacere tra tutti i nodi non visitati; quindi si crea una coda *first-in/first-out* Q contenente il solo s . Lo stesso nodo viene poi marcato come visitato, ed inserito nel sottoinsieme indipendente massimale S . La funzione $\mathcal{F}(\cdot)$, che

Algoritmo 4.12 Calcolo di un insieme indipendente massimale ad h passi.

Input: grafo di vicinato G , numero di passi h

Output: sottoinsieme indipendente massimale S

crea un insieme di nodi visitati $T = \emptyset$

crea il sottoinsieme indipendente massimale $S = \emptyset$

while $G \neq T$ **do**

$s \leftarrow g \in (G \setminus T)$

 [*] crea una coda FIFO $Q = \{s\}$

$\mathcal{F}(s) \leftarrow 0, T \leftarrow T \cup \{s\}, S \leftarrow S \cup \{s\}$

while $Q \neq \emptyset$ **do**

 estrai il primo nodo q dalla coda Q

 ricerca i nodi $a \in A$ adiacenti a q

for all $a \in (A \setminus (A \cap T))$ **do**

$\mathcal{F}(a) \leftarrow \mathcal{F}(q) + 1, T \leftarrow T \cup \{a\}$

if $\mathcal{F}(a) = h$ **then**

$s \leftarrow a$

jump [*]

end if

 aggiungi il nodo a alla coda Q

end for

end while

end while

return S

ha lo scopo di tener traccia della profondità delle visite effettuate contando il numero di passi effettuati, viene posta uguale a zero.

Dentro ad un ulteriore ciclo viene estratto il primo nodo q dalla coda Q , e ne vengono memorizzati tutti i nodi adiacenti $a \in A$. Si tratta poi ciascuno dei nodi adiacenti a non visitati, marcandolo come visitato, e si incrementa di uno il contatore di profondità $\mathcal{F}(a)$ rispetto ad $\mathcal{F}(q)$. Nel caso in cui tale contatore abbia raggiunto il numero desiderato h , si considera a come nuovo nodo di inizio e si ricomincia il ciclo principale. Altrimenti, si inserisce il nodo a in fondo alla coda Q e si prosegue fino a quando essa non sarà vuota.

Diagramma di Voronoi sul grafo

Una volta in possesso di un sottoinsieme indipendente massimale S , è necessario costruire il diagramma di Voronoi “sul grafo” rispetto ad esso. In poche parole questo è un diagramma di Voronoi discreto nel quale si associa ad ogni sito $s \in S$ una regione di influenza, composta da tutti quei nodi che hanno s come loro sito più prossimo in termini di passi sul grafo.

Un simile diagramma può essere rappresentato con un ulteriore grafo L , in cui ogni nodo corrisponde ad un sito s . Tale grafo L può essere costruito in maniera piuttosto semplice: è sufficiente una ricerca parallela in ampiezza nel grafo G a partire dai nodi in S (algoritmo 4.13).

Dopo aver creato una coda Q di tipo *first-in/first-out*, la si riempie con i nodi s contenuti nel sottoinsieme indipendente massimale. Per ognuno dei medesimi nodi s si effettuano altre due operazioni:

- si memorizza il sito associato al nodo s , ovvero in questo caso il sito stesso, denotando l'operazione mediante $\mathcal{L}(s)$;
- si crea un nuovo nodo associato al nodo s all'interno del grafo L , adoperando la notazione $l(s)$ per identificarlo.

Il primo nodo q della coda Q viene dunque estratto, ed i suoi nodi adiacenti su G vengono ricercati. Per ogni nodo adiacente $a \in A$, si agisce in maniera differente in base al valore del sito associato $\mathcal{L}(a)$:

- se non esiste nessun sito associato, si memorizza lo stesso sito associato a q , ossia $\mathcal{L}(q)$; quindi si inserisce a nella coda FIFO Q ;
- se il sito associato è differente rispetto a $\mathcal{L}(q)$, allora ci si trova in una zona di confine; i nodi $l(\mathcal{L}(a))$ ed $l(\mathcal{L}(q))$ vengono pertanto connessi tramite un arco.

La procedura prosegue fintantoché sono presenti nodi all'interno di Q , restituendo infine il grafo L .

Algoritmo 4.13 Creazione di un grafo L per la rappresentazione del diagramma di Voronoi di G rispetto ad S .

Input: grafo G , sottoinsieme indipendente massimale S

Output: grafo L per il diagramma di Voronoi

```

crea una coda FIFO  $Q = \emptyset$ 
for all  $s \in S$  do
     $Q \leftarrow Q \cup \{s\}$ 
     $\mathcal{L}(s) \leftarrow s$ 
    crea un nuovo nodo  $l(s)$  in  $L$ 
end for
while  $Q \neq \emptyset$  do
    estrai il primo nodo  $q$  dalla coda  $Q$ 
    ricerca i nodi  $a \in A$  adiacenti a  $q$ 
    for all  $a \in A$  do
        if  $\mathcal{L}(a)$  non è impostato then
             $\mathcal{L}(a) \leftarrow \mathcal{L}(q)$ 
            inserisci il nodo  $a$  nella coda  $Q$ 
        else if  $\mathcal{L}(a) \neq \mathcal{L}(q)$  then
            connetti  $l(\mathcal{L}(a))$  ed  $l(\mathcal{L}(q))$  con un arco in  $L$ 
        end if
    end for
end while
return  $L$ 

```

Riconoscimento dei cicli minimi

Dopo aver eseguito i precedenti passaggi, siamo in possesso delle relazioni di adiacenza tra le regioni di Voronoi sul grafo G , rappresentate mediante gli archi del grafo L . Determinando i cicli minimi presenti nel grafo L , si è in grado di risalire alle facce che approssimano la superficie dell'oggetto acquisito.

Per trovare i cicli minimi viene in aiuto il test di planarità di Boyer e Myrvold [9]. Questo test permette di stabilire se un grafo possa essere disegnato in un piano senza che nessuno dei suoi archi si intersechi. Ogni grafo planare appartiene ad una classe di equivalenza chiamata *embedding*, ed è definibile mediante una lista degli archi adiacenti in senso orario ad ogni nodo nel grafo.

L'*embedding* per il grafo L suddivide il piano in più regioni, chiamate facce, limitate da cicli minimi del grafo. Una volta ottenute tali facce, è possibile trasformarle in facce nello spazio tridimensionale sfruttando la conoscenza delle coordinate dei nodi l . La superficie dell'oggetto viene finalmente ricostruita congiungendo tra loro tutte queste facce.

Capitolo 5

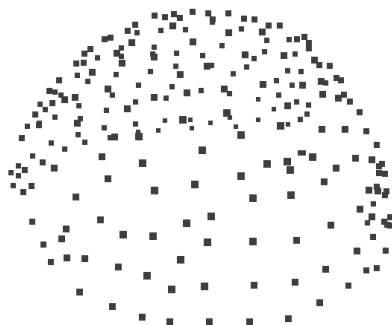
Risultati

In questo capitolo si presentano i risultati relativi agli algoritmi trattati durante questo lavoro di tesi. In primo luogo si confrontano i due approcci, uno basato sulla triangolazione di Delaunay, l'altro sulle relazioni di prossimità, evidenziandone pregi e difetti. Si discutono quindi i tempi di esecuzione di una ricerca all'interno del grafo di prossimità, confrontandoli con una struttura per il partizionamento dello spazio quale il *kd-tree*, punto di riferimento in letteratura. Si mostrano infine alcuni esperimenti di manipolazione svolti.

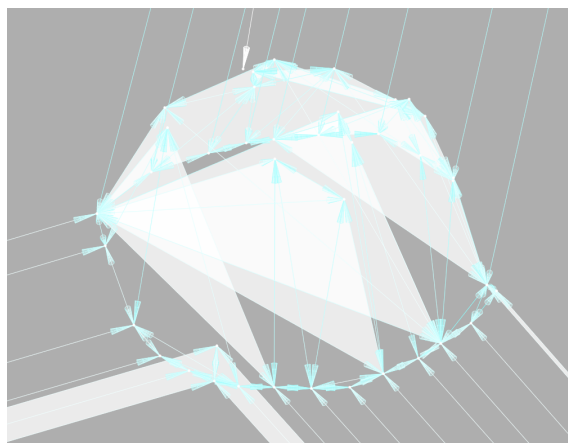
5.1 Costruzione di *mesh*

È stato sviluppato il metodo per la costruzione di triangolazioni di pseudo-Delaunay descritti nella sezione 4.1. In particolare, il sistema si compone di un *quad-edge* per la rappresentazione di suddivisioni, e di procedure per la costruzione incrementale. In figura 5.1 è riportato il risultato parziale della costruzione di un oggetto.

Come anticipato, talvolta il risultato della ricostruzione non è soddisfacente. I problemi riscontrati, elencati nel seguito, sono stati attentamente analizzati, ed ove possibile si è tentato di porvi un rimedio. Per alcune particolari configurazioni di punti il volume del tetraedro è indeterminato,



(a) Nuvola di punti relativa ad un oggetto.



(b) Ricostruzione parziale della superficie.

Figura 5.1: Risultato parziale della ricostruzione di un oggetto.

rendendo così impossibile la riduzione del problema di localizzazione da tridimensionale a bidimensionale. Non è stato possibile dimostrare la validità di un generico criterio di ottimalità di Delaunay per ogni configurazione dei punti nello spazio, limitandosi pertanto all'analisi dei criteri con risultati migliori. Infine, non è stata trovata soluzione per l'innescarsi di cicli infiniti durante la visita del grafo.

In aggiunta è stato realizzato il metodo per la costruzione di *mesh* basata su relazioni di prossimità discusso nella sezione 4.2. Invece di utilizzare primitive geometriche, che soffrono di problemi di robustezza ed affidabilità,

si impiega il grafo di prossimità ottenuto sfruttando le caratteristiche del sistema di acquisizione. Tutte le fasi di costruzione consistono in operazioni topologiche su tale struttura. I benefici in termini di tempi di esecuzione vengono meglio illustrati nella prossima sezione.

5.2 Tempi di ricerca dei vicini

Numerosi algoritmi, inclusi quelli sviluppati e quelli presenti nella *Point Cloud Library*, fanno uso delle informazioni sul vicinato di ogni punto in una nuvola. L'operazione di ricerca è piuttosto comune: ad esempio, nel filtraggio di rimozione degli *outlier* è necessario calcolare la distanza media di ogni punto dai suoi k -vicini. Ancora, il raggruppamento di punti in sottoinsiemi omogenei viene effettuato proprio in base alla distanza dei punti dal relativo vicinato.

Le procedure sopra elencate si appoggiano a strutture dati classiche per la localizzazione di punti, come il *kd-tree* [6], che partizionano lo spazio k -dimensionale. In molti algoritmi, in particolare quelli supportati dalla libreria PCL, la fase di ricerca è l'operazione più dispendiosa in termini computazionali, che determina la complessità dell'intero algoritmo.

Il grafo di prossimità illustrato nella sezione 4.2 consente di svolgere operazioni di ricerca del k vicinato e dell'intorno. Tale struttura dati permette di effettuare la ricerca in tempi pressoché costanti, purché venga scelto un opportuno numero di vicini. Il grafo di prossimità è stato integrato tramite un'appropriata interfaccia con la *Point Cloud Library*.

È quindi lecito domandarsi quali siano le prestazioni delle strutture per la localizzazione dei vicini in termini di tempo di ricerca. Nella tabella 5.1 sono riportati i tempi medi di ricerca del vicinato di un punto. Tali tempi sono relativi alla versione del *kd-tree* attualmente presente nella libreria PCL, interrogando la struttura per un vicinato di dimensione $k = 10$.

Sempre in tabella, sono presenti i tempi medi di ricerca relativi ad grafo di prossimità costruito sugli stessi dati sperimentali del *kd-tree*. Per la co-

Esperimento	Dimensione nuvola	Grafo di prossimità		<i>kd-tree</i>
		Correttezza del vicinato	Tempo medio di ricerca (μs)	Tempo medio di ricerca (μs)
Doll1	21578	86.1%	3.71	6.02
Doll2	10681	90.4%	2.80	4.68
Jug1	20912	84.4%	3.35	6.21
Jug2	22402	85.5%	3.12	5.80
Horse1	13044	92.2%	3.06	5.37
Horse2	15092	85.3%	3.31	5.30
Table1	18577	94.3%	2.69	5.92
Table2	30946	97.4%	3.23	6.46
T1	6244	98.1%	3.20	6.41
T2	12231	79.9%	2.45	4.91

Tabella 5.1: Risultati della ricerca dell'intorno di un punto mediante grafo di prossimità e *kd-tree*.

struzione, sono state adoperate finestre di dimensione $w_s = 5$ per le scansioni precedenti e $2w_p = 10$ per i raggi laser adiacenti. Come si può notare dai dati, per ciascun esperimento considerato i tempi di esecuzione di ogni singola ricerca sono circa la metà dell'equivalente *kd-tree*. Bisogna doverosamente osservare che il vicinato ottenuto con il grafo di prossimità ha una percentuale di correttezza pari all'80–90% rispetto al *kd-tree*. La percentuale, risultante comunque in grado di garantire una buona affidabilità, può essere aumentare ulteriormente mediante un'attenta pianificazione della traiettoria.

Nei tempi non viene tenuto in considerazione il sovraccarico necessario per generare le due strutture dati, siccome non sufficientemente oggettivo. Infatti, il grafo di prossimità viene costruito incrementalmente, e l'inserimento di ciascun punto ha un costo pressoché costante. Il medesimo grafo, inoltre, è una struttura dati che supporta l'inserimento o l'eliminazione di punti, e la realizzazione di tali procedure ha un impatto significativo sui tempi di esecuzione. Al contrario, la creazione di un *kd-tree* è un'operazione che deve essere

svolta sull'intero insieme di dati. Ogni modifica di tale insieme richiederebbe la costruzione di un nuovo albero, affinché esso risulti bilanciato.

5.3 Esperimenti di manipolazione

A conclusione del lavoro svolto sono stati messi in pratica alcuni esperimenti, riguardanti un compito di alto livello quale la presa e la manipolazione di un singolo oggetto ricostruito. L'esperimento è consistito nelle seguenti fasi: acquisizione dei dati di prossimità, elaborazione e *clustering* della nuvola di punti, ricostruzione della superficie, identificazione delle parti dell'oggetto, pianificazione del compito e sua esecuzione.

Mediante il software OpenRAVE per la pianificazione del moto ed il calcolo delle prese, è stato ricostruito un ambiente virtuale riprodotto la configurazione del laboratorio. In tale ambiente sono stati collocati gli oggetti ricostruiti, nella posizione corrispondente alla realtà. Si è quindi provveduto a generare una traiettoria che consentisse di prelevare l'oggetto e spostarlo in un altro punto, utilizzando alcune procedure per risolvere la cinematica inversa del manipolatore. La traiettoria nello spazio dei giunti è stata quindi comunicata al manipolatore Comau Smart SiX, ed eseguita con successo nel laboratorio. In figura 5.2 sono mostrate, nell'ordine, il posizionamento della pinza in corrispondenza di una parte dell'oggetto, il movimento del robot con il tavolo stretto nella pinza, il rilascio dell'oggetto nella destinazione desiderata.

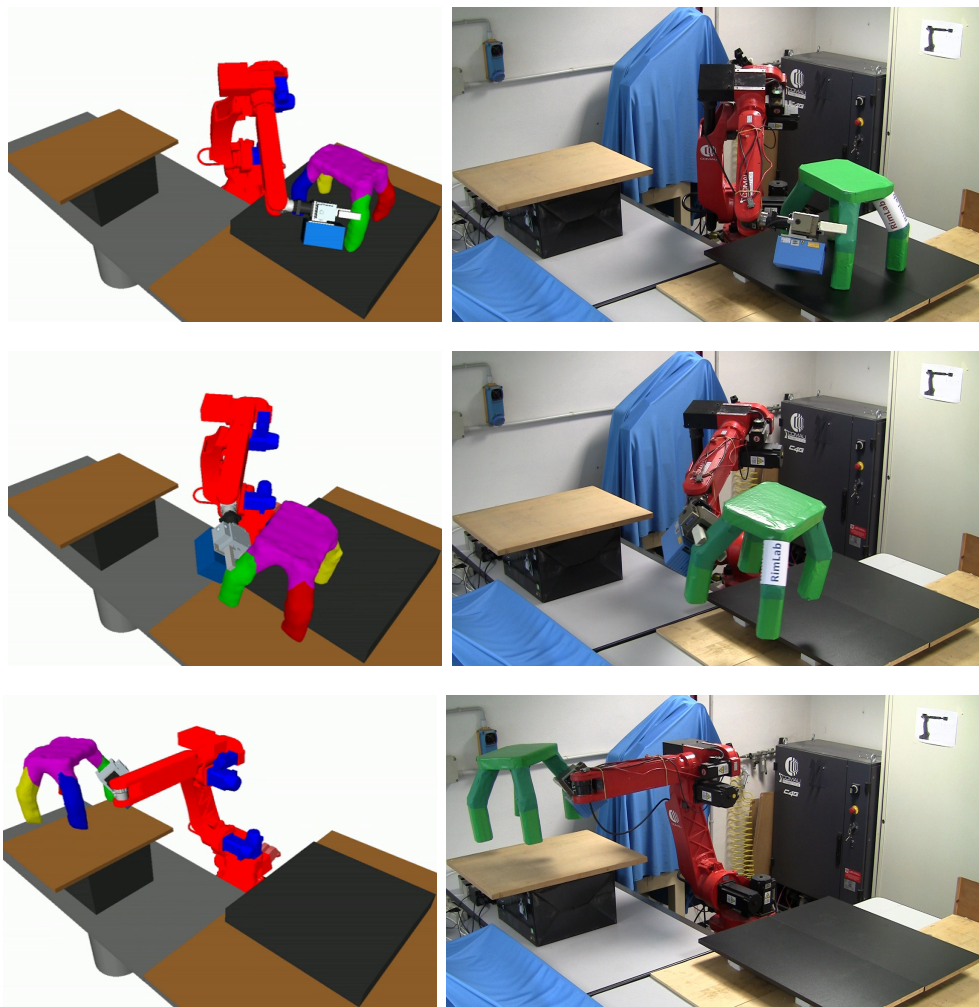


Figura 5.2: Prove sperimentali svolte nell'ambiente OpenRAVE (a sinistra) e nel laboratorio RIMLab (a destra).

Capitolo 6

Conclusioni

In questa tesi è stato presentato lo sviluppo un sistema integrato per l'acquisizione, l'elaborazione, la ricostruzione e la manipolazione di oggetti. In particolare, sono stati mostrati sia i risultati dello studio di ogni singolo sottosistema che quelli relativi all'integrazione del sistema generale.

È stato perfezionato il coordinamento tra i vari moduli coinvolti nell'acquisizione, ossia il programma di controllo del robot manipolatore ed il *driver* per il sensore laser. Inoltre sono state migliorate le prestazioni della fase di trasformazione delle coordinate. Si sono studiate quindi procedure di calibrazione, al fine di acquisire dati più precisi ed irrobustire le successive elaborazioni.

Per quanto riguarda l'elaborazione dei dati acquisiti, sono state studiate le classi di operazioni sulle nuvole di punti, comprendenti tra l'altro la rimozione di eventuale rumore, lo *smoothing*, il sottocampionamento, il *fitting* di un modello del piano, l'estrazione dei raggruppamenti di punti. In particolare, per ciascun tipo di elaborazione sono stati configurati i parametri necessari al fine di ottenere un buon risultato.

Le caratteristiche dell'intero apparato sono state attentamente analizzate, terminando con la formulazione di una serie di ipotesi sul sistema di acquisizione. La prima ipotesi è che ogni punto viene osservato da un determinato punto di vista nello spazio, coincidente con il centro del sensore. La seconda,

che i punti all'interno di una singola scansione giacciono sullo stesso piano, e che inoltre punti consecutivi sono spesso vicini tra loro. La terza, che scansioni temporalmente consecutive sono anche poco distanti nello spazio.

Sono stati seguiti due approcci differenti per attuare la ricostruzione della superficie di un oggetto. Il primo algoritmo sviluppato si avvale delle ipotesi sul punto di vista e sul piano di scansione, ed utilizza una triangolazione di Delaunay allo scopo di suddividere una varietà bidimensionale nello spazio tridimensionale. La struttura dati *quad-edge* è stata adoperata per rappresentare la suddivisione della varietà in vertici, archi e facce, estendendo alcuni risultati ottenuti in passato da Guibas e Stolfi [18].

Il secondo algoritmo, basato sulle relazioni di prossimità all'interno di una scansione e tra scansioni consecutive, consiste nella costruzione di un "grafo di prossimità" in grado di tenere traccia del vicinato di ogni punto. È stata quindi sviluppata una procedura per ricostruire le facce dell'oggetto utilizzando le sole informazioni di prossimità contenute in suddetto grafo, teoricamente dimostrata da Dumitriu [13]. Quest'ultimo approccio, oltre ad essere molto promettente per quanto riguarda la ricostruzione, ha permesso di ridurre in maniera considerevole i tempi di esecuzione di altre elaborazioni che necessitano del vicinato.

I risultati ottenuti dall'intero sistema sono stati impiegati per svolgere un compito di alto livello. Dopo aver suddiviso ciascun oggetto in parti, si sono studiate le possibili prese per ognuna di esse. Sono stati dunque effettuati alcuni esperimenti di manipolazione degli stessi, prima in ambiente simulato e poi con il sistema reale.

In molte aree c'è ancora margine di miglioramento, soprattutto per quanto riguarda lo studio dei criteri di ottimalità di Delaunay e la ricostruzione della superficie in base al grafo di prossimità. In futuro sarebbe altresì interessante una ulteriore integrazione dei singoli sottosistemi, in modo tale da permettere la manipolazione di oggetti in precedenza ignoti senza alcun intervento da parte dell'uomo.

Bibliografia

- [1] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry*, 22:481–504, 1998.
- [2] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the sixteenth annual symposium on Computational geometry*, SCG '00, pages 213–222, New York, NY, USA, 2000. ACM.
- [3] N. Amenta, S. Choi, and R. K. Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications*, 19:127–153, 2000.
- [4] N. Amenta, S. Choi, and R. K. Kolluri. The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, SMA '01, pages 249–266, New York, NY, USA, 2001. ACM.
- [5] R. Beltrami. Acquisizione di scansioni per la ricostruzione tridimensionale e la manipolazione di oggetti. Progetto del Corso di Robotica, Università degli Studi di Parma, 2011.
- [6] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, September 1975.
- [7] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5:349–359, 1999.

-
- [8] J.-D. Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Trans. Graph.*, 3:266–286, October 1984.
- [9] J. M. Boyer and W. J. Myrvold. On the cutting edge: Simplified $o(n)$ planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8:2004, 2004.
- [10] P. J. Brown and C. T. Faigle. A robust efficient algorithm for point location in triangulations. Technical Report UCAM-CL-TR-728, University of Cambridge, Computer Laboratory, Feb. 1997.
- [11] F. Cazals and J. Giesen. Delaunay Triangulation Based Surface Reconstruction: Ideas and Algorithms. Technical Report RR-5393, INRIA, November 2004.
- [12] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.
- [13] D. Dumitriu, S. Funke, M. Kutz, and N. Milosavljevic. How much geometry it takes to reconstruct a 2-manifold in \mathbb{R}^3 . *ACM Journal of Experimental Algorithmics*, 14, 2009.
- [14] A. Ferrazzano. Realizzazione di un software per la scansione di oggetti 3D. Progetto del Corso di Robotica, Università degli Studi di Parma, 2011.
- [15] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.
- [16] S. Foix, G. Alenyà, J. Andrade-Cetto, and C. Torras. Object modeling using a ToF camera under an uncertainty reduction approach. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1306–1312, 2010.

-
- [17] C. Guarino Lo Bianco. *Cinematica dei manipolatori*. Pitagora, 2004.
- [18] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, 4:74–123, April 1985.
- [19] M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii. Topology matching for fully automatic similarity estimation of 3D shapes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 203–212, New York, NY, USA, 2001. ACM Press.
- [20] B. K. P. Horn, H. Hilden, and S. Negahdaripour. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society America*, 5(7):1127–1135, 1988.
- [21] U. Klank, D. Pangercic, R. B. Rusu, and M. Beetz. Real-time CAD model matching for mobile manipulation and grasping. In *The 9th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Paris, France, 12/2009 2009.
- [22] K. Klasing, D. Wollherr, and M. Buss. Realtime segmentation of range data using continuous nearest neighbors. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation, ICRA'09*, pages 2011–2016, Piscataway, NJ, USA, 2009. IEEE Press.
- [23] D. Klimentjew, M. Arli, and J. Zhang. 3D scene reconstruction based on a moving 2D laser range finder for service-robots. In *Proceedings of the 2009 international conference on Robotics and biomimetics, ROBIO'09*, pages 1129–1134, Piscataway, NJ, USA, 2009. IEEE Press.
- [24] D. Lischinski. *Incremental Delaunay triangulation*, pages 47–59. Academic Press Professional, Inc., San Diego, CA, USA, 1994.
- [25] A. Maldonado, U. Klank, and M. Beetz. Robotic grasping of unmodeled objects using time-of-flight range data and finger torque information.

- In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 18-22 2010.
- [26] Z.-C. Marton, D. Pangercic, N. Blodow, J. Kleinhellefort, and M. Beetz. General 3D Modelling of Novel Objects from a Single View. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 18-22 2010.
- [27] P. Neckar and M. Adamek. Software and hardware specification for area segmentation with laser scanner SICK LMS 400, 2011.
- [28] S. Novikov and A. Fomenko. *Basic elements of differential geometry and topology*. Mathematics and its applications (Kluwer Academic Publishers).: Soviet series. Kluwer Academic Publishers, 1990.
- [29] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, New York, NY, USA, 2nd edition, 1998.
- [30] M. Pattera. Sviluppo driver ethernet e caratterizzazione statistica del sensore laser Sick LMS 400. Progetto del Corso di Robotica, Università degli Studi di Parma, 2010.
- [31] R. B. Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009.
- [32] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.
- [33] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz. Towards 3D Point Cloud Based Object Maps for Household Environments. *Robotics and Autonomous Systems Journal (Special Issue on Semantic Knowledge)*, 2008.

-
- [34] Schunk GmbH. *Servo electric 2-finger parallel gripper type PG 70: assembly and operating manual.*
- [35] J. R. Shewchuk. Robust adaptive floating-point geometric predicates. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, pages 141–150. Association for Computing Machinery, May 1996.
- [36] Sick AG. *LMS 400 laser measurement system: pole position for robotics and material handling.*
- [37] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. Point feature extraction on 3d range scans taking into account object boundaries. In *International Conference on Robotics and Automation*, May 2011.
- [38] J. Strom, A. Richardson, and E. Olson. Graph-based segmentation for colored 3D laser point clouds. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2010.
- [39] M. Tarasconi. Sviluppo di un driver in ambiente orchestra per il controllo di un gripper robotico. Tesi di Laurea in Ingegneria Informatica, Università degli Studi di Parma, 2010.
- [40] L. Torabi and K. Gupta. Integrated view and path planning for an autonomous six-DOF eye-in-hand object modeling system. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4516–4521, October 18-22 2010.
- [41] D. Valeriani. Sviluppo di una libreria software per la programmazione del robot manipolatore Comau Smart SiX. Tesi di Laurea in Ingegneria Informatica, Università degli Studi di Parma, 2010.