Università degli Studi di Padova

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Tesi di Laurea Magistrale in Ingegneria Informatica

Sistema software per la stima della posa di oggetti industriali da dati tridimensionali

Laureando

Nicolò Frezzato

Relatore

Prof. Emanuele Menegatti

Correlatore

Stefano Tonello

15 Aprile 2014

Anno Accademico 2013/2014

Dedica...

Citazione *Autore*

Sommario

Lo scopo del presente lavoro è sviluppare un algoritmo per individuare la posa di oggetti utilizzando point cloud tridimensionali ottenute con un sistema di triangolazione laser. La posa 6DOF viene individuata utilizzando un algoritmo per il riconoscimento di oggetti basato su PCL (Point Cloud Library). Tale algoritmo è sviluppato per essere utilizzato in un sistema industriale per bin picking.

La soluzione proposta per il riconoscimento di oggetti si basa su *correspondences grouping*, in particolare, utilizzando una versione della GHT (Generalized Hough Transform) estesa allo spazio tridimensionale. L'algoritmo proposto è stato confrontato con una soluzione che utilizza, in combinazione, clustering e registrazione di point cloud, attualmente utilizzata in campo industriale.

Dai test effettuati la soluzione proposta è una valida alternativa all'approccio correntemente utilizzato.

Indice

Introduzione

Negli ultimi anni sono stati fatti grandi progressi nel campo della *computer vision* applicata alla robotica industriale. Lo studio di sistemi di visione per robot in campo industriale è stato, infatti, incentivato da diversi fattori come, ad esempio, l'implementazione di algoritmi per l'elaborazione di immagini e dati tridimensionali sempre più efficaci e ottimizzati, la diminuzione del costo di hardware dotato di grande potenza di calcolo e sensori di visione sempre più precisi ed economici.

Sistemi industriali robotizzati sprovvisti di un sistema di visione non hanno la percezione dell'ambiente, hanno quindi la necessità di un contesto strutturato e ben definito per operare correttamente. Tale vincolo influenza l'intera catena di produzione in quanto è necessario studiare soluzioni ad hoc per manipolare i prodotti tra una fase della produzione e la successiva. Questo tipo di sistemi non è vantaggioso ad esempio per situazioni in cui si producono quantitativi limitati di diverse tipologie. Spesso, infatti, durante alcune fasi della produzione vari tipi di prodotti vengono depositati alla rinfusa in contenitori.

Un sistema dotato di percezione è configurabile per la lavorazione di diverse tipologie di oggetti senza la necessità di sviluppare sistemi di caricamento adattati ad ogni prodotto.

Nel presente elaborato si focalizza l'attenzione sull'individuazione della posa di oggetti disposti alla rifusa in un contenitore in modo da consentirne la manipolazione da parte di un robot. La stima della posa avviene attraverso algoritmi di visione per il riconoscimento di oggetti in scansioni tridimensionali ottenute con un sistema di triangolazione laser. Questo task fa parte del problema del *bin picking*.

A tale scopo è stato sviluppato un algoritmo per il riconoscimento di oggetti basato su *correspondence grouping* sfruttando PCL (Point Cloud Library), un framework open-source per la manipolazione e l'elaborazione di strutture dati tridimensionali, su cui sono basate un gran numero di applicazioni utilizzate nel campo della robotica.

L'algoritmo proposto è stato pensato e sviluppato cercando un approccio

al riconoscimento alternativo rispetto a quello esistente (basato su *clustering* e *registrazione*), utilizzato in un software dedicato al *bin picking* correntemente sfruttato in sistemi di produzione industriali. Proprio l'introduzione di un secondo algoritmo ha motivato l'implementazione di un framework per testare le diverse configurazioni del software e confrontare i risultati ottenuti con i due diversi algoritmi di riconoscimento.

La ricerca di un'alternativa è di fondamentale importanza per estendere le capacità del sistema esistente, per incrementare ulteriormente le già ottime performance in termini di velocità di esecuzione o la gamma di oggetti riconoscibili dal sistema.

Capitolo 1

Sistema per bin picking

6DOF è l'acronimo di **sei gradi di libertà** (*Six Degrees Of Freedom*) e si riferisce al movimento nello spazio tridimensionale, ovvero l'abilità di traslare liberamente avanti/indietro, in alto/in basso, sinistra/destra (traslare in tre assi perpendicolari); i movimenti di traslazione possono essere combinati a rotazioni attorno ai tre assi (imbardata, beccheggio, rollio).

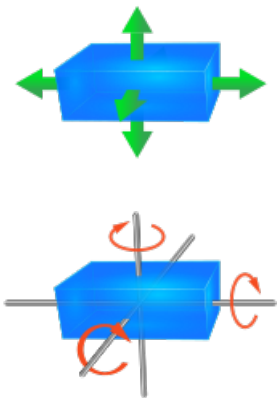

Figura 1.1: Rappresentazione dei tre possibili movimenti di traslazione e i tre movimenti di rotazione

Il problema del *6DOF pose extimation* consiste nel trovare e determinare la posa di un oggetto/modello in una scena tridimensionale. Con il termine posa si intende l'individuazione della matrice di rototraslazione che permette di allineare un modello all'oggetto individuato nella scena. La soluzione a questo problema è di fondamentale importanza per il *bin picking*.

1.1 Cos'è il bin picking

Il problema del *bin picking* consiste nell'individuare e prelevare, degli oggetti posti alla rinfusa all'interno di un contenitore (*bin*) utilizzando un robot manipolatore. La soluzione generale al problema è complessa e consiste nell'essere in grado di individuare e manipolare oggetti senza che il sistema ne conosca la forma a priori. Per questo motivo si semplifica il problema operando riconoscimento e prelievo di oggetti di cui si conosce la forma. I campi di applicazione sono molteplici, specialmente in ambito industriale. Ad esempio, nei sistemi di packaging, nell'assemblaggio o in particolari fasi della lavorazione i componenti escono da fasi di lavorazione precedenti in contenitori disposti alla rinfusa.

Il sistema per *bin picking* deve eseguire le seguenti operazioni:

1. Acquisire dati, cioè ottenere una scansione tridimensionale del contenitore.
2. Identificare oggetti, cioè rilevare la posa esatta degli oggetti nel contenitore.
3. Prelevare e estrarre un oggetto dal contenitore.

Si possono identificare quindi tre componenti fondamentali del sistema: un sistema di visione, un sistema che si occupa dell'identificazione e localizzazione degli oggetti, un robot manipolatore che si occupa del prelievo dell'oggetto dal contenitore.

1.2 Componenti del sistema

Il sistema per bin picking utilizzato è composto principalmente da:

- **Sistema di visione**: è il sistema di acquisizione dati, composto da telecamere o altri sensori e dall'illuminazione (laser, luce strutturata).
- **Supporto per sistema di visione**: può essere fisso (telecamere 2D) o mobile (per scansioni 3D) ad esempio montato su un robot.
- **Bin o contenitore**: contiene gli oggetti o i prodotti da prelevare.
- **Braccio robotico o manipolatore**: è dotato di una uno strumento di presa, intercambiabile a seconda dei prodotti da prelevare (6/8 g.d.l).
- **PLC**: si occupa di mettere in comunicazione e sincronizzare i vari componenti del sistema.

- **Computer**: utilizzato per l'esecuzione degli algoritmi di visione, computazionalmente troppo complessi per essere eseguiti dal PLC.
- **Software**: implementazione degli algoritmi di visione e comunicazione con il PLC.

Si analizzano ora i componenti più rilevanti nell'ambito degli algoritmi di riconoscimento di oggetti.

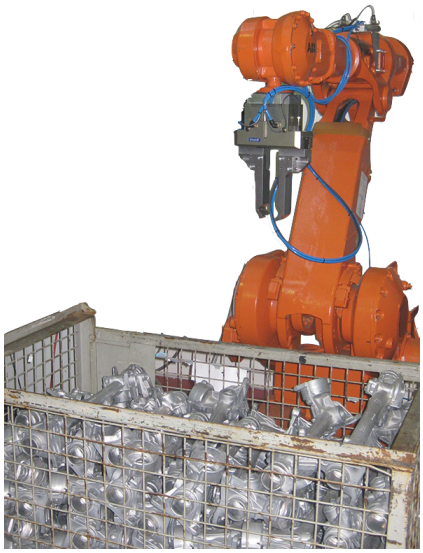

Figura 1.2: Braccio robotico e *bin* contenete i prodotti da individuare e raccogliere.

1.3 Sistema di visione

Il sistema di visione utilizzato dal sistema è montato su un supporto mobile al di sopra del cesto che contiene i prodotti da prelevare. Tale sistema permette di ottenere una scansione tridimensionale mediante triangolazione laser. Le tre componenti principali sono:

- una telecamera ad alta risoluzione (28 fps - risoluzione 1600x1200 px) ;
- due proiettori laser ($5mW$ - λ 640-670nm).

Sensore e proiettori sono fissati su un supporto: la telecamera al centro e i laser simmetricamente rispetto a quest'ultima.

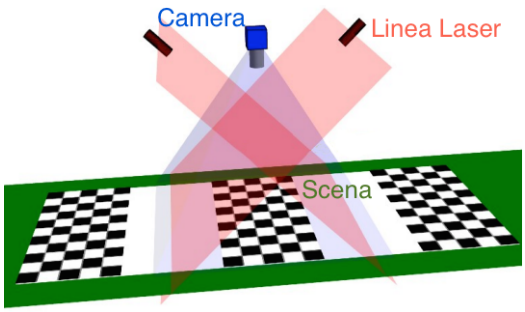

Figura 1.3: Sistema di visione.

La tecnica di triangolazione calcola l'altezza di ogni pixel basandosi sulla sulla deviazione della luce causata dalla presenza di un oggetto. Per ogni frame acquisito dalla telecamera viene calcolata l'altezza di ogni pixel sulla linea, in questo modo ogni frame fornisce una linea di altezze.

Utilizzando questo sistema si ottengono delle nuvole di punti (*point cloud*) che vengono utilizzate come input per il software di riconoscimento che si occupa di individuare la posa degli oggetti nel *bin*.

1.4 Sistema Software

1.4.1 Smart Pick 3D

Smart Pick 3D è il software sviluppato da IT+Robotics per l'identificazione, tramite visione, della posizione e dell'orientamento dei prodotti all'interno della linea di produzione. Attraverso i dati acquisiti dal sistema di visione è possibile controllare i movimenti del robot, permettendo il prelievo dei pezzi da lavorare da un cassone o da un nastro trasportatore disposti in maniera totalmente casuale.

Nello sviluppo di Smart Pick 3D sono state utilizzate molte librerie software e API per ottenere algoritmi di visione efficienti e all'avanguardia. Per quanto riguarda il presente lavoro, i componenti più rilevanti sono:

- **PCL**: Point Cloud Library è un framework open-source che include numerosi algoritmi per l'elaborazione di immagini 2D/3D e *point cloud* n-dimensionali.
- **OpenMP**: API multipiattaforma per la creazione di applicazioni parallele su sistemi a memoria condivisa.

Smart Pick 3D è sviluppato e rilasciato per Microsoft Windows.

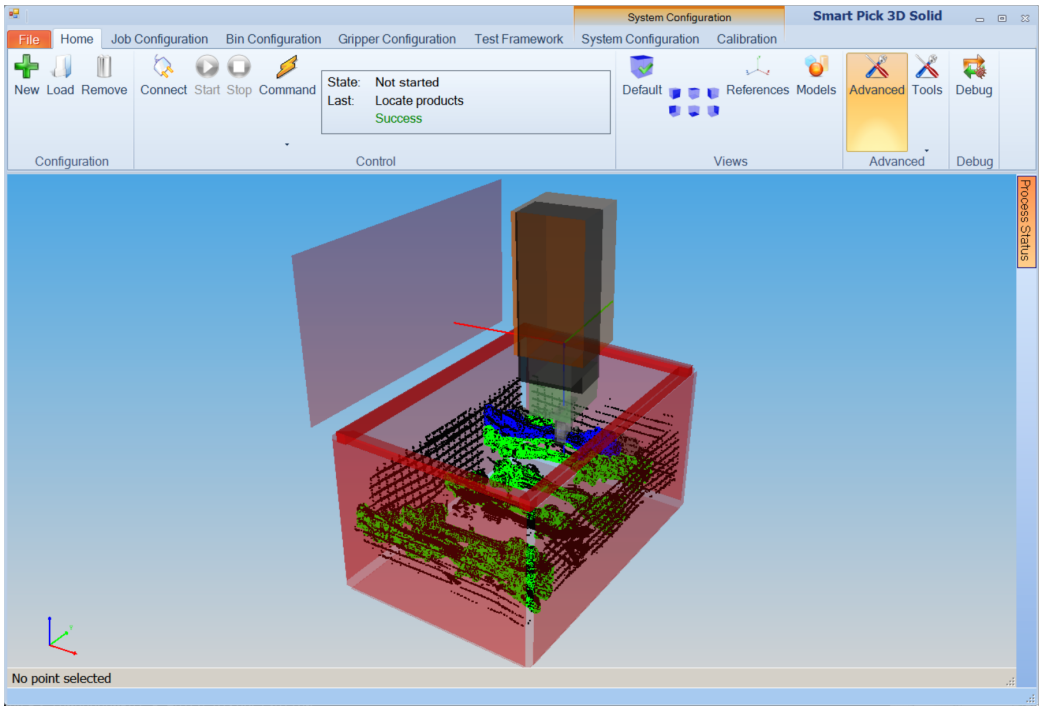

Figura 1.4: Screenshot di Smart Pick 3D. É visibile il risultato di un ciclo di esecuzione: in verde, gli oggetti individuati all'interno del contenitore, in rosso. Il componente in blu è stato scelto per il prelievo.

1.4.1.1 Esecuzione offline e *dump*

Durante l'esecuzione del software in produzione, per ogni ciclo viene salvato un *dump*. Ognuno di essi contiene in sintesi:

- Configurazione corrente (algoritmi da utilizzare, parametri, prodotto da prelevare, ecc.)
- Stato corrente
- Point cloud ottenuta dal sistema di visione
- Posizione del prodotto prelevato nel ciclo precedente

Tali *dump* sono eseguibili in un secondo momento in modalità *offline*, ovvero senza necessità di essere collegati al sistema di visone né al robot. Hanno principalmente due scopi:

- risolvere eventuali problemi che si sono verificati in produzione.

- consentire il debug di nuove funzionalità o algoritmi senza essere collegati al sistema.

1.4.2 Point Cloud Library

PCL[1] è un progetto open-source, cross-platform, rilasciato con licenza BSD. Compilabile e distribuibile su Linux, Mac OS X, Windows, Android e iOS.

Il framework contiene numerosi algoritmi allo stato dell'arte che includono filtraggio, calcolo di *feature*, registrazione, segmentazione e molti altri. Questi algoritmi possono essere usati ad esempio per filtrare dati rumorosi, unire nuvole di punti, segmentare parti di una scena, estrarre keypoint e calcolare descrittori per il riconoscimento di oggetti. Tali funzionalità insieme a molte altre sono alla base di qualsiasi algoritmo di visione.

Per rendere più semplice lo sviluppo utilizzando questo vasto framework, PCL è diviso in una serie di librerie più piccole che possono essere compilate separatamente. La scelta si rivela vincente, per esempio, per distribuire PCL su piattaforme con capacità computazionale ridotta. Il grafico delle librerie di PCL [2] è riportato in figura 1.5.

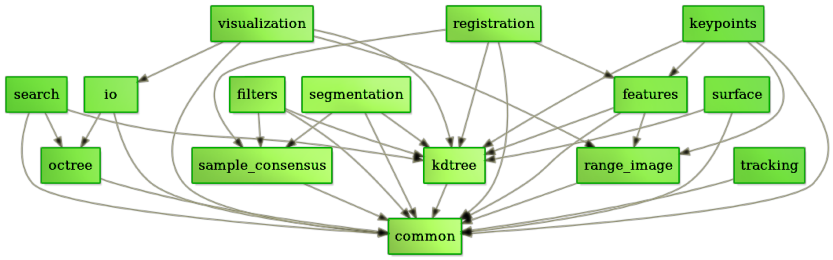

Figura 1.5: Grafico dei moduli di PCL alla versione 1.6.

I moduli principali e utilizzati per lo sviluppo sono:

- **common** contiene le strutture di dati comuni e i metodi usati dalla maggior parte delle librerie PCL. La struttura dei dati di base include la classe `Point Cloud` e diversi tipi di punti che sono utilizzati per rappresentare punti semplici, le normali delle superfici, i valori di colore RGB e i descrittori delle *features*. Contiene inoltre numerose funzioni per calcolare distanze/norme, medie e covarianze, conversioni angolari e trasformazioni geometriche.

- **kdtree** fornisce la struttura dati `kd-tree` (albero a k dimensioni), che usa l'implementazione `FLANN`; con questa struttura dati è possibile partizionare lo spazio che memorizza un *set* di punti a k-dimensioni in un

albero che permette efficienti *range search* e *nearest neighbor search*. Le *nearest neighbor searches* sono operazioni fondamentali quando si lavora con *point cloud* e possono essere utilizzate per trovare corrispondenze tra gruppi di punti, descrittori di *features*, o per definire una *neighborhood* locale attorno a un punto o ad un insieme di punti.

- **filters** contiene algoritmi di rimozione di *outlier* e rumore per il filtraggio di *point cloud* 3D. Contiene inoltre filtri generici usati per estrarre sottoinsiemi di *point cloud*, per escluderne una parte o effettuare il *downsampling*. Viene utilizzato ad esempio per il filtraggio `voxel`.

- **features** utilizzato per la stima delle *features 3D* sulle *point cloud*. Queste descrivono pattern geometrici basati sulle informazioni disponibili intorno ad un punto, dove la *neighborhood* del punto può essere definita o come l'insieme dei punti contenuti in una sfera di centro il punto considerato e raggio arbitrario o in termini di *k-neighborhood* (i k punti più vicini al punto considerato).

- **keypoints** contiene l'implementazione di vari algoritmi per l'individuazione di punti di interesse nelle *point cloud*, punti cioè che siano distintivi della cloud considerata e che, una volta che su di essi siano stati calcolati dei descrittori, siano in grado di formare una rappresentazione compatta (il numero di *keypoints* è minore del numero di punti della *point cloud*) e tuttavia completa e distintiva dei dati originali.

- **registration** implementa una serie di algoritmi di la registrazione di *point cloud*, tra cui algoritmi per il *correspondence finding*, *correspondence rejection*, *transformation estimation* e *ICP (Iterative Closest Point)*.

- **segmentation** contiene algoritmi per la segmentazione di point cloud in cluster. Questo è utilizzato in fase preliminare nell'approccio basato su clusterizzazione della scena.

- **sample_consensus** contiene metodi basati su *SAmple Consensus* come RANSAC e modelli (linee, piani, cilindri, sfere etc.) può essere utilizzato per il `plane detection` o per individuare oggetti con una struttura geometrica comune.

- **io** contiene classi e funzioni per la lettura e la scrittura di *point cloud* da/su disco.

- **visualization** viene utilizzato per visualizzare graficamente i risultati degli algoritmi operanti su *point cloud* tridimensionali.

1.4.2.1 Versione 1.7

Nel Luglio 2013 è stata rilasciata ufficialmente la versione 1.7 del progetto seguita, pochi mesi dopo, dalla versione 1.7.1 che introduce alcune migliorie e bugfix.

Con il rilascio ufficiale dell'ultima versione è stato aggiunto il modulo **recognition** che implementa alcuni algoritmi per il riconoscimento di oggetti basati su *correspondece grouping*. L'introduzione di tale modulo ha motivato lo sviluppo di un nuovo sistema di riconoscimento da confrontare a quello attualmente esistente e funzionate.

1.5 OpenMP API

OpenMP è una *Application Program Interface* (API)[3], definita congiuntamente da un insieme dei maggiori produttori hardware e software. OpenMP fornisce un modello portabile e scalabile per lo sviluppo di applicazioni parallele con memoria condivisa. Sono supportati i linguaggi C/C++ e Fortran su una vasta gamma di architetture.

I componenti di OpenMP possono essere sintetizzati in:

- Direttive del compilatore
- Runtime Library
- Variabili d'ambiente

I programmi che usano OpenMP sono parallelizzati attraverso l'uso di *thread*. Un *thread* è la più piccola unità di un processo che può essere schedulata dal sistema operativo.

Inoltre OpenMP è un modello di programmazione esplicito, che offre al programmatore il pieno controllo della parallelizzazione.

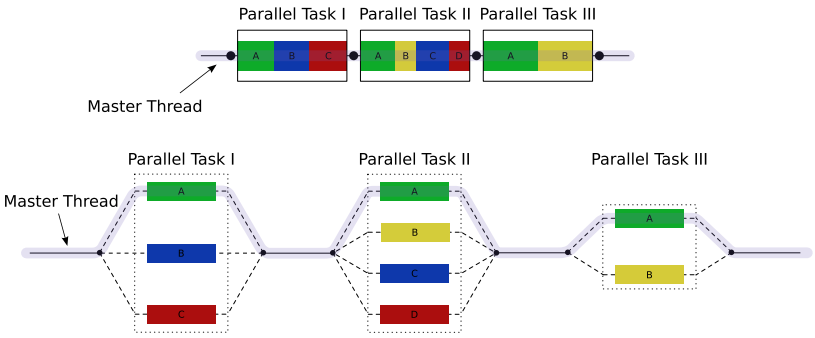

Figura 1.6: Esempio di *multithreading* il *master thread* esegue una *fork* e crea altri *thread* che eseguono blocchi di codice in parallelo.

Capitolo 2

Il riconoscimento

Nel campo della *computer vision* il riconoscimento di oggetti (*object recognition*) consiste nell'individuare in una scena, bidimensionale o tridimensionale, l'oggetto ricercato. L'uomo è in grado di riconoscere in pochi istanti un grande numero di oggetti, anche variando il punto di vista o la scala; non incontra particolari difficoltà nemmeno se gli oggetti sono parzialmente nascosti alla vista (occlusioni).

Per una macchina questo compito è un problema complesso e computazionalmente molto dispendioso, specialmente se la scena in cui si vogliono riconoscere gli oggetti è tridimensionale.

Al giorno d'oggi, in *computer vision*, la ricerca di un algoritmo di riconoscimento robusto, accurato e veloce è di fondamentale importanza per una moltitudine di applicazioni come la realtà aumentata, o come nel caso trattato, per il rilevamento della posa di oggetti per la manipolazione robotica.

Con il rilascio ufficiale delle *Point Cloud Library* 1.7, avvenuto nel luglio 2013, sono stati introdotti dei metodi specifici per l'*object recognition*, prima non supportati. In questo elaborato è stato sviluppato un algoritmo di riconoscimento di oggetti industriali per la stima della posa di quest'ultimi. Le point cloud dei modelli sono ottenute partendo da disegni CAD. Il sistema opera il riconoscimento elaborando le scansioni tridimensionali ottenute con il sistema di triangolazione laser. L'algoritmo sviluppato, basato su **Hough3DGrouping** implementato in PCL 1.7, è stato confrontato con un algoritmo di riconoscimento basato su registrazione [4] e clustering, sviluppato antecedentemente all'introduzione del modulo **recognition**.

Un esempio di *object recognition* è riportato in Figura 2.1: in questo caso lo scopo è riconoscere nella scena più istanze di contenitori diversi per forma e dimensione.

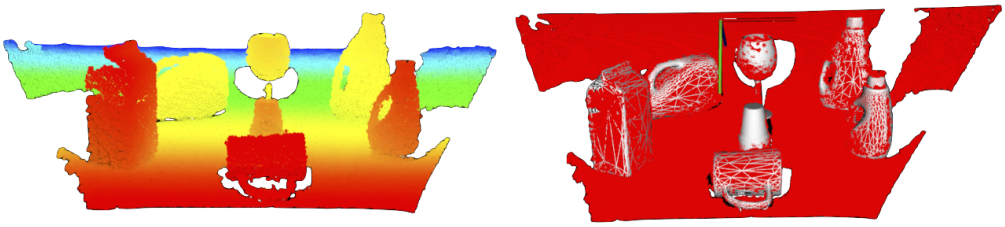

Figura 2.1: Esempio di riconoscimento: a sinistra scansione trimensionale della scena; a destra, in bianco, modelli CAD degli oggetti riconosciuti e allineati sulla scena, in rosso.

Nel presente capitolo viene definito il problema del riconoscimento, analizzata la teoria alla base degli algoritmi e proposti due approcci di soluzione generale allo stesso.

2.1 Definizione del problema

L'obiettivo finale, nel caso in esame, è quello di trovare le coordinate del punto di presa degli oggetti presenti all'interno della scena scansionata dal sistema di triangolazione laser. Posto che il punto di presa può variare a seconda della natura degli oggetti cercati, si semplifica questa parte del problema cercando, nella fase di riconoscimento, una matrice di roto-traslazione che sovrappone un modello (partendo da una posizione nota) ad una istanza dell'oggetto trovata nella scena: l'allineamento tra scena e modello deve essere preciso per consentire la presa del braccio robotico.

2.1.1 Input

I dati di partenza per effettuare un riconoscimento sono essenzialmente il modello e la scena:

- ***modelview*:** partendo dal modello CAD dell'oggetto da riconoscere si ottengono una serie di point cloud, una per ogni faccia o vista del modello (Figura 2.2). Questa scelta facilita molto la fase di riconoscimento, in cui ogni vista del modello viene considerata come un modello da riconoscere, in quanto riduce la complessità computazionale riducendo notevolmente il numero di punti del modello. Inoltre, tale scelta è motivata dal sistema utilizzato per ottenere la scansione del *bin*: la scansione laser è ottenuta dell'alto e difficilmente mostra più facce di

una istanza di un oggetto. Il fatto di usare le facce del modello riduce l'errore di rotazione e di traslazione nella ricerca della posa[5].

- **scena:** point cloud ottenuta dal sistema di scansione, contiene gli oggetti disposti in posizione casuale e il cesto che li contiene (Figura 2.3). Utilizzando un filtro voxel si riduce la risoluzione della scena per ridurre la complessità computazionale.

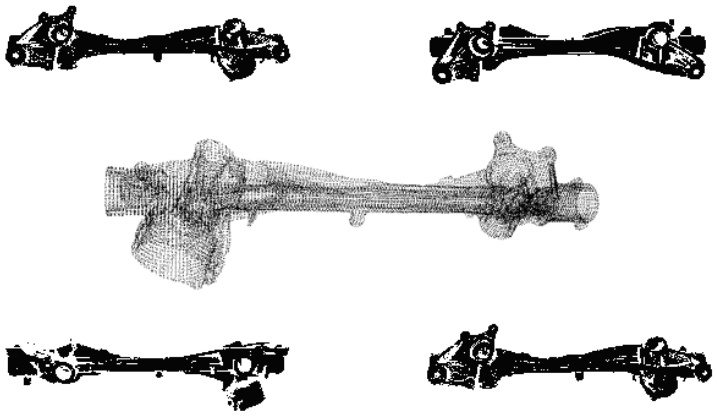

Figura 2.2: Modelli: quattro viste del modello completo visibile al centro. Le point cloud del modello e delle viste sono ottenute da CAD.

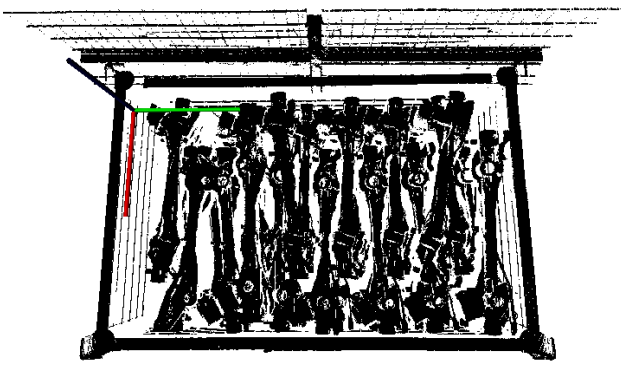

Figura 2.3: Scansione della scena: il *bin* contente gli oggetti da individuare

2.1.2 Output

La soluzione al problema è costituita, quindi, da un set di matrici di rototraslazione utilizzate per allineare la point cloud del modello alla parte visibile nella scena ottenuta con il sistema di visione.

Una trasformazione rigida è rappresentata da una matrice 4x4:

$$\begin{bmatrix} r11 & r12 & r13 & t1 \\ r21 & r22 & r23 & t2 \\ r31 & r32 & r33 & t3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La matrice 4x4 in coordinate omogenee, da cui si ottiene la trasformazione rigida, è costituita da: una sottomatrice 3x3 ($r_{i,j}$) che rappresenta la rotazione nello spazio 3D e dal vettore ($\begin{bmatrix} t1 & t2 & t3 \end{bmatrix}$) che rappresenta la traslazione .

2.2 Riconoscimento basato su *registrazione*

Il metodo per allineare due o più nuvole di punti tridimensionali è chiamato **registrazione**. Il processo consiste brevemente nell'identificare i punti corrispondenti tra più viste di una scena o di un oggetto, trovare una trasformazione che minimizza la distanza tra tali punti in modo da allineare perfettamente le *point cloud* ricostruendo la scena. Iterativamente si cerca di raggiungere un allineamento perfetto.

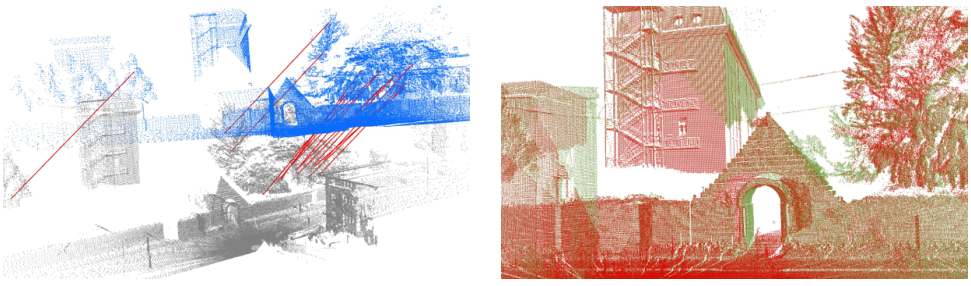

Figura 2.4: Esempio di registrazione tra due *point cloud*: identificazione dei punti corrispondenti (sinistra), risultato della registrazione minimizzando la distanza tra i punti (destra).

Il problema del riconoscimento può essere considerato un caso particolare della registrazione, in cui si cerca di allineare un modello alla scena. Tuttavia, la presenza di più istanze di un determinato oggetto nella stessa scena mette in crisi il metodo di registrazione in quanto ad un punto nel modello ne possono corrispondere uno o più nella scena, entrambi validi. Ma lo scopo di questa tecnica è trovare la migliore trasformazione che allinea due *point cloud* quindi, in questo caso, la tecnica di registrazione ha un'alta probabilità di fallire.

Per questo motivo si adotta la tecnica di *clustering*[6] per separare gli oggetti da riconoscere nella scena prima di utilizzare la tecnica di registrazione. Successivamente è possibile registrare il modello o una vista di quest'ultimo con ogni cluster.

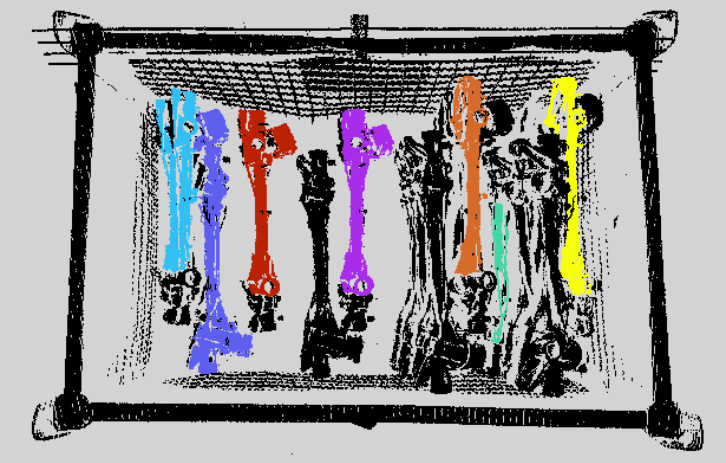

Figura 2.5: Esempio di clustering: ogni colore identifica un cluster, su ogni cluster si utilizza l'algoritmo di registrazione.

In breve, la procedura di registrazione di due point cloud consiste nei seguenti passi:

- per ogni nuvola di punti si identificano i **punti di interesse** o ***keypoint*** che meglio rappresentano la scena in entrambi i data set
- per ogni *keypoint* si calcola un **descrittore**
- dall'insieme dei descrittori, combinati alla loro posizione nello spazio, si identificano una serie di corrispondenze basate sulla somiglianza tra i descrittori e la loro posizione

- si scartano le corrispondenze non valide che contribuirebbero in maniera negativa alla registrazione
- dalle corrispondenze rimanenti si stima una trasformazione rigida

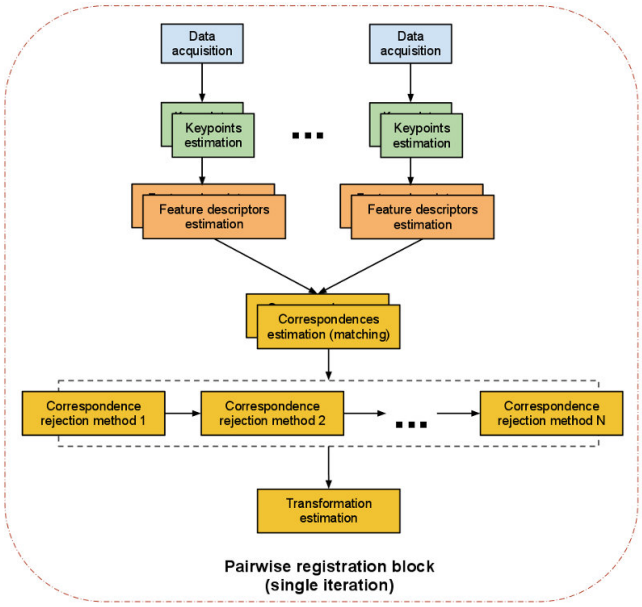

Figura 2.6: Point Cloud Library *Registration API* (immagine tratta dalla documentazione ufficiale di PCL [4]).

Iterando questi passi fino a raggiungere determinati criteri di convergenza è possibile raggiungere l'allineamento. L'implementazione dell'algoritmo ICP (Iterative Closest Point) è un esempio di adattamento della pipeline appena descritta. Solitamente ICP viene utilizzato nella fase finale della registrazione per perfezionare l'allineamento.

2.3 Riconoscimento basato su *correspondence grouping*

Supponiamo di avere un modello di un oggetto e una scena che contiene lo stesso più volte. A differenza della tecnica di registrazione che richiede il clustering della scena, l'approccio al riconoscimento basato sul *correspondence grouping* è in grado di gestire e riconoscere multiple istanze del modello cercato nella stessa scena.

Si analizza ora il flusso di lavoro di un algoritmo di riconoscimento basato su *correspondence grouping* mediante Hough Voting. Le fasi principali del riconoscimento si possono riassumere nello schema in figura 2.7

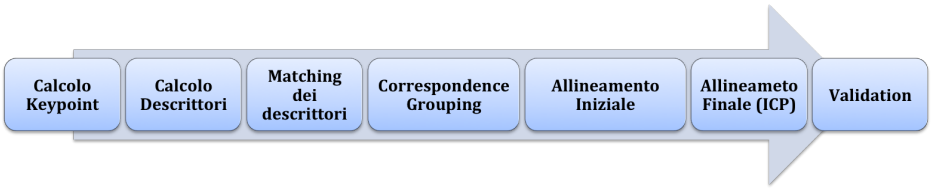

Figura 2.7: Object Recogintion pipeline.

Nello specifico:

1. **Keypoint:** il primo passo consiste nel calcolare dei *keypoint*, o punti chiave, per il modello e per la scena.

2. **Descrittori:** per ogni *keypoint* si calcola un descrittore (*feature*), ovvero una caratteristica unica che lo distingue dagli altri keypoint.

3. **Corrispondenze:** utilizzando le *feature* estratte al punto precedente si calcolano le corrispondenze modello-scena.

4. **Correspondence Grouping:** le corrispondenze trovate vengono clusterizzate, ogni cluster rappresenta una ipotesi di posa.

5. **Allineamento iniziale:** ogni gruppo di corrispondenze permette di calcolare una trasformazione del modello sulla scena, ottenendo così una serie di istanze che sono una prima ipotesi della posa del modello nella scena.

6. **Allineamento finale:** utilizzando ICP si cerca di migliorare l'allineamento iniziale del modello.

7. **Validazione**: come ultimo passaggio vengono rimossi falsi positivi o duplicati utilizzando una funzione di validazione.

2.3.1 Keypoint

I *keypoint* (o punti di interesse) sono punti in un'immagine o in una nuvola di punti che sono stabili, caratteristici o distintivi e che possono essere identificati utilizzando un criterio ben definito. Tipicamente il numero di *keypoint* è molto più piccolo del numero totale di punti nella *point cloud* e, quando

usati in combinazione con i descrittori, si ottiene una descrizione compatta dei dati originali.

Esistono diversi tipi di *keypoint*, ci si limita a riportare una breve introduzione di quelli testati e scelti durante lo studio.

2.3.1.1 SIFT Keypoint

Un SIFT (Scale-Invariant Feature Transform) keypoint è un adattamento al 3D del metodo studiato da David Lowe [7] per l'estrazione di keypoint. Un SIFT keypoint è una regione circolare che possiede un'orientazione, questo è descritto da un frame di quattro parametri: le coordinate x e y del centro, il raggio della regione che comprende (i.e. la sua "scale") e la sua orientazione espressa come un angolo in radianti. I parametri che influenzano il rilevamento di questi keypoint sono diversi: anzitutto essi vengono cercati su più "scale", o cosiddetti "livelli di smoothing", mediante la costruzione di un Gaussian space scale, cioè una collezione di immagini ottenute applicando iterativamente filtri gaussiani all'immagine di partenza. Questi filtri apportano appunto un effetto di smoothing e comportano pertanto una graduale perdita di risoluzione.

2.3.1.2 Uniform Keypoints

Lo *uniform sampling* funziona in maniera analoga al filtro *voxel* l'unica differenza che si può osservare è data dal fatto che il *keypoint* risultante è un punto della *point cloud* di partenza, vincolo che non è necessariamente rispettato dal *voxel filter* in quanto, in questo caso, per ogni voxel viene scelto il centroide dei punti al suo interno. La procedura, dunque, è la seguente:

1. viene costruita una griglia di cubi di dimensione fissata come parametro;

2. viene computato il *voxel point* per ciascun cubo;

3. viene scelto come *keypoint* per ciascun cubo il punto della *point cloud* più vicino al *voxel point*.

Tipicamente le *feature* di oggetti complessi e articolati non funzionano: non vengono cioè riconosciute nella fase di ricerca delle corrispondenze se vi sono dei cambiamenti nella geometria interna tra due point cloud processate. Tuttavia, l'algoritmo **SIFT** porta all'uso di un numero piuttosto elevato di *feature*, cosa che riduce il contributo di errori causati da queste variazioni locali durante la fase di *feature matching* per il calcolo delle corrispondenze.

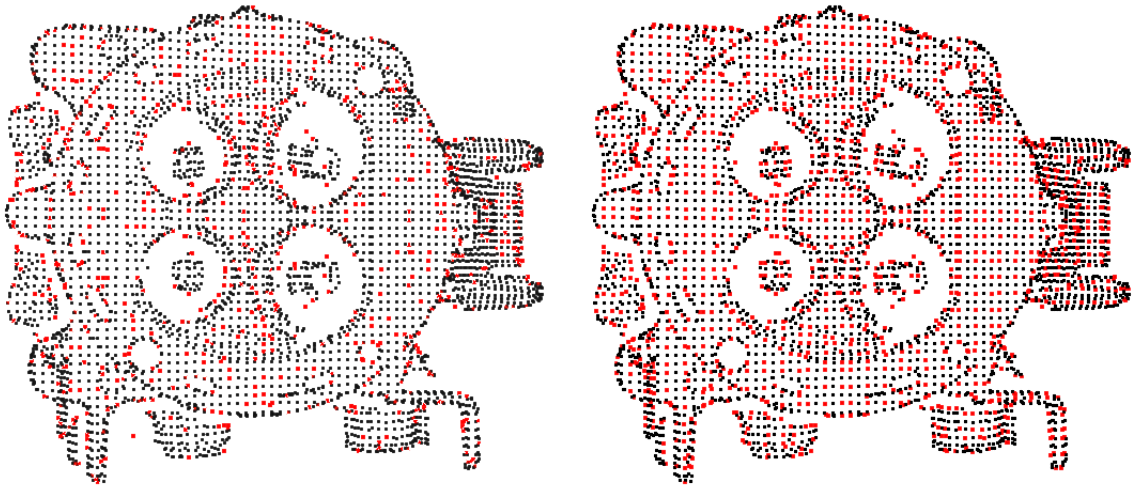

Figura 2.8: Confronto tra *keypoints* `SIFT` (sinistra) e `uniform` (destra). Si noti come nel secondo caso sia ben visibile la struttura a griglia precedentemente discussa.

Nel presente lavoro è stato scelto di utilizzare gli *uniform keypoints* in quanto, per la natura degli oggetti trattati, la scelta di *keypoint* con un particolare algoritmo (come SIFT) non ha portato vantaggi significativi, anzi riducendo il numero di keypoint si riducono il numero di corrispondenze trovate. Infine il tempo di estrazione dei keypoint è significativamente maggiore con SIFT.

Calcolo keypoint

Per il calcolo degli *Uniform keypoint* si utilizza la classe `pcl::UniformSampling` di PCL che prevede:

- **Input**: point cloud di cui calcolare i keypoint
- **Parametri**:
  - `leaf_size`: dimensione delle foglie della griglia 3D
- **Output**: point cloud di keypoint

2.3.2 Descrittori

I descrittori sono delle strutture che contengono informazioni utili alla descrizione sintetica dei punti di una *point cloud*. Sono utilizzate per cercare

le corrispondenze tra i *keypoint* di modello e scena. Nel presente lavoro sono stati testati due tipi di descrittori [8]: FPFH e SHOT352.

2.3.2.1 FPFH (Fast Point Feature Histogram)

L'algoritmo che calcola i descrittori FPFH [9] (o Fast-PFH) è, come suggerisce il nome, una implementazione dell'algoritmo PFH [10] che riduce drasticamente la complessità computazionale e temporale senza inficiare eccessivamente l'informazione apportata. I descrittori FPFH sono delle *feature* locali basate sulle normali alla superficie della *cloud* e utilizzano l'informazione della *k-neighborhood* del punto corrente (i k punti più vicini).

La principale differenza tra PFH-descriptors e FPFH-descriptors è data dal fatto che, mentre l'algoritmo PFH calcola l'*edge* di ogni coppia della *mesh* che la *k-neighborhood* crea (k^2 *edges*), FPFH tiene conto solo delle k connessioni che collegano il punto in esame a ciascun punto della *neighborhood*. La complessità si riduce quindi da $O(n \cdot k^2)$ a $O(n \cdot k)$.

Il calcolo del descrittore FPFH per il generico punto p_q avviene in due *step*:

$$FPFH(p_q) = SPFH(p_q) + \frac{1}{k} \sum \frac{1}{\omega_k} \cdot SPFH(p_k)$$

Come suggerisce la formula, va anzitutto calcolata la **SPFH (Simplified-PFH)** di tutti i punti, quindi l'istogramma FPFH per ciascun punto, utilizzando i valori pesati degli SPFH dei suoi vicini.

Calcolo FPFH

Per il calcolo delle *feature* FPFH si utilizzano le classi di PCL `pcl::FPFHEstimation` o `pcl::FPFHEstimationOMP` per la versione parallelizzata che prevede:

- **Input**:
  - Point cloud: si usa una versione voxellizzata di quella originale in modo da ridurre il numero di punti
  - Normali: calcolate sulla point cloud in input
  - Keypoint: punti sui quali vengono calcolate le *feature*
- **Parametri**:
  - `normal_rad`: raggio utilizzato per il calcolo delle normali
  - `fpfh_rad`: raggio utilizzato per il calcolo
- **Output**: point cloud di *feature*, una per ogni keypoint

2.3.2.2 SHOT352 (Signature of Histograms of OrienTations)

Il descrittore SHOT[11] è basato sull'idea di ottenere un *local reference frame* utilizzando la *eigenvalue decomposition* attorno a ciascun punto. Dato questo *reference frame*, viene costruita una griglia di sfere centrate ciascuna in ciascun punto, in modo da dividere i punti in un insieme di *neighborhood* e in modo tale che ogni in ogni sfera si ottenga un istogramma pesato di normali. Il descrittore concatena tutti questi istogrammi nella *signature* finale. Esso utilizza 9 valori per codificare il *reference frame*, inoltre vengono utilizzati 11 *shape bins* e 32 divisioni di griglie "sferiche", cosa che porta a 352 (32×11) valori aggiuntivi, da cui il nome **SHOT352**. I descrittori sono infine normalizzati per sommare a 1.

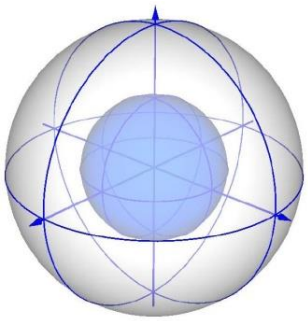

Figura 2.9: Struttura di una SHOT *signature*.

In seguito ai risultati ottenuti durante primi test si è scelto di non utilizzare questo tipo di descrittori, preferendo quelli di tipo FPFH in quanto più performanti sia computazionalmente che qualitativamente nel calcolo delle corrispondenze per il tipo di modelli utilizzati.

2.3.2.3 BOARD LRF (Local Reference Frame)

Il BOrder Aware Repeatable Directions (BOARD[12]) è un algoritmo studiato per il calcolo del Local Reference Frame di un keypoint. L'algoritmo è robusto anche quando lo spazio adiacente al punto di cui si vuole calcolare l'orientazione (LRF) è occluso o non visibile (Border Point). Orientare le *feature* nello spazio tridimensionale è molto importante quando si cerca un match tra viste parziali di una superficie. I LRF sono utilizzati nella fase di correspondence grouping.

La libreria PCL fornisce una implementazione di questo algoritmo nella classe `pcl::BOARDLocalReferenceFrameEstimation`. Il calcolo prevede:

- **Input**:
  - Point cloud: si usa una versione voxellizzata di quella originale in modo da ridurre il numero di punti e di conseguenza la complessità computazionale
  - Normali: calcolate sulla point cloud in input
  - Keypoint: punti sui quali vengono calcolate le *feature*
- **Parametri**:
  - `normal_rad`: raggio utilizzato per il calcolo delle normali
  - `rf_rad`: raggio utilizzato per il calcolo
- `Output`: point cloud di `pcl::ReferenceFrame`, uno per ogni keypoint

2.3.3 Corrispondenze

In seguito al calcolo dei descrittori si ricercano corrispondenze tra quelli di tipo FPFH. Una corrispondenza, infatti, non è altro che una coppia di *feature*, una del modello e una della scena, che presentano una forte somiglianza. Essendo i descrittori calcolati per ogni *keypoint*, in seguito al calcolo delle corrispondenze si ottengono una serie di coppie costituite da un *keypoint* della scena e uno del modello.

La classe `pcl::registration::CorrespondeceEstimation` di PCL implementa due metodi per il calcolo delle corrispondenze tra due *point cloud* di descrittori (FPFHSignature33 in questo caso). Entrambi utilizzano un KDTreeFLANN [13] per ricercare le corrispondenze tra una sorgente (modello) e un obiettivo (scena). I metodi di PCL sono pensati principalmente per effettuare il processo di registrazione, in cui si cercano le corrispondenze sorgente-target (modello-scena in questo caso). Tuttavia, questo approccio non è adatto al caso in cui la scena contenga più istanze dello stesso oggetto.

Nel riconoscimento di oggetti basato su *correspondence grouping*, invece, la fase di ricerca delle corrispondenze si effettua cercando, per ogni descrittore della scena, il descrittore più vicino del modello; si consideri che la distanza di due descrittori indica il loro grado di somiglianza. In questo modo è poi possibile, con algoritmi di *correspondence grouping* come Hough3DGrouping, dividere le corrispondenze in cluster, uno per ogni istanza del modello nella scena.

Per questo motivo il metodo di ricerca delle corrispondenze è stato riscritto in modo da restituire le corrispondenze modello-scena, nonostante vengano calcolate le corrispondenze scena-modello. In breve, il metodo di ricerca è lo

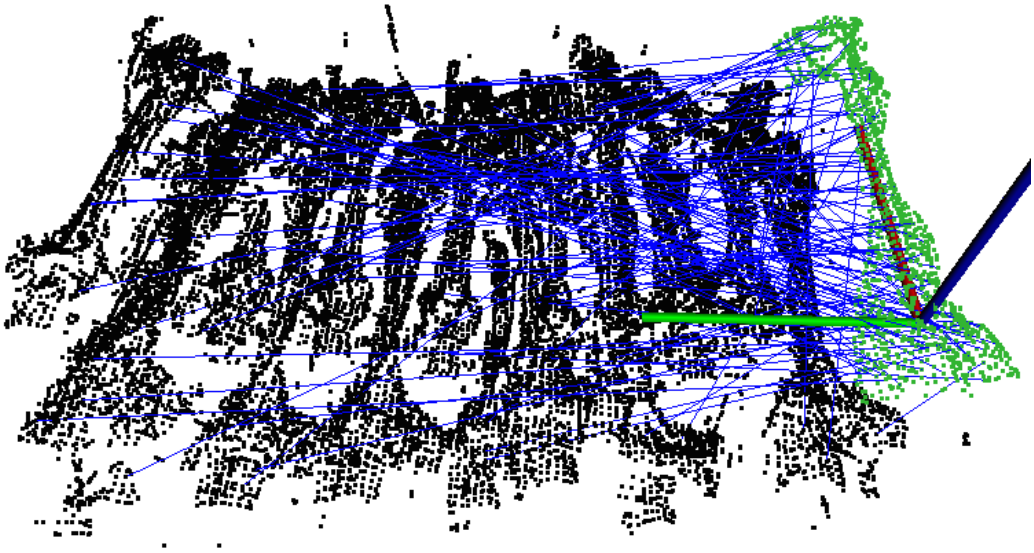

Figura 2.10: Esempio di corrispondenze in blu (per motivi di leggibilità è mostrata una corrispondenza su 15 rispetto a quelle realmente), in nero i keypoint della scena, in verde la vista del modello.

stesso utilizzato da PCL ma al momento della creazione della corrispondenza vengono scambiati gli indici di source e target.

Infine, le corrispondenze che hanno una distanza tra i descrittori superiore a un threshold vengono scartate.

```
void findCorrespondences ( PointCloud<FPFHSignature33>::Ptr source,
    PointCloud<FPFHSignature33>::Ptr target, CorrespondencesPtr
    correspondences )
{
    // Inizializzazione KdTree
    search::KdTree<FPFHSignature33> tree;
    tree.setInputCloud(target);

    std::vector<int> index (1);
    std::vector<float> distance (1);
    unsigned int nr_valid_correspondences = 0;

    // per ogni punto della scena
    for(int j = 0; j < source->size(); ++j)
    {
        // cerca il descrittore piu' simile nel modello
        tree.nearestKSearch(source->at(j), 1, index, distance);

        // se la differenza tra i 2 descrittori supera una soglia fissata
        // ignora la corrispondenza
        if(distance[0] > distance_threshold)
            continue;

        // crea corrispondenza
```

```
        Correspondence corr;

        // indice della feature/kp del modello
        corr.index_query = index[0];
        // indice della feature/kp della scena
        corr.index_match = j;
        corr.distance = distance[0];

        // aggiungi corrispondenza all'output
        correspondences->at(nr_valid_correspondences++) = corr;
    }
    correspondences->resize(nr_valid_correspondences);
}
```

Listing 2.1: Funzione di calcolo corrispondenze

2.3.4 Correspondence grouping

É in questa fase che il processo di riconoscimento si distingue maggiormente dal problema di registrazione, in cui si utilizzano algoritmi come RANSAC (RANdom SAmple Consensus) per filtrare le false corrispondenze e si cerca di individuare la miglior trasformazione dell'oggetto sulla scena (Figura 2.11). Tuttavia, il metodo basato su registrazione fallisce poiché la scena contiene multiple istanze dello stesso oggetto. Per questo motivo è necessario clusterizzare la scena a monte, prima della ricerca delle corrispondenze, e risolvere il problema della registrazione per ogni cluster.

Figura 2.11: In alto: corrispondenze tra sorgente e target. In basso: corrispondenze filtrate con RANSAC

Un algoritmo di *correspondence grouping* è studiato per gestire multiple istanze dello stesso oggetto nella scena. Il suo scopo principale è dividere le corrispondenze in gruppi, ognuno dei quali è utilizzato per determinare una trasformazione del modello cercato nella scena. In questo lavoro è stata utilizzata la classe `pcl::Hough3DGrouping`, un sistema di *correspondence grouping* basato sulla trasformata generalizzata di Hough applicata al 3D.

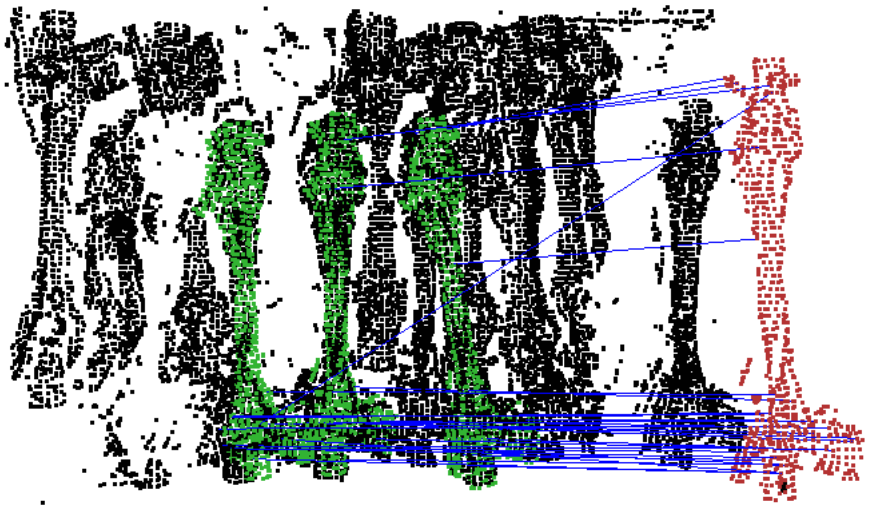

Figura 2.12: Tre possibili istanze trovate in seguito a grouping delle corrispondenze

Le corrispondenze visualizzate in figura 2.10 vengono divise in cluster e, per ogni cluster che soddisfa determinate condizioni, si stima una possibile trasformazione iniziale del modello. Il risultato è visibile in figura 2.12.

2.3.4.1 Hough 3D Grouping

La **trasformata generalizzata di Hough** (GHT) [14] è un metodo che consente di individuare istanze di una particolare forma in un'immagine. Si basa su un sistema di voto che viene effettuato nello spazio dei parametri. Qualsiasi forma può essere rappresentata da un insieme di parametri: ad esempio, un cerchio può essere rappresentato (e quindi trasformato) in un set di 3 parametri che rappresentano le coordinate del centro e il raggio;

una retta viene mappata in una coppia di parametri cioè l'arcotangente del coefficiente angolare e il termine noto, e così via. La trasformata generalizzata di Hough usa i principi del template matching per permettere, non solo di individuare un oggetto descrivibile tramite un'equazione analitica, ma anche oggetti dalla forma arbitraria descritti con un loro modello. Il problema diviene, quindi, quello di trovare la posizione del modello nella scena o, più precisamente, trovare i parametri della matrice di roto-traslazione che mappa il modello nell'immagine. La GHT usa le informazioni sui bordi per definire una mappatura dall'orientazione di un edge point ad un reference point.

Una pipeline generale della GHT consiste:

1. Lo spazio n-dimensionale dei parametri viene discretizzato in *bin* (iper-rettangoli di n dimensioni)

2. Per ogni edge point dell'immagine viene messo un voto in un bin dello spazio dei parametri che può aver generato quel punto, ossia ogni punto vota per quello che riconosce come proprio reference point

3. Vengono scelti i *bin* che hanno il maggior numero di voti

La trasformata di Hough è piuttosto robusta al rumore e conseguentemente accadrà che i punti cosiddetti "rumorosi" non voteranno in maniera consistente per una singola posa del modello. I dati che risultano da questi voti inesatti non hanno alcuna conseguenza apprezzabile fintanto che rimangono abbastanza *edge point* che concordano su un determinato reference-point.

Come già anticipato, la classe `pcl::Hough3DGrouping` implementa un algoritmo di correspondence grouping basato sulla **GHT** in uno spazio tridimensionale [15] [16].

Come nel caso 2D, il sistema di voti di Hough ha l'obiettivo di accumulare prove della presenza dell'oggetto cercato. Se abbastanza *feature* votano per la presenza dell'oggetto in una data posizione, allora l'oggetto è riconosciuto e un allineamento iniziale è determinato utilizzando corrispondenze votanti.

Il primo passo consiste nel calcolare un *reference point* univoco per il modello, C^M, per comodità lo si fa coincidere con il centroide (punto rosso in figura 2.13). Successivamente si calcolano i *reference frame* (RF), ovvero vettori tra ciascuna feature, F_i^M, e il centroide (frecce blu in 2.13). Questi vettori sono memorizzati utilizzando il sistema di riferimento locale (LRF) in modo da rendere il sistema invariante a rototraslazioni. In questo modo, sfruttando le corrispondenze è possibile riportare i vettori sulla scena con lo stesso orientamento secondo un sistema di riferimento locale. A questo punto ogni feature F_i^M dà un voto per una posizione di un reference point nella scena.

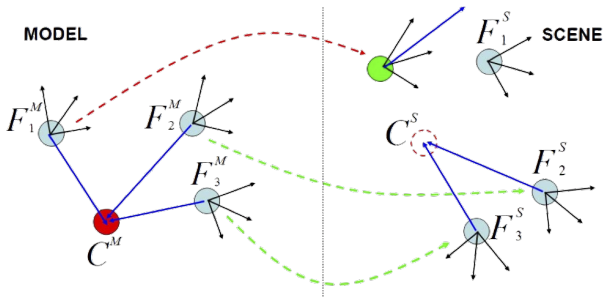

Figura 2.13: Esempio di 3D Hough Voting basato su RF.

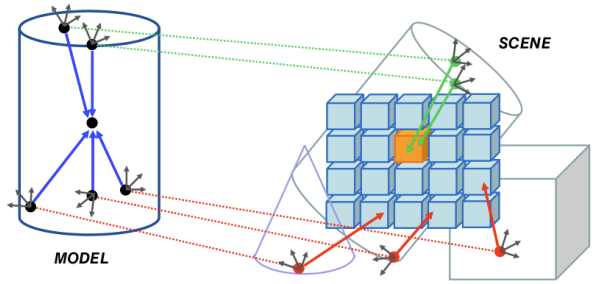

Figura 2.14: Esempio di 3D Hough Voting basato su LRF.

La scena viene divisa in una griglia tridimensionale di *bin* che raccolgono i voti delle feature. Gli spazi che raccolgono il maggior numero di voti determinano la presenza di una istanza. Grazie a questo sistema è possibile trovare più istanze dello stesso oggetto contemporaneamente.

Riassumendo, la classe `pcl::Hough3DGrouping` richiede:

- **Input**:
  - **Keypoint** del modello e della scena
  - **LRF** calcolati sui keypoint del modello e della scena
  - **Corrispondenze** tra i keypoint confrontando i descrittori FPFH
- **Parametri**:
  - `hough_bin_size`: dimensione dei *bin* in cui viene suddiviso lo spazio di voto

  - `hough_threshold`: numero minimo di voti in un *bin* per considerare una possibile istanza

- **Output**: lista di matrici di roto-traslazione, una per ogni istanza che ha raggiunto un numero sufficiente di voti

2.3.5 Allineamento iniziale

L'allineamento iniziale consiste nella trasformazione del modello utilizzando la matrice di roto-traslazione che Hough3DGrouping fornisce in output. Tale matrice è calcolata utilizzando le corrispondenze clusterizzate al passo precedente, ogni gruppo di corrispondenze permette di calcolare una matrice di roto-traslazione. L'insieme delle trasformazioni permette di ottenere una serie di *guess* della posa dell'oggetto cercato.

2.3.5.1 Pre-validazione

Molte delle ipotesi di posa del modello, restituite dalla fase di *correspondence grouping*, sono errate o falsi positivi (disallineate o modello sbagliato). Queste istanze verranno sicuramente scartate nella fase successiva in seguito all'allineamento finale con ICP. Sperimentalmente si è verificato che l'algoritmo ICP è un *bottleneck* considerevole e per questo motivo viene operata una prima validazione delle istanze in seguito all'allineamento iniziale, in questo modo il tempo di esecuzione viene drasticamente ridotto.

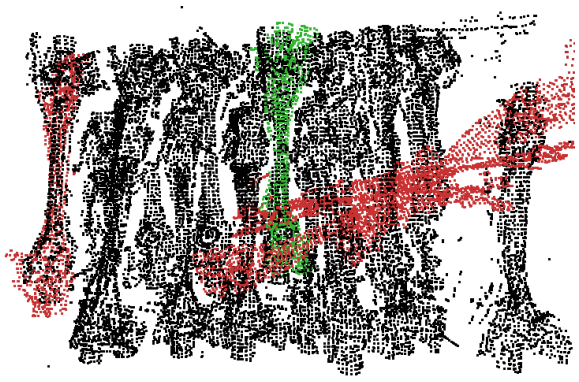

Figura 2.15: Esempio di output in seguito a prima validazione: le istanza scartate in rosso, le istanze che passano alla fase successiva in verde, la scena in nero.

In questa fase viene calcolata la *Squared Euclidean Distance* media tra la point cloud del modello (i.e. il modello roto-traslato secondo le matrici

rilevate al passo precedente) e la scena. Per un modello di K punti ed una scena di N punti, ad esempio, essa è calcolata come:

$$\text{Average Squared Euclidean Distance} = \frac{1}{K}\sum_{i=0}^{K-1} min_{j=0}^{N-1} d(\text{istanza.i, scena.j})$$

dove d(istanza.i, scena.j) rappresenta la *Squared Euclidean Distance* tra l'i-esimo punto dell'istanza ed il j-esimo punto della scena:

$$d(A, B) = (A^x - B^x)^2 + (A^y - B^y)^2 + (A^z - B^z)^2$$

Successivamente le istanze con uno score superiore a un *threshold* vengono scartate in quanto si ipotizza che siano frutto di un *match* errato; quelle "sopravvissute" a questa fase verranno infine allineate alla scena e subiranno un processo di verifica più precisa.

L'implementazione di tale metodo è mostrata nel listato 2.2.

```
double squared_euclidean_distance(const PointCloud::Ptr model, const
    PointCloud::Ptr scene)
{
    // per ricerca K nearest neighbor
    pcl::KdTree<PointXYZ> kdtree;
    kdtree.setInputCloud (scene);

    double distance = 0;

    std::vector<int> pointIdxNKNSearch(1);
    std::vector<float> pointNKNSquaredDistance(1);

    for (int i = 0; i < model->size(); i++)
    {
        kdtree.nearestKSearch (searchPoint, 1, pointIdxNKNSearch,
            pointNKNSquaredDistance)
        distance += pointNKNSquaredDistance[0];
    }
    return distance / model->points.size();
}
```

Listing 2.2: Calcolo della distanza tra due *pointcloud*.

2.3.6 Allineamento finale

L'allineamento iniziale non è preciso a sufficienza per consentire la presa da parte di una pinza guidata da un braccio robotico. É necessario quindi affinare la trasformazione in modo che il modello sia allineato alla perfezione con l'oggetto presente nella point cloud della scena.

Per raggiungere questo scopo si utilizza l'algoritmo **ICP** (Iterative Closest Point) implementato in `pcl::IterativeClosestPoint`. ICP calcola le

corrispondenze tra coppie di punti più vicini fra loro e iterativamente cerca la trasformazione geometrica che minimizza la distanza tra le nuvole di punti.

2.3.6.1 Validazione

In seguito all'allineamento finale dei *match* più promettenti viene calcolato un nuovo *score* per valutare la qualità dell'allineamento finale e, di conseguenza, se aggiungere la trasformazione ottenuta alla lista dei match oppure se scartarla e passare alla successiva ipotesi.

Questa seconda funzione di validazione calcola la percentuale di *outlier*[6] presenti tra la scena ed il modello allineato. Il calcolo dello *score* segue la seguente procedura:

1. per ciascun punto del modello viene individuato il punto più vicino della scena
2. se la distanza tra due punti è inferiore ad una soglia fissata tale punto viene considerato un *inlier*; in caso contrario viene considerato come *outlier*,
3. si calcola la percentuale di punti del modello considerati come *outlier*.

Le istanze che superano entrambe le fasi di validazione vengono considerate oggetti riconosciuti di cui si conosce la posa. Queste istanze verranno utilizzate per il calcolo del punto di presa e della traiettoria di estrazione del pezzo da parte del braccio robotico senza causare collisioni con altri componenti o con le pareti della cassa.

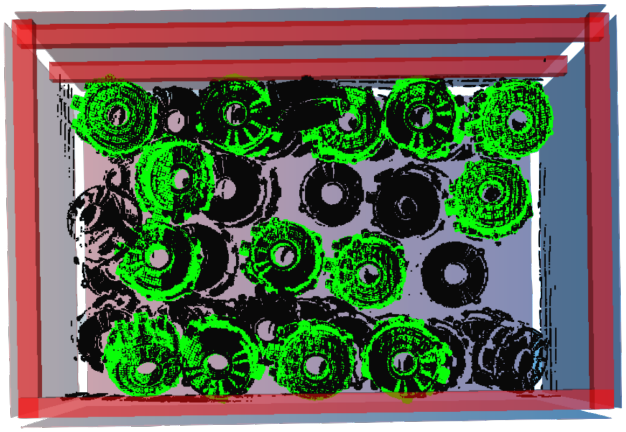

Figura 2.16: Risultato finale del riconoscimento. Gli oggetti riconosciuti (in verde) sono allineati sulla scena (in nero) all'interno del *bin*.

Capitolo 3

Algoritmi

In questo capitolo si analizzano due algoritmi di riconoscimento che utilizzano i due differenti approcci presentati nel capitolo precedente. Il primo algoritmo, basato sulla registrazione, è utilizzato come punto di riferimento, in quanto ampiamente collaudato e testato. Attualmente SmartPick3D utilizza una versione altamente ottimizzata di questo algoritmo. In questo lavoro è stato sviluppato il secondo algoritmo proposto, basato su *correspondence grouping*, da affiancare a quello esistente.

Queste procedure sono state sviluppate iterando alcune fasi del riconoscimento o della registrazione, in modo da riconoscere più istanze di un oggetto utilizzando più modelli o più viste dello stesso modello (ognuna è trattata come un modello differente). Inoltre, alcuni passi sono ripetuti variando la configurazione dei parametri. Questa scelta è motivata dal fatto che per quanto si possano cercare di ottimizzare tali valori, a causa di rumore variabile introdotto durante l'acquisizione o differenti angolazioni di posa degli stessi oggetti, non è possibile trovare un'unica configurazione di parametri che garantisca il riconoscimento di tutte le istanze in una singola iterazione.

3.1 Algoritmo 1: *clustering + registrazione*

Come già anticipato nel capitolo precedente, il primo passo consiste nel clustering della scena, successivamente si procede come riportato nello pseudocodice dell'algoritmo 1.

3.1.1 Analisi

In questo algoritmo si calcolano più allineamenti iniziali per ogni cluster della scena, quello con il punteggio migliore sarà la miglior trasformazione iniziale

per quel cluster. Ogni trasformazione è calcolata da un set di corrispondenze stimate da un set di FPFH calcolate con lo stesso raggio.

Algoritmo 1 clustering + registrazione

Input: Cluster della scena e modelli da cercare
Output: Lista di trasformazioni e punteggio

```
for all cluster della scena do
  - trova uniform keypoint del cluster e dell'oggetto
  for all raggio FPFH do
    - calcolo descrittori FPFH del cluster e dell'oggetto
    - ricerca corrispondenze tra i descrittori
    for all SAC threshold do
      - filtraggio delle corrispondenze con RANSAC
      - stima di una trasformazione rigida utilizzando le corrispondenze
      rimaste che minimizzi la distanza
      - score1 ⇐ Euclidean Distance
      - conserva la trasformazione iniziale con score1 migliore
    end for
  end for
end for
for all trasformazione iniziale con score1 > threshold1 do
  - uniform sampling di scena e oggetto
  - calcolo trasformazione finale utilizzando ICP su cloud ridotte
  - score2 ⇐ Percentuale Outlier
  if score2 inferiore a soglia then
    - aggiungi trasformazione a oggetti riconosciuti
  end if
end for
```

Nel caso di riconoscimento, utilizzando più modelli o più viste dello stesso modello si introduce un ulteriore ciclo innestato all'interno del secondo, quindi si calcolano le corrispondenze tra un cluster e ogni vista del modello.

I vantaggi di questo approccio che utilizza il clustering sono:

- Riduzione del rumore nella fase di allineamento iniziale operando su cluster della scena
- Parallelizzabilità: possibilità di effettuare il riconoscimento in parallelo su più cluster contemporaneamente
- Rimozione di duplicati: operando il riconoscimento su cluster si riduce la probabilità di riconoscere più volte la stessa istanza

- Falsi negativi: cercando un match per ogni cluster si riduce la percentuale di falsi negativi

Tuttavia, il clustering non è sempre vantaggioso in quanto non semplice da effettuare per alcuni tipi di prodotti. Per la natura di alcuni oggetti, per esempio, sono necessarie più fasi di clustering che introducono ulteriore complessità computazionale.

Questa analisi ha motivato la ricerca di un algoritmo che non preveda la ricerca di cluster. Tuttavia, il processo di validazione studiato in questo algoritmo si è rivelato robusto, cioè in grado di validare una trasformazione con una percentuale di falsi positivi prossima a zero. Ciò è stato tenuto in considerazione e preso come spunto nello sviluppo di un nuovo algoritmo.

3.2 Algoritmo 2: *correspondence grouping*

Come nel caso precedente la pipeline descritta in letteratura è stata modificata inserendo delle iterazioni per alcuni parametri considerati più influenti durante l'esecuzione.

In questo caso, come già anticipato, si cercano le corrispondenze tra ogni modello e la scena. In seguito, utilizzando tali corrispondenze, si individuano una serie di ipotesi di posa degli oggetti. Tali ipotesi sono valutate con la funzione di *score* basata su distanza euclidea; in seguito all'allineamento finale si valida o si scarta la posa trovata a seconda del risultato. Lo pseudocodice dell'algoritmo è riportato a pagina successiva (Algoritmo 2).

Rispetto alla versione precedente si rimuove un ciclo innestato quindi la complessità diminuisce; tuttavia, ogni ciclo opera sull'intera scena invece che su cluster, quindi il calcolo delle *feature* e delle corrispondenze richiede maggior tempo, coinvolgendo un numero maggiore di punti.

La scelta di rimuovere i match validi alla fine di ogni iterazione è motivata dal fatto che:

- Si riduce il numero di punti della scena e di conseguenza la complessità dell'input dell'iterazione successiva

- Si evita di individuare istanze già individuate: variando solo i parametri tra un ciclo e il ciclo successivo è molto probabile individuare nuovamente le stesse pose a discapito di altre non ancora individuate

Nel caso di riconoscimento di più modelli contemporaneamente, è necessario introdurre un ulteriore ciclo in cui si inserisce la parte di algoritmo che va dal calcolo di keypoint e di descrittori alla prima validazione delle ipotesi. Tale scelta è dovuta a motivi di performance e parallelizzazione.

Algoritmo 2 *correspondence grouping*

Input: Point cloud: *scena* - point cloud della scena, *model* - point cloud del modello da cercare.
Output: *matches* - lista di trasformazioni e punteggio.

```
for all dimensione bin do
  - trova uniform keypoint e descrittori FPFH di scene
  - trova uniform keypoint e descrittori FPFH di model
  - ricerca corrispondenze descrittori di scene e di model
  - hypotheses ⇐ lista di ipotesi di posa ottenute clusterizzando le
  corrispondenze con Hough3DGrouping
  for all hypothesis in hypotheses do
    - score ⇐ punteggio calcolato con Euclidean Distance
    if score non valido then
      - hypothesis scartata
    end if
  end for
  for all hypothesis in hypotheses do
    - match ⇐ allineamento finale mediante ICP
    - score ⇐ punteggio calcolato con Euclidean Distance
    if score valido then
      - aggiungi match a matches
    end if
  end for
  - rimozione duplicati
  - riduzione scene sottraendo matches
end for
```

3.2.1 Versione parallelizzata

Utilizzando le API **OpenMP** è stata implementata anche una versione parallelizzata dell'algoritmo. Come già anticipato nella sezione 1.5, OpenMP è un modello di programmazione per sistemi di tipo *shared memory*, fattore di cui si è dovuto tenere conto durante la parallelizzazione del codice.

In questa versione si possono individuare tre regioni parallele: una regione principale che contiene al suo interno due regioni annidate. La regione principale introduce parallelizzazione a livello di modelli da individuare. Si è deciso di introdurre le regioni annidate in modo che, nel caso in cui il numero di modelli da individuare sia inferiore al numero di processori disponibili, non rimangano *core* inutilizzati. Le regioni parallele innestate operano a livello di validazione iniziale e finale. Nel capitolo **Prestazioni e risultati** (Capitolo

5) si esegue un ulteriore analisi delle prestazioni.

3.2.2 Parametri

Si presentano ora i parametri numerici necessari all'utilizzo dall'algoritmo 2. La maggior parte dei valori possono essere modificati agendo sulla configurazione dell'algoritmo. Alcuni parametri che non necessitano di modifiche sono definiti all'interno del codice, gli altri sono passati come input nella configurazione del software.

La scelta dei seguenti parametri è stata fatta tenendo in considerazione gli effetti che essi hanno sulle performance complessive.

- `resolution` = 2.0 [mm]
  Dimensione della griglia del filtro voxel iniziale

- `kp_initial_resolution` = 9.0 [mm]
  Dimensione della griglia utilizzata per il calcolo dei keypoint (Uniform Sampling) per l'allineamento iniziale
  Condizioni: > `resolution`

- `fpfh_rad` = 25.0 [mm]
  Raggio utilizzato per il calcolo dei descrittori FPFH
  Condizioni: > `normal_rad`

- `normal_rad` = 0.8·`fpfh_rad` [mm]
  Raggio utilizzato per il calcolo delle normali
  Condizioni: < `fpfh_rad`

- `lrf_rad` = `fpfh_rad` [mm]
  Raggio utilizzato per il calcolo dei Local Reference Frame

- `num_bin_size` = 4-8 [numero di cicli]

- `min_bin_size` = 8-10% della misura lato più lungo del modello [mm]

- `max_bin_size` = 15-20% della misura lato più lungo del modello [mm]
  Parametri utilizzati per il calcolo del `bin_size` nel ciclo: vengono effettuate `num_bin_size` iterazioni partendo da `min_bin_size` fino a `max_bin_size`

- `hough_threshold` = 5 [numero di corrispondenze]
  Numero minimo di corrispondenze per considerare ipotesi valida

- `kp_initial_resolution` = 9.0 [mm]
  Dimensione della griglia utilizzata per il calcolo dei keypoint (Uniform Sampling) per l'allineamento iniziale
  Condizioni: > `resolution`

- `kp_initial_resolution` = 9.0 [mm]
  Dimensione della griglia utilizzata per il calcolo dei keypoint (Uniform Sampling) per l'allineamento iniziale
  Condizioni: > `resolution`

- `corr_distance_threshold` = 300
  Soglia per considerare valida una corrispondenza: se viene trovata una corrispondenza tra due descrittori che hanno distanza maggiore di `corr_distance_threshold` quest'ultima viene scartata

- `icp_max_iteration` = 50

- `icp_epsilon` = 10^{-7}
  Condizioni di terminazione per l'algoritmo ICP

- `initial_score_threshold` = 0.4

- `final_score_threshold` = 0.3
  Soglie per la validazione dell'allineamento iniziale e finale

Capitolo 4

TestFramework

Lo sviluppo di uno strumento software per effettuare test, con diverse configurazioni di parametri o di algoritmi utilizzati, è stato necessario allo scopo di confrontare l'algoritmo proposto con quello esistente. Lo stesso strumento potrà anche essere utilizzato per scopi di *parameter tuning*. In questo capitolo si analizzano requisiti e architettura del framework sviluppato.

4.1 Analisi dei requisiti

Lo scopo principale è fornire uno strumento per testare la configurazione dell'applicazione **SmartPick3D**, dove per configurazione si intende tipologia di prodotti da riconoscere, parametri e algoritmi da utilizzare. Il TestFramework deve avere le seguenti funzionalità:

- Gestire l'esecuzione automatica di *dump*
- Caricare più configurazioni variando parametri e algoritmi da utilizzare per l'esecuzione dei test
- Raccogliere i risultati in formato analizzabile

4.2 Architettura

SmartPick3D è sviluppato all'interno di un framework specifico per applicazioni di visione, sviluppato da IT+Robotics. L'ambiente di sviluppo fornito offre diversi vantaggi, tra cui una maggiore semplicità nel riutilizzo dei componenti sviluppati nelle varie applicazioni che vengono implementate usando il tale framework.

L'implementazione del TestFramework ha rispettato i vincoli imposti dallo stile architetturale preesistente.

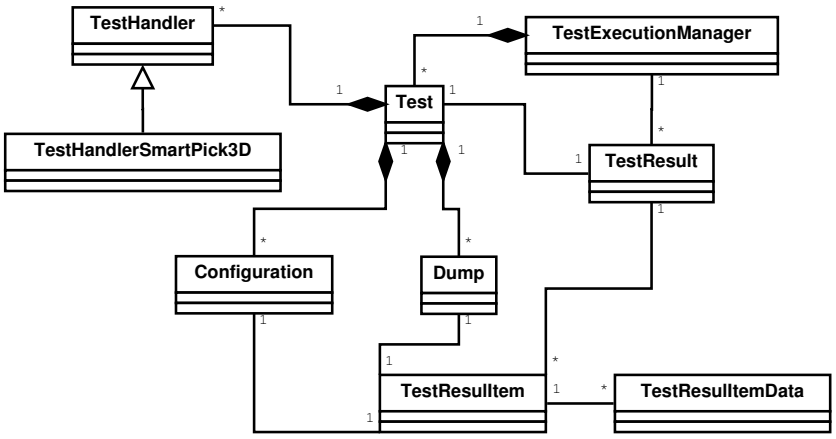

Figura 4.1: Schema UML delle principali classi del TestFramework.

Le classi che implementano il TestFramework sono:

- **Test**: rappresenta il test da eseguire. Ogni test prevede l'esecuzione di uno o più *dump* eseguiti con una o più configurazioni.

- **Dump**: contiene le informazioni di stato salvate durante un ciclo di esecuzione di una applicazione.

- **Configuration**: configurazione di parametri.

- **TestHandler**: rappresenta una combinazione dump/configurazione da eseguire. Per ogni applicazione è presente un **TestHandler** specifico. Per SmartPick3D è presente **TestHandlerSmartPick3D**.

- **TestExecutionManager**: gestisce l'esecuzione dei **Test**.

- **TestResult**: rappresenta i risultati di un **Test**. Contiene per ogni combinazione configurazione/dump una lista di **TestResultItem**.

- **TestResultItemData**: rappresenta un risultato dell'esecuzione di un *dump*, ad esempio il tempo di esecuzione o il numero di prodotti trovati.

4.3 Interfaccia Grafica

L'interfaccia grafica realizzata consente di caricare i file di configurazione e i *dump*. Durante l'esecuzione di un test è possibile visualizzare lo stato e i risultati ottenuti.

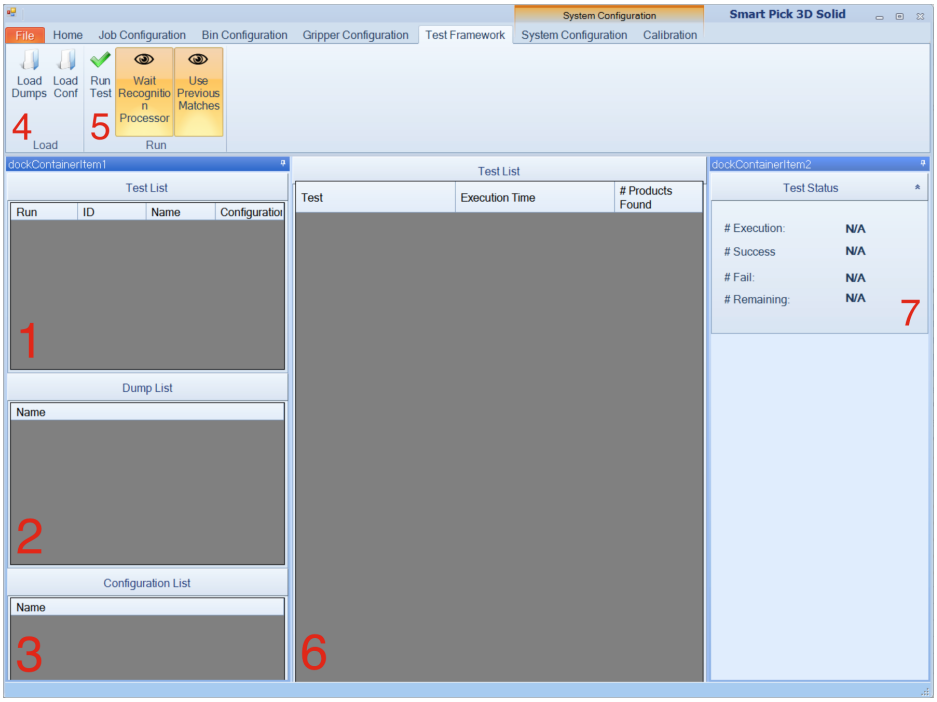

Figura 4.2: Screenshot della GUI del Test Framework sviluppata per Smart Pick 3D.

La GUI visibile in figura 4.2 è composta dalle seguenti parti:

1. Lista dei test da eseguire
2. Lista dei *dump* caricati
3. Lista dei file di configurazione caricati
4. Pulsanti per caricare *dump* e configurazioni
5. Pulsanti per avviare l'esecuzione dei test
6. Tabella in cui vengono riportati i risultati dei test
7. Stato dell'esecuzione in corso: numero di test eseguiti, falliti e rimanenti.

Capitolo 5

Prestazioni e Risultati

Per valutare le prestazioni dell'algoritmo sviluppato (algoritmo 2) si confrontano i risultati con quelli ottenuti con la versione esistente (algoritmo 1). Sono stati eseguiti test con diverse tipologie di oggetti e diversi set di scansioni ottenute con il sistema di triangolazione laser. Il dataset a disposizione proviene da un sistema attualmente utilizzato in produzione.

I test 1 e 2 hanno lo scopo di valutare le prestazioni della soluzione proposta, il test 3 valuta lo speedup ottenuto con la versione parallela mentre il test 4 mette a confronto le prestazioni dei due algoritmi.

I test sono stati effettuati utilizzando una macchina avente le seguenti specifiche:

- CPU: Intel Core i7 dual-core @ 2.8GHz, Turbo Boost fino a 3.3GHz
- RAM: 16GB di SDRAM DDR3L a 1600MHz
- Hard Disk: Unità flash PCIe da 512GB
- Sistema Operativo: OS X Mavericks

Su questo sistema è stata installata una macchina virtuale grazie al software **VMWare Fusion 6**. Le specifiche della macchina *guest* utilizzata per sviluppare e testare il software sono le seguenti:

- CPU: 4 core virtuali condivisi con l'*host*
- RAM: 8GB dedicati
- HD: 128GB
- Sistema Operativo: Microsoft Windows 7 SP1

La GPU non è specificata in quanto non rilevante per i test eseguiti.

5.1 Oggetti e dataset

Gli oggetti di cui si esegue il riconoscimento sono dei componenti di alluminio ottenuti tramite pressofusione. Tali prodotti devono essere prelevati da un braccio robotico per essere caricati nella fase di lavorazione successiva.

Sono stati scelti tre prodotti di forme e dimensioni diverse tra loro in modo da verificare le performance su varie tipologie di modelli. Si riporta una breve descrizione dei modelli e del numero di viste utilizzate per il riconoscimento (sezione 2.1.1).

5.1.1 Prodotto A

Il prodotto A ($500 \times 450 \times 250\,mm$) è un blocco di alluminio compatto e pesante, ad eccezione di un anello laterale, ed un alloggiamento scavato al suo interno. In figura 5.1 si possono vedere la faccia superiore e inferiore del prodotto.

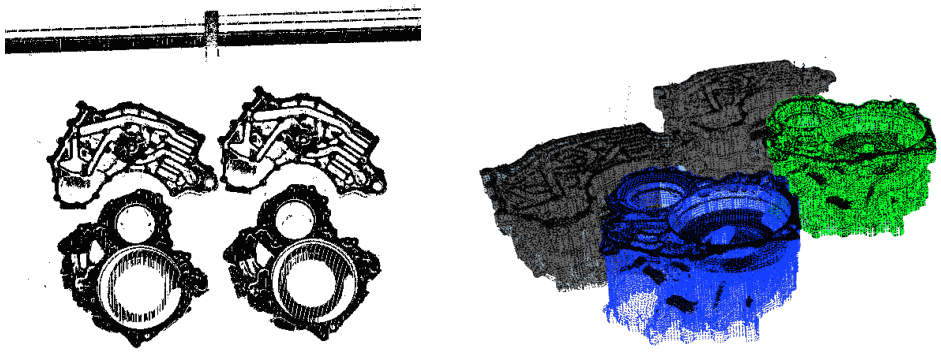

Figura 5.1: A sinistra: esempio di scansione di scena contenente quattro prodotti di tipo A visti dell'alto. Nella parte superiore è visibile una parete del contenitore. A destra: risultato del riconoscimento; il modello è stato allineato ai quattro oggetti individuati (point cloud in blu verde e grigio) nella scena (parti in nero). Il modello per questo tipo di prodotto non viene mostrato in quanto troppo complesso per essere leggibile.

Data la geometria e la tecnica utilizzata per impilare questo tipo di prodotti si utilizzano solo due viste del modello per il riconoscimento.

5.1.2 Prodotto B

Il prodotto B ($200 \times 200 \times 150\,mm$) è un carter in alluminio di forma cilindrica, il lato di copertura presenta un foro al centro. Anche questo componente è

ottenuto con la tecnica di pressofusione.

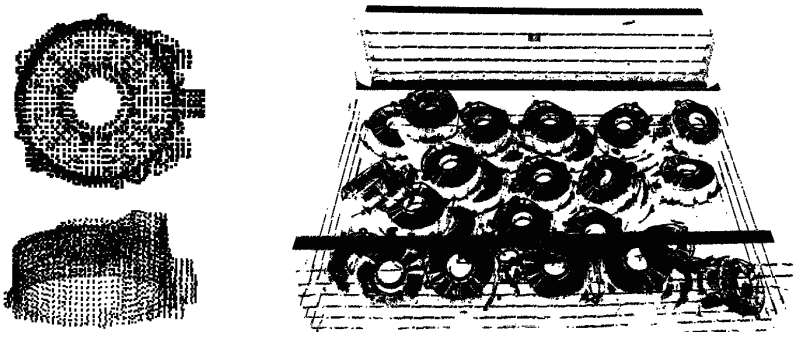

Figura 5.2: A sinistra: modello prodotto B. A destra: scansione contenitore contenente il prodotto B.

5.1.3 Prodotto C

Il prodotto C (550 mm di lunghezza, 150 mm di diametro) è un componente di un cambio meccanico, costituito da un tubo alle cui estremità sono presenti i supporti per il fissaggio agli altri componenti e l'alloggiamento per gli ingranaggi.

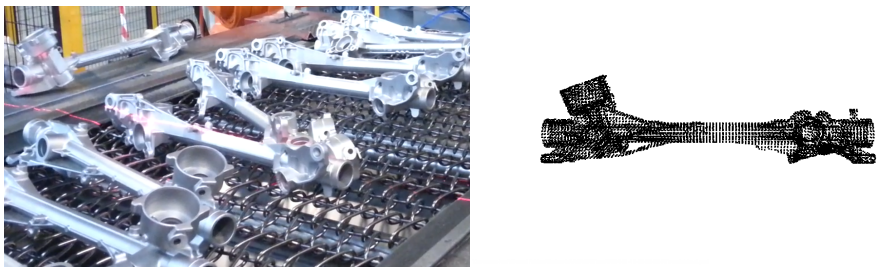

Figura 5.3: A sinistra: prodotti C su un nastro trasportatore. A destra: modello prodotto C.

Per il riconoscimento di questo oggetto si utilizzano otto viste. Questa scelta è dovuta al fatto che la forma dell'oggetto permette la rotazione a 360°, a differenza dell'oggetto precedente più pesante e stabile come forma.

5.1.4 Dataset

Il dataset disponibile per i test è costituito da una serie di scansioni di contenitori contenenti i prodotti. Nel dettaglio, si hanno a disposizione:

- 18 scansioni per Prodotto A
- 40 scansioni per Prodotto B
- 35 scansioni per Prodotto C

5.2 Configurazione

Come già anticipato, i parametri riguardanti la dimensione del *bin* per il calcolo della trasformazione iniziale devono essere scelti in base alla struttura ed alla dimensione del modello che si vuole riconoscere. In questa sezione si riportano i parametri per ciascun prodotto testato. I parametri omessi sono fissati ai valori riportati nella sezione 3.2.2.

L'algoritmo esegue, quindi, un massimo di `num_bin_size` cicli. Per il ciclo i viene utilizzato $\texttt{bin\_size}_i$ calcolato con la seguente formula:

$$bin\_size_i = min\_bin\_size + i \cdot \frac{max\_bin\_size - min\_bin\_size}{num\_bin\_size - 1}$$

Tali valori sono fissati per tutti i test. Infine, tutte le prove sono effettuate con la versione parallela, ad eccezione del test 3 in cui si valuta lo *speedup*.

5.2.1 Configurazione prodotto A

La configurazione per il prodotto A prevede:

- `num_bin_size` = 12
- `min_bin_size` = 40 mm
- `max_bin_size` = 100 mm

5.2.2 Configurazione prodotto B

La configurazione per il prodotto B prevede:

- `num_bin_size` = 8
- `min_bin_size` = 20 mm
- `max_bin_size` = 60 mm

5.2.3 Configurazione prodotto C

La configurazione per il prodotto C prevede:

- `num_bin_size` = 8
- `min_bin_size` = 30 mm
- `max_bin_size` = 70 mm

5.3 Test 1: *rate* di riconoscimento

Il Test 1 valuta il tasso di riconoscimento dell'algoritmo proposto (Algoritmo 2). Per questo esperimento è stato scelto il prodotto A. Questo oggetto è l'unico per cui risulta possibile individuare per ispezione il numero esatto di istanze riconoscibili nella scena (si veda esempio di scansione in figura 5.1).

N. di scene processate	18
N. oggetti totali (TOT)	47
N. oggetti riconosciuti Veri positivi (TP)	32
N. oggetti riconosciuti non corretti Falsi positivi (FP)	0
N. oggetti non riconosciuti Falsi negativi (FN)	15
Tempo di esecuzione medio	7,7s

Tabella 5.1: Risultati Test 1

Dai dati ottenuti è possibile calcolare il *recognition rate* con la formula:

$$RecognitionRate = \frac{TP}{TOT}\%$$

Si ottiene un tasso di riconoscimento del 68%. L'algoritmo 1 ha riconosciuto correttamente 27 prodotti nelle stesse scene.

5.4 Test 2: numero di cicli

Il secondo Test valuta la percentuale media di prodotti riconosciuti in relazione al numero di cicli effettuati dall'algoritmo. Le prove sono state effettuate sui prodotti di tipo B e C e le relative scansioni, presenti nel dataset. Per questo tipo di prodotti è difficile determinare a priori il numero di oggetti riconoscibili nella scena, si prende, quindi, come riferimento il massimo numero di prodotti trovati dagli algoritmi (l'algoritmo 1 viene usato come riferimento).

5.4.1 Configurazione

Per questo test si mantiene la configurazione già riportata per l'algoritmo 2, ad eccezione del parametro `num_bin_size` che, per il prodotto B, viene impostato a 12. L'algoritmo 1 viene eseguito fino a terminazione per avere un riferimento sul totale dei prodotti trovati.

5.4.2 Risultati prodotto B

I risultati e i tempi di esecuzione[1] sono riportati in tabella:

	Algoritmo 1	Algoritmo 2
Oggetti individuati	277	229
Tempo esecuzione medio	20,8 s	16,0 s

Tabella 5.2: Risultati del Test 2 sul prodotto B

L'algoritmo 2 ha riconosciuto l'82,7% dei prodotti rispetto all'algoritmo 1. Nel grafico in figura 5.4 si riporta l'andamento della percentuale di oggetti riconosciuti in relazione al numero di cicli.

Si osserva che i cicli dal nono al dodicesimo apportano solo il 5,2% del totale dei prodotti riconosciuti. Il numero di cicli massimo di default è stato quindi fissato a 8.

5.4.3 Risultati prodotto C

Analogamente si riportano i risultati ottenuti per il prodotto C.

Con questo tipo di prodotto l'algoritmo 2 ha riconosciuto l'8,1% di prodotti in più rispetto alla versione 1.

[1] I tempi di esecuzione del test dipendono dai parametri scelti.

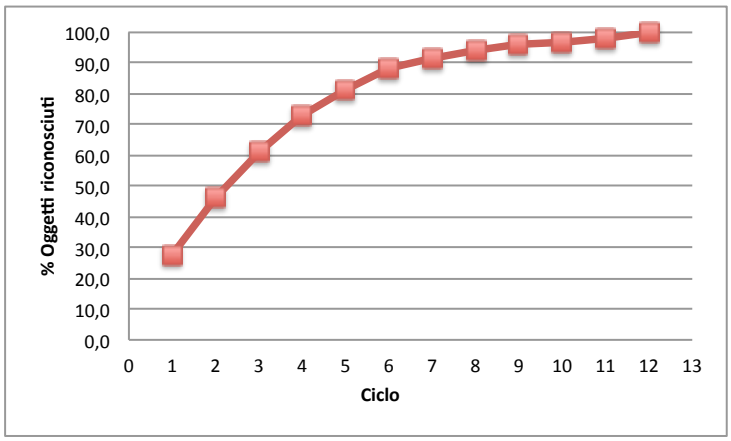

Figura 5.4: Andamento della percentuale di oggetti riconosciuti in relazione al numero di cicli per algoritmo 2 e prodotti di tipo B.

	Algoritmo 1	Algoritmo 2
Oggetti individuati	149	161
Tempo esecuzione medio	20,2 s	16,2 s

Tabella 5.3: Risultati Test 2 sul prodotto C

L'andamento è analogo a quello ottenuto per il prodotto B, l'80% degli oggetti viene riconosciuto entro il quarto/quinto ciclo di esecuzione.

5.5 Test 3: valutazione *speedup*

Con questo test si valuta lo *speedup* ottenuto con la versione parallela dell'algoritmo 2. Per questo test è stata imposta una condizione di terminazione aggiuntiva, ovvero si termina l'elaborazione nel momento in cui sono stati trovati almeno 5 oggetti nella scena.

5.5.1 Configurazione

Il test è stato eseguito con i prodotti di tipo B e C sulle relative scene a disposizione nel dataset. Le prove sono state effettuate su un singolo processore e su 4 processori per la versione parallela.

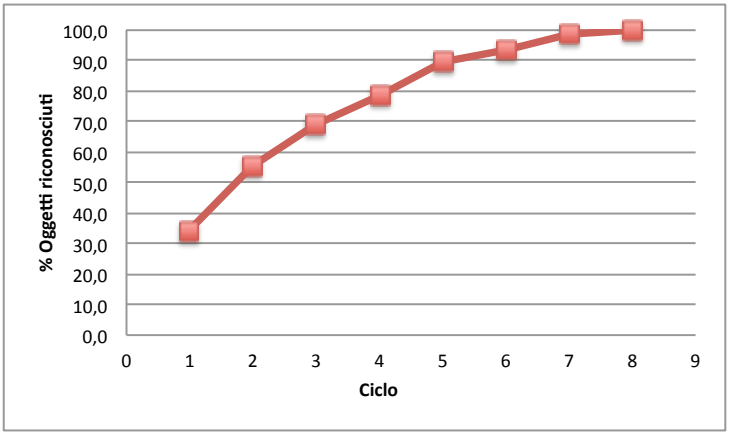

Figura 5.5: Andamento della percentuale di oggetti riconosciuti in relazione al numero di cicli per algoritmo 2 e prodotti di tipo C.

5.5.2 Risultati

I risultati sono riportati in forma grafica in figura 5.6.

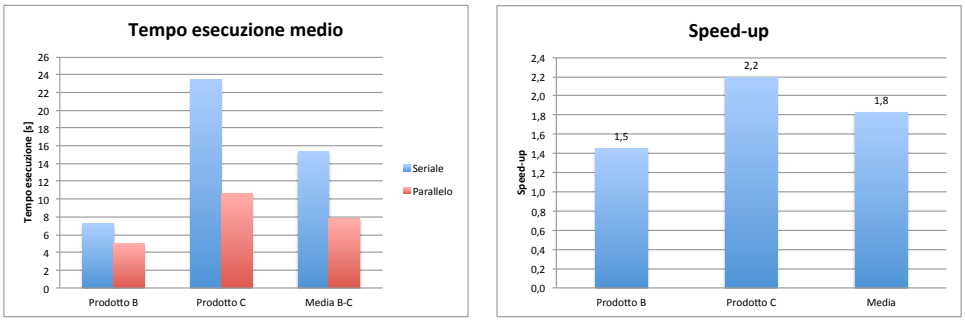

Figura 5.6: Risultati della valutazione dello *speedup*. A sinistra: tempo di esecuzione medio per scena, si confronta la versione seriale e quella parallela. A destra: *speedup* ottenuti testando i prodotti B e C e *speedup* medio.

Si ottiene uno *speedup* maggiore per gli oggetti di tipo C. La notevole differenza è motivata dal fatto che, una delle regioni parallele dell'algoritmo opera a livello di viste dei modelli, per riconoscere i prodotti C si utilizzano 8 viste del modello completo, mentre per il prodotto B se ne utilizzano solamente 2.

5.6 Test 4: confronto

Lo scopo di questo test è confrontare le prestazioni degli algoritmi studiati. In questo caso si imposta la stessa condizione di terminazione usata per valutare lo *speedup*, ovvero l'esecuzione viene terminata quando vengono individuati almeno n match nella scena. Questo esperimento è motivato dal campo di applicazione del sistema studiato. Infatti, non è necessario individuare ad ogni esecuzione tutti i prodotti ma la funzione fondamentale è quella di individuare almeno un oggetto prelevabile nel minor tempo possibile.

5.6.1 Configurazione

Il test è stato eseguito con i prodotti di tipo B e C sulle relative scene a disposizione nel dataset. Le configurazioni utilizzate sono quelle già riportate. Il numero minimo di prodotti n da individuare è stato impostato a $\{1,2,3,4\}$.

5.6.2 Risultati Prodotto B

Per ogni serie di tempi di esecuzione, ottenuti dall'esecuzione di un algoritmo fissando n a un valore, sono stati calcolati media e deviazione standard e distribuzione normale.

I dati ottenuti sono riportati in tabella:

	$n=1$		$n=2$		$n=3$		$n=4$	
Algoritmo	1	2	1	2	1	2	1	2
μ [s]	6,32	3,14	5,93	3,29	7,39	6,09	8,45	7,55
σ^2[s]	1,87	0,45	2,15	0,92	2,43	2,48	2,47	2,89

Tabella 5.4: Risultati del Test 4

In figura 5.7 sono riportati i grafici delle funzioni di densità.

Dai grafici si deduce che l'algoritmo 2 è in grado di individuare con alta probabilità il primo oggetto in un tempo minore rispetto all'algoritmo 1. Nelle prove effettuate con n crescente il tempo di esecuzione medio dei due algoritmi tende ad allinearsi e le due curve arrivano ad avere lo stesso andamento. Una motivazione è data dal fatto che il primo l'algoritmo deve attendere il completamento del clustering prima di iniziare la fase di riconoscimento mentre l'algoritmo 2 inizia subito la fase di riconoscimento.

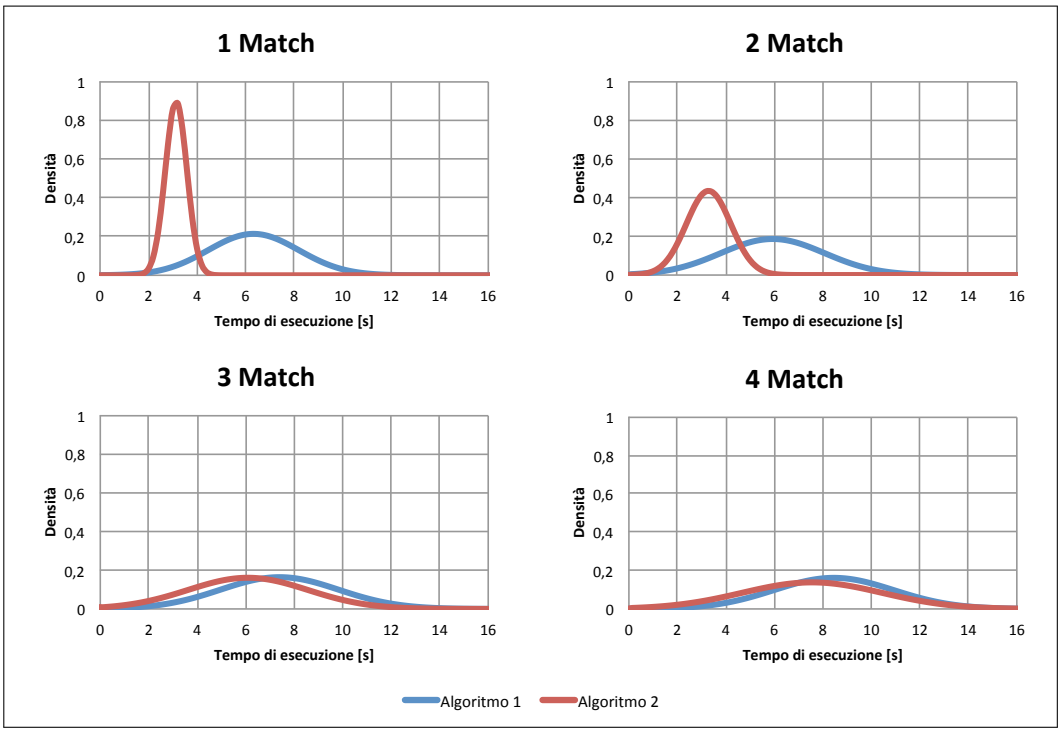

Figura 5.7: Curve gaussiane ottenute dai risultati dei test con il prodotto B. In ogni grafico si confronta la funzione di distribuzione dei tempi dei due algoritmi impostando come condizione di terminazione $n = \{1,2,3,4\}$.

Tuttavia, una volta completato il clustering, l'algoritmo 1 procede più velocemente nella fase di riconoscimento in quanto il calcolo avviene sui singoli cluster che sono composti da molti meno punti rispetto alla scena.

5.6.3 Risultati Prodotto C

I dati ottenuti con questo tipo di prodotti sono riportati in tabella:

	$n=1$		$n=2$		$n=3$		$n=4$	
Algoritmo	1	2	1	2	1	2	1	2
μ [s]	6,53	4,85	8,68	6,57	10,88	9,93	12,74	12,13
σ^2[s]	1,75	1,02	3,08	3,07	4,16	3,69	5,23	4,29

Tabella 5.5: Risultati del Test 4

In figura 5.8 sono riportati i grafici delle funzioni di densità.

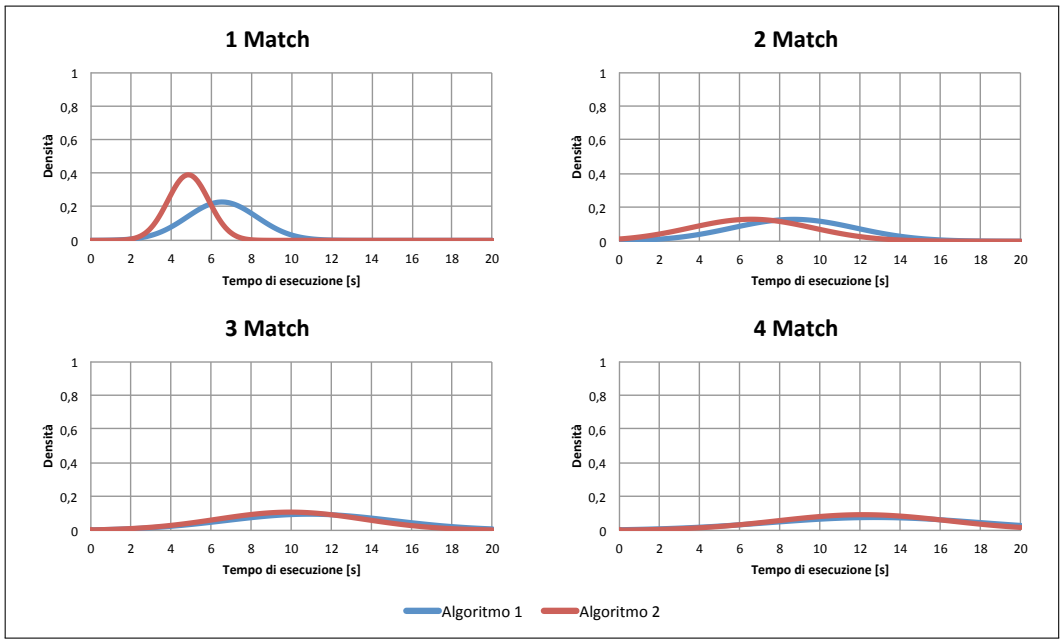

Figura 5.8: Curve gaussiane ottenute dai risultati dei test con il prodotto C. In ogni grafico si confronta la funzione di distribuzione dei tempi dei due algoritmi impostando come condizione di terminazione $n = \{1,2,3,4\}$.

L'andamento dei risultati, anche se meno evidente, è analogo a quello osservato nel test effettuato con gli oggetti di tipo B.

Capitolo 6

Conclusioni

Il presente lavoro ha portato allo sviluppo di un algoritmo di riconoscimento allo scopo di individuare prodotti industriali in point cloud tridimensionali. Dai test effettuati la soluzione proposta, basata su *correspondence grouping*, è una valida alternativa alla soluzione attualmente utilizzata, basata su *clustering* e registrazione.

La scelta di non clusterizzare la scena ha ridotto notevolmente il tempo necessario per fase di preprocessing dell'input. I vantaggi sono evidenti osservando il tempo necessario per individuare il primo oggetto, in cui in alcuni casi si riescono a dimezzare i tempi di esecuzione. Questo risultato è molto rilevante in quanto, ad ogni ciclo di esecuzione, non è fondamentale individuare tutti gli oggetti presenti nella scena ma individuare il primo oggetto prelevabile nel minor tempo possibile.

Per individuare un numero maggiore di oggetti i test non hanno rivelato sostanziali differenze tra le due soluzioni analizzate.

La necessità di confrontare due algoritmi ha portato, inoltre, allo sviluppo di un tool che consente di testare diverse configurazioni di parametri in modo da valutare le prestazioni in relazione agli algoritmi scelti e al tipo di prodotti che devono essere individuati. Il tool sviluppato è stato utilizzato per eseguire tutti i test riportati.

I parametri scelti rendono l'algoritmo affidabile e robusto. Inoltre, questi sono stati fissati cercando un giusto compromesso tra velocità di esecuzione e tasso di riconoscimento. Tutti i parametri sono comunque configurabili per consentire l'adattamento dell'algoritmo ad ogni tipologia di prodotto.

I test effettuati su oggetti diversi per forma e dimensione hanno avuto esito positivo, evidenziando la flessibilità dell'algoritmo e rendendolo integrabile nel sistema correntemente utilizzato.

Si propone come proseguimento del presente lavoro lo sviluppo di un approccio ibrido, in grado di sfruttare i vantaggi di entrambe le soluzioni

studiate. Il sistema potrebbe essere in grado di scegliere l'algoritmo più adatto a seconda dei casi, in base al tipo di prodotti, alla fase di produzione o al risultato desiderato.

Bibliografia

[1] R. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," *Robotics and Automation (ICRA), 2011 ...*, pp. 1–4, 2011.

[2] "Pcl documentation." [Online]. Available: http://pointclouds.org/documentation

[3] OpenMP Architecture Review Board, "OpenMP application program interface version 4.0," 2014. [Online]. Available: http://openmp.org/wp/openmp-specifications/

[4] "The PCL registration api." [Online]. Available: http://pointclouds.org/documentation/tutorials/registration_api.php

[5] a. S. Mian, M. Bennamoun, and R. a. Owens, "A Novel Representation and Feature Matching Algorithm for Automatic Pairwise Registration of Range Images," *International Journal of Computer Vision*, vol. 66, no. 1, pp. 19–40, Jan. 2006.

[6] S. Squizzato and M. Menegatti, "Robot bin picking: 3D pose retrival based on Point Cloud Library," Master's thesis, Università degli Studi di Padova, 2012.

[7] D. Lowe, "Local feature view clustering for 3D object recognition," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–682–I–688, 2001.

[8] L. Alexandre, "3D descriptors for object and category recognition: a comparative evaluation," *Workshop on Color-Depth Camera Fusion in Robotics ...*, 2012.

[9] R. B. Rusu, N. Blodow, and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D registration," *2009 IEEE International Conference on Robotics and Automation*, pp. 3212–3217, 2009.

[10] R. Rusu, N. Blodow, Z. Marton, and M. Beetz, "Aligning point cloud views using persistent feature histograms," *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3384–3391, Sep. 2008.

[11] F. Tombari, S. Salti, and L. Di Stefano, "Unique Signatures of Histograms for Local Surface Description," *Computer Vision – ECCV 2010*, vol. 6313, pp. 356–369, 2010.

[12] A. Petrelli and L. Di Stefano, "On the repeatability of the local reference frame for partial shape matching," *2011 International Conference on Computer Vision*, pp. 2244–2251, Nov. 2011.

[13] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Application (VISSAPP09)*. INSTICC Press, 2009, pp. 331–340.

[14] D. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[15] F. Tombari and L. Di Stefano, "Object Recognition in 3D Scenes with Occlusions and Clutter by Hough Voting," *2010 Fourth Pacific-Rim Symposium on Image and Video Technology*, pp. 349–355, Nov. 2010.

[16] F. Tombari and L. D. Stefano, "Hough Voting for 3D Object Recognition under Occlusion and Clutter," *IPSJ Transactions on Computer Vision and Applications*, vol. 4, pp. 20–29, 2012.