UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Meccanica

Sistema di rilievo ostacoli per applicazione in robot collaborativi

Obstacle detection system for collaborative robot application

Candidato:
Maria Teresa Calcagni

Relatore:
Prof. Paolo Castellini

Anno Accademico 2019-2020

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Meccanica

Sistema di rilievo ostacoli per applicazione in robot collaborativi

Obstacle detection system for collaborative robot application

Candidato:
Maria Teresa Calcagni

Relatore:
Prof. Paolo Castellini

Anno Accademico 2019-2020

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Meccanica
Via Brecce Bianche – 60131 Ancona (AN), Italy

"È facile avere un'idea complicata.
La cosa davvero molto, molto
complicata è avere un'idea semplice"
Carver Mead

Sommario

La principale trattazione di questo progetto di ricerca è la collaborazione uomo-macchina resa possibile dallo studio e dalla rielaborazione di dati acquisiti da sensori di visione, ma è indispensabile fare un passo indietro per comprendere qual è l'evoluzione ideologica che ha portato alla necessità di questo nuovo concetto, in avanzamento ogni giorno di più. L'umanità, fin dai tempi più remoti, ha sfruttato il suo ingegno per fronteggiare le faticose (e non solo) attività quotidiane inventando macchine, strategie e metodi di semplificazione. Il progresso tecnologico ha consentito di portare avanti questa visione potenziandola ulteriormente: si è passati ad esempio, dalla progettazione e realizzazione di mezzi di trasporto alla costruzione di robot elettrici adatti ad ogni applicazione, da domestiche a industriali. Nel caso particolare delle macchine automatiche, il primo androide funzionante risale al diciottesimo secolo quando furono realizzati dei manichini mobili. [1] In realtà, robot propriamente detti, furono ideati dopo gli anni Quaranta (1940) con l'avvento del computer. Negli anni Sessanta (1966), in USA, Shakey fu progettato dai ricercatori per impilare verticalmente carichi: era dotato di videocamera che trasferiva dati ad un computer e li elaborava.[2] Negli anni successivi l'attenzione si è spostata sullo studio dell'intelligenza artificiale: è stato ideato un robot in grado di vincere a scacchi contro il campione mondiale della stessa disciplina. [3] L'attuale tendenza degli studi moderni è quella di combinare le due tipologie di ricerca: da una parte la possibilità di controllare fluidamente i movimenti del meccanismo e dall'altra l'incremento della capacità di calcolo, in modo da realizzare quasi-macchine che percepiscano la realtà circostante così da non lesionare persone e cose. Le esigenze sono cambiate nel tempo e hanno permesso un'ulteriore agevolazione della routine grazie alla curiosità e all'innovazione digitale: il monitoraggio dei sistemi automatici durante lo svolgimento di compiti è quello che interessa principalmente l'argomentazione qui riportata in quanto contribuisce a verificare la sicurezza dello spazio d'interesse. I processi industriali sono caratterizzati da un'elevata produttività a discapito di pericolo causato dai macchinari in costante movimento, nel momento in cui una macchina viene montata all'interno di una linea produttiva ne consegue il rispetto della norma di conformità europea(Testo Unico sulla Sicurezza) e l'abilità da parte degli operatori di utilizzare le specifiche macchine, in quanto consapevoli di quali sono i rischi, espressi dal costruttore, che l'uso di tale funzionalità comporti e come evitarli. Parallelamente, un'analisi minuziosa sulla sicurezza avviene durante la scelta della configurazione dello spazio di lavoro di un robot per salvaguardare gli operatori e la produzione. Il robot, essendo considerato dalla normativa una quasi macchina, una volta equipaggiato, è una macchina a tutti gli

effetti, ma la sua natura non permette di sottostare a determinate norme di sicurezza prestabilite: ricopre funzionalità talmente vaste che non si possono generalizzare criteri di sicurezza per una qualsiasi macchina automatica. Infatti, in base al tipo di applicazione, effettuata all'interno della cella di lavoro, vanno presi in considerazione tutti i rischi possibili e va assegnato un grado di pericolosità più o meno elevato, che guida alla scelta di determinate infrastrutture piuttosto che altre. La maggior parte delle celle di lavoro è caratterizzata da pareti protettive che non consentono l'accesso agli operatori all'interno durante il funzionamento, nell'eventualità in cui qualcuno entri c'è il blocco istantaneo del robot ma ciò non permette di ripartire da dove si era lasciata la produzione, l'arresto delle operazioni avviene tramite presenza di sensori di pressione o infrarossi all'ingresso, per intercettare o meno il passaggio di persone. Con l'avvento del co-bot, robot collaborativo, il rischio durante la lavorazione è ridotto di molto in quanto il robot è molto più fluido e controllato nei movimenti questo continuo controllo e capacità di stop e ripresa delle operazioni permette di non dover sempre utilizzare grate di protezione, in quanto gli operatori possono tranquillamente lavorare in prossimità del robot stesso, ovviamente è necessario lo sviluppo di un software che monitori in tempo reale l'area di interesse. Lo studio qui presentato, si basa completamente sulla realizzazione di un programma che trasformi un normale robot in un co-bot in modo tale che lavori in prossimità dell'operatore, quindi sprovvisto di grate e che sia controllato da un sistema di visione. Ciò è stato realizzato partendo da un normale braccio meccanico, inizialmente sprovvisto di sensori, e movimentato da teach-pendant, dispositivo di comando che permette lo spostamento del robot. L'obbiettivo è quello di rendere il robot 'cosciente' di ciò che lo circonda tramite elaborazione e manipolazione dei dati acquisiti da sensori visivi esterni, così da rilevare in tempo reale informazioni riguardanti il campo di lavoro d'interesse, e vengano successivamente trasferite al robot, il quale si comporterà di conseguenza fermandosi o rallentando in presenza di un eventuale ostacolo captato e analizzato da questi 'occhi' esterni. Rendere collaborativo il robot a disposizione è stato possibile tramite lo sviluppo di un software che permetta visualizzazione dello spazio attorno allo stesso in maniera continua. Questo programma è stato redatto considerando i passaggi di seguito riportati.

Prima fase: preparazione spazio di lavoro per garantire la costante visione al robot, tramite utilizzo di due sensori di visione, opportunamente montati attorno all'area di interesse.

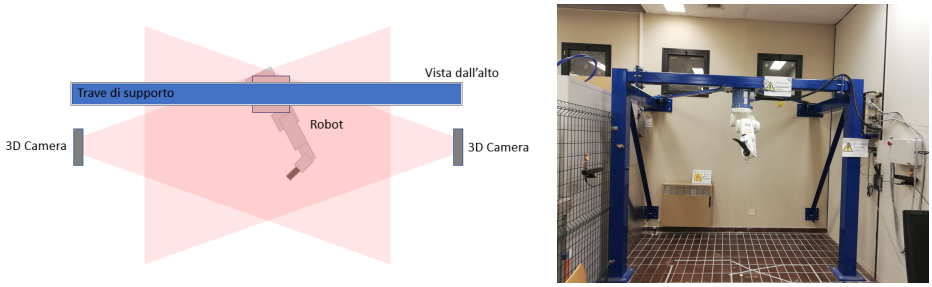
Seconda fase: sviluppo della prima parte del codice, che prevede l'estrapolazione dei parametri necessari per il corretto funzionamento del sistema, tramite funzioni di calibrazione e allineamento delle acquisizioni, in modo da assicurare un sistema di riferimento comune durante le successive elaborazioni e definizione del campo visivo.

Terza fase: utilizzando i parametri precedentemente calcolati, è stato ideato un algoritmo che permettesse di discernere in maniera piuttosto chiara quale fosse il numero di target e distinguerli dimensionalmente, in modo che il robot abbia davvero degli "occhi" che gli permettano di evitare la collisione con eventuali oggetti o persone nello spazio. Per ottenere i dati in maniera continua, tutto il programma è stato compattato e dotato di un ciclo while, che assicura un'acquisizione dell'area d'interesse in tempo reale.

Indice

Metodi e strumenti

La zona utilizzata per il progetto è uno spazio delimitato strutturalmente attorno a un braccio meccanico, all'interno del laboratorio di facoltà: i sensori di visione sono stati montati rispettivamente sul lato destro e sinistro del robot, che è incernierato ad un ponte d'acciaio e quindi sollevato da terra. All'estremità dello stesso, sono montati i due dispositivi che consentono il monitoraggio dello spazio di lavoro, come è illustrato in Figura 0.1a e in Figura 0.1b rispettivamente dall'alto e frontalmente,e comunicano dati, successivamente elaborati.



(a) Vista dall'alto. (b) Vista frontale.

Figura 0.1: Area di lavoro.

Di seguito, saranno spiegati in dettaglio gli strumenti e le metodologie usate per lo sviluppo dell'algoritmo definitivo che permette al robot di diventare collaborativo.

Capitolo 1

Dispositivi

1.1 Robot e target

Lo scopo di questo studio è la realizzazione di un sistema di controllo dell'area di movimentazione di un robot: il braccio meccanico a disposizione è installato centralmente sull'estremità superiore di un ponte rettangolare d'acciaio ed è rivolto verso il basso, è necessario quindi rilevare gli elementi in avvicinamento in tempo reale in modo che possa fermarsi o rallentare per evitare eventuali impatti. Il robot è stato determinante per la scelta del sistema di riferimento utilizzato e adattato ai sensori, in modo tale che i dati acquisiti potessero combaciare. L'origine degli assi è disposto sulla cerniera che vincola il braccio al supporto, l'asse z si estende lungo il robot ed è positivo verso il soffitto, l'asse y è positivo verso la kinect di sinistra, vista dalla postazione di lavoro, l'asse x è positivo verso il robot rispetto alla visuale dalla postazione di lavoro, la visuale della postazione di lavoro coincide con l'immagine sottostante (Figura 1.1). Il braccio meccanico usato è Denso, robot Kuka , con sei assi, antropomorfo, la movimentazione è stata fatta tramite teach pendant, uno strumento di comando manuale che ha permesso di raggiungere le configurazioni necessarie per le prove.

Figura 1.1: Robot Denso e sistema di riferimento.

Tipologie diverse di target sono state usate durante i test in modo da rendere le prove caratterizzanti, come si vedrà in dettaglio più avanti. Per la calibrazione è stata usata una lastra piana verticale, sorretta da supporti in metallo(Figura 1.2a), per visualizzare una figura umana in assenza di persone è stato usato un manichino (Figura 1.2b), per prove di matching un cartonato della lettera 'F' posta su un supporto (Figura 1.2c), persone in movimento sono state introdotte durante le prove in real time. La diversificazione dei target è stata necessaria per comprendere l'effettiva visibilità delle nuvole di punti nello spazio: ciò ha permesso di visualizzare e riconoscere le sagome introdotte, nella loro integrità in quanto l'acquisizione tramite entrambi i sensori in contemporanea ne garantiva l'effettiva profondità e dimensione dell'oggetto nello spazio, cosa non banale considerando la perdita di effettiva densità dell'oggetto durante l'acquisizione di uno solo dei due dispositivi.

(a) Lastra. (b) Manichino. (c) F.

Figura 1.2: Alcune tipologie di target.

1.2 Sensori di visione

I sensori di visione utilizzati sono Kinect: il dispositivo posizionato a destra, è Kinect per Windows, denominato "uno", l'altro, sulla sinistra, è per Xbox ed è denominato "due", le denominazioni sono state necessarie nello sviluppo del programma per un rapido riconoscimento dei due. L'unico elemento distintivo tra i due strumenti riguarda l'ampliamento del campo visivo tramite comando "near mode" che nel "due" non è previsto, ma a parte questo, caratteristiche tecniche e stato dell'arte sono le stesse. I sensori sono provvisti dei componenti di seguito elencati e visualizzati in Figura 1.3 ma i più utilizzati nello studio fatto sono i primi tre della lista:

- Telecamera a colori.
- Proiettore a infrarossi (IR).

- Sensore di profondità a infrarossi (IR).
- Microfono.
- Motore di posizionamento Kinect.
- LED.

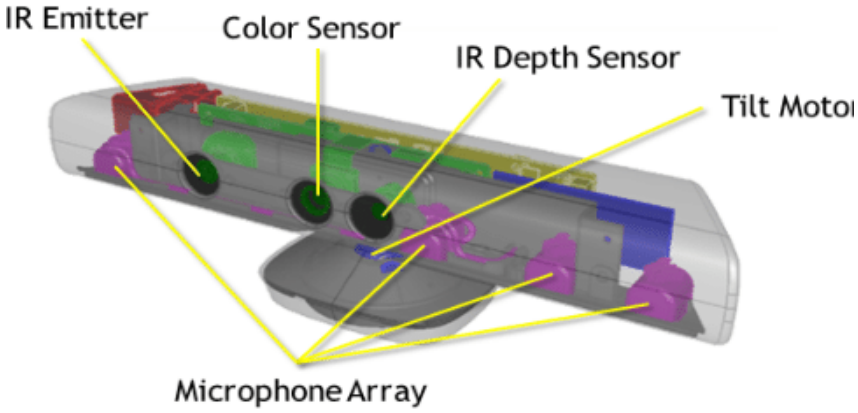
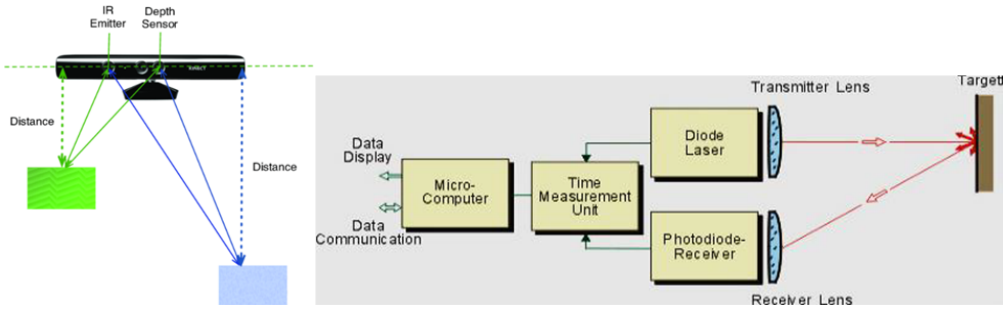


Figura 1.3: Componenti.

La tecnologia delle Kinect differisce in base alla versione dei sensori. I dispositivi disponibili sono provvisti di versione uno (v1), basata sul light coding. Di solito, il calcolo delle distanze nei sensori di profondità si fonda su un tipo di triangolazione, calcolo delle distanze tra punti (pixel in questo caso) in base a triangoli, ed è facilitato da due telecamere; ciò viene semplificato nei sensori Kinect tramite il metodo della luce strutturata che permette di azionare una luce costante e invisibile che, colpendo un filtro, si disperde andando a formare un pattern di piccoli punti nello spazio, il pattern è predefinito e si deforma in base all'oggetto su cui si poggia, questo viene irradiato dall' unico emettitore IR, e percepito dal sensore di profondità IR, così che la distanza viene calcolata a partire dalla luce riflessa degli infrarossi. L'immagine che viene ricevuta indietro nel ricettore IR è poi decodificata dallo stesso Kinect confrontandolo con un'immagine precedentemente registrata ad una profondità costante nota. I dati di profondità vengono mandati tramite USB a un altro dispositivo (computer solitamente) per altri processamenti. Viene usato l'IR in quanto la luce strutturata non può interferire con altri sensori (Figura 1.4a). Il sensore Kinect predisposto con la versione due (v2) utilizza una tecnologia diversa chiamata tempo di volo (in inglese, time of flight, ToF) che consiste nel misurare il tempo che intercorre tra proiezione di segnali luminosi, da parte di un diodo laser che trasforma in luce una sequenza di impulsi, a ricezione da parte di un fotodiodo che lo trasforma in segnale elettrico, tramite un contatore temporale che ne calcola la distanza dell'oggetto nello spazio in base al tempo che intercorre tra emissione del segnale e ricezione, inoltre ne determinano anche l'intensità della luce (Figura 1.4b).



(a) Light coding. (b) Time of flight.

Figura 1.4: Confronto tecnologico.

Le differenze tra le due versioni sono riportate nell' illustrazione sottostante (Figura 1.5). [4]

Feature	Kinect v1	Kinect v2
Depth Sensing Technology	Triangulation with structured light	Time of flight
Color Image Resolution	640x480 30fps 1280x960 12fps	1920x1080 30fps (12fps low light)
IR Image Resolution	640x480 30fps	512x424 30fps
Depth Sensing Resolution	640x480 30fps 320x240 30fps 80x60 30fps	512x424 30fps
Field of View	43° vertical 57° horizontal	> 43° vertical 70° horizontal [60]
Depth Sensing Range	0.4m - 3m (near mode) 0.8m - 4m (default mode)	0.5m - 4.5m Up to 8m without skeletonization
Skeleton Tracking (with full skeleton)	Up to 2 subjects 20 joints per skeleton	Up to 6 subjects 25 joints per skeleton
Built-in Gestures	None	Hand state (open, close, lasso) Hand pointer controls; lean
Unity Support	Third party	Yes
Face APIs	Basic	Extended massively
Runtime Design	Can run multiple Kinect sensors per computer; One app per Kinect	At most one Kinect per computer; Multiple apps share same Kinect
Windows Store	Cannot publish to	Yes

Figura 1.5: Confronto caratteristiche.

Capitolo 2

Caratterizzazione software

Inizializzazione

Il progetto prevede la manipolazione dei dati raccolti dai sensori Kinect per ottenere il riconoscimento degli oggetti nello spazio, per farlo è necessario che il software Matlab, ambiente per il calcolo numerico, sia aggiornato alla versione R2020a. Di seguito sono mostrati i passaggi necessari per garantire il controllo dei due dispositivi da computer. Innanzitutto, sono stati installati i driver SDK (Software Development Kit) versione 1.8, che consentono di creare programmi Kinect su Windows, compatibile con la versione uno dei dispositivi. Skanect è un programma utilizzato per la visualizzazione di ciò che effettivamente rilevano sia la telecamera RGB sia la telecamera di depth ed è necessario per ispezionare il corretto funzionamento di entrambi sensori: per le verifiche con tale programma è necessario che i dispositivi siano collegati non in contemporanea al computer. Entrambi i programmi sono facilmente rintracciabili e scaricabili dal motore di ricerca a propria disposizione. Successivamente sono stati installati dei toolbox di Matlab, Image Acquisition Toolbox, che permettono l'acquisizione di immagini e il controllo dei dati acquisiti, banalmente tramite il programma stesso: nella finestra 'Apps' selezionare 'Install App', comparirà la finestra di esplorazione degli Add-on di Matlab.

2.1 Parametri e delimitazioni

2.1.1 Coordinate carteisiane

I parametri ricavati dalla calibrazione hanno la funzionalità di rendere univoco il punto di vista tra i sensori ma soprattutto associare l'unità di misura dei mm, che stabilisce la distanza reale di un punto nello spazio, ai pixel che descrivono la dimensione dell'immagine acquisita dai dispositivi. In particolare, è stata effettuata la taratura statica di un target, nello specifico la lastra sopra illustrata, all'interno dello spazio di lavoro e sono stati definiti azimut ed elevazione, tramite la funzione 'filtratopol3', relativi al calcolo del rapporto tra pixel e mm con la funzione di taratura che ha permesso di ricavare le coordinate x,y,z per ognuno dei due sensori. Di seguito il codice di acquisizione con la funzione di calibrazione. Nelle immagini sottostanti, è possibile visualizzare nel dettaglio i passaggi precedentemente sintetizzati: nella

prima parte di codice si definiscono i range di acquisizione, rispettivamente in mm e in pixels, vengono importate le curve di taratura (Figura 2.1). Alla fine di Figura 2.1 e all'inizio di Figura 2.2 si attua una manipolazione delle righe e delle colonne della matrice di pixel per definire azimut ed elevazione, input necessari per la computazione delle coordinate cartesiane tramite la funzione 'filtratopol3' (Figura 2.4), nelle righe successive. Nel mezzo di Figura 2.2, avviene l'acquisizione di profondità da parte dei due dispositivi- in questo caso, si acquisisce l'immagine tramite il sensore di distanza con la sua massima risoluzione, più alta è la risoluzione maggiori sono i punti acquisiti nello spazio. Nella parte finale di Figura 2.2 e iniziale di Figura 2.3 avviene la calibrazione vera e propria che rileva le coordinate cartesiane delle due parti.

```
close all
clc
%% Calibrazione
%%
% range acquisizione in mm
minimo=1000;
massimo=2000;
% righe e colonne (pixel)
r=480;
c=640;
%centro assi
tr_x_centro=[-621 -1400 -650];
%% caricamento della curva z index
d_zz=load('curva_z_uk.mat', 'fitresult');
d_z=d_zz.fitresult;
clear d_zz
%% caricamento della curva di taratura dei pixel in mm
d_pp=load('curva_taratura.mat','fitresult');
d_p=d_pp.fitresult;
clear d_pp

rr=1:r;
rr=squeeze(rr-floor(r/2));
az=repmat(rr',1,c);
az=reshape(az, c*r,1);
```

Figura 2.1: Prima parte del codice.

```
cc=1:c;
cc=squeeze(cc-floor(c/2));
ele=repmat(cc,r,1);
ele=reshape(ele, c*r,1);

info = imaqhwinfo('kinect');
%% Kinect 1
vid1 = videoinput('kinect', 2,'Depth_640x480','FramesPerTrigger',1);
src1 = getselectedsource(vid1);
triggerconfig(vid1,'manual');
start(vid1);
%% Kinect2
vid2 = videoinput('kinect', 4,'Depth_640x480');
src2 = getselectedsource(vid2);
triggerconfig(vid2,'manual');
start(vid2);

pause(2)
%% nuvola 1
%% acquisiamo i punti dalla prima camera
c1 = getsnapshot(vid1);
c1=flip(c1,2);
[x1,z1,y1]=filtrato_pol3(c1,az,ele,minimo,massimo,d_z,d_p);
X1=[x1 y1 -z1]+tr_x_centro;
```

Figura 2.2: Seconda parte del codice.

```
%% nuvola 2
%% acquisiamo i punti dalla seconda camera
c2 = getsnapshot(vid2);
c2=flip(c2,2);
[x2,z2,y2]=filtrato_pol3(c2,az,ele,minimo,massimo,d_z,d_p);
X2=[x2 y2 -z2]+tr_x_centro;
```

Figura 2.3: Terza parte del codice.

In questo progetto l'obbiettivo è stabilire la distanza di uno specifico punto nello spazio, in particolare localizzare una nuvola di punti - la telecamera di depth traduce l'oggetto presente nello spazio considerato come un elevato numero di punti che descrivono la sagoma dello stesso , ciò viene definito 'nuvola di punti' o 'point cloud', in inglese - e rilevare che sia più o meno vicina al robot, per farlo si lavorerà con la telecamera di depth dei sensori per ottenere effettiva profondità spaziale. Quindi, la conversione è stata necessaria per uniformare i dati di uscita delle due kinect al sistema di misura utilizzato e per evitare la distorsione causata dall'angolo d'apertura dell'obbiettivo dei due sensori. L'acquisizione tramite Kinect è simile a quella che avviene con altri strumenti di visione, con la differenza che ha DeviceID diversi: il sensore di colore mostra immagini a colori di dati. Il sensore di 'depth' mostra risultati di distanza e lo scheletro degli oggetti. Inoltre, i dati individuati possono

essere acquisiti in tempo reale e su quattro fronti: video a colori, informazioni di distanza in pixels, la tracciabilità dello scheletro umano (tramite sensore di depth) e l'audio. Il dispositivo Kinect può tracciare fino a sei persone di cui, completo tracciamento di due persone, tracciamento di posizione anche sulle altre 4.

```
function [x,y,depth_sel]=filtrato_pol3(c,az,ele,minimo,massimo,d_z,d_p)
depth=reshape(c,size(c,1)*size(c,2),1);

ind=find(depth>minimo & depth<massimo);
depth_sel=depth(ind);

ele=ele(ind);
az=az(ind);

depth_sel=double(depth_sel);


z=d_z(depth_sel);


x=ele./d_p(z);
y=az./d_p(z);
```

Figura 2.4: Funzione di calibrazione.

2.1.2 Campo visivo

L'effettiva estensione del campo visivo dei sensori è stata denotata eseguendo delle prove con target antropomorfi in modo da comprendere la sagoma reale e l'integrità dello stesso nello spazio. I dispositivi utilizzati, per quanto caratterizzati dalla stessa tecnologia, divergono per la rilevazione del campo visivo: Kinect per Windows acquisisce ad una distanza di 80 cm, di default, dal sensore stesso che può essere ridotta a 57 cm tramite modalità 'Near' (Figura 2.5), mentre Kinect per Xbox individua gli oggetti a una distanza di 120 cm dal proprio obbiettivo, la modalità precedentemente citata non è utilizzabile in questo caso nonostante sia stata aggiunta per verificarne l'eventuale utilizzo,ma di conseguenza commentata. Le acquisizioni

```
%% Kinect_destra(1)
vid1 = videoinput('kinect', 2,'Depth_640x480','FramesPerTrigger',1);
src1 = getselectedsource(vid1);
src1.DepthMode='Near';
triggerconfig(vid1,'manual');
start(vid1);
```

```
%% Kinect_sinistra(2)
vid2 = videoinput('kinect', 4,'Depth_640x480');
src2 = getselectedsource(vid2);
% src2.DepthMode='Near';

triggerconfig(vid2,'manual');
```

Figura 2.5: Modalità 'Near'.

fatte con lo script di Matlab (in Figura 2.6) che si aggiunge a (Figura 2.5) hanno permesso di determinare quali fossero effettivamente le distanze coperte da ciascun dispositivo: variando la posizione del target dai sensori di visione è stata delimitata l'area d'azione individuando se l'oggetto d'interesse rientrasse nell'uno e/o nell'altro campo visivo. Per visualizzare i target ad ogni postazione è stato usato il comando di 'plot' riassunto nella funzione 'plot nuvola colore', così da diversificare in maniera evidente la nuvola di punti visualizzata da un sensore rispetto all'altro.

La nuvola verde rappresenta il target percepito dal sensore 'uno' e indicato dal colore 'g' nel codice('g' sta per green), il manichino viene visto frontalmente, la nuvola rossa rappresenta il target osservato dal sensore 'due', identificato 'r' nel codice('r' sta per red), il manichino viene osservato di schiena infatti si intravedono i dettagli della sedia su cui è poggiato (Figura 1.2b), come si può confrontare in Figura 2.7. Il fatto che i due insiemi di punti non siano coincidenti è dovuto alla percezione del sistema di riferimento globale da parte dei dispositivi, questa problematica sarà affrontata nel paragrafo successivo. La regione visualizzata dai due sensori è stata rappresentata in Figura 2.8 sulla base dei risultati visivi estrapolati dal codice: l'intersezione delle linee continue verdi e rosse rappresenta la completa copertura visiva dello spazio, che coincide con la completa visualizzazione delle due nuvole che presentano gli stessi

```
%% nuvola 1
%% acquisizione punti dalla prima camera
c1 = getsnapshot(vid1);
c1=flip(c1,2);
[x1,z1,y1]=filtrato_pol3(c1,az,ele,minimo,massimo,d_z,d_p);
XX1=[x1 y1 -z1]+tr_x_centro;
%% nuvola 2
%% acquisizione punti dalla seconda camera
c2 = getsnapshot(vid2);
c2=flip(c2,2);
[x2,z2,y2]=filtrato_pol3(c2,az,ele,minimo,massimo,d_z,d_p);
XX2=[x2 y2 -z2]+tr_x_centro;

%% VIsualizzazione nuvole
X=[XX1' XX2']';
plot_nuvola_colore(XX1,'g'), hold on
axis equal
plot_nuvola_colore(XX2,'r'),hold off
```

Figura 2.6: Codice acquisizione nuvole.

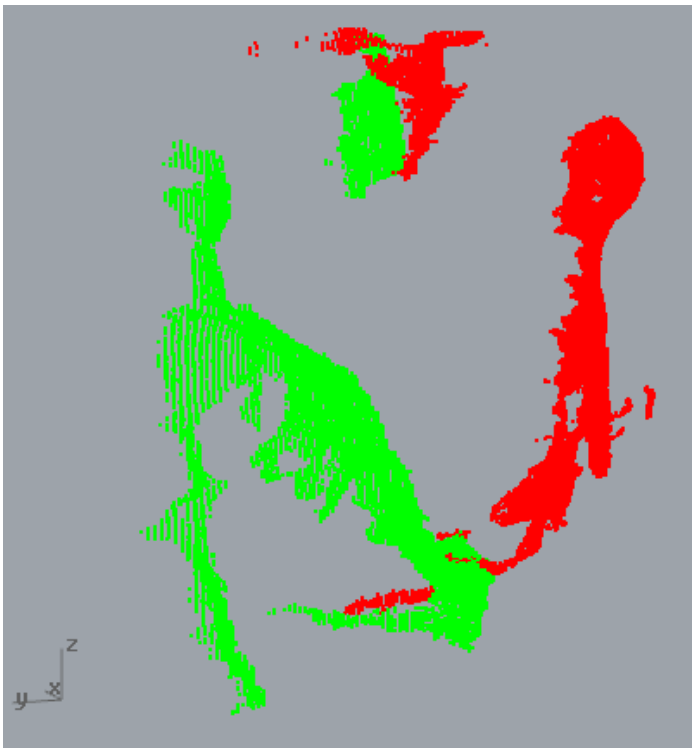


Figura 2.7: Nuvole acquisite.

colori nel grafico del codice. Quando una delle due nuvole non è percepita da uno dei due sensori, non rientra in una delle due aree di visualizzazione quindi non si ottiene la perfetta integrità del target.

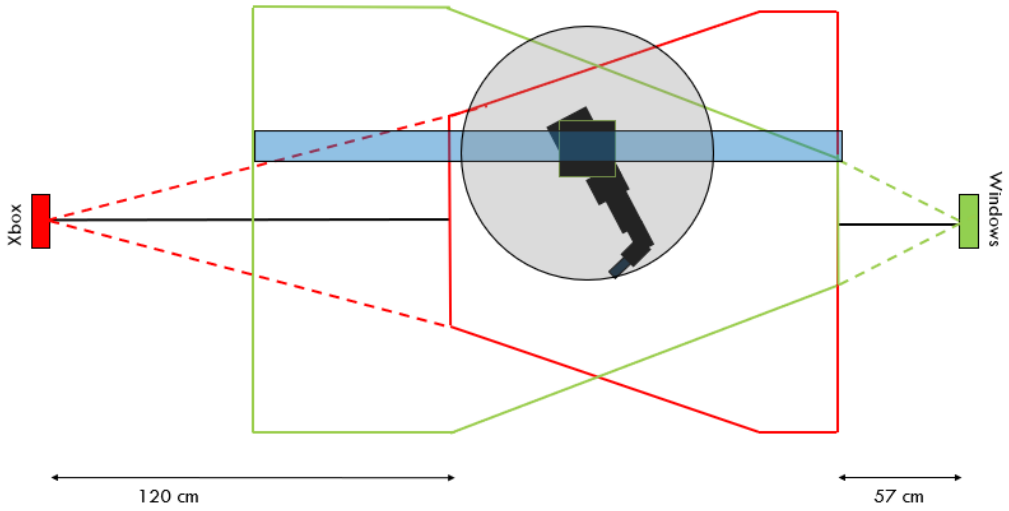


Figura 2.8: Rappresentazione campo visivo.

L'obbiettivo è quello di conoscere qual è perfettamente l'area coperta da entrambi i sensori in modo tale da ridurre i punti ciechi al fine di permettere al robot di conoscere esattamente la posizione del target con cui non deve impattare. Uno sviluppo futuro attuabile per questo campo visivo consiste nel cambiare disposizione delle telecamere: ponendole a distanza maggiore l'una dall'altra oppure modificare eventualmente l'angolazione tra i due dispositivi e/o aggiungerne uno o più di uno, in modo tale da ridurre ulteriormente i punti poco identificabili in prossimità del robot e ampliarne il raggio di visione.

2.1.3 Matrice di rototraslazione

L'unificazione del sistema di riferimento per i due sensori va associata ad una corrispondenza delle due nuvole di punti acquisite, in modo da visualizzare l'oggetto d'interesse nella sua interezza. La coincidenza delle due acquisizioni si raggiunge tramite una matrice di rototraslazione, che come suggerisce la parola stessa, ruota e trasla una delle due nuvole rispetto all'altra. Tramite Matlab è stato sviluppato un algoritmo che ha come obbiettivo il match delle due nuvole utilizzando una funzione che attua automaticamente la rototraslazione per raggiungere la conformità desiderata, la matrice che permette ciò, può essere salvata e riutilizzata nel software finale. Per il calcolo della matrice, viene usato un target d'acquisire così da salvare la matrice di matching in modo da assicurare l'allineamento per qualsiasi altra applicazione futura. Questa operazione garantisce anche l'implementazione del progetto con più dispositivi di visione in quanto basta correlare tramite questo codice il dispositivo d'interesse al dispositivo di riferimento in modo da ottenere la matrice di matching e utilizzarla durante le esecuzioni successive. Di seguito verranno mostrati i passaggi per il calcolo e il salvataggio della matrice di rototraslazione che verrà applicata nel programma finale. Inizialmente sono stati usati target simmetrici per l'estrapolazione della matrice di rototraslazione ma non fornivano l'allineamento corretto poiché le nuvole potevano realizzare un matching su più fronti ma spesso non era con quello reale o addirittura le due nuvole risultavano disallineate. Per agevolare Matlab nel calcolo della matrice, è stata intagliata una sagoma di cartone a forma di 'F' (Figura 2.9) in quanto sia una lettera di natura asimmetrica e di facile realizzazione, inoltre è stata attuata una prerotazione di 180° attorno all'asse z così che l'accoppiamento risultasse immediato. Una volta assicurata la coincidenza dell'insieme di punti sono

Figura 2.9: Cartonato F.

state salvate sia la matrice di rototraslazione calcolata a seguito dell'accostamento delle nuvole sia la matrice di prerotazione. Le matrici salvate sono necessarie per garantire l'allineamento di qualsiasi target che entrerà all'interno dello spazio di lavoro durante l'esecuzione del programma finale. Di seguito, compare il codice

di allineamento delle nuvole che descrive quanto descritto sopra -le nuvole sono state acquisite e salvate tramite il programma descritto nel precedente paragrafo. Inizialmente (Figura 2.10) vengono caricate le nuvole ed elaborate eliminando i punti singoli e distanti, successivamente si definiscono la nuvola che si movimenterà, in questo caso quella vista dal dispositivo '2' e quella che invece rimarrà ferma, acquisita da '1'. Alla fine della prima immagine si inserisce la possibilità di mettere in un grafico l'acquisizione dei target prima del calcolo della matrice di rototraslazione, per ottenere la Figura 2.12. In Figura 2.11, si definisce la matrice di prerotazione e sulla base di questa si calcola la matrice di rototraslazione. Dato il grafico impostato a fine codice, si vedrà il matching del target considerato in Figura 2.13, affinché si possa visualizzare ampiamente il confronto e il risultato ambito.

```
close all
clc
%%
cc=load('F1_z.mat');
f=cc.XX1;clear cc
ff=f(f(:,3)>-700,:); %F
f=[ff(:,1) ff(:,2) ff(:,3)];
clear ff
cc=load('F2_z.mat');
m=cc.XX2;clear cc
mm=m(m(:,3)>-700,:);
m=[mm(:,1) mm(:,2) mm(:,3)];

fixed=pointCloud(f);
moving=pointCloud(m);
movingDownsampled =moving;
fixedDownsampled =fixed;
%%
% Display the downsampled point clouds before registration.
figure
pcshowpair(movingDownsampled,fixedDownsampled,'MarkerSize',50)
xlabel('X')
ylabel('Y')
zlabel('Z')
title('Point clouds before registration')
```

Figura 2.10: Prima parte del codice: matrice rototraslazione

Utilizzando questa matrice nel programma di acquisizione descritto nel precedente paragrafo si otterrebbe il matching perfetto dei target come in Figura 2.14.

```
%% prerotazione
theta=pi;
rotz=[cos(theta) -sin(theta) 0; sin(theta) cos(theta) 0;0 0 1 ];
trans=[0 0 0];
tform_pre=rigid3d(rotz,trans);
movingReg_pre = pctransform(movingDownsampled,tform_pre); %prerotazione di 180 gradi
tform= pcregistericp(movingReg_pre,fixedDownsampled,'MaxIterations',10^7,'Tolerance',[0.1 0.5]);
movingReg = pctransform(movingReg_pre,tform);
```

```
%%
% Display the downsampled point clouds after registration.
figure
pcshowpair(movingReg,fixedDownsampled,'MarkerSize',50)
xlabel('X')
ylabel('Y')
zlabel('Z')
title('Point clouds after registration')
```

Figura 2.11: Seconda parte del codice: matrice rototraslazione

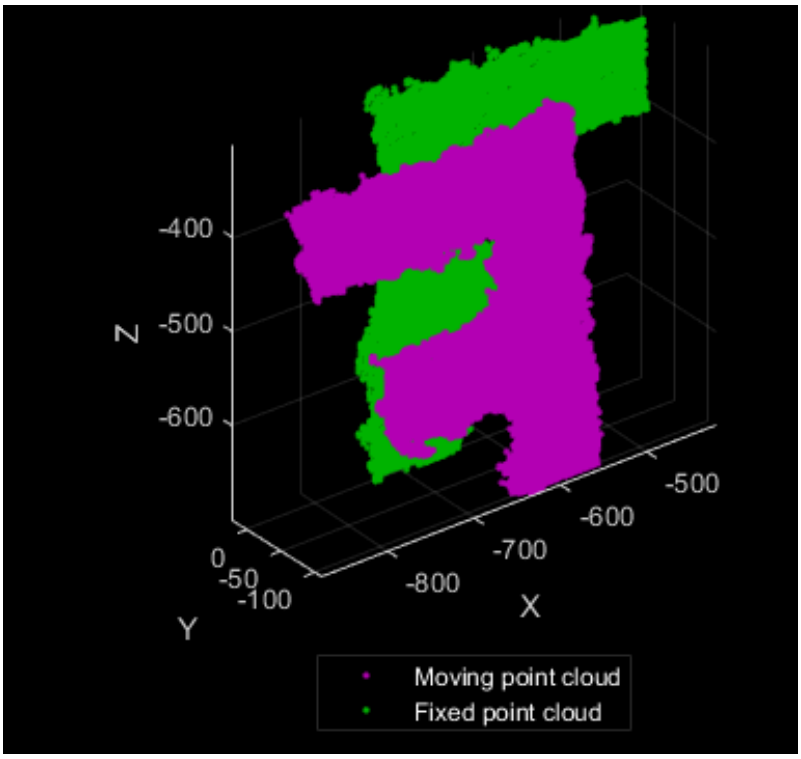


Figura 2.12: Target prima dell'allineamento.

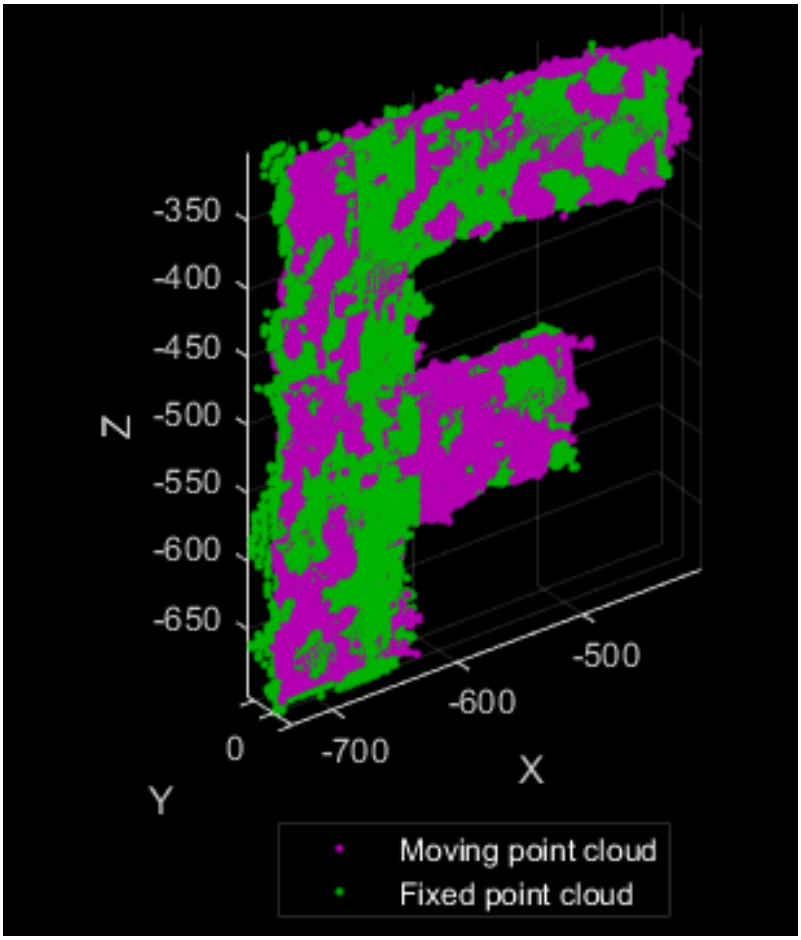


Figura 2.13: Allineamento.

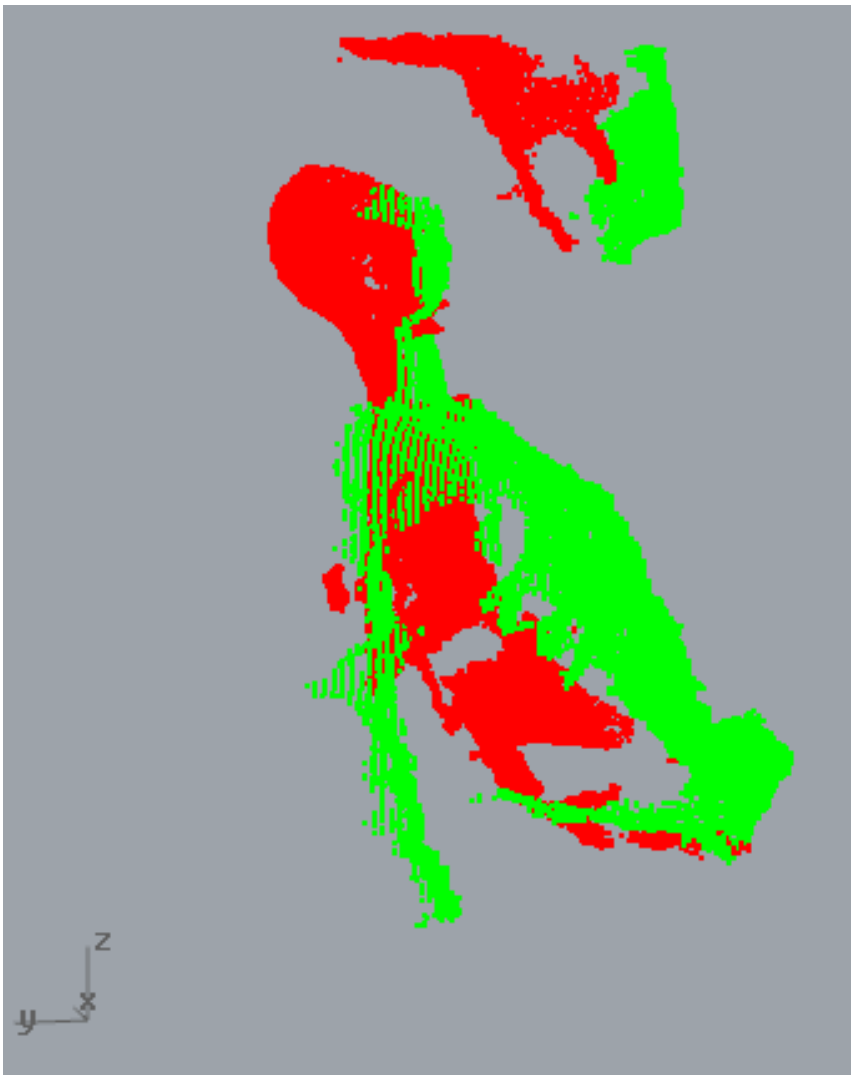


Figura 2.14: Matching nuvole.

Capitolo 3

Software definitivo

Identificazione oggetti

La distinzione e la ricerca degli oggetti è l'argomento portante di questa ricerca per garantire la sicurezza della regione d'interesse durante il funzionamento, infatti la capacità collaborativa del robot deve assicurare l'eventuale inserimento dello stesso all'interno di una catena di produzione. Precedentemente sono state descritte le modalità fondamentali che caratterizzano le acquisizioni, necessarie per rilevare le entità nell'area di lavoro. In questo capitolo si discuteranno le dinamiche relative al discernimento dei target. Di seguito sono presentati i metodi presi in considerazione, per categorizzare gli ostacoli nello spazio, durante lo sviluppo del programma.

3.1 Object detection

Object Detection è una tecnica che rientra nella Computer Vision, insieme di processi che generano un modello tridimensionale e reale partendo da immagini bidimensionali. Permette come suggerisce la denominazione, il rilevamento di oggetti in uno spazio predefinito. Matlab permette di implementare questa modalità, utilizzando il Toolbox di Computer Vision e di Deep Learning. In particolare, l'ambito della Computer Vision si specializza in due settori: Machine Learning, costruzione automatica di modelli analitici basata sull'apprendimento di dati e Deep Learning, apprendimento automatico basato sulle caratteristiche di fattori o concetti posti in fasce gerarchiche in modo tale che i concetti di alto livello si fondano su quelli di livello più basso, quindi incentrato su una stratificazione delle reti neurali in cui ogni livello calcola i valori per raggiungere il successivo fino ad ottenere l'informazione calcolata in modo più completo. Di seguito verranno affrontate le due modalità di riconoscimento immagine, evidenziandone pro, contro e un'eventuale combinazione.

3.1.1 Machine Learning

Questa tecnica prevede dapprima un allenamento del sistema ('training') tramite l'introduzione di immagini, la numerosità dipende da numero di entità da classificare, diverse tra loro ma appartenenti allo stesso ambito d'interesse. Successivamente, si indicano le caratteristiche tramite le quali si vuole attuare il riconoscimento e che

potrebbero differenziare un tipo di immagine, precedentemente introdotta, dall'altra('feature extraction'), così si costruisce un modello che distingue le caratteristiche precedentemente estrapolate in modo tale che se gli venga fornita un'immagine, sia in grado di associarla a una categoria specifica dell'ambito considerato all'inizio. In Matlab, è possibile impostare la predizione delle caratteristiche che differenziano le varie categorie con un'app di classificazione, propria del programma 'Classification Learning App', ciò potrà essere usato per tutte le successive librerie da analizzare. Con questo metodo si ottengono più classificatori, quindi buoni risultati con quantità minimale di dati. Il problema sarà relativo alla scelta delle caratteristiche di classificazione e la migliore combinazione per ottenere un risultato ottimale.

3.1.2 Deep Learning

La prima fase, come nel paragrafo precedente, prevede un allenamento del sistema tramite immagini di uno stesso ambito, ma differenti tra loro in modo tale che sia possibile attuare il riconoscimento. Queste vanno introdotte in una CNN(Convolutional Neural Network), rete neurale convoluzionale, ossia una rete neurale artificiale, in grado di captare in maniera automatica gli elementi per il riconoscimento delle entità mediante la sola introduzione delle immagini nel sistema. Fatto questo sarà possibile nell'immediato ottenere il risultato, introducendo un elemento per testarlo. Questa tecnica prevede l'uso di molti dati per creare una CNN con caratteristiche apprese, che garantisca il risultato immediato oppure l'utilizzo di una quantità media di dati in cui però è necessario pre-allenare la rete, quindi creare un nuovo strumento di classificazione e infine ottenere il risultato. Un'altra modalità prevede un intervento umano maggiore del caso precedente in quanto è necessario modificare l'algoritmo di Matlab estendendo la CNN in base alle proprie necessità così da allenare al massimo il programma anche tramite l'introduzione di immagini e video quanto più nitide e generiche possibile. Maggiore è la qualità delle immgini, maggiore è l'accuratezza, dovuta anche al fatto che le caratteristiche di riconoscimento sono scelte dal programma stesso. I dati richiesti sono molti, inoltre la soluzione si basa su un programma a scatola chiusa di conseguenza non è la tecnica più conveniente.

3.1.3 Combinazione

La possibilità di combinare le due tecniche non è surreale, potrebbe agevolare il risultato, ovviamente sarà necessario fare numerosi tentativi prima di trovare la connessione ottimale per la specifica applicazione. In genere, si inizia con una classificazione, dunque una CNN viene importata nel programma, l'accuratezza è in tal modo ottimizzata. In particolare, si fa riferimento al Deep Learning sull'estrazione delle caratteristiche ementre il Machine Learning interviene sulla fase di generazione dei classificatori. I metodi sopra presentati sono particolarmente complessi, di conseguenza avrebbero richiesto una quantità di tempo non indifferente: allenare il programma alla visualizzazione e riconoscimento dei target che in ogni caso non

sono standard e vengono ampliati a casi più generici non è banale, sarebbe stato necessario generare un set di immagini dei target più utilizzati in laboratorio in modo tale da abituare il programma e consentire un riconoscimento automatico.[5]

3.2 Clustering

Il termine ''Cluster" (inglese) significa "gruppo" (se sostantivo), o "raggruppare" (se verbo), in diverse discipline il clustering rappresenta il raccoglimento di una serie di oggetti secondo caratteristiche comuni, nel caso particolare di programmazione è necessario identificare un metodo che permetta l'unificazione tramite entità comuni e soprattutto applicarlo al caso tridimensionale in quanto la nuvola di punti acquisita complessiva è rilevata nello spazio e soprattutto la distinzione che si vuole attuare in questo specifico caso è tra robot e il resto della nuvola nello spazio oltre che tra gli oggetti che non siano robot, tra loro. Questa tecnica è utile, in particolare in questa ricerca, quando l'etichettamento dei dati è frequente e costoso, quindi maggiormente usato quando le nuvole di punti vanno associate a un oggetto, la classificazione risulterebbe particolarmente faticosa e soggetta a errori di valutazione, mentre un algoritmo di raggruppamento riesce facilmente a captare i punti che appartengono a uno stesso oggetto e necessita solo di un intervento umano che assegni la classe da riconoscere.

Valutazione clusters

Esistono metodi in grado di valutare l'efficacia di una suddivisione in cluster, importante per evitare raggruppamenti casuali o per confrontare algoritmi di clustering diversi. In generale, questi criteri di analisi si distinguono in 3 categorie:

1. Validazione interna: utilizza le informazioni interne al processo di clustering per valutare l'efficacia della struttura di un cluster senza riferimenti all'esterno, usata anche per la stima del numero di cluster e per l'algoritmo di cluster ottimale senza dati esterni. Il metodo maggiormente usato è il riconoscimento di oggetti dello stesso cluster quanto più simili tra loro e la la identificazione degli oggetti molto diversi tra loro per ottenere una distanza media tra cluster quanto più piccola possibile. in sintesi, può essere misurato tramite l'indice di seguito, alpha e beta sono i pesi:

$$Index = \frac{(\alpha \quad x \quad Separation)}{(\beta \quad x \quad Compactness)}$$

Silhouette

Altro indice di valutazione è quello esplicitato tramite coefficiente di silhouette che stima la distanza media tra i cluster. Il grafico che lo rappresenta, manifesta

quanto sono vicini i punti tra loro all'interno di ogni cluster e quanto distano tra i punti dei raggruppamenti circostanti.

Dunn

Questo indice prende in considerazione la distanza da ogni cluster e gli elementi in ogni raggruppamento e gli elementi negli altri raggruppamenti. Si considera la distanza minore tra i due punti delle precedenti situazioni considerate, per ogni raggruppamento si calcola la distanza tra gli elementi dello stesso cluster. Si usa poi la massima distanza all'interno dei cluster come manifestazione di compattezza. L'indice si calcola come segue:

$$D = \frac{min.separation}{max.diameter}$$

2. Validazione esterna: compara i risultati dell'analisi di cluster confrontando i dati noti di un risultato esterno, come una classificazione esterna già fornita, il numero di cluster da ottenere è già noto, solitamente usato per scegliere giusto algoritmo di cluster per dati specifici. Prevede la comparazione dei cluster identificati con cluster noti usati come riferimento, in modo da attribuire gli identificati a quelli esistenti.

3. Validazione relativa: valuta le strutture di cluster cambiando i valori dei parametri in uno stesso algoritmo(es. cambiando il numero di cluster.) Usato per trovare il numero ottimale di cluster.[6]

Un' altra importante distinzione tra i metodi di clustering è definita in diversi modi: clustering partizionante, suddivisione in sottinsiemi non sovrapposti, ogni oggetto appartiene ad un solo cluster e clustering gerarchico, insieme di cluster organizzati come albero gerarchico; esclusivo e non esclusivo, in quest'ultimo i punti possono appartenere a più cluster; fuzzy e non-fuzzy, un punto appartenente ad un fuzzy cluster appartiene a tutti i cluster che hanno peso tra 0 e 1, la somma dei pesi per ogni punto deve essere 1; parziale e completo, i punti del cluster parziale potrebbero non appartenere a nessuno dei cluster; eterogeneo e omogeneo, in un gruppo eterogeneo i cluster possono avere dimensioni, forme e densità molto diverse.

Tipologie di clustering

I criteri per la suddivisione in cluster sono esplicitati da caratteristiche che contribuiscono alla suddivisione e alla diversificazione della stessa nei raggruppamenti:

a) Compatezza di ogni cluster: ogni cluster è caratterizzato da un centro piuttosto che dal punto centrale di un altro cluster. Tale punto è definito centroide. Ciò può essere anche basato sulla densità: un cluster è una regione densa di punti ed è separata da regioni a bassa densità, dalle altre zone ad alta densità.

b) Separazione netta tra i cluster: i punti appartenenti a un cluster sono molto più vicini a ogni altro punto di quel cluster rispetto a ogni altro punto che non appartiene al cluster.

c) Connessione: il punto d'appartenenza ad un cluster è più vicino ad almeno uno dei punti che non vi appartiene.

Si illustrano di seguito gli algoritmi di clustering maggiormente usati in quanto analizzati per lo sviluppo del codice effettivo.[6]

K-means clustering

La tecnica del K-means è di tipo partizionante, ogni cluster appartiene a un centroide quindi ogni punto è attribuito al centroide più prossimo ma la quantità di cluster va specificata tramite K: selezionando K come il numero di centrodi iniziali, si formano K-clusters assegnando tutti i punti al centroide più vicino, allora si ricalcolano i centroidi di ogni cluster, finchè il numero di centroidi rimane lo stesso. I centroidi iniziali solitamente sono definiti casualmente ma i cluster variano ad ogni computazione: il centroide è la media dei punti del raggruppamento. La vicinanza tra i punti può essere calcolata in diverse modalità (distanza euclidea, correlazione, ...). La convergenza dell'algoritmo solitamente avviene nelle prime iterazioni. Questa tipologia di algoritmo è stata utilizzata nel progetto come primo tentativo di discernimento tra robot e non robot. Alcuni svantaggi legati al K sono stati rilevati in fase di sperimentazione. Questi vanno presi in considerazione fin dall'inizio, quindi se fosse necessario un controllo continuo dello spazio di lavoro in cui più di un target non-robot entrino in scena, il programma non sarebbe in grado di diversificare i nuovi oggetti in nuovi cluster, in quanto il valore di K non sarebbe prevedibile in un controllo in tempo reale. Inoltre l'etichettamento dei target cambia continuamente nel tempo anche tra target e robot stesso quindi non assicura nemmeno la distinzione tra entrambi. Ciò comporta che se un oggetto fosse eccessivamente vicino al robot, potrebbe essere riconosciuto all'interno del cluster del robot stesso non permettendone le funzionalità di robot collaborativo. In genere, questo algoritmo è sconsigliato per cluster di diverse dimensioni e densità. Tendenzialmente l'algoritmo trova centroidi che garantiscano una simile dimensione dei cluster che appunto non corrisponde al caso in analisi, stesso discorso va considerato per la densità.

Clustering gerarchico

Questo algoritmo organizza i cluster in un albero gerarchico: il diagramma formato illustra una sequenza di combinazioni tra cluster e per diverse tipologie di cluster possono corrispondere quantità diverse di punti. In questo caso il numero di cluster non va definito a priori, un ulteriore vantaggio è che si ottiene un certo ordine gerarchico di concetti. Esistono due tipi di approcci: agglomerativo, cluster formati da elementi singoli e ad ogni passo si combinano ai due più vicini fino a che non

rimane solo un cluster (o K cluster), e divisivo, si parte da un unico cluster che comprende tutti gli elementi e ad ogni passo si separa il cluster più distante fino a che i cluster non contengono un solo elemento (o sono stati creati K cluster). Questo metodo non è stato considerato per l'applicazione al robot in quanto il nostro obbiettivo non era unificare gli elementi dei diversi cluster.

DBSCAN Clustering

Questa tecnica considera la densità dei target, quindi numero di punti che compongono l'oggetto illustrato. Gli elementi del cluster si dividono in: core point, punti interni, border point, che si trovano vicino al core point ma hanno densità minore e noise point, tutti i punti che non sono i precedenti. Questo metodo è resistente al rumore e garantisce cluster di forma e dimensioni differenti. Il problema applicativo di questo metodo sta nell'elevata dimensionalità, rende difficile definire efficacemente il concetto di densità a causa di un'elevata sparsità. Il data set avrà densità variabili. Anche questo metodo risulta macchinoso da adattare al nostro progetto in quanto considerando un'acquisizione real time di oggetti in movimento i cluster non saranno calcolati sempre allo stesso modo e non vengono riconosciuti, e alto sarebbe il rischio di confondere il target col robot nel momento in cui la distanza tra i due sia molto piccola. La distinzione di ogni algoritmo è basata sulla misurazione di caratteri simili. L'analisi dei cluster è usata in diverse applicazioni, oltre che in quello meccatronico. Questa sintesi sulle possibili opzioni è necessaria per parlare di ciò che è stato sviluppato per raggiungere lo scopo d'interesse, infatti le tecniche precedentemente spiegate, anche se non integralmente usate sono state d'ispirazione per la stesura del programma finale, per cui queste metodologie saranno di seguito illustrate.[7]

3.2.1 Sviluppo algoritmo di clustering

Nel precedente paragrafo sono stati mostrati gli algoritmi normalmente più usati per il clustering e le giustificazioni che hanno portato a una composizione di un programma personalizzato. Inoltre, tale codice risulta più intuitivo e semplice rispetto a quelli basati sulle teorie sopra citate in quanto è più flessibile rispetto alle esigenze necessarie. Il software rappresentato di seguito (Figura 3.1) è nella sua forma finale, il tutto avviene tramite la funzione 'clusterizzazione rt', (Figura 3.2) che raggruppa un insieme di operazioni che permettono la suddivisione in classi in tempo reale.

```
%% Clusterizzazione

[~,ellissoidi,~,~]=clusterizzazione_rt(X,minDistance,min_cl_size,n_stddev);
```

Figura 3.1: Definizione programma di clustering

```
function [id_cluster,ellissoidi,Y,n_cl]=clusterizzazione_rt(Y,minDistance,min_cl_size,n_stddev)
    Y=double(Y);
    ptCloud =pointCloud(Y);
    [id_cluster,n_cl] = pcsegdist(ptCloud,minDistance);
    [Y,id_cluster]=filtro_cl(Y,id_cluster,min_cl_size);
    [id_cluster,n_cl]=ridimensionamento_cl(id_cluster,min_cl_size,n_cl);
    ellissoidi=cluster(Y,id_cluster,n_stddev);
end
```

Figura 3.2: Funzione di clusterizzazione.

Y (Figura 3.2) rappresenta l'insieme di punti acquisito, la funzione 'pointCloud' trasforma Y in una nuvola di punti, azione necessaria per le successive operazioni. Si cercano gli 'id cluster' che sono indici di riconoscimento di ogni cluster e il numero stesso di cluster, 'n cl' con la funzione 'pcsegdist', funzione propria di Matlab che definisce i cluster tramite la minima distanza Euclidea tra i raggruppamenti. La funzione 'filtro cl', in Figura 3.3 e Figura 3.4 è stata invece creata per riconoscere gli indici di cluster e il numero di cluster per poi passarli in rassegna in base alle dimensioni: nella prima parte si definisce la dimensione dei cluster basata sugli indici e se è più piccolo della dimensione minima dei cluster gli viene attribuito un indice pari a zero con l'applicazione di 'if', all'interno del ciclo for che itera tutti i cluster trovati. Nella seconda parte, Figura 3.4 si considerano gli indici che non

```
function [Y,id_cluster, u_cl]=filtro_cl(Y,id_cluster,min_cl_size)

n_cl=size(unique(id_cluster),1);

for kk=1:n_cl
    ind=find(id_cluster==kk);
    tester=size(id_cluster(ind),1);
    if(tester<min_cl_size)
        id_cluster(ind)=0;
    end
end
```

Figura 3.3: Funzione di filtro: parte uno.

sono trascurabili quindi che siano propriamente definiti e saranno quelli che poi espliciteranno il numero effettivo di cluster: nel primo ciclo for del codice sottostante, tra tutti i cluster presenti, si trovano quelli propriamente definiti, nel secondo si definisce la dimensione per ogni cluster.

L'attuazione effettiva di 'filtro cl' sarà fatta tramite la funzione 'ridimensionamento cl' (Figura 3.5) in quanto trasforma quegli indici definiti pari a zero in NaN, Not a Number, quindi non vengono considerati né nella nuvola di punti né nella suddivisione in cluster.Di seguito, avviene una vera e propria manipolazione dei dati preesistenti in modo da ottenere gli indici e il numero di cluster ricalcolati senza i raggruppamenti trascurabili. La funzione successiva presente nel programma è 'cluster' (Figura 3.6) che dà come risultato ellissoidi cioè il riconoscimento degli ostacoli in ingresso come ellissoidi di dimensione corrispondente alla dimensione reale. Nel dettaglio, dapprima c'è il riconoscimento dell'indice e del cluster relativo al robot: il robot si

```
id_cluster=id_cluster(id_cluster>0);
Y=Y(id_cluster>0,:);
u_cl=unique(id_cluster);
n_cl=size(u_cl,1);
kk=1;
for ii=1:n_cl
    id_cluster(id_cluster==u_cl(ii))=kk;
     kk=kk+1;
end



for kk=1:n_cl
    ind=find(id_cluster==u_cl(kk));
    dim_cl(kk)=size(id_cluster(ind),1);
end
```

Figura 3.4: Funzione di filtro: parte due.

```
function [id_cluster,n_cl]=ridimensionamento_cl(id_cluster,min_cl_size,n_cl)
for kk=1:n_cl
        if(size(id_cluster==kk,1)<min_cl_size)
            ind=find(id_cluster==kk);
            Y(ind,:)=NaN;
            id_cluster(id_cluster==kk)=NaN;
        end
    end
```

Figura 3.5: Funzione di ridimensionamento.

definisce considerando la distanza minima rispetto all'origine, e nei passaggi seguenti, l'algoritmo di negazione definisce la nuvola target, eliminando la percezione del robot. La nuvola di target poi, si suddivide in più ellissi che si adattano ad ogni target che la compone, infatti il parametro di deviazione standard ('n stddev') aggiusta il solido elementare in relazione all'oggetto tenendo conto delle sue dimensioni. Qui si definisce 'ellissoidi' come vettore per richiamare la visualizzazione degli ellissi quindi verificare l'effettiva coincidenza dei cluster con gli ostacoli reali oltre che per una comunicazione futura più sintetica dei dati al robot. Di seguito un illustrazione dei passaggi effettuati.

```
function ellissoidi=cluster(Y, id_cluster,n_stddev)

    n_cl=size(unique(id_cluster),1);
    % Ricerca robot
    for jj=1:n_cl
        ind=find(id_cluster==jj);
        YYY=Y(ind,:);
        [mm(jj),imin]=min(sqrt(sum(YYY.^2,2)));
    end
    [mmm,cl_robot]=min(mm);
    % Eliminazione robot
    ellissoidi=[];
    Y_tot=[];
    for kk=1:n_cl
        if(kk~=cl_robot)
            YY=Y(id_cluster==kk,:);
            Y_tot=[Y_tot; YY];
            ellissoide=ellipsoid_t(YY,n_stddev);

            ellissoidi=[ellissoidi; ellissoide];
        end
    end
    plot_ellissoidi(ellissoidi, Y_tot);
```

Figura 3.6: Funzione di cluster.

3.2.2 Composizione programma finale

Questo paragrafo illustra la stesura del programma definitivo sulla base dei precedenti codici sviluppati individualmente. Il software è composto da tutte le funzioni sopra mostrate e provvisto di modalità di acquisizione real time così da comunicare al robot gli ostacoli che di volta in volta si interfacciano nell'area di lavoro. L'acquisizione può essere fermata in qualsiasi momento premendo la barra spaziatrice. Questo programma, nello specifico, raccoglie tutti i passaggi relativi alla caratterizzazione dei parametri e all'algoritmo di clustering, all'interno di un ciclo while che ne permette appunto la condivisione di dati in tempo reale. Questi dati verranno raccolti e trasferiti al robot che si muoverà di conseguenza. Lo script primario del codice è illustrato nella figure di seguito (Figura 3.7). Lo script si apre col la possibilità di scegliere con quale risoluzione acquisire: 'Lo' sta per Low, quindi una risoluzione minore della telecamera di depth equivalente a 320x240, questo tipo di scelta prevede un'acquisizione d'immagini più rapida come conseguenza di un insieme di punti maggiore. 'Hi' d'altro canto, sta per 'High', caratterizza la risoluzione massima della telecamera di depth quindi 640x480, massimo numero di punti acquisibile, caratterizzato da una comunicazione di dati un po' meno prestante della precedente. Si può scegliere l'una o l'altra opzione selezionando opportunamente il comando 'resolution'. Per completezza si riportano anche i parametri visti precedentemente che seguono il codice sopra riportato (Figura 3.8 eFigura 3.9).

```
close all
clc

%% Calibrazione
% generalizzazione sistema di riferimento e dati acquisiti
minimo=10;
massimo=2200;

resolution='Lo';   %Hi oppure Lo
switch resolution
    case 'Hi'
        r=480;
        c=640;
        formato='Depth_640x480';
    case'Lo'
        r=240;
        c=320;
        formato='Depth_320x240';
end

% Cluster parameters
minDistance=50;
min_cl_size=200;
n_stddev=4;
```

Figura 3.7: Prima parte, codice finale.

Nella parte finale del codice si definisce il ciclo while, in particolare 's': è una struttura che al suo interno prevede tutti i parametri neccessari per il richiamo delle funzioni (Figura 3.10). Dunque, in ellissoidi while loop (Figura 3.11) c'è il richiamo di tutte le funzioni precedentemente sviluppate ma riportate all'interno di un ciclo while che può essere bloccato tramite l'uso della barra spaziatrice. Dal programma sottostante si nota come l'output ellissoidi, quindi gli ostacoli d'interesse da comunicare al robot, vengano calcolati tramite la funzione 'elaborazione nuvola': La funzione 'elaborazione nuvola', mostrata di seguito in Figura 3.12, racchiude al suo interno tutte le funzioni viste sopra, qui nello specifico vengono estratte dalla struttura le variabili necessarie per le funzioni da richiamare: Per completezza verrà riportato anche ciò che resta nel codice della funzione elaborazione nuvola (Figura 3.13):

```
%% cluster parameters
minDistance=50;
min_cl_size=200;
n_stddev=4;
%% centro assi
tr_x_centro=[-621 -1400 -650];
%% caricamento della curva z index
d_zz=load('curva_z_uk.mat', 'fitresult');
d_z=d_zz.fitresult;
clear d_zz
%% caricamento della curva di taratura dei pixel in mm
d_pp=load('curva_taratura.mat','fitresult');
d_p=d_pp.fitresult;
clear d_pp

rr=1:r;
rr=squeeze(rr-floor(r/2));
az=repmat(rr',1,c);
az=reshape(az, c*r,1);

cc=1:c;
cc=squeeze(cc-floor(c/2));
ele=repmat(cc,r,1);
ele=reshape(ele, c*r,1);
```

Figura 3.8: Seconda parte, codice finale.

Se la visualizzazione fosse lasciata attivata si potrebbero vedere in tempo reale i target in ingresso e in uscita dallo spazio di lavoro (Figura 3.13). Questo è stato fondamentale in fase di implementazione in quanto permetteva la visualizzazione di ciò che ancora non era concreto.

```
%% Inizializzazione Kinect 1 e 2
info = imaqhwinfo('kinect');
% Kinect 1

vid1 = videoinput('kinect', 2,formato,'FramesPerTrigger',1);
src1 = getselectedsource(vid1);
src1.DepthMode='Near';
triggerconfig(vid1,'manual');
start(vid1);

% Kinect2

vid2 = videoinput('kinect', 4,formato);
src2 = getselectedsource(vid2);
triggerconfig(vid2,'manual');
start(vid2);

pause(2)
%% Allineamento dispositivi video (Kinect 1(DX)è quella di riferimento)
tformF_pree=load('tformF_pre.mat');
tformF_pre=tformF_pree.tform_pre;
tformmF=load('tformF.mat');
tformF=tformmF.tform;
clear tformF_pree
clear tformmF
```

Figura 3.9: Terza parte, codice finale.

```
%% Acquisizione while loop
s=struct('f',{vid1,vid2,az,ele,minimo,massimo,d_z,d_p,tr_x_centro,tformF_pre,tformF,minDistance,min_cl_size,n_stddev});

ellissoidi_while_loop(s);
```

```
%%

stop(vid1)
stop(vid2)
delete(vid1)
delete(vid2)
imaqreset
```

Figura 3.10: Quarta parte, codice finale.

```
%% While Loop ___ blocco con tasto
function ellissoidi_while_loop(s)
global KEY_IS_PRESSED
KEY_IS_PRESSED=0;
gcf
set(gcf,'KeyPressFcn',@myKeyPressFcn)


while ~KEY_IS_PRESSED
      tic;

        ellissoidi=elaborazione_nuvola(s);
        drawnow

      size(ellissoidi,1)
      toc;
      frequenza=1/toc;


end
disp('loop ended')
```

Figura 3.11: Codice definizione while loop, si può fermare con tasto.

```
function ellissoidi=elaborazione_nuvola(s)
vid1=s(1).f;
vid2=s(2).f;
az=s(3).f;
ele=s(4).f;
minimo=s(5).f;
massimo=s(6).f;
d_z=s(7).f;
d_p=s(8).f;
tr_x_centro=s(9).f;
tform_pre=s(10).f;
tform=s(11).f;
minDistance=s(12).f;
min_cl_size=s(13).f;
n_stddev=s(14).f;
```

Figura 3.12: Estrazione da struttura.

```
%% Nuvole
fixed=pointCloud(X1);
moving=pointCloud(X2);
%% Allineamento nuvole_rottraslmatr:possibilità di configurare n-cameras
movingg=pctransform(moving,tformF_pre);
movinggg=pctransform(movingg,tformF);
X=[fixed.Location' movinggg.Location']'; %nuvola complessiva

X=[X(:,1),X(:,2),X(:,3)];
%% Clusterizzazione

[~,ellissoidi,~,~]=clusterizzazione_rt(X,minDistance,min_cl_size,n_stddev);
```

Figura 3.13: Definizione nuvola e cluster.

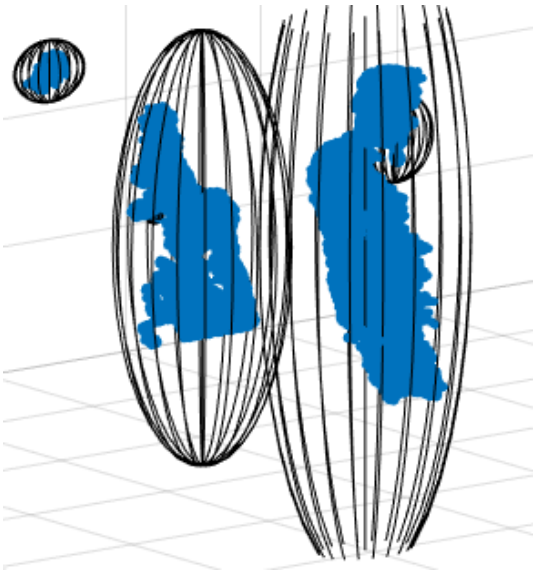


Figura 3.14: Visualizzazione clustering.

Risultati

L'obbiettivo del progetto era quello di ideare un programma che potesse agevolare il riconoscimento di oggetti nello spazio. Lo sviluppo dell'algoritmo sopra presentato ha sicuramente soddisfatto le richieste in quanto è possibile comunicare al robot quali siano i punti che caratterizzino il robot stesso ma soprattutto è possibile rilevare in tempo reale gli oggetti che entrano nello spazio di lavoro circostante e l'associazione di forme elementari ai target, diversificandole in base alle dimensioni reali dei target stessi. La forma elementare scelta è un ellisoide in quanto nelle prove eseguite con target di forma e dimensioni molto diverse tra loro era quella che si adattava meglio alle variazioni, descrivendo sempre in maniera calzante ogni osacolo. Questo contribuisce ad avere un sistema collaborativo in cui il monitoraggio dell'area di lavoro avviene nel tempo corrente quindi il robot sarebbe in grado di mantenere le opportune precauzioni durante un'eventuale compito in prossimità di un operatore. Il programma sviluppato risulta anche molto veloce nell'elaborazione dei dati in quanto tutte le operazioni fatte in fase di sperimentazione sono state trasformate in funzioni e sono stati ridotti al minimo gli effetti di visualizzazione per la validazione del programma stesso, in questo modo il processo di calcolo e conseguente trasferimento di dati è fatto in maniera più efficiente. Per agevolare la computazione è anche possibile scegliere tra due modalità: una ad alta risoluzione 'Hi', quindi vengono acquisiti maggiori punti, una bassa risoluzione 'Lo', vengono acquisiti minori punti, l'alleggerimento delle acquisizioni di gran lunga rende più veloce il trasferimento di informazioni, in queste applicazioni può essere usato in quanto non è rilevante il quantitativo di punti, un alleggerimento del target nello spazio non comporterebbe errori di riconoscimento ma renderebbe il processo più veloce.

Bibliografia

[1] A. Doyon e L. Liaigre. Jacques vaucanson, mécanicien de génie. *PUF*, 1966.

[2] Nils J. Nilsson. Shakey the robot. *SRI International*, 1984.

[3] IBM. Joel benjamin playing a practice game with deep blue. *Computer History Museum*, 1997.

[4] Roanna Lun e Wenbing Zhao. A survey of applications and human motion recognition with microsoft kinect. *International Journal of Pattern Recognition and Artificial Intelligence*, 2015.

[5] Johanna Pingel. Object recognition: Deep learning and machine learning for computer vision.

[6] Alboukadel Kassambara. Cluster validation statistics: must know methods.

[7] Matteo Golfarelli. Clustering.