UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea triennale in Ingegneria Meccanica

SVILUPPO DI UN SISTEMA DI RILEVAMENTO DI OSTACOLI IN 3D PER APPLICAZIONE SU ROBOT COLLABORATIVI

DEVELOPMENT OF AN OBSTACLE DETECTION SYSTEM IN 3D FOR APPLICATION ON COLLABORATIVE ROBOTS

Relatore:

Chia.mo Prof. Paolo Castellini

Tesi di Laurea di:

Marzia Di Tommaso

Anno Accademico 2019 / 2020

INDICE

INTRODUZIONE

I robot sono diventati parte integrante e fondamentale per il lavoro industriale: dalle catene di montaggio fino ad arrivare alle piccole imprese le nuove tecnologie, si affiancano all'uomo affinché le operazioni diventino più veloci, più precise e più sicure. Infatti i robot moderni in collaborazione con il lavoratore, permettono alle aziende di produrre più velocemente con meno rischio di errore e di danno, sollevando il lavoratore da compiti più gravosi e ripetitivi.

La ricerca svolge un compito fondamentale nel continuare a sviluppare nuovi metodi e sinergie tra robot e uomo, per migliorare qualità ed efficienza di questi macchinari.

Questo lavoro di tesi si pone come obiettivo quello di far si che in un ambiente di lavoro un robot riesca a riconoscere gli ostacoli che si trova davanti in tempo reale, in modo da diminuire il rischio del danno fisico del lavoratore o di danno materiale ai beni prodotti.

Per far si che il robot diventi consapevole dell'ambiente circostante utilizzeremo un dispositivo sensibile al movimento del corpo umano: la Kinect di Microsoft, inizialmente progettata per lavorare con le console per videogiochi e poi utilizzate anche per applicazioni ludiche.

Con la Kinect possiamo determinare forma e distanza degli ostacoli in un ambiente definito; questi input vengono poi trasmessi al robot tramite un software programmato in MATLAB in grado di classificare e visualizzare i vari target presenti in un ambiente di lavoro in tempo reale.

In particolare si vuole essere in grado di distinguere cosa è il robot e cosa il target nel campo visivo sia che il target sia statico che in movimento, e che il riconoscimento sia effettuato in maniera continuativa e in tempo reale.

La tesi andrà a descrivere come è stato programmato questo software a partire dal setup delle Kinect, le acquisizioni fatte e il processamento delle immagini.

Nel dettaglio:

- Nel primo capitolo viene descritta la strumentazione utilizzata: il braccio meccanico e le due Kinect e MATLAB.

- Il secondo capitolo tratta la taratura statica delle Kinect: questa è un'operazione essenziale affinché le immagini che si osservano dalle sue Kinect, siano coerenti tra loro.

- Nel terzo capitolo vengono descritte sia l'acquisizione che l'elaborazione delle immagini: in particolare verrà spiegato come il programma MATLAB è in grado di sequenziare le immagini in modo da ottenere un loop di immagini stesse in tempo reale. Definito il loop, in questo capitolo viene illustrato anche come sono definiti i vari cluster, definendo i target come ellissoidi.

- Nel quarto capitolo vengono esposti i risultati ottenuti nelle acquisizioni fatte in laboratorio, che hanno permesso di verificare la validità e il corretto funzionamento del programma, secondo le specifiche richieste.

1. Strumentazione e banco di prova

In questo capitolo verranno descritti tutti gli strumenti utilizzati per lo svolgimento di questa tesi, ovvero del robot, delle due Kinect e di MATLAB.

1.1.Il Robot

Il robot utilizzato per sviluppare questo progetto è un braccio meccanico, il quale è incernierato sulla sommità di una struttura. Il suo corpo è formato da tre parti collegate tra loro e in grado singolarmente di ruotare di 360° su se stesse, e di sollevarsi e abbassarsi. All'estremità è presente una superficie con forma di cuneo la quale riesce a ruotare su se stessa attorno all'ultimo tratto del braccio meccanico.

Un telecomando collegato al robot permette, oltre al movimento delle singole parti che lo compongono, anche di controllare la velocità con il quale si possono muovere.

1.2.Microsoft Kinect

Per questa tesi i sensori utilizzati sono state le Kinect di Windows e per X-box 360.

La Kinect (Figura 1 Microsoft KinectFigura 1) è un dispositivo di input che rileva il movimento del corpo umano in tempo reale. È stata progettata per essere utilizzata in sostituzione ai classici controller per videogiochi, e per far sì che il giocatore potesse essere il controller stesso senza l'utilizzo altri strumenti.

Figura 1 Microsoft Kinect

È formata una barra orizzontale collegata ad un perno, che permette alla Kinect di muoversi lungo l'asse verticale per tutti i movimenti, e da vari sensori: una fotocamera RGB, un sensore di profondità e un array di microfoni.

- Il sensore di profondità è da un proiettore laser a infrarossi combinato con un sensore monocromatico che acquisisce video in 3D. Ha un flusso video di risoluzione VGA con matrice 640x480 pixel.
- La videocamera RGB ha una risoluzione VGA di 640x480 pixel ed una capacità di lettura a 30 fps.
- L'array di microfoni è una sorgente audio che permette di acquisire ed elaborare dati sonori dell'ambiente in cui il sensore si trova.

1.3. MATLAB

MATLAB, abbreviazione di Matrix Laboratory, è un software prodotto da MathWorks scritto nel linguaggio C. È un ambiente per il calcolo numerico, analisi statistica, elaborazione immagini e segnali e molto altro.

Questo software lavora prevalentemente con dati matriciali ma riesce anche a visualizzare dati e funzioni, implementare algoritmi ed interfacciarsi con altri programmi. Si possono trovare in esso dei toolboxes, cioè delle librerie di file che estendono la funzionalità del programma di base.

L'interfaccia (Figura 2) è formata da varie finestre. Le principali sono:

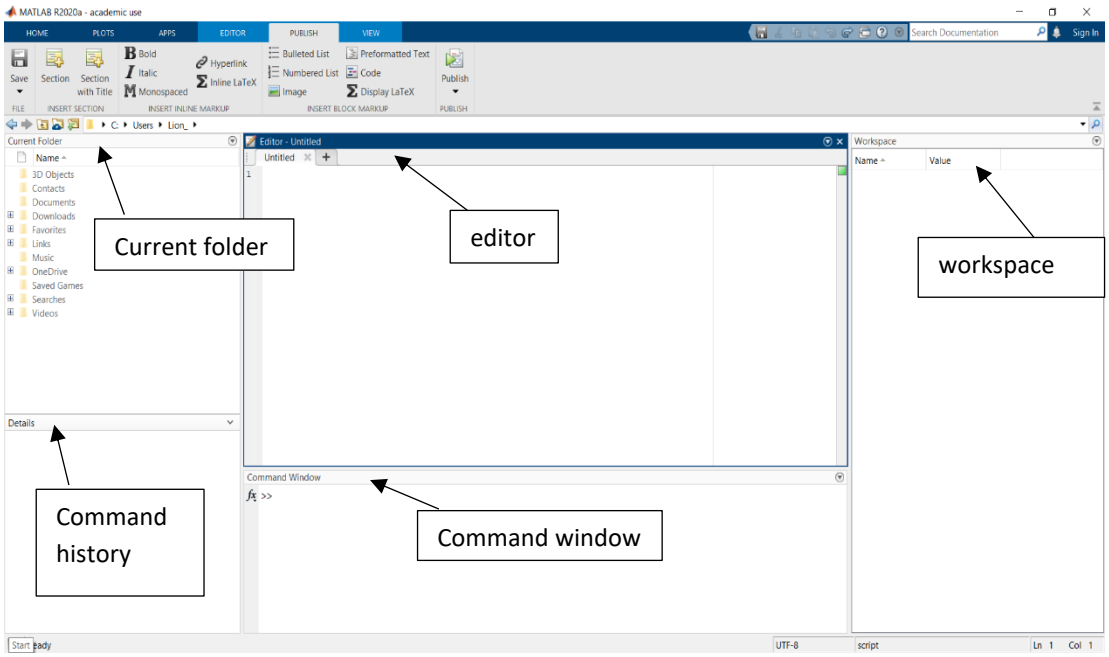

Figura 2 Interfaccia Matlab

- Editor: è la finestra dove scrivere il codice
- Command Window: è la finestra dove vengono inseriti i comandi eseguiti in tempo reale
- Workspace: è uno spazio di lavoro che andrà a contenere tutte le variabili definite nella Command Window
- Current Folder: contiene tutti i file e le cartelle utili
- Command History: contiene tutta la lista dei comandi definiti nella Command Window

2. La Taratura

In questo capitolo tratteremo della taratura delle Kinect. Definiremo cos'è una taratura statica di uno strumento e le caratteristiche della taratura stessa; e successivamente verranno descritti i passaggi effettuati per ottenere le due curve di taratura delle Kinect.

2.1. Sistema di riferimento

Il primo step è definire il sistema di riferimento che verrà utilizzato nell'analisi dell'intero progetto, sia per le singole immagini, che per il loop di immagini che vengono acquisite dalle due Kinect.

In questa tesi il sistema di riferimento è stato impostato secondo la posizione del robot, ponendo lo zero sulla sommità del robot definendo così le direzioni degli assi cartesiani (Figura 3):

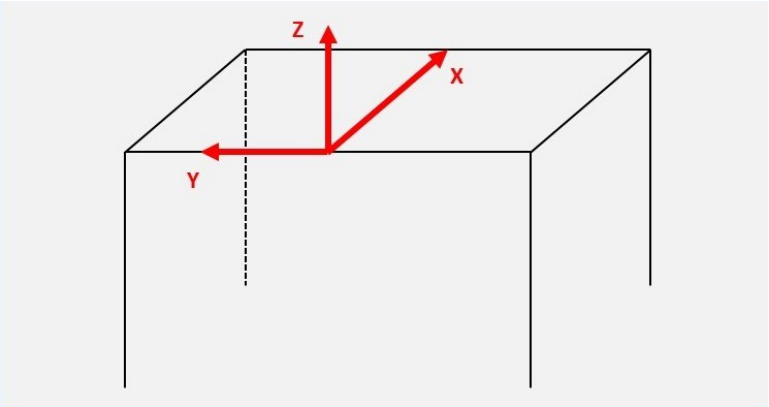

Figura 3 Sistema di riferimento utilizzato

Bisogna valutare anche il campo visivo delle due Kinect. Nel setup utilizzato, queste sono poste l'una di fronte l'altra alla stessa altezza da terra. Il campo visivo ottenuto in questo caso è il seguente (Figura 4):

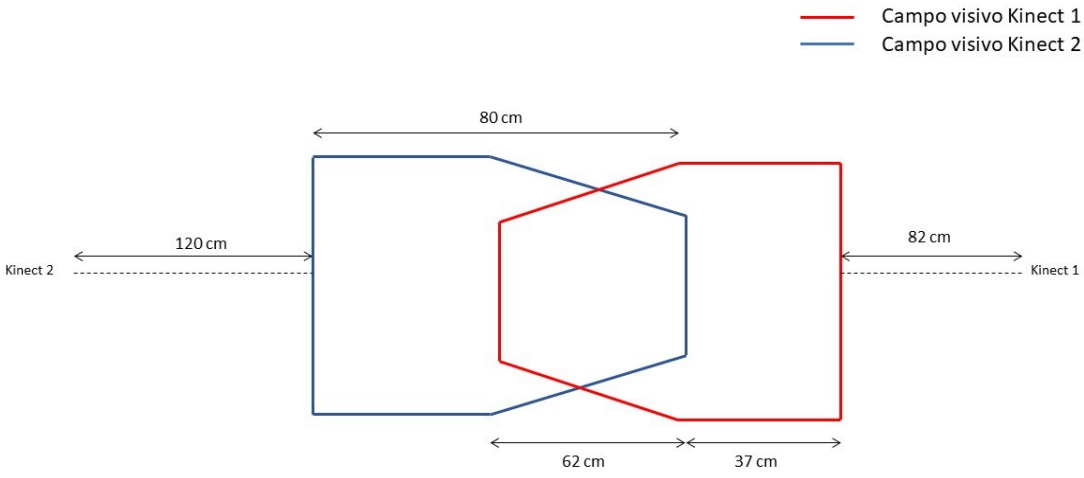

Figura 4 Campo visivo delle Kinect

2.2.Traslazione del sistema di riferimento e unione immagini acquisite

Le immagini acquisite dalle Kinect possono essere visualizzate come nuvole di punti, utilizzando un tool presente in Matlab.

Tramite questa rappresentazione delle immagini, si è notato come ciò che acquisiva la Kinect 1 non combaciava perfettamente con ciò che acquisiva la Kinect 2.

Per questo motivo abbiamo apportato una traslazione del sistema di riferimento sul programma principale scritto in Matlab, affinché le acquisizioni corrispondessero esattamente.

La traslazione è stata fatta andando a valutare la distanza presente tra un punto della nuvola ottenuta da Kinect 1 e lo stesso punto ma della nuvola acquisita dalla Kinect 2. La valutazione delle distanze è stata eseguita per tutte e tre le coordinate del sistema di riferimento (Figura 5) traslando quindi l'immagine sia lungo l'asse x, sia lungo l'asse y e l'asse z.

```
%% Calibrazione
%%
minimo=1000;
massimo=2000;
r=480;
c=640;
%centro assi
tr_x_centro=[-621 -1400 -650];
```

Figura 5 Calibrazione e traslazione del sistema di riferimento

2.3.Taratura statica

La taratura statica è un procedimento che viene effettuato in laboratorio, definito da una serie di misure su campioni i quali presentano delle grandezze di riferimento.

Serve a determinare sia le caratteristiche statiche degli strumenti di misura, che a verificare il corretto funzionamento degli strumenti stessi trascorso un determinato tempo di utilizzo.

Il processo di taratura può essere sintetizzato in due fasi. Nella prima fase si definisce una relazione le grandezza e le rispettive incertezze di misura; mentre nella seconda fase si stabilisce la relazione matematica che intercorre tra le grandezze considerate per ottenere un risultato di misura.

La taratura statica, nella maggior parte dei casi, viene tradotta tramite una relazione lineare (Figura 6) , che correla gli ingressi sull'asse delle ascisse e le rispettive uscite sull'asse della ordinate. Questo ci permette di ottenere per l'appunto una relazione lineare di ingresso-uscita per lo strumento che si sta tarando.

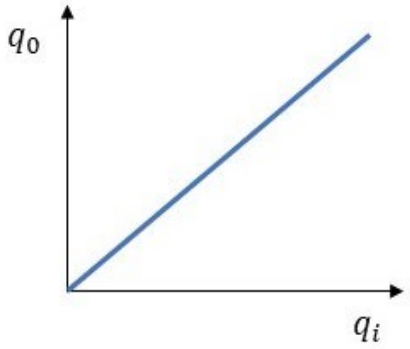

Figura 6 Curva taratura lineare

La curva di taratura, oltre che ad essere lineare, può essere definita anche da una curva polinomiale (Figura 7), che presenterà un'incertezza di linearità a percentuale del fondo scala.

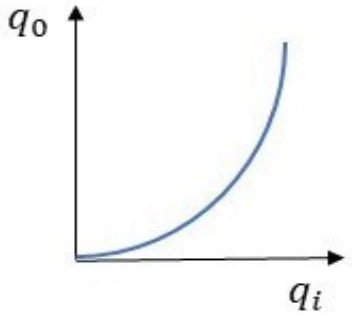

Figura 7 Curva taratura polinomiale

Importante è il controllo preliminare dello strumento, che consiste nel ripetere la misura per un determinato valore di ingresso fisso e identificare poi, i possibili errori tramite i confronti con il campione.

Successivamente alle fase preliminare, si passa alla determinazione sperimentale delle relazioni che sussistono tra ingresso e uscita dello strumento. Questo avviene mantenendo gli ingressi fissi e costanti eccetto uno che varierà con continuità in modo da definire la relazione ingresso-uscita che di desidera.

La misura verrà ripetuta in modo da ottenere una sequenza di valori in uscita dallo strumento, e per ognuna di queste si può calcolare una distribuzione di ampiezza. Se quest'ultime presentano la stessa deviazione standard, possiamo definire la curva ottimale di interpolazione dei valori medi delle diverse distribuzioni; la quale viene chiamata curva dei minimi quadrati.

La curva dei minimi quadrati si ottiene andando a minimizzare la somma dei quadrati delle distanze di ogni punto di taratura dalla curva stessa. Infatti se lo strumento presenta un comportamento lineare si possono interpolare tutti i punti con una retta tramite questa tecnica, ottenendo la seguente equazione:

$$q_0 = mq_i + b \qquad [1.1]$$

La sensibilità dello strumento è il valore associato al coefficiente m, rappresentata dalla pendenza della retta della curva di taratura. Questa viene definita dal rapporto che intercorre tra la variazione in uscita e quella corrispondente all'ingresso (Figura 8);

$$S = \frac{dq_0}{dq_i} \qquad [1.2]$$

se la curva di taratura è lineare la sensibilità è costante in tutto il campo di misura; diversamente ciò non accade.

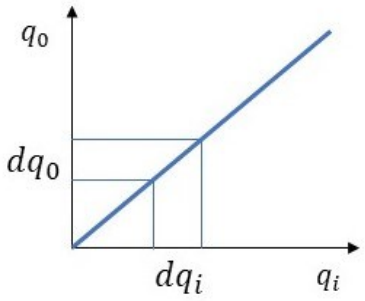

Figura 8 Sensibilità

2.3.1. Caratteristiche statiche

Per ogni strumento sul quale si effettua una taratura statica bisognerà considerare le caratteristiche statiche associate.

Si definisce risoluzione (Figura 9) la più piccola variazione di ingresso che lo strumento riesce a misurare.

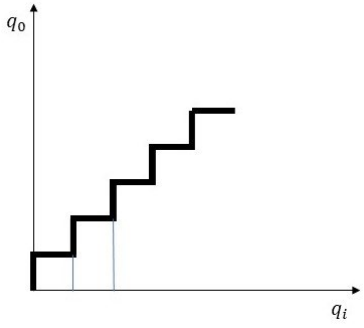

Figura 9 Risoluzione

Mentre la soglia(Figura 10) è il più piccolo ingresso misurabile dello strumento, ovvero un valore al di sotto del quale lo strumento non rileva una variazione nell'uscita.

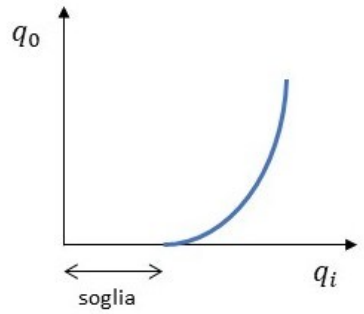

Figura 10 Soglia

Chiameremo fondo scala il limite superiore del campo di misura, cioè il più grande ingresso che lo strumento riesce a misurare. Questa caratteristica viene anche utilizzata per adimensionalizzare tutte le altre.

Altra caratteristica di cui dobbiamo tener conto, per quanto riguarda lo strumento, è l'incertezza. Per valutarla bisogna ripetere per la misura sul campione un determinato numero di volte, ovvero per ogni ingresso valuteremo la deviazione standard delle sequenze dei valori in uscita dello strumento.

Se ipotizziamo di avere la stessa deviazione standard per ogni frequenza, questa si potrà calcolare dalla differenza tra i valori della retta dei minimi quadrati e ogni singola lettura evitando così di ripetere il calcolo più volte per ogni valore che abbiamo di riferimento.

Utilizzeremo una relazione matematica che ci permette di determinare l'incertezza intrinseca, cioè:

$$S_{q_0} = \sqrt{\frac{1}{N-2}\sum(mq_i + b - q_0)} \qquad [1.3]$$

L'incertezza intrinseca dello strumento rappresenta l'incertezza minima che si può ottenere durante la taratura in condizioni ottimali di impiego in laboratorio. Può essere valutata anche come percentuale del fondo scala come:

$$S_{q_i} = \frac{S_{q_0}}{m} \qquad [1.4]$$

Da considerare è anche la (non) linearità che è la massima deviazione dei punti di taratura della retta di regressione definita dal metodo dei minimi quadrati o espressa come percentuale del fondo scala.

Tra le caratteristiche statiche abbiamo anche l'isteresi la quale rappresenta la massima differenza tra la curva di carico e quella di scarico dello strumento. Si può

considerare quindi come una componente dell'incertezza, questo perché nel momento in cui utilizziamo lo strumento non sappiamo se il misurando sta aumentando o diminuendo durante la lettura.

Durante una taratura statica bisogna effettuare le misure prima aumentando il valore in ingresso dello strumento, e successivamente diminuendolo.

2.4. Taratura Kinect

Per questo progetto i sensori utilizzati sono la Kinect sulle quali abbiamo effettuato, prima dell'effettivo utilizzo, due tarature statiche che ci hanno permesso di ottenere una corrispondenza tra le immagini ottenute dalla prima Kinect e quelle della seconda Kinect visualizzabili in scala.

La prima taratura effettuata sulle Kinect permette di ottenere gli oggetti visualizzati dai sensori in scala; la seconda taratura servirà come convertitore di coordinate, passando così da pixel a millimetri.

Per entrambe le tarature effettuate abbiamo utilizzato lo stesso target (Figura 11), ovvero una lastra di materiale legnoso fissata a dei supporti in metallo, la quale può essere spostata lungo la linea che unisce la due Kinect permettendo così la taratura.

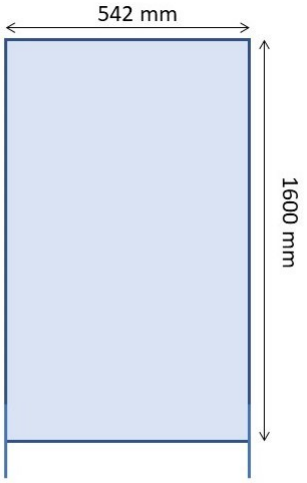

Figura 11 Target utilizzato per la taratura statica

La definizione delle due curve verrà descritta tramite la trascrizione di un programma in Matlab, che permette di ottenere la taratura istante per istante per tipologie di target differenti.

2.4.1. Curva di taratura per immagini in scala

Questa taratura statica effettuata sulle Kinect ha come scopo quella di far si che le immagini acquisite dai sensori siano in scala con il target reale.

Per la definizione di questa curva di taratura abbiamo preso come sensore di riferimento la Kinect 1 e subito dopo si è definita una tabella (Tabella 1) nella quale sono stati messi in relazione i valori misurati in millimetri della distanza effettiva tra la Kinect con il target Z, e la medesima distanza ma in valore di index ottenuto da Matlab successivamente all'acquisizione dell'immagine del target $indici\ u\_k$.

Z	Indici u_k
920	921
1080	1069
1300	1280
1530	1509
1590	1556
1730	1711
2070	2045
2310	2274

Tabella 1 Tabella dati per la taratura

Questi valori tabellati sono stati inseriti in Matlab sottoforma di matrice 8x2, formata da 8 righe e 2 colonne.

$$b = \begin{pmatrix} 920 & 921 \\ 1080 & 1069 \\ 1300 & 1280 \\ 1530 & 1509 \\ 1590 & 1556 \\ 1730 & 1711 \\ 2070 & 2045 \\ 2310 & 2274 \end{pmatrix}$$

Denominiamo d_z il vettore formato da tutti gli elementi della prima colonna della matrice, e con d_uk il vettore formato da tutti gli elementi della seconda colonna della matrice:

$$d\_z = b(:,1) \quad [1.5]$$

$$d\_uk = b(:,2) \quad [1.6]$$

Utilizzando il tool 'Curve Fitting' offerto dal software questi valori andranno a definire una curva di taratura statica lineare (Figura 12); ottenendo l'equivalente relazione matematica generata automaticamente da Matlab (Figura 13), che definisce sia l'equazione della curva che i valori e i parametri associati ad essa.

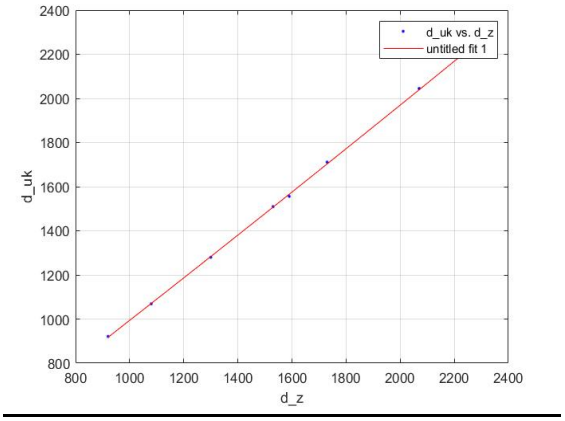

Figura 12 Curva taratura statica per immagini in scala

```
val =

     Linear model Poly1:
     val(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =      0.9792  (0.9648, 0.9935)
       p2 =       11.99  (-11.33, 35.31)
```

Figura 13 Fitresult curva taratura immagine in scala

Dunque grazie al curve fitting si riesce ad ottenere questa curva di taratura che verrà inserita poi, in un programma generale sempre in Matlab (Figura 14), affinché questa taratura possa essere ripetuta in tempo reale per ogni acquisizione effettuata dalla Kinect.

```
%% dimensioni di Z in mm
b=load('dati_Z_ind.txt');
d_z=b(:,1);
d_uk=b(:,2);

%% Fit: 'untitled fit 1'.
[xData, yData] = prepareCurveData( d_z, d_uk );

% Set up fittype and options.
ft = fittype( 'a+b/tan((x-c)/(-d))', 'independent', 'x', 'dependent', 'y' );
opts = fitoptions( 'Method', 'NonlinearLeastSquares' );
opts.Algorithm = 'Levenberg-Marquardt';
opts.Display = 'Off';
opts.MaxFunEvals = 6000;
opts.MaxIter = 4000;
opts.StartPoint = [0 0.075 1088.23 4621.07];

% Fit model to data.
[fitresult, gof] = fit( xData, yData, ft, opts );

% Plot fit with data.
figure( 'Name', 'untitled fit 1' );
h = plot( fitresult, xData, yData );
legend( h, 'd_uk vs. d_z', 'untitled fit 1', 'Location', 'NorthEast', 'Interpreter', 'none' )
% Label axes
xlabel( 'd_z', 'Interpreter', 'none' );
ylabel( 'd_uk', 'Interpreter', 'none' );
grid on
```

Figura 14 Definizione curva taratura in Matlab

2.4.2. Curva taratura per conversione coordinate

La seconda taratura statica verrà eseguita al fine di ottenere, per le immagini che le Kinect acquisiscono, una conversione delle coordinate trasformando i pixel in millimetri.

Come per la taratura statica precedente si misura la distanza effettiva tra la Kinect 1 e il target *Z*; poi si andrà a valutare volta acquisita l'immagine, il valore della larghezza del target in pixel rispetto ad una determinata distanza *deltax*. I valori che ottenuti vengono tabellati; in particolar modo i valori che abbiamo in pixel sono dati dalla differenza di due punti opposti della larghezza del target valutati in index (Tabella 2):

Z	deltax
920	338
1080	289
1300	239
1530	206
1590	197
1730	183
2070	147
2310	134

Tabella 2 Tabella dati per la taratura

Questi valori vengono inseriti in Matlab e come per la curva precedente saranno in forma matriciale sempre 8x2, ovvero 8 righe e 2 colonne.

$$a = \begin{pmatrix} 920 & 338 \\ 1080 & 289 \\ 1300 & 239 \\ 1530 & 206 \\ 1590 & 197 \\ 1730 & 183 \\ 2070 & 147 \\ 2310 & 134 \end{pmatrix}$$

Denominiamo d_m il vettore formato da tutti gli elementi della prima colonna della matrice; mentre con d_p il vettore formato da tutti gli elementi della seconda colonna, i quali sono convertiti da pixel a millimetri in Matlab.

$$d\_m = a(:,1) \quad [1.7]$$

$$d\_p = (a(:,2)/2)/270 \quad [1.8]$$

Anche in questo caso con il tool ' Curve Fitting' in Matlab otteniamo una curva di taratura (Figura 15) che, a differenza della prima, è una curva polinomiale.

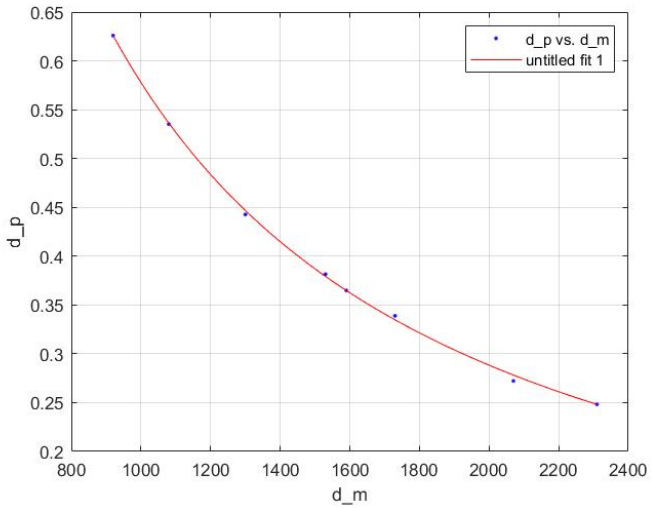

Figura 15 Curva taratura statica per convertire le coordinate

In questo caso si otterrà questa tipologia di curva di taratura non lineare poiché le telecamere infrarossi presenti nella Kinect formano con il target un angolo ϑ (Figura 16) che esprime la profondità in questa tipologia di sensori.

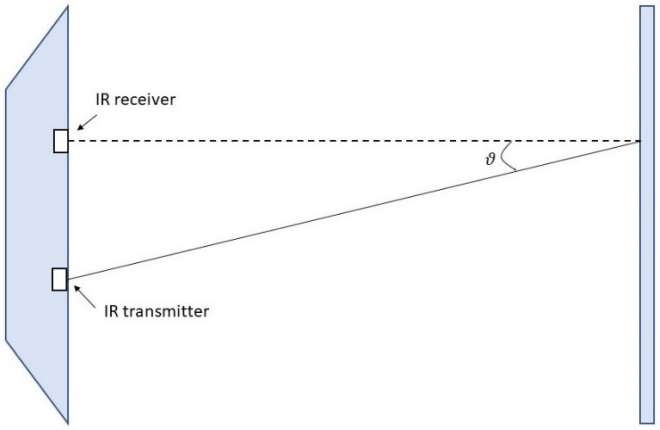

Figura 16 Profondità videocamere delle Kinect

Questo angolo verrà poi considerato nella relazione matematica generata da Matlab con il 'fitresult' (Figura 17) che si ottiene dalla curva di taratura polinomiale.

```
val =

     General model:
     val(x) = (atan(a/x)-b)/c
     Coefficients (with 95% confidence bounds):
       a =          312  (-53.97, 678)
       b =     0.007672  (-0.0188, 0.03414)
       c =       0.5101  (-0.02359, 1.044)
```

Figura 17 Fitresult curva taratura polinomiale

Come per la taratura statica precedente tramite il tool di Matlab riusciamo a definire la taratura statica utilizzabile per qualsiasi tipo di target; e verrà inserita nel programma generale che lavorerà in real time (Figura 18).

```
a=load('dati_eq_tara.txt');
d_m=a(:,1);
d_p=(a(:,2)/2)/270;
% d_p=270/(a(:,2)/2);


%% Fit: 'untitled fit 1'.
[xData, yData] = prepareCurveData( d_m, d_p );

% Set up fittype and options.
ft = fittype( '(atan(a/x)-b)/c', 'independent', 'x', 'dependent', 'y' );
opts = fitoptions( 'Method', 'NonlinearLeastSquares' );
opts.Display = 'Off';
opts.MaxIter = 4000;
opts.Robust = 'LAR';
opts.StartPoint = [0.699076722656686 0.890903252535798 0.959291425205444];

% Fit model to data.
[fitresult, gof] = fit( xData, yData, ft, opts );

% Plot fit with data.
figure( 'Name', 'untitled fit 1' );
h = plot( fitresult, xData, yData );
legend( h, 'd_p vs. d_m', 'untitled fit 1', 'Location', 'NorthEast', 'Interpreter', 'none' );
% Label axes
xlabel( 'd_m', 'Interpreter', 'none' );
ylabel( 'd_p', 'Interpreter', 'none' );
grid on
```

Figura 18 Definizione curva di taratura polinomiale in Matlab

3. Acquisizione e lettura video

In questo capitolo verrà esposta la scrittura del programma in Matlab con il quale si è definito un loop di immagini acquisite in tempo reale dalle due Kinect. Queste immagini vengono visualizzate come nuvole di punti, andando a distinguere e classificare il robot e i vari target che potranno muoversi o essere semplicemente presenti nel suo ambiente di lavoro.

3.1. Definizione parametri utili e programmazione camere

In questo capitolo andremo a descrivere in modo esaustivo il programma Matlab che sarà utilizzato per studiare l'ambiente di lavoro nel quale sono presenti il robot e i vari target che si muovono o che lavorano al suo fianco.

Molto importante è la definizione dei parametri fondamentali che serviranno per tutte le varie fasi della scrittura del programma.

Per prima cosa impostiamo la distanza massima e minima rispetto alle quali entrambe le Kinect acquisiscono un' immagine chiara. Queste grandezze sono definite in millimetri, come tutte le altre grandezze lineari del sistema.

$$massimo = 2000 \quad [1.9]$$

$$minimo = 1000 \quad [1.10]$$

Altro parametro fondamentale da definire è la matrice di pixel definita dalle due Kinect, ovvero una matrice 640x480. Imponiamo che:

$$r = 480 \quad [1.11]$$

$$c = 640 \quad [1.12]$$

Questi due valori, dopo aver effettuato le due tarature statiche, verranno manipolati, affinché si possa arrivare ad ottenere i valori *ele* e *az*, che rappresentano l'altezza e la profondità di un target, come una matrice che è descritta in funzione dei pixel.

Per la manipolazione del valore azimutale *az*, corrispondente alla profondità, definisco un vettore *rr* formato considerando le colonne con valori che vanno da 1 a 480; questi valori vengono di seguito elaborati riducendo il loro numero a 240 ovvero la metà del valore di *r*. Infine si va a specificare apportando le variazioni dei parametri il valore azimutale non più in millimetri ma in pixel (Figura 19).

```
rr=1:r;
rr=squeeze(rr-floor(r/2));
az=repmat(rr',1,c);
%figure, pcolor(az), shading interp
%figure, pcolor(c1), shading interp
az=reshape(az, c*r,1);
%figure, pcolor(az), shading interp
```

Figura 19 Manipolazione valore az

Il medesimo procedimento viene definito per il valore dell'elevazione *ele*, si specifica il vettore *cc* formato dalle colonne che vanno da 1 a 640; anche in questo caso i valori considerati sono pari alla metà del valore *c*, quindi 320. Si definisce quindi la

manipolazione del valore *ele* che non sarà più considerato in millimetri ma verrà espresso in pixel (Figura 20).

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

Figura 20 Manipolazione valore ele

Per far si che le immagini vengano acquisite dalle due Kinect, bisogna scrivere in Matlab delle funzioni che permettano l'acquisizione delle immagini (Figura 21). Consideriamo quindi le Kinect come input mentre tramite Matlab visualizziamo e lavoriamo le immagini prese dalle Kinect, e dunque consideriamo il pc come output.

```
%% Kinect 1
info = imaqhwinfo('kinect')

vid1 = videoinput('kinect', 2,'Depth_640x480','FramesPerTrigger',1);
src1 = getselectedsource(vid1);
triggerconfig(vid1,'manual');
start(vid1);

%% Kinect2
vid2 = videoinput('kinect', 4,'Depth_640x480'); %,'Depth_320x240'); '
src2 = getselectedsource(vid2);
triggerconfig(vid2,'manual');
start(vid2);
```

Figura 21 Acquisizione Kinect su Matlab

3.2. Acquisizione delle nuvole di punti

Le immagini acquisite da entrambe le Kinect vengono visualizzate come nuvole di punti rappresentanti coerentemente alla realtà, questo grazie alle tarature statiche che sono effettuate inizialmente e definite affinché si possano adattare a qualsiasi tipo di target che si presenta nel campo visivo delle Kinect.

L'intenzione principale è quella di esplicitare in Matlab un programma che permetta di definire le coordinate delle nuvole di punti che si acquisiscono partendo dalla considerazione di diversi parametri, quali: il video, le grandezze *az* e *ele*, il massimo e il minimo del campo visivo, e i valori *d_p* e *d_z* definiti con la taratura statica.

Le nuvole di punti che vengono acquisite dalle due Kinect, e sottoposte alla variazione delle coordinate, sono poi unite tra loro e da questo momento in poi nell'analisi del programma andremo a considerare direttamente le nuvole di punti matchate.

L'acquisizione della singola immagine viene estesa ad un determinato numero di frames maggiori di 1, in questo modo si effettueranno più acquisizioni generando un

loop di immagini consecutive. In questo modo otteniamo un programma in Matlab che funzioni in tempo reale, e che vada a valutare frame by frame quello che accade nell'ambiente di lavoro nel quale i vari target si trovano e cooperano assieme al robot.

3.2.1. Funzione ' filtrato_pol3': definizione dei punti

Nella definizione del programma per l' acquisizione delle nuvole di punti si è reso necessario esplicitare una funzione in Matlab che permettesse di ottenere delle grandezze già sottoposte a tarature statiche, e che tenesse in considerazione tutti i parametri indispensabili per ottenere un acquisizione corretta.

I parametri iniziale considerati come input della funzione sono: il valore c in pixel, i coefficienti di ele e az, il massimo e il minimo valore che riguarda il campo visivo delle Kinect e infine i valori d_z e d_p dati dalla taratura statica. Mentre in uscita si vorranno avere i valori delle coordinate x,y,z che tengano conto di tutti i parametri precedenti. Nella funzione viene definita come:

$$[x\ y\ depth\_sel] = filtrato\_pol3(c, az, ele, minimo, massimo, d\_z, d\_p)$$

[1.13]

dove il $depth\_sel$, è un vettore che deve essere tra il suo valore maggiore del minimo definito, e tra il suo valore minimo rispetto al massimo definito; inoltre questa grandezza deve essere in pixel come lo è il valore di $c=640$ (Figura 22).

```
depth=reshape(c,size(c,1)*size(c,2),1);
ind=find(depth>minimo & depth<massimo);
depth_sel=depth(ind);
```

Figura 22 Definizione depth_sel

Questo valore servirà per definire le coordinate x, y, z dell'immagine acquisita come coordinate cartesiane invece che in pixel; infatti tramite i valori manipolati di ele e az e grazie anche ai valori di d_z e d_p ottenuti dalla taratura statica si ottengono le coordinate necessarie come output di questa funzione (Figura 23).

```
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 23 Coordinate cartesiane definite dalla funzione

3.3. Identificazione dei cluster e definizione delle funzioni correlate

Per quanto riguarda l' operazione della clusterizzazione dei target presenti nell'ambiente di lavoro andremo a definire, anche in questo caso, dei parametri che permettono di leggere in modo coerente e corretto le immagini acquisite dalle due Kinect.

Lo scopo a questo punto è quello di cercare di rendere il più rapida e semplice possibile l'identificazione dei vari target attorno al robot; quindi siamo andati a valutare un numero di punti minori rispetto a quelli che formano la nuvola.

Questo viene permesso nel momento in cui abbiamo considerato un valore minimo della dimensione di questi cluster che è pari a 200, una deviazione standard pari a 4 e una decimazioni dei punti che formano la nuvola pari a 10. Con queste informazioni definiamo un vettore Y formata dai valori dati dalle coordinate ottenute dall'acquisizione che vengono però decimate secondo il parametro definito.

Questo vettore di parametri decimati viene utilizzato insieme alla distanza minima e alla minima dimensione del cluster per definire una funzione generale che serve a definire il cluster stesso e il numero di target presenti nell'ambiente.

Questa funzione globale sarà definita a sua volta da altre funzioni che permettono: di filtrare le nuvole di punti che formano i vari cluster eliminando i punti che vengono considerati rumori, di ridimensionare il cluster compattando i punti isolati, e infine di identificare i vari target come degli ellissoidi rendendo possibile così considerare anche un fattore di sicurezza per il target stesso (Figura 24).

```
function [id_cluster,baricentri,Y,n_cl]=clusterizzazione_rt(Y,minDistance,min_cl_size)
    % Cut NaN
    iNaN=isnan(Y(:,1))&isnan(Y(:,2))&isnan(Y(:,3));
    Y=Y(~iNaN,:);
    % Cut Z>0 values
    Y=Y(Y(:,3)<0,:);
    %%Cluster & visualize
    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);
    [baricentri]=cluster(Y,id_cluster);
end
```

Figura 24 Funzione clusterizzazione

3.3.1. Funzione "filtro_clu"

Dall' acquisizione abbiamo notato come le nuvole di punti presentavano dei punti che non facenti parte del target considerato e che successivamente andavano a creare delle problematiche sulla ricerca del cluster stesso.

Per questo motivo abbiamo definito una funzione che servisse a filtrare questi punti estranei al cluster in modo da ottenere delle nuvole di punti ben definite e facilmente deducibili.

Gli input considerati per definire questa funzione sono: il numero minimo di cluster, l'identificazione del cluster e il numero dei cluster stessi. Questi ci permettono di ottenere l'identificazione corretta dei cluster e del vettore delle coordinate decimate della nuvola di punti Y.

3.3.2. Funzione "ridimensionamento_clu"

Il ridimensionamento del cluster che avviene tramite la funzione permette di compattare i punti che risultano isolati rispetto la nuvola di riferimento. I punti isolati vengono identificati in quanto presentano un valore minore del $min\_cluster$.

Il risultato di questa funzione è l'individuazione di cluster di punti isolati che verranno identificati come notanumber (NaN). Questo tipo di identificazione renderà possibile, tramite la funzione descritta nel paragrafo 3.3.3 Funzione "cluster", l'eliminazione di questi punti isolati.

3.3.3. Funzione "cluster"

Con questa funzione si vuole includere il cluster, precedentemente identificato, all'interno di una figura geometrica, ovvero un' ellissoide.

Per prima cosa vengono tagliati i punti isolati definiti come NaN e successivamente cerchiamo il robot tra tutti i cluster presenti nello spazio di lavoro e lo identifichiamo come jj (Figura 25).

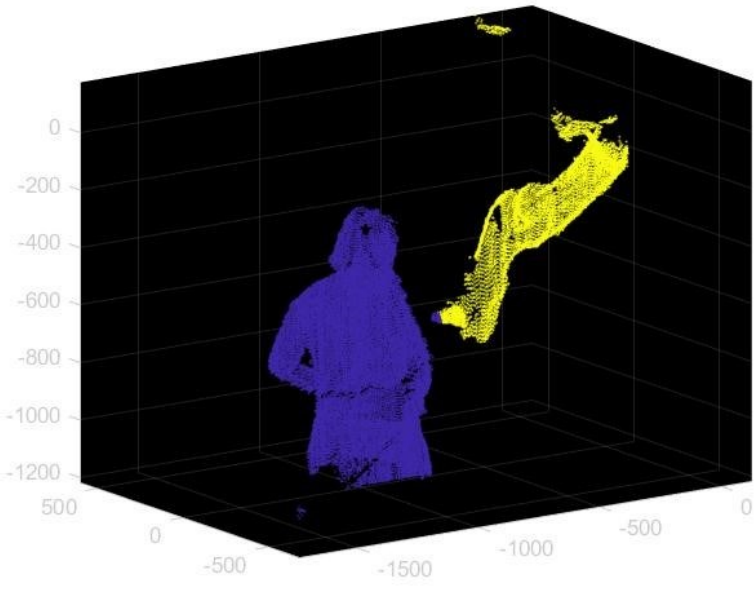

Figura 25 Visualizzazione cluster

Le coordinate del robot possiamo indentificarle in un vettore di zeri in quanto è l'origine del sistema di riferimento. Avendo definito la sua posizione andremo ad escludere il robot dalla visualizzazione globale dei cluster.

Poniamo come kk il numero di cluster nell'ambiente di lavoro e definiamo che questo non corrisponda al cluster del robot. Questi valori dei kk vengono poi associati a Y, il quale rappresenta il valore delle nuvole di punti decimate.

Successivamente siamo passati dalla definizione degli ellissoidi per i singoli cluster.

Il primo step è stato quello di definire una funzione che generi gli ellissoidi. Questo è possibile se in essa definiamo le coordinate dell'asse verticale e del centro dell'ellissoide stesso, tenendo conto della distanza e della deviazione standard dei punti.

Nel secondo step andiamo a scrivere il vettore $baricentro$ formato dalle coordinate del centro dell'ellissoide e di conseguenza plottiamo quanto ottenuto.

Il risultato di questa operazione è appunto quello di visualizzare nell'ambiente di lavoro i vari cluster identificati all'interno di ellissoidi (Figura 26); questo permette di aggiungere un margine di sicurezza per il riconoscimento del target da parte del robot.

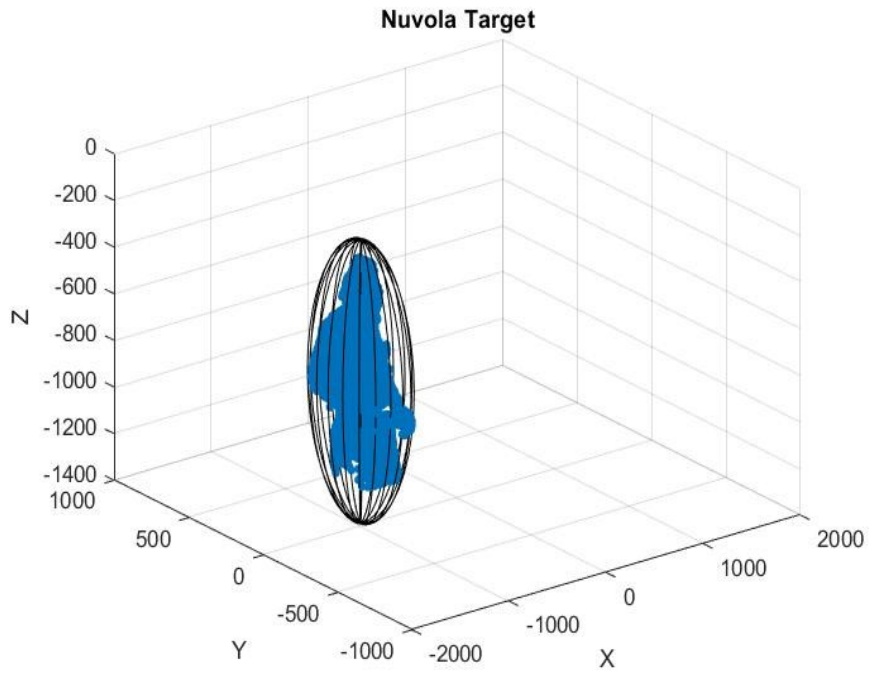

Figura 26 Cluster assimilato all'ellissoide

4. Risultati

In questo capitolo vengono illustrati e discussi i risultati ottenuti dallo sviluppo di un programma in Matlab, che permette di verificare in tempo reale ciò che accade in un ambiente di lavoro nel quale si presenta una collaborazione tra il robot definito nel banco di prova e i vari target che possono essere presenti.

Le prime misure sono state eseguite per verificare e valutare il funzionamento del programma, partendo dall'acquisizione di pochi frames.

In questo primo step si è considerato un solo target nelle vicinanze del robot; questo target veniva correttamente identificato come nuvola di punti in tempo reale (Figura 27).

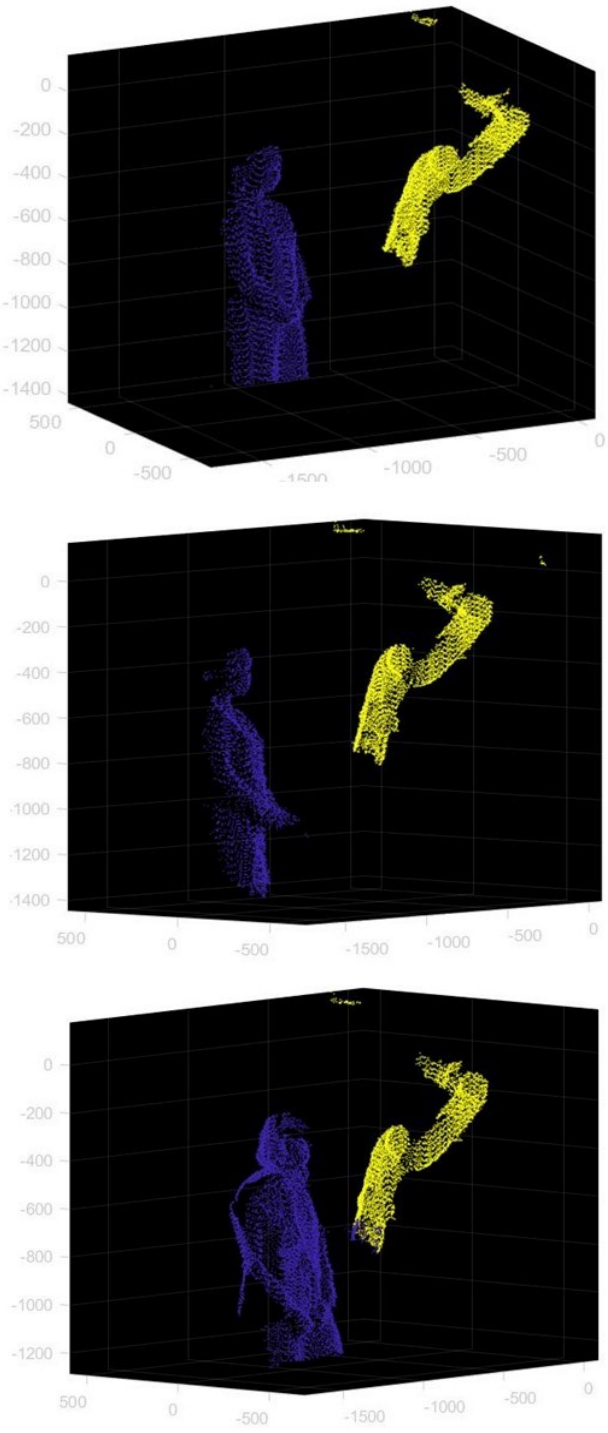

Figura 27 Acquisizione di pochi frames delle nuvole di punti dell'uomo nelle vicinanze del robot

Una volta stabilito che quanto è acquisito dalle due Kinect corrisponde alla realtà, sono state definite e utilizzate le funzioni in Matlab per escludere il robot dalla visualizzazione, mentre il singolo cluster viene definito all'interno di un' ellissoide (Figura 28).

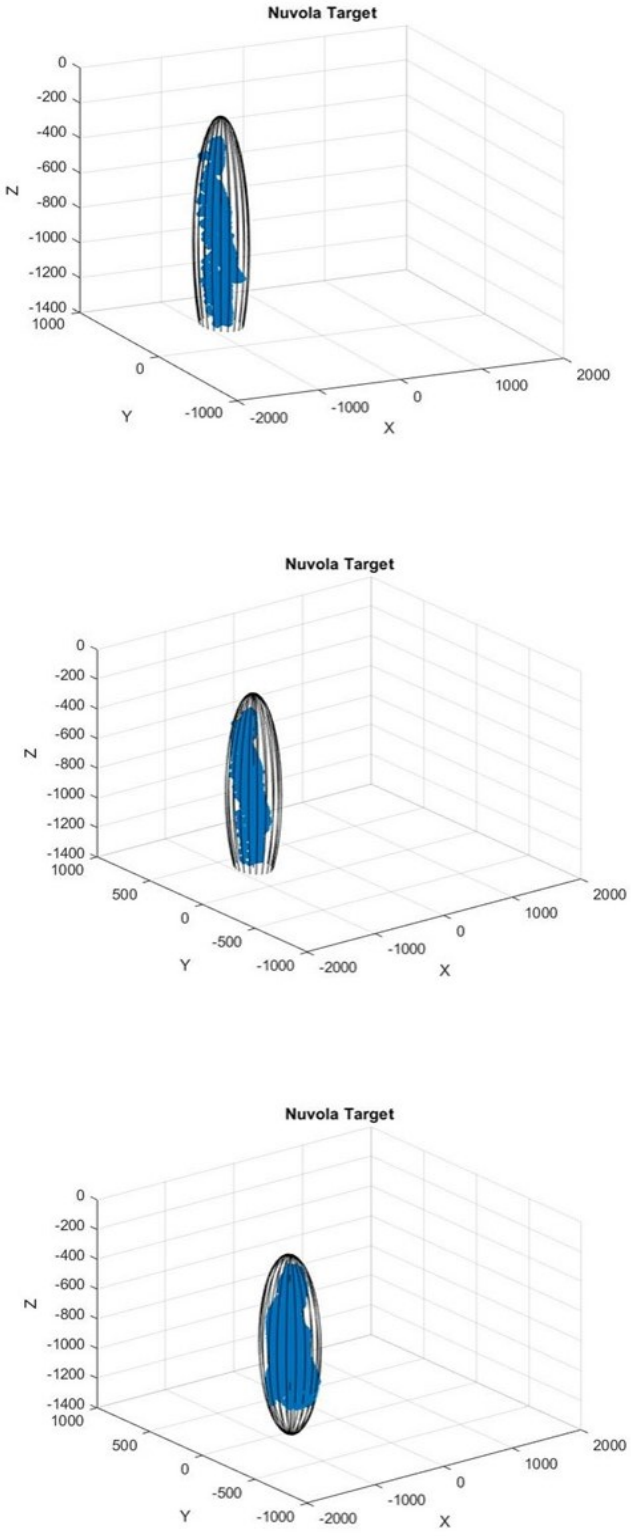

Figura 28 Definizione del cluster nell'ellissoide

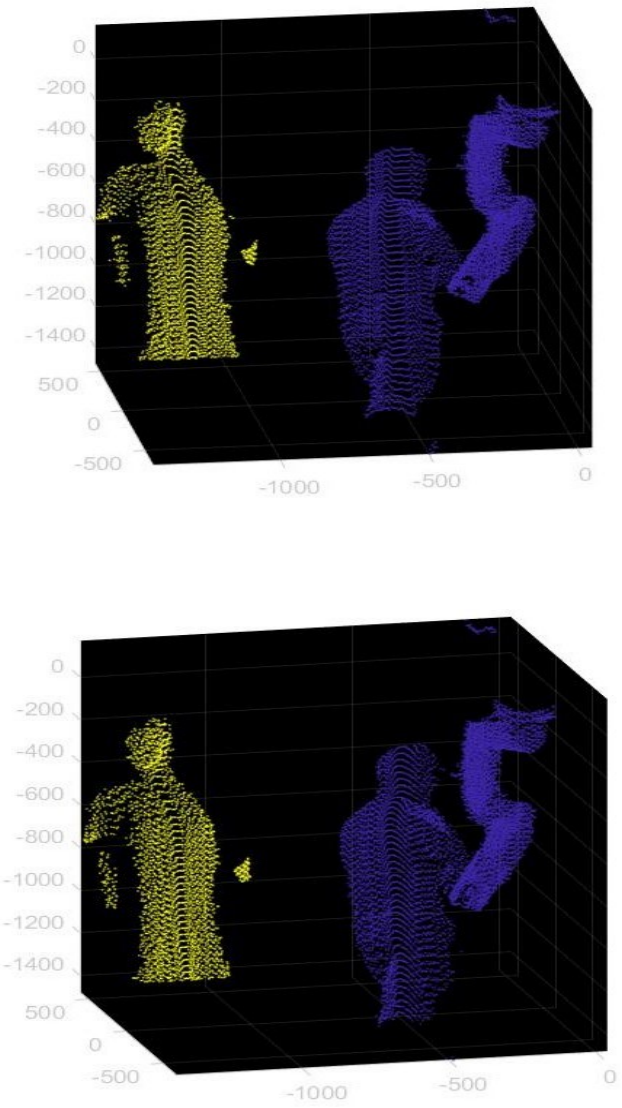

Figura 29 Nuvole di punti di due persone nelle vicinanze del robot

Si è visto dunque che per un solo target questo programma funziona in modo corretto e coerente con quella che è effettivamente la realtà. Acquisendo sempre pochi frames, abbiamo esteso le considerazioni e le misure anche ad un ambiente di lavoro con più target che lavorano e collaborano con o nei pressi del robot(Figura 29) (Figura 30).

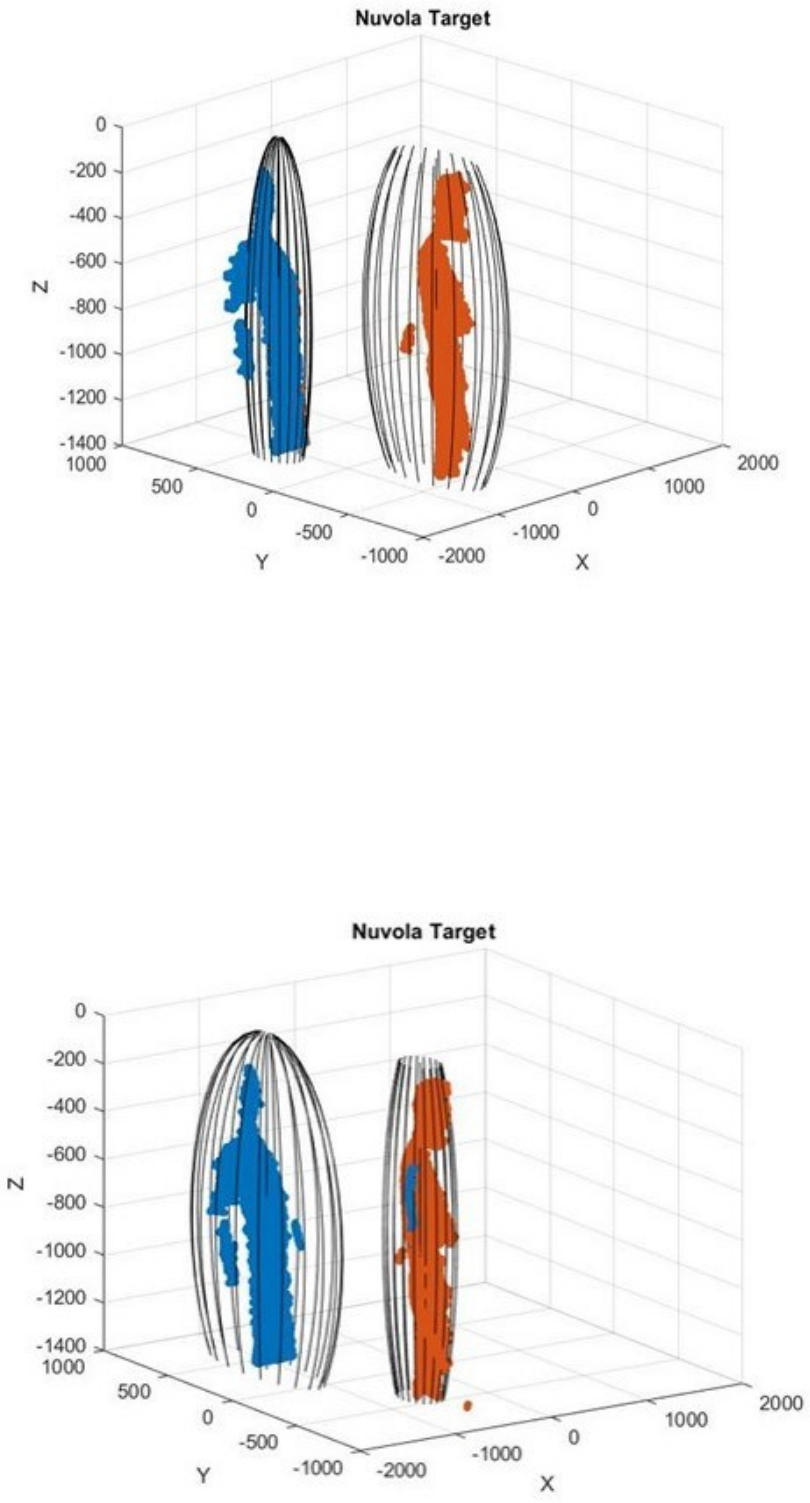

Figura 30 Definizione dei due cluster nei rispettivi ellissoidi

Verificato che quanto viene scritto nel programma di Matlab fa si che si possano acquisire una serie di immagini in tempo reale e che si distinguono chiaramente gli oggetti e persone presenti nell'ambiente circostante al robot, si estende l'acquisizione e la lettura per un elevato numero di frames.

Facendo ciò abbiamo notato che nel momento in cui si vanno a definire i vari cluster all'interno degli ellissoidi nell'ambiente di lavoro talvolta non tutti i punti che formano la nuvola vengono assimilati al cluster ma vengono portati al di fuori dell'ellissoide e questo comporta la formazione di un nuovo cluster (Figura 31).

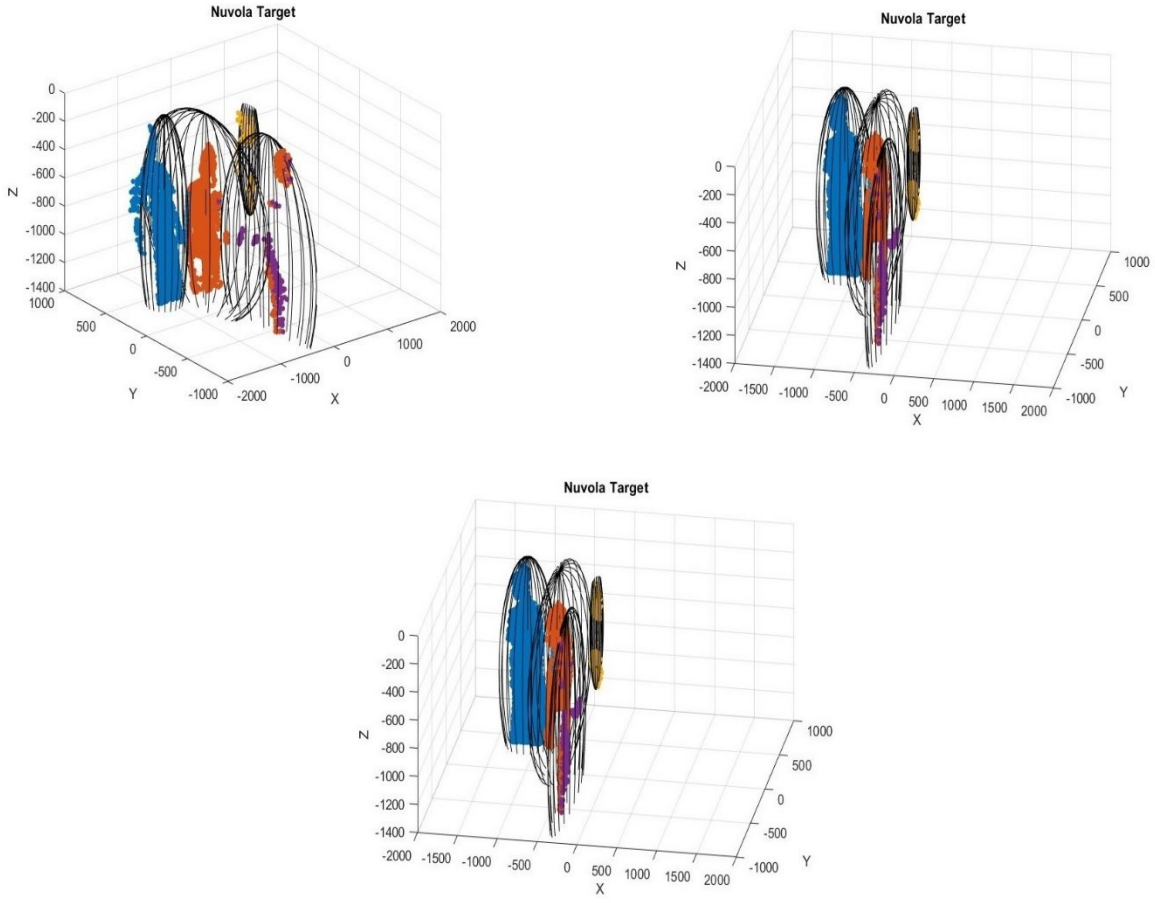

Figura 31 Formazioni ellissoidi per punti isolati

Questo però non risulta essere un problema che compromette la totale validità del progetto in quanto grazie alle opportune funzioni di filtraggio si riesce a minimizzare questo effetto, e rende comunque possibile comunicare al robot che in quel determinato punto è presente un ostacolo.

Il programma è stato poi valutato su 100 frames, per accertare che anche su un elevato numero di frames il programma funzionasse ma soprattutto risultasse valido.

Di seguito vengono riportare alcune delle immagini acquisite della Kinect, in questi 100 frames, nel quale nonostante le nuvole di punti non siano perfettamente identificate vengano comunque definite come dei cluster e inglobate nei rispettivi ellissoidi. (Figura 32) (Figura 33)

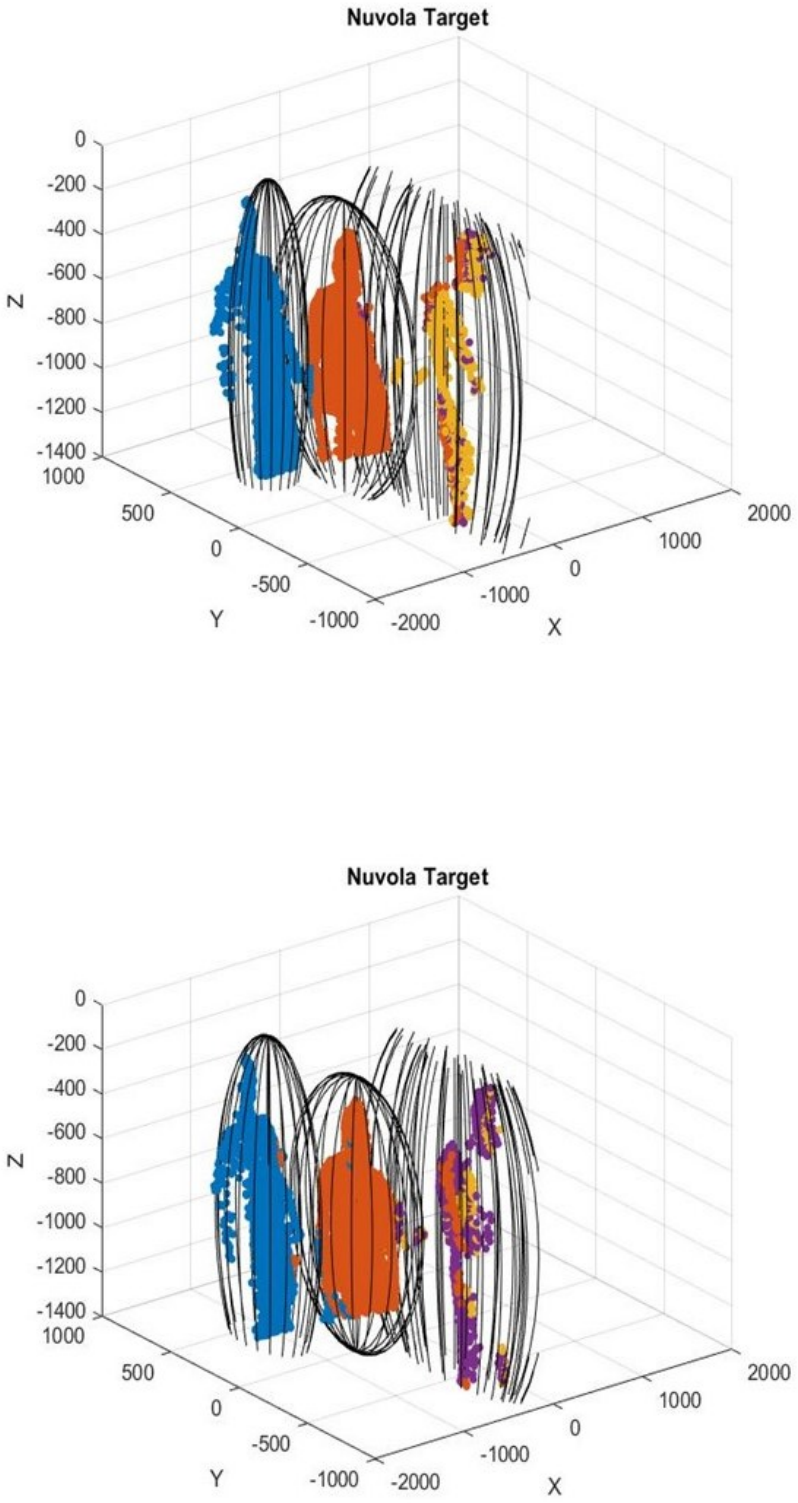

Figura 32 Acquisizione Kinect

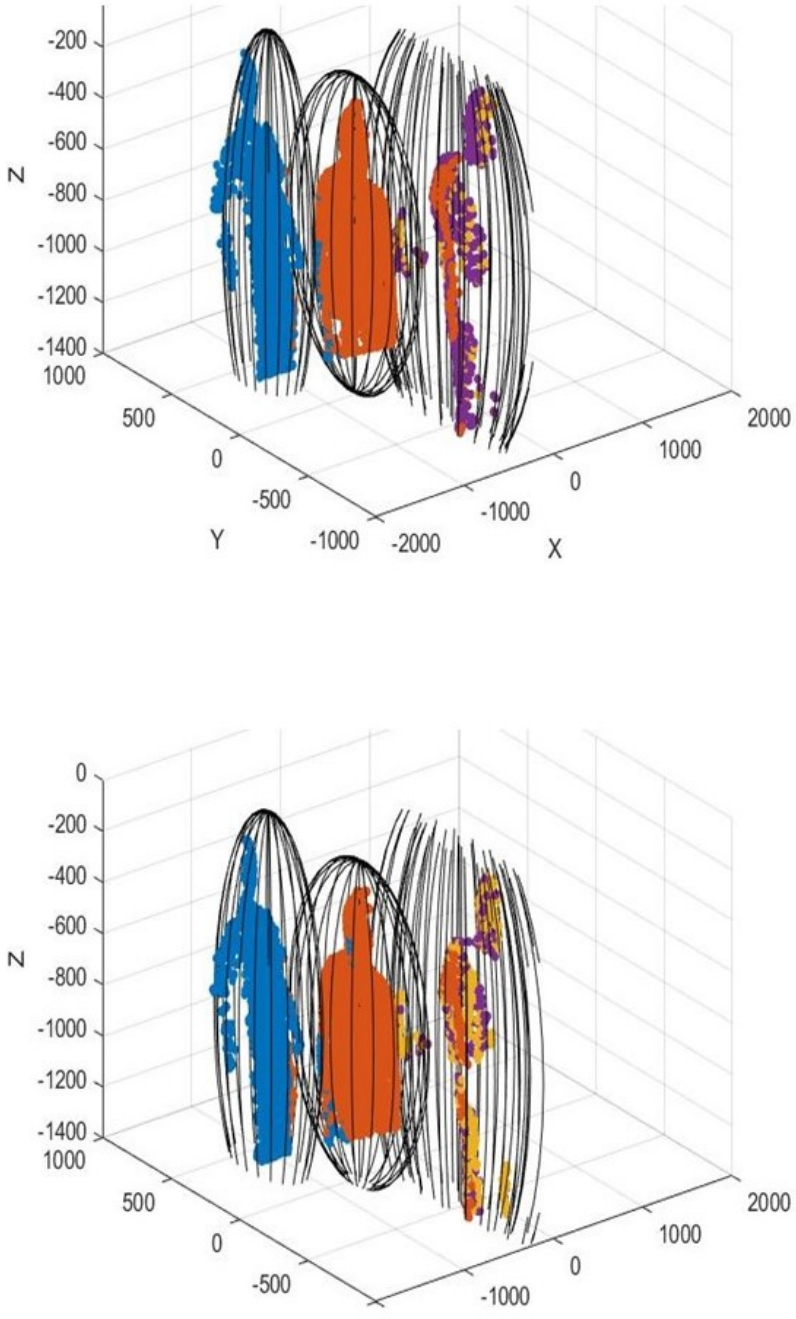

Figura 33 Acquisizione Kinect

CONCLUSIONI

L'obiettivo di questa tesi è quello di avere un programma in Matlab che permette, grazie all'utilizzo delle Kinect, l'acquisizione e la lettura di immagini di un ambiente di lavoro dove il robot collabora con i vari target presenti.

Tramite le misure effettuate è stata verificata la validità del programma sviluppato. Si è visto come la distinzione tra target e tra robot avviene in modo chiaro, mentre nella definizione dei vari cluster si nota come non sempre questi siano distinti in modo chiaro ma presentano degli errori che possono far considerare dei cluster in più rispetto a quelli che sono nella realtà.

Questo programma può essere migliorato applicando una matrice di rototraslazione, definita dopo aver effettuato una taratura statica, che permette un perfetto appaiamento delle nuvole di punti acquisite anche in presenza di più Kinect.

Un altro miglioramento che può essere apportato è quello di rendere ancora più veloce il programma scritto in Matlab tramite la definizione di un 'while loop' nel quale possiamo valutare le acquisizioni tramite due valori diversi di pixel. Si può infatti pensare al loop in due modi: nel primo caso andiamo a considerare per l'acquisizione una matrice di pixel pari a 640x480, mentre nel secondo caso si può avere un'acquisizione con una matrice di pixel inferiore pari a 320x240.

In questo modo si otterrebbe un programma funzionante in tempo reale che permette una corretta valutazione di ciò che accade in un ambiente di lavoro nel quale ci sarà una collaborazione tra il robot e l'uomo.