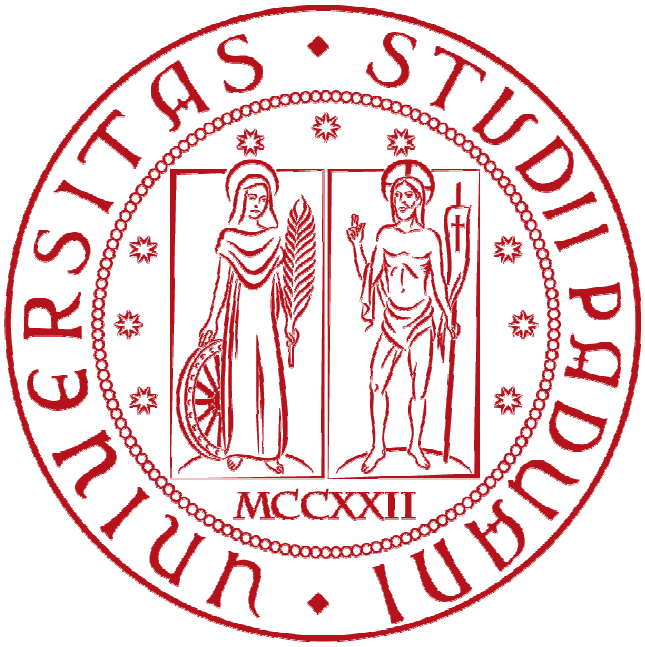
UNIVERSITÀ DEGLI STUDI DI PADOVA



FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria dell'Informazione

TESI DI LAUREA TRIENNALE

Sviluppo di un robot inseguitore di colori, con visione omnidirezionale

Relatore:

Prof. Michele MORO

Laureando:

Marco PESCE

Anno Accademico 2012–2013

A Riccardo e Mirna .

Sommario

Il presente documento tratterà le tematiche riguardanti lo sviluppo di un robot mobile, atto a ricercare all'interno del suo campo visivo oggetti statici e mobili, identificandoli per colore sfruttando l'installazione su di esso di un sistema a visione omnidirezionale. L'analisi partirà dalla matematica governante la geometria ottica dei sistemi a singolo punto di vista efficace, passando alla costruzione e calibrazione di tali sistemi e infine studiando l'implementazione pratica del robot inseguitore di colori. Il sistema omnidirezionale è composto dalla telecamera NXTCam V2, ortogonale a uno specchio di natura iperboloide, e il robot si basa su tecnologia LEGO MINDSTORMS®. La maggiore difficoltà da affrontare sarà la costruzione di un apparato così complesso e articolato su di una piattaforma, quale LEGO MINDSTORMS, di natura prettamente didattica e dimostrativa.

Indice

Capitolo 1

Introduzione

Affinché un robot possa interagire con l'ambiente in cui esso si trova in modo integrale, occorre porre un occhio di riguardo all'implementazione del suo apparato sensoriale, pensando di creare un interfaccia macchina-mondo reale, in maggior modo esauriente, migliorando così l'interazione dellautoma con tutto ciò che lo circonda. In questa tesi si studierà la possibilità di installare su di un modello di robot della LEGO MINDSTORMS®[S1] un sistema di visione omnidirezionale con la facoltà di riconoscere oggetti, eventualmente mobili, del quadro ambientale in cui esso si trova.Possedere una vista totale della scena può essere molto vantaggiosa in numerose applicazioni, integrandola a tutte le informazioni vengono memorizzate in un singolo fotogramma. Lo sviluppo di un robot inseguitore sceglie di affrontare differenti problemi di natura implementativa-sperimentale, in quanto la difficoltà maggiore si cela nella migrazione di metodi su tecnologie all avanguardia e specifiche verso una tecnologia più semplice come LEGO MINDSTORM®, indirizzata ad un uso didattico.

Per un sistema di visione panoramico, ci avvaliamo di una telecamera omnidirezionale, che per definizione, è un sistema che fornisce un campo di vista a 360° della scena. Tale panoramica può essere ottenuta utilizzando più telecamere sincronizzate, combinazioni di telecamere prospettive e specchi, o solo telecamere con obiettivi grandangolari. In questo lavoro si utilizzerà un apparato composto da camera NXTCamV2[S2] di Lego Mindstorm,posizionata ortogonalmente a uno specchio iperboloide.Tale installazione cela un' importante e corposa modellizzazione matematica dellottica geometrica attraverso lenti e specchi, studiata per consentire di creare un sistema a singolo punto di vista efficace Single View Point, in seguito abbreviato con SVP, utile a poter predire la posizione di un dato punto dello spazio, calcolando come il raggio di luce che esso riflette interseca il piano iperboloide dello specchio, possiamo utilizzare un sistema di proiezione omogeneo degli oggetti appartenenti alla scena.

Un sistema SVP, come verrà approfondito in seguito, richiede una certa cura nel

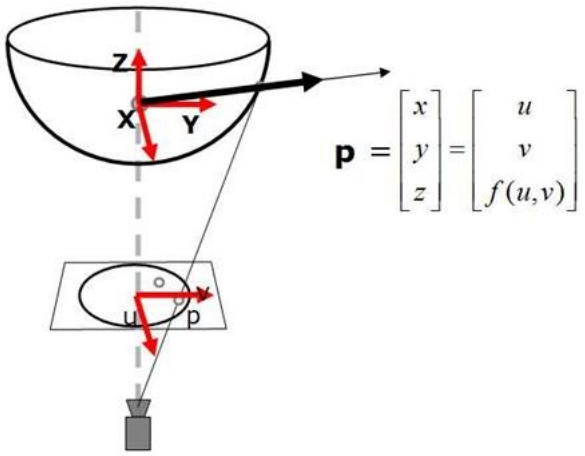


Figura 1.1: Telecamera Omnidirezionale

montaggio e nella calibrazione della fotocamera, utile a eliminare varie distorsioni, conseguenze di imperfezioni intrinseche di diottri e specchi di tale struttura e del montaggio manuale. Per tale calibrazione, si è scelto di utilizzare una procedura assistista grazie all'ambiente interattivo di calcolo MATLAB e un suo tool specifico, OcamCalib[S3]: strumento MATLAB[S4], che consente, tramite una semplice interfaccia di calibrare il sistema Camera Specchio Conico, ergo creare un modello matematico per il calcolo delle proiezioni da fotogramma ,catturato dalla Cam, al mondo reale in un sistema SVP. Il robot è sviluppato sulla tecnologia Brick di Lego, cioè è costituito da un corpo principale, appunto il Brick, mattoncino programmabile con linguaggio di programmazione NXC, C based, organizzato a task asincroni, editato tramite il semplice e duttile editor Bricxcc[S6], semplice ide per la programmazione brick .

La funzione di trasformazione di coordinate da 2D a 3D è stata trasferita da linguaggio matlab a una versione semplificata in NXC. Ovviamente il robot, non potendo sviluppare un movimento in verticale, trovandosi su un piano fisso, non valuterà la coordinata z, limitandosi a muoversi nel suo piano di appartenenza. Linseguimento di colori è stato realizzato grazie a una funzionalità intrinseca presente nel firmware della NXTCam, la quale è in grado di riconoscere nei vari frame da lei processati, un particolare intervallo di colori appartente al modello RGB.Questa modalità detta Tracker Object permetterà allinseguitore di identificare nel fotogramma scattatato dalla camera una regione dove si troverà il target.Queste coordinate degli estremi d'area dell'oggetto verranno processate per eliminare errori e per effettuare l'effettiva trasformazione di coordinate, e così il robot si muoverà in direzione opportuna. I movimenti del robot sono stati curati per creare un inseguimento realistico, per una futura dimostrazione a scopo didattico, ovviamente

senza trascurare i limiti tecnici della strumentazione posseduta. I capitoli seguenti sono stati scritti ed enumerati in modo tale da rappresentare una virtuale creazione del robot finale, difatti seguendo tale percorso di analisi, di prove sperimentali e implementative si è arrivati al reale e concreto robot finale.

Capitolo 2

Strumenti utilizzati

2.1 Lego® Mindstorms®

Lego® Mindstorms® è una tecnologia di Lego®, che unisce ai basilari mattoncini Lego, componenti della serie Lego Tecnhics, come ingranaggi,ruote,distanziatori e parti pneumatiche a una serie di componenti programmabili: il mattoncino intelligente NXT e altre varie componenti della più comune robotica hobbystica come sensori di luce,servomotori, sensori di contatto ecc.., la serie Lego Mindstorm Nxt fornisce un intero kit per lo sviluppo di robot lego, per l ambito puramente educativo e didattico: difatti la programmazione e la creazione di robot è resa possibile da manuali, tecniche e strumenti di facile e rapido uso e apprendimento.

2.1.1 NXT® Mindstorms®

Il mattoncino intelligente NXT, detto anche Brick Nxt, cuore di ogni robot realizzato con suddetta tecnologia presenta al suo interno la completa integrazione di parti elettroniche, che lo rendono un ottimo mini-calcolatore in grado di offrire un potenziale in termini di funzioni tra le più ampie di tutta la robotica hobbistica. Il brick presenta al suo interno:

- Il mattoncino intelligente NXT,dotato di un processore a 32 bit Atmel AT91SAM7S256 (classe ARM7) a 48 MHz. con 256k flash memory 64k RAM, un coprocessore 8 bit Atmel ATmega48 (classe AVR: è un RISC a 8 bit) a 8 MHz, con 4k flash e 512 byte RAM
- uno schermo LCD con una risoluzione di 60x100 pixel
- una porta USB 2.0
- connettività Bluetooth

Figura 2.1: Il Brick

- Il Mindstorms NXT possiede quattro porte di ingresso e tre di uscita, ma avendo delle connessioni digitali, sarà possibile aumentarne il numero con dei moduli esterni
- Altoparlante integrato, da 8 kHz.
- Il mattoncino richiede 6 batterie di tipo AA (che potranno anche essere ricaricabili) oppure la Batteria al Litio della casa.

É possibile programmare il Brick tramite più linguaggi di programmazione: ufficiali, forniti da Lego prettamente grafici, a sottolineare la natura didattica del kit di sviluppo, ed altri ufficiosi, basati su linguaggi di programmazione più rilevanti come Java e C, oppure veri e propri indipendenti linguaggi di programmazione. La direzione presa è stata per il linguaggio NXC,Not eXactly C, linguaggio di programmazione ad alto livello, simile al più famoso C, basato su un altro linguaggio di programmazione brick detto NBC, Next Byte Codes.Il linguaggio pratico e effeciente, è di tipo imperativo e procedurale organizzato task asincroni, per la possibilità di gestire situazioni complesse con poche e semplici righe di codice, questo ha avuto una grande diffusione soprattutto nella competizione RobotCup Junior.

2.1.2 NXTCam V2 ®

Il modulo Hw di maggiore importanza utilizzato nel progetto è stata la NxtCamV2,componente plug and play gestibile attraverso il brick NXT tramite una delle 4 connessioni I^2C di entrata. Il sistema è una camera che consente la gestione di frame in tempo reale, oltre al corpo diottrico e il sensore di immagine, è creata su un processore accessibile tramite interfaccia microUsb. Linterfaccia della NxtCam consente l'accesso a informazione extra dell'immagine, come le coordinate di delineazione di un oggetto target che possiede determinate caratteristiche di colore.La camera può tener traccia fino a 8 oggetti associati a un intervallo di valori del modello RGB. Una prima configurazione della camera può avvenire tramite

connessione USB ad un PC, la compatibilità è garantita per i maggiori sistemi operativi come Windows XP e Windows Vista. Gli oggetti ricercati dalla Cam, in un'immagine, sono identificati tramite un opportuno valore RGB: questo valore deve essere caricato nella camera,appunto in un registro detto ColorMap, questo registro può contenere fino ad un massimo di 8 valori RGB, conseguenza di ciò è che il processo di tracciamento può ricavare posizione e informazione di al più otto elementi. La colormap come tutte le istruzione di controllo per NXTCAM è interfacciabile tramite protocollo I2C, classica comunicazione seriale tramite un master ed un slave tramite mezzo bifilare comune nella comunicazione tra circuiti integrati, in questo ambito tra Camera e Computer o tra Camera e Brick NXT.

Esistono due modalità di tracciamento di coordinate di oggetti, l Object Tracker e il Line Tracker; entrambi si basano sull'utilizzo di un registro digitale contenuto all'interno del sistema detto Colormap. LObject Tracker e il Line Tracker si distinguono come metodi di tracciamento sull uso del registro ColorMap e dalla gestione del suo contenuto.Infatti object tracker utilizza tutti gli 8 colori della ColorMap e lavora per evitare la sovrapposizione degli oggetti.Limitato è invece l'utilizzo della ColorMap nel caso del Line Tracker che utilizza solamente il primo colore contenuto nel registro. Per un utilizzo ottimale del sensore, viene consigliato dal costruttore di utilizzare per l illuminazione lampade fluoriescenti, onde a evitare rilevazioni di colori alterati dalla realtà; altri tipi di impostazioni come contrasto e luminosità possono essere fornite tramite I^2C.

2.2 Specchio iperboloide

Per camera omnidirezionale si intende una camera che consente una visione panoramica a 360°: tale tipo di sistema viene ottenuto combinando un certo tipo di specchi con opportune camere.Queste componenti insieme formano un sistema di visione con singolo punto di vista efficace permettendo di ricreare efficacemente la reale proiezione di un punto nello spazio, catturato nell immagine della camera. Gli specchi sferici, ottenuti da calotte sferiche, possono essere concavi o convessi, secondo che la riflessione avvenga sulla superficie concava o su quella convessa. Lo specchio dà un'immagine reale di un punto sull'asse ottico a distanza dal vertice maggiore della focale; se invece il punto è a distanza dal vertice minore della focale l'immagine è distorta. Per gli specchi sferici valgono le approssimazioni di Gauss: le immagini sono con buona approssimazione stigmatiche per raggi poco inclinati rispetto all'asse ottico e per angoli minori di 1,5°. L'immagine data da uno specchio sferico convesso è sempre distorta, diritta e rimpicciolita. Oltre agli specchi sferici vengono spesso utilizzati in varie applicazione come telescopi o riflettori di automobili, specchi nei quali le superfici riflettenti hanno forma di paraboloidi di rivoluzione, di ellissoidi di rivoluzione e di iperboloidi di rivoluzione, i quali riesco-

no fondamentalmente ad allargare il campo visivo. Nel progetto in questione, la scelta è ricaduta su un catadiottrico di tipo iperboloide che consentirà una visione omnidirezionale del mondo reale, in quanto la camera scatterà frame dello specchio che catturerà tutte le onde luminose per un completo angolo giro di 360°.

2.3 Software di supporto

Fondamentale ai fini della realizzazione del progetto, è stato appoggiarsi a strumenti software robusti e affidabili, adatti a vari compiti ma utilizzati per fini specifici come: configurazione della singola camera, la calibrazione del sistema camera-specchio, e la programmazione finale del brick nxt.

2.3.1 MatLab®

Matlab® (acronimo delle parole inglesi MATrix LABoratory) è un software basato sulla manipolazione di matrici, molto utilizzato nel campo della ricerca scientica, non solo matematica, per la sua grande portabilità (infatti è disponibile sia per grandi workstation che per comuni personal computer), unita ad una notevole facilità d'uso e alle potenzialità di calcolo. Inoltre luso di Matlab è reso facile dalla presenza di un manuale dei comandi in linea, che può essere invocato tramite il comando help, e dalla presenza del comando demo che presenta numerosi e signicativi esempi di applicazioni di tutte le funzioni Matlab. Piuttosto che l'utilizzo del programma Matlab, nello specifico si è implementata una procedura contenuta in toolbox, strumento software estrinseco dellambiente, OcamCalibToolbox, in seguito descritto.

2.3.2 OCamCalib®

Il toolbox, creato dal Dott. Davide Scaramuzza, consente all'utente di calibrare ogni camera panoramica oppure ogni sistema camera specchio centralizzate, per ottenere un singolo punto di vista efficace, questo software si basa prevalentemente sullo svolgimento di due passi fondamentali ai fini della calibrazione. Primo, richiede all'utente di collezionare un set di immagini relative a una scacchiera standard in diverse posizioni e angolazioni.Succesivamente è richiesto di estrarre i punti degli angoli delle immagini precedentemente caricate, così da ottenere un una griglia di punti utili al processo matematico per la proiezione dei punti, proveniente dall immagine spedita dalla camera, nel mondo reale. Oltre a tale calibrazione il toolbox permette anche l'utilizzo di due funzioni CAM2WORLD e WORLD2CAM, che effettuano trasformazioni di coordinate rispettivamente dal 2D allo spazio 3D

e viceversa, eliminando vari tipi di errori raccogliendo e utilizzando i risultati della calibrazione precedentemente descritta.

Queste funzioni verranno poi migrate a codice NXC in una loro versione più pratica ma sufficiente per tale progetto.

2.3.3 Bricxcc

Il software utilizzato per la programmazione dei Robot NXT è BrixCC. BrixCC è un ambiente di programmazione che permette la scrittura dei programmi, la loro compilazione, la ricerca degli errori, ed il trasferimento del codice al robot. Può gestire programmi scritti in vari linguaggi, ma quello che è stato utilizzato per l'implementazione dei robot è NXC (Not Exactly C), molto simile al C. BrixCC è un programma open source, liberamente scaricabile ed utilizzabile, e gira su sistemi operativi Microsoft.

2.3.4 NXTCamView

Per visualizzare il campo visivo dell'NxtCam, acquisire l'immagine per analizzare e configurare la Colormap per processi onboard, è necessario installare e utilzzare il software di configurazione appropriato sul proprio pc NXTCamView[S5]. Per MS-Windows XP/ Vista e dai test effettuati anche MS-Windows 7,scaricare il software da :
`http://nxtcamview.sourceforge.net/`

Capitolo 3

Visione Omnidirezionale

Un sistema omnidirezionale è formato dalla composizione di telecamere e specchi conici. La telecamera presa in esame appartiene alla classe di modelli Pin-hole, basato sul principio della camera oscura illustrato in Fig. 3.1.

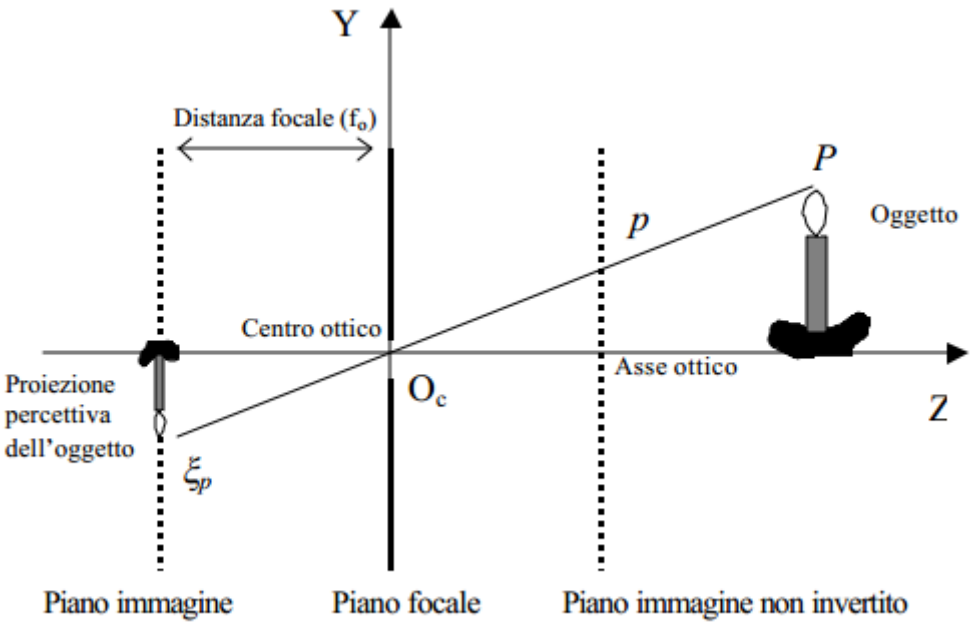


Figura 3.1: Modello camera Pin-Hole.

Si consideri una scatola chiusa in cui è stato praticato un piccolo foro su una delle sue superfici. Si indichi con F il piano in cui è presente il foro. Da un oggetto, per esempio una candela, posto dinnanzi il piano F, partono raggi luminosi per

emissione diretta o riflessa. Alcuni di questi raggi finiscono nel foro e formano sul piano I, dove è presente il sensore digitale: dispositivo fotosensibile costituito da una matrice di fotodiodi in grado di trasformare un segnale luminoso in un segnale elettrico, l'immagine invertita della candela. I piani F e I sono detti rispettivamente piano focale e piano immagine (o piano retina) e la distanza tra loro è la distanza focale f_0. Il punto dove è stato praticato il foro si indica come centro ottico mentre la retta normale al piano focale passante per il foro è detto asse ottico. Infine la proiezione invertita dell'oggetto sul piano immagine è detta proiezione percettiva delloggetto. Nella pratica può risultare comodo riferirsi non alla proiezione percettiva dell'oggetto sul piano immagine (che fornisce una versione ribaltata delloggetto stesso) ma bensì alla sua proiezione sul piano immagine non invertito.

Per ottenere un'immagine panoramica all'interno del piano immagine della telecamera, occorre una tecnica per confluire tutti i raggi luminosi dell'ambiente circostante all'interno dell'asse ottico della camera, a questo scopo ci serviamo di uno specchio conico che rende possibile tale condizione catturando così un campo visivo orizzontale a 360° . Ma non tutti i sistemi ricavati dalla composizione di telecamere pin-hole ortogonali a specchi conici garantiscono una corretta ricostruzione dell'immagine reale a partire da ciò che la camera rileva, solo un sistema di visione ad unico punto di vista efficace, Single Effective View Point, rende possibile la fedele ricostruzione di un immagine prospettica. Infatti un raggio luminoso incidente sullo specchio si interseca su di uno e un solo punto nello spazio 3D(centro di proiezione). Sistemi SVP vengono detti a proiezione centralizzata(central projection systems).

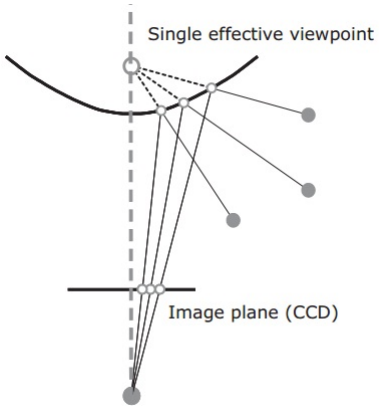


Figura 3.2: Visione omnidirezionale tramite specchio iperboloide

Alcuni sistemi a proiezione centrale consentono una mappatura di coordinate

dei punti omogenea; altri come i sistemi dati dalla combinazione di lenti e specchi, dato il loro forte fattore di distorsione, non possono essere trattati in maniera semplice e tradizionale. Dualmente, quando un sistema non riesce a creare una proiezione dell'immagine del punto nello spazio reale univoca, questo è chiamato sistema a proiezione non centrale.

Un sistema ad un unico punto di vista efficace è così desiderabile perché permette la generazione di immagini prospettiche geometricamente corrette dalle immagini acquisite dalla telecamera omnidirezionale. Questo è possibile perché, sotto il vincolo di unico punto di vista, ogni pixel rilevato nell'immagine, misura l'irradiamento della luce che passa attraverso il punto di vista in una direzione particolare. Quando la geometria della telecamera omnidirezionale è nota e inoltre quando la telecamera è calibrata, si può calcolare la direzione per ciascun pixel. Pertanto, il valore di irradiazione misurato da ciascun pixel può essere mappato su un piano a qualsiasi distanza dal punto di vista per formare una immagine prospettica planare.

3.1 Stato dellarte

Il concetto di centrali telecamere catadiottriche apparso già nella presentazione di René Descartes nel 1637 nel Discours de la Methode, ha mostrato che una rifrazione su superfici ovali riflettenti (lenti coniche e specchi) si concentra in un unico punto di luce, se illuminati da un altro punto. L'idea è stata poi riformulata, da Feynman nel 1963 o Hecht e Zajac nel 1974, e reso popolare in un linguaggio moderno e presentato alla comunità visione artificiale nel 1999 da Baker e Nayar[A5]. Baker e Nayar derivarono la classe completa di sensori catadiottrici con un unico punto di vista efficace e che può essere costruito utilizzando solo una singola lente convenzionale e un unico specchio. Hanno dimostrato che gli specchi da poter utilizzare a tale scopo rientrano in una classe geometrica ben definita, specchi con superficie appartenente alla classe delle sezioni coniche ruotate. Allinterno di questa classe, hanno indicato quattro possibili configurazioni di camera e specchio non degeneri, configurazioni combinate da una camera ortogonale con uno specchio paraboloide, iperboloidale, ellissoidale o planare. Una teoria unificante per tutti i sistemi centrali catadiottrici è stata proposta nel 2000 da Geyer e Daniilidis. Essi hanno mostrato e dimostrato che ogni proiezione prospettica catadiottrica (parabolica, iperbolica, ellittica) e standard è isomorfa ad una mappatura proiettiva da una sfera (centrata nell'effettivo punto di vista) ad un piano con il centro di proiezione sulla perpendicolare piano.

Infine, in [A6], Micusik ha proposto un nuovo formalismo per la rappresentazione di tutti i modelli di proiezione centrale, sia diottrica e catadiottrica (compresi punto di vista modelli). Dal suo formalismo, ha mostrato che è possibile descrivere

Figura 3.3: Telecamera Omnidirezionale

la relazione tra i punti dell'immagine e vettori 3D corrispondenti al reale punti mediante due funzioni g ed f.

3.2 Modello parametrico per la telecamera Omnidirezionale

Nel modello generale in una telecamera omnidirezionale è dovere identificare due riferimenti distinti: l'immagine della telecamera piano(u', v'), e il piano del sensore (u'', v''). In Fig. 3. 3 i due piani di riferimento sono presenti nel caso di un sistema catadiottrico. Tutte le coordinate saranno espressi nella sistema di coordinate collocato in O, con l'asse z allineato con l'asse del sensore.

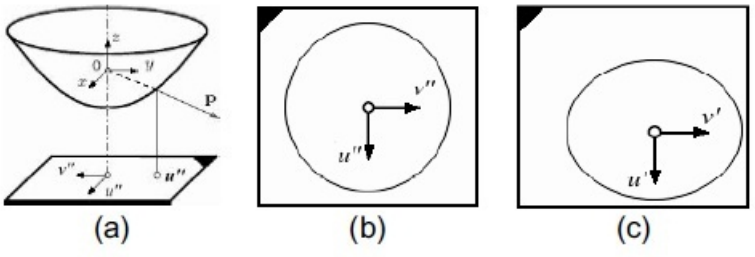


Figura 3.4: a)cordinate spaziali nel sistema catadiottrico b)piano del sensore c)immagine catturata dalla telecamera. b) e c) sono in relazione tramite una trasformazione affine

Sia X un punto appartenente allo spazio reale, assumiamo che il vettore colonna $u'' = [u'', v'']^T$, sia la proiezione del punto X sul piano dello specchio, e sia invece $u' = [u', v']^T$ l'immagine catturata dalla camera.
I punti sui due piani sono correlati da una trasformazione affine, che considera disallineamenti tra il corpo diottrico e lo specchio ed errori nella digitalizzazione dell'immagine, infatti $u'' = Au' + t$ dove $A \in \Re^{2\times2}$ e $t \in \Re^{2\times1}$. Quindi introduciamo g che correla un punto u'' sul piano del sensore al vettore p proiezione del raggio luminoso proveniente dal punto reale X incidente sullo specchio passante per il punto di vista O .
Per cui il modello completo per una telecamera omndirezionale, su sistema catadiottrico è:

$$\lambda p = \lambda g(u'') = \lambda g(Au' + t) = PX[1]$$

dove λ è un fattore di scala (o di profondita), $X \in \Re^4$,vettore delle coordinate omogenee, e $P \in \Re^{3\times4}$, matrice di proiezione. Con la calibrazione del sistema, si intende la stima della matrice A, del vettore t e della funzione non lineare g, che soddisfi l'equazione (1). Assumiamo che g sia nella forma:

$$g(u'', v'') = (u'', v'', f(u'', v''))^T[2]$$

assunta f tale che g sia rotazionalmente simmetrica. Questo significa assumere lo specchio come corpo perfettamente simmetrico, un ipotesi ragionevole, in quanto tali specchi sono creati con una precisione micrometrica. La funzione f è dipendente dal modello geometrico dello specchio in questione, ma essa, è calcolabile come sommatoria polinomiale, nella forma:

$$f = a_0 + a_1 p'' + a_2 p''^2 + .. + a_N p''^N[3]$$

Con $p = \sqrt{u''^2 + v''^2}$, distanza euclidea, e i coefficenti polinomiali a_i, $i = 0, 1, 2..N$, N+1 incognite, calcolabili attraverso la calibrazione. Quindi l'equazione [1] può essere riscritta come:

$$\lambda \begin{pmatrix} u'' \\ v'' \\ w'' \end{pmatrix} = \lambda g(Au' + t) = \lambda \begin{pmatrix} Au' + t \\ f(u'', v'')' \end{pmatrix} = PX, \lambda > 0$$

3.3 Calibrazione Telecamera Omnidirezionale

La calibrazione avrebbe il compito di stimare i paramentri A,t e i i coefficenti polinomiali a_i, $i = 0, 1, 2..N$, ma per abbassare il numero dei parametri incogniti in gioco, è opportuno porre un'approssimazione su i due parametri A e t, infatti assumendo che $A = I$ (matrice identità) e t=0 si ipotizza che il piano del sensore sia coincidente con l'immagine riflessa dallo specchio, ovvero non ci sia alcun tipo di trasformazione affine. In seguito il valore esatto della matrice A e del parametro t verranno calcolati, rispettivamente, tramite un'ottimizzazione non lineare e un metodo iterativo.
Utilizzando questa strategia, e quindi considerando $u'' = u'$, inserendola nell'equazione (4) e facendo uso della (3), si ottiene:

$$\lambda p'' = \lambda \begin{bmatrix} u' \\ v' \\ a_0 + a_1 p' + a_2 {p'}^2 + .. + a_N {p'}^N \end{bmatrix} = P \cdot X$$

dove ora u' e v' sono le coordinate rispetto al centro delle circonferenza, e p' la loro distanza euclidea. Inoltre si noti che la stima che si sta effettuando è su i soli coefficenti polinomiali, definiti come parametri intrinseci del sistema.
Durante la calibrazione un 'pattern planare' verrà utilizzato come modello di rife rimento per la stima dei parametri in quanto, mostrando alla telecamera tale pattern, posizionato in diverse posizioni, si potranno calcolare: la matrice di Rotazione R e la matrice di traslazione T; entrambe le matrici rientrano nei parametri estrinseci dal sistema catadiottrico.

Sia $M_{ij} = [X_{ij}, Y_{ij}, Z_{ij}]$ coordinate di un'ipotetica immagine nel sistema cartesiano riferito alle coordinate del pattern, e sia $m_{ij} = [u_{ij}, v_{ij}]^T$ le coordinate dei pixel riferiti alla stessa immagine catturata dalla telecamera. Senza perdita di generalita assumiamo che il pattern sia planare, quindi Zi=0, e l' equazione (5) diventa:

$$\lambda_{ij} p_{ij} = \lambda_{ij} \begin{bmatrix} u_{ij} \\ v_{ij} \\ a_0 + .. + a_N {p_{ij}}^N \end{bmatrix} = P^i X = \begin{bmatrix} {r_1}^i {r_2}^i {r_3}^i t^i \end{bmatrix} \cdot \begin{bmatrix} X_{ij} \\ Y_{ij} \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} {r_1}^i {r_2}^i t^i \end{bmatrix} \cdot \begin{bmatrix} X_{ij} \\ Y_{ij} \\ 1 \end{bmatrix} \quad (6)$$

Ai fini di estrarre le caratteristiche estrinseche del sistema, precedentemente citate, possiamo trascurare il fattore di profondità λ moltiplicando ambi i membri per il fattore p_{ij}

$$\lambda_{ij} p_{ij} \wedge +_{ij} = p_{ij} \wedge \begin{bmatrix} {r_1}^i {r_2}^i t^i \end{bmatrix} \cdot \begin{bmatrix} X_{ij} \\ Y_{ij} \\ 1 \end{bmatrix} = 0 \Rightarrow \begin{bmatrix} u_{ij} \\ v_{ij} \\ a_0 + .. + a_N {p_{ij}}^N \end{bmatrix} \wedge \begin{bmatrix} {r_1}^i {r_2}^i t^i \end{bmatrix} \cdot \begin{bmatrix} X_{ij} \\ Y_{ij} \\ 1 \end{bmatrix} = 0 (7)$$

Ogni punto p_{ij} contribuisce a tre equazioni omogenee non lineari,della forma:

$$v_j \cdot (r_{31}X_j + r_{32}Y_j + t_3) - f(p_j) \cdot (r_{21}X_j + r_{22}Y_j + t_2) = 0 (8.1)$$

$$f(p_j) \cdot (r_{11}X_j + r_{12}Y_j + t_1) - u_j \cdot (r_{31}X_j + r_{32}Y_j + t_3) = 0 (8.2)$$

$$u_j \cdot (r_{21}X_j + r_{22}Y_j + t_2) - v_j \cdot (r_{11}X_j + r_{12}Y_j + t_1) = 0 (8.3)$$

Dove X_j,Y_j e Z_j sono note. Inoltre inserendo tutti i parametri incogniti r_{11},r_{12},r_{21},r_{22},t_1,t_2 in un opportuno vettore, possiamo riscrivere le equazioni precedenti come un sistema di equazioni lineari :

$$M \cdot H = 0 \ (9)$$

dove

$$H = [r_{11}, r_{12}, r_{21}, r_{22}, t_1, t_2]^T$$

e

$$M = w \begin{bmatrix} -v_1X_1 & -v_1Y_j & u_1X_1 & u_1Y_1 & -v_1 & u_1 \\ & & \cdots\cdots & & & \\ & & \cdots\cdots & & & \\ -v_LX_L & -v_LY_L & u_LX_L & u_LY_L & -v_L & u_L \end{bmatrix}$$

Una stima lineare di H può essere ottenuta utilizzando il criterio di minimizzazione dei minimi quadrati su $||M \cdot H||^2$ con $||H||^2 = 1$: ciò viene compiuto tramite una decomposizione a valori singolari. I parametri r_{11},r_{12},r_{21},r_{22},t_1,t_2 verranno calcolati per ogni posa del pattern di riferimento; invece il parametro di traslazione t_3 verrà calcolato al passo successivo, durante la stima della funzione di proiezione.
Per calcolare invece i parametri intrinseci al sistema : a_i, $i = 0, 1, 2..N$,che descrivono la forma della funzione di proiezione g, e per calcolare l'ultimo parametro estrinseco rimasto incompiuto t_3, inseriamo tutti i dati delle equazioni (8. 1) e (8. 2) in un vettore si riscrivano come un sistema di equazioni lineari. Ottenendo il seguente sistema:

$$\begin{bmatrix} A_1 & A_1p_1 & .. & A_1{p_1}^N & -v_1 & 0 & .. & 0 \\ C_1 & C_1p_1 & .. & C_1{p_1}^N & -u_1 & 0 & .. & 0 \\ & & & \cdots\cdots & & & & \\ & & & \cdots\cdots & & & & \\ A_k & A_kp_k & .. & A_k{p_k}^N & 0 & 0 & .. & -v_k \\ C_k & C_kp_k & .. & C_k{p_k}^N & 0 & 0 & .. & -u_k \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ . \\ . \\ a_N \\ {t^1}_3 \\ . \\ . \\ {t^k}_3 \end{bmatrix} = \begin{bmatrix} B_1 \\ D_1 \\ . \\ . \\ B_k \\ D_k \end{bmatrix}$$

Dove

$$A_i = {r_{21}}^i X^i + {r_{22}}^i Y^i + {t_2}^i,\ B_i = v_i \cdot ({r_{31}}^i X^i + {r_{32}}^i Y^i),$$

$C_i = {r_{11}}^i X^i + {r_{12}}^i Y^i + {t_1}^i$ and $D_i = u^i \cdot ({r_{31}}^i X^i + {r_{32}}^i X^i)$.
La soluzioni per i minimi quadrati,ottenuta tramite il calcolo della matrice pseudoinversa, determina tutti i parametri intrinsechi al sistema, in particolare i coefficienti polinomiale a_i, $i = 0, 1, 2..N$, che descrivono il modello.

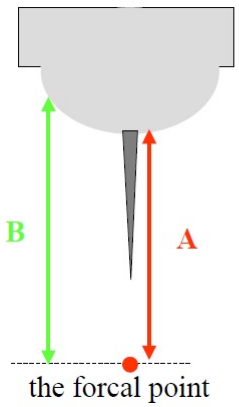
Capitolo 4

Montaggio e calibrazione SVP

4.1 Montaggio SVP

Un sistema catadiottrico mediante specchio iperbolico è centrale se e solo se è impostato sulla condizione di Single effective ViewPoint (SVP). La condizione SVP si avvera solo se il sensore d'immagine e il fuoco dell'iperboloide sono a una distanza pari a 2 volte la distanza focale f(istanza in mm tra il centro della lente e il suo fuoco) lungo l'asse centrale dello specchio [A5].
Una volta calibrato, un sistema catadiottrico centrale potrà essere trattato in maniera simile ad una camera prospettica. La risoluzione dipende dalla distanza dal centro sul piano immagine.

Nel nostro caso, le dimensioni dello specchio iperboloide, corpo catottrico del nostro sistema, seguono quelle del modello VS-C15 nella tabella seguente.



Omnisensor	A	B
VS-C450	60.858mm	68mm
VS-C42	94.4mm	106mm
VS-C15	28.5mm	32.7mm

Figura 4.1: Dimesioni specchio.

Quindi nel montaggio del sistema SVP, il sensore della NXTCam dovrà porsi a 5.7 cm(la distanza focale si trova nella colonna denominata come A) dal fuoco dello specchio,il cui asse è individuato da un piccolo ago nero uscente proprio dal punto di fuoco dell'iperboloide.

Per fare ciò si è deciso di porre lo specchio su di un distanziatore cilindrico in cartone, che potesse fissarne la distanza dal case di Lego creato per contenere senza consentire alcuna oscillazione alla camera posizionata in cima al robot.

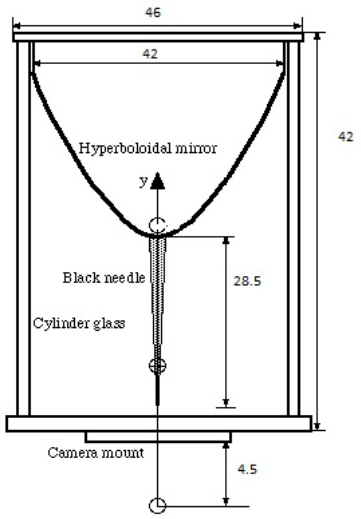


Figura 4.2: Dimensioni del sistema SVP.

Seguentemente a tale installazione si è proseguiti con la calibrazione, seguendo il modello parametrico precedentemente analizzato, utilizzando il toolbox OCam-Calib per Matlab® .Questa calibrazione consiste nella stima polinomiale della funzione di proiezione e della sua inversa (per il nostro scopo trascureremo la sua inversa); inoltre il toolbox estrae parametri estrinseci e intrinseci del sistema, compensando gli errori indotti dalla condizione SVP non perfetta e dell'Hardware utilizzato.

4.2 Calibrazione SVP

La procedura proposta è molto veloce e in linea generale automatica. L'utente ha solo il compito di raccogliere alcune immagini dal sistema catadiottrico di una scacchiera, e cliccare sui suoi angoli. Questa tecnica si svincola dalle specificità dei diversi sensori omndirezionali e quindi si prefigge la calibrazione di qualsiasi sistema omnidirezionale.Il toolbox assume che la funzione di mappatura dell'immagine

possa essere descritta da uno sviluppo in serie di Taylor i cui coefficenti sono stimati risolvendo un problema di minimi quadrati lineari, e seguentemente, un'analisi di minimizzazione tramite criterio di massima verosimiglianza .Il tool mostra poi la precisione di tale calibrazione proposta, proiettando un analisi degli errori, e visualizzando una simulazione, quanto più accurata, delle proiezioni delle immagini caricate in un ambiente 3D.

4.2.1 OcamCalib : Requisiti minimi

La casella degli strumenti OCamCalib per Matlab è stata testata con successo in Matlab, versione 6.5, 7.0, 2007 per Windows e Linux, 2009, 2010, e 2011. Lo strumento di affinamento di calibrazione richiede il Matlab Optimization Toolbox, in funzione `lsqnonlin`, che si dovrebbe avere per impostazione predefinita.

4.2.2 OcamCalib : Download e avvio

È possibile scaricare il Toolbox OcamCalib da indicati `http://robotics.ethz.ch/~scaramuzza/Davide_Scaramuzza_files/software/Scaramuzza_OCamCalib_v2.0.zip`
Per eseguire il tool di lavoro : decomprimere il file, eseguire Matlab, e digitare dalla riga di comando MatLab : `ocam_calib`. Verrà quindi, visualizzata la seguente finestra di dialogo:

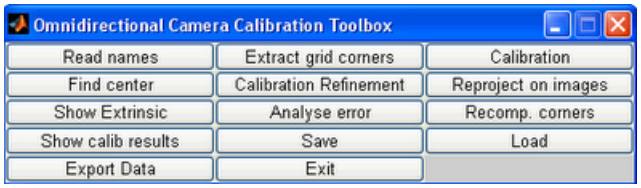


Figura 4.3: Casella strumenti OCamCalib .

4.2.3 OcamCalib : Pattern o griglia di riferimento

Il pattern planare di riferimento, rappresenta una scacchiera di 4 quadrati in altezza e 5 in lunghezza, scaricabile dal seguente link : `http://robotics.ethz.ch/~scaramuzza/Davide_Scaramuzza_files/software/pattern.pdf`
Dopo avere scaricato l'immagine e averla stamparla su un foglio standard A4, incollarla su di un cartoncino, poi assicurarsi che vi sia sempre bordo bianco lungo tutto il modello. Questo bordo bianco è necessario per lo strumento automatico per facilitare l'estrazione dei vari angoli. Utilizzare poi NXTCamView, e la sua

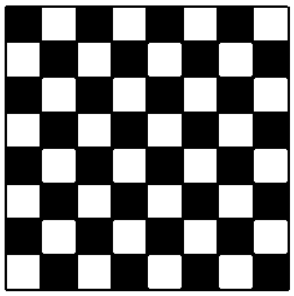


Figura 4.4: Griglia di riferimento .

funzione snapshot ,per la cattura delle immagini con la fotocamera NXTCam all'interno del sistema precedentemente analizzato.Un numero tra 5 e dieci immagini può essere sufficiente. Ovviamente per una buona calibrazione sarebbe utile cercare di ottenere più angolazioni possibili.

4.2.4 OcamCalib : Caricamento Immagini

Il primo passo per calibrare la fotocamera omnidirezionale consiste nel caricare le immagini della scacchiera, precedentemente scattate in diverse posizioni e orientamenti.

Se le immagini sono pronte, si può iniziare a caricarle nel toolbox.Ma prima di caricare le immagini è doveroso assicurarsi che siano nella stessa cartella dei file di OCamCalibToolbox. Quindi, fare clic sul pulsante Read Names. Verrà visualizzato il seguente messaggio nella shell di MatLab:

```
Basename camera calibration images (without number nor suffix):
```

Questo tipo di messaggio domanda all'utente di inserire la stringa rappresentante il 'nome base' dei file di immagine, senza aggiungere il formato del file. Ad esempio, se le immagini scelte per l'inseriemento nella casella degli strumenti OcamCalib sono del tipo: Image0.gif, Image1.gif e Image2.gif,significa che il 'nome base' delle immagini è Image. Quindi, digitare:

```
Basename camera calibration images (without number nor suffix): Image
```

Poi, verrà chiesto di digitare il formato dell'immagine. Per cui, dovrà essere digitata la lettera associata al formato utilizzato:

```
Image format: ([]='r'='ras', 'b'='bmp', 't'='tif', 'g'='gif', 'p'='pgm',
'j'='jpg', 'm'='ppm') » g
```

Se l immagine era in formato gif, sarà necessario digitare g. A questo punto, la casella degli strumenti caricherà tutte le immagini con lo stesso 'nome base' precedentemente indicato:

```
Loading image 1...2...3...4...5...6...7...8...9...10...
Done
```

Alla fine, la casella degli strumenti visualizzerà le miniature delle immagini inserite per la calibrazione .

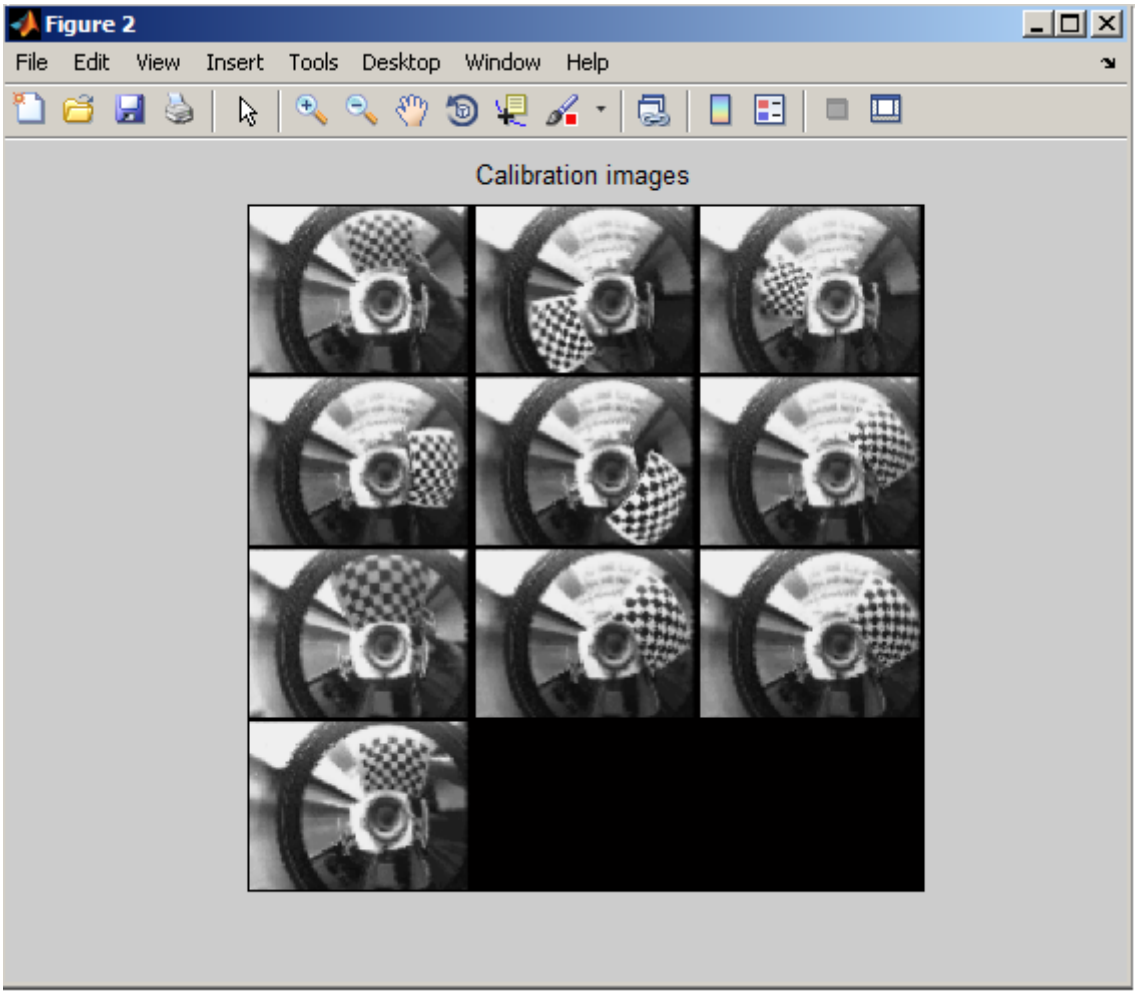


Figura 4.5: Collezioni di immagini utili alla calibrazione .

Caricamento immagini: Limiti tecnologici

È stato verificato empiricamente che caricando immagini di risoluzione troppo bassa, il toolbox andava in crash.Infatti nei passi successivi, il software richiede di individuare gli angoli della griglia di riferimento e seppur il programma presentasse uno zoom automatico, questo non bastava al toolbox per identificare tali corner. Per risolvere tale problema si è decisi di caricare immagini ingrandite tramite altri programmi di image editing e quindi dalla risoluzione di default 176x144 si è passati a una risoluzione di 521x395. Di questo fatto ne verrà tenuto conto in seguito durante la stesura del programma NXC, riguardante la trasformazione di coordinate da 2D a 3D, in quanto la calibrazione basata su una risoluzione ingrandita produrrà risultati che dovranno essere riportati alla risoluzione naturale della NXTCam tramite un fattore di scala.

4.2.5 OcamCalib : Estrazione angoli della griglia

È la fase più importante per la calibrazione, e i risultati di calibrazione dipendono direttamente dalla posizione degli angoli della scacchiera in ciascuna immagine. Nella versione 2.0 del toolbox OCamCalib è possibile scegliere di utilizzare l'estrazione automatica della scacchiera oppure manuale. Nel primo caso, il toolbox non chiederà alcuna interazione all'utente per trovare tutti gli angoli. Mentre nella seconda modalità di estrazione, l'utente avrà il compito di cliccare su ciascun angolo richiesto dal toolbox.Per la bassa qualità di risoluzione e per le dimensioni ridotte, delle immagini scattate dalla NXTCam, tutte le immagini scattate dalla camera subiscono un ingrandimento di un fattore di scala e l'estrazione degli angoli deve essere compiuta tramite modalità manuale. Cliccando su Extraction grid corner otterrete questo messaggio:

```
Extraction of the grid corners on the images

Type the images you want to process (e.g. [1 2 3], [] = all images)
=
```

Digitare INVIO se si desidera elaborare ogni immagine, o digitare il vettore contenente il numero di immagini che si desidera elaborare. Nel nostro tutorial vogliamo elaborare tutte le immagini, quindi dobbiamo semplicemente premere INVIO. Il messaggio successivo è il seguente:

```
Number of squares along the X direction ([]=10) =
```

Digitare il numero di quadrati lungo la direzione X, e quelli lungo l'asse Y. Nel nostro caso, per esempio, vogliamo utilizzare 4 pedine piene lungo la direzione verticale e 5 pedine lungo Y (ad esempio la direzione orizzontale). Così Tipo:

```
Number of squares along the X direction ([]=10) = 4 Number of squares
along the Y direction ([]=10) = 5
```

Ora digitate rispettivamente la dimensione della piazza lungo gli assi X e Y. In questo le nostre immagini è di 30 mm.

```
Size dX of each square along the X direction ([]=30mm) = 30 Size dY of
each square along the Y direction ([]=30mm) = 30
```

Se si preme INVIO, il Toolbox carica i parametri di default ([] = 30mm). Importante osservare, tuttavia, che la dimensione del controllo è utilizzata solo per recuperare le posizioni assolute delle scacchiere.

Nel messaggio successivo, il Toolbox chiederà la posizione (righe, colonne) del centro dell'immagine omnidirezionale. Perché la Casella degli strumenti Ocam-Calib è in grado di determinare automaticamente la posizione del centro, si può semplicemente lasciare vuoto questo campo premendo il tasto ENTER. In questo caso, il Toolbox in un primo stadio prendere come centro , il centro dell'immagine, trascurandone la reale posizione dell'asse focale . Questo passo, tuttavia, non è di gran rilievo; infatti successivamente il toolbox ricaverà automaticamente il centro del sistema , confrontando tutte le immagini caricate.

```
X coordinate (along height) of the omnidirectional image center = ([]=197)
= Y coordinate (along width) of the omnidirectional image center = ([]=260)
=
```

Ora la casella degli strumenti chiederà all'utente, di specificare se si desidera utilizzare l'estrazione automatica o quella manuale. L'estrazione automatica è la più semplice e meno tediosa, in quanto non chiede alcun click all'utente.È importante per questa routine utilizzare una griglia di scacchiera con bordo bianco di grandi dimensioni come precedentemente annunciato.

Estrazione angoli della griglia automatica

Se si è optato per l'estrazione automatica allora la casella degli strumenti visualizzerà il seguente messaggio:

```
Processing image 1..
```

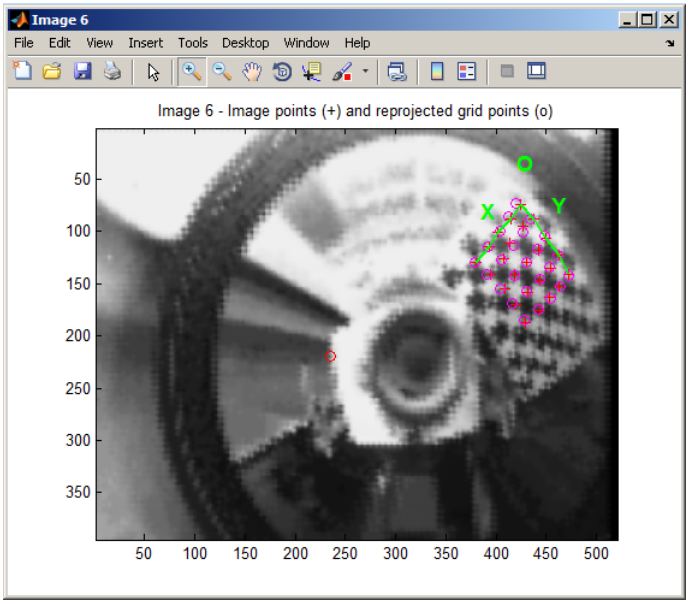


Figura 4.6: Angoli estratti da immagine scacchiera .

Conseguemente verrà visualizzata l'immagine postprocesso, evidenziandone gli angoli catturati dal toolbox, come nell'esempio seguente:

`Would you like to reposition any of the assigned corners ([] = yes, other = no)?`

Se si è soddisfatti basta scegliere no, digitando qualsiasi carattere. In caso di imprecisioni del software, il toolbox vi chiederà di riposizionare qualsiasi angolo assegnato in precedenza,utilizzando il tasto sinistro per riposizionarlo e il tasto destro del mouse per uscire dalla modalità di riposizionamento. Se mancano alcuni degli angoli, la casella degli strumenti vi chiederà di fare clic sui punti mancanti, seguendo l'ordine indicato sulla parte superiore della figura.

Prima di elaborare l'immagine successiva, il toolbox cambierà la numerazione dei punti e visualizzerà l'orientamento degli assi XY e all'origine degli assi.Quindi premere INVIO per continuare.

Estrazione angoli della griglia manuale

In questa fase, la Casella degli strumenti OCamCalib vi chiederà di fare clic sui punti d'angolo di ogni immagine della scacchiera. Per facilitare l'estrazione degli angoli, durante il selezionamento tramite click, la casella degli strumenti si avvale di un sensore d'angolo in grado in interpolare la posizione ottimale l'angolo della griglia, attorno al punto cliccato.

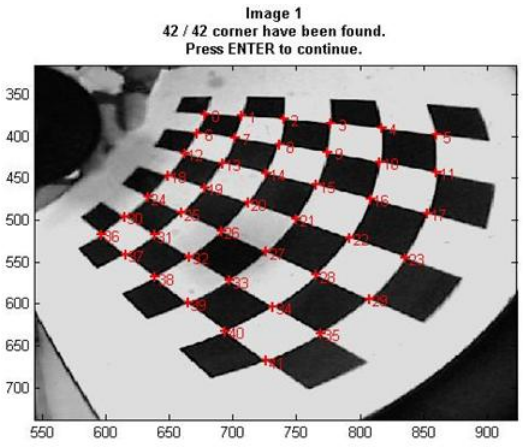


Figura 4.7: Esempio di estrazione automatica .

```
Do you want your clicking to be assisted by a corner detector ? ( []
= yes, other = no )
```

Per accettare l'assistenza durante la selezione degli angoli, basta premere INVIO, premendo invece qualsiasi altro tasto, l'assistente verrà deprecato.

Se l'assistente di identificazioni di angoli è stato abilitato, il Toolbox chiederà di digitare la dimensione della finestra del rivelatore d'angolo. La finestra è la dimensione $(2 * winx + 1) * (2 * winy + 1)$

```
   Window size for corner finder (wintx and winty): wintx ([] = 8) =
winty ([] = 8) =
```

Di solito, i valori dati come default dovrebbe funzionare bene, in caso contrario, (se ad esempio il luogo scelto dal rivelatore angolo è troppo lontano dal punto in cui si fa clic su) si può provare a scegliere un valore inferiore. Al contrario, se la risoluzione delle immagini è molto elevata (fino a 5 megapixel!), si consiglia di scegliere valori più grandi.

Se invece si continuerà approvando le impostazioni di default, il toolbox visualizzerà:

```
   wintx ([] = 8) =
winty ([] = 8) =
Window size = 17x17

   Processing image 1... Using (wintx,winty)=(8,8) - Window size = 17x17
```

```
Press ENTER and then Click on the extreme corners of the rectangular
complete pattern (the first clicked corner is the origin)...
```

ATTENZIONE:
Gli angoli devono essere individuati da sinistra verso destra.Inoltre, nell'elaborazione delle immagini rimanenti, è dovere porre attenzione a preservare le corrispondenze dei punti cliccati,mantenendo l'ordine di tale estrazione. Quindi, è buona prassi mantenere le corrispondenze dei punti; questo fa in modo di mantenere lo stesso orientamento per gli assi di riferimento di ogni snapshot. Per quanto già anticipato, sui problemi intrinseci del sistema utilizzato in questo progetto, il numero di immagini effettivamente utilizzate durante questa procedura, da un numero iniziale di 10 è stato ridotto a solamente 5 immagini, in quanto le altre, pur essendo scattate in condizioni di luce ottimali, alteravano il corretto funzionamente del toolbox, bloccando l'esecuzione del programma stesso e rendendo impossibile quindi una corretta riuscita dell'estrazione.

Prima di iniziare a cliccare sugli angoli della griglia, è consentito ingrandire la zona dell'immagine, che contiene la scacchiera. Quando questa è ingrandita, è chiesto di premere INVIO. In questo modo, la forma del cursore muta in un quadrato, il che significa che si è in modalità di scatto. Quindi, è possibile iniziare a cliccare su ogni punto d'angolo, ricordando che il primo punto identifica l'origine degli assi XY del sistema di riferimento della scacchiera.

4.2.6 OcamCalib : Calibrazione

Per effettuare questa operazione, fare clic sul pulsante di calibrazione. Verrà visualizzato il seguente messaggio:

```
Degree of polynomial expansion ([]=4) =
```

Questo parametro permette di scegliere l'ordine massimo del polinomio che approssima la funzione che proietta indietro ogni punto da pixel allo spazio 3D. Diversi esperimenti su diversi modelli di fotocamere hanno dimostrato che un ordine polinomiale = 4 dà i risultati migliori. Se non si é soddisfatti dei risultati ottenuti, cercare di diminuire o aumentare l'ordine polinomiale. In questo procedimento si è scelto di mantenere il valore di default pari a 4. Una volta scelto l'ordine polinomiale, la taratura viene eseguita molto velocemente, perché viene utilizzato un metodo di minimizzazione a quadrati lineari. Al termine della calibrazione, la casella degli strumenti visualizza il grafico seguente, che mostra il grafico della funzione f, e la forma di THETA, angolo di inclinazione vettore di luce reale, rispetto all'orizzonte.

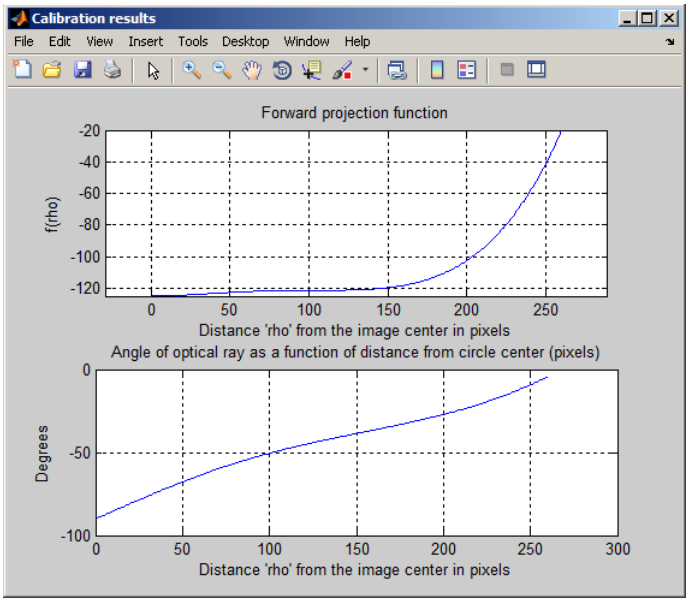


Figura 4.8: Risultati della calibrazione .

4.2.7 OcamCalib : Centro immagine

.

Questa routine cercherà di estrarre il centro dell'immagine automaticamente. Se durante l'estrazione angolo griglia non avete impostato i valori corretti per il centro dell'immagine omnidirezionale, è possibile utilizzare il rilevamento automatico del centro. Per farlo, è sufficiente fare clic sul pulsante Trova il centro, e OcamCalib Toolbox avvia un metodo iterativo per calcolare il centro dell'immagine, che riduce al minimo l'errore di riproiezione di tutti i punti della griglia. Il rilevamento automatico del centro richiede solo pochi secondi.
`Iteration1 ... 2 ... 3 ... 4 ... 5 ... 6 ... 7 ... 8 ... 9 ...`

Alla fine, il Toolbox ricalcola tutti i parametri di calibrazione per la nuova posizione del centro, ed emette le nuove coordinate del centro. Vedi sotto.

3.74 ± 2.08
2.80 ± 1.34
1.91 ± 1.22
2.24 ± 1.48
3.41 ± 2.24
2.73 ± 1.64

```
Average error [pixels]
2.804652

Sum of squared errors
1661.774558

xc =
180.6391

yc =
340.7161

»
```

L'errore medio è la media dell'errore di riproiezione calcolato su tutte le scacchiere, mentre Somma degli errori al quadrato è ovviamente la somma in caso di errori di riproiezione al quadrato. I parametri di calibrazione sono i ocam_model.ss variabili. Questa variabile contiene i coefficienti del polinomio della funzione F. Vorrei ricordare che F ha la seguente forma: $F = a_0 + a_1 \cdot p + a_2 \cdot p^2 + .. + a_n \cdot p^n$

dove è la distanza dal centro dell'immagine omnidirezionale, misurata in pixel. In ocam_model.ss i coefficienti vengono memorizzati dal minimo al massimo ordine, cioè ocam_model.ss = [a0, a1, a2 ...].

Se in qualsiasi momento si desidera modificare le coordinate del centro, si può semplicemente modificare il valore delle variabili e ocam_model.xc ocam_model.yc, che contengono rispettivamente riga e colonna della posizione centrale.

4.2.8 OcamCalib: Rifinimento calibrazione

Facendo clic sul pulsante di calibrazione Refinement, il Toolbox avvia un algoritmo di ottimizzazione non lineare di calibrazione di Levenberg-Marquadt. L'ottimizzazione viene eseguita nel tentativo di minimizzare la somma degli errori di riproiezione al quadrato. La raffinatezza della calibrazione viene eseguita attraverso il Matlab Optimization Toolbox, e, in particolare, richiede la funzione `lsqnonlin`, che si dovrebbe avere per impostazione predefinita. Questo algoritmo si basa su due passi. In primo luogo calcola i parametri della telecamera estrinseci, cioè, le matrici di rotazione e traslazione di ciascun scacchiera rispetto alla macchina. Poi ricalcola i parametri intrinseci della telecamera (cioè `ocam_model`).

Dopo aver fatto clic sul pulsante di Perfezionamento di calibrazione, la Casella degli strumenti chiede se si è sicuri di voler avviare la raffinatezza non lineare, e informa che la procedura può richiedere alcuni secondi.

```
   This function refines calibration parameters (both EXTRINSIC and INTRINSIC)
by using a non linear minimization method Because of the computations
involved this refinement can take some seconds Press ENTER to continue
OR Crtl-C if you do not want
   Starting refinement of EXTRINSIC parameters
   Optimization terminated: search direction less than TolX.
Chessboard pose 1 optimized
Optimization terminated: search direction less than TolX.
Chessboard pose 2 optimized
Optimization terminated: search direction less than TolX.
Chessboard pose 3 optimized
Optimization terminated: search direction less than TolX.
Chessboard pose 4 optimized
Optimization terminated: search direction less than TolX.
Chessboard pose 5 optimized
```

4.2.9 OcamCalib:Visualizzazione parametri estrinseci

Facendo clic sul pulsante Show Estrinsic, la casella degli strumenti visualizza la posizione di ogni scacchiera rispetto al sistema di riferimento della telecamera omnidirezionale.

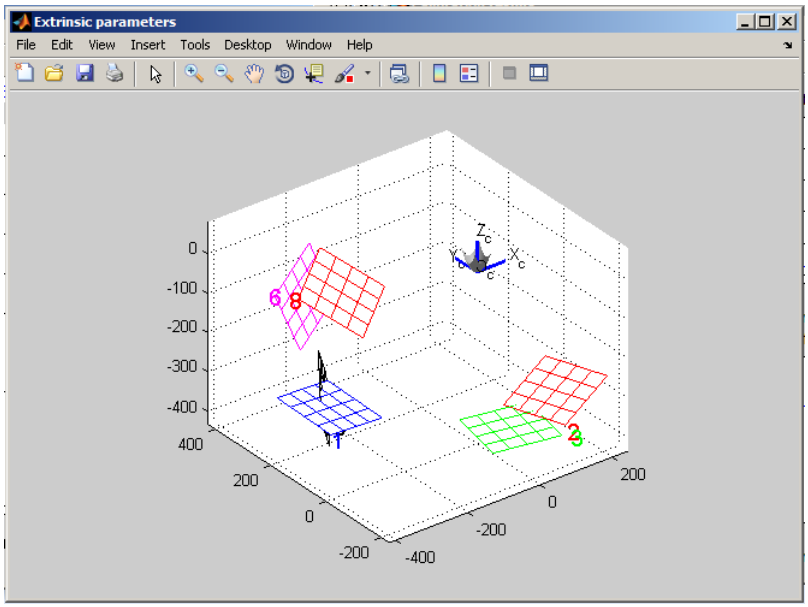


Figura 4.9: Parametri estrinseci.

4.2.10 OcamCalib:Analisi errori

Cilccando sul pulsante Analizza errore si può constatare la distribuzione dell'errore di riproiezione di ciascun punto per tutte le scacchiere. Colori differenti si riferiscono a immagini differenti.

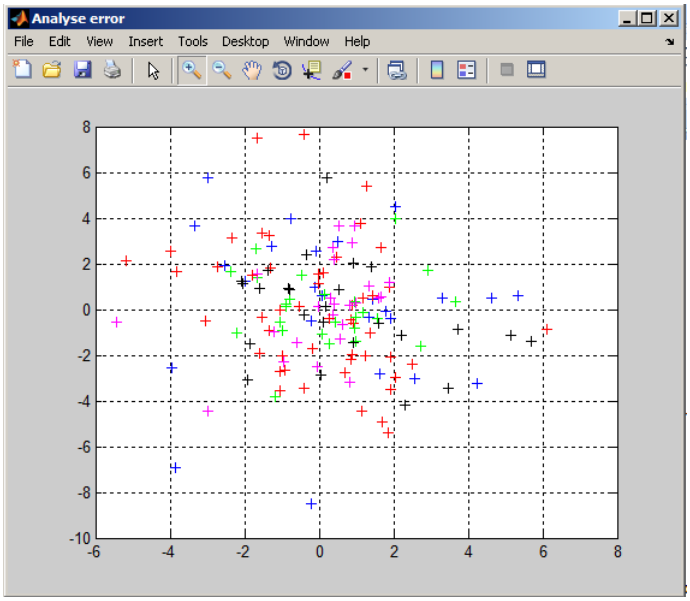


Figura 4.10: Analisi degli errori .

4.2.11 OcamCalib:Risultati calibrazione

Facendo clic sul pulsante Show Calib Result , viene visualizzato un riepilogo testuale e grafico dei passi procedurali appena percorsi.

4.3 Analisi dei risultati

Conseguentemente ai limiti del campo visivo di copertura del sistema omnidirezionale e dalla scarsa risoluzione dei vari fotogrammi scattati dalla camera NXT-CamView,la calibrazione presenta degli errori con una dispersione in pixel maggiore di quanto presentato nella documentazione ufficiale di OCamCalib [S3], la quale prediceva un errore di riproiezione dell'immagine in un sistema 3D, in media di 0.5 pixel, differentemente da quanto mostra la fig 4.10 che presenta una media d'errore in pixel di circa 5. Similarmente alla documentazione officiale, la funzione f, stimata con modello ai cofficenti polinomiali di Taylor si distribuisce esponenzialmente all'aumentare dei pixel dal centro dell'immagine. Discorso analogo per

quanto riguarda l'inclinazione dei raggi luminosi incidenti nel piano dello specchio in funzione della distanza dal centro immagine, vedi Fig.4.8.

Capitolo 5

Il Robot

5.1 Panoramica

Il robot inseguitore è stato implementato interfacciando al mattoncino intelligente Nxt il sistema di visione omnidirezionale, composto da una telecamera NXTCam V2 diretta verso il fuoco di uno specchio iperboloide e da un secondo modulo di sensori costituito da una coppia di NXT Interactive Servo Motor, particolare tipo di motore di piccola potenza, le cui condizioni operative sono soggette ad ampie e spesso repentine variazioni sia nel campo della velocità che della coppia motrice, eseguite con grande precisione.

Figura 5.1: NXT Interactive Servo Motor.

I servomotori qui citati, lavorano in corrente continua (DC) e hanno le seguenti caratteristiche:

- Tensione di alimentazione 9V (DC)
- Potenza meccanica a 9V 2,03W
- Velocità massima 170 rpm (117 rpm a 9V)

- Potenza elettrica a 9V 4,95W
- Ecienza a 9V 41%
- Assorbimento a 9V 0.55A
- Corrente di No-Load 60mA
- Coppia a 9V 16,7 N*cm

Questi motori comprendono, al loro interno, un sensore di rotazione in grado di fornire una misura con precisione di 1 grado rispetto allasse esterno; questo integra un controllore retroazionato di tipo P.I.D., il quale consente una precisa gestione dell'angolo di rotazione che, tramite opportune operazioni si riflette in un efficace controllo della distanza percorsa. Un ulteriore controllo è quello sulla potenza impressa sugli assi di rotazione, che si traduce in un semplice controllo della velocità impressa al robot; questa presenta una precisione in misura minore al controllo sull'angolo di rotazione, in quanto la potenza è limitata allo stato della batteria. Come si vedrà in seguito, per ottenere delle rotazioni da parte dell'inseguitore, ci si è serviti di una tecnica basata sulla diversa direzione di rotazione di ambi i servo, puntando il robot su di un ruotino, formato da una sfera metallica ,equivalente alla punta di una penna biro.

Figura 5.2: Ruota a sfera metallica.

Assemblando insieme queste parti è stato creato tale robot.

5.2 L'inseguimento

Il sistema deve essere in grado di riconoscere un oggetto di particolare colore appartenente al modello RGB. Tale inseguimento inizia nel momento in cui il target entra nel campo di visione del robot, e si conclude appena l'inseguitore dista dal suo obiettivo con una distanza minima ,studiata per evitare collisioni. Inoltre l'inseguitore diminuirà la potenza impressa alle ruote al diminuire del distacco con il target per dare il tempo alla telecamera di inviargli le coordinate dove il robot

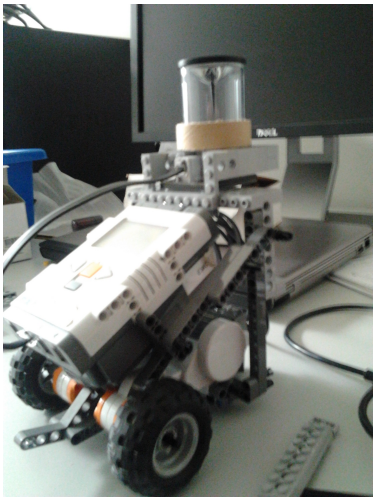


Figura 5.3: Robot Inseguitore.

dovrà spostarsi. Durante la navigazione, il costante aggiornamento della posizione del target, verrà implementato tramite un sistema multitasking . L'uso di molteplici task e di subroutine consente di rendere i programmi più leggibili e più compatti. Un programma NXC consiste di un numero di task a piacere, tra 1 e 255. Il task main deve sempre esistere,dato che è il principale task del programma, ed il primo ad essere eseguito. I vari task per la condivisione e l'accesso alle risorse, saranno gestiti da un insieme di variabili semaforo.

5.2.1 Limiti tecnici

Il sistema SVP creato, presenta varie imperfezioni che lo rendono un sistema quasi centrale.Pesano infatti, sul funzionamento finale del robot, il disallineamento intrinseco tra l'asse del sensore con l'asse del corpo della telecamera e le varie imperfezioni, dovute all'installazione dello specchio conic . Questi problemi, di natura tecnico-implentativa, costituiscono la causa di una non trascurabile limitazione nella navigazione . Infatti l'immagine catturata dal fotosensore digitale sul piano dello specchio non risulta totalmente panoramica ma presenta alcune regioni oscurate e quindi trascurate durante il tracking.

La figura 5.4 evidenzia come, nelle regioni contrassegnate dal colore rosso, il sistema di visione trascuri alcune aree spaziali.Nell'immagine seguente è rappresentata la regione che il sistema omnidirezionale riesce realmente a coprire.Questa limitazione peserà sulla dinamica dell'inseguimento del robot, che cercherà di mantenere il target nella regione di massima copertura, evidenziata in fig. 5.5 dal colore verde. Analizzando il tracking tramite NXTCamView, è stato possibile verifica-

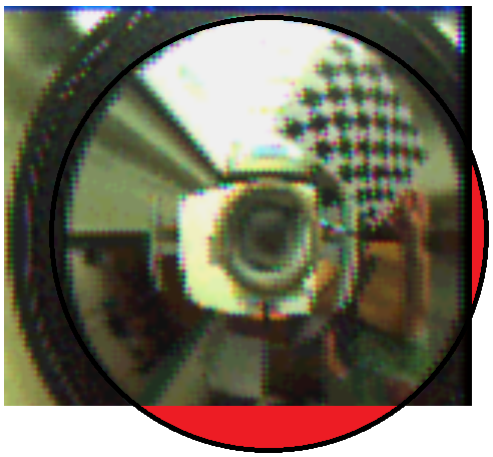


Figura 5.4: Immagine di test scattata durante la calibrazione.

re empiricamente quanto potesse valere il maggior raggio di visione coperto dal sistema omnidirezionale.I risultati si trovano espressi in cm nella figura 5.5.

5.2.2 ColorMaps

Gli oggetti di interesse sono riconosciuti dalla NXTCam, confrontando i valori del colore memorizzato con l'immagine rilevata. Per fare questo, i valori cromatici degli oggetti di interesse devono essere salvati nella memoria della NXTCam. Questo registro di memoria viene definito come ColorMap. La telecamera è in grado di memorizzare fino a 8 ColorMaps e riesce a fornire informazioni riguardandi le coordinate degli oggetti corrispondenti a tali colori. Nella figura 5.6 vi è uno screenshot del software NXTCamView, che mostra gli oggetti di interesse e le loro informazioni di monitoraggio. La finestra in alto a sinistra nella foto qui sotto mostra il campo di vista di NXTCam. Gli oggetti di interesse da questo punto di vista sono le penne rosse e blu. Nella modalità definita Tracker Object, le informazioni indicate dal software riguardano posizione e dimensione delle regioni d'interesse, tale modo di funzionamento verrà richiamato dal codice caricato nel brick per svolgere l'attività di tracking.

La mappa di colori è un buffer di 48 byte presente nella memoria della NXTCam. I primi 16 byte sono per il rosso, i secondi 16 byte per il verde e i rimanenti per il blu. Ognuno dei 16 byte è associato univocamente agli intervalli:0-16-32-48-64-80-96-112-128-144-160-176-192-208-224-240-255,rappresentanti ciascuno un intervallo di colore del modello RGB. Il byte salva al suo interno, come valore, il numero di oggetti riconoscibili per tale colore associato. Ad esempio, la penna blu riconosciuta nella foto precedente, possiede i seguenti intervalli di colore RGB :

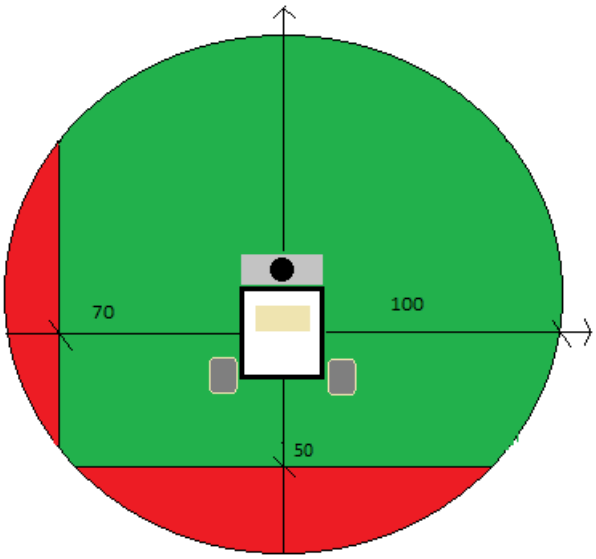


Figura 5.5: Regione di copertura del sistema omndirezionale.In verde l'area dove un target verrebbe riconosciuto,in rosso la zona dove questo verrebbe trascurato.

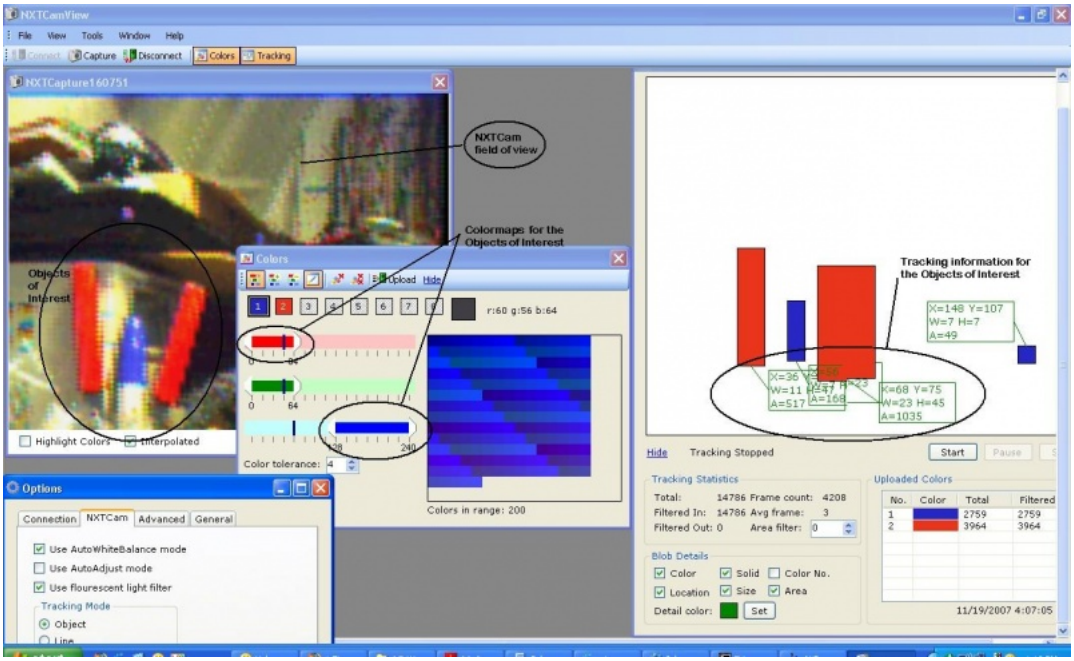


Figura 5.6: Il software NXTCamView visualizza il tracking in modalità Tracker Object.

- Red: 0-64
- Green:0-64
- Blu:128-255

Per impostare tale colore nella ColorMap per un solo oggetto riconoscibile, sarà neccessario salvare un 1 nei primi quattro byte per il rosso,dal 17esimo al 20esimo byte per il verde e dal 41esimo al 48esimo byte per il blu. Per evitare di dover implementare un tedioso controllo di registro mediante I2C, protocollo dedito alla gestione della ColorMap, è possibile utilizzare la libreria per NXC `nxtcamlib` al seguente URL:
`http://www.mindsensors.com/index.php?module=documents&JAS_DocumentManager_op=viewDocument&JAS_Document_id=153`
che fornisce all'utente molte semplici funzioni wrapper del protocollo I2C,le quali scrivono e leggono a loro volta nella colorMap .

Per la scrittura di un colore all'interno del registro ColorMap, ci si serve della presenza di due funzioni nel pacchetto nxtcamlib :

- `NXTCam_PrepareColorMap` : funzione che, conseguentemente agli intervalli RGB in forma decimale passati come input, insieme al numero di oggetti da

ricercare, precarica un array di 48 byte con i valori corretti per le maschere di ogni byte della ColorMap

- `NXTCam_WriteColorMap` : salva l'array di byte ,precedentemente creato dalla funzione `NXTCam_PrepareColorMap` , tramite il protocollo I2C nel buffer effettivo della camera

5.2.3 Il tracking

I passi da seguire, utilizzati per il tracking del colore prescelto, saranno i seguenti:

1. Caricare il colore nella ColorMap
2. Inizializzare il flusso di dati, tra la NXTCam e il Brick NXT
3. Effetturare un polling mediante task, per ricevere in modalità Tracker Object le informazioni riguardanti la posizione del target, vista dal sensore dallo specchio.

Il punto uno è stato studiato precedentemente; rimangono da definire come è possibile implementare i punti 2 e 3 . La libreria scaricabile `nxtcamlib` provvede a fornire funzioni semplici e veloci, basate sul protocollo I2C. In particolare si utilizza la funzione `NXTCam_Init` per instaurare un flusso di Tracking Object da camera a processore NXT. Il tracking, appena inizializzato, fornisce al task o alla funzione chiamante gli oggetti associati al colore precaricato nella ColorMap ordinati per dimensione. Per problemi dovuti alla luminosità dell'ambiente di lavoro, pur essendoci un unico target, il sensore potrebbe rilevare più oggetti di interesse, questo perché la camera frammenta l'immagine del target individuato in più regioni; per tale motivo si è assunto di utilizzare le informazioni riguardanti l'oggetto di massime dimensioni rilevato dalla camera. Il polling eseguito sulla NXTCam è consentito grazie ad un'ulteriore funzione presente nella libreria nxtcam : `NXTCam_GetBlobs()`,questa appoggiandosi allo stream precedentemente instaurato, carica le coordinate dei rettangoli (blobs) delimitanti le aree di colore ricercate,in variabili array, presupponendo che il Tracker Object possa essere eseguito per la ricerca di più di un oggetto.

5.2.4 La trasformazione di coordinate

Le coordinate ricevute na `NXTCam_GetBlobs` sono ovviamente le coordinate lette dalla lente sullo specchio, quindi coordinate che devono essere processate dalla trasformazione affine inversa. Dalle coordinate ricevute si individua il centro dell'area d'interesse: questo è il punto su cui bisogna applicare la funzione g inversa,

di parametri ricavati durante la calibrazione del sistema omnidirezionale(Capitolo 4). Nel capitolo 3, è stata studiata la funzione di trasformazione g, che rappresenta una trasformazione affine di ogni punto del mondo reale nel piano del sensore. La sua inversa indica come occorra calcolare la matrice inversa della matrice chiamata A e sottrargli il centro dell'immagine rappresentante il vettore di traslazione t, per ottenere da un punto dello specchio il suo corrispettivo nello spazio 3D. La coordinata z, come già largamente trattato nel capitolo precedente, viene approssimata grazie ad una serie di Taylor approssimante la funzione f; questa però ai fini pratici verrà trascurata in quanto l'inseguimento simulato si svolgerà su un piano fisso ($z = 0$).

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} C & D \\ E & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + t \quad [5.1]$$

Dove (u', v') è il punto rilevato sul piano del sensore, e quindi per ricavare le coordinate dei punti reali si calcola la sua inversa come segue:

$$A^{-1} \cdot \left[\begin{bmatrix} u' \\ v' \end{bmatrix} - t \right] = \begin{bmatrix} x \\ y \end{bmatrix} \quad [5.2]$$

Questa procedura è descritta all'interno del file `cam2world.nxc` .

5.2.5 Analisi del movimento del robot

Si assuma che il robot si muova su un sistema di coordinate polari mobile incentrato, con una certa approssimazione, sul robot stesso.Quindi ogni qual volta, il robot percepisce la posizione del target, lo identifica tramite una coppia di valori rappresentanti rispettivamente l'angolo di rotazione e la distanza che separa i due soggetti. La dinamica quindi è composta da due semplici step:

1. Rotazione di un angolo α
2. Diminuzione della distanza tramite moto lineare a tratti.

La rotazione del robot si sviluppa sull'asse del robot ortogonale all'asse delle sue ruote, passante per il punto medio. Infatti imprimendo la stessa velocità angolare alle due ruote ma con versp di rotazione opposto, il robot gira su se stesso facendo perno sul suo asse e possedendo la capacità di effettuare un controllo sull'angolo percorso da ciascun motore, è immediato decidere a priori la rotazione da eseguire con una buona precisione.

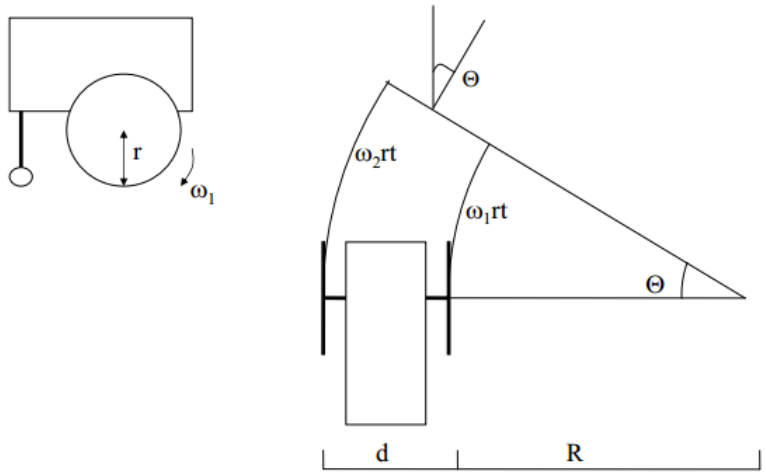


Figura 5.7: Moto curvo centrato sull'asse delle ruote del robot.

Siano a_1 e a_2, gli archi di circonferenza percorsi dalle due ruote del robot , con stesso angolo di rotazione Θ rispetto all'asse ortogonale a quello delle ruote, ma con diverso raggio di rotazione .

$$a_1 = r\theta_1 = R\Theta$$

$$a_2 = r\theta_2 = (R+d)\Theta$$

$$\Theta = \frac{a_1}{R} = \frac{a_2}{R}$$

Nell'ipotesi che le due ruote siano della stessa grandezza.

$$\omega_1(R+d) = \omega_2(R)$$

Per cui R non dipende da r (raggio della ruota) ma dalla distanza d tra le due ruote.Infatti:

$$R = d\frac{\omega_1}{\omega_1 - \omega_2}$$

Sostituendo a ω_1 il valore $-\omega_2$ il moto curvo avviene lungo l'asse del robot, punto medio tra la due ruote, difatti:
Per $\omega_1 \to -\omega_2$ si ottiene $R \to -\frac{d}{2}$

Inoltre l'angolo Θ nel caso che $\omega_1 \to -\omega_2$ e quindi $R \to -\frac{d}{2}$ vale:

$$\Theta = -2\theta\frac{r}{d}$$

Conoscendo il raggio delle ruote e la distanza tra esse, imprimendo ad ambi i servomotori due angoli di ugual modulo ma differente verso, è possibile gestire la rotazione con una precisione sufficiente .

Conclusa la rotazione, il moto del robot per avvicinarsi al target assume le caratteristiche dinamiche di un moto lineare a tratti: imprimendo una potenza costante del 60%, se il target si trova oltre di una certa distanza definita come distanza di guardia, e un altra potenza pari al 30% se il robot dista dal target una distanza minore o uguale della distanza di guardia. Questa tecnica è stata pensata per evitare, come già anticipato, collisioni e di dare il tempo al task col compito di monitorare la camera, di informare il robot di eventuali spostamenti improvvisi del target.

I task asincroni, peculiarità del linguaggio NXC, garantiscono un aggiornamento continuo sulle coordinate del target e quindi di un suo possibile spostamento, e se questo comportasse un'uscita dall'area controllata dal robot inseguitore, il task addetto allo spostamento farebbe compiere al robot una spirale (di raggio limitato) provando così di coprire una regione dello spazio più ampia possibile, fintanto che il task dedito alla ricerca del target non dia una nuova segnalazione .

Il movimento del robot è contenuto all'interno del file `movement.nxc`

Capitolo 6

Codice NXC

6.1 Task concorrenti

É importante comprendere che i task avviati vengono eseguiti in concorrenza. Questo comporta il dover gestire il problema del tentativo simultaneo di accedere alla medesima risorsa da parte dei task. Se queste molteplici richieste effettuate da parte di ogni task non vengono gestite, si va incontro a risultati inattesi. Ad esempio,se entrambi i task cercano di far muovere i motori simultaneamente, chi dei due (o più) task attivi ha la precedenza?.

Una tecnica standard per risolvere il problema è quello di usare una variabile per indicare quale dei task sta usando i motori. Gli altri task non sono abilitati ad usare i motori e il primo task non indica, usando la variabile, che ha liberato le risorse. Una variabile di questo tipo è chiamata semaforo.Questo semplice ma sufficente stratagemma verrà utilizzato prevalentemente per gestire l'accesso ai motori.

6.2 Codice

Invito a un'attenta lettura della tesi per avere un quadro completo sugli algoritmi utilizzati.

6.2.1 Codice nxtcamlib

Questo frammento di codice appartiene alla libreria `nxtcamlib.nxc` , nella quale oltre ad altre funzionalità non citate, le seguenti funzioni:

- NXTCam_Init: Inizializza uno stream in modalità Tracker Object con la NXTCam.

- NXTCam_GetBlobs: Invia dalla NXTCam le informazioni riguardanti i target trovati dalla modalità Tracker Object.
- NXTCam_PrepareColorMap:Precarica un buffer di 48 byte rappresentante la ColorMap.
- NXTCam_WriteColorMap:Carica nella ColorMap il buffer creato da NXTCam_PrepareColorMap.

```
int NXTCam_Init(byte camPort, byte camAddr)
{
        string s, msg;
        int i;

  SetSensorLowspeed(camPort);

  s = "";
        i = 0;
        Wait(100);
        // wait until you can read the vendor name mndsnsrs
        // compare only first letter.
        // try a few times
  while (s[0] != 'm' && i < 200 ) {
        s = i2cReadString(camPort, camAddr, 0x08, 8);
#ifdef MS_DEBUG
    msg = "connecting:";
    msg += s;
    msg += "  ";
    TextOut(0, LCD_LINE8, msg, true);
    Wait(10);
                msg = "i: ";
    msg += NumToStr(i);
    TextOut(0, LCD_LINE7, msg, false);
#else
    Wait(40);
#endif
                i++;
  }
  NXTCam_SendCommand(camPort, camAddr, 'A'); // Sort by size
  NXTCam_SendCommand(camPort, camAddr, 'E'); // Start finding
        return i;
}

/*
void NXTCam_GetBlobs() - loads the current blobs into the global data structure over
Get the blob information of all the blobs that NXTCam is tracking.
There could be upto 8 blobs being tracked by NXTCam.
This function will return color and coordinate information for all the blobs.
```

```
   If the NXTCam has found 3 blobs, the information for 4th, 5th, 6th and 7th blob is no
41 Port: the NXT port to which the device is connected.
   Nblobs: number of blobs found by the NXTCam
43 Color[]: a array of colors for the blobs
   Left[], top[], right[] bottom[]: the coordinate values of the blobs.
45 */
   void NXTCam_GetBlobs(byte port, int &nblobs,
47                       int &color[], int &left[],
                         int &top[], int &right[], int &bottom[])
49 {
     byte nByteReady = 0;
51   byte msg[3];
     byte reply[5];
53   int i;
     int n;
55   int x;
     byte buf[10];
57   string str;

59   // Initialize the returning arrays with zeros.
     for (i = 0; i < 10; i++) {
61     color[i]  = 0;
       left[i]   = -1;
63     top[i]    = -1;
       right[i]  = -1;
65     bottom[i] = -1;
     }
67
     NXTCam_Flush(port);
69
     // Request number of blobs from the count register
71   // NXTCAM_REG_COUNT is where number of detected objects are stored.
     ArrayBuild(msg, CAMADDR, NXTCAM_REG_COUNT);
73
           while (I2CStatus(port, nByteReady) == STAT_COMM_PENDING);
75
     n = I2CWrite(port, 1, msg);
77
     if ( n != NO_ERR ) {
79 #ifdef MS_DEBUG
       string msg2;
81     msg2 = "WriteError1:";
       msg2 += NumToStr(n);
83     msg2 += "  ";
       TextOut(0, LCD_LINE8, msg2, false);
85 #endif
       return;
87   }
```

```
89          while (I2CStatus(port, nByteReady) == STAT_COMM_PENDING);

91    // Get the reply and put into nblobs global
      n = I2CRead(port, 1, buf);
93    if ( n != NO_ERR ) {
  #ifdef MS_DEBUG
95      string msg2;
        msg2 = "ReadError1:";
97      msg2 += NumToStr(n);
        msg2 += "  ";
99      TextOut(0, LCD_LINE8, msg2, false);
  #endif
101     return;
      }
103         while (I2CStatus(port, nByteReady) == STAT_COMM_PENDING);

105   nblobs = buf[0]& 0x00FF;

107   // work around for out of bounds nblobs value.
      if ( nblobs < 0 || nblobs > 8 ) {
109     nblobs = 0;
      }
111
      // Get nblobs of blob data from the camera
113   for (i = 0; i < nblobs; i++) {

115     // Request blob data
        x = NXTCAM_REG_DATA+(i*5);
117     ArrayBuild(msg, CAMADDR, x);
                  while (I2CStatus(port, nByteReady) == STAT_COMM_PENDING);
119     n = I2CWrite(port, 5, msg);
        if ( n != NO_ERR ) {
121 #ifdef MS_DEBUG
          string msg2;
123       msg2 = "WriteError2:";
          msg2 += NumToStr(n);
125       msg2 += "  ";
          TextOut(0, LCD_LINE8, msg2, false);
127 #endif
          return;
129     }

131     // Get blob data reply
                  while (I2CStatus(port, nByteReady) == STAT_COMM_PENDING);
133
        n = I2CRead(port, 5, buf);
135     if ( n != NO_ERR ) {
  #ifdef MS_DEBUG
137       string msg2;
```

```
        msg2 = "ReadError2:";
139     msg2 += NumToStr(n);
        msg2 += "  ";
141     TextOut(0, LCD_LINE8, msg2, false);
#endif
143     return;
      }
145
                while (I2CStatus(port, nByteReady) == STAT_COMM_PENDING);
147
      // Put data into global variables.
149   color[i]  = buf[0] & 0x00FF;
      left[i]   = buf[1] & 0x00FF;
151   top[i]    = buf[2] & 0x00FF;
      right[i]  = buf[3] & 0x00FF;
153   bottom[i] = buf[4] & 0x00FF;

155   // NXTCam_Flush(port);
    }
157 /*

159 This function prepares colormap for the objects.
Colormap[]: the prepared colormap is stored in this variable.
161 Object: the object number (1 to 8) for this colormap being defined.
Red_low, green_low, blue_low: the lowest range of that color you need to match.
163 Red_high, green_high, blue_high: hte highest range of that color you need to match.

165 pass a colormap array (of size 48)
object numbers range from 1 to 8
167 pass low and high range color values of RGB

169 More about colormaps:
The colormap is a 48 byte buffer.
171 From this buffer each of RGB color is assigned 16 bytes for that color.
i.e. first 16 bytes are for Red, next 16 for green and remaining 16 for blue.
173
These 16 bytes store matching preference for that color's absolute value ranges as
175 0-16-32-48-64-80-96-112-128-144-160-176-192-208-224-240-255
i.e. first byte is set for color range 0 to 16, second byte for range 16 to 32 and so
177 each bit in the byte contains mask for each object (i.e. 8 objects) which is set to 1

179 example -
if you want 1st object to match red color in range of 16 to 48, you will set
181 first bit to 1 in byte 1 and byte 2.
or if you want 4th object to match any shade of red (0 to 255) then you will set
183 4th bit of all 16 bytes to 1.

185
*/
```

```
187 void NXTCam_PrepareColormap(byte & colormap[], byte object,
                              byte red_low, byte red_high,
189                           byte green_low, byte green_high,

191 {
        /* 0-16-32-48-64-80-96-112-128-144-160-176-192-208-224-240-255
193      */

195   byte mask, nmask;
        byte a;
197     byte i, rl, rh, gl, gh, bl, bh;

199     if ( object > 8 ) object = 8;

201   mask = 0x01 << (8-object);
        nmask = 0xFF ^ mask;
203
        rl = red_low/16;
205     rh = (red_high/16) + 1;
        gl = (green_low/16) + 16;
207     gh = (green_high/16) + 17;
        bl = (blue_low/16) + 32;
209     bh = (blue_high/16) + 33;

211     if ( rl <  0 || rl > 16) rl = 0;
        if ( rh <  0 || rh > 17) rh = 17;
213     if ( gl < 16 || gl > 32) gl = 16;
        if ( gh < 16 || gh > 33) gh = 33;
215     if ( bl < 32 || bl > 47) bl = 32;
        if ( bh < 32 || bh > 48) bh = 47;
217
        for (i=0; i < 48; i++) { colormap[i] |= mask; }
219
        for (i= 0; i< rl; i++) { colormap[i] &= nmask; }
221     for (i=rh; i< 16; i++) { colormap[i] &= nmask; }
        for (i=16; i< gl; i++) { colormap[i] &= nmask; }
223     for (i=gh; i< 32; i++) { colormap[i] &= nmask; }
        for (i=32; i< bl; i++) { colormap[i] &= nmask; }
225     for (i=bh; i< 48; i++) { colormap[i] &= nmask; }

227   return ;
  }
229
  void NXTCam_WriteColormap (byte colormap[], byte addr, byte port)
231 {
        string msg, x0,x1,x2,x3,x4,x5,x6,x7, i0, loc;
233     byte location, val0, val1, val2, val3, val4, val5, val6, val7;
        int i;
235   byte nByteReady = 0;
```

```
    byte message[20];
237
        for (i=0; i< 6; i++) {
239       location = 0x80 + (i*8);
              val0 = colormap[(i*8)+0];
241           val1 = colormap[(i*8)+1];
              val2 = colormap[(i*8)+2];
243           val3 = colormap[(i*8)+3];
              val4 = colormap[(i*8)+4];
245           val5 = colormap[(i*8)+5];
              val6 = colormap[(i*8)+6];
247           val7 = colormap[(i*8)+7];

249           ArrayBuild(message, addr, location,
                                                                                  val0,
251           while (I2CStatus(port, nByteReady) == STAT_COMM_PENDING);
              I2CWrite(port, 0, message);
253
              while (I2CStatus(port, nByteReady) == STAT_COMM_PENDING);
255
              x0 = NumToStr(val0);
257           x1 = NumToStr(val1);
              x2 = NumToStr(val2);
259           x3 = NumToStr(val3);
              x4 = NumToStr(val4);
261           x5 = NumToStr(val5);
              x6 = NumToStr(val6);
263           x7 = NumToStr(val7);

265           i0 = NumToStr(i);
              loc = NumToStr(location);
267           msg = "loc " + loc + ":    ";
              TextOut(0, LCD_LINE2, msg, false);
269           msg = " " +x0+ " " +x1+ " " +x2 + " " + x3 + "    ";
              TextOut(0, LCD_LINE3, msg, false);
271           msg = " " + x4+ " " +x5+ " " +x6+ " " +x7 + "    ";
              TextOut(0, LCD_LINE4, msg, false);
273
              while (I2CStatus(port, nByteReady) == STAT_COMM_PENDING);
275           NXTCam_SendCommand(port, addr, 'S'); // Save the colormap
              while (I2CStatus(port, nByteReady) == STAT_COMM_PENDING);
277           Wait(2000);
        }
279
    }
281 }
```

6.2.2 Caricamento del colore prescelto

Utilizzando le due funzioni presentate precedentemente:NXTCam_PrepareColorMap e NXTCam_WriteColorMap, salva nel registro ColorMap un colore prescelto(nell'esempio un rosso acceso).

```
/*****************************************************************************/
/*
*/
/* Program Name: colormap-write.nxc
*/
/* ========================
*/
/* Load on ColorMap the color value .
*/
/*
*/
/*
*/
/*****************************************************************************/


#include "nxtcamlib.nxc"

const byte sensorPort  =  IN_1;
#define ADDR     0x02

//Print sensor information from its register
void ShowSensorInfo(byte prt, byte Addres )
{
   if( i2cread(prt,Addres,0x00,1)>1)
   {
                  // sensor name
     TextOut(0, LCD_LINE8, i2cReadString(prt, Addres, 0x10, 8), false);
                  // sensor firmware version.
     TextOut(50, LCD_LINE8, i2cReadString(prt, Addres, 0x00, 8), false);

   }
}

task main()
{
  byte colormap[48];
        int init;

  init = NXTCam_Init(sensorPort, ADDR);
        NXTCam_PrepareColormap(colormap, 1, 80,144,0,32,0,32);
        NXTCam_WriteColormap(colormap, ADDR, sensorPort);
```

```
38
}
```

6.2.3 Tracking

Il seguente file si incarica di ricercare le coordinate reali di un target visualizzato nello specchio dalla telecamera, utilizzando l'oppurtuna funzione cam2world, che applica la trasformazione affine per tale scopo.

```
1 /*************************************************************************/
/*
*/
3 /* Program Name: findtarget.nxc
*/
/* ========================
*/
5 /* Return the coordinates of a target found on sensor image plane  .
*/
/* this piece of code use NXTCam View on Tracker Object Mode
*/
7 /*
*/
/*************************************************************************/
9
const byte camPort  =  IN_1;
11
#define CAMADDR     0x02
13 #include "nxtcamlib.nxc"
#include "cam2world.nxc"
15 float nan=-1;
float x3D;
17 float y3D;
float z3D;
19
float getXPosition(){
21
      return x3D ;
23 }
float getYPosition(){
25
      return y3D;
27 }
float getZPosition(){
29
      return z3D;
31 }
task findTargetPosition ()
33 {
```

```
35  x3D= sqrt(nan);
    y3D= sqrt(nan);
37  z3D= sqrt(nan);
        int nblobs;
39  int stop_now = 0;
    int bc[10];
41  int bl[10];
    int bt[10];
43  int br[10];
    int bb[10];
45      int init, n, i;
        int center_x, center_y;
47  string msg, topdown, leftright;

49  init = NXTCam_Init(camPort, CAMADDR);          //Initialize stream between BrickNXT

51 /*
   *  This program assumes one color is
53 *  uploaded on your NXTCam and tries
   *  to keep object of that color at the
55 *  center of the NXTCam.
   */
57

59  while (1) {
        //      TextOut(40, LCD_LINE1, "", true);
61    n ++;
      NXTCam_GetBlobs(camPort, nblobs, bc, bl, bt, br, bb);      //Get region of a dete
63              if ( nblobs > 0 ) {

65      center_y = (bl[0] + br[0])/2;
                  center_x = bb[0];
67
        msg = "CX ";
69      msg += NumToStr(i);
        msg += ": ";
71      msg += NumToStr(center_x);

73      msg = "CY ";
        msg += NumToStr(i);
75      msg += ": ";
        msg += NumToStr(center_y);
77
        msg = "                ";
79
        float  y_calib=center_y*521/144;     //     Enlarge the sensor
  image center, for the calibration
81      float x_calib=center_x*395/176;
```

```
83        cam2world(x_calib,y_calib);
          x3D=getX3D()/(395/176)*(-1);          //Reduce the coordinates of the sensor
   image center.
85        y3D=getY3D()/(521/144);
          z3D=getZ3D();
87        TextOut(40, LCD_LINE1, "X:"+ NumToStr(x3D), false);
          TextOut(40, LCD_LINE2,"Y:"+ NumToStr(y3D), false);
89
              }
91            else{
              if (!(isNAN(x3D)))
93                    Wait(1000);

95
              x3D= sqrt(nan);     //if i don't see any target , load NaN value on coordina
97            y3D= sqrt(nan);
              z3D= sqrt(nan);}
99          }

101 }
```

```
 1 /*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
   /*                                                                               */
   */
 3 /*Project Name: cam2world.nxc
   */
   /*===============================
   */
 5 /*Conversion of coordinates , with Affine Trasformation
   */
   /*studied in chap.3
   */
 7 /*
   */
   /*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
 9


11
   //value stored on ocam_calib.ss   result of the calibration's toolbox
13 const float NUMBERS_OF_DATA =1.00;
   const float CALIB_PARAM_SS_0 =-125.4936;
15 const float CALIB_PARAM_SS_1 =0.000;
   const float CALIB_PARAM_SS_2=0.00025        ;
17 const float CALIB_PARAM_SS_3 =-0.00           ;
   const float CALIB_PARAM_SS_4 =0.00            ;
19 const float CALIB_PARAM_XC =190               ;
   const float CALIB_PARAM_YC= 400          ;
21 const float CALIB_PARAM_WIDTH = 521      ;
```

```
const float CALIB_PARAM_HEIGHT=  395      ;
const float CALIB_PARAM_C = 1.0024            ;
const float CALIB_PARAM_D = 0.0019             ;
const float CALIB_PARAM_E =0.0023    ;
const float UNO =1.0001             ;
const float N_POINTS=1.000                 ;


float x_target_3D;
float y_target_3D;
float z_target_3D;


float getX3D()
{
      return x_target_3D;
}
float getY3D()
{
      return y_target_3D;
}
float getZ3D()
{
      return z_target_3D;
}
void cam2world(int x_target_2d, int y_target_2d ){


      int n_points=N_POINTS;

      float xc= CALIB_PARAM_XC ;     //Load center image point.
      float yc= CALIB_PARAM_YC;


      float A_matrix [2][2]={{CALIB_PARAM_C,CALIB_PARAM_D},{CALIB_PARAM_E,UNO}};
//Load A(affine trasformation) matrix.

      float determinant=A_matrix[0][0]*A_matrix[1][1]-A_matrix[0][1]*A_matrix[1][0];
//Determinant of matrix A .


      //Determinate the inverse matrix A^-1
      const float mat00=  (A_matrix[1][1]/determinant);
      const float mat01=  -1*(A_matrix[1][0]/determinant);
      const float mat10= -1* (A_matrix[0][1]/determinant);
      const float mat11=  (A_matrix[0][0]/determinant);
```

```
69      // float inverse_A_matrix[2][2]={{mat00,mat01},{mat10,mat11}};

71

73      //Traslation
        x_target_2d=x_target_2d-xc;
75      y_target_2d=y_target_2d-yc;

77      //Calculation of X ,Y and Z 3D point.
        x_target_3D = mat00*x_target_2d+mat01*y_target_2d;
79      y_target_3D=mat10*x_target_2d+mat11*y_target_2d;
        float xx3D=pow(x_target_3D,2);
81      float yy3D=pow(y_target_3D,2);
        z_target_3D= CALIB_PARAM_SS_4*(pow(sqrt(xx3D+yy3D),4) )
83      +CALIB_PARAM_SS_3*(pow(sqrt(xx3D+yy3D),3))
        +CALIB_PARAM_SS_2*(pow(sqrt(xx3D+yy3D),2))
85      +CALIB_PARAM_SS_1*((sqrt(xx3D+yy3D)))
        +CALIB_PARAM_SS_0;
87

89

        return ;
91

93 }
```

6.2.4 Movimento del tracker

Il file `movement.nxc` governa l'esecuzione di tutti i file appena citati e di `movementRobot`, subtask dedito a controllare rotazione e avanzamento del robot conseguentemente alle informazioni inviate dagli altri subtask.

```
/*//////////////////////////////////////////////////////////////////////*/
2 /*
*/
/* Program Name: movement.nxc
*/
4 /* ==========================
*/
/*
*/
6 /*Move robot tracker to a target with a determinate color stored on
*/
/*ColorMap.
*/
8 /*
*/
```

```
/*
*/
/*****************************************************************************/

#include "findtarget.nxc"

bool isInRotation = false;       //Semaphore variable, if is true task of refresh coor
float distanza=0.0;              //distance between tracker and target
float degreeRot=0.0;             //Degree rotation (radial reference)
const  int  AXE_LENGTH=100;        //distance between wheels
const int   ANGLE_SPEED=50;       //angolar speed
const int  RADIUS_WHEEL =28;        // radius wheel
float x_3D=0.0;                     //where is the target .
float y_3D=0.0;


float getDistance(float  x, float y)
{

       //Determination of distance .
       if (isNAN(x)){
         return -1;     //invalid distance if i don't see a target..
       }
       else{
            return sqrt(x*x+y*y);         //distance between target and tracker
       }
}
float getDegreeRotation(int x,int y)
{     if (isNAN(x)){
         return 0;     //I don't rotate if i don't see a target(..spiral movement..)
      }
      else                  //else.. rotation between [-PI,PI)
      {
          if (x>0) {
                return atan(y/x);
          }
          else {
                if (y<0)
                      return    atan(y/x)-PI;
                else
                      return atan(y/x)+PI;
          }
       /*   if (y>0) {
                return atan(x/y);
          }
          else {
                if (x<0)
                      return    atan(x/y)-PI;
```

```
                else
58                      return atan(x/y)+PI;
            }       */
60      }

62 }
   void rotateRobot(float degreeRef){
64
        float degree=degreeRef;    //Omega, degree's rotation of the tracker
66      isInRotation=true;         //I don't refresh target position while tracker rotati

68

70    //  TextOut(0, LCD_LINE5, "OMEGA:  "+NumToStr(degree), false);
        float wheelDegree=    -1*(degree/2)*(AXE_LENGTH/RADIUS_WHEEL);//See chapter 5, c
72      degree= degree*180/(PI);//rad to degree  [-180°,180°)
        wheelDegree =       wheelDegree*180/(PI);
74      TextOut(0, LCD_LINE4, "THETA:  "+NumToStr((wheelDegree)), false);
       TextOut(0, LCD_LINE5, "OMEGA:  "+NumToStr(degree), false);
76
        if  ((degree)<0){
78

80            PlayTone(100,200);
              RotateMotorEx(OUT_AB, ANGLE_SPEED, (wheelDegree),100, true, true);
82                      //clockwise direction      .

84      }
        else{
86
            TextOut(0, LCD_LINE4, "THETA:  "+NumToStr(wheelDegree), false);
88          RotateMotorEx(OUT_AB, ANGLE_SPEED, wheelDegree,-100, true, true);

90                              //Anticlockwise direction       ..

92

94      }
        isInRotation=false;
96      Wait(800);
        //   Unlock semaphore 'isInRotation'  ,now i can refresh target's position
98      Off(OUT_AB); //stop the motor

100

102 }
   task  getXYTarget()             // refreshing target's position Task
104 {
        while (1){
```

```
106             while (!isInRotation){

108                     x_3D=getXPosition();
                        y_3D=getYPosition();
110                     distanza=getDistance(x_3D,y_3D);
                        degreeRot=getDegreeRotation(x_3D,y_3D);
112                     TextOut(0, LCD_LINE6, "X3D:   "+NumToStr(x_3D), false);
                        TextOut(0, LCD_LINE7, "Y3D:   "+NumToStr(y_3D), false);
114
                        Wait(500);
116             }
                Wait(500);
118     }

120 }
    task movementRobot()
122 {
            int i;
124         while (1){
            //move the robot to the target..
126         i=100;
            while (isNAN(x_3D)){
128

130                 OnFwd(OUT_A, ANGLE_SPEED);                  //spiral movement
    .. i don't find any target..
                    OnFwd(OUT_B, ANGLE_SPEED);
132
                    Wait(i);
134                 OnRev(OUT_A, ANGLE_SPEED);                          //clockwise direct
    ..
                    OnFwd(OUT_B, ANGLE_SPEED);
136
                    Wait(100);
138                 Off(OUT_AB);
                    if (i<700)
140                     i+=100;
            }
142         //VADO IN ROTAZIONE

144         if ((abs(degreeRot))> ((PI/9)*4) )
            {
146
               rotateRobot(degreeRot);
148         }
           while (((abs(degreeRot))<((PI/9)*4))&&(distanza>20)){
150                 OnRev(OUT_A, 60);                           //Go to the target.. whil
                    OnRev(OUT_B, 60);
152                 while ((distanza<=30)&&(distanza>20)&&(abs(degreeRot)<((PI/9)*4))){
```

```
                    OnRev(OUT_A, 40);
154                 OnRev(OUT_B, 40);
                }
156         }
            Off(OUT_AB);
158         while ((distanza<=20)&&(distanza>0)){
                PlayTone(100,100);      //target vicino a me.. suono Allarme
160
            };
162
            Wait(500); //ERA 800  METTO 500
164
            }
166


168
    }
170 task main()
    {
172
        isInRotation=false  ;
174
        start findTargetPosition;       // see  'findtarget.nxc'
176     start getXYTarget;             // identify Target by degree's rotation and distan
        start movementRobot;                // this task compute movement tracker
178
    }
```

Capitolo 7

Conclusioni

Conseguentemente alle descrizioni riportate in questo lavoro di tesi sono noti i problemi causati dai limiti tecnologici degli strumenti utilizzati, in quanto seppur il modulo NXTCam-V2 presenti al suo interno la modalità Tracker Object, di un indiscutibile praticità e semplicità, la sua limitata risoluzione d'immagine e i problemi legati agli effetti di una luminosità non omogenea dell'ambiente che la circonda, limita sia la velocità dell'inseguitore che quella del target mobile. Difatti questo vincolo comporta che il robot inseguitore riceva informazioni delle coordinate del target quanto più corrette, poiché a velocità superiori si produrrebbe un errato rilevamento dei contorni del target da parte della camera, distorcendo oltremisura i frame scattati e portando così ad un errato tracking del robot. Una valutazione della strumentazione utilizzata consente di affermare che avendo a disposizione un sistema omnidirezionale più centrale possibile, la simulazione risulterebbe sicuramente più realistica e significativa. Nonostante ciò, la tecnologia LEGO MINDSTORMS si è dimostrata ancora una volta estendibile ad un nuovo tipo di sperimentazione, svincolata dai normali scopi didattici, dimostrando una notevole flessibilità.

Bibliografia

[A1] D. Scaramuzza, R. Siegwart, A Practical Toolbox for Calibrating Omnidirectional Cameras, Vision Systems Book, 2007.

[A2] D. Scaramuzza, R. Siegwart, A New Method and Toolbox for Easily Calibrating Omnidirectional Cameras, Proc. of The International Conference on Computer Vision Systems (ICVS), Workshop on Camera Calibration Methods for Computer Vision Systems, 2007.

[A3] D. Scaramuzza, A. Martinelli, R. Siegwart, A Toolbox for Easy Calibrating Omnidirectional Cameras.

[A4] D. Scaramuzza, A. Martinelli, R. Siegwart, A Flexible Technique for Accurate Omnidirectional Camera Calibration and Structure from Motion, Proc. of The IEEE International Conference on Computer Vision Systems (ICVS), 2006.

[A5] S. Baker and S. K. Nayar. A theory of catadioptric image formation. In Proceedings of the International Conference on Computer Vision (ICCV), pages 35-42. Narosa Publishing House, January 1998.

[A6] Micusik B.: Two View Geometry of Omnidirectional Cameras, PhD Thesis, TR No. CTU–CMP–2004–07, Center for Machine Perception, Czech Technical University in Prague, 2004.

[S1] Lego Mindstorm , *http://mindstorms.lego.com/*

[S2] NXTCam V2, *http://www.mindsensors.com/index.php?module=pagemasterPAGEuserop=vieu*

[S3] OCamCalib toolbox , *https://sites.google.com/site/scarabotix/ocamcalib-toolbox*

[S4] Matlab, *http://www.mathworks.it/products/matlab/*

[S5] NXTCamView, *http://nxtcamview.sourceforge.net/*

[S6] Bricxcc, *http://bricxcc.sourceforge.net/*

[S7] Wikipedia,*http://www.wikipedia.com*