Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Sistema di visione cognitiva per la rappresentazione interna di un robot

A. Chella, I. Infantino, I. Macaluso, G. Scardino

Rapporto Tecnico N.:
RT-ICAR-PA-05-04 **aprile 2005**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Sistema di visione cognitiva per la rappresentazione interna di un robot

A. Chella[2], I. Infantino[1], I. Macaluso[2], G. Scardino[1]

Rapporto Tecnico N.:
RT-ICAR-PA-05-04

Data:
aprile 2005

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sezione di Palermo Viale delle Scienze edificio 11 90128 Palermo
[2] Università degli Studi di Palermo Dipartimento di Ingegneria Informatica Viale delle Scienze 90128 Palermo

SOMMARIO

Il lavoro che si propone è quello di associare un motore grafico tridimensionale che crea in tempo reale l'ambiente in cui il robot si muove; in questo modo si ha una conoscenza tridimensionale dell'ambiente, che può essere d'aiuto nell'elaborazione dei dati provenienti dall'esterno. Un possibile confronto dei dati è quello fra le immagini provenienti dalle telecamere con le immagini create dal simulatore grafico tridimensionale, in modo tale da effettuare delle operazioni di riconoscimento di oggetti o dell'ambiente facendo riferimento alle immagini sintetiche. Un altro confronto è possibile farlo con la lettura dei laser del robot con la lettura dei laser simulati, questo porta anche a filtrare i dati provenienti dalla struttura dell'ambiente in modo da individuare oggetti esterni alla struttura stessa.

In questa tesi verrà anche trattato un modulo di autolocalizzazione per il robot RWI - B21r sfruttato in un'applicazione di guida museale robotica. I problemi affrontati in questo modulo riguardano la distorsione dell'immagine, la triangolazione e la ricerca di marker in un'ambiente.

INDICE

Indice iv
Indice delle Figure vi
1. Introduzione 1
1.1. Sistema di visione cognitiva 1
1.1.1. L'Area Concettuale 2
1.2. Rappresentazione interna della scena 5
2. Interazione del livello sensoriale con la pianificazione 7
2.1. La percezione attraverso le tecniche di visione artificiale 9
2.1.1. Correzione della distorsione e calibrazione 9
2.1.2. Ricerca del Marker 14
2.1.3. Triangolazione 16
2.1.4. Autolocalizzazione del robot basata sulla visione 18
2.1.5. Tecniche per il miglioramento dell'autolocalizzazione 20
3. Rappresentazione interna mediante simulatore 3D 23
3.1. Simulatore usato 23
3.2. Introduzione dei dati dei sensori laser nella rappresentazione interna della scena 25
3.3. Matching basato su marker 27
4. Sperimentazioni 28
4.1. Coppia stereo 30
4.1.1. Sensori di acquisizione immagini del robot 30
4.1.2. Unità Pantilt 32
4.2. Dati relativi alla triangolazione 33
4.3. Analisi dell'errore di autolocalizzazione 34
4.4. Scenario 1: Dipartimento di Ingegneria Informatica - "DINFO" 37
4.5. Scenario 2: Museo Archeologico di Agrigento 40
4.5.1. Autolocalizzazione nel museo 41
4.6. Rilevamento dinamico di ostacoli 43
5. Architettura distribuita e possibili estensioni 47

5.1. Distribuzione dei moduli di elaborazione 47
5.2. Prototipo di interazione Utente – Robot tramite sistema palmare..... 50
5.2.1. Scelta del motore grafico 3D per il palmare 52
6. Bibliografia..... 55
Appendici..... 57
Descrizione della struttura di Quake 2 57
Modifiche effettuate ai vari file di Quake 2..... 58
Programma di controllo 64
Visualizzazione Mappa..... 66
Visualizzazione Entità..... 67
Modifica delle immagini..... 69
Architettura del sistema di visione del Robot RWI-B21r 78
Programma di Visione..... 80
Programma di Sintesi Vocale 83

INDICE DELLE FIGURE

Figura 1-1 L'architettura basata su spazi concettuali 2
Figura 1-2 Esempio di immagine ricostruita virtualmente e di immagine reale 6
Figura 2-1 Modello dell'interazione tra conoscenza gestita tramite spazi concettuali e pianificazione 8
Figura 2-2 Esempio di distorsione dell'immagine 10
Figura 2-3 Esempio di scacchiera per la calibrazione 12
Figura 2-4 Immagine con distorsione corretta 13
Figura 2-5 Punti di intersezione del marker trovati 15
Figura 2-6 Triangolazione con due telecamere 16
Figura 2-7 Calcolo coordinate spaziali del Robot 18
Figura 2-8 Intersezione di due circonferenze usate nell'algoritmo di localizzazione 20
Figura 2-9 Marker visto frontalmente e visto di sbiego 21
Figura 3-1 Differenze fra i due motori 3D utilizzati 24
Figura 3-2 Sfasamento delle letture laser reali e virtuali 26
Figura 4-1 Robot RWI B21r 28
Figura 4-2 Telecamera Sony XC–999 30
Figura 4-3 Unità Pan–Tilt del robot B21r 32
Figura 4-4 Triangolazione con due telecamere 33
Figura 4-5 Porzione della mappa del DINFO 37
Figura 4-6 Esempio di sovrapposizione fra immagine reale e simulata 38
Figura 4-7 Esempio di sovrapposizione fra immagine reale e simulata con trasparenza diversa 38
Figura 4-7 Sovrapposizione fra immagine reale e sintetica per il riconoscimento di un'entità nell'ambiente. 39
Figura 4-8 Riconoscimento di una porta nell'ambiente simulato 39
Figura 4-9 Sala dei Telamoni del museo archeologico di Agrigento 41
Figura 4-10 Esempio di ostacolo non conosciuto a priori 43
Figura 4-11 Matching di letture laser 43
Figura 4-12 Rappresentazione delle letture del laser reali e simulate 45

Figura 5-1 Architettura distribuita completa del Robot 47
Figura 5-2 Sistema Robot - Palmare..... 51
Figura 5-3 Dettaglio sistema Robot – Palmare 51
Figura 6-1 Finestra principale programma di controllo..... 64
Figura 6-2 Finestra della mappa 66
Figura 6-3 Riconoscimento di due porte 67
Figura 6-4 Finestra di elaborazione delle immagini..... 70
Figura 6-5 Applicazione del filtro Edge Detection..... 71
Figura 6-6 Modifica della Luminosità e del Contrasto 71
Figura 6-7 Modifica dei valori del filtro Soglia 72
Figura 6-8 Finestra del filtro Harris..... 73
Figura 6-9 Risultato del filtro Harris 74
Figura 6-10 Finestra della matrice di convoluzione 75
Figura 6-11 Finestra di visualizzazione degli istogrammi..... 77
Figura 6-12 Architettura dei computer usati durante gli esperimenti..... 78
Figura 6-13 Architettura del Programma di Visione 80
Figura 6-14 Finestra di Debug del programma di Visione..... 81

1. INTRODUZIONE

In questo primo capitolo di questa tesi verrà illustrata l'architettura degli spazi concettuali per quanto concerne la visione robotica; seguirà una descrizione della rappresentazione interna che un robot può avere, ponendo l'attenzione su l'utilizzo di una rappresentazione virtuale tridimensionale.

1.1. Sistema di visione cognitiva

I dati provenienti dai sensori di visione devono essere opportunamente elaborati per creare una base di conoscenza accessibile a vari livelli di astrazione. Il processo di acquisizione, elaborazione, rappresentazione simbolica può essere trattato in maniera unificata attraverso una definizione di un'architettura cognitiva. In questa tesi vengono implementati e sperimentati algoritmi di visione artificiale che si inseriscono su un'architettura basata su spazi concettuali [1],[2],[3]. Tale architettura è organizzata in due aree computazionali come mostrato in Figura 1-1. L'area subconcettuale concerne con il processo a basso livello dei dati percettivi che vengono da sensori di visione. Può essere visto come una parte filogenetica semplificata di un sistema che implementa i comportamenti basati su un set di meccanismi visuali incorporati come ad esempio la segmentazione dei colori o l'estrazione dei contorni. Si può chiamare area subconcettuale perché una categorizzazione concettuale dell'informazione non è stata ancora effettuata.

Nell'area concettuale, l'informazione dall'area subconcettuale è organizzata in categorie concettuali che sono ancora indipendenti da ogni caratterizzazione linguistica. L'informazione dall'area subconcettuale è organizzata in categorie concettuali attraverso una serie di spazi metrici con un livello incrementale della complessità rappresentativa, nella quale sono acquisite diverse proprietà della scena percepita. Tale area è anche appropriata per ancorare dati percepiti in categorie linguistiche per un'ulteriore elaborazione cognitiva.

Nell'area linguistica, la rappresentazione e l'elaborazione sono basati sulla classificazione di primitive di movimento.

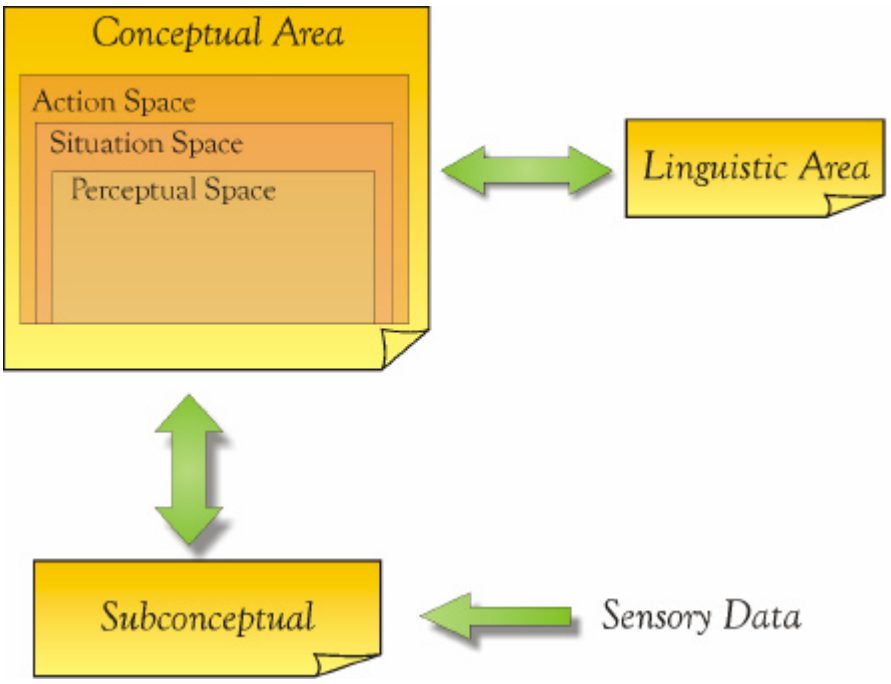


Figura 1-1 L'architettura basata su spazi concettuali: i dati provenienti dai sensori di visione devono essere opportunamente elaborati per creare una base di conoscenza accessibile a vari livelli di astrazione. Il processo di acquisizione, elaborazione, rappresentazione simbolica può essere trattato in maniera unificata attraverso una definizione di un'architettura cognitiva.

L'area sub-concettuale riguarda l'elaborazione a basso livello dei dati percepiti che provengono dai sensori di visione. Può essere vista come una parte filogenetica semplificata di un sistema che implementa i comportamenti basati su un set di meccanismi visuali incorporati. Essa reagisce allo stimolo visuale, per esempio ad un particolare colore o forma e compie un set di semplici azioni su quello stimolo, come ad esempio la segmentazione del colore dell'immagine o l'estrazione di particolari caratteristiche. Questo è equivalente agli istinti genetici in sistemi biologici e ad ogni istinto elementare fa associare una funzione, con la quale calcola un set di quantità per un'ulteriore elaborazione; ad esempio se viene trovato un obiettivo, si può classificare e si può calcolare la sua posizione ed orientamento in riferimento ad alcuni sistemi di riferimento.

1.1.1. L'Area Concettuale

Gli spazi concettuali sono basati sul trattamento geometrico di *concetti* e sulle rappresentazioni di conoscenza [13], dove i concetti non sono indipendenti l'uno

dall'altro, ma sono strutturati in *domini* (per esempio il colore, la forma, le cinematiche e quantità dinamiche e così via). Ogni dominio è definito in termini di un set di *dimensioni di qualità* (per esempio il dominio di colore può essere descritto in termini di quantità di tinta / saturazione / intensità, mentre il dominio della cinematica è descritto da angoli). In uno spazio concettuale, gli oggetti con la loro evoluzione temporale e spaziale sono rappresentati come punti che descrivono le dimensioni di qualità di ogni dominio. Di conseguenza, è possibile misurare la somiglianza tra individui diversi nello spazio per mezzo di una metrica appropriata. Nel contesto dell'apprendimento dall'ambiente vengono proposti tre spazi concettuali che inglobano diversi aspetti temporali e spaziali della scena come descritti di seguito.

The Perceptual Space (PS). Lo spazio percettivo è uno spazio metrico le cui dimensioni sono strettamente relazionate con le quantità processate nell'area subconcettuale. Per analogia col termine pixel, un punto nello PS è chiamato knoxel e può essere considerato un elemento primitivo nel livello considerato dell'analisi. I Knoxels nel PS sono di solito primitive geometriche che descrivono la scena statica acquisita (e.g. forma di oggetto e descrittori di posizione, angoli).

The Situation Space (SS). Nello Spazio delle Percezioni gli oggetti in movimento dovrebbero essere rappresentati come un set di punti che corrispondono ad ogni istante di tempo. Comunque, tale rappresentazione non cattura il moto nella sua interezza. Per dare conto degli aspetti dinamici delle azioni si propone il *Situation Space* SS nel quale ogni punto rappresenta un semplice moto. In questo senso, lo spazio dinamico può essere considerato una "esplosione" dello spazio statico nella quale ogni asse è diviso in un numero di nuovi assi, ognuno corrisponde a descrittori di moto appropriati.

The Action Space (AS). In una *Situation*, il moto di tutti i componenti nella scena avviene simultaneamente, per esempio corrispondono ad una sola configurazione di knoxels nello spazio concettuale. Considerando una composizione di molti moti organizzata in accordo ad una sequenza temporale, si può introdurre una nozione dell'azione secondo Allen [6]. Un'azione corrisponde ad un "diffusione" da una *Situation* ad un'altra *Situation* di knoxels nello spazio concettuale. Si può presumere

che le situazioni all'interno di un'azione sono separate da eventi istantanei. Nella transizione tra due configurazioni consecutivi, avviene almeno una "diffusione" di uno knoxel. Questo corrisponde ad una discontinuità nel tempo che è associato ad un evento istantaneo. Ogni punto nello spazio delle azioni è una raccolta di situazioni. In questo contesto, le primitive di movimento sono considerate una sequenza di azioni che portano a termine un comportamento meta-diretto [18].

1.2. Rappresentazione interna della scena

Uno dei grandi problemi della robotica è l'interpretazione tramite la visione dell'ambiente esterno. Generalmente i robot fino ad oggi hanno fatto uso di telecamere o sensori di prossimità per avere una conoscenza dell'ambiente in cui si muovono; molto spesso si fa uso contemporaneamente sia di telecamere che di sensori. Un compito complesso e difficile è rappresentato dal riconoscimento degli oggetti che si trovano nell'ambiente in cui il robot si muove, che si affianca a quello di ricostruire le strutture dell'ambiente stesso. Con i sensori di prossimità, è possibile sapere se c'è o meno un ostacolo intorno al robot, ma non è possibile identificarlo, a meno che l'ostacolo stesso abbia delle particolari caratteristiche o emetta un segnale di riconoscimento.

Una soluzione largamente utilizzata è quella di utilizzare marker artificiali, in modo da avere riferimenti univoci che possano essere estratti dalle immagini con una complessità computazionale limitata e che consenta l'implementazione di sistemi real-time. Tramite l'informazione derivata dai marker il robot può stabilire la sua posizione all'interno della scena in cui si muove.

Il lavoro che si propone è quello di associare un motore grafico tridimensionale che crea in tempo reale l'ambiente in cui il robot si muove; in questo modo si ha una conoscenza tridimensionale dell'ambiente, che può essere d'aiuto nell'elaborazione dei dati provenienti dall'ambiente esterno. Si pensi che, i dati provenienti dalle telecamere possono essere confrontati con le immagini create da un simulatore grafico, in modo tale da effettuare delle operazioni di riconoscimento di oggetti o dell'ambiente sulle immagini sintetiche. Un altro confronto è possibile farlo con la lettura dei laser del robot con la lettura dei laser simulati; questo porta anche a filtrare i dati provenienti dalla struttura dell'ambiente in modo da individuare oggetti esterni alla struttura stessa. Un grande vantaggio nell'uso del motore grafico che crea una ricostruzione virtuale della scena accettabile è quello di avere un'immagine sintetica, che riproduce la struttura dell'ambiente senza le alterazioni o rumori che si possono avere nella realtà. La sovrapposizione dell'immagine reale con quella sintetica aiuta a poter individuare più facilmente gli

eventuali oggetti che non fanno parte della struttura dell'ambiente stesso. In Figura 1-2 si può vedere un esempio di immagine costruita dal motore grafico 3D, che rappresenta la struttura di un corridoio; come si può notare non ci sono molte imperfezioni che possono complicare il lavoro di riconoscimento di oggetti.

Figura 1-2 Esempio di immagine ricostruita virtualmente e di immagine reale. E' evidente la necessità di dover allineare i due punti di vista per consentire confronti significativi che portino all'estrazione di informazioni utili alla navigazione del robot.

Per struttura dell'ambiente si intende la ricostruzione di elementi dell'ambiente stesso che non possono essere spostati facilmente. Ovviamente questo tipo di approccio può essere applicato a robot che fanno uso di una programmazione deliberativa. Tutte queste operazioni servono per poter estrarre delle informazioni dall'ambiente e utilizzare questa conoscenza sul mondo per poter muoversi in sicurezza.

2. INTERAZIONE DEL LIVELLO SENSORIALE CON LA PIANIFICAZIONE

Uno spazio concettuale è uno spazio metrico le cui dimensioni sono relazionate alla quantità elaborata dai sensori del robot. Alcune dimensioni sono relazionate alla forma dell'oggetto, mentre altre dimensioni sono relazionate al dislocamento nello spazio del movimento delle forme 3D. Il termine *knoxel* viene usato per denotare un punto nello spazio concettuale. Per dare conto della percezione delle scene dinamiche, si sceglie di adottare uno spazio concettuale intrinsecamente dinamico. Il robot con uno spazio concettuale interno è adibito a rappresentare entità in movimento e che interagiscono con il sistema in questo modo il generico movimento di un oggetto è rappresentato nella sua totalità, piuttosto che come una sequenza di frame statici.

Lo spazio concettuale dinamico permette al robot di immaginare le possibili interazioni future con l'ambiente. L'evoluzione della scena è rappresentata come una sequenza di set di knoxel che è immaginata e simulata nello spazio concettuale prima che l'interazione effettivamente rientri nelle aspettative del mondo reale.

Il robot, utilizza la sua aspettativa per la scelta di un obiettivo, esso genera un piano di azioni per produrre le situazioni favorevoli. Questo livello di rappresentazione può essere considerata come un'immaginazione mentale del robot. Durante l'esecuzione del piano, il robot percepisce l'ambiente e può comparare i risultati aspettati delle sue azioni, come sono simulati durante la generazione del piano, e i risultati reali in accordo con la sua percezione corrente.

A questo livello di astrazione è possibile associare ai diversi comportamenti del robot vari tipi di personalità, distinguibili attraverso le modalità con cui si aggiornano le aspettative e si condizionano i piani futuri di azione.

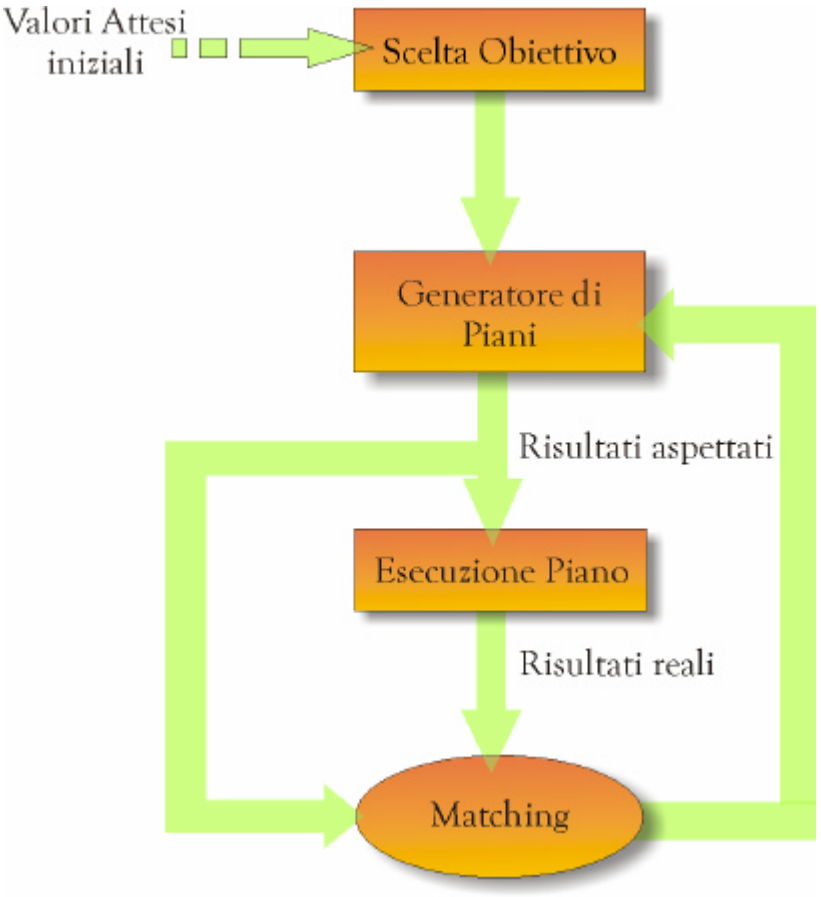


Figura 2-1 Modello dell'interazione tra conoscenza gestita tramite spazi concettuali e pianificazione

2.1. La percezione attraverso le tecniche di visione artificiale

Un robot oltre a poter ricevere dei dati sensoriali da sensori di prossimità, può anche sfruttare la percezione visiva utilizzando le telecamere. L'utilizzo delle telecamere porta a grandi problemi di interpretazione dei dati, ma con dei piccoli accorgimenti largamente usati si possono ottenere dei buoni risultati per quanto riguarda l'autolocalizzazione.

Un problema della visione robotica è quella di potersi localizzare in un ambiente. A causa delle grosse difficoltà di poter riconoscere degli oggetti in una scena, si ricorre all'uso di marker particolari che sono facilmente individuabili in un'immagine. Per poter risalire alla distanza dal marker è necessario avere una coppia stereo di immagini, ovvero delle immagini acquisite da telecamere poste ad una distanza conosciuta. Si preferisce porre le telecamere parallele fra di loro per semplificare il calcolo della distanza dal marker.

2.1.1. Correzione della distorsione e calibrazione

Una telecamera a causa della curvatura delle lenti fornisce un'immagine con diverse distorsioni; una molto diffusa è la distorsione radiale che crea il caratteristico effetto a botte. Questo effetto è molto deleterio durante la fase di triangolazione, infatti, sposta i pixel del punto da considerare in altre posizioni fornendo dei valori non corretti.

Un esempio di questo effetto si ha nella Figura 2-2.

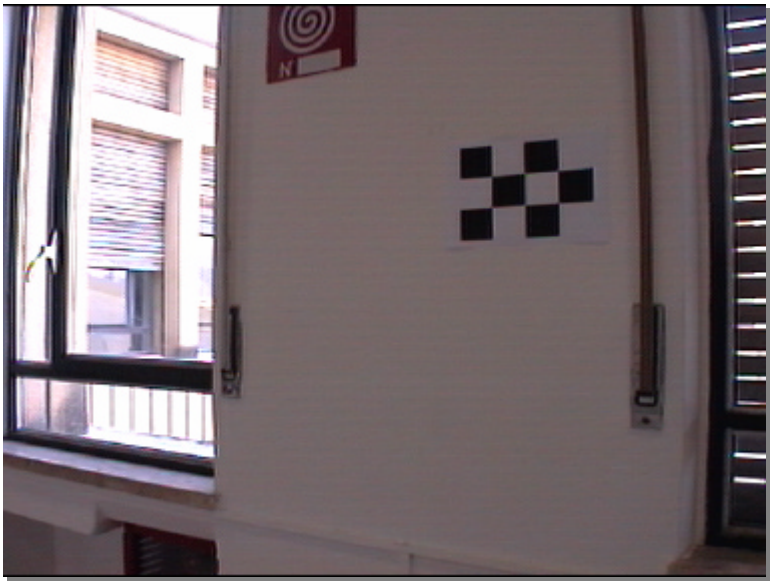


Figura 2-2 Esempio di distorsione dell'immagine

come si può notare le linee verticali sono caratterizzati da una distorsione che assomiglia alla forma di una botte. Per cercare di risolvere questo problema si cerca di minimizzare questo effetto utilizzando dei metodi legati ai parametri della telecamera. I parametri intrinseci ed estrinseci della telecamera stessa vengono calcolati tramite la calibrazione; tali parametri descrivono una particolare configurazione delle telecamere [16].

I parametri intrinseci che specificano le caratteristiche di una telecamera sono:

- Lunghezza focale, che misura la distanza fra le lenti della telecamera e il piano dell'immagine.
- Il punto centrale dell'immagine in pixel.
- Dimensione effettiva in pixel.
- Coefficiente di distorsione radiale delle lenti.

I parametri estrinseci della telecamera descrivono la relazione spaziale fra la telecamera e il mondo esterno e sono:

- Matrice di rotazione.
- Vettore di traslazione.

La relazione tra un punto 3D e la sua proiezione nell'immagine è data dalla formula:

$$m = A[R \quad t]M$$

dove A è la matrice intrinseca della telecamera: $A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$,

(c_x, c_y) rappresentano le coordinate del punto centrale dell'immagine;

(f_x, f_y) rappresentano le lunghezze focali lungo l'asse x e y;

R rappresenta la matrice di rotazione e *t* il vettore di traslazione, questi relazionano le coordinate del sistema mondo con quelle del sistema della telecamera.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \text{ e } t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}.$$

La distorsione radiale è caratterizzata da quattro coefficienti: *k1,k2,p1,p2*.

$$\hat{x} = x + x[k_1 r^2 + k_2 r^4] + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$\hat{y} = y + y[k_1 r^2 + k_2 r^4] + [2p_2 xy + p_2(r^2 + 2y^2)]$$

dove x, y sono le coordinate reali nell'immagine non distorta e $\hat{x}, \hat{y}$ sono le coordinate del punto nell'immagine distorta, mentre $r^2 = x^2 + y^2$.

Un algoritmo descritto da Z. Zhang [21] consente di trovare i parametri della telecamera e quindi di effettuare una calibrazione della stessa. L'algoritmo descritto in passi è il seguente:

1. Trovare un'omografia per tutti i punti su una serie di immagini, dove un'omografia è una matrice di trasformazione prospettica tra il piano di un pattern di calibrazione e il piano di vista della telecamera.
2. Inizializzare i parametri intrinseci e impostare la distorsione a 0.
3. Trovare i parametri estrinseci per ogni immagine del pattern.
4. Effettuare un'ottimizzazione globale minimizzando l'errore dei punti di proiezione con tutti i parametri.

Come pattern per la calibrazione si può usare una scacchiera con dei quadrati di dimensione conosciuta, come ad esempio quella di Figura 2-3:

Figura 2-3 Esempio di scacchiera per la calibrazione

Sperimentalmente si è visto che si ottengono dei risultati migliori usando una scacchiera che ha un numero pari di righe e dispari di colonne o viceversa, mentre con un numero pari di righe e di colonne possono ottenersi dei valori non ottimi. Per effettuare questa calibrazione si è fatto uso del pacchetto di librerie software OpenCV della Intel [8], creato appositamente per la visione robotica.

Le librerie forniscono un filtro utilizzabile con GraphEdit delle DirectX della Microsoft che consente di effettuare una calibrazione della telecamere semplicemente muovendo il pattern davanti alla telecamera stessa. Le stesse librerie OpenCV, forniscono una funzione che consente di minimizzare la distorsione radiale una volta che si conoscono i parametri della telecamera, infatti, in tempo reale si possono avere delle immagini con errori di distorsioni minimi. Considerando la Figura 2-2, si ottengono i seguenti risultati:

Figura 2-4 Immagine con distorsione corretta

Come si può notare nell'immagine l'errore radiale è quasi scomparso, consentendo di applicare l'algoritmo di triangolazione ottenendo dei risultati molto buoni. Dopo aver corretto l'immagine il prossimo passo sarà quello di ricercare il marker all'interno di un'immagine.

2.1.2. Ricerca del Marker

I marker usati per la localizzazione rappresentano una scacchiera di tre righe e quattro colonne, e la dimensione usata di un quadrato è di 6, 5 o 4 cm. Per rendere meno invasivo il marker si è usato un quadrato di quattro cm in modo di occupare 16 x 12 cm^2.

Facendo sempre uso delle librerie OpenCV [8], i marker vengono facilmente individuati in un'immagine tramite delle semplici funzioni che restituiscono l'elenco dei punti di intersezione dei quadrati. L'algoritmo usato per la ricerca del marker ha dato ottimi risultati anche in condizioni di luminosità molto bassa e con forti controluce, mentre in condizioni di basso contrasto non sempre si riusciva a trovare il marker. L'Api che si è utilizzato per l'individuazione del marker è la: `cvFindChessBoardCornerGuesses` [12] alla quale si passa l'immagine di partenza e il numero di punti del marker. L'algoritmo usato per la ricerca dei marker può essere riassunto nei seguenti passi:

1. si applica un filtro di Blur [17];
2. si dilata l'immagine;
3. si effettua una segmentazione adattiva;
4. a partire dal numero dei quadrati del marker, si costruisce un reticolo di punti che verrà mappato all'immagine stessa nelle intersezione dei quadrati;
5. usando la funzione `FindQuadNeibors` [12] si rilevano i quadrati vicini fra di loro nell'intera immagine;
6. dai quadrati connessi fra di loro ottenuti precedentemente si ricavano le coordinate x e y delle intersezioni;
7. si raggruppano le coordinate trovate in modo da sapere a quale quadrato appartengono;
8. si effettua un'operazione sui subpixel per ottenere una precisione sulle coordinate.

Durante la ricerca delle intersezioni ovviamente vengono effettuate dei controlli sul numero delle intersezioni stesse in modo che non si possono avere un numero maggiore o minore rispetto a quelle che ci aspettavamo.

In merito alla Figura 2-4, il risultato ottenuto è quello mostrato nella Figura 2-5. Si può notare come i quadrati vicini sono stati individuati senza molti problemi e i cerchietti indicano i punti di intersezione dei quadrati stessi.

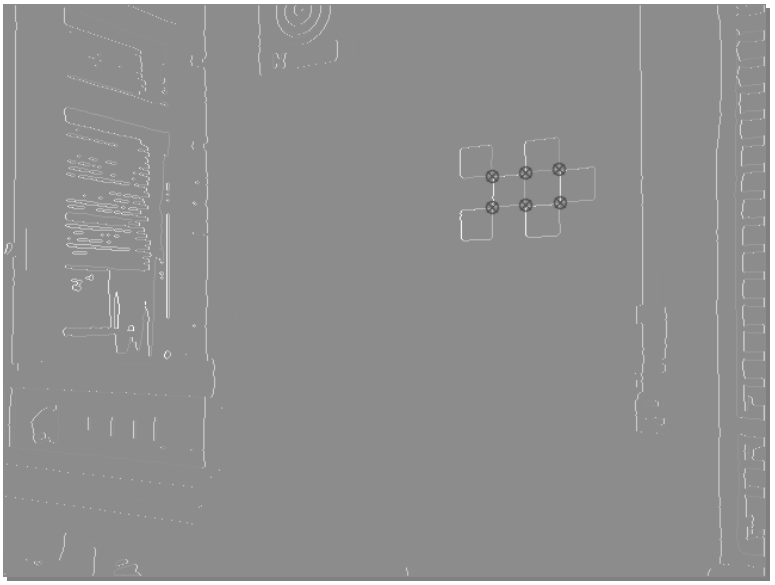


Figura 2-5 Punti di intersezione del marker trovati

Questo algoritmo di ricerca dei punti di intersezione viene anche usato per la calibrazione della telecamera, infatti, usando più immagini che contengono la stessa scacchiera è possibile risalire ai parametri della telecamera.

Conoscendo i punti di intersezione della scacchiera, si può usare la visione stereo per poter determinare la distanza di tali punti dalle telecamere.

2.1.3. Triangolazione

Una situazione semplificata, che tiene conto della sola vista dall'alto, di due telecamere poste parallele fra loro che inquadrano un punto della scena è del tipo:

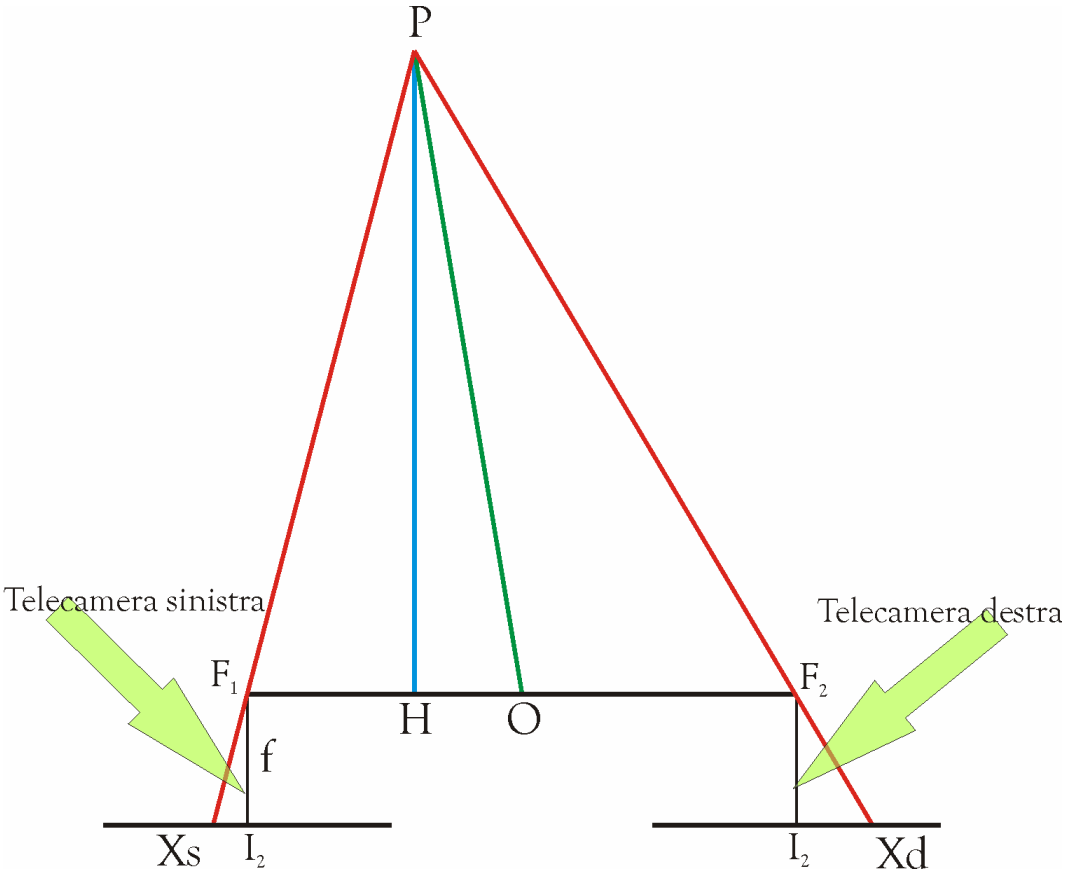


Figura 2-6 Triangolazione con due telecamere

In Figura 2-6 si può notare come il punto P viene visualizzato nelle due immagini destra e sinistra riprese dalle rispettive telecamere. La figura, per semplicità, non rispecchia il ribaltamento del raggio luminoso dovuto all'effetto ottico delle lenti, ma di tale effetto se ne tiene in considerazione nell'algoritmo implementato. Le due telecamere hanno una medesima focale f e sono poste ad una distanza fissa. Il punto P viene rappresentato nell'immagine destra dal pixel Xd e nell'immagine sinistra dal pixel Xs.

Consideriamo i due triangoli simili: $F_1 \hat{P} F_2$ e l'unione dei due triangoli $X_s \hat{F}_1 I_2$ ed $X_d \hat{F}_2 I_2$, i due triangoli sono simili perché hanno i tre angoli uguali. Scriviamo la relazione PH : f = F_1F_2 : ((Xs – I_2) – (Xd + I_2)).

In questo modo otteniamo l'altezza PH del triangolo $F_1 \hat{P} F_2$, in quanto f, F_1F_2 ed (X_s – I2) – (X_d + I2) sono delle quantità note. Con pochi passaggi si possono trovare i segmenti HO e PO; quest'ultimo rappresenta la distanza reale dal centro delle telecamere al punto P. Chiamando a il coefficiente che lega i pixel dell'immagine ai millimetri dell'area sensoriale della telecamera, la descrizione dell'algoritmo di triangolazione è la seguente:

$$I2s = (x1 - CentroXs) \cdot a$$

$$I2d = (x2 - CentroXd) \cdot a$$

$$PH = \frac{f \cdot F1F2}{I2s - I2d}$$

$$HO = \frac{I2s \cdot F1F2}{I2d - I2s} + \frac{F1F2}{2}$$

$$PO = \sqrt{HO^2 + PH^2}$$

$$alphax = p - \arctan\left(\frac{PH}{HO}\right)$$

Purtroppo la sola conoscenza della distanza dal marker, mediata su più punti del marker stesso, non è sufficiente a fornire le coordinate assolute per potersi autolocalizzare, quindi è necessario avere un altro punto di riferimento per ottenere delle coordinate spaziali.

2.1.4. Autolocalizzazione del robot basata sulla visione

Dopo aver calcolato con una buona approssimazione la distanza che c'è fra il pantilt del robot e il marker, è necessario ritornare al software che si occupa della navigazione del robot stesso, le nuove coordinate che contengono meno errori rispetto a quelle che l'odometria del robot stesso fornisce. Una situazione tipica è la seguente:

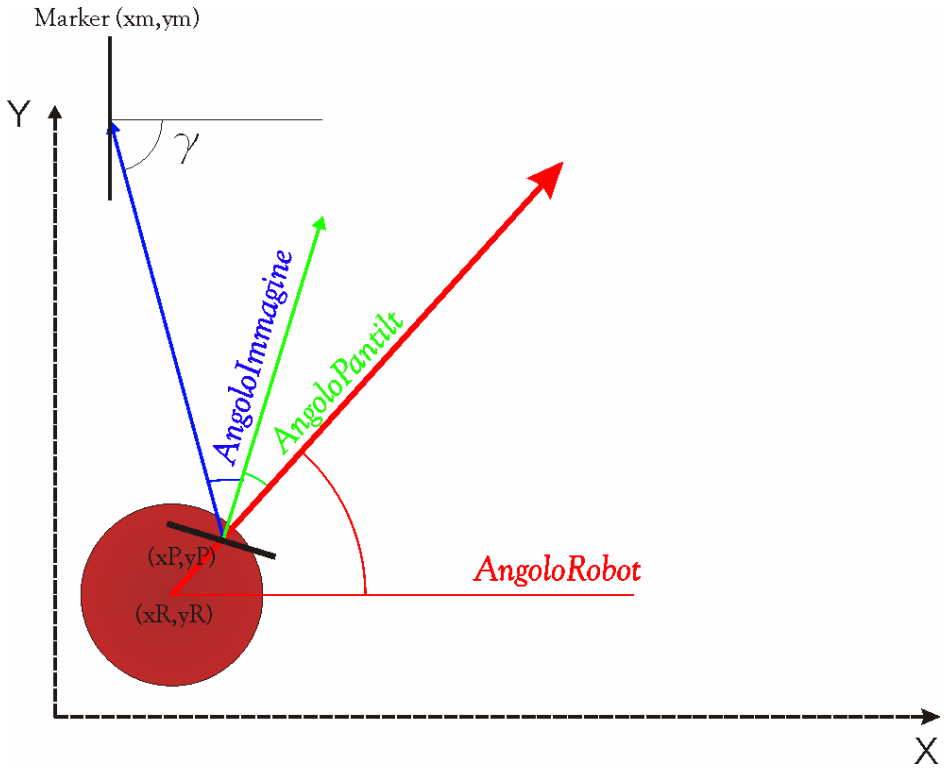


Figura 2-7 Calcolo coordinate spaziali del Robot

Il calcolo delle coordinate si basa a partire dall'angolo del Robot nel sistema mondo. Uno degli aspetti negativi di questo metodo è che se l'angolo del Robot è errato, la localizzazione fornisce dei risultati con degli errori. La descrizione dell'algoritmo è la seguente:

1) Si individua il marker più vicino, mediante la distanza euclidea, e si memorizza le sue coordinate spaziali x, y. In base a queste coordinate e all'angolo che il software di navigazione fornisce, si calcola di quanto deve girare il pantilt per consentire alle telecamere di trovare il marker al centro dell'immagine. Se l'angolo di rotazione del pantilt è maggiore di quanto il

pantilt stesso può ruotare in senso orario o in senso antiorario, l'algoritmo ritorna gli stessi valori forniti dal software di navigazione, perché fisicamente non è si è in grado di fare una tale rotazione.

2) Si cattura un'immagine dalla telecamera sinistra e dopo aver minimizzato gli errori di distorsione radiale si cerca il marker. L'operazione di cattura dell'immagine viene eseguita fino a tre volte sulla stessa telecamera, affinché si elimini l'occlusione temporale del marker da parte di un visitatore o altro. Lo stesso procedimento di acquisizione viene effettuato con la telecamera destra. Se in una delle due immagini non si riesce a trovare il marker allora l'algoritmo ritorna gli stessi valori forniti dal software di navigazione.

3) Si effettua la triangolazione su tutti e sei i punti del marker e si fa una media della distanza e dell'angolo che F2OP della Figura 2-6. Se la distanza è minore di zero o maggiore di quello consentito, significa che la triangolazione non ha avuto successo, in quanto o si è applicato l'algoritmo sulle immagini invertite, o la differenza di posizione fra i pixel dell'immagine destra e sinistra è troppo piccola per poter fornire dei risultati attendibili.

4) Si calcola l'angolo g che uguale a 180 gradi meno la somma dei tre angoli: *AngoloRobot*, *AngoloPanTilt* e *AngoloImmagine*. Ricordando che *AngoloPantilt* è l'angolo di rotazione del PanTilt e *AngoloImmagine* è l'angolo che forma il marker con il centro dell'immagine.
$g = p - (AngoloRobot + AngoloPanTilt + AngoloImmagine)$

5) Conoscendo le coordinate del marker, l'angolo g e la distanza fra il PanTilt e il marker, possiamo calcolare le coordinate (xP, yP) del PanTilt:
$xP = \cos(g) \cdot Distanza + xm; \quad yP = \cos(g) \cdot Distanza + ym$.

6) Adesso si possono calcolare le coordinate del robot sapendo che la distanza fra centro robot e pantilt è di 170 mm, otteniamo:
$xR = xP - 170 \cdot \cos(AngoloRobot); \quad yR = yP - 170 \cdot \sin(AngoloRobot)$.

2.1.5. Tecniche per il miglioramento dell'autolocalizzazione

Il metodo di autolocalizzazione presenta un inconveniente, ovvero l'essere legato all'angolo che il robot ha nel momento della localizzazione. Per ovviare a questo problema si sono cercate delle soluzioni alternative per fare a meno dell'angolo del robot.

Correzione della posizione assoluta del robot.

Una possibile soluzione è quella di far riferimento su più punti della scena in modo da ricavare una posizione assoluta a partire dalle due distanze dai punti. Dato che si hanno molte variabili nel calcolo della distanza da un marker, non si può avere una localizzazione spaziale corretta se ci si basa soltanto della distanza di due punti vicini fra loro nello spazio. Consideriamo la figura seguente:

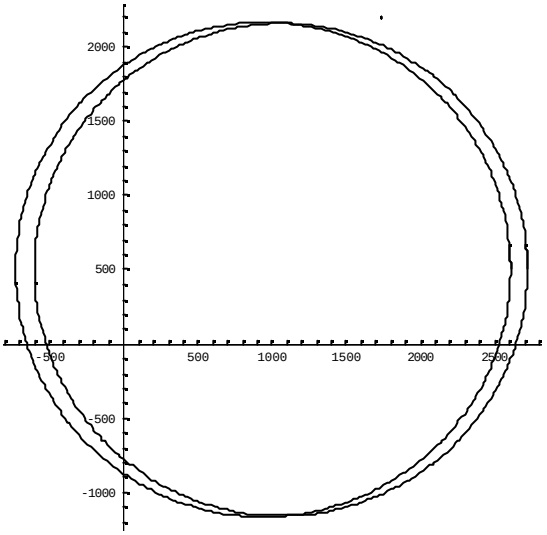


Figura 2-8 Intersezione di due circonferenze usate nell'algoritmo di localizzazione

Ipotizzando che i centri delle due circonferenze siano dei punti o dei marker vicini e conoscendone la distanza dal robot, potremmo dire che una delle due intersezioni delle due circonferenze è la posizione spaziale del robot stesso. Purtroppo questo metodo richiede delle misure molto precise della distanza tra il robot e il punto o marker che si sta considerando. Facendo delle prove si è visto che si ottengono facilmente degli errori di anche 50 cm dalla posizione reale del robot stesso, infatti, anche delle differenze di pochi centimetri su una misura della distanza, causano dei grossi errori.

Per avere una buona localizzazione spaziale, bisogna tenere presente anche dell'angolazione che ha il robot rispetto al marker. Un'altra soluzione per avere una localizzazione che non tiene conto dell'angolo del robot è quella di usare due marker posti fra di loro ad una distanza tale da avere dei risultati accettabili. Il grosso problema di utilizzare due marker posti distanti fra di loro è quella di effettuare una doppia localizzazione ogni volta che ce ne è bisogno e quindi un tempo almeno doppio rispetto a quello necessario per una semplice localizzazione. Considerando il robot in un ambito operativo come quello di guida museale o come metronotte, la doppia localizzazione è un grosso limite, infatti, fa sprecare molto tempo al robot nella sua navigazione nell'ambiente.

Correzione dell'angolo tra il robot e il marker

Una soluzione per ripristinare l'angolo del robot è stata quella di posizionarsi davanti ad un marker e con piccoli movimenti di rotazione in senso orario o antiorario, il robot si mette perpendicolare al marker stesso. Il procedimento di allineamento viene eseguito dal robot dopo che ha finito di effettuare una spiegazione e ritorna al punto di partenza, in modo da annullare gli errori sull'angolo. Dopo aver posizionato il marker in un punto noto e con un angolo, che forma il marker con il sistema di riferimento assoluto, anch'esso conosciuto, l'algoritmo cerca di far allineare i tre punti superiori (o anche inferiori) creati dall'intersezione dei quadrati bianchi e neri. Una condizione necessaria è che il marker sia perfettamente perpendicolare al pavimento.

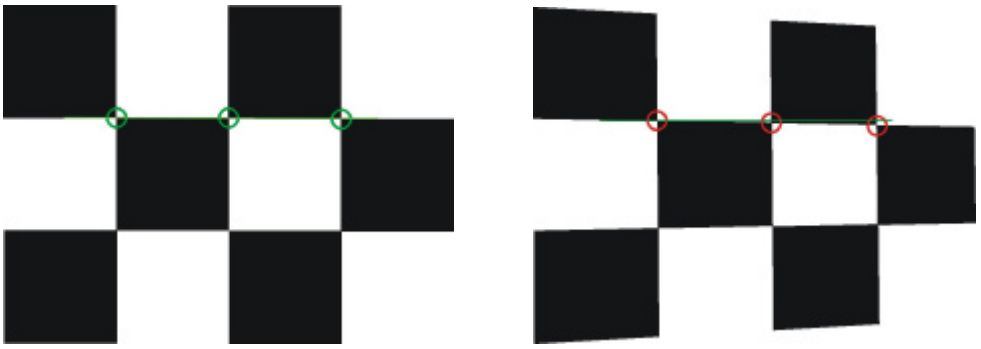


Figura 2-9 Marker visto frontalmente e visto di sbiego

Come si può vedere in Figura 2-9, nel caso in cui il robot si trovi perpendicolare al marker i punti superiori sono allineati nell'immagine acquisita, mentre nel secondo caso non lo sono ed è necessario far ruotare il robot in senso antiorario per posizionarsi perpendicolarmente. A causa delle variazioni di luminosità e dall'algoritmo di ricerca di marker che fa uso di una soglia adattiva, i tre punti si possono considerare allineati quando la loro differenza in pixel lungo l'asse y è al di sotto di un certo delta. Per velocizzare l'allineamento si fornisce al robot un comando di rotazione proporzionale alla differenza in pixel fra il primo e il terzo punto di intersezione. L'algoritmo proposto fornisce valori con precisione subpixel dato che esso effettua un'interpolazione ricorsiva sui punti di intersezione.

3. RAPPRESENTAZIONE INTERNA MEDIANTE SIMULATORE 3D

Per avere una rappresentazione interna delle conoscenze del robot, si può far uso di un simulatore 3D che permette di rappresentare il mondo esterno al robot stesso, e ne permette una visualizzazione completa all'utilizzatore del robot stesso. L'approccio del simulatore 3D facilita anche la programmazione di un robot, perché consente di testare dei comportamenti da far fare al robot senza utilizzarlo, in questo modo diminuiscono i tempi di prova e si utilizza l'hardware del robot in maniera minore.

3.1. Simulatore usato

La simulazione tridimensionale è stata fatta sia con il motore grafico di Quake 2, dopo aver apportato delle modifiche al codice, che con il motore grafico Irrlicht nella versione 0.5 o successive. Il vantaggio di usare il motore grafico open source Irrlincht [9] è quello di avere un controllo quasi totale sul codice prodotto, inoltre, l'SDK viene fornito con una buona documentazione delle funzioni incluse. Quake 2 ha il vantaggio che è stato fatto un adattamento del codice per palmare. Pur avendo il codice fornito dalla casa produttrice, non si ha una documentazione dettagliata, quindi, bisogna scoprire le funzioni o la logica interna del codice stesso.

Un grandissimo vantaggio nell'usare il motore grafico Irrlicht è quello di poter creare un simulatore 3D con pochissime righe di codice, infatti, dopo aver creato una mappa con dei programmi come Quark [15] o GtkRadiant [14] si crea un oggetto albero a cui si connettono le telecamere, la mappa e altri oggetti. Le mappe create sono dello stesso tipo di quelle che sono utilizzate per il gioco Quake 3, infatti, hanno anche una rappresentazione migliore rispetto a quelle usate per il simulatore Quake 2. Una gran limitazione delle mappe per il gioco di Quake 2 è quella di utilizzare delle tessiture a bassa risoluzione e con pochi colori, infatti, tutto il gioco fa riferimento ad una sola tavolozza di colori che viene usata per tutte le tessiture. Utilizzando il motore grafico Irrlicht si evita sia il problema della tavola

dei colori comune che la limitazione di usare delle tessiture a bassa risoluzione, consentendo di avere una rappresentazione dell'ambiente più fotorealistica.

Figura 3-1 Differenze fra i due motori 3D utilizzati

Anche se nelle due foto le differenze non si possono apprezzare bene, si può notare che il motore Irrlicht ha delle tessiture più dettagliate, inoltre, la gamma cromatica è più vasta rispetto a quella di Quake 2. La differenza della gamma cromatica è dovuta al fatto che Quake 2 utilizza, come detto in precedenza, una sola tavola di colori per tutte le texture, inoltre non è possibile utilizzare delle tessiture ad alta risoluzione; questi limiti sono superati dall'altro motore grafico Irrlicht.

3.2. Introduzione dei dati dei sensori laser nella rappresentazione interna della scena

I robot in genere sono dotati di sensori laser che consentono di avere delle letture sulla distanza dal punto di impatto della luce laser con un oggetto dal robot. Queste informazioni sono molto utili perché consentono di poter conoscere degli ostacoli che si trovano davanti al robot stesso. Il laser fornisce una lettura molto precisa sulla distanza dell'ostacolo, ma ha un grosso inconveniente, infatti, non funziona con superfici perfettamente trasparenti come il vetro. La quasi assenza di informazioni da una superficie trasparente porta grossi problemi soprattutto quando si è in ambienti con molte vetrine, come ad esempio in un museo. Un grosso vantaggio del laser è quello di non richiedere alcuna successiva elaborazione, oltre a quella necessaria per la lettura dall'hardware, per interpretare i dati come accade nella visione.

I valori forniti dal laser possono essere interpretati come degli ostacoli nell'ambiente tridimensionale interno del robot e quindi inserire nella sua rappresentazione interna dei nuovi elementi in base al contesto. I dati letti dall'ambiente esterno possono essere confrontati con quelli del simulatore, in modo da poter estrarre delle informazioni dei soli oggetti che non fanno parte della struttura dell'ambiente. Una situazione tipica è quando il robot si trova vicino ad un muro alla sua destra e dai sensori laser individua degli ostacoli alla propria destra, con l'aiuto del simulatore, il robot interpreta questi dati come degli oggetti facenti parte della struttura statica dell'ambiente in cui si trova e quindi facendo un matching si possono individuare strutture più fini che non fanno parte della conoscenza a priori dell'ambiente.

Ovviamente può sorgere un problema di non allineamento fra i dati reali e quelli ricostruiti. Una possibile soluzione è quella di ricercare la sovrapposizione che minimizzi la distanza fra le letture reali e ricreate. Per esempio nell'implementazione che sarà illustrata in seguito, lo spazio di ricerca prevede delle rotazioni di 15 gradi in senso orario e in senso antiorario dello spazio semicircolare che si riferisce alle misure. Lo scopo è quello di correggere l'errore sull'angolo di

rotazione del robot. Un'ulteriore modifica dello spazio di ricerca prevede anche la ricerca del fattore di scala ottimale. Per esempio, variando la scala delle letture di prossimità si può correggere l'errore di traslazione del robot posto più o meno vicino al muro.

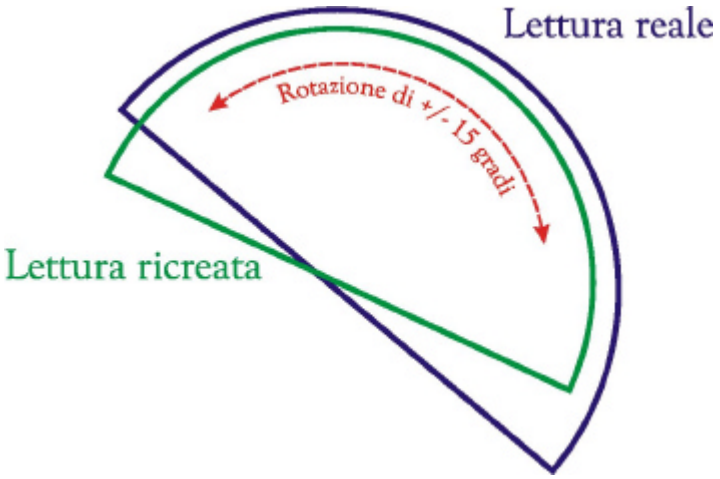


Figura 3-2 Sfasamento delle letture laser reali e virtuali

3.3. Matching basato su marker

Inserendo nel motore tridimensionale interno dei marker proprio dove sono stati posti nella realtà, è possibile fare un confronto fra la posizione del marker nell'immagine reale e nell'immagine simulata, in modo da poter fornire un ulteriore aiuto all'autolocalizzazione. Uno dei grandi problemi che si ha nell'utilizzo di tale tecnica è quello di tenere allineata la posizione del robot con quella del simulatore tridimensionale. Questa tecnica di confronto è utilizzata su una sola telecamera, quindi non è possibile fare delle considerazioni sulla distanza come è stato fatto precedentemente con la visione stereo, ma è possibile fare delle considerazioni sui sei punti che sono presenti nell'intersezioni tra quadrati neri del marker.

Uno sviluppo futuro potrebbe essere quello di effettuare un riconoscimento di oggetti che non siano dei semplici marker, ma estendere la ricerca ad altre entità che per caratteristiche intrinseche possono essere riconosciute in una scena.

4. SPERIMENTAZIONI

Tutte le sperimentazioni sono state fatte su un robot B21r della Real World Interface; tale robot è composto da due parti strutturali: *base* ed *enclosure*. Nella Figura 4-1 è possibile notare tali parti strutturali.

Figura 4-1 Robot RWI B21r

Nella base sono presenti i motori, ruote e batterie, ovvero tutto l'apparato che ne consente il movimento, inoltre è presente un laser che consente l'individuazione di ostacoli vicini e una fascia di sensori sonar. La parte superiore chiamata *enclosure* è composta da diversi sensori, come i sensori ad infrarossi, sonar e sensori di contatto. Nell'estremo superiore è presente un'unità pantilt che consente di far ruotare due telecamere poste su tale unità; inoltre, è presente anche una bussola che permette di individuare l'angolo della direzione di marcia del robot.

I sonar e i sensori ad infrarosso ricoprono il robot su due fasce orizzontali, in modo da avere delle informazioni da tutto l'ambiente circostante. I sensori ad infrarosso ricoprono la parte superiore o *enclosure*, mentre i sensori sonar sono presenti sia nella parte superiore che in quella inferiore. Un sensore sonar riesce a leggere un cono di 15° circa ed in genere sono usati per individuare ostacoli distanti; i sensori ad infrarosso, invece, vengono usati per individuare ostacoli vicini. I sensori di contatto non sono altro che degli interruttori posti nei pannelli inferiori e superiori. In tutte le sperimentazioni non si sono potuti sfruttare

nessuno dei 3 tipi citati di sensori, infatti, quelli di contatto non funzionavano, i sonar fornivano delle letture sparse e non costanti nel tempo, mentre gli infrarossi a causa di difficoltà tecniche non sono stati usati.

I sensori laser consentono di avere una lettura della distanza degli ostacoli dei 180° frontali, ma questi forniscono un'informazione dettagliata degli ostacoli posti davanti al robot. Un problema dei sensori laser è che forniscono una lettura ogni 9 decimi circa, facendo sì che il robot non possa andare veloce durante il suo moto perché resterebbe "cieco" per molti cm di percorrenza. Un altro problema dei sensori laser è che la luce emessa non viene riflessa dalle superfici trasparenti, come ad esempio i vetri; questo comporta molti problemi in ambienti in cui sono presenti molte vetrine, come ad esempio nel museo regionale di Agrigento.

Il robot è dotato di quattro motori, due sono adibiti al movimento traslatorio del robot stesso, mentre gli altri due si occupano al movimento rotatorio.

La bussola si trova nella parte superiore del robot e come detto precedentemente indica la direzione di marcia del robot. Per ragioni sconosciute questo apparato non forniva dei valori corretti, infatti, aveva delle variazioni sull'angolo reale di 15 gradi, facendo sì che il suo utilizzo era poco utile. Dato che, durante gli esperimenti fatti, nella parte superiore del robot vi era collocato un notebook, i valori forniti dalla bussola erano maggiormente corrotti a causa dei campi magnetici creati dal notebook stesso.

Gli strumenti necessari alla localizzazione sono le telecamere e l'unità pantilt, infatti con le prime è possibile acquisire le immagini dall'ambiente circostante, e con il pantilt è possibile rotare le telecamere rispetto al robot stesso, oltre che poterne alzare o abbassare la visuale.

4.1. Coppia stereo

In questi paragrafi verrà descritto l'hardware del robot relativo alla visione, in particolare le telecamere usate e il supporto che consente di farle ruotare.

4.1.1. Sensori di acquisizione immagini del robot

Le telecamere usate nel robot sono le Sony XC–999, le quali consentono di avere una buona risoluzione e per le loro dimensioni e peso possono essere usate senza problemi su un'unità pan-tilt. Una telecamera tipicamente è composta da due parti: un sistema ottico che acquisisce le intensità di luce e tramite una particolare lente le convoglia in un'area che cattura tali intensità di luce; l'altro sistema converte queste intensità di luce in segnali che possono essere facilmente utilizzate da un computer.

Figura 4-2 Telecamera Sony XC–999

La risoluzione massima delle telecamere è di 768 x 494 pixel in NTSC (National Television System Committee), ovvero il formato video standard usato negli Stati Uniti e nel Giappone. Queste telecamere utilizzano delle lenti con una focale di 6 mm e sono collegate ad un'unità Horita BSG–50 che le alimenta. L'uscita è di tipo S–VCD ed è quindi facilmente utilizzabile da diverse apparecchiature. A causa della configurazione del sistema, la parte di visione viene completamente gestita dal portatile posto sopra il robot, quindi, non potendo usare dei dispositivi PCI è stato necessario usare dei grabber USB 2 per potere effettuare l'acquisizione video. La risoluzione usata per la localizzazione è di 640 x 480 in modo da non appesantire eccessivamente il portatile.

I valori dei parametri di una telecamera sono:

$$A = \begin{bmatrix} 659.697 & 0.000 & 342.978 \\ 0.000 & 597.521 & 235.512 \\ 0.000 & 0.000 & 1.000 \end{bmatrix}$$

$$R = \begin{bmatrix} 0.977566 & -0.059572 & -0.202029 \\ -0.030793 & -0.989286 & 0.142709 \\ -0.208365 & -0.133287 & -0.968926 \end{bmatrix};$$

$$t = \begin{bmatrix} -16.280697 \\ 5.654477 \\ 81.432854 \end{bmatrix}$$

$$D = \begin{bmatrix} -0.298529 \\ 0.24298975 \\ -0.00347875 \\ 0.0133665 \end{bmatrix}$$

I risultati dell'algoritmo per attenuare la distorsione applicando questi valori sono visibili nella Figura 2-4.

4.1.2. Unità Pantilt

Il pantilt è un supporto motorizzato che serve a sostenere le telecamere. L'unità è composta da due motori che consentono di far ruotare le telecamere orizzontalmente e verticalmente. Il modello usato è il PT-46-17.5, esso viene pilotato tramite seriale RS–232 / RS–485 e consente di far ruotare le telecamere orizzontalmente di 154.20° in senso orario e in senso antiorario, mentre può ruotare in altro di 30.84° e in basso di 46.26°. Nella foto seguente si può vedere il pantilt con le due telecamere Sony XC–999 montate.

Figura 4-3 Unità Pan–Tilt del robot B21r

I comandi vengono inviati direttamente alla seriale tramite un server che funge da driver ed è comandabile da remoto, nell'appendice vengono descritti in dettaglio i comandi da inviare a tale server per far muovere il pantilt. Per evitare che il pantilt possa essere comandato da più utenti, il server accetta una sola connessione per volta.

4.2. Dati relativi alla triangolazione

Riprendendo l'immagine usata precedentemente per rappresentare la triangolazione, ne elenchiamo i valori ottenuti dalla calibrazione e dai dati strutturali del robot.

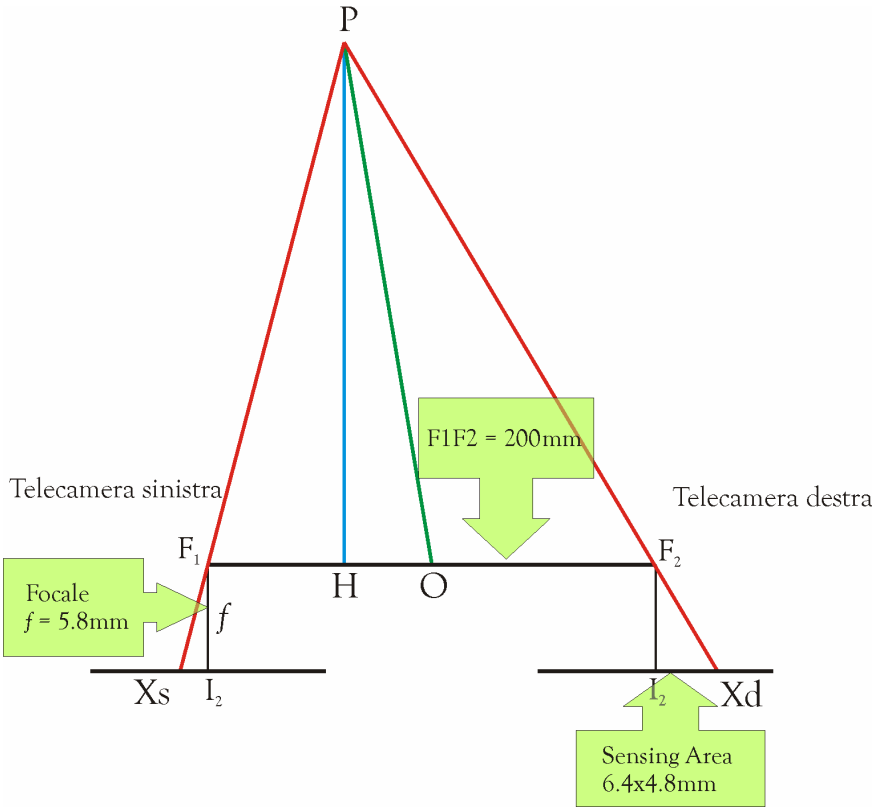


Figura 4-4 Dati per la triangolazione con due telecamere

La focale *f* ottenuta dalla calibrazione delle due telecamere è 5.8 mm, che si avvicina ai 6 mm forniti dalla casa produttrice, ma si è visto che si ottengono dei risultati migliori con un valore della *f* pari a 5.

Le due telecamere sono poste ad una distanza di 200 mm, ovvero F1F2 = 200.

Le telecamere in uso hanno una sensing area di 6.4 mm per 4.8 mm. Le acquisizioni vengono fatte con una risoluzione di 640 x 480 pixel, quindi hanno una densità di 100 pixel / mm. Il coefficiente a che lega i pixel ai millimetri è 0.01.

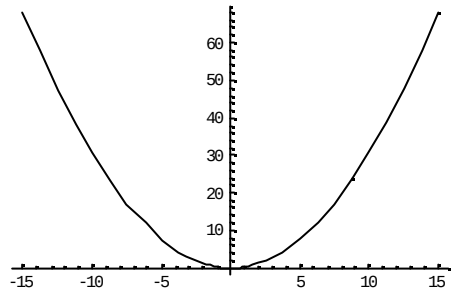
4.3. Analisi dell'errore di autolocalizzazione

Di seguito sono mostrati gli errori che si hanno lungo l'asse x e lungo l'asse y, facendo variare gamma con una variazione di più o meno 15 gradi.

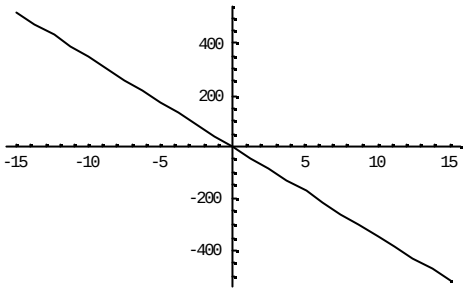
Con una distanza di 2000 mm si ha:

gamma = 0 gradi

x

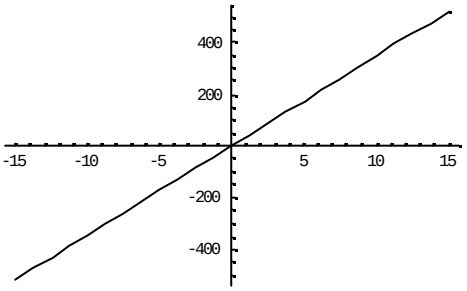


y

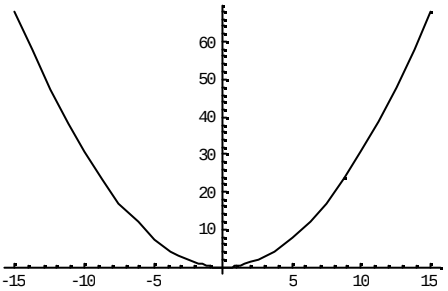


gamma = 90 gradi

x

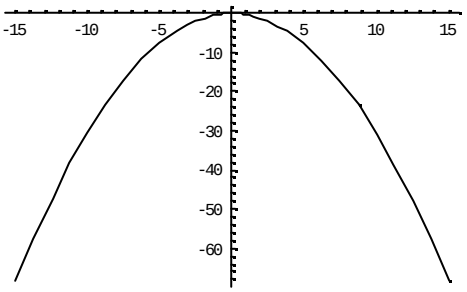


y

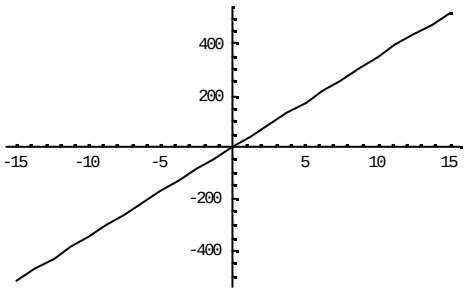


gamma = 180 gradi

x

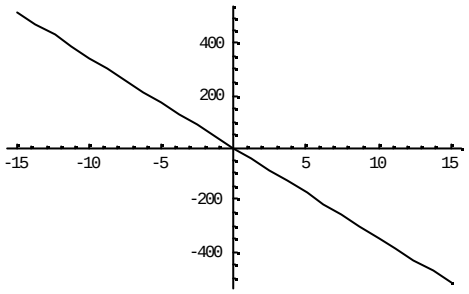


y

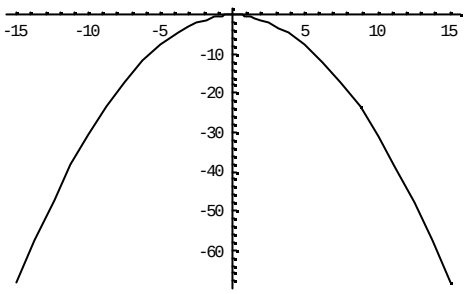


gamma = 270 gradi

x



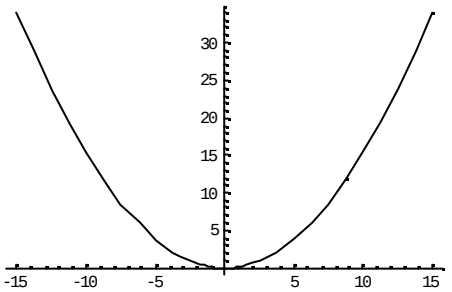
y



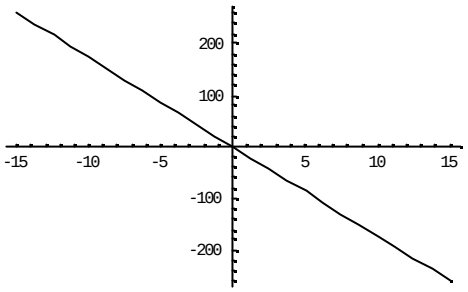
Con una distanza di 1000 mm si ha:

gamma = 0 gradi

x

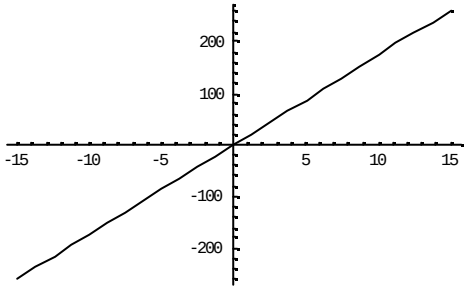


y

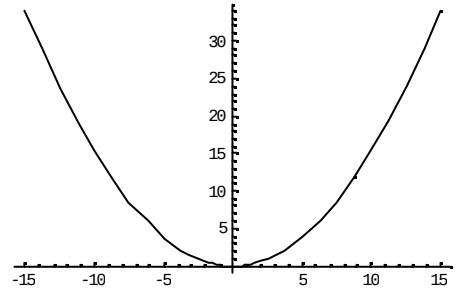


gamma = 90 gradi

x

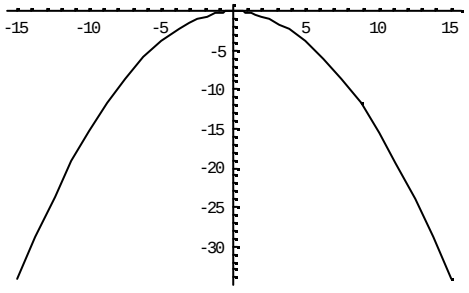


y

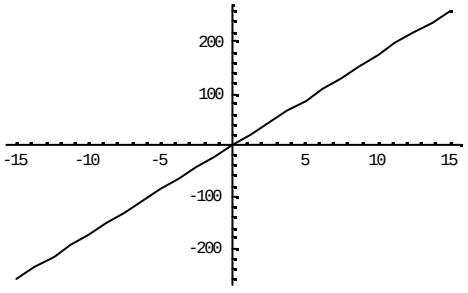


gamma = 180 gradi

x

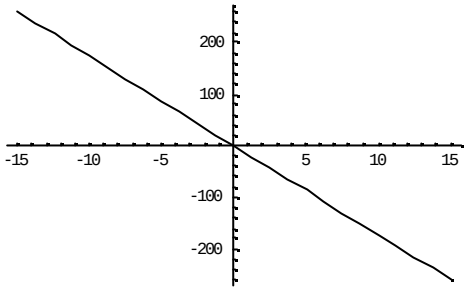


y

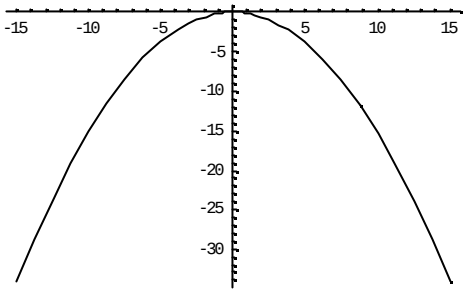


gamma = 270 gradi

x



y



Come si può notare si ha che a 0 e a 180 gradi si ha un grosso errore su l'asse y e un piccolo errore su l'asse x per piccole variazioni dell'angolo gamma, mentre a 90 e a 270 gradi si ha un grosso errore lungo l'asse x e piccolo lungo l'asse y. L'errore è direttamente proporzionale alla distanza che c'è fra il marker e il pantilt, quindi più lontano è il robot, più l'errore sarà maggiore se l'angolo di partenza del robot stesso non è corretto. Considerando una distanza di 2000 mm e un errore dell'angolo del robot di 5 gradi, si ha un errore di circa 200 mm lungo l'asse x o y a secondo di come è messo il robot stesso.

4.4. Scenario 1: Dipartimento di Ingegneria Informatica - "DINFO"

Per testare il software è stato effettuato un esperimento con un robot dotato di telecamere posizionato dentro il laboratorio del DINFO. In particolare nel laboratorio di robotica e nel corridoio principale.

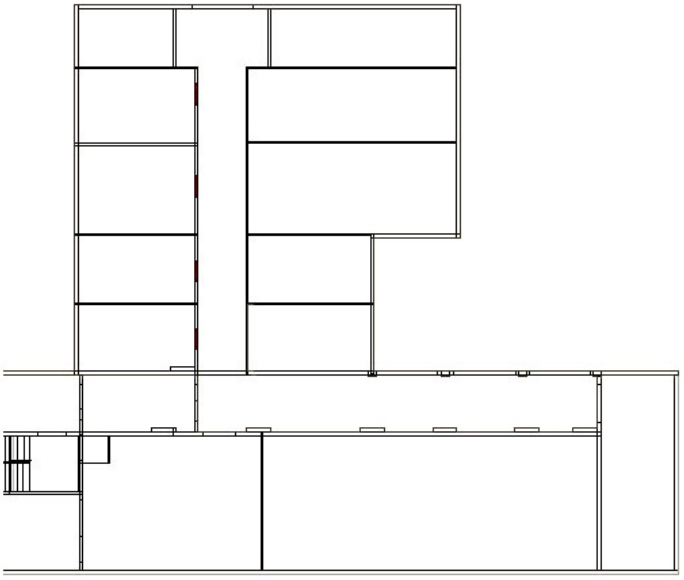


Figura 4-5 Porzione della mappa del DINFO

In questo esperimento si è cercato di far sovrapporre l'immagine acquisita dal robot con quella creata internamente con il simulatore 3D.

Dopo aver preso la posizione del robot nella stanza si è posizionato il robot virtuale dell'ambiente ricostruito in maniera da rispettare le posizioni reali e la grandezza focale dell'obiettivo. Fatto questo, con le barre di spostamento del programma si fa in modo da far coincidere le due immagini sovrapposte, nelle figure seguenti si possono vedere i risultati con due diversi valori di trasparenza:

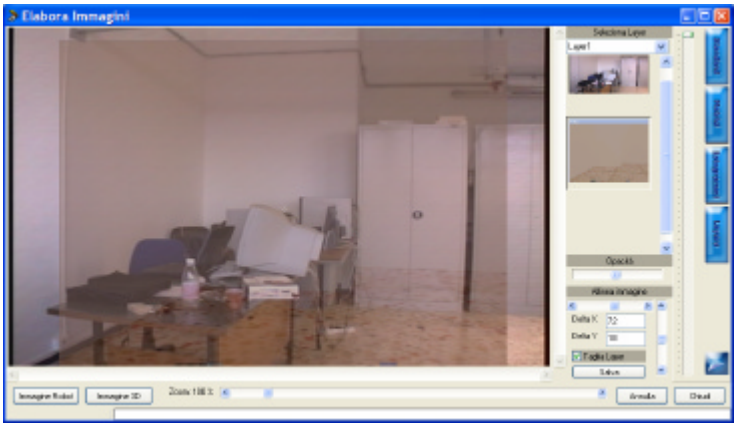


Figura 4-6 Esempio di sovrapposizione fra immagine reale e simulata

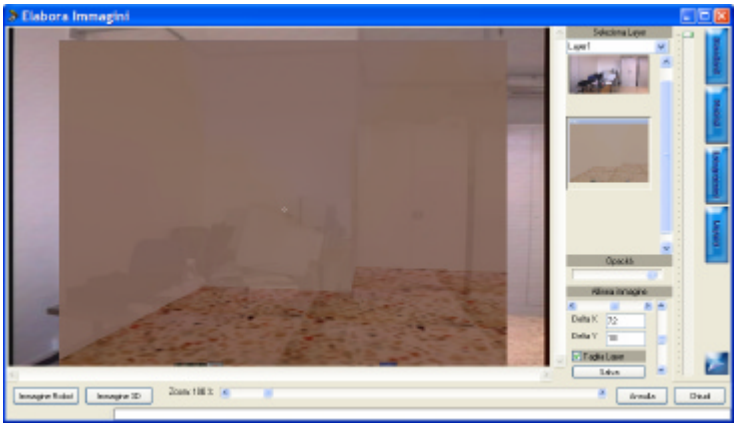


Figura 4-7 Esempio di sovrapposizione fra immagine reale e simulata con trasparenza diversa

Come si può notare il risultato è accettabile, inoltre dopo aver fatto coincidere le due immagini, non è più necessario effettuare questa operazione perché acquisendo delle nuove immagini, il programma mantiene la posizione della sovrapposizione fissa.

Un aspetto interessante di questo sistema è la possibilità di poter riconoscere dei particolari oggetti presenti nella scena. Gli oggetti devono essere specificati nello scenario tridimensionale che viene visualizzato dal motore 3D. Ad esempio in una stanza si è etichettata una porta in modo che durante la visualizzazione può essere riconosciuta dal sistema. Le immagini della telecamera del robot e della ricostruzione tridimensionale, sono mostrate nella Figura 4-8 e Figura 4-9.

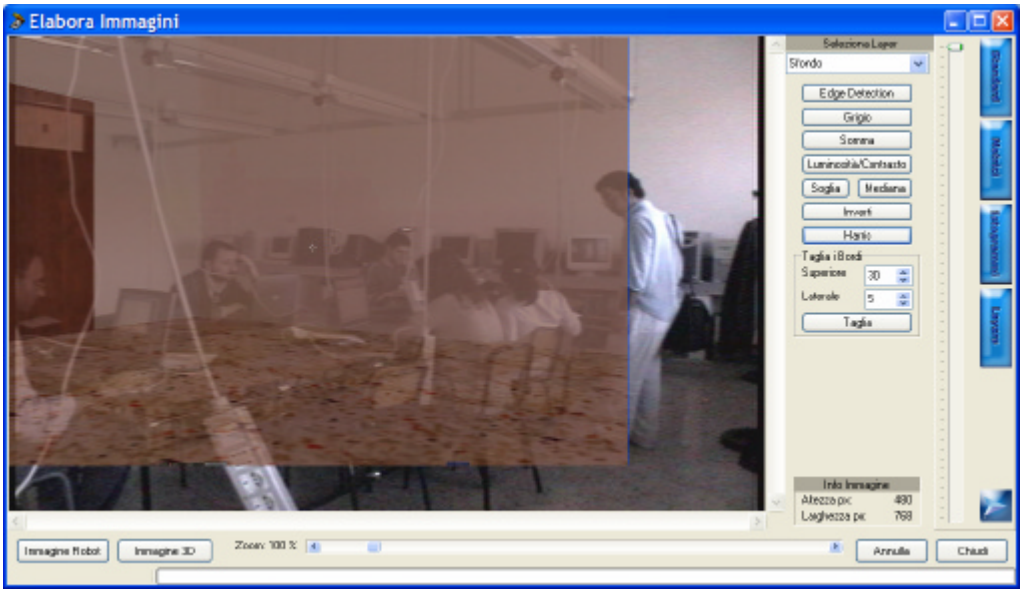


Figura 4-8 Sovrapposizione fra immagine reale e sintetica per il riconoscimento di un'entità nell'ambiente.

Nella finestra della mappa si ha una visione dall'alto che mostra il robot con la sua visualizzazione (in verde) dentro la quale ricade la porta della stanza; nella finestra delle entità vengono elencati gli oggetti riconosciuti che in questo caso è la porta numero uno.

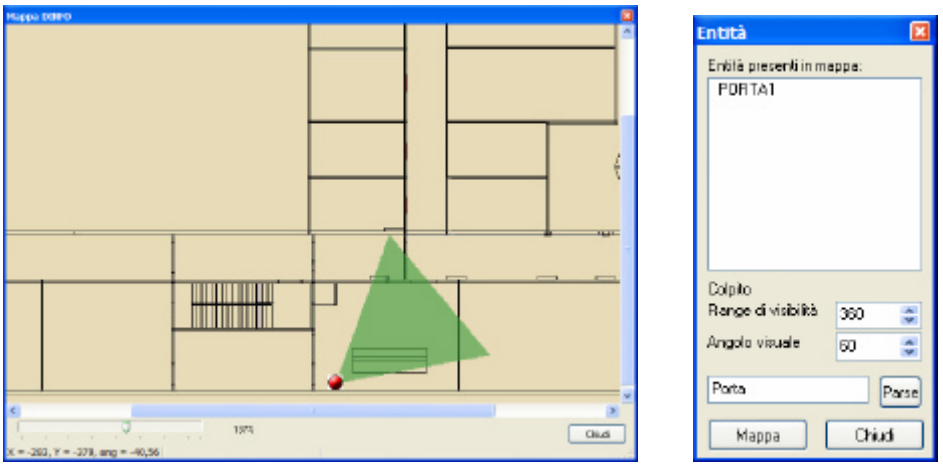


Figura 4-9 Riconoscimento di una porta nell'ambiente simulato

Questo risultato può essere inviato ad un motore referenziale, come ad esempio cyc, che potrebbe essere integrato nel sistema in uso.

4.5. Scenario 2: Museo Archeologico di Agrigento

Una sperimentazione di robot museale [20] è stata effettuata presso il Museo Archeologico Regionale di Agrigento, dove il robot percorreva il corridoio di una sala e nel frattempo descriveva le varie vetrine, spiegandone i relativi oggetti.

Un utente poteva scegliere un percorso da effettuare, premendo l'apposito pulsante sull'interfaccia, e il robot iniziava la visita scegliendo le vetrine adeguate, mentre l'utente seguiva il robot ascoltandone le spiegazioni. In alcune vetrine il robot si fermava, si girava e inquadrava nel suo schermo l'oggetto che stava descrivendo. Se il robot trovava una vetrina occupata, allora esso proseguiva il suo percorso per poi ritornare a spiegare la vetrina successivamente. Nel caso in cui il robot trovava un ostacolo davanti al suo percorso, esso suonava un clacson, per invitare le persone che si trovano davanti ad esso a spostarsi. Purtroppo a causa delle condizioni ambientali non favorevoli si è dovuto abolire il comportamento di aggirare gli ostacoli, in quanto molti oggetti come vetrine e ringhiere di vetro non sono riconoscibili dal robot. Il non riconoscimento degli ostacoli, come vetrine e ringhiere, potrebbe essere superato avendo una buona conoscenza della sua posizione nell'ambiente, ma a causa dei grossi errori di odometria e la mancanza di una bussola, il robot dopo aver percorso alcuni metri dalla localizzazione visuale non ha una grande precisione sulla conoscenza della sua posizione spaziale. Il problema potrebbe essere risolto in parte sfruttando degli algoritmi che utilizzano i dati sensoriali dei laser e/o sonar, ma al momento questi algoritmi non possono essere applicati con successo con l'hardware attuale.

4.5.1. Autolocalizzazione nel museo

Il metodo della localizzazione è stato applicato con un buon successo al robot B21r all'interno del Museo Archeologico Regionale di Agrigento, dove le condizioni di luminosità, seppur costanti nel tempo, presentavano diversi problemi come ad esempio dei fari proiettati contro la telecamera durante la fase di localizzazione. A causa di un malfunzionamento della bussola il sistema di localizzazione non forniva sempre un valore preciso, infatti, come è stato mostrato nelle note sugli errori, si ottenevano dei risultati che discostavano anche di 20 cm dalla posizione assoluta del robot sia lungo l'asse x che lungo l'asse y.

Nella Figura 4-10 è mostrata la sala dei Telamoni dove il robot faceva la sua presentazione delle opere.

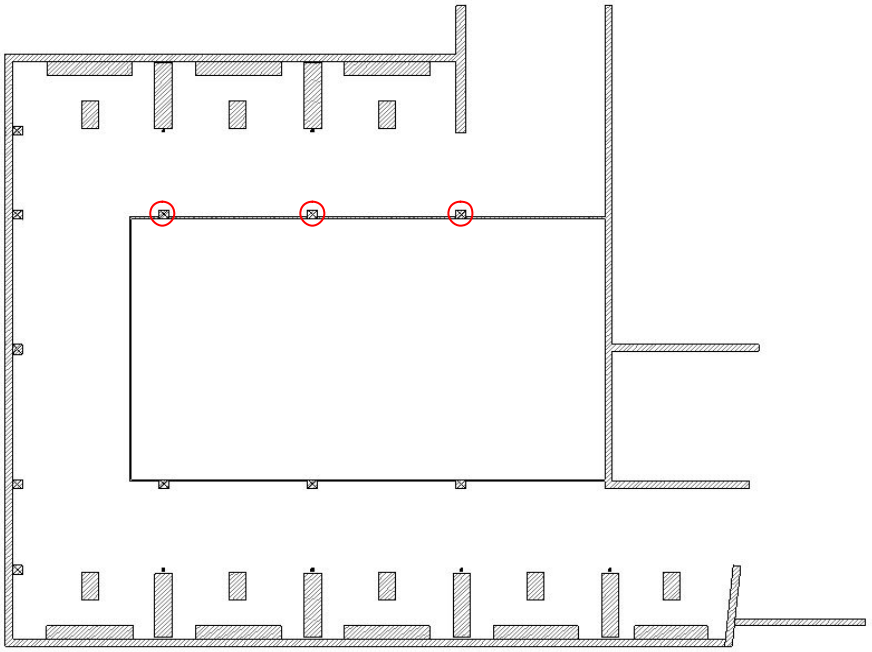


Figura 4-10 Sala dei Telamoni del museo archeologico di Agrigento

I cerchi rossi indicano i punti nella pianta in cui si trovavano i marker. Il robot conoscendo la loro posizione, la distanza ottenuta dalla triangolazione e il suo angolo, riusciva a localizzarsi. Durante le sperimentazioni il robot faceva una descrizione di mezza sala del museo, in seguito basterà aggiungere altri marker nei pilastri rimanenti per poter effettuare una visita completa della sala. Il robot riesce

ad avere la sua posizione angolare grazie all'odometria dei motori, ma questa pur fornendo dei buoni valori viene disturbata dal pavimento non proprio piatto del museo che fa ruotare il robot quando ci sono degli avvallamenti. Questi avvallamenti creano degli errori che sono difficilmente individuabili senza l'utilizzo di una bussola che fornisce un valore più esatto di quello dato dall'odometria.

4.6. Rilevamento dinamico di ostacoli

Al museo regionale di Agrigento sono stati fatte delle sperimentazioni su come eliminare i dati sensoriali provenienti da elementi costruttivi, come ad esempio un muro, in una lettura laser dell'ambiente circostante. La situazione è la seguente:

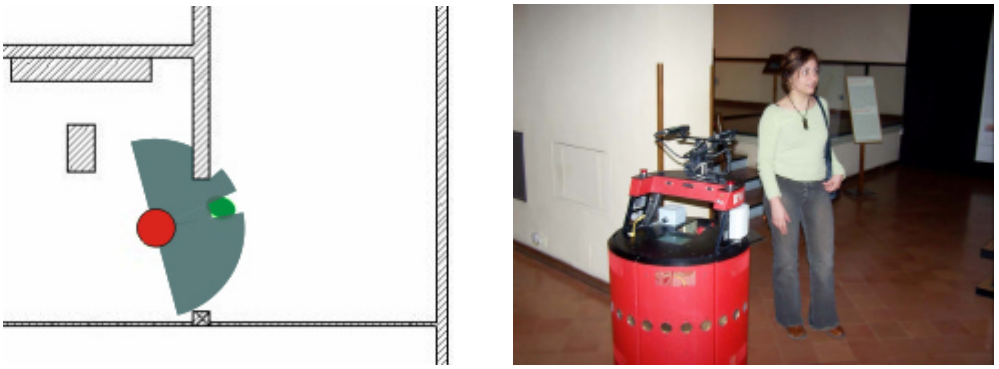


Figura 4-11 Esempio di ostacolo non conosciuto a priori

In questo caso il simulatore è nella stessa posizione spaziale del robot, fornisce una lettura simulata del laser senza l'ostacolo, mentre il laser reale fornisce la lettura completa. Eseguendo una comparazione fra le due letture, è possibile estrarre i dati del solo ostacolo che non fanno parte della struttura dell'ambiente.

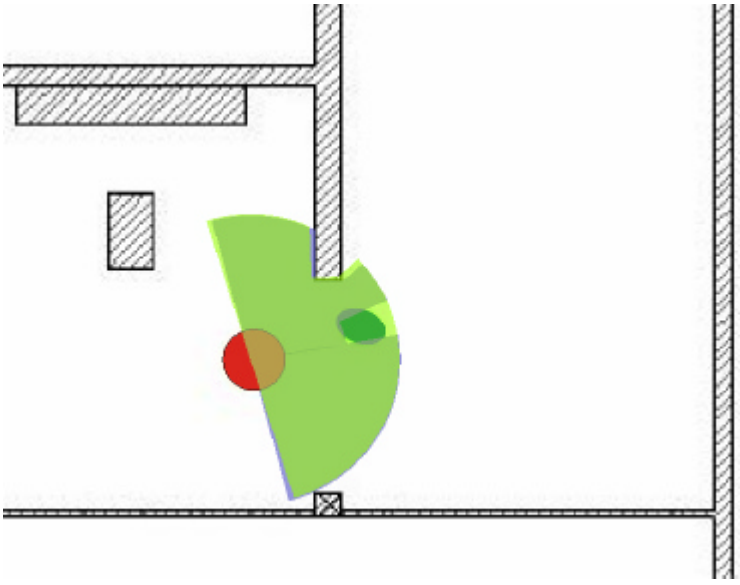


Figura 4-12 Matching di letture laser

Come si può vedere nella Figura 4-12, i dati della lettura reale e quelli della lettura del simulatore sono leggermente ruotati, ma il fattore determinante dell'esperimento è che viene riconosciuto l'ostacolo che il robot ha davanti a se, il quale non fa parte della struttura dell'ambiente esterno.

Se consideriamo per semplicità di scrittura i dati del laser compresi fra 80 e 120 gradi, proprio dove si trova l'ostacolo, e facciamo slittare i dati simulati di 2 gradi in senso orario, otteniamo la seguente tabella che indica la distanza dell'ostacolo dal centro nell'angolo considerato:

Angolo	Artificiale	Reale	Differenza
80	190	192	–2
81	190	190	0
82	190	191	–1
83	190	188	2
84	190	190	0
85	190	189	1
86	190	190	0
87	190	187	3
88	165	159	6
89	164	163	1
90	163	160	3
91	161	158	3
92	160	159	1
93	158	155	3
94	156	159	3
95	156	153	3
96	159	154	4
97	162	160	2
98	165	159	6
99	190	189	1
100	190	192	–2

Angolo	Artificiale	Reale	Differenza
101	190	191	–1
102	190	187	3
103	190	190	0
104	190	189	1
105	190	193	–3
106	190	190	0
107	190	188	2
108	190	189	1
109	190	190	0
110	168	171	–3
111	166	168	–2
112	164	166	–2
113	162	167	–5
114	160	162	–2
115	159	163	–4
116	160	160	0
117	162	161	1
118	164	162	2
119	165	164	1
120	166	165	1

Tabella 1 Valori delle letture reali e simulate

Il laser non fornisce una lettura priva di errori, infatti, in ogni lettura c'è sempre un rumore sulle letture che porta errore di qualche cm, quindi non si ha una lettura perfetta; in pratica in uno spazio privo di ostacoli, la lettura del laser non crea un perfetto semicerchio, ma il contorno di questo semicerchio è frastagliato. Al contrario nel simulatore questo problema di frastagliamento non è presente. Per far si che le due letture possano essere confrontate, si fa uso di una soglia oltre la quale le due letture si possono considerare diverse. Come si può notare nella tabella

precedente i valori simulati sono molto costanti e privi di rumore, mentre quelli reali hanno un rumore di fondo che porta al frastagliamento delle letture.

Nella Figura 4-13 vengono rappresentati le due letture del laser:

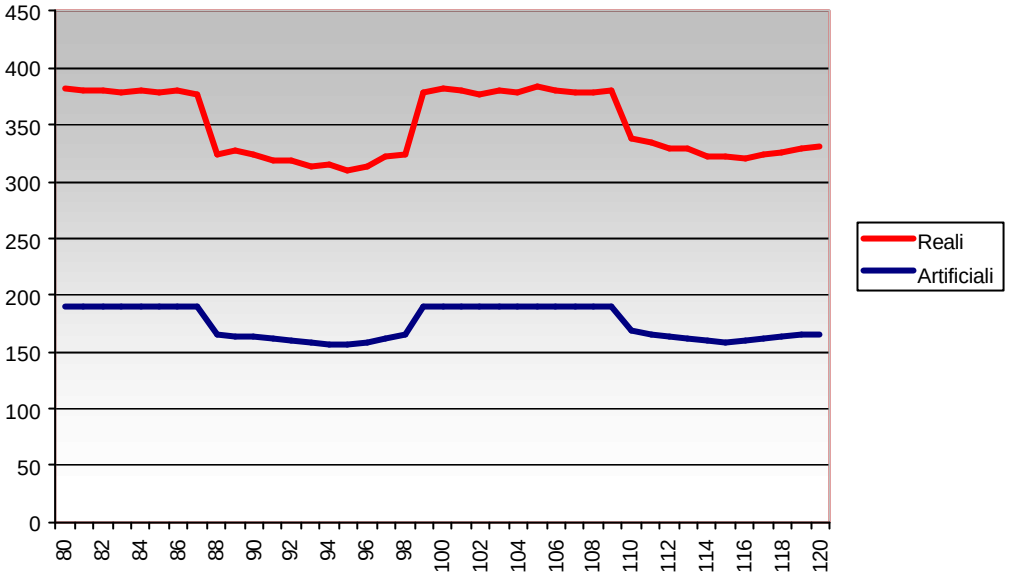


Figura 4-13 Rappresentazione delle letture del laser reali e simulate

Se calcoliamo la varianza, considerando la media della differenza dei dati $\bar{d} = 0{,}6585$ otteniamo: $s^2 = 5{,}98;\ s = 2{,}445$.

Trattandosi di due vettori di 180 valori ciascuno, possiamo tranquillamente effettuare un controllo esaustivo su tutti i valori, in pratica, si effettuano 5.340 confronti. Al momento per non appesantire molto i calcoli non si è deciso di effettuare un'ulteriore confronto sull'errore di traslazione, infatti, supponendo di voler considerare 10 cm in meno e 10 cm in più rispetto alla posizione del robot, bisognerebbe effettuare 106.800 confronti per ogni lettura di laser. Avendo a disposizione un laser che legge più volte al secondo possiamo tranquillamente arrivare a circa 1.000.000 di confronti al secondo; tenendo presente che il calcolatore deve effettuare anche i calcoli per il simulatore 3D, questo tipo di confronto al momento non è consigliabile. Naturalmente si può pensare di effettuare un confronto completo non ad ogni lettura, ma dopo un certo periodo.

Per scegliere l’angolo che meglio approssima le letture reali, si è scelto di calcolare l’errore quadratico medio di tutti i confronti e considerare la rotazione che ha un errore minore.

5. ARCHITETTURA DISTRIBUITA E POSSIBILI ESTENSIONI

Questo capitolo descrive l'architettura utilizzata durante le sperimentazioni, illustrandone le varie parti del sistema, inoltre, viene suggerita anche una possibile estensione al sistema che aiuterebbe l'utente che intende visitare un museo.

5.1. Distribuzione dei moduli di elaborazione

L'intera architettura del sistema è distribuita su più calcolatori, infatti, la comunicazione fra i vari programmi è gestita tramite socket; questo tipo di comunicazione rende indipendente anche il tipo di calcolatore che può essere diverso sia come sistema operativo che come architettura hardware. Nella Figura 5-1 è rappresentata l'intera architettura del sistema di navigazione del robot.

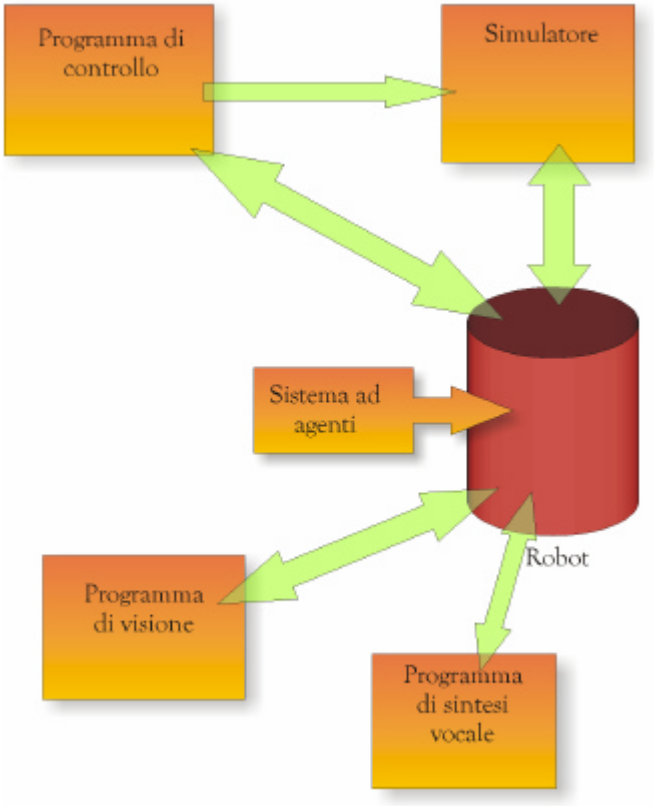


Figura 5-1 Architettura distribuita completa del Robot

Il simulatore usato è comandabile da remoto tramite socket dal programma di controllo descritto nell'appendice o direttamente dal robot. I comandi inviati al simulatore consentono lo spostamento in avanti o indietro e la rotazione su se stesso in senso orario e in senso antiorario; in realtà questi sono i gradi di libertà del robot, quindi sono presi in considerazione solo questi movimenti. Ovviamente il comando di avanzamento viene considerato anche con l'angolo che il robot ha in quel momento, onde evitare che l'avanzamento venga fatto solo sull'asse delle x. Il flusso video del simulatore potrebbe essere sfruttato in seguito per aiutare il robot a localizzarsi o come viene fatto con il laser a riconoscere degli oggetti che non fanno parte della struttura dell'ambiente.

Il robot è gestito da una struttura ad agenti che pianificano e consentono il movimento del robot stesso sfruttando dei driver per la comunicazione con l'hardware. Gli agenti inviano anche le informazioni sensoriali al motore grafico che permette di visualizzare le immagini sintetiche che rappresentano quello che dovrebbe vedere il robot. Il simulatore consente anche di ricreare le informazioni sensoriali prive di rumore e di oggetti non appartenenti alla struttura, queste informazioni vengono inviati al sistema di agenti che li elaborano per migliorare la navigazione.

Il programma di controllo serve a comandare il robot e/o il motore grafico. Il programma di controllo permette anche di poter vedere in anteprima nell'ambiente simulato cosa potrebbe vedere il robot, inoltre, può comandare direttamente il robot per spostarlo da un posto all'altro. Un'applicazione di movimento del robot comandato a distanza potrebbe essere quella di video sorveglianza, infatti, da remoto si potrebbe vedere il flusso video del robot, al quale può essere inviato il punto di arrivo. Nel caso in cui il robot venga comandato a distanza, esso non viene soltanto radiocomandato, ma esso è dotato di una sua "intelligenza" che gli permette di aggirare in maniera autonoma gli ostacoli.

Il programma di visione si occupa della localizzazione e di gestire il flusso video del robot. Per controllare le telecamere, il programma comunica direttamente con il driver del pantilt, il quale, gli consente di poter girare le telecamere poste sopra il pantilt stesso. Tutti i calcoli della distorsione della telecamera, del riconoscimento

del marker, del calcolo della distanza dal marker e del riallineamento dell'angolo sono effettuati da tale programma. Il programma di visione si occupa anche dell'interfaccia grafica da mostrare all'utente che segue il robot durante le sue spiegazioni.

Il programma di sintesi vocale si occupa di leggere le informazioni relative ad una vetrina o ad un particolare oggetto esposto. Il motore di sintetizzatore vocale sfruttato è quello della Microsoft, il quale, consente di ottenere un sintetizzatore vocale in maniera molto veloce, anche se la qualità ottenuta non è eccellente. Il sistema ad agenti comunica direttamente con questo programma con i socket, indicando quale file di testo deve leggere per spiegare una determinata vetrina. Questo programma si occupa anche di poter eseguire un file audio pre–registrato in maniera simile a come gestisce i file di testo da leggere. Un'altra funzionalità di questo programma è quella di suonare un clacson quando il robot non ha abbastanza spazio per poter proseguire il suo percorso.

5.2. Prototipo di interazione Utente – Robot tramite sistema palmare

L'interazione tra utente e robot può avvenire tramite l'uso di un palmare che con l'interazione dell'utente con una tastiera. Il sistema che fa uso di un palmare ha il compito di interagire con l'utente che visita un museo o un sito di particolare interesse tramite un palmare o un robot.

L'utente visitando un sito potrebbe avere bisogno di ulteriori informazioni su una particolare opera, avere informazioni su dove è locata la stessa nel museo o creare un percorso di visita delle opere secondo le proprie esigenze. Nel sistema, infatti, è possibile avere una visualizzazione tridimensionale del sito sul palmare in modo da simulare il percorso che si è scelto di fare, inoltre, si possono vedere le opere in anteprima sul palmare stesso.

Il sistema è costituito da un server che contiene diverse mappe che verranno inviate al palmare in base alle richieste del palmare stesso. Per non appesantire la memoria del palmare, un intero sito può essere diviso in diverse zone, quindi quando un utente passa da una zona all'altra, il palmare richiede al server la nuova mappa che verrà rappresentata sul palmare. I file delle mappe occupano pochi migliaia di byte e possono essere inviate in pochissimo tempo, uno dei grandi vantaggi di questo procedimento è quello di avere le mappe sempre aggiornate su un locale, infatti, basta aggiornare le mappe sul server e tutti i palmari quando richiederanno le mappe, avranno sempre quella aggiornata.

Sfruttando un sistema di riconoscimento di oggetti è possibile aggiornare le mappe in maniera automatica sul server e quindi su tutti i palmari.

I palmari verranno dotati di un sistema di localizzazione spaziale, come ad esempio sfruttando il sistema GPS per luoghi all'aperto o altri metodi per luoghi al chiuso, in questo modo oltre a poter localizzare i vari utenti e poter monitorare le opere più visitate o più apprezzate, i palmari stessi conoscendo la propria posizione spaziale, sono in grado di dare delle ulteriori informazioni sulle opere che sono vicino all'utente stesso. L'utente può sfruttare il palmare per visualizzare in anteprima altre zone del sito, infatti, utilizzando il motore 3D, ci si può muovere

all'interno del sito come ci si muove in un videogioco con una visualizzazione in prima persona.

Nella Figura 5-2 si vede una rappresentazione grafica del sistema in oggetto:

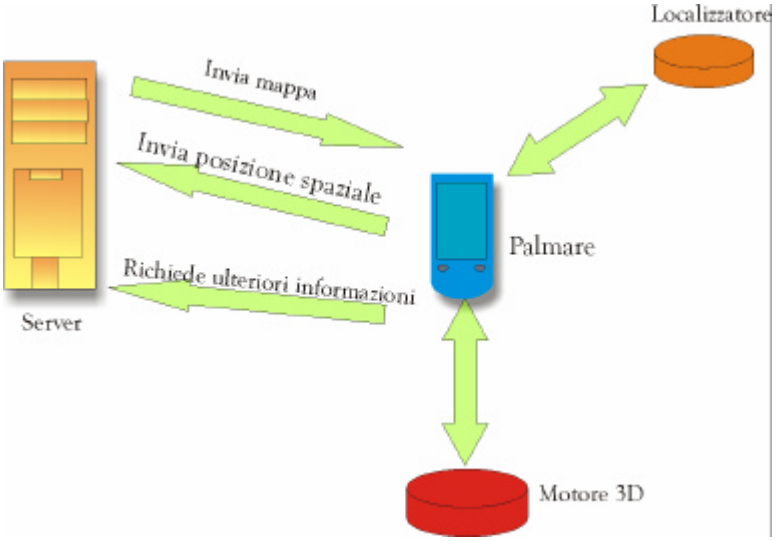


Figura 5-2 Sistema Robot - Palmare

Nella Figura 5-3 è mostrata la struttura del sistema in maniera dettagliata.

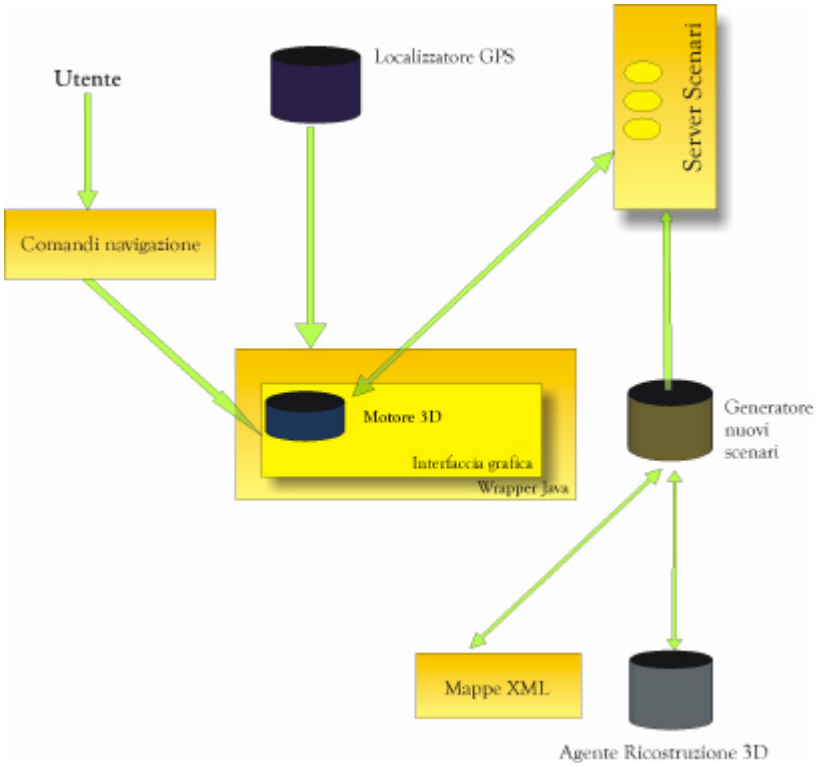


Figura 5-3 Dettaglio sistema Robot – Palmare

Come detto precedentemente, l'utente da i comandi di navigazione al palmare premendo i relativi pulsanti sul display touch–screen, ovvero interagisce con l'interfaccia grafica. Il motore 3D effettua il render della scena in maniera da fornire almeno 15 fotogrammi al secondo per avere una fluidità dell'immagine accettabile. Tramite un'interfaccia (Wrapper) Java si interagisce con un localizzatore GPS. L'interfaccia è necessaria perché si vuole sfruttare un'esistente implementazione di localizzatore GPS.

Il motore 3D in base alla posizione di visualizzazione richiede la mappa al server, il quale invia lo scenario aggiornato ed eventualmente delle nuove tessiture.

Il server ha il compito di tenere le mappe ed eventualmente aggiornarle. L'aggiornamento può essere fatto manualmente da una persona addetta, oppure automaticamente tramite l'ausilio di immagini o video acquisiti da telecamere. Il generatore dei nuovi scenari ha proprio il compito di creare o aggiornare le mappe. Gli scenari vengono creati da mappe scritte precedentemente in XML o tramite un Agente di Ricostruzione 3D, il quale ha l'incarico di ricostruire gli scenari partendo da immagini o video.

5.2.1. Scelta del motore grafico 3D per il palmare

Il motore grafico deve riuscire ad avere un buon livello di rendering ed inoltre deve essere fluido durante la visualizzazione. Attualmente, non è pensabile usare motori 3D che fanno uso di Java, in quanto, oltre ad essere necessaria una notevole potenza di calcolo, la rappresentazione a video non è delle migliori.

L'esigenza porta alla scelta di motori grafici che interagiscono direttamente con l'hardware del dispositivo, per riuscire ad ottenere delle buone prestazioni. Per far questo, è necessario far uso di un linguaggio come il C o C++ e vincolarsi al sistema operativo e al tipo di processore.

Un grande problema dell'utilizzo di un motore grafico in un palmare è che il palmare non è dotato di grandissime potenze di calcolo, quindi bisogna usare del codice ottimizzato in ogni sua parte.

Un altro problema è quello della scelta del sistema operativo su cui deve girare questo motore 3D, infatti, in commercio ci sono due diverse aree di utilizzo.

Un'area è quella del Palm OS, il quale è presente nel mercato da diversi anni, questa sua esperienza porta ad una maggiore affidabilità del sistema operativo stesso; infatti, un Palm OS raramente bisogna resettarlo per problemi software, mostrando una migliore stabilità.

L'altra area è quella dei Pocket PC che utilizzano Windows CE. A differenza del Palm OS, Windows CE è relativamente giovane, ed è più frequente vedere degli errori che portano al crash del palmare stesso. Di contro ha il gran vantaggio di poter fare agevolmente l'adattamento del codice sul palmare di un progetto sviluppato su sistemi Win32, e per aiutare maggiormente lo sviluppatore la Microsoft ha creato un compilatore multiprocessore con un IDE molto simile a quello di Visual Studio. I due sistemi operativi sono molto diversi fra di loro e non è molto semplice fare il porting del codice fra uno e l'altro.

Il Palm OS utilizza il compilatore CodeWarrior della Metrowerks, il quale è un ottimo prodotto, ma oltre a non essere freeware, il codice usato non è facilmente utilizzabile con altri compilatori. Un altro punto a sfavore del Palm OS è la poca disponibilità di progetti open source in rete.

La scelta del sistema operativo tende ai Pocket PC con Windows CE 2003, perché offrono una notevole portabilità di codice da Win32 e inoltre la Microsoft tenderà a migliorare i suoi prodotti perché vuole accaparrarsi questa fetta di mercato.

Facendo una rapida ricerca in internet ci si accorge che esistono dei motori grafici 3D freeware che possono essere usati per Pocket PC. Consultando il sito: www.3dengines.net si può trovare qualche motore grafico esclusivamente per Pocket PC. Un interessante motore è Klimt [10], in quanto, oltre ad essere open source e con delle API simili a quelle dell'OpenGL, in futuro gli autori supporteranno il PalmOS e il sistema operativo Symbian. Un altro motore 3D è Traktor [11], anche questo è freeware e come il precedente il codice completo è disponibile su richiesta; l'unico inconveniente che si può notare dopo aver scaricato gli header delle librerie è che i formati dei file trattati sono poco diffusi, anche se è possibile importare gli scenari creati per il gioco Quake 3.

Esistono altri motori 3D, ma alcuni hanno una resa grafica non paragonabile ai precedenti, mentre altri non sono freeware e il loro costo non è indifferente.

Un motore che suscita molto interesse è quello di Quake 2 della id Software [19]. Uno dei grandi vantaggi nell'utilizzo di questo motore grafico, oltre ad essere uno dei migliori in circolazione, è quello di poter usufruire gratuitamente di programmi per la creazione di scenari che esistono da diversi anni.

Provandolo su un palmare si nota che i movimenti sono fluidi, infatti, si riesce ad avere anche 15 frame al secondo. Un altro vantaggio nell'uso di questo codice è che si tratta di un vero e proprio gioco commerciale, ottimizzato nella velocità e completo in tutte le sue parti; quindi a differenza dei precedenti in cui bisogna creare il codice per gestire i movimenti o altro, qui bisogna eliminare delle funzionalità che per le nostre esigenze sono superflue. Sono state effettuate poche modifiche a partire dal codice originale per poterlo adattare all'utilizzo in un Pocket PC ed inoltre in questo lavoro si sono corrette delle parti di codice per poter compilare lo stesso progetto sia per palmare che per Win32.

6. BIBLIOGRAFIA

1. A. Chella, H. Dindo, I. Infantino, *"A Cognitive Framework for Learning by Imitation"*, ICRA '05, workshop on the Social Mechanisms of Robot Programming by Demonstration, April 18-22, 2005, Barcelona, Spain.
2. A. Chella, M. Frixione, and S. Gaglio. *A cognitive architecture for artificial vision*. Artificial Intelligence, pp. 89:73–111, 1997.
3. A. Chella, M. Frixione, S.Gaglio, *Understanding dynamic scenes*, Artificial Intelligence, Vol 123, pp. 89-132, 2000.
4. A. Ruisi, M. Cossentino, I. Infantino, A. Chella, R. Pirrone *"An Agent Based Architecture for Cooperative Robotics in Vision Tasks*", IROS-2002 Workshop on Cooperative Robotics, Lausanne, Switzerland, October 1, 2002.
5. B. Caci, M. Cardaci, A. Chella, A. D'Amico, I. Infantino, I. Macaluso, "*Personality and Learning in Robots. The Role of Individual Motivations/Expectations/ Emotions in Robot Adaptive Behaviours*", Agents that want and like: motivational and emotional roots of cognition and action, a symposium of the AISB'05 Convention, University of Hertfordshire, April 14-15, 2005, Hatfield, UK.
6. J.F. Allen, "*Towards a general theory of action and time*", Artificial Intelligence, vol. 23(2), pp. 123–154, 1984.
7. James D. Foley, Andries van Dam, Steven K. Deiner, John F. Hughes, "*Computer Graphics Principles and Practice (Second edition in C)*", Addison Wesley, 2002.
8. Librerie *OpenCV* della Intel Corporation, scaricabili da: http://www.intel.com/research/mrl/research/opencv/
9. Motore grafico *Irrlicht* http://irrlicht.sourceforge.net/
10. Motore grafico *Klimt* http://studierstube.org/klimt/
11. Motore grafico *Traktor* http://www.pepperoni.nu/index.php

12. *OpenCV Reference Manual*, scaricabile da http://www.intel.com/research/mrl/research/opencv/
13. P. Gärdenfors, "*Conceptual Spaces*", MIT Press-Bradford Books, Cambridge, MA, 2000.
14. Programma per la creazione di mappe http://www.qeradiant.com/
15. *Quark* programma Open Source per la creazione di mappe http://www.sourceforge.net/projects/quark
16. R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, 2004.
17. Rafael C. Gonzales, Richard E. Wood, "*Digital Image Processing*", Prentice-Hall 2nd Edition, 2002.
18. S. Schaal, *"Is imitation learning the route to humanoid robots?"*, Trends in Cognitive Sciences 3, 233-242, 1999.
19. Sito di riferimento per quake su pocket pc: http://quake.pocketmatrix.com/
20. W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, S. Thrun, *Experiences with an interactive museum tour-guide robot*, Artificial Intelligence 114 (1999) 3–55. (DA INSERIRE NEL MUSEO DI AG COME ALTRI ESEMPI)
21. Z. Zhang. "A Flexible New Technique for Camera Calibration." IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(11):1330-1334, 2000.

APPENDICI

I compilatori usati per questi esperimenti sono stati il Microsoft Visual C++ 6.0 e il Borland C++ Builder 5.0. Il primo è stato utilizzato per la creazione del motore grafico 3D, mentre il secondo per tutto il resto. I driver sul robot sono stati compilati con il compilatore gcc, trattandosi di una piattaforma linux.

DESCRIZIONE DELLA STRUTTURA DI QUAKE 2

Quake è un videogioco creato dalla IdSoftware nel 1997. Questo gioco sfrutta la visuale in prima persona per aumentare il realismo della scena e per far questo i programmatori hanno creato un motore 3D che offre una rappresentazione a video molto accurata se paragonata all'anno di uscita del gioco stesso. Dopo qualche anno la casa produttrice del gioco ha rilasciato i sorgenti del gioco stesso con licenza GNU *General Public License*. I sorgenti di questo gioco sono stati facilmente portati su palmari con sistemi operativi WinCE, mostrando la grande portabilità del codice stesso.

Il gioco consiste in un eseguibile che ha la funzione di server, di librerie che gestiscono la parte grafica e di una libreria che contiene la logica del gioco stesso. Questa struttura modulare facilita molto l'utilizzo del *multigame* con più utenti sia su rete LAN che su internet ed è possibile cambiare la logica del gioco semplicemente usando un'altra libreria.

Un analogo ragionamento è possibile fare per la grafica, infatti, se si vuole sfruttare un particolare scheda video o un nuovo acceleratore hardware, basta cambiare la sola libreria relativa al rendering grafico. Attualmente le librerie grafiche in uso sono quelle in OpenGL sfruttando le capacità della scheda video e quelle che non usano queste librerie e sfruttano la sola capacità di calcolo del processore. Per adattare il gioco all'esigenza dell'esperimento, è stato necessario effettuare alcune modifiche ai vari moduli del gioco.

MODIFICHE EFFETTUATE AI VARI FILE DI QUAKE 2

cl_parse.c

Nella funzione `CL_CheckOrDownloadFile` si commenta la funzione: `Com_Printf("Downloading %s\n",cls.downloadname);`

In questo modo non si mostrano gli errori dovuti al caricamento di un file. Analogamente si commenta la stessa chiamata nella funzione: `CL_Download_f`.

cl_view.c

Come in altri file si include il file dove sono immessi altri comandi `#include "../peppe/altricom.h"`

Nella funzione `V_RenderView` si imposta `cl.refdef.fov_x = Zoom;` ovvero si memorizza il valore che il motore di rendering utilizza per l'ingrandimento, tale valore può essere cambiato dall'utente a piacimento.

menu.c

Le modifiche effettuate alla funzione `M_Quit_Key` consentono di poter uscire dal menù premendo i tasti 's' o 'S' invece di 'y' o 'Y'.

g_func.c

Per consentire di non levare vitalità al robot quando tocca una porta, si commenta la chiamata alla funzione:

```
T_Damage(other, self,self, vec3_origin, other->s.origin, vec3_origin, 100000, 1, 0, MOD_CRUSH);
```

g_weapon.c

Con il codice della funzione `blaster_touch`

```
globals.PuntoImpatto[0] = self->s.origin[0];
globals.PuntoImpatto[1] = self->s.origin[1];
globals.PuntoImpatto[2] = self->s.origin[2];
```

Si salvano nella variabile globals le tre coordinate del punto di impatto di un proiettile con un oggetto. La variabile globals è di tipo game_export_t ovvero una struttura che viene passata al motore 3D da un client del gioco.

Un analogo procedimento viene effettuato per salvare la normale nel punto di impatto:

```
globals.PuntoImpattoN[0] = plane->normal[0];
globals.PuntoImpattoN[1] = plane->normal[1];
globals.PuntoImpattoN[2] = plane->normal[2];
```

Commentando il codice: `bolt->s.effects |= effect` nella funzione `fire_blaster` viene disabilitato l'effetto luminoso che viene prodotto da un proiettile quando c'è uno sparo. Inoltre, per non caricare il modello tridimensionale del proiettile e per eliminare il suono dello sparo, si commentano le seguenti due chiamate:

```
bolt->s.modelindex=gi.modelindex
("models/objects/laser/tris.md2");
bolt->s.sound = gi.soundindex ("misc/lasfly.wav");
```

p_weapon.c

Nella funzione `Blaster_Fire` commento il codice relativo al flash e al rumore dello sparo:

```
gi.WriteByte (svc_muzzleflash);
gi.WriteShort (ent-g_edicts);
if (hyper)
      gi.WriteByte (MZ_HYPERBLASTER | is_silenced);
else
      gi.WriteByte (MZ_BLASTER | is_silenced);
gi.multicast (ent->s.origin, MULTICAST_PVS);
```

Nella funzione `P_ProjectSource` faccio in modo che lo sparo sia sempre nella zona centrale dello schermo; per fare ciò commento il codice:

```
if (client->pers.hand == LEFT_HANDED)
      _distance[1] *= -1;
else if (client->pers.hand == CENTER_HANDED)
      _distance[1] = 0;
```

e inserisco il codice:

```
_distance[1] = 0;
```

game.h

Si aggiungono i due vettori

```
    float       PuntoImpatto[3];  //Salvo il punto di impatto
    float       PuntoImpattoN[3];//Salvo la normale al punto di impatto
```

alla struttura `game_export_t`.

g_phys.c

Commento il codice relativo alla gravità relativa ad un proiettile

```
    if (ent->movetype != MOVETYPE_FLY
    && ent->movetype != MOVETYPE_FLYMISSILE)
          SV_AddGravity (ent);
```

m_soldier.c

Per inserire un altro oggetto in movimento all'interno dell'ambiente 3D, si può associare questo oggetto a quella di un soldato del gioco quake. Per evitare che l'oggetto inserito nell'ambiente si comporti in modo bellicoso nei confronti dell'osservatore che rappresenta il robot, si commenta il codice della funzione `soldier_fire`.

Bisogna fare un analogo ragionamento per le funzioni `soldier_fire1`, `soldier_attack1_refire2`, `soldier_fire2`, `soldier_attack2_refire1` e `soldier_attack2_refire2`.

gl_draw.c

Il gioco utilizza diverse immagini che visualizzano nello schermo la vitalità, numero di proiettili e così via, che consentono al giocatore di avere un resoconto visivo. Allo stato attuale queste informazioni non sono necessarie per le nostre necessità, quindi bisogna commentare il codice che visualizza a schermo il mancato caricamento di un'immagine.

Il codice da commentare è:

```
ri.Con_Printf (PRINT_ALL, "Can't find pic: %s\n", pic);
```

Questo codice è ripetuto nelle funzioni `Draw_StretchPic`, `Draw_Pic` e `Draw_TileClear`.

gl_image.c

Analogamente a quanto detto per il file gl_draw.c bisogna commentare il codice:

`ri.Con_Printf(PRINT_ALL,"GL_FindImage: can't load %s\n",name);`

nella funzione `GL_LoadWal`.

gl_mesh.c

In questo file vengono trattati i modelli 3D, quindi per le stesse ragioni del file gl_draw si commenta il codice:

`ri.Con_Printf (PRINT_ALL, "R_CullAliasModel %s: no such frame %d\n", currentmodel->name, e->frame);`

Nelle funzioni `R_CullAliasModel` e `R_DrawAliasModel`.

I file r_draw.c, r_image.c hanno le stesse modifiche di gl_draw.c e gl_image.c, in quanto r_draw e r_image si riferiscono al rendering software del motore 3D, mentre quelli con prefisso gl_ si riferiscono al rendering opengl.

sv_main.c

Questo file fa parte del server del gioco, il quale si connette con il client usato per pilotare il robot. Nella funzione `SV_RunGameFrame`, si copiano i valori del punto di impatto e normale al punto in variabili locali.

```
PuntoImpatto[0] = ge->PuntoImpatto[0];
PuntoImpatto[1] = ge->PuntoImpatto[1];
PuntoImpatto[2] = ge->PuntoImpatto[2];
PuntoImpattoN[0] = ge->PuntoImpattoN[0];
PuntoImpattoN[1] = ge->PuntoImpattoN[1];
PuntoImpattoN[2] = ge->PuntoImpattoN[2];
```

in_win.c

Nella funzione `IN_Move`, si aggiunge il codice relativo al movimento del robot. I comandi vengono ricevuti dal client tramite un thread in ascolto in una porta, quindi per comunicare i vari comandi, si fa uso di variabili globali che contengono il valore di quanto il robot si deve spostare.

```
if(Avanti != 0)
```

```
{     cmd->forwardmove = Avanti * 4;
      if(Avanti > 0)
      {     Avanti -= SPOSTADI;
            if(Avanti < 0)
                  Avanti = 0;
      }
      else
      if(Avanti < 0)
      {     Avanti += SPOSTADI;
            if(Avanti > 0)
                  Avanti = 0;
      }
}
if(Destra != 0)
{     cmd->sidemove = Destra * 4;
      if(Destra > 0)
      {     Destra -= SPOSTADI;
            if(Destra < 0)
                  Destra = 0;
      }
      else
      if(Destra < 0)
      {     Destra += SPOSTADI;
            if(Destra > 0)
                  Destra = 0;
      }
}
if(AngoloO != 0)
{
      if(AngoloO > 0)
            cl.viewangles[YAW] -= DELTAANGOLO;
      else
            cl.viewangles[YAW] += DELTAANGOLO;
      AngoloO = 0;
}
AngoloOrizzontale = cl.viewangles[YAW];
AngoloVerticale   = cl.viewangles[ROLL];
```

sys_win.c

Nella funzione `WinMain` si aggiunge il seguente codice:

```
    if(strcmp(lpCmdLine,"") == 0)
          strcpy(lpCmdLine,"+map dinfo");

    CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)InizializzaSocke
t,NULL,0,(LPDWORD)&tID);
    Avanti                  = 0;
    Destra                  = 0;
    AngoloO                 = 0;
    AngoloV                 = 0;
    AttivaComandiInput      = 1;
    Zoom                    = 90;
```

Questo consente di avviare il programma con una mappa prestabilita se non viene passato nessun parametro all'avvio, inoltre crea un thread che sta in ascolto su una porta e alla fine inizializza il valore di alcune variabili.

PROGRAMMA DI CONTROLLO

Questo programma ha la funzione di gestire la visuale del robot, di inviare i comandi di moto al robot stesso, di acquisire l'immagine prodotta e di fare un confronto con l'immagine reale.

Il programma si collega tramite socket, usando la porta 27015, al server del motore grafico, il quale comunica ripetutamente la posizione del robot, l'angolo della telecamera e le coordinate del punto del primo oggetto visibile alla telecamera stessa. Dopo aver acquisito questi dati, il programma di controllo, posiziona il robot in una sua piantina in modo da poter avere una visione spaziale della sua posizione; con le coordinate del primo oggetto visibile è possibile individuare degli oggetti che hanno un particolare rilievo sulla scena. Al momento della creazione dell'ambiente tridimensionale è possibile dare delle etichette a degli oggetti che poi possono essere riconosciuti a run-time con un semplice confronto di coordinate.

Figura 6-1 Finestra principale programma di controllo

Il programma si presenta all'utente con questa finestra dove è possibile inserire l'indirizzo IP della macchina dove è in esecuzione il motore 3D da gestire. Dopo

aver inserito l'indirizzo basta premere il pulsante *Connetti* per effettuare la connessione. A connessione avvenuta, il programma lo segnala all'utente con la dicitura "*Connesso*".

La parte centrale della finestra consente di effettuare dei movimenti alla visione che crea il motore 3D, infatti, si può andare avanti e indietro, muoversi verso destra o verso sinistra ed è possibile ruotare il punto di vista sia in senso orario che antiorario. Naturalmente i comandi si devono considerare sempre in relazione al punto di vista che si sta vedendo, cioè se si vuole spostare il robot verso destra si deve considerare il movimento come se fossimo dietro al robot stesso.

Il pulsante *"Ripeti"* consente di tenere premuto un comando di spostamento evitando di premere diverse volte con il mouse su un pulsante, per esempio questo accade quando si vuole camminare avanti o indietro in un lungo corridoio.

Selezionando la casella *Richiedi posizione ogni secondo*, il programma chiede continuamente la posizione spaziale del robot e del motore 3D, in modo da sincronizzare la mappa e poter riconoscere eventuali oggetti già presenti nella mappa stessa. In alternativa se non si vuole appesantire il sistema con la richiesta continua della posizione, l'utente può richiederla premendo il pulsante *Posizione*, quando lo ritiene necessario.

Premendo il tasto *Vis* si può modificare l'angolo di visuale del motore 3D, questo perché ogni telecamera ha un proprio angolo visivo che causa delle modifiche di prospettiva. Per ovviare l'incongruenza fra la prospettiva della telecamera e quella del motore 3D, è stato necessario aggiungere questo parametro al programma in modo da renderlo indipendente dal tipo di telecamera usata.

Il pulsante *Acquisisci*, invia al robot e al motore 3D il comando di acquisizione di un'immagine che sarà poi elaborata manualmente dall'utente. Per accedere alla sezione di elaborazione delle immagini basta premere il pulsante *Elabora.*

I pulsanti *Piantina* ed *Entità* fanno visualizzare rispettivamente la piantina dell'ambiente e le entità o oggetti riconosciti nell'ambiente stesso.

VISUALIZZAZIONE MAPPA

Questa finestra consente di avere una visualizzazione della mappa con una vista dall'alto proprio come in una cartina. Oltre a vedere la mappa, è possibile visualizzare la posizione del robot e la sua visuale [7], in modo da avere un rapporto visivo di dove si trova il robot e cosa stia guardando.

Il programma consente di effettuare uno zoom, in modo da visualizzare meglio la posizione del robot, inoltre in basso sono presenti le coordinate spaziali e l'angolo di direzione del robot stesso.

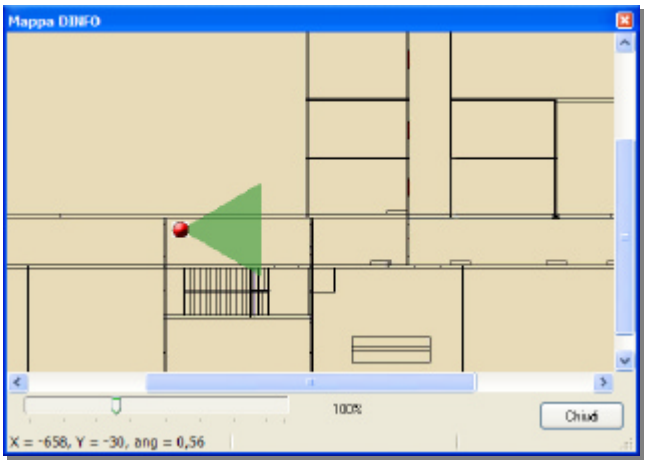


Figura 6-2 Finestra della mappa

VISUALIZZAZIONE ENTITÀ

Una parte dell'esperimento consiste nel riconoscere alcuni oggetti della scena. Gli oggetti da riconoscere sono stati precedentemente indicati durante la creazione dello scenario, adesso, se il robot ha un oggetto dentro la sua visuale il programma elenca cosa ha riconosciuto.

In un esempio fatto, si sono indicate le porte del corridoio del DINFO, quindi quando il robot è passato nel corridoio, si è avuta questa situazione:

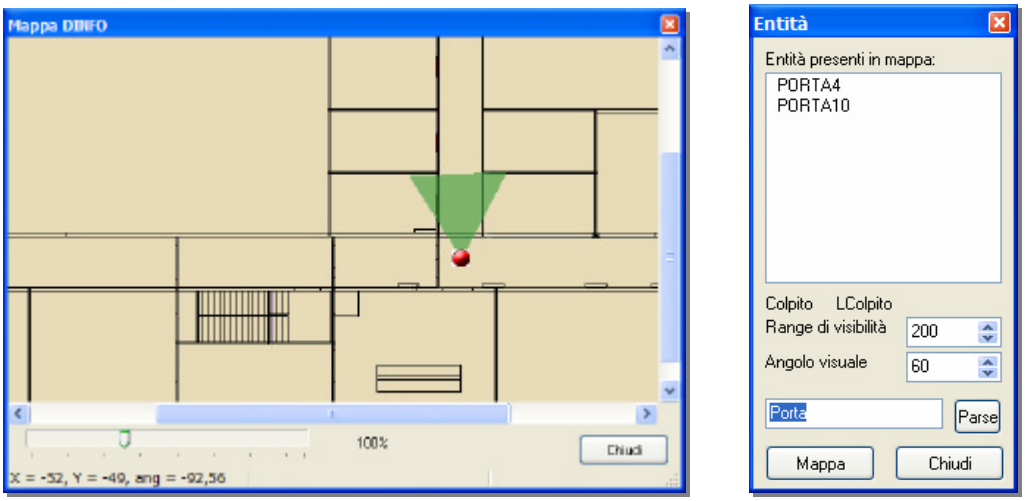


Figura 6-3 Riconoscimento di due porte

Come si può vedere, all'interno della visuale del robot ricadono due porte le quali sono state riconosciute ed elencate dal programma.

Nella finestra delle Entità è possibile cambiare il "*Range di visibilità*" e l'angolo di visuale che consentono di modificare la distanza di visibilità, ovvero quanto lontano il programma deve riconoscere gli oggetti e l'angolo della visuale stessa.

Si possono riconoscere altri oggetti nella mappa, infatti, basta indicare il nome dell'oggetto e premere il pulsante "*Parse*" con il quale si effettua una scansione del file della mappa che si sta usando. Per modificare il file dello scenario bisogna premere il pulsante "*Mappa*".

Ovviamente al momento della creazione di uno scenario si mette un nome all'oggetto che dopo viene riconosciuto dal programma attraverso l'analizzatore.

Un esempio di oggetto che viene riconosciuto è il seguente:

```
  ; Entity 15
  ; Map structure:g[2] -> PorteD:g[1] -> func_door_rotating:b[2]
  {
   "classname" "func_door_rotating"
   "distance" "90"
   "origin" "-12 100"
   "sounds" "1"
   "dmg" "0"
  ; Brush 0
  ; Map structure:g[2] -> PorteD:g[1] -> func_door_rotating:b[2]
-> Porta1:p[1]
   {
    ( -12 56 -44 ) ( -12 56 158.27161 ) ( -12 102.93333 -44 )
OAKPLY1 -153 -28 0 0.36667 1.58025 ;TX1
  …
    ( -14 100 -44 ) ( -14 30.46914 -44 ) ( -11.86666 100 -44 )
OAKPLY1 840 -184 0 0.01667 -0.54321 ;TX1
   }
  }
```

Questo è una parte del codice generato dal programma Quark relativo ad una porta. Il codice verrà poi compilato creando un file bsp, ovvero un file che contiene molte informazioni pre–calcolate per effettuare una rappresentazione veloce durante l'utilizzo del motore 3D.

MODIFICA DELLE IMMAGINI

Il programma consente anche di poter modificare le immagini, infatti, una volta acquisite le immagini reali del robot e quelle create dal motore 3D, bisogna fare un "confronto" fra le due in modo da poter distinguere degli oggetti o poter individuare la posizione del robot per mezzo delle immagini stesse.

La sezione relativa alla modifica delle immagini è dotata di alcune funzionalità di base:

- Correzione della luminosità e contrasto;
- Taglio dei bordi dell'immagine stessa;
- Gestione di più layer contemporaneamente, in quanto è possibile gestire più facilmente la sovrapposizione delle immagini stesse;
- Conversione in grigio dell'immagine;
- E' possibile effettuare delle operazioni di soglia sull'immagine;
- Inversione dei colori dell'immagine;
- Applicare il filtro di Mediana;
- Applicare il filtro di Harris
- Applicare un filtro di Edge Detection;
- Possibilità di fare la convoluzione dell'immagine con una matrice qualsiasi;
- Visualizzazione degli istogrammi del canale Rosso, Verde, Blue, nonché quelli H, S, L;
- Per ogni layer è possibile impostare la propria opacità e lo spostamento rispetto all'immagine di sfondo;
- Possibilità di salvare ogni singola immagine del layer su file;
- Possibilità di fare lo zoom sull'immagine stessa;

La sezione relativa alle immagini si presenta all'utente finale con questa finestra:

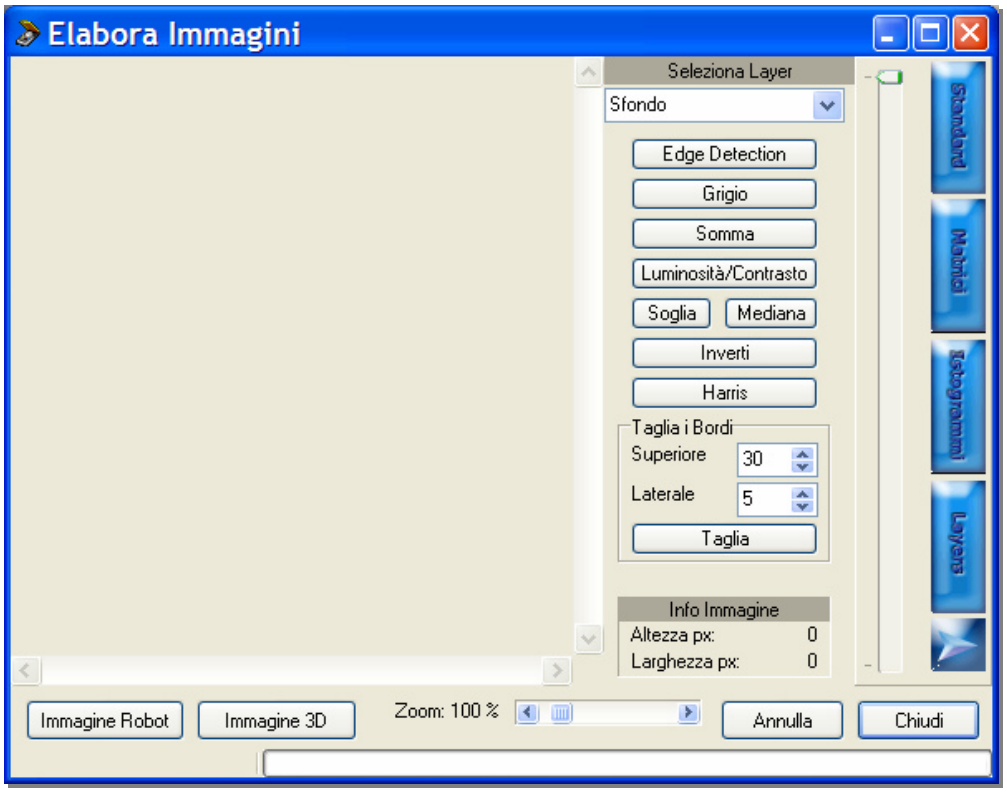


Figura 6-4 Finestra di elaborazione delle immagini

Come si può notare dall'interfaccia, è possibile caricare l'immagine visualizzata dal robot e quella creata dal motore 3D, tramite gli appositi pulsanti posti in basso a sinistra. Sempre in basso si notano i pulsanti per uscire, quello per annullare un'operazione e la barra per effettuare lo zoom.

All'estrema destra ci sono dei pulsanti che consentono di selezionare le categorie di operazioni che si possono effettuare alle immagini. Premendo il pulsante *Standard* compare il pannello visualizzato in Figura 6-4, nella stessa figura si può notare una barra verticale che consente di stabilire fino a dove bisogna applicare il filtro che si vuole eseguire.

Una volta selezionato il layer si possono effettuare varie operazioni, ad esempio l'Edge Detection, ovvero dopo che l'immagine è stata convertita in grigio, filtrata con la matrice di Prewitt [17] per determinare i punti orizzontali, si fa la somma con

l'immagine filtrata con la matrice di Prewitt per i punti verticali; il risultato è un'immagine che mostra i contorni della scena:

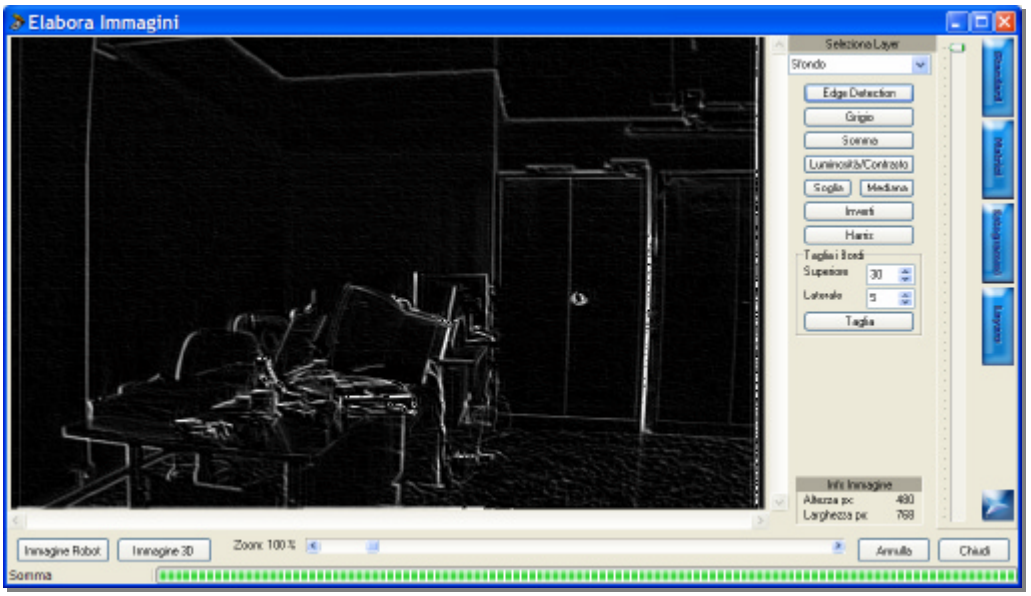


Figura 6-5 Applicazione del filtro Edge Detection

Gli altri pulsanti consentono di convertire l'immagine in scala di grigio e di effettuare la somma fra i due layer visibili.

Premendo il pulsante *Luminosità/Contrasto* viene visualizzata la seguente finestra:

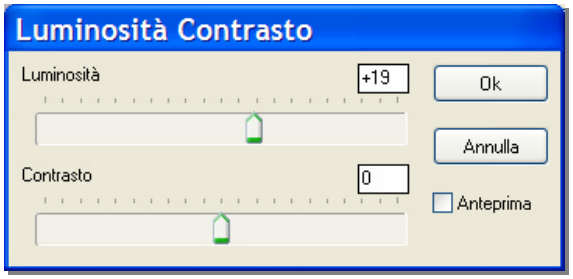


Figura 6-6 Modifica della Luminosità e del Contrasto

con la quale è possibile regolare la luminosità e il contrasto dell'immagine, inoltre selezionando la casella *Anteprima*, si può vedere il risultato sull'immagine stessa.

Un'analoga finestra viene visualizzata quando si preme il pulsante *Soglia*:

Figura 6-7 Modifica dei valori del filtro Soglia

Anche in questo caso è possibile variare il limite superiore dei tre canali di colore oltre il quale si ha lo stesso valore, ovvero 255. Selezionando la casella *Blocca* le tre barre assumono lo stesso valore. Premendo il pulsante *Grigio* l'immagine viene convertita in scala di grigi e si lavora sul solo canale della luminosità.

Gli altri pulsanti consento di effettuare il filtro Mediana all'immagine. Il filtro, che appartiene ad una classe chiamata order–statistics [17], non fa altro che considerare un intorno di 3 x 3 pixel e sostituire al pixel centrale la media dei nove pixel, in modo da eliminare il rumore che può esserci nell'immagine. Per evitare degli eventuali spike isolati di intensità, al posto della media aritmetica dei nove pixel, si sostituisce con il valore dei pixel che si avvicina di più alla media.

Il pulsante inverti non fa altro che invertire i colori dell'immagine tramite la formula: 255 – componente colore.

La sezione relativa ai bordi consente di eliminare eventuali bordi da un'immagine, questo perché a volte alcune videocamere creano dei bordi scuri nell'immagine stessa.

Il filtro di Harris serve ad individuare gli eventuali punti spigolosi che ci sono nell'immagine. Il filtro è descritto con il seguente pseudocodice di matlab:

Indicando con dx una maschera derivativa del tipo:

```
-1 0 1
-1 0 1
-1 0 1
```

mentre con dy indichiamo la sua trasposta, otteniamo due immagini Ix ed Iy:

```
Ix = convolvi(immagine, dx);
Iy = convolvi(immagine, dy);
```

Generiamo un filtro Gaussiano di ampiezza il minimo fra 1 e `6 * sigma (+/- 3 * sigma)` e lo indichiamo con g.

Calcoliamo le immagini Ix2, Iy2 e Ixy dalle convoluzioni di g con (Ix * Ix), (Iy * Iy), e (Ix * Iy)

```
Ix2 = convolvi(Ix * Ix, g);
Iy2 = convolvi(Iy * Iy, g);
Ixy = convolvi(Ix * Iy, g);
```

Si calcola la misura dei punti angolosi di Harris con il metodo di Nobel, in modo da eliminare un parametro.

```
cim = (Ix2 * Iy2 – Ixy * Ixy) / (Ix2 + Iy2 + eps);
```

Con una dilatazione morfologica in scala di grigio si estraggono i massimi locali, quindi si trovano i punti che combaciano con l'immagine dilatata e inoltre sono più grandi della sogliatura.

```
sze = 2 * radium + 1;          % Dimensione della maschera.
mx  = ordfilt2(cim, sze^2, ones(sze));  % Dilatazione.
cim = (cim == mx) & (cim > thresh);     % Trova i massimi.
[r,c] = find(cim);  % Trova le coordinate di riga e colonna.
```

Premendo il pulsante *Harris* compare la seguente finestra:

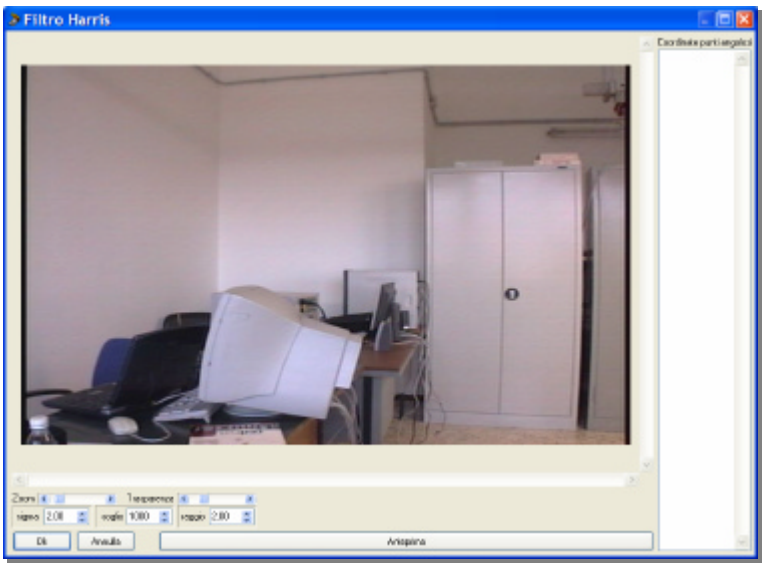


Figura 6-8 Finestra del filtro Harris

effettuando un'anteprima dopo aver scelto una Deviazione Standard Gaussiana, un valore di sogliatura, e un raggio otteniamo un immagine con i punti spigolosi ottenuti, i quali sono elencati alla destra della finestra. La trasparenza dell'immagine con i punti spigolosi può essere modificata, in modo da poter visualizzare meglio i punti stessi.

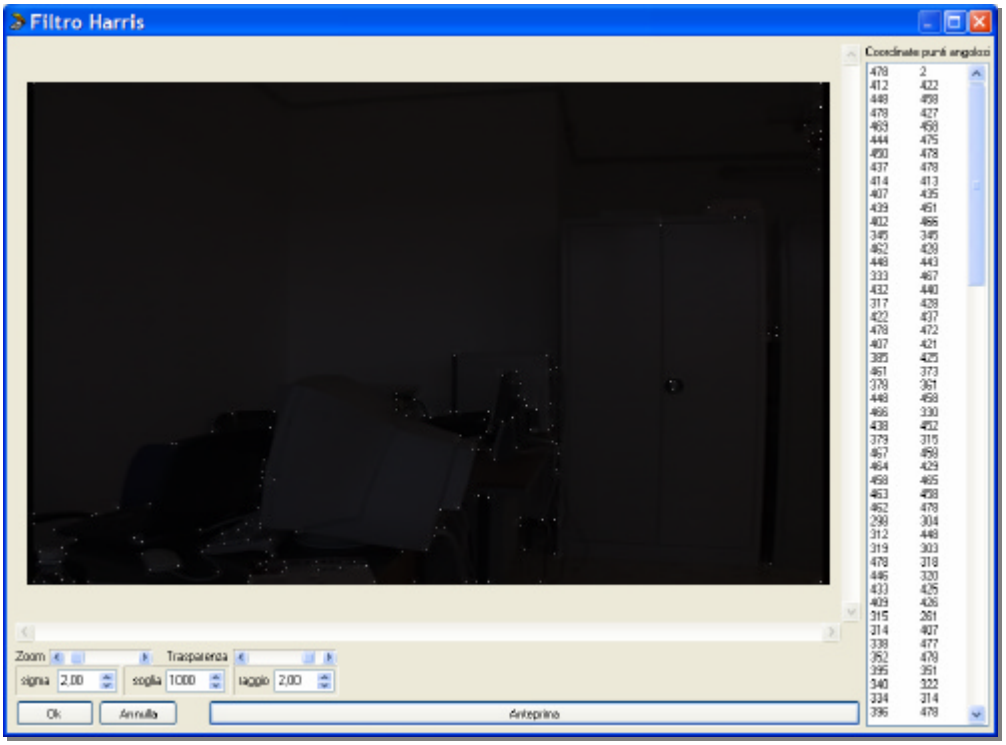


Figura 6-9 Risultato del filtro Harris

Sezione relativa alle operazioni di convoluzione:

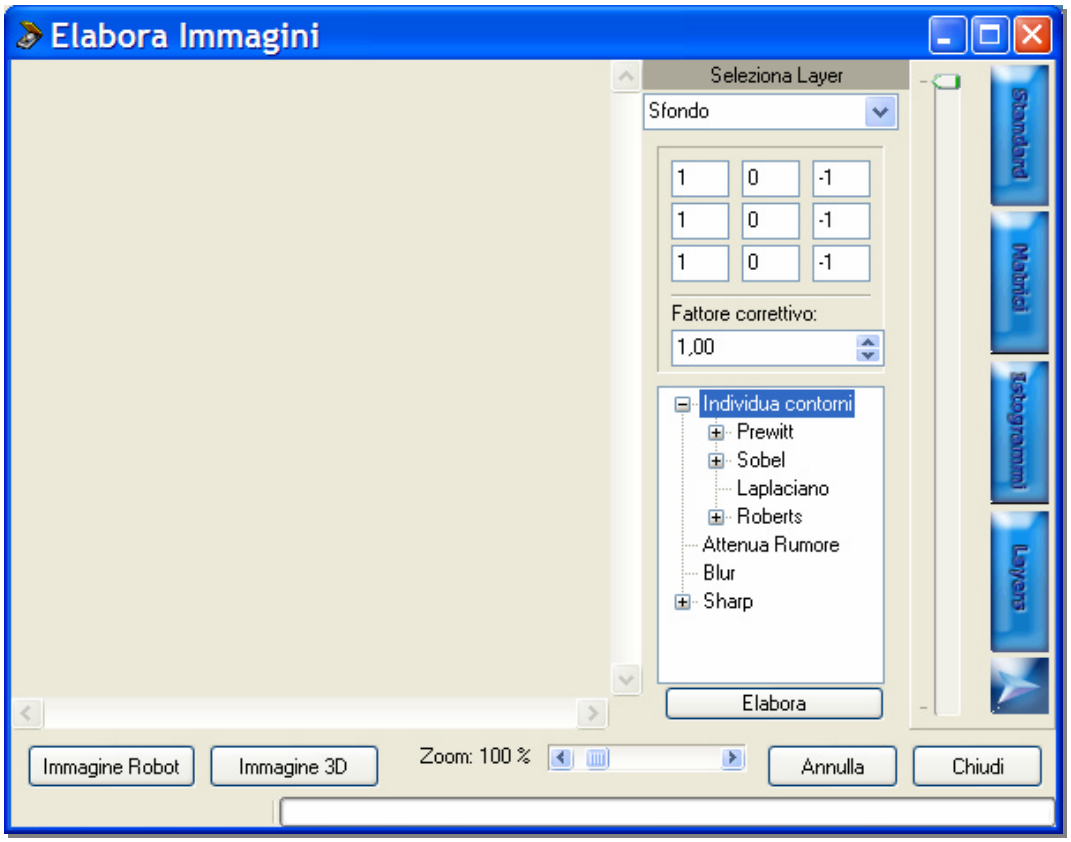


Figura 6-10 Finestra della matrice di convoluzione

Premendo il pulsante *Matrici* viene visualizzato il seguente pannello dove è possibile selezionare una matrice di convoluzione da un elenco prestabilito, oppure crearne una personale modificando i vari elementi della matrice. I valori che si possono immettere sono degli interi, per ragioni di velocità di elaborazione, ma è possibile usare un fattore correttivo che viene premoltiplicato alla matrice.

Per applicare il filtro di convoluzione, dopo che è stato impostato, basta premere il pulsante *Elabora.*

In basso verrà visualizzato lo stato di avanzamento della convoluzione, inoltre, è possibile ritornare all'immagine precedente premendo il tasto *Annulla.*

I filtri reimpostati sono raggruppati in Edge Detection che hanno la funzione di individuare i contorni tramite le matrici di Prewitt [17], Sobel [17], Roberts [17] o con il metodo del Laplaciano [17].

Il filtro di Blur fa la media dei valori dei pixel nell'intorno definito dalla maschera, in modo da ridurre l'entità delle differenze di grigio tra i punti vicini. In questo caso si è usata una matrice tre per tre formata da uno e premoltiplicata per 1/9.

I filtri di Sharp al contrario di quelli di Blur, servono ad accentuare i dettagli di un'immagine; bisogna fare attenzione che questi filtri oltre ai dettagli accentuano anche il rumore che può esserci in un'immagine. Le matrici di default sono due ed entrambi sono basati sul Laplaciano. Una matrice di convoluzione elimina le informazioni di sfondo ed è costruita in modo da rendere isotropo il laplaciano per rotazioni di 45° prendendo in considerazioni le derivate seconde lungo le due direzioni diagonali. L'altra matrice consente di effettuare un ripristino delle informazioni di sfondo, infatti, si somma l'immagine di partenza all'uscita del laplaciano.

Sezione relativa alla visualizzazione degli istogrammi:

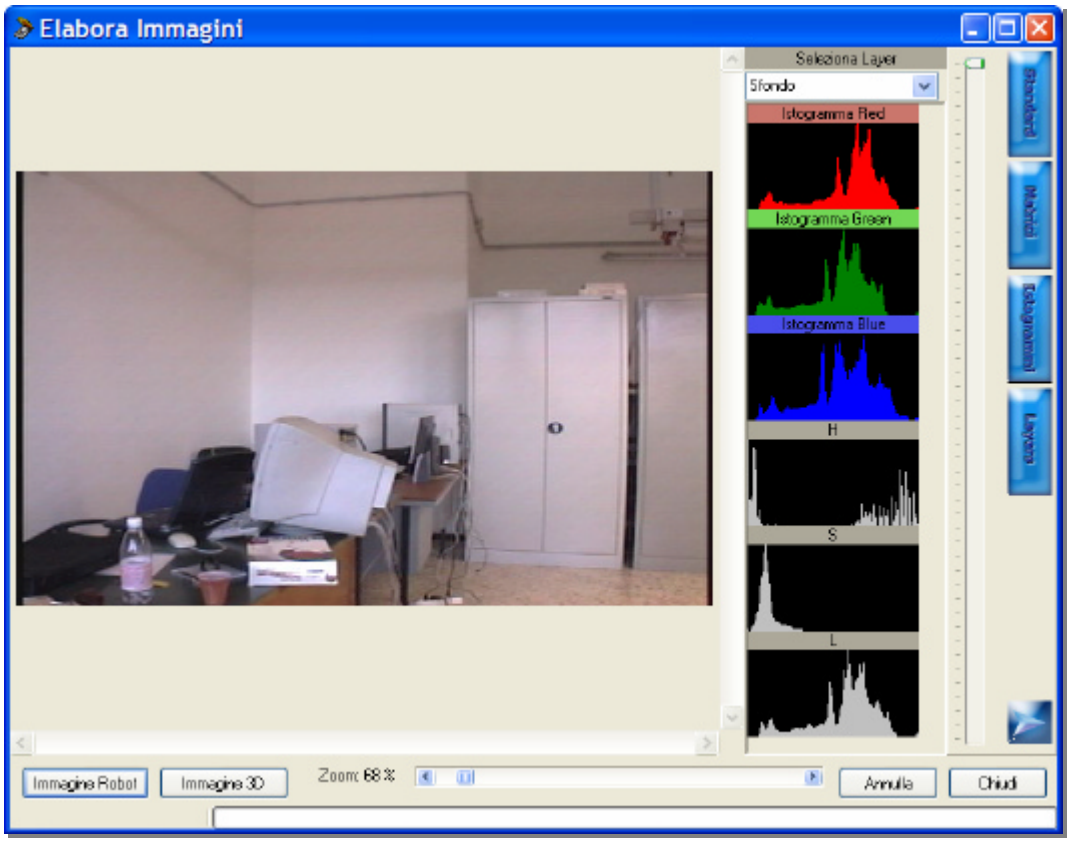


Figura 6-11 Finestra di visualizzazione degli istogrammi

Questa finestra consente di visualizzare i vari istogrammi di un'immagine, infatti, si possono vedere quelli relativi alle tre componenti del colore, e quelli relativi alla rappresentazione HSL.

Per ottenere ogni diagramma si scorrono tutti i pixel dell'immagine con due cicli e si incrementa il valore della posizione del vettore relativa al colore stesso:

```
pixel   = LayerSelezionato()->Pixel[i][j];
ValoreR = RedComponent(pixel);
CompoR[ValoreR]++;
ValoreG = GreenComponent(pixel);
CompoG[ValoreG]++;
ValoreB = BlueComponent(pixel);
CompoB[ValoreB]++;
```

Un analogo ragionamento viene effettuato per la rappresentazione HSL.

ARCHITETTURA DEL SISTEMA DI VISIONE DEL ROBOT RWI-B21R

Il sistema di visione usato nel robot è un sistema distribuito su più macchine che comunicano fra di loro tramite socket su protocollo TCP/IP. In Figura 6-12 si possono vedere le macchine che interagiscono nel sistema, come si può notare sono presenti due notebook, di cui uno sopra il robot e il calcolatore interno del robot stesso.

Il calcolatore interno esegue dei programmi che si interfacciano con l'hardware e la parte del sistema ad agenti che si occupa del movimento. Il notebook remoto ha la funzione di far partire il software che c'è sul robot, tramite connessione telnet, inoltre esegue la restante parte del sistema ad agenti che si occupa della pianificazione del percorso. Il rimanente notebook sopra il robot ha la funzione di interfacciarsi con l'utente, mostrando a video le possibili visite che si possono fare ed inoltre è possibile visualizzare il flusso video di una telecamera del robot, in modo da poter vedere quello che il robot indica. Il notebook posto sopra il robot dato che gestisce il flusso video delle telecamere, si occupa anche della localizzazione e della gestione del pantilt. La gestione del parlato, tramite sintetizzatore vocale o file registrato, viene gestito da un altro programma che viene eseguito anche sul notebook sopra il robot.

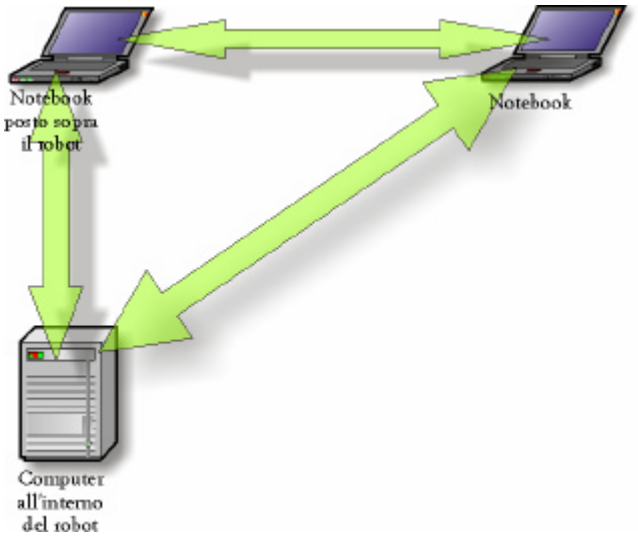


Figura 6-12 Architettura dei computer usati durante gli esperimenti

DRIVER PANTILT

Per poter comandare da remoto il pantilt è necessario creare un server TCP che si interfacci con l'hardware. Il server accetta una singola connessione per volta e nel nostro esperimento, si instaura una connessione all'avvio dei programmi e si chiude alla fine. Il protocollo è molto semplice, infatti, il client e il server si scambiano delle normali stringhe di testo, dopo queste stringhe vengono interpretate dal server per eseguire il corrispettivo comando. La stringa contiene tre interi, di cui il primo indica il comando e i rimanenti due, nel solo caso di movimento, vengono interpretati come angolo pan e angolo tilt. Nella tabella è possibile vedere l'intero protocollo:

Stringa	Comando
-1 x x	chiude il server
0 x x	reset e calibrazione del pantilt
1 x x	posizione il pantilt al centro
2 alfapan alfatilt	muove il pantilt di un angolo alfapan sul piano orizzontale e un angolo alfatilt sul piano verticale

Tabella 2 Comandi server PanTilt

L'unità pantilt può muoversi soltanto di 154.20° nella direzione orizzontale a destra e a sinistra, in alto di 30.84° e in basso di 46.26°, inoltre i comandi all'hardware vengono impartiti in unità numeriche, e ogni unità vale 0.0514°, quindi i limiti dei valori del pan sono da –3000 a 3000, mentre per il tilt i valori vanno da –600 a 900.

Il server utilizza la porta TCP 1666 e resta in ascolto su tale porta finchè un client non si colleghi e gli invii dei comandi. I comandi vengono inviati direttamente alla porta */dev/ttyR1* dopo che sono stati filtrati da valori errati. Prima di inviare un qualsiasi comando, bisogna inizializzare il pantilt tramite il comando interno al driver *pt_init.* Ogni volta che si è inviato un comando, bisogna mettere in attesa il pantilt con il comando interno *pt_wait.*

PROGRAMMA DI VISIONE

Il programma di visualizzazione che risiede sul notebook sopra il robot, consente all'utente di scegliere una visita da effettuare e visualizzare il flusso video di una telecamera; inoltre lo stesso programma si occupa della localizzazione. Come si può vedere in Figura 6-13 l'utente sceglie una visita da effettuare premendo uno dei pulsanti: Visita 1, Visita 2, Visita 3.

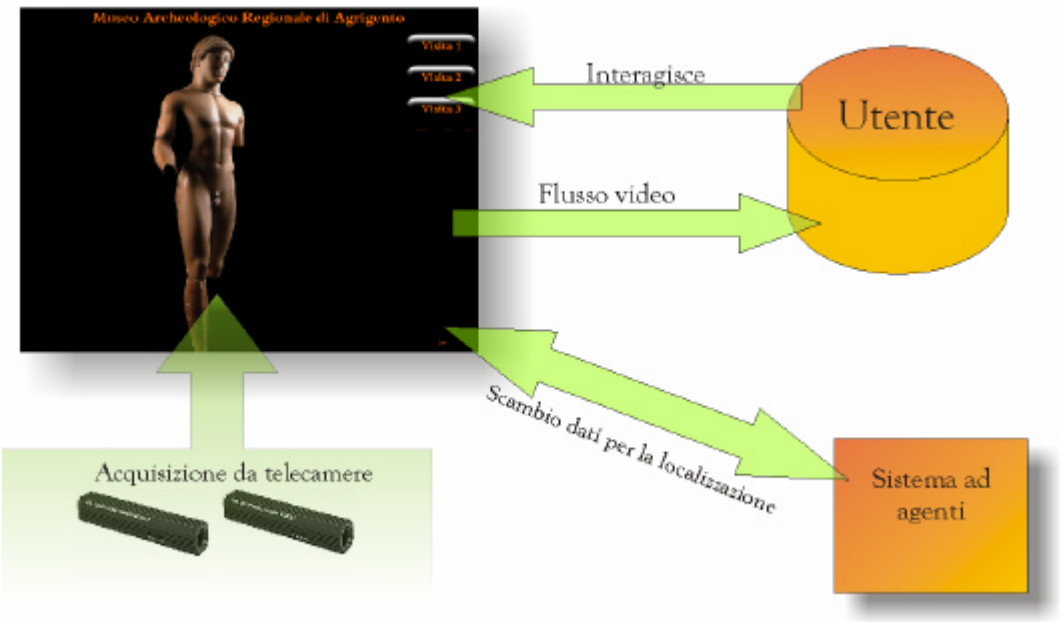


Figura 6-13 Architettura del Programma di Visione

Dopo aver scelto la visita il robot la inizia e il programma mostra il flusso video di una telecamera, inoltre, quando il sistema ad agenti decide che bisogna localizzarsi, questo, tramite protocollo TCP, comunica al programma di visualizzazione il comando da effettuare e la posizione attuale. Il protocollo di comunicazione è molto simile a quella usata per il pantilt, inoltre questo programma è il client che comanda il server del pantilt. La stringa che il sistema ad agenti comunica al programma è formata da interi e stringhe di testo, il primo intero indica il comando da eseguire.

Nell'interfaccia sono anche presenti dei pulsanti di debug, che consentono di visualizzare delle informazioni utili durante le nostre prove, come ad esempio la misura della distanza dal marker, o le coordinate calcolate dalla localizzazione. Nella

finestra di debug è inoltre possibile vedere il flusso video di entrambe le telecamere, oltre che poter forzare manualmente alcuni valori. Tali informazioni sono state fondamentali per migliorare il sistema di localizzazione.

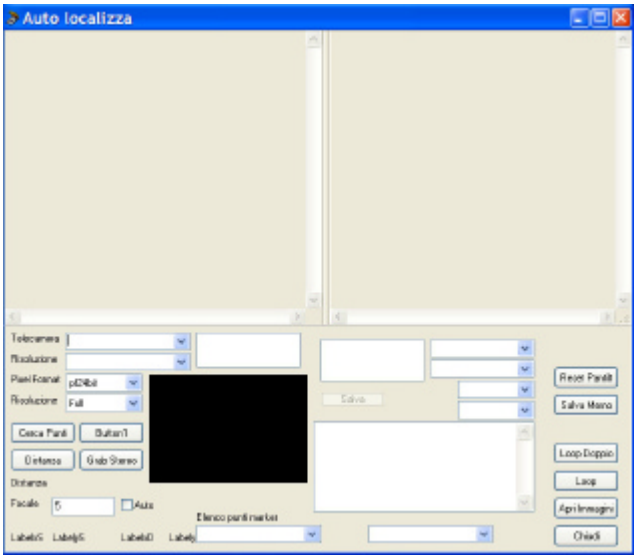


Figura 6-14 Finestra di Debug del programma di Visione

Nella tabella seguente sono elencati i comandi del programma di visione:

Stringa	Comando
0 posx posy alfa	Effettua una localizzazione partendo dalle coordinate *posx*, *posy* e angolo *alfa*, questo comando ritorna le posizioni aggiornate
1 alfa x x	Muove il pantilt di un certo angolo *alfa* per poi ritornare dopo qualche secondo alla posizione centrale
2 alfa x x	Comando non più utilizzato
3 alfa x x	Muove il pantilt di un angolo *alfa* senza ritornare
4 alfa nomefile	Muove il pantilt di un angolo alfa ed esegue il file audio *nomefile*

Tabella 3 Comandi programmi localizzazione

A causa del grosso carico di calcolo che si ha quando si visualizza il flusso video, è consigliabile avere un calcolatore adeguato, infatti, per evitare calcoli inutili, l'altra telecamera viene usata soltanto quando si effettua una localizzazione. Tenendo

conto che durante una localizzazione il robot sta fermo non ci sono grossi problemi di sincronismo fra di due frame.

PROGRAMMA DI SINTESI VOCALE

Il programma di sintesi vocale è un server che accetta dei comandi tramite connessione TCP. La funzione di tale programma è quella di "leggere" dei file di testo, quando viene inviato l'apposito comando; in questo modo il robot può far conoscere all'utente le informazioni su una determinata vetrina o oggetto. Il programma gestisce anche dei file audio pre–registrati, in modo da poter sfruttare sia la funzionalità della lettura dei file che quella di eseguire dei file registrati da voce umana o dal sintetizzatore stesso. La funzione di eseguire i file pre–registrati ha il grande vantaggio di non appesantire il carico della CPU.

Un'altra funzionalità è quella di eseguire un file audio che emette il suono di un clacson, infatti, questa necessità si ha quando il robot non ha spazio libero davanti a se per proseguire nel suo percorso e in questo modo fa capire alle persone poste davanti a lui di spostarsi.

Ci sono altre funzionalità non usate durante l'esperimento, ma potrebbero essere utili successivamente; ad esempio si può mettere in pausa una spiegazione e riprenderla successivamente.

Trattandosi di un vero e proprio server che elabora i comandi inviati da remoto, il programma non ha un'interfaccia grafica particolare, se non quella di visualizzare un pulsante per chiudere il programma stesso.