UNIVERSITÀ DEGLI STUDI DI PARMA
FACOLTÀ DI INGEGNERIA
Corso di Laurea in Ingegneria Informatica

PROGETTAZIONE E REALIZZAZIONE DI UN SISTEMA DI ASSERVIMENTO VISIVO PER UN ROBOT MANIPOLATORE

Relatore:
Chiar.mo Prof. STEFANO CASELLI

Correlatori:
Ing. JACOPO ALEOTTI

Tesi di laurea di:
ALESSIO NIZZOLI

ANNO ACCADEMICO 2005-2006

Indice

Elenco delle figure

Elenco delle tabelle

Alla mia famiglia

“Sono convinto che l’informatica abbia molto in comune
con la fisica. Entrambe si occupano di come funziona il
mondo a un livello abbastanza fondamentale. La differenza,
naturalmente, è che mentre in fisica devi capire come
è fatto il mondo, in informatica sei tu a crearlo.
Dentro i confini del computer, sei tu il creatore.
Controlli, almeno potenzialmente, tutto ciò che vi succede.
Se sei abbastanza bravo, puoi essere un dio. Su piccola scala.”

Linus Torvalds

Capitolo 1

Introduzione

1.1 Premessa

La robotica è una branca relativamente giovane dell'ingegneria che coinvolge diversi settori come la cinematica/dinamica dei sistemi, la teoria del controllo, l'informatica e l'intelligenza e la visione artificiale.
Fino agli anni '80 le applicazioni della robotica erano confinate all'industria meccanica, dove i robot compiono azioni ripetitive in ambienti strutturati, cioè le cui caratteristiche sono note a priori, e costruiti appositamente secondo le esigenze del robot. Grazie all'abbattimento dei costi, l'affinamento delle tecniche di controllo, e l'enorme progresso dei calcolatori elettronici, la robotica trova ora molteplici applicazioni in ambiti diversi. Tra essi la medicina, per la creazione di protesi che eseguano movimenti il più vicino possibili a quelli naturali; la chimica, dove algoritmi *path-finding* vengono sfruttati per la creazione di molecole complesse; la vita domestica, dove robot che puliscono pavimenti o tagliano l'erba dei prati stanno diventando sempre più comuni; persino nel settore ludico diversi produttori di giocattoli forniscono piccoli robot con semplici comportamenti programmabili.
La medicina moderna ha di fatto allungato l'età media della popolazione, per cui malattie degenerative ed invalidanti sono sempre più comuni nella popolazione anziana; soprattutto in Europa ed in Giappone, l'invecchiamento della popolazione e la necessità di servizi di assistenza rappresentano una vera e propria sfida sociale. Anche in questo settore la robotica trova un possibile campo di applicazione,

nella costruzione di sistemi di supporto per persone con difficoltà motorie, o impedimenti fisici. Una delle maggiori difficoltà nella progettazione dei sistemi robotici risiede nella creazione di sistemi facilmente usufruibili da persone che non abbiano una formazione ingegneristica o scientifica, e nel dotare tali sistemi di un certo grado di autonomia, ossia nel non dipendere totalmente da comandi umani per lo svolgimento di azioni. Mentre la ricerca più avanzata si concentra sullo studio di interfacce neurali, in cui il cervello è direttamente connesso al sistema robotico, una possibile alternativa è la creazione di sistemi di controllo basati su realtà virtuale.

Nel sistema proposto in fig.1.1 una persona tramite personal computer imparti-

Figura 1.1: Sistema robotico per l'assistenza.

sce ordini ad un sistema robotico composto da una base mobile, su cui è installato un manipolatore. Nell'ambito del progetto di ricerca Lareer il Dipartimento di Ingegneria dell'Informazione mira alla realizzazione di un sistema robotico mobile in grado di compiere compiti di manipolazione, identificazione, prelievo e consegna di oggetti. Il Software PbD (*Programming by Demonstration*), creato presso il Laboratorio di Robotica dell'Università di Parma [1], si propone come sistema di interazione uomo-robot basato su realtà virtuale. Il software simula un ambiente reale in cui è presente un robot mobile in grado di eseguire compiti di manipolazione: l'utente imposta nel mondo virtuale il compito che il robot dovrà eseguire

specificando, la posizione attuale del robot, il luogo dove dovrà svolgersi il compito, ed il tipo di compito, ad esempio il recupero di un oggetto. Il software provvede a simulare il robot nell'esecuzione dell'azione: lo spostamento del robot, la ricerca dell'oggetto, e la sua presa. È importante specificare che l'utente assegna un compito al sistema robotico, ma non specifica come poi questo debba essere eseguito; è quindi necessario che il robot abbia un certo grado di autonomia. Per questo il robot avrà bisogno di un qualche tipo di "senso" che gli permetta di avere un riscontro sia sull'ambiente che sul proprio operato. L'estensione naturale di questo progetto è il passaggio dalla simulazione all'esecuzione reale di un compito di manipolazione mobile.

L'obbiettivo di questa tesi è lo sviluppo di un'architettura visual servoing per un robot manipolatore per compiti di servizio. Il sistema proposto si compone del robot Nomad 200 come base mobile e del manipolatore Manus, entrambi in dotazione del Laboratorio di Robotica ed utilizza il software di simulazione come interfaccia utente, attraverso la quale si impostano le operazioni richieste dal robot.

1.2 Il ruolo del Visual Servoing

La maggior parte dei robot, oggi, opera in ambienti *strutturati*, cioè direttamente controllabili ed adattatabili alle esigenze del robot stesso, come ad esempio le fabbriche. Al contrario, in ambienti dinamici, o in ogni caso di cui si conosce poco a priori e su cui non si può intervenire, l'applicazione di sistemi robotici è poco diffusa a causa della limitazione delle capacità sensoriali da parte dei robot stessi.
La visione artificiale, imitando la visione umana, è uno strumento molto potente per la sensorialità di un robot, perché consente misurazioni dell'ambiente *passive*, cioè senza la necessità del contatto fisico. Tipicamente visione e manipolazione sono combinati in un ciclo aperto "guarda poi muovi"; una scelta per migliorare la precisione di questo sistema è realizzare un ciclo di comando dei movimenti del robot con retroazione visiva. In altri termini, la visione artificiale può fornire, in un ciclo chiuso di posizionamento, le informazioni necessarie al braccio robotico, formando un sistema che prende il nome di "visual servoing" (asservimento visivo).
Dal primo sistema di visual servoing (VS), sviluppato nei primi anni '80[2], i progressi nel controllo visuale dei robot è stato abbastanza lento, ma negli ultimi anni la potenza computazionale degli elaboratori ha oltrepassato la soglia che consente l'analisi della scena in tempi tali da permettere di comandare un robot manipolatore praticamente in tempo reale. Nel 1980 Sanderson e Weiss[3] introdussero una tassonomia con la quale è possibile classificare tutti i sistemi VS. Il loro schema pone essenzialmente due domande:

1. La struttura di controllo è gerarchica, con la visione che fornisce l'ingresso al controllore del robot, o è direttamente la visione a comandare il robot a livello dei giunti?

2. L'errore di posizionamento è definito nello spazio di lavoro del robot (vale a dire in un ambiente tridimensionale), o in termini di caratteristica dell'immagine acquisita?

Le risposte ad ognuna di queste domande sono due, e combinate tra loro generano quattro tipi di paradigmi VS, che si differenziano a seconda del tipo di controllo che si opera sul robot, ed in base alle informazioni che si ottengono dall'algoritmo di

		Robot Control	
		dynamic look and move	*direct visual servo*
Information from Vision	*position based*	I	II
	image based	III	IV

Tabella 1.1: Tassonomia di Sanderson-Weiss per i sistemi di visual servoing

visione.
I quattro paradigmi sono i seguenti:

- **dynamic look and move.** La visione fornisce i punti nel quale il manipolatore si deve portare, ed il controllore aziona i giunti in modo da assumere la configurazione necessaria per il raggiungimento di tali punti.
- **direct visual servo.** Il controllore è eliminato poiché il sistema visivo controlla direttamente i giunti.
- **position-based.** Le informazioni estratte dall'immagine (*features*) vengono usate per ricostruire la posa (posizione/orientazione.) 3D corrente dell'oggetto rispetto alla telecamera. Si genera un segnale di errore "cartesiano" dalla differenza tra la posa 3D desiderata e quella attuale.
- **image-based.** L'errore viene calcolato in 2D nel piano immagine, senza stimare la posizione degli oggetti esaminati. Il robot si muove in modo da portare le *features* 2D correnti osservate verso i loro valori desiderati.

Quasi tutti i sistemi adottano il primo tipo d'architettura in quanto più semplice, visto che assicurare l'elaborazione d'immagini e il contemporaneo controllo della cinematica del manipolatore è un compito complesso sia dal punto di vista computazionale che del controllo. Inoltre molti manipolatori sono già forniti di controllori programmabili che si occupano della cinematica, sia nello spazio dei giunti sia nello spazio cartesiano; in particolare la capacità di movimento nello spazio cartesiano facilita di molto la costruzione di sistemi VS visto che è possibile relazionare i dati provenienti dalla telecamera con gli spostamenti del robot una volta conosciuto l'orientamento relativo dei due sistemi di riferimento.
Per quanto riguarda la parte di visione, con entrambe le tipologie, è neccessario

avere delle conoscenze a priori sull'oggetto di interesse, ovvero sul "target" verso al quale viene riferito il compito di manipolazione.
Nel caso in cui la telecamera inquadri il bersaglio e il manipolatore (o solo il suo organo terminale) si parla di sistemi "Eye to Hand", che sono caratterizzati da una telecamera che mantiene una posizione fissa o al più con orientamento variabile. Nei sistemi "Eye in Hand" la telecamera è fissata sul manipolatore, tipicamente sulla pinza, e tramite il movimento del robot si cerca di mantenere il bersaglio nella visuale della telecamera. Una variazione a questi modelli include il posizionamento di telecamere multiple, con un telecamera montata sul braccio che "aggancia" il bersaglio, ed una telecamera che guarda sia l'obbiettivo sia la pinza del robot. L'uso di telecamere multiple porta soluzioni ibride che compensano i difetti delle due tipologie, ma oltre ad avere un costo più elevato, presentano anche problemi di gestione e di elaborazione di doppie immagini. Per questa tesi verrà implementato un modello *dynamic look and move*, e *position based*, *eye in hand*. Nella figura 1.2 è illustrato

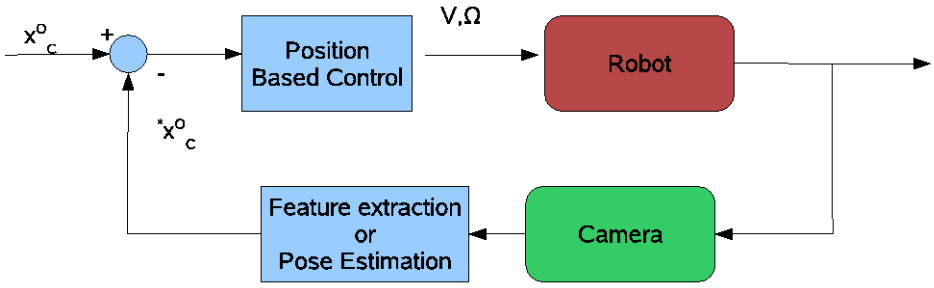


Figura 1.2: Modello Visual Servoing *dynamic look and move* and *position based*

un esempio di architettura *visual servoing*: si noti che il controllo del robot avviene nello spazio di lavoro (V rappresenta la velocità lungo le tre direzioni x,y,z mentre Ω rappresenta l'orientazione). Il controllo poi viene confrontato col *feedback* visivo. Dalla immagine acquisita vengono estratte delle caratteristiche che, unite alle conoscenze a priori del sistema, servono per il calcolo della posizione relativa. Da notare che l'architettura non specifica quante telecamere vengono impiegate e come esse siano collocate (fisse o sul robot).

1.3 Alcuni progetti significativi di visual servoing

Il metodo di controllo di un robot con feedback visuale è noto da anni in letteratura, e nel corso del tempo si è via via affinato. Pertanto, i modelli sviluppati oggi sono capaci di grande robustezza ed affidabilità. Nell'ambito della ricerca, il *visual servoing*, e più in generale il controllo di robot mediante visione, sono oggi di grande interesse, in quanto in grado di produrre, con i giusti investimenti di studio e denaro, risultati non solo applicabili all'industria, ma anche ad ambiti "civili". Questo paragrafo presenta una panoramica di alcuni interessanti progetti di ricerca in cui il VS riveste un ruolo importante.

UJI Librarian Robot

Il laboratorio di Robotica Intelligente dell'Università Jaume I in Castellon in Spagna ha realizzato un sistema robotico mobile pensato per compiti di recupero libri all'interno di un archivio o biblioteca [4].

Figura 1.3: Il robot bibliotecario UJI.

Il sistema, mostrato in fig. 1.3, si compone di:

Figura 1.4: Processo di visione.

- Robot mobile ActivMedia PowerBot [5], un robot mobile di dimensioni medio piccole in grado di trasportare carichi anche pesanti (fino a 100Kg), dotato di sensori sonar, bumper, e laser (opzionale). Dispone di due ruote mobili dotate di encoder per il calcolo della distanza percorsa, ed è alimentato da due batterie da 24 volt.

- Robot manipolatore PA-10 dotato di sei gradi di libertà (sei giunti rotoidali più il gripper). Il basso peso e l'alta capacità di movimento lo rendono ideale per lavori in ambienti semi strutturati, come nel caso di una biblioteca.

- Una telecamera stereo posta sul gripper e parallelamente ad esso

Il sistema proposto è in grado di effettuare il prelievo da uno scaffale di un determinato libro specificato dall'utente. L'utente si limita a specificare il volume scelto ed una sua posizione approssimativa, il sistema si porta nella posizione specficata ed inizia la ricerca del libro. Il robot acquisisce immagini dello scaffale con la telecamera, e mediante un sistema di visione artificiale basato su OCR, elabora le etichette dei volumi alla ricerca del testo specificato (come in fig.1.4).
Una volta che il volume è stato riconosciuto viene utilizzato l'algoritmo di visual servoing: dalla visione si ricavano le informazioni relative alla posizione del volume, e in base ad esse si comandano i giunti del manipolatore per la presa dell'ob-

Figura 1.5: Presa dell'oggetto.

biettivo (fig. 1.5).

Ishikawa Namiki Nomura Laboratory

L'Ishikawa Namiki Nomura Laboratory dell'Università di Tokio in Giappone ha realizzato un sistema robotico pensato esclusivamente per il compito di "grasping" mediante *feedback* visuale ad alta velocità [6].

Un braccio robotico dotato di sei gradi di libertà con giunti rotoidali e un organo

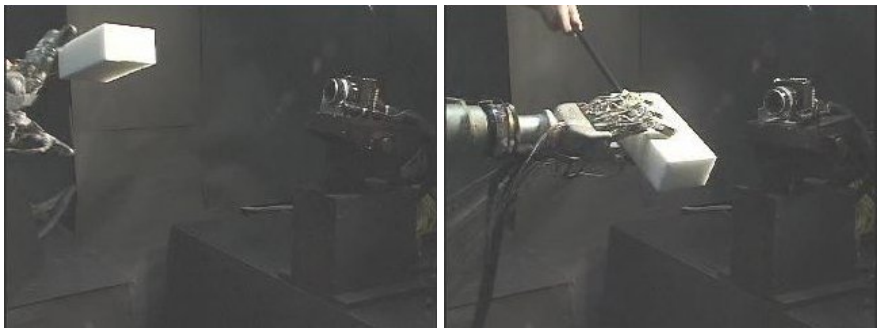


Figura 1.6: Inseguimento e cattura di oggetti

terminale a forma di mano dotato di quattro dita è comandato da un sistema di visione denominato SPE-256. L'apparato SPE-256 sviluppato dal laboratorio stesso si basa su un sistema integrato di una matrice 64x64 pixel, con un processore apposito per l'elaborazione di immagini in real-time.

In questo caso la soluzione adottata è quella di un sistema visual servoing, *eye*

Figura 1.7: Stretta di mano.

to hand: la telecamera in questo caso non è montata direttamente su manipolatore, ma di fronte ad esso; il manipolatore è fisso, mentre la videocamera è in gra-

do di cambiare il proprio orientamento in modo da seguire meglio l'oggetto. I filmati di esperimenti, visualizzabili in http://www.k2.t.u-tokyo.ac.jp/fusion/VisualFeedbackGrasping/index-e.html, di cui si ritrovano in fig.1.6 e fig.1.7 delle immagini, stupiscono, in particolare, per la precisione e l'elevata velocità con cui il sistema risponde agli stimoli visivi.

Grasping with a Humanoid Robot

Nicolas Mansard nell'ambito del gruppo di ricerca Lagadic [7] dell'Università di

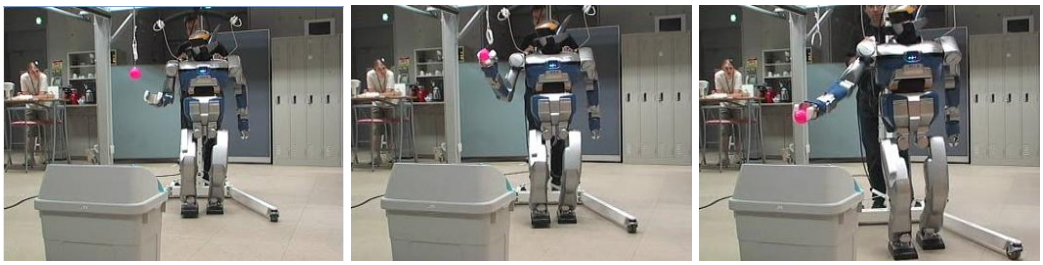


Figura 1.8: Lagadic Robot mentre cammina per prendere l'oggetto.

Figura 1.9: visuale del robot.

Beaulieu, ed in collaborazione con il Joint Robotic Laboratory, Tsukuba in Giappone, sta lavorando alla realizzazione di operazioni complesse con un robot umanoide, in particolare nella presa di oggetti in movimanto. L'hardware utilizzato è il robot HRP-2 dotato di gambe e braccia mobili e sensori vari, in particolare una telecamera posta nella testa del robot. Un algoritmo di visual servoing *position based, eye to hand* viene utilizzato per comandare il braccio del robot, nella presa dell'oggetto. Le figure 1.8 1.9 mostrano il robot in movimento mentre raccoglie una palla rossa, che dovrà poi essere gettata nel cestino.

Mobile Service Robotics

L'Institute for Man-Machine-Interaction della Aachen University [8] in Germania

Figura 1.10: Sistema di supporto per disabili basato su carrozzina, manipolatore e telecamera.

lavora alla realizzazione di sistemi per il supporto a persone disabili. Il sistema illustrato in fig.1.10 si compone di una carrozzina elettrica fornita da Invacare GmbH Deutschland, un manipolatore Manus, uno schermo tattile come interfaccia per l'utente, e diverse telecamere usate come sensori. In questo caso tecniche di visual servoing vengono usate per comandare il manipolatore in semplici task, come ad esempio in fig.1.11 per prendere una tazza . La telecamera stereo (posta sopra la

Figura 1.11: Compito di manipolazione su una tazza.

testa) viene usata per una misura indicativa di posizione e distanza dell'oggetto. Il

sistema tenta di ricostruire un modello tridimensionale dell'oggetto per determinare la presa ideale. Una telecamera montata sulla pinza provvede all'esplorazione dello spazio ed alla estrazione delle caratteristiche principali dell'oggetto (dato che una ricostruzione dettagliata della forma, non è necessaria, ma anzi appesantisce inutilmente il sistema).

1.4 Descrizione del problema

L'implementazione di un sistema robotico mobile per compiti di manipolazione è un problema la cui realizzazione può essere in parte facilitata attraverso una progettazione modulare del sistema.
Partendo dal livello più alto possibile di astrazione, è necessaria un'interfaccia attraverso la quale l'utente specifichi il compito che il robot deve assolvere. Tale interfaccia può essere implementata in vari modi: come riga di comando, come file, come metodi di una classe messe a disposizione in un linguaggio di programmazione ecc. Nella premessa si è però posta enfasi sul fatto che obbiettivo ideale di questi sistemi è il supporto e l'assistenza di persone con possibili difficoltà sia fisiche che mentali, e dunque i requisiti che tale interfaccia deve soddisfare sono:

- l'immediatezza, per cui l'utente capisce immediatamente ciò che il sistema è in grado di compiere;
- la facilità, per cui è immediato per l'utente specificare un compito.

Un'interfaccia basata su realtà virtuale ricostruisce in computer grafica una rappresentazione dell'ambiente, e soddisfa appieno entrambi i requisiti, soprattutto se fruibile attraverso dispositivi intuitivi come ad esempio un mouse.
Scendendo ad un livello inferiore di astrazione sono necessari almeno altri tre moduli fondamentali in modo da ottenere un'architettura del tipo schematizzata in fig.1.12.

È necessario un sistema di *navigazione* che consenta al sistema di muoversi nell'ambiente, per raggiungere il luogo dove avverrà poi il compito di manipolazione vero e proprio. Tale sistema deve conoscere la propria posizione nello spazio e la posizione finale, e dovrà poi, interfacciandosi a basso livello con la parte mobile del sistema robotico, gestire il movimento. È dunque necessaria una pianificazione della traiettoria che può essere svolta dal calcolatore in maniera completa o parziale specificando un numero limitato di punti. Si possono poi implementare dei comportamenti robotici imponendo delle direttive che il sistema cerca di seguire come ricerca/inseguimento di un cammino (*path finding/following*) e, se il robot è dotato

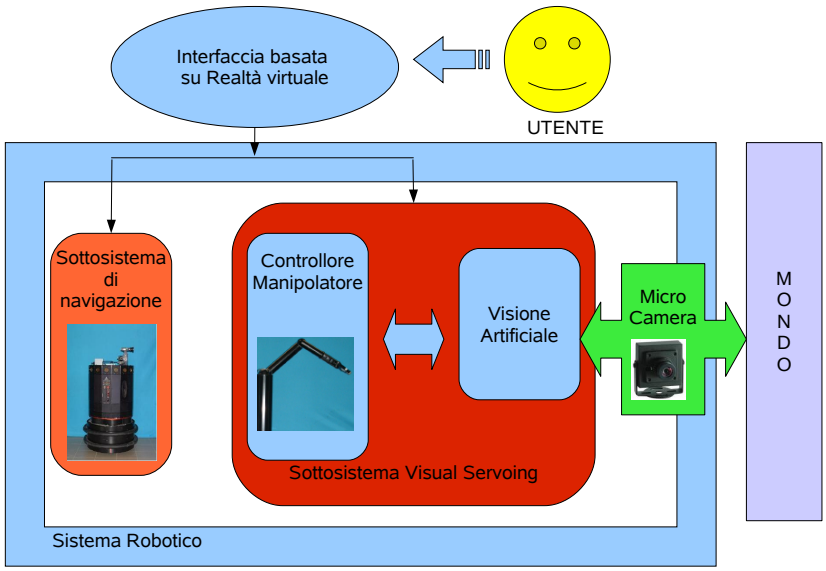


Figura 1.12: Schema generale del sistema progettato.

di sensorialità, è possibile inserire comportamenti complessi come elusione di ostacoli (*obstacle avoidance*).

Per quanto riguarda la manipolazione vera e propria, è necessario un sistema di visione artificiale che fornisca il *feedback* al manipolatore. È stato assunto infatti come premessa che l'ambiente non sia perfettamente noto, per cui è necessario che il robot disponga di un "senso" per l'interazione con l'ambiente. Il sistema sarà quindi dotato di una telecamera attraverso la quale dovrà acquisire ed elaborare le immagini. L'analisi dell'immagine attraverso la visione è fondamentale: permette infatti di avere le informazioni relative all'oggetto mentre il manipolatore si muove, e di comandare il robot di conseguenza cercando di minimizzare gli errori sul posizionamento. Come già illustrato, la visione può fornire informazioni sull'immagine mediante l'estrazione di caratteristiche, o calcolare direttamente il posizionamento reciproco tra bersaglio e *end-effector*.

Infine è necessario un controllore che si occupi del movimento del manipolatore in base ai dati forniti dalla visione. Il controllo può essere fatto comandando dire-

tamente i giunti, mediante l'uso di algoritmi di cinematica diretta o inversa, o se il manipolatore lo prevede, utilizzare i movimenti nello spazio di lavoro, forniti direttamente dal controllore hardware del robot.

Come modulo di interfaccia verrà utilizzato il programma di realtà virtuale PbD, mentre i moduli "operativi" verranno creati appositamente ed integrati sia tra loro che con l'interfaccia. Per il modulo di navigazione bisognerà prevedere la possibilità di ricevere i dati relativi alla traiettoria da eseguire, ad esempio attraverso un meccanismo client-server, già in uso peraltro, all'interno dello stesso programma PbD. Il modulo di visual servoing (visione + controllore) invece riceve poche informazioni dall'interfaccia: il compito da eseguire (tipicamente la cattura di un oggetto), e un identificatore dell'obbiettivo nel caso nella scena siano presenti più oggetti. Visione e controllore opereranno invece con uno scambio continuo di dati per l'esecuzione del compito, per cui saranno necessari meccanismi di comunicazione e sincronizzazione appositi.

Capitolo 2

Dispositivi e Strumenti

In questo capitolo saranno descritte le varie componenti fisiche che compongono il sistema robotico proposto. Come accennato, il robot dovrà essere in grado di muoversi nello spazio, quindi avrà bisogno di una base mobile programmabile, che gli consenta di arrivare al luogo dove sarà previsto il compito di manipolazione.
Successivamente verrà descritto il robot manipolatore vero e proprio: le sue modalità di comando in teleoperazione e programmabile, i modi di funzionamento in modalità cartesiana e nello spazio dei giunti, i suoi pregi ed i suoi limiti. Inoltre verrà descritta la scheda che ne consente l'interfacciamento con un Personal Computer.
Poichè per un compito di *visual servoing* è necessario un apparato di visione, sarà usata una microcamera per l'acquisizione delle immagini che andrà collegata al braccio robotico. Per l'elaborazione delle immagini è necessaria una scheda frame grabber che converta il segnale analogico, fornito dalla microcamera, in formato digitale, e dunque elaborabile al calcolatore.
Infine verrà illustrato anche il dispositivo Cyberglove, che si propone come strumento integrante per l'interfaccia utente basata su realtà virtuale.

2.1 Nomad 200

Figura 2.1: Il robot Nomad 200.

Il Nomad 200, mostrato in fig.2.1, è un robot mobile prodotto dalla "Nomadic Techonolgies Inc." nella prima metà degli anni '90, ampiamente usato in progetti di ricerca in università sia europee che americane. Maggiori informazioni sul robot, dotazione di sensori, architettura interna, modalità di comando sono contenute in [9, 10, 11, 12, 13, 14]. Il Nomad è stato utilizzato come base mobile per il sistema robotico, e la sua capacità di elaborazione viene sfruttata per eseguire "in locale" il lato server dell'architettura software proposta.
Per questa tesi l'equipaggiamento hardware è stato ulteriormente ampliato: l'adattatore di rete WiFi USB è stato sostituito in quanto non consentiva l'avviamento automatico dei daemon e dei servizi necessari per la rete. Ora viene usata una scheda PCI D-Link AirPlus G+.Sono state inserite in slot PCI la scheda CAN-Bus e il frame grabber (descritti nei par.2.5 2.3) anche se con notevoli difficoltà a causa dello spazio limitato tra il connettore ISA occupato dalla scheda di controllo dei motori e il più vicino slot PCI e tra slot PCI stessi.

2.2 Manus

Il Manus è un manipolatore prodotto dalla ditta olandese “Exact Dynamics” [15], allo scopo di essere montato su carrozzine elettriche e servire come sistema di supporto per disabili. Grazie ad alcuni accorgimenti come l’inserimento dei meccanismi di attuazione nel corpo centrale, la cavità dei bracci e la scelta di materiali leggeri, il Manus è caratterizzato da un basso peso (13Kg) e da un ridotto consumo di energia elettrica.
Dal punto di vista più prettamente robotico, il Manus è un manipolatore antropomorfo, con sei gradi di libertà: sei giunti rotoidali, e la possibilità di aprire e chiudere il gripper.
Per quanto concerne le dimensioni, riportate in fig.2.2, in posizione di *fold-in* il

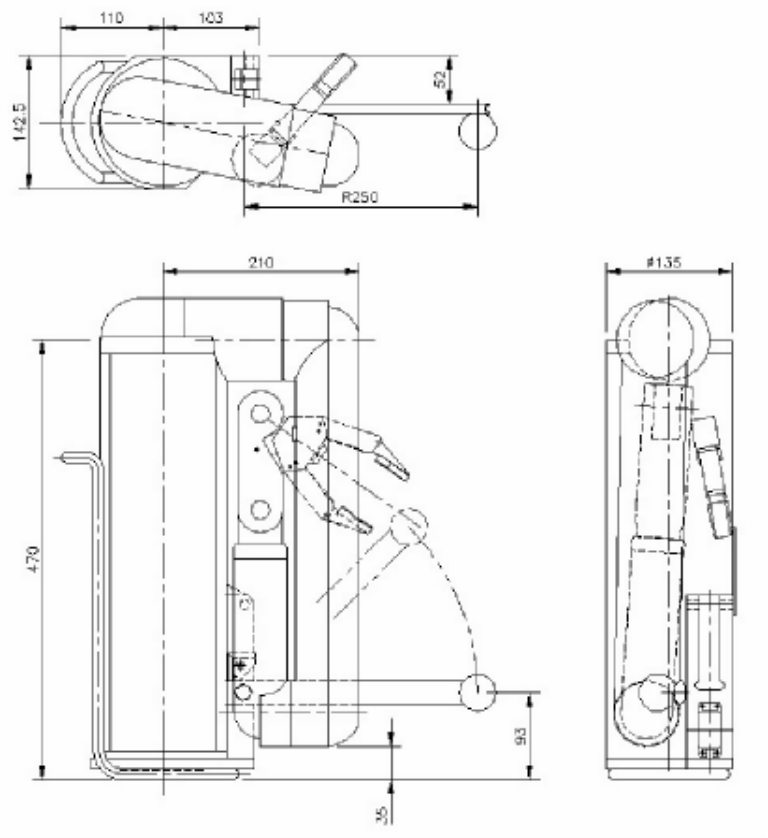


Figura 2.2: Schema del manipolatore in posizione di Fold-in.

Manus è completamente racchiuso su se stesso ed occupa lo spazio minimo, assimilabile grosso modo ad un cilindro di altezza 50cm e diametro 25cm. Il comando può avvenire in due modi fondamentali, tramite teleoperazione, in cui l'utente specifica direttamente le azioni da compiere attraverso un tastierino, o tramite la modalità "trasparente" in cui i comandi gli vengono forniti da un PC. Sia in teleoperazione che in modalità trasparente il Manus può essere comandato nello spazio cartesiano o nello spazio dei giunti; in entrambi i casi è inoltre presente un piccolo display a matrice di led che comunica lo stato del robot, e l'eventuale presenza di situazioni critiche come il blocco di un giunto.
In modalità cartesiana si possono dare i comandi di traslazione rispetto alla terna fissa (posta alla base), mentre i movimenti di rotazione dell'estremità, ossia del gripper, sono espressi rispetto alla terna sul polso, che rimane fissa.
In modalità trasparente è inoltre possibile la lettura dello stato del manipolatore: è possibile leggere la posizione e l'orientamento della terna del polso rispetto alla base, nel caso il comando sia nello spazio di lavoro, e gli scostamenti angolari, in caso di comando nello spazio dei giunti.

2.2.1 La modalità trasparente

Nel caso si voglia comandare il manipolatore da PC, bisogna considerare l'architettura di funzionamento [16] che si basa sullo scambio di messaggi con una temporizzazione di 20ms.

Il controllore trasmette sul bus tre tipi diversi di messaggi, riportati in tabella 2.1 (le informazioni sono espresse in incrementi, le cui unità di misura sono quelle riportate in tab.2.2), con una temporizzazione di $20ms$, da cui l'utente può leggere le diverse informazioni riguardanti lo stato del manipolatore. I messaggi di tipo 1 e 2, identificati dal controllore con la configurazione esadecimale 0x350 e 0x360 rispettivamente, sono messaggi in cui viene comunicato lo stato del manipolatore e dei suoi giunti/gradi di libertà. Il messaggio di tipo 3, caratterizzati dall'identificatore 0x37*X*, sono quelli corrispondenti alla comunicazione dei comandi da eseguire.
Supponiamo di considerare come istante $t = 0$ come quello in cui il manipolatore comunica il messaggio di tipo 1; per $t = 20ms$ il manipolatore comunicherà il

ID	Byte	Valore	Cartesian Mode	Joint Mode
0x350	1	Errore di movimento	Stato	Stato
	2	Blocked DOF	messaggio	messaggio
	3	MSB	X	giunto 1
	4	LSB		
	5	MSB	Y	giunto 2
	6	LSB		
	7	MSB	Z	giunto 3
	8	LSB		
0x360	1	MSB	Yaw	giunto 4
	2	LSB		
	3	MSB	Pitch	giunto 5
	4	LSB		
	5	MSB	Roll	giunto 6
	6	LSB		
	7	MSB	Gripper	Gripper
	8	LSB		

ID	RTR	DLC	Descrizione
0x370	0	0	Inizializzazione del controllore
0x371	0	8	Spostamento nello spazio cartesiano
0x374	0	8	Spostamento nello spazio dei giunti
0x375	0	0	Fold Out
0x376	0	0	Fold IN

Tabella 2.1: Messaggi inviati dalla Control Box del manus.

messaggio di tipo 2, e a $40ms$ quello di tipo 3. I comandi al Manus possono essere inviati solo nella finestra temporale, anch'essa di $20ms$, che intercorre tra la fine del messaggio di tipo 3 ($t = 60ms$), e il nuovo messaggio di tipo 1. A questo punto il comando inviato viene eseguito a partire da $t = 80ms$ per una durata complessiva di $60ms$, cioè fino a quando non viene modificato. In base al manuale [16], di cui si riporta la tabella 2.2, la risoluzione del manipolatore è piuttosto elevata: si parla di spostamenti minimi di 0.022mm e rotazioni minime di decimi di grado, sia per quanto riguarda l'attuazione che la lettura dello stato. Nella pratica tuttavia questa risoluzione non risulta apprezzabile. Questa limitazione nelle prestazioni è dovuta,

Byte	Parametro	Incremento	Min. Incremento	Max Incremento
1	Lift Unit	-1 0 +1	-1	+1
modalità giunto				
2	Giunto 1	0.1°	-10	10
3	Giunto 2			
4	Giunto 3			
5	Giunto 4	0.1°	-10	10
6	Giunto 5			
7	Giunto 6			
8	Gripper	0.1mm	0	15
modalità cartesiana				
2	X	0.022mm	0	127
3	Y			
4	Z			
5	Yaw	0.1°	0	10
6	Pitch			
7	Roll			
8	Gripper	0.1mm	0	15

Tabella 2.2: Manus: risoluzione dei comandi di movimento.

sia all'imprecisione nella lettura dei dati, sia al sistema di trasmissione delle coppie che, essendo realizzato mediante cinghie, non riesce a contrastare gli attriti interni per movimenti piccoli [17].

2.3 Scheda Can-Bus

Figura 2.3: Scheda CAN-Bus PCI.

Exact Dynamics[15] fornisce insieme al manipolatore Manus una scheda per rete CAN di tipo ISA per la comunicazione fra la Control Box e PC nella modalità d'uso denominata *transparent mode*.
Col passare degli anni però la tecnologia ISA è diventata obsoleta, ed i produttori di schede madri non includono più questo tipo di slot nei loro prodotti. É stato quindi necessario adeguare il sistema di comunicazione tramite una scheda per rete CAN basata su slot PCI, di cui è presentata l'immagine in fig.2.3.
Il controller in questione è una scheda modello CANnes prodotta da Trinamic GmbH e distribuita in Italia da Celte. Una caratteristica importante che accompagna questa scheda è la possibilità di funzionamento con il sistema operativo Linux mediante i driver forniti dal produttore. Questa scheda utilizza l'integrato SJA1000[18] prodotto da Philips Semiconductors largamente utilizzato in tutti i dispositivi di questo tipo mentre l'interfacciamento con il bus PCI viene invece realizzato tramite il PCIController PCI9052 prodotto da PLX Technology [19]. Le due uscite fornite di tipo CAN-bus sono indipendenti l'una dall'altra, ma per essere collegate al con-

trollore del Manus è necessario costruire un apposito cavo seriale (RS232) con la piedinatura (lato saldatura maschio e femmina) mostrata in fig.2.4:
I piedini da ciascuna parte del cavo vanno terminati tra loro con una resistenza da

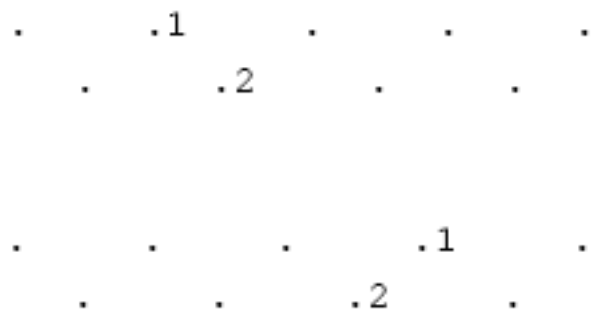


Figura 2.4: Connettori per il cavo di collegamento Manus/PC.

120 Ω.

2.4 Microcamera CCD

Figura 2.5: Microcamera CCD

L'apparato di visione è costituito da una telecamera CCD a colori, mostrata in fig.2.5, con sensore Sharp $1/4$" dotata delle seguenti caratteristiche:

- Risoluzione 420 linee (NTSC 510(H)x492(V) pixel; PAL:(500(H)x582(V) pixel)
- Uscita video: 1.0Vp-p 75 Ω
- Temperatura di lavoro -10°C + 50°C RH 95%
- Sensibilità 1,5 Lux (F:2.0)
- Bilanciamento automatico
- Alimentazione 12V DC
- Dimensioni 32x32mm

La microcamera è stata collegata ad un supporto metallico a *L* con del velcro, ed il supporto metallico è collegato alla pinza del Manus, anch'esso con del velcro, in modo da realizzare un sistema *Eye in Hand*. In questo modo è possibile staccare ed attaccare la telecamera con estrema semplicità, in modo da utilizzarla solo quando necessario, e lasciare libero il manipolatore per altre applicazioni che non

richiedano la visione. L'unico svantaggio dato dal collegamento col velcro è la non rigidezza dell'ancoraggio del supporto, che comporta degli errori per quanto riguarda l'orientazione tra visuale di telecamera e gripper, ma tali errori sono abbastanza piccoli da poter essere di fatto trascurabili. Per questa tesi non sono state eseguite correzioni prospettiche sull'immagine catturata; i parametri usati per la telecamera sono quelli generici che fornisce ARToolkit (par.3.3). È possibile seguire invece il processo di calibrazione specificato dalla libreria per generare un file di parametri "personalizzato" per la telecamera utilizzata ed ottenere un lieve miglioramento della precisione.

2.5 Frame Grabber

La scheda Frame grabber utilizzata per l'interfacciamento della telecamera è la Terratec Cinergy 250 PCI [20], una scheda di acquisizione audio-video in grado di gestire i seguenti tipi di segnale:

Figura 2.6: Scheda di acquisizione Terratec Cinergy 250 PCI.

1. Immagine televisiva tramite antenna
2. Immagini televideo ad alta velocità
3. Immagini provenienti da sorgenti e/o apparecchi esterni
4. Segnali audio esterni
5. Segnali radio FM

Il sistema, illustrato in fig.2.6, è basato sul chipset Philips Semiconductor della famiglia saa713x (nel caso saa7133[21]) che si occupa sia della parte di acquisizione dei vari tipi di segnali, sia della loro elaborazione. Questo modello di chipset è ampiamente diffuso in schede di questo tipo tanto che ne è stato implementato un driver direttamente nel kernel di Linux già dalle prime versioni 2.6 per poi essere migliorato nelle versioni successive solo per il riconoscimento/supporto delle schede più recenti che ne fanno uso.

La scheda, come illustrato in fig.2.7, è dotata di quattro ingressi:

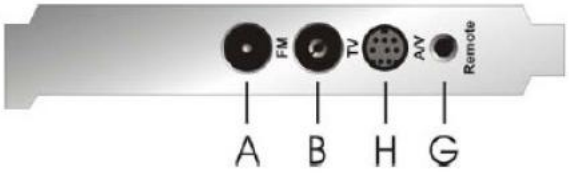


Figura 2.7: Ingressi della scheda di acquisizione.

A un ingresso per l'antenna radio FM (ora non più presente)
B un ingresso per l'antenna televisiva analogica
H un ingresso per le sorgenti audio-video esterne
G un ingresso per il ricevitore ad infrarossi per il telecomando

In particolare all'ingresso H viene collegato un cavetto che si divide in più spi-

Figura 2.8: Cavo di ingresso della scheda di acquisizione.

notti consentendo di collegare diversi tipi di sorgenti. La scheda si occupa poi in automatico del riconoscimento del tipo di segnale acquisito.
Come mostrato in fig.2.8, è dunque possibile acquisire segnali audio stereo (spinotti C), ingressi video di tipo composito (spinotto D) come quello fornito dalla micro-camera, segnali di tipo S-Video (spinotto E). È inoltre presente un'ulteriore uscita audio (spinotto F). La scheda viene fornita di manuale in formato elettronico, CD con i driver e software per Windows2000/XP, cavo doppio per il collegamento alla scheda audio, e telecomando ad infrarossi. La scheda è stata usata prima su PC,

poi è stata installata su slot PCI a bordo del Nomad ma, causa del poco spazio tra slot PCI, è stato necessario rimuovere l'ingresso Radio FM, che andava a urtare il connettore dell'antenna della scheda wireless di rete.

2.6 Il guanto Cyberglove ed il dispositivo Fastrak

Figura 2.9: Il dispositivo CyberGlove

Il dispositivo CyberGlove, prodotto da Virtual Technology, [22] e mostrato in fig.2.9, è costituito da un guanto destro su cui sono stati inseriti diciotto sensori che rilevano lo sforzo meccanico. Indossando il guanto e piegando le articolazioni della mano si attivano i sensori che, inseriti in corrispondenza delle giunture, rilevano la sollecitazione ed inviano i dati tramite l'interfaccia CGIU (CyberGlove Interface Unit) al personal computer. In tal modo è possibile ricavare la configurazione dei giunti della mano ed elaborando i dati ricostruire i movimenti delle dita. Questa importante caratteristica può essere utilizzata nella costruzione di un modello di mano virtuale, per visualizzare in tempo reale i movimenti dell'utente. Oltre ai citati sensori, il guanto dispone di altri sei attuatori di tipo vibrotattile, in grado di restituire un *feedback* sensoriale all'utente.

In fig.2.10 è riportato il dispositivo Fastrak prodotto dalla Polhemus [23]. Il tracker elettromagnetico Fastrack è composto da un trasmettitore fisso ed un ricevitore mobile che nel caso viene collegato al guanto. I dati vengono inviati attraverso una

Figura 2.10: Il dispositivo CyberGlove

porta seriale (RS-232) ad alta velocità, al personal computer. Grazie ad esso è possibile tracciare la posizione del guanto all'interno dello spazio tridimensionale. Fastrak e Cyberglove costituiscono nell'insieme un dispositivo molto versatile, che si colloca come strumento ideale nell'ambito dei progetti basati su realtà virtuale[1].

Capitolo 3

Librerie Software

Il software costituisce, in un certo senso, l'anima di un sistema robotico. Grazie ad esso il progettista riesce ad avere accesso alle funzionalità messe a disposizione dall'hardware. La realizzazione di sistemi complessi richiede l'uso di strumenti in grado di astrarre il compito del programmatore, in modo da consentirgli di concentrarsi il più possibile su funzioni ad alto livello, cercando di tralasciare dettagli di implementazione. In questo capitolo vengono analizzate le tecnologie ed alcune librerie utilizzate per la realizzazione del programma che governa i vari componenti robotici del sistema.

L'interfaccia utente viene realizzata tramite un programma di simulazione: all'interno di un ambiente virtuale l'utente impartisce ordini ad un robot, anch'esso virtuale. La simulazione risulta comunque utile per l'utente che può così verificare l'ordine impartito ed avere un riscontro indicativo, sull'operato futuro del robot.
Le prime due tecnologie illustrate in questo capitolo sono legate alla interfaccia, e riguardano la *computer graphics*: la libreria OpenGL, ad un livello di astrazione più basso, si occupa della creazione di oggetti, e di come questi possono essere manipolati grazie alle funzionalità delle schede video. VRML opera ad un livello più alto, e permette la definizione di ambienti virtuali.
Per la realizzazione di un sistema *visual servoing position based* è necessario che la visione fornisca la posizione relativa tra telecamera ed oggetto. Questo compito è assolto dalla libreria ARToolkit (anch'essa basata in parte su OpenGL), che me-

diante una conoscenza *a priori* dell'oggetto permette il calcolo sia della posizione che dell'orientazione.
In tutti i compiti ingegneristici è prevista una formulazione matematica del modello del problema, ma spesso le funzioni matematiche messe a disposizione dai linguaggi di programmazione sono o insufficienti, o poco pratiche; perciò l'uso di librerie che permettano la creazione e la manipolazione di strutture matematiche più o meno complesse è indispensabile. Le librerie Nurbs e Newmat sono state utilizzate in questa tesi per la realizzazione del sistema di VS. La libreria Nurbs mette a disposizione delle funzioni per la manipolazione di curve ed è utilizzata all'interno del programma di interfaccia utente per la creazione e manipolazione delle traiettorie. La libreria Newmat consente di eseguire su matrici gran parte delle operazioni dell'algebra lineare ed è impiegata per l'implementazione delle funzioni di cinematica del manipolatore.

3.1 OpenGL

OpenGL [24] (Open Graphics Library) costituisce un'interfaccia di programmazione per la grafica 2D e 3D. Nasce nel 1992 da un accordo tra diverse aziende del settore hardware e software come ATI Technologies (ora acquisita da AMD), NVIDIA, Apple Computer, IBM, Dell, Microsoft (la quale ne è uscita dopo un anno), Sun Microsystems ed altri che negli anni si sono aggiunti. OpenGL assolve a due compiti fondamentali:

1. nascondere la complessità di interfacciamento con acceleratori 3D differenti, offrendo al programmatore una API unica ed uniforme;

2. nascondere le peculiarità dei diversi acceleratori 3D, richiedendo che tutte le implementazioni supportino completamente l'insieme di funzioni OpenGL, ricorrendo ad un'emulazione software se necessario.

In ambiente Unix/Linux, OpenGL si è imposto come standard *de facto* per la programmazione grafica mentre, in ambiente Windows, Microsoft ufficialmente non supporta OpenGL in favore delle API proprietarie Direct X. Sono comunque disponibili versioni OpenGL per sistemi Microsoft che funzionano senza problemi.

I vantaggi derivanti dall'uso di tale libreria sono la possibilità di operare con i più diversi sistemi operativi e con i linguaggi di programmazione più diffusi come C, C++, Java, Fortran, Phyton, ed altri, che forniscono port e binding appositi. OpenGL opera a basso livello e richiede passi precisi per disegnare una scena, obbliga quindi i programmatori ad avere una buona conoscenza della pipeline grafica stessa, ma al contempo lascia una certa libertà per implementare complessi algoritmi di rendering. Lo standard aperto ha permesso la costruzione di estensioni sia per quanto riguarda librerie aggiuntive, come GLU e GLUT, sia per quanto riguarda operazioni e funzionalità messe a disposizione dalle GPU di ultima generazione. Per esempio, l'abbreviazione di NVIDIA (NV) viene usata nel definire la loro funzione proprietaria `glCombinerParameterfvNV` e la costante `GL_NORMAL_MAP_NV`.

3.2 VRML

L'acronimo VRML [25] sta per **V**irtual **R**eality **M**odeling **L**anguage. VRML costituisce un linguaggio specifico per la creazione di grafica 3D in applicazioni multimediali. Lo scopo di questo linguaggio è facilitare gli sviluppatori alla creazione di ambienti grafici per la pubblicazione su pagine internet, attraverso un formato di linguaggio aperto e facilmente manipolabile.
Per visualizzare direttamente sul browser gli ambienti è necessario un programma apposito, ma oltre ad essere disponibili dei *parser* liberamente scaricabili (come Cortona), i principali browser come IE, Opera, Mozilla, Firefox, forniscono dei plug-in che assolvono a questo compito.
Un file VRML altro non è che un file di testo in cui una specifica sintassi definisce una gerarchia di oggetti ognuno con delle proprietà specifiche. La costruzione della scena avviene mediante una struttura ad albero dotato di rami e foglie dove a ciascun oggetto vengono associati attributi semplici, come dimensioni, colore, posizione, o anche complesse, come texture, brillantezza, trasparenza ecc. Vrml mette a disposizione primitive come cubi, cilindri e altri, ma per oggetti complessi è possibile definire la struttura mediante assemblamento di triangoli nello spazio. È inoltre possibile associare degli URL ad oggetti che si riferiscono a file locali o non locali. VRML consente dunque di creare scene semplici o complesse con estrema facilità

e flessibilità e per questo si è imposto come protagonista nella modellazione 3D per web sia in ambiti professionali che per utenti non professionisti. La prima versione di VRML è stata specificata nel Novembre 1994 ed era largamente basata su un precedente progetto di SGI, mentre la specifica attuale è uno standard ISO denominato VRML97. La prossima versione, in fase di definizione, sarà chiamata X3D. Il Web3D Consortium [26] è il consorzio incaricato di coordinare gli sviluppi del linguaggio.

3.3 ARToolkit

ARToolkit è una libreria di visione artificiale, scritta in linguaggio C e basata su OpenGL, che consente l'acquisizione di un flusso video da una telecamera e l'elaborazione in tempo reale delle immagini ricevute. Il software è in grado di riconoscere e calcolare la posizione e l'orientamento di specifici oggetti all'interno dei frame ricevuti, grazie all'utilizzo di appositi "marker", ovvero di immagini con caratteristiche particolari che il software è "addestrato" a riconoscere. La libreria consente lo sviluppo di programmi di "realtà aumentata" in cui ad immagini di un ambiente reale vengono aggiunti oggetti realizzati in computer grafica. È possibile infatti definire degli oggetti virtuali mediante primitive OpenGL che vengono inseriti nelle immagini acquisite e visualizzati attraverso un flusso video in una finestra grafica. In fig.3.1 è presentato un tipico esempio applicativo costruito mediante ARToolkit: il sistema acquisisce immagini da una telecamera, le immagini vengono elaborate per la ricerca di un marker, ed in corrispondenza ad esso, viene visualizzato un oggetto virtuale costituito, in questo caso, da un omino con la spada.
Le prinicipali caratteristiche di questa libreria [27] sono:

- La possibilità di definire marker personalizzati, basati su quadrati neri.
- Semplici procedure di calibrazione.
- Supporto multipiattaforma (Windows, Linux, MacOS).
- Codice sorgente aperto.

In questa tesi ARToolkit viene sfruttata per la sua capacità di calcolare la posizione e l'orientazione relativa tra la telecamera ed un specifico marker. L'utilizzo di

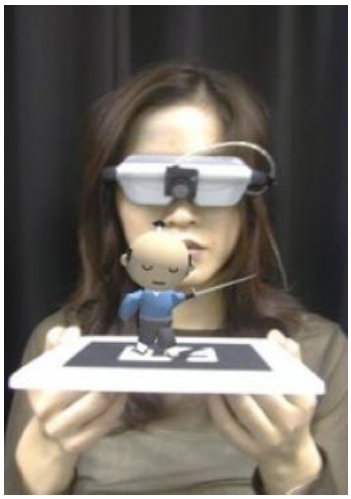


Figura 3.1: ARToolkit: esempio di realtà aumentata.

un marker consente di rendere riconoscibile al sistema l'oggetto di interesse del compito di manipolazione. Grazie ad ARToolkit è nota la posizione dell'oggetto rispetto alla telecamera per cui, conoscendo la posizione della telecamera rispetto al manipolatore, è possibile risalire alla posizione dell'oggetto rispetto al manipolatore. Si riescono pertanto ad ottenere tutte le informazioni necessarie per imporre, nell'algoritmo di VS, uno spostamento del braccio robotico verso il bersaglio. La funzionalità di visualizzazione di oggetti virtuali può essere utilizzata come metodo di conferma dell'avvenuto riconoscimento dell'oggetto di interesse.

L'algoritmo di visione artificiale integrato in ARToolkit è basato su tecniche di rilevamento ed estrazione di contorni ed angoli dalle immagini, grazie alle quali il sistema dimostra buona precisione e robustezza.
La fase di elaborazione è suddivisa nei passi elencati in fig.3.2 :

1. La telecamera acquisisce l'immagine della scena e la invia al computer.

2. Si cercano all'interno del frame acquisito delle aree di interesse, contenenti qualsiasi forma quadrata.

3. Le zone di interesse vengono esaminate per la ricerca dei marker specificati.

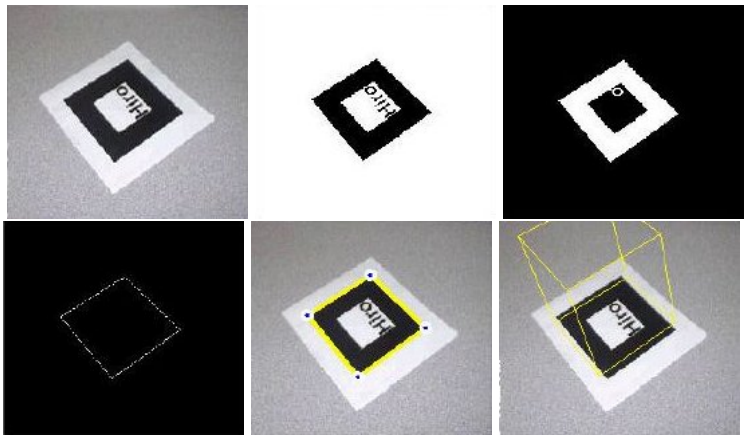


Figura 3.2: Le diverse fasi del processo di elaborazione visuale di ARToolkit: Acquisizione, Ricerca quadrati, Estrazione Contorni, Calcolo della Posizione, Rendering dell'oggetto virtuale

4. Viene calcolata la matrice di trasformazione tra i marker trovati e la telecamera.

5. Al frame viene aggiunto l'oggetto virtuale e l'immagine finale viene visualizzata nella finestra grafica.

Sia la tipologia che la dimensione del marker devono essere specificate appositamente nel programma, che in questo modo è in grado di calcolarne il posizionamento nello spazio senza l'utilizzo di telecamere stereoscopiche. Questa soluzione consente la creazione di sistemi non solo più economici, ma anche più veloci dal punto di vista dell'elaborazione.
La robustezza e la facilità di utilizzo hanno reso ARToolkit molto popolare come ap-

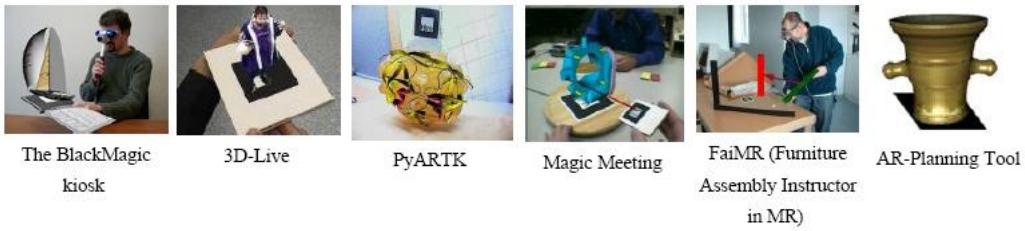


Figura 3.3: Progetti internazionali che utilizzano ARToolkit.

plicativo di visione artificiale in molti progetti di realtà virtuale sia locali e nazionali [28], sia in ambito internazionale (fig.3.3) [29].

3.4 Nurbs++

Nurbs++ è una libreria per C++ che consente di gestire curve e superfici utilizzando rappresentazioni esatte, parametriche, o approssimate mediante interpolazione. Nella rappresentazione approssimata per definire una curva è necessario fornire dei punti che costituiscono coordinate spaziali, Nurbs provvede poi alla parametrizzazione della curva attraverso *spline* utilizzando metodi di interpolazione (in cui la curva passa obbligatoriamente per i punti definiti) o ai minimi quadrati (in cui la curva tenta di avvicinarsi il più possibile ai punti). La scelta del modello e del metodo di approssimazione dipende dall'uso che se ne vuol fare. Non esiste un metodo migliore in assoluto, è consigliabile invece utilizzare approcci differenti confrontando poi quale di questi fornisce i migliori risultati.
É possibile inoltre operare su tali curve ricavando informazioni come lunghezza, derivate globali e puntuali, sottocampionamenti e sovracampionamenti, e inserendo punti di controllo. Gli oggetti "curva" una volta creati non sono statici, ma possono essere modificati inserendo o rimuovendo altri punti, o cambiando la parametrizzazione. Inoltre è possibile fondere più curve, dividere una curva in più punti, e infine è prevista la possibilità di esportare la curva Nurbs in un file di tipo VRML in modo da facilitarne la visualizzazione.

3.5 NewMat

NewMat [30] è una libreria sviluppata in C++ che mette a disposizione dell'utente un'implementazione di matrici e vettori intesi come strutture matematiche. A differenze di classi analoghe disponibili per il linguaggio C++, come ad esempio Ublas (parte del progetto BOOST[31]), Newmat si distingue la leggerezza e la semplicità di utilizzo.
La libreria è totalmente libera e scaricabile dal sito `http://www.robertnz.net/index.html`; una volta scaricata è necessaria la compilazione, per la quale

vengono forniti diversi Makefile a seconda del compilatore utilizzato. Le strutture dati che la classe implementa sono diverse, e rappresentano varie tipologie di vettori e matrici, come ad esempio matrici standard, matrici sparse, matrici identità, ed altre. Il tipo di dato è automatico, ed è sempre in virgola mobile. Oltre alle strutture sono definite diverse operazioni dell'algebra lineare, come somma e prodotto, ottenute mediante *overloading* degli operatori, e metodi di classe orientati principalmente alla risoluzione di sistemi lineari.
La libreria è pensata, in particolare, per i programmatori che devono realizzare formule matematiche per applicazioni nel campo della statistica, ingegneria, fisica, ed applicazioni scentifiche in generale. In questa tesi NewMat è utilizzata per la realizzazione delle funzioni per il calcolo della cinematica diretta ed inversa del manipolatore Manus.

Capitolo 4

Analisi del Problema

In questo capitolo sono approfonditi gli argomenti di visual servoing e problema cinematico dei manipolatori, introdotti brevemente in sez.1.4. I problemi verranno definiti ed analizzati in maniera rigorosa, ricorrendo al formalismo matematico. Per una introduzione alla geometria analitica e alle notazioni relative ai sistemi di riferimento, si rimanda alla lettura dell'appendice A.

4.1 Visual Servoing

La tipologia di visual servoing sviluppata in questa tesi è *dynamic look and move*, *position based*, *eye in hand*. Le informazioni che si ricavano dalla visione sono quindi riferite allo spazio di lavoro. I dati in base ai quali si comanda il manipolatore sono spostamenti ed orientazione relativi ad un sistema di riferimento prescelto, in questo caso il sistema della telecamera.
Definiamo alcune terne di riferimento (fig.4.1):

- $[B]$ *base frame* la terna di riferimento del manipolatore (spesso definita anche terna $[0]$)
- $[T]$ *tool frame* la terna solidale alla pinza del manipolatore
- $[M]$ *marker frame* la terna obbiettivo
- $[C]$ *camera frame* la terna associata alla microcamera.

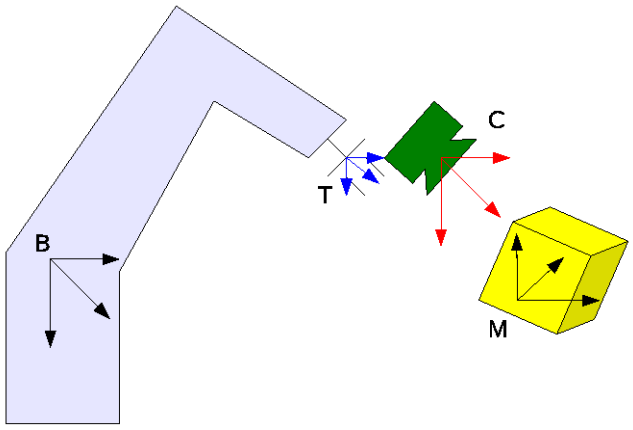


Figura 4.1: Terne di riferimento.

Supponiamo che, grazie agli algoritmi di visione artificiale, si riesca a ricavare P_C^M (posizione del marker descritto secondo il sistema di riferimento della telecamera); come illustrato nella sez.3.3, questa ipotesi non solo è ragionevole, ma anche verificata. È lecito supporre di conoscere anche P_C^T (posizione della telecamera rispetto al sistema di riferimento della pinza) visto che la telecamera ha sempre la stessa posizione. Infine supponiamo di conoscere P_B^T (posizione della pinza rispetto alla terna di riferimento del manipolatore); anche questa è un'ipotesi verificata, come mostrato più avanti nel sez.4.2.
Possiamo allora calcolare la posa dell'oggetto rispetto alla terna di riferimento del manipolatore:

$$P_B^M = P_B^T \cdot P_T^C \cdot P_C^M$$

Il compito di manipolazione può essere definito come moto del manipolatore verso la posizione dell'oggetto,

$$P_B^T \longmapsto P_B^M$$

Se il manipolatore è controllabile nello spazio cartesiano, allora basterà calcolare i singoli scostamenti delle componenti della posa, ed imporre il movimento a ciascun

grado di libertà

$$P_B^T - P_B^M = \begin{bmatrix} x_B^T - x_B^M \\ y_B^T - y_B^M \\ z_B^T - z_B^M \\ \alpha_B^T - \alpha_B^M \\ \beta_B^T - \beta_B^M \\ \gamma_B^T - \gamma_B^M \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ \alpha_c \\ \beta_c \\ \gamma_c \end{bmatrix} \tag{4.1}$$

In (4.1) $[x_c, y_c, z_c, \alpha_c, \beta_c, \gamma_c]$ sono i gradi di libertà che il manipolatore possiede quando si muove nello spazio cartesiano. Bisogna però considerare che:

- La visione fornisce dati precisi ma non esatti, quindi $P_C^M \rightsquigarrow \hat{P_C^M}$, cioè si passa da una posizione esatta, ad una stima approssimata $\hat{P}$.
- Come accennato nella sez.2.4, anche la posa della microcamera è imprecisa a causa dell'ancoraggio non rigido della telecamera al manipolatore $P_T^C \rightsquigarrow \hat{P_T^C}$.
- I dati forniti dagli encoder del manipolatore sono anch'essi soggetti ad errore, quindi la posizione della pinza non è esatta $P_B^T \rightsquigarrow \hat{P_B^T}$.
- Anche l'attuazione dei giunti è soggetta ad errori a causa degli attriti interni e della resistenza elastica delle cinghie del Manus (si veda sez.2.2).

Perciò anche la posizione del marker rispetto alla terna base è affetta da errori $P_B^M \rightsquigarrow \hat{P_B^M}$. Bisogna allora cercare di ridurre il più possibile l'errore:

$$E(P) = P_B^M - \hat{P_B^M}$$

Mentre gli errori sulla posa, della telecamera, e sui dati forniti dagli encoder del manipolatore sono di un ordine di grandezza fissato, l'errore della visione è tanto più grande quanto maggiore è la distanza tra la telecamera e l'oggetto. Uno spostamento dalla posizione attuale a quella stimata riporterebbe tutto l'errore dato dalla visione mentre, come mostrato in fig.4.2, uno spostamento ridotto nella direzione stimata comporta un errore di posizionamento più piccolo. Nella figura l'errore di posa corrisponde alla distanza tra il vettore di posa esatto e quello approssimato: si può notare che dalla posizione attuale l'errore cresce man mano ci si avvicina all'oggetto inquadrato. Con spostamenti ridotti invece l'errore (E1) è sempre presente,

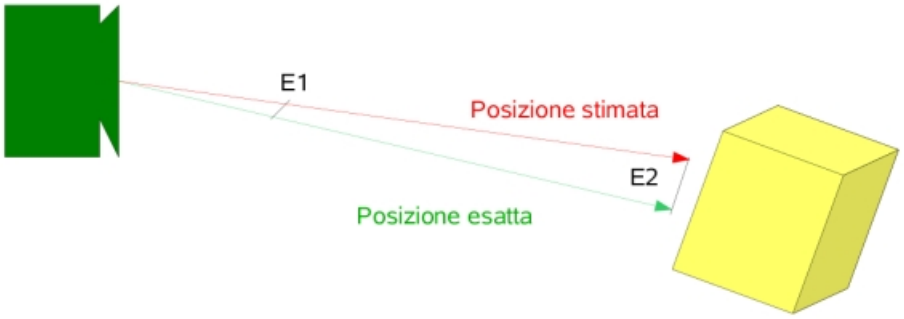


Figura 4.2: Errore introdotto dalla visione.

ma risulta trascurabile rispetto all'errore globale di posa (E2). Per queste ragioni è opportuno introdurre un metodo di avvicinamento per passi, in cui per ogni passo:

1. Si calcola la posizione $\hat{P_B^M}$.
2. Si calcola uno spostamento incrementale [$\Delta x_c, \Delta y_c, \Delta z_c, \Delta \alpha_c, \Delta \beta_c, \Delta \gamma_c$] verso la posizione stimata $\hat{P_B^M}$.
3. Si esegue lo spostamento nello spazio cartesiano.

Si ripete fino a quando l'errore tra la posizione attuale e la posizione desiderata è sufficientemente piccolo:

$$\hat{P_B^T} - \hat{P_B^M} \leq \varepsilon \iff \begin{bmatrix} \hat{x}_B^T - \hat{x}_B^M \\ \hat{y}_B^T - \hat{y}_B^M \\ \hat{z}_B^T - \hat{z}_B^M \\ \hat{\alpha}_B^T - \hat{\alpha}_B^M \\ \hat{\beta}_B^T - \hat{\beta}_B^M \\ \hat{\gamma}_B^T - \hat{\gamma}_B^M \end{bmatrix} \leq \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \varepsilon_\alpha \\ \varepsilon_\beta \\ \varepsilon_\gamma \end{bmatrix}$$

Per completare il controllore si deve trovare una relazione tra la distanza e i gli incrementi di movimento che il manipolatore deve eseguire.
In letteratura sono ben note tecniche di controllo di manipolatore mediante asservimento visivo basato su posizione. Un approccio classico consiste nel ricavare la posizione attuale dell'organo terminale, calcolare la posizione dell'obiettivo, e co-

struire una traiettoria che consenta al manipolatore di arrivare al punto desiderato. La traiettoria viene inizialmente calcolata nello spazio di lavoro, per poi essere "convertita" nello spazio dei giunti, per l'attuazione del robot. È inoltre neccessaria l'implementazione di un controllore software del robot, di solito realizzato mediante controllo proporzionale/derivativo (PD) come in [32], o solo proporzionale come in [33]. Questo metodo ha però lo svantaggio di essere pesante dal punto di vista computazionale.
In questa tesi si è deciso di utilizzare un approccio diverso basato su un controllo di tipo *stop and go*, eliminando la parte di pianificazione della traiettoria. La scelta è motivata anche dal fatto che il Manus dispone di una modalità di spostamento nello spazio cartesiano fornita direttamente dalla control box hardware. Comandando il robot unicamente mediante incrementi, è eliminata anche la necessità di conoscere la posizione attuale della pinza: basta solo riportare i dati della visione dal sistema di riferimento della telecamera a quello della terna di riferimento del manipolatore. Anche il controllore che ne risulta è semplificato, infatti è sufficiente un controllo proporzionale, ma operante unicamente sulla posizione dei giunti e non sulle velocità di attuazione. Supponendo che i gradi di libertà del manipolatore nello spazio cartesiano siano corrispondenti alle componenti del vettore di posa si ottiene

$$C_i = k_i \cdot (\hat{P_B^T} - \hat{P_B^M})_i \tag{4.2}$$

In (4.2) C_i identifica il grado di libertà nello spazio di lavoro del manipolatre, il pedice i della parentesi corrisponde all'i-esimo elemento del vettore posa, e k rappresenta la costante proporzionale di spostamento corrispondente al grado di libertà. Un incremento uniforme lungo tutti i gradi di libertà, differenziando come ovvio i gradi di libertà traslazionali da quelli rotazionali, non rappresenta una soluzione adeguata. Infatti nel caso l'oggetto sia molto spostato verso una delle tre direzioni x, y, z il muoversi della stessa quantità, anche se piccola, in le direzioni porta, prima o poi, all'uscita dell'oggetto dalla visuale della telecamera.
Una possibile soluzione è quella di imporre uno spostamento massimo, analizzare il grado di libertà in cui l'oggetto appare più distante, imporre lo spostamento massimo per tale grado di libertà e normalizzare gli altri spostamenti. Un miglioramento ulteriore consiste nell'utilizzare un grado di spostamento massimo, in funzione della

distanza: semplificando si può scegliere uno spostamento $\tilde{x}$ per distanze superiori a $dist_{MAX}$ e spostamento $\tilde{x}/2$ per distanze inferiori. La distanza "di soglia" $dist_{MAX}$ e lo spostamento $\tilde{x}$ vanno scelti in funzione della risoluzione e precisione del manipolatore e dell'angolo di visuale della microcamera, tuttavia non rappresentano dei parametri critici del sistema. Con questa soluzione il manipolatore si muove più velocemente quando si trova lontano dall'oggetto e più lentamente quando si trova maggiormente vicino. Infatti spostamenti troppo grandi in prossimità dell'oggetto rischiano di farlo uscire dalla visuale della videocamera. Lo stesso discorso vale per i gradi di libertà angolari α, β, γ.

La presa dell'oggetto è automatica se la telecamera si trova dentro la pinza del manipolatore; nel sistema proposto invece la telecamera è posta sulla pinza. Questo significa che se la pinza si trova davanti all'oggetto, allora l'oggetto è fuori dalla visuale. Pertanto l'unica soluzione possibile è posizionare la microcamera di fronte all'oggetto, ed effettuare la presa "alla cieca" con un movimento preimpostato. L'efficacia di questa soluzione è tanto maggiore quanto più è preciso il processo di posizionamento della telecamera.
La trattazione esposta rimane valida con la seguente correzione: invece di

$$\hat{P_B^T} \rightsquigarrow \hat{P_B^M} \qquad \text{si ha} \qquad \hat{P_B^T} \rightsquigarrow \hat{P_B^M} - \hat{P_B^C}$$

Questa formulazione del problema deve infine essere corretta, considerando un termine ulteriore di scostamento, visto che portando $\hat{P_B^T}$ in $\hat{P_B^M} - \hat{P_B^C}$ avremmo una collisione tra telecamera ed oggetto. Pertanto si avrà

$$\hat{P_B^T} \rightsquigarrow \left(\hat{P_B^M} - \hat{P_B^C}\right) + T \tag{4.3}$$

In (4.3) T rappresenta un termine di traslazione che permette il posizionamento della telecamera davanti al marker con una distanza minima (tipicamente qualche centimetro), ma con orientazione esatta. La realizzazione del moto di avvicinamento

rappresentato dalla (4.3) non è un problema, dato che sia $\hat{P_B^C}$ che T sono costanti e noti a priori.

4.2 Cinematica

Si è detto che la Control Box ha la possibilità di comandare il Manus direttamente nello spazio di lavoro. Tale modalità di comando tuttavia, è efficace solo per il posizionamento: infatti i comandi di orientazione della pinza *(roll, pitch, yaw)* non corrispondono all'orientamento rispetto alla terna fissa, bensì alla terna del polso, la quale a sua volta cambia la propria orientazione al variare della posizione.
Tenendo conto che le informazioni relative alla posa dell'oggetto sono espresse nello spazio di lavoro, si rende necessario utilizzare la modalità di comando nello spazio dei giunti attraverso l'uso delle equazioni di cinematica diretta ed inversa adattate al manipolatore in uso. Mentre in precedenti lavori di tesi i calcoli della cinematica venivano realizzati totalmente [34] o parzialmente [35], con l'utilizzo di librerie esterne come Roboop [36], in questo caso si è provveduto alla realizzazione di funzioni apposite per il calcolo della cinematica sia inversa che diretta del manipolatore Manus.

4.2.1 Cinematica Diretta

Le matrici di trasformazione da una terna di riferimento alla successiva si ricavano dalle terne fissate utilizzando, per l'orientamento, la notazione angolare per assi fissi detta anche *Roll Pitch Yaw*, in cui *Roll* rappresenta l'angolo attorno all'asse *x* , *Pitch* attorno all'asse *y* e *Yaw* attorno all'asse *z* secondo il verso destrorso. I pedici nelle formule si riferiscono agli angoli di giunto del manipolatore.

$$T_1^0 = \begin{bmatrix} c_1 & 0 & -s_1 & 0 \\ s_1 & 0 & c_1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_2^1 = \begin{bmatrix} c_2 & -s_2 & 0 & L_2c_2 \\ s_2 & c_2 & 0 & L_2s_2 \\ 0 & 0 & 1 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^2 = \begin{bmatrix} c_3 & 0 & s_3 & 0 \\ s_3 & 0 & -c_3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_4^3 = \begin{bmatrix} c_4 & 0 & -s_4 & 0 \\ s_4 & 0 & c_4 & 0 \\ 0 & -1 & 0 & L_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_5^4 = \begin{bmatrix} c_5 & 0 & s_5 & 0 \\ s_5 & 0 & -c_5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_6^5 = \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ s_6 & c_6 & 0 & 0 \\ 0 & 0 & 1 & L_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Da queste si ricava la matrice di trasformazione di coordinate dalla terna 0 alla terna 6, (cioè da base a polso):

$$T_6^0 = T_1^0 \cdot T_2^1 \cdot T_3^2 \cdot T_4^3 \cdot T_5^4 \cdot T_6^5 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

Dove gli elementi della matrice di (4.4) sono:

$$
\begin{aligned}
r_{11} &= ((c_1c_{23}c_4 - s_1s_4)c_5 - c_1s_{23}s_5)c_6 - (c_1c_{23}s_4 + s_1c_4)s_6 \\
r_{12} &= -((c_1c_{23}c_4 - s_1s_4)c_5 - c_1s_{23}s_5)s_6 - ((c_1c_{23}s_4 + s_1c_4)c_6 \\
r_{13} &= ((c_1c_{23}c_4 - s_1s_4)s_5 + c_1s_{23}c_5 \\
x &= ((c_1c_{23}c_4 - s_1s_4)s_5 + c_1s_{23}c_5)L_4 + c_1s_{23}L_3 + c_1c_2L_2 - s_1L_1 \\
r_{21} &= ((s_1c_{23}c_4 + c_1s_4)c_5 - s_1s_{23}s_5)c_6 - (s_1c_{23}s_4 - c_1c_4)s_6 \\
r_{22} &= -((s_1c_{23}c_4 + c_1s_4)c_5 - s_1s_{23}s_5)s_6 - (s_1c_{23}s_4 - c_1c_4)c_6 \\
r_{23} &= ((s_1c_{23}c_4 + c_1s_4)s_5 + s_1s_{23}c_5 \\
y &= ((s_1c_{23}c_4 + c_1s_4)s_5 + s_1s_{23}c_5)L_4 + s_1s_{23}L_3 + s_1c_2L_2 + c_1L_1 \\
r_{31} &= -(s_{23}c_4c_5 + c_{23}s_5)c_6 + s_{23}s_4s_6 \\
r_{32} &= (s_{23}c_4c_5 + c_{23}s_5)s_6 + s_{23}s_4c_6 \\
r_{33} &= c_{23}c_5 - s_{23}c_4s_5 \\
z &= (c_{23}c_5 - s_{23}c_4s_5)L_4 + c_{23}L_3 - s_2L_2
\end{aligned}
$$

Nelle formule illustrate s_i rappresenta il $sin(\theta_i)$ ed analogamente c_i rappresenta $cos(\theta_i)$. Il doppio pedice indica la somma di angoli, così che s_{23} rappresenta $sin(\theta_2+\theta_3)$. I termini L_i rappresentano le lunghezze dei bracci, come in riferimento alla tabella in appendice B.2.

4.2.2 Cinematica Inversa

Nel problema cinematico inverso si deve ricavare, conoscendo un punto nello spazio operativo descritto tramite parametri cinematici, la configurazione dei valori dei giunti che consentono il posizionamento del manipolatore in tal punto. Il metodo utilizzato per la soluzione della cinematica inversa del Manus è quello analitico. La soluzione analitica non sempre esiste, e se esiste può ammettere una o più soluzioni. Nel caso di un manipolatore a sei giunti rotoidali come il Manus la presenza di un polso sferico come organo terminale del robot garantisce l'esistenza di almeno una soluzione in forma chiusa. Inoltre il Manus è un manipolatore con giunti tutti ro-

toidali di cui gli ultimi 3 (relativi al polso) hanno gli assi che si intersecano tutti in un unico punto. Questa peculiarità consente di risolvere il problema cinematico inverso mediante la tecnica del disaccoppiamento cinematico [37], ossia ricavando la soluzione completa in due passi. Se consideriamo i parametri cinematici, $x, y, z, roll(\gamma), pitch(\beta), yaw(\alpha)$, si ha che il semplice posizionamento alle coordinate indicate dipende unicamente dai primi tre giunti, mentre l'orientamento della terna utensile, cioè del gripper, dipende unicamente dagli ultimi tre giunti. La posizione del polso si ricava facilmente con la matrice R_6^0 (anche se non è necessario calcolarla interamente):

$$p_3^0 = p_6^0 - L_4 R_6^0 \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} = \begin{bmatrix} x - L_4(c_\alpha s_\beta c_\gamma + s_\alpha s\gamma) \\ y - L_4(s_\alpha s_\beta c_\gamma - c_\alpha s_\gamma) \\ z - L_4 c_\beta c_\gamma \end{bmatrix} \tag{4.5}$$

Tutte le variabili in (4.5) sono note e dunque x_w, y_w, z_w sono valori calcolati. É possibile confrontare queste formule con quelle della matrice di trasformazione omogenea T_4^0 della cinematica diretta (4.4), dove però ora $\theta_1, \theta_2, \theta_3$ sono incognite:

$$p_3^0 = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} = \begin{bmatrix} c_1 s_{23} L_3 + c_1 c_2 L_2 - s_1 L_1 \\ s_1 s_{23} L_3 + s_1 c_2 L_2 + c_1 L_1 \\ c_{23} L_3 - s_2 L_2 \end{bmatrix}$$

Si ricava dunque la soluzione relativa alle prime tre variabili di giunto:

$$\tilde{\theta}_1 = 2arctan(\frac{-x_w \pm \sqrt{x_w^2 + y_w^2 - L_1}}{y_w + L_1})$$

$$\tilde{\theta}_3 = arcsin(\frac{x_w^2 + y_w^2 + z_w^2 - L_1^2 - L_2^2 - L_3^2}{2L_2L_3})$$
$$\tilde{\theta}_3 = \pi - arcsin(\frac{x_w^2 + y_w^2 + z_w^2 - L_1^2 - L_2^2 - L_3^2}{2L_2L_3})$$

$$\tilde{\theta}_2 = atan2(\frac{c_3L_3(c_1x_w + s_1y_w) - z_w(s_3L_3 + L_2)}{z_wc_3L_3 + (s_3L_3 + L_2)(c_1x_w + s_1y_w)})$$

Dove devono essere rispettate le condizioni:

$$\tilde{\theta}_1 : \qquad x_w^2 + y_w^2 \geqslant L_1^2$$

$$\tilde{\theta}_3 : \qquad L_1^2 + (L_2 + L_3)^2 \leqslant x_w^2 + y_w^2 + z_w^2 \leqslant L_1^2 + (L_2 + L_3)^2$$

$\tilde{\theta}_2$ dipende univocamente dai valori di $\tilde{\theta}_1$ e $\tilde{\theta}_3$. Le formule presentate restituiscono quattro possibili soluzioni al problema; è dunque necessario operare una scelta. La realizzazione della cinematica inversa utilizzata in questa tesi per il controllo del Manus, considera la soluzione più vicina alla configurazione attuale, ossia quella che minimizza la norma della distanza tra i vari giunti, tenendo conto della circolarità del dominio dei giunti.

$$\begin{bmatrix} \tilde{\theta}_1 \\ \tilde{\theta}_2 \\ \tilde{\theta}_3 \end{bmatrix} = min_i \begin{bmatrix} ||\tilde{\theta}_{i,1} - \theta_{0,1}|| \\ ||\tilde{\theta}_{i,2} - \theta_{0,2}|| \\ ||\tilde{\theta}_{i,1} - \theta_{0,1}|| \end{bmatrix} \tag{4.6}$$

In (4.6) $\tilde{\theta}_{i,j}$ rappresenta la soluzione *i*-esima per il *j*-esimo giunto, mentre $\theta_{0,j}$ rappresenta l'attuale configurazione del *j*-esimo giunto. Con i primi tre giunti è ora possibile ricavare $R_6^3(\tilde{\theta})$ mediante R_6^0 della cinematica diretta (4.4):

$$R_6^3(\tilde{\theta}) = R_3^0(\tilde{\theta})^{-1} R_6^0$$

Gli elementi della matrice dipendono dalla soluzione trovata al passo precedente, e dalla orientazione finale che si vuole ottenere:

$$\begin{aligned}
r_{11} &= c_\beta c_{23} c_{1-\alpha} + s_\beta s_{23} \\
r_{12} &= s_\beta s_\gamma c_{23} c_{1-\alpha} + c_\gamma c_{23} s_{1-\alpha} - c_\beta s_\gamma s_{23} \\
r_{13} &= s_\beta s_\gamma c_{23} c_{1-\alpha} + s_\gamma c_{23} s_{1-\alpha} - c_\beta c_\gamma s_{23} \\
r_{21} &= c_\beta s_{\alpha-1} \\
r_{22} &= c_\gamma c_{1-\alpha} + s_\beta s_\gamma s_{\alpha-1} \\
r_{23} &= s_\beta c_\gamma s_{\alpha-1} - s_\gamma c_{\alpha-1} \\
r_{31} &= c_\beta s_{23} c_{1-\alpha} - s_\beta c_{23} \\
r_{32} &= s_\beta s_\gamma s_{23} c_{1-\alpha} + c_\gamma s_{23} s_{1-\alpha} + c_\beta s_\gamma c_{23} \\
r_{33} &= s_\beta c_\gamma s_{23} c_{\alpha-1} + s_\gamma s_{23} s_{\alpha-1} + c_\beta c_\gamma c_{23}
\end{aligned}$$

La stessa matrice può essere espressa in funzione delle ultime tre variabili di giunto, che ora sono le incognite:

$$R_6^3(\theta) = \begin{bmatrix} c_4 c_5 c_6 - s_4 s_6 & -c_4 c_5 s_6 - s_4 c_6 & c_4 s_5 \\ s_4 c_5 c_6 + c_4 s_6 & -s_4 c_5 s_6 + c_4 c_6 & s_4 s_5 \\ -s_5 c_6 & s_5 s_6 & c_5 \end{bmatrix}$$

Eguagliando le due matrici si ricava un sistema la cui soluzione permette di ricavare le ultime variabili di giunto:

$$\begin{aligned} \hat{\theta}_4 &= atan2(r_{23}, r_{13}) & \hat{\theta}_4 &= atan2(-r_{23}, -r_{13}) \\ \hat{\theta}_5 &= atan2(\sqrt{r_{13}^2 + r_{23}^2}, r_{33}) & \hat{\theta}_5 &= atan2(-\sqrt{r_{13}^2 + r_{23}^2}, r_{33}) \\ \hat{\theta}_6 &= atan2(r_{32}, -r_{31}) & \hat{\theta}_6 &= atan2(-r_{32}, r_{31}) \end{aligned}$$

La soluzione trovata è valida se $sin(\hat{\theta}_5) \neq 0$ altrimenti la soluzione alternativa risulta:

$$\begin{aligned} \hat{\theta}_5 &= 0 & \hat{\theta}_5 &= \pi \\ \hat{\theta}_6 &= \theta_{0,5} & \hat{\theta}_6 &= \theta_{0,5} \\ \hat{\theta}_4 &= atan2(r_{21}, r_{11}) - \hat{\theta}_6 & \hat{\theta}_4 &= atan2(-r_{21}, -r_{11}) + \hat{\theta}_6 \end{aligned}$$

Anche in questo caso le formule propongono soluzioni multiple; a differenza del caso precedente la scelta è obbligata alla soluzione che presenta valori positivi della rotazione del giunto 5 in quanto la meccanica del Manus impedisce posizionamenti negativi per tale giunto.

Bisogna considerare alcuni aspetti del manipolatore:

- i sei giunti del Manus non sono del tutto indipendenti tra loro; esiste infatti un accoppiamento tra il secondo ed il terzo giunto: ad una rotazione del secondo giunto, il controllore del Manus compie in maniera trasparente una rotazione contraria del terzo giunto. Questo al fine di mantenere un'orientazione costante dell'asse del terzo braccio.
- I valori restituiti dagli encoder non sono concordi con la notazione Denavit-Hartenberg.

Per questi motivi, sia prima della cinematica diretta che prima della cinematica inversa è necessario operare degli aggiustamenti sugli angoli sia per compensare

l'accoppiamento (4.7), sia per rendere i valori consistenti alla notazione Denavit Hartenberg e riportarli nell'intervallo $[-\pi, \pi]$ (4.10)

$$\dot{\theta_3} = \theta_3 + \pi/2 - \theta_2 \tag{4.7}$$

$$\ddot{\theta_2} = -\theta_2 \tag{4.8}$$

$$\begin{cases} \dot{\theta_3} \geqslant 0 & \ddot{\theta_3} = \pi - \theta_3 \\ \dot{\theta_3} \leqslant 0 & \ddot{\theta_3} = -\pi - \theta_3 \end{cases} \tag{4.9}$$

$$\ddot{\theta_4} = \theta_4 + \pi \tag{4.10}$$

Inoltre gli angoli di giunto restituiti dalla cinematica inversa sono conformi alla notazione DH mentre quelli del Manus no, quindi anche nel controllore software sarà necessario tenere conto degli aggiustamenti visti.

Capitolo 5

Architettura del sistema

Come discusso nella sez.1.4, il problema di visual servoing è intrinsicamente modulare, pertanto la struttura generale del programma è organizzabile secondo un modello client/server:

- La parte client costituisce l'interfaccia in cui avviene la generazione in *computer graphics* dell'ambiente virtuale, ed attraverso la quale l'utente specifica la collocazione del robot e il compito da eseguire.
- La parte server riceve l'ordine relativo al compito e lo esegue.

In particolare sono presenti due server: uno è il simulatore che esegue il compito nello spazio virtuale e lo mostra all'utente sottoforma di animazione computerizzata; l'altro è il server reale che si occupa dell'esecuzione del compito nel spazio reale, mediante la navigazione e l'utilizzo del visual servoing. La parte client e la parte di server simulato sono già realizzate dal programma PbD, mentre la parte del server reale è stata progettata e realizzata nell'ambito in questa tesi.

Il server reale si deve occupare del controllo diretto dei sottosistemi robotici in base alle direttive ricevute dal client. Dato che lo scopo da perseguire è la realizzazione di un sistema autonomo, la comunicazione tra client e server (reale) è minima, e si limita all'informazione relativa il compito da eseguire. Il client non vede il server come un sistema robotico, ma come un esecutore di compiti; pertanto l'implementazione del server deve essere realizzata mediante una classe che metta a disposizione le possibili operazioni robotiche che il sistema è in grado di eseguire, mascherando struttura e capacità dei singoli robot. Sarà poi il server a richiamare le singole

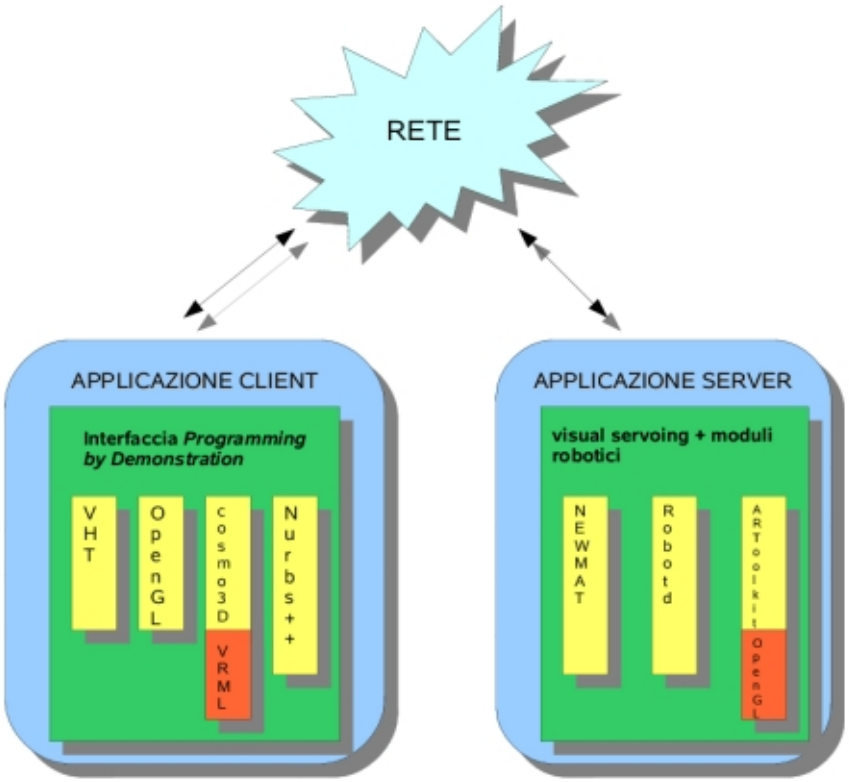


Figura 5.1: Schema generale dell'architettura proposta per il sistema di visual servoing.

funzionalità dei robot, a seconda del compito assegnato. Vista l'eterogeneità dei sottositemi da controllare, la scelta di realizzare classi diverse per ogni dispositivo fisico non è solo una questione di stilistica per mantenere la modularità, ma rappresenta una una via obbligata. Avremo quindi una classe per l'interfacciamento del server con il client, che si appoggerà su altre classi per la realizzazione delle azioni robotiche. Seguendo lo schema in fig.1.12 sono state realizzate le classi *thread* per il controllo del Nomad, per il controllo del Manus, e per la visione.
In fig.5.2 è presente uno schema UML delle classi realizzate; come illustrato una classe Master mette a disposizione delle funzioni che corrispondono ai compiti robotici che il sistema è in grado di seguire. A seconda del compito selezionato, venogono richiamati i moduli che controllano i singoli robot. Ad esempio per l'azione di spostamento si richiama la classe Nomad, mentre per la presa di oggetti ci si ap-

poggia sulla classe Manus.
Le tre classi principali Manus, Vision, Nomad, ereditano dalla classe Thread; all'interno della classe Vision, viene richiamato un metodo della classe Manus per risolvere il problema della comunicazione dei dati di visione, pertanto le due classi sono poste in relazione. In figura si nota anche che la classe Manus si appoggia sulla classe Cinematica, per la realizzazione dei metodi di spostamento del manipolatore.

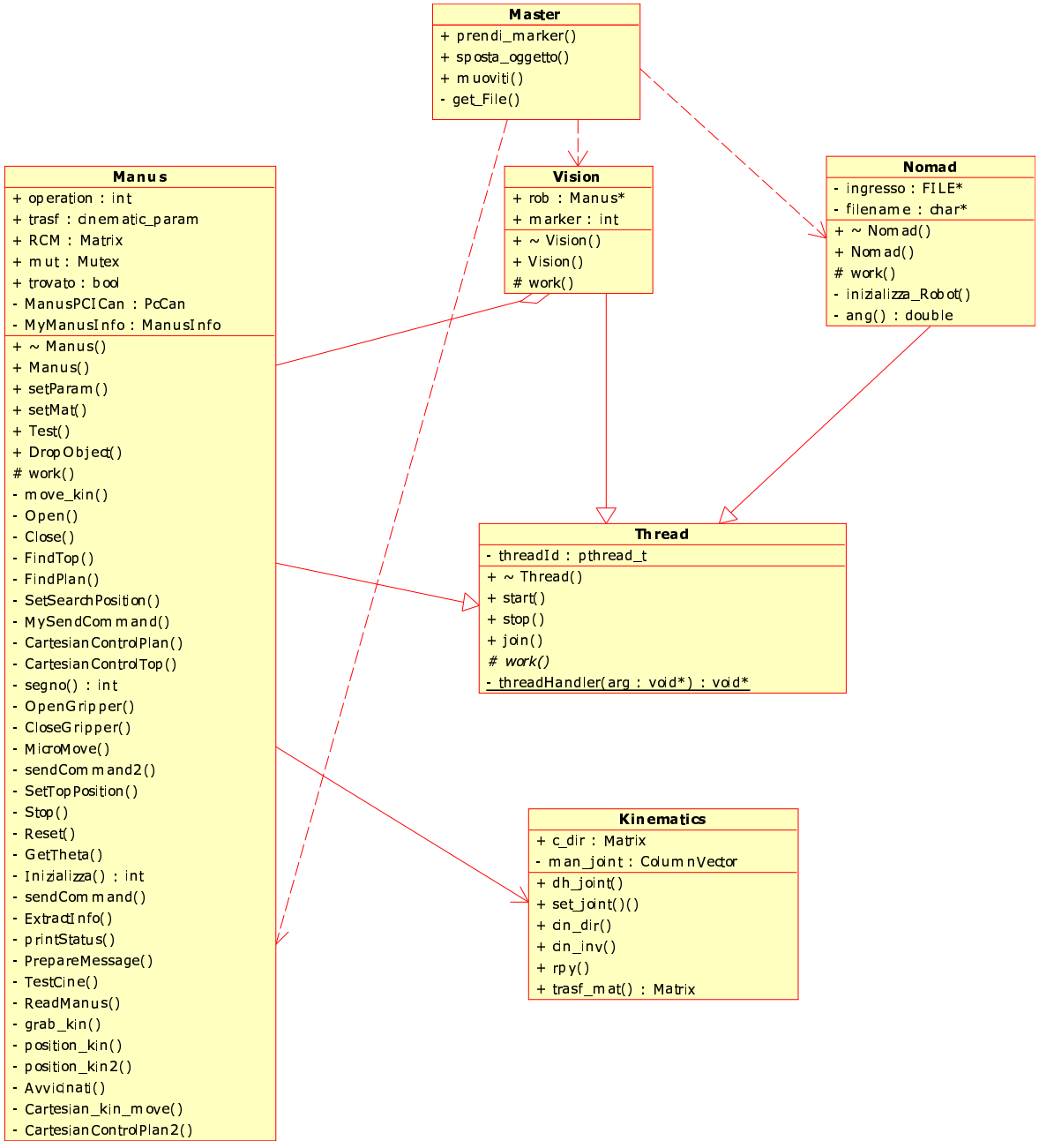


Figura 5.2: Diagramma delle classi del sistema software realizzato.

5.1 PbD

Il software PbD (*Programming by Demonstration*) è stato realizzato presso il laboratorio di robotica dell'Università di Parma [1]. Nell'applicazionedi interesse per questa tesi, esso viene utilizzato come sistema di interazione uomo-robot basato su realtà virtuale simulando un ambiente reale (nel caso parte della Palazzina 1 della sede scientifica della Facoltà di Ingegneria) in cui è presente un robot mobile in grado di eseguire compiti di manipolazione.
Il sistema si propone come interfaccia semplificata per l'utente, che, attraverso dispositivi di puntamento intuitivi come mouse e guanto, è in grado di pianificare ed eseguire (virtualmente) compiti robotici.
In questa tesi è utilizzata la modalità di esecuzione tramite dimostrazione: l'utente si limita a specificare un compito che ora non solo viene simulato, ma anche eseguito avvalendosi del programma di visual servoing.
Inizialmente il programma apre una finestra grafica, visualizzando la pianta dell'ambiente virtuale simulato realizzato come file VRML in cui è definita una struttura gerarchica di oggetti che compongono la scena. I file VRML vengono elaborati mediante la libreria Cosmo3D, per la generazione di un albero di oggetti che rappresenta la struttura dati sopra la quale opera direttamente il programma durante la simulazione.
Nella versione originale venivano usati due differenti modelli di ambiente, utilizzando due diversi file VRML: uno per la rappresentazione dell'ambiente in pianta e per la visualizzazione della traiettoria, l'altro per la modalità di dimostrazione ed esecuzione del compito robotico. In questa tesi, il programma è stato modificato per utilizzare un modello unico per entrambe le fasi, sfruttando le diverse modalità di visualizzazione della telecamera virtuale. In questo modo per eventuali modifiche dell'ambiente virtuale basta aggiornare un solo file.
In fig.5.3, è mostrato come all'interno della finestra si posiziona il robot mediante la pressione del tasto sinistro del mouse, per poi specificare la traiettoria che il sistema robotico deve eseguire. Sempre premendo il tasto sinistro si impostano dei punti di via, contrassegnati da delle "x", e alla pressione del tasto destro viene visualizzato il percorso impostato mediante una linea rossa come visibile in fig.5.4. Questa linea rappresenta la traiettoria ottenuta dai punti contrassegnati inseriti mediante una

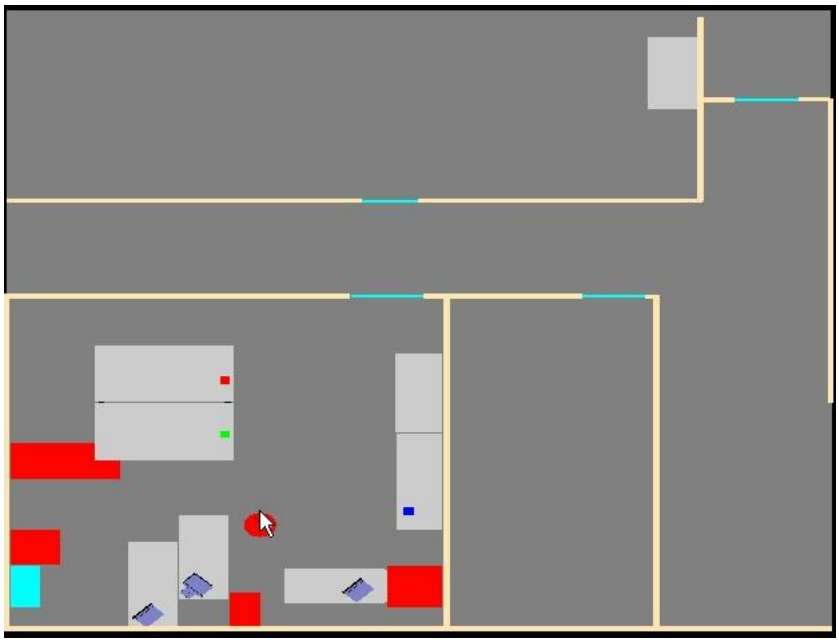


Figura 5.3: Posizionamento del robot nell'ambiente virtuale.

curva Nurbs interpolante. Chiudendo la finestra viene avviata la simulazione: i dati relativi alla traiettoria vengono inviati al server di simulazione che li utilizza per la visualizzazione del movimento del robot. Le stesse informazioni devono essere inviate al server reale in ascolto sul Nomad per l'esecuzione del movimento. In questo caso però viene eseguito un sottocampionamento, visto che i punti che compongono la traiettoria sono centinaia, e darebbero luogo ad una risoluzione inferiore a quella praticabile col Nomad. Chiudendo la finestra, i dati vengono inviati mediante *socket* al Nomad, ed inizia la visualizzazione di movimento nel server simulato. A questo punto il programma passa alla visualizzazione dell'ambiente in assonometria, in cui è presente una mano virtuale antropomorfa, presentata in fig.5.1. I movimenti che l'utente esegue col guanto nel mondo reale sono rilevati e salvati dal dispositivo tracker ed inviati al server virtuale, che li riporta nel sistema di riferimento virtuale. Muovendo la propria mano l'utente seleziona l'oggetto nell'ambiente virtuale, che rappresenta l'obbiettivo del compito di manipolazione. Un identificatore viene inviato al server reale presente sul Nomad, per indicare all'algoritmo di visual servoing quale oggetto prendere fra quelli presenti nella scena reale.

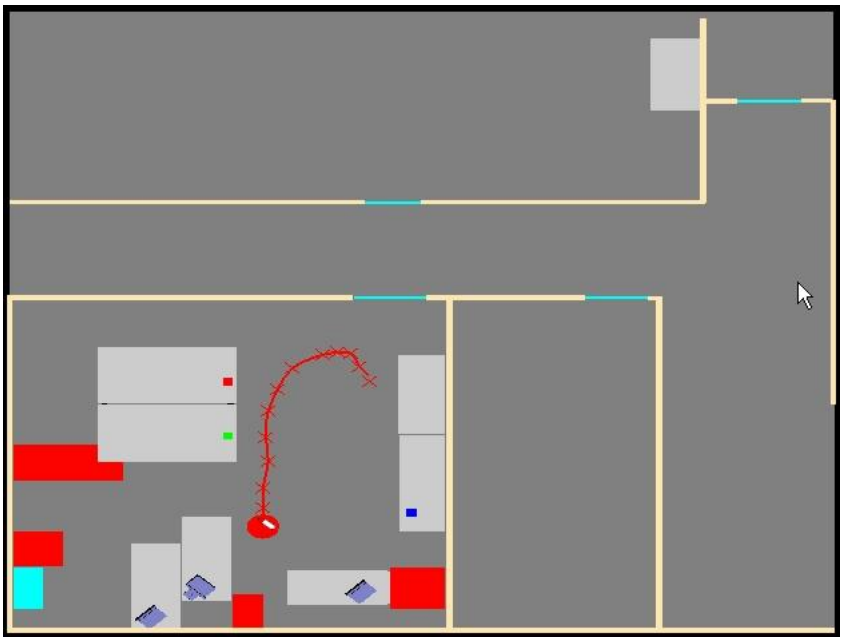


Figura 5.4: Immissione della traiettoria del robot.

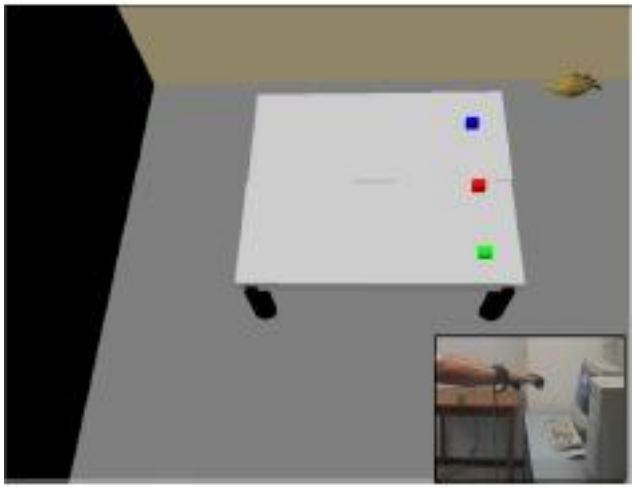


Figura 5.5: Indicazione del compito robotico col guanto.

5.2 Modulo Master

Il modulo master costituisce l'interfaccia per il sistema robotico proposto, mettendo a disposizione dell'utente le macrooperazioni che il robot può eseguire. È realizzato mediante una classe in cui i metodi rappresentano le operazioni richiamabili da lato client. La classe non ha alcuna variabile membro; si limita invece a creare degli oggetti locali ai metodi. Queste variabili sono istanze degli oggetti che rappresentano i due robot del sistema Manus e Nomad rispettivamente. Si analizzano ora i singoli metodi costituenti la classe, fornendo per ciascuno di essi una breve spiegazione:

- `sposta_oggetto( ... )` consente di specificare al `Manus` dove posare un oggetto. Le coordinate e l'orientazione sono relative alla terna di riferimento del `Manus`. La funzione si limita a creare un oggetto `Manus` e a passargli la posizione finale dell'oggetto. Attualmente non viene eseguito nessun controllo per verificare se il manipolatore è effettivamente in possesso dell'oggetto. L'oggetto `Manus` viene distrutto una volta che il compito è stato eseguito.

- `muovi( ... )` istanzia un oggetto `Nomad` passandogli un file in cui sono contenuti i punti relativi alla traiettoria da eseguire per lo spostamento della base mobile. La funzione si occupa anche della ricezione del file da lato server, richiamando al proprio interno il metodo privato `ricevi_file()`.

- `ricevi_file( ... )` si pone in "ascolto" su una *socket* in attesa che il client, cioè l'interfaccia utente, comunichi i dati relativi alla traiettoria. Le informazioni vengono salvate su FILE che poi verrà passato prima al metodo chiamante `muovi( ... )`, poi all'istanza `Nomad` creata.

- `presa_oggetto( ... )` è il metodo che istanzia due oggetti `Manus` e `Vision`, per avviare il compito di visual servoing. All'oggetto `Vision` viene passato un identificatore dell'oggetto della manipolazione, prevedendo che sulla scena ne siano presenti diversi, ognuno identificato da un marker proprio.

Come illustrato, questa classe non esegue praticamente nessuna operazione,ma funge da contenitore/istanziatore di oggetti, delegando l'esecuzione di compiti agli

```
class Master  {
public:
 void sposta_oggetto(double x,double y,double z
        ,double roll,double pitch,double yaw);
 void muovi(FILE* nurbs);
 void muovi(char* nurbs);
 void presa_oggetto(int m);
protected:
private:
void ricevi_file(char *filename);
};
```

Listato 5.1: Master Class Header

oggetti robotici corrispondenti. Gli oggetti robotici sono tutti classi *thread* le cui operazioni vengono eseguite in maniera concorrente.

5.3 Modulo Nomad

Questo modulo comanda la base mobile del sistema robotico, che nel sistema proposto è costituito dal robot Nomad. La traiettoria viene creata e pianificata all'interno del programma PbD: l'utente posiziona il robot nell'ambiente virtuale e imposta una traiettoria "cliccando" col mouse. In questo modo si specificano dei *via-point* che la base mobile insegue, mentre la traiettoria completa viene calcolata tramite interpolazione (attraverso il calcolo di una *spline* con NURBS++ (sez.3.4)). Ciò che si ottiene è un insieme di punti nello spazio, definito come insieme di coordinate relative al sistema di riferimento dell'ambiente virtuale.
La traiettoria a questo punto viene sottocampionata ed esportata all'interno di un file di testo; in questo modo, tutto il software "a valle" è in grado di operare sui dati forniti senza il bisogno di utilizzare Nurbs++. Il file viene rispedito mediante *socket* ad un programma server attivo sul Nomad, che lo riceve e ne salva una copia locale.
Il software di comando sfrutta le librerie *robotd* 2.1, che mettono a disposizione semplici istruzioni per lo spostamento ed il controllo del Nomad. La traiettoria a questo punto è costituita da una "spezzata", ancora riferita al sistema di riferimen-

Algoritmo 1 main

```
1: Inizializzazione robotd
2: Apertura del file di coordinate locale
3: Impostazione del primo punto come coordinata attuale
4: while punti della traiettoria disponibili do
5:     Lettura da file del prossimo punto
6:     Calcolo degli spostamenti nel sistema locale
7:     Esecuzione movimento
8:     Impostazione del punto successivo come posizione attuale
9: end while
10: return
```

to virtuale, perciò il sistema calcola gli spostamenti necessari per arrivare al punto successivo nel sistema di riferimento locale. I comandi di spostamento vengono eseguiti attraverso le funzioni messe a disposizione da *robotd*, e viene ricalcolato lo spostamento da compiere per arrivare al punto successivo. Il modulo Nomad è stato implementato con una classe che esegue unicamente il compito di spostamento, nel momento in cui la variabile viene istanziata. Anche se il compito di navigazione viene eseguito in maniera autonoma dai moduli di manipolazione e visione, è stato scelto di implementarlo comunque come una *thread* nel caso in futuro si vogliano aggiungere azioni più complesse, come spostamenti di oggetti da una stanza all'altra, o comportamenti avanzati come elusione di ostacoli. Inoltre in questo modo si mantiene la struttura più modulare possibile, rispettando le scelte di progetto iniziali.

5.4 Modulo Vision

Anche il modulo di visione è stato realizzato ereditando dall'oggetto *thread*: l'oggetto viene creato ed istanziato; viene poi fatto eseguire il suo metodo *work*. L'unico dato locale di questa classe è un puntatore all'oggetto *Manus* che serve, come illustrato più avanti, per esportare i dati riguardanti la visione. Al momento della creazione dell'oggetto, la variabile riceve come dato iniziale un identificatore del marker scelto per il compito di manipolazione.
Si distingue allora `marker_selezionato`, come quello corrispondente all'operazione robotica, `marker_impostato`, come uno dei marker che il sistema riconosce come validi, e come `marker` il vettore dei possibili marker rilevati all'interno dell'immagine da ARToolkit. L'elenco dei marker "validi" è contenuto all'interno di un file che viene letto al momento dell'inizializzazione del processo di visione di ARToolKit. Il metodo *work* si limita a richiamare la funzione di visione, la quale si occupa innanzi tutto dell'inizializzazione della telecamera per poi iniziare con la visione vera e propria. L'algoritmo di visione artificiale sfrutta le caratteristiche messe a disposizione dalla libreria ARToolkit 3.3: il flusso di programma richiama la funzione argMainLoop() che attraverso una CallBack richiama all'infinito la funzione MainLoop(). È infatti proprio MainLoop() il nucleo del sottosistema di visione: Su tutti i marker rilevati e appartenenti all'elenco di marker impostati viene sovrapposta nella finestra grafica un oggetto virtuale, ottenuto mediante chiamate OpenGL. Sui marker riconosciuti viene disegnata una cornice verde, mentre in corrispondenza del marker selezionato viene disegnato un quadrato rosso.
Il ciclo di CallBack attraverso il quale si continua a richiamare la funzione MainLoop è infinito, e termina solo se viene chiusa la finestra grafica o se in corrispondenza ad essa viene premuto il tasto ESC.
Il fatto di avere la funzione di elaborazione dell'immagine all'interno di un ciclo callback impone il problema dell'esportazione dei dati. Infatti le variabili locali alla funzione MainLoop non sono accessibili all'esterno di tale funzione, mentre modifiche a variabili globali, all'interno della funzione MainLoop, non vengono applicate. Tale problema è stato risolto creando un puntatore all'oggetto `Manus`. In questo modo i dati relativi alla posizione del marker vengono passati come parametri ad un metodo dell'oggetto. La modifica ed il salvataggio di tali dati non avviene dunque

Algoritmo 2 MainLoop

```
1: Acquisizione frame video (arVideoGetImage)
2: trovato=false;
3: Rilevamento dei markers nel frame acquisito (arDetectMarker)
4: for i = 1, ··· , n_marker_rilevati do
5:    for j = 1, ··· , n_marker_impostati do
6:       if (Marker[i]==Marker_impostato[j] ) then
7:          imposto Marker[i] come marker riconosciuto.
8:       end if
9:       if (Marker[i] == Marker_selezionato) then
10:         Estrazione della matrice di trasformazione da Marker[i] (arGetTran-
            sMat) in patt_trans
11:         manus->trovato=true
12:      end if
13:   end for
14: end for
15: if manus->trovato==true then
16:   Estrazione di posizione e angoli di orientamento da patt_trans
17:   Salvataggio delle informazioni dentro manus->trasf
18:   Visualizzazione dell'ogetto virtuale in corrispondenza dei marker trovati
19: end if
20: return
```

nella classe `Vision`, ma direttamente nella classe `Manus`. Durante la fase di avvicinamento, la struttura dati viene modificata dalla classe `Manus` che ne scambia i valori dei campi per riportare i dati nel sistema di riferimento locale. Se la classe `Vision` aggiornasse le informazioni prima che il Manus abbia eseguito il movimento, avremmo parte dei dati riferiti a sistemi di riferimento diversi. Perciò è stato utilizzato un semaforo per regolare l'accesso alla struttura dati in modalità esclusiva. Nel tempo in cui la visione elabora l'immagine, la struttura non è trattenuta, e i dati sono liberamente accessibili alla classe `Manus`, viceversa `Manus` trattiene la struttura nel momento in cui deve impostare il movimento da eseguire.
I dati salvati sono la posizione, espressa come coordinate *x,y,z*, e gli angoli di orientamento *roll,pitch,yaw*. Si poteva direttamente salvare la matrice di trasformazione, ma in questo modo è più semplice effettuare correzioni sui dati. Ad esempio è presente uno sfasamento di 180° sull'angolo di *roll* che viene corretto da una semplice sottrazione, mentre usando la matrice di trasferimento si sarebbe dovuto compensare lo sfasamento con una moltiplicazione per una matrice di rotazione (tale matrice dovrebbe compensare l'angolo senza invertire gli assi). Inoltre nella fase di avvicinamento (che avviene in modalità cartesiana) tali informazioni avrebbero dovuto essere estratte (e corrette) comunque dalla matrice di trasformazione.

5.5 Modulo Manus

In questo modulo viene racchiuso tutto il codice che si occupa del comando e controllo del manipolatore; l'implementazione si basa sulla realizzazione di una classe "Manus" che mette a disposizione dei metodi per la ricerca e la cattura dell'oggetto. Il codice che si interfaccia a basso livello con la Control Box si basa in parte su lavori precedentemente svolti sul Manus [34] e si occupa della lettura e scrittura dei messaggi, mentre le funzioni di livello di astrazione più alto sono state implementate *ex-novo*. Il problema che si pone, nella ricerca del marker, è quello di impostare una posizione ed orientazione del braccio robotico tale per cui la telecamera posta sulla pinza riesca a vedere e riconoscere l'obbiettivo. La soluzione adottata semplifica il problema a due casi generali:

1. Il marker si trova circa in posizione normale rispetto al piano d'appoggio

2. Il marker è circa parallelo al piano d'appoggio

Nelle due condizioni descritte si fanno rientrare tutte le possibili posizioni ed orientazioni del bersaglio, per cui non vengono imposte limitazioni.
Prima di intraprendere la ricerca vera e propria del *target* il manipolatore esegue la manovra di *fold-out*, un movimento preimpostato del controllore hardware del Manus che apre il braccio a partire dalla posizione richiusa (detta di *fold-in*). Il braccio assume delle posizioni prefissate per iniziare un movimento circolare tale per cui la telecamera, che segue una circonferenza con centro il corpo centrale, "spazza" l'ambiente alla ricerca del marker. Quando il marker entra nella visuale della microcamera e viene riconosciuto dal programma di visione, il movimento rotante viene interrotto ed inizia la fase di avvicinamento. Nel caso invece la ricerca non abbia avuto successo, il braccio si porta nella posizizione di ricerca "dall'alto" ed inizia un nuovo movimento rotatorio per la ricerca del marker (la rotazione avviene nel verso contrario rispetto a quella effettuata per la ricerca in piano).
Una volta trovato il marker, sia nel caso in piano che dall'alto, il programma esegue tre fasi distinte: l'avvicinamento, il posizionamento, e la presa. L'avvicinamento, tramite le informazioni fornite dalla visione, realizza i movimenti per portarsi in direzione del bersaglio comandando il robot attraverso la modalità cartesiana della Control Box.
Purtroppo la cinematica cartesiana fornita dal controllore hardware del Manus consente di esprimere solo i parametri di traslazione x, y, z rispetto alla terna fissa per cui l'orientamento della videocamera rispetto al marker viene realizzato solo in parte. Infatti se il marker è relativamente vicino e con orientamento indicativo verso la terna fissa, che nel caso del Manus è nella base del manipolatore (corrispondente alla spalla del braccio robotico), la posizione di ricerca assunta in piano rende gli assi della microcamera e della terna fissa paralleli ed ortogonali (a meno di una semplice rotazione attorno ad un asse), per cui gli angoli di orientazione risultano solo scambiati. Al contrario se il marker risulta troppo lontano, o con orientazioni arbitrarie, gli assi della telecamera e del Manus non sono più allineati, e visto che la cinematica cartesiana del Manus restituisce solo la posizione rispetto alla terna fissa, non è possibile stabilire l'orientazione esatta.
La funzione MovePlan realizza l'avvicinamento della microcamera al marker mediante l'utilizzo della modalità di comando cartesiana. La funzione MoveTop è con-

Algoritmo 3 visualservoing

```
1: Esegui fold-out
2: Apri Gripper
3: Imposta posizione di ricerca "in piano"
4: while ((marker non trovato) ∨ (raggiunta rotazione massima)) do
5:    Continua la rotazione
6: end while
7: if (marker trovato) then
8:    Presa in piano (Moveplan)
9: else
10:   Imposta posizione di ricerca "dall'alto"
11:   while ((marker non trovato) ∨ (ragginta rotazione massima)) do
12:      Continua la rotazione
13:   end while
14:   if (marker trovato) then
15:      Presa dall'alto (movetop)
16:   else
17:      Marker non trovato
18:   end if
19: end if
20: return
```

Algoritmo 4 MovePlan

```
1: avvicinato←false
2: while (avvicinato≠true) do
3:    Attendi i dati dalla visione
4:    Riporta i parametri cinematici impostati dalla visione al sistema di
      riferimento fisso
5:    Imposta i parametri di spostamento cartesiani
6:    Imposta i parametri di orientazione
7:    if (sono sufficientemente vicino) then
8:       Diminuisci i parametri di spostamento
9:    end if
10:   Invia il comando di movimento alla Control Box
11:   Verifica se avvicinato=true
12: end while
13: Posizionamento
14: Presa
15: return
```

Algoritmo 5 MoveTop

```
1: avvicinato←false
2: while (avvicinato≠true) do
3:    Attendi i dati dalla visione
4:    Riporta i parametri cinematici al sistema di riferimento fisso
5:    Imposta i parametri di spostamento cartesiani
6:    if (sono sufficientemente vicino) then
7:       Diminuisci i parametri di spostamento
8:    end if
9:    Invia il comando di movimento alla Control Box
10:   Verifica se avvicinato=true
11: end while
12: Posizionamento
13: Presa
14: return
```

cettualmente uguale alla MovePlan, ciò che cambia a livello di codice sono le istruzioni per riportare i dati della visione al sistema di riferimento fisso. Nel caso di avvicinamento dall'alto, il problema dell'orientazione è ancora più critico, infatti l'avvicinamento (sempre nello spazio di lavoro cartesiano) avviene solo modificando la posizione del polso, in quanto modifiche anche minime dell'orientamento ruotano il sistema telecamera rispetto al sistema fisso in un modo non calcolabile per i motivi appena espressi. La fase di orientamento si troverà allora a dover esplorare in uno spazio più vasto rispetto al caso planare.

In entrambi i casi si rende necessario introdurre una fase di orientamento che funzioni diversamente dalle fase di avvicinamento; tale fase viene realizzata comandando il Manus nello spazio dei giunti utilizzando le funzioni di cinematica diretta ed inversa.

Il processo avviene per passi con un continuo *feedback* della visione: per ogni passo la posizione e l'orientazione attuali vengono aggiornate, mentre la posizione e ed orientazione finale ricalcolate; il processo si conclude quando gli errori sulla posizione e sull'orientamento sono sufficientemente piccoli.

La presa dell'oggetto deve avvenire per forza "alla cieca", cioè senza il suppor-

Algoritmo 6 Orientamento

```
1: Leggi Stato dei giunti
2: Calcola con la cinematica diretta la posizione attuale
3: posizionato←false
4: while (posizionato≠true) do
5:    Attendi i dati dalla visione
6:    Esegui la Trasformazione da Sistema Teleamera a Sistema Pinza
7:    Esegui la Trasformazione da Sistema Pinza a Terna 0
8:    Calcola con la Cinematica Inversa la posizione finale dei giunti
9:    Imposta i valori di spostamento di giunto
10:   Invia il comando di movimento alla Control Box
11:   if (errore su posizione ed orientazione sufficientemente piccolo) then
12:      posizionato←false
13:   end if
14:   Leggi Stato dei giunti
15:   Calcola con la cinematica diretta la posizione attuale
16: end while
17: return
```

Algoritmo 7 Presa

```
1: Leggi Stato dei giunti
2: Calcola con la cinematica diretta la posizione attuale
3: Imposta un movimento verso l'alto (riferito al sistema telecamera)
4: Calcola con la cinemtica Inversa la posizione finale dei giunti
5: while (posizione attuale(giunti)≠posizione finale(giunti)) do
6:    Imposta valori di giunto (posizione finale-attuale)
7:    Invia il comando di movimento alla Control Box
8: end while
9: Leggi Stato dei giunti
10: Calcola con la cinematica diretta la posizione attuale
11: Imposta un movimento in avanti (riferito al sistema telecamera)
12: Calcola con la cinemtica Inversa la posizione finale dei giunti
13: while (posizione attuale(giunti)≠posizione finale(giunti)) do
14:   Imposta valori di giunto (posizione finale-attuale)
15:   Invia il comando di movimento alla Control Box
16: end while
17: Chiudi Gripper
18: return
```

to della visione.Infatti essendo microcamera e marker molto vicini ed essendo la pinza sotto la microcamera, per posizionare la pinza alla stessa altezza del marker, necessariamente il marker esce dalla scena ripresa dalla microcamera. Quest'ultimo movimento di presa viene eseguito in modo automatico, cioè è uguale in tutte le situazioni e viene realizzato comandando il manus nello spazio dei giunti attraverso le funzioni di cinematica.

5.6 Kinematic

Questa classe, a differenza delle precedenti, non interviene direttamente sul sistema robotico, ma fornisce le funzioni per il calcolo della cinematica diretta ed inversa, necessarie al controllo del Manus. I metodi adottati si limitano ad implementare le formule matematiche illustrate nel sez.4.2, utilizzando per la gestione delle matrici la libreria NewMat (sez.3.5).
La funzione `dh_joint(double old_j[6],double dh_j[6])` riceve in ingresso un vettore contenente i valori dei giunti, così come sono letti dagli encoder del Manus, ed effettua le correzioni opportune per renderli coerenti con la notazione Denavit Hartenberg.

`set_joint(double j[6])` imposta i valori di giunto all'interno della struttura `man_joint` della classe. Questo metodo deve essere eseguito sia prima di eseguire la cinematica diretta, sia prima di eseguire la cinematica inversa.

`cin_dir(...)` esegue il calcolo della cinematica diretta: in base ai valori di giunto presenti nella struttura `man_joint`, restituisce la posizione nello spazio di lavoro. Il risultato è espresso sottoforma di posizione (x, y, z) ed angolazione $(roll, pitch, yaw)$ (le unità di misura sono metri e radianti).

`cin_inv(...)` esegue il calcolo della cinematica inversa, riceve in ingresso la posizione nello spazio di lavoro e ritorna i valori di giunto corrispondenti. L'ingresso puó essere sottoforma di parametri cinematici $(x, y, z, roll, pitch, yaw)$, o sottoforma di matrice di trasformazione (sia come puntatore a `double` che come elemento `Matrix`).

```
class Kinematic{
public:
  void dh_joint(double old_j[6],double dh_j[6]);
  void set_joint(double j[6]);
  void cin_dir(double *x,double *y,double *z,
       double *roll,double *pitch,double *yaw);
  void cin_inv(double x,double y,double z,
       double gamma, double beta,
       double alpha,double t_new[6]);
  void cin_inv(double *mat, double t_new[6]);
  void cin_inv(Matrix A, double t_new[6]);
  void rpy(Matrix mat,double *roll, double *pitch,
       double *yaw);
  Matrix trasf_mat(double x,double y,double z,
       double roll,double pitch,double yaw);
  Matrix  c_dir;
private:
  ColumnVector man_joint;
};
```

Listato 5.2: Kinematic Class Header

`rpy()` e `trasf_mat()` sono funzioni di utilità: la prima estrae gli angoli di Eulero da una matrice di trasformazione 4×4, mentre la seconda restituisce la matrice di trasformazione corrispondente ai parametri cinematici forniti in ingresso.

Capitolo 6

Prove Sperimentali

In questo capitolo sono illustrati i vari test svolti durante lo sviluppo della tesi per la verifica della correttezza e delle prestazioni del sistema sviluppato. In una prima fase è stato testato l'algoritmo di *visual servoing* mantenendo il Manus nella sua postazione fissa. Nella fase finale il Manus è stato installato sul Nomad e sono stati realizzate prove facendo eseguire al sistema compiti di navigazione e ricerca/presa di oggetti.

6.1 Postazione fissa

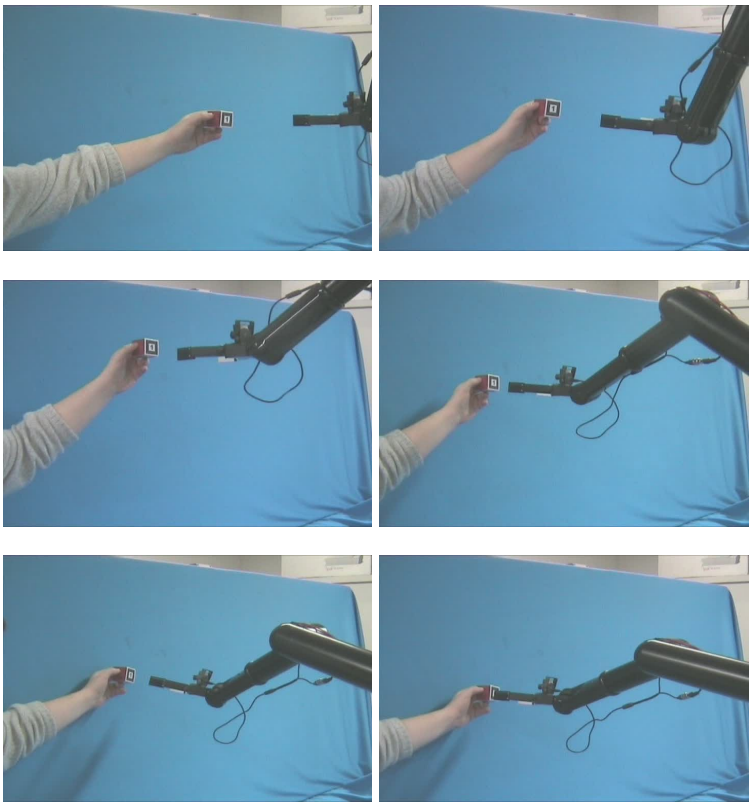


Figura 6.1: Esperimento1: inseguimento dell'oggetto.

Un primo test è stato svolto per verificare il comportamento dell'algoritmo di visual servoing nella fase di *tracking dinamico* di un oggetto. Nella prova eseguita, di cui sono mostrate alcune immagini in fig.6.1, è stato posto davanti alla microcamera un cubetto di legno sul quale era stato opportunamente apposto un marker[1]. Il movimento del cubo è stato eseguito cercando di mantenere sempre il marker nel campo visivo della microcamera; questo è stato possibile verificando in tempo reale, tramite l'output video fornito da ARToolkit e visibile in fig.6.2, le informazioni visive provenienti dalla telecamera. Dalle immagini é possibile notare come, nella visuale della microcamera, il marker non sia visibile, ma al suo posto sia presente

[1] Il cubo ed il marker sono entrambi di quattro centimetri di lato.

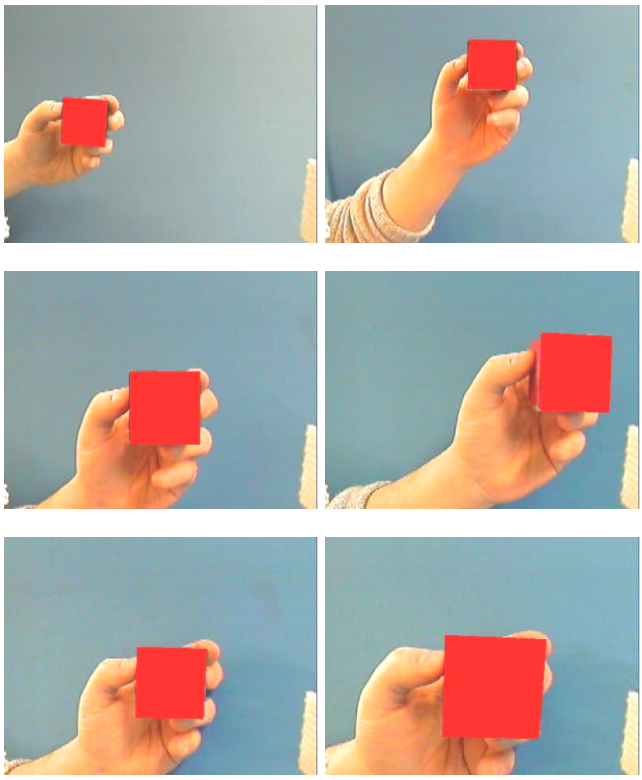


Figura 6.2: Esperimento1: inseguimento dell'oggetto vista della microcamera.

un quadrato rosso virtuale. Il test ha confermato l'efficacia dell'algoritmo di avvicinamento, il quale impone i movimenti al braccio robotico cercando di mantenere sempre il marker nel centro del campo visivo.

Un secondo tipo si esperimento è stato condotto per verificare la capacità di cattura di oggetti: Nelle fig.6.3,6.4 si possono osservare le immagini di una delle varie

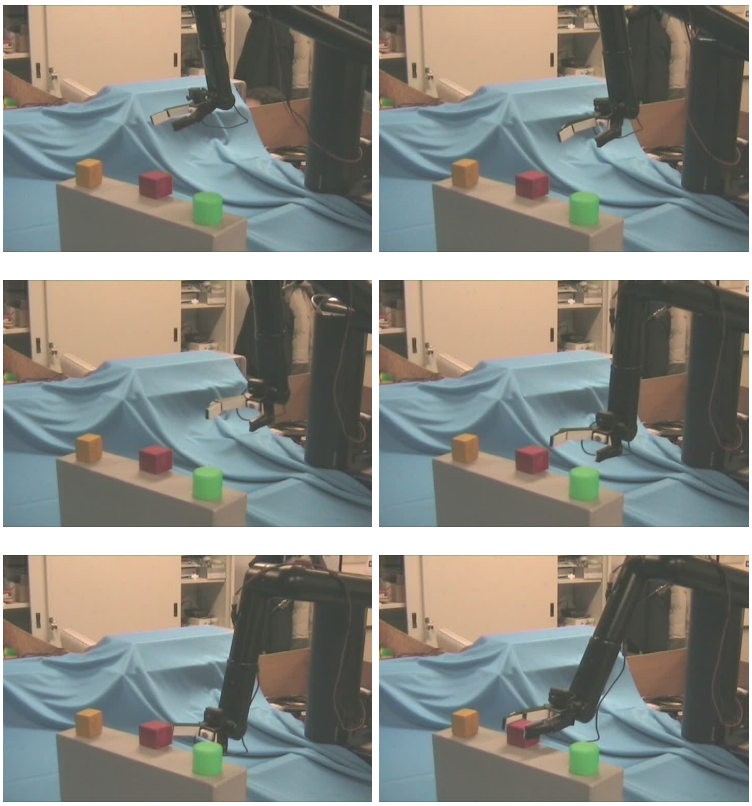


Figura 6.3: Esperimento2: Avvicinamento e presa.

prove effettuate. Da questo tipo di esperimento sono emersi problemi nella fase di posizionamento e presa, dovuti alla scarsa accuratezza che il manipolatore possiede nell'esecuzione di movimenti di piccola entità, dell'ordine di uno o due centimetri.

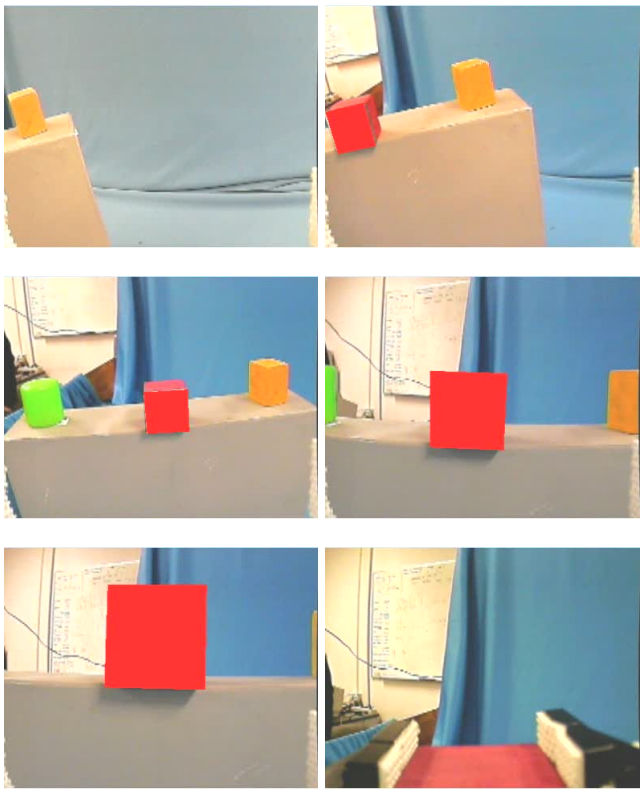


Figura 6.4: Esperimento2: Avvicinamento e presa vista della microcamera.

Durante il corso delle prove si è anche tenuto sotto controllo l'andamento generale della traiettoria 3D della pinza del braccio robotico. Elaborando i dati della visione è stato possibile definire una funzione di errore relativa alla posa della telecamera, e conseguentemente della pinza, rispetto al marker. Nei grafici proposti in fig.6.5 sono riportati i dati di alcuni esperimenti. L'errore menzionato a cui i dati si riferiscono è dato dalla distanza euclidea della telecamera rispetto al marker:

$$\epsilon = \sqrt{x^2 + y^2 + z^2}$$

il fatto che il sistema prosegue "nella esplorazione" nonostante abbia individuato un minimo, indica che in corrispondenza di tale configurazione non è stato raggiun-

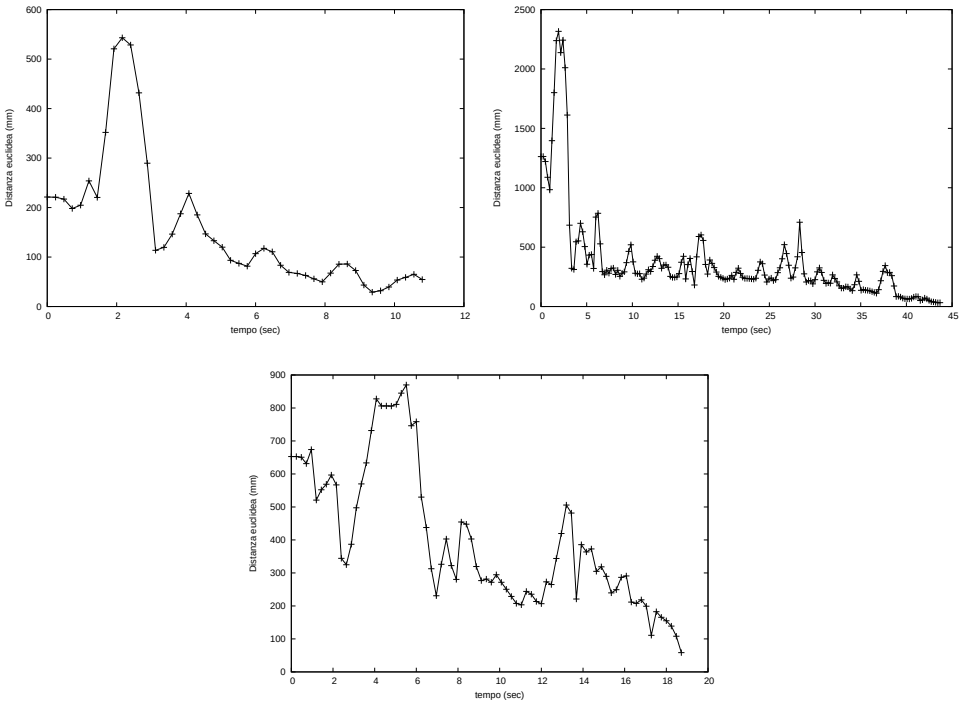
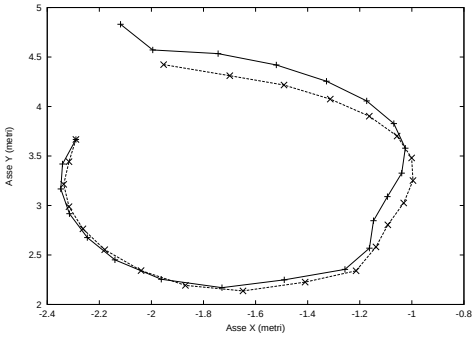


Figura 6.5: Distanza tra la telecamera e oggetto nella fase di visual servoing in tre esperimenti distinti.

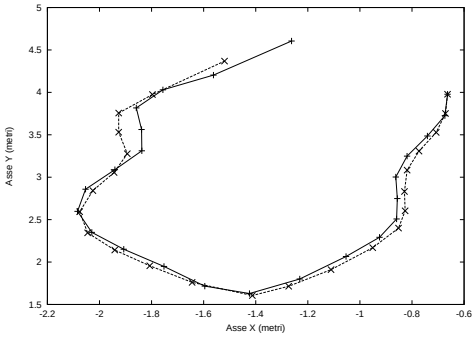
to un orientamento corretto della telecamera. Durante l'azione di orientamento il marker, da posizione centrata,si sposta all'interno della visuale, perciò viene riconosciuto un aumento della distanza. Il sistema allora provvede a modificare, oltre che l'orientazione, anche la posizione cercando di riportare il marker nel centro del campo visivo. Nell'algoritmo di avvicinamento gli spostamenti sono calcolati rispetto al sistema telecamera, ed approssimati per riportarli al sistema fisso, dando un peso maggiore agli spostamenti sul piano immagine rispetto alla profondità. Questa approssimazione produce un movimento all'indietro, che ha lo svantaggio di allontanare il marker, ma nel contempo ne assicura la visibilità da parte della microcamera.

6.2 Navigazione

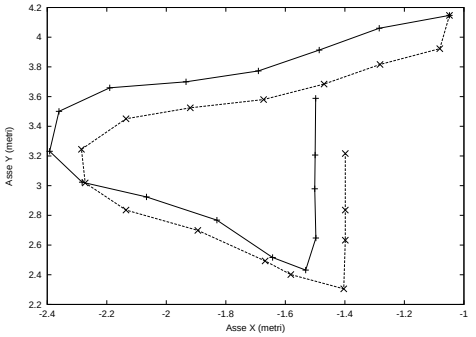
Un ulteriore test è stato svolto per verificare la precisione del modulo di navigazione della base mobile. Infatti l'errore con cui il sistema robotico si porta nella zona di compimento dell'azione robotica è un fattore altamente critico. Si consideri che se il posizionamento della base mobile (e dunque anche del manipolatore), è affetto da errore l'oggetto di interesse potrebbe risultare al di fuori dello spazio di lavoro, e pertanto il compito robotico risulterebbe non praticabile.
Per verificare la precisione della traiettoria sono stati messi a confronto i punti della traiettoria inviate al Nomad ed i corrispondenti valori di posizionamento letti dai sensori di odometria. In base ai dati elaborati si è calcolata media e varianza tra la posizione comunicata, e quella indicata dall'odometria (che a sua volta è notoriamente affetta da errori). Le lunghezze indicative delle traiettorie eseguite e riportate in fig.6.6 variano dai 4 a 5 metri.



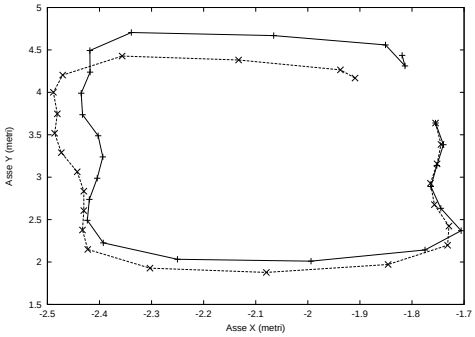
(a) Test di Navigazione 1 errore media $\eta = 0.1470$metri dev.std. $\sigma = 0.126$metri.



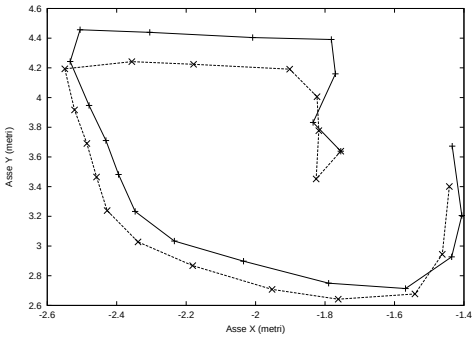
(b) Test di Navigazione 2 errore media $\eta = 0.1140$metri dev.std. $\sigma = 0.084$metri.



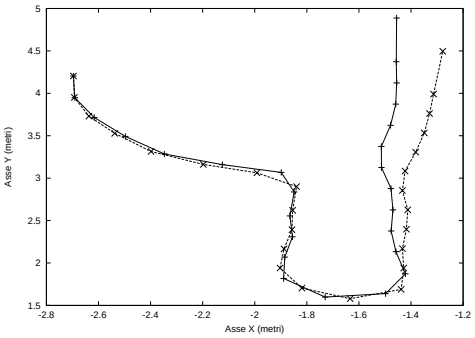
(c) Test di Navigazione 3 errore media $\eta = 0.1292$metri dev.std. $\sigma = 0.112$metri.



(d) Test di Navigazione 4 errore media $\eta = 0.1565$metri dev.std. $\sigma = 0.165$metri.



(e) Test di Navigazione 5 errore media $\eta = 0.1147$metri dev.std. $\sigma = 0.113$metri.



(f) Test di Navigazione 6 errore media $\eta = 0.1092$metri dev.std. $\sigma = 0.110$metri.

Figura 6.6: Test di Navigazione.

6.3 Esecuzione di compiti di manipolazione mobile

Una volta testati i singoli comportamenti del sistema robotico, si è provveduto all'esecuzione di prove complesse. Attraverso l'interfaccia si è impartito un ordine di navigazione verso una zona dell'ambiente contenente oggetti vari identificati da marker e specificando al sistema robotico l'identificatore del marker corrispondente all'oggetto da prelevare.

Figura 6.7: Esperimento 4: Navigazione del robot.

Nelle figure 6.7 e 6.8 sono illustrate alcune sequenze dell'esecuzione del compito da parte del sistema robotico.

Figura 6.8: Esperimento 4: Avvicinamento e presa.

Capitolo 7

Conclusioni

In questa tesi è stato realizzato un sistema di asservimento visivo per robot manipolatori, finalizzato all'esecuzione di azioni robotiche complesse, come ricerca e presa di oggetti. Il contributo principale di questo lavoro è stata la creazione di un programma di controllo per un braccio robotico basato su dati estrapolati da un sistema di visione artificiale, mediante la tecnica del *visual servoing*. Il sistema robotico realizzato consente l'esecuzione di compiti di manipolazione mediante l'utilizzo di una semplice interfaccia utente basata su realtà virtuale. Le prove eseguite sono state in grado di far emergere le potenzialità del sistema realizzato, evidenziando i vantaggi e gli svantaggi delle soluzioni tecniche adottate. È necessario commentare i test svolti analizzando le prestazioni di ogni funzionalità che il sistema mette a disposizione.

Per quanto riguarda il modulo di navigazione, risulta sempre presente un certo scostamento della traiettoria seguita rispetto alla traiettoria impostata, dovuto principalmente agli errori di odometria presenti negli attuatori dei motori del Nomad. In particolare le debolezze del sistema si riscontrano in caso di inseguimento di traiettorie complesse in cui sono richiesti rapidi cambiamenti di direzione. Bisogna inoltre considerare che la traiettoria specificata dall'utente nell'interfaccia viene sottocampionata, al fine di ottenere dei movimenti compatibili con la risoluzione di spostamento della base mobile, e gli effetti di questa operazione sono riscontrabili maggiormente proprio in corrispondenza di curve strette. Inoltre, a prescindere

dalla precisione del programma di navigazione, il successo del compito di posizionamento è legato anche alla precisione con cui il robot viene inserito nell'ambiente virtuale, e di conseguenza anche alla precisione della ricostruzione in grafica computerizzata del luogo in cui il sistema opera.

La parte di visione artificiale, in questo lavoro, si basa totalmente su funzioni messe a disposizione dalla libreria ARToolkit. Le prestazioni ottenute da questo modulo, sono più che buone. L'utilizzo di marker consente di ridurre il carico computazionale in maniera significativa, e di portare l'esecuzione di tutto il software che costituisce il "server reale" sulla macchina Nomad. La precisione dei dati ottenuti, tenendo conto dell'utilizzo di marker di ridotte dimensioni e microcamere economiche, è anch'essa di buon livello: la stima di posizionamento, ad una distanza media di 30cm, risente di errori nell'ordine del centimetro sulla posizione e dell'ordine di qualche grado nell'orientamento. Le oscillazioni dei valori di stima calano rapidamente col diminuire della distanza tra marker e telecamera, tanto che a distanze minime sono ampiamente trascurabili.

Le funzionalità di avvicinamento e presa, essendo realizzate con algoritmi diversi, offrono prestazioni differenti. È necessario innanzi tutto ricordare che anche nella realizzazione di questo modulo si è cercato di ridurre il carico computazionale, in modo che il sistema di controllo del manipolatore e l'algoritmo di visione potessero essere eseguiti anche su elaboratori di prestazioni medio/basse come quello presente sul Nomad. Pertanto è stata eliminata la parte di elaborazione di traiettoria, privilegiando un controllo iterativo basato su spostamenti incrementali. La funzione di avvicinamento è realizzata utilizzando la modalità cartesiana messa a disposizione via hardware, la quale garantisce un movimento abbastanza preciso, a patto di compiere spostamenti minimi in ogni passo di elaborazione. Al contrario la parte di posizionamento e presa, che viene realizzata comandando il manipolatore nello spazio dei giunti e per cui è necessaria la lettura degli encoder del braccio robotico, risulta poco precisa e tende a produrre un movimento "a scatti". Questa limitazione nelle prestazioni è dovuta alla presenza di elasticità e giochi, causati dalla struttura a cinghie del manipolatore stesso. Il Manus infatti, come descritto nella sez.2.2, è stato progettato per essere inserito in carrozzine elettriche per disabili, privilegiando il

basso peso e la sicurezza. Proprio queste caratteristiche impongono delle soluzioni costruttive che ne limitano la precisione, a differenza dei manipolatori industriali.

Gli sviluppi futuri di questa tesi riguardano principalmente l'ampiamento dei compiti eseguibili dal sistema come:

- Spostamento di un oggetto in una diversa locazione.
- Miglioramento delle strategia di controllo in caso di perdita del marker.
- Analisi preliminare della scena per la sola ricerca di marker.
- Possibilità di interazione con l'utente durante l'esecuzione del compito.

Un miglioramento dell'architettura proposta prevede inoltre l'integrazione di un modulo di localizzazione automatica per il robot.

Appendice A

Introduzione ai sistemi di riferimento

Sia P la posizione di un vettore nello spazio tridimensionale; sia allora P_0 la posizione di tale vettore descritto secondo una terna di riferimento 0.
Consideriamo ora un vettore P nello spazio e due sistemi di riferimento centrati, che chiameremo A e B, allora avremo che P_A descrive il vettore P rispetto al sistema A e P_B descrive il vettore P rispetto al sistema B; indichiamo dunque R_A^B la matrice di rotazione del sistema B rispetto al sistema A allora possiamo scrivere che

$$P_A = R_A^B * P_B$$

Se invece consideriamo i due sistemi di riferimento traslati l'uno rispetto all'altro, allora l'equazione diventa:

$$P_A = R_A^B * P_B + t_A^B \tag{A.1}$$

In (A.1) $t_A^B = \begin{bmatrix} {t_x}_A^B \\ {t_y}_A^B \\ {t_z}_A^B \end{bmatrix}$ rappresenta la posizione dell'origine di B rispetto agli assi di A, ma può essere omesso se al posto della matrice di rotazione si usa la matrice di rototraslazione

$$\bar{R_A^B} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & {t_x}_A^B \\ r_{21} & r_{22} & r_{23} & {t_y}_A^B \\ r_{31} & r_{32} & r_{33} & {t_z}_A^B \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

In A.2 r_{xy} sono gli elementi della matrice di rotazione.

Quando un vettore è posto nello spazio, oltre alla sua posizione viene definita implicitamente anche una rotazione. La matrice di rotazione ci consente di esprimere la rotazione di un vettore rispetto ad un sistema di riferimento, ma è una rappresentazione ridondante, visto che gli elementi della matrice non sono tutti indipendenti. Utilizziamo allora il sistema minimo dell'orientamento, detto **angoli di Eulero** o **notazione roll pitch yaw**. Un vettore nello spazio può allora essere definito mediante la sua posizione ed orientazione:

$$P_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ \alpha_0 \\ \beta_0 \\ \gamma_0 \end{bmatrix} \tag{A.3}$$

x_0, y_0, z_0 contenuti in A.3 rappresentano la proiezione del vettore sui rispettivi assi X_0, Y_0, Z_0 della terna 0, mentre α_0 rappresenta l'angolo di rotazione del vettore rispetto all'asse X_0, β_0 rappresenta l'angolo di rotazione del vettore rispetto all'asse Y_0 ed infine γ_0 rappresenta l'angolo di rotazione del vettore rispetto all'asse Z_0. Chiameremo dunque P_0 posa del vettore P rispetto al sistema 0.

Appendice B

Parametri di Denavit Hartenberg del Manipolatore Manus

Il problema della cinematica di un manipolatore [38] consiste nel passare dallo spazio di lavoro del manipolatore allo spazio dei giunti e viceversa. Il problema della cinematica diretta mira ad ottenere la posizione del manipolatore nello spazio di lavoro in funzione della configurazione dei giunti. Al contrario il problema della cinematica inversa mira a ricavare la configurazione dei giunti del manipolatore che consente il posizionamento in un dato punto dello spazio operativo. Entrambi trovano soluzione mediante il posizionamento di terne fittizie in corrispondenza dei giunti del manipolatore stesso. Tali terne vengono fissate con delle regole precise che prendono il nome di metodo di Denavit-Hartenberg [39].
In questa appendice, dopo un breve richiamo al metodo generale, vengono ricavati i parametri cinematici DH per il robot Manus. Nel caso generale, si considerino due generici assi $i-1$ ed i dei giunti del manipolatore, orientati nello spazio, e si definiscono:

- a_{i-1} è la distanza minima fra asse i-1 e i
- Se si trasla l'asse i lungo a_i, allora i due assi si incrociano in un punto formando un angolo α_{i-1}
- Si chiamino d_i le distanze lungo l'asse i tra i segmenti a_i e a_{i-1}.

- Supponendo di traslare a_i lungo l'asse i allora a_i e a_{i-1} si incrociano in un punto con un angolo θ_i

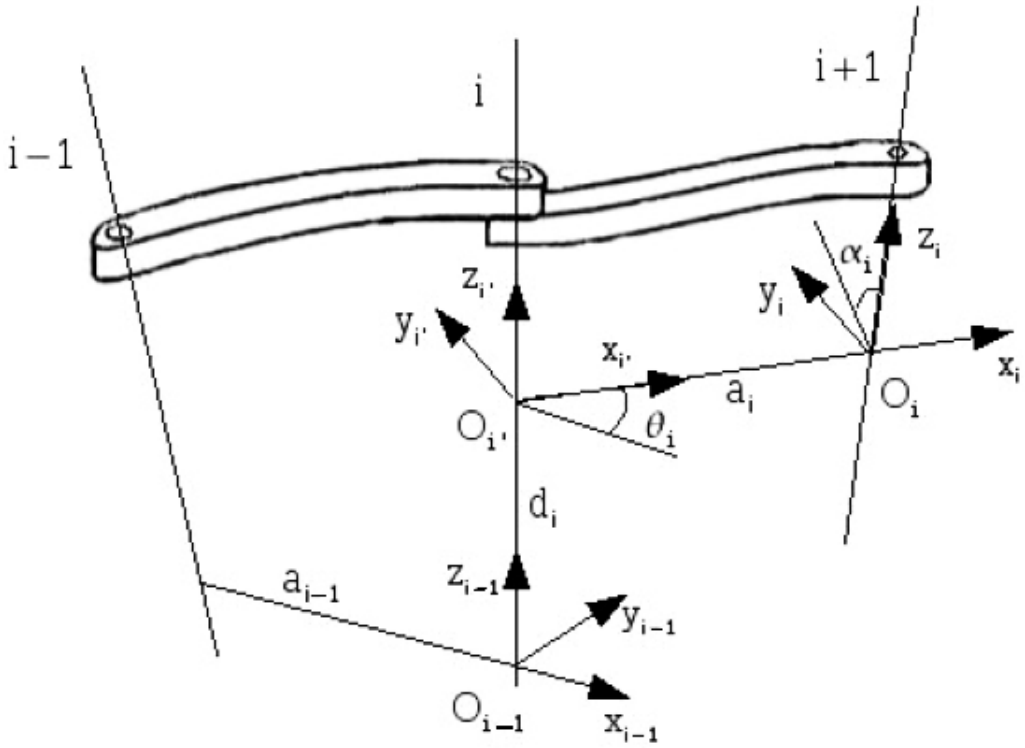


Figura B.1: Parametri Cinematici Denavit-Hartenberg

Secondo D.H. per passare dal sistema di riferimento di un generico link i-1, al successivo i, è sufficiente una trasformazione che è la composizione di quattro trasformazioni elementari ricavabili osservando la fig.B.1:

1. una traslazione lungo z_{i-1} di una quantitá *d* per portare l'origine del sistema di riferimento i-1 all'altezza della retta a minima distanza tra i due assi z_{i-1} e z_i

2. una rotazione intorno all'asse z_{i-1} di una quantitá θ per direzionare l'asse x_{i-1} lungo la retta a minima distanza che coinciderá anche con la direzione dell'asse x_i.

3. una traslazione lungo l'asse x_{i-1} di una quantitá a per far coincidere le origini e per sovrapporre i due assi x

4. una rotazione intorno all'asse x_i di una quantitá α per sovrapporee l'asse z_{i-1} all'asse z_i

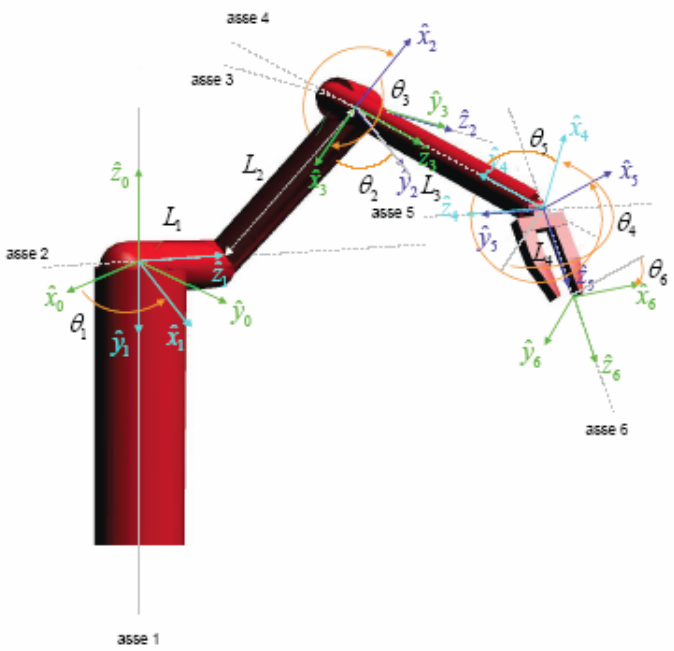


Figura B.2: Il Manus secondo la convenzione D.H.

Applicando queste regole al manipolatore Manus si ottiene l'insieme di parametri riportato in tabella B.2:

braccio	L_1	L_2	L_3	L_4
lunghezza (metri)	0.105	0.4	0.32	0.16

Tabella B.1: Lunghezze dei bracci del Manus.

In tab.B.1 sono riportate le lunghezze dei bracci del manipolatore.

i	a_i	α_i	d_i	θ_i
1	0	$-\pi/2$	0	θ_1
2	L_2	0	L_1	θ_2
3	0	$\pi/2$	0	θ_3
4	0	$-\pi/2$	L_3	θ_4
5	0	$\pi/2$	0	θ_5
6	0	0	L_4	θ_6

Tabella B.2: Parametri cinematici del Manus.

Bibliografia

[1] A. Melzi. Un sistema di interazione uomo macchina per compiti di manipolazione mobile basato su realtà virtuale. Tesi di laurea in ingegneria elettronica, Università degli studi di Parma, 2005.

[2] H. Inoue J. Hill. Real time control of a robot with a mobile camera. In *Proc. 9th ISIR*, pages 233–246, Washington D.C., March 1979.

[3] L. E. Weiss A. C. Sanderson. Imaged-based visual servo control using relational graph error signals. In *Proc. IEEE*, pages 1074–1077, 1980.

[4] M. Prats, P.J. Sanz, and A.P. del Pobil. Model-based tracking and hybrid force-vision control for the uji librarian robot. Robotic Intelligence Laboratory Universitat Jaume I, Castellon - Spain.

[5] Activemedia Robotics. Mobile robots. http://www.activrobots.com/ROBOTS.

[6] M. Ishikawa A. Namik, K. Hashimoto. Hierarchical control architecture for high-speed visual servoing. Int. J. Robotics Research to appear.

[7] F. Chaumette N. Mansard, O. Stasse and K. Yokoi. Visually-guided grasping while walking on a humanoid robot. In *IEEE Int. Conf. on Robotics and Automation, ICRA'07*, Roma, Italia, April 2007.
http://www.irisa.fr/lagadic/team/Nicolas.Mansard-eng.html.

[8] F. Bley V. Schmirgel. Autonomous manipulation in assistive mobile robotics. 10th International Student Conference on Electrical Engineering, May 2006. RWTH Aachen University - Aachen.

[9] F. Monica. Progettazione di un'architettura modulare aperta ed in tempo reale per un robot mobile. Tesi di laurea in ingegneria informatica, Università degli studi di Parma, 2003.

[10] D. Pallastrelli F. Monica. Controllo remoto per il nomad. Technical Report, 2001.

[11] L. Capuzzello. Sviluppo di compiti di navigazione mediante supporti evoluti di programmazione concorrente. Tesi di laurea in ingegneria elettronica, Università degli studi di Parma, 1997.

[12] C. Bertani. Navigazione di robot mobili basata su apprendimento con rinforzo. Tesi di laurea in ingegneria elettronica, Università degli studi di Parma, 1996.

[13] A. Calafiore. Navigazione di robot mobili in ambiente strutturato basata su visione artificiale. Tesi di laurea in ingegneria elettronica, Università degli studi di Parma, 1996.

[14] Nomadic Techonologies Inc. The nomad 200 user guide, December 1993.

[15] Exact dynamics. http://www.exactdynamics.nl.

[16] Exact Dynamics. *ARM User Manual*. Exact Dynamics, 9 edition, August 2002.

[17] J.A. Van Woerden A.H. Versluis, B.J. Driessen and B.J. Krose. Enhancing the usability of the manus manipulator by using visual servoing. University of Amsterdam, Netherlands.

[18] Philips Semiconductor. Sja1000: Stand-alone can controller, 2000.

[19] Plx technology. www.plxtech.com.

[20] Terratec cinergy. www.terratec.it/prodotti/schede_tv/cinergy_250_pci.shtml.

[21] Philips Semiconductor. www.npx.com/pip/saa7134hl.

[22] Immersion corporation. www.immersion.com.

[23] Polhemus. www.polhemus.com.

[24] Opengl. www.opengl.org.

[25] J. Hartman and J. Wernecke. *The VRML 2.0 Handbook*. Adison-Wesley, 1996.

[26] Web 3d consortium. www.wen3d.org.

[27] Artoolkit. www.hitl.washington.edu/artoolkit.

[28] F. Denaro. Un sistema per la modellazione di ambienti mediante visione artificiale. Tesi di laurea in ingegneria informatica, Università degli studi di Parma, 2006.

[29] Progetti internazionali basati su artoolkit. http://www.hitl.washington.edu/artoolkit/projects/.

[30] Newmat documentation. http://www.robertnz.net/index.

[31] The boost library. http://www.boost.org.

[32] G. S. Bell W. J. Wilson, C. C. Williams Hulls. Relative end-effector control using cartesian position based visual servoing. *IEEE Transaction on Robotics and Automation*, 12(5), October 1996.

[33] F. Janabi-Sharifi L. Deng, W. J. Wilson. Dynamic performance of the position-based visual servoing method in the cartesian and image space. In *Int. Conference on Intelligent Robots and Systems*, Las Vegas - Nevada, 2003. IEEE.

[34] G. Ferrari. Realizzazione di un'architettura software di governo per un robot manipolatore per compiti di assistenza. Tesi di laurea in ingegneria informatica, Università degli studi di Parma, 2004.

[35] L. Marchini. Sviluppo di un modulo software per la soluzione della cinematica di un manipolatore manus. Tesi di laurea in ingegneria informatica, Università degli studi di Parma, 2004.

[36] Roboop. http://www.cours.polymtl.ca/roboop/.

[37] C. G. Lo Bianco. *Cinematica dei manipolatori*. Pitagora Editrice, 2004.

[38] J. J. Craig. *Introduction to Robotics, Mechanics and Control*. Pearson Prentice Hall, 2005.

[39] J. Denavit and R. S. Hartenberg. A kinematic notation for lower pair mechanisms based on matrices. *ASME Journal of Applied Mechanics*, June 1955.

[40] P. Ochi. Progettazione e realizzazione del sistema di controllo di un robot manipolatore per compiti di assistenza. Tesi di laurea in ingegneria elettronica, Università degli studi di Parma, 2003.

[41] D. Kragic and H. I Christensen. Survey on visual servoing for manipulation. Centre for Autonomous System Numerical Analysis and Computer Science, Stockholm, 2004.

[42] B. Yoshimi P. K. Allen, A. Timcenko and P. Michelman. Automated tracking and grasping of a moving object with a robotic hand-eye system. *IEEE Transaction on Robotics and Automation*, 9(2), April 1993.

[43] P. Rives B. Espiau, F. Chaumette. A new approach to visual servoing in robotics. *IEEE Transaction on Robotics and Automation*, 8(3), June 1992.

[44] E. Marchand G. Flanding, F. Chaumette. Eye in hand-eye to hand cooperation for visual servoing. In *Int. Conference on Robotics and Automation*, San Francisco - CA, April 2000. IEEE.

[45] M. F. de Mathelin J. A. Gangloff. Visual servoing of a 6-dof manipulator for unknown 3-d profile following. *IEEE Transaction on Robotics and Automation*, 18(4), August 2002.

[46] J.M.C. Sousa P. J. Sequeira Gonçalves, L.F. Mendoça and J.R. Caldas Pinto. Improving visual servoing using fuzzy filters. Escola Superior de Tecnologia de Castel Branco - Technical University of Lisbona, Budapest, July 2004.

[47] B. Espiau R. Horaud, F. Dornaika. Visually guided object grasping. *IEEE Transaction on Robotics and Automation*, 14(4), August 2004.

[48] P. I Corke S. Hutchinson, G. D. Hager. A tutorial on visual servo control. *IEEE Transaction on Robotics and Automation*, 12(5), October 1996.

[49] D. R. Buthenhof. *Programming with POSIX threads*. Adison-Wesley Professional Computing Series, 1997.

A differenza degli esami, la tesi di laurea è l'esperienza più coinvolgente che lo studente compie nell'arco della sua carriera. Nel mio caso poi questi mesi sono stati particolarmente ricchi, ed appaganti per qualità ed acquisizione di conoscenza, per merito specialmente delle persone che mi hanno seguito ed accompagnato nello svolgimento del lavoro.

Devo pertanto ringraziare il mio relatore prof. Stefano Caselli che mi ha permesso di svolgere questa tesi, l'ing. Francesco Monica e l'ing. Dario Lodi Rizzini per i consigli e l'aiuto fornito in diverse occasioni.

Un ringraziamento particolare al mio correlatore Ing. Jacopo Aleotti, per la disponibilità e l'infinita pazienza con la quale ha supervisionato il mio lavoro.

Un grazie a tutti gli studenti e non, frequentatori abituali del laboratorio di robotica per i bei momenti passati in queste numerose settimane.

Ringrazio tutti gli amici di questi anni, specialmente Aramini, Marchi, il Falcio, Fossa, il Bolzo, il Pianfo, ed il Fisto compagni di discussioni, studio, e mangiate!

Un ultimo ringraziamento va alla mia famiglia per avermi sempre sostenuto confortato.