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

PROGETTAZIONE E REALIZZAZIONE DEL SISTEMA DI CONTROLLO DI UN ROBOT MANIPOLATORE PER COMPITI DI ASSISTENZA

Relatore:
Chiar.mo Prof. STEFANO CASELLI

Correlatori:
Dott. Ing. MONICA REGGIANI
Dott. Ing. FRANCESCO MONICA

Tesi di laurea di:
PIETRO OCCHI

Anno Accademico 2003-2003

Ai miei genitori,
Enrico ed Anna
e a Francesca.

Il tempo passato a svolgere questa tesi è stato sicuramente segnato dalla presenza di persone che mi hanno offerto la loro professionalità, il loro impegno oltre che la loro amicizia. Queste righe vorrebbero poter offrire a tutti un piccolo "grazie".

Per prima cosa vorrei ringraziare il Prof. Caselli che mi ha dato la possibilità di svolgere questa tesi offrendomi l'opportunità di lavorare ad un progetto di rilievo e, soprattutto, per me di grande interesse.

Ringrazio anche l'Ing. Reggiani per l'aiuto prezioso nello svolgimento del mio lavoro ma soprattutto per avermi fatto intravedere il mondo della vera programmazione. Inoltre la ringrazio per tutti i consigli e la vicinanza dimostratami.

Ringrazio anche sia Francesco sia Eleonora, o meglio l'Ing. Monica e l'Ing. Fantini, con i quali ho avuto l'opportunità di lavorare fianco a fianco. Proprio per questo mi auguro di avere l'occasione di poter lavorare ancora in futuro con persone come loro, vista le loro conoscenze e soprattutto disponibilità.

Sicuramente la maggior parte del tempo l'ho passato a contatto con il manipolatore MANUS, ma mi sembra esagerato attribuirgli facoltà di ragionamento visto anche che non sempre si è comportato come avrebbe dovuto. Ritengo però di inserire un ringraziamento speciale ad Alexandros che ha studiato il robot molto prima di me e che mi ha dato un prezioso aiuto.

Ringrazio la Dott.sa Ghizzoni che si è "spontaneamente" offerta per la realizzazione del filmato del robot data la sua indiscutibile esperienza in campo video. Inoltre ringrazio anche la Dott.sa Orsi per avermi fornito assistenza dato che le lingue non sono mai stato il mio forte.

Infine ringrazio tutti i ragazzi dell'indimenticabile laboratorio di robotica con i quali ho condiviso momenti di lavoro e momenti di svago, nonché impreviste ed esplosive esperienze.

Indice

Capitolo 1

Introduzione

1.1 Progetto *RoboCare*

Negli ultimi anni la ricerca si è dedicata allo sviluppo di sistemi finalizzati a fornire aiuto a persone disabili in ambito domestico o ospedaliero. Per tutti coloro che devono trascorrere parte della propria vita nell'impossibilità di compiere normali movimenti e attività, infatti, si rende necessario affidarsi a persone che possano portare loro assistenza. Questa esigenza di assistenza da un lato crea una sensazione di dipendenza e di scarsa autonomia nella persona assistita, e dall'altro, determina costi rilevanti per la persona e per il sistema sociale nel suo complesso. In quest'ottica la robotica si pone l'obiettivo fondamentale di creare strumenti che possano rispondere alle esigenze dell'utente e lo aiutino nello svolgimento di alcune azioni di vita quotidiana, permettendo di migliorare lo stato di indipendenza dell'assistito grazie ad una sua maggiore operatività nell'ambiente domestico.

In questo contesto, nasce nel Novembre del 2002 il progetto *RoboCare*, finanziato dal *MIUR* e a cui partecipa anche il *Dipartimento di Ingegneria dell'Informazione dell'Università di Parma*. Il progetto ha come scopo, appunto, la creazione di un'interazione tra robot autonomi, sistemi intelligenti e persone per l'assistenza ad anziani e disabili. Risulta essenziale sottolineare come, nel progetto RoboCare, a differenza di altri, alla base di tutto si trovi una forte idea di cooperazione: molti gruppi di ricerca che affrontano questo tema, infatti, spesso valutano soltanto lo sviluppo di soluzioni tecnologiche adeguate senza la preoccupazione di quella che poi

dovrà risultare l'interazione che si instaurerà tra uomo e robot e tra robot e robot, operanti nel medesimo ambiente.

Il progetto RoboCare, quindi, si pone l'obiettivo di studiare la realizzazione di un sistema multi-agente sviluppato su piattaforme distribuite ed eterogenee. Esso si propone cioè di sviluppare robot studiati per eseguire diversi compiti sia singolarmente sia sfruttando l'idea di cooperazione. L'intento finale è quello di realizzare un sistema completo che possa fornire servizi efficaci integrando automazione domestica, robot autonomi intelligenti e sistemi di comunicazione evoluti.

1.2 Obiettivi

All'interno del progetto RoboCare l'unità dell'Università di Parma si propone di sviluppare un robot mobile dotato di braccio manipolatore con il quale eseguire compiti di identificazione, prelievo e consegna di oggetti. Per esempio, come illustrato in figura 1.1, il robot dovrà essere in grado di identificare, prendere e quindi riportare all'utente un oggetto ben determinato che l'utente stesso gli ha indicato in qualche modo in precedenza.

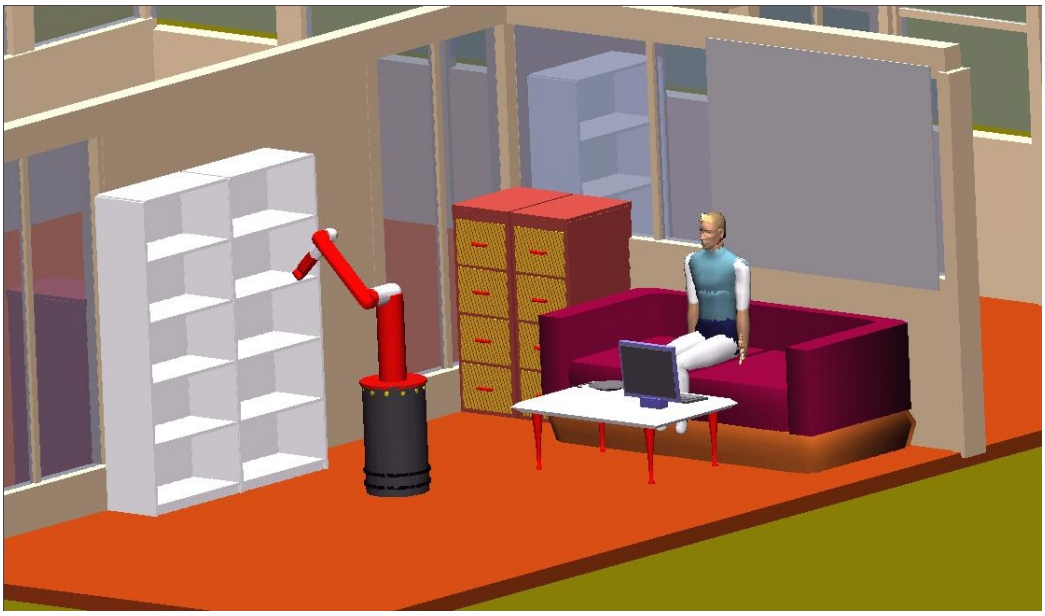

Figura 1.1: Esempio di task per un robot manipolatore di assistenza

Per la realizzazione di un sistema che fornisca tali servizi, occorre disporre di un robot mobile dotato di ampie capacità sia motorie che sensoriali, soprattutto per la navigazione, e, inoltre, di adeguati strumenti di elaborazione per estrarre l'informazione dai sistemi sensoriali più evoluti, per esempio da telecamere usate per l'individuazione degli oggetti. Il sistema, inoltre, deve essere dotato di un robot manipolatore e di adeguati strumenti per controllarne il moto.

In particolare questa tesi si propone di realizzare un sistema di controllo per il robot manipolatore MANUS adottato nel progetto e descritto in dettaglio nei capitoli successivi.

1.3 Organizzazione della tesi

La tesi è organizzata come segue. Nel *capitolo 2* è illustrato lo stato dell'arte nella robotica per assistenza. Dopo una breve introduzione in cui sono indicati gli obiettivi principali di questo filone di studio, vengono classificati i principali progetti presenti nel panorama internazionale. Successivamente, sono illustrati con maggior dettaglio le scelte architetturali adottate in alcuni progetti maggiormente significativi.

Nel *capitolo 3* è presentato il progetto dal punto di vista dei componenti e delle interfacce. Sono quindi descritti i robot utilizzati, per poi discutere le specifiche di sviluppo dell'interfaccia necessaria per permettere ai robot di scambiare informazioni. Vengono infine analizzati in dettaglio i problemi legati allo scambio dei messaggi CAN e al protocollo adottato dal costruttore del manipolatore per realizzare la comunicazione.

Nel *capitolo 4* si affrontano i problemi principali dal punto di vista software. In questo capitolo vengono illustrati inizialmente gli algoritmi utilizzati per il calcolo delle traiettorie relative alla generazione del moto del manipolatore. Quindi viene descritta l'architettura software sviluppata, presentando anche la suddivisione delle classi.

Nel *capitolo 5* vengono riportati i risultati sperimentali ottenuti impostando il moto del manipolatore tramite un file di testo contenente alcuni nodi della traiettoria. In questo capitolo vengono presentati alcuni esempi di traiettorie esegui-

te per evidenziare il comportamento del generatore e le prestazioni del sistema di controllo.

La tesi si conclude (*capitolo 6*) con una breve discussione dei risultati ottenuti e delle possibili evoluzioni del progetto e del sistema di controllo.

Capitolo 2

Robotica per l'assistenza

2.1 Introduzione

La robotica per assistenza sta rivestendo in questi ultimi anni un ruolo essenziale nell'ambito robotico internazionale. Numerosi gruppi di ricerca stanno lavorando per realizzare sistemi in grado di essere di aiuto a persone affette da handicap e, per questo, incapaci di svolgere azioni e compiti tipici della vita di tutti i giorni. In questo capitolo, dopo una breve descrizione su che cosa significa *Robotica per Assistenza*, si introdurrà una classificazione dei progetti svolti nell'ambito robotico internazionale; quindi si descriveranno in dettaglio alcuni progetti che prendono in considerazione l'utilizzo del manipolatore *MANUS* utilizzato in questo lavoro di tesi. Alla fine verranno anche citate quelle che possono essere le maggiori limitazioni allo sviluppo della robotica per assistenza.

2.2 Obiettivo della Robotica per Assistenza

Il miglioramento delle condizioni di vita[1] per quelle persone che a causa dell'età o perché disabili, non possono avere un'esistenza normale, sta diventando un obiettivo essenziale per la società di oggi. Un aspetto importante per queste persone bisognose di supporto nelle attività quotidiane è quello di riuscire ad essere comunque integrate nella vita sociale superando quelle che risultano le loro disabilità. Lo scopo principale, quindi, dei progetti di ricerca è quello di ottenere nei prossimi an-

ni la realizzazione di strumenti capaci di aiutare le persone affette da problemi fisici ad affrontare la realtà quotidiana.

Per quanto riguarda l'aspetto economico, è il singolo utente che deve farsi carico di tutti o di una parte dei costi di un eventuale struttura di assistenza basata su sistemi avanzati o robotici. Risulta perciò importante cercare di ridurre i costi per far sì che sempre più persone possano usufruire di questi benefici.

Per una persona anziana o affetta da problemi fisici, affidarsi ad uno strumento che lo aiuti a svolgere numerosi compiti nella vita di tutti i giorni significa:

- migliorare il proprio senso di indipendenza;
- migliorare la qualità della propria vita;
- aumentare le possibilità di movimento;
- ottenere aiuto anche nei periodi in cui il personale addetto all'assistenza risulta assente;
- aumentare la sicurezza personale;
- migliorare l'integrazione sociale mediante le comunicazioni video
- ridurre i costi per gli interventi medici a domicilio
- operare più facilmente nell'ambiente domestico

2.3 Stato dell'arte

L'applicazione della robotica nella riabilitazione e, in particolare, nell'assistenza alle persone disabili e agli anziani, è stata oggetto di studio approfondito da parte di svariati gruppi di ricerca, soprattutto negli ultimi anni. Questo tentativo di fornire una risposta ai bisogni crescenti della società moderna ha portato allo sviluppo di numerosi progetti che si differenziano tra loro per molti aspetti: primo tra tutti l'aspetto della complessità tecnologica. Nel seguito si metteranno in luce i più significativi studi effettuati a livello mondiale e i risultati raggiunti cercando di suddividere i progetti di ricerca in una grossolana classificazione basata sulle diverse peculiarità.

2.3.1 Postazioni fisse

Le postazioni fisse[2] sono state il primo tentativo di applicare la robotica al servizio delle persone e in particolare rappresentano il risultato della robotica industriale applicata all'assistenza. Solitamente queste soluzioni sono caratterizzate dall'utilizzo di un braccio manipolatore di tipo industriale fissato ad un tavolo utilizzato per compiti di trasporto di oggetti in un ben delimitato spazio di lavoro. Normalmente l'utente controlla il manipolatore tramite un'interfaccia grafica con la quale dialoga con un personal computer, che a sua volta comunica al robot la scelta tra alcuni task predefiniti da eseguire.

Figura 2.1: Il sistema DEVAR.

Esempi di questo tipo di configurazione sono il *Devar System* sviluppato dal Centro di Ricerca e Sviluppo per la Riabilitazione del Veterans Affairs Department in California (Fig. 2.1). Il progetto Devar ha subito di recente un'evoluzione nel nuovo *Provar System*: esso risulta basato sempre sull'utilizzo di un manipolatore, ma stavolta il sistema è stato sviluppato per alti gradi di disabilità come, per esempio, persone quadriplegiche ma con buone capacità cognitive. Il sistema, di cui è mostrato uno schema in figura 2.2, si basa sull' utilizzo di un robot manipolatore *PUMA 260* e sfrutta un'interfaccia utente semplificata ma in cui sono state aggiunte varie possibilità come l'utilizzo del telefono e di Internet oltre che la possibilità di realizzare, tramite un bus X10, l'automazione della casa.

Il progetto *RAID* (*Robot for Assisting the Integration of the Disabled*) è anch'esso sempre basato sull'idea di una stazione di lavoro fissa per l'ambito domestico o per l'ufficio e permette a persone disabili ma con piene capacità cognitive di essere

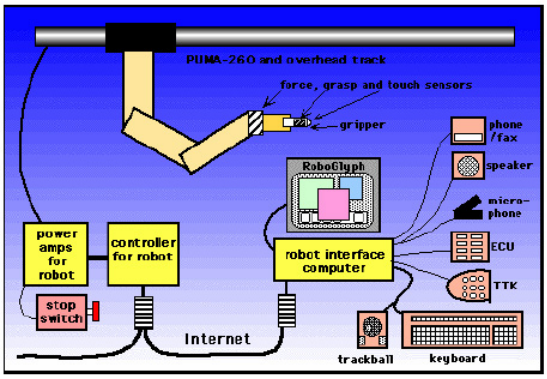

Figura 2.2: Il sistema PROVAR

Figura 2.3: Il sistema RAID

indipendenti nell'esecuzione di compiti come prendere e manipolare oggetti, o in task più complessi come bere o mangiare. Il sistema RAID (vedi fig. 2.3) inizialmente sviluppato all'interno del MASTER RAID PROJECT coordinato dal CEA-STR in Francia, fu clinicamente testato negli anni 1992 e 1996 all'interno del programma TIDE della Comunità Europea.

Come ultimo esempio delle postazioni fisse robotiche può essere citato il progetto *CAPDI* coordinato da UPC (Università Politecnica di Catalogna) e supportato dal Ministero Spagnolo per gli Affari Sociali. Il sistema consiste anch'esso in un robot manipolatore, che in questo caso è montato all'interno di una cucina strutturata ad hoc per permettere all'utente di realizzare qualsiasi task potendo raggiungere qualsiasi oggetto dislocato in essa. Nella figura 2.4 viene mostrato sia uno schema di una cucina strutturata sia la schermata dell'interfaccia grafica per l'utente.

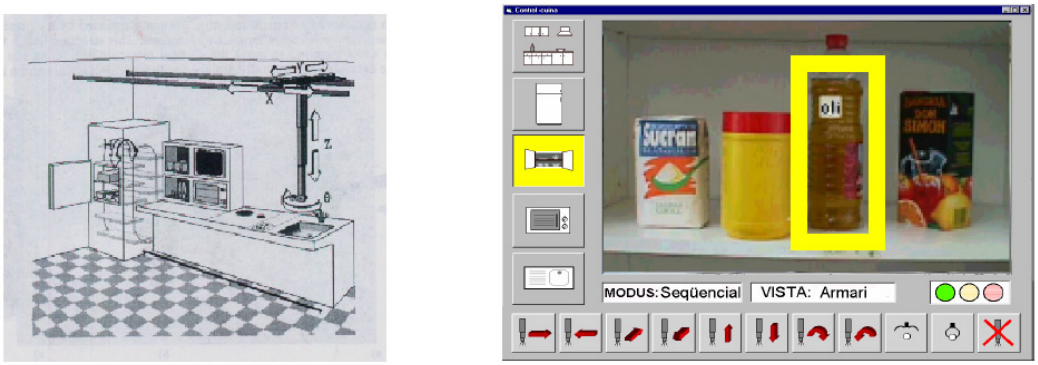

Figura 2.4: CAPDI System

Come si nota dagli esempi precedenti, con le postazioni fisse robotiche per poter operare con successo, nasce l'esigenza di strutturare opportunamente l'ambiente circostante in modo da renderlo noto al sistema[3]. Tale strutturazione ad una prima analisi può sembrare una limitazione, ma in realtà questo porta ad una notevole velocizzazione nello svolgimento dei compiti. Risulta infatti possibile utilizzare delle procedure preprogrammate con l'effettivo miglioramento delle prestazioni sia in termini di velocità sia di efficienza. È però evidente che l'area di lavoro del robot risulta piuttosto limitata, e di conseguenza sarà necessario utilizzare robot manipolatori con una buona estensione.

2.3.2 Postazioni a carrozzella

Le postazioni di questo tipo permettono all'utente di utilizzare il robot in contesti differenti sia all'interno sia all'esterno grazie al fatto di poter sfruttare il robot manipolatore unitamente alla mobilità offerta dalla carrozzella elettrica. Questa soluzione presenta alcuni svantaggi tecnici dovuti soprattutto alla scarsa accuratezza del sistema a causa del fatto che esso non conosce a priori l'ambiente circostante e, in questo modo, risulta impossibile utilizzare il robot in modo autonomo. Solitamente il moto del manipolatore viene controllato passo dopo passo dall'utente con la conseguente diminuzione di prestazioni soprattutto dal punto di vista della precisione e della velocità.

Un esempio di questo tipo di configurazione è proposta nel progetto *SPRINT-IMMEDIATE* che si pone l'obiettivo di ottenere svariati prototipi utilizzando il ma-

nipolatore *MANUS* prodotto da *Exact Dynamics* (di cui si avrà modo di parlare più ampiamente nei capitoli successivi). In questo progetto il Manus è montato su diversi modelli di carrozzella elettrica utilizzando lo standard M3S per l'interfaccia.

Figura 2.5: Il Manus nella configurazione TGR Explorer

In Italia è stata creato il *TGR Explorer* che utilizza, congiuntamente al MANUS, una carrozzella elettrica cingolata come mostrato in figura 2.5.

Figura 2.6: Progetto Handy1

In figura 2.6 viene mostrato il progetto *Handy 1* prodotto da *RehabRobotics* in Gran Bretagna. Esso è costituito da un semplice braccio manipolatore a 5 gradi di libertà con il quale risulta possibile compiere anche complessi task come per esempio prendere del cibo o un bicchiere d'acqua da portare alla bocca dell'utente.

In parallelo allo sviluppo di carrozzelle elettriche su cui montare i robot manipolatori, ma comunque sempre pilotate da un utente, la ricerca robotica ha dedicato

numerosi sforzi anche per studiare lo sviluppo di carrozzelle intelligenti. Tra le principali caratteristiche di questo tipo di carrozzelle sono le migliori performance di mobilità ottenute dotandole di sistemi di sensori e di controllo capaci di aiutare l'utente nel movimento oppure di muoversi autonomamente, anche in presenza di ostacoli. Il progetto *TIDE-OMNI* rappresenta un esempio di questo tipo di approc-

Figura 2.7: Progetto TIDE-OMNI

cio. Il sistema mostrato in figura 2.7 è equipaggiato con dei sensori sia ad ultrasuoni sia infrarossi e con un elaboratore che aiuta l'utente nel movimento della carrozzella evitando gli ostacoli presenti cercando di ridefinire la traiettoria stabilita dall'utente in base ai dati provenienti dall'ambiente circostante rilevati con i sensori.

Figura 2.8: Progetto ESCLATEC

Il progetto *ESCLATEC* (Fig. 2.8) è anch'esso un progetto basato sull'utilizzo di una carrozzella ed è stato sviluppato dall'Università Politecnica della Catalogna e supportato dal Ministero per gli Affari Sociali del governo Spagnolo. Questo

progetto è stato sviluppato per realizzare un mezzo che potesse aiutare le persone disabili ad affrontare terreni difficili come può essere, ad esempio, una spiaggia. Anche in questo caso sono stati aggiunti numerosi sensori in modo da facilitare la navigazione.

Figura 2.9: Sistema SPERIC 3x3

Infine il progetto *SPHERIC 3x3* (mostrato in figura 2.9) consiste in un robot mobile sviluppato dall'Università Politecnica della Catalogna per la Commissione per la Ricerca e la Tecnologia della Catalogna. La peculiarità di questo robot è che esso fonda il suo movimento su sei sfere configurate in modo da ottenere tre ruote con ampia mobilità. Per questo motivo esso può anche essere utilizzato nella configurazione tipica di una carrozzella.

Come evidenziato dagli esempi precedenti, i manipolatori utilizzati su carrozzelle, pur essendo meno precisi godono di tre gradi di libertà in più rispetto ai precedenti, forniti dalla mobilità della carrozzella. In questo modo risulta possibile per il braccio raggiungere uno spazio di lavoro maggiore e, di conseguenza, ci si può affidare a robot manipolatori di dimensioni minori. Esistono comunque alcune limitazioni: in questo caso, infatti, si deve cercare di mantenere l'ingombro, creato dal manipolatore, il più limitato possibile per evitare che ci siano problemi in zone strette della casa come porte o corridoi. Inoltre per poter assicurare una buona flessibilità del sistema, deve essere possibile scollegare in maniera rapida il manipolatore per facilitarne il trasporto.

2.3.3 Sistemi robotici autonomi

La soluzione che offre maggiore flessibilità nell'ambito della robotica per assistenza è sicuramente rappresentata dai robot mobili autonomi equipaggiati con un manipolatore e con sistemi di sensori per eseguire compiti di manipolazione e trasporto di oggetti. Questo sistema appare la soluzione più promettente per tutti quegli utenti che sono affetti da gravi disabilità fisiche o che devono passare la maggior parte del loro tempo sdraiati in un letto. Fornendo ad essi un'interfaccia adeguata, risulta possibile per l'utente interagire con il sistema anche a livello vocale. Questo tipo di sistema supera i problemi imposti dalla configurazione a carrozzella, ma rappresenta comunque la soluzione dotata di maggior complessità. Essa si deve tuttavia scontrare con problemi tecnici tutt'ora irrisolti soprattutto nell'ambito dell'autonomia nell'assistenza all'utente, dove la precisione del sistema, come si può ben immaginare, risulta fortemente critica.

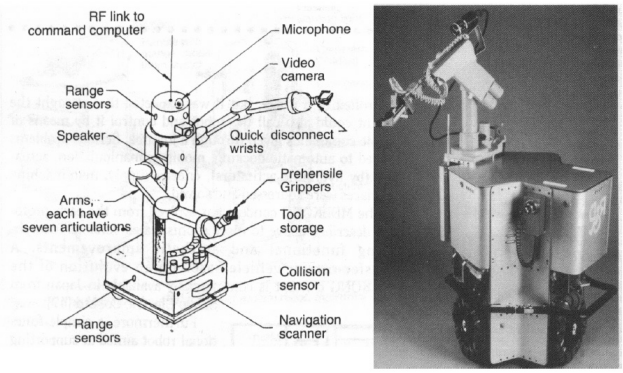

Figura 2.10: MOVAR Robot

L'idea di un sistema autonomo per assistenza fu introdotta per la prima volta dal gruppo di ricerca della *Stanford University*. Il sistema proposto (presentato in figura 2.10) denominato *MOVAR* si basa sull'idea di robot mobile dotato di ampia gamma di sensori per una buona autonomia. Allo stesso modo il progetto *URMAD* che significa *Unità Robotica Mobile per l'assistenza ai Disabili* sviluppato dalla *Scuola Superiore Sant'Anna* a Pisa (presentato in figura 2.11) consiste in una base mobile omnidirezionale equipaggiata con un braccio manipolatore ad otto gradi di libertà, con una mano robotica a due gradi di libertà e con un pan tilt dotato di due telecamere. La base mobile, inoltre, risulta equipaggiata con un anello di sensori

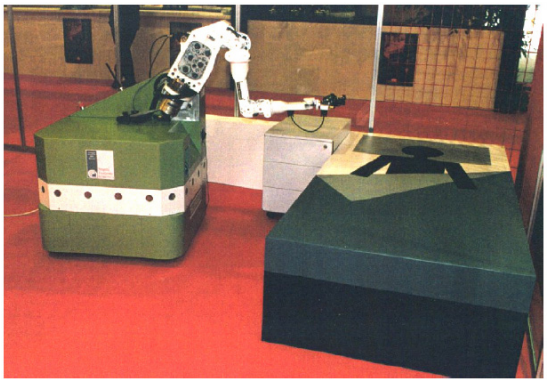

Figura 2.11: URMAD Robot

ad ultrasuoni e con un sistema di visione capace di rilevare gli ostacoli non visti dai sonar, mentre la mano robotica risulta dotata di sensori di forza per controllare la presa degli oggetti. È stata altresì creata un'interfaccia grafica installata su di un personal computer che permette all'utente di comunicare alla base mobile i task da svolgere. Questa interfaccia permette all'utente di scegliere tra due modalità di funzionamento: il modo *autonomo* con il quale è possibile chiedere al sistema di eseguire alcuni task preprogrammati o definirne di nuovi tramite l'inserimento di comandi per via vocale; oppure il modo di funzionamento *teleoperato* con il quale l'utente può controllare ogni singola parte del robot passo dopo passo fornendogli una sequenza di movimenti.

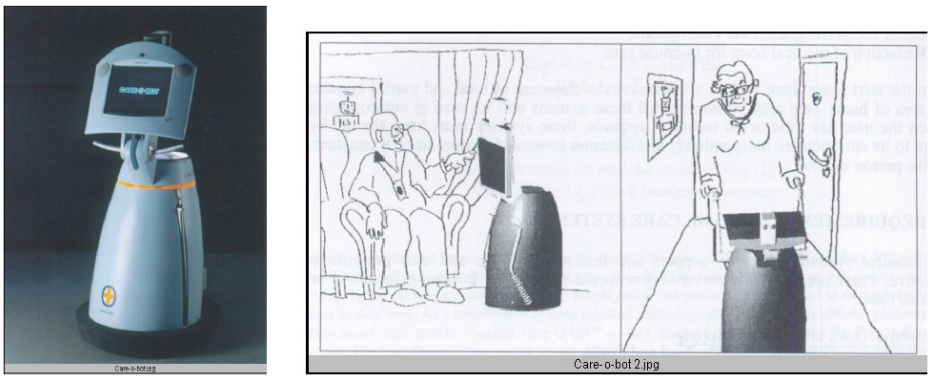

Figura 2.12: Sistema Care-O-bot

Nella figura 2.12 viene invece presentato un progetto sviluppato dell'*IPA* di Francoforte denominato *Care-O-bot*. In questo caso si tratta di un robot mobile

nato per assistenza in ambito domestico per persone disabili o per anziani. Il sistema è stato sviluppato per eseguire compiti di manipolazione e trasporto di oggetti come cibo, bevande o fiori ed, inoltre, per eseguire semplici compiti tipici dell'ambiente domestico. Particolare attenzione va posta all'interfaccia del sistema: essa, infatti, risulta dotata di un sistema di video telefono che permette al robot di contattare servizi di assistenza in caso di emergenza, oltre che interagire con tutte le apparecchiature presenti nell'ambiente, non solo come TV o DVD, ma anche con porte e finestre tramite un'opportuna automazione della casa. Inoltre va detto che questo robot permette anche il trasporto delle persone. Un prototipo di questo robot fu presentato alla fiera di Hannover nel 1998.

Altra categoria importante che va analizzata all'interno della sezione dei robot mobili è quella relativa ai robot studiati per essere utilizzati in ambiente ospedaliero. Il primo robot effettivamente costruito per eseguire compiti di assistenza all'inter-

Figura 2.13: Robot HelpMate

no di un ospedale è risultato essere *Pyxis HelpMate* (vedi foto 2.13) sviluppato nella prima metà degli anni '90 dalla *Pyxis Corp.*[4]. Il robot in questione fu il primo ad utilizzare una navigazione interamente basata sull'utilizzo di sensori sonar. Un'analisi approfondita mostra che esso risultava molto più veloce e preciso nella

distribuzione delle medicine ai pazienti rispetto ad un infermiere in carne ad ossa addetto al medesimo compito. Per questo motivo si stimò che in un anno di lavoro questo robot permettesse di risparmiare da 5000$ a 10000$. Si può affermare che in America alla fine degli anni novanta risultarono in uso un centinaio di HelpMate utilizzati in vari ambiti come ospedali, aziende farmaceutiche e laboratori clinici.

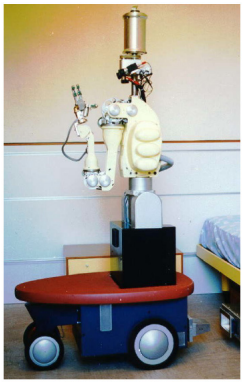

Figura 2.14: MOVAID System

Un altro progetto di particolare interesse è il progetto *MOVAID*. Si, è infatti già analizzato in precedenza quelli che risultano i vantaggi sia delle postazioni fisse, per quanto riguarda la reattività, sia dei sistemi autonomi, appunto, per quanto riguarda l'autonomia rispetto agli altri tipi di configurazione. Il sistema *MOVAID* (MObility and actiVity Assistance system for Disabled) è un progetto TIDE (Technology Initiative for Disabled and Elderly people) della Comunità Europea con l'intento di sviluppare un sistema robotico autonomo e modulare per assistenza in ambito domestico a persone disabili o ad anziani. Durante le fasi iniziali del progetto, si eseguì uno studio approfondito su quelle che potessero essere le aspettative di persone disabili intervistando numerosi individui sia in Francia sia in Italia sia in Svizzera. La maggior parte delle persone intervistate sottolineavano il bisogno di un robot che potesse eseguire compiti particolari come far cuocere un cibo surgelato nel forno a microonde e quindi servirlo all'utente, oppure pulire la cucina o la stanza da letto. Da queste ricerche, quindi, si pensò di sviluppare un robot con cui cercare di unire le performance delle postazioni fisse con l'idea di autonomia dei robot mobili. Il sistema risultante è fondamentalmente basato sull'utilizzo di una base fissa dotata di un braccio manipolatore che può essere posizionata su di una base mobile.

Quest'ultima possiede la tecnologia che le consente di navigare in autonomia in un ambiente domestico. Il sistema fu dunque sviluppato come un sistema distribuito che includeva, appunto, la base mobile e un paio di postazioni fisse: una posizionata in cucina e l'altra nella camera da letto. Le basi fisse sono dotate di un elaboratore nel quale risiedono tutti i moduli necessari al loro funzionamento e sono interconnesse ad una rete Ethernet. Per quanto riguarda, invece, l'interfaccia utente, essa si basa su di un elaboratore con il quale l'utente, in modo grafico, può accedere a quattro livelli di funzionamento.

- Beginner level: in questo caso il robot ha ampia autonomia e l'utente può solamente scegliere tra alcuni task predefiniti.

- Standard level: in questo caso l'utente può destreggiarsi tra i task predefiniti ma anche comporne di nuovi usando alcune primitive come, per esempio "vai in cucina" o "prendi un bicchiere". Il robot, quindi, esegue questi compiti senza l'intervento dell'utente a meno che sia il robot a richiederlo nel caso in cui, per esempio, si trovi in una situazione di errore o non riesca a localizzare un oggetto. Nel caso in cui l'utente non dia risposta il robot esegue un comportamento standard.

- Advanced level: in questo caso l'utente può costruire i singoli task utilizzando primitive più generali rispetto alle precedenti (come "vai a", "cucina", "bicchiere", ...). Inoltre risulta possibile operare in teleoperazione controllando in modo indipendente ogni singolo componente del sistema. In questo livello, comunque, la teleoperazione viene supervisionata dal sistema ed integrata con l'informazione sensoriale per evitare, per esempio, che si verifichino collisioni.

- Expert level: questo risulta essere una sorta di livello per supervisionare il sistema. L'utente, infatti, può controllare completamente ogni sua componente senza che esso faccia nessun tipo di controllo. Si potrebbe affermare che è il modo di funzionamento che agisce a più basso livello.

Dopo la fase di realizzazione, il sistema fu testato con successo nei paesi in cui era stata effettuata la ricerca preliminare. Il risultato fu, in generale, molto positivo so-

prattutto per le persone disabili a livello motorio ma comunque con buone capacità cognitive.

2.3.4 Sviluppi futuri

Analizzando gli esempi di robotica per assistenza raccolti negli ultimi vent'anni, soprattutto a livello ospedaliero, possono essere fatte alcune considerazioni. Prima di tutto va detto che, in generale, i robot sono stati utilizzati in sostituzione di personale in tutti quei compiti come la pulizia dei locali, il trasporto dei campioni medici nei laboratori o dei farmaci ai pazienti e così via. In generale si può dire che i risultati sono sicuramente a favore delle macchine sia dal punto di vista dell'affidabilità sia dal punto di vista del risparmio economico. Il bilancio è altrettanto positivo se si considerano i robot che sono stati utilizzati in compiti altamente pericolosi per gli uomini come trasporto e manipolazione di materiali tossici o radioattivi.

Analizzando, invece, l'utilizzo dei robot in ambiente domestico per assistenza a disabili o a persone anziane, va sottolineato che i risultati non sono così rosei non solo a livello tecnico, ma anche a livello sociale. In questi casi, infatti, il più delle volte risulta difficile riuscire a fare accettare un robot, soprattutto ad una persona anziana. Per questo motivo, forse, ci si dovrà aspettare una lenta diffusione per la robotica finalizzata a questo tipo di compiti e ambiti di lavoro.

2.4 Progetti di ricerca che utilizzano il MANUS

Come mostrato in precedenza, i gruppi di ricerca che negli ultimi anni hanno dato vita a progetti di sistemi robotici per assistenza a persone disabili o anziani sono molteplici. Di seguito verranno presentati alcuni progetti che sono stati ritenuti significativi nel panorama internazionale sia come esempi di sistemi robotici per assistenza, sia, soprattutto, per aver affrontato il problema con differenti approcci di interazione con l'utente. Per questi progetti, nei quali viene utilizzato il robot manipolatore *MANUS*, si è ritenuto opportuno entrare con maggior dettaglio nell'analisi del loro sviluppo.

2.4.1 FRIEND[5]

Figura 2.15: Progetto FRIEND

Il gruppo di ricerca dell'Università di Breme, capeggiato dal Prof. Axel Graser sta sviluppando un sistema basato su di una configurazione a carrozzella. Il gruppo ha scelto di posizionare il manipolatore MANUS su di una carrozzella elettrica basando il controllo su di un elaboratore a bordo di essa. La cosa interessante risulta sicuramente il tipo di interfaccia con l'utente: in questo caso, infatti, oltre che con un normale monitor, l'utente può interagire con il sistema tramite la propria voce (è stato infatti utilizzato un software di riconoscimento vocale) oltre che tramite un *data glove* che permette di eseguirne il controllo tramite il cosiddetto *programming by demonstration*[6]. Per quanto riguarda il controllo vocale, il programma utilizzato (IBM ViaVoice Gold) si occupa di tradurre le parole impartite dall'utente al sistema, in comandi che, una volta inviati alla parte di controllo, vengono utilizzati per il moto. Un interessante approccio sta nel fatto che i comandi devono essere dati al sistema seguendo un tipo di sequenza ad albero in modo da minimizzare gli errori dovuti a rumore o a interpretazioni sbagliate da parte del software di riconoscimento. Ogni volta che un comando vocale viene riconosciuto come valido, il sistema lo mostra sul monitor nella zona dell'albero di stato a cui appartiene. In questo modo

l'utente può facilmente identificare cosa effettivamente il sistema stia facendo. Per quanto riguarda, invece, l'algoritmo di *programming by demostration*, esso viene utilizzato soprattutto per far apprendere al sistema nuovi comandi e nuovi movimenti. Esso può essere utilizzato tramite tastiera controllando il manipolatore in modo point-to-point (cioè spostando il gripper da un punto al successivo): in questo modo il sistema registra la traiettoria per poterla successivamente rieseguire. Esiste anche la possibilità di mostrare al manipolatore quale sia il compito da eseguire utilizzando la propria mano. Il sistema è, infatti, dotato di un *data glove* con il quale l'utente può mostrare al robot sia semplici comportamenti di tipo pick-and-place, sia elaborate traiettorie per compiti più complessi.

2.4.2 TAURO[7]

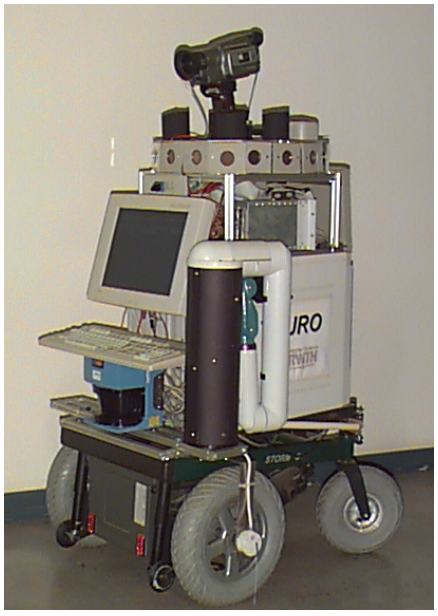

Figura 2.16: Progetto TAURO

Il progetto TAURO è stato sviluppato in Germania dall'Istituto di Scienza dell'Informazione presso l'Università di Aachen. Anche questo progetto rientra nell'idea di *robotica di assistenza*, ma in questo caso, a differenza del precedente, il progetto risulta fondato sull'idea di un robot completamente autonomo. Anche

TAURO utilizza il MANUS come robot manipolatore. La configurazione meccanica utilizzata è differente da quella del progetto FRIEND. Nel TAURO, infatti, viene utilizzata una carrozzella elettrica solo come struttura meccanica portante per il robot; quest'ultimo è dotato, oltre che di un paio di elaboratori, anche di una sensorialità molto eterogenea che va da una serie di sensori ad ultrasuoni fino ad un sensore CCD per la visione. Per questo progetto, quindi, non è necessario avere un utente a bordo del sistema per indicare di volta in volta quale sia il task da svolgere, ma il robot risulta completamente autonomo.

La configurazione del sistema è di grande interesse soprattutto dal punto di vista della realizzazione del movimento del manipolatore. Esso, infatti, risulta controllato tramite una retroazione basata sulla visione, fornita dal sensore CCD posizionato sul gripper del MANUS. In questo modo, il sistema ottiene in tempo reale la visione di un ipotetico operatore che si trovasse posizionato sopra la pinza del robot. Acquisendo immagini in successione[8] quindi, dopo un'adeguata elaborazione tramite una rete neurale, analizzando due immagini successive, il sistema di controllo riesce ad estrapolare le informazioni di moto necessarie. La via del controllo del manipolatore tramite retroazione visiva è un ottimo approccio: esso implica, infatti, che qualsiasi tipo di errore nel movimento, causato da rumore nella visione o da mal posizionamento di un giunto, venga compensato istante per istante, permettendo al gripper di posizionarsi con estrema precisione anche se forse non con grande reattività a causa delle complesse elaborazioni necessarie per ottenere l'informazione di moto.

2.4.3 Intelligent Sweet Home[9]

Questo progetto viene sviluppato dal gruppo di ricerca capeggiato dal Prof. Ju-Jang Lee presso il KAIST (Korea Advanced Institute of Science and Technology) in Corea. Questo centro nasce per condurre ricerche in campo robotico, sia in ambito civile che in quello industriale. Il progetto porta il nome di *Intelligent Sweet Home* perchè l'obiettivo di questo gruppo di ricerca è quello di creare un ambiente altamente integrato e intelligente dotato di svariati dispositivi capaci di interagire per trasformare un normale ambito civile, come un'abitazione, in un ambiente capace di essere di aiuto a chi lo abita ed in questo contesto trova collocazione anche l'idea

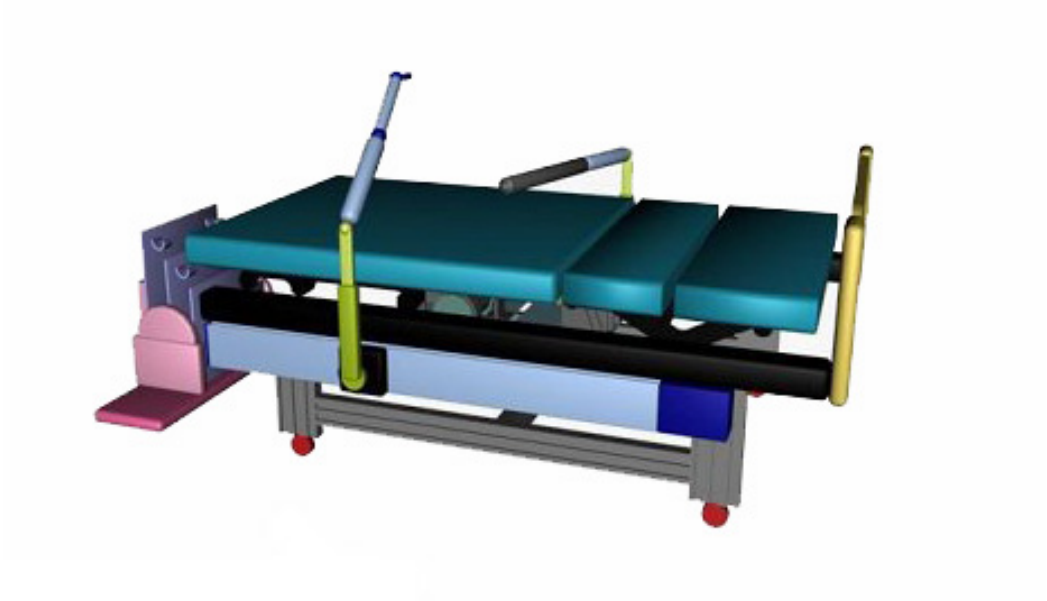

Figura 2.17: Progetto Intelligent Sweet Home

della robotica per assistenza. Nel prototipo *Intelligent Sweet Home*, vengono utilizzati svariati sistemi robotici, tra i quali un manipolatore MANUS e un robot mobile PIONEER. Con questo esempio, quindi, si vuole mettere in luce meglio l'idea di configurazione a postazione fissa anche se con alcune differenze rispetto a come è stata presentata in precedenza. Innanzitutto va detto che, in questo caso, il controllo dei dispositivi da parte dell'utente avviene tramite l'utilizzo di reti neuro-fuzzy capaci di riconoscere la gestualità e il moto. Utilizzando tre telecamere si procede acquisendo le immagini della mano dell'utente e, dopo una prima elaborazione, si tenta di ricostruirne basandosi su di esse, l'orientazione 3D. In questo modo tramite l'uso di un set di 28 immagini preregistrate e utilizzate come esempi di modi di comportamento il sistema tenta di fare un matching dell'immagine acquisita con una delle 28 presenti per poter scegliere un compito da eseguire. Il sistema integra inoltre un algoritmo per il riconoscimento della gestualità. In questo caso, però, l'utente non si limita a comunicare un compito da eseguire ad uno dei dispositivi, ma controlla direttamente il moto tramite il moto della mano. È importante sottolineare che nel progetto ISH la configurazione del manipolatore è a base fissa, essendo essa posizionata sulla sponda di un letto. In questo caso la limitazione dello spa-

zio di lavoro viene superata sfruttando l'interazione tra sistemi robotici: il MANUS esegue compiti di manipolazione di oggetti, mentre il trasporto degli stessi viene eseguito da un robot mobile PIONEER ed il tutto viene coordinato dal sistema di tre telecamere.

2.5 Requisiti per un sistema robotico di assistenza

Un sistema robotico sviluppato per compiti di assistenza dovrebbe prendere in considerazione, e, di conseguenza agire, su alcuni aspetti della vita quotidiana del singolo utente che vive in un ambiente domestico. Di seguito verrà elencata una serie di compiti che dovrebbero poter essere eseguiti da un robot per l'assistenza a persone disabili o anziane.

- Esecuzione di compiti domestici: il sistema dovrebbe essere predisposto per eseguire i normali compiti tipici di un ambiente domestico come, per esempio, trasportare cibi e bevande o pulire pavimenti o finestre.

- Comunicazione: il sistema deve permettere di comunicare in vari modi con l'utente tramite un'interfaccia semplificata, per esempio, di tipo visivo o vocale. Deve, però, anche essere in grado di dialogare con i dispositivi tipici dell'ambiente domestico, come TV o VCR, o anche con termostato per la regolazione della temperatura. Inoltre deve poter essere in grado di comunicare all'esterno, per esempio con centri medici (per chiamate di soccorso) o con le autorità.

- Organizzazione delle infrastrutture domestiche: il sistema deve permettere di agire su tutte quelle infrastrutture domestiche come aria condizionata o luci, o anche con porte o finestre per poter adattare l'ambiente alle richieste dell'utente.

- Supporto alla mobilità: il sistema deve poter fornire supporto alla mobilità dell'utente sia nel momento in cui esso debba essere guidato da una zona all'altra della casa, sia nel momento in cui esso necessiti di aiuto per alzarsi da un letto o da una poltrona.

- Supporto nel manipolare oggetti: questo risulta senza dubbio uno degli obiettivi primari di ogni sistema di assistenza. Il sistema, infatti, deve poter essere in grado di individuare e riconoscere gli oggetti che l'utente gli indica per poi procedere con la manipolazione.

- Integrazione sociale: il sistema dovrebbe poter garantire all'utente anche di mantenere i propri contatti con persone conosciute grazie a strumenti come telefono, videoconferenza o Internet.

- Sicurezza personale: il sistema dovrebbe essere in grado di monitorare la sicurezza e la salute dell'utente. Sarebbe utile tenere sotto controllo tutti i parametri vitali come pressione, pulsazioni e temperatura. Nel momento in cui viene registrato qualcosa di anomalo, il sistema dovrebbe essere in grado di attivare una chiamata di emergenza ad un centro competente.

2.6 Implicazioni sociali

Sebbene il tono utilizzato nella letteratura che illustra i progetti di ricerca sulla robotica per assistenza sia in generale piuttosto ottimistico, bisogna prendere in considerazione anche alcune questioni di carattere sociale che, da un certo punto di vista, possono mitigarne l'euforia.

- Vanno tenuti ben presente i rischi di mal funzionamento del sistema, e di conseguenza risulta necessario tenere in considerazione un piano di intervento.

- L'esperienza insegna che non sempre l'accettazione da parte dell'utente dei sistemi robotici risulta facile. Gli utenti, infatti, sono ben disposti verso i robot solo dopo che ne hanno avuto esperienza. Sarebbe necessario, quindi, promuovere un piano di inserimento graduale.

- L'esperienza insegna anche che non sempre questi sistemi sono ben accettati non solo dai pazienti ma anche dal personale preposto alla loro cura. Di conseguenza sarebbe necessario creare un piano di addestramento del personale specializzato per garantire una conoscenza più approfondita dei robot.

- Non sempre le dimensioni risultano compatibili con un normale ambiente domestico e non sempre strutturare ad hoc l'abitazione è possibile.
- Come tutti i dispositivi meccanici, anche i robot sono soggetti a rotture e a manutenzione. È per questo motivo che i costruttori dovrebbero prevedere piani di manutenzione dei robot durante il loro normale utilizzo presso l'utente.
- Nel momento in cui si utilizzano più sistemi robotici nello stesso ambiente, è importante valutare bene a priori l'iteroperabilità e la possibilità di cooperazione.

Capitolo 3

I componenti del manipolatore mobile

3.1 Introduzione

In questo capitolo si metteranno in luce gli aspetti costitutivi del progetto che ha ispirato questo lavoro di tesi. L'obiettivo fondamentale è realizzare un robot mobile dotato di braccio manipolatore con il quale sia possibile eseguire compiti di assistenza come mostrato in figura 3.1.

Per prima cosa verrà svolta una breve analisi iniziando con il descrivere i robot che sono stati utilizzati per il progetto. Quindi si descriverà la creazione di un'interfaccia per la comunicazione tra i robot, realizzata con l'obiettivo di dare la possibilità alla base mobile di controllare il manipolatore. Le parti costitutive fondamentali della struttura robotica sviluppata in questo progetto risultano, pertanto, le seguenti:

- Il *NOMAD* è il robot mobile utilizzato come base su cui verrà posizionato il manipolatore.
- Il *MANUS* è il robot manipolatore scelto per poter svolgere compiti di manipolazione di oggetti.
- L'interfaccia costituisce ciò che unisce i due robot a livello di hardware e di protocollo.

Figura 3.1: Il manipolatore mobile.

3.2 La base mobile Nomad 200

Il Nomad 200[10] è un robot mobile sviluppato dalla società americana *Nomadic Technologies* durante la prima metà degli anni novanta. È un robot di medie dimensioni con una sensoristica molto eterogenea. Come evidenziato in figura 3.2, esso ha una struttura molto compatta (50 cm di diametro e 80 cm di altezza) nel quale sono stati alloggiati tutti i dispositivi sia di controllo sia di sensoristica in modo tale da creare un robot con caratteristiche quasi di olonomie, capace di muoversi in un ambiente indoor.

Il movimento del robot è realizzato tramite tre ruote che lo sorreggono e trovano alloggiamento nella base. Un unico motore, tramite un sistema di trasmissione,

Figura 3.2: Il Nomad 200.

produce la coppia necessaria al loro movimento. Risulta possibile altresì ruotare la base in cui sono presenti le ruote per far si che il robot possa scegliere una direzione piuttosto che un'altra.

Per quanto riguarda la sensorialità, il Nomad è dotato di una vasta gamma di sensori di varia natura. Si può fare una netta suddivisione tra sensorialità *propriocettiva* e sensorialità *eterocettiva*. Per quanto riguarda la sensorialità propriocettiva vanno citate l'odometria che permette al robot, tramite le informazioni provenienti dalla parte di controllo dei motori, di tenere traccia del moto compiuto e la bussola che permette di avere un'informazione sempre precisa con la quale il robot può orientarsi rispetto al nord magnetico. Per quanto riguarda la sensorialità eterocettiva, invece, vanno citati i sensori di contatto, i sensori ad infrarossi e quelli ad ultrasuo-

ni. I primi, realizzati tramite una serie di microinterruttori, sono utili soltanto nel momento in cui il robot collide con un ostacolo. Sia i sensori ad infrarossi sia gli ultrasuoni, invece, permettono al robot di misurare la distanza alla quale vengono rilevati gli ostacoli. Il robot è dotato anche di un sistema di visione stereo utilizzato per ottenere informazioni particolareggiate sull'ambiente circostante e per poter, per esempio, pianificare una traiettoria di moto.

Il sistema di elaborazione, a seguito di un recente intervento di aggiornamento è costituito da un processore *Intel Pentium III* a 1GHz con 256 MB di memoria RAM. Il robot, infatti, in precedenza, era nato con un PC con processore *Intel 486DX2* e 16MB di memoria RAM. Dovendo, però eseguire operazioni per controllare sia la parte motoria sia la parte sensoriale oltre che quella relativa all'elaborazione dei dati per la pianificazione del moto, si è deciso di effettuare l'aggiornameto ottenendo, così, un elaboratore con maggiori risorse computazionali.

3.3 Il manipolatore Manus

Il *Manus* è un robot manipolatore sviluppato dalla società olandese *Exact Dynamics*. Esso è nato principalmente con lo scopo di fornire assistenza a persone affette da handicap e incapaci di compiere alcune delle normali azioni quotidiane. La società olandese ha iniziato a sviluppare il progetto del manipolatore già all'inizio degli anni novanta con l'intento di realizzare uno strumento finalizzato ad essere posizionato su di una carrozzella elettrica ed essere controllato tramite un sistema che potesse essere utilizzato da persone non in grado di muovere gli arti[11].

Il *MANUS*[12] è un robot manipolatore dotato di sei gradi di libertà grazie alla cui configurazione il gripper può essere orientato liberamente nello spazio di lavoro, in ogni direzione. Mediante un semplice controllo da parte dell'utente basato sull'utilizzo di un keypad o di un joystick, è possibile che il gripper del manipolatore venga mosso e posizionato in modo opportuno al fine di realizzare compiti che altrimenti sarebbero impossibili da realizzare. Per esempio è possibile afferrare oggetti posizionati a terra o su piani rialzati, aprire e chiudere porte o finestre e manipolare oggetti di vario genere.

I sei gradi di libertà del MANUS gli conferiscono la caratteristica di poter assumere tutte le posizioni nello spazio di lavoro. Il concetto di spazio di lavoro risulta

però fondamentale per poter capire quale sia l'effettiva raggiungibilità dei punti di interesse. La figura 3.3 evidenzia graficamente i limiti del campo di lavoro del manipolatore. Come si può notare lo spazio di lavoro risultante è approssimativamente sferico.

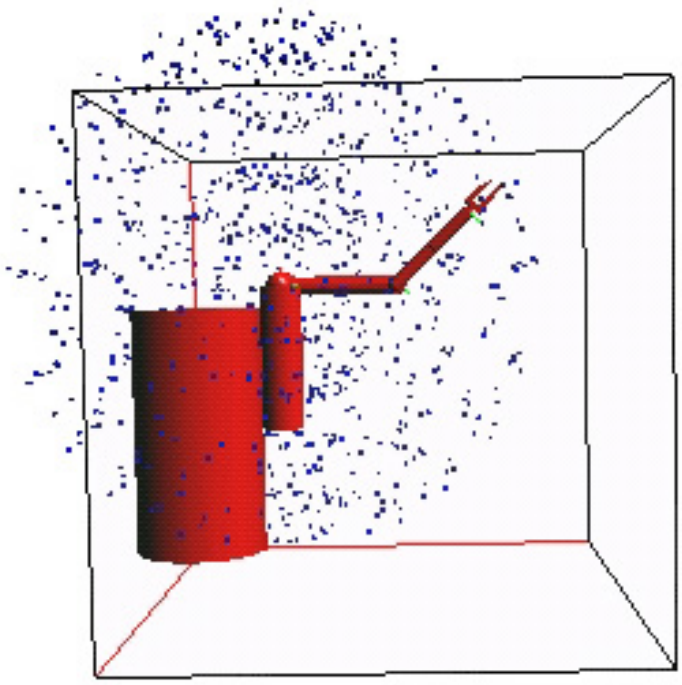

Figura 3.3: Spazio di raggiungibilità.

Si nota inoltre che se il manipolatore risulta posizionato in un punto fissato, il campo di lavoro risulta piuttosto ristretto, come si verifica in generale per i robot manipolatori a base fissa.

3.3.1 Caratteristiche meccaniche del manipolatore

La figura 3.4 presenta le dimensioni del manipolatore in posizione di riposo. Per ridurre gli ingombri e per facilitarne il trasporto, infatti, il manipolatore è stato pre-

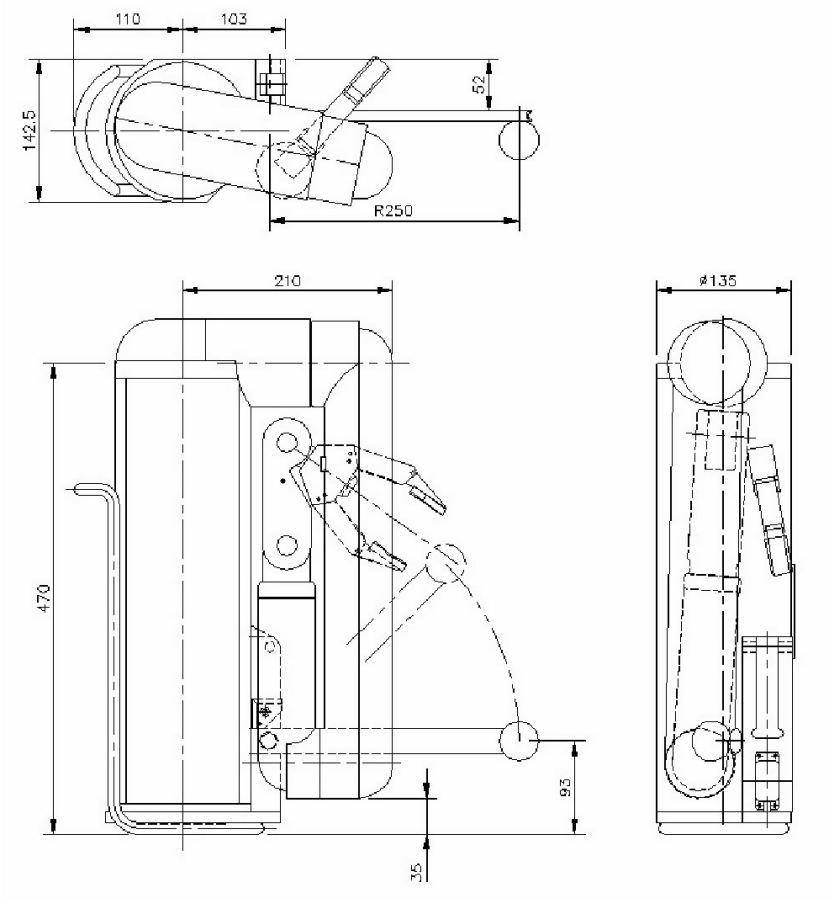

Figura 3.4: Dimensioni.

disposto di due apposite procedure che ne permettono la chiusura (*Fold-IN*) nel momento in cui non viene utilizzato e il successivo riposizionamento in posizione iniziale (*Fold-OUT*).

La figura 3.5 mostra i sei gradi di libertà del manipolatore.

Per quanto riguarda le dimensioni dal punto di vista dell'estensione, il MANUS può raggiungere una lunghezza di 80 cm. In figura 3.6 viene mostrato il limite di questa estensione. Questa estensione permette al MANUS di avere un apprezzabile spazio di lavoro una volta che esso venga posizionato su di una carrozzella. Per migliorare, comunque, le prestazioni in tutti i casi in cui sia necessario raggiungere oggetti o posti molto in basso, per esempio a terra, o posti molto in alto, per esempio su scaffali o librerie, è stata aggiunta la possibilità, sempre per il caso del posiziona-

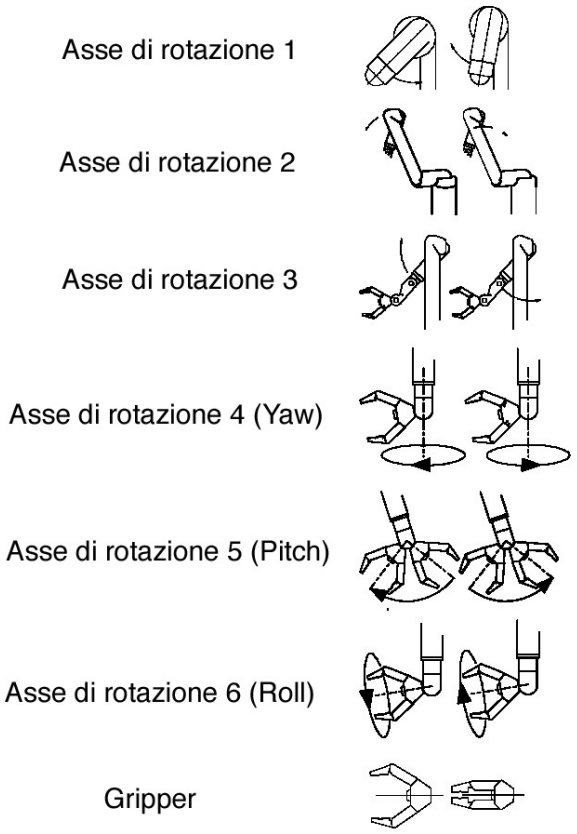

Figura 3.5: Gradi di libertà.

mento su carrozzella, di utilizzare un giunto aggiuntivo, il cosiddetto *lift joint*. Esso permette alla base del manipolatore di assumere due posizioni variandone l'altezza di aggancio.

Analizzando la struttura meccanica, si nota che i giunti, oltre ad essere controllati tramite encoder assoluti che garantiscono un'ottima precisione nel posizionamento, sono dotati di una peculiarità importante: il moto, infatti, viene realizzato tramite un sistema di cinghie che trasferiscono ai singoli gradi di libertà il moto proveniente dai motori che trovano alloggiamento tutti nella base del manipolatore. Questa caratteristica è stata studiata appunto per cercare di limitare le dimensioni e il peso del manipolatore. Esso, infatti pesa soltanto 13Kg. Naturalmente questo tipo di approccio alla realizzazione del moto pone alcuni vincoli, in particolare al massimo

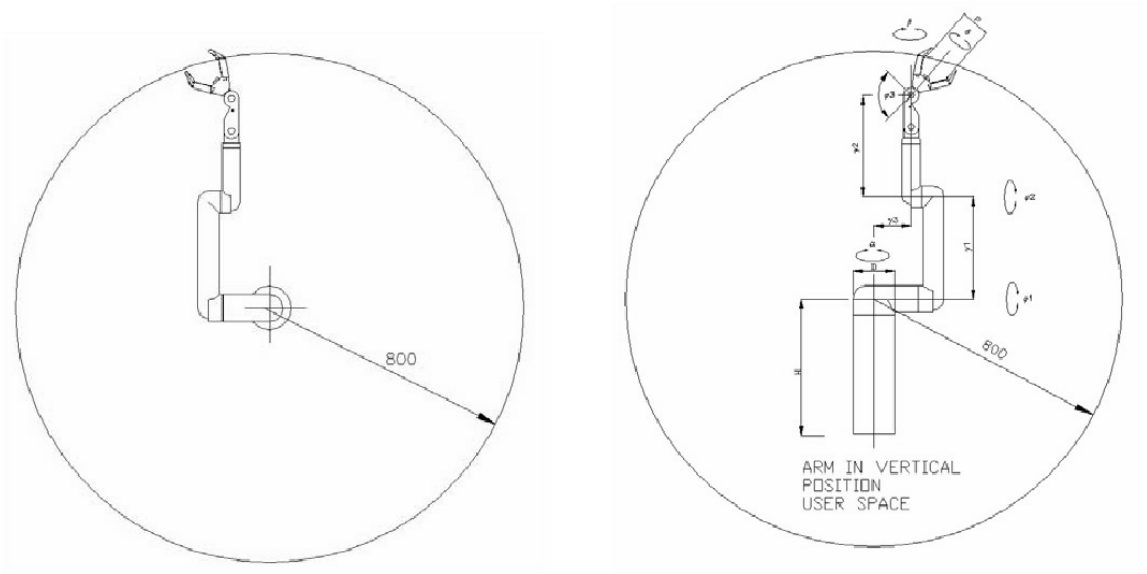

Figura 3.6: Estensibilità.

peso trasportabile che risulta, limitato a 1,5Kg. Parlando dell'aspetto meccanico, va accennato anche l'aspetto della sicurezza: è, infatti, essenziale, visto l'ambito di lavoro a cui il MANUS è destinato, che siano presenti alcuni accorgimenti che rendano il manipolatore assolutamente privo di qualsiasi pericolosità per l'utente. Per prima cosa va osservato, oltre al funzionamento del *MANUS* a bassa tensione, anche che la potenza degli attuatori è stata volutamente limitata soprattutto in termini di velocità e reattività. Osservando, infatti, il manipolatore in movimento e confrontandolo con un manipolatore industriale come per esempio il *PUMA560*, si nota proprio come il primo risulti molto meno reattivo e più impreciso rispetto al robot industriale. Inoltre ogni giunto risulta dotato di un ulteriore struttura di sicurezza realizzata mediante l'utilizzo di *slipring*: essi infatti permettono sia di muovere manualmente i giunti senza creare danni agli attuatori, sia, cosa molto più importante, permettono di evitare che il movimento dei giunti possa creare danni nel momento in cui si verifica una collisione. Infatti, la forza esercitata risulta limitata dalla presenza di questo accorgimento che provoca il movimento dell'attuatore evitando così che vengano provocati danni.

Da quanto detto, risulta chiaro che, per un progetto in ambito robotico in cui

sia richiesto l'utilizzo di un manipolatore e, in particolare, il suo interfacciamento con un robot preesistente, le caratteristiche sopra esposte risultano essenziali sia dal punto di vista dell'ingombro sia da quello del peso che risulta limitato e, per questo, adattabile ad una base mobile.

3.3.2 Elettronica di controllo del manipolatore

Il sistema di controllo permette di muovere il manipolatore sia nello spazio dei giunti, impostando cioè singolarmente il valore di ogni giunto, sia nello spazio cartesiano, muovendo cioè il manipolatore lungo gli assi cartesiani nello spazio. Analizzando in dettaglio il sistema di controllo, mostrato in figura 3.7, si individuano tre microprocessori. Il processore principale è l'INTEL 80C186 ed è rappresentato dal

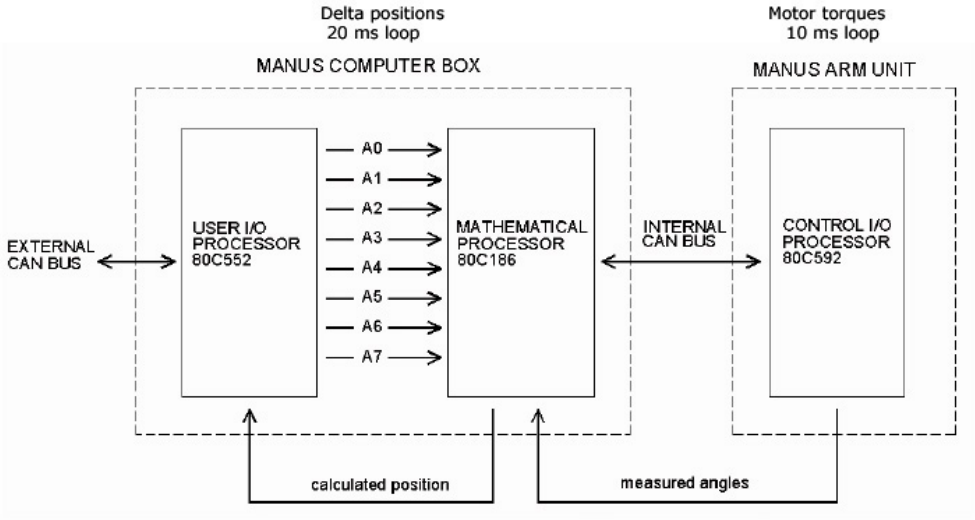

Figura 3.7: Struttura di controllo del manipolatore.

Mathematical processor. Esso si occupa di eseguire le trasformate quando il manipolatore opera in *Cartesian Mode*, di verificare le traiettorie per evitare eventuali autocollisioni ed, inoltre, calcola con un ciclo di 10ms le coppie richieste dai motori con un controllo di tipo PI (proporzionale-integrativo) in base alla retroazione sulle velocità. Per quanto riguarda, invece, il posizionamento che il MANUS deve assumere, il set point viene comunicato al *Mathematical processor* dall'utente esterno

tramite un secondo processore (80C552), lo *User I/O processor*. Quest'ultimo si occupa, appunto, di elaborare le informazioni provenienti dal mondo esterno. Una volta conclusa l'elaborazione da parte del 80C186, i dati di coppia vengono trasmessi ai motori tramite il terzo microprocessore (80C592): il *Control I/O processor*. La comunicazione tra il *Mathematical processor* e il *Control I/O processor* viene realizzata tramite il protocollo CAN-Bus ed è identificata da *Internal CAN bus* che si distingue dall'*External CAN bus*. Il primo, infatti realizza la comunicazione tra il control box e gli attuatori mentre il secondo realizza la comunicazione tra l'utente e il control box. L'*External CAN bus* viene utilizzato, infatti, quando si richiede che il manipolatore venga comandato da un elaboratore. Non risulta possibile, invece, connettere direttamente un elaboratore all'*Internal CAN bus*.

È proprio il sistema di controllo del manipolatore che permette di impostare alcuni stati di funzionamento che l'utente esterno deve selezionare per procedere nel moto del MANUS. La figura 3.8 ne mostra una schematizzazione:

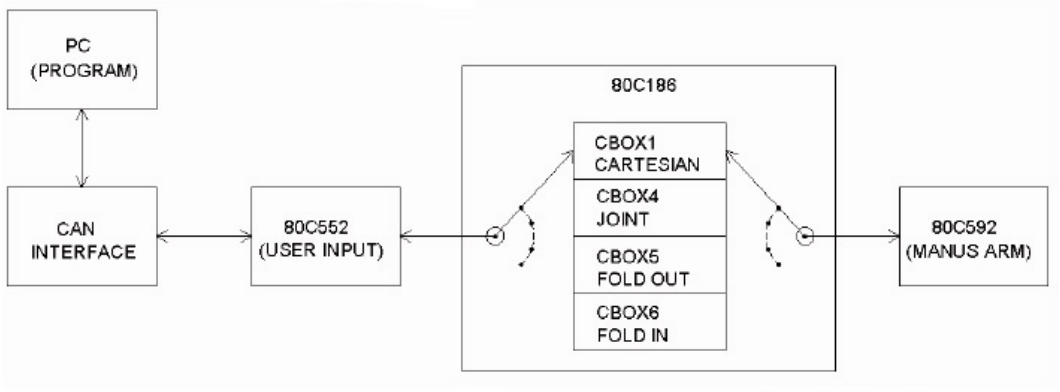

Figura 3.8: Stati di funzionamento del manipolatore.

- Nello stato cartesiano (cbox1) il gripper del manipolatore Manus può essere traslato lungo gli assi cartesiani *x, y, z*.
- Nello stato dei giunti (cbox4) ogni singolo asse di rotazione di ogni grado di libertà del manipolatore può essere orientato liberamente. Questo stato di funzionamento risulta essere quello maggiormente libero da vincoli da parte

del sistema di controllo del manipolatore. Nello stato precedente, infatti, il sistema di controllo si occupa di effettuare, oltre che la cinematica inversa, anche alcuni controlli di sicurezza per evitare eventuali autocollisioni. Questo non risulta presente in questo stato di funzionamento dato che si agisce direttamente su di ogni singolo grado di libertà.

- Gli stati cbox5 e cbox6 permettono di accedere a due procedure preprogrammate dal costruttore che fanno sì che il manipolatore rispettivamente partendo dalla posizione di riposo si posizioni in quella operativa e, al contrario, da qualsiasi posizione operativa vada a posizionarsi nella posizione di riposo in modo totalmente automatico.

Come ultima osservazione, alla luce del come è stato sviluppato il manipolatore, risulta chiaro che l'alimentazione e l'assorbimento debbano essere compatibili con la configurazione standard in cui viene utilizzato. Esso, infatti, è stato ottimizzato sotto il profilo della tensione di alimentazione e della corrente assorbita. L'alimentazione necessita, infatti, di una tensione di 24 V e di una corrente di 2 A continui.

3.4 L'interfaccia elettronica e di comunicazione

In questo paragrafo verrà prima di tutto analizzato l'hardware utilizzato per creare il canale di comunicazione tra il sistema di controllo del manipolatore e l'elaboratore esterno. Quindi, dopo aver brevemente spiegato il funzionamento del protocollo CAN-Bus, si vedrà come poter sfruttare lo scambio dei messaggi CAN in modo da soddisfare le specifiche richieste dal costruttore del MANUS per realizzare la comunicazione.

3.4.1 L'hardware

Come accennato nel paragrafo precedente, per poter comunicare con il sistema di controllo del manipolatore, risulta necessario inviare messaggi su di una rete CAN per la quale è necessario utilizzare un controller.

Figura 3.9: Controller CAN.

Il costruttore del manipolatore fornisce tra i vari accessori, anche un kit denominato *Transparent Mode* nel quale sono contenute, oltre ad una piccola utility che mostra le funzioni ottenibili tramite il controllo esterno, anche una scheda ISA 16-bit dotata di un controller per gestire la comunicazione sulla rete CAN. Il problema essenziale di questa scheda è che essa è poco adattabile ai sistemi odierni, dato che il bus ISA risulta ormai sorpassato e sostituito dal più moderno bus PCI. Questa considerazione ha spinto a cercare una scheda PCI che fornisse un'interfaccia CAN-Bus. Analizzando le varie possibilità presenti sul mercato dell'automazione industriale, ci si è indirizzati verso l'acquisto di 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.

La scheda ISA 16-bit fornita nel kit *Transparent Mode* dal costruttore, è dotata di un integrato 82C200 per la comunicazione sulla rete CAN. Essa risulta molto semplice dal punto di vista architetturale: utilizza, infatti, un indirizzo di memoria fissato a priori e non modificabile con il quale si può accedere al controller per realizzare la comunicazione; inoltre non viene previsto l'utilizzo di nessun tipo

di interrupt per la gestione di eccezioni. Per queste ragioni, essa non necessita di particolari applicazioni che ne leghino il funzionamento al kernel del sistema operativo. Al contrario, analizzando il codice sorgente dell'utility fornita dal produttore si vede come la comunicazione sia realizzata accedendo direttamente all'hardware utilizzando l'indirizzo base della scheda fissato a 0x300 per le funzioni di I/O.

Il controller scelto, invece, è una scheda PCI che realizza la comunicazione su di una rete CAN sfruttando l'integrato *SJA1000* prodotto da *Philips Semiconductors* e largamente utilizzato in tutti i dispositivi di questo tipo. L'interfacciamento con il bus PCI viene invece realizzato tramite un PCI-Controller: il *PCI9052* prodotto da *PLX Technology*. In questo caso, quindi, la gestione dell'accesso alla scheda risulta più complessa rispetto al caso precedente, permettendo comunque una maggior flessibilità nell'utilizzo degli interrupt generati dal controller SJA1000. A differenza della scheda precedente, ora risulta necessario utilizzare un driver che si occupa di ottenere dal Kernel l'indirizzo assegnato alla scheda dal sistema. Con questo indirizzo, risulta dunque possibile accedere per la scrittura e la lettura dei messaggi ai buffer presenti sul controller. In questo caso è lo stesso produttore che fornisce un driver per la gestione della scheda e una libreria nella quale sono presenti molte funzioni tra cui `CannesGetMessage`[13] e `CannesSendMessage` rispettivamente per inviare e leggere un messaggio e `CannesGetInQueueCount` per notificare la presenza di messaggi nel buffer di ricezione.

Per approfondire la presentazione del sistema di comunicazione, risulta necessario analizzare in dettaglio il funzionamento del dispositivo SJA1000. Esso infatti rappresenta il cuore della comunicazione per quanto riguarda la rete CAN. Va innanzitutto detto che sulla scheda fornita nel kit *transparent mode* viene utilizzato il chip 82C200, il quale, però, presenta assoluta compatibilità elettronica con il fratello maggiore SJA1000[14]. Quest'ultimo è un controller stand-alone per la gestione della comunicazione su rete CAN. Esso ad una prima analisi appare come un microcontrollore, un dispositivo cioè basato sull'idea di I/O mappato in memoria. L'area di memoria del dispositivo consiste, infatti, nel set di indirizzi corrispondenti ai registri sia dell'area di controllo, sia dei comandi da eseguire, sia dei buffer di trasmissione. I registri di controllo vengono scritti in fase di inizializzazione, appunto per configurare i parametri per la comunicazione. In questa fase, infatti, devono venire impostate tutte le informazione relative sia alla configurazione delle maschere

di accettazione dei messaggi, sia i parametri relativi al timing della comunicazione e al bitrate. Dato che questi dati devono essere modificati solo in fase di inizializzazione, sono stati creati due modi di funzionamento ben distinti: il *Reset Mode* e l'*Operating Mode*. Il primo, attivato ogni qualvolta si procede ad un reset dell'hardware, rappresenta il solo modo di funzionamento mediante il quale si ha la possibilità di modificare i registri di configurazione. Una volta effettuato il passaggio all'*Operating Mode* resettando il request bit nel *Control Register*, i registri di configurazione non possono più essere modificati e le uniche aree di memoria a cui si può accedere in I/O sono quelle relative alla comunicazione.

Per la comunicazione, risulta necessario prima di tutto sapere quando è opportuno leggere o scrivere nei buffer, quando cioè in lettura è disponibile un messaggio nel buffer di lettura o quando il buffer di scrittura è libero. Questa informazione viene notificata dal controller per mezzo dello *Status Register*, che deve essere periodicamente letto prima di effettuare ogni operazione di I/O. Inoltre non basta leggere o scrivere i dati operando sui relativi buffer, ma è necessario anche informare il controller dopo aver terminato la scrittura o la lettura, in modo che esso possa procedere inviando o cancellando i dati presenti.

3.4.2 Comunicazione CANBus

Il *CAN-Bus* è un protocollo di comunicazione introdotto per la prima volta da Robert Bosch nel febbraio del 1986 a Detroit. Letteralmente CAN sta per Controller Area Network. Il protocollo CAN fu creato per scambiare messaggi tra dispositivi intelligenti in ambito industriale secondo lo schema master-slave. Già dal 1987 *INTEL* iniziò la commercializzazione di un integrato (82526) studiato per realizzare la comunicazione di vari dispositivi tramite queste specifiche. Successivamente anche *Philips Semiconductors* iniziò la commercializzazione di un controller (82C200) con le medesime funzionalità. A tutt'oggi il protocollo risulta largamente utilizzato in ambito industriale, e soprattutto in ambito automobilistico: in Europa, infatti, ogni nuova autovettura risulta equipaggiata con almeno una rete CAN con la quale è possibile controllare tutta la struttura elettronica del veicolo.

Come accennato, il protocollo *CAN-Bus* è stato sviluppato su specifiche che lo hanno reso molto adatto ad essere utilizzato in ambito industriale. In questo con-

testo dove la forte presenza di interferenza E.M. e le lunghe distanze rendono non facile la comunicazione tra dispositivi elettronici intelligenti (come attuatori o sensori), non sempre risulta semplice realizzare una comunicazione robusta. Proprio per questo a livello fisico il CAN-Bus risulta simile al protocollo seriale RS485, anch'esso sviluppato per un ambito industriale. Per entrambi, infatti, la trasmissione si realizza su doppino in modo differenziale. Per il CAN-Bus le distanze massime tra due dispositivi possono variare da qualche decina di metri nel caso che si trasmetta alla velocità massima (1 Mbps) fino ad oltre un chilometro per bit rate più bassi.

Il CAN-Bus è un bus di tipo multi-master: i messaggi non sono diretti a destinazioni specifiche, ma istante per istante ogni dispositivo o endpoint può assumere il controllo del bus e spedire un messaggio. Ogni messaggio ha un ID che lo identifica e ogni endpoint decide autonomamente (in base a maschere che si possono impostare) se il messaggio risulta di sua competenza. Questo ovviamente implica che si possano spedire messaggi anche a più endpoint contemporaneamente. Chiaramente una struttura di questo tipo presuppone il fatto che sia possibile assegnare diverse priorità ai messaggi. È infatti possibile che due endpoint cerchino di assumere il controllo del bus contemporaneamente per spedire il proprio messaggio. In questo caso la priorità può garantire una latenza massima nel caso in cui il bus sia usato per scopi di controllo realtime.

Nel protocollo CAN-Bus è presente anche una buona immunità agli errori tramite una ottima failure detection che permette di collegare più endpoint contemporaneamente e nel caso che un endpoint sia malfunzionante, il bus è in grado di escluderlo autonomamente. Sui messaggi viene effettuato anche il controllo/correzione degli errori, ed è garantito che ciascun messaggio sia ricevuto da tutti oppure da nessun endpoint. In caso di errore, infatti, viene effettuata la ritrasmissione automatica non appena possibile.

Un messaggio del protocollo CAN-Bus possiede un ID per l'identificazione, 8 byte che contengono i dati codificati, un flag (rtr) che indica al sistema in ascolto se il messaggio necessita di una risposta ed, infine, un byte che indica la lunghezza del campo dati. La tabella 3.1 riassume le struttura di un messaggio CAN.

Parametro	Tipo di dato	Esempio
ID	numero esadecimale	350, 37F, ...
rtr	flag	0 o 1
len	numero	0 ... 8
data	8 Bytes	12345678

Tabella 3.1: Formato di un messaggio CAN-Bus.

3.4.3 Protocollo del manipolatore

Come descritto in precedenza, il *CAN-Bus* supporta uno scambio di messaggi identificati da un *ID*, spediti a tutti i dispositivi in ascolto sul network e filtrati tramite una maschera che in base all'ID ne permette l'identificazione. Anche il manipolatore MANUS sfrutta questo tipo di protocollo e nel momento in cui si intende controllare il moto del manipolatore tramite un elaboratore dotato di controller CAN-Bus, si devono approfondire oltre alle specifiche per lo scambio di messaggi, anche l'architettura e la tempistica con la quale procedere nella comunicazione con il sistema di controllo.

Nel seguito vengono descritti i parametri di configurazione che bisogna impostare per stabilire una comunicazione con il sistema di controllo del manipolatore. Tali parametri sono stati ricavati tramite un'operazione di "reverse engineering" del software fornito dal costruttore. Per prima cosa risulta necessario disabilitare la resistenza di terminazione presente sul controller esterno. Il protocollo CAN-Bus permette infatti di abilitare o meno una resistenza di valore 120 Ω per terminare il bus. Il protocollo permette, inoltre, il riconoscimento dei messaggi che transitano sulla rete tramite alcune maschere utilizzate per accettare solo quei messaggi effettivamente indirizzati al dispositivo. In questo caso, essendo la rete CAN costituita soltanto da due endpoint (il sistema di controllo del manipolatore e l'elaboratore esterno), risulta utile accettare qualsiasi messaggio che transita sulla rete. Di conseguenza le maschere per il filtraggio dei messaggi possono essere neutralizzate associando ad esse il valore `0xff`. Come ultimo parametro va definito il bitrate. Dal codice sorgente fornito dal costruttore del MANUS si comprende che esso deve avere un valore pari a 250kbps. Si veda l'appendice A per il codice sorgente.

Un discorso più articolato è necessario in merito alla tempistica per lo scambio dei messaggi. L'interazione tra il controller CAN esterno e lo *User I/O processor*

deve avvenire, infatti, secondo un preciso schema di tipo *domanda e risposta*. Lo *User I/O processor* spedisce sul CAN-Bus esterno un messaggio ogni 20 ms. I primi due messaggi spediti contengono informazioni sullo stato del manipolatore in quel momento: presenza di eventuali warning o errori, stato del manipolatore (stato cartesiano, joint...) e sua posizione. Dopo i primi due messaggi, ne viene spedito un terzo che richiede al controller CAN in ascolto sul bus di inviare un messaggio di risposta. Ne deriva che il dispositivo esterno deve inviare sul bus un messaggio ogni 60 ms. Non è sempre necessario che il messaggio di risposta venga inviato. Il manipolatore, infatti, con il messaggio ricevuto dall' esterno riceve le informazioni relative al suo stato futuro e al movimento che dovrà effettuare. Se non viene fornito nessun messaggio, esso continua ad elaborare le ultime informazioni ricevute. Se, al contrario, il dispositivo esterno invia un nuovo messaggio in risposta, le informazioni precedenti vengono sovrascritte. Le tabelle 3.2 e 3.3 riassumono le tipologie di messaggi e di informazioni contenute e scambiate.

Informazione User I/O			**Risposta**	**Descrizione**
ID	RTR	LEN		
0x350	0	8	nessuna	stato e orientazione assi da 1 a 3
0x360	0	8	nessuna	orientazione assi da 4 a 6

Tabella 3.2: Messaggi CAN-Bus.

Domanda User I/O			**Risposta esterna**			**Stato**	**Descrizione**
ID	RTR	LEN	ID	RTR	LEN		
0x37f	1	0	0x370	0	0	cbox0	Inizializzazione
0x37f	1	0	0x371	0	8	cbox1	Valore spostamento assi
0x37f	1	0	0x374	0	8	cbox4	Valore angoli giunti
0x37f	1	0	0x375	0	0	cbox5	Fold-Out
0x37f	1	0	0x376	0	0	cbox6	Fold-In

Tabella 3.3: Risposte CAN-Bus.

È da sottolineare che la tempistica della risposta al terzo messaggio (ID 37F) del manipolatore non risulta essere critica: essa infatti può essere inviata in un qualsiasi istante compreso nei 20 ms successivi al ricevimento del terzo messaggio. Se però il messaggio esterno perviene al sistema di controllo del manipolatore oltre questi 20 ms, esso agirà come se non fosse stato inviato nessun messaggio. Va inoltre

sottolineato il fatto che, nel momento in cui viene stabilita per la prima volta la comunicazione, per prima cosa dall'esterno deve pervenire il tipo di stato da utilizzare. La tabella 3.4 mostra un esempio di comunicazione.

TEMPO	DESCRIZIONE
20ms	85C552 spedisce un messaggio (ID 0x350). Informazioni di posizione
40ms	85C552 spedisce un messaggio (ID 0x360). Informazioni di posizione
60ms	85C552 spedisce un messaggio (ID 0x37F). Richiesta stato e movimento
60ms + ??	Risposta: stato = Joint, movimento = 0,0,0,0,0,0,0,0
80ms	85C552 spedisce un messaggio (ID 0x350). Informazioni di posizione
100ms	85C552 spedisce un messaggio (ID 0x360). Informazioni di posizione
120ms	85C552 spedisce un messaggio (ID 0x37F). Richiesta stato e movimento
120ms + ??	Risposta: stato = Joint, movimento = x,x,x,x,x,x,x,x

Tabella 3.4: Esempio di comunicazione

Negli stati di funzionamento cbox1 e cbox4 (Cartesiano e Joint) la posizione in cui si trova il manipolatore risulta comunicata dallo *User I/O processor* al dispositivo esterno tramite l'invio dei primi due messaggi (ID 0x350 e 0x360). Le informazioni risultano codificate secondo le tabelle 3.5 e 3.6. I primi due byte del

Byte	Valore	cbox1	cbox4
1	Errore di movimento	Byte di stato	Byte di stato
2	DOF bloccato	Messaggio di errore	Messaggio di errore
3	MSB	Valore asse X	Valore giunto 1
4	LSB	Valore asse X	Valore giunto 1
5	MSB	Valore asse Y	Valore giunto 2
6	LSB	Valore asse Y	Valore giunto 2
7	MSB	Valore asse Z	Valore giunto 3
8	LSB	Valore asse Z	Valore giunto 3

Tabella 3.5: Informazione proveniente dal messaggio con ID 350.

messaggio con ID 0x350 comunicano lo stato del manipolatore ed eventuali messaggi di errore come possibili gradi di libertà bloccati. I restanti byte codificano l'informazione di posizione del MANUS secondo la codifica indicata in tabella.

I byte del messaggio CAN che codificano l'informazione da inviare al sistema di controllo del manipolatore assumono significati diversi in base allo stato di funzionamento.

Byte	Valore	cbox1	cbox4
1	MSB	Yaw	Valore giunto 4
2	LSB	Yaw	Valore giunto 4
3	MSB	Pitch	Valore giunto 5
4	LSB	Picth	Valore giunto 5
5	MSB	Roll	Valore giunto 6
6	LSB	Roll	Valore giunto 6
7	MSB	Gripper	Gripper
8	LSB	Gripper	Gripper

Tabella 3.6: Informazione proveniente dal messaggio con ID 360.

Byte	Parametro	Delta di incremento	Range Minimo	Range Massimo
1	Lift Joint	Up / Off / Down	Down	Up
2	Asse X	0.022 mm	0	127
3	Asse Y	0.022 mm	0	127
4	Asse Z	0.022 mm	0	127
5	Yaw	0.1 °	0	10
6	Pitch	0.1 °	0	10
7	Roll	0.1 °	0	10
8	Gripper	0.1 mm	0	15

Tabella 3.7: Cbox1: codifica per lo stato cartesiano

Byte	Parametro	Delta di incremento	Range Minimo	Range Massimo
1	Lift Joint	Up / Off / Down	Down	Up
2	Giunto 1	0.1 °	0	10
3	Giunto 2	0.1 °	0	10
4	Giunto 3	0.1 °	0	10
5	Yaw	0.1 °	0	10
6	Pitch	0.1 °	0	10
7	Roll	0.1 °	0	10
8	Gripper	0.1 mm	0	15

Tabella 3.8: Cbox4: codifica per lo stato dei giunti

Nelle tabelle 3.7 e 3.8 viene indicata la modalità di codifica dell'informazione. Per esempio, se viene selezionato lo stato cbox1 e viene inviato un messaggio dove il byte numero due è settato al valore 10, il manipolatore si muoverà di 0.22 mm(=10x0.022) dopo 60 ms: questo infatti è dovuto al fatto che l'informa-

zione viene ricevuta dallo *User I/O processor*, elaborata, quindi inviata al *Mathematical processor*. Se nel ciclo successivo l'informazione non varia il manipolatore fisserà una velocità di 0.22 mm ogni 20 ms dato che lo *User I/O processor* invierà al *Mathematical processor* la stessa informazione con un ciclo di 20 ms. Oltre alla traslazione lungo gli assi, bisogna fornire al sistema di controllo del manipolatore anche i valori dei giunti degli ultimi tre gradi di libertà, permettendo in questo modo, anche la possibilità di raggiungere il punto prefissato specificando l'orientazione del gripper.

Analizzando ora il metodo di controllo del manipolatore, si nota come il *Mathematical processor* controlli il movimento sulla base della velocità che viene impostata dall'esterno. L'elaboratore, infatti, comunica una posizione *p* (nello stato cartesiano o in quello dei giunti) che dovrà essere sommata alla posizione attuale. Una volta che il messaggio è stato elaborato dello *User I/O processor*, esso viene passato al *Mathematical processor* ogni 20 ms. In definitiva il parametro di controllo fornito al manipolatore è una velocità di valore $v = \frac{p}{20} 10^{-3}\ \mathrm{mm/s}$. Il controllo di tipo PI che agisce sui motori assicura che il MANUS si muova alla velocità fissata. Se poi, il delta di posizione che viene comunicato dall'esterno non subisce variazione, ad ogni nuovo ciclo viene rielaborata la medesima informazione ottenendo una velocità costante.

Capitolo 4

Realizzazione del sistema

4.1 Introduzione

In questo capitolo si descriveranno in maniera approfondita gli aspetti principali dell'architettura software sviluppata per realizzare il controllo del moto del braccio manipolatore.

Per prima cosa verranno descritti gli algoritmi utilizzati per la soluzione dei problemi tipici del moto dei robot manipolatori. Verranno illustrate la *cinematica diretta*, la *cinematica inversa* e la *generazione delle traiettorie* in particolare facendo riferimento agli algoritmi utilizzati nella libreria *RoBoop* su cui è stato basato il lavoro di sviluppo della tesi. La prima è utilizzata per il calcolo della posizione cartesiana in funzione della configurazione dei gradi di libertà; la seconda, invece, permette di calcolare la configurazione dei gradi di libertà in funzione della posizione cartesiana. Per la generazione delle traiettorie verranno illustrati l'interpolazione dei *via point* e il controllo che assicura un corretto posizionamento nel moto.

Nella seconda parte del capitolo, si analizzeranno meglio le scelte implementative che hanno portato a suddividere l'architettura in alcune parti o macro blocchi principali e si discuterà come questa suddivisione si rifletta nello sviluppo delle classi che costituiscono la libreria, introducendo una descrizione delle funzioni e delle strutture dati in esse contenute.

Nell'ultima sessione verranno descritte le peculiartià della libreria *RoBoop* che è stata utilizzata per il calcolo della cinematica. Si tratta di una libreria scritta da

terzi e sviluppata ad hoc per risolvere i problemi tipici da affrontare nel controllo dei robot manipolatori.

4.1.1 Obiettivi

Come discusso in precedenza si vuole ottenere un sistema software capace di controllare il moto di un manipolatore. Nel capitolo precedente abbiamo analizzato il protocollo di interazione e le tempistiche utili allo scambio dei messaggi tra l'elaboratore e il sistema di controllo del braccio. Ora, sfruttando queste conoscenze di basso livello del sistema, si procederà con la descrizione della parte software realizzata. Si è cercato, infatti, di realizzare una libreria con la quale poter controllare il moto del MANUS facendogli eseguire una traiettoria basata sull'interpolazione di alcuni punti forniti dall'esterno.

4.2 Algoritmi

In questo paragrafo verranno illustrati gli algoritmi utilizzati per il calcolo della cinematica diretta e inversa e gli algoritmi utilizzati per generare una traiettoria sfruttando la lista di punti nodali o *via point*.

4.2.1 Calcolo della cinematica

Per descrivere la configurazione di un braccio manipolatore occorre costruire in base ai parametri del *Denavit-Hartenberg* le *matrici di rotazione* R e i *vettori posizione* p come segue:

$$\mathbf{R_i} = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & 0 \\ \sin\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} \\ \sin\theta_i\sin\alpha_{i-1} & \cos\theta_i\sin\alpha_{i-1} & \cos\alpha_{i-1} \end{pmatrix}. \tag{4.1}$$

$$\mathbf{p_i} = \begin{pmatrix} a_{i-1} \\ -d_i\sin\alpha_{i-1} \\ d_i\cos\alpha_{i-1} \end{pmatrix}. \tag{4.2}$$

In questo modo si procederà descrivendo la posizione e l'orientazione dell'i-esimo link rispetto al precedente. Alla fine componendo le matrici di rotazione con i vettori posizione, si otterrà la matrice T detta *matrice omogenea* nella forma:

$$^{\mathbf{i-1}}\mathbf{T_i} = \begin{pmatrix} ^{\mathbf{i-1}}\mathbf{R_i} & ^{\mathbf{i-1}}\mathbf{p_i} \\ 0 & 1 \end{pmatrix}. \tag{4.3}$$

Nel caso in cui si tratti di un braccio manipolatore costituito da n *gradi di libertà*, per calcolare la cinematica diretta occorre costruire la matrice ^0T_n per descrivere la posizione e l'orientazione dell'ultimo link in funzione del primo in questo modo:

$$^{\mathbf{0}}\mathbf{T_n} = {}^{\mathbf{0}}\mathbf{T_1}{}^{\mathbf{1}}\mathbf{T_2}{}^{\mathbf{2}}\mathbf{T_3}\cdots{}^{\mathbf{n-1}}\mathbf{T_n} \tag{4.4}$$

da cui essendo:

$$^{i}\mathbf{R} = {}^{\mathbf{0}}\,\mathbf{R_{i-1}}\,{}^{\mathbf{i-1}}\mathbf{R_i} \tag{4.5}$$

$$^{0}\mathbf{p} = {}^{\mathbf{0}}\,\mathbf{p_{i-1}} + {}^{\mathbf{0}}\,\mathbf{R_{i-1}}\mathbf{p_i} \tag{4.6}$$

risulta:

$$^{0}\mathbf{T_i} = \begin{pmatrix} ^{0}\mathbf{R_i} & ^{0}\mathbf{p_i} \\ 0 & 1 \end{pmatrix}. \tag{4.7}$$

Quindi, alla fine, per ricorsione si ottiene una matrice di trasformazione che riporta la posizione e la rotazione del'ultimo giunto in funzione della posizione e rotazione del primo.

L'algoritmo utilizzato per il calcolo della cinematica inversa utilizza la tecnica Newton-Raphson. Gli approcci possibili sono due. Nel primo si utilizzano le seguenti equazioni:

$$^{0}\mathbf{T_n}(\mathbf{q}^*) = {}^{\mathbf{0}}\mathbf{T_n}(\mathbf{q}+\delta\mathbf{q}) \approx {}^{\mathbf{0}}\mathbf{T_n}(\mathbf{q})\delta\mathbf{T}(\delta\mathbf{q}) = \mathbf{T_{obj}} \tag{4.8}$$

$$\delta\mathbf{T}(\delta\mathbf{q}) = (^{0}\mathbf{T_n}(\mathbf{q}))^{-\mathbf{1}}\mathbf{T_{obj}} = \mathbf{I} + \mathbf{\Delta} \tag{4.9}$$

$$\mathbf{\Delta} = \begin{pmatrix} 0 & -\delta_z & \delta_y & d_x \\ \delta_z & 0 & -\delta_x & d_y \\ -\delta_y & \delta_x & 0 & d_z \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.10}$$

$${}^{n}\delta\chi = \begin{pmatrix} d_x & d_y & d_z & \delta_x & \delta_y & \delta_z \end{pmatrix}^{T} \tag{4.11}$$

$${}^{n}\delta\chi \approx {}^{n}\mathbf{J}(\mathbf{p})\delta\mathbf{q} \tag{4.12}$$

Questo primo metodo risulta sicuramente più veloce ma non si ha la certezza che esso converga. Per questo motivo si procede utilizzando lo sviluppo in serie di Taylor:

$${}^{0}\mathbf{T_n}(\mathbf{q}^*) = {}^{\mathbf{0}}\mathbf{T_n}(\mathbf{q}+\delta\mathbf{q}) \approx {}^{\mathbf{0}}\mathbf{T_n}(\mathbf{q}) + \sum_{\mathbf{i=1}}^{\mathbf{n}} \frac{\partial {}^{\mathbf{0}}\mathbf{T_n}}{\partial \mathbf{q_i}}\delta\mathbf{q_i} \tag{4.13}$$

In questo secondo caso si ha la certezza della convergenza.

4.2.2 Realizzazione del moto

Analizzando in modo dettagliato la procedura di generazione del moto del manipolatore, è utile introdurre una suddivisione mediante uno schema a blocchi rappresentato dal diagramma 4.1, nel quale sono identificabili il flusso di funzionamento e le operazioni principali che il sistema deve eseguire. Al sistema viene a priori fornita una lista di via point contenente le coordinate dei nodi della traiettoria nello spazio cartesiano e gli intervalli temporali di esecuzione del moto tra un nodo e il successivo. Osservando il diagramma, si nota come questi punti debbano in un primo tempo essere rielaborati in modo da ottenere un vettore contenente un set di punti in coordinate assolute sommando le coordinate fornite, alla posizione iniziale in cui si trova il manipolatore. Dato che le informazioni risultano in coordinate cartesiane, occorre procedere, quindi con il calcolo della cinematica inversa per ottenere la configurazione dei gradi di libertà ad esse associata. Una volta ottenute tutte le configurazioni in cinematica diretta, è utile analizzare se queste siano corrette o meno visto che il problema del calcolo di questo tipo di cinematica risulta non sempre definito: è possibile, infatti, che ad un set di coordinate cartesiane corrispondano o più

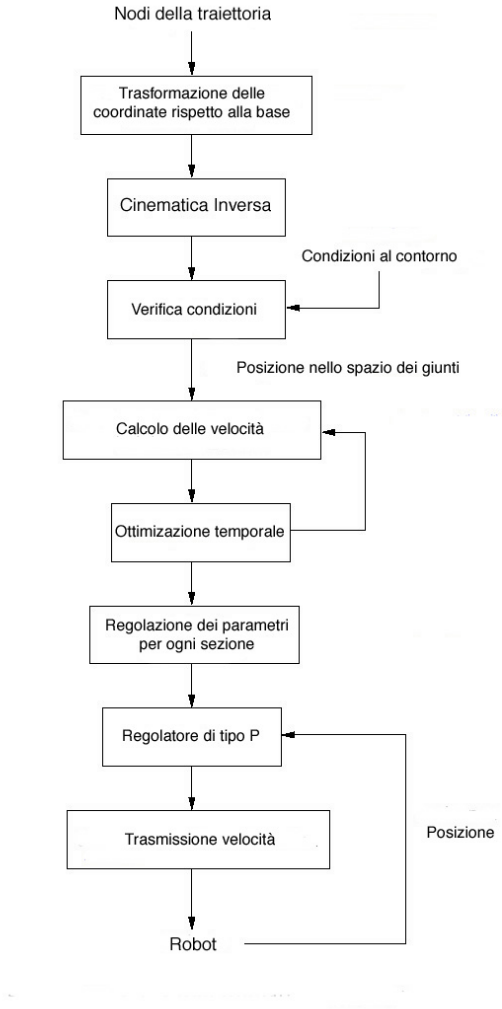

Figura 4.1: Schema a blocchi per la realizzazione delle traiettorie

set di configurazioni dei gradi di libertà, oppure nessuno se il punto risulta esterno allo spazio di lavoro del manipolatore.

Una volta terminata la prima fase, risulta necessario procedere con l'interpolazione dei via point. Dato che per ogni coppia di punti sono note sia le posizioni iniziali e finali sia le velocità calcolate in base alla continuità delle accelerazioni come verrà mostrato successivamente, si può pensare di utilizzare un polinomio di interpolazione cubico per realizzare una traiettoria adeguatamente *smooth*[15]. Il

polinomio avrà la seguente forma:

$$\theta_i(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{4.14}$$

Quello che risulta, perciò, necessario calcolare, sono i coefficienti a_0, a_1, a_2 e a_3. Partendo dalla 4.14 si può facilmente calcolarne la derivata prima e la derivata seconda:

$$\dot{\theta}_i(t) = a_1 + 2a_2 t + 3a_3 t^2 \tag{4.15}$$

$$\ddot{\theta}_i(t) = 2a_2 + 6a_3 t \tag{4.16}$$

A questo punto note le posizioni e le velocità, sostituendo nella 4.14 e 4.15, si ottiene:

$$\theta_i(t_j) = a_{0j} \tag{4.17}$$

$$\dot{\theta}_i(t_j) = a_{1j} \tag{4.18}$$

$$\theta_i(t_{j+1}) = a_{0j} + a_{1j} t_f + a_{2j} t_f^2 + a_{3j} t_f^3 \tag{4.19}$$

$$\dot{\theta}_i(t_{j+1}) = a_{1j} + 2a_{2j} t_f + 3a_{3j} t_f^2 \tag{4.20}$$

avendo considerato che $t_f = t_{j+1} - t_j$. Se ora si sostituiscono i coefficienti a_{0j} e a_{1j} nelle 4.19 e 4.20, si arriva alle forme seguenti:

$$a_{0j} = \theta_i(t_j) \tag{4.21}$$

$$a_{1j} = \dot{\theta}_i(t_j) \tag{4.22}$$

$$a_{2j} = \frac{3}{t_f^2}(\theta_i(t_{j+1}) - \theta_i(t_j)) - \frac{2}{t_f}(\dot{\theta}_i(t_j)) - \frac{1}{t_f}\dot{\theta}_i(t_{j+}) \tag{4.23}$$

$$a_{3j} = -\frac{2}{t_f^3}(\theta_i(t_{j+1}) - \theta_i(t_j)) - \frac{1}{t_f^2}((\dot{\theta}_i(t_j)) + \dot{\theta}_i(t_{j+})) \tag{4.24}$$

In questo modo, per ogni sezione vengono calcolati i quattro coefficienti a_0, a_1, a_2 e a_3 in base ai quali si procede con l'interpolazione.

L'esecuzione dell'interpolazione richiede, come visto, la conoscenza sia di posizione sia di velocità relative ad ogni via point. Per eseguire il calcolo di quest'ultima deve essere imposto, come prima condizione, che il punto iniziale e quello finale presentino ovviamente velocità nulla. In tutti gli altri punti, invece, le velocità non possono essere conosciute a priori e, di conseguenza, devono essere calcolate utilizzando l'ipotesi di continuità. Essendo noti le posizioni dei punti della traiettoria e i tempi t_j tra un nodo e il successivo, si può procedere considerando l'accelerazione e osservando che essa deve risultare continua. Siano $\theta_i(t_j)$, $\theta_i(t_{j+1})$ e $\theta_i(t_{j+2})$ tre punti consecutivi della traiettoria. Imponendo la continuità dell'accelerazione nel punto $\theta_i(t_{j+1})$ di mezzo, deve risultare dalla 4.16 che:

$$2a_{2j} + 6a_{3j}t_{j+1} = 2a_{2(j+1)} \tag{4.25}$$

Inserendo dunque i coefficienti ottenuti dalle 4.21-4.24, si ottiene: la seguente formula:

$$\frac{2\dot{\theta}_i(t_j)}{t_{j+1}-t_j} + \left(\frac{4}{t_{j+1}-t_j} + \frac{4}{t_{j+2}-t_{j+1}}\right)\dot{\theta}_i(t_{j+1}) + \frac{2\dot{\theta}_i(t_{j+2})}{t_{j+2}-t_{j+1}} = \frac{6(\theta_i(t_{j+2})-\theta_i(t_{j+1}))}{(t_{j+2}-t_{j+1})^2} + \frac{6(\theta_i(t_{j+1})-\theta_i(t_j))}{(t_{j+1}-t_j)^2} \tag{4.26}$$

in questo modo possono essere formulate $k-1$ equazioni con $\dot{\theta}_i(t_j)$, $j = 1, ..., k-1$ come incognite. Queste $k-1$ equazioni possono essere anche interpretate sotto forma di matrice:

$$\mathbf{A}\tilde{\dot{\theta}}_\mathbf{i} = \vec{b} \tag{4.27}$$

$$\mathbf{A} = \begin{pmatrix} \beta_0 & \gamma_0 & 0 & 0 & \cdots & 0 & 0 \\ \alpha_1 & \beta_1 & \gamma_1 & 0 & \cdots & 0 & 0 \\ 0 & \alpha_2 & \beta_2 & \gamma_2 & 0 & \cdots & 0 \\ \vdots & & & & \ddots & & \vdots \\ 0 & \cdots & & 0 & \alpha_{k-3} & \beta_{k-3} & \gamma_{k-3} \\ 0 & \cdots & & 0 & 0 & \alpha_{k-2} & \beta_{k-2} \end{pmatrix}. \tag{4.28}$$

$$\vec{\dot{\theta}_i} = (\dot{\theta}_i(t_1), \dot{\theta}_i(t_2, \ldots, \dot{\theta}_i(t_{k-1})^T \tag{4.29}$$

$$\vec{\dot{b}} = (\delta_0, \delta_1, \ldots, \delta_{k-2})^T \tag{4.30}$$

$$\alpha_j = \frac{2}{t_{j+1} - t_j} \tag{4.31}$$

$$\beta_j = \frac{4}{t_{j+1} - t_j} + \frac{4}{t_{j+2} - t_{j+1}} \tag{4.32}$$

$$\gamma_j = \frac{2}{t_{j+2} - t_{j+1}} \tag{4.33}$$

$$\delta_j = \frac{6(\theta_i(t_{j+2}) - \theta_i(t_{j+1})}{(t_{j+2} - t_{j+1})^2} + \frac{6(\theta_i(t_{j+1}) - \theta_i(t_j)}{(t_{j+1} - t_j)^2} \tag{4.34}$$

Inoltre δ_0 e δ_{k-2} devono essere calcolati nel modo seguente:

$$\delta_0 = \frac{6(\theta_i(t_2) - \theta_i(t_1)}{(t_2 - t_1)^2} + \frac{6(\theta_i(t_1) - \theta_i(t_0)}{(t_1 - t_0)^2} - \frac{2}{t_1 - t_0}\dot{\theta}_i(t_0) \tag{4.35}$$

$$\delta_j = \frac{6(\theta_i(t_k) - \theta_i(t_{k-1})}{(t_k - t_{k-1})^2} + \frac{6(\theta_i(t_{k-1}) - \theta_i(t_{k-2})}{(t_{k-1} - t_{k-2})^2} - \frac{2}{t_k - t_{k-1}}\dot{\theta}_i(t_{k-2}) \tag{4.36}$$

Quindi ora risulta possibile calcolare le velocità di ogni singolo giunto per ogni nodo della traiettoria nel seguente modo:

$$\vec{\dot{\theta}_i} = \mathbf{A^{-1}}\vec{b} \tag{4.37}$$

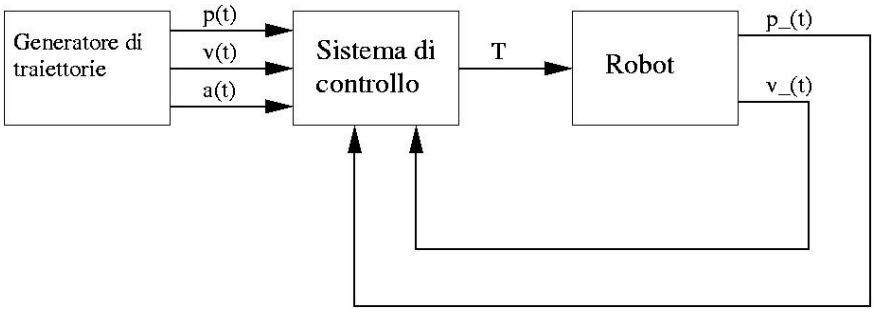

Figura 4.2: Schema sistema di retroazione per il controllo.

L'algoritmo di *Run* della traiettoria realizza un controllo di tipo P basandosi sulla retroazione di posizione. Per realizzare il moto tra due punti nodali della traiettoria, si procede prima di tutto calcolando il tempo a cui l'iterazione corrisponde. In base ad esso e al polinomio interpolatore, come mostrato nello schema 4.2 viene calcolata la posizione corretta. A questo punto, dopo aver letto l'informazione proveniente dal MANUS si calcola l'errore di posizione $E = p(t) - p\_(t)$ quindi si procede sfruttando la derivata del polinomio interpolatore per calcolare la velocità teorica in base al tempo di iterazione. Sfruttando l'errore di posizione calcolato in precedenza risulta così possibile, tramite una costante di proporzionalità, modulare la velocità teorica in modo da ottenere un dato reale con il quale eseguire il controllo del moto. Inoltre, per far sì che il controllo rimanga stabile, si valuta se l'errore E rimane contenuto in un intervallo prefissato. In caso contrario si procede allo stop immediato del manipolatore.

4.3 Realizzazione

Come si è già accennato, l'obiettivo principale di questo progetto è di ottenere un sistema di controllo con cui gestire il movimento del robot manipolatore MANUS in modo tale da creare un'interfaccia tra di esso e un elaboratore.

Analizzando nel complesso il problema, possono essere individuate alcune parti fondamentali che per la loro natura e funzionalià è ragionevole separare e circoscrivere in macro blocchi che rappresentano la parti costitutive dell'architettura.

- Basso livello: è la parte relativa allo scambio di informazione tra il sistema di controllo del manipolatore e il controller CAN-Bus. In questa sezione,

partendo dalla descrizione del protocollo introdotta nel capitolo precedente, verranno sviluppate le funzioni atte a realizzare lo scambio dei messaggi secondo la tempistica indicata, oltre che la decodifica dei dati in ingresso e la codifica dei dati in uscita.

- Alto livello: questa è la parte relativa all'elaborazione dei dati per la generazione delle traiettorie. In base agli algoritmi visti in precedenza, verranno inserite alcune funzioni per la trattazione della lista di via point inseriti, per la loro elaborazione, e quindi per la generazione vera e propria dei dati di moto e il loro invio alla parte di basso livello.

- Comunicazione: in questa parte saranno messi in luce gli aspetti coinvolti dallo scambio delle informazioni tra il basso e l'alto livello. Come verrà specificato meglio successivamente, le due parti descritte in precedenza dovranno essere eseguite in parallelo e quindi è necessario adottare un sistema che possa permettere di leggere e scrivere i dati in strutture condivise cercando di mantenerne la consistenza, evitando cioè che alcuni dati vengano sovrascritti mentre contemporaneamente vengono letti altrove.

Quello che si intende creare, quindi, è un architettura capace di gestire gli aspetti prncipali del controllo senza che l'utilizzatore si debba preoccupare di alcunchè se non dell'inserimento dei dati di moto.

4.3.1 Basso livello: hardware e comunicazione

Nella sezione di basso livello il sistema si occupa principalmente di gestire la comunicazione con il manipolatore. Nel capitolo 3 si è già descritto sia il protocollo CAN-Bus sia quello imposto dal costruttore del Manus per realizzare la comunicazione. In questa parte, quindi, viene realizzato il codice che gestisce la codifica dei messaggi CAN, il loro invio e la loro ricezione.

Come si evince dal codice 4.1, questa risulta essere la classe di più basso livello nella quale vengono implementate le funzioni principali per la comunicazione con il controller CAN. Qui, infatti, dopo l'inizializzazione per opera della funzione `CanInit`, dove vengono configurati i parametri relativi al timing e alla resistenza di terminazione, si procede con la realizzazione vera e propria del-

```
class PcCan
{
    private:
        int canDevicePortId;
        canmsg_t CanMessageTransmit,CanMessageReceive;

    public:
        bool CanInit(char *canDevice);
        bool CanTransmit();
        bool CanReceive();
        void CanCode(int messageID,int messageLength,
                                    unsigned char messagePackage[8]);
        void CanDecode(int *messageID,int *messageLength,
                                    unsigned char messagePackage[8]);
};
```

Listato 4.1: Classe per la gestione della comunicazione CAN

la comunicazione. Nella funzione `CanReceive` viene implementata la ricezione dei messaggi nel seguente modo: la funzione `CannesGetInQueueCount` realizza la temporizzazione eseguendo una lettura bloccante sul buffer di ricezione. Nel momento in cui è presente un messaggio all'interno del buffer, la funzione `CannesGetMessage` ne esegue la lettura inserendo i dati all'interno della struttura `CanMessageReceive`. A questo punto, tramite la funzione `CanDecode` si procede decodificando i dati ed estrapolando le informazioni utili. Per quanto riguarda la trasmissione, invece, dopo che i dati sono stati codificati dalla funzione `CanCode` nella struttura `CanMessageTransmit`, essa viene realizzata tramite la funzione `CanTransmit` dove `CannesSendMessage` realizza la scrittura nel buffer di trasmissione del controller.

La classe `RobotCanCom` rappresenta, invece, la classe principale della parte di basso livello accessibile dall'esterno. Ereditando `PcCan`, essa acquista tutti gli strumenti necessari per la trattazione della comunicazione con il controller CAN. Il codice 4.2 evidenzia che per prima cosa va effettuata l'inizializzazione del sistema che si basa sia sull'impostazione dei parametri relativi al CAN-Bus tramite la funzione contenuta nella classe precedente, sia sulla lettura dei dati relativi al DH del manipolatore. In questa classe la funzione principale risulta essere `RobotComLoop`

```
class  RobotCanCom: public RobotCom, protected PcCan
  {
  private:
        int  messageID;
        int  messageLength;
        unsigned char messageData[8];

  public:
        void  InitRobotCom();
        void  ReceivePosition ();
        void  TransmitSpeed();
        void  RobotComLoop();
  };
```

Listato 4.2: Classe per la gestione dell'informazione proveniente dal Manus

che deve essere utilizzata nel processo di comunicazione. Come prima cosa essa si occupa di leggere i dati di moto provenienti dalla parte di alto livello inserendoli in una struttura contenuta nella classe `RobotCom` che viene ereditata pubblicamente. Nella funzione `RobotComLoop`, inoltre, è implementato il ciclo di controllo per la ricezione e lo scambio dei messaggi secondo il protocollo già discusso. Sfruttando un ciclo di tre letture successive, essa elabora la ricezione dei tre messaggi provenienti dal manipolatore nei quali risulta codificata la totalità dell'informazione. Nel momento in cui l'ID di uno di questi corrisponde al valore `0x37f` la funzione attua la trasmissione mediante `TransmitSpeed` che si occupa di inviare i dati di moto letti in precedenza. La ricezione, invece, è realizzata tramite la funzioni `ReceivePosition`. Essa, dopo che è avvenuta la lettura del singolo messaggio CAN e la successiva decodifica, in base all'ID del messaggio identifica il significato dell'informazione e lo inserisce all'interno di una struttura contenuta nella classe `RobotCom`. Una volta che il ciclo principale risulta terminato, la funzione `RobotComLoop` si occupa di scrivere i dati di posizione pervenuti dal manipolatore nella struttura condivisa con la parte di alto livello. Come si vede dal codice sono presenti anche alcune variabili di appoggio (`messageID`, `messageLength` e `messageData`): queste risultano utilizzate ogni volta che si procede alla codifica o decodifica di un messaggio CAN per comunicare con la classe `PcCan`.

4.3.2 Alto livello: cinematiche e generatore di traiettorie

Completato il progetto della parte di basso livello, è necessario sviluppare la parte più significativa del sistema e cioè quella relativa all'elaborazione delle traiettorie in base ai dati immessi nel sistema. Questo discorso risulta molto ampio e, per questo, va diviso in alcuni ambiti ben distinti.

Prima di tutto è necessario chiarire quali siano le informazioni che devono essere immesse nel sistema perché questo generi una traiettoria e, soprattutto, come queste debbano essere elaborate per generare dei dati utili per la parte di comunicazione. Avendo già analizzato all'inizio del capitolo gli algoritmi, qui, si vedrà come venga memorizzata l'informazione e come gli algoritmi siano stati implementati.

Dovendo fornire al sistema un'informazione utile alla generazione di una traiettoria, si provvederà a fornire una lista di punti o via point con i quali definire la sequenza di coordinate. Queste possono essere espresse sia in forma angolare, indicando già le configurazioni dei gradi di libertà del robot e non richiedendo nessun tipo di trasformazione, sia in forma cartesiana. In questo caso va definito se queste coordinate debbano rappresentare uno spostamento del gripper del manipolatore rispetto alla posizione iniziale del moto o rispetto all'ultima posizione assunta, oppure se esse siano assolute nel sistema di riferimento della base del manipolatore. Risulta, inoltre, utile definire nel caso del moto cartesiano anche un dato con il quale specificare l'orientazione finale che il gripper deve assumere, per esempio, nel caso in cui si debba afferrare un oggetto in una particolare posizione. In questo caso, si definiscono anche tre valori angolari per i gradi di libertà *yaw*, *pitch* e *roll*.

Position	**Position velocity**	**Orientation**	**Orientation velocity**	**Time**	**Type**
(x, y, z)	(V_x, V_y, V_z)	$(yaw, pitch, roll)$	$(V_{yaw}, V_{pitch}, V_{roll})$	t	C_{type}

Tabella 4.1: Formato della lista di via point.

La tabella 4.1 riassume la forma che la lista di punti deve assumere. Nella lista sono stati anche inseriti due campi contenenti le velocità nei quali a priori non è contenuto nessun dato significativo, visto che le velocità sono calcolate in un secondo tempo.

Una volta definita la forma della lista, è stata dichiarata la classe `pointListJoint` nella quale prima vengono salvati i dati relativi ai via point nella forma di un

```
class pointListJoint : public vector<elementType>
  {
  public:
      int LoadFromFile( const char *fileName );
      int SaveToFile ( const char *fileName );
      void InsertPos (elementType element , int after );
      void DeletePos( int atPosition );
  };
```

Listato 4.3: Classe per la gestione dei dati in ingresso.

vettore `vector` di tipo `elementType` (vedi tabella 4.1). Inoltre, come illustrato dal codice 4.3, in questa classe vengono dichiarate anche le funzioni utili per operare su questo vettore. Le funzioni `LoadFromFile` e `SaveToFile` permettono di leggere e scrivere le informazioni in un file di testo, mentre le funzioni `InsertPos` e `DeletePos` permettono di inserire o cancellare una posizione dal vettore direttamente quando esso risulta già creato.

Una volta definita la forma dell'informazione che deve essere inserita nel sistema, è necessario capire come questo tipo di informazione debba essere elaborato per ottenere dei dati di moto da passare alla parte di basso livello per la comunicazione al manipolatore. Avendo già descritto l'algoritmo utilizzato, ora si limiterà l'analisi a come esso è stato implementato.

La classe `JointTrajectoryVR`, come evidenziato dal codice 4.4, si occupa della parte di moto effettiva oltre che dell'elaborazione dei dati inseriti. Essa eredita immediatamente la classe `pointListJoint` dove sono contenuti i via point e le funzioni per la manipolazione di questi dati. Una volta compilato il vettore contenente i via point, deve essere chiamata la funzione `StartTrajectory` all'interno del processo principale che si occupa della generazione delle traiettorie. Come prima cosa, essa procede chiamando a sua volta la funzione `CalculateVelocities` che in un primo tempo si occupa di calcolare in base al tipo di informazione (se assoluta o relativa alla posizione attuale del gripper) la sequenza di posizioni cartesiane assolute. Quindi, partendo da queste posizioni viene effettuata la cinematica inversa per ottenere le configurazioni nello spazio dei giunti. Infine si procede al calcolo delle velocità per ogni giunto dato che, come detto in precedenza, risulta necessaria

```
class JointTrajectoryVR : public pointListJoint
  {
  public:
    int StartTrajectory ();
    int RunTrajectoryPartial ( startPoint , endPoint );
    int CalculateVelocities ( void );
   void ReceivePosition (RobotPosition *robotPos , RobotStatus * robotStat );
   void TransmitSpeed(RobotSpeed *robotSpeed,RobotStatus * robotStat );

    private:
       Matrix * startPos , * endPos,
                    * startVel , * endVel;
       RobotSpeed *nextSpeed;
       RobotStatus * actualStatus ;
       RobotPosition * actualPosition ;
  };
```

Listato 4.4: Classe gestione dell'elaborazione dei dati.

anche questa informazione nel momento in cui si procederà all'effettiva generazione del moto. A questo punto, viene chiamata la funzione `RunTrajectoryPartial` che si occupa appunto di generare in base al controllo di tipo P di cui si è già parlato, dei dati di velocità che al termine di ogni iterazione vengono scritti all'interno di una struttura condivisa per la successiva trasmissione al sistema di controllo del manipolatore.

4.3.3 Thread e comunicazione tra basso e alto livello

Terminata la descrizione di come sono state sviluppate la parte di alto e quello di basso livello, occorre ora vedere come esse dovranno essere eseguite e, in particolare, come dovranno interagire tra loro.

Entrambe, per il loro funzionamento, necessitano di autonomia, infatti ognuna svolge operazioni assai differenti rispetto all'altra. La parte di alto livello risulta abbastanza pesante dal punto di vista computazionale a causa dei numerosi calcoli che devono essere eseguiti per la cinematica inversa partendo delle coordinate cartesiane. Questi calcoli, però non pongono particolari vincoli temporali di esecu-

zione. La parte relativa al basso livello, invece, deve eseguire solo l'algoritmo di comunicazione che implica l'invio e la ricezione dei messaggi CAN e la codifica e decodifica dei dati. In questo caso le operazioni da svolgere non comportano una grande spesa a livello computazionale ma comportano il rispetto di una tempistica molto critica soprattutto a livello di trasmissione dei messaggi. Per questo motivo si è scelto di eseguire ciascuna delle due parti all'interno di una *thread* indipendente. Questo approccio ha portato i vantaggi sperati, ma ha anche aperto un problema relativo all'accesso ai dati. La parte di basso livello, infatti, attua una comunicazione dei dati che vengono elaborati da quella di alto livello. Dato che le esecuzioni di entrambe risultano indipendenti, nasce il problema di garantire che i dati scambiati siano consistenti e che non si sovrappongano letture e scritture da parti differenti. Per garantire consistenza si è deciso di serializzare gli accessi ai dati creando delle strutture intermedie in cui salvare temporaneamente questi dati in attesa che vengano letti ed elaborati.

Nel capitolo 3 si è evidenziato come l'informazione proveniente dal sistema di controllo del manipolatore risulta sostanzialmente differente da quella che, invece, gli viene comunicata. I dati provenienti dal Manus forniscono la posizione attuale del manipolatore nel preciso istante in cui vengono inviati, mentre quelli diretti al Manus specificano lo stato e le velocità desiderate. Per questo motivo risulta necessario creare tre classi nelle quali inserire questi dati.

- La classe `RobotPosition` contiene funzioni e strutture per i dati che provengono dal manipolatore relativi alla posizione attuale. Nel caso in cui esso operi nello spazio cartesiano, le posizioni lungo gli assi x, y, z verranno salvate nella struttura di tipo `RobotCartesianPosition`, nel caso in cui, invece, il manipolatore operi nello spazio dei giunti, i valori dei gradi di libertà verranno inseriti all'interno della struttura di tipo `RobotJointPosition`. In questa classe, inoltre, sono presenti anche le funzioni per l'elaborazione di questi dati. `GetCartesianCoordinateSystem` e `GetInverseKinematics` si occupano, appunto, del calcolo della cinematica rispettivamente diretta e inversa utilizzando gli algoritmi già visti.

- La classe `RobotSpeed` contiene le strutture per gestire i dati che devono essere inviati al manipolatore provenienti dall'elaborazione della traiettoria.

```
class RobotPosition
  {
  private:
      struct LocalPosition {
          struct RobotCartesianPosition  cartesianPosition ;
          struct RobotJointPosition  jointPosition ;
      } robotPosition ;

  public:
      void GetPosition ( robotPosition );
      void SetPosition ( robotPosition );
      Matrix GetCartesianCoordinateSystem(int k,
                                          enum RobotModi robotModus);
      void GetJointFromCart(int k,enum RobotModi robotModus);
  };
```

Listato 4.5: Classe per la gestione della posizione.

```
class RobotSpeed
  {
  private:
      struct LocalSpeed{
          struct RobotCartesianSpeed cartesianSpeed
          struct RobotJointSpeed jointSpeed
      } robotPosition ;

  public:
      void SetSpeed( robotPosition );
      void GetSpeed( robotPosition );
  };
```

Listato 4.6: Classe per la gestione delle velocità.

Anche in questo caso sono presenti due strutture di tipo `RobotCartesianSpeed` e `RobotJointSpeed` per distinguere i dati nel caso che si operi nello stato cartesiano piuttosto che in quello dei giunti.

- La terza classe `RobotStatus` viene utilizzata per la gestione dello stato del manipolatore. In questa classe vengono salvati i dati relativi al funzionamento

```
class  RobotStatus
  {
  public:
        char  robotStatus ;
        enum RobotError robotMessage;
        enum RobotModi robotWantedMode, robotActualMode;

        char GetRobotStatus ();
        enum RobotModi GetRobotActualMode();
        enum RobotModi GetRobotWantedMode();
        enum RobotError GetRobotMessage();

        void  SetRobotStatus (char  status );
        void  SetRobotActualMode(enum RobotModi mode);
        void  SetRobotWantedMode(enum RobotModi mode);
        void  SetRobotMessage(enum RobotError message);
  };
```

Listato 4.7: Classe per la gestione dello stato del manipolatore.

(cartesiano, spazio dei giunti, fold-in, fold-out) e gli eventuali errori che il Manus segnala, come, per esempio gradi di libertà bloccati o posizione errata del gripper (se esterno all'area di lavoro).

Osservando il codice, si nota che in tutte e tra le classi sono presenti una funzione `Get` e una funzione `Set`. Per garantirne, infatti, la consistenza dei dati contenuti nelle strutture e per evitare che alcuni di questi vengano letti per esempio dalla parte di basso livello mentre vengono sovrascritti da quella di alto livello, si è proceduto inserendo in ogni classe due funzioni per accedere ad essi in modo protetto tramite una mutua esclusione.

Si deve, infine, osservare che per rendere la trattazione di questi dati più flessibile e di facile identificazione, si è proceduto raccogliendo le tre classi precedenti in un'unica classe che gestisce ogni forma di operazione di I/O su ogni tipo di dato in modo globale: la classe in questione è `RobotIO` che eredita pubblicamente `RobotPosition`, `RobotSpeed` e `RobotStatus`.

4.4 Libreria RoBoop

All'inizio di questo capitolo sono stati esaminati gli algoritmi utilizzati per il calcolo della cinematica diretta e inversa. Essendo l'approccio al problema non immediato e, soprattutto, essendo disponibili numerose librerie in cui sono implementati questi strumenti, si è proceduto analizzandone alcune per identificare una libreria da adattare nella realizzazione del sistema di controllo. Tra le varie librerie disponibili, si è scelto di utilizzare *ROBOOP*[16] sviluppata da Richard Gourdeau, che rappresenta, appunto, una collezioni di funzioni scritte in linguaggio C++ da utilizzare per sintesi e simulazione di modelli matematici per robot manipolatori. Essa è liberamente scaricabile dal sito `http://www.cours.polymtl.ca/roboop/` in cui è possibile reperire oltre ai sorgenti, anche svariata documentazione con cui si può approfondire il funzionamento e gli algoritmi utilizzati.

La libreria è molto ampia e permette di risolvere numerosi problemi matematici oltre che simulare il controllo del moto di un manipolatore. Per quello che riguarda il sistema di controllo del manipolatore che è stato progettato in questo lavoro di tesi, è stato sufficiente prendere in considerazione una minima parte delle funzionalità della libreria, limitando il campo di utilizzo al calcolo della cinematica diretta e inversa secondo gli algoritmi introdotti all'inizio di questo capitolo.

La libreria *Roboop* fa uso a sua volta di un'ulteriore libreria denominata *NEWMAT* per il calcolo delle matrici e per la definizione dei tipi. Essa, infatti, basa il calcolo sull'utilizzo di variabili di tipo `Matrix` e vettori di tipo `ColumnVector`. Il primo rappresenta un tipo utilizzato per creare un oggetto di tipo matrice, con il vantaggio di contenere, oltre alla struttura dati di tipo `array`, anche tutti i metodi utili per eseguire i calcoli, come per esempio la moltiplicazione riga per colonna o la trasposizione. Il secondo tipo, invece, risulta essere utilizzato per specificare un vettore e può essere considerato come direttamente ottenuto dal tipo precedente specificando una matrice 3×1.

La libreria Roboop utilizza come classe base la classe `Robot` nella quale sono contenuti tutti i parametri di configurazione per il modello del robot manipolatore, e cioè:

- il numero dei gradi di libertà *n* (`int dof`);

- il valore dell'accelerazione di gravità in formato vettoriale
- la matrice *n*-dimensionale di oggetti di tipo `Link`

Cinematica	**Inerzia**	**Motori**
`int joint_type`	`Real m`	`Real Im`
`Real theta, d, a, alpha`	`ColumnVector r`	`Real Gr`
`Matrix R`	`Matrix I`	`Real B`
`ColumnVector p`		`Real Cf`
`Bool DH`		

Tabella 4.2: Dati caratterizzanti ogni singolo *link*.

Nella tabella 4.5 vengono indicati tutti i parametri contenuti nella classe `Link`. Essa, infatti, raccoglie sia i dati sia le funzioni necessarie a caratterizzare ogni singolo *link* del manipolatore inteso nella accezione definita da Denavit e Hartenberg[17].

Questa classe deve essere inizializzata fornendo le informazioni iniziali riguardanti il tipo di giunto (`int joint_type`: rotazionale = 0, prismatico = 1) oltre che i parametri *theta, d, a, alpha* (`Real theta, d, a, alpha`) e un valore booleano `Bool DH` (`true` = standard, `false` = modificato). Inoltre in questa classe sono contenuti i parametri inerziali come *m* massa, *r* vettore posizione dei centri di massa e il tensore con i valori dei momenti di inerzia. Si veda l'appendice B per i parametri relativi al modello DH.

Nella classe principale `Robot` sono state inserite, inoltre, tutte le funzioni necessarie alla simulazione del modello del robot manipolatore.

Funzione	**Descrizione**
`get_q`	legge il valore dei giunti
`set_q`	scrive il valore dei giunti

Tabella 4.3: Funzioni relative alle configurazioni dei giunti.

Nelle tabelle 4.3 e 4.4 vengono riassunte le principali funzioni che sono state utilizzate all'interno del progetto dell'architettura per il controllo del manipolatore MANUS. Le funzioni `set_q` e `get_q` procedono con la lettura e scrittura dei valori dei giunti: la libreria, infatti, nel momento in cui viene effettuato qualsiasi tipo di calcolo, deve conoscere la configurazione attuale dei gradi di libertà del manipolatore. Le funzioni `kine` e `inv_kine` si occupano di calcolare la cinematica

Funzione	Descrizione
`inv_kin`	calcolo della cinematica inversa
`jacobian`	calcolo dello Jacobiano del robot
`kine`	calcolo della cinematica diretta
`dTdqi`	derivate parziali della cinematica diretta

Tabella 4.4: Funzioni relative alla cinematica.

diretta e inversa implementando gli algoritmi visti. Inoltre le funzioni `jacobian` e `dTdqi` si occupano di calcolare rispettivamente lo jacobiano del manipolatore (relazione tra velocità nello spazio dei giunti e velocità nello spazio cartesiano) e le derivate parziali della matrice di omogenea.

Per ogni giunto viene creata una matrice R di tipo `Matrix` e un vettore p di tipo `ColumnVector`: la prima rappresenta la matrice di rotazione nella quale sono contenuti i valori rotazionali dei giunti, mentre il secondo rappresenta il vettore di posizione.

Per utilizzare la libreria all'interno del sistema di controllo del manipolatore, risulta necessario per prima cosa istanziare nel codice un oggetto di tipo `Robot`. Quindi, per eseguire l'inizializzazione di questa classe, devono essere forniti al sistema i parametri che saranno poi inseriti nelle strutture di controllo. È essenziale notare che la libreria può essere inizializzata in modi diversi a seconda del dettaglio dei calcoli di cui si necessita. Fornendo, infatti, i dati da inserire nella prima matrice $n_{dof} \times 5$ si procede inizializzando la libreria per eseguire soltanto calcoli di cinematica. Se, invece, si procede con una matrice $n_{dof} \times 15$, si forniscono al sistema i dati necessari per il calcolo oltre che della cinematica, anche dell'inerzia. Inoltre se si fornisce una matrice $n_{dof} \times 19$ si aggiungono anche i dati relativi alla velocità e alla coppia dei motori. Si comprende, quindi, come si ottenga via via un risultato più dettagliato. Nella tabella 4.5 vengono riassunti i vari significati delle posizioni nelle matrici precedenti.

Colonna	**Variabile**	**Descrizione**
1	σ	Tipo di giunto (rotazionale = 0, prismatico = 1)
2	θ	Parametro del Denavit-Hartenberg
3	d	Parametro del Denavit-Hartenberg
4	a	Parametro del Denavit-Hartenberg
5	α	Parametro del Denavit-Hartenberg
6	m	Massa del giunto
7	c_x	Centro di massa lungo asse x
8	c_y	Centro di massa lungo asse y
9	c_z	Centro di massa lungo asse z
10	I_{xx}	Elemento xx del tensore di inerzia
11	I_{xy}	Elemento xy del tensore di inerzia
12	I_{xz}	Elemento xz del tensore di inerzia
13	I_{yy}	Elemento yy del tensore di inerzia
14	I_{yz}	Elemento yz del tensore di inerzia
15	I_{zz}	Elemento zz del tensore di inerzia
16	I_m	Inerzia del motori
17	Gr	Rapporto di rotazione dei motori
18	B	Coefficiente di viscosità della frizione dei motori
19	C_f	Coefficiente di Coulomb della frizione dei motori

Tabella 4.5: Variabili da inizializzare.

Capitolo 5

Risultati sperimentali

Nei capitoli precedenti sono stati affrontati tutti gli aspetti che costituiscono il sistema di controllo progettato per il manipolatore MANUS. Ora si analizzeranno in dettaglio i risultati ottenuti con l'utilizzo di questo sistema di controllo. Per prima cosa si vedrà il comportamento della parte di basso livello analizzando i messaggi che effettivamente transitano sul CAN-Bus e le relative temporizzazioni. Successivamente verranno analizzate le prestazioni della parte relativa alla generazione delle traiettorie.

5.1 Comunicazione: obiettivi

Partendo dalle considerazioni presentate nel capitolo 3, è stata sviluppata la parte relativa alla comunicazione per gestire lo scambio dei messaggi tra l'elaboratore e il sistema di controllo del MANUS attraverso una rete CAN-Bus. In questo caso è necessario valutare la sequenza dei messaggi scambiati per verificare sia l'effettivo invio della risposta da parte del sistema di controllo, sia la corretta tempistica in modo da soddisfare il protocollo imposto dal costruttore del manipolatore. Per questo motivo, avendo a disposizione, oltre al controller PCI utilizzato per la comunicazione principale anche la scheda ISA 16 Bit contenuta nel Transparent Mode (fornito come accessorio del MANUS) è stato creato un dispositivo di log per i messaggi CAN.

Dopo aver installato la scheda ISA all'interno di un secondo elaboratore, si è

proceduto scrivendo una utility con la quale registrare qualsiasi messaggio transitante sul bus. Va ricordato, infatti, che una peculiarità di questo tipo di bus sta nel fatto che ogni messaggio che transita su di esso viene ricevuto da tutti i dispositivi connessi alla rete; quindi per regolarne la ricezione vengono impostate alcune maschere che filtrano i messaggi in base all'ID. Lo schema del sistema logger è mostrato in figura 5.1

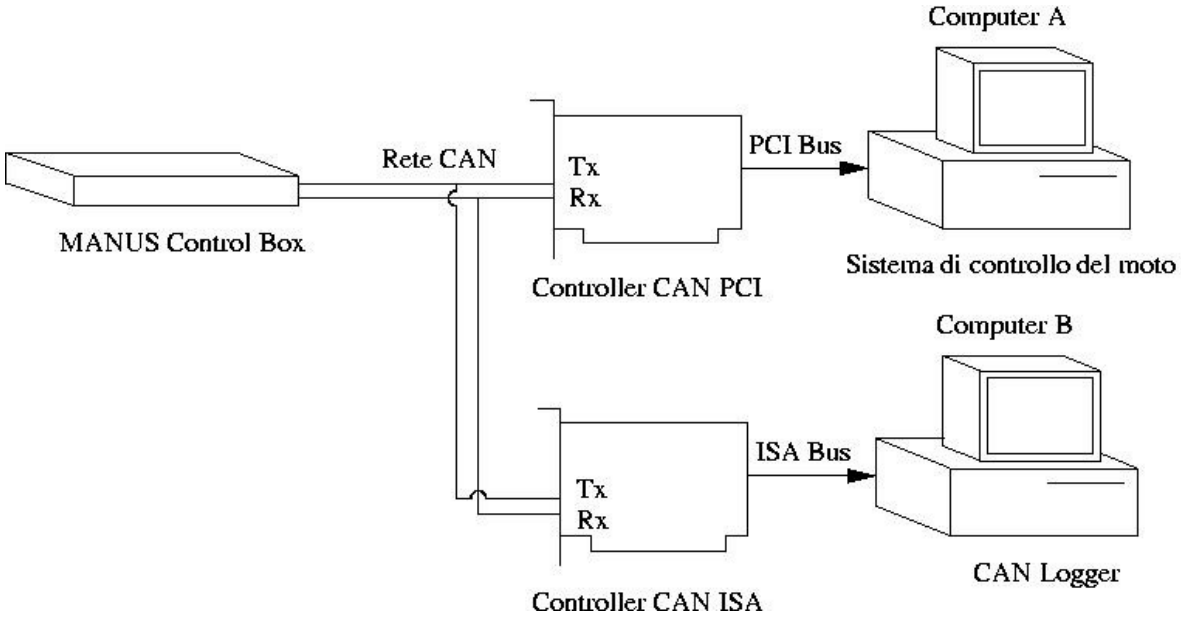

Figura 5.1: Schema del logger per i messaggi CAN

Osservando lo schema in figura 5.1, si nota che si è scelto di realizzare il sistema di log installando il controller ISA su un elaboratore a parte. Questa scelta è nata dal fatto che, per la tempistica critica che deve essere mantenuta nello scambio dei messaggi, un ulteriore processo sul medesimo elaboratore poteva falsare sia l'invio dei messaggi al manipolatore sia la ricezione da parte dell'utility di log. Il computer A quindi si occupa dell'esecuzione del sistema di controllo per attuare il moto del manipolatore, mentre il computer B, in modo totalmente trasparente alla comunicazione, si pone "in ascolto" sul bus. Nel momento in cui viene registrato il passaggio di un messaggio (dal manipolatore al sistema di controllo sul computer A, oppure da quest'ultimo verso il MANUS), il computer B registrare l'id, le informazioni contenute, e l'istante temporale in cui è avvenuta la comunicazione. In questo mo-

do, in base all'id risulta facile risalire al mittente e al destinatario del messaggio, e in base al tempo si può risalire ad eventuali errori di trasmissione.

5.1.1 Risultati

Di seguito viene riportata una parte dei risultati prodotti dal logger.

```
t=14.265s:
Extended  ID: 350  Data: 00 03 fd 80 01 bb fe ac
t=14.285s:
Extended  ID: 360  Data: 05 eb 01 bb 05 23 d8 f0
t=14.305s:
Extended  ID: 37f  Data: 00 ff 00 00 05 23 d8 f0
t=14.325s:
Extended  ID: 350  Data: 00 03 fd 80 01 bb fe ac
t=14.345s:
Extended  ID: 360  Data: 05 eb 01 bb 05 23 d8 f0
t=14.365s:
Extended  ID: 37f  Data: 00 ff 00 00 05 23 d8 f0
t=14.367s:
Extended  ID: 374  Data: 00 00 00 00 00 00 00 00
t=14.386s:
Extended  ID: 350  Data: 00 03 fd 80 01 bb fe ac
t=14.405s:
Extended  ID: 360  Data: 05 eb 01 bb 05 23 d8 f0
t=14.425s:
Extended  ID: 37f  Data: 00 ff 00 41 05 23 d8 f0
t=14.427s:
Extended  ID: 374  Data: 00 00 00 00 00 00 00 00
t=14.446s:
Extended  ID: 350  Data: 00 03 fd 80 01 bb fe ac
t=14.465s:
Extended  ID: 360  Data: 05 eb 01 bb 05 23 d8 f0
t=14.485s:
```

```
Extended  ID: 37f  Data: 00 ff 00 41 fe 4c d8 f0
t=14.487s:
Extended  ID: 374  Data: 00 00 00 00 00 00 00 00
```

Nell'esempio di log precedente, si osserva che in un primo tempo, fino a $t = 14.325\ \mathrm{s}$, si ha un ciclo di messaggi inviati dal manipolatore senza che il sistema di controllo invii alcuna risposta. In un primo tempo, infatti, il sistema di controllo si occupa di leggere il file di dati e quindi di elaborare questi dati per generare una traiettoria. Di conseguenza la thread di alto livello è nello stato di run mentre quella di basso livello risulta essere nello stato di stop. Si nota anche che ogni messaggio spedito dal manipolatore segue la tempistica specificata dal costruttore con cadenza di un invio ogni 20 ms . All'istante $t = 14.367\ \mathrm{s}$ viene spedito il primo messaggio proveniente dal sistema di controllo, esattamente 2 ms dopo l'arrivo del messaggio con ID 37f. Questo risulta sensato dato che si era già visto come l'invio non fosse critico, se, però, contenuto nei successivi 20 ms e non oltre. In caso contrario il messaggio verrebbe ignorato. In generale, si nota che la comunicazione, verificata poi anche a livello visivo con l'attuazione del moto, avviene senza problemi.

5.2 Generatore di traiettorie

In questa sezione verranno messi in luce i risultati ottenuti nella realizzazione del moto tramite il generatore di traiettorie. Come già discusso nel capitolo 4, è stato realizzato un algoritmo che, in base ad una lista di via point, calcola istante per istante le velocità da inviare al sistema di controllo del manipolatore. Quindi, in base alla retroazione di posizione proveniente dal sistema di controllo del manipolatore e in base al polinomio interpolatore, viene calcolata la posizione teorica e l'errore rispetto a quella attuale. In base a questo si realizza un controllo di tipo P che modula le velocità teoriche che vengono inviata al Manus. L'obiettivo che si vorrebbe raggiungere in questa fase di testing è far compiere al manipolatore una traiettoria con la quale verificare sia il corretto funzionamento della cinematica inversa, fornendo quindi una lista di punti in coordinate cartesiane, sia la corretta interpolazione degli stessi punti dalla quale, come già evidenziato, dovrebbe risultare una traiettoria smussata e non rettilinea.

5.2.1 Risultati per la generazione del moto nello spazio cartesiano

Per prima cosa si è proceduto fornendo al sistema un solo via point in coordinate cartesiane per verificare solamente l'aspetto di calcolo della cinematica e per ottenre alcuni risultati utili per stimare la precisione nei movimenti. In questo caso, si vede chiaramente che il gripper del manipolatore esegue un moto rettilineo posizionandosi nella posizione finale richiesta con buona precisione. Dopo svariati test eseguiti a diverse velocità si è potuto notare che lo scarto tra la posizione finale inserita e quella effettivamente raggiunta per la maggior parte dei casi non supera i 5 mm . Solo in alcuni casi si sono verificati scarti sensibilmente superiori anche di 15 mm su esecuzioni di segmenti di traiettorie di 20-30 cm anche se tali casi risultano sporadici.

Per verificare, poi, l'effettivo funzionamento dell'aspetto di interpolazione, si è proceduto fornendo al sistema una lista di punti estrapolati da una circonferenza cercando di far eseguire al manipolatore una traiettoria circolare.

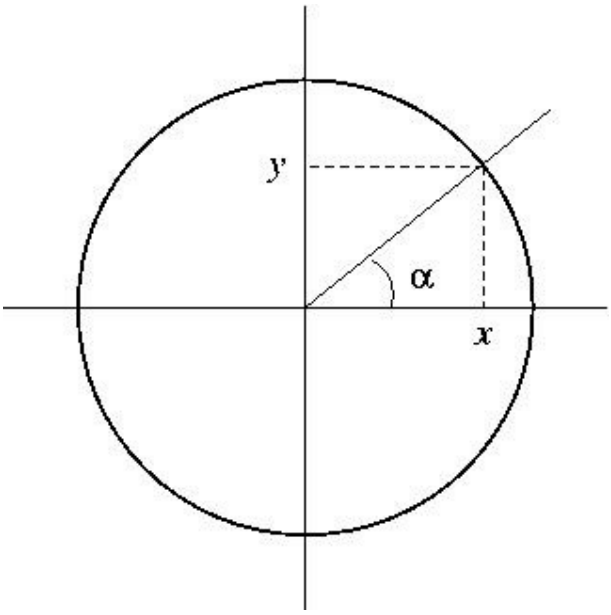

Figura 5.2: Definizione coordinate cartesiane per traiettorie.

Come mostrato in figura 5.2, la circonferenza viene suddivisa scegliendo un angolo α di una certa ampiezza in modo tale da rendere la descrizione puntuale più

o meno dettagliata aumentando o diminuendo il numero di punti secondo la formula $n = \frac{360°}{\alpha}$ con n numero di punti. Quindi si calcolano le coordinate con le seguenti formule: $x = \cos i\alpha$ e $y = \sin i\alpha$ con $i = 1 \dots n$. Di seguito viene inserita parte della lista dei punti.

```
 0.0    0.0    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  0 1
 0.0  300.0    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 10 1
 0.0  295.4   52.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 13 1
 0.0  281.9  102.6 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 15 1
 0.0  259.8  150.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 17 1
 0.0  229.8  192.8 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 19 1
 0.0  192.8  229.8 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 21 1
 0.0  150.0  259.8 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 23 1
 0.0  102.6  281.9 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 25 1
 0.0   52.1  295.4 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 27 1
 0.0    0.0  300.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 29 1
 0.0  -52.1  295.4 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 31 1
 0.0 -102.6  281.9 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 33 1
 0.0 -150.0  259.8 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 35 1
 0.0 -192.8  229.8 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 37 1
 0.0 -229.8  192.8 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 39 1
 0.0 -259.8  150.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 41 1
 0.0 -281.9  102.6 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 43 1
 0.0 -295.4   52.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 45 1
 0.0 -300.0    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 47 1
```

Per prima cosa va sottolineato che la circonferenza in questione si sviluppa sul piano yz e ha raggio pari a $30\ \mathrm{cm}$, come si nota dal 300 presente nella prima riga. In questo primo esempio si è scelto di utilizzare $\alpha = 9\ °$ in modo tale da ottenere esattamente 40 via point. Come si nota dall'esempio (riportante solo i punti per $180\ °$), tutte le velocità sono state impostate ad un valore fittizio pari a 0 dato che verranno calcolate successivamente in base al parametro temporale inserito nella penultima colonna. Come si vede, infatti, essa contiene i valori degli intervalli temporali entro i quali devono essere eseguiti i segmenti di traiettoria tra un via point e il successivo.

L'ultima colonna fissa il tipo di coordinata settando il sistema in modo da calcolare le coordinate in funzione del punto in cui esso si trova all'inizio del moto. Come

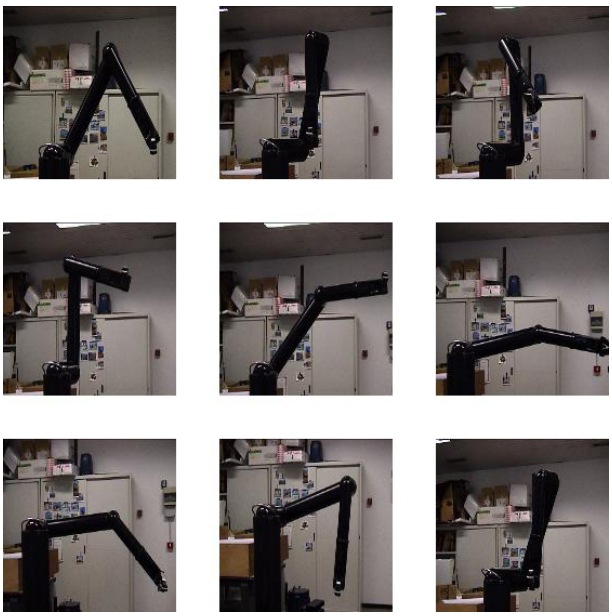

Figura 5.3: Sequenza.

illustrato in figura 5.3, dopo aver fatto eseguire questa serie di punti al generatore, si osserva che il manipolatore descrive una circonferenza di raggio $30\ \mathrm{cm}$ come richiesto. L'osservazione che può essere effettuata, è che il moto risulta piuttosto scattoso e poco fluido. In questo caso, dato che il numero di via point risulta piuttosto elevato, non si ritiene che sia significativo il fatto che il moto descritto risulti esattamente circolare: il numero di punti, infatti, è troppo elevato per poter verificare il fattore di smooth del generatore.

Ora si esegue lo stesso esperimento, ma stavolta riducendo drasticamente il numero dei via point proprio per verificare se la circonferenza descritta risulta ancora circolare o se, invece, il moto tende ad assumere qualche altra forma, magari più vicina ad un andamento rettilineo. In questo caso si sceglie $\alpha = 45\ ^\circ$ in modo da ottenere in totale 8 via point come mostrato nella lista seguente.

```
0.0    0.0    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  0 1
0.0  300.0    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 10 1
0.0  212.1  212.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 19 1
```

```
0.0    0.0  300.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 28 1
0.0 -212.1  212.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 37 1
0.0 -300.0    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 46 1
0.0 -212.1 -212.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 55 1
0.0    0.0 -300.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 64 1
0.0  212.1 -212.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 73 1
0.0  300.0    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 82 1
```

In questo caso, i tempi di percorrenza vengono mantenuti ai valori precedenti in modo da forzare il sistema di controllo alla generazione di una traiettoria con la stessa velocità. In questo caso si nota come il sistema funzioni nettamente meglio mantenendo un andamento molto più costante e, comunque, descrivendo ugualmente un moto circolare.

Capitolo 6

Conclusione

6.1 Conclusione

In questo lavoro di tesi è stato sviluppato il sistema di controllo per un robot manipolatore con l'obiettivo di creare un'interfaccia hardware e software che permetta di gestirne il moto in modo trasparente ed efficace. Il progetto è stato sviluppato cercando di tenere suddivise le parti funzionalmente concordi, creando alcuni macro blocchi nei quali vengono raccolti gli aspetti comuni del problema.

La prima fase del lavoro ha comportato lo studio del protocollo *CAN-Bus* e delle specifiche di comunicazione del manipolatore *MANUS*. Il robot è, infatti, concepito sia per essere direttamente utilizzato da un utente attraverso un'interfaccia manuale, sia per poter essere controllato dall'esterno attraverso un elaboratore connesso alla rete CAN. Per lo sviluppo di questa interfaccia si è proceduto nella realizzazione di una parte software di basso livello per la gestione dell'invio e della ricezione dei messaggi in modo da rispettare tutti i vincoli temporali richiesti dal costruttore. Tramite una utility di log della rete CAN si è potuto subito verificare il suo corretto funzionamento.

Il lavoro di maggior interesse è stato lo sviluppo della parte software di alto livello con la quale sono realizzati la generazione e il controllo vero e proprio del moto del manipolatore. Sfruttando come base dati una lista di via point forniti dall'utente, dopo averne calcolato la cinematica inversa, il sistema di controllo calcola una traiettoria sfruttando un algoritmo di interpolazione. Mediante un controllo in

retroazione viene quindi attuato il movimento. Lo sviluppo di questa parte ha richiesto l'approfondimento di problematiche relative al controllo dei robot manipolatori e la valutazione degli algoritmi più idonei alla loro soluzione.

L'ultima fase del lavoro di tesi ha comportato l'esecuzione di una serie di test per verificare il corretto funzionamento del sistema. In particolare si è verificato che il controllo in retroazione garantisce una precisione accettabile, se si considera l'ambito di lavoro per cui è stato sviluppato il manipolatore. Inoltre sono state costruite alcune traiettorie ad hoc che hanno permesso di verificare la correttezza dell'algoritmo di interpolazione.

Alcune possibili estensioni a questo lavoro di tesi sono le seguenti:

- L'evoluzione del driver del controller CAN verso una migliore gestione della comunicazione nell'ottica dell'utilizzazione all'interno di un sistema real time.
- La costruzione di un insieme di API che forniscano l'accesso al sistema da parte di un utente tramite primitive di alto livello con le quali attuare con facilità il moto.
- La creazione di un sistema di controllo ad anello aperto con il quale poter realizzare, per esempio un sistema di tracking, sfruttando l'inserimento e l'elaborazione dei via a point mentre il manipolatore esegue il moto.

Appendice A

Codice di comunicazione con il manipolatore

In questa appendice viene inserito il codice sorgente del programma fornito con il kit *Transparent Mode* adeguatamente commentato per ottenere una conoscenza piú approfondita della parte di basso livello e della comunicazione.

Listato A.1: Codice che realizza l'utility di comunicazione.

```
//*************************************************************
 // M O D U L O : PCCAN
 //
 // D E S C R I Z I O N E : invio e ricezione messaggi CAN
//*************************************************************

 // Includes

#include < stdio .h>
#include <conio.h>
#include <dos.h>

 // Indirizzi esadecimali dei registri del 82C200

#define CONT_REG 0x300
```

```
#define COMM_REG 0x301
#define STAT_REG 0x302
#define INT_REG 0x303
#define ACC_REG 0x304
#define ACM_REG 0x305
#define BT0_REG 0x306
#define BT1_REG 0x307
#define OUTP_REG 0x308
#define TEST_REG 0x309

#define TXID_REG 0x30A
#define TRTR_REG 0x30B
#define TXB1     0x30C
#define TXB2     0x30D
#define TXB3     0x30E
#define TXB4     0x30F
#define TXB5     0x310
#define TXB6     0x311
#define TXB7     0x312
#define TXB8     0x313

#define RXID_REG 0x314
#define RRTR_REG 0x315
#define RXB1     0x316
#define RXB2     0x317
#define RXB3     0x318
#define RXB4     0x319
#define RXB5     0x31A
#define RXB6     0x31B
#define RXB7     0x31C
#define RXB8     0x31D

#define CLKD_REG 0x31F

 // Define dei valori di stato ed degli errori

#define ERROR_ID 1
#define EXIT     'o'

#define CARTESIAN 1
#define SET_ZERO 2
```

```
#define JOINT        4
#define FOLD_OUT 5
#define FOLD_IN   6

#define STUCK_GRIPPER      0
#define NOT_ATTACHE        1
#define ARM_FOLDED_STRECHED 2
#define BLOCKED_DOF        3
#define MAX_M1_ROTATION 4
#define MOVEMENT_ERROR 5
#define MEMORY186_FULL  6

#define MAX_CART             70
#define MAX_JOINT             2
#define MAX_JOINT_GRIP    10
#define MAX_CART_GRIP    12
#define MAX_JOINT_YPR      3
#define MAX_CART_YPR       6

 // Variabili globali

int pos[8];
int speed[8];
char cbox;
unsigned char manus_status, manus_message;
int error;
unsigned char actual_cbox;

typedef struct
{
  unsigned int ident;
  char rtr, len, dat[8];
} message;

void caninit (void);
void transmit (message *package);
void receive (message *package);
```

```
//************************************************************
 //                          Main
//************************************************************

void main()
{
   message rcvmessage, xmitmessage;
   char i;

    /* Pulisce lo schermo */
    printf ("%c[2J",27);
   cbox = 0;
    printf ("\n\nMANUS␣II␣PC␣CAN␣PROGRAM\n");
    printf (__DATE__);printf("\n\n");
    caninit ();
   help ();

   while(1)
   {
      while(! kbhit ())
      {
            /* Ciclo di trasmissione principale */
            for ( i = 0; i < 3; i++)
            {
                  receive (&rcvmessage);
                 decode(&rcvmessage, &xmitmessage);
                  transmit (&xmitmessage);
                   print_status ();
        }
      }
      readkey ();
   }
}
```

```
//*************************************************************
//                        D E C O D E
//*************************************************************
// Questa funzione implementa la decodifica / codifica dei dati .
// In base al tipo di messaggio ricevuto viene costruita
// l'informazione secondo le specifiche del costruttore .
// Quindi in base alle scelte effettuate dall'utente si procede
// a codificare il messaggio che successivamente verra' inviato
// con i dati di moto

decode(message *rcvmessage, message *xmitmessage)
{
  int i;
  static char echo = 1;
  switch(rcvmessage->ident)
  {
    /* Primo messaggio inviato dal manipolatore */
    case 0x350:
    manus_status  = rcvmessage->dat[0];
    manus_message = rcvmessage->dat[1];

    if ((manus_status & 0x80) == 0x80)  /* message 186 */
    {
        if ((manus_message == 2) && (echo == 1))
        {
            printf ("MANUS gripper is ready\n");
            echo = 2;
        }
        if ((manus_message == 3) && (echo == 2))
        {
            printf ("MANUS absolute angles are ready\n");
            echo = 0;
        }
      }
      /*Costruzione dell'informazione di posiziobe */
      pos [1] = ((unsigned int)rcvmessage->dat [2] << 8) +
                (unsigned char)rcvmessage->dat[3];
      pos [2] = ((unsigned int)rcvmessage->dat [4] << 8) +
                (unsigned char)rcvmessage->dat[5];
      pos [3] = ((unsigned int)rcvmessage->dat [6] << 8) +
                (unsigned char)rcvmessage->dat[7];
```

```
    xmitmessage->ident = rcvmessage->ident;
    xmitmessage->rtr   = 0;
    xmitmessage->len   = 0;
    break;

  /* Secondo messaggio inviato dal manipolatore */
  case 0x360:
  /* Costruzione informazione orientazione graipper */
    pos [4] = (( unsigned int )rcvmessage->dat [0] << 8) +
              (unsigned char)rcvmessage->dat[1];
    pos [5] = (( unsigned int )rcvmessage->dat [2] << 8) +
              (unsigned char)rcvmessage->dat[3];
    pos [6] = (( unsigned int )rcvmessage->dat [4] << 8) +
              (unsigned char)rcvmessage->dat[5];
    pos [7] = (( unsigned int )rcvmessage->dat [6] << 8) +
              (unsigned char)rcvmessage->dat[7];

    xmitmessage->ident = rcvmessage->ident;
    xmitmessage->rtr   = 0;
    xmitmessage->len   = 0;
    break;

  /* Terzo messaggio inviato dal manipolatore */
  /* con relativa codifica della risposta */
  case 0x37F:
    xmitmessage->rtr = 0;      /* per ogni cbox */
    switch(cbox)
    {
      case CARTESIAN:
        xmitmessage->ident = 0x371; /* Stato Cartesiano */
        xmitmessage->len = 8;
        for( i  = 0;  i < xmitmessage->len; i++)
          xmitmessage->dat[i] = speed[ i ];
        break;

      case JOINT:
        xmitmessage->ident = 0x374; /* Stato dei Giunti */
        xmitmessage->len = 8;
        for( i  = 0;  i < xmitmessage->len; i++)
          xmitmessage->dat[i] = speed[ i ];
```

```
            break;

          case FOLD_OUT:
            xmitmessage->ident = 0x375; /* Fold-Out */
            xmitmessage->len = 0;
            break;

          case FOLD_IN:
            xmitmessage->ident = 0x376; /* Fold-In */
            xmitmessage->len = 0;
            break;

          default :       /* Fermo */
            xmitmessage->ident = 0x370;
            xmitmessage->len = 0;
            break;
        } /* fine dello cbox switch */
        break;
    } /* fine del rcvmessage switch */
}

//************************************************************
//                          print status
//************************************************************
// Questa funzione di occupa di comunicare all'utente sia lo
// stato in cui si trova il manipolatore si gli eventuali
// errori che si verificano.

print_status ()
{
  int i;

  actual_cbox = manus_status & 0x0F;

  switch(actual_cbox)
  {
    case 0:  printf ("No move mode\n"); break;
    case 1:  printf ("Cartesian mode\n"); break;
    case 4:  printf ("Joint mode\n"); break;
    case 5:  printf ("Fold out mode\n"); break;
    case 6:  printf ("Fold in mode\n"); break;
```

```
    default :   printf ("Unknown␣mode\n"); break;
  }

  if  (( manus_status & 0xC0) == 0xC0)   /* error  186 */
  {
    switch(manus_message)
    {
          default :
          printf ("ERROR:␣%d\n",manus_message);
          break;
    }
  }
  else  if  (( manus_status & 0x40) == 0x40)   /* warning 186 */
  {
    switch(manus_message)
    {
       case  STUCK_GRIPPER:
           printf ("WARNING:␣stuck␣gripper\n");
          break;
       case  NOT_ATTACHE:
           printf ("WARNING:␣not␣allowed␣to␣attache\n");
          break;
       case  ARM_FOLDED_STRECHED:
           printf ("WARNING:␣arm␣folded␣or␣streched\n");
          break;
       case  BLOCKED_DOF:
           printf ("WARNING:␣blocked␣dof\n");
          break;
       case  MAX_M1_ROTATION:
           printf ("WARNING:␣max␣rotation␣M1␣reached\n");
          break;
       case  MOVEMENT_ERROR:
           printf ("WARNING:␣MANUS␣moving␣without␣input\n");
          break;
       case  MEMORY186_FULL:
           printf ("WARNING:␣memory␣buffer␣full\n");
          break;
        default :
           printf ("WARNING:␣unknown␣warning\n");
          break;
    }
```

```
   }
   else if ((manus_status & 0x80) == 0x80) /* message 186 */
   {
      if (manus_message == 0) printf("MANUS folded\n");
      if (manus_message == 1) printf("MANUS unfolded\n");
   }

   if (error == ERROR_ID)
      printf ("ERROR: wrong CAN identifier received\n");
}

//************************************************************
 //                          print fold status
//************************************************************
 // Legge lo stato del manipolatore e lo visualizza

 print_fold_status ()
{
   int i;
   if ((manus_status & 0x80) == 0x80) /* message 186 */
   {
      if (manus_message == 0) printf("F\n");
      else if (manus_message == 1) printf("U\n");
   }
   else printf ("No fold info\n");
}

//************************************************************
 //                       Readkey
//************************************************************
 // Legge un carattere con il quale l'utente comunica cosa fare
 // al sistema di controllo.

readkey()
{
   int ch, i;
   ch = getch();
   switch(ch)
   /********** Control Box ***********/
   { case '0':
        cbox = 0; break;
```

```
    case '1':
      cbox = 1; break;
    case '4':
      cbox = 4; break;
    case '5':
      cbox = 5; break;
    case '6':
      cbox = 6; break;
  /********* Movimenti *********/
    case 'q':
      if (cbox == 1) speed[1] = MAX_CART;
      else speed[1] = MAX_JOINT;
      break;
    case 'a':
      if (cbox == 1) speed[1] = -MAX_CART;
      else speed[1] = -MAX_JOINT;
      break;
    case 'w':
      if (cbox == 1) speed[2] = MAX_CART;
      else speed[2] = MAX_JOINT;
      break;
    case 's':
      if (cbox == 1) speed[2] = -MAX_CART;
      else speed[2] = -MAX_JOINT;
      break;
    case 'e':
      if (cbox == 1) speed[3] = MAX_CART;
      else speed[3] = MAX_JOINT;
      break;
    case 'd':
      if (cbox == 1) speed[3] = -MAX_CART;
      else speed[3] = -MAX_JOINT;
      break;
    case 'r':
      if (cbox == 1) speed[4] = MAX_CART_YPR;
      else speed[4] = MAX_JOINT_YPR;
      break;
    case 'f':
      if (cbox==1) speed[4] = -MAX_CART_YPR;
      else speed[4] = -MAX_JOINT_YPR;
      break;
```

```
  case 't':
    if (cbox == 1) speed[5] = MAX_CART_YPR;
    else speed[5] = MAX_JOINT_YPR;
    break;
  case 'g':
    if (cbox == 1) speed[5] = -MAX_CART_YPR;
    else speed[5] = -MAX_JOINT_YPR;
    break;
  case 'y':
    if (cbox == 1) speed[6] = MAX_CART_YPR;
    else speed[6] = MAX_JOINT_YPR;
    break;
  case 'h':
    if (cbox == 1) speed[6] = -MAX_CART_YPR;
    else speed[6] = -MAX_JOINT_YPR;
    break;
  case 'u':
    if (cbox == 1) speed[7] = MAX_CART_GRIP;
    else speed[7] = MAX_JOINT_GRIP;
    break;
  case 'j':
    if (cbox == 1) speed[7] = -MAX_CART_GRIP;
    else speed[7] = -MAX_JOINT_GRIP;
    break;
  case ' ':
    for(i=0;i<8;i++)
      speed[i] = 0;
    if ((cbox == 6) || (cbox == 5))
      cbox = 0;
    break;

/********** Scelte generali **********/
  case 'p':
     printf ("Controlbox: %d   ", (actual_cbox = manus_status & 0x0F));
     printf ("Position : ");
    for(i = 1; i < 8; i++)
       printf ("%5d ",pos[i]);
     printf ("\n");
    break;
  case 'i':
     printf ("M A N U S   S T A T U S:\n\n");
```

```
        print_status ();
        break;
    case 'l':
        print_fold_status ();
        break;
    case EXIT:
        printf ("...␣Done␣...\n");
        exit (1);
    case '?':
        help ();
        break;
    default:
        for(i=0;i<8;i++)
            speed[i] = 0;
        if ((cbox == 6) || (cbox == 5))
            cbox = 0;
        break;
  }   /* Fine switch */
}  /* Fine readkey */

//************************************************************
//                          Help
//************************************************************
// Visualizza l'help con le scelte possibili per l'utente

help()
{   printf("type␣'?'␣for␣help\n");
    printf("type␣'o'␣for␣quit\n");
    printf("type␣'p'␣for␣printing␣of␣the␣positions\n");
    printf("type␣'i'␣for␣manus␣status␣info\n");
    printf("type␣'l'␣for␣fold␣status\n");
    printf("type␣'1'␣for␣carthesian␣control\n");
    printf("type␣'4'␣for␣joint␣control\n");
    printf("type␣'5'␣for␣fold␣out\n");
    printf("type␣'6'␣for␣fold␣in\n\n");

    printf("type␣'q/a'␣for␣A1␣or␣X␣(depending␣the␣mode)\n");
    printf("type␣'w/s'␣for␣A2␣or␣Y␣(depending␣the␣mode)\n");
    printf("type␣'e/d'␣for␣A3␣or␣Z␣(depending␣the␣mode)\n");
    printf("type␣'r/f'␣for␣A4␣or␣yaw␣(depending␣the␣mode)\n");
    printf("type␣'t/g'␣for␣A5␣or␣pitch␣(depending␣the␣mode)\n");
```

```
    printf ("type␣'y/h'␣for␣A6␣or␣roll␣(depending␣the␣mode)\n");
    printf ("type␣'u/j'␣for␣gripper␣open/close␣(A7)\n\n");
    printf ("type␣any␣other␣key␣for␣STOP␣movements\n");
}

//*************************************************************
//        CANINIT
//*************************************************************
// Questa funzione permette di impostare sul controller i
// parametri necessari per settare la comunicazione impostando
// i valori nei registri di configurazione
//*************************************************************

void caninit (void)
{
  int i,p;
  unsigned char r;

  /* Resetta il controller */
  outp(CONT_REG, 0x00);

  /* Effettua un check per verificare la presenza dell'hw */
  p = TXID_REG;

  for(i = 0; i <= 10; i++)
  {
    outp(p, 0xAA);
    r = inp(p);
    if (r ^ 0xAA)
    {
        printf ("\nHardware␣error:␣CAN␣driver␣not␣found\n");
        exit (1);
    }
    p++;
  }

  /* Setta il request bit per entrare in configurazione */
  outp(CONT_REG, 0x01);

  /* Satta la maschera di accettazione dell'ID */
  outp(ACC_REG, 0xFF);
```

```
  /* Setta la maschera di accettazione dei bit                    */
  outp(ACM_REG, 0xFF);

  /* Setta il timing a tscl = 0.5us, sjw= 1.5us                   */
  outp(BT0_REG, 0x83);

  /* Setta un sample per bit tseg_1= 2us, tseg_2 = 1.5 us         */
  outp(BT1_REG, 0x23);

  /* Setta il normal output mode, push/pull                       */
  outp(OUTP_REG, 0xFA);

  /* Rilascia il request bit per entrare in operating mode        */
  outp(CONT_REG, 0x00);
}

//*************************************************************
// TRANSMIT
//*************************************************************
// Quetsta e' la funzione che si occupa della trasmissione dei
// messaggi CAN scrivendo direttamente nei registri
//*************************************************************

void transmit(message *package)
{
    unsigned char msb, lsb;
    unsigned int l;
    int i;

    /* Codifica i dati relativi all'ID, rtr e lunghezza */
    i = ((package->ident >> 3) & 0xFF);
    msb = i;
    i = ((package->ident & 0x07) << 5);
    i += ((package->rtr & 0x01) << 4);
    i += (package->len & 0x0F);
    lsb = i;

    /* Aspetta che il controller sia libero per la trasmissione */
    for (l = 0; l < 5000; l++)  /* wait until ready */
    {
```

```
        if (((inp(STAT_REG) & 0x04) » 2) == 1);
        {
            break;
        }
    }
    if (l == 5000)
    {
        printf ("ERROR: Transmit timeout\n");
        exit (1);
    }

    /* Scrive i dati relativi all'ID e len nei buffer */
    outp(TXID_REG, msb);
    outp(TRTR_REG, lsb);

    /* Scrive i dati nel buffer di trasmissione */
    if (package->rtr != 1)
    {
        for(i = 0; i < package->len; i++)
        {
            outp(TXB1 + i, package->dat[i]);
        }
    }

    /* Invia */
    outp(COMM_REG, 0x01);
}

//*************************************************************
// Receive
//*************************************************************
// Questa funzione si occupa della ricezione dei messaggi CAN
// leggendo direttamente dai registri del controller i dati
//*************************************************************

void receive(message *package)
{
    unsigned int msb, lsb, i;
    unsigned long l;
    unsigned char ch;
```

```
    /* Legge il registro di stato per notificare quando e'    */
    /* presente un messaggio                                   */
    for (l = 0; l <= 1000000; l++)
    {
        if ((inp(STAT_REG) & 0x01) != 0) break;
    }

    if (l >= 1000000)
    {
        printf ("ERROR: Receive timeout\n");
        exit (1);
    }

    /* Legge il contenuto dei registri */
    msb = inp(RXID_REG);
    lsb = inp(RRTR_REG);

    /* Decodifica l'informazione */
    package->ident = ((msb & 0xff ) << 3) + ((  lsb & 0xE0 ) >> 5);
    package->rtr = (( lsb & 0x10 ) >> 4);
    package->len = (lsb & 0x0F);

    /* Legge i dati contenuti nei buffer di trasmissione */
    if (package->rtr != 1)
    {
        for(i = 0; i < package->len; i++)
        package->dat[i] = inp(RXB1 + i);
    }
    /* Cancella i dati contenuti nel buffer */
    outp(COMM_REG, 0x04);
}
```

Appendice B

Parametri di Denavit-Hartenberg per il manipolatore MANUS

In questa appendice viene presentata la tabella con i valori dei parametri del modello Denavit-Hartenberg per il manipolatore MANUS utilizzata nel sistema di controllo progettato. In base a questi valori, come si è visto, viene costruita la cinematica del robot.

Giunto	a_i	α_i	d_i	θ_i
1	0.0	$-90.0\,^\circ$	0.0	θ_1
2	400.0	0.0	105.0	θ_2
3	0.0	$90.0\,^\circ$	0.0	θ_3
4	0.0	$-90.0\,^\circ$	320.0	θ_4
5	0.0	$90.0\,^\circ$	0.0	θ_5
6	0.0	0.0	160.0	θ_6

Tabella B.1: Tabella contenete i valori del DH per il MANUS

Bibliografia

[1] C. Schaeffer and T. May. Care-o-bot': A system for assisting elderly or disabled person in home environment. Technical report, Fraunhofer-Institut fur Produktionstechnik und Automatisierung (IPA), 2001.

[2] P. Dario, R. Dillman, and H. Christensen. Euron research roadmaps. http://www.euron.org, March 2004.

[3] H. H. Kwee. Integrated control of manus manipulator and wheelchair enhanced by environmental docking. *Robotica*, 16:491–498, 1998.

[4] J. C. Bauer. Service robots in health care: The evolution of mechanical solution to human resource problems. *TEWS - Future of Service Robots in Health Care*, pages 1–10, June 2003.

[5] B. Borgerding, O. Ivlev, C. Martens, N. Ruchel, and A. Gräser. Friend: Functional robot arm with user friendly interface for disabled people. *http://www.iat.uni-bremen.de/Projekte/HTML_e/FRIEND.htm*.

[6] C. Martens, N. Ruchel, O. Lang, O. Ivlev, and A. Graser. A friend for assistive handicapped people. *IEEE Robotics and Automation Magazine*, pages 57–65, March 2001.

[7] A. Matsikis. Mobile robot tauro. *http://www.techinfo.rwth-aachen.de/Forschung/MSR/Tauro/index.html*.

[8] A. Matsikis. Mobile service robots: Manus. Technical report, Lehrstuhl fur Technische Informatik, 1997/98.

[9] VSMM2002. *Human-Friendly Interaction in Intelligent Sweet Home*, 2002.

[10] F. Monica. Progettazione di un'architettura modulare, aperta ed in tempo reale per un robot mobile. Tesi di Laurea in Ingegneria Elettronica, Università degli Studi di Parma, 2003.

[11] H.H. Kwee and J.J. Duimel. Rehabilitation robotics: The manus concept. 1991.

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

[13] Trinamic Microchips GmbH. *CANnes PC CAN Interface Manual*. Trinamic Microchips GmbH, 1.2 edition, April 2002.

[14] Philips Semiconductors. *SJA1000: Stand-alone CAN controller*. Philips Semiconductors, Jan 2000.

[15] J. J. Craig. *Introductions to robotics: Mechanics and Control*. Addison-Wesley Publishing Company, 2nd edition edition, 1989.

[16] R. Gourdeau. Object oriented programming for robotic manipulators simulation. *IEEE Robotics and Automation Magazine*, 4(3):21–29, 1997.

[17] J. Denavit and R. S. Hartenberg. A kinematic notation for lower pair mechanisms based on matrices. *ASME Jour. of Applied Mechanics*, pages 215–221, June 1955.

[18] R. Boch. Can homepage of robert bosch gmbh. `http://www.can.bosch.com`.

[19] 15 years of can. `http://www.canopen.us/history.htm`.

[20] H. Boternbrood. *CANopen: high-level protocol for CAN-bus*. NIKHEF, Amsterdam, 3.0 edition, March 2000.