

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

REALIZZAZIONE DI UN' ARCHITETTURA
SOFTWARE DI GOVERNO PER 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:

GIORDANO FERRARI

Anno Accademico 2004-2005

Alla mia famiglia.

Indice

1	Robotica per l'assistenza	1
1.1	Progetti di robotica per l'assistenza	4
1.1.1	Il progetto <i>RoboCare</i>	4
1.1.2	Altri progetti di robotica per l'assistenza	5
1.2	Obbiettivi della tesi	9
1.3	Organizzazione della tesi	10
2	Pianificazione di traiettorie e controllo del moto	12
2.1	Pianificazione di traiettorie	12
2.1.1	Pianificazione nello spazio dei giunti	13
2.1.2	Moto punto a punto	13
2.1.3	Moto su percorso assegnato	16
2.2	Controllo del moto	19
3	Il manipolatore Manus	21
3.1	Caratteristiche meccaniche	22
3.2	Cinematica	25
3.3	Elettronica di controllo	30
3.4	Protocollo di comunicazione	33
3.4.1	Il protocollo CAN BUS	33
3.4.2	Comunicazioni con il manipolatore	38
4	Realizzazione del sistema	44
4.1	Obbiettivi e requisiti del sistema	44
4.1.1	YARA	45

4.2	Architettura del sistema	48
4.2.1	Il modulo <i>Encoder</i>	48
4.2.2	Il modulo <i>Command</i>	53
4.2.3	Il modulo <i>Control</i>	56
5	Risultati sperimentali	70
5.1	Risposta al gradino del controllore PI	70
5.2	Test sul generatore di traiettorie	72
6	Conclusioni	77
	Appendici	80
A	I manipolatori robotici	80
A.1	Struttura di un manipolatore robotico	80
A.2	Descrizioni e trasformazioni spaziali	82
A.3	Cinematica diretta	86
A.3.1	Convenzione di Denavit-Hartenberg	88
A.3.2	Convenzione di Denavit-Hartenberg modificata	94
A.4	Cinematica inversa	96
A.4.1	Formulazione del problema cinematico inverso	96
A.4.2	Risolubilità del problema cinematico inverso	96
A.4.3	Disaccoppiamento cinematico	99
A.4.4	Soluzione del manipolatore antropomorfo	100
A.5	Cinematica differenziale	102
B	Parametri di Denavit-Hartenberg modificati del <i>Manus</i>	106
	Bibliografia	106

Capitolo 1

Robotica per l'assistenza

La società moderna sta affrontando il serio problema del progressivo invecchiamento della popolazione. Spesso gli anziani sono costretti a lasciare la propria abitazione per andare a vivere con i familiari o essere ospitati in istituti in grado di fornire loro l'assistenza adeguata. Questo comporta una diminuzione del senso di autonomia ed indipendenza della persona assistita, a cui si vanno ad aggiungere oneri sociali ed economici legati ai costi che i singoli individui e le istituzioni devono sostenere.

In quest'ottica, data la necessità di fornire una qualità di vita soddisfacente a queste persone e la crescente richiesta di personale, negli ultimi anni molti gruppi di ricerca hanno indirizzato i propri studi al nuovo campo dell'applicazione della robotica all'assistenza di anziani e disabili, ed alle attività di riabilitazione. A tutt'oggi il lavoro di questi gruppi ha reso disponibili una serie di soluzioni a diversi livelli di complessità tecnologica, e ha portato all'individuazione dell'insieme di funzionalità di base che un robot di assistenza dovrebbe poter offrire ai propri utilizzatori, illustrato di seguito:

- Svolgimento delle tradizionali mansioni domestiche:
 - trasporto e manipolazione di oggetti.
 - pulizia della casa.
 - gestione delle infrastrutture tecniche (impianto di riscaldamento, aria condizionata o sistema d'allarme).

- Supporto alla mobilità: il sistema robotico deve essere in grado di aiutare gli utenti nei loro spostamenti all'interno dell'ambiente domestico e, nel caso di persone immobilizzate, il robot con il proprio operato deve comunque contribuire all'indipendenza ed all'autonomia dell'assistito.
- Comunicazione ed integrazione sociale: l'assistente robotico deve consentire e facilitare i rapporti tra l'utente, i suoi familiari ed il mondo esterno, fornendo degli strumenti semplificati per l'utilizzo dei tradizionali media (telefono, televisione, radio).
- Cura della salute: l'assistente robotico deve garantire un continuo monitoraggio della salute dell'assistito. Questo compito può essere svolto in due modi:
 - instaurando un canale di comunicazione dedicato tra utente e strutture medico-sanitarie.
 - tramite intervento diretto dell'assistente: ricordando ad esempio all'utente di assumere i medicinali nei tempi e nelle modalità corrette, o monitorando i suoi parametri vitali in modo da poter consentire un intervento di soccorso tempestivo in caso di emergenza.

I primi studi ed i primi progetti realizzati, hanno portato all'identificazione di cinque aree di ricerca fondamentali per la realizzazione pratica dei robot assistenti in grado di soddisfare i requisiti sopra elencati.

- *Comunicazione uomo-macchina*: la cooperazione tra utente e robot deve essere garantita anche nelle situazioni più complesse. In quest'ottica l'assistente robotico dovrebbe essere dotato di sistemi di comunicazione multimodali, che gli consentano di comprendere i diversi tipi di comando impartiti dall'utente (comandi vocali, tattili o provenienti da interfacce grafiche).
- *Analisi ed interpretazione della scena*: la collaborazione tra uomo e macchina è influenzata da come quest'ultima percepisce l'ambiente circostante, e da come viene interpretato il compito da svolgere all'interno del contesto.

- *Apprendimento*: la sola intelligenza di cui è dotato inizialmente il robot non è sufficiente per lo svolgimento di un effettivo compito di assistenza. È necessario anche un trasferimento di conoscenza tra l'uomo e la macchina. Una possibile forma di apprendimento è rappresentata dalla programmazione per dimostrazione.
- *Pianificazione del moto e coordinamento*: l'interazione uomo-macchina e la realizzazione di compiti comporta una accurata pianificazione del moto, ed elevate capacità di coordinamento da parte del robot.
- *Sicurezza*: l'assistente robotico durante il suo operato deve garantire l'integrità delle persone e degli oggetti che lo circondano. Gli eventi esterni che minacciano il corretto funzionamento del sistema, o eventuali errori interni, devono essere classificati in base al fattore di rischio che comportano, al fine di evitare il verificarsi di situazioni potenzialmente pericolose.

Oltre agli aspetti prettamente tecnici, alcuni gruppi di ricerca si dedicano allo studio dell'impatto psicologico e sociale che l'introduzione dei robot ha sugli anziani, tradizionalmente diffidenti nei confronti delle innovazioni tecnologiche. Oggi i ricercatori che si occupano di sviluppare tecnologie per l'assistenza agli anziani devono tenere in considerazione il fatto che le persone a cui sono destinati i prodotti sono culturalmente distanti dalle soluzioni tecniche proposte. Gli anziani invece che per primi usufruiranno pienamente di queste nuove tecnologie nei prossimi anni, avranno fatto parte attivamente della società dell'informazione ed avranno quindi un rapporto differente con la tecnologia, anche se alcune delle problematiche riguardanti individui con problemi motori o mentali rimarranno invariate. Lo studio sull'impatto delle nuove tecnologie non si limita all'analisi del livello di accettazione da parte di coloro che devono ricevere l'assistenza, ma investe anche quello degli operatori che riceveranno la collaborazione dei robot nel loro lavoro quotidiano, e che in parte si vedranno sostituiti nelle loro tradizionali mansioni da questi ultimi.

Risultati incoraggianti sono stati ottenuti da diversi progetti attivi in Europa, Stati Uniti e Giappone, dimostrando non solo la realizzabilità tecnologica, ma anche

la potenzialità della robotica nel settore dell'assistenza personale. Nel successivo paragrafo sono illustrate le caratteristiche di alcuni di questi progetti, iniziando da *RoboCare*, in cui rientra questo lavoro di tesi.

1.1 Progetti di robotica per l'assistenza

1.1.1 Il progetto *RoboCare*

RoboCare [1] è un progetto lanciato in Italia nel Dicembre 2002, finanziato dal *MIUR* (Ministero Italiano per l'Università e la Ricerca) e portato avanti dall'attività congiunta di diversi gruppi di lavoro tra cui il Dipartimento di Ingegneria dell'Informazione dell'Università di Parma. L'obiettivo del progetto è di sviluppare un sistema distribuito in cui un insieme eterogeneo di agenti fissi e mobili, agenti software, robot autonomi, sensori e personale umano, collaborano assieme con l'intento comune di fornire cura e supporto costante ad anziani e disabili, aiutando queste persone in diversi aspetti della loro vita, come il prendere decisioni, ricordare le scadenze giornaliere o avvertirli su pericoli imminenti.

Gli scenari previsti per l'utilizzo del sistema sono principalmente due:

- L'ambiente domestico, in cui l'accoppiata di tecnologie domotiche e sistemi esperti consentirà agli anziani di essere assistiti nelle loro attività quotidiane.
- Le residenze sanitarie assistite, in cui gli agenti robotici si affiancheranno ed in parte sostituiranno nelle attività di assistenza e monitoraggio i tradizionali operatori umani.

L'utilizzo di agenti autonomi e tecnologie cognitive e di calcolo distribuite rappresenta la base per la realizzazione di un simile sistema esperto basato su comportamenti collaborativi. Esistono già tecnologie che consentono a squadre di robot di cooperare tra loro, e l'intelligenza artificiale ha già in parte trovato una soluzione alle questioni della pianificazione dei compiti e della risoluzione automatica dei problemi. Un sistema di assistenza deve però saper integrare le tecnologie di questi due ambiti, e ciò deve essere fatto tramite il controllo di un supervisore che sia in grado

di mantenere una visione d'insieme del sistema. Il supervisore è il responsabile sia della sintetizzazione di un piano per il raggiungimento degli obiettivi prefissi, sia del controllo dell'avanzamento di questo piano, e deve inoltre poter fornire delle funzionalità di controllo agli operatori umani. E' proprio la natura *multiagente* dell'architettura e la presenza di un supervisore a differenziare il progetto *RoboCare* da altri progetti di robotica per l'assistenza quali *Care-o-Bot* [2] *Morpha* [3], *NurseBot* [4] ed *Assisted Cognition Project* [5].

Nell'ambito del progetto *RoboCare*, l'attività del Dipartimento di Ingegneria dell'Informazione dell'Università di Parma è mirata all'allestimento di una base mobile equipaggiata con un manipolatore per compiti di assistenza a persone affette da difficoltà motorie, ed in particolare agli arti inferiori. L'azione simultanea della base mobile e del manipolatore consentirà alla piattaforma di portare a termine compiti combinati di navigazione e trasporto di oggetti, a tutto vantaggio di una maggior autonomia ed indipendenza dell'utente. Il sistema deve ad esempio essere in grado di spostarsi all'interno di un ambiente non completamente reingegnerizzato, riconoscere un oggetto indicatogli precedentemente dall'utente mediante un apposito sistema di input, afferrarlo e consegnarlo a quest'ultimo.

La base mobile della piattaforma è rappresentata da un *Nomad 200*, prodotto da *Nomadic Technologies*, mentre il manipolatore è il *Manus*, realizzato da *Exact Dynamics*. Si tratta di un manipolatore a bassa impedenza meccanica appositamente progettato per compiti di assistenza, che è stato scelto per le sue caratteristiche di leggerezza e sicurezza intrinseca, descritte in modo dettagliato nel capitolo 3.

1.1.2 Altri progetti di robotica per l'assistenza

Nel paragrafo 1.1.1 si accenna ad una analogia tra il progetto *RoboCare* ed altri progetti di robotica per l'assistenza. Vediamo quali sono gli obiettivi e le caratteristiche di alcuni di questi progetti.

Care-o-Bot

Care-o-Bot è un robot mobile di servizio, progettato e realizzato dall'istituto Fraunhofer di Stoccarda, in grado di realizzare compiti di manipolazione e trasporto in



Figura 1.1: La base mobile *Nomad 200* e il manipolatore *Manus*.

ambiente domestico, in aggiunta ad altre operazioni di supporto a persone anziane o disabili. Il primo modello realizzato risale al 1998.

Dal punto di vista tecnico le caratteristiche principali di *Care-o-Bot* sono:

- Interfaccia utente multimodale: anche senza alcuna conoscenza tecnica l'utente del robot deve poter impartire dei comandi in modo semplice ed intuitivo, potendo ricorrere a canali di comunicazione multipli, come la parola, il tatto, o i gesti.
- Pianificatore di compiti interattivo: *Care-o-Bot* è equipaggiato con un'architettura di controllo ibrida *reattiva-deliberativa*. Ciascun comando immesso dall'utente è trasferito al pianificatore simbolico, che genera la lista di azioni necessarie per portare a termine il compito assegnato. Per l'esecuzione dei compiti il robot si affida ad un modello del mondo precedentemente caricato da una base di dati. Il modello dell'ambiente è successivamente e continuamente aggiornato dal robot sulla base dei dati sensoriali provenienti dall'esterno. Nel suo complesso questa architettura consente a *Care-o-Bot* di svolgere in modo autonomo compiti complessi.



Figura 1.2: Robot mobile del progetto *Nursebot*.

- Manipolazione: *Care-o-Bot* è dotato di un manipolatore appositamente progettato per esser utilizzato su un robot mobile di servizio, per la manipolazione dei tipici oggetti presenti in una abitazione, come una bottiglia o dei piatti.
- Assistenza alla deambulazione: sono presenti appositi supporti e sistemi di controllo che consentono al robot di supportare in modo sicuro e confortevole gli utilizzatori nei loro spostamenti.

Nursebot

Nursebot è un progetto interdisciplinare condotto dalla Carnegie Mellon University in collaborazione con la Pittsburg University. L'obiettivo è quello di sviluppare dei robot mobili di servizio che siano in grado di assistere nella loro vita quotidiana persone anziane affette da malattie croniche. La scelta di utilizzare dei robot mobili è legata al fatto che questa tecnologia presenta le caratteristiche adeguate in termini



Figura 1.3: Robot mobile del progetto *Nursebot*.

di robustezza e capacità d'azione, a cui si aggiunge il fattore importante nel settore sanitario, rappresentato dai costi non eccessivamente elevati.

Tra i servizi che ciascun robot deve essere in grado di svolgere una particolare attenzione è rivolta alle seguenti attività:

- Aiutare quei pazienti che non hanno una vita autonoma a causa della loro memoria, ricordandogli ad esempio di bere, assumere i medicinali o andare dal medico.
- Consentire un collegamento remoto che permetta un'interazione diretta e continuata tra l'anziano ed il medico, evitando in questo modo visite superflue.
- Realizzare un monitoraggio dei parametri vitali dei pazienti più critici in modo da cercare di prevenire possibili situazioni di emergenza.
- Aiutare tramite compiti di manipolazione gli anziani che presentano problemi di mobilità agli arti.



Figura 1.4: Robot mobile del progetto *Morpha*

- Svolgere compiti di interazione sociale con gli anziani che vivono soli, privi di contatti con altri individui.

Morpha

Morpha è un progetto finanziato dal ministero tedesco per l'educazione e la ricerca, e condotto da un consorzio di 16 enti suddivisi fra centri di ricerca ed industrie. L'obiettivo del progetto è quello di fornire a robot di servizio o con compiti di assistenza capacità comunicative che consentano loro di interagire con gli esseri umani in modo intuitivo e naturale, così da accrescerne le possibilità collaborative. L'aumento dell'interazione uomo-robot ha il duplice ruolo di consentire all'utente di insegnare al robot come risolvere i nuovi problemi che l'ambiente circostante propone e che sono al di fuori delle sue competenze iniziali, e facilitare l'indicazione dei compiti che devono essere svolti.

L'interazione uomo-robot richiede la presenza di molteplici strumenti di comunicazione, e per questo motivo il progetto prevede di affiancare alle tradizionali interfacce quelle che da sempre costituiscono i canali di comunicazione privilegiati dall'uomo: la parola ed i gesti, a cui è previsto che si possa aggiungere una vera e propria interazione fisica durante le fasi di insegnamento, agevolate dall'adozione di tecnologie come i sensori tattili.

1.2 Obiettivi della tesi

Nell'ambito dell'attività svolta dal Dipartimento di Ingegneria dell'Informazione dell'Università di Parma all'interno del progetto *RoboCare*, l'obiettivo di questo lavoro di tesi consiste nella progettazione ed implementazione di un'architettura di

controllo per il manipolatore a bassa impedenza *Manus*, che consenta la realizzazione di task di manipolazione. A tal proposito, le tre aree di principale interesse ai fini della realizzazione del sistema sono:

- Comunicazione tra sistema di controllo e manipolatore.
- Pianificazione del moto.
- Controllo del moto.

1.3 Organizzazione della tesi

Vediamo brevemente come sono organizzati i prossimi capitoli.

Il *capitolo 2* contiene una breve discussione delle tematiche relative alla *pianificazione* ed al *controllo* del moto, utili per inquadrare gli algoritmi utilizzati nella realizzazione del sistema

Il *capitolo 3* descrive le caratteristiche del manipolatore *Manus*, che stanno alla base della realizzazione del sistema di controllo realizzato: struttura meccanica e cinematica, le componenti elettroniche, ed il protocollo *CAN* utilizzato per la comunicazione tra calcolatore e manipolatore.

Nel *capitolo 4* è presentata l'architettura del sistema controllo: i moduli che lo compongono, le attività svolte da ciascun modulo, le librerie esterne usate per l'implementazione, ed il modo in cui i diverse componenti interagiscono tra loro per garantire il corretto funzionamento.

Nel *capitolo 5* sono riportati i risultati dei test realizzati sul sistema. É stata eseguita una prova preliminare per valutare le caratteristiche del controllore PI integrato nell'elettronica di controllo nel manipolatore, e successivamente sono stati eseguiti dei test per valutare la qualità delle traiettorie generate.

Per concludere, nel *capitolo 6* sono esposte alcune considerazioni sui risultati ottenuti, sulle soluzioni adottate, sulle possibili evoluzioni future da apportare al sistema.

Capitolo 2

Pianificazione di traiettorie e controllo del moto

In questo capitolo sono introdotti i concetti di base dei due argomenti attorno a cui si è concentrato il lavoro sviluppato in questa tesi: la *pianificazione di traiettorie*, ed il *controllo del moto*.

2.1 Pianificazione di traiettorie

Lo scopo della *pianificazione di traiettorie* [6] è quello di fornire in ingresso al sistema di controllo del moto un riferimento da utilizzare per imporre al manipolatore l'esecuzione di un movimento specifico all'interno del proprio spazio di lavoro. Prima di vedere come opera il pianificatore è bene specificare il concetto di *percorso* e *traiettoria*. Con il primo termine si intende una sequenza di punti nello *spazio dei giunti* o in quello *operativo* (cartesiano) che il manipolatore deve seguire per eseguire il compito assegnato. Con il termine *traiettoria*, si intende invece un percorso su cui è specificata per ciascun punto una legge oraria del moto in termini di velocità e accelerazione. In linea di principio il pianificatore prende come ingresso un percorso definito dall'utente in cui vengono specificati i punti estremi, eventuali punti intermedi e vari tipi di vincoli, quali vincoli temporali, o sulle velocità ed accelerazioni massime consentite, e produce in uscita una sequenza temporale di valori che specificano posizione, orientazione, velocità ed accelerazione dell'organo terminale

del manipolatore, che serviranno da riferimento per il controllo del moto.

La definizione del percorso può avvenire sia nello spazio operativo sia in quello dei giunti. La prima soluzione offre il vantaggio di una determinazione più intuitiva del compito che il manipolatore deve svolgere, e facilita la definizione dei vincoli legati alla presenza di ostacoli all'interno dello spazio di lavoro.

2.1.1 Pianificazione nello spazio dei giunti

La pianificazione nello spazio dei giunti prevede che il calcolo della traiettoria sia fatto in funzione delle variabili di giunto del manipolatore. Rispetto alla pianificazione nello spazio operativo, quella nello spazio dei giunti elimina le problematiche legate all'esistenza di configurazioni singolari del manipolatore. Nel caso molto comune in cui il percorso contenente i punti di passaggio sia assegnato nello spazio operativo è necessario trasformare questi punti in equivalenti variabili di giunto tramite inversione cinematica. Fatto questo si procede con il calcolo di una funzione $q(t)$ che interpola i diversi punti di passaggio rispettando gli eventuali vincoli presenti nella definizione del percorso. La funzione interpolatrice $q(t)$ deve garantire la continuità della posizione, della velocità e in alcuni casi anche dell'accelerazione delle variabili di giunto, e deve garantire una curvatura priva di irregolarità ed oscillazioni eccessive.

2.1.2 Moto punto a punto

Vediamo com'è possibile scegliere una funzione interpolante che abbia caratteristiche soddisfacenti partendo dal caso di un semplice *moto punto-punto* che consiste nel far muovere l'organo terminale del manipolatore da un punto iniziale ad uno finale in un dato intervallo di tempo $t_f - t_i$, disinteressandoci per il momento del comportamento del manipolatore nei punti intermedi.

Polinomi cubici

La funzione interpolatrice di ciascun giunto deve garantire che al tempo t_i il giunto si trovi nella posizione iniziale q_i , e al tempo t_f in quella finale q_f . I primi due vincoli che $q(t)$ deve rispettare sono quindi:

$$\begin{aligned}q(t_i) &= q_i \\q(t_f) &= q_f\end{aligned}\tag{2.1}$$

Oltre ai due vincoli sulle posizioni si possono imporre anche due vincoli sulla velocità iniziale \dot{q}_i e su quella finale \dot{q}_f che solitamente sono nulle.

$$\begin{aligned}\dot{q}(t_i) &= \dot{q}_i \\ \dot{q}(t_f) &= \dot{q}_f\end{aligned}\tag{2.2}$$

Questi quattro vincoli possono essere soddisfatti da un *polinomio cubico* del tipo:

$$q(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0\tag{2.3}$$

a cui corrisponde una velocità con profilo parabolico ed una accelerazione lineare:

$$\dot{q}(t) = 3a_3 t^2 + 2a_2 t + a_1\tag{2.4}$$

$$\ddot{q}(t) = 6a_3 t + 2a_2\tag{2.5}$$

Dalla combinazione delle equazioni 2.3, 2.4 e 2.5 con i vincoli 2.1 e 2.2 e ponendo $t_0 = 0$ si ottiene il seguente sistema di quattro equazioni in quattro incognite:

$$q_i = a_0\tag{2.6}$$

$$\dot{q}_i = a_1\tag{2.7}$$

$$q_f = a_3 t_f^3 + a_2 t_f^2 + a_1 t_f + a_0\tag{2.8}$$

$$\dot{q}_f = 3a_3 t_f^2 + 2a_2 t_f + a_1\tag{2.9}$$

Risolviendo il sistema si ottengono le espressioni dei coefficienti a_i dell'equazione 2.3:

$$a_0 = q_i \quad (2.10)$$

$$a_1 = \dot{q}_i \quad (2.11)$$

$$a_2 = \frac{3}{t_f^2} (q_f - q_i) - \frac{\dot{q}_f}{t_f} - \frac{\dot{q}_i}{2t_f} \quad (2.12)$$

$$a_3 = \frac{2}{t_f^3} (q_f - q_i) + \frac{1}{t_f^2} (\dot{q}_f + \dot{q}_i) \quad (2.13)$$

Polinomi di grado superiore

Qualora si vogliono imporre dei valori iniziali e finali anche sulle accelerazioni i vincoli da soddisfare diventano sei e si rende quindi necessario l'uso di un polinomio di quinto grado del tipo:

$$q(t) = a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0 \quad (2.14)$$

L'imposizione dei vincoli di posizione, velocità ed accelerazione per l'equazione 2.14 porta al sistema di sei equazioni in sei incognite seguente:

$$q_i = a_0 \quad (2.15)$$

$$\dot{q}_i = a_1 \quad (2.16)$$

$$\ddot{q}_i = 2a_2 \quad (2.17)$$

$$q_f = a_5 t_f^5 + a_4 t_f^4 + a_3 t_f^3 + a_2 t_f^2 + a_1 t_f + a_0 \quad (2.18)$$

$$\dot{q}_f = 5a_5 t_f^4 + 4a_4 t_f^3 + 3a_3 t_f^2 + 2a_2 t_f + a_1 \quad (2.19)$$

$$\ddot{q}_f = 20a_5 t_f^3 + 12a_4 t_f^2 + 6a_3 t_f + 2a_2 \quad (2.20)$$

La soluzione del sistema è:

$$a_0 = q_i \quad (2.21)$$

$$a_1 = \dot{q}_i \quad (2.22)$$

$$a_2 = \frac{\ddot{q}_i}{2} \quad (2.23)$$

$$a_3 = \frac{20q_f - 20q_i - (8\dot{q}_f + 12\dot{q}_i)t_f - (3\ddot{q}_i - 2\ddot{q}_f)t_f^2}{2t_f^3} \quad (2.24)$$

$$a_4 = \frac{30q_i - 30q_f + (14\dot{q}_f + 16\dot{q}_i)t_f + (3\ddot{q}_i - 2\ddot{q}_f)t_f^2}{2t_f^4} \quad (2.25)$$

$$a_4 = \frac{12q_f - 12q_i - (6\dot{q}_f + 6\dot{q}_i)t_f - (\ddot{q}_i - \ddot{q}_f)f_f^2}{2t_f^2} \quad (2.26)$$

2.1.3 Moto su percorso assegnato

Fino a ora abbiamo preso in considerazione solo il caso di un moto punto-punto in cui il percorso è specificato solamente dalla posizione iniziale e finale che l'organo terminale del manipolatore deve occupare; solitamente però nella descrizione di un percorso si utilizzano più di due punti con l'introduzione di valori intermedi. La densità di questi punti è scelta in base al livello di precisione del moto che si vuole ottenere, andando ad esempio ad aumentarne il numero in prossimità di eventuali ostacoli o zone a curvatura accentuata. Con a disposizione n punti una possibile funzione interpolante è rappresentata da un polinomio di grado $n - 1$. Questa soluzione è però insoddisfacente per alcuni motivi: per prima cosa all'aumentare di n aumentano le oscillazioni introdotte dal polinomio, aumenta il carico computazionale richiesto per i calcoli dei coefficienti e diminuisce la precisione nella determinazione degli stessi. In secondo luogo i coefficienti del polinomio dipendono da tutti gli n punti, e la modifica ad uno solo di questi obbliga ad un nuovo calcolo dell'intera funzione. La soluzione al problema consiste nel sostituire il polinomio di grado $n - 1$ con delle funzioni interpolatrici di grado più basso, unite fra loro nei punti intermedi del percorso. Il polinomio cubico visto in precedenza, con i suoi quattro coefficienti ha i requisiti minimi per poter essere utilizzato assicurando la continuità della velocità, e garantendo allo stesso tempo il passaggio per i punti specificati. Il calcolo dell'intera traiettoria si realizza quindi interpolando ciascuna coppia di pun-

ti consecutivi del percorso tramite le equazioni 2.10, 2.11, 2.12 e 2.13. Per calcolare i coefficienti è necessario conoscere le velocità dei punti intermedi, e nel caso in cui queste non siano esplicitamente dichiarate dall'utente nella definizione del percorso possono essere determinate imponendo la condizione di continuità dell'accelerazione nei punti di raccordo dei polinomi interpolanti. Vediamo nel dettaglio il calcolo. Siano $q_i(t_j)$, $q_i(t_{j+1})$ e $q_i(t_{j+2})$ i tre valori consecutivi del percorso specificato, imponendo la continuità dell'accelerazione nel punto intermedio $q_i(t_{j+1})$ dall'equazione 2.5 si ottiene:

$$2a_{2j} + 6a_{3j}t_{j+1} = 2a_{2j+1} \quad (2.27)$$

Sostituendo i coefficienti a_i in 2.27 con le espressioni trovate in 2.10-2.13 si ottiene:

$$\begin{aligned} \frac{2\dot{q}_i(t_j)}{t_{j+1} - t_j} + \left(\frac{4}{t_{j+1} - t_j} + \frac{4}{t_{j+2} - t_{j+1}} \right) \dot{q}_i(t_{j+1}) + \frac{2\dot{q}_i(t_{j+2})}{t_{j+2} - t_{j+1}} = \\ \frac{6(q_i(t_{j+2}) - q_i(t_{j+1}))}{(t_{j+2} - t_{j+1})^2} + \frac{6(q_i(t_{j+1}) - q_i(t_j))}{(t_{j+1} - t_j)^2} \end{aligned} \quad (2.28)$$

Variando j da 1 a $n - 1$, con n il numero dei punti del percorso si ottengono $n - 1$ espressioni della velocità $\dot{q}_i(t_j)$ rappresentabili in forma matriciale:

$$A\dot{q}_i = \vec{b} \quad (2.29)$$

Dove A , \dot{q}_i , b hanno rispettivamente la seguente forma:

$$\mathbf{A} = \begin{bmatrix} \beta_0 & \gamma_0 & 0 & 0 & \dots & 0 & 0 \\ \alpha_1 & \beta_1 & \gamma_1 & 0 & \dots & 0 & 0 \\ 0 & \alpha_2 & \beta_2 & \gamma_2 & 0 & \dots & 0 \\ \vdots & & & & \ddots & & \vdots \\ 0 & \dots & 0 & \alpha_{k-3} & \beta_{k-3} & \gamma_{k-3} \\ 0 & \dots & 0 & 0 & \alpha_{k-2} & \beta_{k-2} \end{bmatrix} \quad (2.30)$$

$$\dot{q}_i = (\dot{q}_i(t_1), \dot{q}_i(t_2), \dots, \dot{q}_i(t_{k-1}))^T \quad (2.31)$$

$$b = (\delta_0, \delta_1, \dots, \delta_{k-2})^T \quad (2.32)$$

Le espressioni di α_j , β_j , γ_j e δ_j sono:

$$\alpha_j = \frac{2}{t_{j+1} - t_j} \quad (2.33)$$

$$\beta_j = \frac{4}{t_{j+1} - t_j} + \frac{4}{t_{j+2} - t_{j+1}} \quad (2.34)$$

$$\gamma_j = \frac{2}{t_{j+2} - t_{j+1}} \quad (2.35)$$

$$\delta_j = \frac{6(q_i(t_{j+2}) - q_i(t_{j+1}))}{(t_{j+2} - t_{j+1})^2} + \frac{6(q_i(t_{j+1}) - q_i(t_j))}{(t_{j+1} - t_j)^2} \quad (2.36)$$

I valori δ_0 e δ_{k-2} devono essere trattati a parte rispetto agli altri δ_j ; le loro espressioni sono:

$$\delta_0 = \frac{6(q_i(t_2) - q_i(t_1))}{(t_2 - t_1)^2} + \frac{6(q_i(t_1) - q_i(t_0))}{(t_1 - t_0)^2} - \frac{2}{t_1 - t_2} \dot{q}_i(t_0) \quad (2.37)$$

$$\begin{aligned} \delta_{k-2} = & \frac{6(q_i(t_k) - q_i(t_{k-1}))}{(t_k - t_{k-1})^2} + \frac{6(q_i(t_{k-1}) - q_i(t_{k-2}))}{(t_{k-1} - t_{k-2})^2} \\ & - \frac{2}{t_k - t_{k-1}} \dot{q}_i(t_{k-2}) \end{aligned} \quad (2.38)$$

A questo punto abbiamo a disposizione tutte le velocità e possiamo calcolare tutti i polinomi cubici che interpolano il percorso voluto.

2.2 Controllo del moto

Nei paragrafi precedenti abbiamo visto in che modo sia possibile determinare un insieme di variabili di giunto che descriva uno specifico movimento dell'organo terminale di un manipolatore; resta da vedere in che modo possano essere utilizzate queste informazioni nell'azione di controllo. Il compito di consentire effettivamente la realizzazione della traiettoria desiderata spetta al *sistema di controllo del moto*. Così come per la pianificazione, anche il controllo può essere realizzato sia nello *spazio dei giunti* che in quello *operativo*. Nel primo caso il controllore deve garantire l'inseguimento dei riferimenti da parte delle variabili di giunto, che sono le grandezze controllate. Questo approccio garantisce una maggiore semplicità realizzativa data dalla possibilità di separare il calcolo della cinematica inversa dall'azione di controllo vera e propria. L'operare direttamente sulle variabili di giunto implica però che il controllo delle grandezze nello spazio operativo sia effettuato in anello aperto mediante la struttura cinematica del manipolatore; questo comporta che eventuali giochi negli organi di trasmissione o la non perfetta rigidità della struttura si ripercuotono interamente sulla precisione del moto. Il controllo nello spazio operativo consente concettualmente di eliminare il problema appena visto, a discapito di una maggior complessità dell'algoritmo, dovuta alla risoluzione del problema cinematico inverso direttamente all'interno dell'anello di controllo. Dal

momento però che spesso i valori dello spazio operativo non sono noti direttamente, ma vengono dedotti dai valori delle variabili di giunto, questo vantaggio teorico si perde.

Per quanto riguarda il controllo nello spazio dei giunti sono possibili due approc-

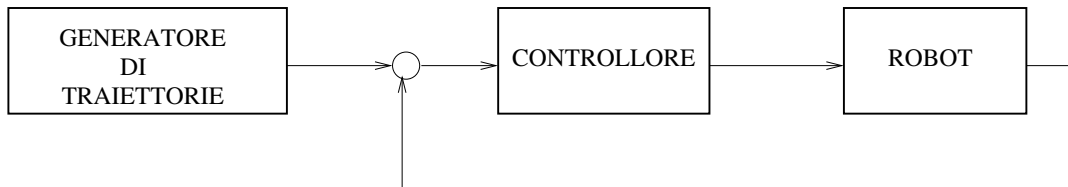


Figura 2.1: Controllo in retroazione nello spazio dei giunti.

ci: il primo consiste nell'utilizzare uno schema di controllo *decentralizzato*, in cui ogni singolo giunto del manipolatore viene controllato in modo indipendente rispetto agli altri. Il secondo approccio ricorre a schemi centralizzati, in cui il controllo dei giunti viene realizzato tenendo conto della reciproca interazione dinamica. In figura 2.1 è schematizzato un classico sistema di controllo in retroazione negativa operante nello spazio dei giunti: vediamo brevemente il funzionamento. Il blocco *GENERATORE DI TRAIETTORIE* opera come si è visto nel paragrafo 2.1: prende in ingresso una sequenza di punti di passaggio, esegue la cinematica inversa nel caso in cui i punti siano definiti nello spazio operativo, e ne fa l'interpolazione nello spazio dei giunti. In questo modo vengono generate delle traiettorie che permettono di fornire al manipolatore un riferimento di posizione, velocità ed accelerazione per le variabili di giunto ad un dato istante di tempo. Ai riferimenti così prodotti vengono sottratti i corrispondenti valori provenienti dal manipolatore tramite l'anello di retroazione, ed il risultato ottenuto passa in ingresso al blocco *CONTROLLO*, che si occupa di generare un opportuno set di segnali da inviare agli attuatori del manipolatore in modo da realizzare la traiettoria voluta.

Capitolo 3

Il manipolatore Manus

In questo capitolo verranno presentate le caratteristiche tecniche del manipolatore *MANUS* che è indispensabile conoscere per la realizzazione del sistema di controllo: la struttura meccanica e cinematica, l'elettronica di controllo e il protocollo di comunicazione tra robot e calcolatore.

Il *Manus* è un robot manipolatore realizzato e commercializzato dall'azienda olandese Exact Dynamics [7]. Si tratta di un prodotto appositamente concepito per essere installato su una sedia a rotelle al fine di assistere le persone disabili nelle loro attività quotidiane più comuni come il bere, il mangiare, o più semplicemente per afferrare oggetti altrimenti irraggiungibili, accrescendo in questo modo l'autonomia di tali individui, e migliorandone la qualità della vita. I principi che hanno guidato la progettazione del *Manus* possono essere riassunti nei seguenti punti [8]:

- L'utente si trova all'interno dello spazio di lavoro del manipolatore e come tale la sicurezza dell'intero sistema riveste il ruolo fondamentale: le forze esercitate dal manipolatore e le velocità di spostamento devono essere limitate ed in qualsiasi istante al verificarsi di un problema il sistema deve poter essere arrestato immediatamente per non arrecare danni agli individui.
- L'utente è parte attiva del sistema di controllo: aziona infatti il manipolatore sulla base degli oggetti da prendere o degli ostacoli da evitare, e può in questo modo compensare con il proprio intervento eventuali imprecisioni del robot.

- Il sistema deve fornire le funzionalità tali da consentire all'utente di operare in modo autonomo per un lungo periodo di tempo. L'interfaccia per l'utilizzo deve essere semplice e flessibile per adattarsi alle esigenze delle diverse tipologie di utenti.

Alcune delle caratteristiche che consentono al *Manus* di essere utilizzato facilmente ed in piena sicurezza a stretto contatto con gli individui, pongono però limitazioni ad un suo utilizzo senza l'intervento di un operatore umano tramite il controllo di un calcolatore. Vedremo nel dettaglio quali sono queste limitazioni nei prossimi paragrafi.

3.1 Caratteristiche meccaniche

Il *Manus* è un manipolatore antropomorfo non ridondante a 6 gradi di libertà, più un settimo grado rappresentato dal *gripper* 3.1. A questi sette gradi di libertà è possibile aggiungerne un ottavo rappresentato da un supporto mobile in grado di variare l'altezza della base del manipolatore. Come discusso nel paragrafo A.1 i sei gradi di libertà consentono al *Manus* di poter posizionare con orientazione arbitraria il gripper in un punto qualsiasi all'interno del proprio spazio di lavoro, che è rappresentato approssimativamente da una sfera di raggio di 80 cm .

Dal momento che il *Manus* è stato concepito per essere collocato su una sedia a rotelle, un'attenzione particolare in fase di progettazione è stata rivolta al problema degli ingombri e dei pesi, in modo da non incidere eccessivamente sulla mobilità dell'utente: in quest'ottica il manipolatore è dotato di una procedura automatica di chiusura su se stesso per i momenti in cui non è utilizzato, a cui si affianca una procedura che provvede a riportarlo in posizione operativa. Per quanto riguarda i pesi, i materiali impiegati, il dimensionamento dei motori elettrici e le tecniche utilizzate per la trasmissione del moto dagli attuatori ai giunti, hanno consentito di limitare a 13 Kg la massa dell'intero manipolatore, fissando però allo stesso tempo a soli 1,5 Kg il peso utile massimo trasportabile dal gripper.

Una delle particolarità costruttive che differenzia il *Manus* dalla maggioranza

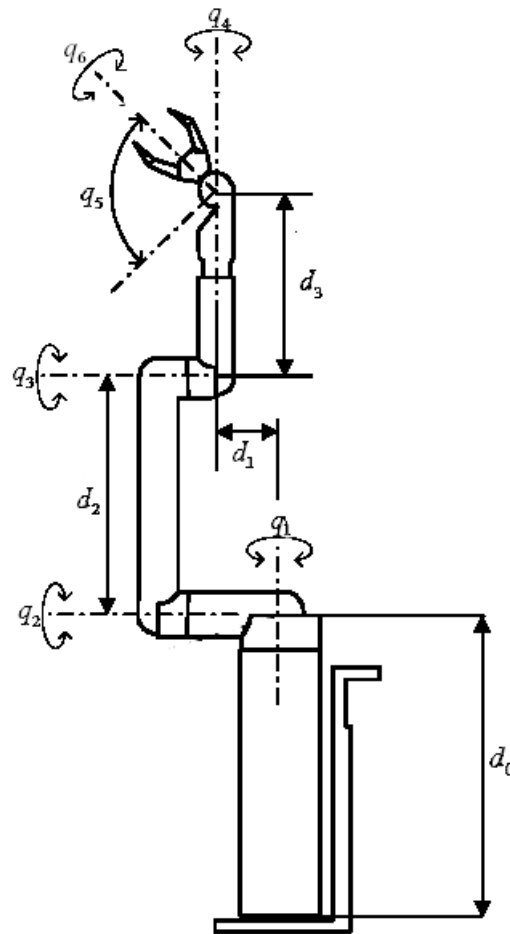


Figura 3.1: Struttura meccanica del Manus.

degli altri manipolatori antropomorfi è il fatto che tutti gli attuatori sono collocati nella base verticale, che viene spostata dal primo giunto della catena cinematica. Tale scelta è stata fatta con l'intento di ridurre il peso complessivo della struttura: il posizionamento degli attuatori nella base ed il conseguente alleggerimento degli altri bracci, garantisce infatti la possibilità di azionare il manipolatore con una minor richiesta di potenza, e quindi tramite l'impiego di motori elettrici più piccoli e più leggeri che consentono consumi energetici inferiori e una maggior durata delle batterie che alimentano il sistema. Un ulteriore vantaggio derivante dalla posizione degli attuatori è la totale assenza di cablaggi all'interno dei bracci, che offre la possibilità di far compiere ai giunti (ad eccezione del quinto) più di un giro completo

attorno al proprio asse di rotazione.

La trasmissione del moto dagli attuatori ai sei giunti ed al gripper avviene tramite cinghie collegate direttamente, o connesse fra loro in modo da poter raggiungere i giunti più distanti dalla base. Tale soluzione, oltre che garantire, come già visto, una diminuzione dei pesi delle parti mobili del manipolatore, fornisce un primo meccanismo di sicurezza, consentendo di limitare tramite lo slittamento delle cinghie la massima forza di rotazione applicata su ciascun giunto.

L'adozione di cinghie comporta però l'introduzione di giochi e non linearità nella catena di trasmissione del moto, che soprattutto per i giunti più lontani dalla base (giunti 4, 5 e 6) limitano la precisione di controllo. A questo va aggiunto che le informazioni disponibili sui valori degli angoli di rotazione dei giunti, sono quelle fornite dagli encoder posti sugli alberi dei motori, che non necessariamente coincidono con i valori reali a causa delle imprecisioni introdotte proprio dalle cinghie.



Figura 3.2: Gripper.

All'estremità opposta del manipolatore rispetto alla base, si trova il *gripper* (Figura 3.2), che consente al manipolatore di afferrare oggetti ed interagire attivamente con l'ambiente circostante. Le due estremità sono state progettate in modo da allottarsi ed avvicinarsi rimanendo sempre parallele fra loro, così da garantire una presa costante. L'apertura e la chiusura del gripper sono realizzate da un attuttore collo-

cato come tutti gli altri nella base del manipolatore, e collegato tramite cinghia, che consente un'apertura massima del gripper di 9 cm.

3.2 Cinematica

Dal punto di vista cinematico il Manus presenta una configurazione non ridondante a sei gradi di libertà (escluso il gripper) conferiti da altrettanti giunti rotoidali. Gli ultimi tre giunti sono disposti in modo tale da formare un polso sferico, che come indicato nel paragrafo A.4.3 consente la risoluzione del problema cinematico inverso in forma chiusa tramite disaccoppiamento. In realtà i sei giunti rotoidali che formano la catena cinematica aperta del Manus, non sono esattamente indipendenti. Esiste infatti una relazione fra il moto del secondo e del terzo giunto: ad una rotazione di un angolo $\Delta\theta_2$ gradi da parte del secondo giunto, il sistema fa compiere una rotazione di $-\Delta\theta_2$ al terzo giunto, in modo da mantenere costante la direzione dell'asse del terzo braccio. Ai fini del controllo è possibile compensare questa dipendenza andando a sommare al valore dell'angolo di rotazione restituito dell'encoder del terzo giunto, il valore dell'angolo del secondo giunto, in modo da poter considerare a tutti gli effetti indipendenti i sei giunti.

$$\theta_3 = \theta_3 + \theta_2 \quad (3.1)$$

Cinematica diretta

In figura 3.3 è rappresentato un modello del Manus completo delle terne solidali a ciascun braccio, posizionate secondo la convenzione di Denavit-Hartenberg. Seguendo la procedura descritta al paragrafo A.3.1 si possono determinare i parametri cinematici del manipolatore elencati nelle tabelle 3.1 e 3.2, dove le lunghezze sono espresse in mm e gli angoli in gradi.

Tramite l'equazione A.21 è ora possibile calcolare per ciascuna coppia di bracci adiacenti, la trasformazione geometrica che lega la terna i con la $i - 1$.

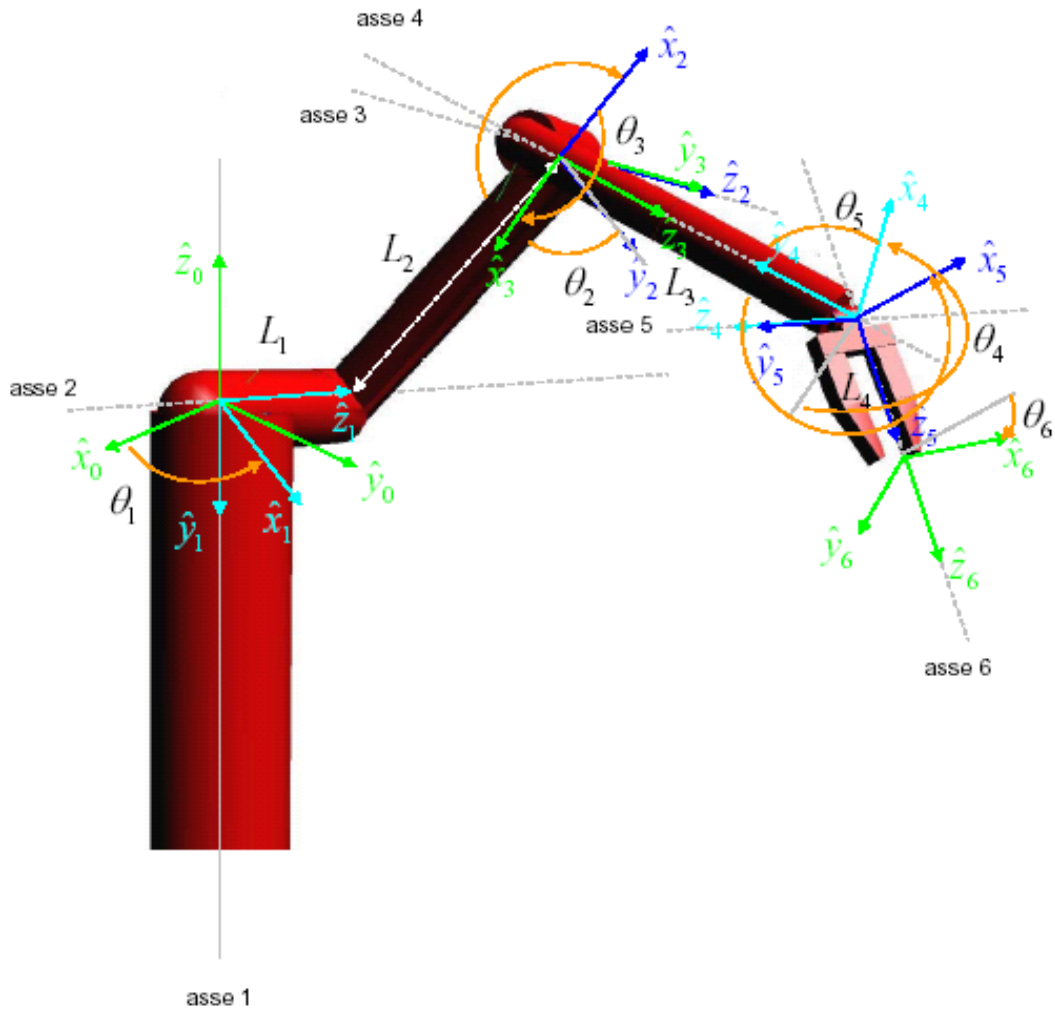


Figura 3.3: Terne di riferimento del *Manus* secondo la convezione di Denavit-Hartenberg

L_1	105
L_2	400
L_3	320
L_4	160

Tabella 3.1: Lunghezze del Manus

i	a_i	α_i	d_i	θ_i
1	0	-90	0	θ_1
2	L_2	0	L_1	θ_2
3	0	90	0	θ_3
4	0	-90	L_3	θ_4
5	0	90	0	θ_5
6	0	0	L_4	θ_6

Tabella 3.2: Parametri cinematici di Denavit-Hartenberg

$$\mathbf{A}_1^0 = \begin{bmatrix} c1 & 0 & -s1 & 0 \\ s1 & 0 & c1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{A}_2^1 = \begin{bmatrix} c2 & -s2 & 0 & L_2c2 \\ s2 & c2 & c1 & L_2s2 \\ 0 & 0 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.2)$$

$$\mathbf{A}_3^2 = \begin{bmatrix} c3 & 0 & s3 & 0 \\ s3 & 0 & -c3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{A}_4^3 = \begin{bmatrix} c4 & -s4 & 0 & 0 \\ s4 & c4 & c1 & 0 \\ 0 & 0 & 0 & L_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.3)$$

$$\mathbf{A}_4^5 = \begin{bmatrix} c5 & 0 & s5 & 0 \\ s5 & 0 & -c5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{A}_5^6 = \begin{bmatrix} c6 & -s6 & 0 & 0 \\ s6 & c6 & 0 & 0 \\ 0 & 0 & 0 & L_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.4)$$

L'espressione della trasformazione di coordinate complessiva del sistema di riferimento del gripper rispetto a quello di base si ottiene per l'equazione A.18 tramite moltiplicazione delle espressioni 3.2, 3.3 e 3.5.

$$\mathbf{T}_0^6 = \mathbf{A}_0^1 \mathbf{A}_1^2 \mathbf{A}_2^3 \mathbf{A}_3^4 \mathbf{A}_4^5 \mathbf{A}_5^6 = \begin{bmatrix} T_{11} & T_{12} & T_{13} & T_{14} \\ T_{21} & T_{22} & T_{23} & T_{24} \\ T_{31} & T_{32} & T_{33} & T_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.5)$$

$$\begin{aligned} T_{11} &= ((c_1c_2c_3c_4 - s_1s_4)c_5 - c_1s_2c_3s_5)c_6 - (c_1c_2c_3s_4 + s_1c_4)s_6 \\ T_{12} &= -((c_1c_2c_3c_4 - s_1s_4)c_5 - c_1s_2c_3s_5)s_6 - c_1s_2c_3s_5s_6 - (c_1c_2c_3s_4 + s_1c_4)c_6 \\ T_{13} &= (c_1c_2c_3c_4 - s_1s_4)s_5 + c_1s_2c_3c_5 \\ T_{14} &= ((c_1c_2c_3c_4 - s_1s_4)s_5 + c_1s_2c_3c_5)L_4 + c_1s_2c_3L_3 + c_1c_2L_2 - s_1L_1 \\ T_{21} &= ((s_1c_2c_3c_4 + c_1s_4)c_5 - s_1s_2c_3s_5)c_6 - (s_1c_2c_3s_4 - c_1c_4)s_6 \\ T_{22} &= -((s_1c_2c_3c_4 + c_1s_4)c_5 - s_1s_2c_3s_5)s_6 - (s_1s_2c_3s_4 - c_1c_4)c_6 \\ T_{23} &= (s_1c_2c_3c_4 + c_1s_4)s_5 + s_1s_2c_3c_5 \\ T_{24} &= ((s_1s_2c_3c_4 + c_1s_4)s_5 + s_1s_2c_3c_5)L_4 + s_1s_2c_3L_3 + s_1c_2L_2 + c_1L_1 \\ T_{31} &= -(s_2c_3c_4c_5 + c_2c_3s_5)c_6 + s_2c_3s_4s_6 \\ T_{32} &= (s_2c_3c_4c_5 + c_2c_3s_5)s_6 + s_2c_3s_4c_6 \\ T_{33} &= c_2c_3c_5 - s_2c_3c_4s_5 \\ T_{34} &= (c_2c_3c_5 - s_2c_3c_4s_5)L_4 - c_2c_3L_3 - l_2s_2 \end{aligned}$$

Cinematica inversa

Come è stato precedentemente anticipato all'inizio del paragrafo, il Manus è dotato di un polso sferico, e per tale motivo è possibile risolvere il problema cinematico inverso in forma chiusa, scomponendolo in due sottoproblemi più semplici: dapprima si determina la posizione della terna di polso e poi l'orientazione di quest'ultimo. Vediamo ora l'espressione della soluzione. Per una trattazione più completa sul metodo risolutivo si rimanda al paragrafo [A.4.3](#).

Sia $[x_w, y_w, z_w]$ il vettore posizione della terna di polso; le soluzioni per i primi tre giunti sono:

$$\theta_1 = 2\text{Atan} \left(\frac{x_w \pm \sqrt{x_w^2 + y_w^2 - L_1^2}}{y_w + L_1} \right) \quad x_w^2 + y_w^2 \geq L_1^2 \quad (3.6)$$

$$\theta_3 = \arcsin \left(\frac{x_w^2 + y_w^2 + z_w^2 - L_1^2 - L_2^2 - l_3^2}{2L_2L_3} \right)$$

$$\theta_3 = \pi - \arcsin \left(\frac{x_w^2 + y_w^2 + z_w^2 - L_1^2 - L_2^2 - l_3^2}{2L_2L_3} \right)$$

$$L_1^2 + (L_2 - L_3)^2 \leq x_w^2 + y_w^2 + z_w^2 \leq L_1^2 + (L_2 + L_3)^2 \quad (3.7)$$

$$\theta_2 = \text{Atan2}(c3L_3(c1x_w + s1y_w) - z_w(s3L_3 + L_2), z_w c3L_3 + (s3L_3 + L_2)(c1x_w + s1y_w)) \quad (3.8)$$

Con i valori dei primi tre giunti appena calcolati, tramite il procedimento indicato in A.4.3 si può ricavare la matrice di rotazione $R_6^3(\theta_4, \theta_5, \theta_6)$, e da qui determinare l'espressione delle tre variabili di giunto mancanti.

$$\mathbf{R}_6^3 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} c4c5c6 - s4s6 & -c4c5s6 & c4s5 \\ s4c5c6 & -s4c5s6 + c4c6 & s4c5 \\ -s5c6 & s5s6 & c5 \end{bmatrix} \quad (3.9)$$

$$\theta_4 = \text{Atan2}(r_{23}, r_{13}) \quad \theta_4 = \text{Atan2}(-r_{23}, -r_{13}) \quad (3.10)$$

$$\theta_5 = \text{Atan2} \left(\pm \sqrt{r_{13}^2 + r_{23}^2}, r_{33} \right) \quad (3.11)$$

$$\theta_6 = \text{Atan2}(r_{32}, -r_{31}) \quad \theta_6 = \text{Atan2}(-r_{32}, r_{31}) \quad (3.12)$$

3.3 Elettronica di controllo

Oltre al manipolatore vero e proprio, il sistema il Manus è dotato di una interfaccia elettronica che si occupa dei seguenti compiti:

- Accetta in ingresso i comandi impartiti dall'utente tramite tastierino numerico o joystick, o quelli provenienti da un dispositivo esterno.
- Genera sulla base dei valori di moto in ingresso i corrispondenti valori delle coppie da inviare ai motori elettrici del manipolatore.
- Segnala all'utente tramite display lo stato di funzionamento del sistema e l'eventuale presenza di errori.
- Consente il passaggio dalla modalità di funzionamento in *keypad mode*, in cui i comandi sono impartiti manualmente dall'utente, alla modalità in *transparent mode* in cui i comandi sono inviati al manipolatore da un dispositivo esterno tramite comunicazione su *bus CAN*.
- Gestisce le procedure automatiche di chiusura ed apertura del manipolatore (*Fold In/Fold Out*).

Il nucleo del sistema di controllo è costituito da tre processori, organizzati secondo lo schema di figura 5.1

Il processore 80C552 si occupa della gestione della comunicazione tramite *CAN bus* tra il controllore e l'esterno. I dati che questo processore riceve contengono le informazioni per muovere il manipolatore e i comandi per eseguire le procedure automatiche di *Fold In* e *Fold Out*. La lettura dei dati sul *CAN bus* avviene periodicamente ad intervalli di 20 ms. Il moto del manipolatore può essere controllato in due differenti modalità:

- *Joint Mode*: in cui è possibile controllare singolarmente la velocità di ciascun giunto.
- *Cartesian Mode*: in cui si può controllare la posizione del gripper lungo i tre assi cartesiani x , y , z ed la sua orientazione nelle tre direzioni *yaw*, *roll*, *pitch*.

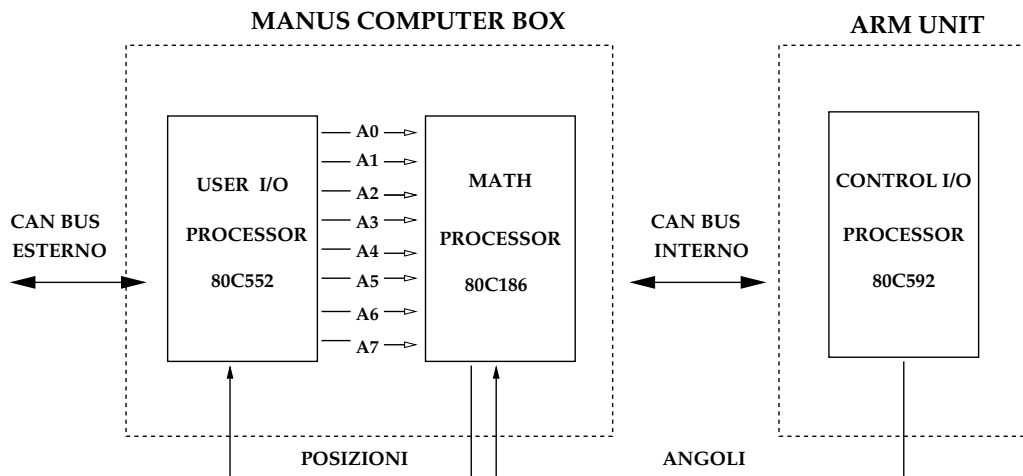


Figura 3.4: Schema dell'elettronica di controllo.

Nel caso della modalità di funzionamento nello spazio dei giunti, le informazioni in ingresso forniscono il valore in gradi di quanto debba essere ruotato ciascun giunto nei successivi 20 ms. Nel caso invece di funzionamento in modalità cartesiana i dati in ingresso specificano al sistema di quanti mm debba essere mosso il gripper lungo ciascun dei tre assi coordinati nei successivi 20 ms e di quanti gradi debba essere ruotato il polso attorno ai tre assi di rotazione. In entrambe le modalità di funzionamento lo spostamento da compiere nei 20 ms viene realizzato a velocità costante da parte del controllore interno, di fatto quindi i dati processati dall'80C552 rappresentano delle informazioni di velocità. A questo punto i dati di ingresso sono inviati al processore matematico 80C186. Nel caso in cui si stia operando in modalità cartesiana, il processore matematico applica la cinematica inversa ai valori ricevuti, controlla il verificarsi di possibili autocollisioni, e poi utilizza questi valori assieme a quelli sugli angoli dei giunti provenienti dal manipolatore per calcolare ogni 10 ms tramite un controllo di tipo proporzionale-integrativo (PI) il valore delle coppie da applicare ai motori. Questi valori di coppia vengono infine inviati tramite un secondo CAN bus (*Internal Can Bus*) al processore 80C592 che si occupa della loro trasmissione agli attuatori.

Completa la dotazione elettronica un controller CAN installato su scheda con interfaccia ISA a 16 bit da utilizzare in caso di funzionamento in *transparent mode*



Figura 3.5: Controller Can su bus ISA.

per la comunicazione tra il manipolatore ed un calcolatore. La scheda, per l'implementazione del protocollo *CAN* si avvale dell'integrato 82C200 prodotto dalla *Philips Semiconductors*. Si tratta di un controller molto semplice che non prevede alcun utilizzo di interrupt e non è provvisto di driver per il suo uso. L'invio e la ricezione dei dati si realizza infatti accedendo direttamente ai registri interni [9] che si suddividono in quattro gruppi:

- Registri di controllo: sono utilizzati per inizializzare i parametri di configurazione della comunicazione come ad esempio il bitrate, la frequenza degli oscillatori o la maschera per il filtraggio dei messaggi in ingresso al dispositivo.
- Registri di comando: la scrittura su questi registri consente l'esecuzione di una data operazione, quali ad esempio il rilascio dei buffer di trasmissione\ricezione, l'immediata sospensione di una trasmissione, o il reset del controller.
- Registri di stato: i loro valori forniscono informazioni sullo stato del sistema quali la disponibilità di un messaggio nel buffer di ricezione, l'avven-

to completamento di una trasmissione, o ancora il verificarsi di un qualche errore.

- Buffer di trasmissione\ricezione: sono le aree di memoria dove risiedono i dati prima del loro invio o immediatamente dopo la loro ricezione.

3.4 Protocollo di comunicazione

Come indicato nel paragrafo 3.3 il sistema di controllo comunica tramite il calcolatore attraverso il protocollo CAN: prima di vedere in dettaglio in che modo avviene lo scambio di dati, si analizzano le caratteristiche generali di questo protocollo.

3.4.1 Il protocollo CAN BUS

Il *Controller Area Network (CAN)* è un protocollo di comunicazione seriale sviluppato nei primi anni ottanta dalla Bosch [10] per la realizzazione sistemi di controllo distribuiti ed in tempo reale. Nel 1993 è divenuto uno standard internazionale ISO ed oggi è ampiamente utilizzato nel settore industriale, ed in particolare in campo automobilistico per la connessione dei dispositivi che presiedono al controllo dei veicoli.

Il protocollo consente la realizzazione di reti con gerarchie di tipo *multimaster*, in cui ciascun dispositivo connesso all'unico *bus* può assumere il controllo del canale di comunicazione e trasmettere i propri dati secondo un modello di trasmissione *broadcast orientata al messaggio*. Un modello di questo tipo prevede che siano definiti i contenuti dei messaggi invece che i nodi e gli indirizzi dei nodi che compongono la rete. In quest'ottica ciascun messaggio è etichettato da un identificatore univoco che descrive il significato dei dati contenuti, e fissa una priorità statica necessaria a risolvere le situazioni di contesa del bus da parte di due o più nodi trasmettenti. Il modello di tipo broadcast comporta che ciascun dispositivo connesso alla rete è in grado di ricevere tutti i messaggi inviati sul bus: è compito dei singoli nodi decidere sulla base dell'identificatore se il messaggio è di propria competenza o vada invece ignorato. Il risultato di tali scelte ha portato ad una architettura che permette la progettazione di sistemi modulari ed altamente configurabili. L'aggiunta,

ad esempio, alla rete di dispositivi riceventi non comporta alcuna modifica software ed hardware ai dispositivi esistenti, mentre la trasmissione in modalità *broadcast* consente la ricezione multipla dei dati e la sincronizzazione di processi distribuiti.

Struttura dei messaggi

L'intero meccanismo di trasferimento dati del protocollo CAN si basa su 4 tipi di messaggi con struttura fissa a lunghezza variabile ma comunque limitata:

- *messaggi dati*: sono utilizzati per la trasmissione di dati fra due o più nodi della rete.
- *messaggi remoti*: sono utilizzati per la richiesta di trasmissione di un messaggio dati con lo stesso identificatore del messaggio remoto trasmesso.
- *messaggi di errore*: sono trasmessi dai dispositivi della rete al rilevamento di un errore sul bus.
- *messaggi di ritardo*: introducono un ritardo tra un messaggio di richiesta remota ed un messaggio dati a fini di sincronizzazione.

S		R	I				A	
O	ID	T	D	DLC	DATA	CRC	C	EOF
F		R	E				K	

Figura 3.6: Struttura di un *messaggio dati* nel protocollo CAN.

Vediamo in dettaglio la struttura di un messaggio dati schematizzata in figura 3.6.

SOF (Start Of Frame) è un singolo bit che contrassegna l'inizio del messaggio; dal momento che un dispositivo può trasmettere solamente in assenza di attività sul bus, il bit SOF svolge anche un ruolo di sincronizzazione fra il nodo che inizia una trasmissione e gli altri nodi della rete.

ID (Identifier) é il codice di identificazione che consente di differenziare i messaggi in base al contenuto; consente inoltre di fissare una gerarchia di priorità statiche dei messaggi da utilizzare in caso di conflitto durante la fase di trasmissione: ai valori di identificazione più bassi corrispondono priorità più elevate. Il campo *ID* è di 11 bit per i messaggi in formato base, 29 per quelli in formato esteso introdotti con la specifiche 2.0 del protocollo.

RTR è il bit per la richiesta di trasmissione remota che consente di distinguere un messaggio di trasmissione dati da uno di richiesta ad un altro nodo della rete.

IDE (Identifier Extension) è il bit utilizzato per distinguere i messaggi in formato base da quelli estesi.

DLC (Data Legth Code) campo formato da 6 bit dei quali 4 sono utilizzati per indicare il numero di byte di dati trasmessi o richiesti dal messaggio, 2 bit sono invece riservati per usi futuri.

DATA campo contenente i dati che il messaggio sta trasportando, il limite massimo è di 8 byte.

CRC (Cyclic Redundant Check) campo contenente una sequenza di controllo calcolata per verificare l'integrità dei dati ricevuti.

ACK campo di due bit utilizzati dal dispositivo trasmittente e ricevente per la conferma di una corretta ricezione di un messaggio.

EOF (End Of Frame) campo di 7 bit che segnala la fine del messaggio.

Rilevazione e segnalazione degli errori

Il protocollo CAN prevede che eventuali errori nella trasmissione siano segnalati non appena ne venga rilevata la presenza.

A livello di messaggio sono disponibili tre meccanismi per l'individuazione e la segnalazione degli errori:

- *Sequenza CRC*: il dispositivo trasmittente calcola la *sequenza CRC* e la spedisce assieme al messaggio. Il nodo ricevente ricalcola la sequenza e la con-

fronta con quella ricevuta: se le due sequenze non coincidono viene segnalato che si è verificato un errore nella fase di trasmissione.

- *Controllo del formato*: ogni dispositivo prima di inviare un messaggio sul bus verifica che la struttura dei campi sia conforme per formato e dimensione al modello previsto dal protocollo.
- *Segnale di riconoscimento*: il nodo che riceve un messaggio di propria competenza deve informare il dispositivo trasmittente dell'avvenuta consegna. In caso di mancata segnalazione il nodo trasmittente segnala la presenza di un errore.

Ai tre meccanismi appena visti se ne aggiungono due operanti a livello di bit:

- Parallelamente alla trasmissione dei dati, ciascun dispositivo esegue un'analisi del segnale del bus in modo da poter rilevare la differenza fra i bit spediti e quelli ricevuti; in questo modo si possono inoltre distinguere gli errori causati dal dispositivo da quelli introdotti dal resto della rete.
- Durante la codifica dei campi SOF, ID, DATA e CRC ad ogni cinque bit consecutivi identici viene accodato un bit di valore complementare. Il dispositivo ricevente verifica che non esistano sequenze di sei bit uguali e consecutivi: in caso di responso positivo viene rimosso il sesto bit e si procede con la decodifica dei dati, in caso contrario viene generato un messaggio di errore.

Nel momento in cui viene rilevata una condizione di errore, il nodo interessato annulla la trasmissione corrente ed invia un messaggio di errore evitando che gli altri nodi ricevano dati corrotti, mantenendo così il sistema in uno stato consistente; subito dopo il dispositivo ripete automaticamente la trasmissione che aveva provocato l'errore.

Livello fisico

Le specifiche del protocollo *CAN* non si limitano al solo livello *data link*, ma definiscono in parte anche quelle che sono le caratteristiche del livello fisico del modello OSI.

A livello di segnale i singoli bit vengono rappresentati tramite una codifica del tipo *NRZ* (*Non-Return-to-Zero*), in cui il livello del segnale rimane costante per l'intera durata del periodo di bit.

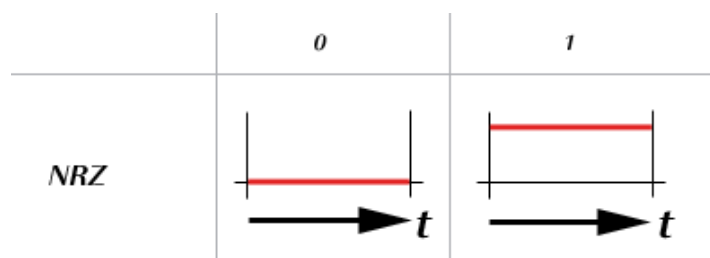


Figura 3.7: Codifica NRZ.

A livello di bit la trasmissione avviene in modalità sincrona: se da un lato questo consente di incrementare le capacità trasmissive, dall'altro rende necessario l'uso di tecniche per mantenere la sincronizzazione. A tale scopo ciascun periodo di bit può essere pensato come la composizione di quattro intervalli temporali non sovrapposti (figura 3.8).

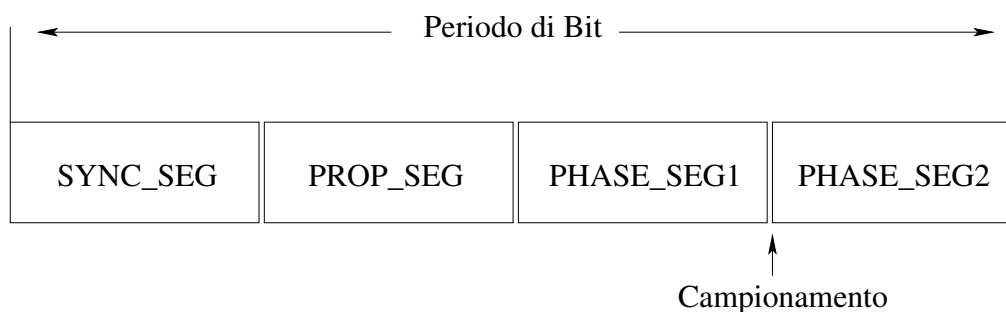


Figura 3.8: Scomposizione del periodo di bit.

- **SYNC_SEG:** è il segmento temporale del periodo di bit utilizzato per sincronizzare i nodi della rete; la sua durata è pari ad un periodo del segnale di clock della rete.
- **PROP_SEG:** questo segmento è utilizzato per compensare il ritardo del segnale sul mezzo fisico; il suo valore è pari al doppio della somma del tempo di propagazione sulla rete, e dei tempi di trasmissione e ricezione dei dispositivi.

- PHASE_SEG1, PHASE_SEG2: questi due segmenti sono utilizzati per compensare eventuali ritardi o anticipi di fase rispetto al segnale di clock della rete. I loro valori sono programmabili e vengono variati in fase di risincronizzazione.

Al termine del segmento *PHASE_SEG1* avviene la lettura del livello del segnale sul bus e l'interpretazione del valore del bit corrispondente.

Esistono due modalità di sincronizzazione:

- *Sincronizzazione forzata*: è realizzata quando in assenza di attività sul bus un nodo riceve il bit *SOF* di un messaggio appena inviato da un altro dispositivo. In seguito a tale ricezione il nodo inizializza un nuovo periodo di bit in modo che il segnale che ha indotto la sincronizzazione si venga a trovare all'interno del segmento *SYNC_SEG* del nuovo periodo di bit.
- *Risincronizzazione*: è realizzata nella fase di ricezione dei messaggi variando il valore dei segmenti *SYNC_SEG1*, *SYNC_SEG* in modo da variare l'istante di campionamento in accordo al messaggio trasmesso.

Dal momento che il periodo di bit dipende dal tempo di propagazione del segnale sul mezzo fisico, la massima frequenza di trasmissione su una rete è strettamente correlata alla lunghezza del bus: quanto più è grande la distanza massima fra due nodi, tanto minore è la frequenza massima di trasmissione utilizzabile. I due casi estremi previsti dal protocollo prevedono un bit rate massimo di 1 Mbps su bus da 40 metri, per arrivare a bus lunghi 1000 metri con velocità di trasmissione di 40 Kbps.

3.4.2 Comunicazioni con il manipolatore

La comunicazione tra manipolatore e calcolatore avviene tramite il CAN bus esterno di figura 5.1. Il processore 80C552 invia al controller CAN posto su calcolatore un messaggio ogni 20 ms. Il ciclo di funzionamento prevede l'invio di tre messaggi: i primi due portano con sé delle informazioni, il terzo si limita a svolgere un ruolo di sincronizzazione, segnalando al controller che dal quel momento può inviare un

Stato	Tipo di messaggio	Codice errore	Descrizione
0	Nessun Messaggio		
1	Warning	0	Collisione del gripper.
		1	Area di lavoro sbagliata.
		2	Manipolatore completamente esteso.
		3	Motore bloccato.
1	Messaggio Generale	0	Manipolatore aperto.
		1	Manipolatore chiuso.
		2	Gripper pronto.
2	Errore	0	errore
		1	Errore I/O 80C552
		2	errore
		3	errore
		4	errore encoder assoluto
		6	errore
		7	errore
		8	errore
		11	errore
		12	errore
		13	errore
		13	Movimento senza input

Tabella 3.3: Messaggi di errore inviati dal manipolatore.

ID	RTR	DLC
0X350	0	8
0X360	0	8
0x37F	1	0

Tabella 3.4: Messaggi spediti dal processore 80C552.

messaggio di risposta, che nel caso in cui venga spedito deve giungere a destinazione entro i successivi 20 ms. Il controllore esterno può quindi inviare un messaggio ogni 60 ms, trovandosi così ad operare ad una frequenza di circa 16 Hz, inferiore alla frequenza tipica di controllo dei manipolatori in commercio, che spazia tipicamente nell'intervallo 20-200 Hz. Qualora la risposta giunga in ritardo o non venga inviata affatto, l'integrato 80C552 replica al processore matematico l'ultimo comando utile ricevuto.

I primi due messaggi, inviati rispettivamente dopo 20 e 40 ms dall'inizio del ciclo di trasmissione, contengono informazioni sullo stato del sistema, e sulla posizione del manipolatore. La tabella 3.5 mostra nel dettaglio come sono organizzate queste informazioni all'interno dei campi dati dei due messaggi per entrambe le modalità di funzionamento: quella nello spazio cartesiano e quella nello spazio dei giunti.

I primi due byte del messaggio 0x350 contengono informazioni sullo stato di funzionamento del manipolatore, e sull'eventuale verificarsi di situazioni anomale, che vengono riportate tramite codici di errore ed avvertimento secondo quanto riportato in tabella 3.3.

Sono disponibili più messaggi di risposta (Tabella 3.6), ciascuno caratterizzato da un proprio identificatore univoco che consente al processore matematico di interpretarne correttamente il contenuto e di selezionare l'azione da eseguire. Vediamo una breve descrizione dei diversi messaggi:

- ID 0x370: comporta la sospensione immediata di qualsiasi attività in esecuzione, e pone il manipolatore in attesa di un nuovo comando.
- ID 0x376: avvia la procedura automatica di chiusura del manipolatore (*Fold*

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

Tabella 3.5: Organizzazione dei dati nei primi due messaggi.

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

Tabella 3.6: Elenco dei possibili messaggi di risposta.

Byte	Parametro	Incremento	Min Incremento	Max Incremento
1	Lift Unit	-1 0 +1	-1	1
2	Giunto 1	0.1 (gradi)	-10	10
3	Giunto 2			
4	Giunto 3			
5	Giunto 4	0.1 gradi	-10	10
6	Giunto 5			
7	Giunto 6			
8	Gripper	0.1 (mm)	0	15

Tabella 3.7: Range degli incrementi utilizzabili nello spazio dei giunti.

In).

- ID 0x375: avvia la procedura di *Fold Out*, che porta il manipolatore dalla posizione di *Fold In* ad una posizione operativa.
- ID 0x371: contiene le informazioni sul moto da far compiere al manipolatore specificate nello spazio cartesiano.
- ID 0x374: contiene le informazioni sul moto da far compiere al manipolatore specificate nello spazio dei giunti.

Il campo dati dei messaggi con identificatori 0x370, 0x375 e 0x376 è vuoto; per quanto riguarda invece i messaggi 0x371 e 0x374, i loro campi dati contengono informazione su quanto e come debba essere modificata la posizione del manipolatore ad intervalli di 20 ms. Le unità di misura con cui vengono espressi questi incrementi spaziali, la loro organizzazione all'interno del campo dati dei messaggi, ed il range dei valori utilizzabili sono rispettivamente rappresentati in tabella 3.7 per il funzionamento nello spazio dei giunti, e in tabella 3.8 per quello nello spazio cartesiano. Com'è già stato evidenziato in precedenza questi incrementi rappresentano di fatto delle velocità: nel caso in cui si stia operando nello spazio dei giunti il controllo del manipolatore avviene quindi indicando ad ogni giunto a quale velocità, fra le dieci disponibili per ciascuno dei due sensi di rotazione, si deve muovere nei successivi 60 ms.

Byte	Parametro	Incremento	Min Incremento	Max Incremento
1	Lift Unit	-1 0 +1	-1	1
2	X	0.022 (mm)	0	127
3	Y			
4	Z			
5	Yaw	0.1 gradi	0	10
6	Pitch			
7	Roll			
8	Gripper	0.1 (mm)	0	15

Tabella 3.8: Range degli incrementi utilizzabili nello spazio cartesiano.

Capitolo 4

Realizzazione del sistema

4.1 Obiettivi e requisiti del sistema

L'obiettivo di questo lavoro di tesi è quello di sviluppare un sistema di controllo software per il manipolatore *Manus* che consenta la realizzazione di compiti di manipolazione, che possono spaziare dal semplice movimento del manipolatore nello spazio libero, al trasporto di oggetti, e all'esecuzione di task che prevedono un'interazione complessa tra robot ed ambiente circostante.

La realizzazione del sistema deve avvenire nel rispetto dei seguenti requisiti:

- Vincoli *real time*: nella fase di comunicazione con il manipolatore, il sistema deve garantire il rispetto delle tempistiche imposte dall'elettronica di controllo (paragrafo 3.4.2), per non compromettere lo scambio di dati e quindi l'azione di controllo.
- L'architettura del sistema deve essere quanto più modulare possibile, in modo da favorirne lo sviluppo e gli aggiornamenti futuri, alcuni dei quali già previsti, come l'integrazione al complesso base mobile + manipolatore di un sistema di visione stereoscopico, per l'individuazione degli obiettivi e degli ostacoli presenti nello spazio di lavoro del manipolatore.
- Sempre nell'ottica di modularizzazione dell'architettura, ed al fine di sfruttare al meglio le caratteristiche di *multitasking* del sistema operativo Linux, i

principali task del sistema, comunicazione con il manipolatore, calcolo delle traiettorie e controllo del moto, dovrebbero essere realizzati come *thread* autonome.

Sulla base di quanto è appena stato esposto, si è deciso di utilizzare le funzionalità messe a disposizione dalla libreria *YARA* per la realizzazione della parte di basso livello del sistema.

Yara è un framework sviluppato presso il Dipartimento di Ingegneria dell'Informazione dell'università di Parma, con l'obiettivo di fornire agli sviluppatori gli strumenti per la realizzazione di applicazioni nell'ambito della robotica mobile. Questi strumenti comprendono meccanismi avanzati per lo *scheduling real time* dei processi e primitive di comunicazione ad alto livello per lo scambio di dati in grado di garantire la trasparenza rispetto ai meccanismi di implementazione utilizzati.

4.1.1 YARA

In questo paragrafo sono illustrate le principali caratteristiche di *YARA* che è indispensabile conoscere per poter comprendere le dinamiche di funzionamento del sistema di controllo del *Manus*; per una trattazione più completa si rimanda a [11].

Moduli

Il componente principale di *Yara* è il modulo, inteso come unità attiva autonoma, in grado di eseguire compiti elementari, rappresentare un *Behaviour* o fornire un'interfaccia verso l'hardware del robot, garantendo però allo stesso tempo la cooperazione con gli altri moduli dell'architettura mediante lo scambio di dati e comandi. I moduli sono realizzati come classi C++ derivate dalla classe astratta *Module*. Al proprio interno ciascun modulo possiede algoritmi che operano sulle strutture dati comuni, ed un certo numero di *attività* che si occupano dell'esecuzione degli algoritmi. I moduli sono identificati univocamente tramite il nome passato al costruttore, che consente al framework una loro indicizzazione e l'attivazione e disattivazione dei processi associati tramite i due metodi `turnOn()` e `turnOff()`.

Attività

Ogni modulo possiede una o più attività che si occupano dell'esecuzione degli algoritmi implementati dai metodi del modulo stesso. Le attività sono realizzate tramite istanze della classe template *Activity*, che consente di specificare il codice da eseguire, rappresentato da un metodo del modulo di tipo `void`. L'esecuzione delle attività è affidata ai meccanismi interni del framework: l'attivazione può essere periodica, in questo caso il framework provvede ad intervalli regolari a mandare in esecuzione il codice associato all'*Activity*, o può essere aperiodica, ed allora l'istante di attivazione è il risultato dell'interazione tra il modulo e gli altri componenti del sistema. All'atto della dichiarazione di una attività viene passato al costruttore un parametro che rappresenta rispettivamente il periodo di attivazione espresso in secondi per le attività periodiche, ed il tempo minimo che deve intercorrere fra due attivazioni consecutive per le attività aperiodiche. Per entrambi i tipi di attività questo parametro ha anche il significato di *deadline*, l'istante temporale entro cui si deve concludere l'esecuzione dell'attività per poter essere considerata corretta. Per quanto riguarda le attività periodiche, la loro attivazione e sospensione può essere invocata direttamente tramite i metodi `startPeriodicRun()` e `stopPeriodicRun()`; per quelle aperiodiche invece, così com'è spiegato in seguito, ci si può solamente limitare ad agganciarne l'attivazione ad uno specifico evento del sistema.

Comunicazione

La cooperazione fra i componenti del sistema si fonda sul fatto che i moduli, trovandosi nello stesso spazio di indirizzamento, possono comunicare fra loro mediante accesso concorrente a strutture dati condivise. Per evitare però allo sviluppatore le difficoltà legate all'uso esplicito della memoria condivisa e della sincronizzazione, il framework è dotato di una serie di primitive ad alto livello per la comunicazione fra i moduli che nascondono la complessità ed i dettagli utilizzati internamente per la loro realizzazione.

Il meccanismo di comunicazione utilizzato per la progettazione del controllore del *Manus* è rappresentato dal pattern *Information*, che consente ai moduli di realizzare uno scambio reciproco di dati. Per la ricezione o l'invio di un dato, il modulo deve dichiarare come propri *data members* uno o più oggetti di tipo *Receiver* e *Sen-*

der. *Receiver* e *Sender* sono classi template e permettono quindi di specificare il tipo di dato da scambiare, e ogni oggetto è dotato di un nome univoco in modo da poter essere identificato dal framework. Affinchè due moduli si possano scambiare un dato, essi si devono registrare rispettivamente come *Sender* e *Receiver* del dato in questione. Il framework stabilisce un collegamento fra i due moduli, in modo che ad ogni aggiornamento dell'oggetto *Sender*, eseguito tramite la chiamata al metodo `update()`, anche il contenuto del corrispondente oggetto *Receiver*, accessibile tramite il metodo `get()`, venga aggiornato. Il pattern *Information* consente anche comunicazioni di tipo *broadcast*: se infatti un solo modulo si può registrare come *Sender* di un dato, più moduli possono invece registrarsi come *Receiver* del dato, consentendo in questo modo l'invio contemporaneo di un dato a più moduli. Due metodi molto importanti della classe *Receiver* sono `setListener` e `clearListener`: il primo consente di legare l'esecuzione di una attività all'aggiornamento dei dati di un oggetto *Receiver*, in modo che ad ogni operazione di `update()` su un oggetto *Sender*, l'attività *listener* dell'equivalente oggetto *Receiver* venga attivata in modo aperiodico. Il secondo metodo si limita semplicemente a rimuovere il legame tra aggiornamento ed attivazione stabilito da una precedente chiamata di `setListener()`, interrompendo così l'attivazione dell'*Activity*.

Scheduler real time

L'esecuzione dinamica del codice associato alle *Activity* è realizzato dal framework *TODS (Timed Object for Distributed Systems)* [12], che consente la realizzazione dello *scheduling real time* di task periodici ed aperiodici. *TODS* opera assegnando ad ogni task una thread di sistema, che verrà quindi schedulata dallo scheduler FIFO standard di Linux. Successivamente tramite una politica di tipo EDF si occupa di calcolare dinamicamente la priorità delle thread in esecuzione. La politica di scheduling EDF consiste nello schedulare ad ogni istante il task con la deadline più vicina, ed assegnare alle thread rimanenti una priorità decrescente con l'allontanarsi delle rispettive deadline.

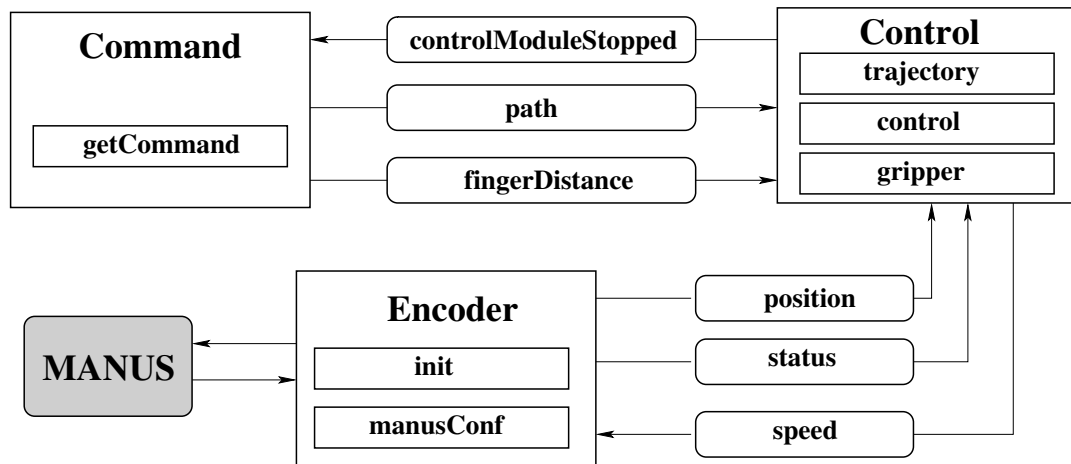


Figura 4.1: Schema del sistema di controllo.

4.2 Architettura del sistema

L'architettura del sistema di controllo, schematizzata in figura 5.1, si compone dei tre moduli *Encoder*, *Command* e *Control*, che complessivamente ricorrono a sei Activity per la realizzazione del controllo e ad altrettanti pattern *Information* per le comunicazioni. Il resto del capitolo è dedicato all'analisi dettagliata di ogni modulo e di ogni attività, agli algoritmi ed al modo in cui i componenti del sistema interagiscono tra loro.

4.2.1 Il modulo *Encoder*

Il modulo *Encoder* è il componente del sistema preposto alla comunicazione tra l'architettura software ed il manipolatore, tramite l'interfaccia hardware costituita dall'elettronica di controllo del *Manus*, e dal controller *CAN* posto sul calcolatore. Nel listato 4.1 sono mostrate le *Activity* del modulo e gli oggetti *Receiver* e *Sender* utilizzati per la comunicazione con gli altri moduli dell'architettura. Vediamo nel dettaglio il compito svolto da ciascuna delle due *Activity*.

4.2.1.1 Activity *init*

init viene attivata come *Activity* periodica all'avvio del sistema. Il suo ruolo è quello di svolgere un'azione di inizializzazione sui giunti del manipolatore. Questa inizia-

```
class EncoderModule : public Module
{
  private:
    void getManusConf();
    void initManus();

  public:
    EncoderModule(): Module("Encoder"),
    init(*this, &EncoderModule::initManus, 0.01),
    manusConf(*this, &EncoderModule::getManusConf, 0.01),
    position(*this, "position"),
    status(*this, "status"),
    speed(*this, "speed") {}

    Activity<void, void> init;
    Activity<void, void> manusConf;

    Sender<JointPosition> position;
    Sender<ManusStatus> status;

    Receiver<JointSpeed> speed;
};
```

Listato 4.1: *Activities, Senders e Receiver della Classe EncoderModule.*

lizzazione si è resa necessaria in seguito ad un'analisi dei valori delle variabili di giunto restituiti dagli encoder, che ha evidenziato una inconsistenza nei valori degli ultimi tre giunti che si manifesta durante i primi movimenti immediatamente successivi all'accensione del manipolatore. Come mostrano i dati riportati in tabella 4.1, trascorsi 240 ms dall'istante temporale t_0 in cui viene impartito il primo comando di movimento ai giunti 4, 5 e 6, il valore delle variabili di giunto subisce una netta variazione, legata probabilmente alla presenza di giochi nella catena di trasmissione del moto (Paragrafo 3.1).

Per evitare che la brusca variazione delle variabili di giunto giunga al controllo provocando la generazione di velocità di controllo errate, viene fatto compiere un

tempo	Giunto 4	Giunto 5	Giunto 6
$t_0 + 60$	179.3	-36.8	82.6
$t_0 + 120$	179.3	-36.8	82.6
$t_0 + 180$	179.3	-36.8	82.6
$t_0 + 240$	-121.8	5	121.3

Tabella 4.1: Valori delle variabili dei giunti 4,5 e 6

movimento impercettibile ai tre giunti interessati prima dell'inizio del ciclo di controllo vero e proprio. L'Activity *init*, rispettando le tempistiche di comunicazione del manipolatore introdotte nel paragrafo 3.4.2, invia per quattro volte consecutive a distanza di 60 ms un comando di movimento ai tre giunti, invertendo ogni volta il verso del moto di rotazione in modo da annullarne l'effetto complessivo. Fatto ciò, l'attivazione periodica di *init* è sospesa, e viene attivata l'Activity *manusConf*.

4.2.1.2 Activity *manusConf*

manusConf è un'Activity periodica con periodo di attivazione di 10 ms, ed è avviata al termine della fase di inizializzazione del manipolatore. Il suo compito è quello di leggere dal CAN bus le informazioni sulla configurazione attuale del manipolatore (stato del sistema e valori delle variabili di giunto), rendere disponibili queste informazioni al modulo che si occupa del controllo, ed inviare al manipolatore sempre tramite CAN bus le velocità dei giunti che si vogliono impostare.

L'operazione di ricezione della configurazione del manipolatore avviene in tre fasi, secondo il protocollo di trasmissione dei messaggi da parte dell'elettronica del manipolatore illustrate nel paragrafo 3.4.2. Dopo la ricezione dei primi due messaggi, con identificatori rispettivamente uguali a 0x350 e 0x360, *manusConf* assembla le informazioni contenute nei due campi dati e le rende disponibili al modulo *Control*. In modo particolare, così com'è indicato dalla tabella 3.5 dai primi due byte del messaggio con identificatore 0x350 vengono estratte le informazioni sullo stato del sistema che sono memorizzate nell'oggetto di tipo *Sender status*, mentre dai rimanenti byte del messaggio 0x350 e da quelli del messaggio 0x360 vengono ricavati i valori delle variabili di giunto e l'informazione sullo stato di apertura del gripper. Prima di essere passati al modulo *Control* tramite l'oggetto *Sender position*, i valori

Correzione
$\theta_2 = -\theta_2$
$\theta_3 = \theta_3 + \theta_2 + 90$
$\theta_4 = \theta_4 + 90$

Tabella 4.2: Correzione delle variabili di giunto.

delle variabili di giunto vengono modificati per compensare la dipendenza che esiste fra il secondo e terzo giunto (Paragrafo 3.2). A questa correzione se ne aggiunge una seconda per far coincidere gli zeri delle variabili di giunto restituiti dagli encoder del manipolatore con gli zeri previsti dalla convenzione di Denavit-Hartenberg. In tabella 4.2 sono riportati le espressioni delle correzioni applicate, dove i valori numerici rappresentano dei gradi.

A questo punto viene chiamato il metodo `update()` per l'oggetto *Sender position*, in modo che i nuovi valori delle variabili di giunto giungano al modulo di controllo, dove saranno utilizzati per il calcolo di un nuovo insieme di velocità dei giunti sulla base della posizione attuale del manipolatore. Il terzo messaggio previsto dal protocollo di comunicazione (ID 0x37F) non contiene alcun dato, ma svolge uno ruolo di sincronizzazione, indicando a *manusConf* che da questo istante e per i prossimi 20 ms può effettuare la trasmissione del messaggio di risposta contenente le informazioni sulle velocità dei giunti da impostare. Queste velocità sono calcolate dal modulo *Control*, e successivamente sono messe a disposizione di *manusConf* tramite il pattern *Information speed*. Dal momento che il controllo del moto è realizzato nello spazio dei giunti (paragrafo 2.2), il messaggio di risposta con le velocità prevede unicamente l'utilizzo dell'identificatore 0x374, corrispondente (tabella 3.6) ad un utilizzo del manipolatore nello spazio dei giunti.

Qualora per il verificarsi di un qualche errore, o per un ritardo del sistema, al momento della trasmissione del messaggio con le velocità, l'oggetto *speed* non contenga alcun valore, *manusConf* provvede ad inviare delle velocità dei giunti nulle in modo da fermare temporaneamente il manipolatore evitando così che si muova senza controllo.

In precedenza si è accennato al fatto che *manusConf* è un'Activity periodica con

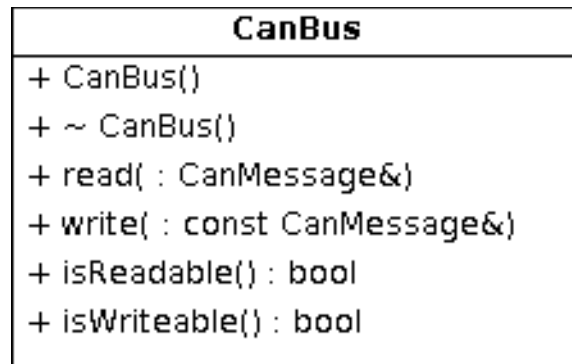


Figura 4.2: Diagramma UML della classe *CanBus*

periodo di attivazione di 10 ms, pari alla metà del periodo con cui vengono inviati i messaggi dal manipolatore verso la scheda di controllo. La mancanza di un driver per il controllo della scheda CAN e la mancata gestione delle code di messaggi da parte dell'integrato 80C200, costringono infatti *manusConf* ad una attivazione a frequenza doppia rispetto a quella necessaria in modo da evitare che un messaggio vada perso perchè sovrascritto dal successivo. Nel caso in cui comunque uno dei messaggi non giunga a destinazione, compromettendo in questo modo il recupero dei dati necessari al controllore, *manusConf*, dopo aver momentaneamente fermato il manipolatore, è in grado di risincronizzare l'attività di ricezione e ristabilire la corretta sequenza dei messaggi.

Per la comunicazione sul CAN bus, entrambe le Activity del modulo si affidano alle classi *CanBus* (figura 4.11) e *CanMessage*. La classe *CanBus* fornisce le funzioni per svolgere le seguenti azioni:

- Inizializzazione e chiusura del canale di comunicazione tra controller e manipolatore, realizzate rispettivamente dal costruttore e distruttore della classe.
- Lettura e scrittura dei messaggi tramite chiamata a delle funzioni `write()` e `read()`.
- Verifica dell'accessibilità in scrittura e lettura del bus tramite le funzioni `isReadable()` e `isWriteable()`.

Tutte le operazioni eseguite dalla classe *CanBus* sono realizzate tramite accesso

diretto ai registri di memoria dell'integrato 80C200, così come indicato nel paragrafo 3.3.

La classe *CanMessage* rappresenta invece un messaggio dati secondo il protocollo CAN, e fornisce tutte le funzioni per la creazione di un messaggio e per l'accesso ai suoi campi, organizzati secondo la struttura di figura 3.6.

Restano da definire le strutture dei due oggetti Sender *position* e *status* e quella dell'oggetto Receiver *speed*:

- *position* è un oggetto `vector<int>` di dimensione pari a otto. Il primo campo non è utilizzato ed è riservato a contenere la posizione dell'unità di sollevamento del *Manus*, non presente nell'attuale configurazione. I successivi sei campi contengono il valore in gradi della posizione dei giunti, mentre l'ottavo campo contiene la distanza in mm tra le dita del gripper.
- *status* è una struttura con due soli campi di tipo `char` contenenti le indicazioni sullo stato di funzionamento del manipolatore ed i codici numerici di eventuali messaggi di errore o avvertimento.
- *speed* è un oggetto `vector<int>` di dimensione otto, contenente le velocità di rotazione da applicare ai sei giunti e la velocità di azionamento del gripper, in aggiunta al campo riservato all'unità di sollevamento del manipolatore non utilizzato.

4.2.2 Il modulo *Command*

Il ruolo del modulo *command* è quello di inoltrare al modulo *control* le indicazioni sul *task* da eseguire, fornendogli contemporaneamente tutti i dati necessari per la realizzazione del compito.

Il tipico task di manipolazione comporta che il manipolatore prenda un oggetto, compia qualche azione con questo oggetto, o molto più semplicemente lo sposti in una nuova posizione dello spazio di lavoro, ed infine lo rilasci. In quest'ottica il sistema deve essere dotato delle due funzionalità di base indispensabili che sono:

- il controllo del moto del manipolatore lungo un percorso assegnato.

- la possibilità di aprire e chiudere il gripper.

```
class CommandModule: public Module
{
  private :
  void getComm ();

  public :
  CommandModule( vector<string> &,vector<double> &);

  Activity< void , void > getCommand;

  Sender< vector<ViaPoint> > path;
  Sender< float > fingerDistance;

  Receiver< bool > controlModuleStopped;
};
```

Listato 4.2: *Activities, Senders e Receivers* del modulo *Command*.

Il costruttore del modulo *Command* si occupa di memorizzare in due vettori distinti le sequenze dei via point dei percorsi da interpolare e le misure di apertura del gripper che devono essere tenute durante le diverse fasi del task. Entrambi i tipi di informazioni sono passate al modulo come argomenti di riga al momento dell'esecuzione del programma. Per quanto riguarda i via point, l'input è rappresentato dal nome dei file su cui sono memorizzati le diverse porzioni di traiettoria che compongono il moto complessivo; le informazioni sullo stato di apertura del gripper sono invece passate direttamente da riga di comando tramite valori numerici che rappresentano la distanza in mm tra le due dita che formano il gripper. I due tipi di argomenti in ingresso devono essere alternati fra loro, e devono essere disposti secondo l'esatta sequenza temporale con cui si devono succedere durante la realizzazione del moto.

I via point sono memorizzati su file nella forma mostrata in tabella 4.3. La posizione (x, y, z) del gripper è espressa in mm l'orientazione (Yaw, Roll, Pitch) in gradi, mentre le velocità sono espresse rispettivamente in mm/s e gradi/s. In realtà

$X Y Z$	$V_x V_y V_z$	$Yaw Roll Pitch$	$V_{Yaw} V_{Roll} V_{Pitch}$	time	type
---------	---------------	------------------	------------------------------	------	------

Tabella 4.3: Formato dei via point

sono esplicitamente indicate solo le velocità del primo ed ultimo via point (fissate solitamente a 0), mentre le velocità dei punti intermedi non sono fornite, ed i loro valori, necessari per il calcolo dei polinomi interpolanti, sono determinati in fase di interpolazione (paragrafo 4.2.3). Il campo *time* rappresenta il tempo in secondi che il manipolatore deve impiegare per percorrere lo spazio che separa il via point attuale da quello precedente. *type* indica infine con il proprio valore come vadano interpretate le informazioni di posizione dei via point ¹:

- 0 : la posizione e l'orientazione del gripper sono espressi in termini assoluti rispetto alla terna di riferimento posta nella base del manipolatore (figura 3.3).
- 1 : i dati di posizione ed orientazione sono relativi al primo via point della sequenza.
- 2 : i dati di posizione ed orientazione sono relativi al via point precedente.

Un caso particolare è rappresentato dal primo via point della sequenza i cui valori di posizione sono tutti nulli e vengono inizializzati prima dell'interpolazione con la posizione e l'orientazione del gripper in quel dato istante. Ciascun via point letto da file viene memorizzato in un oggetto di tipo *ViaPoint*, ed inserito in un vettore assieme agli altri punti del percorso.

Conclusa la fase di acquisizione dei dati, avviene l'attivazione di *getCommand*.

4.2.2.1 Activity *getCommand*

getCommand è un'Activity aperiodica e il suo funzionamento è riconducibile a quella di una macchina a stati, con due soli stati che commutano ad ogni attivazione. Assumendo come modello elementare di un task di manipolazione una sequenza alternata di movimenti del manipolatore e azioni di controllo del gripper,

¹Indipendentemente dal valore di *type*, prima dell'interpolazioni tutte le informazioni spaziali dei via point vengono comunque espresse in coordinate assolute; le coordinate relative sono state introdotte semplicemente per agevolare la stesura 'manuale' delle traiettorie da eseguire.

abbiamo che nel primo stato di funzionamento *getCommand* si occupa di inviare al modulo *Control* i punti del percorso da interpolare. Il passaggio dei via point fra i due moduli avviene tramite l'operazione di `update()` sull'oggetto *Sender path* da parte di *getCommand*. A questa operazione di aggiornamento è agganciata l'attivazione dell'Activity *trajectory*, (figura 4.5), a cui fa seguito la sospensione di *getCommand*. La nuova attivazione di *getCommand* è legata all'aggiornamento dell'oggetto *Receiver controlModuleStopped*, che segnala la conclusione dell'attività di controllo in corso e la possibilità da parte del controllore di iniziare un nuovo task di controllo. Alla sua riattivazione *getCommand* si trova nel secondo stato di funzionamento in cui provvede ad avviare la procedura di apertura/chiusura del gripper tramite aggiornamento dell'oggetto *Sender fingerDistance* a cui è associata l'attivazione dell'Activity *gripper*. Anche in questo caso all'operazione di `update()` segue la sospensione di *getCommand* fino ad una sua nuova riattivazione, che la riporterà nuovamente nel primo stato di funzionamento. L'alternanza fra i due stati prosegue fino al completamento dell'intero task di manipolazione.

Per quanto riguarda la natura degli oggetti *Sender* e *Receiver* dichiarati dal modulo *Command*, *path* è un oggetto di tipo `vector<ViaPoint>`, un vettore quindi in cui sono memorizzati tutti i via point della sequenza da interpolare. *fingerDistance* è di tipo `float` e contiene il valore dell'apertura del gripper in mm mentre *ControlModuleStopped* è un `bool` ed ha semplicemente il compito di segnalare la conclusione del ciclo di controllo.

4.2.3 Il modulo *Control*

Il modulo *Control* rappresenta il componente centrale dell'architettura di controllo del *Manus*. Al suo interno infatti, le tre *Activity* presenti si occupano della generazione delle traiettorie che l'organo terminale del manipolatore deve eseguire, ne controllano l'esecuzione e gestiscono l'apertura e la chiusura del gripper durante i compiti di manipolazione. Le comunicazioni con gli altri due moduli del sistema sono garantite dai sei pattern *Information* mostrati nel listato 4.3, quattro di tipo *Receiver* e due di tipo *Sender*; questi oggetti oltre che garantire uno scambio continuo di dati, svolgono un importante ruolo di sincronizzazione dei moduli, dal momento

che le attivazioni e le sospensioni di alcune *Activity* sono legate alle operazioni di `update()` sui pattern *Information* stessi.

```
class ControlModule : public Module
{
private:
    void controlLoop();
    void computeTrajectory();
    void moveGripper();

public:
    ControlModule();

    Activity< void , void > trajectory;
    Activity< void , void > gripper;
    Activity< void , void > control;

    Receiver< JointPosition > position;
    Receiver< ManusStatus > status;
    Receiver< vector<ViaPoint> > path;
    Receiver< float > fingerDistance;

    Sender< JointSpeed > speed;
    Sender< bool > controlModuleStopped;
};
```

Listato 4.3: *Activities, Senders e Receivers* del modulo *Control*.

4.2.3.1 Activity *trajectory*

Il compito dell'Activity *trajectory* è quello di generare partendo da una sequenza di punti di passaggio una traiettoria, che in un secondo momento è utilizzata come riferimento spaziale-temporale dal controllore del moto del manipolatore. *trajectory* è aperiodica, ed è attivata ad ogni operazione di `update()` sul pattern *Information path* da parte dell'Activity *getCommand* (paragrafo 4.2.3). L'aggiornamento di *path*, e la conseguente attivazione di *trajectory* comporta il passaggio dal modulo *command* al modulo *control* del vettore contenente i *via point* del percorso che il

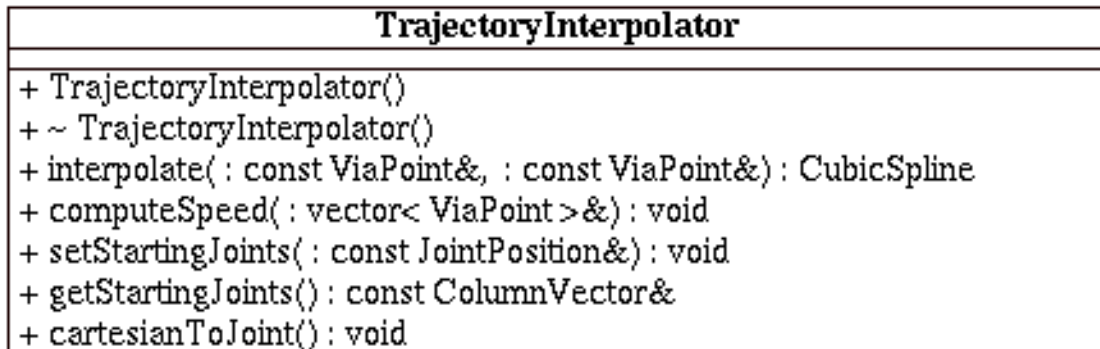


Figura 4.3: Diagramma UML della classe *trajectoryInterpolator*

gripper durante il suo movimento deve seguire. Partendo dal vettore di via point, trajectory esegue le seguenti azioni:

- Trasforma le coordinate cartesiane dei via point in coordinate nello spazio dei giunti tramite risoluzione del problema cinematico inverso.
- Interpola nello spazio dei giunti le coppie di via point adiacenti tramite polinomi cubici.

Per la prima delle due operazioni *trajectory* utilizza la classe *Manipulator* (figura 4.4), che fornisce tutte le funzioni per la gestione della cinematica del *Manus*. L'interpolazione dei via point è invece realizzata tramite le funzioni della classe *TrajectoryInterpolator* (figura 4.3).

Con riferimento al diagramma di figura 4.5 vediamo in dettaglio dal momento dell'attivazione di *trajectory* la sequenza di operazioni e di funzioni utilizzate per la generazione della traiettoria.

1. Viene letto tramite la chiamata di `get()` sull'oggetto Receiver *path*, il vettore contenente la sequenza di via point da interpolare.
2. `setStartingJoints()` inizializza l'istanza dell'oggetto *Manipulator*, membro della classe *TrajectoryInterpolator*, con i valori correnti delle variabili di giunto, ricevute dal modulo *Encoder* tramite il pattern *Information position*.

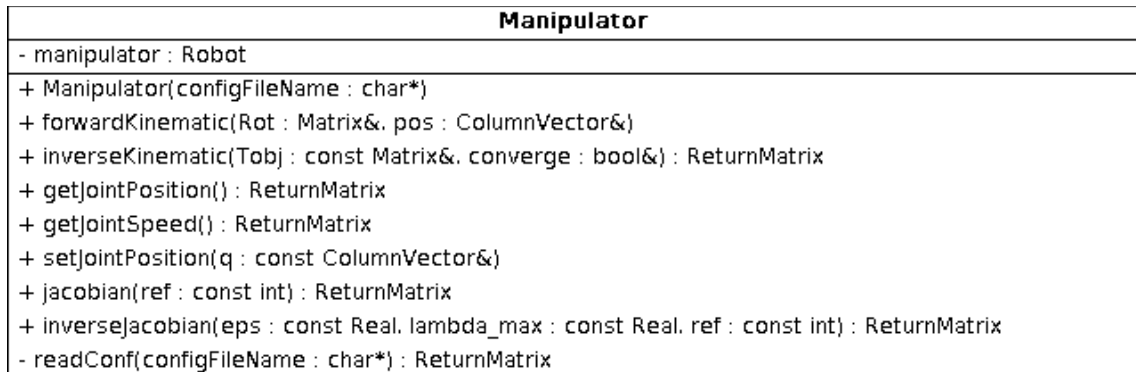


Figura 4.4: Diagramma UML della classe *Manipulator*

3. `setStartingViaPoint()` inizializza il primo via point della sequenza con le coordinate del punto corrispondente all'attuale posizione del gripper. In questo modo il manipolatore è in grado di percorrere il cammino che lo separa dalla posizione che segna l'inizio della traiettoria da eseguire.
4. `cartesianToJoint()` trasforma le informazioni di posizione rappresentate dai via point dallo spazio cartesiano a quello dei giunti tramite l'algoritmo di cinematica inversa implementato dalla funzione `inverseKinematic()`. Prima di fare questo però, qualora le informazioni spaziali contenute nei via point siano di tipo relativo, le coordinate vengono trasformate in assolute, sommando al via point iniziale ottenuto al punto 3 gli incrementi spaziali ed angolari espressi dai rimanenti punti.

Terminata la fase di acquisizione e trasformazione dei via point, ha inizio quella della loro interpolazione nello spazio dei giunti, che si svolge in due passi:

5. `computeSpeed()` determina le velocità di passaggio dei via point intermedi che non sono note a priori. Il calcolo viene fatto imponendo l'ipotesi di continuità dell'accelerazione nei punti di raccordo delle funzioni interpolanti, espressa dalle relazioni [2.27](#) e [2.28](#).
6. `interpolate()` calcola per ogni coppia di via point consecutivi i coefficienti del polinomio cubico interpolante secondo le espressioni [2.10-2.13](#),

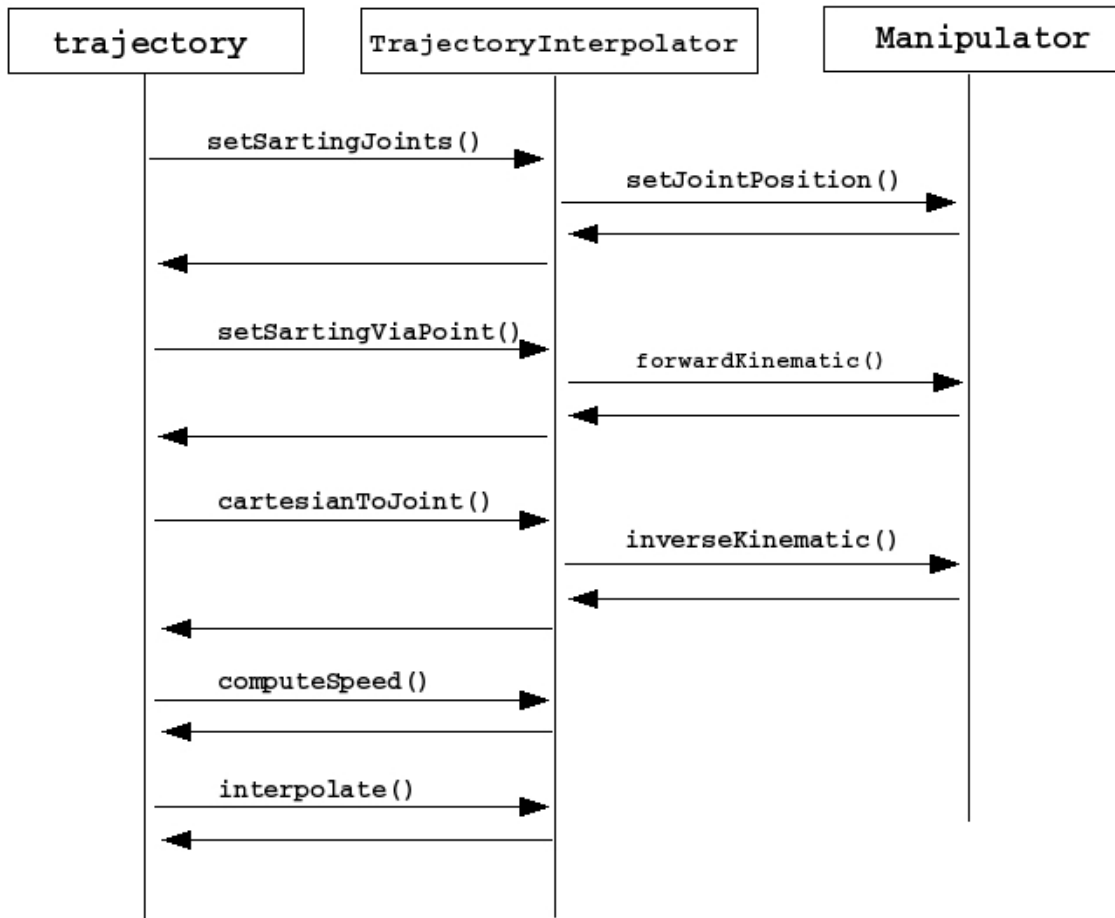


Figura 4.5: Diagramma di sequenza dell'Activity *trajectory*

e successivamente i parametri di ogni polinomio sono memorizzati in un oggetto di tipo *CubicSpline* ed inseriti in un vettore.

Al termine dell'interpolazione, *trajectory* prima di essere sospesa dallo scheduler del framework in attesa di una nuova attivazione, aggancia l'esecuzione dell'Activity *control* alla ricezione dei dati contenuti nell'oggetto *position*. In questo modo, al successivo aggiornamento di *position* da parte dell'attività *manusConf* il controllore viene attivato, ed ha inizio la generazione del moto.

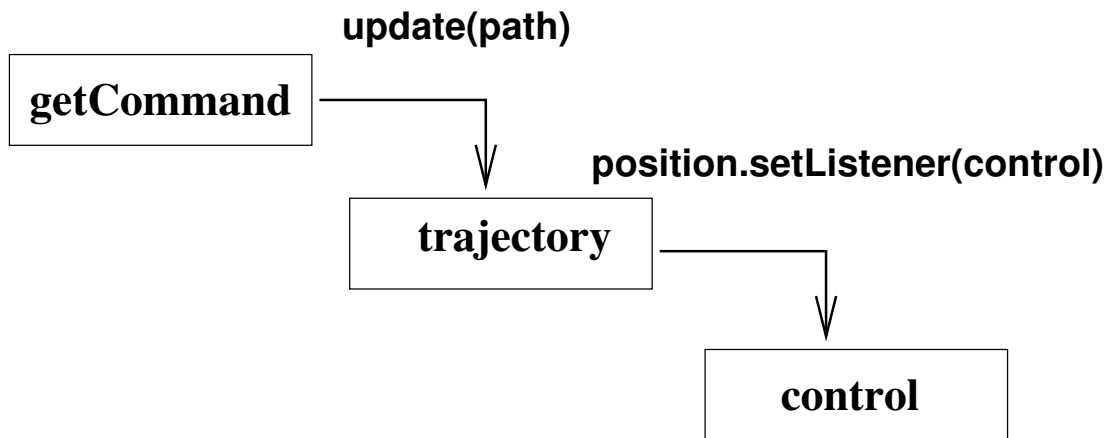


Figura 4.6: Sequenza di attivazione di *trajectory*

Fra tutte le attività del sistema, l'operazione di pianificazione della traiettoria eseguita da *trajectory* è quella che presenta una maggior complessità computazionale, tuttavia, dal momento che il calcolo della traiettoria avviene a manipolatore fermo ed all'esterno del ciclo di controllo del moto, non esiste alcun problema di rispetto dei vincoli temporali imposti dall'elettronica del *Manus*.

La realizzazione delle funzioni di cinematica diretta ed inversa da parte della classe *Manipulator* si basa sulla libreria *Roboop* [13]. I metodi di *Manipulator* rappresentati nel diagramma UML di figura 4.4 si limitano infatti a richiamare semplicemente le corrispondenti funzioni implementate in *Roboop*. Il fatto di non utilizzare direttamente i metodi e le classi di *Roboop*, consente di poter ricorrere a librerie differenti, o di realizzarne nuove, senza cambiare, o comunque apportando modifiche limitate al resto dell'architettura, dal momento che l'interfaccia di *Manipulator* rimane inalterata, e ciò che cambia è solo l'implementazione.

4.2.3.2 Roboop

La libreria *Roboop* è una raccolta portabile di classi e funzioni C++ sviluppata presso il Politecnico di Montreal, che fornisce all'utilizzatore tutti gli strumenti per la realizzazione e la simulazione di sistemi di controllo per robot manipolatori. La parte matematica è affidata a *NewMat*, una libreria per l'algebra lineare realizzata

da Robert Davies [14], che mette a disposizione tutte le funzioni più comuni per l'elaborazione di matrici e vettori.

L'elenco che segue riporta brevemente e suddivise per argomento le funzionalità implementate da *Roboop*:

- Descrizioni e trasformazioni spaziali: sono presenti le funzioni per la realizzazione di trasformazioni geometriche elementari, quali rotazioni e traslazioni, e trasformazioni geometriche omogenee, in aggiunta alla definizione di una classe *Quaternion* per la descrizione delle rotazioni tramite quaternioni unitari.
- Cinematica: sono presenti funzione per il calcolo della cinematica diretta secondo l'espressione A.18 , ed una funzione per la risoluzione numerica del problema cinematico inverso applicabile ad un manipolatore generico, a cui si affiancano le tre soluzioni analitiche per i manipolatori *Rhino*, *Schilling* e *Puma*.
- Cinematica differenziale: sono presenti i metodi per il calcolo dello jacobiano diretto ed inverso (paragrafo A.5) e delle relative derivate per il calcolo delle accelerazioni delle variabili di giunto, note le velocità.
- Dinamica: funzioni per la determinazione del modello dinamico del manipolatore, comprendente l'inerzia, i momenti torcenti e gli effetti prodotti dalle forze di Coriolis, centrifughe e gravitazionale.
- Generazione di traiettorie: classi e metodi per l'interpolazione nello spazio dei giunti, o in quello cartesiano, di un percorso assegnato mediante spline cubiche.
- Controllo del moto: la libreria dispone di quattro controllori del moto, le funzioni per la loro selezione e gestione, e quelle per l'interazione con gli attuatori del manipolatore.
- Grafica: sono presenti una serie di funzioni per la creazioni di grafici 2D, realizzati tramite l'applicazione *GnuPlot*.

La classe *Manipulator* ricorre alle funzionalità di *Roboop* esclusivamente per quanto riguarda la cinematica diretta ed inversa, e per quella differenziale, che non è comunque utilizzata nell'attuale implementazione del sistema di controllo.

Tutte le funzioni cinematiche di *Roboop* sono funzioni membro della classe *Robot*, di cui è presente un'istanza in *Manipulator* (figura 4.4). Al proprio interno *Robot* conserva tutti i parametri di configurazione del manipolatore che rappresenta: il numero di gradi di libertà n_{DOF} , i parametri inerziali e quelli cinematici, memorizzati in oggetti di tipo *Link*. La classe *Link* contiene i parametri necessari a caratterizzare un braccio della catena cinematica del manipolatore secondo la convenzione i Denavit-Hartenberg standard (A.3.1) o quella modificata (A.3.2), tali parametri sono: il tipo di giunto (rotazionale o prismatico), i quattro parametri DH a , d , θ ed α , e un parametro indicante il tipo di convenzione DH adottata (standard o modificata). A questi parametri, peraltro già sufficienti per la caratterizzazione cinematica del manipolatore, si aggiungono i parametri inerziali, quali la massa, il vettore dei centri di massa e la matrice di inerzia, e quelli riguardanti i motori (inerzie, attriti e rapporti di riduzione). L'inizializzazione dell'istanza della classe *Robot* presente in *Manipulator* avviene tramite lettura da file, ed è limitata ai soli parametri cinematici. Per quanto riguarda il sottoinsieme dei metodi di *Roboop* utilizzati da *Manipulator* (figura 4.4), l'unica annotazione importante riguarda il metodo `inverseKinematic`, che per l'inversione cinematica utilizza il metodo della classe *Robot* `inv_kin_puma`. Quest'ultima è una delle tre funzioni citate in precedenza, in grado di risolvere in forma chiusa il problema cinematico inverso per il manipolatore Puma in modo analitico tramite disaccoppiamento (paragrafo A.4.3). La sua applicazione al *Manus* è possibile in quanto le strutture cinematiche dei due manipolatori sono equivalenti.

4.2.3.3 Activity gripper

gripper ha il compito di inoltrare al controllore un comando di apertura o chiusura della pinza del gripper. È un'Activity aperiodica e la sua attivazione è legata all'operazione di `update()` sull'oggetto *fingerDistance*, realizzato dall'Activity *getCommand*. In seguito all'aggiornamento dell'*Information* pattern, *fingerDistance* contiene il valore in mm della distanza a cui devono essere portate le estremità delle due

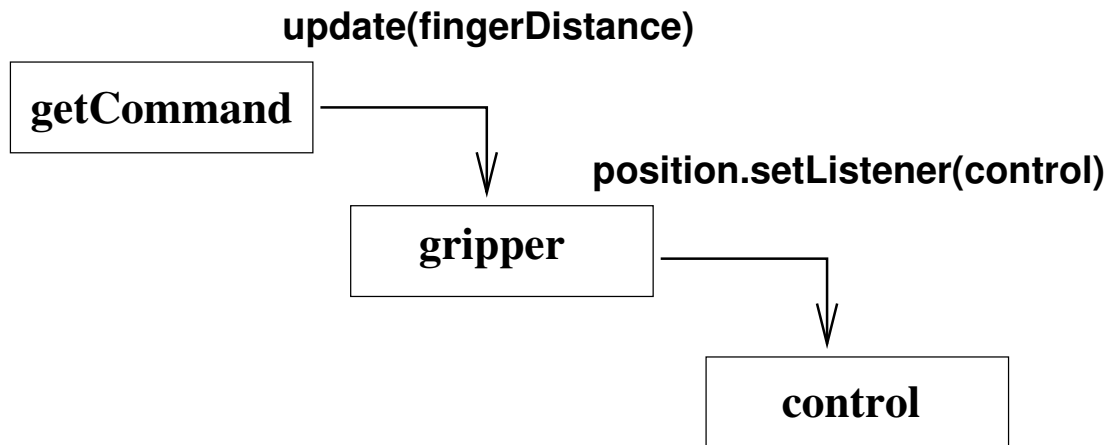


Figura 4.7: Sequenza di attivazione di *gripper*

dita che formano i gripper (figura 3.2). L'Activity si limita a recuperare mediante il metodo `get()` il valore memorizzato in *fingerDistance*, rendendolo disponibile al controllore, e poi, prima di essere sospesa in attesa di una nuova attivazione, in modo del tutto equivalente a quanto avviene per l'Activity *trajectory*, aggancia l'attivazione dell'attività di controllo all'evento di aggiornamento dell'oggetto Receiver *position*, in modo da garantire l'inizio della procedura di azionamento del gripper nei successivi 60 ms.

4.2.3.4 Activity control

L'Activity *control* implementa il sistema di controllo vero e proprio del *Manus*. L'attivazione di *control* è legata alla ricezione di una nuovo set di valori delle variabili di giunto nel pattern *Information position*. Il vincolo tra aggiornamento di *position* ed attivazione di *control* non è permanente, si instaura infatti al termine dell'attività delle Activity *trajectory* o *gripper* (paragrafi 4.2.3.1-4.2.3.3), che provvedono entrambe a registrare *control* come oggetto *Listener* di *position*, e termina con la fine dell'azione di controllo.

Il ciclo di controllo inizia con la lettura dei due campi memorizzati nell'oggetto *status*, che contengono i codici indicanti lo stato di funzionamento del *Manus* (ta-

bella 3.3). Il contenuto di *status* viene analizzato, in caso di errore il manipolatore viene fermato, in caso contrario *control* procede scegliendo una fra le due possibili azioni di controllo:

- Controllo del moto del manipolatore nello spazio dei giunti (paragrafo 2.2).
- Apertura e chiusura del gripper.

Il codice che implementa le due azioni di controllo fa parte dello stesso metodo `controlLoop()`, e la scelta su quale delle due eseguire avviene sulla base del valore di un flag, che viene impostato dall'Activity che, fra *trajectory* e *gripper* ha determinato l'attivazione di *control*. La presenza di una sola Activity di controllo consente di sovrapporre temporalmente il movimento del manipolatore con la chiusura/apertura del gripper, semplicemente eseguendo entrambi i cicli di controllo in modo sequenziale ad ogni attivazione e poi inviando simultaneamente al manipolatore i sei byte con le velocità delle variabili di giunto ed il byte con l'informazione sul gripper.

Controllo del moto

L'obiettivo del controllo del moto è quello di garantire che l'organo terminale del manipolatore si muova all'interno dello spazio di lavoro rispettando i vincoli spaziali e temporali rappresentati dai via point dati in ingresso al sistema. Il controllo si basa sul classico schema con retroazione di posizione, dove la chiusura dell'anello di retroazione proveniente dal manipolatore è realizzata dal modulo *encoder* tramite il pattern *Information position*. I valori di riferimento per le sei variabili di giunto sono ottenuti dai polinomi cubici calcolati dall'Activity *trajectory* e memorizzati in oggetti *CubicSpline*, che come mostrato dal diagramma UML di figura 4.8, mettono a disposizione del controllore i metodi per ottenere i valori di posizione, velocità ed accelerazione ad un dato istante. La gestione temporale è invece affidata ad un oggetto di tipo *Timer*, che consente di misurare il tempo trascorso dall'inizio del ciclo di controllo (figura 4.9). La classe *Controller* (figura 4.10) infine si occupa di verificare lo stato del sistema e calcolare un nuovo set di velocità dei giunti sulla base della posizione attuale del manipolatore, tramite un controllore di tipo proporzionale.

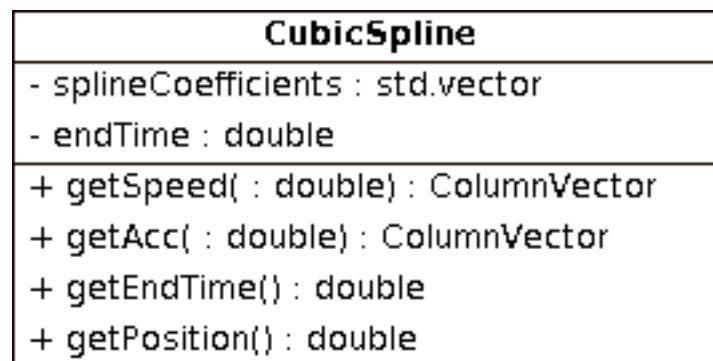


Figura 4.8: Diagramma UML della classe *CubicSpline*

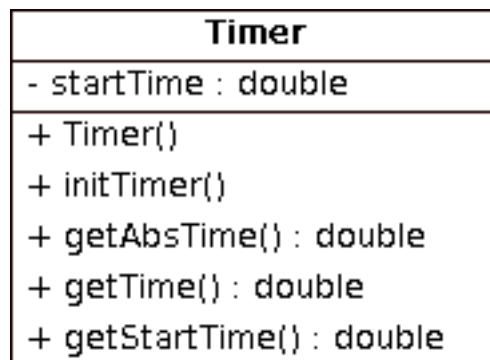


Figura 4.9: Diagramma UML della classe *Timer*

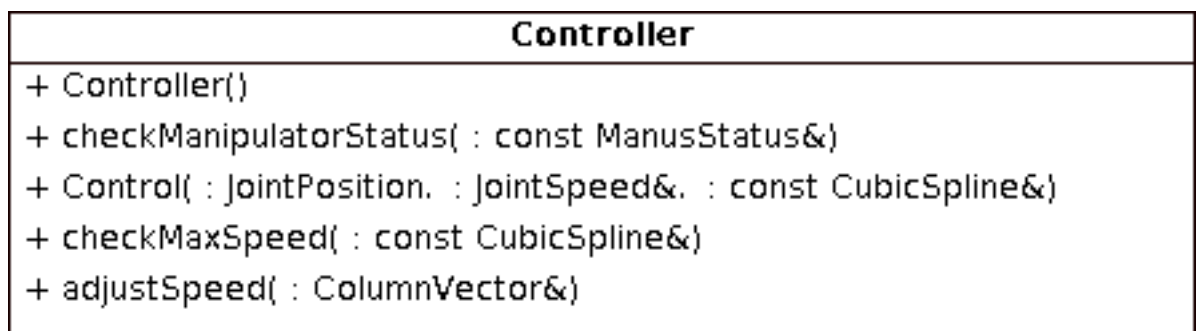


Figura 4.10: Diagramma UML della classe *Controller*

Il ciclo di controllo inizia con la selezione dal vettore contenente tutti i polinomi interpolanti della spline cubica corrispondente alla porzione di traiettoria che vogliamo generare e con l'inizializzazione del timer di sistema, e continua con i tre passaggi che seguono:

1. lettura tramite l'oggetto Receiver *position* della posizione corrente q dei giunti.
2. calcolo tramite la spline cubica della posizione e della velocità teoriche q_0 e \dot{q}_0 all'istante di tempo attuale.
3. calcolo della velocità \dot{q} a cui azionare i giunti secondo l'espressione:

$$\dot{q} = \dot{q}_0 + K(q - q_0) \quad (4.1)$$

con K costante di proporzionalità.

Al termine del terzo passo del ciclo di controllo abbiamo a disposizione i valori di velocità dei giunti che consentono al manipolatore di realizzare la traiettoria specificata. Prima del loro invio al modulo *Encoder*:

- Si verifica che le velocità calcolate dal controllore non eccedano la massima velocità di rotazione ammessa dal manipolatore, procedendo eventualmente ad una scalatura.
- Le velocità finora espresse in gradi/secondo vengono trasformate in incrementi spaziali espressi in gradi, e successivamente approssimati ad uno dei 20 valori interi ammessi dall'elettronica di controllo del *Manus* (tabella 3.7).
- Viene compensata la dipendenza fra secondo e terzo giunto, sommando all'incremento spaziale di quest'ultimo quello del secondo giunto.

A questo punto viene eseguito l'`update()` sull'oggetto *speed*, e le velocità calcolate sono inviate in forma di incrementi spaziali al modulo *Encoder* che provvede ad inviarle al manipolatore.

I passaggi 1,2,3 vengono ripetuti ad ogni attivazione di *control* fino al raggiungimento del via point o allo scadere del tempo a disposizione del manipolatore per quel tratto di traiettoria. Nel primo caso il via point è considerato raggiunto quando la sua distanza dalla posizione corrente del gripper è inferiore ad una certa soglia. Se il via point raggiunto non è l'ultimo della sequenza, il controllo procede con la selezione della spline cubica successiva ed inizia un nuovo ciclo. Se invece il via point

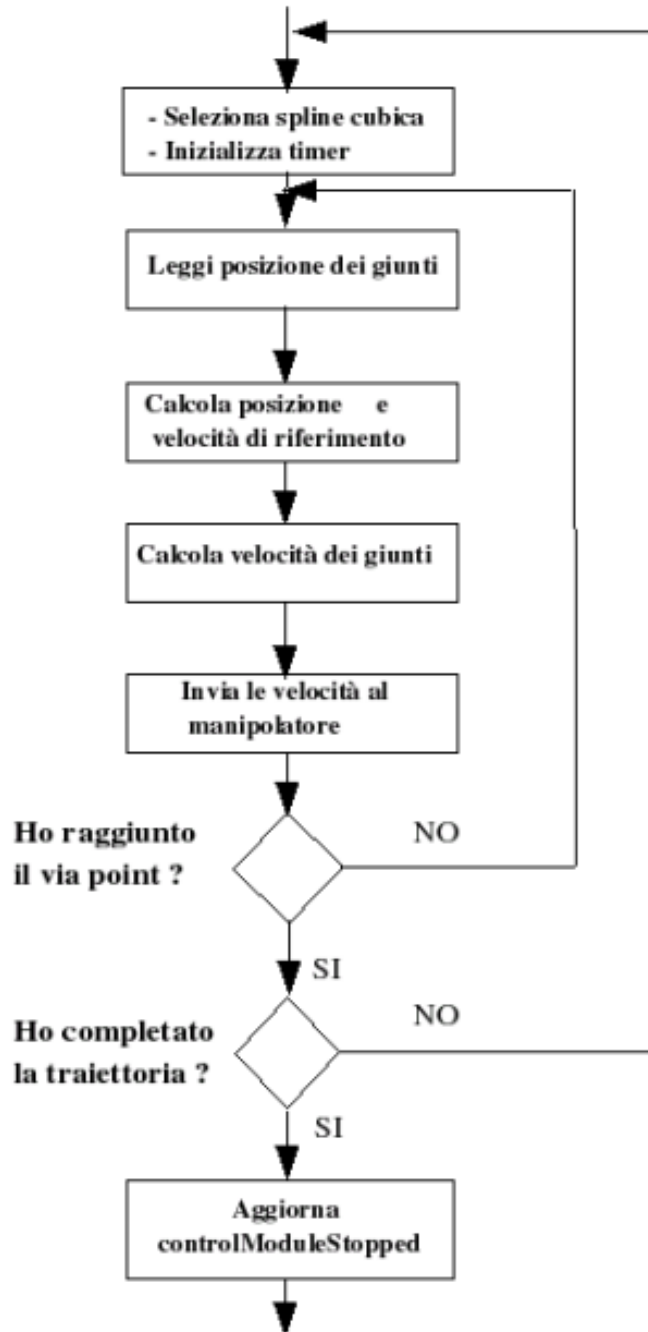


Figura 4.11: Diagramma di flusso del ciclo di controllo del *motodiagrammaFlussoControlloMoto*

appena raggiunto è l'ultimo della sequenza, *control* ha terminato la propria attività, aggiorna il pattern *Information controlModuleStopped* in modo da segnalare al modulo *command* la possibilità di inviare un nuovo comando, e viene sospeso in attesa di una nuova attivazione. Nel caso in cui allo scadere del tempo a disposizione il gripper non abbia ancora raggiunto il via point, e se il via point in questione non è l'ultimo della sequenza si procede come nel caso precedente con la selezione della spline successiva. Se invece il via point chiude la sequenza, viene valutato l'errore di posizione del gripper, e se eccessivo viene calcolata una nuova spline che interpola la posizione attuale del gripper con quella conclusiva della traiettoria e viene avviato un nuovo ciclo di controllo in modo da riportare l'errore di posizione al di sotto di una soglia il cui valore può essere regolato sulla base della precisione che vogliamo ottenere.

Controllo del gripper

La seconda azione di controllo che l'Activity *control* realizza, riguarda l'apertura e la chiusura del gripper. Anche in questo caso come nell'azione di controllo del moto, l'attivazione di *control* è legata all'aggiornamento del contenuto di *position*, con la differenza che in questa circostanza è l'Activity *gripper* e non *trajectory* ad occuparsi di registrare *control* come *Listener* di *position*. Ad ogni attivazione *control* legge l'ottavo elemento dell'array memorizzato in *position*, che fornisce la misura dell'attuale distanza fra le due estremità del gripper, e confronta questo valore con quello che si desidera ottenere. Se il gripper 'è più aperto del necessario, *control* provvede a far chiudere la pinza andando a scrivere nell'ottavo elemento dell'array contenuto in *speed* un valore di incremento negativo. Nel caso invece in cui sia necessario aprire il gripper viene inviato all'attuatore del gripper un valore di velocità positivo. Quando la differenza tra apertura desiderata ed apertura reale scende al di sotto un valore di soglia, il ciclo di controllo termina, *control* cancella se stessa come *Listener* di *position* tramite la chiamata al metodo `clearListener()`, ed aggiorna il contenuto dell'oggetto Sender *controlModuleStopped* in modo da attivare l'Activity *getCommand*.

Capitolo 5

Risultati sperimentali

Nel capitolo precedente è stata analizzata in tutti i suoi componenti l'architettura del sistema di controllo: le attività presenti in ciascun modulo, le sequenze di attivazione, gli algoritmi utilizzati e le modalità di comunicazione. In questo capitolo sono invece esposti i risultati riguardanti alcuni test condotti per valutare sia le caratteristiche intrinseche del manipolatore, sia quelle del sistema di controllo oggetto di questo lavoro di tesi.

5.1 Risposta al gradino del controllore PI

Per prima cosa è stato realizzato un test per valutare preliminarmente l'efficienza del controllore PI presente nell'unità di controllo del *Manus*, ed implementato dal processore matematico 80C186 (figura 5.1). Questo controllore, come già illustrato nel capitolo 3, si occupa di impostare le velocità di rotazione dei giunti, pari ai valori di riferimento calcolati dal sistema di controllo esterno, ed espressi in termini di incrementi angolari, producendo in uscita un adeguato set di valori per gli attuatori del manipolatore. Il test è consistito nel valutare il comportamento del controllore in risposta a segnali di ingresso a gradino. Il giunto prescelto per l'analisi è stato il secondo, ed il manipolatore è stato collocato in posizione orizzontale e completamente distesa in modo da massimizzare gli effetti dell'inerzia della struttura al momento dell'applicazione del gradino.

Partendo da questa posizione con il manipolatore fermo, è stato impartito al secon-

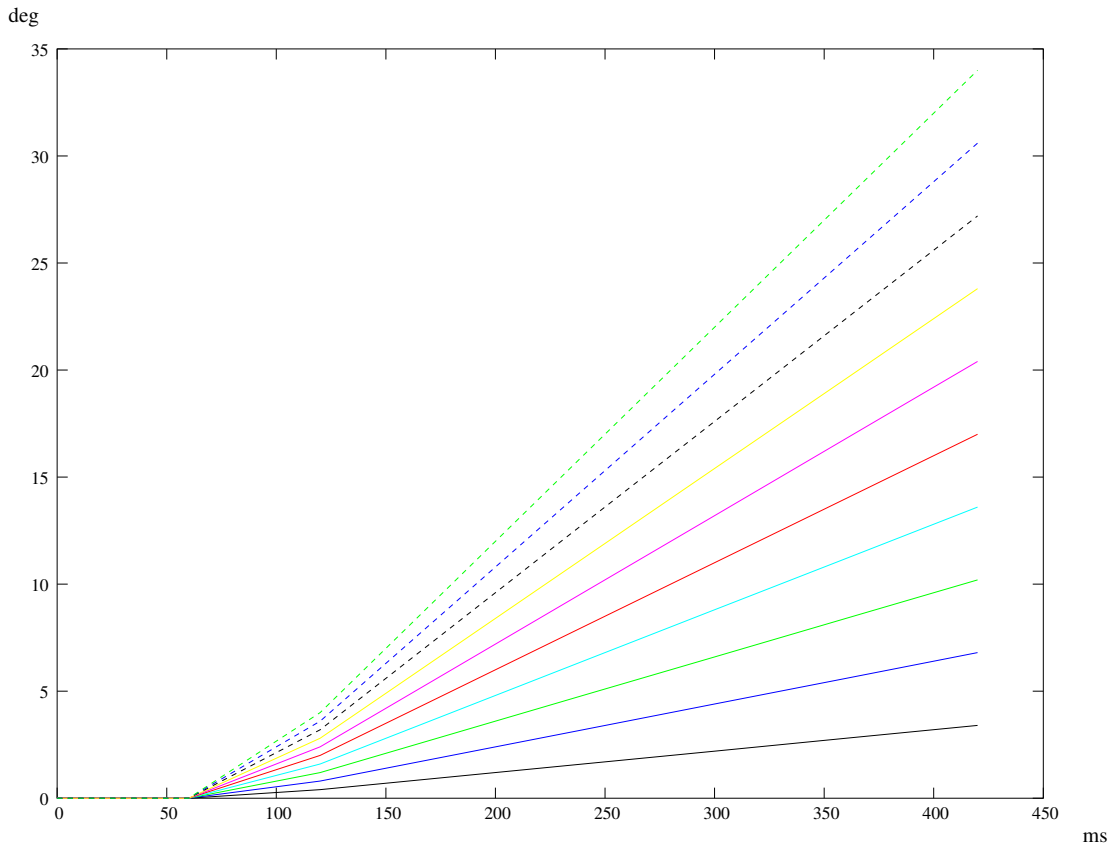


Figura 5.1: Risposta al gradino del controllore PI.

do giunto un comando di movimento costante, con valore di incremento angolare minimo pari ad 1° , e si è contemporaneamente provveduto a registrare il valore della seconda variabile di giunto. La prova è stata successivamente ripetuta per tutte le 10 velocità ammesse dalla modalità di funzionamento nello spazio dei giunti.

La figura 5.1 riporta il valore dell'angolo di rotazione del secondo giunto (espresso in gradi) al trascorrere del tempo (espresso in ms) per i dieci possibili valori di velocità in ingresso, con l'istante temporale 0 corrispondente al momento di applicazione del gradino.

Per prima cosa si può osservare come per i primi 60 ms il giunto non si muova: questo è un comportamento previsto e documentato dell'elettronica di controllo del manipolatore, che esegue il primo comando ricevuto con un ritardo proprio di 60

¹corrispondente, secondo quanto riportato dei dati della tabella 3.7, ad una velocità angolare di 5 *gradi/sec*

ms. Trascorsi 120 ms, la pendenza delle rette, ovvero la velocità angolare del giunto si stabilizza sul valore di riferimento, che da quel momento viene mantenuto fino a quando non ne viene fissato uno nuovo dall'esterno. Nell'intervallo temporale 60-120 ms ha invece luogo, sotto l'azione del controllore, il transitorio che porta la velocità al valore fissato. In realtà, poichè i valori delle variabili di giunto sono disponibili solamente ad intervalli periodici di 60 ms, non è possibile stabilire con esattezza il tempo impiegato dal controllore per portare la velocità di rotazione al valore di riferimento, ed inoltre, poiché il controllore software può inviare un nuovo set di valori di riferimento ogni 60 ms, non si può determinare per quale percentuale del periodo di controllo i giunti si muovano realmente alla velocità voluta. Tuttavia si può ritenere soddisfacente il comportamento del controllore PI.

5.2 Test sul generatore di traiettorie

Il secondo test realizzato è finalizzato alla verifica dell'efficacia complessiva del sistema di controllo software nella realizzazione delle traiettorie, ed a una valutazione della precisione con cui queste sono generate.

Le prove sono state condotte fornendo al sistema alcuni percorsi specificati tramite sequenze di via point, secondo il formato visto nel paragrafo 4.2.2. La traiettoria ottenuta in seguito all'azione di controllo è stata valutata andando a leggere i valori delle variabili di giunto provenienti dall'anello di retroazione di posizione del controllore, ed ottenendo, tramite l'applicazione della cinematica diretta, la posizione del gripper nello spazio cartesiano.

La prima traiettoria realizzata consiste nel muovere il gripper lungo un quadrato di lato 30 cm, posizionato sul piano yz dello spazio di lavoro del *Manus*, mantenendone costante l'orientazione. Il quadrato è stato specificato fornendo 3 punti di passaggio per ogni vertice, così come mostra la sequenza dei via point, espressi in coordinate relative (listato 5.1), in cui i valori del primo punto sono sostituiti dai valori della posizione cartesiana occupata dal gripper nell'istante di tempo in cui ha inizio l'interpolazione della traiettoria.

Nel grafico di figura 5.2 sono rappresentate in coordinate cartesiane le posizioni occupate dal gripper durante l'esecuzione della traiettoria quadrata. Da una prima

0.0	0.0	0.0	0 0 0	0 0 0	0 0 0	0.0	0
0.0	50.0	0.0	0 0 0	0 0 0	0 0 0	0.8	2
0.0	100.0	0.0	0 0 0	0 0 0	0 0 0	2.4	2
0.0	100.0	0.0	0 0 0	0 0 0	0 0 0	4.0	2
0.0	50.0	0.0	0 0 0	0 0 0	0 0 0	4.8	2
0.0	0.0	-50.0	0 0 0	0 0 0	0 0 0	5.6	2
0.0	0.0	-100.0	0 0 0	0 0 0	0 0 0	7.2	2
0.0	0.0	-100.0	0 0 0	0 0 0	0 0 0	8.8	2
0.0	0.0	-50.0	0 0 0	0 0 0	0 0 0	9.6	2
0.0	-50.0	0.0	0 0 0	0 0 0	0 0 0	10.4	2
0.0	-100.0	0.0	0 0 0	0 0 0	0 0 0	12.0	2
0.0	-100.0	0.0	0 0 0	0 0 0	0 0 0	13.6	2
0.0	-50.0	0.0	0 0 0	0 0 0	0 0 0	14.4	2
0.0	0.0	50.0	0 0 0	0 0 0	0 0 0	15.2	2
0.0	0.0	100.0	0 0 0	0 0 0	0 0 0	16.8	2
0.0	0.0	100.0	0 0 0	0 0 0	0 0 0	18.4	2
0.0	0.0	50.0	0 0 0	0 0 0	0 0 0	19.2	2

Listato 5.1: Via point per la generazione di un quadrato.

analisi della curva si può notare come i due lati verticali presentino un andamento più regolare rispetto a quelli orizzontali. Questo comportamento è giustificabile con il fatto che l'esecuzione dei tratti orizzontali prevede la rotazione del primo giunto, che risulta sensibilmente più soggetta ad oscillazione ed imprecisioni rispetto ai rimanenti giunti della catena cinematica, dal momento che comporta lo spostamento dell'intero manipolatore, attuatori compresi. La realizzazione dei tratti verticali non prevede invece l'azionamento del primo giunto, con evidenti benefici sulla qualità della traiettoria ottenuta.

Un'ulteriore fonte di imprecisione è legata al metodo con cui vengono determinate le velocità di controllo. Com'è già stato discusso nel paragrafo 4.2.3.4, la velocità di controllo ottima calcolata dal controllore per ciascun giunto deve essere ricondotta ad uno dei venti ² valori accettati dal controllore PI del *Manus*. Questa operazione di quantizzazione, fatta tramite approssimazione, produce una velocità di controllo effettiva maggiore o minore rispetto a quella calcolata. Questa differenza porta ad un errore di posizione che il controllore provvede a compensare nei cicli di controllo

²10 per ognuno dei due sensi di rotazione

successivi, che a loro volta però sono soggetti allo stesso problema, con il conseguente innesco di oscillazioni a carattere permanente, che accompagnano l'intera esecuzione della traiettoria. Nel corso dei test è risultato inoltre evidente come la presenza delle oscillazioni sia esaltata dall'utilizzo di velocità di azionamento dei giunti troppo basse o troppo elevate. Ai fini della qualità delle traiettorie generate, è necessario quindi determinare dei tempi di percorrenza del percorso ottimali, che consentano di limitare le oscillazioni e gli errori di posizione del gripper.

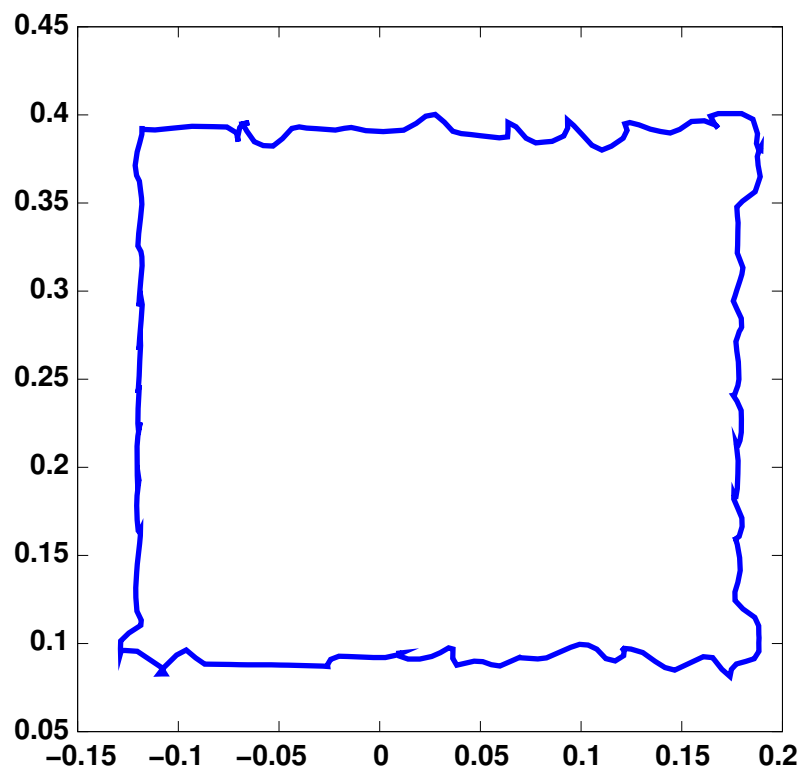


Figura 5.2: Esecuzione di una traiettoria quadrata da parte del *Manus*.

L'utilizzo delle *spline cubiche* garantisce il transito del gripper per i via point specificati, ma non consente di controllarne la traiettoria al di fuori dei punti di passaggio. Inoltre queste spline non si prestano alla realizzazione di tratti di traiettorie rettilinei come quelli presenti nel quadrato visto in precedenza, a meno di non utilizzare un numero elevato di via point ravvicinati tra loro. Per questo motivo il secondo test ha riguardato la realizzazione di una traiettoria circolare, che meglio si adatta,

quanto a conformazione e curvatura, alla natura dei polinomi cubici. Il listato 5.2 contiene la sequenza di via point di una circonferenza di raggio 20 cm posizionata sul piano yz . I punti di passaggio sono disposti uniformemente ad una distanza angolare di 20 gradi l'uno dall'altro, ed anche i tempi di percorrenza dei tratti fra via point adiacenti sono costanti.

0.0	0.0	0.0	0 0 0	0 0 0	0 0 0	0.00	2
0.0	-12.06	68.40	0 0 0	0 0 0	0 0 0	1.40	2
0.0	-34.73	60.15	0 0 0	0 0 0	0 0 0	2.80	2
0.0	-53.21	44.65	0 0 0	0 0 0	0 0 0	4.20	2
0.0	-65.27	23.76	0 0 0	0 0 0	0 0 0	5.60	2
0.0	-69.46	0.00	0 0 0	0 0 0	0 0 0	7.00	2
0.0	-65.27	-23.76	0 0 0	0 0 0	0 0 0	8.40	2
0.0	-53.21	-44.65	0 0 0	0 0 0	0 0 0	9.80	2
0.0	-34.73	-60.15	0 0 0	0 0 0	0 0 0	11.20	2
0.0	-12.06	-68.40	0 0 0	0 0 0	0 0 0	12.60	2
0.0	12.06	-68.40	0 0 0	0 0 0	0 0 0	14.00	2
0.0	34.73	-60.15	0 0 0	0 0 0	0 0 0	15.40	2
0.0	53.21	-44.65	0 0 0	0 0 0	0 0 0	16.80	2
0.0	65.27	-23.76	0 0 0	0 0 0	0 0 0	18.20	2
0.0	69.46	0.00	0 0 0	0 0 0	0 0 0	19.60	2
0.0	65.27	23.76	0 0 0	0 0 0	0 0 0	21.00	2
0.0	53.21	44.65	0 0 0	0 0 0	0 0 0	22.40	2
0.0	34.73	60.15	0 0 0	0 0 0	0 0 0	23.80	2
0.0	12.06	68.40	0 0 0	0 0 0	0 0 0	25.20	2

Listato 5.2: Via point per la generazione di un cerchio.

Il risultato dell'esecuzione della traiettoria è riportato in figura 5.3. Nel complesso la traiettoria ottenuta è soddisfacente, anche se sono evidenti le oscillazioni della posizione del gripper rispetto alla circonferenza ideale, causate dal problema di approssimazione delle velocità di controllo, di cui si è discusso in precedenza.

Dal punto di vista quantitativo, l'errore di posizione sui punti di passaggio, nel corso dei test si è attestato mediamente sui 5 mm, con valori massimi nell'ordine del cm. Per quanto riguarda l'orientazione del gripper, la situazione risulta leg-

germente più critica in seguito al sommarsi durante il movimento del manipolatore degli effetti dei giochi e delle non linearità che affliggono gli ultimi tre giunti della catena cinematica, e che nel complesso portano ad errori di orientazione maggiori rispetto a quelli di posizione.

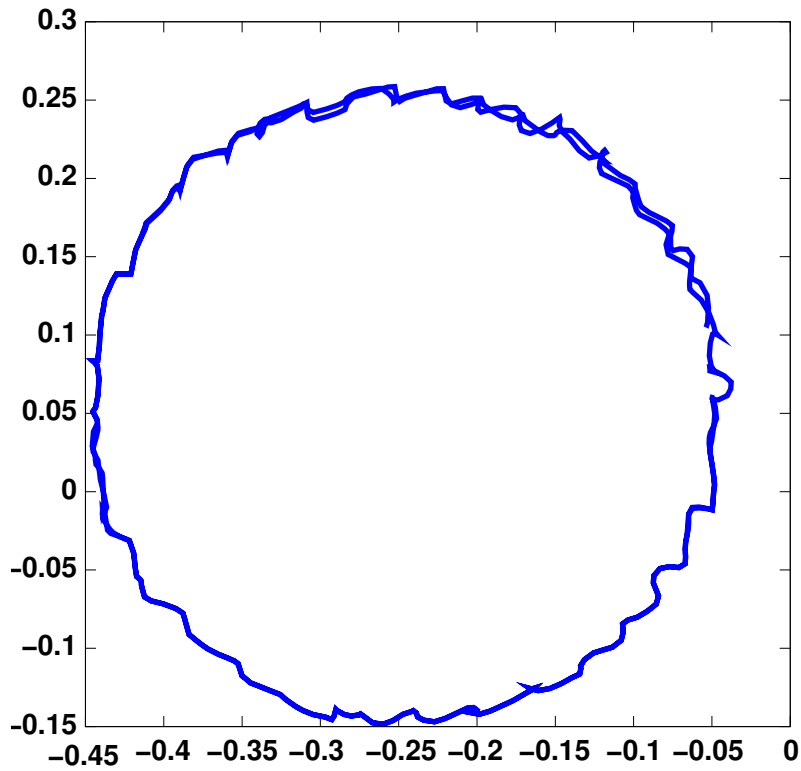


Figura 5.3: Esecuzione di una traiettoria circolare da parte del *Manus*.

Capitolo 6

Conclusioni

In questo progetto di tesi è stata sviluppata un'architettura software per il controllo di un robot manipolatore a bassa impedenza con compiti di assistenza a persone anziane e disabili. Il sistema di controllo è stato progettato, grazie anche all'utilizzo degli strumenti messi a disposizione dal *framework YARA*, con una struttura modulare, in grado di garantire rapidi ed agevoli aggiornamenti ed espansioni. La scomposizione dell'architettura in *moduli* ed *attività* é avvenuta nel rispetto della suddivisione naturale dettata dai blocchi funzionali omogenei presenti nel sistema: *comunicazione, generazione delle traiettorie, controllo del moto*.

Per prima cosa è stato sviluppato il modulo di comunicazione, per lo scambio di informazioni tra manipolatore e sistema di controllo, tramite protocollo seriale *CAN bus*. Le informazioni scambiate sono rappresentate dalle posizioni e dalle velocità delle variabili di giunto del manipolatore, e dal suo stato di funzionamento. Il risultato di questa prima parte di lavoro è stato un primo prototipo del sistema in grado di dialogare correttamente con il manipolatore. Il passo successivo é consistito nell'implementazione delle primitive per la generazione delle traiettorie che il manipolatore deve realizzare durante l'esecuzione del task di manipolazione. Il calcolo delle traiettorie viene fatto tramite l'interpolazione mediante polinomi cubici delle sequenze di via point fornite in ingresso al sistema. Il risultato é rappresentato da una serie di funzioni interpolanti che forniscono il riferimento spaziale-temporale utilizzato nella fase di controllo del moto. La terza fase ha riguardato proprio la realizzazione del controllore del moto, che utilizzando le informazioni prodotte dalle

funzioni interpolanti, e le informazioni di posizione provenienti dal manipolatore tramite l'anello di retroazione, calcola il set di velocità dei giunti in grado di far compiere al manipolatore il movimento voluto. Parallelamente allo sviluppo dei singoli moduli, sono stati curati gli aspetti riguardanti le comunicazioni ed il coordinamento dei componenti del sistema, in modo da garantire il funzionamento dell'architettura nel suo complesso.

Da ultimo, è stata realizzata una serie di test al fine di valutare l'efficienza del sistema di controllo nella realizzazione di diversi tipi di traiettorie. Nel complesso i risultati ottenuti sono stati soddisfacenti, soprattutto se si tiene conto che ad un manipolatore di questo tipo non si richiedono i livelli di precisione che possono invece essere garantiti dai manipolatori industriali. L'analisi della struttura del manipolatore e i risultati dei test hanno evidenziato come, allo stato attuale, uno dei limiti maggiori all'esecuzione di traiettorie più precise sia costituito dall'interfaccia di comunicazione del manipolatore, che limita le possibilità di controllo esterne.

Evoluzioni future del sistema

Per concludere discutiamo alcune possibili future evoluzioni del sistema:

- Sostituzione della lettura dei via point da file con un pianificatore del moto in grado di determinare, sulla base delle informazioni provenienti da un sistema di visione ed in funzione dell'obiettivo da raggiungere, la sequenza di punti dello spazio di lavoro che il manipolatore deve seguire.
- Gestione dinamica dei via point: attualmente tutti i punti del percorso da eseguire devono essere noti al momento dell'interpolazione, e non è possibile modificare la traiettoria con il manipolatore in movimento. Un uso più flessibile del sistema, anche in previsione delle evoluzioni proposte nel punto precedente, dovrebbe prevedere la possibilità di gestire dinamicamente l'arrivo di nuovi via point tramite una nuova interpolazione, che consenta di variare il movimento in atto, offrendo così la possibilità di eseguire operazioni di inseguimento.

- Sviluppo di una serie di procedure sul modello di *RCI-RCCL* [15], per il controllo ad alto livello del moto del manipolatore.
- Sostituzione dell'attuale controllore di tipo proporzionale con uno schema di controllo appositamente realizzato per sopperire alle limitate capacità di input offerte dall'elettronica del manipolatore.

Appendice A

I manipolatori robotici

A.1 Struttura di un manipolatore robotico

La struttura di un robot manipolatore può essere schematizzata con una serie di bracci connessi da giunti in modo da formare una catena cinematica. Lo schema più ricorrente è quello a *catena cinematica aperta*. In termini topologici una catena si definisce aperta quando la sequenza dei bracci che unisce i due estremi del manipolatore, la base e l'organo terminale, è unica; si parla invece di *catena cinematica chiusa*, quando esiste una sequenza di bracci connessi fra loro in modo tale da formare un anello chiuso. I giunti che garantiscono il collegamento meccanico dei bracci possono essere semplici, come i giunti *rotoidali* o quelli *prismatici*, o possono avere una struttura più complessa come i giunti sferici. Ad ogni giunto della prima categoria è associato un solo grado di mobilità: la rotazione tra bracci adiacenti nel caso dei giunti *rotoidali*, la traslazione nel caso di quelli *prismatici*. I giunti complessi possiedono invece più gradi di mobilità: ad esempio tre nel caso del giunto sferico; la loro struttura può comunque essere pensata come una successione di giunti semplici rotoidali o prismatici connessi tra loro da bracci di lunghezza nulla. Per questo motivo da ora in avanti si assumerà l'ipotesi di trattare esclusivamente manipolatori costituiti da giunti semplici.

In un manipolatore a catena cinematica aperta ad ogni giunto corrisponde un *grado di mobilità*, e dalla distribuzione dei gradi di mobilità lungo la struttura dipende il numero dei *gradi di libertà*. Un manipolatore per poter posizionare con

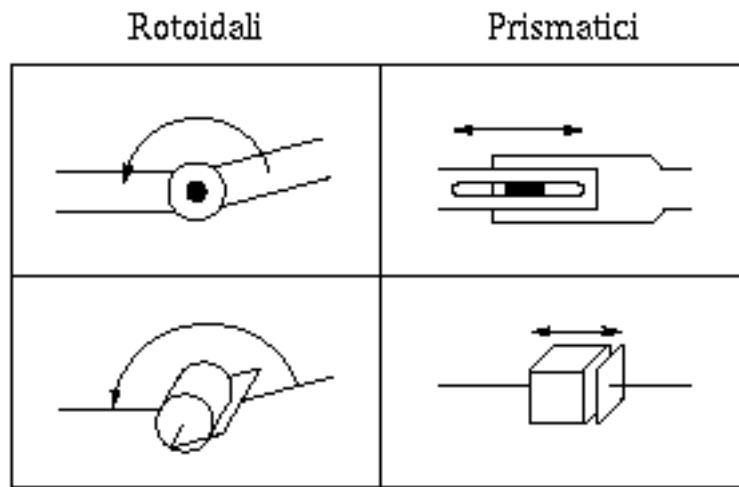


Figura A.1: Rappresentazione simbolica dei giunti rotoidali e prismatici.

un'orientazione arbitrario l'organo terminale in una qualsiasi posizione del proprio spazio di lavoro deve possedere almeno sei gradi di libertà: tre per il posizionamento e tre per l'orientazione. Una struttura che presenta un numero di gradi di mobilità maggiore del numero di gradi di libertà si definisce *cinematicamente ridondante*.

Lo *spazio di lavoro* del manipolatore a cui si è fatto riferimento in precedenza, rappresenta il volume totale dello spazio raggiungibile dall'organo terminale, ed è determinato dalla geometria del manipolatore, oltre che dai vincoli meccanici sui giunti. Spesso inoltre si fa un'ulteriore distinzione fra spazio di lavoro *raggiungibile* e spazio di *destrezza*; il primo è costituito dall'insieme dei punti che l'organo terminale può raggiungere con almeno un'orientazione, mentre il secondo è identificato dai punti che possono essere raggiunti con un'orientazione arbitrario, e rappresenta un sottoinsieme dello spazio raggiungibile.

In figura A.2 è riportata a titolo d'esempio la struttura cinematica schematizzata di un manipolatore sferico ed il relativo spazio di lavoro.

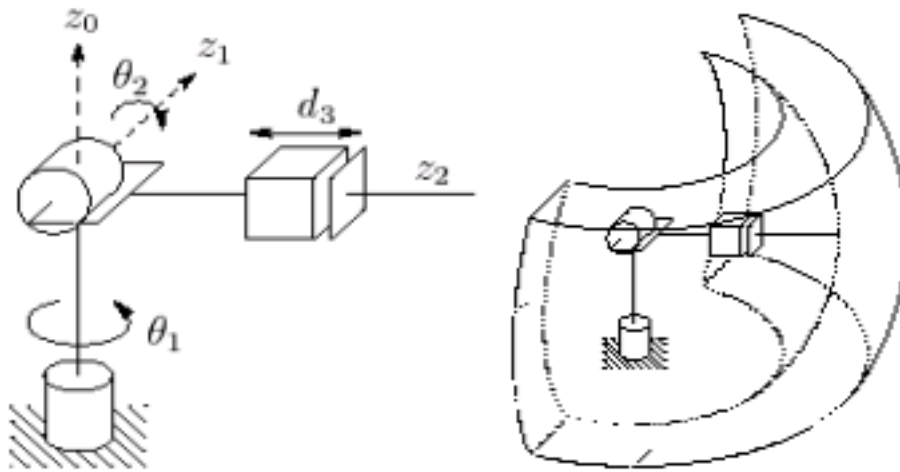


Figura A.2: Manipolatore sferico e spazio di lavoro.

A.2 Descrizioni e trasformazioni spaziali

Come si è visto in precedenza, dal punto di vista meccanico un *manipolatore* può essere rappresentato da una catena cinematica composta da corpi rigidi connessi tramite giunti. Un estremo della catena è vincolato ad una base, mentre all'estremo opposto è collocato l'organo terminale. Il moto complessivo del manipolatore si realizza tramite la composizione dei moti elementari di ciascun singolo braccio. Per fare questo è indispensabile fissare con precisione i sistemi di coordinate che forniscono la posizione e l'orientazione dei bracci rispetto ad un riferimento fisso, e le relazioni geometriche delle trasformazioni che intercorrono tra gli stessi sistemi di coordinate.

Posizione ed orientazione di un corpo rigido

La posizione di un punto p di un corpo rigido rispetto ad una terna di riferimento di coordinate $O-xyz$ è espressa dalla relazione

$$p = p_x x + p_y y + p_z z \quad (\text{A.1})$$

dove p_x, p_y, p_z rappresentano le componenti del vettore p lungo gli assi della terna.

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \quad (\text{A.2})$$

Per definire l'orientazione di un corpo è necessario esprimere i versori della terna ortonormale $O-x'y'z'$ solidale al corpo stesso in funzione dei versori della terna di riferimento $O-xyz$

$$\begin{aligned} x' &= x'_x x + x'_y y + x'_z z \\ y' &= y'_x x + y'_y y + y'_z z \\ z' &= z'_x x + z'_y y + z'_z z \end{aligned} \quad (\text{A.3})$$

I tre versori x', y', z' dell'equazione A.3 possono essere combinati nella *matrice di rotazione* \mathbf{R}

$$\mathbf{R} = \begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{bmatrix} \quad (\text{A.4})$$

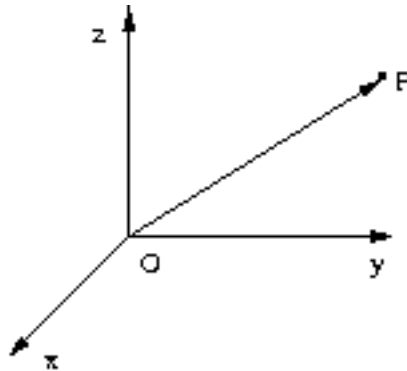
I vettori che formano le colonne di R sono versori ortonormali con modulo unitario vale quindi:

$$\mathbf{R}^T \mathbf{R} = \mathbf{I} \quad (\text{A.5})$$

con I la matrice identità, da cui si ricava:

$$\mathbf{R}^T = \mathbf{R}^{-1} \quad (\text{A.6})$$

La matrice di rotazione consente quindi di poter esprimere l'orientazione di un sistema di coordinate rispetto ad una terna di riferimento; rappresenta inoltre una



trasformazione tra le coordinate di uno stesso punto riferite a sistemi di coordinate diversi, ed è anche l'operatore matriciale utilizzato per la rotazione di un vettore attorno ad un generico asse nello spazio.

Vediamo brevemente le espressioni delle matrici per una rotazione di un dato angolo attorno agli assi coordinati x, y, z .

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{bmatrix} \quad (\text{A.7})$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \quad (\text{A.8})$$

$$\mathbf{R}_z(\gamma) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{A.9})$$

Una rotazione generica può essere interpretata come composizione di più rotazioni parziali ciascuna definita rispetto alla terna ottenuta dalla trasformazione precedente (*terna corrente*); la matrice complessiva si ricava moltiplicando da sinistra a destra le singole matrici seguendo l'ordine imposto dalla sequenza di rotazione.

$$\mathbf{R}_2^0 = \mathbf{R}_1^0 \mathbf{R}_1^2 \quad (\text{A.10})$$

In alternativa le rotazioni parziali possono essere definite facendo sempre riferimento ad una *terna fissa*: in questo caso la matrice complessiva si ottiene moltiplicando da destra a sinistra le singole matrici di rotazione.

$$\mathbf{R}_2^0 = \mathbf{R}_1^2 \mathbf{R}_1^0 \quad (\text{A.11})$$

I nove elementi di una matrice di rotazione forniscono una descrizione ridondante dell'orientazione di una terna, sono infatti sufficienti solamente tre parametri per definire un'orientazione: a tal proposito esistono diverse convenzioni che consentono di ricavare l'espressione della una generica matrice di rotazione a partire dai valori dei tre angoli assegnati φ , θ e ψ . Gli *angoli di Eulero ZYZ* ad esempio consentono di ottenere \mathbf{R} come composizione di rotazioni rispetto alla terna corrente compiendo prima una rotazione della terna di partenza di un angolo ϕ attorno all'asse z , successivamente una rotazione di θ attorno all'asse y' ottenuto dalla trasformazione precedente, ed infine un'ulteriore rotazione di ψ attorno all'asse z''

$$\mathbf{R} = \mathbf{R}_z(\phi) \mathbf{R}_{y'}(\theta) \mathbf{R}_{z''}(\psi) \quad (\text{A.12})$$

Vediamo infine com'è possibile rappresentare un punto \mathbf{p} in due differenti sistemi di coordinate. Dalla figura A.2 si ricava

$$\mathbf{p}^0 = \mathbf{0}_1^0 + \mathbf{R}_1^0 \mathbf{p}^1 \quad (\text{A.13})$$

e la relazione inversa

$$\mathbf{p}^1 = -\mathbf{R}_1^0 \mathbf{0}_1^0 + \mathbf{R}_1^0 \mathbf{p}^1 \quad (\text{A.14})$$

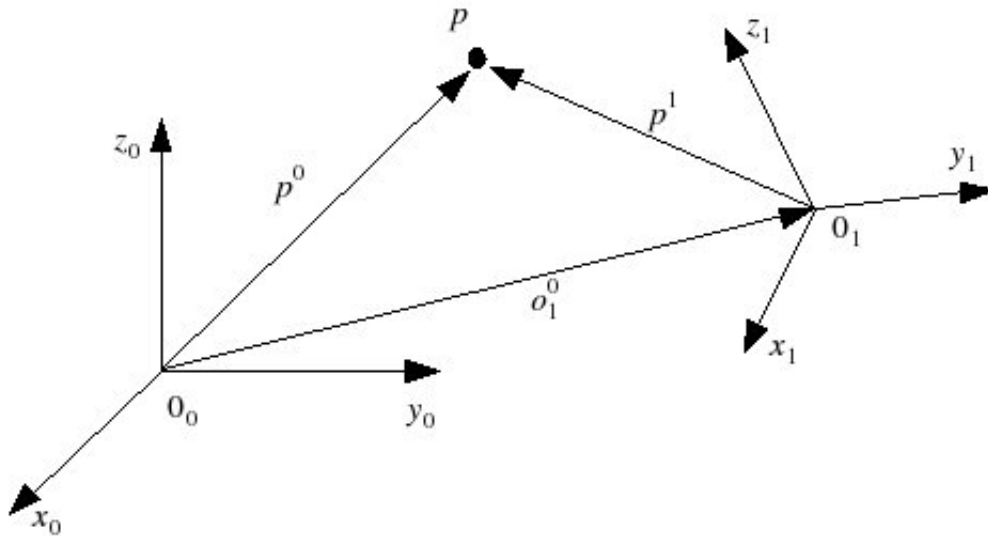


Figura A.3: Rappresentazione di un punto in 2 diverse terne.

Dalle relazioni A.13 e A.14, utilizzando per i vettori di posizione la loro rappresentazione omogenea ottenuta aggiungendo una quarta componente unitaria, si può ricavare la *matrice di trasformazione omogenea* A_1^0 che esprime in forma compatta la trasformazione di coordinate fra le due terne, trasformazione intesa come operazione di traslazione e rotazione. Vedremo nel paragrafo A.3 come la A.15 sia ampiamente utilizzata nel calcolo della funzione cinematica diretta dei manipolatori.

$$A_1^0 = \begin{bmatrix} R_0^1 & o_0^1 \\ 0 & 1 \end{bmatrix} \quad (A.15)$$

A.3 Cinematica diretta

Il problema cinematico diretto consiste nel determinare la posizione e l'orientazione dell'organo terminale di un manipolatore in funzione dei valori delle sue *variabili di giunto*, dove per variabile di giunto si intende l'angolo fra due bracci consecutivi nel caso di giunti rotazionali o l'estensione del braccio nel caso di giunti prismatici.

Definita una terna di riferimento base $O_b-x_b y_b z_b$, la funzione di cinematica diretta è espressa dalla matrice di trasformazione omogenea A.16

$$\mathbf{T}_e^b(\mathbf{q}) = \begin{bmatrix} n_e^b(q) & s_e^b(q) & a_e^b(q) & p_e^b(q) \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{A.16})$$

dove \mathbf{q} è il vettore ($n \times 1$) delle variabili di giunto, \mathbf{n}_e , \mathbf{s}_e , \mathbf{a}_e sono i versori di una terna solidale all'organo terminale del manipolatore (*terna utensile*) e \mathbf{p}_e è il vettore posizione della 'origine della terna utensile rispetto all'origine della terna base.

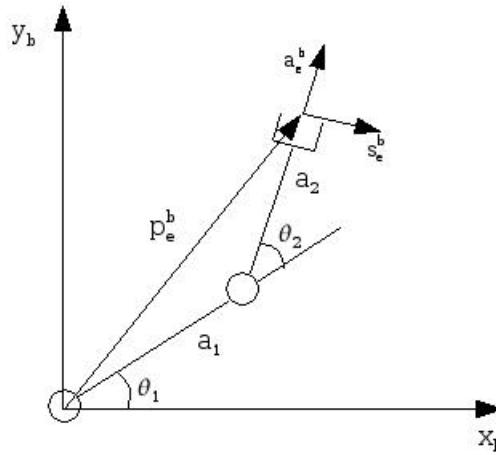


Figura A.4: Schema di un manipolatore planare a due giunti.

Un primo approccio alla risoluzione del problema cinematico diretto può essere basato sull'analisi della struttura geometrica del manipolatore. In figura A.3 è riportata la schematizzazione di un manipolatore planare con due giunti rotazionali. Dalla disposizione delle terne, applicando alcune relazioni trigonometriche si ottiene l'espressione della matrice di trasformazione omogenea

$$\mathbf{T}_e^b(\mathbf{q}) = \begin{bmatrix} 0 & \sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) & a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) \\ 0 & -\cos(\theta_1 + \theta_2) & \sin(\theta_1 + \theta_2) & a_1 \sin(\theta_1) + a_2 \sin(\theta_1 + \theta_2) \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{A.17})$$

Una soluzione geometrica come quella appena vista è concretamente applicabile solo a manipolatori con struttura semplice. Per manipolatori aventi una complessità ed un numero di giunti maggiori è preferibile utilizzare una soluzione meno diretta ma basata su procedure standard più generali. Per i manipolatori a *catena cinematica aperta* la struttura stessa suggerisce una procedura operativa per la soluzione del problema cinematico diretto. Consideriamo un manipolatore a *catena aperta* formato da $n + 1$ bracci connessi da n giunti; consideriamo per convenzione il braccio 0 fisso a terra. A questo punto dopo aver fissato una terna di riferimento solidale a ciascun braccio per tutti i bracci da 0 a n (figura A.5) possiamo esprimere la trasformazione di coordinate complessiva della terna n rispetto alla 0 come:

$$\mathbf{T}_n^0(\mathbf{q}) = \mathbf{A}_1^0(\mathbf{q}_1)\mathbf{A}_2^1(\mathbf{q}_2) \cdots \mathbf{A}_n^{n-1}(\mathbf{q}_n) \quad (\text{A.18})$$

La funzione cinematica diretta si ottiene quindi ricorsivamente tramite la moltiplicazione delle matrici di trasformazione omogenea $A_i^{i-1}(q_i)$ (per $i = 1, \dots, n$) ognuna delle quali è funzione di una sola variabile di giunto.

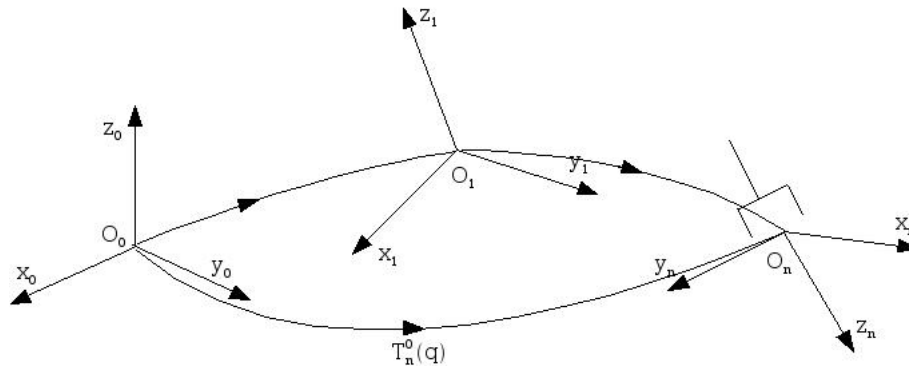


Figura A.5: Trasformazione di coordinate in una catena cinematica aperta.

A.3.1 Convenzione di Denavit-Hartenberg

La convenzione di *Denavit-Hartenberg* [16] fornisce un metodo generale e sistematico per esprimere la posizione e l'orientazione relativi di due bracci consecutivi. Con riferimento alla figura A.6 si opera nel seguente modo:

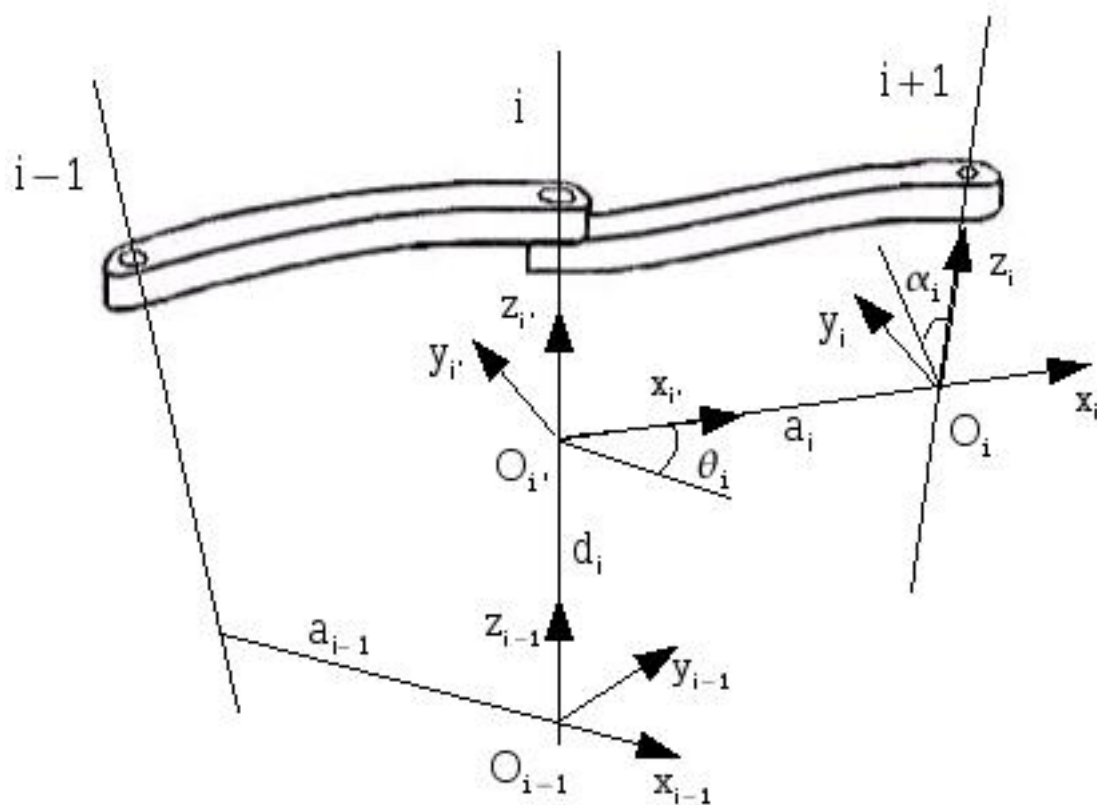


Figura A.6: Parametri cinematici di Denavit-Hartenberg.

- si sceglie l'asse z_i giacente lungo l'asse del giunto $i + 1$
- si individua O_i all'intersezione dell'asse z_i con la normale comune agli assi z_{i-1} e z_i , e con $O_{i'}$ si indica l'intersezione della normale comune con z_{i-1} ;
- si sceglie l'asse x_i diretto lungo la normale comune agli assi z_{i-1} e z_i con verso positivo dal giunto i al giunti $i + 1$;
- si sceglie l'asse y_i in modo da completare una terna levogira.

Nei seguenti casi la convenzione di Denavit-Hartenberg non fornisce una definizione univoca della terna:

- nella terna 0 solo la direzione dell'asse z_0 risulta specificata, O_0 e x_0 possono essere scelti arbitrariamente
- non essendovi un giunto $n + 1$ nella terna n l'asse z_n non è univocamente definito, mentre l'asse x_n deve essere normale all'asse z_{n-1} ;
- quando due assi consecutivi sono paralleli, in quanto la normale comune tra di loro non è univocamente definita
- quando due assi consecutivi si intersecano, in quanto il verso di x_i è arbitrario;
- quando il giunto i è prismatico, in questo caso l'unica direzione determinata è quella dell'asse z_i .

Nei casi di indeterminazione si può semplificare la procedura ricercando ad esempio delle condizioni di allineamento tra gli assi delle terne. A questo punto la posizione e l'orientazione della terna i rispetto alla terna $i - 1$ risultano completamente definite dai seguenti parametri:

a_i distanza di O_i da $O_{i'}$,

d_i coordinata su z_{i-1} di $O_{i'}$,

α_i angolo intorno all'asse x_i tra l'asse z_{i-1} e l'asse z_i valutato positivo in senso antiorario,

θ_i angolo intorno all'asse z_{i-1} tra l'asse x_{i-1} e l'asse x_i valutato positivo in senso antiorario.

a_i e α_i sono sempre costanti e dipendono solamente dalla geometria di connessione dei giunti consecutivi tramite il braccio. Tra d_i e α_i solo uno è variabile in base al tipo di giunto utilizzato nella connessione dei bracci i e $i - 1$; in particolare:

- se il giunto è *rotoidale* la variabile è θ_i ,
- se il giunto è *prismatico* la variabile è d_i .

A questo punto si può ricavare la trasformazione che lega la terna i ed $i - 1$ secondo i seguenti passi:

- si parte da una terna coincidente con la terna $i - 1$;
- si trasla la terna di d_i lungo l'asse z_{i-1} e la si ruota di θ_i attorno allo stesso asse; questa operazione che è descritta dalla trasformazione omogenea [A.19](#) porta da una sovrapposizione della terna $i - 1$ ed i'

$$\mathbf{A}_i^{i-1} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & 0 \\ \sin(\theta_i) & \cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{A.19})$$

- si trasla ora la terna ottenuta nei passaggio precedente di a_i lungo l'asse x_i e la si ruota di α_i attorno allo stesso asse portando alla sovrapposizione fra la terna in questione e la terna i come rappresentato dalla matrice [A.20](#)

$$\mathbf{A}_i^{i'} = \begin{bmatrix} 1 & 0 & 0 & \alpha_i \\ 0 & \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{A.20})$$

- la trasformazione di coordinate complessiva si ottiene moltiplicando le singole trasformazioni come segue:¹

¹Nell'equazione [A.21](#) s e c sono rispettivamente un'abbreviazione per \sin e \cos .

$$\mathbf{A}_i^{i-1}(\mathbf{q}_i) = \mathbf{A}_i^{i-1} \mathbf{A}_i^i = \begin{bmatrix} c(\theta_i) & -s(\theta_i)c(\alpha_i) & s(\theta_i)s(\alpha_i) & a_i c(\theta_i) \\ s(\theta_i) & c(\theta_i)c(\alpha_i) & -c(\theta_i)s(\alpha_i) & a_i s(\theta_i) \\ 0 & s(\alpha_i) & c(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{A.21})$$

Si può riassumere la procedura appena vista basata sulla convenzione di Denavit-Hartenberg nel seguente algoritmo che consente la derivazione della funzione cinematica diretta per qualsiasi manipolatore a catena aperta.

1. Individuare e numerare consecutivamente gli assi dei giunti ed assegnare rispettivamente le direzioni agli assi z_0, \dots, z_n .
2. Posizionare la terna 0 sull'asse z_0 e scegliere x_0 e y_0 in modo da formare una terna levogira.
Eeguire i passi **3**, **4** e **5** per $i = 1, \dots, n - 1$:
3. Individuare l'origine O_i all'intersezione di z_i con la normale comune agli assi z_{i-1} e z_i . Se gli assi in questione sono paralleli ed il giunto i è rotoideale, posizionare O_i in modo da annullare d_i ; Se invece il giunto è prismatico scegliere O_i in corrispondenza di una posizione di riferimento della corsa del giunto.
4. Fissare l'asse x_i diretto lungo la normale comune agli assi z_{i-1} e z_i con verso positivo dal giunto i al giunto $i + 1$.
5. Fissare y_i in modo da ottenere una terna levogira.
6. Fissare la terna n allineando z_n lungo la direzione di z_{n-1} se il giunto è rotoideale, scegliere invece z_n in modo arbitrario nel caso di giunto prismatico.
7. Costruire per ciascun giunto la tabella dei parametri $a_i, d_i, \alpha_i, \theta_i$ e calcolare sulla base di questi parametri le matrici di trasformazione omogenea $\mathbf{A}_i^{i-1}(\mathbf{q}_i)$
8. Calcolare tramite [A.18](#) la trasformazione omogenea $T_n^0(q)$ che fornisce posizione ed orientazione della terna n rispetto alla terna 0.

Una volta assegnate la terna di base T_0^b e quella di utensile T_e^n la posizione di quest'ultima rispetto alla prima si ottiene da:

$$T_e^b(\mathbf{q}) = T_0^b T_n^0 T_e^n \quad (\text{A.22})$$

Vediamo ora a titolo di esempio il calcolo della funzione cinematica diretta tramite la convenzione di Denavit-hartenberg del manipolatore planare a due giunti trattato in precedenza.

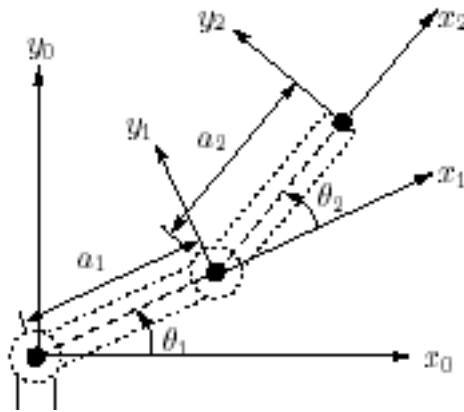


Figura A.7: Manipolatore planare a due giunti .

Applicando la procedura vista sopra si ottiene la seguente tabella dei parametri di Denavit-Hartenberg dove θ_1 e θ_2 sono variabili visto che entrambi i giunti sono rotoidali.

Braccio	a_i	α_i	d_i	θ_i
1	a_1	0	0	θ_1
2	a_2	0	0	θ_2

Dall'equazione A.21 si ricavano le due matrici di trasformazione:

$$\mathbf{A}_1^0 = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & a_1 \cos(\theta_1) \\ \sin(\theta_1) & \cos(\theta_1) & 0 & a_1 \sin(\theta_1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{A.23})$$

$$\mathbf{A}_2^1 = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & a_2 \cos(\theta_2) \\ \sin(\theta_2) & \cos(\theta_2) & 0 & a_2 \sin(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{A.24})$$

A questo punto per ottenere la matrice di trasformazione complessiva del manipolatore basta moltiplicare fra loro le matrici A.23 e A.24:

$$\mathbf{T}_2^0 = \mathbf{A}_1^0 \mathbf{A}_2^1 = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & a_2 \cos(\theta_2) \\ \sin(\theta_2) & \cos(\theta_2) & 0 & a_2 \sin(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{A.25})$$

A.3.2 Convenzione di Denavit-Hartenberg modificata

Esiste anche una variante della procedura vista nel paragrafo A.3.1 nota col nome di *convenzione di Denavit-Hartenberg modificata* [17]. Dal momento che questa seconda convenzione non si discosta concettualmente molto dalla precedente ci si limita ad enunciare i passi dell'algoritmo di identificazione dei sistemi di riferimento solidali ai bracci del manipolatore e l'espressione analitica della matrice di trasformazione che ne consegue.

1. Individuare ed enumerare consecutivamente gli assi dei giunti ed assegnare le direzioni agli assi z_i .

Eseguire i passi 2 , 3, 4 e 5 per ogni coppia di assi adiacenti:

2. Individuare l'origine O_i all'intersezione dell'asse i con la normale comune.
3. Fissare l'asse x_i lungo la perpendicolare comune, o se gli assi si intersecano , fissare la direzione di x_i in modo da essere perpendicolare al piano contenente i due assi.
4. Fissare y_i in modo da ottenere una terna levogira.
5. Assegnare il sistema di riferimento 0 in modo da farlo coincidere con la terna 1 quando la prima variabile di giunto è nulla. Per la terna n scegliere x_n liberamente.

Questa procedura porta all'individuazione dei seguenti parametri:

a_i distanza fra l'origine della terna i e $i + 1$ lungo x_i

α_i angolo fra z_i e z_{i+1} attorno all'asse x_i

d_i distanza fra gli assi x_{i-1} e x_i misurata lungo z_i

θ_i angolo fra gli assi x_{i-1} e x_i attorno all'asse z_i

La rotazione di α_{i-1} attorno all'asse x , seguita dalla traslazione a_{i-1} lungo x , seguita da una rotazione di θ_i attorno all'asse z e da una traslazione di d_i lungo lo stesso asse porta alla seguente espressione analitica per la matrice di trasformazione omogenea \mathbf{A}_i^{i-1}

$$\mathbf{A}_i^{i-1} = \begin{bmatrix} c(\theta_i) & -s(\theta_i) & 0 & a_{i-1} \\ s(\theta_i)c(\alpha_{i-1}) & c(\theta_i)c(\alpha_{i-1}) & -s(\alpha_{i-1}) & -d_i s(\alpha_{i-1}) \\ s(\theta_i)s(\alpha_{i-1}) & c(\theta_i)s(\alpha_{i-1}) & c(\alpha_{i-1}) & d_i c(\alpha_{i-1}) \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{A.26})$$

A.4 Cinematica inversa

Nella sezione precedente si è visto come sia possibile determinare la posizione dell'organo terminale del manipolatore in funzione delle variabili di giunto tramite l'equazione A.18. La *cinematica inversa* affronta il problema opposto, ossia quello della determinazione dei valori delle variabili di giunto una volta che siano noti posizione ed orientazione dell'organo terminale. La soluzione al problema cinematico inverso è necessaria in tutte quelle circostanze in cui si vuol far eseguire all'organo terminale un dato moto le cui specifiche sono assegnate nello *spazio operativo* e necessitano di essere espresse in termini delle variabili di giunto.

A.4.1 Formulazione del problema cinematico inverso

In termini matematici il problema può essere enunciato come segue. Data una trasformazione omogenea 4x4

$$\mathbf{H} = \begin{bmatrix} \mathbf{R} & 0 \\ 0 & 1 \end{bmatrix} \quad (\text{A.27})$$

con R matrice di rotazione 3x3 si devono trovare le soluzioni dell'equazione

$$\mathbf{T}_n^0(q_1, \dots, q_n) = \mathbf{H} \quad (\text{A.28})$$

dove H rappresenta la posizione e l'orientazione dell'organo terminale del manipolatore, mentre q_1, \dots, q_n sono le variabili di giunto che devono essere determinate.

A.4.2 Risolvibilità del problema cinematico inverso

A differenza del problema cinematico diretto in cui la sola conoscenza delle variabili di giunto è sufficiente a garantire l'univocità e la correttezza della soluzione, il problema cinematico inverso presenta una complessità maggiore dovuta alle seguenti ragioni:

- possono non esistere soluzioni.
- si possono avere soluzioni multiple.
- si possono avere infinite soluzioni come nel caso di manipolatori ridondanti.
- le equazioni da risolvere sono generalmente non lineari e non sempre è possibile trovare una soluzione analitica al problema.

Vediamo più nel dettaglio le problematiche della cinematica inversa.

Esistenza della soluzione

Il problema dell'esistenza o meno di una soluzione al problema cinematico inverso è necessariamente collegato al concetto di *spazio di lavoro* visto in [A.1](#); una soluzione può infatti esistere solamente se la posizione specificata da H nell'equazione [A.27](#) si trova all'interno dello spazio di lavoro del manipolatore, e per la precisione all'interno dello spazio di destrezza. Nella valutazione dello spazio di lavoro bisogna tenere in considerazione anche le eventuali restrizioni di tipo meccanico sui giunti che determinano una riduzione dello spazio stesso, con la conseguenza che una soluzione matematicamente corretta può essere fisicamente irrealizzabile.

Soluzioni multiple

Un altro possibile problema è la presenza di soluzioni multiple (figura [A.8](#)), che in un manipolatore a sei gradi di mobilità e privo di fine-corsa meccanici possono essere fino a sedici. Questo fatto implica che il sistema di controllo del manipolatore deve essere in grado di scegliere una soluzione fra tutte quelle possibili in base a qualche criterio. Un criterio ragionevole può essere quello di scegliere la soluzione più vicina all'attuale configurazione del manipolatore in modo da minimizzare lo spostamento di ciascun giunto. Nel caso molto comune di manipolatori costituiti da tre bracci di grandi dimensioni seguiti da tre bracci più piccoli la scelta della soluzione ricade spesso su quella che privilegia lo spostamento dei bracci piccoli rispetto a quelli grandi. La presenza di ostacoli all'interno dello spazio di lavoro può richiedere la necessità di abbandonare i criteri di scelta fin qui visti al fine di evitare

la collisione. In generale è necessario poter disporre di tutte le soluzioni possibili, e questo ha delle ripercussioni sulla scelta del metodo di risoluzione (A.4.2.1).

Il numero di soluzioni al problema cinematico inverso dipende dal numero di giunti del manipolatore e dai parametri $\alpha_i, a_i, d_i, \theta_i$. In generale, maggiore è il numero di parametri non nulli, maggiore sono le soluzioni possibili. Il caso limite per quanto riguarda il problema delle soluzioni multiple lo si ha con i manipolatori ridondanti, che come si è già visto, sono caratterizzati da un numero di gradi di mobilità maggiore di quello dei gradi di libertà, con la conseguenza di avere infinite soluzioni del problema cinematico inverso. La soluzione in questo caso si trova fissando uno dei gradi di mobilità.

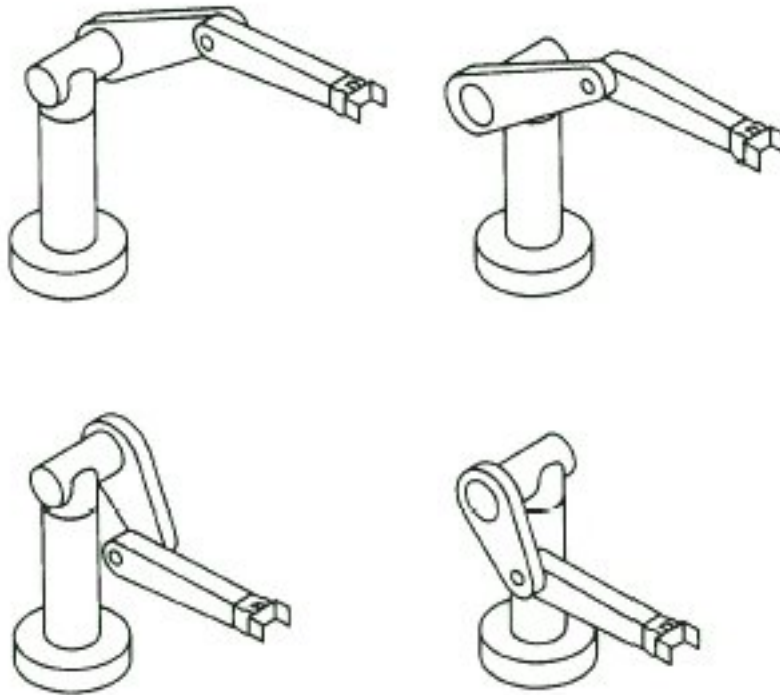


Figura A.8: Quattro possibili soluzioni al problema cinematico inverso.

A.4.2.1 Metodi risolutivi

Nell'ambito della cinematica inversa un manipolatore è detto *risolvibile* se è possibile determinare tutti i possibili insiemi delle variabili di giunto corrispondenti ad una data posizione ed orientazione.

Trovare una soluzione al problema in *forma chiusa (analitica)* significa determinare una relazione esplicita del tipo:

$$q_k = f_k(h_{11}, \dots, h_{34}), \quad k = 1, \dots, n \quad (\text{A.29})$$

La non linearità dell'equazione non garantisce però la possibilità di trovare per tutti i manipolatori una soluzione analitica, obbligando così all'utilizzo di una *soluzione numerica*.

Le soluzioni in forma chiusa sono preferibili a quelle in forma numerica principalmente per due ragioni:

- la natura iterativa delle soluzioni numeriche implica tempi risolutivi maggiori e spesso incompatibili con le frequenze a cui operano i controllori dei manipolatori.
- l'uso della della forma analitica consente la determinazione delle eventuali soluzioni multiple e consente quindi di fissare un criterio per scegliere una specifica soluzione fra tutte quelle disponibili; l'approccio numerico anche se assicura la risolvibilità del problema per tutti i manipolatori con giunti prismatici o rotoidali a catena cinematica aperta con un totale di sei gradi di libertà, non consente la gestione delle soluzioni multiple.

A.4.3 Disaccoppiamento cinematico

In generale non è possibile risolvere in forma chiusa il problema cinematico inverso per un manipolatore a sei gradi di libertà ma fanno eccezione le strutture cinematiche che rispettano una delle seguenti condizioni:

- tre assi di giunti rotoidali adiacenti che si intersecano in un punto.

- tre assi di giunti rotoidali adiacenti paralleli.

All'interno della struttura dei manipolatori che soddisfano una delle due condizioni indicate sopra è possibile individuare un punto opportuno la cui posizione è esprimibile in funzione di un numero ridotto di variabili di giunto. La presenza di tale punto consente di disaccoppiare il problema cinematico inverso in due sotto-problemi più semplici, quello relativo alla determinazione della posizione e quello relativo alla determinazione dell'orientazione [18].

Un caso tipico di struttura cinematica in cui è possibile applicare il disaccoppiamento cinematico è il manipolatore con polso sferico in cui gli ultimi tre giunti soddisfano la prima delle due condizioni viste sopra. In questo caso il punto w che gode della proprietà di disaccoppiamento coincide con il punto di intersezione degli assi dei tre giunti di polso (figura A.4.3). Una volta fissate la posizione p e l'orientazione R della terna utensile, la posizione del centro del polso è individuata dall'equazione:

$$\mathbf{p}_w = \mathbf{p} - \mathbf{d}_6 \mathbf{a} \quad (\text{A.30})$$

che dipende esclusivamente dalle variabili di giunto dei primi tre bracci.

Complessivamente il calcolo della cinematica inversa per un manipolatore dotato di polso sferico si esegue con la seguente procedura:

- calcolare tramite l'equazione A.30 la posizione del polso $p_w(q_1, q_2, q_3)$.
- risolvere la cinematica inversa per (q_1, q_2, q_3) .
- calcolare $R_0^3(q_1, q_2, q_3)$.
- calcolare $R_6^3(\theta_4, \theta_5, \theta_6) = R_3^{0T} R$
- risolvere la cinematica inversa per l'orientazione $(\theta_4, \theta_5, \theta_6)$

A.4.4 Soluzione del manipolatore antropomorfo

A seguire viene mostrata la soluzione al problema cinematico inverso di un manipolatore antropomorfo con polso sferico (figura A.4.3), la stessa struttura cinematica del *manus* oggetto di questa tesi o del *PUMA*. Siano dati la posizione della terna

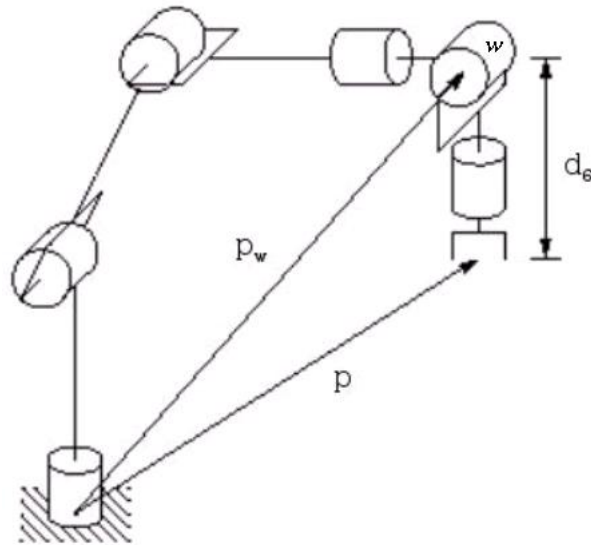


Figura A.9: Manipolatore con polso sferico.

utensile p e la matrice di rotazione R della stessa e p_w la posizione del centro del polso sferico:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (\text{A.31})$$

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \quad (\text{A.32})$$

Per l'equazione A.30 le coordinate del centro del polso risultano:

$$p_{w_x} = p_x - d_6 r_{13} \quad (\text{A.33})$$

$$p_{w_y} = p_y - d_6 r_{23} \quad (\text{A.34})$$

$$p_{w_z} = p_z - d_6 r_{33} \quad (\text{A.35})$$

Un possibile insieme di variabili di giunto è dato da:

$$\theta_1 = \text{Atan}(p_{w_x}, p_{w_y}) \quad (\text{A.36})$$

$$\theta_2 = \text{Atan} \left(\sqrt{p_{w_x}^2 + p_{w_y}^2 - d^2}, p_{w_z} \right) - \text{Atan} (a_2 + a_3 c_3, a_3 s_3) \quad (\text{A.37})$$

$$\theta_3 = \text{Atan} \left(D, \pm \sqrt{1 - D^2} \right)$$

$$\text{dove} \quad D = \frac{p_{w_x}^2 + p_{w_y}^2 - d^2 + p_{w_z}^2 - a_2^2 - a_3^2}{2a_2 a_3} \quad (\text{A.38})$$

$$\theta_4 = \text{Atan}(c_1 c_{23} r_{13} + s_1 c_{23} r_{23} + s_{23} r_{33},$$

$$-c_1 s_{23} r_{13} - s_1 s_{23} r_{23} + c_{23} r_{33}) \quad (\text{A.39})$$

$$\theta_5 = \text{Atan} \left(s_1 r_{13} - c_1 r_{23}, \pm \sqrt{1 - (s_1 r_{13} - c_1 r_{23})^2} \right) \quad (\text{A.40})$$

$$\theta_6 = \text{Atan} (-s_1 r_{11} + c_1 r_{21}, s_1 r_{12} - c_1 r_{22}) \quad (\text{A.41})$$

A.5 Cinematica differenziale

Nei paragrafi A.3 e A.4 sono state introdotte le relazioni che legano i valori delle variabili di giunto alla posizione ed all'orientazione dell'organo terminale di un dato manipolatore. La *cinematica differenziale* descrive invece i legami tra le velocità dei giunti e la velocità angolare e lineare dello stesso organo terminale. Tale relazione può essere descritta da una matrice di trasformazione dipendente dalla configurazione del manipolatore nota con il nome di *Jacobiano geometrico*, ma può anche essere ricavata differenziando la funzione cinematica diretta, ottenendo in questo secondo caso lo *Jacobiano analitico*, che in generale differisce da quello geometrico.

Jacobiano geometrico

Sia nota la funzione cinematica diretta per un manipolatore ad n gradi di mobilità

$$\mathbf{T}(\mathbf{q}) = \begin{bmatrix} \mathbf{R}(\mathbf{q}) & p(q) \\ 0 & 1 \end{bmatrix} \quad (\text{A.42})$$

L'obbiettivo della cinematica differenziale è quello di esprimere il vettore $\dot{\mathbf{p}}$ delle velocità lineari e ω delle velocità angolari in funzione delle velocità di giunto $\dot{\mathbf{q}}$.

$$\dot{\mathbf{p}} = \mathbf{J}_P(\mathbf{q})\dot{\mathbf{q}} \quad (\text{A.43})$$

$$\omega = \mathbf{J}_O(\mathbf{q})\dot{\mathbf{q}} \quad (\text{A.44})$$

Le equazioni A.43 e A.44 possono essere riscritte nella forma più compatta

$$\mathbf{v} = \begin{bmatrix} \dot{\mathbf{p}} \\ \omega \end{bmatrix} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \quad (\text{A.45})$$

La matrice \mathbf{J} rappresenta lo *Jacobiano geometrico* del manipolatore: il numero delle sue righe corrisponde al numero di gradi di libertà, mentre il numero delle colonne corrisponde al numero dei giunti.

\mathbf{J} può essere partizionata come segue:

$$\mathbf{J} = \begin{bmatrix} J_P \\ J_O \end{bmatrix} = \begin{bmatrix} J_{P_1} & \dots & J_{P_n} \\ \vdots & & \vdots \\ J_{O_1} & \dots & J_{O_n} \end{bmatrix} \quad (\text{A.46})$$

I termini del tipo $\dot{\mathbf{q}}\mathbf{J}_{P_i}$ rappresentano il contributo del giunto i -esimo sulla velocità lineare dell'organo terminale, mentre i termini del tipo $\dot{\mathbf{q}}\mathbf{J}_{O_i}$ rappresentano il contributo apportato alla velocità angolare. Il calcolo dello Jacobiano geometrico consiste proprio nel determinare gli effetti di ciascun singolo giunto sulla velocità complessiva.

Nel caso di giunti rotoidali la velocità di rotazione ω del braccio $i + 1$ equivale

a quella del braccio i più la componente introdotta dal giunto $i + 1$

$$\omega_{i+1} = \mathbf{R}_i^{i+1} \omega_i + \dot{\mathbf{q}}_{i+1} \mathbf{z}_{i+1} \quad (\text{A.47})$$

per la velocità lineare $\dot{\mathbf{p}}$ vale l'espressione analoga

$$\dot{\mathbf{p}}_{i+1} = \mathbf{R}_i^{i+1} (\dot{\mathbf{p}}_i + \omega_i \times \mathbf{r}_{i+1}) \quad (\text{A.48})$$

con r_{i+1} il vettore posizione della terna $i + 1$ rispetto alla terna i .

Nel caso in cui il giunto $i + 1$ sia prismatico valgono le relazioni:

$$\omega_{i+1} = \mathbf{R}_i^{i+1} \omega_i \quad (\text{A.49})$$

$$\dot{\mathbf{p}}_{i+1} = \mathbf{R}_i^{i+1} (\dot{\mathbf{p}}_i + \omega_i \times \mathbf{r}_{i+1}) + \dot{\mathbf{d}}_{i+1} \mathbf{z}_{i+1} \quad (\text{A.50})$$

Singularità cinematiche

Lo Jacobiano geometrico introdotto in A.5 dipende in generale dai valori assunti dalle variabili di giunto; possono esistere delle configurazioni del manipolatore in corrispondenza delle quali il rango di \mathbf{J} diminuisce: tale configurazioni sono definite *singularità cinematiche*. La determinazione delle singularità di una data struttura è di notevole importanza per i seguenti motivi:

- in corrispondenza di una configurazione singolare possono esistere infinite soluzioni al problema cinematico inverso.
- una struttura cinematica che si trova in una configurazione singolare va incontro ad una perdita di gradi di libertà.
- in prossimità delle singularità, a velocità ridotte nello spazio operativo possono corrispondere velocità elevate nello spazio dei giunti.

Tutti i manipolatori presentano delle singularità sui bordi del proprio spazio di lavoro raggiungibile in corrispondenza di configurazioni che vedono la struttura completamente distesa o ripiegata su sè stessa. In generale queste singularità non comportano gravi difficoltà in fase di controllo. Ben più problematiche sono invece

le singolarità che si trovano all'interno dello spazio di lavoro in quanto possono interferire con le traiettorie pianificate nello spazio operativo (paragrafo 2.1) rendendo impossibile la determinazione delle velocità dei giunti $\dot{\mathbf{q}}$ partendo dalla velocità \mathbf{v} specificata nello spazio cartesiano.

$$\dot{\mathbf{q}} = \mathbf{J}^{-1}(\mathbf{q})\mathbf{v} \quad (\text{A.51})$$

Jacobiano analitico

Se la posizione e l'orientazione dell'organo terminale del manipolatore sono dati nello spazio operativo utilizzando una rappresentazione minima (ad esempio gli angoli di Eulero visti in A.2) è possibile utilizzare lo *Jacobiano analitico* al posto di quello geometrico. La velocità di traslazione $\dot{\mathbf{p}}$ della terna utensile è vale

$$\dot{\mathbf{p}} = \frac{\partial \mathbf{p}}{\partial \mathbf{q}} \dot{\mathbf{q}} = \mathbf{J}_p(\mathbf{q}) \dot{\mathbf{q}} \quad (\text{A.52})$$

Se ϕ è il vettore di tre elementi che costituiscono la rappresentazione minima dell'orientazione minimo della terna utensile, il vettore delle velocità angolari è espresso da:

$$\dot{\phi} = \frac{\partial \phi}{\partial \mathbf{q}} \dot{\mathbf{q}} = \mathbf{J}_\phi(\mathbf{q}) \dot{\mathbf{q}} \quad (\text{A.53})$$

In generale $\dot{\phi}$ differisce dal vettore delle velocità angolari calcolato con lo jacobiano geometrico in quanto non rappresenta la velocità angolare ω espressa in terna base. In definitiva lo jacobiano analitico è espresso da

$$\mathbf{J}_a(\mathbf{q}) = \begin{bmatrix} J_p(q) \\ J_\phi(q) \end{bmatrix} \quad (\text{A.54})$$

Appendice B

Parametri di Denavit-Hartenberg modificati del *Manus*

Nel paragrafo 3.2 sono stati introdotti i parametri di *Denavit-Hartenberg* del *Manus* utilizzati nella realizzazione del sistema di controllo per la risoluzione della cinematica inversa e diretta. Per completezza, in questa appendice sono illustrati i parametri cinematici secondo la convenzione di *Denavit-Hartenberg modificata* introdotta nel paragrafo A.3.2;

In figura B.1 sono riportate le terne di riferimento per ciascun braccio del manipolatore, mentre le tabelle B.2 e B.1 contengono i parametri di *Denavit-Hartenberg* modificati.

i	a_i	α_i	d_i	θ_i
1	0	0	0	θ_1
2	0	-90	0	θ_2
3	L_2	0	L_1	θ_3
4	0	90	L_3	θ_4
5	0	-90	0	θ_5
6	0	90	L_4	θ_6

Tabella B.1: Parametri cinematici di Denavit-Hartenberg modificati.

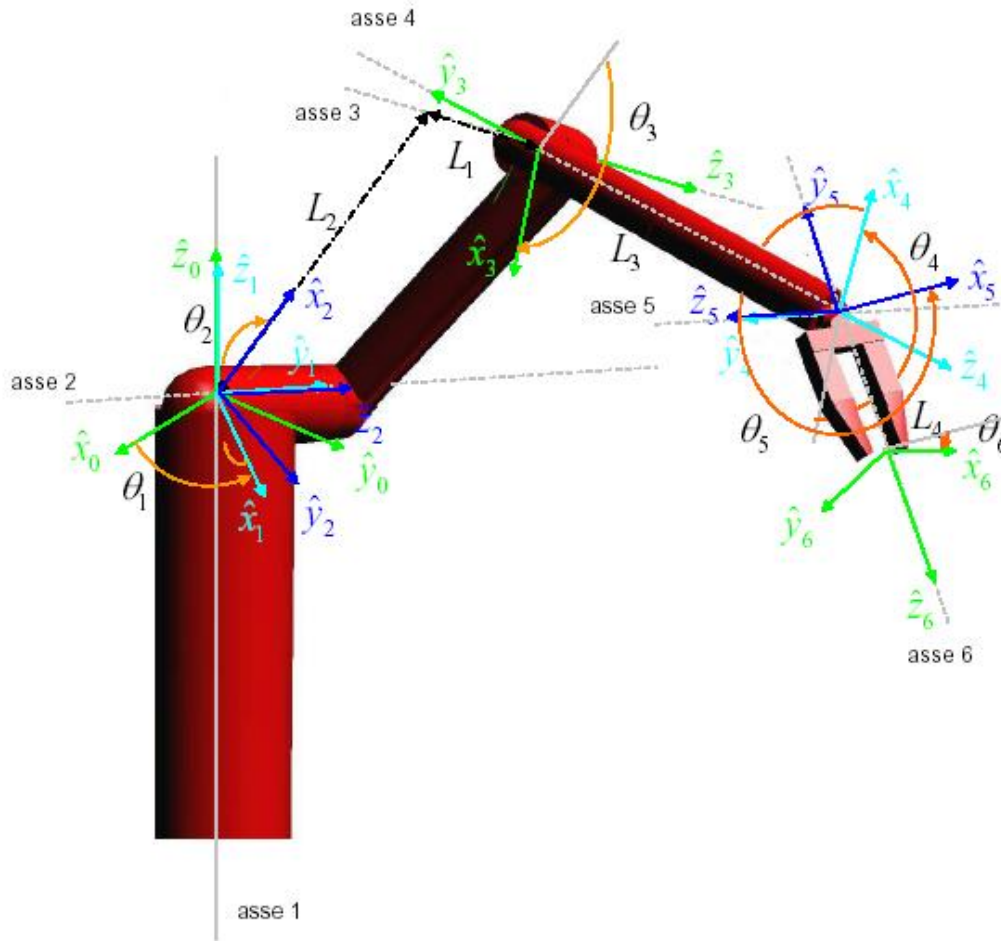


Figura B.1: Terne di riferimento del *Manus* secondo la convenzione di Denavit-Hartenberg modificata.

L_1	105
L_2	400
L_3	320
L_4	160

Tabella B.2: Lunghezze del manus secondo la convenzione di *Denavit-Hartenberg* modificata (espresse in mm)

Bibliografia

- [1] RoboCare. <http://pst.ip.rm.cnr.it/robocare/>.
- [2] B. Graf, M. Hans, J. Kubacki, and R.D. Schrft. Robotic home assistant care-0-bot 2. *Fraunhofer Institute, Stuttgart, Germany*.
- [3] Morpha. <http://www.morpha.de>.
- [4] Nursebot. <http://www-2.cs.cmu.edu/~nursebot/>.
- [5] Assisted Cognition Project. <http://www.cs.washington.edu/assistcog/>.
- [6] C. Melchiorri. *Traiettori Per Azionamenti Elettrici*. Esculapio Ed. Bologna, 2000.
- [7] Exact Dynamics. <http://www.exactdynamics.nl>.
- [8] J.C. Rosier, H.H. Kwee, and J.J. Smits. Rehabilitation robotics: The manus concept. *IEEE Transactions on Robotics and Automation*, 1991.
- [9] Philips Semiconductors. *SJA1000: Stand-alone CAN Controller*. Philips Semiconductors, 2000.
- [10] BOSCH. *CAN Specification 2.0*. 1991.
- [11] F. Monica. Progettazione e 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.
- [12] D. Pallastrelli. Studio e Realizzazione di un Framework Orientato agli Oggetti per Applicazioni Real-time. Tesi di Laurea in Ingegneria Informatica, Università degli Studi di Parma, 2002.
- [13] Roboop. <http://www.cours.polymtl.ca/roboop/>.
- [14] Robert Davies. http://www.robertnz.net/nm_intro.htm.
- [15] John Lloyd's. <http://www.cs.ubc.ca/spider/lloyd/rccl.html>.
- [16] L. Sciavicco and B. Siciliano. *Robotica Industriale, Modellistica e Controllo di Manipolatori*. McGraw-Hill, 2000.
- [17] J.J. Craig. *Introduction to Robotics, Mechanics and Control*. Addison-Wesley, 1989.
- [18] C. Guarino Lo Bianco. *Cinematica dei Manipolatori*. Pitagora Editrice, 2004.
- [19] J.M. Selig. *Introductory Robotics*. Prentice Hall, 1992.
- [20] P. Ochi. Progettazione e realizzazione del sistema di controllo di un robot manipolatore per compiti di assistenza. Tesi di Laurea in Ingegneria Elettronica, Università degli Studi di Parma, 2004.

- [21] S. Bahadori, A. Cesta, L. Iocchi, G.R Lone, D. Nardi, F. Pecora, and R. Rasconi. Towards ambient intelligence for the domestic care of the elderly. *RoboCare Technical Reports*.
- [22] A. Dario, R. Dillman, and H Christensen. Euron research roadmaps. <http://www.euron.org/>, 2004.
- [23] A. Cappelli and E. Giovannetti. L'interazione uomo-robot. *RoboCare Technical Reports*.