Università Politecnica delle Marche

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Meccanica

Progettazione Meccanica

Tesi di Laurea Magistrale

Controllo in Impedenza basato su Force Field con Modalità di Training Adattive per Robot Riabilitativo TIAGo

Force Field-Based Impedance Control with Adaptive Training Modes for the TIAGo Rehabilitation Robot

---

Relatore:

Prof. Palpacelli Matteo

Claudio

Candidato:

Gagliardini Marco

Anno Accademico 2025/2026

Sommario

La riabilitazione robotica degli arti superiori rappresenta un campo in rapida evoluzione, dove l'interazione uomo-robot gioca un ruolo fondamentale nel recupero motorio dei pazienti. Il presente lavoro di tesi si propone di sviluppare e validare un sistema di controllo in impedenza per il robot riabilitativo TIAGo, implementando una strategia di assistenza graduata basata su force field.

Il sistema proposto si basa sul paradigma *Assist-As-Needed*, dove il robot mantiene un comportamento prevalentemente passivo, lasciando al paziente il ruolo di motore primario del movimento. L'assistenza viene fornita in modo graduato attraverso un campo di forze virtuali costruito attorno alla traiettoria di riferimento, modulando l'intervento in base alle effettive necessità del paziente.

L'architettura di controllo prevede tre modalità operative: *Active Guidance* (AG), *Assist-As-Needed* (AAN) e *Restrictive Protection* (RP), con transizioni automatiche basate sull'errore di tracking. Il force field è composto da tre componenti: una forza normale che riporta il paziente verso il centro del tunnel virtuale, una forza tangenziale che fornisce guida lungo la traiettoria, e una forza viscosa che garantisce stabilità e fluidità del movimento.

La simulazione numerica è stata realizzata interamente in ambiente Python, utilizzando la libreria TRAC-IK per la cinematica inversa e Pinocchio per il calcolo della dinamica del manipolatore. L'approccio adottato consente di integrare le equazioni del moto nello spazio dei giunti, garantendo coerenza fisica e accuratezza numerica.

I risultati ottenuti dimostrano il corretto funzionamento del sistema di controllo, con una distribuzione bilanciata tra le modalità AG e AAN, coppie agli attuatori ampiamente entro i limiti di sicurezza, e un comportamento dinamico coerente con le aspettative teoriche.

Parole chiave: controllo in impedenza, riabilitazione robotica, Assist-As-Needed, force field, robot TIAGo, dinamica del manipolatore.

Abstract

Robotic rehabilitation of upper limbs represents a rapidly evolving field, where human-robot interaction plays a fundamental role in patients' motor recovery. This thesis aims to develop and validate an impedance control system for the TIAGo rehabilitation robot, implementing a graduated assistance strategy based on force fields.

The proposed system is based on the *Assist-As-Needed* paradigm, where the robot maintains a predominantly passive behavior, leaving the patient as the primary driver of movement. Assistance is provided gradually through a virtual force field constructed around the reference trajectory, modulating intervention based on the patient's actual needs.

The control architecture includes three operating modes: *Active Guidance* (AG), *Assist-As-Needed* (AAN), and *Restrictive Protection* (RP), with automatic transitions based on tracking error. The force field consists of three components: a normal force that brings the patient back toward the center of the virtual tunnel, a tangential force that provides guidance along the trajectory, and a viscous force that ensures stability and smoothness of movement.

The numerical simulation was entirely developed in Python environment, using the TRAC-IK library for inverse kinematics and Pinocchio for manipulator dynamics computation. The adopted approach allows integration of the equations of motion in joint space, ensuring physical consistency and numerical accuracy.

The results demonstrate the correct functioning of the control system, with a balanced distribution between AG and AAN modes, actuator torques well within safety limits, and dynamic behavior consistent with theoretical expectations.

Keywords: impedance control, robotic rehabilitation, Assist-As-Needed, force field, TIAGo robot, manipulator dynamics.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Contesto: riabilitazione robotica degli arti superiori

La riabilitazione motoria degli arti superiori rappresenta una sfida significativa nel campo della medicina riabilitativa. Pazienti affetti da ictus, lesioni del midollo spinale o altre patologie neurologiche necessitano di programmi riabilitativi intensivi e prolungati per recuperare le funzionalità motorie compromesse. Tradizionalmente, questi programmi si basano su sessioni di fisioterapia manuale, che richiedono la presenza costante di personale specializzato e risultano difficilmente scalabili rispetto al numero crescente di pazienti.

In questo contesto, la robotica riabilitativa si è affermata come una tecnologia promettente per supportare e potenziare i tradizionali approcci terapeutici. I robot riabilitativi offrono diversi vantaggi: la possibilità di eseguire movimenti ripetitivi con elevata precisione, la capacità di modulare l'assistenza in tempo reale, e la disponibilità di dati oggettivi sulle prestazioni del paziente durante ogni sessione. Un ulteriore beneficio è rappresentato dalla possibilità di effettuare sessioni riabilitative in ambiente domestico: dopo un adeguato addestramento iniziale sotto supervisione specialistica, il paziente può proseguire la terapia a casa con l'assistenza di un familiare, riducendo la necessità di frequenti spostamenti verso strutture sanitarie e aumentando la frequenza delle sessioni riabilitative.

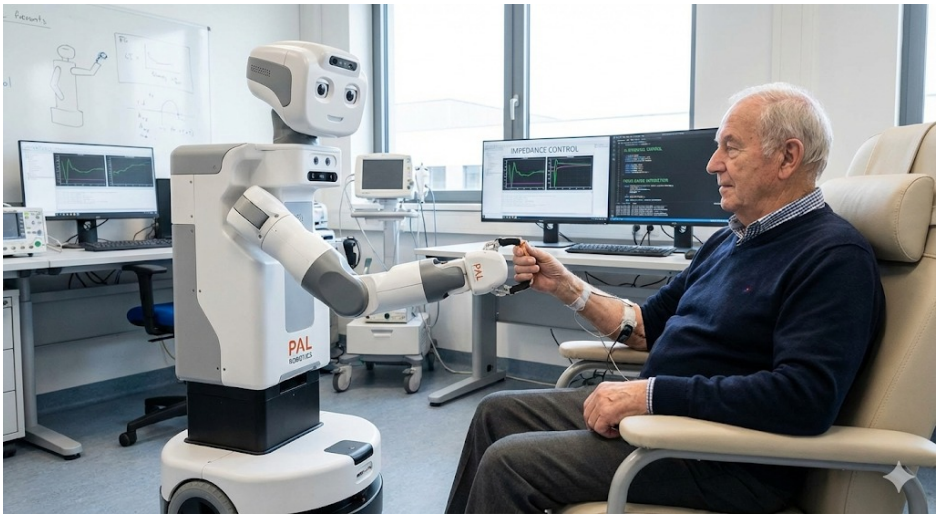

Figura 1.1: Rappresentazione concettuale dello scenario di riabilitazione assistita con robot TIAGo. Il paziente esegue un task di reaching guidato dal sistema di controllo AAN, con monitoraggio in tempo reale delle forze di interazione (immagine generata tramite intelligenza artificiale).

Gli studi clinici hanno dimostrato che i sistemi robotici con strategie di controllo che massimizzano la partecipazione attiva del paziente risultano più efficaci nel promuovere la neuroplasticità e il recupero motorio [1]. Questo principio ha portato allo sviluppo di controllori che forniscono assistenza minima, intervenendo solo quando strettamente necessario.

1.2 Stato dell'arte: controllo in impedenza e strategie di assistenza adattiva

Le strategie di controllo per robot riabilitativi possono essere classificate in due categorie principali: approcci ad alto livello (come il controllo adattivo) e approcci a basso livello (come il controllo PID). Tra questi, il controllo in impedenza si è affermato come paradigma di riferimento per applicazioni che richiedono interazione fisica sicura tra robot e paziente [2].

Il controllo in impedenza definisce una relazione dinamica tra la posizione dell'end-effector e la forza esercitata, tipicamente modellata come un sistema massa-molla-smorzatore virtuale. Questo approccio consente al robot di "cedere" alle forze applicate dal paziente, garantendo un'interazione naturale e sicura.

All'interno di questo paradigma, le strategie basate sul principio "Assist-As-Needed" (assistenza secondo necessità) rappresentano l'evoluzione più recente. L'idea fondamentale è che il robot debba fornire solo l'assistenza strettamen-

te necessaria, lasciando al paziente il ruolo attivo nel completamento del movimento. Questo approccio si basa sull'evidenza che la partecipazione attiva durante l'esercizio riabilitativo è fondamentale per stimolare i meccanismi di neuroplasticità.

Zhang et al. [3] hanno proposto uno schema di controllo basato su force field che implementa questo principio. Un campo di forze virtuali viene costruito attorno alla traiettoria di riferimento, creando un "tunnel" virtuale che guida il paziente lungo il percorso desiderato. L'intensità dell'assistenza viene modulata in base all'errore di tracking, permettendo al sistema di adattarsi dinamicamente alle capacità motorie del paziente. Lo schema prevede tre modalità operative, con transizioni automatiche basate sull'entità della deviazione dalla traiettoria:

- **Active Guidance (AG):** attivata quando l'errore di tracking è contenuto entro la soglia D_t. In questa condizione il paziente dimostra sufficiente capacità motoria per seguire la traiettoria in modo autonomo; il force field fornisce una guida leggera principalmente attraverso la componente tangenziale.

- **Assist-As-Needed (AAN):** attivata quando l'errore supera D_t ma rimane inferiore alla soglia critica D_n. Il robot fornisce assistenza graduata proporzionale alla deviazione, con la componente normale del force field che riporta progressivamente il paziente verso il centro del tunnel virtuale.

- **Restrictive Protection (RP):** attivata quando l'errore supera la soglia critica D_n. In questa condizione il training corrente risulta eccessivamente impegnativo per le capacità motorie del paziente; il sistema aumenta significativamente le forze di richiamo e, nel caso di persistenza della condizione, segnala la necessità di modificare i parametri della sessione riabilitativa.

Nota terminologica: Nel presente lavoro, il termine "AAN" viene utilizzato con due accezioni distinte in base al contesto. Quando ci si riferisce alla filosofia generale di controllo, indica il principio di "assistenza secondo necessità" che permea l'intero schema. Quando invece si discute delle modalità operative, "AAN mode" indica specificamente la modalità intermedia tra AG e RP. Il contesto renderà sempre chiaro a quale significato ci si riferisce.

1.3 Obiettivi della tesi

Il presente lavoro di tesi si propone di sviluppare e validare un sistema di controllo in impedenza per il robot TIAGo di PAL Robotics, implementando lo schema basato su force field proposto da Zhang et al. Gli obiettivi specifici sono:

1. **Sviluppo di un simulatore numerico:** realizzazione di un ambiente di simulazione completamente numerico che integri cinematica inversa (TRAC-IK) e dinamica del manipolatore (Pinocchio), senza dipendenza da simulatori fisici esterni.

2. **Implementazione del controllo in impedenza:** progettazione e implementazione del force field con le tre componenti (normale, tangenziale, viscosa) e la logica di switching tra le modalità AG, AAN e RP.

3. **Modellazione dell'interazione robot-paziente:** definizione di un modello del paziente che includa forza volontaria, inerzia del braccio, gravità e smorzamento neuromuscolare.

4. **Validazione del sistema:** verifica del corretto funzionamento attraverso simulazioni numeriche, analisi delle forze di interazione, e verifica del rispetto dei limiti di sicurezza degli attuatori.

1.4 Struttura del documento

Il presente documento è organizzato come segue:

- Il **Capitolo 2** descrive la piattaforma robotica TIAGo, con particolare attenzione alla cinematica del braccio e alle specifiche degli attuatori.

- Il **Capitolo 3** presenta i fondamenti teorici del controllo in impedenza, dello schema di assistenza adattiva basato su force field, e della dinamica del manipolatore.

- Il **Capitolo 4** illustra l'ambiente di sviluppo utilizzato, dalla configurazione del sistema operativo all'installazione delle librerie software, inclusa la fase esplorativa con Gazebo.

- Il **Capitolo 5** descrive l'architettura del simulatore numerico, dettagliando il flusso di calcolo dalla cinematica inversa all'integrazione dinamica.

- Il **Capitolo 6** presenta l'implementazione del controllo in impedenza, con la struttura del force field, il modello del paziente e la logica di switching.
- Il **Capitolo 7** riporta i risultati delle simulazioni, con analisi delle traiettorie, delle forze di interazione e della distribuzione delle modalità di training.
- Il **Capitolo 8** conclude il lavoro con una sintesi dei risultati ottenuti e una discussione sugli sviluppi futuri.

Le appendici contengono il codice sorgente principale, i parametri del modello TIAGo e i comandi per la configurazione dell'ambiente di sviluppo.

Capitolo 2

Piattaforma Robotica TIAGo

Questo capitolo presenta la piattaforma robotica TIAGo di PAL Robotics, utilizzata come base per lo sviluppo del sistema di controllo in impedenza. Vengono descritte le caratteristiche generali del robot, la struttura cinematica del braccio manipolatore, le specifiche degli attuatori e il modello URDF utilizzato per la simulazione.

2.1 Descrizione del robot TIAGo

TIAGo è un robot di servizio sviluppato da PAL Robotics (Barcellona, Spagna), progettato per applicazioni di ricerca, assistenza domestica e riabilitazione. Il nome è un acronimo di "Takes Instructions And Goes", che riflette la sua capacità di interpretare comandi vocali o gestuali ed eseguire autonomamente i compiti richiesti.

Il robot è disponibile in diverse configurazioni, che variano per la presenza o assenza del braccio manipolatore, il tipo di end-effector (gripper parallelo, Hey5 hand, o nessuno), e la configurazione dei sensori. La versione utilizzata in questo lavoro è dotata di base mobile differenziale, torso estensibile, braccio manipolatore a 7 gradi di libertà e gripper parallelo a due dita.

Caratteristiche generali del braccio

Le specifiche principali del braccio manipolatore sono riassunte nella Tabella 2.1.

Tabella 2.1: Caratteristiche generali del braccio manipolatore.

Parametro	**Valore**
Peso del braccio	10 kg
Payload massimo	2.8 kg
Reach massimo	86 cm
Gradi di libertà	7

Il reach massimo di 86 cm definisce l'area di influenza del braccio, ovvero la zona in cui non devono essere presenti ostacoli durante l'esecuzione di movimenti. Questo dato è fondamentale per la pianificazione delle traiettorie in ambiente riabilitativo, dove la sicurezza del paziente è prioritaria.

Applicazioni in ambito riabilitativo

TIAGo presenta caratteristiche che lo rendono particolarmente adatto ad applicazioni riabilitative. La sua struttura antropomorfa, con un braccio che replica approssimativamente le proporzioni e i gradi di libertà dell'arto superiore umano, facilita l'interazione naturale con il paziente. Il payload di 2.8 kg è sufficiente per supportare il peso di un avambraccio umano durante esercizi di riabilitazione assistita.

Recenti studi hanno esplorato l'utilizzo di TIAGo in contesti di assistenza agli anziani e a persone con disabilità motorie. Bajrami et al. [4] hanno analizzato l'ottimizzazione della postura del robot per operazioni di grasping, fornendo una formalizzazione completa della cinematica diretta e differenziale che costituisce un riferimento fondamentale per lo sviluppo di applicazioni di controllo avanzate.

2.2 Cinematica del braccio

La struttura cinematica del TIAGo comprende una base mobile differenziale, un giunto prismatico per l'elevazione del torso, e un braccio manipolatore a 7 giunti rotoidali. Ai fini del presente lavoro, ci si concentra sulla catena cinematica dal torso all'end-effector, che include il giunto prismatico del torso e i 7 giunti del braccio.

Struttura della catena cinematica

La catena cinematica rilevante per il controllo in impedenza comprende:

- **Torso:** 1 giunto prismatico (`torso_lift_joint`) che permette l'elevazione verticale del tronco del robot nell'intervallo 0–350 mm.

- **Braccio:** 7 giunti rotoidali (`arm_1_joint` ... `arm_7_joint`) che forniscono i gradi di libertà necessari per il posizionamento e l'orientamento dell'end-effector nello spazio.

La convenzione per i sistemi di riferimento segue quella adottata nel Robotics Systems Toolbox di MATLAB e descritta in dettaglio da Bajrami et al. [4]. Per ogni giunto rotoidale, l'asse di rotazione coincide con l'asse z locale del sistema di riferimento associato. La Figura 2.1 mostra i sistemi di riferimento assegnati a ciascun giunto del braccio in una configurazione flessa rappresentativa.

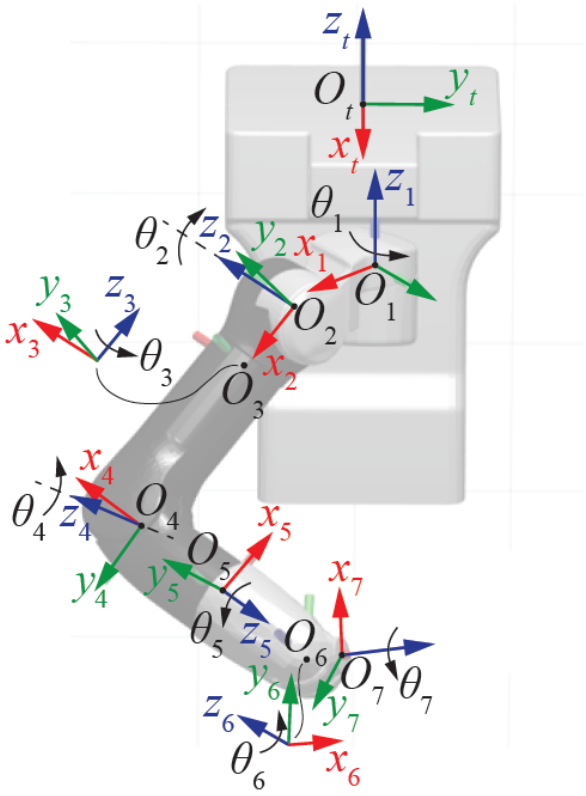

Figura 2.1: Sistemi di riferimento del braccio TIAGo. Sono indicati i frame $O_t, O_1, \ldots, O_7$ con i rispettivi assi e gli angoli $\theta_1 \ldots \theta_7$ per ogni giunto. Fonte: Bajrami et al. [4].

Limiti articolari

I limiti meccanici dei giunti sono riportati nella Tabella 2.2. Questi vincoli sono fondamentali per la pianificazione delle traiettorie e devono essere rispettati dal risolutore di cinematica inversa.

Tabella 2.2: Limiti articolari della catena cinematica torso-braccio.

Giunto	**Tipo**	**Limite inferiore**	**Limite superiore**
torso	prismatico	0 mm	350 mm
arm_1	rotoidale	0°	157.5°
arm_2	rotoidale	−90°	62.5°
arm_3	rotoidale	−202.5°	90°
arm_4	rotoidale	−22.5°	135°
arm_5	rotoidale	−120°	120°
arm_6	rotoidale	−90°	90°
arm_7	rotoidale	−120°	120°

Nota: Per la versione con sensore di forza/coppia al polso, il range di arm_6 è ridotto a $[-81°, 81°]$.

Ridondanza cinematica

Il braccio a 7 gradi di libertà è cinematicamente ridondante rispetto al task di posizionamento e orientamento dell'end-effector nello spazio 3D, che richiede 6 DOF (3 per la posizione, 3 per l'orientamento). Questa ridondanza offre vantaggi significativi:

- Possibilità di evitare singolarità cinematiche durante il movimento
- Capacità di ottimizzare la configurazione per criteri secondari (es. manipolabilità)
- Maggiore flessibilità nell'evitamento di ostacoli e limiti articolari

La gestione della ridondanza viene affidata al risolutore TRAC-IK, che sfrutta il grado di libertà extra per massimizzare la probabilità di convergenza verso una soluzione cinematicamente valida.

2.3 Specifiche degli attuatori e limiti di coppia

Il braccio TIAGo utilizza attuatori modulari con diverse caratteristiche per i giunti prossimali (moduli 1–4) e distali (polso). La Tabella 2.3 riporta le specifiche tecniche di ciascun attuatore.

Tabella 2.3: Specifiche tecniche degli attuatori del braccio TIAGo.

Attuatore	**Riduzione**	**Vel. max [rpm]**	**Coppia nom. [Nm]**	**Encoder mot.**	**Encoder ass.**
1st module (arm_1)	100:1	18	39	12 bit	12 bit
2nd module (arm_2)	100:1	18	39	12 bit	12 bit
3rd module (arm_3)	100:1	22	22	12 bit	12 bit
4th module (arm_4)	100:1	22	22	12 bit	12 bit
Wrist 1st (arm_5)	336:1	17	3	11 bit	12 bit
Wrist 2nd (arm_6)	336:1	17	5	11 bit	13 bit
Wrist 3rd (arm_7)	336:1	17	5	11 bit	13 bit

Modalità di controllo

Gli attuatori supportano diverse modalità di controllo a bordo:

- **Moduli 1–4:** controllo in posizione, velocità e corrente
- **Polso (5–7):** controllo in posizione e velocità

La disponibilità del controllo in corrente per i moduli prossimali è particolarmente rilevante per applicazioni di controllo in impedenza, poiché consente l'implementazione di strategie di controllo di coppia a basso livello.

Considerazioni per il controllo in impedenza

I limiti di coppia nominale rappresentano un vincolo importante per la sicurezza del sistema di controllo. Durante la simulazione, le coppie comandate vengono verificate rispetto a questi limiti per garantire che il robot operi sempre in condizioni sicure. In particolare, le coppie dei giunti prossimali (39 Nm e 22 Nm) sono dimensionate per compensare il peso del braccio stesso e del payload, mentre i giunti del polso (3–5 Nm) sono sufficienti per l'orientamento fine dell'end-effector.

2.4 Modello URDF e parametri dinamici

Il modello del robot utilizzato per la simulazione è descritto in formato URDF (Unified Robot Description Format), uno standard XML ampiamente adottato in ambito ROS per la descrizione di robot. Il file URDF contiene la definizione completa della struttura cinematica e dinamica del robot.

Struttura del file URDF

Il file URDF è organizzato in elementi principali:

- **Link:** definiscono i corpi rigidi del robot, ciascuno con proprietà di massa, centro di massa e tensore d'inerzia.
- **Joint:** definiscono le connessioni tra link, specificando il tipo (revolute, prismatic, fixed), l'asse di rotazione/traslazione, e i limiti meccanici.
- **Inertial:** per ogni link, specifica massa, posizione del centro di massa, e matrice d'inerzia rispetto al centro di massa.

Parametri dinamici

I parametri dinamici contenuti nel file URDF sono essenziali per il calcolo della dinamica del robot mediante la libreria Pinocchio. In particolare:

- **Massa dei link:** necessaria per il calcolo dei termini gravitazionali $\boldsymbol{G}(\boldsymbol{q})$ e della matrice di massa $\boldsymbol{M}(\boldsymbol{q})$.
- **Tensori d'inerzia:** matrici 3×3 simmetriche che descrivono la distribuzione di massa di ciascun link, necessarie per il calcolo delle accelerazioni angolari.
- **Posizione del centro di massa:** definita rispetto al sistema di riferimento del link, influenza il calcolo dei momenti.

Il modello URDF completo del TIAGo, generato dai file XACRO forniti da PAL Robotics, contiene 68 gradi di libertà totali (inclusi base mobile, testa e gripper) e 59 giunti. Per la simulazione del controllo in impedenza vengono considerati solo i giunti del braccio (`arm_1_joint` ... `arm_7_joint`), identificati per nome all'interno del modello.

Utilizzo con Pinocchio

La libreria Pinocchio carica il file URDF e costruisce automaticamente il modello dinamico del robot. Le funzionalità principali utilizzate nel simulatore includono:

- Calcolo della matrice di massa $\boldsymbol{M}(\boldsymbol{q})$ mediante l'algoritmo CRBA (Composite Rigid Body Algorithm)
- Calcolo dei termini di Coriolis e centrifughi $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$
- Calcolo del vettore di gravità $\boldsymbol{G}(\boldsymbol{q})$
- Forward dynamics mediante l'algoritmo ABA (Articulated Body Algorithm)
- Calcolo del Jacobiano geometrico per la mappatura forze-coppie

Questi algoritmi, implementati in modo efficiente da Pinocchio, costituiscono il nucleo computazionale del simulatore e verranno descritti in dettaglio nel Capitolo 5.

Capitolo 3

Fondamenti Teorici

Questo capitolo presenta i fondamenti teorici alla base del sistema di controllo sviluppato. Vengono introdotti il paradigma del controllo in impedenza, la filosofia Assist-As-Needed, la struttura del force field proposta da Zhang et al. [3], e le equazioni della dinamica del manipolatore.

3.1 Controllo in impedenza

Il controllo in impedenza è un paradigma di controllo che definisce una relazione dinamica tra la posizione dell'end-effector e la forza esercitata sull'ambiente. A differenza del controllo in posizione, che cerca di mantenere una traiettoria indipendentemente dalle forze esterne, il controllo in impedenza permette al robot di "cedere" quando incontra resistenza, garantendo un'interazione sicura con l'ambiente e con l'operatore umano.

Definizione di impedenza meccanica

In analogia con i sistemi elettrici, dove l'impedenza definisce il rapporto tra tensione e corrente, l'impedenza meccanica definisce il rapporto tra forza e velocità (o, equivalentemente, tra forza e spostamento). Un sistema meccanico lineare può essere caratterizzato da tre parametri: massa (M), smorzamento (D) e rigidezza (K).

Il comportamento desiderato dell'end-effector viene specificato mediante un modello massa-molla-smorzatore virtuale:

$$M \cdot \ddot{x} + D \cdot \dot{x} + K \cdot x = F_{\text{ext}} \tag{3.1}$$

dove x rappresenta lo spostamento dalla posizione di equilibrio, $\dot{x}$ e $\ddot{x}$ sono rispettivamente velocità e accelerazione, e F_{ext} è la forza esterna applicata.

Formulazione per il controllo

Nella pratica del controllo robotico, si considera tipicamente la deviazione dalla traiettoria desiderata. Definendo l'errore di posizione come $\Delta x = x_{\text{des}} - x_{\text{actual}}$ e l'errore di velocità come $\Delta\dot{x} = \dot{x}_{\text{des}} - \dot{x}_{\text{actual}}$, la forza di interazione desiderata può essere espressa come:

$$F = K \cdot \Delta x + D \cdot \Delta\dot{x} \tag{3.2}$$

Questa formulazione è nota come controllo in impedenza puro quando $M = 0$, ovvero quando si trascura il termine inerziale. In applicazioni riabilitative, dove le velocità sono tipicamente basse, questa semplificazione è spesso accettabile e riduce la complessità computazionale.

Vantaggi per applicazioni riabilitative

Il controllo in impedenza offre vantaggi significativi per la riabilitazione robotica:

- **Sicurezza intrinseca:** il robot cede alle forze del paziente, evitando situazioni di conflitto che potrebbero causare lesioni.
- **Interazione naturale:** il comportamento del robot è percepito come "morbido" e collaborativo, facilitando l'accettazione da parte del paziente.
- **Modulabilità dell'assistenza:** variando i parametri K e D è possibile regolare il livello di assistenza fornito, da quasi nullo (robot trasparente) a molto rigido (guida forzata).
- **Feedback propriocettivo:** il paziente percepisce la resistenza del robot e può adattare il proprio sforzo, stimolando i meccanismi di apprendimento motorio.

3.2 Strategia Assist-As-Needed

La strategia Assist-As-Needed (AAN), traducibile come "assistenza secondo necessità", rappresenta l'evoluzione più recente nel campo del controllo per robot

riabilitativi. Il principio fondamentale è che il robot debba fornire solo l'assistenza strettamente necessaria per completare il movimento, lasciando al paziente il ruolo attivo primario.

Fondamento neuroscientifico

La motivazione della strategia AAN deriva dalle evidenze neuroscientifiche sull'apprendimento motorio e sulla neuroplasticità. Studi clinici hanno dimostrato che la partecipazione attiva del paziente durante l'esercizio riabilitativo è fondamentale per stimolare i meccanismi di riorganizzazione corticale che sottendono il recupero motorio.

Un'assistenza eccessiva, sebbene possa facilitare il completamento del movimento nel breve termine, rischia di indurre una dipendenza dal supporto robotico e di ridurre lo sforzo volontario del paziente. Al contrario, un'assistenza calibrata che interviene solo quando necessario mantiene alto il livello di engagement del paziente e massimizza il beneficio terapeutico.

Confronto con approcci tradizionali

Gli approcci tradizionali alla riabilitazione robotica prevedono tipicamente un livello di assistenza fisso, determinato a priori sulla base di una valutazione delle capacità del paziente. Questo approccio presenta due limitazioni principali:

1. L'assistenza rimane costante anche quando il paziente potrebbe farne a meno, riducendo lo stimolo all'impegno attivo.

2. Non vi è adattamento in tempo reale alle fluttuazioni delle capacità motorie, che possono variare significativamente durante una singola sessione a causa di affaticamento o altri fattori.

La strategia AAN supera queste limitazioni modulando continuamente l'assistenza in base alle prestazioni istantanee del paziente, realizzando un controllo adattativo che risponde dinamicamente alle esigenze del momento.

3.3 Force field per riabilitazione

Zhang et al. [3] hanno proposto uno schema di controllo basato su force field che implementa la filosofia AAN mediante la costruzione di un "tunnel virtuale" attorno alla traiettoria di riferimento. All'interno di questo tunnel, un campo di

forze guida il paziente lungo il percorso desiderato con un'intensità che varia in funzione della deviazione dalla traiettoria.

Concetto di tunnel virtuale

Il tunnel virtuale è una struttura spaziale che circonda la traiettoria predefinita. La sua sezione trasversale è caratterizzata da due parametri fondamentali:

- D_n **(raggio del canale normale):** definisce il confine oltre il quale l'errore è considerato critico.
- D_t **(raggio del canale tangenziale):** definisce la zona di tolleranza entro cui l'assistenza è minima.

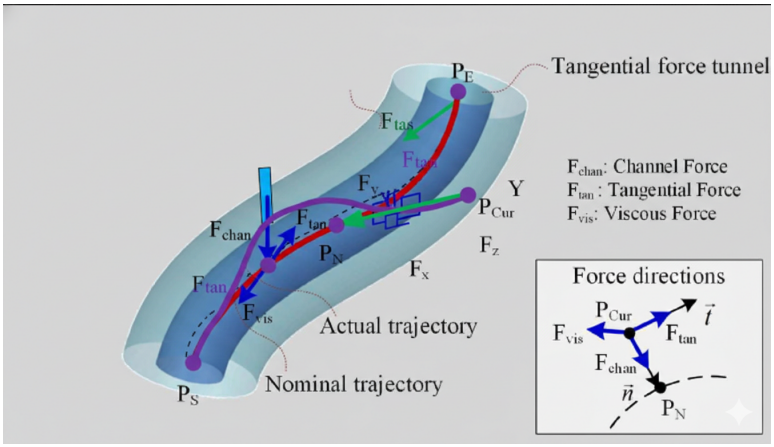

Figura 3.1: Struttura tipica del canale di forza 3D. Le traiettorie predeterminata e attuale sono rappresentate rispettivamente dalla curva rossa e viola, mentre i canali tangenziale e normale sono rappresentati rispettivamente in blu e verde. Fonte: Zhang et al. [3].

Calcolo dell'errore di tracking

L'errore di tracking $\Delta \boldsymbol{L}$ viene calcolato come la distanza tra la posizione attuale dell'end-effector $\boldsymbol{P}_{\text{Cur}}$ e il punto più vicino sulla traiettoria predefinita $\boldsymbol{P}_N$:

$$\Delta \boldsymbol{L} = \boldsymbol{P}_{\text{Cur}} - \boldsymbol{P}_N(R) \tag{3.3}$$

dove $\boldsymbol{P}_{\text{Cur}} \in \mathbb{R}^{3\times 1}$ è la posizione attuale e $\boldsymbol{P}_N \in \mathbb{R}^{3\times 1}$ è il punto più vicino sulla traiettoria predefinita. Questa formulazione, basata sulla distanza spaziale piuttosto che su un riferimento temporale, garantisce al paziente libertà temporale nell'esecuzione del movimento.

Componenti del force field

Il campo di forze assistivo è composto da tre componenti ortogonali, ciascuna con una specifica funzione:

Forza normale ($\boldsymbol{F}_{\mathbf{chan}}$). La forza normale, o forza di canale, agisce perpendicolarmente alla traiettoria e ha la funzione di riportare il paziente verso il centro del tunnel quando si verifica una deviazione. La sua intensità aumenta progressivamente con la distanza dalla traiettoria:

$$\boldsymbol{F}_{\text{chan}} = -K_{\text{chan}} \cdot (\|\Delta \boldsymbol{L}\|_2 - D_n) \cdot \hat{\boldsymbol{n}} \qquad \text{per } \|\Delta \boldsymbol{L}\|_2 > D_n \tag{3.4}$$

dove K_{chan} è il coefficiente di rigidezza del canale e $\hat{\boldsymbol{n}}$ è il versore normale che punta da $\boldsymbol{P}_{\text{Cur}}$ verso $\boldsymbol{P}_N$. All'interno della zona di tolleranza ($\|\Delta \boldsymbol{L}\|_2 \leq D_n$), la forza normale è nulla.

Forza tangenziale ($\boldsymbol{F}_{\mathbf{tan}}$). La forza tangenziale agisce lungo la direzione della traiettoria e ha la funzione di guidare il paziente in avanti. La sua intensità varia in funzione della posizione all'interno del tunnel:

$$\boldsymbol{F}_{\text{tan}} = K_{\text{tan}} \cdot \hat{\boldsymbol{t}} \qquad \text{per } \|\Delta \boldsymbol{L}\|_2 \leq D_t \tag{3.5}$$

dove K_{tan} è il coefficiente della forza tangenziale e $\hat{\boldsymbol{t}}$ è il versore tangente alla traiettoria nel punto $\boldsymbol{P}_N$. Quando l'errore supera D_t, la forza tangenziale viene progressivamente ridotta per evitare di spingere il paziente in avanti mentre è significativamente fuori traiettoria.

Forza viscosa ($\boldsymbol{F}_{\mathbf{vis}}$). La forza viscosa introduce uno smorzamento proporzionale alla velocità, contrastando movimenti troppo rapidi o bruschi:

$$\boldsymbol{F}_{\text{vis}} = -K_d \cdot \boldsymbol{V} \tag{3.6}$$

dove K_d è il coefficiente di smorzamento viscoso e $\boldsymbol{V} = \boldsymbol{J}^T(\boldsymbol{q}) \cdot \dot{\boldsymbol{q}}$ è la velocità cartesiana dell'end-effector, ottenuta dal prodotto del Jacobiano trasposto per la velocità ai giunti.

Forza totale del force field. La forza assistiva totale è data dalla somma delle tre componenti:

$$\boldsymbol{F}_r = \boldsymbol{F}_{\text{tan}} + \boldsymbol{F}_{\text{chan}} + \boldsymbol{F}_{\text{vis}} \tag{3.7}$$

3.4 Modalità di training: AG, AAN, RP

Lo schema di controllo prevede tre modalità operative distinte, con transizioni automatiche basate sull'entità dell'errore di tracking. Questa struttura permette di adattare il comportamento del robot alle capacità motorie istantanee del paziente.

Active Guidance (AG)

La modalità Active Guidance si attiva quando l'errore di tracking è contenuto entro la soglia tangenziale:

$$\|\Delta \boldsymbol{L}\|_2 < D_t \tag{3.8}$$

In questa condizione, il paziente dimostra sufficiente capacità motoria per seguire la traiettoria in modo sostanzialmente autonomo. Il force field fornisce una guida leggera, principalmente attraverso la componente tangenziale che accompagna il movimento senza correggerlo. La forza normale è nulla o molto ridotta.

Questa modalità rappresenta la condizione ottimale: il paziente è protagonista del movimento e il robot assume un ruolo di puro accompagnamento.

Assist-As-Needed (AAN)

La modalità Assist-As-Needed si attiva quando l'errore supera la soglia tangenziale ma rimane inferiore alla soglia critica:

$$D_t \leq \|\Delta \boldsymbol{L}\|_2 < D_n \tag{3.9}$$

In questa condizione, il paziente non riesce a mantenere autonomamente la traiettoria desiderata e necessita di assistenza. Il robot interviene con la componente normale del force field, che riporta progressivamente il paziente verso il centro del tunnel. L'intensità dell'assistenza è proporzionale alla deviazione: maggiore è l'errore, maggiore è la forza correttiva.

Contemporaneamente, la forza tangenziale viene ridotta per evitare di spingere il paziente in avanti mentre è ancora lontano dalla traiettoria corretta.

Restrictive Protection (RP)

La modalità Restrictive Protection si attiva quando l'errore supera la soglia critica:

$$\|\Delta \boldsymbol{L}\|_2 \geq D_n \tag{3.10}$$

Questa condizione indica che il training corrente è eccessivamente impegnativo per le capacità motorie attuali del paziente. Il sistema aumenta significativamente le forze di richiamo e, nel caso di persistenza della condizione, può segnalare la necessità di modificare i parametri della sessione riabilitativa o di interrompere l'esercizio.

La modalità RP ha principalmente una funzione di protezione: evita che il paziente si allontani troppo dalla traiettoria sicura e previene situazioni di frustrazione dovute a task troppo difficili.

Logica di switching

La transizione tra le modalità avviene automaticamente in base all'errore di tracking istantaneo. Lo pseudocodice seguente descrive la logica di switching:

1: **if** $\|\Delta \boldsymbol{L}\|_2 < D_t$ **then**
2: Mode $\leftarrow$ Active Guidance (AG)
3: **else if** $D_t \leq \|\Delta \boldsymbol{L}\|_2 < D_n$ **then**
4: Mode $\leftarrow$ Assist-As-Needed (AAN)
5: **else if** $\|\Delta \boldsymbol{L}\|_2 \geq D_n$ **then**
6: Mode $\leftarrow$ Restrictive Protection (RP)
7: **end if**

3.5 Dinamica del manipolatore

La simulazione del sistema richiede la modellazione accurata della dinamica del manipolatore. In questa sezione vengono presentate le equazioni del moto, i metodi per il loro calcolo efficiente, e la relazione tra forze cartesiane e coppie ai giunti.

Equazione del moto

La dinamica di un manipolatore rigido a n gradi di libertà è descritta dall'equazione di Eulero-Lagrange:

$$\boldsymbol{M}(\boldsymbol{q}) \cdot \ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \cdot \dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau} \tag{3.11}$$

dove:

- $\boldsymbol{q} \in \mathbb{R}^n$ è il vettore delle coordinate generalizzate (posizioni ai giunti)
- $\dot{\boldsymbol{q}} \in \mathbb{R}^n$ è il vettore delle velocità ai giunti
- $\ddot{\boldsymbol{q}} \in \mathbb{R}^n$ è il vettore delle accelerazioni ai giunti
- $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n \times n}$ è la matrice di massa (o matrice d'inerzia), simmetrica e definita positiva
- $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{n \times n}$ è la matrice dei termini di Coriolis e centrifughi
- $\boldsymbol{G}(\boldsymbol{q}) \in \mathbb{R}^n$ è il vettore dei termini gravitazionali
- $\boldsymbol{\tau} \in \mathbb{R}^n$ è il vettore delle coppie (o forze generalizzate) applicate ai giunti

Significato fisico dei termini

Matrice di massa $\boldsymbol{M}(\boldsymbol{q})$. La matrice di massa rappresenta l'inerzia del sistema vista dallo spazio dei giunti. L'elemento M_{ij} indica quanta coppia deve essere applicata al giunto i per produrre un'accelerazione unitaria al giunto j, mantenendo fermi tutti gli altri giunti. La matrice è configurazione-dipendente perché la distribuzione delle masse rispetto agli assi di rotazione cambia al variare della postura del robot.

Il calcolo di $\boldsymbol{M}(\boldsymbol{q})$ può essere effettuato mediante l'algoritmo CRBA (Composite Rigid Body Algorithm), che ha complessità $O(n^2)$.

Termini di Coriolis e centrifughi $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$. I termini di Coriolis e centrifughi rappresentano le forze fittizie che compaiono a causa del moto relativo tra i link del robot. Il termine di Coriolis è proporzionale al prodotto delle velocità di due giunti diversi, mentre il termine centrifugo è proporzionale al quadrato della velocità di un singolo giunto.

Il prodotto $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \cdot \dot{\boldsymbol{q}}$ può essere calcolato efficientemente come sottoprodotto dell'algoritmo RNEA (Recursive Newton-Euler Algorithm).

Vettore di gravità $\boldsymbol{G}(\boldsymbol{q})$. Il vettore di gravità rappresenta le coppie necessarie per mantenere il robot in equilibrio statico contro la forza di gravità. Dipende dalla configurazione perché il momento esercitato dal peso di ciascun link rispetto a ciascun giunto varia con la postura.

Il calcolo di $\boldsymbol{G}(\boldsymbol{q})$ si ottiene valutando la dinamica inversa con velocità e accelerazioni nulle: $\boldsymbol{G}(\boldsymbol{q}) = \text{RNEA}(\boldsymbol{q}, \boldsymbol{0}, \boldsymbol{0})$.

Forward dynamics

Il problema della forward dynamics (dinamica diretta) consiste nel calcolare le accelerazioni ai giunti $\ddot{\boldsymbol{q}}$ note le coppie applicate $\boldsymbol{\tau}$, la configurazione $\boldsymbol{q}$ e le velocità $\dot{\boldsymbol{q}}$. Riarrangiando l'Equazione 3.11:

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}(\boldsymbol{q})^{-1} \cdot [\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \cdot \dot{\boldsymbol{q}} - \boldsymbol{G}(\boldsymbol{q})] \tag{3.12}$$

Il calcolo esplicito dell'inversa della matrice di massa è computazionalmente costoso ($O(n^3)$). L'algoritmo ABA (Articulated Body Algorithm), sviluppato da Featherstone [5], risolve questo problema con complessità $O(n)$, rendendolo adatto ad applicazioni real-time.

Articulated Body Algorithm (ABA)

L'ABA calcola direttamente le accelerazioni ai giunti senza passare per l'inversione della matrice di massa. L'algoritmo si articola in tre passate ricorsive lungo la catena cinematica:

1. **Passata in avanti (dalla base all'end-effector):** calcola le velocità di ciascun link a partire dalle velocità ai giunti.

2. **Passata all'indietro (dall'end-effector alla base):** calcola le inerzie articolate e i bias di forza per ciascun link.

3. **Passata in avanti finale:** calcola le accelerazioni ai giunti propagando l'accelerazione dalla base verso l'end-effector.

L'implementazione dell'ABA è fornita dalla libreria Pinocchio attraverso la funzione `aba(model, data, q, v, tau)`, che restituisce direttamente il vettore delle accelerazioni $\ddot{\boldsymbol{q}}$.

Jacobiano e mappatura forze-coppie

Il Jacobiano geometrico $\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{6\times n}$ mette in relazione le velocità ai giunti $\dot{\boldsymbol{q}}$ con la velocità cartesiana dell'end-effector (velocità lineare $\boldsymbol{v}$ e velocità angolare $\boldsymbol{\omega}$):

$$\begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} = \boldsymbol{J}(\boldsymbol{q}) \cdot \dot{\boldsymbol{q}} \tag{3.13}$$

Il principio dei lavori virtuali stabilisce che, se una forza $\boldsymbol{F}$ è applicata all'end-effector nello spazio cartesiano, la coppia equivalente $\boldsymbol{\tau}$ nello spazio dei giunti è data da:

$$\boldsymbol{\tau} = \boldsymbol{J}(\boldsymbol{q})^T \cdot \boldsymbol{F} \tag{3.14}$$

Questa relazione è fondamentale per il controllo in impedenza: le forze del force field, calcolate nello spazio cartesiano secondo le Equazione 3.4–(3.6), vengono mappate in coppie ai giunti mediante il Jacobiano trasposto.

Integrazione numerica

La simulazione della dinamica richiede l'integrazione numerica delle equazioni del moto. Partendo dallo stato corrente $(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k)$ e dalle coppie applicate $\boldsymbol{\tau}_k$, le accelerazioni vengono calcolate mediante l'ABA:

$$\ddot{\boldsymbol{q}}_k = \mathrm{ABA}(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \boldsymbol{\tau}_k) \tag{3.15}$$

Lo stato al passo successivo viene quindi ottenuto mediante integrazione di Eulero esplicita:

$$\dot{\boldsymbol{q}}_{k+1} = \dot{\boldsymbol{q}}_k + \Delta t \cdot \ddot{\boldsymbol{q}}_k \tag{3.16}$$

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \Delta t \cdot \dot{\boldsymbol{q}}_{k+1} \tag{3.17}$$

dove Δt è il passo di integrazione. Sebbene metodi di integrazione più sofisticati (es. Runge-Kutta) offrano maggiore accuratezza, il metodo di Eulero è sufficiente per le velocità relativamente basse tipiche delle applicazioni riabilitative, a patto di utilizzare un passo di integrazione sufficientemente piccolo.

Capitolo 4

Ambiente di Sviluppo e Strumenti Software

Questo capitolo descrive in dettaglio l'ambiente di sviluppo utilizzato per la realizzazione del simulatore. La documentazione è volutamente dettagliata per consentire la riproducibilità del lavoro e facilitare eventuali sviluppi futuri.

Il percorso di sviluppo si articola in due fasi principali: una fase esplorativa condotta con il simulatore Gazebo all'interno di un container Docker, e una fase implementativa basata su simulazione numerica pura.

Riferimenti ufficiali. L'installazione e la configurazione dell'ambiente seguono le guide ufficiali PAL Robotics disponibili all'indirizzo: http://wiki.ros.org/Robots/TIAGo/Tutorials

4.1 Configurazione WSL Ubuntu 20.04

Il lavoro è stato sviluppato su sistema operativo Windows 10/11 utilizzando Windows Subsystem for Linux (WSL) con distribuzione Ubuntu 20.04 LTS. Questa scelta è motivata dalla necessità di utilizzare ROS Noetic, ufficialmente supportato solo su Ubuntu 20.04.

Installazione di WSL

L'installazione di WSL su Windows avviene tramite PowerShell con privilegi di amministratore:

```
# Abilitare WSL (richiede riavvio)
wsl --install

```

```
# Dopo il riavvio, installare Ubuntu 20.04
wsl --install -d Ubuntu-20.04

# Verificare l'installazione
wsl -l -v
```

4.2 Stack ROS Noetic

ROS (Robot Operating System) Noetic è stato installato seguendo la procedura ufficiale documentata all'indirizzo: http://wiki.ros.org/noetic/Installation/Ubuntu

Procedura di installazione step-by-step

Step 1: Configurazione dei repository ROS.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu \
  $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-
   latest.list'
```

Step 2: Aggiunta delle chiavi GPG.

```
sudo apt install curl
curl -s https://raw.githubusercontent.com/ros/rosdistro/
   master/ros.asc \
  | sudo apt-key add -
```

Step 3: Aggiornamento e installazione.

```
sudo apt update
sudo apt install ros-noetic-desktop-full
```

Step 4: Configurazione dell'ambiente.

```
# Aggiungere il source al file di configurazione della
   shell
```

```
echo "source /opt/ros/noetic/setup.bash" >> $HOME/.bashrc
source $HOME/.bashrc
```

Step 5: Verifica dell'installazione.

```
rosversion -d
# Output atteso: noetic
```

Creazione del workspace catkin

Il workspace catkin per il progetto TIAGo viene creato nella home directory dell'utente:

```
# Creare la struttura del workspace
mkdir -p $HOME/tiago_public_ws/src
cd $HOME/tiago_public_ws

# Inizializzare il workspace
catkin_make

# Configurare l'ambiente
echo "source $HOME/tiago_public_ws/devel/setup.bash" >>
   $HOME/.bashrc
```

4.3 Fase esplorativa: Docker e Gazebo

Prima di sviluppare il simulatore numerico, è stata condotta una fase esplorativa utilizzando il simulatore Gazebo all'interno di un container Docker fornito da PAL Robotics. La procedura segue il tutorial ufficiale: http://wiki.ros.org/Robots/TIAGo/Tutorials/Installation/Installing_Tiago_tutorial_docker

Questa fase è stata fondamentale per:

- Familiarizzare con il robot TIAGo e il suo modello cinematico/dinamico
- Validare le traiettorie cartesiane in ambiente simulato
- Estrarre configurazioni articolari iniziali (seed) per la cinematica inversa
- Verificare i limiti operativi del manipolatore e le singolarità

Installazione Docker e rocker

Step 1: Installazione di Docker.

```
sudo apt install docker.io
sudo usermod -aG docker $USER
# IMPORTANTE: effettuare logout e login per applicare le
   modifiche
```

Step 2: Installazione di rocker.

```
pip3 install rocker
```

Step 3: Download dell'immagine Docker TIAGo.

```
docker pull palroboticssl/tiago_tutorials:noetic
```

Avvio del container TIAGo

Il container viene avviato con il comando seguente:

```
rocker --home --user --nvidia --x11 --privileged \
  palroboticssl/tiago_tutorials:noetic
```

I flag utilizzati hanno il seguente significato:

- `-home`: monta la directory `$HOME` dell'host nel container
- `-user`: esegue il container come utente corrente (non root)
- `-nvidia`: abilita il supporto GPU NVIDIA
- `-x11`: abilita il forwarding del display X11 per le GUI
- `-privileged`: concede privilegi estesi (necessario per alcuni driver)

Workflow con Gazebo

L'utilizzo tipico del simulatore Gazebo prevede l'apertura di tre terminali simultanei. La procedura è documentata nel tutorial: http://wiki.ros.org/Robots/TIAGo/Tutorials/motions/runtiago_gazebo

Terminale 1 — Avvio del simulatore Gazebo.

```
cd $HOME/tiago_public_ws
source devel/setup.bash
roslaunch tiago_gazebo tiago_gazebo.launch public_sim:=true
    \
  end_effector:=pal-gripper
```

Terminale 2 — Esecuzione dello script di controllo.

```
cd $HOME/tiago_public_ws
source devel/setup.bash
rosrun <nome_pacchetto> <nome_script>.py
```

Terminale 3 — Visualizzazione in RViz.

```
cd $HOME/tiago_public_ws
source devel/setup.bash
rosrun rviz rviz
```

4.4 Transizione all'approccio numerico puro

Dopo la fase esplorativa, si è deciso di abbandonare l'approccio basato su Gazebo in favore di una simulazione completamente numerica. Le motivazioni di questa scelta sono:

1. **Controllo completo sulla dinamica:** Gazebo utilizza engine fisici (ODE, Bullet) che introducono approssimazioni non sempre documentate. Con Pinocchio si ha accesso diretto alle equazioni del moto.

2. **Riproducibilità:** la simulazione numerica è completamente deterministica. A parità di condizioni iniziali, i risultati sono identici.

3. **Efficienza computazionale:** eliminando l'overhead del rendering 3D, i tempi di esecuzione si riducono significativamente.

4. **Indipendenza da ROS runtime:** lo script può essere eseguito senza avviare roscore o altri nodi ROS.

Creazione del workspace per la simulazione numerica

```
# Creare la struttura del progetto
mkdir -p $HOME/tesi_ws/src/aan_simulation
cd $HOME/tesi_ws/src/aan_simulation

# Creare le directory per i risultati
mkdir -p results

# Creare il file principale
touch ImpedanceControlTiago.py
chmod +x ImpedanceControlTiago.py
```

Generazione del file URDF

Per la simulazione numerica è necessario disporre del modello URDF (Unified Robot Description Format) del robot TIAGo. Il modello originale fornito da PAL Robotics utilizza il formato XACRO (XML Macro), che deve essere convertito in URDF puro per l'utilizzo con Pinocchio.

Il file XACRO principale del TIAGo si trova nel workspace catkin:

```
$HOME/tiago_public_ws/src/tiago_robot/tiago_description/
   robots/tiago.urdf.xacro
```

La conversione richiede che l'ambiente ROS sia correttamente configurato, poiché il file XACRO include riferimenti ad altri package (come `tiago_description_calibration`). È quindi necessario eseguire il source del workspace prima della conversione:

Step 1: Configurazione dell'ambiente.

```
cd $HOME/tiago_public_ws
source devel/setup.bash
```

Step 2: Verifica della disponibilità dei package.

```
# Verificare che xacro trovi tutti i package necessari
rospack find tiago_description_calibration
# Output atteso: /home/<user>/tiago_public_ws/src/
   tiago_description_calibration
```

Step 3: Generazione del file URDF.

```
cd $HOME/tesi_ws/src/aan_simulation
rosrun xacro xacro $HOME/tiago_public_ws/src/tiago_robot/
   tiago_description/robots/tiago.urdf.xacro > tiago_full.
   urdf
```

Step 4: Verifica del file generato.

```
# Verificare dimensione (deve essere > 0)
ls -lh tiago_full.urdf
# Output atteso: circa 129K

# Contare le righe
wc -l tiago_full.urdf
# Output atteso: circa 3595 righe
```

Il file URDF generato contiene il modello completo del robot TIAGo, inclusi base mobile, torso, braccio a 7 DOF, testa e gripper. Per la simulazione del controllo in impedenza verranno utilizzati solo i giunti del braccio, ma il modello completo è necessario per il corretto calcolo della dinamica.

Step 5: Test di caricamento con Pinocchio.

```
import pinocchio as pin

urdf_path = "tiago_full.urdf"
model = pin.buildModelFromUrdf(urdf_path)

print(f"Model loaded successfully!")
print(f"  Total DOF: {model.nq}")
print(f"  Joints: {model.njoints}")
```

L'output atteso del test è:

```
Model loaded successfully!
  Total DOF: 68
  Joints: 59
```

Il modello presenta 68 gradi di libertà totali e 59 giunti, che includono tutti i componenti del robot. I 7 giunti del braccio (`arm_1_joint` ... `arm_7_joint`) sono un sottoinsieme di questo modello e verranno identificati per nome nello script di simulazione.

4.5 Librerie utilizzate

4.5.1 TRAC-IK

TRAC-IK è una libreria open-source per la risoluzione della cinematica inversa che combina due approcci complementari: un risolutore numerico basato su KDL e un risolutore SQP (Sequential Quadratic Programming). La combinazione dei due metodi garantisce un elevato tasso di convergenza anche per manipolatori ridondanti come il braccio a 7 DOF del TIAGo.

Installazione:

```
sudo apt install ros-noetic-trac-ik
python3 -c "from trac_ik_python.trac_ik import IK; print('
   TRAC-IK OK')"
```

4.5.2 Pinocchio

Pinocchio è una libreria C++ (con binding Python) per il calcolo efficiente della dinamica di sistemi articolati. Implementa algoritmi allo stato dell'arte, tra cui RNEA per la dinamica inversa, ABA per la forward dynamics, e CRBA per il calcolo della matrice di massa.

Installazione:

```
sudo apt install ros-noetic-pinocchio
python3 -c "import pinocchio; print(f'Pinocchio {pinocchio.
   __version__}')"
```

4.6 Riepilogo configurazione finale

La Tabella 4.1 riassume la configurazione software finale utilizzata per lo sviluppo.

Tabella 4.1: Configurazione software finale.

Componente	**Strumento**	**Versione**
Sistema operativo host	Windows 10/11	—
Ambiente Linux	WSL Ubuntu	20.04 LTS
Framework robotica	ROS	Noetic
Cinematica inversa	TRAC-IK	1.6.6
Dinamica	Pinocchio	3.3.0
Modello robot	URDF TIAGo	PAL Robotics
Linguaggio	Python	3.8

Struttura del progetto

La struttura finale del progetto è la seguente:

```
$HOME/tesi_ws/src/aan_simulation/
|-- ImpedanceControlTiago.py    # Script principale simulazione
|-- tiago_full.urdf             # Modello URDF completo del robot
|-- results/                    # Directory output
    |-- simulation_results.csv  # Dati numerici
    |-- *.png                   # Grafici generati
```

Checklist di verifica ambiente

Prima di procedere con l'esecuzione del simulatore, verificare che tutti i componenti siano correttamente installati:

```
#!/bin/bash
echo "=== Verifica ambiente di sviluppo ==="

# 1. Verifica ROS
echo -n "ROS Noetic: "
rosversion -d 2>/dev/null || echo "NON INSTALLATO"

# 2. Verifica TRAC-IK
echo -n "TRAC-IK: "
```

```
python3 -c "from trac_ik_python.trac_ik import IK; print('OK')" \
  2>/dev/null || echo "NON INSTALLATO"

# 3. Verifica Pinocchio
echo -n "Pinocchio: "
python3 -c "import pinocchio; print(pinocchio.__version__)" \
  2>/dev/null || echo "NON INSTALLATO"

echo "=== Fine verifica ==="
```

L'output atteso in caso di installazione corretta è:

```
=== Verifica ambiente di sviluppo ===
ROS Noetic: noetic
TRAC-IK: OK
Pinocchio: 3.3.0
=== Fine verifica ===
```

Capitolo 5

Architettura del Simulatore

Questo capitolo descrive l'architettura del simulatore numerico sviluppato per validare il sistema di controllo Assist-As-Needed. Il simulatore integra cinematica inversa, calcolo dinamico e integrazione numerica in un flusso computazionale coerente, permettendo di riprodurre il comportamento del sistema robot-paziente senza la necessità di un ambiente di simulazione fisica come Gazebo.

5.1 Schema generale del sistema

Il simulatore è strutturato come una pipeline di elaborazione che trasforma una traiettoria cartesiana di riferimento in traiettorie di posizione, velocità, forze e coppie ai giunti del manipolatore. Lo schema generale comprende i seguenti blocchi funzionali:

1. **Generazione traiettoria:** definizione della traiettoria cartesiana di riferimento ($\boldsymbol{P}_{\text{target}}$) mediante interpolazione lineare tra punto iniziale e finale.

2. **Cinematica inversa:** conversione della traiettoria cartesiana in traiettoria nello spazio dei giunti mediante TRAC-IK.

3. **Modello paziente:** generazione della traiettoria "intended" che simula il tentativo imperfetto del paziente di seguire il riferimento.

4. **Force field:** calcolo delle forze assistive ($\boldsymbol{F}_{\text{chan}}$, $\boldsymbol{F}_{\text{tan}}$, $\boldsymbol{F}_{\text{vis}}$) in base alla deviazione dalla traiettoria.

5. **Dinamica:** calcolo dei termini dinamici ($\boldsymbol{M}$, $\boldsymbol{C}$, $\boldsymbol{G}$) e delle accelerazioni mediante Pinocchio.

6. **Integrazione:** propagazione dello stato del sistema nel tempo mediante integrazione di Eulero.

Flusso dei dati

Il flusso dei dati nel simulatore può essere schematizzato come segue:

1. Traiettoria cartesiana $\boldsymbol{P}_{\text{target}}(t)$
2. $\rightarrow$ TRAC-IK (cinematica inversa) $\rightarrow$ $\boldsymbol{q}_{\text{target}}(t)$
3. $\rightarrow$ Pinocchio (FK, Jacobiano, Dinamica)
4. $\rightarrow$ Calcolo forze: $\boldsymbol{F}_{\text{patient}} + \boldsymbol{F}_{\text{field}}$
5. $\rightarrow$ Mappatura: $\boldsymbol{\tau} = \boldsymbol{J}^T \cdot \boldsymbol{F}$
6. $\rightarrow$ Forward dynamics: $\ddot{\boldsymbol{q}} = \text{ABA}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau})$
7. $\rightarrow$ Integrazione Eulero: $\boldsymbol{q}(t + \Delta t)$, $\dot{\boldsymbol{q}}(t + \Delta t)$
8. $\rightarrow$ Output: CSV con tutti i dati

Separazione tra robot e controllo

Un aspetto fondamentale dell'architettura è la separazione netta tra il modello del "plant" (il robot fisico simulato) e il controllore:

- **Plant:** include la dinamica completa del robot (massa, Coriolis, gravità) calcolata mediante l'algoritmo ABA di Pinocchio.
- **Controllore:** compensa solo la gravità ($\boldsymbol{\tau}_{\text{control}} = \boldsymbol{G}$), mantenendo il robot "trasparente" rispetto alle forze esterne. Questo approccio è coerente con la filosofia AAN di minimo intervento.

5.2 Cinematica inversa con TRAC-IK

TRAC-IK (Track-IK) è un risolutore di cinematica inversa open-source sviluppato specificamente per manipolatori robotici ridondanti [6]. Rispetto ai risolutori tradizionali basati su metodi iterativi (es. Newton-Raphson, Jacobiano pseudoinverso), TRAC-IK combina due strategie complementari: un solver basato su Sequential Quadratic Programming (SQP) e un solver basato su KDL con random restarts.

Vantaggi di TRAC-IK

La scelta di TRAC-IK per questo lavoro è motivata da diversi fattori:

- **Success rate elevato:** superiore al 95% su manipolatori ridondanti, come documentato in letteratura.
- **Gestione automatica dei limiti:** il solver rispetta i limiti articolari definiti nel file URDF senza necessità di implementazione esplicita.
- **Gestione della ridondanza:** ottimizzazione implicita del null-space per manipolatori con più di 6 DOF.
- **Velocità:** convergenza tipica in 5–20 ms per waypoint, adatta a preprocessing offline di traiettorie.

Configurazione del risolutore

Il risolutore TRAC-IK è stato configurato con i seguenti parametri:

```
from trac_ik_python.trac_ik import IK

ik_solver = IK(
    base_link="torso_lift_link",    # Base della catena cinematica
    tip_link="arm_7_link",          # End-effector
    urdf_string=urdf_content,
    timeout=0.005,                   # 5 ms timeout per waypoint
    epsilon=1e-5,                    # Tolleranza di convergenza
    solve_type="Speed"              # Modalita' ottimizzata per velocita'
)
```

La scelta di `torso_lift_link` come base della catena cinematica, anziché `base_footprint`, deriva dalla decisione progettuale di mantenere il torso vincolato durante la simulazione. Questo semplifica il problema IK a 7 DOF (solo il braccio) pur permettendo di variare l'altezza del torso lungo la traiettoria.

Strategia multi-seed

Per massimizzare il success rate, è stata implementata una strategia multi-seed che tenta la risoluzione IK con diverse configurazioni iniziali. I seed sono stati selezionati per coprire regioni diverse dello spazio di lavoro:

```
seed_candidates = [
    # Seed 1: Configurazione home TIAGo
    [0.15, 0.20, -1.34, -0.20, 1.94, -1.57, 1.37],
    # Seed 2: Braccio esteso frontalmente
    [0.20, 0.00, -1.00, 1.50, -1.50, 0.50, 0.00],
    # Seed 3: Braccio raccolto lateralmente
    [0.15, 0.50, -0.50, 0.00, 1.50, 0.00, 0.50],
    # Seed 4: Configurazione intermedia
    [0.25, 0.30, -1.20, 1.20, -1.80, 0.30, 0.30],
    # Seed 5: Ottimizzato per workspace laterale
    [0.20, 0.70, -1.40, -0.50, 2.00, -1.20, 1.00]
]
```

Il processo di risoluzione itera sui seed in ordine fino a trovare una soluzione valida:

```
def solve_ik(target_position, target_orientation):
    for seed in seed_candidates:
        q_solution = ik_solver.get_ik(
            seed,
            target_position,      # [x, y, z]
            target_orientation,   # [qx, qy, qz, qw]
            position_tolerance,   # 1 mm
            orientation_tolerance # 0.01 rad
        )
        if q_solution is not None:
            return q_solution
    return None  # Fallimento
```

Risultati della risoluzione IK

La risoluzione IK è stata eseguita offline per l'intera traiettoria di 20000 waypoint (20 secondi a 1000 Hz). L'errore di forward kinematics (FK) è stato calcolato riapplicando la cinematica diretta alle soluzioni IK e confrontando la posizione

ottenuta con il target. Errori inferiori a 0.1 mm sono trascurabili per applicazioni riabilitative, dove le soglie cliniche sono tipicamente nell'ordine di 5 mm.

5.3 Calcolo dinamico con Pinocchio

Pinocchio è una libreria C++ (con binding Python) per il calcolo efficiente della cinematica e dinamica di sistemi multi-corpo [7]. Sviluppata presso LAAS-CNRS e INRIA, implementa gli algoritmi ricorsivi di Featherstone [5] con complessità $O(n)$, dove n è il numero di gradi di libertà.

Caricamento del modello

Il modello del robot viene caricato dal file URDF e automaticamente analizzato per costruire le strutture dati necessarie al calcolo dinamico:

```
import pinocchio as pin

# Caricamento modello
model = pin.buildModelFromUrdf("tiago_full.urdf")
data = model.createData()

# Identificazione frame end-effector
ee_frame_id = model.getFrameId("arm_7_link")

# Verifica
print(f"DOF totali: {model.nq}")    # 68
print(f"Giunti: {model.njoints}")   # 59
```

Matrice di massa $\boldsymbol{M}(\boldsymbol{q})$

La matrice di massa $\boldsymbol{M}(\boldsymbol{q})$ viene calcolata mediante l'algoritmo CRBA (Composite Rigid Body Algorithm). Per il braccio TIAGo a 7 DOF, $\boldsymbol{M}(\boldsymbol{q})$ è una matrice 7×7 simmetrica definita positiva:

```
# Calcolo matrice di massa
pin.crba(model, data, q)
M = data.M[arm_indices, :][:, arm_indices]  # Estrai
   sottomatrice 7x7
```

La dipendenza di $\boldsymbol{M}$ dalla configurazione $\boldsymbol{q}$ riflette il fatto che la distribuzione delle masse rispetto agli assi di rotazione cambia al variare della postura del robot.

Termini di Coriolis e gravità

I termini di Coriolis $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ e gravità $\boldsymbol{G}(\boldsymbol{q})$ vengono calcolati mediante la funzione `computeAllTerms`, che esegue internamente l'algoritmo RNEA (Recursive Newton-Euler Algorithm):

```
# Calcolo tutti i termini dinamici
pin.computeAllTerms(model, data, q, q_dot)

# Estrazione termini per il braccio
C = data.C[arm_indices, :][:, arm_indices]  # Matrice
   Coriolis 7x7
G = data.g[arm_indices]                      # Vettore
   gravita' 7x1
```

Il termine di Coriolis $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \cdot \dot{\boldsymbol{q}}$ rappresenta le forze fittizie dovute al moto relativo tra i link. Il termine gravitazionale $\boldsymbol{G}(\boldsymbol{q})$ rappresenta le coppie necessarie per mantenere il robot in equilibrio statico.

Forward dynamics con ABA

L'algoritmo ABA (Articulated Body Algorithm) calcola direttamente le accelerazioni ai giunti senza passare per l'inversione esplicita della matrice di massa. Questo approccio ha complessità $O(n)$ rispetto a $O(n^3)$ dell'inversione diretta:

```
# Forward dynamics
q_ddot = pin.aba(model, data, q, q_dot, tau)

# Estrazione accelerazioni braccio
q_ddot_arm = q_ddot[arm_v_indices]
```

L'ABA risolve implicitamente l'equazione:

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}(\boldsymbol{q})^{-1} \cdot [\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \cdot \dot{\boldsymbol{q}} - \boldsymbol{G}(\boldsymbol{q})] \tag{5.1}$$

includendo automaticamente tutti i termini dinamici (massa, Coriolis, gravità) nel calcolo delle accelerazioni.

Flusso causale della simulazione

È importante chiarire come viene risolta l'equazione di Lagrange nella simulazione. L'equazione del moto:

$$\boldsymbol{M}(\boldsymbol{q}) \cdot \ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \cdot \dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau}_{\text{totale}} \tag{5.2}$$

potrebbe sembrare avere due incognite ($\ddot{\boldsymbol{q}}$ e $\boldsymbol{\tau}_{\text{totale}}$), ma nella simulazione forward dynamics esiste una sola incognita: l'accelerazione $\ddot{\boldsymbol{q}}$. Questo perché la coppia $\boldsymbol{\tau}_{\text{totale}}$ viene calcolata completamente *prima* di risolvere l'equazione dinamica.

Il flusso causale si articola in due fasi distinte:

- **Fase A — Calcolo delle coppie:** a partire dallo stato corrente $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ vengono calcolate le forze del force field e del paziente, mappate in coppie ai giunti, e sommate alla compensazione di gravità. Al termine di questa fase, $\boldsymbol{\tau}_{\text{totale}}$ è completamente determinato.
- **Fase B — Calcolo accelerazione:** l'equazione dinamica viene risolta dall'ABA con $\boldsymbol{\tau}_{\text{totale}}$ noto, ottenendo $\ddot{\boldsymbol{q}}$ come *unica* incognita.

Questa struttura garantisce che non vi siano ambiguità nel calcolo: la coppia è completamente determinata nella Fase A, e l'accelerazione è l'unica incognita nella Fase B.

5.4 Mappatura forze-coppie (Jacobiano trasposto)

Il Jacobiano geometrico $\boldsymbol{J}(\boldsymbol{q})$ mette in relazione le velocità ai giunti con la velocità cartesiana dell'end-effector. La stessa relazione, trasposta, permette di mappare forze cartesiane in coppie ai giunti.

Calcolo del Jacobiano

Il Jacobiano viene calcolato rispetto al frame dell'end-effector ed espresso nel sistema di riferimento world:

```
# Aggiornamento cinematica
pin.forwardKinematics(model, data, q)
pin.updateFramePlacements(model, data)

# Calcolo Jacobiano (6x68 per modello completo)
J_full = pin.computeFrameJacobian(
    model, data, q, ee_frame_id,
```

```
    pin.ReferenceFrame.LOCAL_WORLD_ALIGNED
)

# Estrazione colonne relative al braccio (6x7)
J_arm = J_full[:, arm_v_indices]

# Parte lineare del Jacobiano (3x7)
J_linear = J_arm[0:3, :]
```

La scelta del reference frame `LOCAL_WORLD_ALIGNED` garantisce che le forze e velocità siano espresse in un sistema con assi paralleli al world frame ma origine nell'end-effector, semplificando l'interpretazione fisica.

Mappatura delle forze

Le forze cartesiane del force field vengono mappate in coppie ai giunti mediante il Jacobiano trasposto:

$$\boldsymbol{\tau}_{\text{ext}} = \boldsymbol{J}(\boldsymbol{q})^T \cdot \boldsymbol{F}_{\text{cartesian}} \tag{5.3}$$

dove $\boldsymbol{F}_{\text{cartesian}} \in \mathbb{R}^3$ è la forza (solo componente lineare) calcolata dal force field e $\boldsymbol{\tau}_{\text{ext}} \in \mathbb{R}^7$ sono le coppie equivalenti ai giunti del braccio.

```
# Forza totale del force field
F_cartesian = F_chan + F_tan + F_vis + F_patient

# Mappatura a coppie ai giunti
tau_ext = J_linear.T @ F_cartesian
```

5.5 Integrazione numerica

La propagazione dello stato del sistema nel tempo avviene mediante integrazione numerica delle equazioni del moto. È stato scelto lo schema di Eulero esplicito per la sua semplicità implementativa e sufficiente accuratezza alle velocità tipiche delle applicazioni riabilitative.

Schema di Eulero esplicito

Lo schema di Eulero esplicito approssima le derivate temporali con differenze finite in avanti:

$$\dot{\boldsymbol{q}}(t + \Delta t) = \dot{\boldsymbol{q}}(t) + \ddot{\boldsymbol{q}}(t) \cdot \Delta t \tag{5.4}$$

$$\boldsymbol{q}(t + \Delta t) = \boldsymbol{q}(t) + \dot{\boldsymbol{q}}(t) \cdot \Delta t \tag{5.5}$$

dove Δt è il passo di integrazione. L'implementazione nel simulatore:

```
# Loop di integrazione
for step in range(num_steps):
    # 1. Calcola dinamica
    tau_total = tau_ext + tau_gravity
    q_ddot = pin.aba(model, data, q, q_dot, tau_total)

    # 2. Integrazione Eulero
    q_dot_new = q_dot + q_ddot * dt
    q_new = q + q_dot * dt

    # 3. Aggiorna stato
    q = q_new
    q_dot = q_dot_new
```

Scelta del timestep

La scelta del passo di integrazione Δt è critica per la stabilità numerica. Con i coefficienti di rigidezza utilizzati ($K_n = 250$ N/m), un passo troppo grande causa instabilità oscillatoria.

Attraverso test empirici è stato determinato che:

- $\Delta t = 10$ ms: instabile, oscillazioni divergenti
- $\Delta t = 1$ ms: stabile, risultati fisicamente plausibili

È stato quindi adottato $\Delta t = 1$ ms (frequenza 1000 Hz), che rappresenta lo standard in simulazioni robotiche con controllo in impedenza. Per una traiettoria di 20 secondi, questo corrisponde a 20000 passi di integrazione.

Condizioni iniziali e transitorio

La simulazione parte da condizioni di riposo: il robot è posizionato sul punto iniziale della traiettoria con velocità nulla. La traiettoria target, invece, è definita con velocità costante di 20 mm/s.

La simulazione parte da condizioni di riposo: il robot è posizionato sul punto iniziale della traiettoria con velocità nulla. La traiettoria target, invece, è definita con velocità costante di 20 mm/s.

Questa discontinuità iniziale tra la velocità nulla del sistema e la velocità del riferimento non introduce singolarità nel controllo in impedenza implementato, poiché la forza correttiva è proporzionale all'errore di posizione, non di velocità.

Al tempo t = 0 l'errore di posizione è pressoché nullo, quindi anche la forza generata dal force field è trascurabile. Nei timestep successivi, il target avanza progressivamente, l'errore cresce in modo graduale, e il sistema accelera seguendo il transitorio naturale di un sistema del secondo ordine fino a raggiungere la velocità di regime.

Questa dinamica è fisicamente corretta e non presenta singolarità. La distinzione fondamentale è tra traiettoria TARGET (riferimento geometrico con velocità costante) e traiettoria ACTUAL (effettivamente percorsa dal robot, che parte da riposo e accelera gradualmente).

Gestione dei limiti articolari

Durante l'integrazione, le posizioni ai giunti possono superare i limiti meccanici. È stato implementato un clipping hard che forza le configurazioni all'interno dei limiti ammissibili:

```
# Clipping ai limiti articolari
q_new = np.clip(q_new, q_min, q_max)

# Annulla velocita' se a contatto con il limite
for i in range(7):
    if q_new[i] == q_min[i] or q_new[i] == q_max[i]:
        q_dot_new[i] = 0.0
```

Questa strategia, sebbene non fisicamente rigorosa (non modella l'urto elastico con il fine corsa), è sufficiente per le applicazioni riabilitative dove il raggiungimento dei limiti è un evento raro e indesiderato.

Parametri della simulazione

I parametri utilizzati per la simulazione sono riassunti nella Tabella 5.1.

Tabella 5.1: Parametri della simulazione.

Parametro	**Simbolo**	**Valore**
Passo di integrazione	Δt	1 ms
Durata simulazione	T	20 s
Numero di passi	N	20000
Frequenza di campionamento	f_s	1000 Hz
Velocità traiettoria target	v_{target}	20 mm/s

Capitolo 6

Implementazione del Controllo in Impedenza

Questo capitolo descrive l'implementazione del sistema di controllo in impedenza basato su force field. Vengono presentate le equazioni implementate nel codice, il modello del paziente utilizzato per la simulazione, la logica di transizione tra le modalità di training, e i parametri calibrati per ottenere il comportamento desiderato.

6.1 Struttura del force field

Il force field implementato segue lo schema proposto da Zhang et al. [3], costruendo un tunnel virtuale attorno alla traiettoria di riferimento. All'interno di questo tunnel, tre componenti di forza cooperano per guidare il paziente: una forza normale che corregge le deviazioni laterali, una forza tangenziale che accompagna il movimento lungo la traiettoria, e una forza viscosa che stabilizza il sistema.

I valori dei coefficienti sono stati scelti in accordo con la letteratura sulla robotica riabilitativa. Studi su sistemi come MIT-Manus e Armeo Power indicano che rigidezze nell'intervallo 100–500 N/m sono appropriate per applicazioni di training assistito, con valori più bassi per pazienti spastici e più alti per modalità di protezione.

È opportuno precisare la natura delle forze nel contesto della presente simulazione. A differenza di un'implementazione su robot reale, dove le forze di interazione sono grandezze fisiche misurabili tramite sensori di forza/coppia, nel simulatore numerico qui sviluppato tutte le forze, la componente normale Fn, la tangenziale Ft, la viscosa Fv e la stessa forza del paziente Fpatient, sono quantità interamente computazionali. Ciascuna è determinata a partire da grandezze

cinematiche calcolate nel loop di simulazione: la differenza tra traiettoria di riferimento e posizione effettiva (per Fn), la direzione della traiettoria e la distanza dal target (per Ft), la velocità dell'end-effector (per Fv), e la differenza tra posizione intesa e posizione attuale (per Fpatient). Nella simulazione, queste forze calcolate vengono applicate assumendo un controllo di coppia ideale agli attuatori, privo di ritardi e attriti non modellati. Sul robot reale, sebbene la legge di controllo in impedenza operi calcolando parimenti forze virtuali a partire da errori cinematici, la presenza di sensori di forza/coppia diventa essenziale per stimare la reale interazione fisica robot-paziente e chiudere i loop di controllo a basso livello. Va inoltre osservato che questa assunzione di idealità non è una limitazione specifica della pipeline adottata (Pinocchio + TRAC-IK), ma è intrinseca a qualsiasi simulazione numerica: anche utilizzando un simulatore fisico come Gazebo, le forze sarebbero state calcolate internamente dal physics engine tramite equazioni numeriche, non misurate da sensori reali. Gazebo avrebbe aggiunto un modello fisico più ricco (attrito, contatto, giochi meccanici), ma l'assunzione di attuazione ideale sarebbe rimasta invariata. L'unico modo per superare questa assunzione è l'implementazione su robot reale con sensori di forza/coppia all'end-effector, che rappresenta uno sviluppo futuro necessario per confermare la trasferibilità dei risultati ottenuti.

6.1.1 Forza normale ($\boldsymbol{F}_{\mathbf{chan}}$)

La forza normale, o forza di canale, agisce perpendicolarmente alla traiettoria di riferimento con la funzione di riportare l'end-effector verso il centro del tunnel quando si verifica una deviazione trasversale. L'implementazione utilizza una legge elastica lineare:

$$\boldsymbol{F}_{\text{chan}} = K_n \cdot \boldsymbol{e}_{\text{trans}} \tag{6.1}$$

dove $K_n = 250$ N/m è il coefficiente di rigidezza normale e $\boldsymbol{e}_{\text{trans}}$ è il vettore di errore trasversale, calcolato come la componente dell'errore di posizione perpendicolare alla direzione della traiettoria.

È fondamentale sottolineare che la forza normale agisce solo sull'errore trasversale e non sull'errore longitudinale (ritardo lungo la traiettoria). Questa scelta implementativa preserva la libertà temporale del paziente: un ritardo nell'esecuzione del movimento non genera forze correttive, coerentemente con la filosofia AAN che privilegia la partecipazione attiva.

```
# Calcolo errore trasversale
```

```
error_vec = x_target - x_actual
tangent = trajectory_tangent  # direzione normalizzata

# Proiezione longitudinale (lungo traiettoria)
error_long = np.dot(error_vec, tangent) * tangent

# Errore trasversale (perpendicolare)
error_trans = error_vec - error_long

# Forza normale
F_normal = KN * error_trans
```

6.1.2 Forza tangenziale ($\boldsymbol{F}_{\mathbf{tan}}$)

La forza tangenziale agisce lungo la direzione della traiettoria con la funzione di accompagnare il paziente nel movimento. A differenza della forza normale che è sempre attiva, la forza tangenziale viene applicata solo in modalità Active Guidance (AG), quando il paziente dimostra buona capacità di seguire la traiettoria.

$$\boldsymbol{F}_{\text{tan}} = K_{\text{tan}} \cdot \alpha_{\text{spatial}} \cdot \alpha_{\text{fade}} \cdot \hat{\boldsymbol{t}} \tag{6.2}$$

dove $K_{\text{tan}} = 10$ N è l'intensità base della forza tangenziale e $\hat{\boldsymbol{t}}$ è il versore tangente alla traiettoria. La forza tangenziale è modulata da due fattori:

- **Fattore spaziale (α_{spatial}):** riduce linearmente la forza all'aumentare dell'errore trasversale, da 1.0 quando l'errore è nullo a 0.0 quando raggiunge la soglia D_t.
- **Fattore di fade-out (α_{fade}):** riduce linearmente la forza negli ultimi 20 mm prima del target finale, evitando overshoot e garantendo convergenza stabile.

Il fade-out spaziale rappresenta un contributo originale rispetto alla formulazione di Zhang et al., introdotto per migliorare la stabilità nella fase finale del movimento. Senza questo accorgimento, la forza tangenziale continuerebbe a spingere il paziente oltre il punto di arrivo, causando oscillazioni indesiderate.

```
if mode == 0:  # AG Mode
    # Modulazione spaziale
```

```
    spatial_factor = max(0.0, 1.0 - error_trans_norm /
DT_THRESHOLD)

    # Fade-out vicino al target (20mm)
    distance_to_target = total_distance -
current_position_along_traj
    if distance_to_target <= 0:
        fade_factor = 0.0
    elif distance_to_target < FADE_OUT_DISTANCE:
        fade_factor = distance_to_target /
FADE_OUT_DISTANCE
    else:
        fade_factor = 1.0

    F_tangent = KTAN * spatial_factor * fade_factor *
tangent_direction
else:
    F_tangent = np.zeros(3)  # Zero in AAN mode
```

6.1.3 Forza viscosa ($\boldsymbol{F}_{\text{vis}}$)

La forza viscosa introduce uno smorzamento proporzionale alla velocità dell'end-effector, con la funzione di stabilizzare il sistema e contrastare movimenti troppo rapidi o bruschi:

$$\boldsymbol{F}_{\text{vis}} = -K_d \cdot \boldsymbol{v} \tag{6.3}$$

dove $K_d = 30$ Ns/m è il coefficiente di smorzamento viscoso e $\boldsymbol{v}$ è la velocità cartesiana dell'end-effector. Il valore di K_d è stato calibrato per ottenere un buon compromesso tra stabilità e trasparenza.

6.1.4 Forza totale del force field

La forza totale generata dal force field è la somma delle tre componenti:

$$\boldsymbol{F}_{\text{field}} = \boldsymbol{F}_{\text{chan}} + \boldsymbol{F}_{\text{tan}} + \boldsymbol{F}_{\text{vis}} \tag{6.4}$$

Questa forza rappresenta l'assistenza che il robot fornisce al paziente e viene mappata in coppie ai giunti mediante il Jacobiano trasposto per essere applicata agli attuatori.

6.2 Modello del paziente

Per simulare l'interazione tra robot e paziente, è necessario un modello che rappresenti le forze esercitate dal braccio umano sull'end-effector. Il modello implementato include quattro componenti fisiche che caratterizzano il comportamento biomeccanico dell'arto superiore.

I parametri del modello paziente sono stati scelti in accordo con la letteratura sulla biomeccanica dell'arto superiore e sulla robotica riabilitativa. Studi su sistemi come MIT-Manus e Armeo Power forniscono range di riferimento per pazienti con diversi livelli di compromissione motoria.

6.2.1 Forza volontaria

La forza volontaria rappresenta l'intenzione motoria del paziente, modellata come una forza elastica che "tira" verso la posizione desiderata:

$$\boldsymbol{F}_{\text{vol}} = K_{\text{human}} \cdot (\boldsymbol{x}_{\text{intended}} - \boldsymbol{x}_{\text{actual}}) \tag{6.5}$$

dove $K_{\text{human}} = 50$ N/m è la rigidezza volontaria del paziente e $\boldsymbol{x}_{\text{intended}}$ è la posizione che il paziente intende raggiungere (che può differire dalla traiettoria di riferimento a causa del deficit motorio).

Il valore $K_{\text{human}} = 50$ N/m corrisponde a un paziente con tono muscolare nella norma. La letteratura riporta valori tipici di 0–20 N/m per pazienti flaccidi (post-ictus in fase acuta), 50–100 N/m per soggetti sani, e 200–600+ N/m per pazienti spastici.

6.2.2 Smorzamento neuromuscolare

Lo smorzamento neuromuscolare rappresenta la resistenza viscosa intrinseca del sistema muscolo-scheletrico:

$$\boldsymbol{F}_{\text{damp}} = -D_{\text{human}} \cdot \boldsymbol{v} \tag{6.6}$$

dove $D_{\text{human}} = 10$ Ns/m è il coefficiente di smorzamento neuromuscolare.

6.2.3 Peso del braccio

Il peso del braccio del paziente agisce come forza costante verso il basso:

$$\boldsymbol{F}_{\text{grav}} = \begin{bmatrix} 0 \\ 0 \\ -M_{\text{human}} \cdot g \end{bmatrix} \tag{6.7}$$

dove $M_{\text{human}} = 1.2$ kg è la massa dell'avambraccio e della mano, e $g = 9.81$ m/s^2 è l'accelerazione di gravità. La forza risultante è circa 11.8 N diretta verso il basso.

6.2.4 Forza inerziale

La forza inerziale rappresenta la resistenza del braccio umano alle accelerazioni:

$$\boldsymbol{F}_{\text{inert}} = -M_{\text{human}} \cdot \boldsymbol{a} \tag{6.8}$$

dove $\boldsymbol{a}$ è l'accelerazione cartesiana dell'end-effector, calcolata mediante differenze finite dalla velocità.

È importante notare che la massa del braccio umano M_{human} non è inclusa nella matrice di massa $\boldsymbol{M}(\boldsymbol{q})$ del robot calcolata da Pinocchio, che considera solo i link del manipolatore. La forza inerziale del paziente deve quindi essere modellata esplicitamente come forza esterna.

6.2.5 Forza totale del paziente

La forza totale esercitata dal paziente è la somma delle quattro componenti:

$$\boldsymbol{F}_{\text{patient}} = \boldsymbol{F}_{\text{vol}} + \boldsymbol{F}_{\text{damp}} + \boldsymbol{F}_{\text{grav}} + \boldsymbol{F}_{\text{inert}} \tag{6.9}$$

6.2.6 Giustificazione fisica del modello

La somma $\boldsymbol{F}_{\text{ext}} = \boldsymbol{F}_{\text{patient}} + \boldsymbol{F}_{\text{field}}$ nella formulazione dinamica merita un chiarimento. Potrebbe sembrare che le due forze debbano bilanciarsi (e quindi non possano essere sommate), ma questa interpretazione è errata e confonde il bilancio all'equilibrio con la formulazione dinamica generale.

Natura delle due forze. Entrambe le forze agiscono simultaneamente sull'end-effector e non sono mutuamente esclusive. La loro somma determina l'accelerazione del sistema secondo la seconda legge di Newton.

Necessità della componente inerziale $\boldsymbol{F}_{\text{inert}}$. Una possibile obiezione è che $\boldsymbol{F}_{\text{inert}} = -M_{\text{human}} \cdot \boldsymbol{a}$ rappresenti un "doppio conteggio" dell'inerzia, già presente nella matrice $\boldsymbol{M}(\boldsymbol{q})$. Questa obiezione è errata per la seguente ragione: il modello dinamico Pinocchio è costruito dall'URDF del robot TIAGo, che contiene solo la massa del robot (∼10 kg per il braccio). La massa del paziente ($M_{\text{human}} = 1.2$ kg) non è inclusa nella matrice $\boldsymbol{M}(\boldsymbol{q})$.

Se $\boldsymbol{F}_{\text{inert}}$ venisse rimossa, il paziente sarebbe modellato come un corpo senza massa, il che è fisicamente scorretto. La forza inerziale è quindi necessaria per rappresentare correttamente la dinamica del sistema accoppiato robot-paziente.

Bilancio vs dinamica. Il bilancio delle forze avviene solo all'equilibrio statico. Durante il movimento dinamico, è proprio lo squilibrio tra le forze a produrre l'accelerazione che fa evolvere il sistema.

6.2.7 Simulazione del deficit motorio

Per simulare un paziente con deficit motorio, la posizione "intended" viene fatta deviare dalla traiettoria di riferimento secondo un profilo sinusoidale:

$$\boldsymbol{x}_{\text{intended}}(t) = \boldsymbol{x}_{\text{target}}(t) + A \cdot \sin(2\pi f t) \cdot \hat{\boldsymbol{x}} \tag{6.10}$$

dove $A = 60$ mm è l'ampiezza della deviazione, $f = 0.1$ Hz è la frequenza, e $\hat{\boldsymbol{x}}$ è la direzione perpendicolare al movimento (asse X). Questa deviazione simula l'incapacità del paziente di mantenere una traiettoria rettilinea, tipica di deficit motori di entità medio-severa.

L'ampiezza di 60 mm è stata scelta per testare le transizioni tra le diverse modalità di training: con deviazioni minori il sistema rimarrebbe sempre in modalità AG, mentre con questa ampiezza si osservano transizioni tra AG e AAN.

6.3 Logica di switching tra modalità

Il sistema di controllo implementa due modalità operative con transizione automatica basata sull'errore di tracking trasversale. La scelta di utilizzare solo l'errore trasversale (e non quello totale) è coerente con il concetto di tunnel virtuale: ciò che conta è quanto il paziente devia lateralmente dalla traiettoria, non quanto è in ritardo lungo di essa.

Soglie di transizione

Le soglie che definiscono le transizioni tra modalità sono:

- $D_t = 39$ mm: soglia tra Active Guidance (AG) e Assist-As-Needed (AAN)
- $D_n = 50$ mm: soglia tra AAN e Restrictive Protection (RP)

Questi valori sono stati calibrati per ottenere transizioni significative tra le modalità con il profilo di deviazione simulato. Il paper di Zhang et al. utilizza valori di $D_t = 30$ mm e $D_n = 70$ mm; la scelta di soglie diverse riflette l'adattamento ai parametri specifici della simulazione.

Comportamento delle modalità

Le due modalità operative si differenziano per il livello di assistenza fornito:

Active Guidance (AG) — errore $<$ 39 mm. Il paziente segue adeguatamente la traiettoria. Il force field fornisce una guida leggera attraverso la forza tangenziale che accompagna il movimento, mentre la forza normale corregge piccole deviazioni. È la modalità ottimale: il paziente è protagonista e il robot accompagna.

Assist-As-Needed (AAN) — errore tra 39 mm e 50 mm. Il paziente ha difficoltà a mantenere la traiettoria. La forza tangenziale viene azzerata per massimizzare la trasparenza, mentre la forza normale aumenta proporzionalmente all'errore per riportare il paziente verso il centro del tunnel. Il robot fornisce solo l'assistenza strettamente necessaria.

Implementazione

La logica di switching è implementata come semplice confronto con soglie:

```
def determine_training_mode(error_trans_norm):
    if error_trans_norm < DT_THRESHOLD:  # 39 mm
        return 0  # AG - Active Guidance
    elif error_trans_norm < DA_THRESHOLD:  # 50 mm
        return 1  # AAN - Assist-As-Needed
    else:
        return 2  # RP - Restrictive Protection
```

La transizione avviene istantaneamente quando l'errore attraversa una soglia. Non è stata implementata isteresi poiché, con i parametri utilizzati, le oscillazioni spurie tra modalità non si sono rivelate problematiche.

6.4 Compensazione di gravità

Il controllore del robot implementa una strategia di compensazione di gravità pura, coerente con la filosofia AAN di minimo intervento:

$$\boldsymbol{\tau}_{\text{control}} = \boldsymbol{G}(\boldsymbol{q}) \tag{6.11}$$

dove $\boldsymbol{G}(\boldsymbol{q})$ è il vettore delle coppie gravitazionali calcolato da Pinocchio. Questa strategia fa sì che il braccio del robot "fluttui" senza cadere sotto il proprio peso, risultando trasparente alle forze esterne.

La coppia totale applicata ai motori è quindi:

$$\boldsymbol{\tau}_{\text{total}} = \boldsymbol{\tau}_{\text{ext}} + \boldsymbol{\tau}_{\text{control}} = \boldsymbol{J}^T \cdot \boldsymbol{F}_{\text{ext}} + \boldsymbol{G}(\boldsymbol{q}) \tag{6.12}$$

dove $\boldsymbol{\tau}_{\text{ext}}$ sono le coppie dovute alle forze esterne (paziente + force field) mappate attraverso il Jacobiano trasposto.

È importante notare che, sebbene la gravità del robot venga compensata dal controllore, i motori devono comunque generare fisicamente questa coppia. Nella simulazione, il termine $\boldsymbol{G}(\boldsymbol{q})$ appare sia nella dinamica del plant (come forza che fa cadere il braccio) sia nel controllore (come coppia che la compensa), annullandosi matematicamente ma rappresentando un carico reale sugli attuatori.

6.5 Parametri e calibrazione

Le tabelle seguenti riassumono tutti i parametri utilizzati nell'implementazione del controllo.

Tabella 6.1: Parametri del force field.

Parametro	**Simbolo**	**Valore**	**Unità**
Rigidezza normale	K_n	250	N/m
Smorzamento viscoso	K_d	30	Ns/m
Forza tangenziale	K_{tan}	10	N
Soglia tangenziale	D_t	39	mm
Soglia normale	D_n	50	mm
Distanza fade-out	–	20	mm

Tabella 6.2: Parametri del modello paziente.

Parametro	**Simbolo**	**Valore**	**Unità**
Rigidezza volontaria	K_{human}	50	N/m
Smorzamento neuromuscolare	D_{human}	10	Ns/m
Massa avambraccio	M_{human}	1.2	kg
Ampiezza deviazione	A	60	mm
Frequenza deviazione	f	0.1	Hz

Tabella 6.3: Parametri della traiettoria.

Parametro	**Valore**	**Unità**
Velocità target	20	mm/s
Durata	20	s
Lunghezza	400	mm
Direzione principale	asse Y	–

Processo di calibrazione

I parametri sono stati calibrati attraverso un processo iterativo con i seguenti obiettivi:

1. **Bilanciamento delle forze:** le forze del force field e del paziente devono essere dello stesso ordine di grandezza per ottenere un'interazione realistica.

2. **Transizioni tra modalità:** le soglie devono essere calibrate in modo che, con il profilo di deviazione simulato, si osservino transizioni significative tra AG e AAN.

3. **Stabilità numerica:** i coefficienti di rigidezza e smorzamento devono essere compatibili con il passo di integrazione scelto (1 ms).

4. **Rispetto dei limiti di coppia:** le coppie risultanti devono rimanere entro i limiti nominali degli attuatori del TIAGo.

Il risultato della calibrazione è un sistema che opera prevalentemente in modalità AG (circa 56% del tempo) con transizioni frequenti in modalità AAN (circa 44%), dimostrando il corretto funzionamento della logica di switching adattiva.

È importante sottolineare che i valori delle soglie D_t e D_n, così come l'ampiezza della deviazione sinusoidale A, non derivano da evidenze cliniche ma sono

il risultato di un processo di calibrazione iterativo finalizzato a ottenere transizioni significative tra le modalità di training nella simulazione. Il parametro $D_t = 39$ mm non rappresenta una soglia clinicamente validata né deriva da dati sperimentali su pazienti. Il valore è stato selezionato mediante calibrazione iterativa all'interno dell'ambiente simulativo con l'obiettivo di ottenere una distribuzione non degenerata tra le modalità AG e AAN durante il task considerato. Analogamente, il valore $A = 60$ mm è stato scelto per generare deviazioni sufficienti ad attivare tutte le modalità di controllo senza raggiungere sistematicamente la soglia di intervento restrittivo. I parametri devono pertanto essere interpretati esclusivamente come parametri di esercizio del modello simulativo.

Capitolo 7

Risultati e Discussione

7.1 Setup della simulazione

La validazione del sistema di controllo AAN è stata condotta attraverso una simulazione numerica completa della durata di 20 secondi. Lo scenario di test prevede un paziente con deficit motorio severo (deviazione sinusoidale di ampiezza 60 mm) che esegue un task di reaching lineare lungo l'asse Y del workspace.

La Tabella 7.1 riassume i parametri della simulazione, suddivisi per categoria funzionale.

Tabella 7.1: Parametri della simulazione.

Categoria	**Parametro**	**Valore**
Force Field	Rigidezza normale (K_n)	250 N/m
	Smorzamento (K_d)	30 Ns/m
	Forza tangenziale ($K_{\tan}$)	10 N
Modello Paziente	Rigidezza muscolare (K_{human})	50 N/m
	Smorzamento neuromuscolare	10 Ns/m
	Ampiezza deviazione (A)	60 mm
	Frequenza deviazione (f)	0.1 Hz
Soglie di Controllo	Soglia AG/AAN (D_t)	39 mm
	Soglia AAN/RP (D_n)	50 mm
Traiettoria	Lunghezza	400 mm
	Durata simulazione	20 s
	Timestep	1 ms (1 kHz)

Tutte le simulazioni riportate nel presente capitolo sono state eseguite utilizzando un singolo set di parametri nominali. Lo scopo dell'analisi non è valutare la sensibilità del sistema rispetto alla variazione dei parametri di controllo, ma

verificare il comportamento del framework implementato in una configurazione di riferimento. L'analisi parametrica e l'identificazione sistematica dei parametri costituiscono attività successive al presente lavoro.

7.2 Criteri di analisi dei risultati

Prima dell'analisi dei risultati è opportuno definire i criteri utilizzati per interpretare il comportamento del sistema simulato.

Poiché l'obiettivo del lavoro è verificare il corretto funzionamento del framework di controllo implementato, l'analisi non viene condotta in termini di efficacia riabilitativa ma attraverso indicatori cinematici e di controllo ottenuti direttamente dalla simulazione.

In particolare, nelle sezioni successive verranno considerate le seguenti grandezze:

- accuratezza di tracking della traiettoria cartesiana rispetto al riferimento;
- distribuzione temporale delle modalità di assistenza (AG, AAN e RP);
- andamento delle forze generate dal force field durante l'esecuzione del task;
- rispetto dei limiti dinamici e delle condizioni di stabilità del modello simulato.

L'interpretazione dei risultati verrà pertanto effettuata valutando se il controllore produce il comportamento atteso nello scenario simulato e se la transizione tra le modalità di assistenza avviene in modo coerente con le soglie definite nel Capitolo 6.

Gli indicatori adottati hanno esclusivamente finalità descrittiva del comportamento del modello implementato e non devono essere interpretati come metriche di efficacia clinica o come validazione sperimentale del sistema su piattaforma reale.

7.3 Validazione cinematica

Prima di analizzare i risultati del controllo, è necessario verificare l'accuratezza della catena cinematica. Come descritto nel Capitolo 5, il sistema utilizza TRAC-IK per la cinematica inversa e Pinocchio per la cinematica diretta e il calcolo dinamico.

La validazione è stata effettuata confrontando le posizioni cartesiane richieste con quelle ottenute tramite forward kinematics sulle configurazioni articolari calcolate da TRAC-IK. I risultati confermano l'elevata accuratezza della pipeline cinematica (Tabella 7.2).

Tabella 7.2: Validazione cinematica.

Metrica	**Valore**
Success rate IK	100%
Errore FK medio	0.04 mm
Errore FK massimo	0.08 mm

L'errore sub-millimetrico garantisce che eventuali discrepanze nei risultati del controllo non siano attribuibili a imprecisioni cinematiche.

La verifica cinematica effettuata consente di validare la consistenza numerica della trasformazione IK–FK adottata nella simulazione. Tale verifica non costituisce una validazione del controllo implementato né delle prestazioni fisiche del robot reale, ma garantisce che gli errori osservati successivamente siano attribuibili al comportamento del controllore e non alla catena cinematica.

7.4 Performance di tracking

La Figura 7.1 mostra l'andamento temporale dell'errore trasversale, ovvero la distanza perpendicolare dell'end-effector dalla linea della traiettoria target. Questo errore determina le transizioni tra le modalità di training.

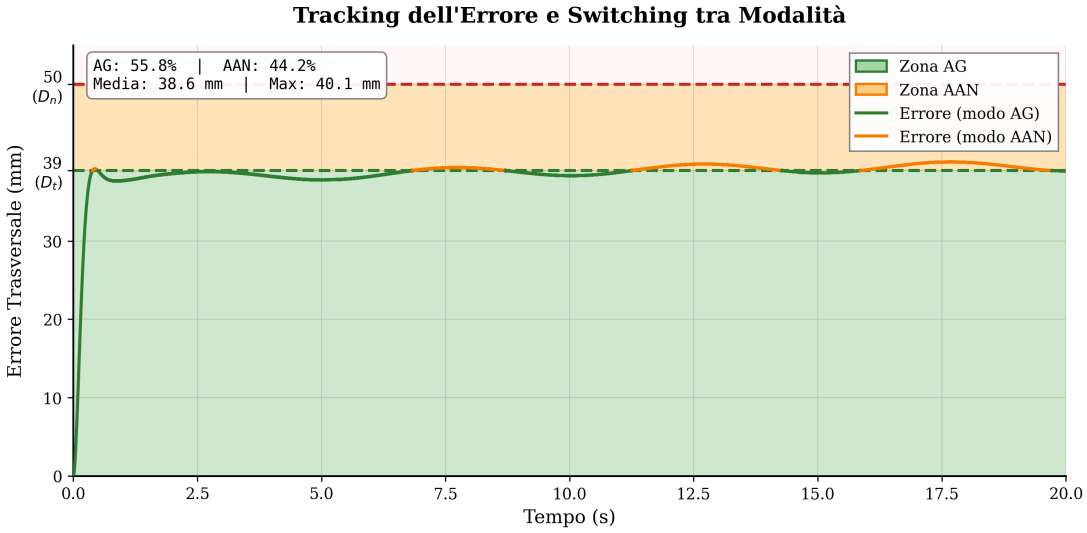

Figura 7.1: Tracking dell'errore trasversale e switching tra modalità.

Dopo un breve transitorio iniziale (∼0.5 s), l'errore converge a un valore di regime che oscilla attorno alla soglia $D_t = 39$ mm. Questo comportamento è coerente con la filosofia di controllo AAN: il sistema si stabilizza naturalmente al confine tra guida attiva (AG) e assistenza minima (AAN), adattandosi continuamente alle deviazioni del paziente.

Tabella 7.3: Statistiche dell'errore di tracking.

Metrica	**Valore**
Errore trasversale medio	38.6 mm
Errore trasversale massimo	40.1 mm
Deviazione standard	0.8 mm

L'errore massimo (40.1 mm) rimane al di sotto della soglia $D_n = 50$ mm, indicando che il sistema non entra mai in modalità RP (Restrictive Protection). Questo conferma che i parametri di controllo sono stati calibrati correttamente per il livello di deficit motorio simulato.

È importante osservare che il force field svolge contemporaneamente due funzioni distinte nel supporto al paziente. La componente principale dell'intervento è il sostegno del peso del braccio lungo l'asse Z: un paziente con deficit motorio severo non è in grado di sostenere autonomamente l'arto superiore (1.2 kg, pari a circa 11.8 N di carico gravitazionale), e il force field compensa questa insufficienza attraverso la forza normale. A questa si aggiunge la correzione delle deviazioni involontarie lungo l'asse X, che simulano lo spasmo o la perdita di coordinazione del paziente, con un picco di circa 3 N. La combinazione dei due contributi spiega perché l'errore trasversale medio (38.6 mm) si colloca in prossimità della soglia Dt = 39 mm: il sistema opera prevalentemente nel sostenere il braccio del paziente e secondariamente nel correggere le deviazioni laterali, producendo la distribuzione bilanciata tra le modalità AG e AAN osservata nella simulazione.

7.5 Distribuzione delle modalità di training

La Figura 7.2 mostra la distribuzione percentuale del tempo trascorso in ciascuna modalità di training durante la simulazione.

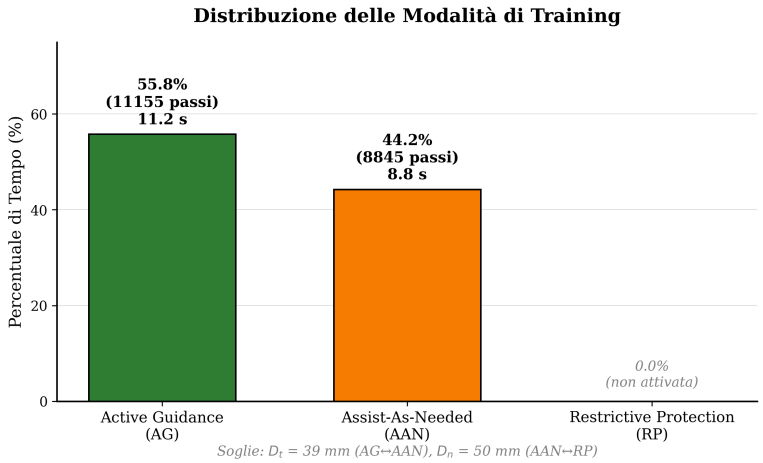

Figura 7.2: Distribuzione delle modalità di training.

La distribuzione bilanciata (55.8% AG, 44.2% AAN, 0% RP) dimostra che il sistema opera nel regime desiderato: il paziente riceve guida attiva quando segue adeguatamente la traiettoria, mentre il robot diventa più trasparente quando l'errore aumenta moderatamente. L'assenza di attivazione della modalità RP indica che il paziente, nonostante il deficit motorio severo, non ha mai raggiunto deviazioni critiche che richiedessero un intervento protettivo.

7.6 Bilancio delle forze

La Figura 7.3 presenta l'analisi del bilancio delle forze nel sistema. Il grafico superiore confronta le forze totali esercitate dal robot (force field) e dal paziente; il grafico inferiore mostra le singole componenti del force field.

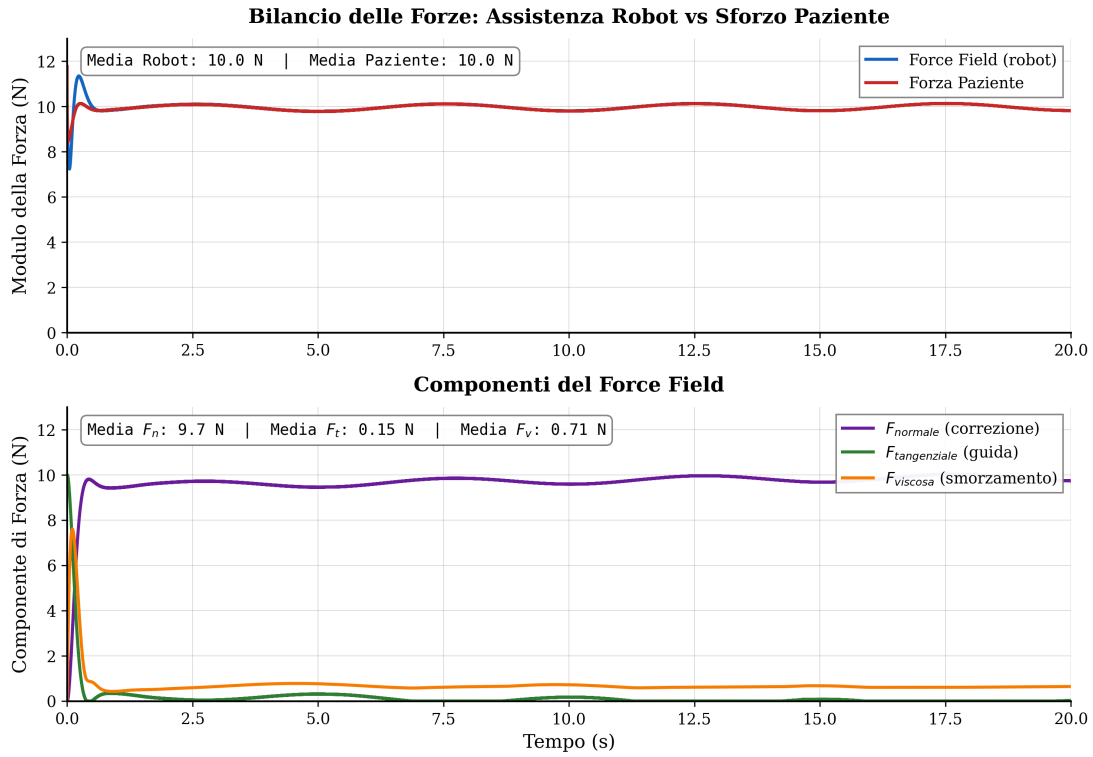

Figura 7.3: Bilancio delle forze robot-paziente.

Grafico superiore. Dopo il transitorio iniziale, le forze del robot e del paziente convergono a valori simili (∼10 N). Questo equilibrio dinamico è coerente con il principio dei lavori virtuali: in regime quasi-stazionario, la forza del force field bilancia la forza volontaria del paziente.

Grafico inferiore. La componente normale F_n (correzione trasversale) domina con una media di 9.7 N, mentre la componente tangenziale F_t (guida lungo traiettoria) risulta quasi nulla (0.15 N). Questo è coerente con il fatto che l'errore oscilla attorno alla soglia D_t: in modalità AAN la forza tangenziale è disattivata, e in modalità AG viene modulata dal fattore spaziale che la riduce quando l'errore si avvicina alla soglia. La componente viscosa F_v contribuisce con 0.71 N medi allo smorzamento delle oscillazioni.

7.7 Analisi delle coppie articolari

La Figura 7.4 confronta le coppie massime raggiunte durante la simulazione con i limiti nominali dei motori del TIAGo.

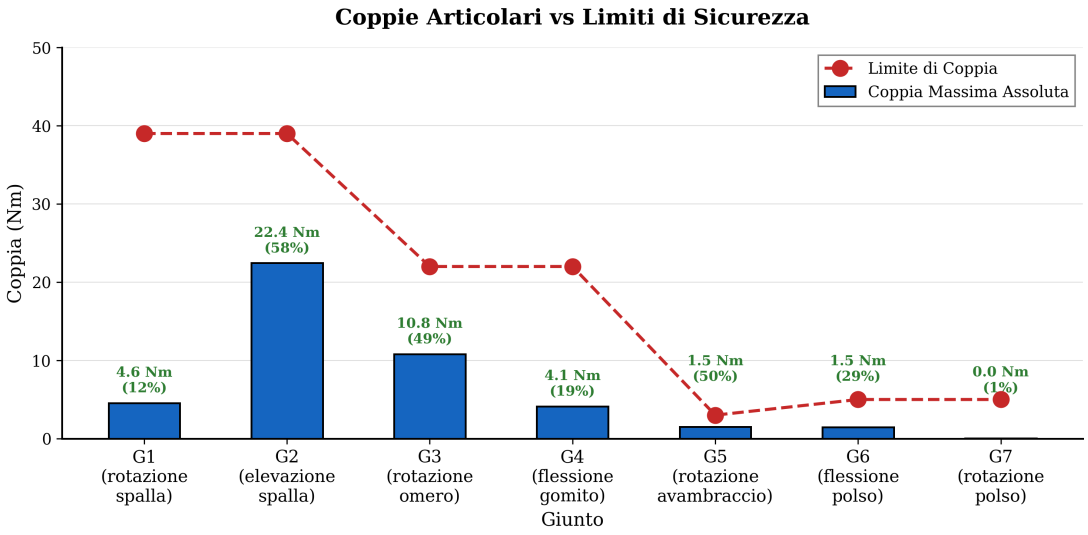

Figura 7.4: Coppie articolari vs limiti di sicurezza.

Tabella 7.4: Coppie articolari massime.

Giunto	**Coppia Max**	**Limite**	**Saturazione**
G1 (rotazione spalla)	4.6 Nm	39 Nm	12%
G2 (elevazione spalla)	22.4 Nm	39 Nm	58%
G3 (rotazione omero)	10.8 Nm	22 Nm	49%
G4 (flessione gomito)	4.1 Nm	22 Nm	19%
G5 (rotazione avambraccio)	1.5 Nm	3 Nm	50%
G6 (flessione polso)	1.5 Nm	5 Nm	29%
G7 (rotazione polso)	0.0 Nm	5 Nm	1%

Il giunto 2 (elevazione della spalla) risulta il più sollecitato con il 58% della coppia nominale. Questo è atteso per un task di reaching orizzontale: il giunto deve sostenere il peso del braccio contro la gravità. Tutti i giunti rimangono comunque al di sotto del 60% dei rispettivi limiti, confermando ampi margini di sicurezza per l'implementazione su robot reale.

7.8 Visualizzazione della traiettoria

La Figura 7.5 offre una visualizzazione tridimensionale della traiettoria eseguita, con i cilindri concentrici che rappresentano i confini del tunnel di controllo.

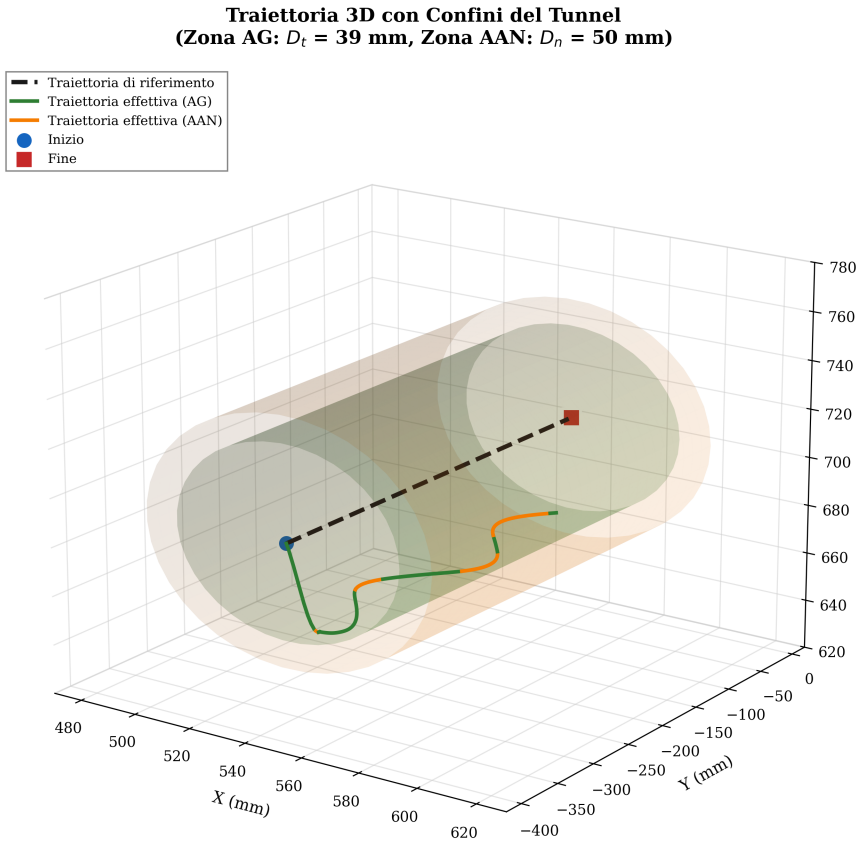

Figura 7.5: Traiettoria 3D con confini del tunnel.

La traiettoria effettiva (linea colorata) rimane interamente contenuta all'interno del tunnel esterno ($D_n = 50$ mm), oscillando tra la zona AG (verde) e la zona AAN (arancione). Il transitorio iniziale è chiaramente visibile come la deviazione verso il basso all'inizio del movimento, seguita dalla convergenza verso la linea della traiettoria target.

La deviazione prevalente lungo l'asse Z, visibile nella traiettoria tridimensionale, è coerente con il contributo gravitazionale discusso nella Sezione 7.3.

7.9 Discussione

Figura 7.6: Rappresentazione concettuale della sessione riabilitativa dal punto di vista del paziente. Sul monitor sono visibili il tunnel del force field e l'errore di tracking in tempo reale (immagine generata tramite intelligenza artificiale).

Coerenza con la letteratura

I risultati ottenuti sono coerenti con l'approccio force field proposto da Zhang et al. [3]. La distribuzione bilanciata tra modalità AG e AAN dimostra che il sistema implementa correttamente la filosofia Assist-As-Needed: il robot fornisce assistenza solo quando necessario, lasciando al paziente la responsabilità del movimento quando le sue capacità lo permettono.

Limiti della simulazione

I risultati ottenuti devono essere interpretati come verifica della coerenza interna del framework simulativo implementato. In particolare, il corretto funzionamento del controllore all'interno dello scenario numerico non implica automaticamente la trasferibilità delle prestazioni osservate a un sistema robotico reale né consente conclusioni sull'efficacia riabilitativa del metodo.

La validazione presenta alcune limitazioni che dovranno essere affrontate in sviluppi futuri:

Modello del paziente semplificato. Il paziente è stato modellato come un sistema massa-molla-smorzatore con deviazione sinusoidale deterministica. Un paziente reale presenterebbe comportamenti più complessi, incluse variazioni sto-

castiche dell'intenzione motoria e affaticamento progressivo. La scelta di un modello deterministico è stata effettuata per isolare il comportamento del controllore dalle variabilità introdotte dal soggetto e consentire una verifica iniziale dell'implementazione software. L'introduzione di modelli stocastici o adattativi del paziente rappresenta un'estensione del lavoro e non un prerequisito per la validazione numerica preliminare qui proposta.

Traiettoria rettilinea. Il task simulato è un semplice reaching lineare. Traiettorie curve o task più complessi (es. reaching-to-grasp) potrebbero sollecitare il sistema in modi diversi.

Assenza di rumore sensoriale. La simulazione non include rumore nelle misure di posizione e forza. Su un robot reale, il controllo dovrà gestire l'incertezza sensoriale.

Trasferibilità al robot reale

I margini di sicurezza sulle coppie articolari (massimo 58%) suggeriscono che il sistema possa essere implementato su robot reale senza modifiche sostanziali ai parametri di controllo. Tuttavia, sarà necessario:

- Implementare un loop di controllo in tempo reale a 1 kHz
- Integrare sensori di forza/coppia per la stima delle forze di interazione
- Validare la stabilità del sistema in presenza di ritardi di comunicazione
- Condurre test di sicurezza secondo le normative IEC 80601-2-78 per dispositivi riabilitativi

Capitolo 8

Conclusioni e Sviluppi Futuri

8.1 Sintesi del lavoro svolto

Il presente lavoro di tesi ha affrontato la progettazione e la validazione di un sistema di controllo in impedenza basato su force field per il robot TIAGo di PAL Robotics, finalizzato ad applicazioni di riabilitazione degli arti superiori secondo la filosofia Assist-As-Needed.

L'obiettivo principale era sviluppare un simulatore numerico in grado di riprodurre l'interazione tra robot e paziente durante un task riabilitativo, implementando lo schema di controllo proposto da Zhang et al. [3] e adattandolo alla piattaforma TIAGo. Il lavoro ha richiesto l'integrazione di diverse componenti software e teoriche:

- Cinematica inversa mediante il risolutore TRAC-IK, con strategia multi-seed per massimizzare il success rate su un manipolatore ridondante a 7 DOF.
- Calcolo dinamico mediante la libreria Pinocchio, utilizzando l'Articulated Body Algorithm per la forward dynamics con complessità $O(n)$.
- Implementazione del force field con tre componenti (normale, tangenziale, viscosa) e logica di switching automatico tra le modalità Active Guidance e Assist-As-Needed.
- Modellazione del paziente con deficit motorio, includendo forza volontaria, smorzamento neuromuscolare, inerzia e peso dell'arto superiore.

I risultati della simulazione hanno confermato il corretto funzionamento del sistema. L'errore di tracking trasversale si stabilizza attorno alla soglia $D_t = 39$ mm,

con transizioni frequenti e bilanciate tra le modalità AG (55.8%) e AAN (44.2%). Le coppie articolari rimangono entro il 58% dei limiti nominali degli attuatori, confermando ampi margini di sicurezza. Il bilancio delle forze mostra un equilibrio dinamico coerente tra le forze del force field e quelle del paziente, con la componente normale che domina la risposta correttiva.

8.2 Contributi originali

Il lavoro presenta diversi contributi rispetto allo stato dell'arte:

Adattamento dello schema force field alla piattaforma TIAGo. Lo schema proposto da Zhang et al. è stato originariamente sviluppato e validato sul sistema RaRTS (Robot-assisted Rehabilitation Training System), un sistema a 6 DOF specifico per la riabilitazione. Il presente lavoro dimostra l'applicabilità dello stesso approccio a un robot di servizio general-purpose a 7 DOF, ampliando le potenziali piattaforme utilizzabili per la riabilitazione robotica.

Simulatore numerico completamente integrato. È stato sviluppato un simulatore che integra cinematica inversa (TRAC-IK), dinamica completa del manipolatore (Pinocchio) e modello del paziente in un unico flusso computazionale, senza dipendenza da simulatori fisici come Gazebo. Questo approccio garantisce completa riproducibilità dei risultati e pieno controllo su ogni aspetto della simulazione.

Fade-out tangenziale nella fase finale del movimento. È stato introdotto un meccanismo di fade-out per la forza tangenziale negli ultimi 20 mm prima del target, non presente nella formulazione originale di Zhang et al. Questo accorgimento previene l'overshoot e migliora la stabilità nella fase di convergenza.

Documentazione completa della metodologia. La tesi fornisce una descrizione dettagliata di ogni scelta implementativa, dalle configurazioni dell'ambiente di sviluppo ai parametri di calibrazione, con l'obiettivo di rendere il lavoro riproducibile e utilizzabile come base per sviluppi futuri.

Il lavoro fornisce una base software e metodologica per future attività di integrazione su piattaforma reale e per successive campagne sperimentali orientate alla caratterizzazione dell'interazione fisica uomo–robot.

8.3 Limitazioni

La validazione del sistema presenta alcune limitazioni che è importante riconoscere:

Modello del paziente deterministico. Il paziente è stato modellato come un sistema massa-molla-smorzatore con deviazione sinusoidale fissa. Un paziente reale presenterebbe comportamenti stocastici, variazioni dell'intenzione motoria, affaticamento progressivo durante la sessione, e risposte non lineari del sistema neuromuscolare.

Traiettoria rettilinea. Il task simulato è un semplice reaching lineare lungo l'asse Y. Traiettorie più complesse (curve, tridimensionali, task funzionali come reaching-to-grasp) solleciterebbero il sistema in modi diversi e potrebbero evidenziare comportamenti non osservati nella simulazione attuale.

Assenza di rumore sensoriale e ritardi di comunicazione. La simulazione opera in condizioni ideali, senza rumore nelle misure di posizione e forza e senza ritardi nel loop di controllo. Su un robot reale, questi fattori potrebbero influire significativamente sulla stabilità e sulle prestazioni del sistema.

Validazione esclusivamente numerica. I risultati sono stati ottenuti in simulazione e non sono stati verificati sperimentalmente su robot fisico né con soggetti umani. La trasferibilità dei risultati al mondo reale, sebbene supportata dai margini di sicurezza osservati, resta da dimostrare.

Modalità RP non attivata. Con i parametri calibrati, la modalità Restrictive Protection non è mai stata attivata durante la simulazione. Sebbene questo indichi una buona calibrazione delle soglie per il livello di deficit simulato, il comportamento del sistema in condizioni di deviazione estrema non è stato verificato.

8.4 Sviluppi futuri

Sulla base dei risultati ottenuti e delle limitazioni identificate, si delineano diverse direzioni per lo sviluppo futuro del sistema:

Validazione su robot fisico. Il passo prioritario è l'implementazione del sistema di controllo sul robot TIAGo reale, con loop di controllo a 1 kHz. Questo richiederà l'integrazione di sensori di forza/coppia al polso per la stima delle forze di interazione e la gestione dei ritardi di comunicazione nel framework ROS.

Traiettorie e task riabilitativi realistici. L'estensione a traiettorie circolari, spirali e task funzionali (es. raggiungere e afferrare oggetti) consentirebbe di valutare il sistema in scenari clinicamente più rilevanti. La traiettoria circolare utilizzata da Zhang et al. nei loro esperimenti rappresenterebbe un primo benchmark comparativo diretto.

Modello del paziente avanzato. L'introduzione di componenti stocastiche nell'intenzione motoria, l'effetto dell'affaticamento (decadimento progressivo di K_{human} durante la sessione), e la modellazione della spasticità (rigidezza dipendente dalla velocità) renderebbero la simulazione più rappresentativa delle condizioni cliniche reali.

Rigidezza adattiva del force field. L'implementazione di un meccanismo di adattamento automatico dei coefficienti del force field (in particolare K_n) in funzione delle prestazioni cumulative del paziente consentirebbe al sistema di regolare progressivamente il livello di assistenza durante la sessione riabilitativa e tra sessioni successive.

Trial clinici con pazienti. La validazione finale del sistema richiederà studi clinici controllati con pazienti affetti da deficit motori dell'arto superiore, in conformità con le normative IEC 80601-2-78 per dispositivi medici riabilitativi. Questi studi dovranno valutare non solo l'efficacia terapeutica ma anche l'accettabilità del sistema da parte dei pazienti e dei terapisti.

Interfaccia per il terapista. Lo sviluppo di un'interfaccia grafica che consenta al terapista di configurare i parametri della sessione (soglie, rigidezze, traiettoria), monitorare in tempo reale le prestazioni del paziente e visualizzare i progressi nel tempo faciliterebbe l'adozione clinica del sistema.

Bibliografia

[1] Laura Marchal-Crespo e David J. Reinkensmeyer. «Review of control strategies for robotic movement training after neurologic injury». In: *Journal of NeuroEngineering and Rehabilitation* 6.1 (2009), p. 20.

[2] Neville Hogan. «Impedance Control: An Approach to Manipulation: Part I—Theory». In: *Journal of Dynamic Systems, Measurement, and Control* 107.1 (1985), pp. 1–7.

[3] Leigang Zhang et al. «Force-Field based assisted control for upper-limb rehabilitation robots». In: *Biomedical Signal Processing and Control* 99 (2025), p. 106896. DOI: 10.1016/j.bspc.2024.106896.

[4] Albin Bajrami et al. «Posture Optimization of the TIAGo Highly-Redundant Robot for Grasping Operation». In: *Robotics* 13.4 (2024). Università Politecnica delle Marche, Ancona, p. 56. DOI: 10.3390/robotics13040056.

[5] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer, 2008. ISBN: 978-0-387-74314-1.

[6] Patrick Beeson e Barrett Ames. «TRAC-IK: An open-source library for improved solving of generic inverse kinematics». In: *IEEE-RAS International Conference on Humanoid Robots* (2015), pp. 928–935.

[7] Justin Carpentier et al. *The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives*. https://github.com/stack-of-tasks/pinocchio. 2019.

Appendice A

Codice Sorgente Principale

In questa appendice viene riportato il codice sorgente principale del simulatore AAN (`ImpedanceControlTiago.py`). Per ragioni di spazio, vengono incluse solo le sezioni più significative: i parametri di configurazione, le funzioni di calcolo del force field e del modello del paziente, e il loop di simulazione. Le funzioni ausiliarie (cinematica inversa, salvataggio CSV, stampa statistiche) sono omesse.

A.1 Parametri di configurazione

```
import numpy as np
import pinocchio as pin
from trac_ik_python.trac_ik import IK

# File paths
URDF_PATH = "/home/marco/tesi_ws/src/aan_simulation/tiago_full.
    urdf"

# Model indices - Configuration space (q)
ARM_Q_INDICES = [23, 24, 25, 26, 27, 28, 29]
TORSO_Q_INDEX = 22

# Model indices - Velocity space (v)
ARM_V_INDICES = [13, 14, 15, 16, 17, 18, 19]
TORSO_V_INDEX = 12

# System parameters
M_ROBOT = 10.0       # kg - robot arm mass
M_HUMAN = 1.2        # kg - human arm mass (forearm + hand)
M_TOTAL = M_ROBOT + M_HUMAN

```

```
# AAN Control Parameters
KN = 250.0            # N/m - normal stiffness (force field)
KD = 30.0             # Ns/m - damping coefficient
KTAN = 10.0           # N - tangential guidance force (AG mode)

# Patient Model Parameters
K_HUMAN = 50.0        # N/m - human voluntary stiffness
D_HUMAN = 10.0        # Ns/m - human neuromuscular damping

# Training mode thresholds (transversal error only)
DT_THRESHOLD = 0.039  # 39mm - AG/AAN threshold
DA_THRESHOLD = 0.050  # 50mm - AAN/RP threshold

# Trajectory parameters
X_START, Y_START, Z_START = 0.53, -0.4, 0.7
X_END, Y_END, Z_END = 0.53, 0.0, 0.7
QX, QY, QZ, QW = 0.0, 0.7071068, 0.0, 0.7071068

# Trajectory geometry (computed automatically)
TRAJ_START = np.array([X_START, Y_START, Z_START])
TRAJ_END = np.array([X_END, Y_END, Z_END])
TRAJ_TANGENT_VECTOR = (TRAJ_END - TRAJ_START)
TRAJ_TANGENT_NORM = np.linalg.norm(TRAJ_TANGENT_VECTOR)
TRAJ_TANGENT_NORMALIZED = TRAJ_TANGENT_VECTOR / TRAJ_TANGENT_NORM
TRAJ_TOTAL_DISTANCE = TRAJ_TANGENT_NORM
FADE_OUT_DISTANCE = 0.020  # 20mm fade-out zone

# Simulation parameters
DURATION = 20.0
DT = 0.001
NUM_STEPS = int(DURATION / DT)

# Patient deviation (severe motor impairment)
PATIENT_DEVIATION_AMPLITUDE = 0.060  # 60mm
PATIENT_DEVIATION_FREQUENCY = 0.1    # 0.1 Hz

# Torque limits (Nm)
TORQUE_LIMITS = np.array([39.0, 39.0, 22.0, 22.0, 3.0, 5.0, 5.0])
```

Listing A.1: Parametri del sistema.

A.2 Struttura della classe principale

```
class AANSimulator:
    """Complete Assist-As-Needed Simulator with Force Field
   Control"""

    def __init__(self):
        """Initialize simulator"""
        with open(URDF_PATH, 'r') as f:
            self.urdf_string = f.read()

        # Initialize TRAC-IK
        self.ik_solver = IK(
            "base_footprint", "arm_7_link",
            timeout=0.005, epsilon=1e-4,
            solve_type="Speed", urdf_string=self.urdf_string
        )

        # Load Pinocchio model
        self.model = pin.buildModelFromUrdf(URDF_PATH)
        self.data = self.model.createData()
        self.ee_frame_id = self.model.getFrameId("arm_7_link")
        self.trajectory_tangent = TRAJ_TANGENT_NORMALIZED

        # Generate trajectory and compute IK
        self.generate_cartesian_trajectory()
        self.compute_ik_trajectory()
        self.verify_fk_accuracy()

    def get_ee_position(self, q_arm, torso_lift):
        """Forward kinematics: q -> x"""
        q_full = pin.neutral(self.model)
        q_full[TORSO_Q_INDEX] = torso_lift
        for i, q_idx in enumerate(ARM_Q_INDICES):
            q_full[q_idx] = q_arm[i]

        pin.forwardKinematics(self.model, self.data, q_full)
        pin.updateFramePlacements(self.model, self.data)
        return self.data.oMf[self.ee_frame_id].translation.copy()

    def get_jacobian(self, q_arm, torso_lift):
        """Compute end-effector Jacobian [3x7]"""
        q_full = pin.neutral(self.model)
        q_full[TORSO_Q_INDEX] = torso_lift
        for i, q_idx in enumerate(ARM_Q_INDICES):
```

```
            q_full[q_idx] = q_arm[i]

        pin.forwardKinematics(self.model, self.data, q_full)
        pin.updateFramePlacements(self.model, self.data)

        J_full = pin.computeFrameJacobian(
            self.model, self.data, q_full, self.ee_frame_id,
            pin.ReferenceFrame.LOCAL_WORLD_ALIGNED
        )
        J_trans = J_full[:3, :]
        J_arm = J_trans[:, ARM_V_INDICES]
        return J_arm
```

Listing A.2: Inizializzazione del simulatore e funzioni cinematiche.

A.3 Calcolo del force field

```
def compute_errors(self, x_target, x_actual):
    """
    Compute longitudinal and transversal errors.
    Transversal error determines tunnel boundaries and mode
    switching.
    """
    error_vec = x_target - x_actual
    tangent = self.trajectory_tangent

    # Longitudinal error (projection onto tangent)
    error_long_scalar = np.dot(error_vec, tangent)
    error_long = error_long_scalar * tangent

    # Transversal error (perpendicular component)
    error_trans = error_vec - error_long
    error_trans_norm = np.linalg.norm(error_trans)

    return error_long, error_trans, error_trans_norm

def compute_force_field(self, mode, error_trans, error_trans_norm
    ,
                        v_actual, x_actual):
    """
    Calcola campo di forze AAN con fade-out spaziale
    della forza tangenziale.

```

```
    - F_normal: sempre attiva, riporta verso traiettoria
    - F_viscous: sempre attiva, oppone velocita'
    - F_tangent: solo in AG mode, con modulazione spaziale e fade-out
    """
    # === FORZE SEMPRE ATTIVE ===
    F_normal = KN * error_trans
    F_viscous = -KD * v_actual

    # === FORZA TANGENZIALE (solo AG mode) ===
    if mode == 0:  # AG Mode
        tangent_direction = self.trajectory_tangent

        # Fattore spaziale: 1.0 quando error=0, 0.0 quando error>=DT
        spatial_factor = max(0.0, 1.0 - (error_trans_norm / DT_THRESHOLD))

        # Fade-out: riduzione lineare negli ultimi 20mm
        current_pos = np.dot(x_actual - TRAJ_START, self.trajectory_tangent)
        distance_to_target = TRAJ_TOTAL_DISTANCE - current_pos

        if distance_to_target <= 0:
            fade_factor = 0.0
        elif distance_to_target < FADE_OUT_DISTANCE:
            fade_factor = distance_to_target / FADE_OUT_DISTANCE
        else:
            fade_factor = 1.0

        F_tangent = KTAN * spatial_factor * fade_factor * tangent_direction
    else:
        # AAN e RP: nessuna spinta tangenziale
        F_tangent = np.zeros(3)

    F_total = F_normal + F_viscous + F_tangent
    return F_total, F_normal, F_tangent, F_viscous
```

Listing A.3: Decomposizione dell'errore e calcolo del campo di forze.

A.4 Modello del paziente

```
def compute_patient_deviation(self, t):
    """Deviazione sinusoidale lungo X (perpendicolare al
    movimento)"""
    dev_x = PATIENT_DEVIATION_AMPLITUDE * np.sin(
        2 * np.pi * PATIENT_DEVIATION_FREQUENCY * t)
    return np.array([dev_x, 0.0, 0.0])

def compute_patient_force(self, x_actual, v_actual, v_previous,
                          dt, x_intended):
    """
    Forza del paziente - modello completo a 4 componenti:
    1. Volontaria (elastica): verso posizione intesa
    2. Smorzamento neuromuscolare: resistenza viscosa
    3. Gravitazionale: peso del braccio umano
    4. Inerziale: resistenza all'accelerazione
    """
    # 1. Forza volontaria
    F_voluntary = K_HUMAN * (x_intended - x_actual)

    # 2. Smorzamento neuromuscolare
    F_damping = -D_HUMAN * v_actual

    # 3. Forza gravitazionale
    F_gravity = np.array([0.0, 0.0, -M_HUMAN * 9.81])

    # 4. Forza inerziale
    if v_previous is None:
        a_actual = np.zeros(3)
    else:
        a_actual = (v_actual - v_previous) / dt
    F_inertial = -M_HUMAN * a_actual

    return F_voluntary + F_damping + F_gravity + F_inertial

def determine_training_mode(self, error_trans_norm):
    """Determina modalita' basata su errore TRASVERSALE (tunnel)
    """
    if error_trans_norm < DT_THRESHOLD:
        return 0  # AG - Active Guidance
    elif error_trans_norm < DA_THRESHOLD:
        return 1  # AAN - Assist-As-Needed
    else:
        return 2  # RP - Restrictive Protection
```

Listing A.4: Modello del paziente con quattro componenti di forza.

A.5 Loop di simulazione

```
def simulate_step(self, t_idx, q_actual, q_dot_actual, v_previous, dt):
    """Simulate one timestep with Euler integration"""
    t = t_idx * dt
    torso_pos = self.torso_trajectory[t_idx]

    # 1. Forward kinematics: q -> x
    x_actual = self.get_ee_position(q_actual, torso_pos)

    # 2. Jacobian for velocity mapping
    J_arm = self.get_jacobian(q_actual, torso_pos)
    v_actual = J_arm @ q_dot_actual

    # 3. Target and intended positions
    x_target = self.x_target_trajectory[t_idx]
    deviation = self.compute_patient_deviation(t)
    x_intended = x_target + deviation

    # 4. Compute errors (longitudinal + transversal)
    error_long, error_trans, error_trans_norm = \
        self.compute_errors(x_target, x_actual)

    # 5. Determine training mode (transversal error only)
    mode = self.determine_training_mode(error_trans_norm)

    # 6. Compute forces
    F_patient = self.compute_patient_force(
        x_actual, v_actual, v_previous, dt, x_intended)
    F_field, F_normal, F_tangent, F_viscous = \
        self.compute_force_field(
            mode, error_trans, error_trans_norm, v_actual, x_actual)
    F_ext_cart = F_patient + F_field

    # 7. Map to joint torques (Jacobian transpose)
    tau_ext = J_arm.T @ F_ext_cart

    # 8. Torque limits
```

```
    tau_ext_clipped = np.clip(tau_ext, -TORQUE_LIMITS,
TORQUE_LIMITS)

    # 9. Build full configuration for dynamics
    q_full = pin.neutral(self.model)
    q_dot_full = np.zeros(self.model.nv)
    q_full[TORSO_Q_INDEX] = torso_pos
    for i, q_idx in enumerate(ARM_Q_INDICES):
        q_full[q_idx] = q_actual[i]
    for i, v_idx in enumerate(ARM_V_INDICES):
        q_dot_full[v_idx] = q_dot_actual[i]

    # 10. Build torque vector
    tau_full = np.zeros(self.model.nv)
    for i, v_idx in enumerate(ARM_V_INDICES):
        tau_full[v_idx] = tau_ext_clipped[i]

    # 11. Gravity compensation (robot passive controller)
    pin.computeAllTerms(self.model, self.data, q_full, q_dot_full
)
    tau_gravity = self.data.g.copy()
    tau_control = tau_gravity.copy()
    tau_total = tau_full + tau_control

    # 12. Forward dynamics (ABA)
    q_ddot_full = pin.aba(
        self.model, self.data, q_full, q_dot_full, tau_total)

    # 13. Extract arm accelerations
    q_ddot_arm = q_ddot_full[ARM_V_INDICES]

    # 14. Euler integration
    q_dot_new = q_dot_actual + q_ddot_arm * dt
    q_new = q_actual + q_dot_actual * dt

    return q_new, q_dot_new, v_actual, data_dict
```

Listing A.5: Singolo passo di simulazione con integrazione di Eulero.

```
def run_simulation(self):
    """Main simulation loop"""
    q_actual = self.q_arm_trajectory[0].copy()
    q_dot_actual = np.zeros(7)
    v_previous = None
    simulation_data = []
```

```
7
8       for i in range(NUM_STEPS):
9           q_new, q_dot_new, v_actual, data_dict = \
10              self.simulate_step(i, q_actual, q_dot_actual,
    v_previous, DT)
11
12          simulation_data.append(data_dict)
13          q_actual = q_new
14          q_dot_actual = q_dot_new
15          v_previous = v_actual.copy()
16
17      self.save_results_csv(simulation_data)
18      self.print_statistics(simulation_data)
19      return simulation_data
20
21  if __name__ == "__main__":
22      sim = AANSimulator()
23      simulation_data = sim.run_simulation()
```

Listing A.6: Loop principale della simulazione.

Appendice B

Parametri del Modello TIAGo

In questa appendice vengono riportati i parametri principali del robot TIAGo estratti dall'URDF e dal datasheet ufficiale.

I valori riportati nella presente appendice rappresentano i parametri finali utilizzati nella versione conclusiva del simulatore. Eventuali valori differenti presenti nel testo descrivono configurazioni intermedie utilizzate durante la fase di calibrazione.

B.1 Limiti dei giunti

Tabella B.1: Limiti di posizione dei giunti del braccio

Giunto	**Min [rad]**	**Max [rad]**	**Range [deg]**
arm_1_joint	-1.18	1.57	157.5
arm_2_joint	-1.18	1.57	157.5
arm_3_joint	-3.14	2.29	311.0
arm_4_joint	-0.32	2.27	148.4
arm_5_joint	-2.07	2.07	237.2
arm_6_joint	-1.39	1.39	159.4
arm_7_joint	-2.07	2.07	237.2
torso_lift_joint	0.00	0.35	350 mm

B.2 Coppie nominali degli attuatori

Tabella B.2: Specifiche degli attuatori del braccio TIAGo

Modulo	**Riduzione**	**Velocità max [rpm]**	**Coppia nom. [Nm]**
1st module (arm_1)	100:1	18	39
2nd module (arm_2)	100:1	18	39
3rd module (arm_3)	100:1	22	22
4th module (arm_4)	100:1	22	22
Wrist 1st DoF (arm_5)	336:1	17	3
Wrist 2nd DoF (arm_6)	336:1	17	5
Wrist 3rd DoF (arm_7)	336:1	17	5

B.3 Parametri del controllo in impedenza

Tabella B.3: Parametri del force field calibrati

Parametro	**Valore**	**Unità**
K_n (rigidezza normale)	250	N/m
K_d (smorzamento)	30	Ns/m
K_{tan} (guida tangenziale)	10	N
D_t (soglia AG/AAN)	39	mm
D_n (soglia AAN/RP)	50	mm

Tabella B.4: Parametri del modello paziente

Parametro	**Valore**	**Unità**
M_{human} (massa braccio)	1.2	kg
D_{human} (smorzamento)	10	Ns/m
Ampiezza deviazione	60	mm
Frequenza deviazione	0.1	Hz

B.4 Indici dei giunti nel modello Pinocchio

Il modello URDF del TIAGo caricato in Pinocchio ha 68 gradi di libertà totali (configurazione) e 58 gradi di libertà di velocità. Gli indici rilevanti per il braccio destro sono:

Tabella B.5: Indici dei giunti nel modello Pinocchio

Giunto	**Indice q (config)**	**Indice v (velocità)**
torso_lift_joint	22	21
arm_1_joint	23	22
arm_2_joint	24	23
arm_3_joint	25	24
arm_4_joint	26	25
arm_5_joint	27	26
arm_6_joint	28	27
arm_7_joint	29	28

Appendice C

Comandi di Setup Ambiente

In questa appendice vengono riportati tutti i comandi necessari per replicare l'ambiente di sviluppo utilizzato in questo lavoro.

C.1 Installazione WSL e Ubuntu

```
# Aprire PowerShell come amministratore

# Installare WSL
wsl --install

# Installare Ubuntu 20.04
wsl --install -d Ubuntu-20.04

# Verificare installazione
wsl -l -v

# Avviare Ubuntu
wsl -d Ubuntu-20.04
```

Listing C.1: Installazione WSL su Windows

C.2 Installazione ROS Noetic

```
# Configurare repository
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(
   lsb_release -sc) main" > /etc/apt/sources.list.d/ros-
   latest.list'

# Aggiungere chiavi GPG
sudo apt install curl -y
curl -s https://raw.githubusercontent.com/ros/rosdistro/
   master/ros.asc | sudo apt-key add -

# Aggiornare sistema
sudo apt update
sudo apt upgrade -y

# Installare ROS Noetic (versione completa)
sudo apt install ros-noetic-desktop-full -y

# Installare dipendenze per build
sudo apt install python3-rosdep python3-rosinstall python3-
   rosinstall-generator python3-wstool build-essential -y

# Inizializzare rosdep
sudo rosdep init
rosdep update

# Configurare ambiente (aggiungere a ~/.bashrc)
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Listing C.2: Installazione completa ROS Noetic

C.3 Creazione workspace e installazione pacchetti TIAGo

```
# Creare workspace
mkdir -p ~/tiago_public_ws/src
```

```
cd ~/tiago_public_ws/src

# Clonare repository TIAGo (opzionale, per modello URDF)
git clone https://github.com/pal-robotics/tiago_robot.git
git clone https://github.com/pal-robotics/tiago_description.git

# Compilare workspace
cd ~/tiago_public_ws
catkin_make

# Configurare ambiente
echo "source ~/tiago_public_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Listing C.3: Setup workspace TIAGo

C.4 Installazione TRAC-IK

```
# Metodo 1: Da repository ROS (consigliato)
sudo apt install ros-noetic-trac-ik -y

# Metodo 2: Compilazione da sorgente
cd ~/tiago_public_ws/src
git clone https://bitbucket.org/traclabs/trac_ik.git
cd ~/tiago_public_ws
catkin_make

# Verificare installazione
python3 -c "from trac_ik_python.trac_ik import IK; print('TRAC-IK OK')"
```

Listing C.4: Installazione TRAC-IK

C.5 Installazione Pinocchio

```
# Installare da repository ROS
```

```
sudo apt install ros-noetic-pinocchio ros-noetic-eigenpy -y

# Configurare PYTHONPATH
echo 'export PYTHONPATH=$PYTHONPATH:/opt/ros/noetic/lib/
   python3/dist-packages' >> ~/.bashrc
source ~/.bashrc

# Verificare installazione
python3 -c "import pinocchio as pin; print('Pinocchio
   version:', pin.__version__)"
```

Listing C.5: Installazione Pinocchio

C.6 Installazione dipendenze Python

```
# Librerie numeriche e grafiche
pip3 install numpy matplotlib pandas --break-system-
   packages

# Verificare installazioni
python3 << EOF
import numpy as np
import matplotlib
import pandas as pd
print(f"NumPy: {np.__version__}")
print(f"Matplotlib: {matplotlib.__version__}")
print(f"Pandas: {pd.__version__}")
EOF
```

Listing C.6: Installazione librerie Python

C.7 Creazione progetto simulazione

```
# Creare struttura progetto
mkdir -p ~/tesi_ws/src/aan_simulation/results
cd ~/tesi_ws/src/aan_simulation

# Copiare URDF del TIAGo (se non presente)
```

```
cp ~/tiago_public_ws/src/tiago_description/robots/
   tiago_full.urdf .

# Creare file principale
touch ImpedanceControlTiago.py
chmod +x ImpedanceControlTiago.py

# Verificare struttura
ls -la
# Dovrebbe mostrare:
# ImpedanceControlTiago.py
# tiago_full.urdf
# results/
```

Listing C.7: Setup progetto simulazione

C.8 Setup Docker e Gazebo (fase esplorativa)

```
# Installare Docker
sudo apt install docker.io -y
sudo usermod -aG docker $USER

# Riavviare sessione per applicare gruppo docker
# (logout e login, oppure: newgrp docker)

# Installare rocker
pip3 install rocker --break-system-packages

# Scaricare immagine TIAGo
docker pull palroboticssl/tiago_tutorials:noetic

# Avviare container con supporto grafico
rocker --home --user --nvidia --x11 --privileged \
    palroboticssl/tiago_tutorials:noetic

# All'interno del container, avviare Gazebo:
# roslaunch tiago_gazebo tiago_gazebo.launch public_sim:=
   true
```

Listing C.8: Setup Docker per Gazebo

C.9 Script di avvio rapido

Per facilitare l'avvio dell'ambiente, si consiglia di creare uno script:

```
#!/bin/bash
# File: ~/setup_env.sh

# Source ROS
source /opt/ros/noetic/setup.bash

# Source workspace TIAGo (se necessario)
if [ -d ~/tiago_public_ws/devel ]; then
    source ~/tiago_public_ws/devel/setup.bash
fi

# Configurare PYTHONPATH per Pinocchio
export PYTHONPATH=$PYTHONPATH:/opt/ros/noetic/lib/python3/
   dist-packages

# Andare al progetto
cd ~/tesi_ws/src/aan_simulation

echo "Ambiente configurato. Pronto per eseguire la
   simulazione."
```

Listing C.9: Script setup_env.sh

Rendere eseguibile e utilizzare:

```
chmod +x ~/setup_env.sh
source ~/setup_env.sh
python3 ImpedanceControlTiago.py
```