Università degli Studi di Modena e Reggio Emilia
Dipartimento di Ingegneria "Enzo Ferrari"

Laurea Magistrale in
Data Engineering and Analytics

Interazione fisica Uomo-Robot mobile per il trasporto collaborativo

Candidato:
Matteo Tagliavini

Relatore:
Prof. Luigi Biagiotti

Anno Accademico 2024-2025

Indice

Abstract

Il lavoro di Tesi propone l'integrazione ed il controllo di un manipolatore collaborativo UR10e con una base mobile omnidirezionale Robotnik RB-KAIROS+, unendo la versatilità di un braccio antropomorfo alla locomozione su ruote.
Il lavoro è articolato in tre parti principali. In una prima fase mi sono concentrato sullo studio dei due sistemi robotizzati separati, sia dal punto di vista hardware che software, per imparare ad utilizzarli in modo indipendente. Mi sono concentrato in particolare su moduli di mappatura, localizzazione e navigazione per il robot mobile, mentre per il manipolatore collaborativo sull'utilizzo del software proprietario, integrazione con ROS e framework MoveIt. In un secondo momento mi sono dedicato alla manipolazione e movimentazione dell'intero sistema assemblato tramite un controllo in ammettenza, proponendo così una diversa modalità di interazione da parte dell'operatore. Questo approccio si basa sulla conversione delle forze esercitate dall'operatore nella parte terminale del braccio robotico, rilevate tramite un sensore integrato, in corrispondenti velocità articolari.
In tal modo il braccio robotico si muove in risposta diretta al contatto fisico con l'operatore, cedendo all'interazione e permettendo un'operatività collaborativa naturale. Successivamente ho implementato nel controllo una tecnica di divisione delle frequenze per separare il dato di velocità in due componenti distinte: una per comandare la base mobile e per controllare il braccio UR10e. L'ultima parte del lavoro presenta una prima valutazione comparativa nella quale il controllo sviluppato viene confrontato con altri possibili metodi ed approcci.
In conclusione, l'approccio proposto offre una modalità innovativa di interazione uomo-robot, potenzialmente applicabile a molteplici scenari industriali ed accademici di manipolazione mobile collaborativa, eliminando la dipendenza da interfacce e da comandi manuali convenzionali.

Introduzione

La presente tesi nasce da una collaborazione tra l'azienda Ideativa e i laboratori universitari AutoLab e IdeaLab, con l'obiettivo di sviluppare un sistema per la movimentazione sicura di carichi in ambienti dinamici tramite l'interazione fisica diretta tra uomo e robot. Questo progetto riveste un grande interesse applicativo per il contesto industriale, in quanto consente di introdurre robot collaborativi mobili nei processi produttivi, migliorando la sicurezza degli operatori e la flessibilità operativa.

Negli ultimi anni l'utilizzo dei Cobot, i robot collaborativi, ha trovato ampia diffusione in molto contesti industriali e di ricerca. Questa tipologia di robot permette di lavorare a stretto contatto con l'operatore umano e di collaborare al raggiungimento e completamento di task condivisi, riducendo lo sforzo umano in attività faticose. Allo stato dell'arte, l'utilizzo dei cobot è spesso confinato a postazioni fisse di lavoro e senza la possibilità di essere riconfigurate.

All'interno di questo scenario, il presente lavoro propone l'integrazione di un manipolatore collaborativo UR10e su una piattaforma mobile omnidirezionale Robotnik RB-KAIROS+. L'obiettivo è consentire un'interazione fisica diretta e intuitiva tra l'operatore umano e il sistema robotico, realizzando un trasporto collaborativo efficiente e sicuro. In altre parole, l'utente ha la possibilità di guidare manualmente il robot afferrando il braccio, combinando i movimenti del manipolatore e della base mobile in modo naturale. Questa soluzione mira a superare le limitazioni delle postazioni fisse, permettendo al cobot di muoversi insieme all'operatore attraverso spazi di lavoro condivisi e dinamici.

I risultati sperimentali ottenuti sul robot reale confermano la validità dell'approccio proposto. Il sistema sviluppato offre una modalità innovativa di interazione uomo-robot in applicazioni pratiche come la movimentazione di materiali, la logistica di magazzino o l'assistenza in linea di produzione; una tale soluzione permette di operare in sicurezza senza dover ricorrere a interfacce di controllo tradizionali. Questo contribuisce ad aumentare l'ergonomia e l'efficacia delle operazioni collaborative, riducendo il carico di lavoro umano e migliorando l'adattabilità del sistema robotico all'ambiente circostante.

L'elaborato di tesi è articolato nei seguenti capitoli:

- **Capitolo 1 - Robotica collaborativa e robot mobili**: fornisce una panoramica dei concetti chiave della robotica collaborativa e mobile, descrivendo separatamente nel dettaglio i due robot utilizzati e le loro caratteristiche tecniche.

- **Capitolo 2 – Configurazione del sistema di navigazione**: descrive il funzionamento e l'implementazione del sistema di navigazione della piattaforma mobile. Vengono illustrate la creazione della mappa dell'ambiente, gli algoritmi di localizzazione e le strategie di pianificazione del movimento messe a disposizione dal robot mobile per muoversi in sicurezza.

- **Capitolo 3 – Setting up manipulation**: viene descritta la configurazione del braccio UR10e tramite il framework MoveIt, la generazione del relativo pacchetto di configurazione e l'utilizzo dell'interfaccia di controllo che consente di pianificare ed eseguire i movimenti del braccio sia in ambiente simulato sia sul robot reale.

- **Capitolo 4 – Controllo**: illustra lo sviluppo del controllo coordinato tra manipolatore e base mobile. Viene proposto un algoritmo di controllo in ammettenza per il sistema combinato, basato su un modello massa-smorzatore virtuale, che permette al braccio robotico di reagire in modo immediato e controllato alle forze esercitate dall'operatore. Inoltre, il capitolo introduce la strategia di micro/macro-manipolazione basata sulla separazione in frequenza del comando, andando così a ripartire i movimenti tra base e braccio.

- **Capitolo 5 – Risultati sperimentali**: illustra i testi svolti sul sistema e presenta i risultati ottenuti. Vengono riportati i confronti quantitativi in termini di energia e sforzo umano tra l'algoritmo di controllo proposto e altri approcci esistenti in letteratura. Inoltre, è inclusa un'analisi qualitativa dell'esperienza d'uso tramite il questionario NASA-TLX sottoposto ad un insieme di partecipanti.

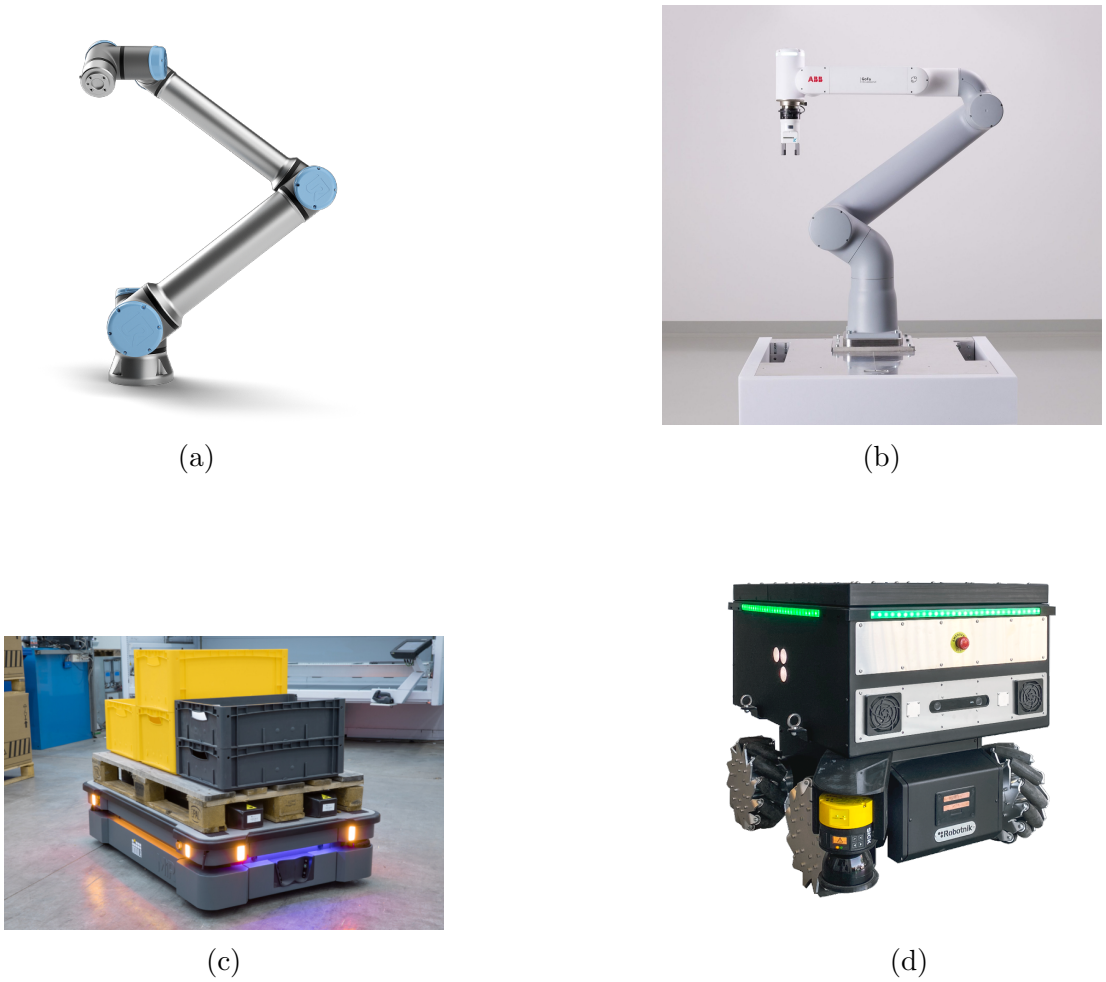
Capitolo 1

Robotica collaborativa e robot mobili

Negli ultimi decenni la robotica ha vissuto un periodo di forte innovazione e profonda evoluzione: i robot, fin ad ora confinati dietro barriere protettive lontano dagli operatori, possono oggi condividere in sicurezza lo spazio di lavoro con l'uomo. Si parla così di robotica collaborativa, in cui i robot (o cobot) sono progettati e realizzati appositamente per poter interagire fisicamente con gli esseri umani nello stesso ambiente operativo. La norma ISO 10218-2 contiene una sezione specifica dedicata al funzionamento dei robot collaborativi, definendolo come un tipo particolare di operatività che prevede la collaborazione tra una persona e un robot nello stesso spazio di lavoro. Tale sezione fornisce informazioni sui requisiti generali, sui requisiti relativi allo spazio di lavoro e descrive le differenti modalità operative che possono essere selezionate per garantire un ambiente di lavoro sicuro [1].

Parallelamente allo sviluppo della robotica collaborativa, sta acquisendo sempre maggiore rilievo anche l'impiego dei robot mobili autonomi (Autonomous Mobile Robots, AMR). Secondo la norma ISO 8373, un robot mobile è definito come un dispositivo robotico in grado di spostarsi autonomamente nell'ambiente operativo sotto il proprio controllo, senza la necessità di guide fisiche o interventi continui da parte di un operatore umano [2]. L'introduzione degli AMR ha permesso un incremento significativo dell'efficienza, della flessibilità operativa e della sicurezza in molteplici ambiti applicativi: nell'industria manifatturiera, per la rapida adattabilità della produzione; nella logistica, per ottimizzare la movimentazione automatizzata delle merci; in ambito sanitario, per compiti di distribuzione e sanificazione; e in contesti pubblici o commerciali, per supportare servizi di consegna, informazione e sorveglianza.

In questo capitolo viene presentata una panoramica tecnica e strutturale delle due piattaforme robotiche usate in questo lavoro: il braccio robotico collaborativo UR10e dell'azienda danase Universal Robots e la piattaforma mobile RB-Kairos+ dell'azienda spagnola Robotnik. Nei paragrafi seguenti verranno analizzati separatamente, per ciascun robot, i seguenti aspetti: la struttura meccanica, le interfacce elettriche e di comunicazione, l'architettura software e gli ambienti di programmazione, le caratteristiche legate alla sicurezza, e alcuni ambiti applicativi descritti in letteratura. Infine, verrà descritta l'integrazione fisica e funzionale tra il robot ur10e e la piattaforma RB-Kairos+, con particolare attenzione agli aspetti legati all'assemblaggio e all'interoperabilità.



(a) (b)

(c) (d)

Figura 1.1: (a) Universal Robot UR10e [3], (b) ABB GoFa CRB 15000 [4], (c) Modula AMR [5], (d) Robotnik RB-Kairos [6]

1.1 Universal Robots UR10e

1.1.1 Composizione meccanica e struttura

L'UR10e è un robot collaborativo a sei gradi di libertà (6-DoF) prodotto da Universal Robots, costruito con una struttura articolata in serie che include base, spalla, gomito e polso con tre giunti di polso, conferendogli così un'architettura antropomorfa. Un grado di libertà in un robot si riferisce alla capacità di movimento indipendente che il robot può effettuare, ogni grado rappresenta un tipo specifico di movimento, come rotazione o traslazione, che contribuisce alla versatilità del robot. [7]. Il carico utile massimo al polso è pari a 10 kg, mentre la portata del braccio è di 1300 mm, il che permette al robot di coprire un'ampia area di lavoro grazie anche all'intervallo di rotazione $\pm 360°$ di tutti i suoi giunti. La ripetibilità di posizionamento dichiarata è di $\pm 0{,}05$ mm, indice di un'elevata precisione, adatta anche ad operazioni di assemblaggio fine. [8].

Per quanto riguarda la struttura, il robot impiega materiali leggeri e resistenti: le strutture dei link sono in lega di alluminio con elementi esterni di plastica tecnica PC/ASA, per un peso complessivo di 33 kg [8].

L'UR10e rappresenta il miglioramento della precedente versione UR10, è dotato infatti di un sensore di forza incorporato nel giunto del polso (flangia tool), permettendo così al robot di misurare forze esterne e rilevare eventuali collisioni [3].

Questa caratteristica distingue la e-Series di Universal Robots dalle versioni precedenti, introducendo nuove funzionalità collaborative e di adattamento alla presenza umana. Dal punto di vista della resistenza e protezione, il braccio ha un grado di protezione IP54, sufficiente a contrastare polvere e spruzzi leggeri, il suo livello di rumorosità è inferiore a 65 dB(A) durante il funzionamento. La base di montaggio ha diametro di 190 mm e dispone di fori per il fissaggio standard [8].

Figura 1.2: Robot collaborativo UR10e

Figura 1.3: UR10e in attività di manipolazione

1.1.2 Interfacce elettriche e di comunicazione

L'architettura hardware dell'UR10e prevede un control box esterno che fornisce alimentazione del robot ed elettronica di controllo. L'unità di controllo standard dell'UR10e opera con alimentazione di rete AC (100-240 VAC) e fornisce al braccio la tensione necessaria; esiste tuttavia anche una versione OEM DC pensata per l'integrazione su piattaforme mobili, che consente di alimentare il robot tramite batterie [8]. Questa versione OEM facilita l'uso dell'UR10e su sistemi mobili come RB-KAIROS+, eliminando la necessità di collegamento alla rete elettrica fissa.

Il control box dell'UR10e mette a disposizione numerose interfacce I/O per l'integrazione con sensori ed attuatori esterni, in particolare, sono presenti 16 ingressi digitali, 16 uscite digitali, 2 ingressi analogici e 2 uscite analogiche sul controllore, con alimentazione ausiliaria a 24 V/2 A disponibile per circuiti esterni. Sulla flangia robot sono inoltre presenti connessioni I/O dedicate: 2 ingressi e 2 uscite digitali, più 2 ingressi analogici, con un'alimentazione selezionabile a 12 V o 24 V (fino a 2 A) per alimentare utensili o end-effector montati sul polso robot [8]. Questo permette di collegare ad esempio pinze elettriche, sensori di prossimità o altri dispositivi direttamente all'estremità del braccio, sfruttando il cablaggio interno.

Per quanto riguarda le interfacce di comunicazione, l'UR10e supporta protocolli industriali su Ethernet come MODBUS TCP, Ethernet/IP e PROFINET nativamente, permettendo di integrarsi facilmente in linee automatizzate e scambiare dati con PLC o altri controller di fabbrica. Il control box dispone di porte di rete RJ45 e consente la comunicazione TCP/IP sia per scopi di programmazione sia per la connettività con sistemi di visione. Inoltre, vi sono porte USB (USB 2.0 e 3.0) sul pannello di controllo che permettono di collegare dispositivi di storage (per aggiornamenti software, log, ecc.) o periferiche supportate [8]. Tramite le interfacce di rete, è possibile utilizzare l'SDK di Universal Robots o driver ROS per comandare il robot in tempo reale: ad esempio, l'Universal Robots ROS Driver ufficiale fornisce un'interfaccia di comunicazione che espone i giunti, l'end-effector e i sensori dell'UR10e all'interno dell'ecosistema ROS [9].

1.1.3 Architettura software e ambienti di programmazione

Al robot può essere integrata l 'unità di programmazione touchscreen a colori, chiamata Teach Pendant, su cui è installato PolyScope, il software proprietario di Universal Robots dedicato alla programmazione. Tramite PolyScope l'operatore può guidare manualmente il braccio in modalità "freedrive" e creare sequenze di movimenti e operazioni attraverso un approccio grafico basato su blocchi, senza la necessità di conoscere un linguaggio di programmazione testuale. Questo rende l'UR10e accessibile anche ad utenti non programmatori, velocizzando la condifenda con il sistema [10]. L'interfaccia consente di impostare waypoint, inserire comandi di I/O, aggiungere logica condizionale e definire routine, il tutto direttamente dal pannello di controllo touch.

In parallelo alla programmazione grafica, l'UR10e supporta un proprio linguaggio di scripting denominato URScript. Ogni programma creato graficamente può infatti essere consultato e modificato come codice URScript, un linguaggio testuale ad alto livello con sintassi semplice, pensato per controllare movimenti, velocità, accelerazioni, I/O e altre funzioni del robot. Gli utenti avanzati possono scrivere script personalizzati per implementare programmi più complessi, algoritmi di controllo o integrazioni con sensori esterni. Il robot espone un'interfaccia socket TCP/IP attraverso cui è possibile inviare comandi URScript da un computer remoto, permettendo così il telecontrollo da parte di un programma esterno in tempo reale. Universal Robots fornisce inoltre una dashboard server (per comandi di alto livello come caricamento programma dalla memoria, avvio/arresto programma, ricevere feedback di stato dal robot) e l'RTDE (Real-Time Data Exchange), un'interfaccia per scambiare dati ad alta frequenza (500 Hz) tra robot e sistemi esterni, rendendolo particolarmente adatto alle applicazioni che richiedono tempi e sincronizzazione precisi [11].

L'ambiente software è estendibile tramite gli URCaps, estensioni software che consentono di integrare hardware aggiuntivo, creare nuove funzionalità o migliorare l'interfaccia utente della GUI. Nel caso di RB-KAIROS+, Robotnik fornisce un URCap dedicato che permette di visualizzare informazioni e dati della la base mobile direttamente dall'interfaccia UR Polyscope [12]. In aggiunta, Universal Robots mette a disposizione un simulatore software (URSim) che replica in locale l'interfaccia PolyScope sul proprio pc, utile per programmare e testare le sequenze offline prima di eseguirle sul robot reale.

1.1.4 Caratteristiche di sicurezza

L'UR10e è progettato sin dall'origine come robot collaborativo (cobot), pensato per operare in sicurezza a fianco degli operatori umani senza barriere fisiche. Per questo motivo integra un insieme di funzioni di sicurezza avanzate sia a livello hardware che software. In primo luogo, come già accennato, il robot è dotato di sensori di forza/coppia sull'ultimo giunto che permette di rilevare prontamente collisioni o contatti imprevisti con persone o oggetti. In caso di urto, l'UR10e è in grado di arrestare il movimento in pochi millisecondi, limitando l'energia trasferita nell'impatto. Il controllo di sicurezza implementa la limitazione di potenza e forza, monitorando costantemente le coppie ai giunti e la forza al TCP, mantenendole entro soglie predefinite in modo da non causare danni in caso di contatto [13].

Il sistema di controllo dell'UR10e include funzioni di sicurezza configurabili (Safety Functions): in totale il cobot offre 17 funzioni di sicurezza parametrizzabili, certificate fino a Performance Level d, Categoria 3 secondo la norma EN ISO 13849-1 [8]. Queste funzioni comprendono, ad esempio, la limitazione di velocità, limitazione di forza, monitoraggio di posizioni/cartesiano, controllo di arresto di emergenza, etc. Tramite l'interfaccia PolyScope l'utente può regolare i parametri di sicurezza (ad es. velocità massima consentita in modalità collaborativa, soglie di forza, zone di sicurezza con limiti di movimento) per adattare il robot al contesto applicativo e alle normative ISO/TS 15066 per la robotica collaborativa.

Dal punto di vista hardware, l'UR10e dispone di un pulsante di Emergency Stop sia sul teach pendant sia sul control box, con circuito di sicurezza ridondato che interrompe immediatamente potenza ai motori in caso di emergenza. Inoltre, il teach pendant è dotato di un pulsante a tre posizioni (deadman switch) che garantisce sicurezza durante la programmazione manuale: se l'operatore rilascia o preme a fondo, il robot si arresta, mentre mantenendolo a metà corsa consente il movimento in modalità manuale controllata. L'insieme di questi accorgimenti rende l'UR10e conforme agli standard internazionali di sicurezza per robot industriali collaborativi (ISO 10218-1, ISO/TS 15066).

1.1.5 Ambiti applicativi in letteratura

L'UR10e, caratterizzato da un'elevata capacità di carico, un'ampia portata e la sicurezza collaborativa, viene impiegato in numerose applicazioni sia industriali che di ricerca. In ambito industriale, questo cobot viene utilizzato per automatizzare processi come: assemblaggio e avvitatura, manipolazione di materiali e asservimento macchine, operazioni di imballaggio e palletizzazione, saldatura, levigatura e lucidatura [14].Universal Robots identifica tra i settori di impiego tipici i settori automotive, alimentare, packaging, elettronica, sanitario e molti altri, testimonianza della versatilità di questo braccio. La natura collaborativa del robot permette spesso di inserirlo in isole di lavoro condivise con operatori umani: ad esempio, l'UR10e può occuparsi di operazioni faticose o ripetitive (come il pick & place di componenti pesanti) mentre l'operatore svolge compiti a maggior valore aggiunto nella stessa area, con il robot che rallenta o si arresta automaticamente quando la persona si avvicina.

In conclusione, l'UR10e rappresenta oggi uno dei cobot più diffusi e studiati: la letteratura riporta applicazioni del settore industria 4.0 fino a impieghi in ambito sanitario (es. assistenza in laboratori, manipolazione di campioni) ed educativo. La sua flessibilità applicativa è uno dei motivi del successo: cambiando semplicemente l'end-effector (pinza, utensile, camera di visione, ecc.) e riprogrammando opportunamente la sequenza, lo stesso robot può diversificare i propri compiti. Questa adattabilità, unita alla facilità di integrazione in ambienti produttivi esistenti, rende l'UR10e una soluzione molto attraente sia per l'automazione delle PMI che per progetti di ricerca avanzata in robotica.

(a) (b)

Figura 1.4: (a) UR10e in operazione di bin picking [15], (b) UR10e impiegato in un'applicazione di verniciatura automatizzata [16]

1.2 Piattaforma mobile Robotnik RB-KAIROS+

1.2.1 Composizione meccanica e struttura

Il RB-KAIROS+ è una piattaforma robotica mobile autonoma (AMR – Autonomous Mobile Robot) progettata da Robotnik per integrare sulla propria base bracci robotici Universal Robots e-Series (UR5e, UR10e, UR16e), rendondolo a tutti gli effetti un manipolatore mobile. Dal punto di vista meccanico ha uno chassis in acciaio molto robusto e compatto [17] di forma rettangolare (circa 0,98 m x 0,78 m) e un baricentro basso per garantire stabilità, con una'altezza della piattaforma è di circa 70–100 cm da terra ed un peso complessivo di 115 kg [18]. Grazie alla sua struttura, RB-KAIROS+ può trasportare fino a 250 kg di payload sul piatto superiore, ad esempio merci, utensili o attrezzature quando usato senza manipolatore, oppure sostenere il manipolatore e gli oggetti da esso afferrati.

La locomozione della piattaforma avviene tramite quattro ruote Mecanum montate agli angoli dello chassis, dotate di rulli inclinati a 45° che consentono movimento omnidirezionale: RB-KAIROS+ è capace di traslare in qualsiasi direzione sul piano (avanti, indietro, lateralmente) e ruotare su sé stesso, con grande agilità anche in spazi ristretti. Ogni ruota è motorizzata in modo indipendente: la piattaforma dispone infatti di 4 motori elettrici da 500 W ciascuno, per una velocità massima di spostamento è circa 1,5 m/s (circa 5,4 km/ h), più che sufficiente per seguire i ritmi di un operatore a piedi nelle operazioni di intra-logistica. La piattaforma può affrontare rampe con pendenza fino al 15% e operare su terreni con irregolarità, tuttavia è destinata esclusivamente ad ambienti indoor dato il grado di protezione IP52 che la rende resistente alla polvere ma non adatta ad intensa esposizione all'acqua [19].

Oltre agli attuatori per il movimento, RB-KAIROS+ integra diversi sensori e componenti funzionali alla navigazione e all'uso come manipolatore mobile, come i laser-scanner perimetrali e telecamere 2D montata sul fronte dello chassis. La parte superiore della piattaforma presenta un piatto di montaggio predisposto per l'attacco del braccio UR: vi sono fori e interfacce compatibili con la base del manipolatore, permettendo un fissaggio rapido e preciso. Internamente, la piattaforma ospita un alloggio per l'elettronica di controllo (computer di bordo, motori, PLC di sicurezza) e per il pacco batteria, la quale offre circa 6 ore di autonomia operativa [19]. La ricarica avviene tramite un'apposita stazione su cui il robot può attraccare autonomamente quando la batteria è in esaurimento, oppure tramite cavo da attaccare manualmente [12].

1.2.2 Interfacce elettriche e di comunicazione

La piattaforma RB-KAIROS+ integra un'architettura elettrica ed elettronica complessa, che comprende sia il sistema di alimentazione a batteria, sia i moduli di controllo movimento, sicurezza e comunicazione. Il sistema di alimentazione è concentrato sulla batteria ricaricabile 48V 30Ah, la quale alimenta direttamente i motori e i principali componenti, fornenfo inoltre linee di alimentazione dedicate per l'eventuale manipolatore e-Serie installato [20]. Le interfacce di comunicazione disponibili su RB-KAIROS+ sono state concepite sia per la teleoperazione che per l'integrazione in reti industriali o laboratoriali, di serie infatti la piattaforma è dotata di connettività Wi-Fi e Bluetooth 4.0, per la comunicazione e controllo wireless, oltre le due porte ethernet poste sul retro [20]. Sono presenti inoltre tre porte USB utili per collegare periferiche, come joystick, dispositivi di storage o sensori aggiuntivi plug&play ed un'uscita HDMI per collegare un monitor.

1.2.3 Architettura software e ambienti di programmazione

Tutti i robot della marca Robotnik, incluso RB-KAIROS+, adottano un'architettura software basata su ROS (Robot Operating System), in particolare ROS 2 nelle versioni più recenti [19]. Ciò significa che la gestione ad alto livello del robot: navigazione, pianificazione del percorso, gestione sensori, è gestita tramite nodi e pacchetti ROS, beneficiando della grande flessibilità e modularità offerte da tale ecosistema. Sul PC a bordo di RB-KAIROS+ è installato il sistema operativo Ubuntu con middleware ROS 1 noetic, su cui vengono eseguiti diversi nodi: ad esempio un nodo di SLAM per costruire ed aggiornare la mappa dell'ambiente , un nodo di localizzazione che fonde dati di odometria con dati dei laser scanner per stimare la posa corrente, e un nodo di path planning per calcolare traiettorie evitando ostacoli [19]. L'adozione di ROS rende l'ambiente di programmazione di RB-KAIROS+ molto aperto a sviluppatori ed utenti esperti: è possibile creare nodi ROS personalizzati per estendere le funzionalità del robot, ad esempio integrando algoritmi di visione artificiale o logiche decisionali a più alto livello. Un aspetto rilevante è l'integrazione software con il manipolatore UR, come già accennato Robotnik ha sviluppato un URcap dedicato che permette di controllare la base mobile direttamente da Polyscope integrando così il piano di movimento della base con quello del braccio [12].

Installando questo URCap sul controller UR10e, l'utente può programmare dal Teach Pendant sequenze di movimento ibride, ad esempio: "muovi la base a una posizione specifica" seguito da "aziona il braccio per prelevare un pezzo" e così via, il tutto all'interno dello stesso ambiente. L'URCap fornisce nell'interfaccia del teach pendant alcuni comandi base per RB-KAIROS+, tra cui: spostamento a un punto predefinito, spostamento relativo, raggiungimento della stazione di ricarica.

1.2.4 Caratteristiche di sicurezza

RB-KAIROS+ è un robot mobile che opera in ambienti condivisi con persone e dispone di numerose misure di sicurezza per prevenire collisioni e situazioni di pericolo durante la navigazione. La piattaforma è equipaggiata con due laser-scanner di sicurezza 2D montati diagonalmente sul telaio per ottenere una copertura di rilevamento ostacoli a 360° intorno al robot [18]. Questi laser creano zone di sicurezza configurabili, normalmente una zona di rallentamento e una di arresto attorno al robot: se un ostacolo o una persona entra nella zona di preallarme, il robot riduce la velocità; se entra nella zona più critica, il PLC di sicurezza comanda un arresto immediato.

Sul lato frontale è installata una camera RGB-D (sensore di profondità) che fornisce una visione 3D dell'ambiente, utile per fare una mappa e calcolare percorsi alternativi in modo più intelligente. Lungo il perimetro del robot sono presenti pulsanti di arresto di emergenza a fungo, ben visibili e raggiungibili, in modo tale che chiunque si trovi nelle vicinanze può premerli in caso di necessità: tali pulsanti tolgono immediatamente energia ai motori tramite il circuito di sicurezza dedicato.

La sicurezza funzionale del sistema mobile è gestita da un Safety PLC separato, certificato secondo le normative macchine (ad es. ISO 13849), esso legge gli input dei laser di sicurezza e dei funghi di emergenza e può eseguire logiche di sicurezza come il ridimensionamento delle zone in base alla velocità attuale. In caso di manipolatore installato, il PLC può essere interfacciato anche con il circuito di sicurezza del braccio robotico (tramite contatti di E-stop in uscita dal control box UR) in modo che un'emergenza su uno dei due sistemi propaghi l'arresto anche all'altro, mantenendo sicuro il comportamento dell'intero sistema.

1.2.5 Ambiti applicativi in letteratura

I robot mobili come RB-KAIROS+ rappresentano uno stato dell'arte nel campo dell'automazione e trovano impiego in svariati scenari avanzati, andando ad estendere le normali e limitate zone di lavoro dei manipolatori. Ciò risulta ideale in applicazioni di logistica interna e asservimento di linee: il robot può prelevare materiale da un magazzino e portarlo a diverse stazioni, dove il braccio UR10e esegue operazioni di carico/scarico macchine o assemblaggio [21]. Altre applicazioni industriali possono essere espanse e migliorate grazie alla mobilità di questo sistema: controllo qualità mobile, ispezioni su prodotti di grande dimensioni, lucidatura di superfici estese [22].

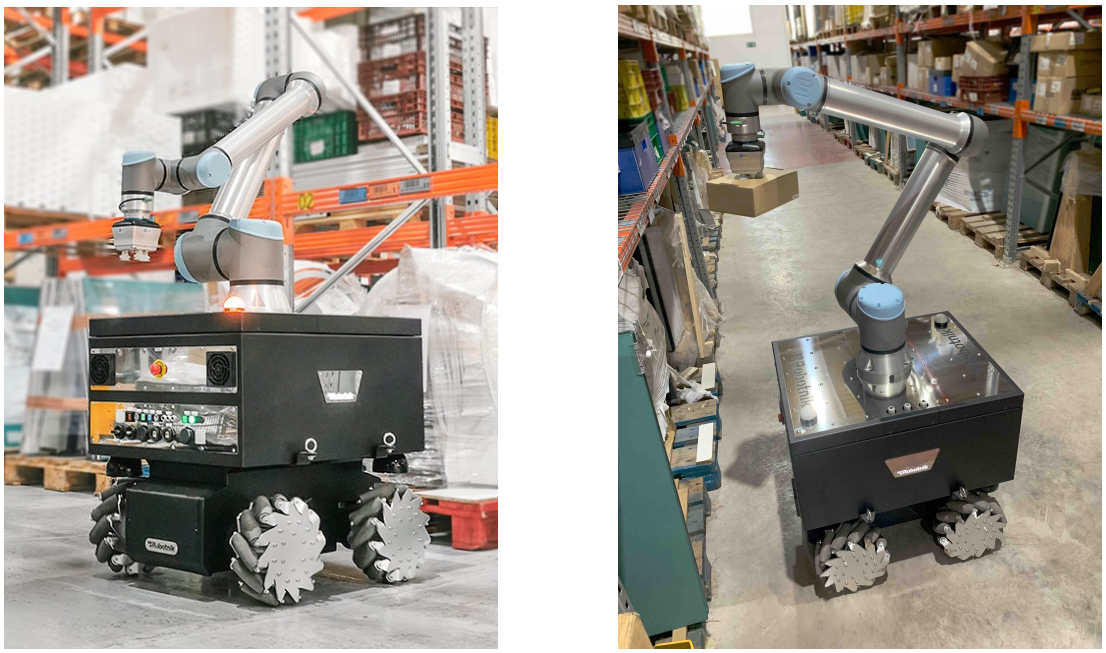


Figura 1.5: RB-KAIROS+ con UR10e in applicazioni di logistica e automazione industriale [23, 24]

Capitolo 2

Configurazione del sistema di navigazione

2.1 Concetti base sulla navigazione

L'obiettivo principale della navigazione autonoma è dare la possibilità ad un robot mobile di potersi spostarsi da un punto iniziale a uno di destinazione evitando gli ostacoli lungo il percorso. In generale, esistono due modalità di locomozione per un robot mobile su ruote:

- trazione differenziale
- trazione omnidirezionale

La differenza principale tra queste modalità risiede nel numero delle componenti di velocità che sono controllabili dal robot. Un robot mobile può infatti muoversi mediante due tipi di velocità: una velocità lineare e una velocità angolare.

- La velocità lineare consente al robot di traslare nello spazio, lungo la direzione di avanzamento
- La velocità angolare permette al robot di ruotare attorno a un asse, ovvero variare orientamento senza traslare

Nel caso di un robot a trazione differenziale, il vettore velocità del robot ammette solo la componente lineare lungo l'asse x (movimento avanti/indietro) e la componente angolare attorno all'asse z (rotazione a destra/sinistra). In un robot a trazione omnidirezionale invece, è possibile imporre anche una velocità lineare lungo l'asse y, abilitando così movimenti laterali e diagonali oltre a quelli longitudinali.

La maggior parte dei robot mobili in configurazione differenziale presenta vincoli non olonomi che impediscono movimenti puramente laterali; ciò significa che, pur potendo teoricamente raggiungere qualsiasi posizione nel piano, essi devono eseguire più movimenti complessi simili ad una manovra di parcheggio in automobile per variare la propria posizione laterale e orientazione. Viceversa, un robot omnidirezionale non è soggetto a tali vincoli: può traslare e ruotare liberamente sul posto, controllando indipendentemente le componenti di velocità lineare lungo x, y e rotazionale in z in modo tale da massimizzare la flessibilità di navigazione in spazi stretti o affollati.

2.2 Navigazione con mappa

Oltre alle differenti modalità di movimento, in ambito di navigazione autonoma, vi sono due strategie principali riguardo l'uso di informazioni sull'ambiente: la navigazione basata su mappa e la navigazione priva di mappa. Di seguito se ne delineano le caratteristiche:

- Navigazione con mappa: si dispone a priori di una mappa dell'ambiente, generalmente ottenuta tramite un processo di mappatura preliminare. La presenza di una mappa consente di localizzare tutti gli ostacoli statici rilevati durante la fase di mapping e di tenerne conto in fase di pianificazione del percorso da un punto di partenza verso un punto di destinazione.
- Navigazione senza mappa: in questo approccio il robot non dispone di alcuna informazione pregressa dell'ambiente circostante. La navigazione avviene utilizzando esclusivamente i dati sensoriali in tempo reale forniti localmente da laser scanner o telecamere.

Nel seguito ci si concentrerà sulla navigazione con mappa, descrivendo i passi necessari a costruire la mappa dell'ambiente e a localizzare il robot al suo interno utilizzando gli strumenti dello stack di navigazione ROS [25].

2.2.1 Creazione della mappa

Il primo requisito per abilitare la navigazione autonoma basata su mappa consiste nel fornire al robot una rappresentazione dell'ambiente operativo in cui è chiamato ad operare; a tal fine, uno degli approcci più comuni consiste nell'affrontare il cosiddetto problema di SLAM (Simultaneous Localization and Mapping), ovvero la costruzione simultanea di una mappa dell'ambiente e la localizzazione del robot al suo interno, considerato in letteratura come uno dei compiti fondamentali della robotica mobile [26]. All'interno di ROS lo stack di navigazione fornisce una comoda implementazione di Gmapping tramite il nodo `slam_gmapping` (pacchetto gmapping), il quale consente di creare una mappa bidimensionale dell'ambiente utilizzando i dati del laser e dell'odometria del robot. In altri termini, il nodo `slam_gmapping` implementa un algoritmo di SLAM laser che produce in modo incrementale una occupancy grid map (mappa di occupazione a celle) a partire dalle letture del sensore e dal moto del robot. Tramite ROS si a disposizione un'implementazione pronta all'uso, che richiede soltanto di configurare alcuni parametri opportuni per il proprio robot [27]. Utilizzando un file di launch è possibile avviare il nodo `slam_gmapping` con la configurazione opportuna, un esempio di launch file per Gmapping è riportato di seguito:

```
<launch>
    <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">

      <remap from="scan" to ="/robot/merged_laser/scan"/>
      <param name="base_frame" value="robot_base_footprint"/>
      <param name="odom_frame" value="robot_odom"/>
      <param name="map_udpate_interval" value="5.0"/>
      <param name="maxUrange" value="7.0"/>
      <param name="sigma" value="0.05"/>
      <param name="kernelSize" value="1"/>
      <param name="lstep" value="0.05"/>
      <param name="astep" value="0.05"/>
      <param name="iterations" value="5"/>
      <param name="lsigma" value="0.075"/>
      <param name="ogain" value="3.0"/>
      <param name="lskip" value="0"/>
      <param name="srr" value="0.1"/>
      <param name="srt" value="0.2"/>
      <param name="str" value="0.1"/>
      <param name="stt" value="0.2"/>
      <param name="linearUpdate" value="0.2"/>
      <param name="angularUpdate" value="0.1"/>
      <param name="temporalUpdate" value="3.0"/>
      <param name="resampleThreshold" value="0.5"/>
      <param name="particles" value="100"/>
      <param name="xmin" value="-50.0"/>
      <param name="ymin" value="-50.0"/>
      <param name="xmax" value="50.0"/>
      <param name="ymax" value="50.0"/>
      <param name="delta" value="0.05"/>
      <param name="llsamplerange" value="0.01"/>
      <param name="llsamplestep" value="0.01"/>
      <param name="lasamplerange" value="0.005"/>
      <param name="lasamplestep" value="0.005"/>
      <param name="throttle_scans" value="1"/>
    </node>
</launch>
```

Script 2.1: File start_mapping.launch

È importante configurare correttamente i principali topic a cui il nodo `slam_gmapping` sottoscrive, adattandoli ai topic effettivamente utilizzati dal robot reale [27]:

- **scan**: il topic su cui vengono pubblicati i dati del sensore laser del robot.
- **tf**: componente che tiene traccia e pubblica le trasformazioni spaziali tra i diversi sistemi di riferimento (frame) del robot. Ogni robot, sensore laser, telecamera ha un proprio sistema di coordinate; affinché tutte queste informazioni siano integrate coerentemente in un unico sistema globale è necessario conoscere esattamente come questi sistemi di coordinate sono collegati tra loro.

Tra i parametri configurabili nel file di launch sopra riportato, rivestono particolare importanza:

- **base_frame**: frame che rappresenta il centro del robot mobile
- **odom_frame**: sistema di riferimento dell'odometria del robot. Frame che tiene che calcola e tiene tracca della posizione relativa del robot rispetto al suo punto di partenza.
- **maxUrange**: indica la distanza massima letta dal sensore laser per la creazione della mappa. Un valore maggiore consente una più rapida creazione della mappa.

È possibile eseguire l'intero nodo di mappatura tramite il comando:

```
roslaunch rbkairos_navigation start_mapping.launch
```

Script 2.2: Comando per eseguire file start_mapping.launch

Per monitorare il processo di mappatura in tempo reale si utilizza RViz, strumento di visualizzazione 3D standard del framework ROS. Consente di ispezionare, tramite un ambiente grafico tridimensionale, lo stato interno del robot ed il suo ambiente facilitando il debug e lo sviluppo di algoritmi di navigazione e percezione [28]. Configurando opportunamente alcune viste, è possibile monitorare in tempo reale:

- **RobotModel**: È necessario impostare il parametro Robot Description sul valore robot/robot_description. Questo permette di visualizzare il modello 3D del robot all'interno di RViz.

- **LaserScan**: impostandolo su /robot/merged_laser/scan è possibile visualizzare in tempo reale i dati provenienti dal sensore laser del robot.
- **Map**: tramite impostazione /map è possibile visualizzare la mappa generata dal processo di SLAM.

Una volta che il nodo viene attivato tramite il comando, è possibile muovere il robot per generare gradualmente la mappa dell'ambiente. Il robot può essere teleoperato tramite joystick-controller oppure da tastiera tramite il nodo `teleop_twist_keyboard` usando il seguente comando:

```
rosrun teleop_twist_keyboard
 ↪  teleop_twist_keyboard.py
 ↪  cmd_vel:=/robot/robotnik_base_control/cmd_vel
```

Script 2.3: Controllo remoto del robot tramite tastiera

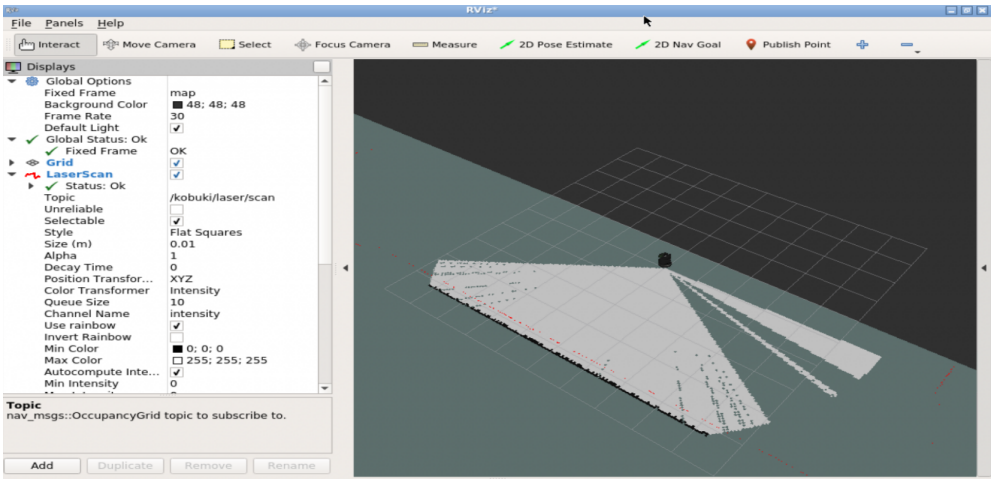


Figura 2.1: Visualizzazione in RViz del processo di mappatura in tempo reale [29].

Durante questa fase, il pacchetto ROS gmapping provvede ad eseguire internamente l'algoritmo di SLAM Gmapping, l'implementazione ROS fornisce quindi una soluzione pronta all'uso senza che l'utente debba programmare da zero alcun aspetto del SLAM. La procedura di mapping mostrata in Fig. 2.1 può essere riassunta come segue: un file di launch preconfigurato avvia il nodo `slam_gmapping`, il quale, durante lo spostamento del robot nell'ambiente, si sottoscrive ai dati provenienti dal sensore laser (topic */robot/merged_laser/scan*) e alle trasformazioni odometriche (topic */tf*), costruendo progressivamente una mappa dell'ambiente circostante.

La mappa generata viene pubblicata in continuo sul topic `/map`, consentendo di visualizzare in RViz l'evoluzione real-time del processo di mappatura. Il topic */map* utilizza il tipo di messaggio *nav_msgs/OccupancyGrid*, che rappresenta la mappa come una griglia di occupazione: in questa codifica il valore 0 indica celle libere, 100 indica celle occupate da ostacoli, mentre il valore -1 denota celle inesplorate e sconosciute [30][31].

2.2.2 Salvataggio della mappa

Un componente fondamentale dello stack di navigazione ROS è il pacchetto `map_server`, il quale consente di acquisire i dati della mappa tramite un servizio ROS e salvarli su file [30]; è possibile inoltre salvare in qualsiasi momento la mappa costruita utilizzando il comando seguente:

```
rosrun map_server map_saver -f nome_mappa
```

Script 2.4: Comando per salvare su file la mappa generata

Il comando consente di recuperare i dati attuali della mappa pubblicati sul topic */map* e di salvarli in due file: "nome_mappa.pgm", contenente la rappresentazione grafica della mappa, e "nome_mappa.yaml", che ne descrive i parametri di configurazione. Il file in formato PGM contiene i dati di occupazione della mappa come immagine in scala di grigi, in cui i pixel neri rappresentano gli ostacoli. Il file YAML contiene i metadati relativi alla mappa, come la risoluzione espressa in metri per cella, l'origine della mappa nel sistema di riferimento, e le soglie che determinano quanto una cella considerata libera o occupata. Questi due file possono essere ricaricati e riutilizzati qualora ci fosse bisogno di ripristinare la mappa in nuove sessioni di navigazione.

```
image: my_map.pgm
resolution: 0.050000
origin: [-10.000000, -10.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Script 2.5: Parametri di configurazione della mappa generata [24]

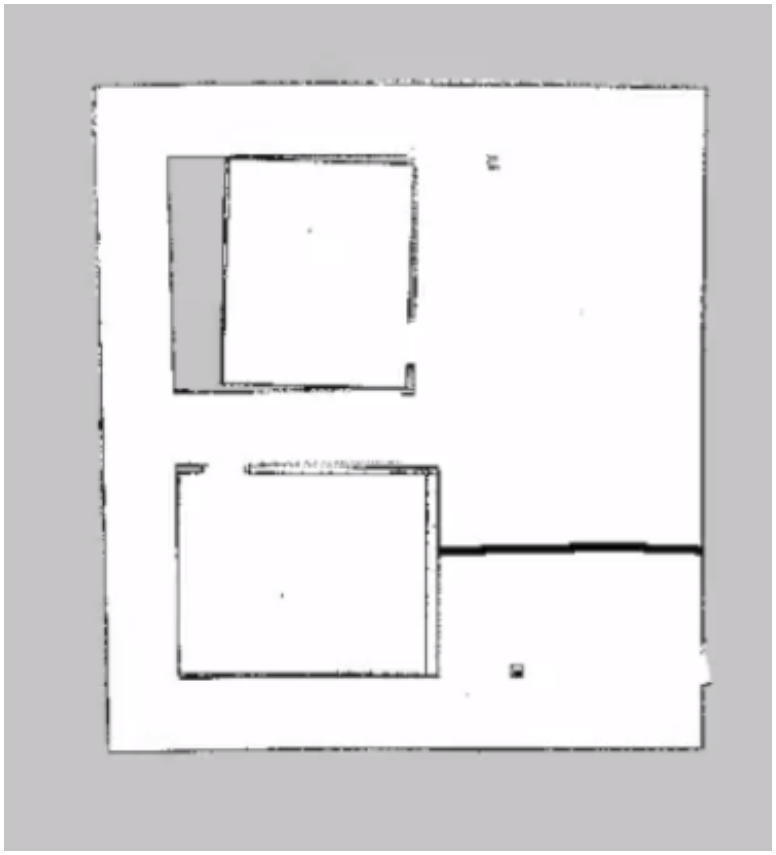


Figura 2.2: File .pgm con ricostruzione dall'alto [32]

2.3 Caricamento della mappa e localizzazione del robot

Al termine della fase di mapping, è necessario rendere la mappa disponibile ai moduli di navigazione per le fasi successive. A tal fine, il pacchetto *map_server* consente di avviare l'omonimo nodo, che rende la mappa accessibile a qualsiasi altro nodo ROS che ne faccia richiesta. In un sistema di navigazione tipico, diversi nodi richiedono la mappa corrente al `map_server`: ad esempio il nodo `move_base` utilizza la mappa per effettuare la pianificazione dei percorsi, mentre il nodo di localizzazione la utilizza per confrontare le osservazioni dei sensori con l'ambiente noto e determinare la posa del robot. Il servizio esposto ed offerto dal nodo `map_server` per ottenere la mappa è *static_map (nav_msgs/GetMap)*, un'interfaccia che consente a qualsiasi componente ROS di richiedere una copia della mappa attualmente disponibile nel map_server, utilizzando un meccanismo basato su servizio anziché sottoscrizione a topic [30][33].

Per avviare il nodo `map_server` con una mappa precedentemente salvata, si può utilizzare il seguente comando:

```
rosrun map_server map_server nome_mappa.yaml
```

Script 2.6: Avvio del nodo map_server con la mappa desiderata

Sebbene sia possibile avviare manualmente il nodo `map_server`, è prassi più comune includerne l'esecuzione all'interno di un file di launch complessivo, in modo da automatizzare la fornitura della mappa ai moduli di navigazione; a titolo esemplificativo, si può definire un file di launch denominato *map_provider.launch*, che avvia il nodo `map_server` specificando il file della mappa da caricare e impostando il frame di riferimento appropriato, come illustrato di seguito:

```
<launch>
  <arg name="map_file" default="$(find rbkairos_navigation)/maps/my_map.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)">
    <param name="frame_id" value="robot_map"/>
  </node>
</launch>
```

Script 2.7: File map_provider.launch per esecuzione del nodo map_server

Una volta creata la mappa e caricata tramite il nodo `map_server`, il robot non è ancora in grado di conoscere la propria posizione all'interno di essa; è infatti necessaria una fase di localizzazione, il cui scopo è quello di allineare il frame di riferimento del robot alla mappa, stimandone la posizione rispetto all'ambiente circostante, così da permettere un utilizzo corretto della mappa nei successivi processi di navigazione. Per ottenere la localizzazione eposizione del robot, si utilizza il nodo AMCL (Adaptive Monte Carlo Localization) fornito dallo stack di navigazione di ROS. Analogamente a quanto fatto per la mappatura, si predispone un file di tipo launch chiamato *start_localization.launch* per l'avvio del nodo:

```
<launch>
   <arg name="map_file" default="$(find rbkairos_navigation)/maps/my_map.yaml"/>

   <!-- Run the map server -->
   <include file="$(find rbkairos_navigation)/launch/map_provider.launch">
        <arg name="map_file" default="$(arg map_file)"/>
   </include>

   <!--Run Amcl node -->
   <node pkg="amcl" type="amcl" name="amcl" output="screen">
        <remap from ="scan" to="/robot/merged_laser/scan"/>
        <remap from ="cmd_vel" to="/robot/robotnik_base_control/cmd_vel"/>
        <remap from ="amcl_pose" to="/robot/amcl_pose"/>

        <param name="global_frame_id" value="robot_map"/>
        <param name="odom_model_type" value="omni"/>
        <param name="odom_alpha5" value="0.1"/>
        <param name="transform_tolerance" value="0.2"/>
        <param name="gui_publish_rate" value="10.0"/>
        <param name="laser_max_beams" value="30"/>
        <param name="min_particles" value="500"/>
        <param name="max_particles" value="5000"/>
        <param name="kld_err" value="0.05"/>
        <param name="kld_z" value="0.99"/>
        <param name="odom_alpha1" value="0.2"/>
        <param name="odom_alpha2" value="0.2"/>
        <!-- translation std dev, m -->
        <param name="odom_alpha3" value="0.8"/>
        <param name="odom_alpha4" value="0.2"/>
        <param name="laser_z_hit" value="0.5"/>
        <param name="laser_z_short" value="0.05"/>
        <param name="laser_z_max" value="0.05"/>
        <param name="laser_z_rand" value="0.5"/>
        <param name="laser_sigma_hit" value="0.2"/>
        <param name="laser_lambda_short" value="0.1"/>
        <param name="laser_lambda_short" value="0.1"/>
```

```
        <param name="laser_model_type" value="likelihood_field"/>
        <!-- <param name="laser_model_type" value="beam"/> -->
        <param name="laser_likelihood_max_dist" value="2.0"/>
        <param name="update_min_d" value="0.2"/>
        <param name="update_min_a" value="0.5"/>
        <param name="odom_frame_id" value="robot_odom"/>
        <param name="base_frame_id" value="robot_base_link"/>
        <param name="resample_interval" value="1"/>
        <param name="recovery_alpha_slow" value="0.0"/>
        <param name="recovery_alpha_fast" value="0.0"/>
        <param name="initial_pose_x" value ="0.0"/>
        <param name="initial_pose_y" value ="0.0"/>
        <param name="initial_pose_a" value ="0.0"/>
    </node>
</launch>
```

Script 2.8: File start_localization.launch per l'attivazione del nodo AMCL

All'interno del file di launch dedicato alla localizzazione è incluso anche l'avvio del nodo `map_server` come definito nello script 2.7, in modo da garantire che il nodo AMCL abbia accesso alla mappa statica necessaria per calcolare la posa del robot [34]. Alcuni importanti topic da configurare nel file di launch sopra riportato sono:

- **scan**: topic su cui vengono pubblicati i dati rielvati dai laser-scanner(LIDAR)
- **cmd_vel**: topic del robot su cui vengono inviati i comandi di velocità
- **amcl_pose**: topic in cui il nodo AMCL di localizzazione pubblica la posizione stimata del robot

Tra i parametri più importanti da impostare nel file `start_localization.launch` vi sono:

- **global_frame_id**: nome del frame globale della mappa
- **min_particles, max_particles**: questi parametri definiscono il numero minimo e massimo di particelle utilizzate dal filtro Monte Carlo per la localizzazione del robot. Un numero maggiore aumenta la precisione della localizzazione, a discapito però del consumo di risorse computazionali.

- **laser_max_range**: distanza massima che il sensore laser può misurare (in metri)
- **odom_model_type**: definisce il tipo di locomozione che verrà utilizzata dal robot. Nel caso del RB-Kairos ,essendo un robot omnidirezionale come descritto prima, è importante definirlo su `omni` in modo che l'odometria venga calcolata correttamente. Per un robot a locomozione differenziale questo parametro dovrà essere impostato su "diff"

Per avviare il nodo di localizzazione AMCL definito precedentemente 2.8 è sufficiente eseguire il seguente comando:

```
roslaunch rbkairos_navigation start_localization.launch
```

Script 2.9: Avvio del nodo di localizzazione AMCL

Una volta avviato il nodo di localizzazione, è possibile visualizzare il processo in tempo reale tramite RViz, infatti è sufficiente ricaricare la stessa configurazione precedentemente utilizzata nella fase di mappatura integrandola con un ulteriore elemento: *Pose Array*. Questo elemento dovrà essere impostato per visualizzare il topic */particlecloud*, che mostra la distribuzione delle particelle utilizzate dal filtro di localizzazione Monte Carlo (AMCL), consentendo la visualizzazione grafica delle particelle verso la posizione stimata del robot.

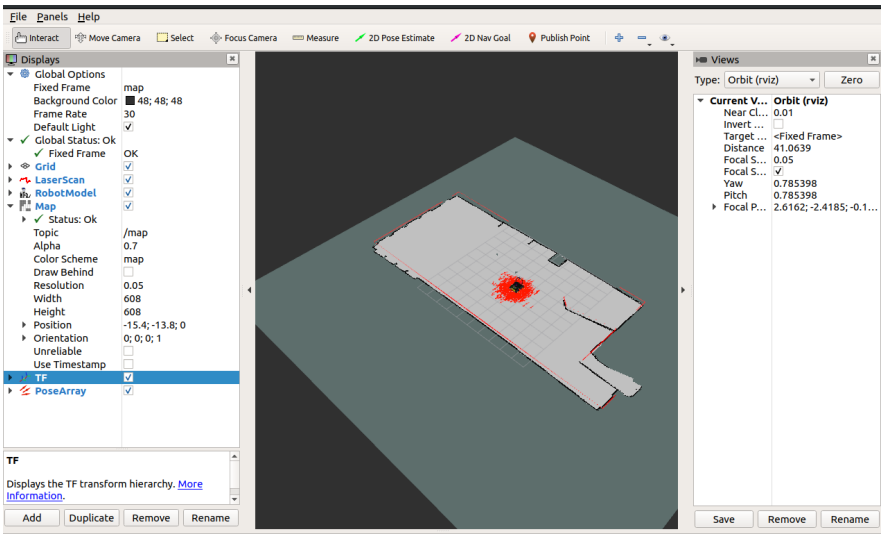


Figura 2.3: Visualizzazione delle particelle AMCL in RViz durante il processo di localizzazione del robot [35]

Una descrizione completa di tutti i parametri configurabili per AMCL è disponibile nella documentazione ROS ufficiale [34]. Questo sitema di localizzazione probabilistico, implementa un filtro di localizzazione basato sull'algoritmo MCL (Monte Carlo Localization, ovvero localizzazione mediante filtro a particelle).
Il termine "Monte Carlo" si riferisce all'utilizzo di un campionamento casuale per rappresentare la distribuzione di probabilità della posa del robot. Durante la fase di inizializzazione, le particelle, ciascuna delle quali rappresenta un'ipotesi di posizione e orientamento del robot, vengono generalmente distribuite in modo uniforme nello spazio, a indicare l'assenza di informazioni iniziali sulla posizione. Con il progredire del movimento del robot e l'acquisizione di dati dall'ambiente, l'algoritmo confronta le letture sensoriali con la mappa, eliminando le particelle incompatibili con le nuove osservazioni e generando nuove particelle attorno a quelle maggiormente coerenti con i dati rilevati. In questo modo, iterativamente, la "nuvola" di particelle converge progressivamente verso la posizione reale del robot. In Figura 2.3 è mostrata la visualizzazione in RViz delle particelle (frecce), inizialmente disperse, esse tenderanno a concentrarsi attorno alla posizione corretta man mano che il robot esplora ed effettua osservazioni.

Una caratteristica fondamentale di AMCL in ROS è l'approccio adattativo al numero di particelle basato su KLD-sampling. Questo approccio, utilizza la divergenza di Kullback-Leibler (KLD) per regolare dinamicamente la dimensione dell'insieme di particelle in funzione dell'incertezza della stima di posa [36]. Quando l'incertezza sulla posizione del robot è elevata, come nel caso della localizzazione globale in cui il robot non conosce ancora la propria posizione sulla mappa, AMCL utilizza un numero elevato di particelle per esplorare l'intero spazio delle possibili pose. Viceversa, quando il robot converge verso una stima più precisa e affidabile della propria posizione, il filtro riduce automaticamente il numero di particelle, mantenendo solo quelle necessarie a seguire i movimenti del robot. In questo modo, l'algoritmo AMCL utilizza tutte le particelle disponibili nelle fasi iniziali della localizzazione e le riduce drasticamente una volta che la posa è determinata con buona confidenza. Questo comportamento garantisce efficienza computazionale, poiché evita di mantenere sempre un numero massimo di campioni. Modificando i parametri *min_particles* e *max_particles*: ad esempio, un numero maggiore di particelle comporterà una nuvola iniziale più densa e una maggiore robustezza a ipotesi errate, a scapito di un maggior carico computazionale.

L'algoritmo di localizzazione descritto è noto pertanto come Monte Carlo Localization (MCL o localizzazione tramite filtro a particelle. Il nodo AMCL implementa questo algoritmo in modalità adattativa, ovvero regolando dinamicamente il numero di particelle in funzione del livello di confidenza sulla posa stimata. Quando l'incertezza sulla posizione è elevata, come nella fase iniziale o dopo grandi spostamenti, AMCL utilizza un insieme di particelle più numeroso; mano a mano che il robot converge verso una soluzione di localizzazione affidabile, il nodo riduce automaticamente il numero di particelle, risparmiando risorse computazionali. Il nodo AMCL si sottoscrive ai dati del laser, alla mappa statica ed alle trasformazioni *tf* del robot, pubblicando in uscita la stima filtrata della posa del robot sul topic */amcl_pose*. All'avvio, il filtro di particelle viene inizializzato secondo i parametri specificati (ad esempio le coordinate iniziali approssimative tramite *initial_pose_x*, *initial_pose_y*, *initial_pose_a*). Una volta avviato AMCL tramite il comando 2.9, il sistema di navigazione disporrà di tutti i componenti fondamentali: la mappa dell'ambiente, la localizzazione corrente del robot nella mappa e il modello del robot.

Capitolo 3

Configurazione del manipolatore

3.1 MoveIt

Per la pianificazione e l'esecuzione di movimenti del braccio in modo autonomo e sicuro, è stata utilizzata la libreria MoveIt, uno dei framework più diffusi in ROS per il motion planning di manipolatori [37]. La libreria MoveIt è un framework open source per ROS [38], fornisce un'architettura modulare che comprende componenti per la risoluzione della cinematica inversa tramite plugin dedicati (es. KDL – Kinematics and Dynamics Library), la pianificazione del movimento (supportando molteplici algoritmi, con OMPL – Open Motion Planning Library come soluzione predefinita), il controllo delle collisioni e la rappresentazione dell'ambiente tramite la Planning Scene [38].

MoveIt si integra strettamente con ROS mediante il nodo centrale `move_group`, che carica la descrizione del robot (URDF/SRDF) e i relativi parametri di configurazione, offrendo servizi e azioni per la pianificazione ed esecuzione delle traiettorie [39]. Il sistema presenta un'architettura a plugin flessibile che lo rende indipendente dall'hardware e ne consente il rapido adattamento a piattaforme diverse, un esempio è il Motion Planning plugin per l'interazione e la visualizzazione in tempo reale della pianificazione [39][38].

3.2 Generazione del pacchetto MoveIt di configurazione del manipolatore

Il controllo e l'utilizzo del robot UR10e tramite MoveIt sono stati realizzati utilizzando il MoveIt Setup Assistant, uno strumento GUI fornito dal framework MoveIt che guida l'utente nella creazione di un package di configurazione specifico per il robot. Questo package conterrà tutti i file necessari affinché MoveIt possa interfacciarsi correttamente con il manipolatore e l'ambiente; di seguito si illustrano i passi eseguiti tramite il MoveIt Setup Assistant:

1. **Caricamento del modello URDF del robot**: Il primo passo consiste nel fornire a MoveIt la descrizione del robot in formato URDF (Unified Robot Description Format), uno standard basato su XML per rappresentare la struttura di un robot (link rigidi, giunti, dimensioni, masse, geometrie di collisione e visibilità) in modo unificato [40]. ROS utilizza un file in formato URDF come descrizione primaria del robot: esso definisce la catena cinematica del manipolatore, ovvero la sequenza gerarchica di collegamenti e giunti dall'elemento base (link 1) fino all'organo terminale (end effector) [41]. Per il sistema utilizzato, il file URDF (o XACRO, una macro-versione dell'URDF) dovrà includere sia la base omnidirezionale RB-KAIROS+ sia il braccio robot UR10e montato su di essa [42]. Una volta selezionato il file URDF, il Setup Assistant analizza la struttura cinematica del robot e carica i parametri relativi a collegamenti e giunti, passaggio è fondamentale poiché tutte le configurazioni ed impostazioni successive si basano sulle informazioni di struttura fornite dall'URDF.

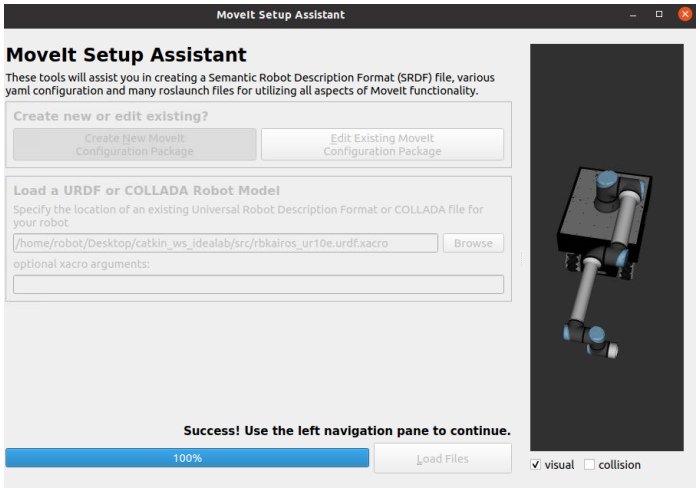


Figura 3.1: Interfaccia grafica del MoveIt Setup Assistant durante il caricamento del modello URDF del robot

2. **Configurazione della matrice di auto-collisione**: Dopo il caricamento del file URDF, il Setup Assistant permette di configurare la Self-Collision Matrix, ovvero la matrice delle coppie di link per cui ignorare il controllo di collisione. MoveIt genera automaticamente una lista di coppie di collegamenti del manipolatore, che possono essere escluse dai controlli di collisione poiché sempre in contatto o cinematicamente adiacenti. Ad esempio, segmenti consecutivi di un robot (come un link e il successivo collegato dallo stesso giunto) risultano sempre in contatto o molto vicini tra loro durante il movimento del braccio, e quindi le collisioni reciproche possono essere ignorate senza correre rischi [43]. Questa ottimizzazione riduce il carico computazionale durante la pianificazione, evitando preventivamente numerosi controlli ridondanti su collisioni interne. Nel Setup Assistant, queste coppie vengono pre-calcolate mostrate all'utente, che può eventualmente modificarle o accettarle. Durante quesot lavoro di tesi, si è mantenuta la configurazione di default generata automaticamente dal file URDF che coinvolge sia UR10e che RB-Kairos+, che esclude dalle verifiche di collisione le coppie di link adiacenti come il collegamento spalla-braccio e braccio-avambraccio.

3. **Definizione dei gruppi di pianificazione (Planning Groups)**: MoveIt organizza i giunti del robot in uno o più planning group, ovvero gruppi di giunti che verranno controllati e pianificati simultaneamente. In pratica, un planning group corrisponde a un manipolatore o sottosistema del robot che si vuole possa essere mosso indipendentemente dagli altri; nella configurazione utilizzata è stato definito un solo gruppo principale: - Gruppo "arm" (braccio): include tutti i giunti del manipolatore UR10e dalla base del braccio fino al polso/flangia a cui è collegato un eventuale end-effector [44]. Questo gruppo rappresenta la catena cinematica principale del braccio robotico ed è quello su cui verranno effettuati i calcoli di cinematica inversa e le pianificazioni di traiettoria per posizionare l'end effector nello spazio operativo, ovvero lo spazio tridimensionale in cui vengono espresse le posizioni ed orientamenti dell'end-effector. Per il gruppo arm si è scelto il solver cinematico basato sulla libreria KDL (Kinematics and Dynamics Library), in particolare il plugin di MoveIt KDLKinematics, che fornisce algoritmi di cinematica inversa generici per manipolatori [44]. Si noti che un planning group può anche essere composto da un singolo giunto o da sotto-insiemi dei giunti totali, qualora si voglia, ad esempio, pianificare solo il movimento di una parte del robot. Definire opportunamente i planning

groups è importante pichè MoveIt utilizza questi gruppi per sapere quali giunti deve controllare durante una richiesta di pianificazione del movimento. Ad esempio, quando si comanda uno spostamento del braccio verso una certa posizione, MoveIt considererà solo i giunti del planning group indicato, lasciandone fermi altri eventuali non inclusi, come la base mobile o la pinza se non esplicitamente coinvolte. Nella configurazione in esame, la base mobile non è stata inserita tra i planning group durante la generazione del pacchetto MoveIt del manipolatore, assumendo quindi il braccio montato in posizione fissa di lavoro. Qualora si volesse pianificare movimenti combinati base e braccio, si potrebbe definire un ulteriore planning group comprendente sia i giunti di locomozione della base che quelli del braccio; andando a creare un coordinaemento di manipolazione mobile più complesso.

4. **Definizione di pose predefinite del braccio**: MoveIt permette di definire delle pose predefinite per uno qualsiasi dei planning group configurati al passaggio precedente. Una posa predefinita è essenzialmente una configurazione specifica di tutti i giunti di un gruppo, salvata con un nome identificativo, allo scopo di poterla richiamare durante le operazioni di pianificazione ed esecuzione [45]. In particolare, si è definita una pose denominata "start" (iniziale) in cui il braccio è in posizione di riposo raccolta con il link terminale (flangia) rivolto verso il basso, e un'altra posa "home" utile per trasporto ed eventuale installazione del robot in altre sedi. Per ciascuna pose, il Setup Assistant ha richiesto di specificare i valori corrispondenti per tutti i giunti del gruppo arm; le pose predefinite rappresentano uno strumento pratico: durante l'uso dell'interfaccia grafica RViz, è possibile comandare il robot direttamente verso una pose salvata senza dover specificare manualmente tutti i valori articolari ogni volta.

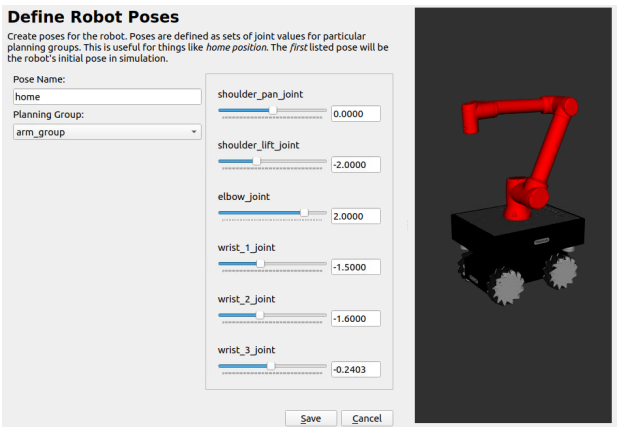


Figura 3.2: Configurazione di pose predefinite in MoveIt

5. **Configurazione dei controller ROS per il braccio**: Una volta definiti robot, gruppi e pose, occorre configurare come MoveIt invierà i comandi di movimento al robot (simulato o reale). MoveIt genera traiettorie, ovvero sequenze di valori di giunti nel tempo, delegando l'esecuzione di esse a specifici controller ROS che interagiscono con l'hardware. Nel caso del robot UR10e in esame, viene utilizzato il controller standard FollowJointTrajectory (parte del framework ros_control di ROS) per il controllo del braccio [46][47]. Il Setup Assistant, nella sezione "ROS Control", offre un pulsante "Auto Add Controllers" che popola automaticamente una configurazione iniziale di un controller chiamato arm_controller di tipo FollowJointTrajectory. Questo corrisponde al controllore che gestisce i 6 giunti dell'UR10e, già in esecuzione nel sistema, ad esempio tramite il driver ROS del UR10e [48]. I parametri principali, modificabili dall'utente, sono il nome del controller (arm_controller), il tipo (plugin di tipo FollowJointTrajectory) e la lista dei giunti controllati (6 giunti del braccio). Il file di configurazione generato verrà salvato come *ros_controllers.yaml* nel pacchetto MoveIt, questi dati vengono salvati sotto la sezione *controller_list*.

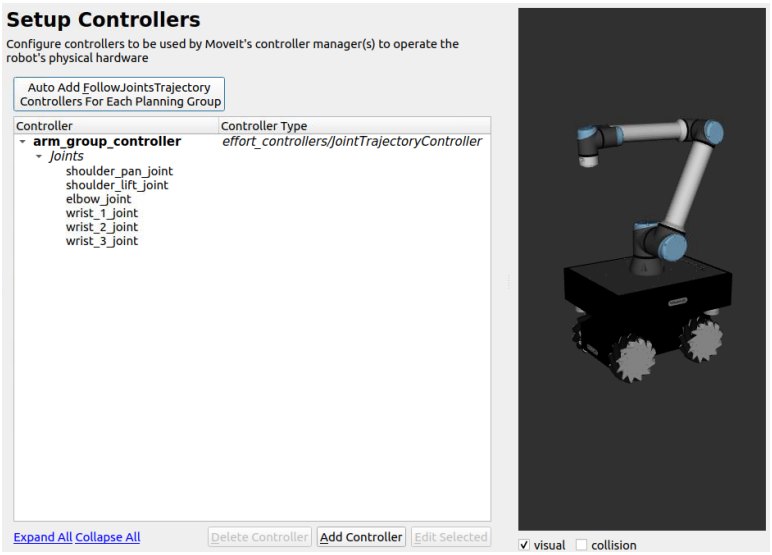


Figura 3.3: Configurazione automatica dei controller nel MoveIt Setup Assistant

6. **Salvataggio e generazione del pacchetto di configurazione**:

   Completati tutti i passi sopra descritti, si procede a generare il pacchetto MoveIt di configurazione.
   Nella scheda finale "Configuration Files" del Setup Assistant si sceglie la directory di destinazione, ad esempio *rbkairos_ur10e_moveit_config* all'interno della cartella di lavoro *catkin_ws/src*). Confermando la creazione, lo strumento genera automaticamente vari file:

   - Il file SRDF (Semantic Robot Description Format), contenente informazioni supplementari rispetto all'URDF di partenza, come i planning groups definiti, le pose predefinite e la matrice di collisione personalizzata [49].
   - File YAML di configurazione: *kinematics.yaml*, specifica il solver di cinematica (KDL) per ciascun planning group [50]. *joint_limits.yaml*, per limitare posizione/velocità/accelerazione per ciascun giunto, dati letti ed importati dal file URDF stesso [47][51]. *ros_controllers.yaml*, la lista dei controller configurati. *ompl_planning.yaml*, per impostare i parametri dei planner OMPL, ad esempio tipologia di planner e tempo massimo di pianificazione[52].
   - File di launch ROS preconfigurati: in genere viene creato un file di launch (es. demo.launch o move_group.launch) che avvia il nodo principale di MoveIt (move_group) con i parametri appena configurati, e un file di launch per RViz con il plugin di MotionPlanning caricato per visualizzare e interagire col robot

Una volta che il pacchetto è stato generato, MoveIt è pronto per eseguire pianificazioni di moto. Conosce la descrizione geometrica e cinematica del robot UR10e, conosce i limiti dei giunti del manipolatore, sa come calcolarne la cinematica inversa, come evitare collisioni, come inviare i comandi al controller per muoverne i giunti.

3.3 Il nodo move_group e architettura di MoveIt

Il core del sistema di manipolazione è il nodo ROS denominato `move_group`, esso può essere considerato il cuore di MoveIt in esecuzione poiché carica tutti i plugin e le configurazioni create offrendo interfacce per la pianificazione ed esecuzione dei movimenti. In fase di run-time, il nodo `move_group` aggrega le informazioni relative al robot ed allo stesso tempo offre servizi e azioni per la pianificazione [53].

Una volta messo in esecuzione, il nodo move_group carica dal Server ROS la descrizione URDF del robot e il file SRDF corrispondente, oltre ai file di configurazione YAML, andando a leggere la struttura cinematica (URDF) e semantica (SRDF, gruppi, pose, collisioni disattivate) del manipolatore. Inoltre, il nodo si sottoscrive a diversi topic per ottenere lo stato attuale del robot: tipicamente il topic *joint_states* che fornisce le misure correnti di posizione articolare [53] e il topic *TF* della configurazione per conoscere la posa relativa dei vari link.

Il nodo move_group espone una serie di interfacce attraverso cui gli utenti o altri nodi possono richiedere operazioni di pianificazione del movimento.
Ad esempio, l'azione MoveGroup (/move_group action) è l'interfaccia principale per chiedere al sistema di pianificare ed eventualmente eseguire un movimento del robot verso una posa obiettivo. È possibile invocarla sia tramite codice (API C++ o Python di MoveIt) sia tramite interfacce grafiche come RViz. Altre interfacce includono servizi per il calcolo di cinematica inversa (/compute_ik), per la verifica di collisioni, per l'aggiunta o rimozione di oggetti nel planning scene, ecc [54]. Il nodo esegue il ciclo di pianificazione ogni qualvolta riceve una richiesta: ad esempio, dati uno start state (stato attuale dei giunti) e un goal (posizione/orientamento desiderati dell'end-effector o valori target per i giunti), `move_group` richiama internamente un motion planner configurato nel pacchetto, per calcolare una traiettoria valida che soddisfi l'obiettivo evitando collisioni (di default uno dei planner del OMPL - Open Motion Planning Library [55][52].

Una volta calcolata la traiettoria, MoveIt può inviarla al sistema di controllo del robot per la sua esecuzione reale, utilizzando i dati in ros_controllers.yaml per sapere a quali action server inoltrarla. In concreto, invia la traiettoria pianificata all'action FollowJointTrajectory offerto dal controller del braccio, quindi ne monitora l'esecuzione. Durante essa, MoveIt aggiorna lo stato corrente dei giunti e verifica eventuali deviazioni o condizioni di errore.

Se durante il movimento intervenisse un cambiamento nell'ambiente (ad es. un nuovo ostacolo percepito), MoveIt potrebbe opzionalmente ricalcolare la traiettoria o fermare il robot grazie alla funzionalità di collision checking continuo. Il nodo `move_group` può essere controllato inoltre tramite l'interfaccia grafica di RViz grazie al plugin di Motion Planning. Questo plugin si connette all'action MoveGroup e ad altri servizi del nodo, consentendo così ad un operatore umano di impostare visivamente posizioni target da raggiungere, ad esempio manipolando virtualmente end-effector del robot [56]. Per quanto riguarda eventuali messaggi di log, MoveIt fornisce informazioni quali: tempi di calcolo, successo/fallimento del planner, eventuali collisioni riscontrate, tramite log su terminale e su RViz. Questi feedback sono utili durante l'utilizzo del sistema per impostare parametri come il tempo massimo di pianificazione e la risoluzione di eventuali collisioni per assicurarsi che il comportamento del robot sia quello atteso, obiettivo fondamentale quando si parla di simulazione.

In definitiva, il nodo `move_group` svolge il ruolo di coordinatore tra percezione, pianificazione e controllo, raccogliendo lo stato del robot e dell'ambiente, calcolando traiettorie valide mediante i plugin di pianificazione e affidando la loro esecuzione ai relativi controller [46]. Grazie alla modularità di MoveIt, il nodo può essere controllato sia tramite comandi di alto livello per automatizzare sequenze di movimento (es. operazioni di pick & place), sia manualmente tramite interfacce come RViz per scopi di test e calibrazione.

3.4 Simulazione ed integrazione con il robot reale UR10e

La configurazione del sistema di manipolazione tramite MoveIt ha previsto due fasi principali: una prima fase di verifica in ambiente simulato e una successiva fase di integrazione e test con il robot reale UR10e. Questa metodologia è stata adottata per validare la correttezza della configurazione e ridurre il rischio di comportamenti imprevisti durante l'operatività effettiva del robot reale.

3.4.1 Fase di Simulazione

La fase di simulazione è stata condotta utilizzando RViz ed il plugin Motion Planning integrato. Il braccio è stato comandato per muoversi verso pose conosciute e salvate precedentemente, ad esempio la pose "start" e la posa "home"; selezionando dall'interfaccia di RViz la posa desiderata come obiettivo, MoveIt calcolava una traiettoria priva collisioni partendo dallo stato attuale dei giunti fino alla configurazione target. La traiettoria pianificata veniva mostrata in anteprima nell'interfaccia grafica di RViz, grazie al movimento simulato del modello 3D robot. Una volta pianificata la traiettoria è possibile eseguirla tramite il pulsante "Execute": il modello del braccio in RViz segue così i movimenti articolari pianificati, confermando che il controller virtuale è in grado di seguire la traiettoria senza errori. Il motion planner è stato testato anche su pose casuali valide generate dal plugin stesso, poiché MoveIt è in grado di generare pose target aleatorie entro i limiti articolari e di calcolare le traiettorie necessarie per raggiungerle. Al termine di questa fase, è stato verificato che il pacchetto MoveIt configurato fosse in grado di pianificare correttamente i movimenti del robot interessato, rispettando i limiti articolari e prevenendo collisioni tra i link stessi del robot. L'ambiente di simulazione RViz è stato utilizzato anche per testare e modificare alcuni parametri presenti nel file joints_limit.yaml creatosi durante la generazione del pacchetto, da qui infatti è possibile personalizzare vincoli articolari (posizione, velocità, accelerazione) di ciascun giunto del robot.

3.4.2 Implementazione e Utilizzo del Robot Reale UR10e

La fase conclusiva ha riguardato l'utilizzo del package MoveIt per il controllo del robot reale, al fine di verificare la corretta trasposizione sul sistema fisico delle funzionalità già testate in simulazione. Per fare ciò, sono state implementate e validate una serie di modifiche e accorgimenti tecnici necessari a garantire l'integrazione.

Nel contesto reale, poiché il controller del robot UR10e in esecuzione utilizza un namespace differente rispetto a quello generato dal MoveIt Setup Assistant, è stato necessario modificare il file *ros_controllers.yaml* del pacchetto MoveIt, sostituendo il nome del controller da "arm_controller" a "scaled_pos_joint_traj_controller", corrispondente al nome reale del controller UR10e di tipo FollowJointTrajectory. Questa modifica ha permesso al nodo move_group di connettersi correttamente agli action server dei controller del robot reale e di inviare le traiettorie dall'ambiente simulato RViz all'hardware effettivo.

Qui di seguito è riportato lo Script 3.1 di configurazione ros_controller.yaml, utilizzato per definire i parametri di controllo del robot reale

```
# Simulation settings for using moveit_sim_controllers
moveit_sim_hw_interface:
  joint_model_group: arm
  joint_model_group_pose: home
# Settings for ros_control_boilerplate control loop
generic_hw_control_loop:
  loop_hz: 300
  cycle_time_error_threshold: 0.01
# Settings for ros_control hardware interface
hardware_interface:
  joints:
    - shoulder_pan_joint
    - shoulder_lift_joint
    - elbow_joint
    - wrist_1_joint
    - wrist_2_joint
    - wrist_3_joint
  sim_control_mode: 1  # 0: position, 1: velocity
# Publish all joint states
# Creates the /joint_states topic necessary in ROS
joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 50
controller_list:
  - name: "scaled_pos_joint_traj_controller"
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    joints:
      - shoulder_pan_joint
      - shoulder_lift_joint
      - elbow_joint
      - wrist_1_joint
      - wrist_2_joint
      - wrist_3_joint

  - name: "joint_group_vel_controller"
    type: velocity_controllers/JointGroupVelocityController
    joints:
      - shoulder_pan_joint
      - shoulder_lift_joint
      - elbow_joint
      - wrist_1_joint
      - wrist_2_joint
      - wrist_3_joint
```

Script 3.1: File ros_controller.yaml utilizzato per il controllo del robot reale

È stato fondamentale verificare che il topic */joint_states* pubblicasse e comunicasse correttamente i dati con i driver ROS UR10e messi a disposizione da Universal Robot, controllando inoltre che i nomi dei giunti fossero consistenti con quelli definiti nei file URDF/SRDF in MoveIt [57]. La mancata corrispondenza avrebbe potuto causare un aggiornamento errato dello stato dei giunti del robot e, di conseguenza, pianificazioni di traiettorie errate. Dopo aver effettuato le verifiche e le modifiche necessarie, è stato eseguito un primo test di pianificazione ed esecuzione con il robot reale, impostando velocità e accelerazione ridotte; comandando una traiettoria verso la posa 'start' tramite RViz, il robot ha iniziato a muoversi seguendo con precisione la traiettoria pianificata in simulazione, fino a raggiungere la posizione finale. Questo ha confermato l'accuratezza della configurazione cinematica e di controllo e la fedeltà del modello MoveIt al comportamento reale del robot. Nel contesto di questo capitolo, la base mobile RB-Kairos+ non è stata integrata nella pianificazione MoveIt, considerando quindi il manipolatore come montato separatamente in posizione fissa di lavoro.

Capitolo 4

Controllo

4.1 Controllo coordinato UR10e e RB-Kairos+

Dopo aver analizzato in dettaglio nei capitoli precedenti le funzionalità e le configurazioni del manipolatore UR10e e della base mobile RB-Kairos+, ponendo così le basi per una piena comprensione del loro funzionamento come sistemi indipendenti, il presente capitolo si concentra sul controllo congiunto e coordinato di entrambi i robot, introducendo una strategia di gestione integrata attraverso l'implementazione di un controllo in ammettenza, una strategia che permette di integrare il movimento del manipolatore con quello della base mobile in modo reattivo con la sola applicazione di forze esterne da parte dell'operatore.
Questo approccio rappresenta una naturale evoluzione del percorso fin qui intrapreso, poiché dopo l'analisi distinta dei due sistemi, diventa cruciale esplorare la loro interazione cooperativa andando a sfruttare al meglio le potenzialità offerte dalla combinazione dei due robot. L'obiettivo è dunque far operare manipolatore e base in sinergia, garantendo una risposta controllata, creando così un sistema unico con cui l'utente può interagire.

4.2 Modello massa-smorzatore virtuale

Il controllo in ammettenza è una strategia basata sull' interazione, nella quale il manipolatore viene comandato a muoversi in risposta a forze esterne, simulando un comportamento desiderato. Più precisamente, l'ammettenza meccanica rappresenta l'inverso dell'impedenza meccanica: mentre l'impedenza definisce la forza risultante a partire da una velocità, l'ammettenza determina invece la velocità conseguente all'applicazione di una forza esterna [58].

La dinamica virtuale considerata è la seguente:

$$\boldsymbol{M}\,\ddot{\boldsymbol{x}}(t) + \boldsymbol{B}\,\dot{\boldsymbol{x}}(t) = \boldsymbol{F}_{\text{ext}}(t)$$

dove $\boldsymbol{M} = \text{diag}(M_i) \in \mathcal{R}^{(6\times6)}$ è la matrice di massa virtuale, $\boldsymbol{B} = \text{diag}(B_i) \in \mathcal{R}^{(6\times6)}B$ è ila matrice dei coefficientie di smorzamento (damping) e $\boldsymbol{F}_{\text{ext}}(t) \in \mathcal{R}^{(6\times1)}$ è il vettore delle forze applicate dall'uomo. Questa equazione descrive la dinamica di un sistema che oppone inerzia e attrito viscoso al movimento: una forza esterna produce un'accelerazione $\ddot{x}$ proporzionale all'inverso della massa, mentre la velocità $\dot{x}$ è limitata dallo smorzamento.

Il controllo in ammettenza consente al robot di manifestare un comportamento dinamico desiderato in risposta a forze esterne, analogamente a un sistema meccanico passivo; nel caso in cui si impostino una massa virtuale e uno smorzamento ridotti, il manipolatore risulta morbido e facilmente spostabile dall'operatore, caratterizzato da un'alta ammettenza, in cui piccole forze generano movimenti rapidi. Viceversa, aumentando massa e smorzamento virtuali, il robot opporrà maggior resistenza al movimento (bassa ammettenza, servono forze più grandi per ottenere la stessa velocità) [59]. La scelta di $\boldsymbol{M}$ e $\boldsymbol{B}$ consente dunque di regolare la sensibilità del robot rispetto alle forze che gli vengono applicate: valori bassi rendono il robot molto sensibile al tocco umano ma possono portare instabilità in caso di variazioni brusche o urti, mentre valori alti garantiscono stabilità ma l'operatore dovrà applicare forze maggiori per muovere il braccio [59].

In termini di controllo, l'ammettenza è realizzata tramite un anello di retroazione sulla velocità. Con riferimento al caso scalare, in ogni istante il controllorer legge la forza $\boldsymbol{F}_{ext}$ applicata sul sensore e calcola la variazione di velocità necessaria affinché l'equazione $\boldsymbol{M}\dot{\boldsymbol{v}} + \boldsymbol{B}\boldsymbol{v} = \boldsymbol{F}_{ext}$ sia soddisfatta (dove $v = \dot{x}$ è la velocità attuale del TCP, punto centrale della flangia robot).

Discretizzando con periodo di campionamento Δt, l'aggiornamento della velocità desiderata v_s può essere approssimato come:

$$\boldsymbol{a}_d[k] = \frac{\boldsymbol{F}_{ext}[k] - \boldsymbol{B}\boldsymbol{v_s}[k-1]}{\boldsymbol{M}},$$
$$\boldsymbol{v_s}[k] = \boldsymbol{v_s}[k-1] + \boldsymbol{a}_d[k]\,\Delta t,$$

dove a_d è l'accelerazione virtuale calcolata. Questo rappresenta un integratore le cui dinamiche interne sono definite da $\boldsymbol{M}$ e $\boldsymbol{B}$. Lo schema a blocchi in Fig. 4.1 illustra il principio del controllo in ammettenza utilizzato: la forza esterna $\boldsymbol{F}_{ext}$ esercitata dall'operatore alimenta il sistema, il blocco Y_v implementa l'equazione dinamica per generare la velocità desiderata $\boldsymbol{v}_s$. Il controllore C gestisce l'errore tra la velocità desiderata e quella reale, andando ad imporre questa velocità al robot Y_r.

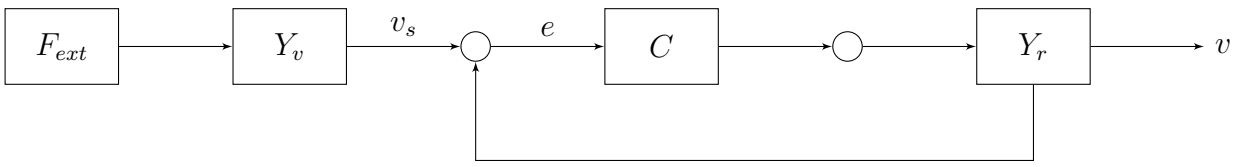


Figura 4.1: Schema semplificato del controllo di ammettenza

4.3 Cinematica differenziale

Il robot UR10e è un manipolatore industriale collaborativo a 6 gradi di libertà (6 giunti rotazionali) prodotto da Universal Robots. Appartenendo alla serie e-Series, il modello UR10e integra un sensore di forza/coppia a 6 assi nel polso, in prossimità della flangia robot [60]. Questo sensore fornisce in tempo reale le tre componenti di forza e le tre componenti di momento (coppie) applicate all'utensile, rendendole leggibili e utilizzabili tramite interfaccia RTDE descritta al paragrafo 1.1.3.

Per descrivere il movimento e l'interazione del manipolatore, si devono richiamare brevemente le equazioni di cinematica e dinamica. Indichiamo con $\boldsymbol{q} = [q_1, \ldots, q_6]^T$ il vettore delle coordinate articolari (angoli di giunto) e con x la posa (posizione e orientazione) dell'endeffector nello spazio operativo (cartesiano). La cinematica differenziale del robot fornisce la relazione tra la velocità dei giunti $\dot{q}$ e la velocità cartesiana $\boldsymbol{v}_s$ dell'end-effector tramite la matrice Jacobiana $\boldsymbol{J}(q) \in R^{6\times6}$, dipendente dalla configurazione attuale dei giunti del robot, mentre $\boldsymbol{v}_s = [v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]^T$ comprende le componenti lineari e angolari della velocità del tool. In configurazioni non singolari, $\boldsymbol{J}$ è una matrice invertibile (nel caso UR10e, J è quadrata 6×6) e quindi si può ottenere la velocità di giunto necessaria a produrre una certa velocità cartesiana desiderata $v_s s$ con la formula:

$$\dot{\boldsymbol{q}}_d = \boldsymbol{J}^{-1}(\boldsymbol{q})\ \boldsymbol{v}_s \tag{4.1}$$

4.4 Implementazione pratica del controllo in ammettenza

L'implementazione è stata realizzata tramite linguaggio Python, utilizzando librerie per la comunicazione e dialogo in tempo reale con l'interfaccia RTDE [11]. Lo script python aggiorna continuamente la lettura delle forze dal sensore integrato, calcola la velocità desiderata in base al modello di ammettenza, la converte in velocità articolari e le invia al controllore del robot tramite un ciclo eseguito ad alta frequenza (500Hz nel caso del controller e-series in dotazione nel nostro caso [11]). Di seguito riporto i principali passi realizzati per l'implementazione:

- Lettura delle forze dal sensore: Ad ogni iterazione, si acquisisce il vettore $\boldsymbol{F}_{ext}$ tramite lettura dal sensore al polso del UR10e. Questo vettore è composto da sei valori: forze $[F_x, F_y, F_z]$ e momenti $[M_x, M_y, M_z]$, restituiti dal metodo getActualTCPForce() [11]. Prima dell'utilizzo di queste forze per creare il modello massa-molla smorzatore, esse possono essere filtrate per ridurre eventuale rumore andando a sottrarre eventuali offset di zero.
  Nell'implementazione considerata, il frame di riferimento scelto è il base frame del robot, ciò significa che una forza positiva F_z farà muovere il robot verso l'alto lungo l'asse z.

- Definizione delle matrici $\boldsymbol{M}$ e $\boldsymbol{B}$: Si inizializzano la matrice di massa virtuale $\boldsymbol{M}$ e di smorzamento $\boldsymbol{B}$ in forma diagonale dimensione 6×6, una massa e un coefficiente di smorzamento per ciascuna dei 6 DoF di movimento del TCP.
  $\boldsymbol{M} = \text{diag}(m_x, m_y, m_z, I_x, I_y, I_z)$
  $\boldsymbol{B} = \text{diag}(b_x, b_y, b_z, d_x, d_y, d_z)$

- Controllo in ammettenza: Il cuore del controllo in ammettenza è un ciclo di calcolo ripetuto in cui la velocità desiderata del TCP viene aggiornata in base alla forza attuale e alla velocità precedente. All'interno dello script Python, il calcolo principale per le componenti cartesiane (lineari e angolari) può essere rappresentato così:

```
#lettura forze e momenti
F_base = np.array(rtde_r.getActualTCPForce())

#calcolo dell'accelerazione virtuale
F_corr = wrench_tool - B_6x6 @ v_s
a = np.linalg.inv(M_6x6) @ F_corr

#integrazione per ottenere velocità desiderata
v_s =  v_s + dt * a
```

Script 4.1: Calcolo della velocità desiderata a partire dalla lettura delle forze

Nel codice riportato, v_s è il vettore 6×1 della velocità attuale del TCP (memorizzata dall'iterazione precedente), $\boldsymbol{M}$ e $\boldsymbol{B}$ sono le matrici di massa e smorzamento, e dt il periodo di campionamento.
La differenza $\boldsymbol{F}$ - $\boldsymbol{B}(\boldsymbol{v}_s)$ implementa la sommatoria $\boldsymbol{F}_{ext}$ - $\boldsymbol{B}(\boldsymbol{v}_s)$ del modello massa-smorzatore, e la moltiplicazione per *np.linalg.inv(M)* calcola l'accelerazione $\boldsymbol{a} = \boldsymbol{M}^{-1}(\boldsymbol{F}\_\boldsymbol{B}\_6x6)$. Moltiplicando per dt e sommando a $\boldsymbol{v}_s$, si ottiene la nuova velocità desiderata.

- Trasformazione della velocità nello spazio dei giunti: Una volta ottenuta la velocità cartesiana desiderata del TCP, questa va convertita in comandi di velocità per ciascun giunto del robot. Si calcola dunque $\dot{q} = J^{-1}(q)\, vs$, utilizzando la matrice Jacobiana del robot calcolata nella configurazione attuale q dei giunti, lette tramite metodo getActualQ() [11]. Nello script Python, la Jacobiana è stata ottenuta tramite l'utilizzo della libreria URDFKDLKinematics, in grado fornire la matrice J a partire dal file URDF e dalla posizione corrente q dei giunti.

```
# Calcolo Jacobiano e velocità articolari

# 1.configurazione attuale dei giunti
q_pos = leggiConfigurazioneGiunti()

# 2.Calcolo della matrice Jacobiana e della
↪  sua inversa
J = calcolaJacobiano(q_pos)  # matrice 6x6

J_inv = invertiJacobiano(J)

# 4. Applica la trasformazione inversa alla
↪  velocità del TCP
q_dot = J_inv * v_tcp  # vettore 6x1
```

Script 4.2: Esempio di pseudocodice Python per la trasformazione della velocità cartesiana del TCP in comandi di velocità articolare mediante calcolo dello Jacobiano

- Trasmissione del comando al robot: I valori di velocità articolare $\dot{q}_d$ vengono infine inviati al robot. Grazie all'interfaccia RTDE è possibile utilizzare il metodo `speedJ()`, il quale imposta una velocità target per ciascun giunto, un'accelerazione massima e un timeout di sicurezza. Il comando viene chiamato ad ogni iterazione, aggiornando continuamente la velocità dei giunti con i nuovi valori calcolati riportati in precedenza negli script 4.1 e 4.2.

```
rtde_c.speedJ(q_dot_arm.tolist(), acceleration=1, time=t)
```

Script 4.3: Invio del comando di velocità articolare al robot

4.5 Micro/Macro Manipolazione

Nel controllo coordinato tra il manipolatore UR10e e la base mobile RB-KAIROS+, la letteratura accademica distingue i concetti di micro e macro manipolazione, in cui un robot caratterizzato da grande portata ma dinamica lenta (macro) è affiancato da un manipolatore più piccolo e agile (micro) [61]. In questa tesi, tali concetti vengono reinterpretati nel contesto di un robot mobile collaborativo, in cui la base mobile assume il ruolo macro (movimenti ampi e lenti) e il manipolatore assume il ruolo micro (movimenti rapidi e precisi). I movimenti lenti e di ampia portata (macro) vengono demandati alla base mobile, mentre i movimenti rapidi e di piccola entità (micro) vengono corretti dal manipolatore.
Per ottenere ciò, il riferimento di velocità V_s, derivante dal controllo in ammettenza descritto nel capitolo precedente, viene separato in due componenti: una a bassa frequenza destinata alla base e una ad alta frequenza destinata al manipolatore. Questa idea di divisione delle frequenze consente al robot mobile collaborativo di eseguire spostamenti grandi in modo stabile attraverso la base, lasciando al braccio il compito di compensare velocemente piccole deviazioni e vibrazioni per un controllo più preciso e naturale. Di seguito sono riportate le metodologie che sono state prese in considerazione per effettuare tale separazione:

- filtro di Kalman
- filtro passa-basso Butterworth
- divisione in frequenza basato su FFT (Fast Fourier Transform)

4.5.1 Analisi delle metodologie

Filtro di Kalman

Il filtro di Kalman è un algoritmo ricorsivo ampiamente utilizzato per stimare lo stato di un sistema dinamico a partire da misure rumorose [62]. Esso opera attraverso due fasi ripetute ad ogni intervallo temporale: una fase di predizione, basata sullo stato precedente stimato e sul modello del sistema, seguita da una fase di aggiornamento, in cui la predizione viene corretta utilizzando la nuova misura acquisita [63]. In questo caso di studio, consideriamo un sistema monodimensionale in cui lo stato x rappresenta la componente lenta della velocità (movimento della base mobile) e la misura z_k corrisponde alla velocità totale (base + braccio) calcolata dal modello massa-smorzatore. Assumendo un modello a velocità costante per la base ($x_k \approx x_{k-1} + w_{k-1}$, con w_k rumore di processo) e considerando che la misura rilevi direttamente lo stato reale ($z_k = x_k + v_k$, con v_k rumore di misura), è possibile scrivere le equazioni ricorsive del filtro di Kalman in forma semplificata.

- Predizione dello stato:

$$x_{k|k-1} = x_{k-1|k-1} \tag{4.2}$$

- Predizione dell'incertezza:

$$P_{k|k-1} = P_{k-1|k-1} + Q_{\text{kalman}} \tag{4.3}$$

- Guadagno di Kalman:

$$K_k = \frac{P_{k|k-1}}{P_{k|k-1} + R_{\text{kalman}}} \tag{4.4}$$

- Correzione dello stato:

$$x_{k|k} = x_{k|k-1} + K_k\big(z_k - x_{k|k-1}\big) \tag{4.5}$$

- Aggiornamento dell'incertezza:

$$P_{k|k} = (1 - K_k) \cdot P_{k|k-1} \tag{4.6}$$

dove $x_{k|k}$ è la stima a posteriori dello stato (velocità lenta stimata della base) al passo k, e $P_{k|k}$ la relativa varianza di errore. In queste equazioni Q_{kalman} e R_{kalman} sono i parametri che rappresentano rispettivamente la varianza del rumore di processo e di misura del filtro, mentre K_k è il guadagno di Kalman calcolato ad ogni iterazione. Nell'implementazione Python fornita, le variabili kalman_state e kalman_P contengono rispettivamente la stima corrente $x_{k|k}$ della componente lenta e la sua covarianza $P_{k|k}$.
Ad ogni ciclo di campionamento il filtro esegue prima la predizione e poi l'aggiornamento con la nuova misura di velocità, calcolando il guadagno K_k e aggiornando di conseguenza kalman_state e kalman_P secondo le equazioni sopra. I valori scelti di Q_{kalman} e R_{kalman} regolano il comportamento del filtro: un Q_{kalman} più grande implica che il filtro si adatti più velocemente alle variazioni, mentre un R_{kalman} più grande fa sì che il filtro consideri le misure meno affidabili e produca quindi stime più levigate ma più lente nel seguire cambiamenti rapidi. Dopo l'aggiornamento, la velocità misurata viene infine scomposta nelle sue due componenti: la stima filtrata kalman_state rappresenta la componente lenta attribuibile al movimento della base, mentre la componente rapida del braccio si ottiene come differenza tra la misura istantanea e lo stato stimato ($v_{\text{braccio}} = z_k - x_{k|k}$). In questo modo $z_k = v_{\text{base}} + v_{\text{braccio}}$, realizzando la separazione desiderata tra moto lento e veloce.

```python
def kalman_filter(z, x_est_old, P_old, Q, R):
    # Predizione
    x_pred = x_est_old
    P_pred = P_old + Q

    # Correzione
    K = P_pred / (P_pred + R)
    x_est = x_pred + K * (z - x_pred)
    P = (1 - K) * P_pred

    return x_est, P
```

Script 4.4: Implementazione python del filtro di Kalman

Filtro Passa Basso Butterworth

Un filtro passa-basso è un sistema in grado di attenuare le componenti ad alta frequenza di un segnale, lasciando invece inalterate quelle a bassa frequenza.
Questo tipo di filtraggio risulta particolarmente utile in applicazioni di controllo real-time, dove la stabilità e la prevedibilità del comportamento del filtro sono prioritarie. Tra le diverse tipologie di filtri passa-basso, il filtro di tipo Butterworth è uno dei più utilizzati per via della sua risposta in frequenza estremamente regolare: in particolare, presenta una banda passante completamente piatta e una transizione monotona verso la banda attenuata [64]. Questo comportamento garantisce che il segnale utile a bassa frequenza non venga distorto, rendendo il Butterworth una scelta efficace in ambiti in cui è importante preservare l'andamento lento del segnale.
Dal punto di vista formale, la funzione di trasferimento di un Butterworth di ordine n (normalizzato) ha modulo dato da:

$$|G(j\omega)| = \frac{1}{\sqrt{1 + \left(\frac{\omega}{\omega_c}\right)^{2n}}} \tag{4.7}$$

dove $|G(j\omega)|$ rappresenta il modulo della funzione di trasferimento, cioè il guadagno in frequenza del filtro; ω è la pulsazione del segnale di ingresso, espressa in radianti al secondo ($\omega = 2\pi f$); $\omega_c = 2\pi f_c$ rappresenta la pulsazione angolare corrispondente alla frequenza di taglio f_c, espressa in hertz; infine, n indica l'ordine del filtro Butterworth. La frequenza di taglio f_c definisce il confine oltre il quale le componenti veloci (micro) vengono attenuate rispetto al segnale di ingresso.

In ambiente Python, il filtro viene implementato mediante la funzione `butter()` della libreria `scipy.signal`, che restituisce i coefficienti numeratore e denominatore del filtro digitale. Un esempio di realizzazione pratica in pseudocodice è mostrato di seguito:

```
from scipy.signal import butter, lfilter_zi, lfilter
b, a = butter(N_butterworth, cutoff / (fs/2), btype='low')
Va_k, zf = lfilter(b, a, x, zi=zi)
Vb_k = v_s - Va_k
```

Script 4.5: Implementazione del filtro Butterworth in Python

Nel contesto in esame, si progetta un filtro passa-basso scegliendo opportunamente l'ordine n e la frequenza di taglio f_c in modo da separare i movimenti macro da quelli micro. Ad esempio fissando $f_c = 1$ Hz, tutte le variazioni di V_s più lente di 1 Hz, passeranno attraverso il filtro e andranno a comandare la base mobile ($V_a(t)$), mentre le componenti più rapide (oscillazioni, vibrazioni, correzioni veloci) verranno fortemente attenuate e rimarranno a carico esclusivo del braccio manipolatore ($V_b(t)$).

Fast Fourier Transform (FFT)

La Fast Fourier Transform (FFT) è uno tra gli strumenti più utilizzati per l'analisi spettrale dei segnali, impiegata in numerosi ambiti applicativi come il controllo di sistemi dinamici, la diagnostica e il monitoraggio delle condizioni operative [65, 66]. Essa costituisce una versione computazionalmente efficiente della Trasformata Discreta di Fourier (DFT), consentendo di scomporre un segnale campionato nel tempo nelle sue componenti sinusoidali fondamentali [66].

Nel contesto del controllo in frequenza, la FFT è utilizzata per separare le componenti lente e veloci di un segnale dinamico, come ad esempio un errore di velocità. Una soglia di frequenza f_s consente di distinguere tra contenuti a bassa frequenza, da affidare ad attuatori lenti (come basi mobili o motori termici), e contenuti ad alta frequenza, delegati ad attuatori più reattivi (come manipolatori o motori elettrici). Tale approccio, noto come frequency-division control, è stato adottato anche da Tebaldi per la gestione duale in veicoli ibridi paralleli, dove il segnale di controllo viene trasformato mediante FFT, filtrato idealmente rispetto alla soglia f_s, e poi riconvertito tramite IFFT in due segnali distinti da inviare separatamente ai due ingressi di controllo [67]. Questo metodo consente una suddivisione netta e priva di modelli espliciti tra le dinamiche lente e rapide, ottimizzando la risposta globale del sistema e migliorando la ripartizione dei compiti tra i sottosistemi coinvolti.

L'algoritmo si ispira al lavoro di Tebaldi et al. sulla gestione energetica di veicoli ibridi [67], dove viene proposta una strategia di controllo basata sulla suddivisione in frequenza dell'errore di velocità $\Delta\omega(k)$. Nell'implementazione proposta in questo lavoro di tesi, tale grandezza è rappresentata da V_s, che corrisponde al vettore delle velocità calcolate a partire dalle forze misurate all'end-effector tramite controllo in ammettenza.

La pipeline dell'algoritmo si sviluppa nei seguenti passaggi:

1. **Acquisizione del segnale**: ad ogni ciclo di controllo (500 Hz), si calcola il vettore:

$$\dot{v}_s = M^{-1}(F - BV_s), \qquad V_s = \text{clip}(v_s, -v_{\max}, v_{\max})$$

   Ogni componente di V_s viene salvata in un buffer circolare di lunghezza $N = 128$.

2. **Preparazione del buffer e calcolo FFT**: una volta riempito il buffer, si calcola la FFT per ciascun canale:

$$v_1, v_2 = \texttt{fft}(V_s)$$

3. **Divisione in frequenza**: si definisce una soglia $f_s = 0.01$ Hz. I coefficienti spettrali vengono separati in:

   - v_1: componenti con frequenza $f < f_s$
   - v_2: componenti con frequenza $f \geq f_s$

   Questa logica riflette l'algoritmo di Tebaldi, dove *"the components of* $F = \texttt{fft}(\Delta\omega)$ *associated with a frequency lower than the threshold* f_s *are stored in* F_1*, others in* F_2*"* [67].

4. **Ricostruzione nel dominio del tempo**: si applica la trasformata inversa per ricostruire i segnali:

$$V_a = \Re(\texttt{ifft}(v_1))[-1], \qquad V_b = \Re(\texttt{ifft}(v_2))[-1]$$

   "The signal is split in two parts through a frequency-based filter [...] The first control input is responsible for the low-frequency components and the second for the high-frequency components. Both components are transformed back into time domain using an inverse FFT, and then applied to the system"

5. **Assegnazione dei comandi**:

   - V_a: assegnato alla base mobile RB-Kairos+
   - V_b: assegnato al manipolatore UR10e

   In analogia al paper: *"the low-frequency disturbances must be up to the first control input τ_1, the high-frequency ones to the second τ_2"*, dove $\tau_1 \leftrightarrow V_a$ e $\tau_2 \leftrightarrow V_b$ [67].

6. **Gestione del transitorio iniziale**: nei primi N cicli, la FFT non è disponibile. Come nel lavoro di Tebaldi: *"During this very short initial transient, the first actuator is responsible for the full control"*[67]. Nel nostro caso, ciò equivale ad assegnare temporaneamente l'intero comando alla base mobile.

Questa architettura di controllo a divisione di frequenza si rivela particolarmente efficace nei sistemi robotici multimodali, consentendo una suddivisione del carico tra attuatori in base al contenuto spettrale del segnale. In questo modo, la base mobile gestisce le componenti a bassa frequenza, mentre il manipolatore compensa le variazioni più rapide, migliorando la reattività complessiva.
Il risultato è una cooperazione fluida tra macro- e micro-manipolazione, utile soprattutto in scenari di interazione fisica. Analogamente a quanto osservato da Tebaldi et al. nel dominio automotive, anche in ambito robotico tale approccio consente una coordinazione efficace tra attuatori con dinamiche differenti, senza richiedere schemi di controllo complessi.

4.5.2 Confronto

Al fine di valutare le prestazioni delle diverse strategie di filtraggio implementate nel sistema di controllo e trovare così una soluzione adeguata al comportamento desiderato del robot, è stato condotto un confronto sistematico tra i tre approcci descritti nelle sezioni precedenti. La valutazione ha preso in considerazione due parametri fondamentali: il tempo di assestamento introdotto dal filtro (*settling time*), ovvero l'intervallo temporale necessario affinchè la risposta del sistema raggiunga e mantenga stabilmente un valore prossimo al valore finale, entro una tolleranza specificata, e la (*deviation*) ovvero la capacità di attenuare le alte frequenze, intesa come deviazione massima rispetto all'ampiezza del segnale di riferimento. Il confronto è stato realizzato tramite simulazioni in ambiente Matlab-Simulink utilizzando tre modelli distinti e variando i parametri caratteristici:

- Filtro Butterworth: frequenza di taglio $f_c \in \{0.25, 0.5, 0.75, 1\}$ Hz e ordine del filtro $n \in \{1, 2, 3\}$.
- Filtro di Kalman: rumore di processo $Q \in \{10^{-10}, 10^{-11}, 10^{-12}\}$.
- Metodo FFT: dimensione della finestra $N \in \{2^9, 2^{10}, 2^{11}\}$ di campioni temporali utilizzati per eseguire la trasformata.

Si riportano di seguito i risultati sperimentali ottenuti in tre scenari comparativi:

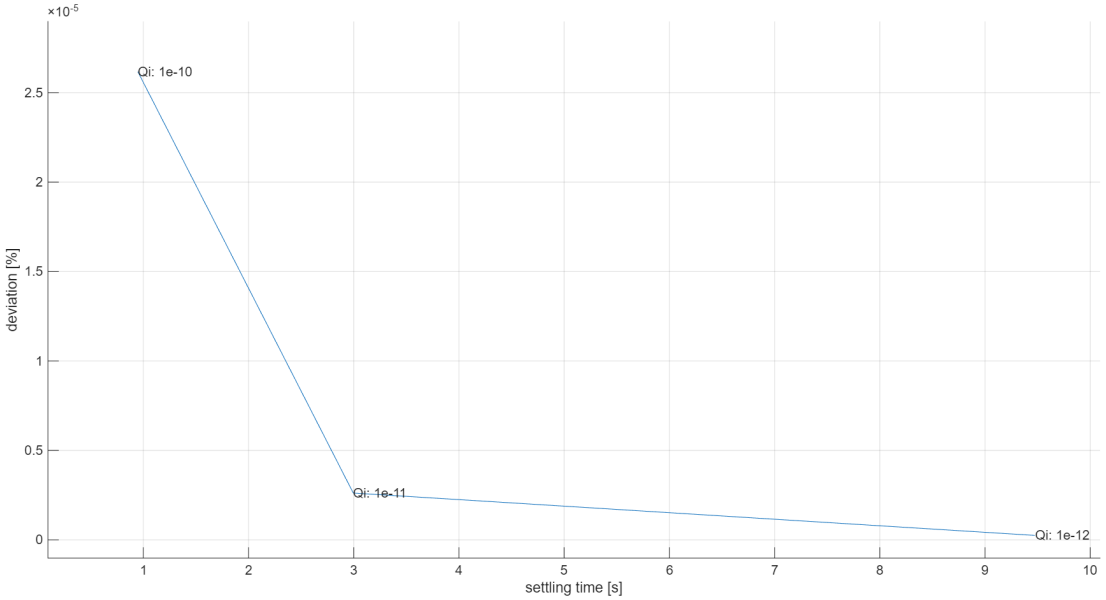


Figura 4.2: Prestazioni del filtro di Kalman

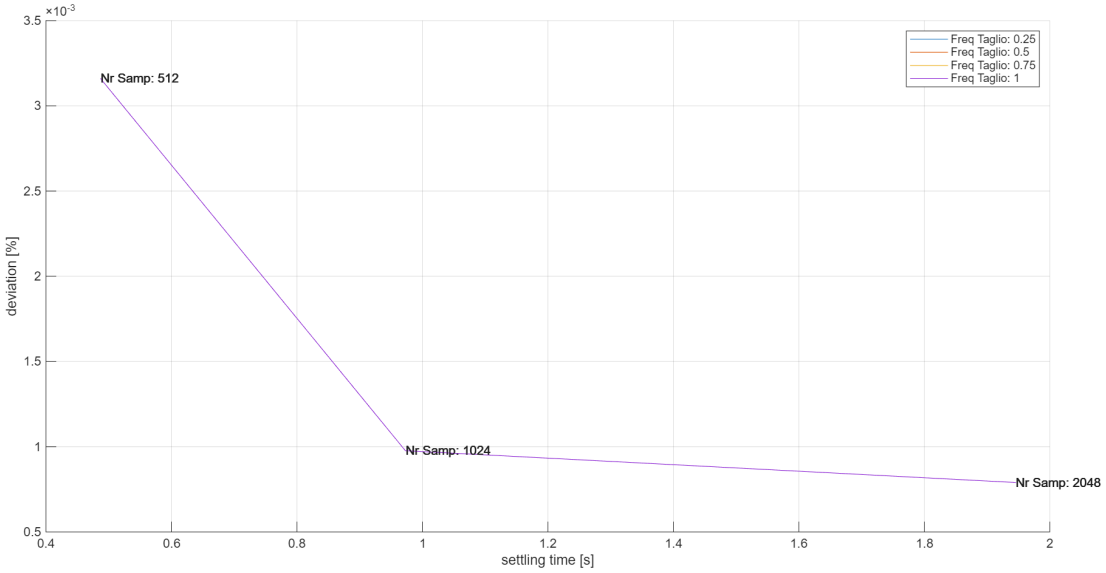


Figura 4.3: Risultati del controllo a divisione di frequenza per lunghezze della finestra FFT

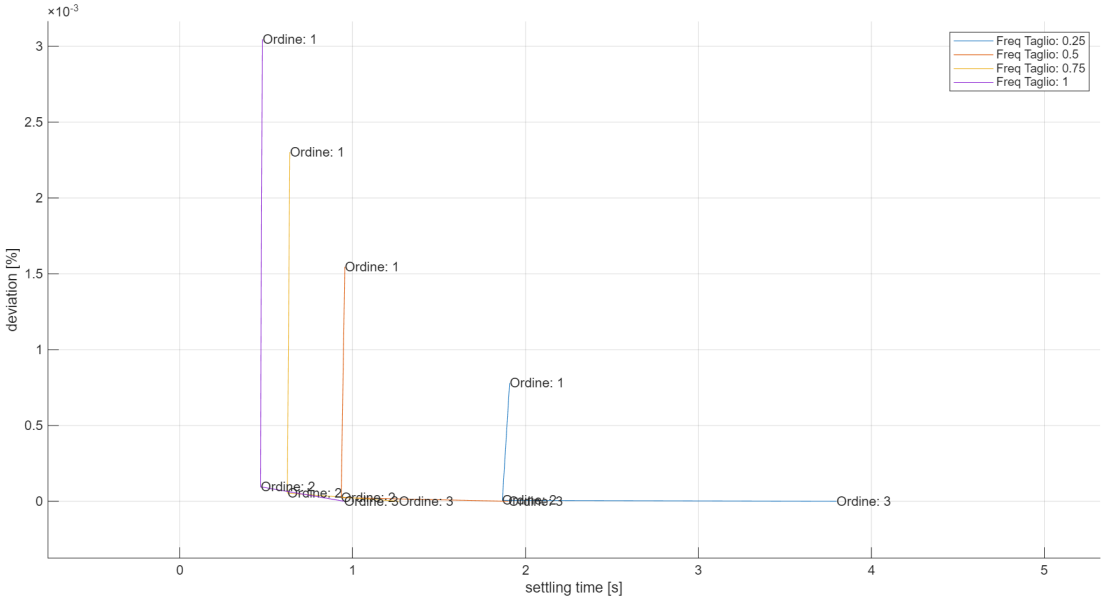


Figura 4.4: Prestazioni del filtro Butterworth per diverse frequenze di taglio ed ordine

Dall'analisi comparativa emerge che, tra le tre tecniche di filtraggio analizzate, il filtro di Kalman mostra la miglior capacità di attenuazione delle componenti ad alta frequenza, ma a scapito di un tempo di assestamento significativamente più elevato. Al contrario, il filtro Butterworth e il controllo con FFT garantiscono tempi di risposta molto più rapidi, mantenendo comunque la deviazione entro valori accettabili.

Da un punto di vista applicativo, un basso tempo di assestamento si traduce in una maggiore prontezza del sistema nel reagire a variazioni improvvise dello stato, garantendo stabilità e una percezione di sicurezza durante l'interazione con l'operatore. Per questo motivo, si è scelto di privilegiare una risposta dinamica più rapida anche a fronte di una moderata perdita in capacità di filtraggio.

In particolare, la scelta finale è ricaduta sul filtro Butterworth del primo ordine con frequenza di taglio pari a 1 Hz, che rappresenta un buon compromesso tra ritardo contenuto e attenuazione spettrale. Frequenze di taglio inferiori o ordini superiori avrebbero comportato un ritardo maggiore, non tollerabile nelle applicazioni di controllo reattivo.

Questa configurazione ha inoltre mostrato un comportamento coerente tra simulazioni e test reali sul robot, confermando la sua efficacia anche in fase di implementazione. La risposta ottenuta si è dimostrata stabile, rapida e intuitiva nell'interazione fisica, pienamente allineata con gli obiettivi progettuali.

Capitolo 5

Risultati sperimentali

Negli scenari di collaborazione fisica tra esseri umani e robot mobili, risulta fondamentale progettare algoritmi di controllo che siano in grado non solo di eseguire movimenti efficaci, ma anche di adattarsi in modo reattivo alle volontà e al comportamento dell'utente. A tal fine, il confronto e comparazione tra diversi approcci di controllo rappresenta un passaggio fondamentale per valutarne l'efficacia ed efficienza.

Il presente lavoro di tesi propone un confronto diretto tra l' algoritmo di controllo sviluppato internamente con metodo di filtraggio descritto nella Sez. 4.5.1(Butterworth)[1], e l'approccio descritto nell'articolo *"SUPER-MAN: SUPER-numerary robotic bodies for physical assistance in huMAN–robot conjoined actions"* [68]. Quest'ultimo è stato selezionato come benchmark e termine di confronto in virtù della sua validazione sperimentale e della centralità attribuita alla cooperazione fisica. Il confronto si è concentrato in particolare sulla modalità di controllo denominata da gli autori con il termine "Locomotion", per la quale il paper fornisce un'implementazione completa, favorendola alla comparazione con l'architettura proposta in questo elaborato.

[1] metodo basato sulla separazione spettrale del comando di velocità tramite un filtro passa-basso Butterworth

5.1 Algoritmo Superman

Il framework SUPER-MAN (SUPERnumerary robotic bodies for physical assistance in huMAN–robot conjoined actions) è stato introdotto come un sistema robotico pensato per assistere fisicamente l'operatore umano attraverso un'interazione corpo-a-corpo [68]. Il sistema integra tre componenti hardware principali: una piattaforma mobile, un manipolatore robotico e un'interfaccia fisica (manopola) che consente all'utente di comandare il robot tramite ammettenza applicata.

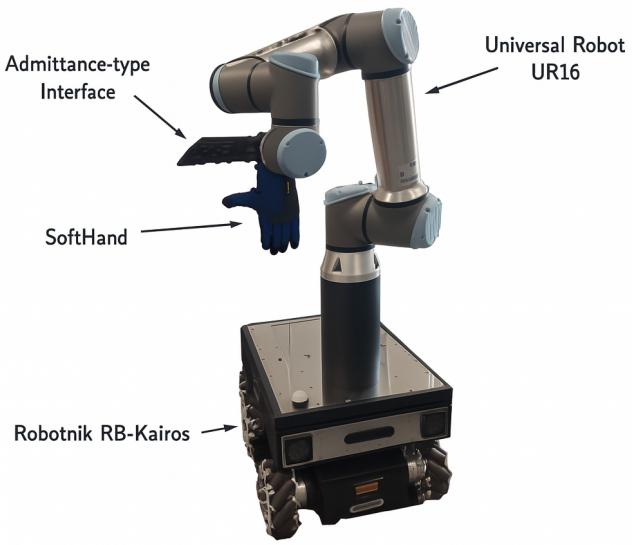


Figura 5.1: Architettura hardware del sistema SUPER-MAN utilizzato come riferimento nello studio. Il sistema è composto da una piattaforma mobile Robotnik RB-Kairos, un manipolatore UR16e [68]

Nell' articolo vengono presentate due modalità operative principali: "Manipulation" e "Locomotion". Nella modalità Manipulation, il controllo si focalizza sulla destrezza ed agilità del braccio robotico, con la base mobile che resta ferma o minimamente coinvolto nel movimento per evitare di compromettere l'accuratezza di manipolazione. Questa modalità è concepita per l'esecuzione di azioni di presa e deposito di oggeti in spazi ristretti, e per la definizione della configurazione preferita-desidera del robot prima del cambio verso la modalità di locomozione.

Al contrario, in modalità Locomotion il sistema sfrutta l'uso della base mobile per eseguire ampi spostamenti nello spazio di lavoro. In questo caso, la priorità del controllo viene assegnata alla componente di traslazione della base, mentre il manipolatore resta pressoché rigido rimanendo in una configurazione globale fissa, non contribuendo al movimento.

Caratteristica	**Modalità Manipulation**	**Modalità Locomotion**
Obiettivo	manipolazione precisa	movimenti ampi e spostamenti
Base mobile	ferma o quasi immobile	reattiva, per spostamenti nel workspace
Manipolatore	agile e flessibile	rigido, in configurazione fissa
Applicazione	interazioni ravvicinate, preleivo e deposito di oggetti	sollevamento e trasporto lungo grandi distanze

Tabella 5.1: Confronto tra le modalità "Manipulation" e "Locomotion" proposte nel framework SUPER-MAN [68].

Nei paragrafi successivi, la modalità Locomotion proposta nell'articolo [68] è stata adottata come riferimento per valutare le prestazioni rispetto alla soluzione sviluppata, in cui il comando di velocità viene separato in componenti ad alta e bassa frequenza e distribuito rispettivamente tra manipolatore e base.

La sezione seguente presenterà un confronto quantitativo e qualitativo tra i due approcci.

5.2 Risultati quantitativi

Per valutare le prestazioni dell'algoritmo proposto basato su filtro passa-basso (Butterworth, rispetto all'approccio SUPER-MAN proposto nell'articolo, sono stati considerati due diversi task rappresentativi:

- Serpentina (Fig.5.2): movimentazione dell'utensile lungo un percorso curvilineo
- Peg-in-Hole (Fig.5.3): inserimento di un puntale in una serie di fori

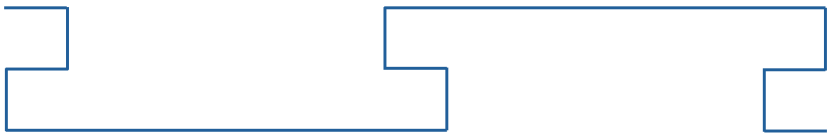


Figura 5.2: Rappresentazione grafica del percorso utilizzato per il task Serpentina

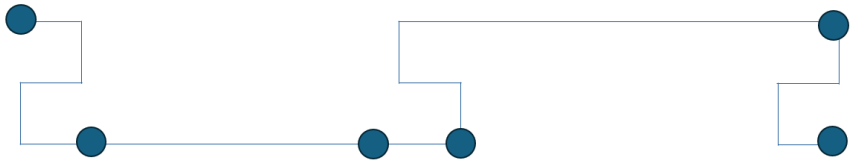


Figura 5.3: Rappresentazione grafica del percorso utilizzato per il task Peg-in-Hole

Su ciascun task sono state prese in considerazione due metriche quantitative di interesse: *l'energia media* erogata dall'intero sistema UR10e + RB-Kairos durante l'esecuzione (espressa in Joule) e lo *sforzo umano* medio esercitato dall'utente durante l'interazione (misurato in Newton). Nelle Tabelle 5.2 e 5.3 sono riportati i valori medi registrati nell'esecuzione dei due task, andando a fare un confronto tra le due diverse modalità di controllo.

	Superman	Butterworth
Average Energy(J)	64,17	32,43
Human Effort(N)	15,43	17,87

Tabella 5.2: Confronto tra gli algoritmi durante il task "Serpentina"

	Superman	Butterworth
Average Energy(J)	46,64	22,23
Human Effort(N)	13,59	14,78

Tabella 5.3: Confronto tra gli algoritmi durante il task "Peg-in-Hole"

Nell'approccio SUPER-MAN in modalità "Locomotion", l'algoritmo di controllo assegna la velocità desiderata quasi interamente alla base mobile, penalizzando fortemente il movimento del manipoaltore; in termini pratici il robot UR10e rimane pressoché rigido e fermo con i giunti, mentre il moto è dato in carico alle ruote del RB-Kairos. Di conseguenza, per inseguire il comando dell'utente la base mobile deve compiere ampi spostamenti superando attriti e inerzie significative[2]. Tale comportamento porta a un maggiore dispendio energetico: l'energia consumata in entrambi i task risulta circa il doppio rispetto al metodo filtrato.

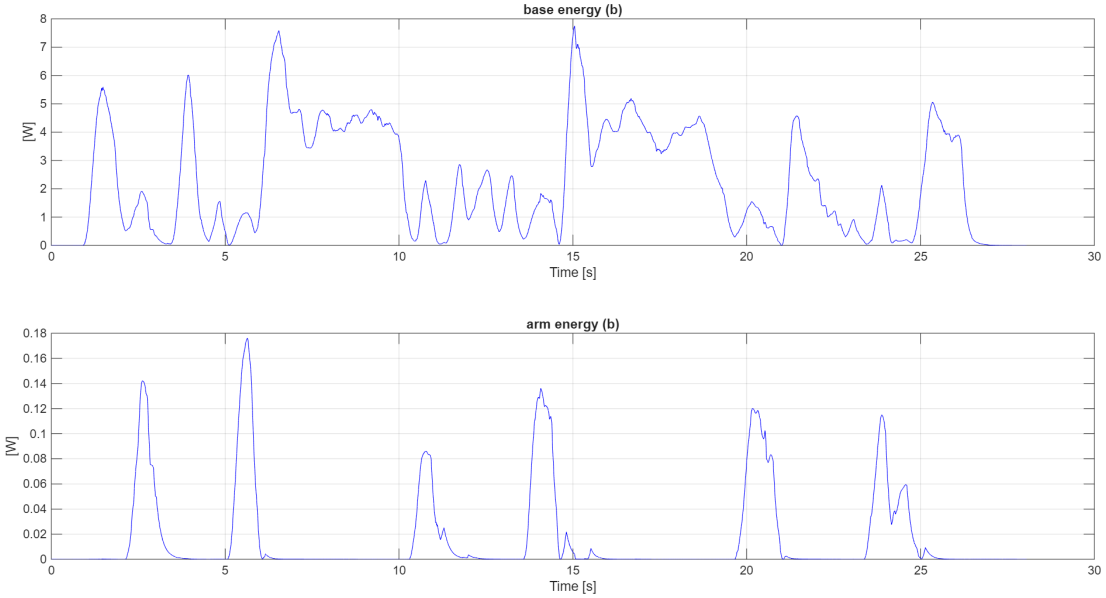


Figura 5.4: Andamento della potenza assorbita dalla base mobile e dal manipolatore durante l'esecuzione del task serpentina in modalità SUPER-MAN.

Al contrario, l'approccio proposto con filtro e separazione delle velocità permette di avere due componenti distinte: una a bassa e una ad alta frequenza. Questa divisione consente una ridistribuzione controllata del movimento tra base e manipolatore, evitando che l'intero carico dinamico sia a carico di un solo sottosistema. In particolare, la componente a bassa frequenza della velocità (variazioni lente) viene assegnata principalmente alla base mobile, mentre la componente ad alta frequenza (variazioni rapide, correzioni iniziali o di fino) viene gestita dal manipoaltore.

[2]L'intero sistema UR10e + RB-Kairos ha un peso di circa 150 Kg.

Il movimento complessivo risulta così più fluido e reattivo, poiché la base non è costretta ad accelerazioni brusche per inseguire ogni piccola variazione del comando. In particolare le fasi iniziali del moto diventano più agevoli: il manipolatore assorbe gli aggiustamenti rapidi, riducendo l'inerzia percepita e lo sforzo fisico richiesto all'utente.

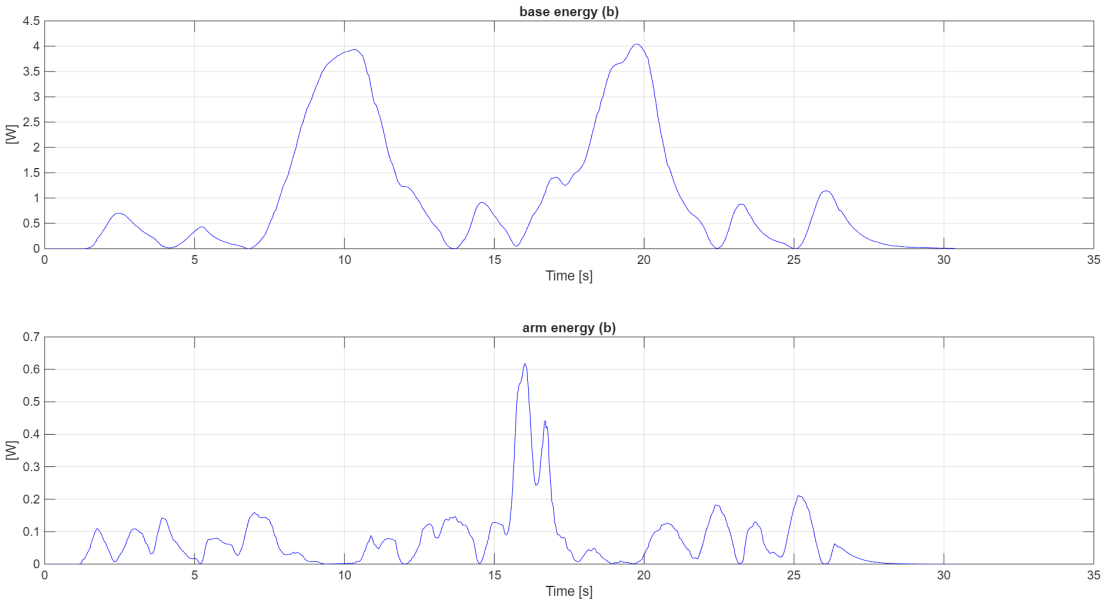


Figura 5.5: Andamento della potenza assorbita dalla base mobile e dal manipolatore durante l'esecuzione del task serpentina in modalità filtrata Butterworth

Dal punto di vista interpretativo, le sostanziali riduzioni di energia e forza con l'approccio Butterworth indicano una maggiore efficienza nell'assistenza: il filtro passabasso elimina componenti di rumore o oscillazioni ad alta frequenza nei comandi di ammettenza, evitando movimenti inutili o bruschi del robot.
Di conseguenza, il sistema dissipa meno energia e l'utente deve applicare meno forza per ottenere lo stesso risultato, beneficiando di movimenti più fluidi e controllati. Al contrario, l'algoritmo SUPER-MAN, pur garantendo la corretta esecuzione del compito, sembra introdurre una dinamica più rigida o reattiva che porta a sforzi maggiori: probabilmente l'assenza di filtraggio dei segnali di input causa leggeri sovra-shoot o micro-oscillazioni che aumentano l'energia impiegata e il carico percepito dall'operatore. In sintesi, l'analisi quantitativa conferma che il controller proposto ottimizza la collaborazione fisica riducendo sia il dispendio energetico del robot sia l'impegno di forza richiesto all'uomo, su entrambe le tipologie di task esaminate. Ciò rappresenta un importante miglioramento in termini di ergonomia e prestazione del sistema uomo-robot.

5.3 Risultati qualitativi

Oltre alle misure oggettive sopra descritte, è stata condotta inoltre una valutazione qualitativa dell'esperienza d'uso attraverso il questionario NASA-TLX, allo scopo di misurare e determinare il carico di lavoro percepito dagli utenti con le due diverse modalità di controllo per entrambi i task precedentemente descritti (Serpentina e Peg-in-Hole) [3]. Il NASA Task Load Index (NASA-TLX) è uno strumento riconosciuto e standardizzato in ambito scientifico per la valutazione soggettiva del carico mentale, fisico ed emotivo associato all'esecuzione di un compito o task. Si basa su sei dimensioni fondamentali: carico mentale, carico fisica, carico temporale, performance percepita, sforzo e frustrazione.
Ciascuna dimensione viene valutata su una scala graduata contribuendo così al calcolo di un indice complessivo di carico di lavoro (workload), che riflette la percezione dell'utente in modo strutturato e comparabile. Questo metodo viene considerato uno standard per l'analisi del workload in contesti operativi, ergonomici e di interazione uomo-macchina [69, 70].
In Figura 5.6 è mostrato il questionario NASA-TLX sottoposto a gli utenti presi in esame.

[3] L'ordine di esecuzione dei task ("Serpentina" e "Peg-in-Hole") e l'ordine di utilizzo degli algoritmi di controllo ("Superman" e "Butterworth") sono stati decisi in modo casuale per ogni partecipanti, evitando così di influenzare e condizionare il risultato.

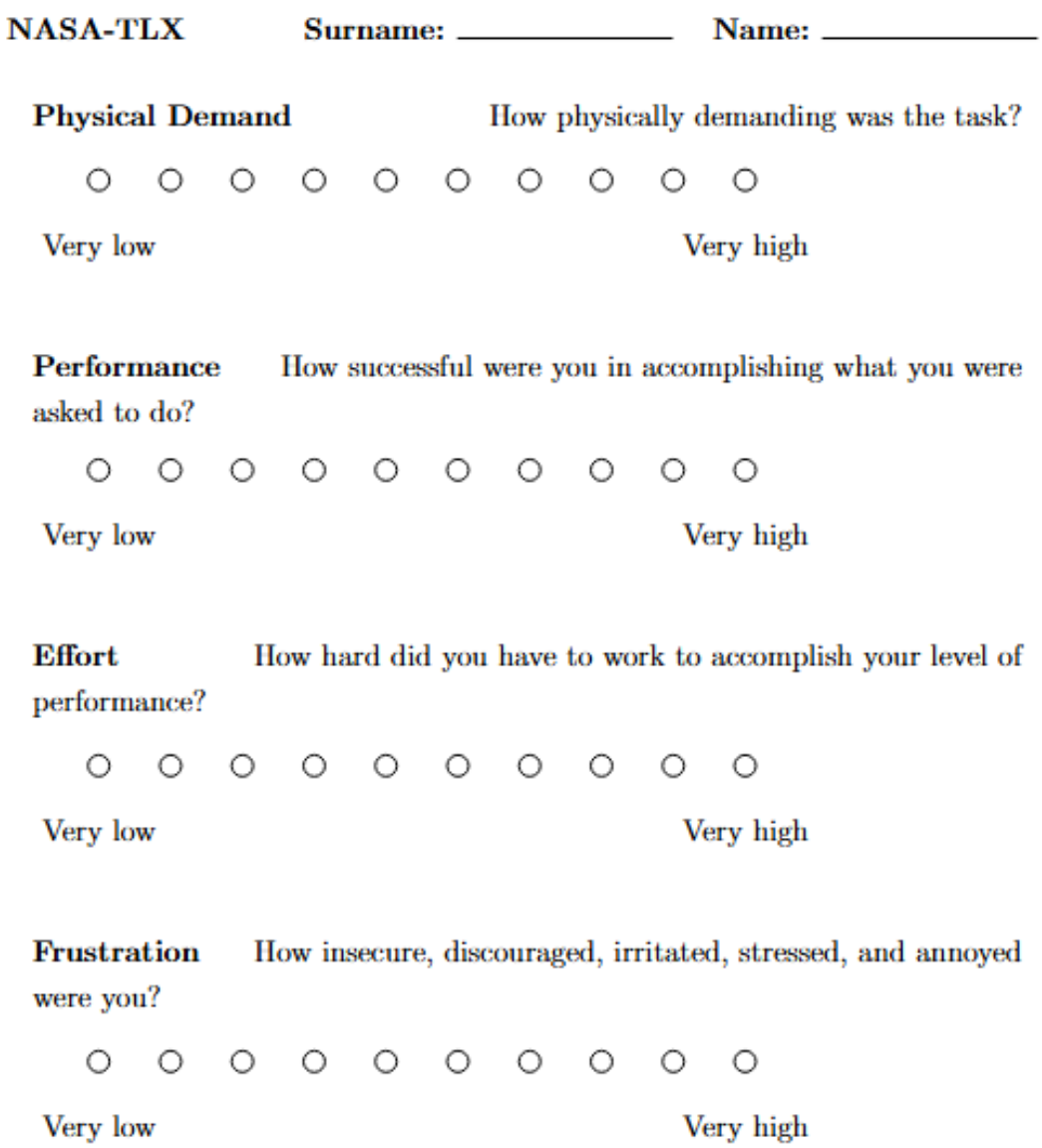


Figura 5.6: Esempio del questionario NASA-TLX utilizzato per la valutazione soggettiva del carico di lavoro

Per il lavoro di tesi sono state prese in considerazione solo quattro di queste sei dimensioni, ritenute le più rilevanti per i due task svolti sul robot reale, omettendo le componenti di domanda temporale e sforzo complessivo. Il questionario è stato somministrato a un campione di persone eterogeneo per sesso ed età, al fine di garantire una maggiore equità e rappresentatività nei risultati ottenuti. Di seguito sono riportate le quattro metriche NASA-TLX considerate e il loro relativo significato:

- *Sforzo Fisico (Physical Demand)*: quantifica il livello di sforzo e fatica muscolare richiesto all'utente per portare a compimento il task.
  Un punteggio alto indica che il compito è stato percepito come fisicamente molto impegnativo.

- *Performance*: rappresenta la valutazione soggettiva della propria performance nel compito. Nel questionario NASA-TLX originale un punteggio elevato corrisponde a bassa performance percepita, ovvero insoddisfazione rispetto agli obiettivi raggiunti; tuttavia, per semplicità interpretativa da parte dell'utente, nel questionario sottoposto per il lavoro di tesi il punteggio è stato ribaltato affinché valori alti andassero ad indicare una migliore performance ed una maggiore soddisfazione dell'utente riguardo al proprio risultato.

- *Sforzo Mentale (Mental Effort)*: indica il carico cognitivo, attenzione richiesta, la concentrazione e lo sforzo mentale richiesti dal compito.

- Frustrazione (Frustration): misura il livello di stress, irritazione e scoraggiamento provato durante l'esecuzione del compito. Un punteggio alto implica che l'attività è stata fonte di notevole frustrazione o insoddisfazione per l'utente, mentre un valore basso indica un'esperienza serena e priva di stress.

Per ciascuna di queste quattro metriche sono stati raccolti i punteggi NASA-TLX da partecipanti, dopo aver eseguito i due diversi task Peg-in-Hole e Serpentina, sia con il controllo tramite divisione in frequenze proposto nell'elaborato di tesi sia con controllo SUPER-MAN riportato nell'articolo precedentemente citato. I risultati sono sintetizzati nei boxplot delle Figure 5.7, 5.8, 5.9, 5.10, che mostrano la distribuzione dei punteggi da 1 a 10 sull'asse y e la combinazione di task-algoritmo sull'asse x.

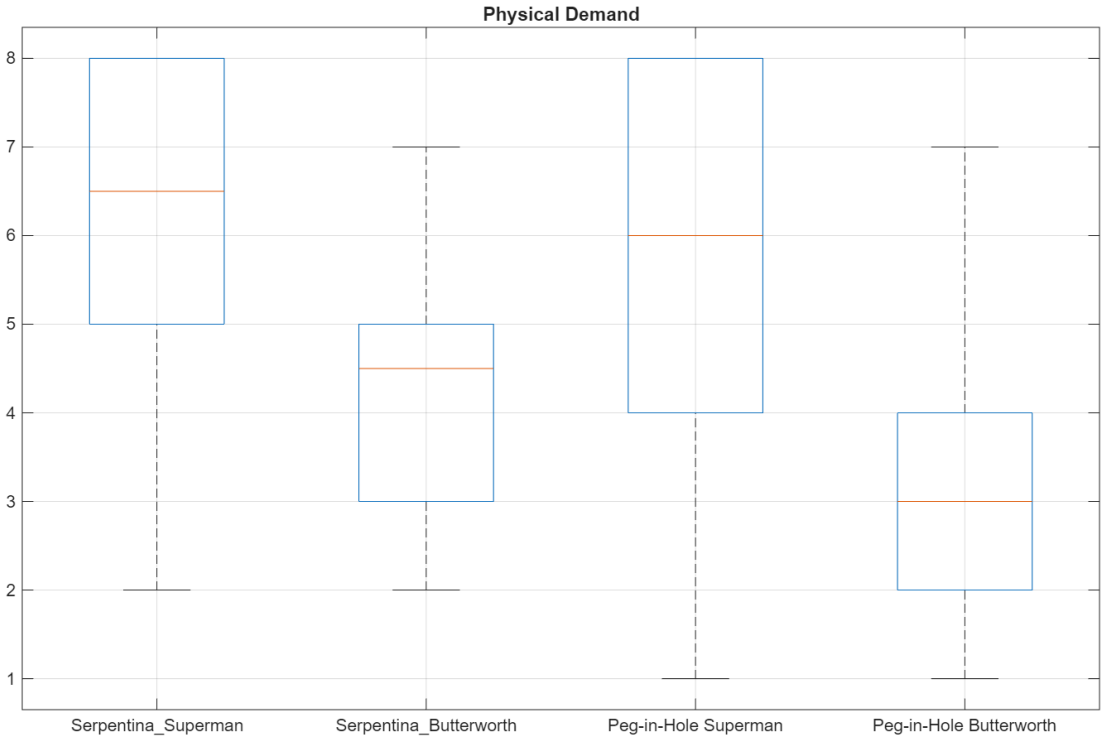


Figura 5.7: Sforzo fisico

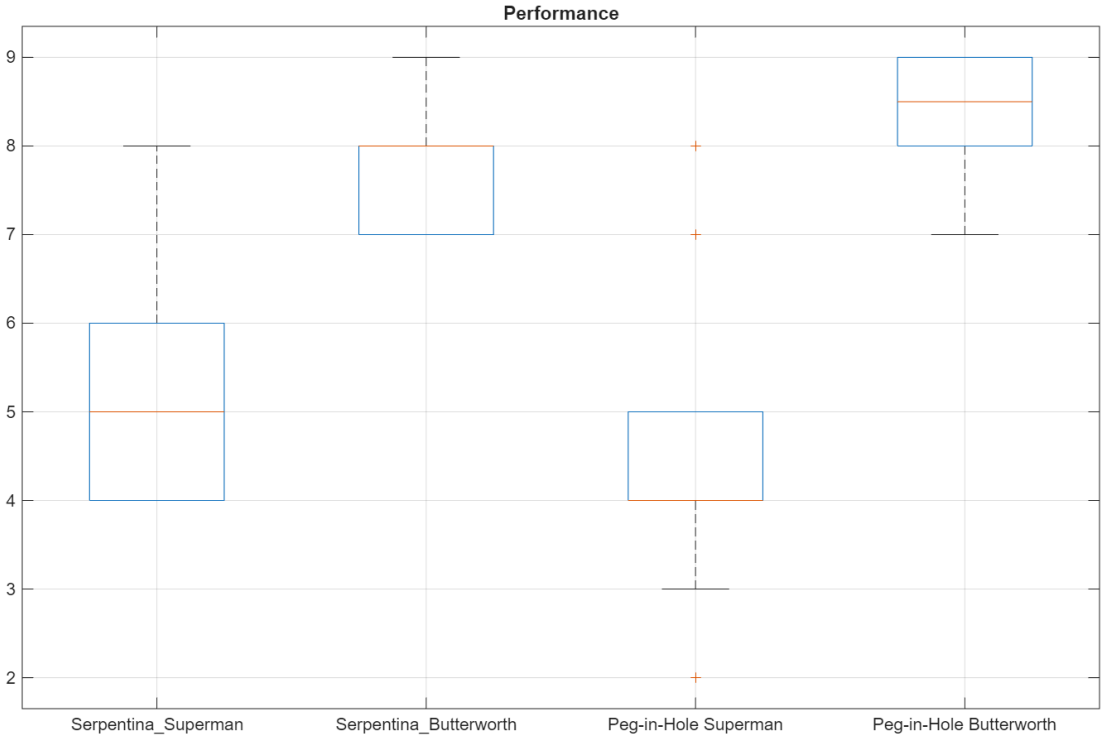


Figura 5.8: Performance

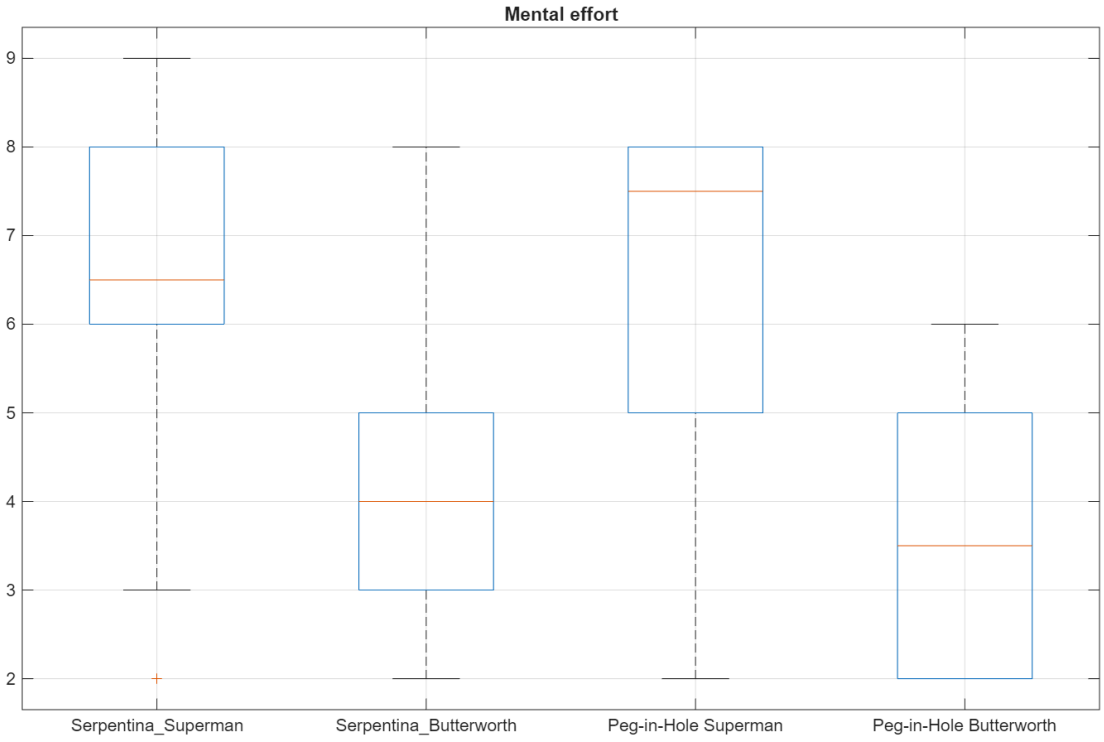


Figura 5.9: Sforzo mentale

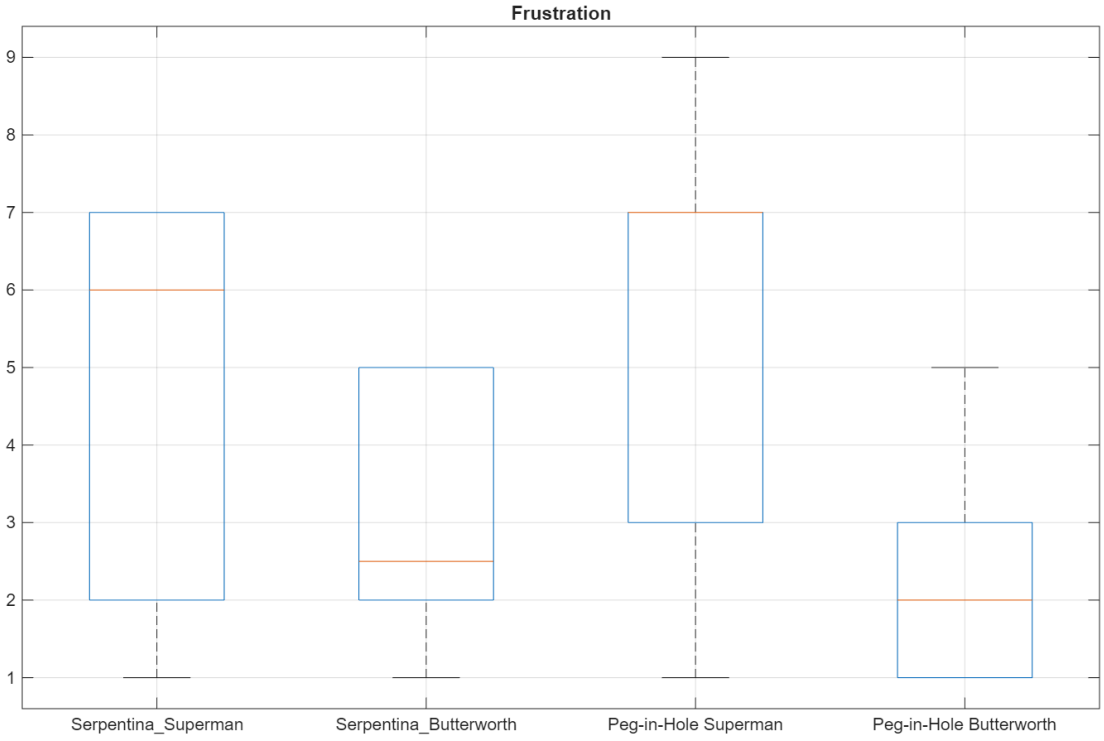


Figura 5.10: Frustrazione

Si osserva in generale una chiara tendenza: in ogni metrica considerata, l'algoritmo di controllo proposto ottiene valutazioni migliori: minore carico percepito e migliore performance rispetto all'algoritmo SUPER-MAN, su entrambi i compiti. Di seguito si analizzano nel dettaglio i vari aspetti.

- Domanda Fisica (Figura 5.7): i punteggi di sforzo fisico percepito risultano nettamente inferiori utilizzando il controllo con filtro Butterworth rispetto all'approccio SUPER-MAN. Nel task Serpentina, ad esempio, il punteggio mediano di Physical Demand passa da circa 6.5 (SUPER-MAN) a 4.5 (Butterworth), evidenziando una sensibile diminuzione della fatica fisica come indicato dagli utenti. Analogamente, per il task Peg-in-Hole la mediana scende da 6 (con SUPER-MAN) a 3 (con Butterworth), con distribuzioni dei punteggi quasi disgiunte tra i due algoritmi. Questi dati indicano che i partecipanti hanno ritenuto il compito meno faticoso fisicamente quando assistiti dal controllo con algoritmo filtrato. Ciò è coerente con le misure quantitative oggettive riportante in precedenza nella Sez. 5.2: la riduzione della velocità della base mobile ed una maggior manovrabilità del manipolatore si traduce in una minore percezione di sforzo muscolare da utilizzare per controllare il sistema robot.

- Performance (Figura 5.8): la valutazione soggettiva della performance mostra marcate differenze a favore del controllo Butterworth. In entrambe le attività, i partecipanti hanno attribuito al loro stesso lavoro dei punteggi di performance, nel grafico si vede che per il task Peg-in-Hole la mediana del punteggio Performance è di circa 8.5/10 con controllo Butterworth, contro un valore vicino a 4/10 con SUPER-MAN. Anche per il task Serpentina è riportata una discrepanza simile: si nota una mediana di 8/10 contro una mediana di 5/10. Ricordando che in questa analisi un punteggio maggiore corrisponde a una migliore performance percepita, possiamo concludere che gli utilizzatori si sono sentiti più efficaci e soddisfatti del risultato ottenuto quando il robot era controllato tramite l'algoritmo proposto, esprimendo anche verbalmente una maggior confidenza e maneggevolezza d'uso.

- Sforzo Mentale (Figura 5.9): anche il carico cognitivo richiesto risulta inferiore con il controllo proposto. I boxplot relativi allo sforzo mentale mostrano punteggi mediani attorno a 4 (Butterworth) rispetto a valori circa doppi intorno a 7 (SUPER-MAN). Ad esempio, per il compito di Peg-in-Hole la distribuzione dei punteggi con SUPER-MAN è centrata su livelli

alti di sforzo mentale, mentre con Butterworth i soggetti riportano livelli significativamente più bassi, con alcuni valori minimi fino a 2/10. Ciò suggerisce che l'interazione filtrata richiede meno concentrazione e stress mentale: probabilmente il movimento più fluido e naturale del manipolatore ha ridotto la necessità di continui aggiustamenti correttivi da parte dell'utente e l'attenzione costante per controllare eventuali oscillazioni involontarie.

- Frustrazione (Figura 5.10): i punteggi di Frustration con controllo attraverso divisione in frequenze sono mediamente molto bassi, indicando che gli utenti hanno affrontato l'attività con minimo stress e irritazione.
  Al contrario, in condizione di controllo SUPER-MAN i valori salgono verso la parte alta della scala (mediane 6-7), con alcuni partecipanti che hanno riportato punteggi di frustrazione fino a 9/10 nel task Peg-in-Hole.
  Questa differenza è indicativa di un'esperienza utente nettamente migliorata tramite l'approccio proposto: gli utenti si sono sentiti più in controllo della situazione e meno infastiditi dal comportamento del robot, rispetto a quanto emergeva con l'approccio SUPER-MAN.

Per ciascuna delle quattro dimensioni considerate, è stato applicato un **t-test** per campioni indipendenti (independent samples t-test), metodo statistico appropriato per confrontare le medie di due gruppi distinti e determinare se le differenze osservate siano statisticamente significative. Il t-test confronta la media dei punteggi ottenuti nei due gruppi e restituisce un valore p, che rappresenta la probabilità che la differenza osservata sia dovuta al caso. Un valore di p inferiore a 0.05 indica una differenza statisticamente significativa tra i gruppi, ossia non attribuibile alla sola variabilità casuale.

I risultati del t-test hanno evidenziato differenze significative tra le due condizioni in tre delle quattro dimensioni analizzate. In particolare, i punteggi associati alla configurazione Butterworth risultano significativamente più bassi nelle dimensioni di Physical Demand ($p = 0.0499$) e Mental Effort ($p = 0.0458$), suggerendo una minore percezione di sforzo fisico e cognitivo da parte degli utenti. Parallelamente, la dimensione Performance mostra valori significativamente più elevati nella stessa configurazione ($p = 0.0002$), indicando una migliore efficacia percepita del controllo. Al contrario, la dimensione Frustration non ha mostrato differenze statisticamente significative ($p = 0.0717$), suggerendo che il livello di frustrazione percepita rimanga sostanzialmente invariato tra le due modalità operative.

I risultati sono riportati nelle seguenti immagini:

```
---- Physical Demand ----
p-value: 0.0499
>> DIFFERENZA SIGNIFICATIVA

---- Performance ----
p-value: 0.0002
>> DIFFERENZA SIGNIFICATIVA

---- Mental Effort ----
p-value: 0.0458
>> DIFFERENZA SIGNIFICATIVA

---- Frustration ----
p-value: 0.0717
>> Nessuna differenza significativa
```

Figura 5.11: Risultati del t-test per le quattro dimensioni del questionario NASA-TLX nel task Serpentina

```
---- Physical Demand ----
p-value: 0.0164
>> DIFFERENZA SIGNIFICATIVA

---- Performance ----
p-value: 0.0000
>> DIFFERENZA SIGNIFICATIVA

---- Mental Effort ----
p-value: 0.0024
>> DIFFERENZA SIGNIFICATIVA

---- Frustration ----
p-value: 0.0021
>> DIFFERENZA SIGNIFICATIVA
```

Figura 5.12: Risultati del t-test per le quattro dimensioni del questionario NASA-TLX nel task Peg-in-Hole

In sintesi, i risultati qualitativi ottenuti tramite il questionario NASA-TLX confermano pienamente le tendenze emerse dalle metriche quantitative della Sez. 5.2. L'implementazione basata su filtro Butterworth ha ottenuto valutazioni superiori in termini di comfort ed usabilità: ha ridotto il carico di lavoro fisico e mentale percepito dagli operatori, migliorato la soddisfazione riguardo alla performance e minimizzato lo stress-frustrazione associato ai task richiesti.

Il riscontro positivo ottenuto dagli utilizzatori mette in luce il potenziale dell'approccio proposto in questo lavoro di tesi, suggerendo una buona base per l'utilizzo in applicazioni in contesti reali.

Capitolo 6

Conclusioni

In conclusione, il lavoro di tesi ha raggiunto diversi obiettivi.
In primo luogo, questa ricerca ha dimostrato il potenziale di un sistema robot composto da un manipolatore ed una base mobile. Unendo il mondo della robotica collaborativa a quello della robotica mobile, si è potuto affrontare nuove tematiche di ricerca e sfide aziendali. In secondo luogo, il controllo in ammettenza implementato ha permesso di governare il sistema in modo naturale e reattivo da parte degli utenti. Ciò ha reso possibile interagire con l'intero sistema robotico attraverso comandi semplici e intuitivi, superando la necessità di utilizzare le interfacce native e spesso macchinose dei singoli sottosistemi.

Infine, questo lavoro di tesi ha dimostrato che i robot possono essere utilizzati, controllati, integrati in contesti aziendali in modo intuitivo e semplice; riducendo il divario tra utenti esperti e non esperti.

Questo progetto offre quindi uno sguardo verso il futuro, in termini di coesistenza, collaborazione e interazione tra esseri umani e robot.

Bibliografia

[1] SICK Sensor Blog. "safety for collaborative robot applications," sick ag,. https://www.sick.com/sk/cs/sick-sensor-blog/safety-for-collaborative-robot-applications/w/blog-safety-collaborative-robot-applications., 2023.

[2] International Organization for Standardization. Iso 8373:2021 – robotics — vocabulary, iso,. https://cdn.standards.iteh.ai/samples/75539/1bc8409322eb4922bf680e15901852d2/ISO-8373-2021.pdf, 2021.

[3] Universal Robots. "universal robots – collaborative robotic automation," universal robots. https://www.universal-robots.com/.

[4] "ABB expands GoFa cobot family. " factory & handling solutions,. https://factoryandhandlingsolutions.co.uk/abb-expands-gofa-cobot-family-2/.

[5] "Autonomous mobile robots. " modula,. https://www.modula.eu/robotic-integrations/autonomous-mobile-robots/.

[6] "RB-KAIROS. " ros components,. https://www.roscomponents.com/product/rb-kairos/.

[7] S&T Automation. "quanti gradi di libertà ha un robot?", s&t automation blog,. https://www.stautomation.eu/blog/approfondimenti/quanti-gradi-di-liberta-ha-un-robot/, 2022.

[8] Universal Robots. "technical specifications – ur10e," universal robots, 2021.

[9] Universal Robots. Universal robots ros driver. https://github.com/UniversalRobots/Universal_Robots_ROS_Driver.

[10] SP Automation. "elevating automation: The universal robots ur10e collaborative robot unveiled," sp automation https://sp-automation.co.uk/elevating-automation-the-universal-robots-ur10e-collaborative-robot-unveiled/.

[11] Universal Robots. "real-time data exchange (rtde) guide", universal robots,. https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/.

[12] Universal Robots. "rb-kairos+ – ur+ certified mobile manipulator," ur+ marketplace website. https://www.universal-robots.com/marketplace/products/01tP40000071NgSIAU/.

[13] Universal Robots. Ur10e e-series user manual, sw 5.9, universal robots a/s,. https://www.universal-robots.com/download/manuals-e-seriesur20ur30/user/ur10e/59/user-manual-ur10e-e-series-sw-59-english-international-en/, 2021.

[14] Qviro. "universal robots ur10e specifications," qviro,. https://qviro.com/product/universal-robots/ur10e/specifications.

[15] "Autonomous bin-picking kit for machine tending. " metalforming magazine,. https://www.metalformingmagazine.com/article/?/pressroom-automation/robotics/autonomous-bin-picking-kit-for-machine-tending.

[16] "Brandt A/S: Automated sanding with collaborative robots. " universal robots,. https://www.universal-robots.com/case-stories/brandt-a-s/.

[17] QViro. "robotnik rb-kairos – specifications and features," qviro robotics database,. https://qviro.com/product/robotnik/rb-kairos/specifications.

[18] J. Cacace et al. "a mobile manipulator for industrial applications in unstructured environments," sensors (basel), vol. 23, no. 23, art. no. 9906,. https://pmc.ncbi.nlm.nih.gov/articles/PMC10728719/., 2023.

[19] Robotnik. "rb-kairos+ datasheet and technical specifications," robotnik, 2024. https://robotnik.eu/products/mobile-robots/rb-kairos-2/.

[20] Robotnik. Rb-kairos+ user manual, versione 1.1, robotnik automation,, 2020.

[21] Deltacon S.r.l. "rb-kairos+ – robot manipolatore mobile robotnik," deltacon,. https://www.deltacon.it/prodotto/rb-kairos-robot-manipolatore-mobile-robotnik/.

[22] Robotnik Automation. "what are the different types of robots? the 5 strong points of 5 autonomous mobile robots," robotnik, https://robotnik.eu/what-are-the-different-types-of-robots-the-5-strong-points-of-5-autonomous-mobile-robots/.

[23] "RB-KAIROS+ – Mobile manipulator with UR10e. " robotnik automation,. https://robotnik.eu/products/mobile-robots/rb-kairos-2/.

[24] "ROS Mobile Manipulators Training. " the construct,. https://www.theconstruct.ai/ros-mobile-manipulators-training/.

[25] ROS. "ros navigation stack documentation,". ROSWiki,https://wiki.ros.org/navigation.

[26] G. Grisetti. C. stachniss, and w. burgard, "improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Trans. Robotics*, vol.23, no.1, pp.34–46,, 2007.

[27] "gmapping Package (ROS laser-based SLAM). " ros wiki,. https://wiki.ros.org/gmapping., 2024.

[28] M. A. Chunab-Rodríguez. J. a. p. lázaro, m. c. romero-ternero e m. a. vázquez, "a free simulation environment based on ros for teaching autonomous vehicle navigation algorithms," applied sciences, vol. 12, n. 14, art. 7277,. https://www.mdpi.com/2076-3417/12/14/7277, 2022.

[29] "ROS Q&A: How to provide a map to the Mapping Tutorial. " the construct,. https://www.theconstruct.ai/ros-qa-119-ros-mapping-tutorial-provide-map/.

[30] "map_server ROS Wiki. " wiki.ros.org. . https://wiki.ros.org/map_server.

[31] "nav_msgs/OccupancyGrid Message Definition. " docs.ros.org,. https://docs.ros.org/en/noetic/api/nav_msgs/html/msg/OccupancyGrid.html, 2025.

[32] "ROS Q&A: How to edit a map generated with GMapping. " the construct,. https://www.theconstruct.ai/ros-qa-136-how-to-edit-a-map-generated-with-gmapping/.

[33] "nav_msgs/GetMap Documentation. " docs.ros.org,. https://docs.ros.org/en/noetic/api/nav_msgs/html/srv/GetMap.html, 2025.

[34] ["amcl ROS Wiki. " wiki.ros.org. . https://wiki.ros.org/amcl.

[35] dazi2_2. "ros: Robot localization," velog.io,. https://velog.io/@dazi2_2/ROS-Robot-Localization.

[36] Open Source Robotics Foundation. "amcl (adaptive monte carlo localization)," ros wiki,. http://wiki.ros.org/amcl, 2014.

[37] J. H. Hwangbo. "ros_control: A generic and simple control framework for ros," journal of open source software, vol. 2, no. 20, p. 456,. https://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf, 2017.

[38] "Announcing MoveIt! ROS robotics news. " ros.org, 2020. . https://www.ros.org/news/2013/05/announcing-moveit.html.

[39] MoveIt. "concepts," moveit documentation,. https://moveit.ai/documentation/concepts/, 2024.

[40] Yaskawa Motoman. "what is ros?". https://knowledge.motoman.com/hc/en-us/articles/4408002862871-What-is-ROS.

[41] A. Pandey. D. t. lofaro, "introducing kinematics with robot operating system (ros)," in *Proc. c Annu. Conf. Expo.*,. https://www.asee.org/public/conferences/56/papers/11460/download, 2015.

[42] "xacro (XML Macros)". Ros wiki,. https://wiki.ros.org/xacro, 2015.

[43] "Generate Self-Collision Matrix. " moveit documentation. https://moveit.picknik.ai/main/doc/examples/setup_assistant/setup_assistant_tutorial.html#step-2-generate-self-collision-matrix.

[44] "Add Planning Groups". Moveit documentation. https://moveit.picknik.ai/main/doc/examples/setup_assistant/setup_assistant_tutorial.html#step-4-add-planning-groups.

[45] "Add Robot Poses. " moveit documentation. https://moveit.picknik.ai/main/doc/examples/setup_assistant/setup_assistant_tutorial.html#step-5-add-robot-poses.

[46] S. Chitta et al. "ros_control: A generic and simple control framework for ros," journal of open source software, vol. 2, no. 20, p. 456, dec. https://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf, 2017.

[47] "MoveIt Controllers. " moveit documentation. https://moveit.picknik.ai/main/doc/examples/setup_assistant/setup_assistant_tutorial.html#step-10-moveit-controllers.

[48] Universal Robots. "universal robots ros driver," github repository. https://github.com/UniversalRobots/Universal_Robots_ROS_Driver.

[49] "URDF and SRDF Tutorial. " moveit documentation,. https://moveit.picknik.ai/main/doc/examples/urdf_srdf/urdf_srdf_tutorial.html.

[50] "Kinematics Configuration Tutorial. " moveit tutorials (noetic),. https://moveit.github.io/moveit_tutorials/doc/kinematics_configuration/kinematics_configuration_tutorial.html.

[51] "MoveIt Configuration Tutorial. " moveit documentation. https://moveit.picknik.ai/main/doc/how_to_guides/moveit_configuration/moveit_configuration_tutorial.html.

[52] "The Open Motion Planning Library. " ompl official website. https://ompl.kavrakilab.org/.

[53] The move_group node. Moveit documentation, . https://moveit.picknik.ai/main/doc/concepts/move_group.html.

[54] EECS C106A/206A Course Staff. "lab 5: Inverse kinematics," dept. of electrical engineering and computer sciences, university of california, berkeley, fall. https://pages.github.berkeley.edu/EECS-106/fa21-site/assets/labs/Lab_5__Inverse_Kinematics.pdf, 2021.

[55] I. A. Şucan. M. moll, and l. e. kavraki, "moveit![ros topics],". https://www.researchgate.net/publication/254057457_MoveitROS_topics.

[56] MoveIt. "motion planning overview," moveit documentation,. https://moveit.picknik.ai/main/doc/concepts/motion_planning.html.

[57] Universal Robots. "universal robots ros driver," github repository. https://github.com/UniversalRobots/Universal_Robots_ROS_Driver.

[58] N. Hogan. "impedance control: An approach to manipulation," asme journal of dynamic systems, measurement, and control, vol. 107, no. 1, pp. 1–24, 1985, doi: 10.1115/1.314, 0702.

[59] H. Kim. W. yang, "variable admittance control based on human–robot collaboration observer using frequency analysis for sensitive and safe interaction", sensors, vol. 21, no. 5, p. 1899, 2021.

[60] R. L. Williams II. "universal robot ure-series cobot kinematics & dynamics", ohio university, jan. 2024.

[61] K. D. Do and J. Pan. "the macro/micro manipulator: An improved architecture for robot control," mechatronics, vol. 13, no. 2, pp. 217–239, mar. 2003, doi: 10.1016/0736-5845(93)90056-p.

[62] MathWorks. "filtro di kalman," mathworks italia. https://it.mathworks.com/discovery/kalman-filter.html.

[63] Ultralytics. "kalman filter (kf) - glossary," ultralytics,. https://www.ultralytics.com/it/glossary/kalman-filter-kf.

[64] MathWorks. "low-pass filter - matlab & simulink," mathworks,. https://it.mathworks.com/discovery/low-pass-filter.html.

[65] "Guide to FFT Analysis. " dewesoft,. https://dewesoft.com/blog/guide-to-fft-analysis.

[66] "Fast Fourier Transform (FFT) – Basics. " nti audio,. https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft.

[67] D. Tebaldi. "frequency-division control for parallel hybrid electric vehicles," in proc. 2024 ieee int. conf. on automation science and engineering (case), bari, italy,, 2024.

[68] A. Giammarino. J. m. gandarias, p. balatti, m. leonori, m. lorenzini, and a. ajoudani, "super-man: Supernumerary robotic bodies for physical assistance in human–robot conjoined actions,". https://arxiv.org/abs/2201.06365.

[69] S. G. Hart. Nasa task load index (tlx): Paper and pencil package, version 1.0, nasa ames research center,. https://ntrs.nasa.gov/api/citations/20000021488/downloads/20000021488.pd.

[70] SKYbrary Aviation Safety. "nasa task load index (nasa tlx)," skybrary,. https://skybrary.aero/articles/nasa-task-load-index-nasa-tlx.

Elenco delle figure

Elenco delle tabelle

Elenco dei comandi