Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria Informatica

Sviluppo di una GUI per il controllo di un esoscheletro assistivo per il cammino

Impostazione parametri e connessione con ROS

Relatore

Dr. Stefano Tortora

Università di Padova

Laureando

Cristian Bassotto

Matricola 2000169

Correlatore

Dr. Edoardo Trombin

Università di Padova

Anno Accademico 2022/2023

Data di Laurea 21.07.2023

Ai miei genitori
che mi hanno sempre spronato a dare il massimo

A Sara
che mi supporta e mi sopporta

Sommario

L'avanzamento tecnologico ha portato ad importanti progressi nel campo della robotica. In particolare, gli esoscheletri robotici si sono rivelati dispositivi di grande rilevanza in vari ambiti. Nel settore medico, purtroppo, i costi di produzione dei robot sono elevati e il loro controllo è spesso affidato a tecnici esperti.

Il presente documento illustra il processo di sviluppo di un'interfaccia grafica per il controllo di ALICE, un esoscheletro assistivo a basso costo per il cammino. Ho realizzato la GUI con lo scopo di fornire uno strumento facile ed intuitivo per gli specialisti del settore medico, come i fisioterapisti. L'obiettivo principale è consentire ai professionisti di concentrarsi sul paziente e sulla scelta del trattamento, anziché sull'utilizzo complesso dell'esoscheletro.

Ho implementato l'interfaccia in C++ con l'ausilio delle librerie Qt per lo sviluppo della grafica e la connessione con Robot Operating System (ROS).Il software offre un'interfaccia grafica intuitiva e user-friendly, che consente agli operatori di controllare il robot in modo efficace e sicuro. L'attenzione principale è stata posta sulla progettazione e l'implementazione delle schermate di movimento, che permettono di gestire le diverse modalità di camminata e regolarne i parametri. L'utilizzo del middleware ROS ha permesso una comunicazione affidabile e bidirezionale tra il software e l'esoscheletro attraverso servizi.

Al termine, ho condotto alcuni test in laboratorio per valutare l'efficacia e l'efficienza del sistema. I risultati hanno dimostrato che le schermate di movimento consentono un controllo preciso e reattivo di ALICE, migliorando l'esperienza dell'utente.

La tesi evidenzia anche la possibilità di ampliare il software con l'aggiunta di nuove modalità di allenamento. L'utilizzo di ROS come framework di comunicazione apre prospettive per lo sviluppo di funzionalità avanzate e personalizzate, che possono essere integrate nel sistema esistente per soddisfare le esigenze specifiche degli utenti.

Indice

Elenco delle Figure

Elenco degli Acronimi

CSV Comma Separated Values

GUI Graphical User Interface

std C++ Standard Library

SDI Single Document Interface

MDI Multiple Document Interface

IDE Ambiente di Sviluppo Integrato

ROS Robot Operating System

SEA Attuatori Elastici in Serie

Hardiman I Human Augmentation Research and Development Investigation and MA-Nipulator I

BLEEX Berkeley Lower Extremity Exoskeleton

DARPA Defense Advanced Research Project Agency

ALICE Assistive lower LImb-Controlled Exoskeleton

WYSIWYG What You See Is What You Get

DDS Data Distribution Service

1 Introduzione

L'avanzamento tecnologico ha innescato significativi progressi nel campo della robotica, aprendo nuove prospettive e possibilità in vari settori. Tra le molte innovazioni che hanno suscitato un notevole interesse negli ultimi anni, gli esoscheletri robotici sono emersi come una tecnologia di grande successo e rilevanza.

In questo capitolo si propone un *excursus* sull'origine e il funzionamento degli esoscheletri, gli ambiti di utilizzo e come questa tecnologia sia stata utilizzata all'interno di questo specifico progetto.

1.1 Esoscheletri robotici e stato dell'arte

Gli esoscheletri robotici, noti anche come esoscheletri indossabili [1], sono dispositivi tecnologici progettati per aumentare, integrare o sostituire la funzione degli arti di chi li indossa. Possono essere definiti come strutture robotiche elettromeccaniche motorizzate che esercitano una quantità adeguata di coppia/forza sull'articolazione dell'arto per ottenere un movimento base o potenziato [2]. L'esoscheletro è progettato seguendo la struttura antropomorfa e simulando l'andatura umana. Lo strumento si muove parallelamente a chi lo indossa e può essere azionato in modo attivo o passivo, addizionando alla forza impressa dall'utente quella necessaria per compiere il movimento.

Gli esoscheletri robotici non sono solo dispositivi progettati per fornire assistenza alle persone, ma hanno un ruolo più ampio e diversificato nell'ambito della robotica. Oltre ad aiutare le persone con limitazioni motorie a svolgere attività quotidiane, stanno trovando applicazione in settori come la riabilitazione, l'industria manifatturiera [3] e l'industria bellica. Essi offrono la possibilità di potenziare le capacità umane, aumentare

la forza e la resistenza fisica, migliorare la precisione dei movimenti e ridurre il rischio di lesioni. Inoltre, gli esoscheletri sono oggetto di ricerca e sviluppo continuo per migliorarne le prestazioni, la sicurezza e l'ergonomia. L'evoluzione della tecnologia degli esoscheletri robotici promette di aprire nuove prospettive e sfide affascinanti nel campo della robotica e dell'interazione uomo-macchina.

1.1.1 Composizione

Attualmente gli esoscheletri sono utilizzati per le funzioni più disparate seppur mantenendo principi e progetti simili. La maggior parte degli esoscheletri, infatti, sono composti da una struttura meccanica, sensori, attuatori e controller [2] [4].

- La **struttura meccanica** costituisce l'ossatura portante dell'esoscheletro e può variare in base al design specifico. È progettata per fornire supporto strutturale e stabilizzare le parti del corpo che richiedono assistenza o potenziamento. La struttura meccanica può essere realizzata utilizzando materiali resistenti come leghe metalliche o compositi leggeri a seconda delle esigenze di resistenza e leggerezza del dispositivo.

- I **sensori** sono elementi cruciali che consentono all'esoscheletro di percepire l'ambiente circostante e rilevare i movimenti e le azioni dell'utente. I tipi comuni di sensori impiegati includono giroscopi, accelerometri, sensori di forza e di pressione. Questi sensori permettono al sistema di ottenere informazioni precise e in tempo reale sulla posizione, l'orientamento e la forza applicata dal corpo umano o da altre fonti esterne.

- Gli **attuatori** sono responsabili della generazione del movimento e della trasmissione di forza all'interno dell'esoscheletro. Possono essere motori elettrici, attuatori pneumatici o idraulici, o altri dispositivi in grado di convertire l'energia elettrica o fluida in movimento meccanico. Gli attuatori consentono al dispositivo di fornire assistenza, resistenza o potenziamento alle azioni muscolari dell'utente, migliorando così la forza o l'efficienza del movimento.

- Il **controller** è l'elemento centrale che coordina il funzionamento dell'esoscheletro. Comprende algoritmi e software che elaborano i dati provenienti dai sensori e determinano le risposte degli attuatori in base alle informazioni raccolte. Il sistema di controllo può essere programmato per rispondere in modo predeterminato a determinati stimoli o può adattarsi dinamicamente alle esigenze e alle azioni dell'utente. Ciò consente di personalizzare l'esperienza e ottimizzare le prestazioni dell'esoscheletro in base alle specifiche necessità dell'utente.

1.1.2 Storia ed evoluzione

Il modello più antico di esoscheletro robotico risale al 1890, ideato dell'ingegnere russo Nicholas Yagn[5].

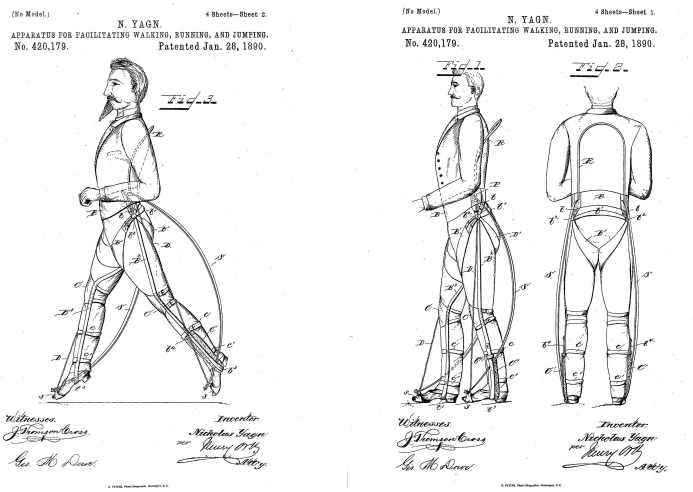

Figura 1.1: Modello concettuale dell'esoscheletro N. Yagn [5]

Tale invenzione aveva uno scopo puramente assistivo: l'esoscheletro consisteva in un lungo arco che funzionava parallelamente alle gambe dell'utente in modo che quest'ultimo fosse facilitato nella deambulazione. Il modello astratto di Yagn, prese forma nel 1965, quando la General Electric Company avviò lo sviluppo dell'esoscheletro Human Augmentation Research and Development Investigation and MANipulator I (Hardiman I) all'interno di un programma militare-navale. Il primo esoscheletro si presentava come un telaio in alluminio e acciaio con attuatori per il movimento e batterie per il funzionamento. L'architettura pesava all'incirca 680 kg [2], per cui risultava nel complesso scomoda e goffa con un movimento impreciso e inefficace. L'azienda si impegnò per migliorare questo progetto producendo esoscheletri più leggeri e modificandone il design.

Con il modello pionieristico di Nicholas Yang che prese forma sia con l'invenzione di Hardiman I che, negli stessi anni e con un progetto simile, da Neil J. Mizen, gli anni '60 segnarono una svolta nello sviluppo degli esoscheletri. A partire da queste basi, l'intera elite scientifica globale si dedicò al miglioramento dell'efficacia dell'esoscheletro. Basterà poi aspettare la metà degli anni '70 per la presentazione del primo esoscheletro per la deambulazione ideato per un impiego assistivo. Lo scienziato serbo Miomir Vukobratovic ampliò così in modo notevole l'utilizzo e il potenziale di questa nuova tecnologia. A differenza del modello militare, quello civile presentava un peso drasticamente inferiore, grazie anche alla minor richiesta energetica. Dal punto di vista puramente sanitario, fu una rivoluzione perché proponeva un futuro, e quindi una speranza, per molti casi di disabilità motoria.

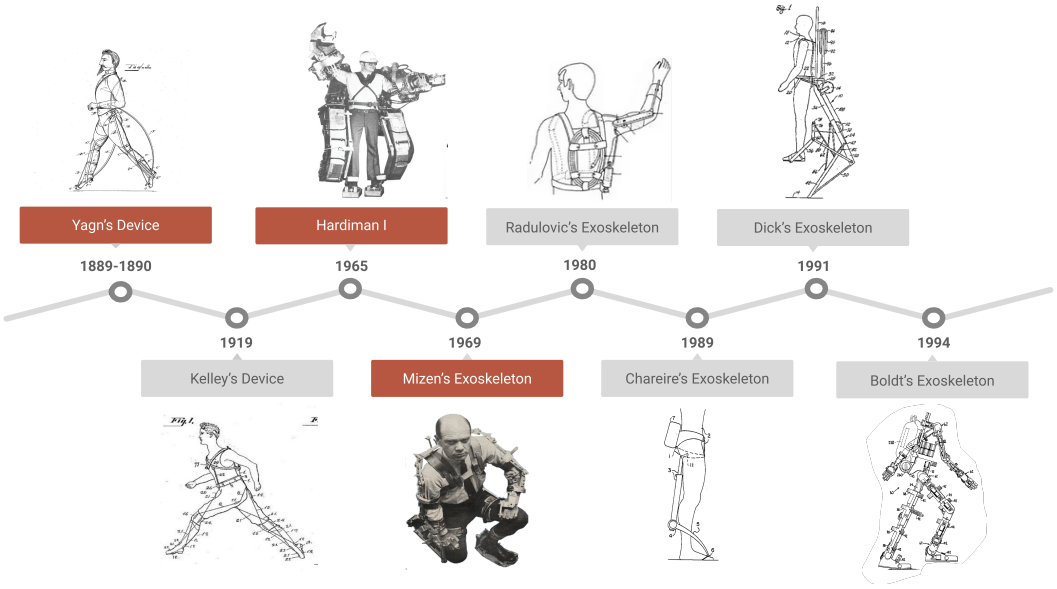

Figura 1.2: Cronologia degli Esoscheletri del Novecento [6]

Grazie a questi modelli di avanguardia e alle menti visionarie che li crearono, il ventennio successivo popolò di progetti significativi che proponevano migliorie sui prototipi pre-esistenti. Gli ambiti di sviluppo erano l'incremento della forza, l'alleggerimento del peso strutturale, l'indipendenza energetica e una maggiore durata dell'allenamento. Si sviluppò così una vasta gamma di esoscheletri ognuno specifico per criteri diversi, come il distretto corporeo, la funzione o il sistema meccanico caratterizzante.

Il nuovo secolo si aprì con il progetto Berkeley Lower Extremity Exoskeleton (BLEEX) [7] . Un esoscheletro per gli arti inferiori ideato dallo Human Engineering and Robotics Laboratory dell'Università della California, Berkeley, e finanziato dalla Defense Advanced Research Project Agency (DARPA). Questo prototipo ad esclusivo uso militare

Figura 1.3: Esoscheletro BLEEX [7]

era energeticamente autonomo e forniva maggiore resistenza e forza nel trasporto dei carichi. Era evidente come lo sviluppo dei computer e dei software avesse incentivato molto la ricerca nel campo degli esoscheletri. Sicuramente la maggior facilità di reperimento di informazioni sulla rete e la libera circolazione delle risorse rese più facile la cooperazione e il rimodellamento delle componenti robotiche. Il modello BLEEX fu particolarmente apprezzato e ciò portò ad un incentivo sulla ricerca, sostenuta economicamente da aziende militari capaci di sostenere anche gravosi costi finanziari. Le innovazioni vennero poi esportate nel campo civile. A farsene carico in questo ambito furono varie aziende, tra cui la Ekso Bionics che ancora oggi partecipa attivamente e competitivamente nel rifornire gli ambienti sanitari di esoscheletri tecnologicamente avanzati.

Il XXI secolo segnò dunque un ulteriore passo avanti, soprattutto nel campo della progettazione degli esoscheletri, consentendo la possibilità di creare modelli personalizzabili sviluppati ad hoc sulla specifica patologia o sul movimento dell'utente.

Come ne è testimonianza la Figura 1.4, dal 2010 in poi, le pubblicazioni, i libri, gli articoli e le conferenze in cui vengono esposti e discussi argomenti legati alla formazione e al funzionamento di nuovi esoscheletri, sono aumentate esponenzialmente. Questo mostra la tendenza, sempre crescente, alla creazione e implementazione di nuovi esoscheletri per aumentarne l'utilizzo e l'area di applicazione.

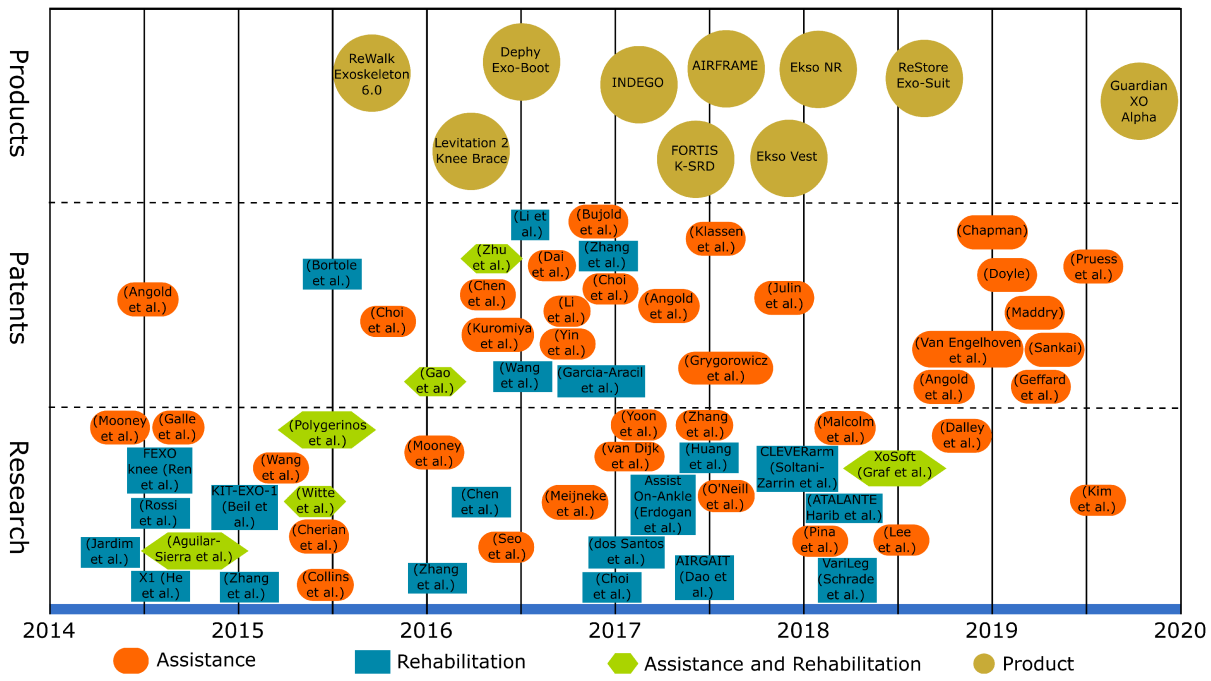

Figura 1.4: Cronologia delle ricerche sugli esoscheletri dal 2014 al 2020 [6]

Risulta necessario dunque creare dei criteri per la classificazione di questi nuovi esoscheletri.

1.1.3 Classificazioni

La categorizzazione generale suggerisce diverse tipologie di esoscheletri realizzabili e ottimizza la comprensione delle capacità dell'esoscheletro stesso. Si tratta di una classificazione gerarchica che comprende classi generali che si dividono in sottoclassi [6]. Come si può vedere dalla Figura 1.5, i criteri più ampi per la categorizzazione sono: la parte del corpo su cui si concentra, la struttura, l'azione, la tecnologia di alimentazione, lo scopo e l'area di applicazione.

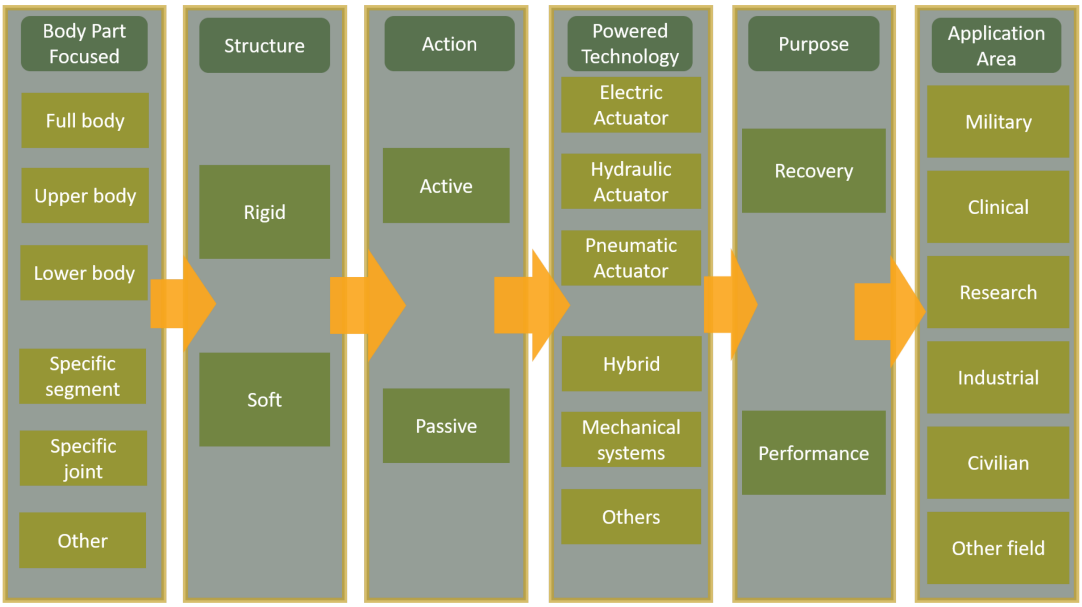

Figura 1.5: Classificazione generale per esoscheletri [6]

Si mostrano ora le varie classificazioni con relativa percentuale di prototipi per classe. Le Figure nelle prossime sezioni sono rielaborazioni di dati ricavati nell'articolo [6] su di un insieme di 75 esoscheletri presi in esame.

Classificazione per parti del corpo

Un esoscheletro può essere progettato seguendo l'intera figura antropomorfa o il singolo distretto corporeo. Ciò significa che è possibile costruire architetture robotiche adattive che assistano il movimento della struttura corporea completa fino a specializzarsi anche ad una singola mano. Questa categoria intende rappresentare ogni parte del corpo su cui può essere utilizzato un esoscheletro. Generalmente ci si riferisce a quale parte del corpo l'esoscheletro è progettato per supportare.

In termini generali, si presenta quelle che sono le sotto-categorie principali di questa classificazione:

- **Corpo intero**: esoscheletri pensati per assistere tutti gli arti o la maggior parte del corpo;
- **Arti superiori**: esoscheletri realizzati per gli arti superiori e che coinvolgono il torace, la testa, la schiena e/o le spalle;
- **Arti Inferiori**: esoscheletri realizzati per gli arti inferiori e che coinvolgono cosce, gambe e/o fianchi;

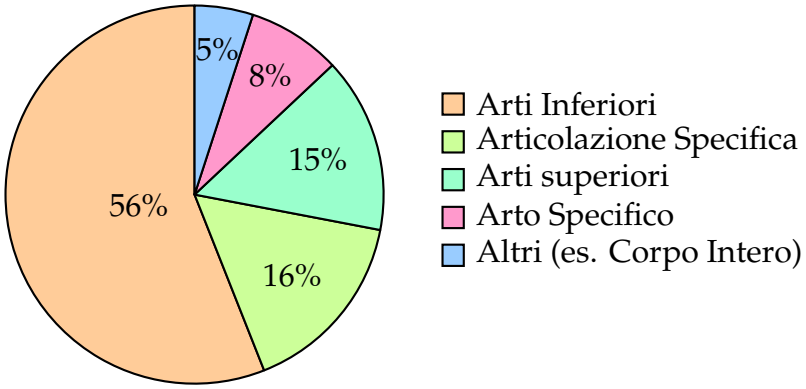

Figura 1.6: Grafico della classificazione in base alle parti del corpo

Notiamo facilmente che la maggior parte degli esoscheletri in esame sono per gli arti inferiori. Il motivo è dovuto ai notevoli benefici per la mobilità che avrà un impatto decisamente rilevante sulla qualità della vita delle persone con disabilità e allo stesso modo , in alcuni settori lavorativi, al sollevamento di pesi o altre attività che richiederanno un sostegno aggiuntivo della parte inferiore rispetto a quella superiore del corpo. Inoltre un ulteriore motivo è la maggiore complessità tecnica associata alla progettazione e implementazi

Tuttavia, gli esoscheletri non sono progettati solo per il corpo intero, superiore o inferiore. Possono essere progettati per adattarsi a una parte specifica del corpo. Ogni gruppo quindi contiene anche gli esoscheletri che si focalizzano su una più specifica articolazione o distretto. Ad esempio, gli esoscheletri specifici per il ginocchio, la caviglia e il piede possono essere racchiusi come un sottogruppo specializzato degli esoscheletri lower body.

Classificazione strutturale

La definizione di esoscheletro secondo la struttura ha portato alla divisione di due classi: esoscheletri rigidi e esoscheletri morbidi.

Quelli **rigidi** sono costituiti da componenti strutturali di materiale duro che si attaccano al corpo dell'utente per fornire principalmente supporto stabile. I materiali in questione sono metalli, plastiche, fibre, ecc.

Le exo-tute, o esoscheletri **morbidi**, sono realizzati invece con materiali elastici di tipo tessile, progettati per consentire un movimento più libero al corpo dell'utente. A differenza degli exoscheletri rigidi, le exo-tute offrono una maggiore flessibilità e adattabilità grazie alla natura dei materiali utilizzati.

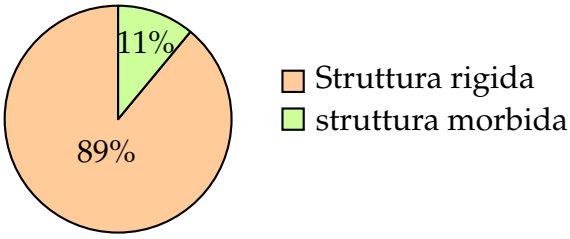

Figura 1.7: Grafico della classificazione in base alla struttura

Classificazione d'azione

Con il termina azione si descrive il tipo di aiuto che l'esoscheletro fornisce all'utente. Si identificano due categorie di esoscheletri: attivi e passivi.

I primi comprendono i modelli che si sostituiscono all'utente, cioè che eseguono il movimento senza che ci lo indossa debba fornire energia o input. La classe **passiva** invece accompagna il movimento dell'utente, cioè è necessario un input iniziate che dia l'incipit al moto.

In altre parole, gli esoscheletri **attivi** hanno bisogno di una fonte esterna di energia perché compiono il movimento; quelli passivi non hanno bisogno di fonti energetiche perché il loro ruolo è la facilitazione del movimento che però, di per sé, é compiuto dall'utente.

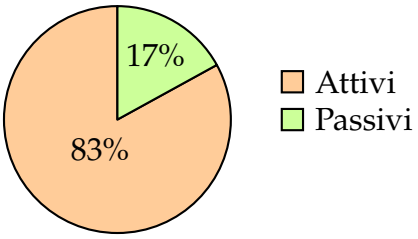

Figura 1.8: Grafico della classificazione d'azione

Classificazione per tecnologia di alimentazione

Questa categoria si ricollega con la precedente legata all'azione. Nello specifico, si identificano: una classe di azione attiva che comprende gli attuatori elettrici, idraulici e pneumatici; una classe di azione passiva che raccoglie i sistemi meccanici; in più, si distingue una classe specifica per l'ibrido e una per raccogliere altre tecnologie non

comuni. Analizzando la classe attiva, gli attuatori **elettrici** comprendono qualsiasi tipo di motore elettrico utile per attivare i movimenti dell'esoscheletro. Analogamente, gli attuatori **idraulici** e **pneumatici** comprendono pistoni e attuatori morbidi. La classe passiva di sistemi **meccanici** comprende i meccanismi usati per l'immagazzinamento energetico meccanico, come le molle, e per la trasmissione, come gli ingranaggi.

Si inseriscono nella classe **ibrida** tutti gli esoscheletri che sfruttano più di un metodo. Un esempio molto utilizzato sono i Attuatori Elastici in Serie (SEA), combinazione di attuatori elettrici e sistemi meccanici.

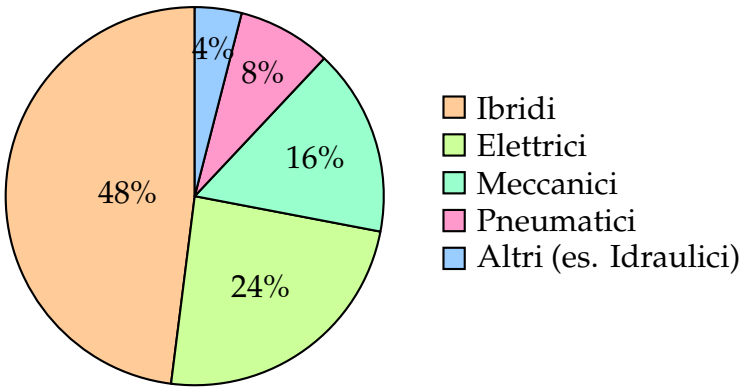

Figura 1.9: Grafico della classificazione per alimentazione

Classificazione funzionale o di scopo

Questa classe definisce l'uso che si farà dell'esoscheletro e comprende due tipologie soltanto: di recupero o da prestazione.

I primi sono sono progettati per fornire supporto e assistenza alle persone che hanno subito lesioni o che presentano limitazioni fisiche. Questi dispositivi sono pensati per aiutare nella riabilitazione (il ripristino con mezzi terapeutici di una migliore condizione fisica), promuovendo il recupero funzionale. Gli esoscheletri di **recupero** possono fornire sostegno alle articolazioni, migliorare la mobilità e consentire a chi li indossa di svolgere attività quotidiane in modo più agevole. Rappresentano quindi un'innovativa soluzione per migliorare la qualità della vita delle persone con disabilità.

D'altra parte, gli esoscheletri da **prestazione** sono progettati per migliorare le capacità fisiche delle persone senza necessariamente compensare una disabilità o una lesione. Questi dispositivi sono spesso utilizzati nel contesto sportivo o in ambiti lavorativi che richiedono uno sforzo fisico particolare. Possono quindi aumentare la forza, l'agilità o l'endurance, consentendo agli individui di raggiungere livelli superiori di performance in determinate attività.

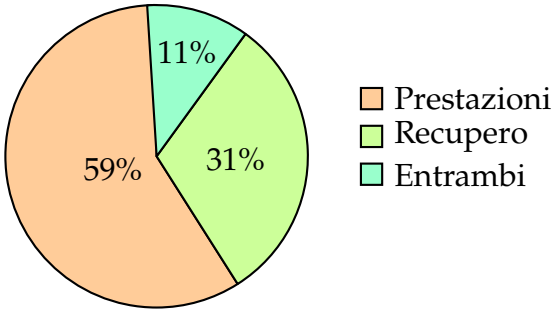

Figura 1.10: Grafico della classificazione funzionale o di scopo

Figura 1.11: Esempio di esoscheletro industriale Guardian Xo [1]

Classificazione per ambito di applicazione

L'esoscheletro, come visto anche nell'excursus storico, può essere realizzato con diversi fini utilitaristici.

Gli esoscheletri per le applicazioni **militari** sono utilizzati per potenziare la forza fisica e la resistenza dei soldati. L'allenamento consiste nel trasportare carichi per tempi lunghi, abbattere il costo metabolico di locomozione in modo da addestrare soldati più agili, meno inclini all'affaticamento e più prestanti.

Gli esoscheletri per le applicazioni non militari sono coinvolti nella branca medica, industriale, civile e di **ricerca**. Quest'ultima ovviamente comprende quelli in fase di sviluppo, che essendo in crescita risulta anche il campo maggiore.

In campo **sanitario**, gli esoscheletri sono coinvolti in attività cliniche e, spesso, sono utilizzati per il recupero: si interfacciano con pazienti target affetti da disabilità fisiche causate da infortuni sportivi, lesioni del midollo spinale o complicazioni dovute a incidenti vascolari cerebrali (ictus). Lo scopo consiste nell'ottimizzare il tempo e la qualità del recupero del movimento.

Nel campo **industriale** (Figura 1.11) sono compresi esoscheletri utilizzati da persone

[1] https://www.sarcos.com/products/guardian-xo-powered-exoskeleton/

senza alcuna patologia che svolgono un'azione di assistenza per evitare danni fisici. Questa caratteristica varia anche per gli esoscheletri militari che, come detto, fungono da potenziamento, qui anche da protezione, rispetto ad azioni svolte da utenti sani.

Infine, per esoscheletri **civili**, si intendono macchinari utili in casa o in spazi pubblici per supportare lavori fisicamente impegnativi

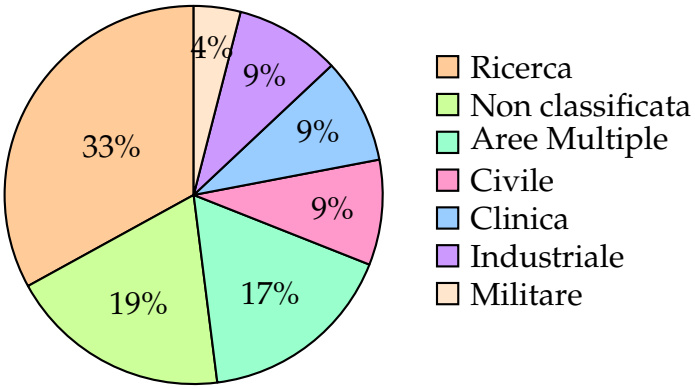

Figura 1.12: Grafico della classificazione per ambito di applicazione

1.2 Progetto ALICE ed Esoscheletro

Il campo della robotica riabilitativa ha mostrato un significativo sviluppo negli ultimi 15 anni, con una crescente richiesta di servizi di riabilitazione che supera le risorse disponibili. Questa discrepanza ha portato a una riduzione della durata delle sessioni di riabilitazione e a liste d'attesa sempre più lunghe, esercitando una pressione significativa sui fisioterapisti. Al fine di affrontare questa sfida, il progetto ALICE [8] si propone di fornire agli operatori medici uno strumento avanzato: **un esoscheletro per la riabilitazione degli arti inferiori**.

Il progetto ALICE è stato sviluppato da INDI [9], un'azienda messicana che si dedica alla progettazione di esoscheletri per aiutare le persone con disabilità motorie alle gambe a recuperare la mobilità. Il team di INDI ha reso open-source il proprio lavoro e si è trasferito in Francia per promuovere un impatto più ampio a livello globale.

L'esoscheletro utilizzato in questo studio è l'Assistive lower LImb-Controlled Exoskeleton (ALICE) disponibile presso lo IAS-Lab dell'Università di Padova. ALICE è un esoscheletro assistivo sviluppato principalmente con parti e materiali open source prodotti in alluminio o tramite stampa 3D, che mira ad abbattere i costi (il costo totale di tutti i componenti è di circa 1500 dollari) per rendere la tecnologia accessibile al maggior numero possibile di persone. Tramite questo progetto di esoscheletro low-cost si vuole quindi cercare di abbattere i costi massicci degli esoscheletri al momento in

commercio (in media quasi un milione di euro ad esoscheletro) per facilitarne l'uso anche per i paesi in via di sviluppo.

L'esoscheletro presenta tre bracci (anca, femore e tibia) e comprende quattro articolazioni attive (due alle anche e due alle ginocchia), azionate da motori a corrente continua spazzolati e da un riduttore a vite. Ogni unità di attuazione fornisce una coppia massima di 26 Nm e una coppia nominale di circa 10 Nm; la tensione nominale è di 12 V e il consumo di energia è di 120 W nominali e 260 W di picco. Ogni giunto è dotato di un encoder lineare per la misurazione della posizione del giunto in tempo reale. L'unità di controllo di basso livello (Arduino Mega 2650 R3), il controllo di alto livello (miniPC) e l'alimentazione sono posizionati fuori bordo per ridurre il peso complessivo indossato dal soggetto, e sono collegati all'esoscheletro attraverso un fascio di cavi di 15 metri. La lunghezza dei collegamenti e le dimensioni del telaio del bacino sono state regolate manualmente da un operatore esperto per adattarsi alle caratteristiche corporee di ciascun soggetto; è regolabile con lunghezze di femore e tibia comprese tra 35 e 50 cm e una larghezza pelvica da 29 a 40 cm. Come è facilmente constatabile, è un esoscheletro progettato ad uso pediatrico con barre di lunghezza ridotta. Per i test di laboratorio questa versione pediatrica è stata scalata per adulti per poter svolgere agevolmente gli esperimenti.

Il profilo di andatura dell'esoscheletro può essere attivato dai partecipanti tentando un passo con la gamba desiderata. L'esoscheletro fornirà quindi una potenza aggiuntiva, integrando la forza delle gambe dell'utente necessaria per completare la traiettoria del passo. Offre un duplice vantaggio: da un lato, supporta i fisioterapisti nel processo riabilitativo, riducendo il loro carico di lavoro; dall'altro, funge da sensore avanzato, fornendo informazioni cruciali sui movimenti del paziente e contribuendo a migliorare la diagnosi e la prognosi di specifiche lesioni. Tuttavia, va sottolineato presenta alcune limitazioni, come la mancanza di movimento attivo alla caviglia, che ne limitano l'utilizzo solo a determinate tipologie di lesioni.

L'esoscheletro è progettato per assistere nella diagnosi del modello di deambulazione e misurare oggettivamente il recupero del paziente nella riabilitazione della deambulazione.

Attualmente, il progetto è nel processo di convalida dei componenti/sottosistemi e di esecuzione di test di laboratorio e simulazioni in un ambiente reale. Se il costo non è più quindi un problema per il suo inserimento negli ospedali, lo è la difficile interazione coi fisioterapisti causata dall'attuale necessità di utilizzo del terminale per il suo azionamento.

[2] `http://www.indi.global/alice`

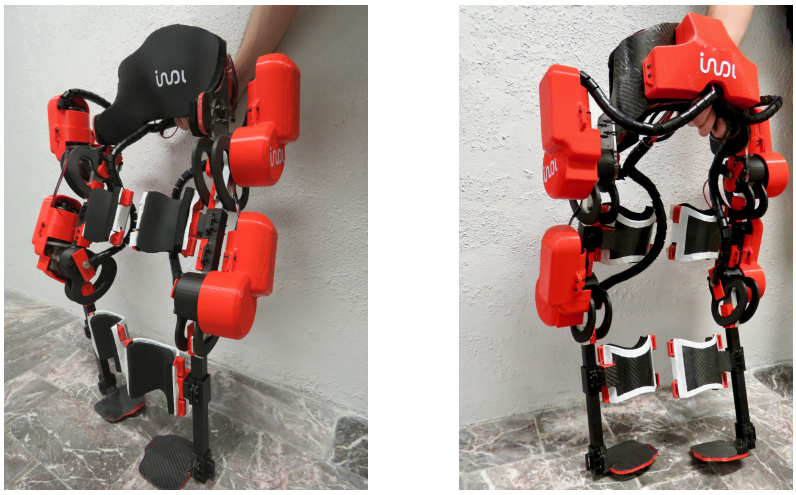

Figura 1.13: ALICE² per uso infantile: visuale frontale a sinistra e posteriore a destra [9].

Altri ostacoli per estendere l'uso dell'esoscheletro nelle cliniche risiedono (a) nella mancanza di credibilità e accettazione sia da parte del personale di riabilitazione che dei pazienti, e (b) nei requisiti normativi in quanto, essendo una macchina di diagnosi e riabilitazione, è necessario sottoporsi ai necessari studi clinici.

L'obiettivo futuro principale di INDI è quello di migliorare la vita delle persone con disabilità motorie, rendendo le soluzioni più accessibili ed economiche. Attraverso il loro lavoro, INDI ha già ottenuto successi significativi, come nel caso di Shawn, un ragazzo che ha riacquistato la capacità di camminare grazie all'uso dell'esoscheletro Alice. L'azienda mira anche a espandere l'uso degli esoscheletri in diversi contesti, come la danza, per promuovere l'inclusione delle persone con disabilità in varie attività.

1.3 Scopo e struttura della tesi

In questa tesi, ho proposto e sviluppato un'applicazione grafica[3] facile ed intuitiva per gli specialisti del settore medico, come i fisioterapisti, cosicché, durante la riabilitazione, possano concentrarsi sul paziente e sul miglior trattamento da applicare piuttosto che sull'utilizzo dell'esoscheletro.

Descriverò prima i linguaggi e le librerie da me utilizzati, per una migliore comprensione delle scelte implementative (capitolo 2), per poi focalizzarmi sul design dell'applicazione e le sue funzioni principali (sia in front-end che back-end) in prospettiva di un futuro progetto agile (approccio di sviluppo collaborativo e iterativo che si basa sulla flessibilità, adattabilità e consegna incrementale del software) (capitolo 3).

[3] https://github.com/cristianbass01/GUI_robotic_exoskeleton

In particolare, l'applicazione è specifica per l'esoscheletro ALICE perché è stata sviluppata in collaborazione con INDI Ingénierie et Design. L'obiettivo della loro cooperazione con l'Università di Padova era quello di adattare l'interfaccia da un sistema basato su terminale a una soluzione intuitiva basata su interfaccia grafica. Nonostante questa specificità, è possibile adattare l'applicazione ad un qualsiasi esoscheletro in commercio, purché implementato con Robot Operating System (ROS), grazie alla comunicazione integrata tra interfaccia e sistema operativo tramite servizi. Tuttavia, ROS risulta essere una forte novità, recentemente introdotta dal gruppo di ricerca dell'Università di Padova, per applicazioni di robotica medica e neuro-robotica. Infatti, fino ad oggi lo standard di questo sistema operativo rimaneva la robotica di servizio e la robotica industriale.

Nell'ultimo capitolo di questa tesi, andrò ad effettuare dei test qualitativi per valutarne la fluidità ed intuitività e a descrivere eventuali ulteriori implementazioni future (capitolo 4).

2

Materiali

In questo capitolo saranno analizzati nel dettaglio il linguaggio di programmazione utilizzato e le rispettive librerie per l'implementazione della Graphical User Interface (GUI) e per l'interazione di quest'ultima con l'esoscheletro.

2.1 Strumenti utilizzati per la Grafica

Per l'implementazione della GUI è stato scelto il linguaggio C++ integrato con la libreria esterna Qt.

2.1.1 Linguaggio C++

C++ [10] è un potente linguaggio di programmazione general-purpose che combina caratteristiche ad alto livello e ad alto rendimento. Nato negli anni '80 come un'evoluzione del linguaggio C, è diventato uno dei linguaggi più diffusi e influenti nell'industria del software. Un aspetto che sicuramente ha giovato al suo ampio impiego è l'implementazione di una vasta gamma di librerie standard che offrono funzioni predefinite per svolgere attività comuni, permettendo agli sviluppatori di focalizzarsi sulla logica del proprio programma.

Una delle principali caratteristiche di C++ è la sua capacità di fornire un controllo di basso livello sulla gestione della memoria, consentendo di scrivere codice efficiente e ottimizzato per le risorse. Per questo, risulta la miglior scelta da adottare per lo sviluppo di un'interfaccia performante, basata sullo scambio di informazioni attraverso puntatori ad oggetti.

Puntatori ad oggetti

I puntatori rivestono un ruolo cruciale nella gestione della memoria e nell'accesso ai dati in C++.

Un puntatore è una variabile speciale che contiene l'indirizzo di memoria di un'altra variabile. Non conserva direttamente il valore effettivo, bensì l'indirizzo di memoria che consente l'accesso al valore desiderato. L'utilizzo dei puntatori risulta particolarmente utile quando si desidera manipolare o passare grandi quantità di dati senza eseguirne una copia fisica. Questa capacità dei puntatori è stata ampiamente sfruttata nella creazione dell'interfaccia grafica, soprattutto per manipolare gli oggetti presenti nelle varie pagine senza doverli reinserire nei diversi contesti. In rari casi, a causa di alcuni casting statici per la manipolazione degli oggetti derivati, il reinserimento si è reso necessario (sezione 2.1.2).
Spesso sono stati utilizzati proprio i puntatori per passare i parametri per riferimento alle funzioni e salvarli nelle classi per rendere fosse possibile la manipolazione diretta degli oggetti e la restituzione dei risultati modificati.

Quando possibile, sono stati utilizzati degli Smart Pointer [11], ovvero dei puntatori intelligenti che gestiscono automaticamente la deallocazione della memoria quando non più necessaria, fornendo un'implementazione più sicura e meno soggetta a errori nella gestione della memoria stessa. In particolare, sono stati utilizzati dei puntatori condivisi (Shared Pointer). Ogni Shared Pointer tiene traccia del numero di riferimenti che puntano all'oggetto condiviso.

Un aspetto importante degli Shared Smart Pointer è che consentono la condivisione di un oggetto tra più puntatori senza il rischio di accessi non validi o di deallocazione prematura della memoria. Quando un nuovo Shared Pointer viene creato e assegnato a un oggetto già condiviso, il conteggio dei riferimenti aumenta. Allo stesso modo, quando l'oggetto puntato esce dal proprio scope o viene esplicitamente distrutto, il conteggio dei riferimenti diminuisce. Quando il conteggio dei riferimenti raggiunge zero, l'oggetto viene deallocato. Questo meccanismo di conteggio dei riferimenti garantisce che l'oggetto condiviso rimanga in vita finché esistono puntatori riferiti ad esso.

Un vantaggio significativo di questi oggetti è che riducono la probabilità di errori nella gestione della memoria rispetto all'allocazione e deallocazione manuale tramite *new* e *delete*. Ciò porta a un codice più sicuro e robusto, contribuendo a migliorare l'affidabilità e la mantenibilità del software, tuttavia, non sempre è possibile il loro utilizzo.

Quando si gestiscono oggetti grafici Qt (sezione 2.1.2), non è possibile salvare puntatori all'interno degli Smart Pointer perchè potrebbero cercare di deallocare la memoria

degli oggetti puntati alla chiusura dell'applicazione, compito affidato alle classi interne della libreria Qt. Analogamente accade per gli oggetti *List* della C++ Standard Library (std) che gestiscono in modo autonomo la memoria e che quindi non necessitano di alcuna deallocazione, ma di cui è possibile eseguire il passaggio per riferimento usando un puntatore che non dovrà poi essere distrutto nemmeno se salvato in una variabile puntatore all'interno della classe dell'oggetto.

Design Pattern

Attraverso C++ è stato anche possibile ottimizzare la gestione del codice rendendolo riutilizzabile e mantenibile in prospettiva ipotetica di una futura cooperazione con altri team in un progetto agile attraverso l'utilizzo di alcuni design pattern [11].

I design pattern sono soluzioni comprovate e generiche per problemi ricorrenti nello sviluppo del software. Essi offrono un approccio strutturato per progettare e organizzare il codice, consentendo di affrontare in modo efficace e efficiente situazioni complesse.

Nel caso specifico, sono stati utilizzati due particolari design pattern:

- Il **Singleton** è un design pattern creazionale che garantisce l'esistenza di una sola istanza di una classe e fornisce un punto di accesso globale per accedere a tale istanza. Questo pattern è stato utilizzato nella classe responsabile della connessione con l'esoscheletro per limitarne il numero di istanze a una sola e garantire quindi che tutto il programma utilizzi la stessa istanza.
  L'implementazione del Singleton prevede l'utilizzo di un metodo statico che si occupa di restituire l'unica istanza creata. Questo metodo controlla se l'istanza è già stata creata e, in caso positivo, la restituisce. Nel caso in cui l'istanza non sia ancora stata creata, ne viene creata una nuova e successivamente viene restituita. È importante notare che il costruttore della classe Singleton è dichiarato privato, impedendo così la creazione di istanze tramite l'utilizzo dell'operatore "new" da parte di altre classi. Tuttavia, è importante notare che la condivisione di uno stato globale può comportare problemi legati alla sincronizzazione e alla dipendenza indesiderata tra le parti del sistema. Pertanto, si è reso necessario l'utilizzo di una classe particolare di Qt per la gestione della sincronizzazione dei thread all'interno dell'applicazione, come descritto nella sezione 2.1.2.

- Il **Composite** è un design pattern strutturale ampiamente utilizzato nel contesto dello sviluppo dei software per organizzare gli oggetti all'interno di una struttura ad albero e fornire un'interfaccia unificata per trattare sia gli oggetti individuali che i gruppi di oggetti. Questo pattern è stato utilizzato per gestire in modo trasparente i diversi tipi di form che possono essere inseriti prima nella pagina

principale (chiamata frame) e poi all'interno del form stesso, creando la struttura ad albero di tipo cornice-form-form (sezione 3.1).

2.1.2 Libreria Qt

La libreria Qt [12] è un framework di sviluppo software ampiamente utilizzato per la creazione di applicazioni con interfacce grafiche di alto livello. Qt fornisce un set completo di strumenti e componenti per la creazione di interfacce utente avanzate, incluse funzionalità come la gestione degli eventi, la manipolazione dei widget, la grafica 2D e 3D, la connettività di rete e l'accesso a database. Inoltre, offre una vasta gamma di funzionalità per lo sviluppo cross-platform (dalla rappresentazione dei caratteri in memoria alla creazione di un'applicazione grafica multithread) , consentendo agli sviluppatori di scrivere il codice una singola volta e distribuirlo su diverse piattaforme senza dover apportare modifiche significative. È proprio grazie all'ampia gamma di funzionalità e alla sua versatilità che è diventata uno degli strumenti più popolari per lo sviluppo di applicazioni desktop con interfacce grafiche (come quella trattata in questa tesi) .

Segnali e Slots

Un aspetto distintivo di Qt è il suo sistema di segnali e slot, che permette una comunicazione efficiente tra gli oggetti all'interno dell'applicazione. Gli oggetti possono emettere segnali per notificare altri oggetti di un evento e gli slot possono essere collegati a questi segnali per eseguire azioni di risposta specifiche. In particolare, è possibile dichiarare prototipi di segnali che potrebbero essere emessi inserendoli nella sezione *signals* della classe e poi collegarli ad un numero indefinito di slots. Allo stesso modo anche gli slots sono metodi che devono essere dichiarati nella loro apposita sezione che può essere pubblica, privata o protetta, ma il metodo sarà comunque connesso a segnali di altre classi indifferentemente. Gli slots verranno poi invocati in risposta ad un segnale; inoltre, potranno essere collegati a più segnali creando quindi una relazione molti-a-molti.

QThread

I QThread sono strumenti disponibili nella libreria Qt che permettono la disponibilità dell'applicazione anche durante l'esecuzione di funzioni lente, evitando il blocco momentaneo del programma che segue l'input, ad esempio, di un pulsante. In altre parole, i thread vengono usati per lanciare in esecuzione una funzione in modo parallelo

e autonomo rispetto all'applicazione stessa, come ad esempio per gestire la camminata dell'esoscheletro avendo la possibilità di stopparla a proprio piacimento.

Per personalizzare la loro esecuzione è necessario sovrascrivere la funzione base $run()$ chiamata dalla funzione $thread.start()$, come vedremo nel prossimo capitolo.

Schermata principale e Widget

Una schermata principale, o Main Windows, è una finestra di primo livello attorno alla quale si basa un'applicazione. Può contenere:

- **Barra dei menù**: fornisce un insieme di opzioni organizzate gerarchicamente per consentire all'utente di accedere alle diverse funzionalità dell'applicazione;
- **Barra degli strumenti**: ospita icone o pulsanti per l'esecuzione di azioni frequentemente utilizzate;
- **Barra di stato**: fornisce informazioni di stato dell'applicazione o dei componenti specifici;
- Altre aree per funzioni di supporto (non utilizzate per l'implementazione di questo progetto).

La schermata principale svolge un ruolo cruciale nel flusso di lavoro dell'applicazione in quanto rappresenta il contenitore principale del documento di lavoro, cioè la rappresentazione dei dati con cui l'utente sta lavorando, e può fungere da punto di accesso per aprire le finestre di dialogo dell'applicazione. Queste ultime sono finestre temporanee che consentono all'utente di inserire informazioni specifiche, fare scelte o eseguire operazioni aggiuntive.

Esistono due implementazioni base per arrangiare i documenti di lavoro nelle schermate principali:

- *Single Document Interface (SDI)*: ogni schermata principale corrisponde ad un documento di lavoro unico contenuto nel widget centrale;
- *Multiple Document Interface (MDI)*: la schermata principale contiene al suo interno uno spazio di lavoro che contiene a sua volta i documenti di lavoro sottoforma di schermate più piccole.

Nel progetto è stato adottato il metodo a schermata singola al cui interno vengono inseriti vari widget personalizzati dinamici.

Un widget è una finestra, o sezione di finestra, che può contenere altri oggetti grafici, noti come widget figli. Questo pannello grafico può essere visualizzato sullo schermo e può interagire con l'utente per ricevere input e reagire ad eventi.

La schermata principale è una classe derivata dalla classe widget, ma implementa funzioni aggiuntive come le barre descritte precedentemente. Widget e Main Windows hanno una serie di proprietà, come dimensioni, posizione, stile, titolo, icone, eccetera, che possono essere configurate per personalizzare l'aspetto e il comportamento del widget. Entrambi possono gestire eventi come il clic del mouse, la pressione dei tasti e le modifiche dello stato. Solitamente i widget sono organizzati in layout, che definiscono la loro posizione e il loro ridimensionamento automatico sulla schermata principale, ma possono anche essere liberamente ancorati al widget padre, questi avranno dunque dimensioni fisse.

Ci sono vari tipi diversi di widget base in Qt. Nell'applicazione ho utilizzato:

- *QPushButton*: pulsante cliccabile, se premuto emette un segnale che invoca una funzione specifica;
- *QLabel*: etichetta di testo non modificabile dall'utente ma solo dal programma, utilizzata per visualizzare testo o immagini;
- *QLineEdit*: campo di testo modificabile in cui gli utenti possono inserire del testo;
- *QCheckBox*: casella di controllo che può essere selezionata o deselezionata;
- *QGroupBox*: contenitore che raggruppa altri widget in una sezione logica o funzionale dell'interfaccia utente;
- *QSpinBox*: campo di input numerico che consente all'utente di selezionare un valore tramite un'interfaccia di incremento/decremento;
- *QProgressBar*: indicatore di avanzamento visuale;
- *QMessage*: finestra di dialogo predefinita utilizzata per mostrare messaggi all'utente, come avvisi, conferme o notifiche;
- *QDialog*: finestra di dialogo personalizzabile che può essere utilizzata per interazioni complesse con l'utente.

Oltre ai widget base implementati in Qt è possibile anche crearne di personalizzati in formato XML per implementare applicazioni più complesse o sovrascrivere funzioni virtuali della classe base QWidget o sue derivate, in modo da adattarlo alle proprie esigenze. Oltre alla struttura dei widget è possibile personalizzarne il comportamento e lo stile collegandolo a parti di codice o inserendo uno StyleSheet (in formato simile a

CSS). Inoltre è possibile inserire i propri widget personalizzati all'interno di altri widget o modificarli dinamicamente durante l'esecuzione del codice.

QtCreator e QtDesigner

Per modificare e implementare in modo più facile ed intuitivo i widget personalizzati e le loro rispettive funzioni, è possibile usare un Ambiente di Sviluppo Integrato (IDE) predefinito: Qt Creator.

QtCreator offre un'interfaccia utente intuitiva e strumenti per la scrittura come l'auto-completamento del codice, l'analisi statica e il debugger. Si tratta di una multi-piattaforma che supporta lo sviluppo su diverse architetture e sistemi operativi. Integrato a QtCreator troviamo anche QtDesigner, un'applicazione visuale che consente di progettare l'aspetto e la disposizione degli elementi dell'interfaccia utente. QtDesigner è uno strumento *What You See Is What You Get (WYSIWYG)* che permette di creare la grafica delle applicazioni Qt tramite un'interfaccia drag-and-drop, inoltre, è possibile collegare segnali e slot tra i widget per definire il comportamento interattivo dell'applicazione. Queste interfacce sono salvate in file XML con formato .ui per essere utilizzate in combinazione con QtCreator.

Nonostante le numerose imperfezioni e difficoltà applicative come ad esempio gli errori nella visualizzazione dell'interfaccia, la difficoltà di inserimento delle risorse e l'impossibilità di modificare il file XML direttamente dall'applicazione, questo IDE offre un plugin per il collegamento con ROS (sezione 2.2), il debug e l'esecuzione dell'applicativo: QtCreator ROS. Purtroppo questo plugin è offerto solo per vecchie versioni dell'IDE e quindi non è stato possibile implementare l'applicazione con l'ultima versione stabile disponibile (Qt6), ma attraverso la libreria Qt5.

2.2 Strumenti utilizzati per l'interazione con l'Esoscheletro

Le funzioni principali che l'esoscheletro può compiere sono implementate attraverso *Arduino*, che funge da controllore del sistema interfacciando sensori e attuatori consentendo di monitorare i movimenti del corpo, rilevare l'intenzione di movimento dell'utilizzatore e attivare i motori dell'esoscheletro in risposta a tali segnali. Questi algoritmi di controllo specifici dell'esoscheletro utilizzato, sono al di fuori dello scopo della presente tesi che propone invece un'applicazione per il collegamento tra un esoscheletro generico e il fisioterapista o l'operatore che ne farà uso. Per questo, ora, discuterò il middleware utilizzato per lo scambio di dati in tempo reale e la gestione dei comandi: ROS.

2.2.1 Robot Operating System (ROS)

ROS è un software libero e open-source per la robotica nato nel 2007 e utilizzato sia in applicazioni commerciali che di ricerca. È progettato per fornire un'architettura modulare e flessibile per la comunicazione tra processi, il controllo dei dispositivi e la gestione dei dati all'interno di un sistema robotico [13].

In particolare, nell'implementazione dell'applicazione, ROS è stato integrato per facilitare la comunicazione e l'interazione tra l'interfaccia grafica e il sistema robotico sottostante attraverso il passaggio di messaggi tramite servizi.

ROS file system

Il file system di ROS include pacchetti, unità atomica di ROS contenente codice sorgente, file di dati, codice compilato, dipendenze e altro ancora; metapacchetti, gruppi di pacchetti simili contenenti solo le dipendenze necessarie; manifesti dei pacchetti, file XML all'interno di un pacchetto che contiene le informazioni principali (nome, descrizione, autore, dipendenze, etc.); cartelle, collezioni di pacchetti che condividono una comune versione; tipi di messaggio e tipi di servizi personalizzati creati dal programmatore.

Attraverso alcuni semplici comandi simili a quelli utilizzati normalmente in linux (roscd, rosls), è possibile navigare all'interno del file system.

Catkin Make

Catkin è un sistema di build utilizzato per la gestione dei pacchetti che fornisce un framework di compilazione e un set di strumenti per creare, compilare e organizzare i pacchetti. Con Catkin, i pacchetti vengono organizzati in una struttura modulare basata su directory, consentendo agli sviluppatori di separare in modo ordinato i file di configurazione, il codice sorgente, le librerie e gli eseguibili, come descritto nel Frammento di Codice 2.1.

```
workspace_folder/     -- Root del workspace
  src/                -- Dir contenente i pacchetti sorgente
    CMakeLists.txt    -- File CMake principale fornito da Catkin
    package_1/        -- Primo pacchetto
      CMakeLists.txt  -- File CMake per il pacchetto 1
      package.xml     -- Manifesto del pacchetto 1
      src/            -- Dir contenente i file sorgente del pacchetto 1
      include/        -- Dir contenente i file di inclusione del pacchetto 1
      ...
    ...
```

```
11    package_n/       -- n-esimo pacchetto
12      CMakeLists.txt -- File CMake per il pacchetto n
13      package.xml    -- Manifesto del pacchetto n
14      src/           -- Dir contenente i file sorgente del pacchetto n
15      include/       -- Dir contenente i file di inclusione del pacchetto n
16      ...
17  build/             -- Dir per la compilazione dei pacchetti
18  devel/             -- Dir per l'installazione dei pacchetti compilati
19  install/           -- Dir per l'installazione finale dei pacchetti
```

Codice 2.1: Esempio di organizzazione di un workspace di catkin[1]

Catkin semplifica anche la gestione delle dipendenze consentendo di specificarle nel file di configurazione del pacchetto e gestendole automaticamente durante il processo di compilazione. Con la compilazione *out-of-source*, i file sorgenti rimangono separati dalla directory di build, mantenendo l'ambiente di sviluppo pulito e facilitando la gestione delle diverse configurazioni di compilazione. L'applicativo fornisce inoltre strumenti di build come catkin_make e catkin build, che semplificano il processo di compilazione dei pacchetti.

Versioni

Esistono due versioni principali: ROS e ROS2. Il primo è basato su una comunicazione peer-to-peer tramite il middleware ROS-1 Core, mentre il secondo utilizza il middleware Data Distribution Service (DDS) per una comunicazione più robusta. Inoltre, ROS2 offre miglioramenti in termini di scalabilità e prestazioni e per questo è molto utilizzato nell'industria nonostante non sia retrocompatibile con ROS.

Esistono poi varie sottoversioni, molte delle quali non più supportate, fanno eccezione: *ROS Noetic Ninjemys* [2], versione utilizzata nel corso di questo lavoro di tesi, e per *ROS2 Humble Hawksbill* e la nuova versione *Iron Irwini* [3].

Programmazione ad alto livello

Questo middleware è vantaggioso perché supporta i più popolari linguaggi di programmazione ad alto livello, inclusi C++, Python, Lisp e sperimentalmente anche Java, Node.js e C#. In più, mette a disposizione librerie per la gestione del sistema operativo direttamente all'interno del codice dell'applicazione.

[1] http://wiki.ros.org/ROS/Tutorials/CreatingPackage
[2] http://wiki.ros.org/Distributions
[3] https://docs.ros.org/en/iron/Releases.html

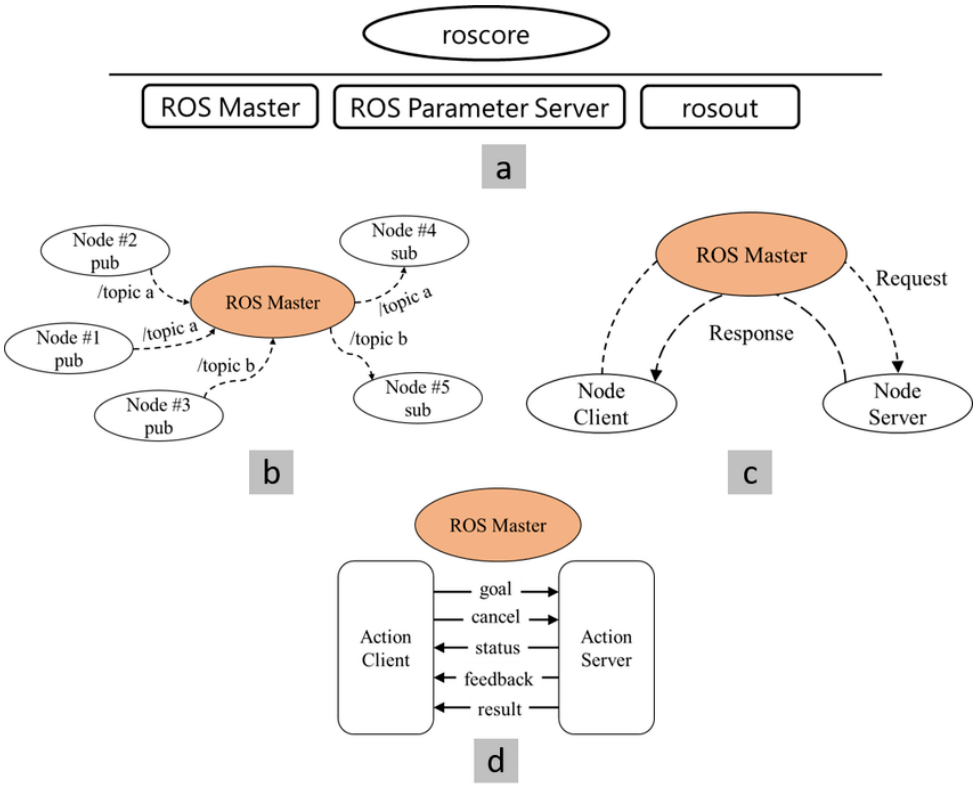

Figura 2.1: Grafo ROS. (a) Grafico computazionale del nucleo, (b) Comunicazione tramite publish e subscribe a topics tra nodi, (c) Comunicazione tramite servizi tra client e server, (d) Comunicazione tramite azioni tra client e server [14]

In questa tesi, ROS è stato utilizzato all'interno di un ambiente programmato in C++ per il collegamento seriale e l'invio di messaggi all'esoscheletro.

Nucleo e Nodi

Il grafo in Figura 2.1 mostra la rete peer-to-peer dei processi di ROS che elaborano i dati contemporaneamente. Roscore è il nucleo del sistema e fornisce una raccolta dei nodi e programmi. Attualmente roscore contiene tre moduli funzionali principali, come mostrato nella Figura 2.1 (a), che sono ROS Master, Parameter Server e rosout [14].

- **ROS Master**: entità di coordinamento centralizzata che facilita la comunicazione tra i diversi nodi di un sistema ROS; mantiene informazioni su tutti i nodi attivi, i loro nomi, i topic e i servizi che offrono o utilizzano. ROS Master è responsabile della gestione dell'infrastruttura di comunicazione, come le connessioni TCP/IP e i protocolli di scambio di messaggi, e fornisce un servizio di denominazione per i nodi in modo che possano individuarsi reciprocamente.
- **Parameter Server**: spazio di memorizzazione chiave-valore che consente ai nodi di memorizzare e recuperare parametri durante l'esecuzione. I parametri vengo-

no utilizzati tipicamente per configurare il comportamento dei nodi, ad esempio per regolare i guadagni del controllo, impostare soglie o definire altri parametri configurabili. Tramite un'API fornita dalla libreria di ROS, è possibile accedere al Parameter Server consentendo ai nodi di reperire e aggiornare facilmente i parametri, consentendo una riconfigurazione dinamica del sistema.

- **Rosout**: meccanismo di registrazione fornito da ROS per catturare e visualizzare i messaggi di log generati dai nodi durante l'esecuzione. I nodi possono utilizzare l'API di registrazione di ROS per generare messaggi di log con diversi livelli di gravità (ad esempio, debug, info, warning, error, fatal). Questi messaggi di log vengono quindi raccolti dal nodo rosout e possono essere visualizzati in varie forme, come output della console, file di log o strumenti visivi come rqt_console. Il sistema di registrazione aiuta nella risoluzione dei problemi e nel monitoraggio del comportamento dei nodi ROS e del sistema nel suo complesso.

Quando un nuovo nodo si vuole iscrivere comunica al ROS Master il suo nome univoco e gli viene assegnato un pid di sistema. I nodi ROS possono localizzarsi a vicenda (richiedendo informazioni al ROS Master) e stabilire comunicazioni peer-to-peer in tre modalità, come indicato nella Figura 2.1 (b-d): publish/subscribe dei messaggi (unidirezionale), server/client dei servizi (unidirezionale ma con feedback) e server/client delle azioni (bidirezionale). Di queste tre modalità appena descritte verrà trattata solo quella attraverso servizi, unica tipologia utilizzata all'interno dell'applicazione.

Servizi

Con servizio si definisce il canale di comunicazione unidirezionale tra client e server dove il client definisce una richiesta (request) ed ottiene una risposta (response) di avvenuta o errata esecuzione dal server che può essere associata ad un'azione specifica del nodo. I servizi sono definiti utilizzando il *ROS Service Description Language (srv)*, che specifica la struttura dei dati. Un nodo che desidera richiedere un servizio invia una richiesta al nodo che offre il servizio. Quest'ultimo elabora la richiesta e restituisce una risposta al nodo richiedente. Richieste o servizi dipendono dalle funzionalità specifiche dei nodi che, durante l'esecuzione della chiamata di servizio, possono essere temporaneamente bloccati in attesa di una risposta.

Nel caso del software che ho realizzato, il servizio creato sarà composto da una prima stringa come valore di request e da una seconda come valore di response (Sezione 3.2).

3 Metodi

In questo capitolo sarà analizzata nel dettaglio la struttura dell'applicazione. Inizialmente verrà descritta interamente la parte di front-end, cioè la grafica. Successivamente, si analizzerà il back-end, cioè lo sviluppo mediante ROS del collegamento con l'esoscheletro.

3.1 Sviluppo UI - Front End

La Figura 3.1 riassume il flusso dell'applicazione sotto forma di diagramma. In essa, è possibile visualizzare tutte le finestre di lavoro principali. Sono assenti le pagine secondarie e di pop-up che verranno descritte successivamente nella sezione 3.1.2.

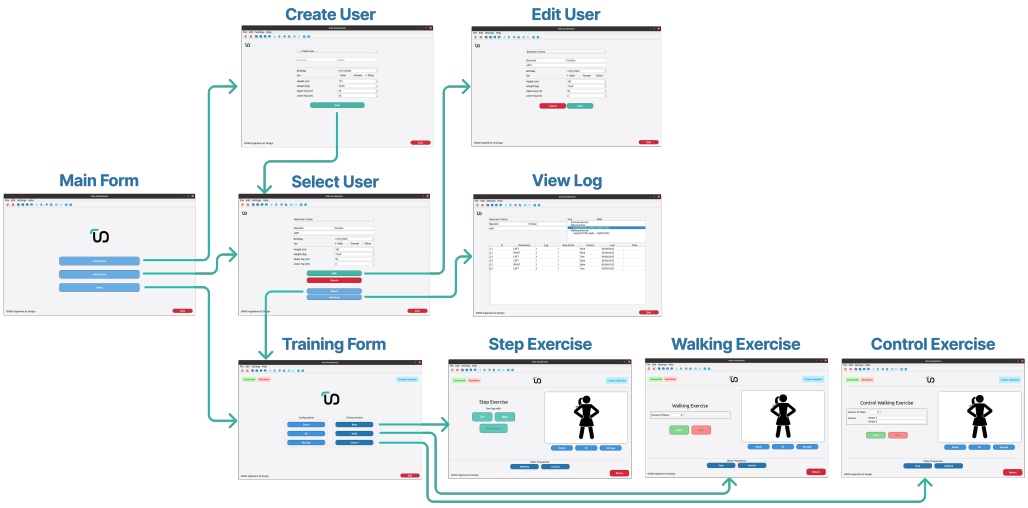

Figura 3.1: Diagramma di flusso dell'applicazione semplificato

3.1.1 Finestre principali

La Figura 3.2 mostra la pagina iniziale dell'applicazione in cui è possibile selezione o creare l'utente che, nel nostro caso, sarà il paziente che necessita della terapia. Invece, scegliendo la modalità demo, è possibile proseguire in anonimato.

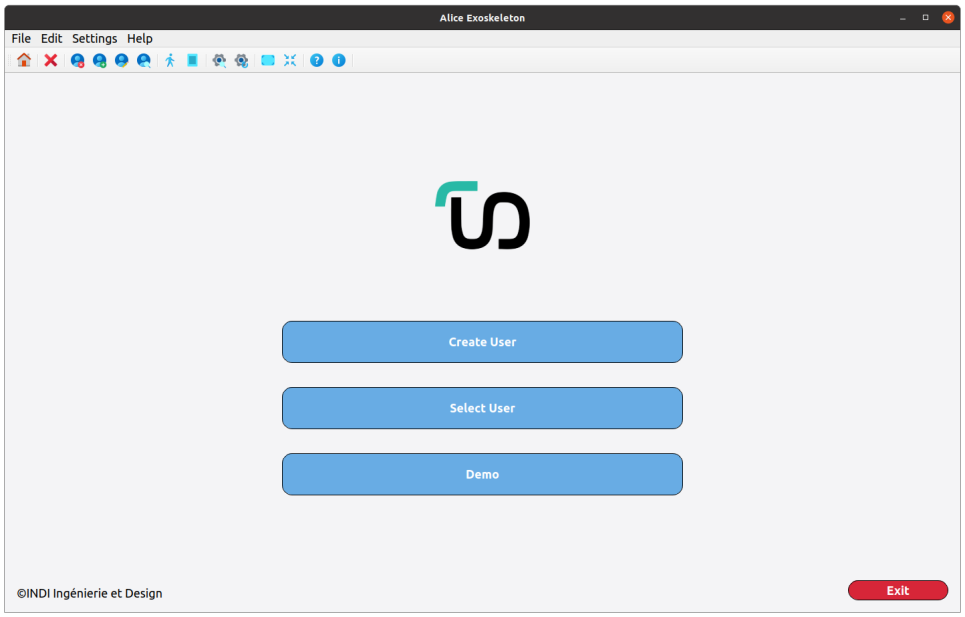

Figura 3.2: Main Form

In alto a sinistra è presente la barra del menù e quella degli strumenti che permettono di muoversi agevolmente all'interno dell'applicazione, di visualizzare o modificare i dati dell'utente in qualsiasi finestra, di calibrare i parametri dell'esoscheletro e, in caso di necessità, di visionare il vademecum del programma. Queste barre saranno presenti in ogni finestra dell'applicazione in quanto parte della FrameWindow, classe contenitore derivata di QMainWindow. Il Widget centrale è personalizzabile attraverso l'inserimento di un documento di lavoro (Main Form nel caso della Figura 3.2).

In basso a destra si trova il tasto *exit* che permette la chiusura dell'applicazione tramite conferma con finestra di pop-up.

Al centro troviamo il logo e in basso a sinistra il nome dell'azienda INDI Ingènierie et Design che, ricordo, ha progettato e sviluppato ALICE.

La creazione e la selezione dell'utente non saranno oggetto di questa tesi così come il salvataggio delle sessioni di allenamento. Per facilitare la comprensione delle successive descrizioni e per maggiore completezza, nella Figura 3.3 si illustra la sequenza di azioni svolte dall'operatore, con relativo feedback, per la scelta dell'utente.

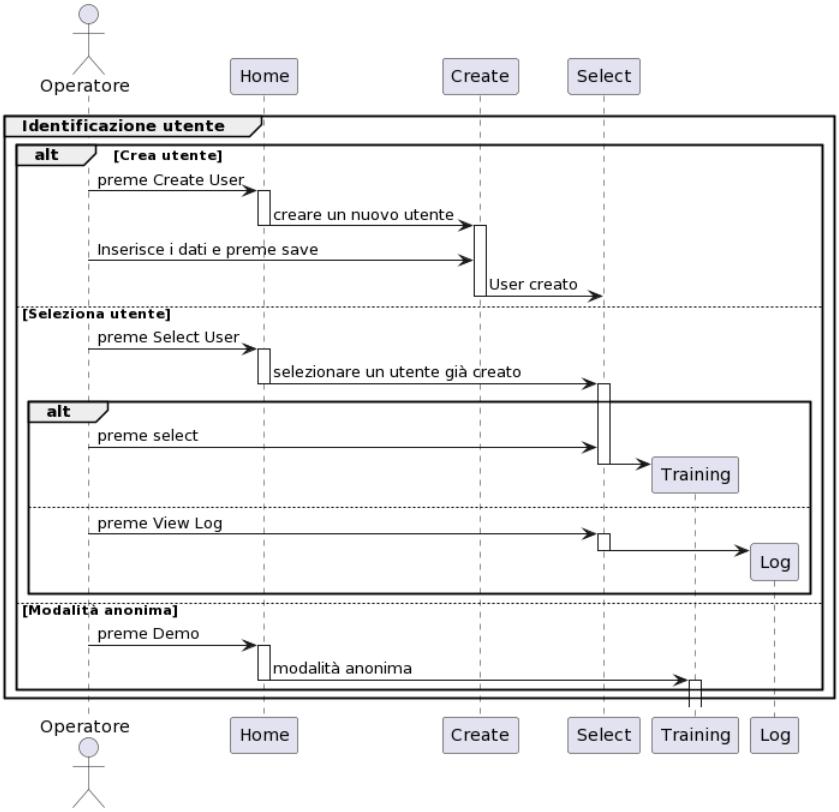

Figura 3.3: Diagramma di sequenza dell'identificazione dell'utente

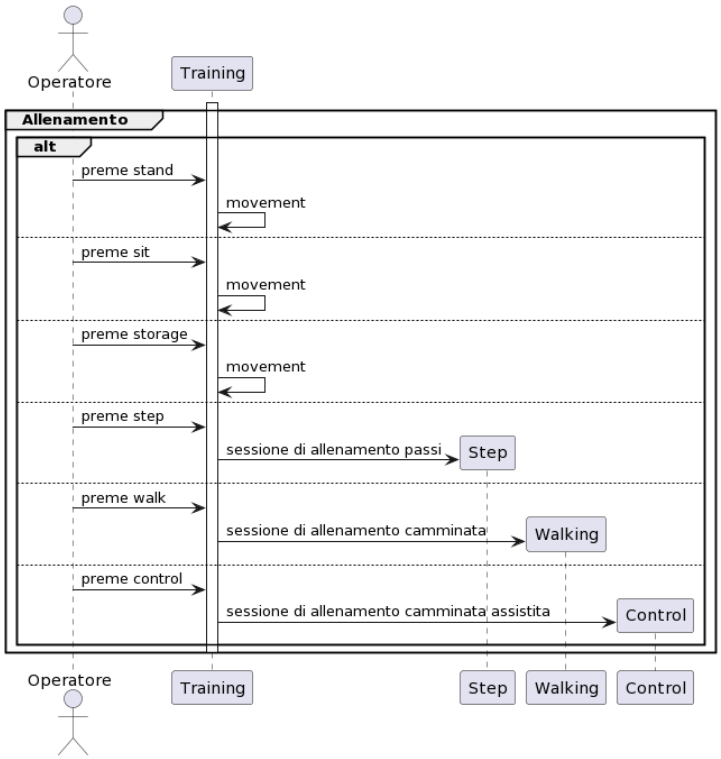

Figura 3.4: Diagramma di sequenza delle tipologie di training

Training

Dopo aver selezionato l'utente o la modalità demo, si accede alla finestra di training (Figura 3.5). Questo pannello presenta le tipologie di allenamento che l'operatore, l'ipotetico fisioterapista, può far svolgere al paziente.

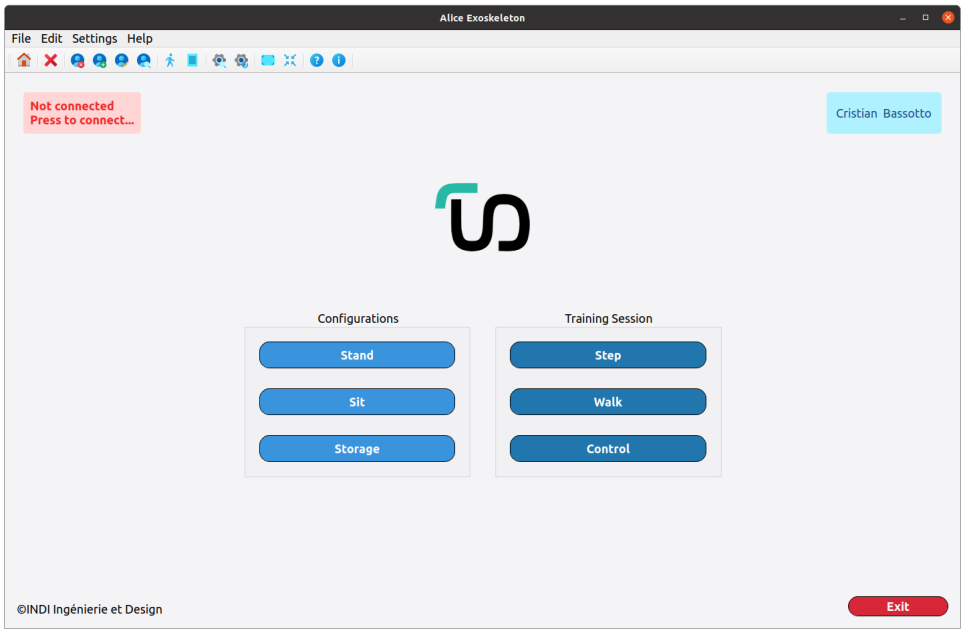

Figura 3.5: Training Form

Al centro della finestra si trovano due box che presentano le modalità di allenamento suddivisibili in due macro-gruppi: *Configurations*, le tre configurazioni di sicurezza che può acquisire l'esoscheletro, e *Training Session*, le tre tipologie di allenamento propriamente detto che analizzerò singolarmente nel dettaglio, ma di cui viene raffigurato un diagramma iniziale nella Figura 3.4.

Sopra i box e in basso a sinistra troviamo sempre il logo e nome dell'azienda che seguirà anche nelle schermate successive.

In alto a destra, all'interno del Widget centrale, si visualizza il nominativo dell'utente precedentemente selezionato, mentre, a sinistra, è possibile controllare lo stato della connessione con l'esoscheletro. Qualora il pulsante presentasse la scritta rossa *Not Connected* basterà, come indicato, premerlo per collegare l'applicazione al robot (in dettaglio sezione 3.2). La visualizzazione di un pulsante verde *Connected* affiancato da *Shutdown* in rosso (bottone attraverso il quale avviene la disconnessione manuale dall'esoscheletro) conferma l'avvenuta connessione.

Nella Figura 3.6 è possibile seguire la sequenza del processo appena illustrato che, se avvenuto con successo, porterà alla visualizzazione di una interfaccia simile alla Figura 3.7.

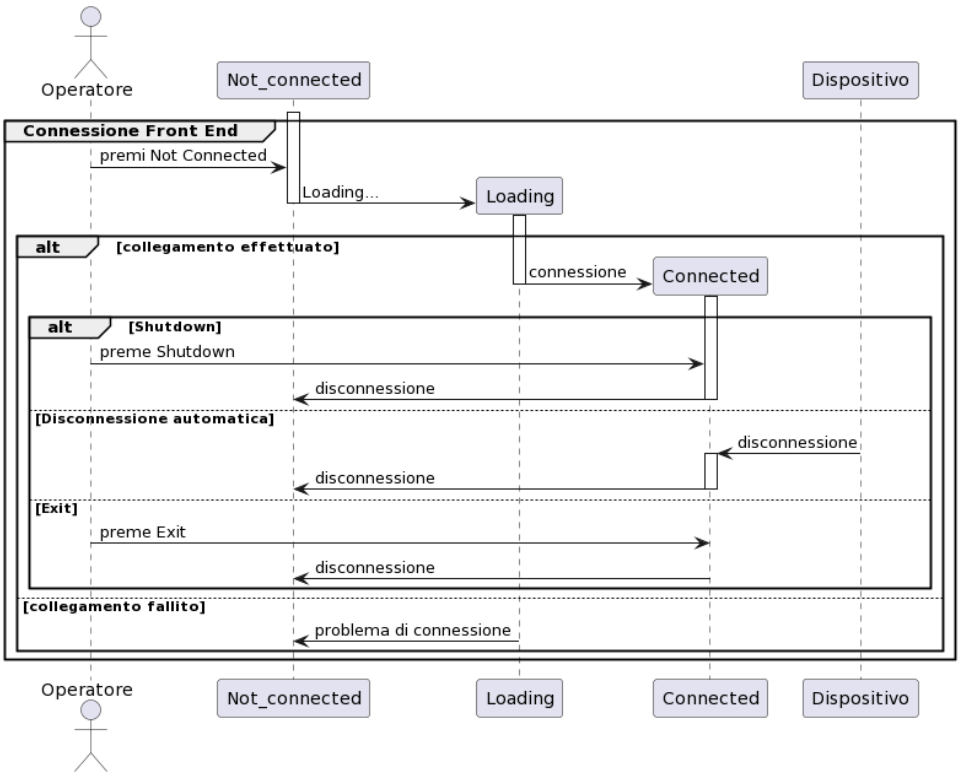

Figura 3.6: Diagramma di sequenza dei pulsanti di connessione

Figura 3.7: Modalità Demo connessa al robot

Configurations

Nel box *Configurations* si trovano tre pulsanti che permettono di ristabilire le conformazioni di base dell'esoscheletro in modo diretto, a differenza del gruppo di training che invece contiene link a finestre secondarie per lo svolgimento delle azioni.

- ***Stand*** permette la messa in sicurezza del robot e lo predispone alle fasi successive di allenamento. All'avvenuta connessione con l'esoscheletro, è necessario configurarlo in *stand* per poter correggere l'assetto degli attuatori. Al termine di ogni sessione di movimenti, sarebbe consigliato ristabilire la posizione eretta.

- ***Sit*** configura l'esoscheletro in posizione seduta. Attualmente, da questo assetto, non è possibile svolgere allenamenti specifici, cosa che potrebbe essere implementata in futuro.

- ***Storage*** è la posizione utilizzata quando l'esoscheletro non è indossato dall'utente perché permette il riassetto del robot in modo che rimanga stabile e fisso quando riposto.

STEP

La Figura 3.8 rappresenta la visualizzazione della sessione di allenamento per singoli passi. Attraverso *Step*, il fisioterapista o l'operatore può seguire e controllare minuziosamente un unico movimento della gamba. Utilizzando i pulsanti, è possibile selezionare il passo di partenza e, al suo compimento, decidere se proseguire o riunire i piedi. Per facilitare il controllo dell'allenamento, i pulsanti di movimento sono disponibili in modo alterno e l'azione è visualizzabile nell'immagine dinamica a destra. A causa di una scelta implementativa, *Feet Toghether* è selezionabile ed eseguibile solamente se l'ultimo passo è stato svolto con la gamba destra.

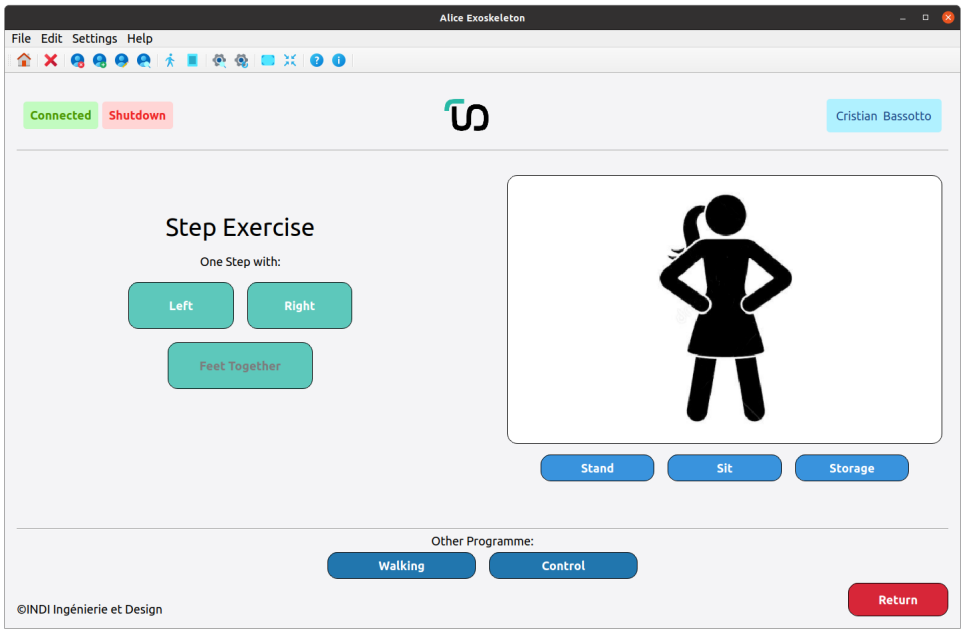

Figura 3.8: Step Form all'interno di Session Form

Ora si procederà con la descrizione di alcuni elementi che rimarranno fissi nelle tre sessioni di *Step*, *Walking* e *Controll Walking*, perché parte della classe Session Form. In basso a destra, si trova il pulsante *Return* che permette di ritornare nella finestra con *Training Form*. In basso al centro, sono presenti gli altri programmi disponibili per l'allenamento per facilitare la navigazione all'interno dell'applicazione. Ad esempio, nella visualizzazione *Step*, ho inserito *Walking* e *Control*. Inoltre, sotto l'immagine, è possibile selezionare le tre modalità di *configurations* per ripristinare l'assetto dell'esoscheletro.

L'icona presente a destra può indicare: l'azione che il robot sta compiendo, la sua configurazione, ma anche lo stato della connessione. Qualora si visualizzasse la Figura 3.9 (g) significherebbe che il collegamento con l'esoscheletro si è interrotto. Si dovrà ripristinare dunque la connessione cliccando il pulsante apposito in alto a sinistra (come visto nella finestra in Figura 3.5).

Figura 3.9: Icone di ALICE: stand (a), passo sinistro (b), passo destro (c), passo di chiusura (d), seduto (e), storage (f), non connesso (g)

Nel diagramma in Figura 3.10 si illustra il processo per l'esecuzione dei movimenti di *Stand*, *Sit*, *Storage* e *Step*.

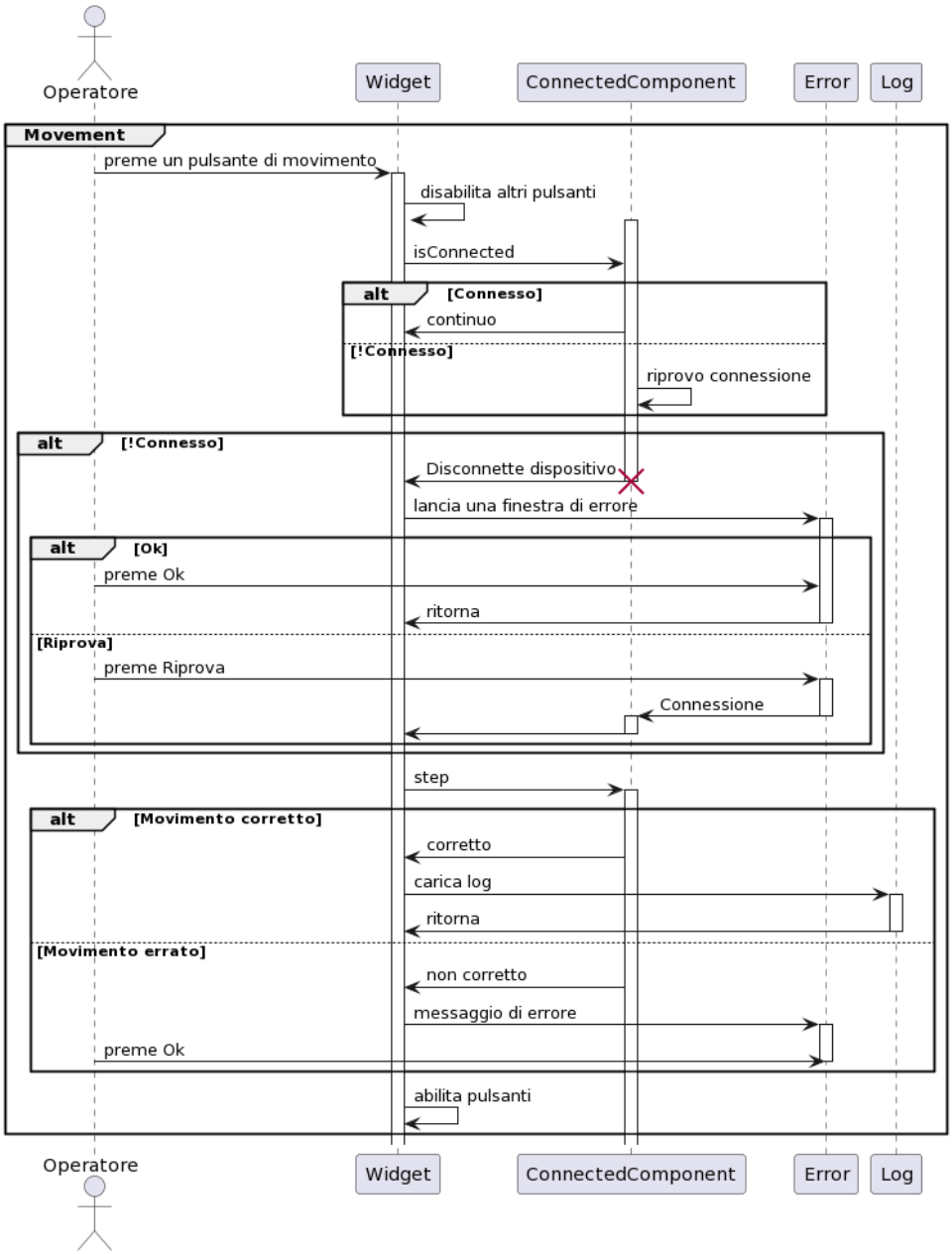

Figura 3.10: Diagramma di sequenza dell'esecuzione di un movimento generico

Walking

Nella Figura 3.11, si illustra solamente il Widget personalizzato per *Walking* inserito in Session Form.

A caratterizzare questo pannello troviamo *QSpinBox* in cui, come indicato, è possibile inserire il numero di passi da far compiere. Premendo *Start* l'esoscheletro eseguirà il primo passo, composto da due falcate, iniziando con la destra e terminando con *Feet Together*, cioè unendo i piedi. Contemporaneamente, apparirà una *QProgressBar* che mostrerà l'avanzare del processo. In qualsiasi istante è possibile bloccare la camminata premendo ***Stop***.

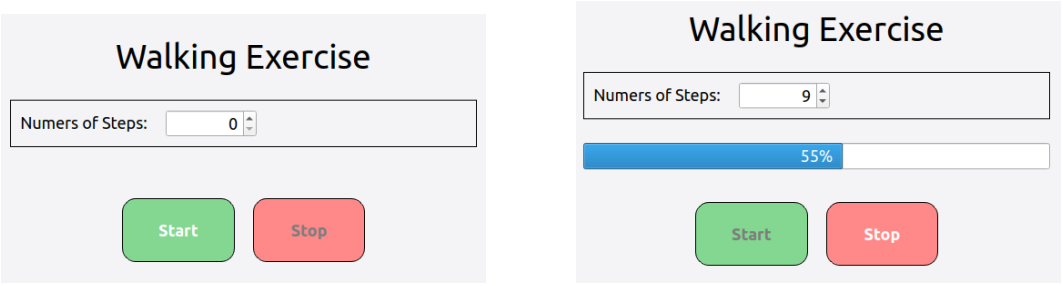

Figura 3.11: Camminata: Prima (a), Durante (b)

Il diagramma 3.12 mostra nel dettaglio quanto appena spiegato.

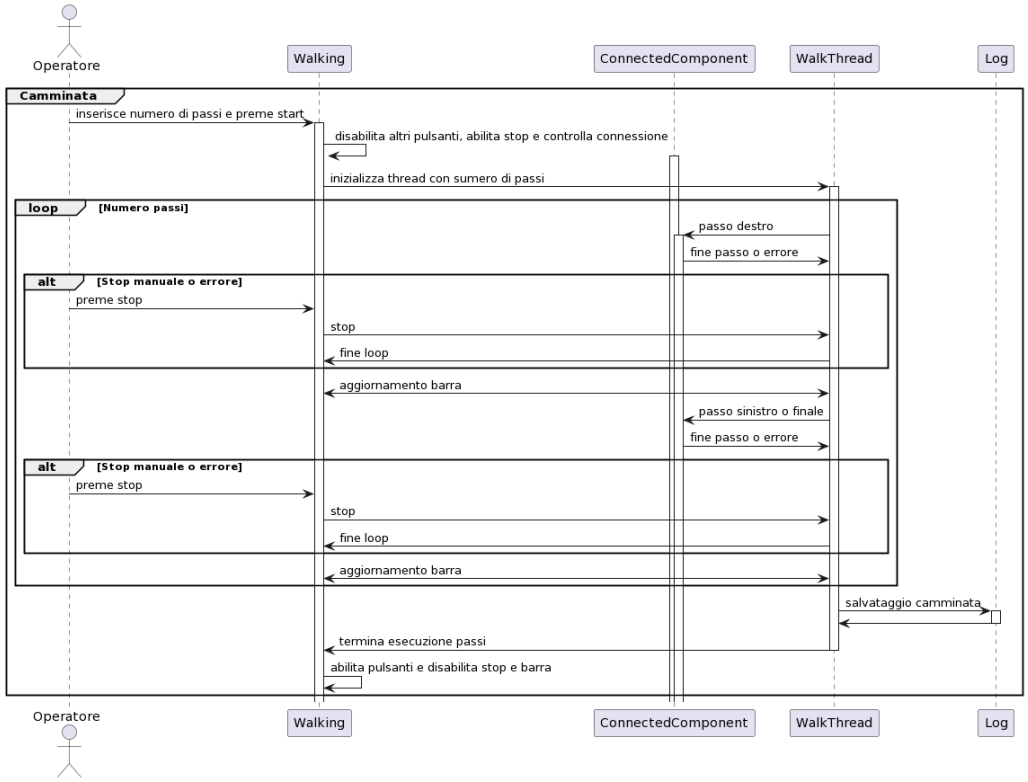

Figura 3.12: Diagramma di sequenza dell'esecuzione della camminata

Control Walking

La sessione *Control Walking* non è attualmente implementata in ALICE ma servirà in futuro come allenamento successivo al *Walking* perché necessita di maggior attività da parte dell'utente. Nello specifico, se in *Walking* è l'esoscheletro a compiere l'azione, in *Control Walking* l'input per il movimento parte dall'utente e l'esoscheletro segue di conseguenza aiutando l'esecuzione.

In questa visualizzazione sono presenti anche alcuni parametri aggiuntivi (da impostare) rispetto a *Walking*.

3.1.2 Finestre secondarie

Si procede con la descrizione di tre tipologie di finestre: impostazioni di connessione, controllo e aiuto. La prima e la terza classe possono essere aperte dalla barra del menù oppure da quella degli strumenti, mentre le finestre di controllo appaiono autonomamente a causa di qualche evento.

Finestra di Impostazioni Connessione

In Figura 3.13 sono presenti le due finestre di impostazioni di connessione che mostrano i **parametri ROS**: (a) è di sola lettura mentre (b), che si visualizza anche solamente premendo *Change Parameters*, è modificabile.

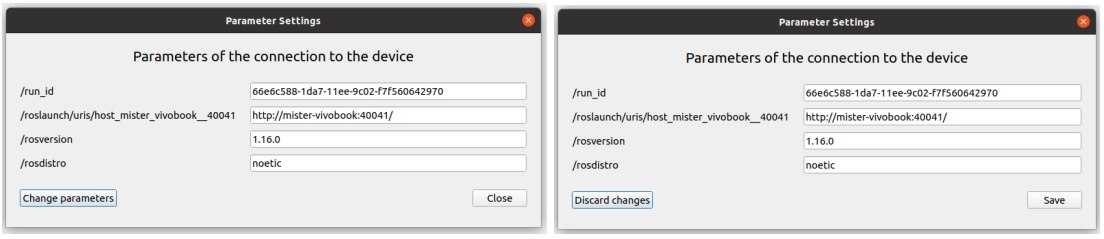

Figura 3.13: Finestra di Impostazioni di Connessione: solo lettura (a), modificabile (b)

Finestre di Controllo

In Figura 3.14 sono presenti le due finestre a comparsa automatica. ***Confirmation*** appare successivamente alla selezione di *exit* o all'icona di chiusura, mentre ***Error*** compare se il processo di connessione non ha avuto successo a causa, probabilmente, di un'errore nella porta di collegamento.

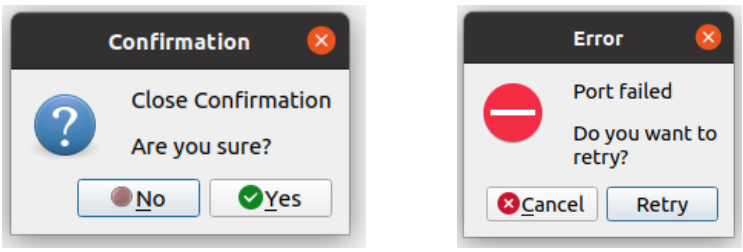

Figura 3.14: Finestre di Controllo: accettazione chiusura (a), errore di porta (b)

Finestre di Aiuto

In Figura 3.15 sono presenti la finestra di aiuto: *Help*, utile per visualizzare il vademecum, e la finestra informativa, *Informations*.

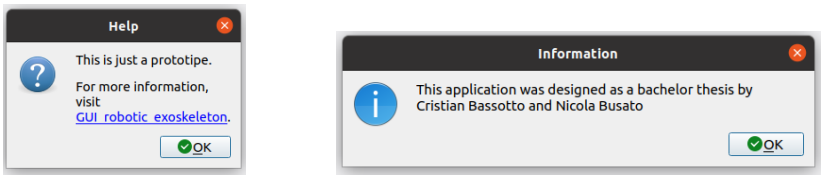

Figura 3.15: Finestre di Aiuto: aiuto (a), informazioni (b)

3.2 Sviluppo ROS - Back End

La back-end si occupa della connessione con ALICE e dell'invio dei servizi attraverso canali di comunicazione affinché avvenga il movimento.

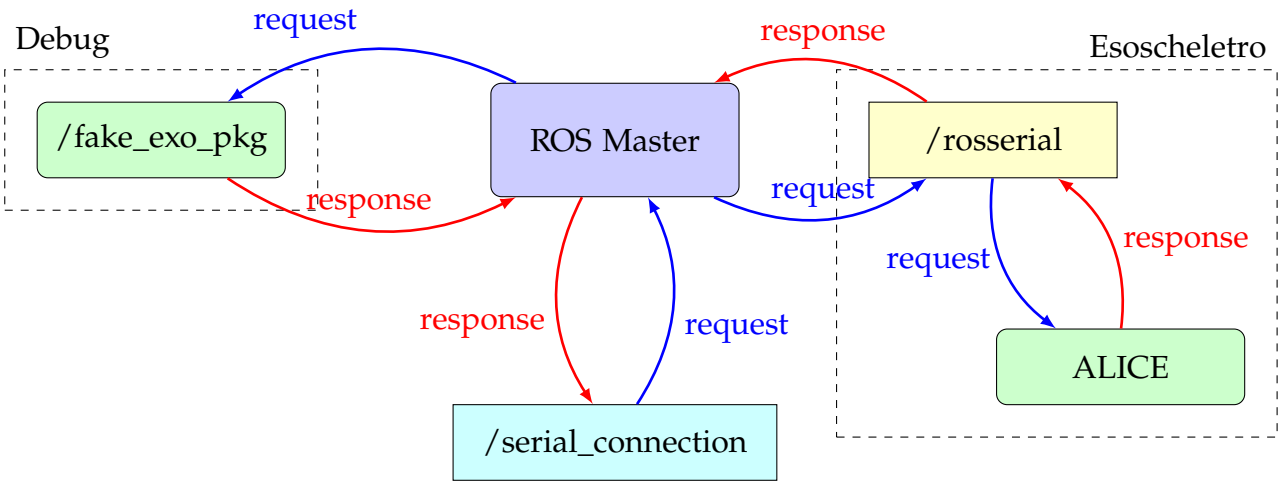

Figura 3.16: Diagramma del collegamento con l'esoscheletro

Per comprendere il funzionamento specifico di ROS in questo software, descriverò la Figura 3.16 che mostra come avviene la risposta alle richieste fatte all'interno dell'applicazione. Selezionando il pulsante per il collegamento all'esoscheletro, il nodo *serial_connection* viene inizializzato all'interno di *Connected Component*. Quando ciò

avviene, *ROS launch* lancia un comando che, in parallelo, permette l'apertura della connessione con *ROS Master* attraverso il nodo *ROS Core* e l'inizializzazione di uno dei seguenti nodi: *Debug* oppure *Esoscheletro*. **Debug** è un esoscheletro fittizio da me creato per poter testare l'apparato di connessione dell'applicazione senza il robot reale. Invece, *Esoscheletro* è l'architettura che comprende *ROS Serial*, nodo che si occupa della connessione seriale attraverso porta USB, e i nodi di Arduino di ALICE.

La richiesta di un servizio detto *Movement_srv* invia una *request* che parte da *serial_connection*, attraversa *ROS Core* e viene reindirizzata all'esoscheletro (o a debug in fase di test) che la rielabora e invia una risposta.

Request e Response sono variabili di tipo *std::string* descritte in *Test.srv*. La richiesta contiene il codice del movimento che ALICE deve svolgere: "12" per il passo sinistro, "34" per il destro, "35" per la chiusura, "s" per stand, "n" per storage e "y" per sit. La risposta invece contiene "0" o "1" in base al successo o fallimento dell'azione.

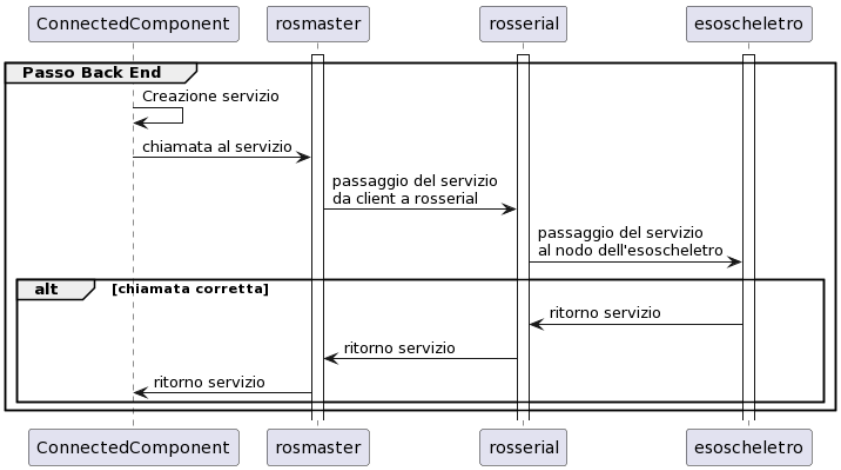

Figura 3.17: Diagramma di sequenza dell'invio del servizio all'esoscheletro

3.2.1 Componente Connessa

La classe principale che si occupa della connessione tra GUI ed esoscheletro si chiama *ConnectedComponent*.

Il processo di collegamento si avvia quando l'operatore preme il pulsante *Not Connected* visto nella sezione 3.1.1. Se la connessione avviene con successo, ogni 5 secondi verrà automaticamente verificata. All'inizio di ogni controllo, il timer viene interrotto per riprendere al termine dell'accertamento se l'esoscheletro è ancora connesso. Se la connessione non è presente, viene verificata l'esecuzione di ROS Master e, se presente, lo si inizializza, altrimenti viene lanciato ROS launch che inizializza sia ROS Master che l'esoscheletro. Successivamente, *serial_connection* si iscrive al servizio *Movement_srv* e si verifica nuovamente la connessione per testare l'avvenuta sottoscrizione al canale.

Se non avviene il collegamento, all'operatore comparirà la finestra di errore vista in Figura 3.14.

Il diagramma di sequenza in Figura 3.17 mostra l'insieme di operazioni svolte dal metodo *connect* di *ConnectedComponent* per inizializzare e controllare la connessione, come appena descritto.

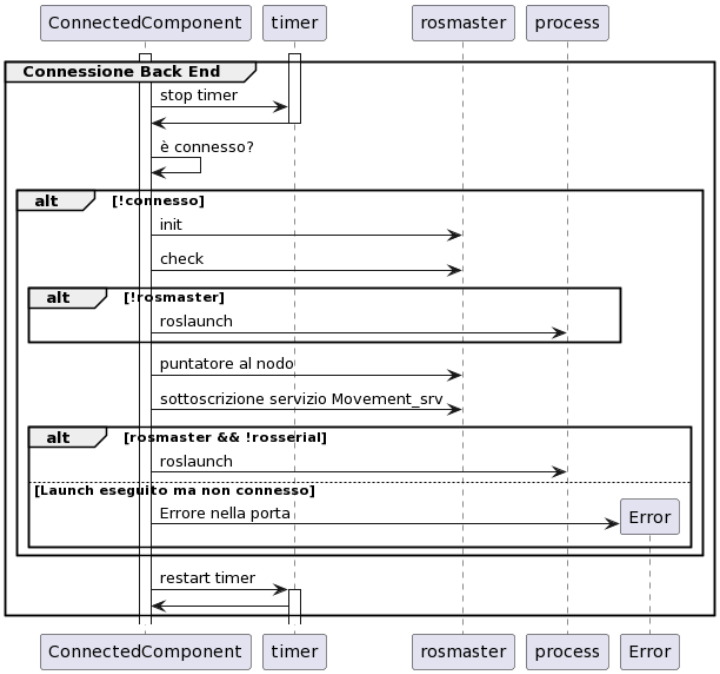

Figura 3.18: Diagramma di sequenza della connessione in background

Nella Figura 3.19 si trova invece la sequenza di operazioni di chiusura manuale della connessione. Lo stesso metodo è anche eseguito alla chiusura dell'applicazione. Per la corretta terminazione di ogni nodo inizializzato è necessario un segnale di kill direttamente al suo processo aperto nel sistema operativo (perchè ROS non implementa questa funzione). Per questo motivo vengono prima richiesti i *pid* dei processi *rosmaster*, *roslaunch* e *serial_node.py* per esser successivamente terminati in serie in un loop.

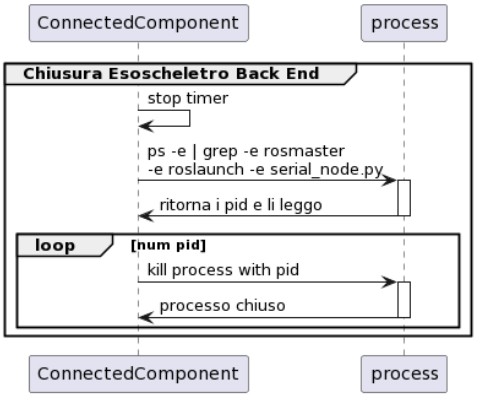

Figura 3.19: Diagramma di sequenza della chiusura manuale della connessione

4

Conclusioni

In questo elaborato è presentata la progettazione e lo sviluppo di una GUI per governare ALICE, un esoscheletro per il cammino assistito. L'innovazione alla base di questa tesi deriva dalla possibilità, per gli operatori, di non dover più utilizzare il terminale per la comunicazione dei comandi.

L'applicazione è stata testata in laboratorio dimostrando la sua efficienza in termini di fluidità ed efficacia nell'invio di comandi corretti. Il software risulta facilmente installabile con *setup.sh* e maneggevole grazie alla suddivisione in pacchetti. La navigazione è intuitiva con uno stile minimal e user-friendly. In prima persona ho potuto testare il funzionamento dell'intera infrastruttura (Figura 4.1).

Figura 4.1: Test eseguiti in laboratorio

Al momento è possibile eseguire l'applicazione solo su **Ubuntu Focal 20.04** perché la versione attuale di ROS Noetic non è ancora stata rilasciata per Windows o MacOS. Seguendo le istruzioni presenti nel seguente link di GitHub sarà possibile installare il software completo di debugger: `https://github.com/cristianbass01/GUI_robotic_exoskeleton`.

Se in un futuro progetto fosse possibile l'installazione di questa versione su Windows, ne risentirebbe solo la prima parte compilativa e il pacchetto ROS. Infatti, il codice C++ è stato implementato per funzionare correttamente anche nel caso di sistema operativo Windows o MacOS.

Per aggirare questo problema è possibile installare un sottosistema operativo Ubuntu in Windows attraverso wsl[1].

La corrente versione dell'applicazione funge da base per un **progetto futuro** che comprenderà l'implementazione di altre modalità di allenamento come quelle da seduto e il *Control Walking* già descritto. In particolare potrebbero essere svolti esercizi di estensione degli arti in posizione eretta o seduta, di controllo di ogni singolo movimento della gamba durante il passo (che al momento comprende sia l'alzata che la distensione dell'arto) e di chiusura delle gambe col piede destro invece che col sinistro in modo da poter scegliere l'arto con cui voler iniziare la camminata.

Inoltre, un progetto futuro che potrebbe migliorare la terapia fornita dal fisioterapista potrebbe coinvolgere i sensori dell'esoscheletro per poter comprendere la forza applicata dal paziente durante il passo o per evidenziare eventuali irregolarità nell'esecuzione dei movimenti.

[1] `https://learn.microsoft.com/it-it/windows/wsl/install`

A

Classi

In questa sezione verrà mostrato il pacchetto creato e il diagramma delle sue classi.

A.1 ALICE_PKG

Alice_pkg (Figura A.1) è un pacchetto ROS che contiene tutte le classi dell'interfaccia implementata. Risulta liberamente scaricabile da GitHub come pacchetto singolo + *setup.sh*. Il pacchetto verrà aggiunto all'interno del src del proprio workspace e il file di setup dovrà esser eseguito nel workspace. Se viene selezionata l'opzione di debug, verrà autonomamente installato il pacchetto fake_exo_pkg nella fase di setup e potrà esser scelta la modalità di esecuzione preferita al lancio dell'applicazione.

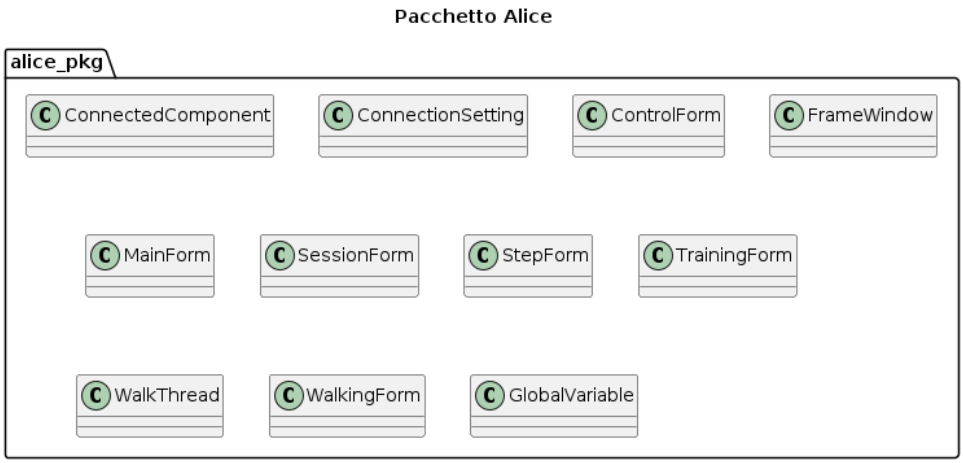

Figura A.1: Diagramma del pacchetto alice_pkg

A.2 Classi di front end della GUI

Verranno ora presentate le classi che ho implementato per l'interfaccia in ordine di comparizione nel capitolo 3: *FrameWindow, MainForm, TrainingForm, SessionForm, StepForm, WalkingForm, ControlForm, WalkThread, ConnectionSetting e ConnectedComponent*.

Da notare è il metodo *customizeWindow* delle classi *FrameWindow* e *SessionForm* attraverso il quale è possibile inserire altri Widget all'interno della grafica e quindi personalizzare l'interfaccia utente. Questo metodo è stato utilizzato per riutilizzare classi uguali su finestre apparentemente diverse, ad esempio *FrameWindow* che viene utilizzato in quasi tutte le finestre.

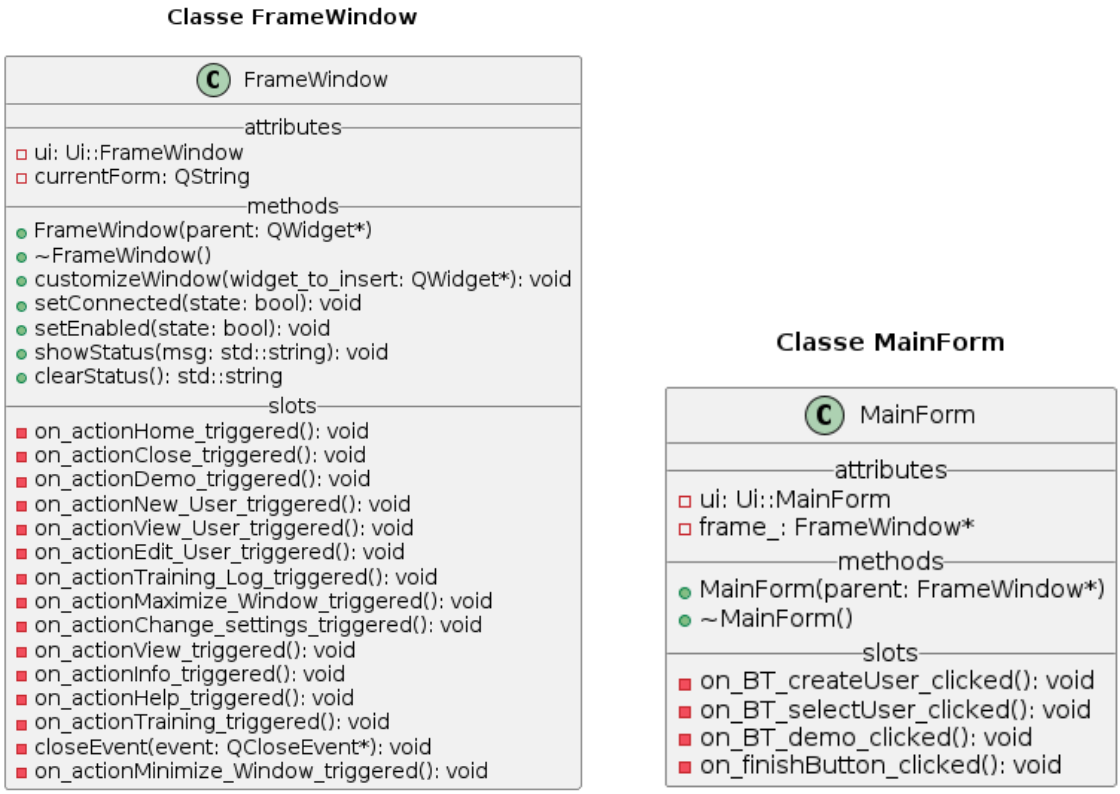

Figura A.2: Diagramma delle classi: FrameWindow (a), MainForm (b)

Le classi *TrainingForm*, *SessionForm* e *StepForm* presentano invece il metodo *movement*. Il funzionamento è lo stesso già descritto nella sezione 3.1.

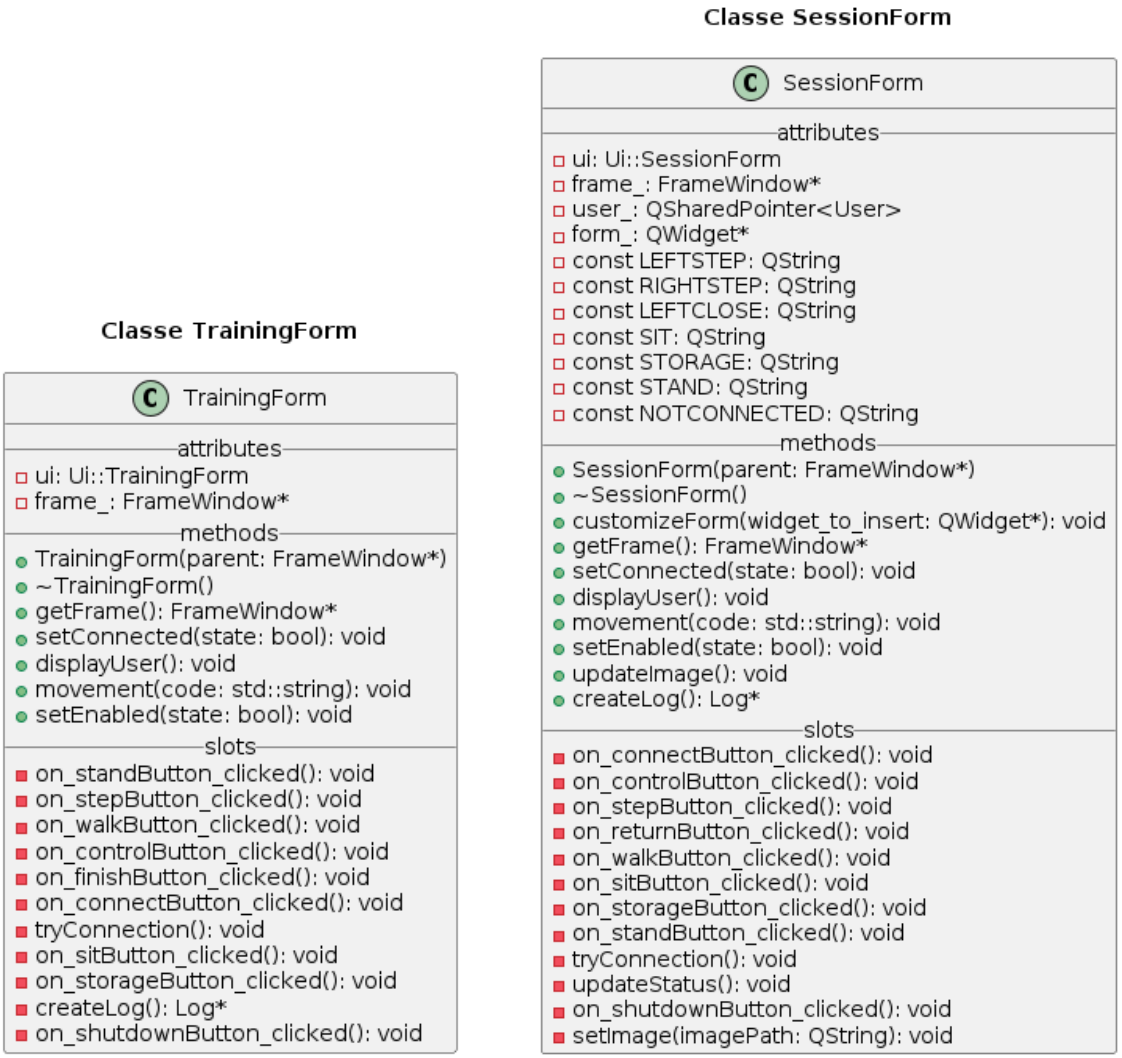

Figura A.3: Diagramma delle classi TrainingForm (a) e SessionForm (b)

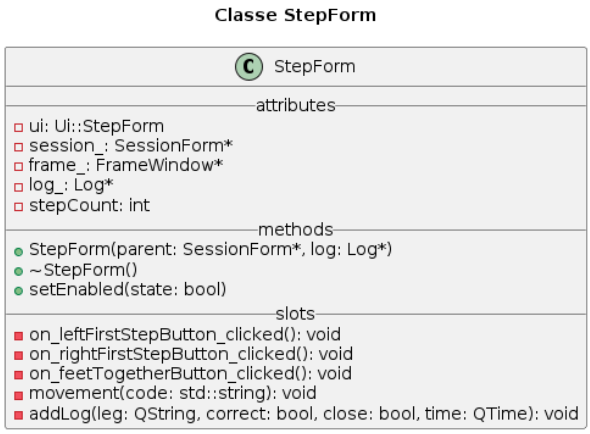

Figura A.4: Diagramma della classe StepForm

WalkingForm (Figura A.5) invece contiene anche il puntatore ad un'istanza *WalkThread* per svolgere la camminata in background.

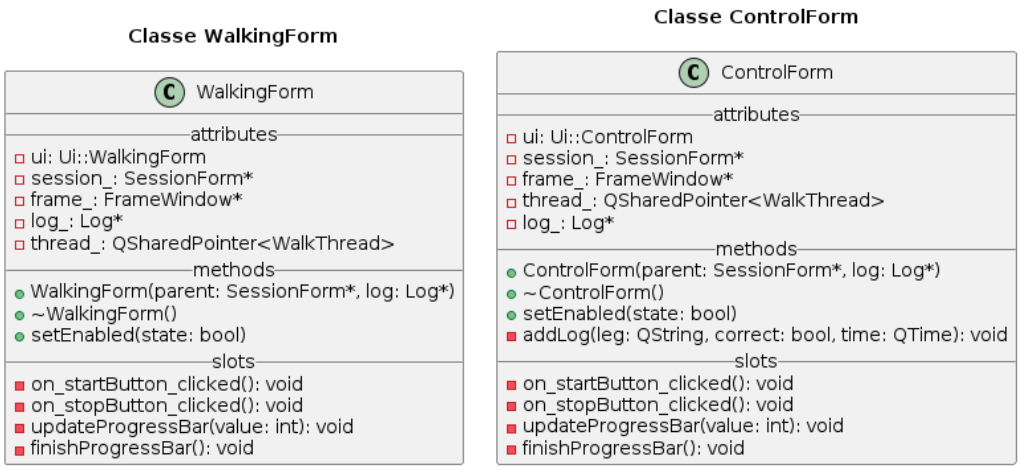

Figura A.5: Diagramma delle classi WalkingForm (b) e ControlForm (c)

La classe *WalkThread* (Figura A.6) invece è l'unica che contiene sia segnali che slot perché richiama metodi che sono connessi ad altri metodi di classe *Walking* e *Log*.

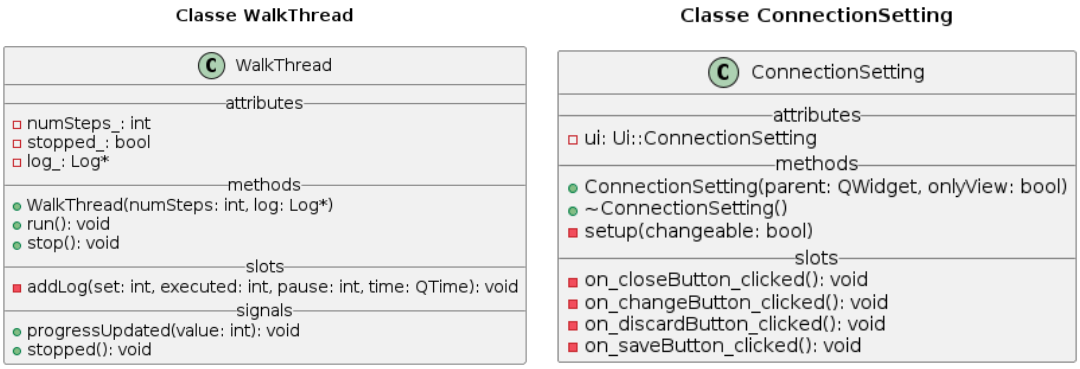

Figura A.6: Diagramma delle classi: WalkThread (a), ConnectionSetting(b)

A.3 Classe di back end ROS

Notiamo che il costruttore della classe *ConnectedComponent* (Figura A.7) è privato e sono stati eliminati i metodi per la copia. Questo è il procedimento standard del design pattern Singleton assieme al metodo *getInstance()* che invece ritorna l'unica istanza di quella classe.

Figura A.7: Diagramma della classe ConnectedComponent

Bibliografia

[1] Leah Morris et al. «The-state-of-the-art of soft robotics to assist mobility: a review of physiotherapist and patient identified limitations of current lower-limb exoskeletons and the potential soft-robotic solutions». English. In: *Journal of neuroengineering and rehabilitation* 20.1 (gen. 2023). DOI: 10.1186/s12984-022-01122-3.

[2] Vikash Kumar, Yogesh V. Hote e Shivam Jain. «Review of Exoskeleton: History, Design and Control». In: *2019 3rd International Conference on Recent Developments in Control, Automation & Power Engineering (RDCAPE)*. 2019, pp. 677–682. DOI: 10.1109/RDCAPE47089.2019.8979099.

[3] A. Voilqué et al. «Industrial Exoskeleton Technology: Classification, Structural Analysis, and Structural Complexity Indicator». In: *2019 Wearable Robotics Association Conference (WearRAcon)*. 2019, pp. 13–20. DOI: 10.1109/WEARRACON.2019.8719395.

[4] Monica Tiboni et al. «Sensors and Actuation Technologies in Exoskeletons: A Review». In: *Sensors (Basel)* 22.3 (2022), p. 884. DOI: 10.3390/s22030884. URL: https://www.mdpi.com/1424-8220/22/3/884.

[5] Nicholas Yagn. «Apparatus for Facilitating Walking, Running, and Jumping». 420,178. United States Patent. Gen. 1890.

[6] Javier A. de la Tejera et al. «Systematic Review of Exoskeletons towards a General Categorization Model Proposal». In: *Applied Sciences* 11.1 (2021), p. 76. DOI: 10.3390/app11010076.

[7] A. Zoss, H. Kazerooni e A. Chu. «On the mechanical design of the Berkeley Lower Extremity Exoskeleton (BLEEX)». In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2005, pp. 3465–3472. DOI: 10.1109/IROS.2005.1545453.

[8] Manuel Cardona et al. «ALICE: Conceptual Development of a Lower Limb Exoskeleton Robot Driven by an On-Board Musculoskeletal Simulator». In: *Sensors* 20.3 (2020). ISSN: 1424-8220. DOI: 10.3390/s20030789. URL: https://www.mdpi.com/1424-8220/20/3/789.

[9] Jesús Tamez-Duque et al. *ALICE Open Source Exoskeleton 2021 Update*. https://hackaday.io/project/176681-alice-open-source-exoskeleton-2021-update. Accessed on June 30, 2023. 2021.

[10] Marc Gregoire. *Professional C++, 4th Edition*. eng. 4th edition. Wrox, 2018. ISBN: 1-5231-1848-2.

[11] Stephan Roth. *Clean C++20 : sustainable software development patterns and best practices*. eng. Second edition. Berkeley, California: Apress L. P., 2021. ISBN: 1-4842-5949-1.

[12] Johan Thelin. *Foundations of Qt Development*. Berkeley, CA: Apress, 2007, p. 528. ISBN: 978-1-5905-9831-3.

[13] Lentin Joseph e Aleena Johny. *Robot Operating System (ROS) for Absolute Beginners: Robotics Programming Made Easy*. Berkeley, CA: Apress, 2022, p. 283. ISBN: 978-1-4842-7750-8.

[14] Hao Deng, Jing Xiong e Zeyang Xia. «Mobile manipulation task simulation using ROS with MoveIt». In: *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*. 2017, pp. 612–616. DOI: 10.1109/RCAR.2017.8311930.

Ringraziamenti

Al termine di questo elaborato, desidero ringraziare tutte le persone che mi hanno sostenuto durante il mio percorso universitario e durante la stesura della tesi.

Innanzitutto, vorrei ringraziare il mio relatore, Stefano Tortora, che mi ha seguito e guidato in ogni fase dell'implementazione di questo progetto. Grazie per la Sua grande disponibilità in questi mesi e per i suoi spunti fondamentali.

Non posso non menzionare i miei genitori, che da sempre mi sostengono nella realizzazione dei miei progetti. Grazie per il vostro costante supporto e incoraggiamento.

Un ringraziamento speciale va a Sara, la persona che più di tutte mi ha aiutato in questo percorso, che è stata sempre al mio fianco tranquillizzandomi nei momenti più bui e che mi ha aiutato passo passo nella realizzazione e correzione di questa tesi. Grazie di cuore! Senza di te non sarei mai riuscito a portare a termine tutto questo.

Desidero inoltre ringraziare i miei amici e i compagni di corso, con i quali ho condiviso gioie e sfide durante questo percorso accademico. Senza di voi, tutto sarebbe stato più cupo: grazie per avermi trasmesso entusiasmo e coraggio. In particolare, vorrei ringraziare Nicola, con cui ho condiviso questo progetto di tesi.

Sono profondamente grato a tutte le persone che mi hanno aiutato lungo questo viaggio e mi hanno permesso di raggiungere questo importante traguardo nella mia formazione accademica.