UNIVERSITÀ
DI PARMA

Dipartimento di Scienze Matematiche, Fisiche e Informatiche

Corso di Laurea Triennale in Informatica

Integrazione e ottimizzazione di un sistema di riabilitazione cognitiva con robot NAO ed AI per il riconoscimento di emozioni

Integration and optimization of a cognitive rehabilitation system with a NAO robot and AI for emotion recognition

Candidato:
Antonio Rubino

Matricola:
336325

Relatore:
Prof. Alessandro dal Palù

Correlatori:
Prof. Riccardo Monica
Prof.ssa Olimpia Pino

Anno Accademico 2023-2024

Alla mia famiglia, a cui devo tutto.
A tutti i miei cari, che ci sono stati e ci saranno sempre.
Grazie

Indice

Elenco delle figure

Elenco degli algoritmi

Elenco delle tabelle

Introduzione

Ci troviamo negli anni più fiorenti dell'informatica, Dallo sviluppo dei Chat-Bot, in grado di cambiare radicalmente le abitudini comuni, alla nascita di Robot che, tramite la particolare struttura umanoide, incrementata da complessi algoritmi, riescono a replicare movimenti umanizzati, in modo quasi naturale. Gli sviluppi che possono esserci a partire da ciò sono infiniti, come lo sono le loro applicazioni; in ambito medico, ad esempio, avere la possibilità di utilizzare a nostro vantaggio uno strumento così potente potrebbe migliorare notevolmente l'interazione con i soggetti che, affascinati dal nuovo, risulterebbero più coinvolti nel processo di riabilitazione. È in questo contesto che nascono i "SAR", acronimo di "Socially Assistive Robots", una tipologia di Robot che si concentra sull'interazione emotiva, piuttosto che fisica e che guida, passo dopo passo, nella riabilitazione, nell'educazione o nell'apprendimento. È semplice intuire come questa branca si possa facilmente applicare in contesti quali l'insegnamento ai bambini, ad esempio, la cui interazione con un robot potrebbe essere un ottimo appiglio per motivare una maggiore concentrazione e una stimolante motivazione, o con gli anziani, nei processi di riabilitazione. È fondamentale e di rilevante importanza sottolineare che il progetto, esposto in questo lavoro, è stato realizzato in collaborazione con il dipartimento di Medicina e Chirurgia, laboratorio di Psicologia cognitiva e il dipartimento di Ingegneria, dell'università di Parma, ponendosi l'obiettivo di utilizzare un SAR, nello specifico il robot autonomo NAO, come strumento applicativo il cui ruolo specifico approfondiremo nel dettaglio nei capitoli successivi. Questo automa umanoide è stato implementato e settato per sostenere delle sedute di riabilitazione cognitiva con soggetti, perlopiù anziani, che presentano problemi di memoria lievi. Le sedute in questione, che verranno chiamate "sessioni" durante tutto l'elaborato, sono delle vere e proprie sedute di riabilitazione, così come verrebbero poste da un esperto, utilizzando tuttavia, in sostituzione all'azione umana, le funzionalità di NAO. Verranno sottoposte ai soggetti una serie di attività, che potranno essere vocali, interattive, o un mix di entrambe, aventi come obiettivo quello di ridurre i deficit comportamentali e cognitivi. Tutto questo verrà realizzato sfruttando ciò che NAO ha da offrirci: sensori, videocamere, microfoni e quant'altro sia proprio alle caratteristiche tecniche

del dispositivo; utilizzata al meglio già oggi, come dimostreremo, e ancor più in futuro, questa modalità applicativa potrà fornirci un responso molto simile se non superiore a quanto ottenibile dal contatto diretto con un operatore medico. In questo elaborato verrà esposto, inoltre, tutto ciò che fino ad ora è stato svolto, fornendo precedentemente un background dell'attività realizzata, senza tralasciare informazioni generali e descrivendo il percorso a partire dai lavori precedenti, fino a ciò che è stato prodotto e sviluppato finora, personalmente. Allo stesso modo, non verranno tralasciati i dettagli tecnici riguardanti le parti hardware e software del dispositivo robotico. Al riguardo saranno, in concomitanza, evidenziate e descritte idee ed eventuali evoluzioni che potranno essere, in seguito, sviluppate al riguardo. Sottolineo che il progetto sta compiendo in questo momento i suoi primi passi e tutto quello che verrà mostrato è in fase di progettazione, rappresentandosi, nei suoi termini essenziali, come una "Demo" del lavoro finale.

Capitolo 1

I Socially Assistive Robots: "SAR" e NAO

In questo capitolo si affronterà nel dettaglio la tipologia di Robot con cui è stato possibile realizzare le diverse fasi del progetto. Inizialmente si descriveranno le caratteristiche generali dei SAR, delle loro applicazioni (in particolare in ambito medico), delle loro caratteristiche e delle più significative e utili particolarità. Si approfondirà poi il robot NAO, una tipologia di SAR con caratteristiche molto simili a quelle di un bambino.

1.1 I SAR

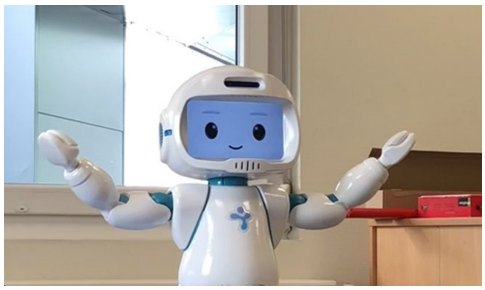
Negli ultimi anni si è assistito ad una notevole crescita nell'industria della robotica, con cambiamenti costanti nella progettazione e nello sviluppo.

Secondo [5], possiamo definire 5 tappe nell'evoluzione della robotica:

- prototipi di robot, la prima tappa nella robotica, caratterizzata da robot molto semplici e con funzioni basiche, principalmente prototipi per l'appunto
- bracci robotici, che hanno rivoluzionato il mondo dell'automazione industriale e progettati per eseguire compiti specifici per l'aumento della produttività e dell'efficienza
- robot ambulanti, in grado di camminare e muoversi autonomamente, progettati per emulare i movimenti umani e operare in ambienti diversi
- robot basati sul comportamento, in grado di prendere decisioni autonome e reagire in base all'ambiente circostante

- robot umanoidi, l'attuale generazione, dotati di interfacce sociali[1] avanzate, con possibilità di interagire in più contesti; in questa categoria troviamo i SAR, in particolare NAO.

Un SAR è una tipologia di robot umanoide abile nel completare una serie di azioni, o task, con l'aggiunta di un'interfaccia sociale in grado di convincere l'utente che il robot sia un partner con cui interagire.[2]



(a) Immagine da: luxai.com [8]

(b) Immagine da: nocamels.com [21]

Figura 1.1: Esempi di Socially Assistive Robots (SAR).

1.1.1 I SAR e le interazioni con anziani

Come già anticipato, una delle principali applicazioni dei SAR è la riabilitazione. Nel contesto corrente, il focus principale è l'interazione con gli anziani, una categoria che con la tecnologia ha poco a che fare, ma che, come hanno dimostrato diversi studi, si troverebbe più stimolata dalla presenza di un robot come operatore. Uno studio [2] dimostra come l'utilizzo di un SAR con soggetti anziani possa migliorare l'umore e alleviare gli stati di disturbo. In particolare, nello studio in questione, sono stati usati 3 robot: "AIBO", "Paro" e "Sophie". Si sono identificati 5 ambiti di utilizzo in cui i SAR sembrerebbero migliorare lo stato degli anziani:

- terapia affettiva: possono migliorare il benessere generale, con interazioni di gruppo che si sono rivelate più efficaci rispetto a quelle individuali
- training cognitivo: i robot sociali si sono dimostrati in grado di migliorare le misure cognitive specialmente per gli anziani cognitivamente sani. In soggetti con demenza, i risultati sono meno chiari a causa delle difficoltà nell'interpretare i dati
- facilitazione sociale: migliorerebbero la socialità, sia in soggetti con demenza che senza, tuttavia sembrerebbe che si evidenzino risultati maggiori in contesti di gruppo

[1] La complessità di uno scenario supportata da un robot

- compagnia: gli studi hanno dimostrato effetti positivi sull'isolamento sociale, evidenziando come i SAR che imitino gli animali possano essere degli ottimi compagni per gli anziani nelle case di cura

- terapia fisiologica: sono stati osservati effetti positivi, come la riduzione della pressione sanguigna e della frequenza cardiaca dopo interazioni con il SAR Paro. Sebbene non ci siano risultati schiaccianti, si evince come i SAR potrebbero svolgere un ruolo importante nell'intervento non farmacologico per l'ipertensione, anche se sono necessari studi futuri per gli effetti a lungo termine

1.1.2 Il futuro dei SAR: considerazioni

Fino ad ora è stata effettuata una panoramica su questa tipologia di Robot che, come ogni elemento innovativo, porta con sé pregi e difetti. L'uso di essi, ad esempio, dovrebbe sempre essere affiancato da un costante supporto di un team di esperti, essenziale nel garantire un corretto utilizzo ed un adeguato approccio etico. In un contesto così delicato, quale la cura degli anziani o dei bambini, è necessario valutare attentamente quali obiettivi si vogliano raggiungere, come interfacciarsi al meglio con essi e le possibili implicazioni psicologiche e sociali.

Come si è evidenziato, i SAR possono alleviare il livello di solitudine e facilitare la socializzazione in contesti di gruppo, ma non devono diventare un sostituto delle relazioni umane autentiche; per questo è necessario mantenere una costante interazione umana. Devono quindi rappresentare un complemento, arricchendo l'esperienza, ma senza privare i contatti diretti tra soggetto e operatore.

Un altro tema importante è la personalizzazione dell'esperienza, fattore pienamente considerato nello sviluppo del progetto. Le sessioni di riabilitazione con i SAR non devono essere alienanti, ma focalizzate sull'utente e sulle sue esigenze specifiche.

"Lo sviluppo dei SAR dovrebbe mirare a un approccio "human first", in grado di garantire la creazione di un ecosistema terapeutico in cui i soggetti si conformino ai requisiti (e ai limiti tecnici) della tecnologia medica. Sistemi "intelligenti" e adattivi come i SAR devono essere progettati per promuovere il benessere umano e il raggiungimento di importanti obiettivi medici e sociali nella riabilitazione. L'integrazione di questa tecnologia nell'ambito specifico richiede la collaborazione tra clinici, soggetti dei test e ingegneri, per comprendere come i SAR possano essere considerati affidabili nel ricoprire ruoli specifici." [3] Di seguito è riportata un'immagine che illustra quanto descritto, in particolare il modo in cui la fiducia si sviluppa e si misura nell'interazione tra SAR ed esseri umani nel contesto della riabilitazione.

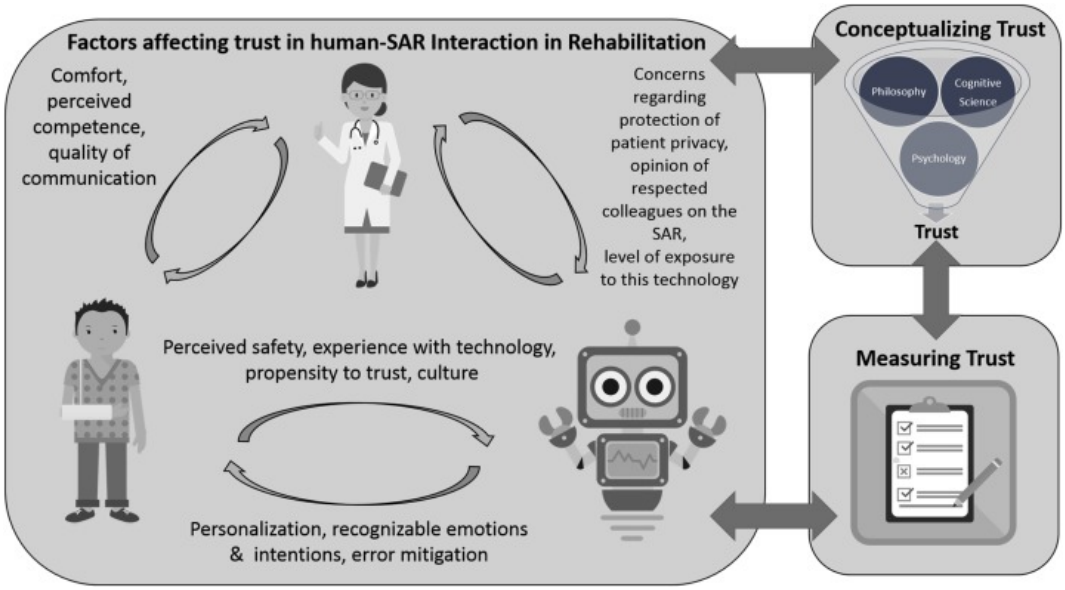


Figura 1.2: Immagine da: sciencedirect.com
[3]

Il futuro di questa tecnologia sarà quindi determinato dal grado di integrazione di tecnologie avanzate, con un approccio focalizzato sull'utente. La multidimensionalità dei SAR rappresenta un enorme potenziale, ma allo stesso tempo porta con sé la necessità di garantire efficienza e affidabilità, garantita solo da un costante sviluppo e monitoraggio per evitare che la tecnologia prenda il sopravvento sull'essenza umana.

1.2 NAO

Parliamo ora del protagonista del lavoro svolto: il robot NAO. NAO è un SAR sviluppato dalla Aldebaran, azienda francese con sede a Parigi. Nel 2013 l'azienda è stata acquisita dalla giapponese SoftBank Mobile. Dal 2008 è stata lanciata la Nao Academics Edition, una versione dedicata al mercato dell'insegnamento e della ricerca, per imprese e università; nel 2012 viene distribuito sul mercato per tutti, attraverso i canali della grande distribuzione.[18] NAO rappresenta appieno l'immagine di un SAR, un robot umanoide con le caratteristiche fisiche di un bambino, con un design rassicurante e amichevole.

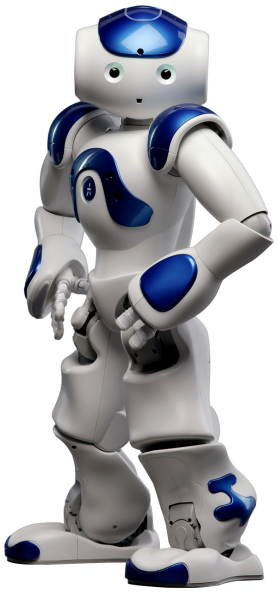


Figura 1.3: Immagine da: rootlab.com

L'uso di NAO si è diffuso in numerosi settori, tra cui l'educazione, la robotica sociale e la ricerca scientifica. Nelle scuole, NAO viene impiegato per avvicinare gli studenti alla programmazione e all'intelligenza artificiale, mentre in ambito terapeutico viene utilizzato per migliorare le capacità cognitive e sociali di persone con disturbi dello spettro autistico o problemi neurologici.

1.2.1 NAO: le caratteristiche Hardware

Si riportano nel dettaglio le sue caratteristiche tecniche; di seguito, una breve scheda tecnica.[6]

Caratteristiche tecniche	**Valore**
Altezza	58 cm
Peso	4,3 kg
Autonomia	90 min.
Gradi di libertà*	da 21 a 25
Microprocessore	x86 AMD GEODE a 500 MHz, ATOM 1, a 1,6 GHz a partire da Nao Next Gen
Sistema operativo integrato	Linux
Sistemi operativi compatibili	Mac OS, Linux, Windows
Linguaggi di programmazione	C++, Python, Urbi, Microsoft .NET
Connettività	Ethernet, Wi-Fi

* I gradi di libertà (o DoF, Degrees of Freedom) rappresentano il numero di movimenti indipendenti che un sistema può compiere nello spazio tridimensionale.

Tabella 1.1: Caratteristiche tecniche di NAO[18]

Un focus sulla testa di NAO:

- 2 altoparlanti, posizionati sulle orecchie
- 3 sensori tattili posizionati sulla testa
- 2 videocamere, una sulla fronte e una al posto della bocca
- 4 microfoni sensoriali, posizionati nella parte alta della testa
- 2 ricetrasmettitori a infrarossi con Led RGB, sugli occhi.

Come descritto, NAO dispone di due altoparlanti, che gli permettono di esprimersi in diverse lingue con una voce sintetizzata, molto simile a quella di un bambino, personalizzabile in intonazione e velocità. I suoi occhi a LED possono cambiare colore, offrendo un ulteriore canale espressivo per trasmettere emozioni e stati interni.

Per quanto riguarda il resto del corpo, troviamo:

- 26 motori
- 3 sensori tattili per mano
- 2 sensori tattili per ogni piede
- 2 sensori ad ultrasuoni

La mobilità del robot è garantita da una batteria ricaricabile che gli assicura un'autonomia di 60-90 minuti, variabile in base alle attività svolte. In aggiunta, è possibile utilizzare un alimentatore esterno. Il movimento avviene tramite due gambe articolate che gli permettono di camminare, sedersi e compiere movimenti complessi, mentre le braccia possono essere utilizzate per gesti espressivi o interazioni con oggetti leggeri.

Come si evince, grazie alla sua avanzata tecnologia e alla presenza di sensori, microfoni, videocamere, tutto ampiamente programmabile, si può avere il pieno controllo di ciò che circonda il robot, aprendo la strada ad una vasta gamma di possibilità.

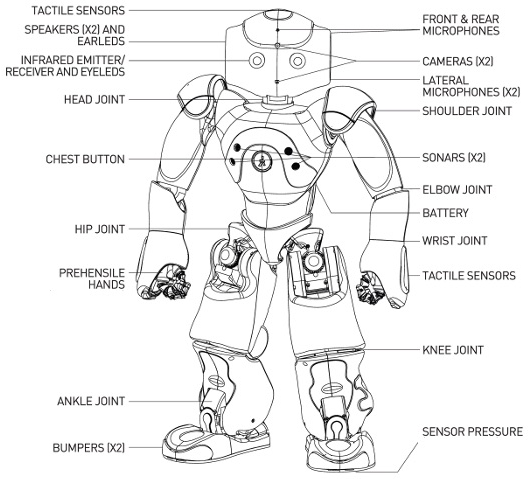


Figura 1.4: Immagine da: researchgate.net
[17]

1.2.2 NAO: le caratteristiche Software

Alla base del funzionamento di NAO troviamo NAOqi OS, una distribuzione basata su Linux progettata per la gestione delle funzioni del robot. Il sistema include NAOqi, un framework modulare che funge da nucleo per controllare tutte le sue funzionalità. Tramite il Software Development Kit (SDK) si può

interagire con le API del robot. È possibile anche utilizzare un'interfaccia grafica chiamata Choreographe, che permette di programmare il robot tramite blocchi visivi senza la necessità di conoscere un linguaggio di programmazione. In Choreographe è anche possibile creare dei moduli personalizzati, ma uno dei principali limiti del software è l'uso di Python 2.7, una versione ormai obsoleta e non più supportata ufficialmente; questa rappresenta una criticità significativa, poiché le nuove librerie attualmente in uso non vengono più sviluppate per questa versione, limitando l'espansione del sistema.[6]

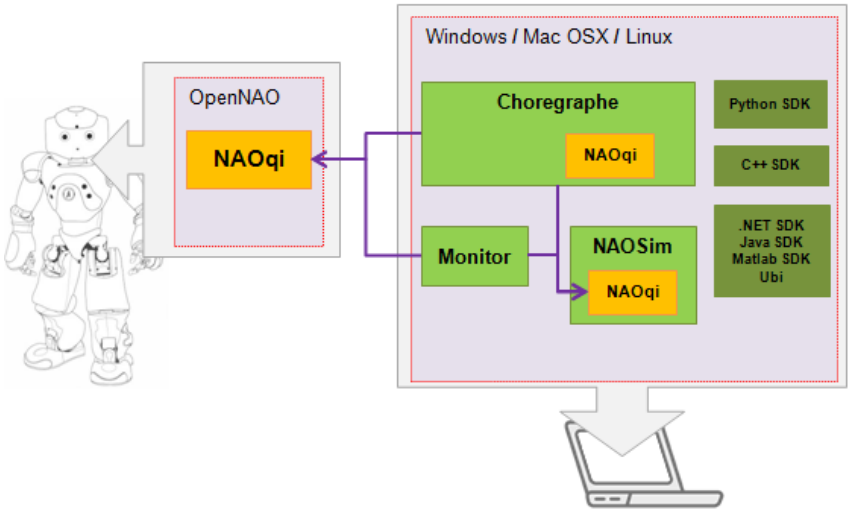


Figura 1.5: Immagine da: cs.emu.edu

1.2.3 NAO: i suoi problemi

Questa tipologia di robot umanoide si presta perfettamente all'educazione, all'apprendimento e alla riabilitazione; in particolare NAO, con il suo aspetto, è perfetto per l'interazione con i più piccoli, grazie anche alla sua capacità di esprimere emozioni, di conversare con una voce molto naturale e di adattarsi all'ambiente circostante.

Tuttavia, ci sono alcuni problemi che caratterizzano NAO, e più in generale l'interazione uomo-robot in questo ambito.

- Nonostante le sue potenzialità, NAO ha una capacità Hardware limitata, che ne condiziona significativamente le operazioni in compiti più complessi. Questo problema è stato ad esempio riscontrato nel progetto qui descritto.
- La durata della batteria rappresenta un ulteriore vincolo, infatti, per sessioni di lavoro più lunghe, NAO deve essere necessariamente collegato all'alimentazione.

- Il costo dei robot in questo momento è molto elevato, ciò limita l'adozione su larga scala di questo sistema, soprattutto in ambito educativo e sanitario.

- Quando si parla di tecnologia sorgono inoltre problemi di accettazione sociale, con l'idea che questi nuovi metodi di apprendimento/ riabilitazione, sfavoriscano le interazioni umane e producano effetti negativi sugli utilizzatori.

1.2.4 I vantaggi nell'uso di NAO: uno studio pilota

È stato già descritto come l'uso dei SAR in ambito clinico-sanitario possa rappresentare un grande vantaggio nell'interazione con esseri umani. In particolare, in uno studio condotto dall'università di Parma [16], è stato utilizzato proprio il robot NAO per valutare l'efficacia dello stesso nel trattamento dei disturbi cognitivi. I partecipanti, selezionati tra i pazienti ambulatoriali del Centro per i Disturbi Cognitivi e la Demenza di Parma, sono stati valutati da specialisti dei disturbi della memoria e sottoposti a screening per i criteri di inclusione, che richiedevano una diagnosi di Mild Cognitive Impairment (MCI), buoni punteggi nei test delle attività quotidiane (ADL e IADL), un'età compresa tra i 45 e gli 85 anni e l'assenza di trattamenti farmacologici. Essi sono stati coinvolti in un programma di allenamento della memoria della durata di 8 settimane, con incontri settimanali di 1,5 ore, condotti da un neuropsicologo esperto. Sono stati formati 3 gruppi, con un totale di 21 persone (10 donne e 11 uomini), a cui sono stati sottoposti dei compiti che rientravano nei task standard tipicamente svolti in un programma senza l'uso del robot.

Le ipotesi iniziali erano che:

- i programmi di memoria avrebbero portato a miglioramenti nell'autovalutazione della memoria, ma senza cambiamenti significativi nei test neuropsicologici

- gli anziani assistiti dal robot NAO durante l'allenamento avrebbero mostrato una valutazione migliore della memoria e livelli più bassi di ansia rispetto a quelli senza il robot.

Tuttavia, non sono emerse differenze significative nel comportamento dei pazienti durante l'interazione con il robot o lo psicologo.

Sorprendentemente, sono stati invece riscontrati cambiamenti significativi nelle misure di memoria narrativa e "fluency" verbale. Non è possibile stabilire se queste differenze siano dovute alla presenza di NAO, a causa del disegno dello studio, ma si suggerisce, per questo motivo, di estendere la ricerca a un campione più ampio per valutare meglio questi effetti. I soggetti, in generale,

hanno valutato positivamente l'usabilità di NAO, apprezzando la sua capacità di svolgere azioni e la sua adattabilità all'ambiente, soprattutto durante le frasi di rinforzo comunicate durante un compito (i feedback), piuttosto che durante l'esecuzione stessa. Questo suggerisce che si potrebbe arricchire l'approccio utilizzato, personalizzandolo, ad esempio pronunciando il nome del soggetto mentre queste vengono dette.

Questo studio dimostra che NAO rappresenta uno strumento innovativo per affrontare il crescente bisogno di approcci alternativi in questo ambito.

Come verrà analizzato più avanti, gli obiettivi del progetto di questa tesi rientrano esattamente in ciò che è stato visto in questo studio.

Capitolo 2

Struttura del progetto

Si passa ora al lavoro vero e proprio. Si inizierà descrivendo la struttura generale del progetto, senza tralasciare i pre-concetti, fondamentali per comprendere al meglio lo sviluppo generale, passando per aspetti più tecnici e analizzando le scelte e le implementazioni adottate. In questo capitolo sarà importante, inoltre, collegare i vari punti, delineando il contesto in cui si inserisce il progetto stesso, citando i contributi dei colleghi che hanno lavorato su aspetti specifici e analizzando tecnologie e punti forti.

Ciò permetterà di avere una struttura chiara di quanto realizzato, facilitando la comprensione dei risultati ottenuti e dei futuri sviluppi.

2.1 Gli obiettivi

Il progetto si inserisce in un contesto esecutivo che cerca di unire due campi: la tecnologia e i processi clinico-sanitari, insieme a tutto ciò che rientra nei termini della neuroscienza. Il risultato ottenuto è stato reso possibile grazie alla collaborazione del dipartimento di Medicina e Chirurgia, laboratorio di Psicologia Cognitiva, che ha fornito tutti gli strumenti e i metodi necessari per poter affrontare al meglio una "Sessione", insieme all'essenziale interazione con i soggetti e i vari "Task" da proporre loro.

Lo scopo del progetto è: progettare un programma di riabilitazione cognitiva strutturato in un ciclo di sessioni mirate al potenziamento di funzioni cognitive, come la memoria, in soggetti con Mild Cognitive Impairment (MCI), una condizione caratterizzata da un declino cognitivo lieve che può precedere la demenza, influenzando memoria e altre abilità cognitive [15]. L'efficacia dell'intervento sarà valutata attraverso procedure psicometriche appropriate per misurare i miglioramenti cognitivi e funzionali.

"Il termine riabilitazione cognitiva identifica interventi, strategie o tecniche, aventi la finalità di consentire ai beneficiari e alle loro famiglie di convivere,

amministrare, by-passare, ridurre e gestire i deficit cognitivi. Le tecniche di riabilitazione sono finalizzate a massimizzare la capacità di mantenere l'autonomia nel proprio ambiente con i limiti imposti dalla patologia, dal danno funzionale e dalle risorse disponibili e ad aiutare la persona ad adattarsi al nuovo status psico-fisico. Il fine ultimo dell'intervento è il miglioramento delle funzionalità nella vita di tutti i giorni." [13]

Ci sono vari modi attraverso i quali stimolare queste abilità. Nel contesto corrente verrà effettuato un ciclo di incontri, o *Sessioni*, nel quale il soggetto verrà sottoposto ad una serie di compiti, o *Task*, che progressivamente andranno ad aumentare di difficoltà, in base a come il soggetto stesso reagisce. Ogni qualvolta risponderà correttamente ai compiti proposti, gli verrà aumentato il punteggio, o score, nella categoria a cui appartiene quel task.

La soglia scelta per il superamento di queste è l'80%. I compiti, è importante sottolinearlo, sono stati sviluppati per allenare determinate capacità cognitive.

2.1.1 La struttura delle sessioni

Una sessione è, quindi, un incontro di un'ora o poco più, in cui vengono proposti casualmente una serie di compiti all'utente e il cui obiettivo è arrivare ad un punteggio minimo, per poter incrementare il livello di difficoltà e, di conseguenza, stimolare le capacità cognitive.

La prima sessione è detta: "*Sessione 0*" ed è per lo più conoscitiva. In questa prima fase i task non verranno scelti casualmente, ma saranno "prestabiliti", anche perché ogni sessione tiene conto delle precedenti. Questo per ribadire che, durante la prima sessione, la scelta del task è indifferente. In totale, i task sono 3 e ad ogni sessione verranno riproposti, evitando di selezionare gli stessi quesiti. Oltre a ciò, nella sessione 0 e prima di ogni task, verranno effettuate delle prove, per verificare che l'utente stesso abbia compreso le regole. Il risultato ottenuto, ovviamente, non verrà conteggiato nello score finale.

Si analizza ora la struttura di una generica sessione:

1. NAO si presenta e fornisce una spiegazione di cosa verrà fatto, introducendo le regole valide per tutta la sessione, chiedendo all'utente se ha compreso il significato e le regole; in caso di risposta negativa, verrà proposta una barzelletta, in caso di non risposta verrà invece ripetuta, in forma ridotta, l'introduzione.

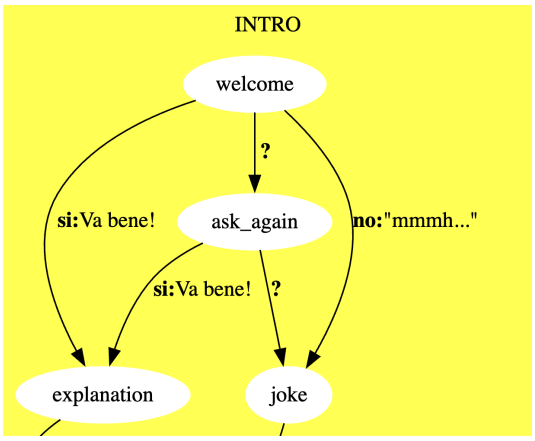


Figura 2.1: Introduzione di una sessione generica

2. Si entra poi nel vivo, selezionando uno dei possibili task tramite uno stato chiamato "*Task Chooser*", che funge da instradatore verso un task generico. Di seguito l'esempio di un task, tralasciando la sua struttura interna.

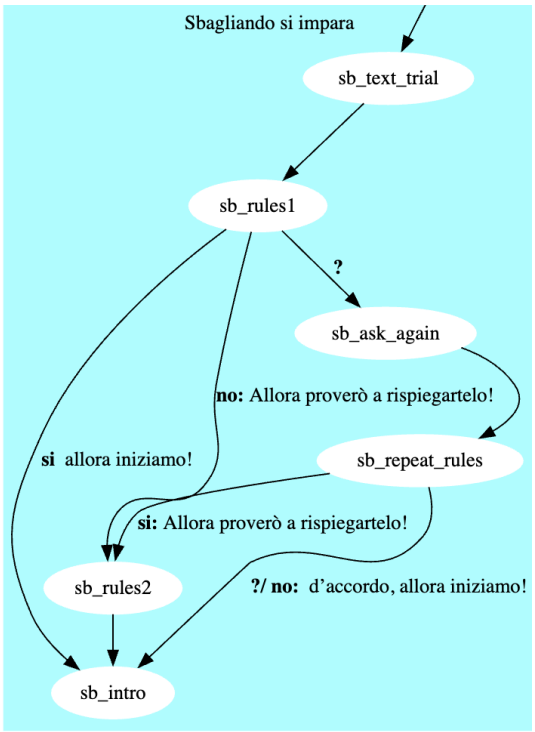


Figura 2.2: Esempio: Task Sbagliando si Impara

3. Alla fine di ogni task si passa ad una sezione che per semplicità è stata chiamata "*Middle/ End*", attraverso la quale si sceglie la successiva azione da intraprendere: se proporre un dialogo di intermezzo, spostarsi verso un altro task, o concludere. Durante la sessione, ci saranno due momenti diversi in cui verrà proposto un compito "bonus" chiamato "*Memoria Prospettica*", un tipo di task con un compito ben preciso e che verrà approfondito meglio nella sezione dedicata alla spiegazione dei Task.

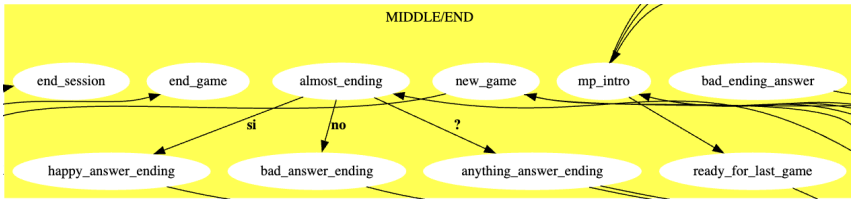


Figura 2.3: Sezione Middle/End

4. Dopo aver completato tutti i task si passerà, un'ultima volta nella sezione Middle/ End, per poter poi entrare nello stato di chiusura, nel quale verrà salutato l'utente con un messaggio positivo, utile a stimolare il soggetto, favorendolo ad affrontare al meglio la sessione successiva.

2.1.2 I dialoghi

Un punto cruciale nello sviluppo del progetto sono i dialoghi con l'utente, che rappresentano, per ora, l'unico mezzo di espressione di NAO.

Nel lavoro precedente, è stata dedicata una particolare attenzione alle modalità di espressione, al tono della voce e alla velocità, ma soprattutto alla sensibilità emotiva con cui certi messaggi venivano comunicati.[1] Bisogna tenere a mente che la riabilitazione cognitiva è un processo delicato, poiché si potrebbe avere a che fare con persone che presentano difficoltà nella comprensione e nell'attenzione. Quindi, è stato fondamentale progettare un modello di conversazione che fosse chiaro, semplice e coinvolgente. Per fare ciò, la collaborazione con il dipartimento citato in precedenza è stata cruciale; sono stati organizzati degli incontri nei quali è stato discusso il modello da utilizzare per le conversazioni, le frasi da non adoperare e le varie configurazioni, per rendere la voce di NAO il più amichevole e comprensibile possibile.

Dopo aver recepito i vari dialoghi e le frasi da utilizzare, sono stati fatti dei test tra i collaboratori del progetto, immedesimandosi nei soggetti, per verificare se i modelli funzionassero nella pratica.

Si illustra ora un esempio di dialogo per capire quanto detto:

Introduzione

Buongiorno X. Io mi chiamo NAO! Spero tu sia felice di essere qui con me oggi, vorrei ti divertissi cercando di migliorare le tue prestazioni cognitive ed io sono qui per aiutarti. A volte ci saranno compiti più semplici e altre volte più complicati, ma non ti preoccupare perché io sono qui per accompagnarti e se possibile aiutarti a fare sempre meglio. A volte anche io potrei fare errori, tutti ne possiamo commettere, tu ad esempio potresti aiutarmi alzando il tono della voce quando rispondi alle mie domande e utilizzando solamente le parole che io ti indicherò per le risposte.Bene, se sei d'accordo, io sono pronto ad iniziare. Tu sei pronto?

2.2 I Task

Si presenta ora un altro punto fondamentale del progetto: i compiti da proporre.

Come anticipato in precedenza, l'utente, durante tutto il corso della sessione, verrà messo alla prova con vari task, che variano in base alla tipologia e alla difficoltà. Ci saranno compiti in cui verrà coinvolto solo l'uso della parola, altri in cui verranno testati anche i riflessi, come ad esempio il battito di mani quando viene scandita una determinata parola.

Ogni task sarà introdotto da un suo Jingle caratteristico, questo per identificare ogni compito e far in modo di creare un'associazione suono-task, utile per coinvolgere l'utente in maniera attiva e stimolare più aree del cervello.

Si ricordi inoltre come ogni task, solo nella sessione 0, venga introdotto da una prova e che inizialmente la difficoltà sarà impostata al livello "Facile", considerandolo superato solo se l'utente oltrepassa la soglia dell'80%.

La struttura generale di tutti i compiti segue un modello ben definito. Dopo la presentazione del compito, verrà posta una domanda, a cui l'utente potrà rispondere sì o no. Da qui, il percorso può svilupparsi in diversi modi:

- se l'utente risponde "no", verranno rispiegate le regole in un formato più breve e con un linguaggio più semplice. Dopo la spiegazione non ne verrà più fornita un'altra e si comincerà immediatamente con il compito.
- se l'utente risponde "sì", il task comincerà subito, senza ulteriori spiegazioni
- nel caso in cui si riceverà una risposta diversa da sì o no, o non verrà percepito nessun suono da NAO, verrà ricordato di rispondere solo con sì o no e la domanda verrà riproposta.

2.2.1 Batti un colpo

Il primo task da presentare è "*Batti un colpo*", un compito che coinvolge non solo le capacità cognitive, ma anche quelle motorie e di attenzione.

Il task è diviso in due fasi: nella prima verrà raccontata una breve storia al cui interno sarà presente una parola definita "Target". Quando NAO pronuncia questa parola, viene aperta una breve finestra di ascolto, durante la quale l'utente deve battere le mani. Nella seconda fase, a storia terminata, dovrà invece rispondere ad una domanda inerente alla storia appena ascoltata. Entrambe le fasi contribuiscono al punteggio finale, con diverse percentuali.

Vediamo un esempio:

Storia:

Mario e Luca sono amici da quando erano bambini. Dopo alcuni anni, si sono rincontrati al bar insieme ai propri bambini ed hanno iniziato a raccontargli i giochi che loro facevano da piccoli.

Target: Bambini

Domanda: Dove si sono rincontrati Mario e Luca?

In questa storia, ad esempio, l'utente dovrà battere le mani per 4 volte e, successivamente, rispondere alla domanda proposta. Lo scopo del compito è quindi duplice: stimolare l'attenzione e la reattività e poi incoraggiare l'elaborazione e la memoria.

2.2.2 Sbagliando si impara

Il secondo task da presentare è "*Sbagliando si impara*", un compito che si ispira a quello proposto nel famoso QuizShow "Avanti un altro". In pratica, verranno poste una serie di domande e l'utente avrà a disposizione due opzioni di risposta: l'obiettivo è scegliere quella **sbagliata**. Ovviamente, lo score aumenterà in base a quante risposte "sbagliate" si daranno. Ogni volta che l'utente non segue la regola ricomincerà dall'inizio.

In base al livello di difficoltà, il numero di domande cambia, ma in tutte le difficoltà il tempo massimo è di **100 secondi**, mentre quello concesso per

ogni risposta è di **10 secondi**; se l'utente supera il limite dei 10 secondi, la domanda verrà considerata sbagliata.

Ecco un esempio di domanda:

Risposta: Quattro o Cinque

Domanda: Quanto fa due più due?

In questo caso quindi, la risposta giusta è quella sbagliata, quindi sarà "**Cinque**". Si noti anche come NAO comunicherà prima le due possibili risposte, poi la domanda.

2.2.3 Riordinando

Riordinando è un task il cui titolo dice tutto: riordinare degli elementi in un insieme o categoria all'interno di un ordine prestabilito. L'obiettivo del task è verificare le capacità del soggetto di comprendere e applicare una sequenza logica, per organizzare gli elementi in modo coerente.

Dopo la consueta intro, in cui verranno spiegate le regole del task, si esprimerà la richiesta di esecuzione del compito, in coerenza con il tipo di ordinamento e di elementi da riordinare in una determinata categoria.

Al termine della richiesta verrà fornito all'utente del tempo per poter riflettere sulla risposta da dare e durante questo intervallo, verrà riprodotto un suono, studiato appositamente per fornire concentrazione. Dopo la pausa, si avrà il tempo necessario per poter rispondere alla domanda.

Vediamo ora un esempio:

Domanda: Elenca in ordine dalla più fredda alla più calda o dalla più calda alla più fredda le seguenti stagioni: autunno, inverno, primavera, estate.

2.2.4 Memoria Prospettica

L'ultimo compito da presentare è "*Memoria Prospettica*", un task che si differenzia dagli altri, in quanto si concentra sulla capacità di ricordare di eseguire un'azione in futuro o nei casi in cui vanno verificate determinate condizioni. Di conseguenza si misura e si valuta una particolare abilità nel soggetto, fondamentale nella vita di tutti i giorni, e particolarmente rilevante tra coloro

che evidenziano particolari difficoltà cognitive o disturbi della memoria, poiché questa condizione influenza direttamente la qualità della vita.

La prova in essere si differenzia non solo per lo scopo, ma anche per la modalità di esecuzione; infatti, il test viene somministrato in un momento preciso della sessione, durante la quale viene pronunciato un termine da associare ad una determinata espressione e in un secondo momento, viene ripetuta la parola di riferimento, sulla quale l'utente dovrà fornire la soluzione associata (talvolta, invece della parola, può essere associata anche un'azione da eseguire).

Target: Insegnante

L'utente risponde: Scuola

2.3 I Feedback

I Feedback rappresentano un altro importante punto del progetto, poiché rappresentano un mezzo fondamentale per il soggetto, non solo per poter comprendere le proprie risposte e migliorare progressivamente le proprie capacità, ma anche per motivarlo quando viene data una risposta corretta.

Di seguito un estratto di un tesista [11] del dipartimento con cui il lavoro è in collaborazione e che ha lavorato al medesimo progetto, focalizzandosi sulle modalità di espressione di NAO.

Si analizzano i risultati di uno studio condotto per analizzare l'effetto di diversi tipi di feedback verbale sull'accettabilità di un robot utilizzato per attività di training cognitivo e fisico.

I partecipanti sono stati suddivisi in gruppi in base alla tipologia di feedback ricevuto:

- Gratificante: frasi positive rivolte all'utente senza necessariamente basarsi sulle sue reali abilità o qualità.
- Positivo: rinforzi incoraggianti sulle capacità dell'utente.
- Negativo: feedback critico sulle prestazioni.

I risultati hanno mostrato che i partecipanti che hanno ricevuto un feedback gratificante o positivo hanno espresso valutazioni più alte riguardo:

- atteggiamento verso la tecnologia
- intenzione di utilizzo

- utilità percepita
- influenza sociale.

Tutti questi aspetti sono stati misurati tramite un questionario di accettazione del robot e risultavano migliori rispetto al gruppo che aveva ricevuto un feedback negativo.

Il risultato è che: il feedback del robot sulle prestazioni degli utenti influisce su esecuzione del compito, divertimento, impegno e motivazione.

A tal proposito, come per tutto il progetto, il team del dipartimento di Medicina e Chirurgia, laboratorio di Psicologia cognitiva, ha fornito una serie di feedback positivi, negativi e intermedi, che permettono di mantenere un linguaggio che non miri tanto alla mera valutazione della risposta, quanto piuttosto a motivare l'utente, fornendo un'esperienza motivante e costruttiva.

Ecco degli esempi ripresi dal task "Batti un colpo" e "Sbagliando si impara":

Feedback negativo:

Non hai battuto le mani! Ricordati che ogni volta che senti la parola TARGET devi battere le mani, ma non ti preoccupare, siamo qui per migliorare!

Feedback positivo:

Grande X! Hai fatto un ottimo lavoro!

2.4 Le emozioni e il loro ruolo

Le emozioni sono un punto fondamentale del progetto, aggiunto di recente, per arricchire le interazioni con l'utente. Grazie al lavoro di un precedente tesista [19], è possibile verificare costantemente lo stato emotivo del soggetto grazie a una delle telecamere di NAO, adattando dinamicamente il flusso del dialogo in base alle emozioni espresse.

A riguardo, ci sono diversi studi effettuati che dimostrano l'efficacia del rilevamento delle emozioni durante un programma di training cognitivo, assistito da un robot sociale, con soggetti con Mild Cognitive Impairment (MCI). In particolare, da dei test effettuati all'università di Bari, in collaborazione con il Dipartimento di Medicina e Chirurgia dell'Università di Parma, [4] e [14], si evincono risultati positivi. Il software risulta essere affidabile nel riconoscere

le espressioni facciali, aggiungendo nuove evidenze sulle variabili coinvolte nell'Interazione Uomo-Robot (HRI). Il test, della durata di 2 mesi e in cui sono stati coinvolti 21 soggetti, prevedeva il rilevamento delle emozioni tramite robot NAO. I video sono stati campionati a **1 frame al secondo** e per ciascun partecipante sono stati analizzati **28.800 frame** (1 frame/secondo X 8 ore di registrazione). Le metriche utilizzate per valutare il riconoscimento delle espressioni sono:

- **nFD (number of detected faces)** → Numero di volti rilevati
- **nFE (number of facial expressions recognized)** → Numero di espressioni facciali riconosciute per ogni frame e per ogni partecipante

Il sistema è riuscito ad analizzare tutto il corpus video rilevando **tutti i volti** dei 21 partecipanti, con una percentuale di successo nella rilevazione (nFD) del **56%**. Per ogni frame analizzato, anche se parzialmente occluso (il soggetto indossava occhiali o aveva le mani davanti al volto), il sistema è stato in grado di riconoscere un'espressione facciale. Per quanto riguarda le espressioni rilevate, le più frequenti sono state:

- Neutrale: 41% dei frame
- Felicità: 17% dei frame
- Tristezza: 15% dei frame

Nel contesto del lavoro corrente, non sono state ancora codificate le modalità di alterazione del dialogo tramite l'uso delle emozioni ma, in futuro, tali adattamenti avranno un ruolo cruciale nel migliorare l'efficacia del sistema, permettendo una maggiore empatia e un'esperienza più coinvolgente.

2.5 Explainable AI (XAI): Un approccio basato su ASP e automi deterministici

In questa sezione si analizzerà brevemente come sono stati implementati i task e, più in generale, l'intera struttura della conversazione nel programma. Il componente responsabile della gestione del flusso del dialogo e delle scelte intraprese nel corso della conversazione prende il nome di "*Ragionatore*", termine che da ora in poi verrà utilizzato durante tutto il lavoro per riferirsi a questa specifica figura.

L'Explainable AI (XAI) si focalizza sulla spiegabilità dei modelli di IA tradizionali, sfruttando il processo decisionale dei modelli e i risultati delle loro previsioni. [20]

Per realizzare un sistema trasparente, con regole chiare e definite, ci si avvelerà dell'utilizzo di "ASP", acronimo di "*Answer Set Programming*", e per la struttura verrà utilizzato un "DFSA", acronimo di "*Automa a Stati Finiti*".

L'obiettivo di questa sezione è, quindi, fornire una panoramica tecnica delle tecnologie utilizzate per l'implementazione di questa parte fondamentale del progetto.

2.5.1 ASP

ASP è un paradigma di programmazione dichiarativa rivolto alla risoluzione di problemi di ricerca complessi.[23] Nel progetto corrente acquisisce un ruolo centrale poiché, tramite esso, è possibile riprodurre la logica e la personalizzazione dei dialoghi, dei task e dei loro livelli di difficoltà. La sua dinamicità e capacità di esprimere esattamente ciò che succede all'interno del contesto è enormemente efficace per adattare il sistema alle specifiche esigenze dei soggetti.

2.5.2 Modellazione di una sessione: I DFSA

Di seguito ciò che serve per poter implementare quanto descritto. Iniziamo dalla struttura, modellizzata tramite un'architettura informatica chiamata Automa a stati finiti, o DFSA, un modello computazionale che consente di rappresentare rigorosamente il flusso logico delle operazioni.

Un "Automa a Stati Finiti Deterministico (DFSA)" è definito formalmente come una quintupla[22]:

$$\mathcal{A} = \langle \Sigma, S, \delta, s_0, F \rangle \tag{2.1}$$

dove:

- $\Sigma = \{a_0, a_1, \ldots, a_n\}$ è un insieme finito di simboli, chiamato **alfabeto**;
- $S = \{s_0, s_1, \ldots, s_m\}$ è un insieme finito di **stati**;
- $\delta : S \times \Sigma \to S$ è la **funzione di transizione** tra stati;
- $s_0 \in S$ è lo **stato iniziale**;
- $F \subseteq S$ è l'insieme degli **stati finali** o terminali.

Tramite questa struttura è possibile muoverci all'interno del flusso in maniera organizzata e controllata: gli stati rappresentano delle fasi distinte della sessione: domande, analisi della risposta, stato di conferma, passando dal precedente al successivo tramite la prima descritta funzione di transizione.

Di seguito la struttura adottata per il DFSA:

- state: indica lo stato corrente del sistema
- answer_type: tipo di risposta dell'utente
- listen_user: rappresenta un booleano yes/no e indica se il sistema aspetta una risposta dall'utente
- new_state: il nuovo stato che il sistema adotta in base alla risposta dell'utente.

In questo modo si ha il controllo sul flusso: ci si sposterà all'interno di esso in base a ciò che succede all'interno della conversazione.

2.6 Gli script di conversione

In questa sezione si descrive brevemente come si arriva dai concetti teorici all'implementazione della struttura del ragionatore.

Si vedrà, in particolare, una serie di script, sviluppati su Python dalla precedente tesista [1], in grado di convertire un file Excel in altre tipologie di file, modellando i dati all'interno di un DFSA.

2.6.1 Da Excel ad ASP

Si inizia con la descrizione di uno script che permette di convertire un file Excel in un file `.asp`. Il primo passo è la lettura, tramite la libreria `pandas`, del file Excel, che viene poi caricato in un "*DataFrame*", una struttura dati tabellare, eseguendo una normalizzazione del testo, rimuovendo gli spazi e convertendo tutte le lettere in minuscole per prevenire errori e rendere il tutto uniforme. Si estrae poi il valore della cella `A2` dalla colonna `da` e si definisce quel valore come stato iniziale dell'automa.

Si itera poi sulle righe del DataFrame per trasformarle in una dichiarazione ASP con i campi elencati in precedenza, eseguendo una standardizzazione. In particolare, viene eseguito un controllo sul valore `answer_type`:

- Se `answer_type` contiene il simbolo **?**, esso viene sostituito con la stringa `unknown`
- Se `answer_type` è vuoto o contiene il valore `nan`, viene anch'esso convertito in `unknown`
- Se `answer_type` contiene espressioni logiche come `if, ==, o ||`, viene trasformato in `unknown`.

Per ogni condizione viene creata e scritta nel file .asp una riga di questa forma: `dfsa(state, answer, listen user, new state)`.

2.6.2 Da Excel a JSON

Uno script simile è stato utilizzato per convertire un file Excel in un file `JSON`, necessario per memorizzare un dizionario contenente frasi da confrontare con le risposte dell'utente.

La funzione principale trasforma il DataFrame in un dizionario JSON, eseguendo prima una verifica e una standardizzazione del testo (rimuovendo spazi e uniformando le lettere), e poi utilizzando la prima colonna come chiave e la seconda come valore. Infine, il file JSON viene salvato nella stessa cartella del file Excel con il nome `generics.json`, correttamente formattato e indentato.

2.6.3 Conversione dei Task

Per i Task, la questione è più ampia e articolata, poiché ciascun Task segue una struttura prestabilita e richiede l'implementazione di specifici script di conversione. Per questo motivo, si eviterà di approfondire nel dettaglio ogni singolo script, ma si fornirà soltanto una panoramica del lavoro svolto, evidenziando i punti comuni che caratterizzano l'intero processo. Gli script condividono diverse caratteristiche fondamentali.

In primo luogo, si occupano di caricare file Excel ed effettuare una pulizia preliminare, come la rimozione di spazi indesiderati nelle colonne, per garantire la corretta elaborazione dei dati. Un altro aspetto essenziale riguarda la separazione e la riorganizzazione delle informazioni, ad esempio suddividendo ID e difficoltà in modo chiaro e strutturato. Successivamente, ogni script elabora le righe del DataFrame, applicando trasformazioni specifiche in base ai requisiti del Task, assicurandosi che i dati vengano trattati correttamente prima di essere esportati. Una volta elaborati, i dati vengono scritti in un file ASP con un formato predefinito, in modo da garantirne la compatibilità con il sistema di destinazione.

Un ulteriore elemento cruciale è la gestione degli errori, integrata per evitare problemi di lettura dei file e per assicurarsi che tutte le colonne richieste siano presenti e correttamente formattate.

Questa struttura consente di standardizzare il processo di conversione, mantenendo, al contempo, la flessibilità necessaria per adattarsi alle caratteristiche di ogni singolo Task. L'obiettivo principale è **ottimizzare l'elaborazione dei dati, rendendola il più efficiente e robusta possibile.**

Capitolo 3

Background

Una volta fatta chiarezza sulla struttura del progetto, possiamo dedicare il capitolo corrente alla spiegazione della parte più tecnica che lo riguarda.

In particolare, verranno illustrate tutte le parti che compongono il lavoro, descrivendo le varie sezioni che costituiscono il software. Verrà posta un'attenzione particolare al protagonista del progetto, NAO, e all'implementazione tecnica delle componenti che, collegate fra loro, danno vita alla sessione vera e propria.

Tutto ciò che verrà menzionato è frutto di un lavoro svolto da diversi tesisti, che hanno lavorato su parti diverse del progetto, fornendo una base solida per sviluppi futuri.

3.1 Le scelte implementative

In questa breve sezione si descriveranno le scelte implementative che sono state prese durante il processo di sviluppo e le motivazioni che hanno portato a prenderle.

3.1.1 NAO: le limitazioni

Si parte sviluppando un'analisi di NAO, con le sue problematiche. Come visto nel capitolo 1, le caratteristiche hardware di NAO sono limitate e non lasciano spazio a molta elaborazione.

Per questo motivo, l'obiettivo del lavoro corrente, che vedremo successivamente, è spostare la computazione e il lavoro "duro" sul PC e lasciare al NAO solo ciò che è strettamente necessario, come ad esempio i movimenti e la riproduzione dei file audio.

Questa scelta permette di sfruttare la potenza di un calcolatore per eseguire azioni, anche pesanti o che prevedono l'uso della GPU, e non far surriscaldare il NAO, visto che questo aspetto rappresenta una sua nota dolente.

Un'altra significativa problematica riguarda l'uso dei microfoni, che non consentono di registrare audio di alta qualità. Inizialmente, questo limite è stato superato mediante l'impiego di un microfono esterno, ma, come approfondiremo più avanti, tale soluzione non è più necessaria.

Per la connessione con NAO, ci sono 2 alternative: collegare NAO al WiFi o utilizzare un cavo Ethernet dietro la sua testa. Se si sceglie la connessione WiFi, ovviamente, la latenza sarà maggiore e, in più, l'antenna di NAO non è delle migliori, ma si avrà il vantaggio di non avere cavi che possono dare fastidio durante la sessione. L'approccio tramite Ethernet, al contrario, permette di avere meno latenza e quindi più velocità, ad esempio nello scambio di dati, ma si avrà costantemente un cavo attaccato al NAO, oltre a quello dell'alimentazione. È importante quindi valutare la scelta più adeguata e scegliere cosa sacrificare.

Per quanto riguarda lo sviluppo, è stato scelto di utilizzare due principali linguaggi di programmazione: Python e C++.

Perché utilizzarne due diversi? Nel primo capitolo si è accennato al fatto di poter sviluppare dei propri script utilizzando le API che NaoQi mette a disposizione, ma che, per quanto riguarda Python, queste non risultano aggiornate. Questa limitazione impone di dover utilizzare C++ per l'interazione con NAO e Python per implementare gli script di gestione del flusso.

3.1.2 La libreria in C++

Avendo due linguaggi di programmazione distinti era necessario stabilire un modo per utilizzare le funzioni scritte in C++ su Python. Chi ha lavorato al progetto in precedenza ha quindi pensato di scrivere una libreria in C++, contenente solo le funzioni di interfacciamento al NAO: l'inizializzazione dei Proxy, il movimento, la gestione del parlato... Si noti come questa libreria venga compilata come "*Shared Library*", con, ad esempio, estensione `.so` su Linux, consentendo l'uso multipiattaforma, quindi utilizzabile su qualunque sistema operativo. La conversione avviene, invece, tramite una libreria chiamata `ctypes`, la quale fornisce tipi di dati compatibili con il C++ e permette di richiamare funzioni in librerie condivise. Può quindi essere utilizzata per creare Wrapper per queste librerie, interamente in Python.[9]

Questo permette, quando se ne ha bisogno, di utilizzare la potenza e la semplicità di Python per gestire il flusso del programma e utilizzare la libreria "*TensorFlow*" per il riconoscimento delle emozioni, richiamando le funzioni

di interfacciamento al NAO tramite il Wrapper creato appositamente per la libreria C++.

3.2 Le componenti del progetto

Per entrare nell'ottica del progetto, ci si pone una semplice domanda: di cosa ha bisogno NAO per sostenere una sessione? L'obiettivo è **riprodurre l'attività di ausilio e assistenza così come verrebbe svolta da un esperto.** È quindi necessario umanizzare NAO e renderlo il più vicino possibile ad un oggetto di cura e conforto riabilitante. Si elencano alcune delle caratteristiche da considerare:

- dialogo, con tempi di risposta veloci e naturali
- visione dell'utente, con riconoscimento di emozioni
- dinamicità nel flusso del dialogo
- interazione multi-modale (gesti, movimenti, dialogo).

Sono stati quindi chiariti i punti su cui è indispensabile lavorare per implementare un sistema coerente e funzionale e che si avvicini il più possibile ad un'interazione umana. Alcune delle componenti elencate sono state realizzate e rese funzionanti, altre sono in fase di sviluppo e altre ancora saranno aggiunte in futuro.

Nelle successive sezioni si vedrà cosa è stato fatto, partendo dai punti già affrontati.

3.2.1 Il dialogo uomo-macchina

L'implementazione di un dialogo con una macchina non è immediata e richiede una particolare attenzione, sia per le scelte che vengono prese, sia per il modo in cui avviene. Bisogna focalizzarsi, prima di tutto, sul tipo di macchina con cui si ha a che fare, su cosa essa ci mette a disposizione e come rendere più semplice il percorso.

Si descrive brevemente come si traduce in informatica il flusso del dialogo:

1. il primo passo è far parlare la macchina, questo avviene tramite un procedimento chiamato "Text To Speech", in cui una frase, o un qualunque file di testo, viene trasformato in un file audio
2. una volta che è stata posta una domanda che prevede una risposta, il secondo step è registrare il riscontro al quesito, tramite un microfono, e renderlo disponibile alla macchina per la riproduzione

3. per miglior definizione, il procedimento che si occupa della trascrizione del file audio, è chiamato "Speech to Text". In successione, una volta comunicata la risposta, il ciclo si ripete, creando così un flusso di dialogo.

Inizialmente, la parte che riguarda il Text To Speech, funzionava tramite un modulo Socket[1][10], che inizializzava il PlayerProxy di NAO, impostando porta e indirizzo IP del socket server (lato Choreographe), ed inviando il file audio da far riprodurre al NAO, una volta ricevuto. Questo limitava particolarmente la procedura poiché era necessario tenere costantemente aperto il software Choreographe per mantenere il socket.[12]

Lato Speech to Text, è stata messa in evidenza la questione dei microfoni di NAO e dei loro problemi; per questo motivo, inizialmente, è stato scelto di utilizzare un microfono esterno per l'acquisizione dell'audio, limitante anche questo perché imponeva di utilizzare un dispositivo esogeno al sistema. Veniva poi trasformato il file audio in testo ed elaborato per un'eventuale risposta da parte del NAO. Il computer si occupava, quindi, dell'acquisizione, dello Speech To Text e del Text To Speech (compreso l'invio dei file audio tramite socket).

Il Text To Speech: l'implementazione precedente

Una volta fatta chiarezza sul funzionamento del Text To Speech, si procede ad analizzare l'implementazione del lavoro precedente e le librerie utilizzate.[12]

Il modulo `text_to_speech` utilizza due sotto-moduli per poter convertire il testo in parlato: `text_to_speech_google` (`t2sg`), online, che richiama le API di Google, e `pyttsx4`, un'alternativa offline che si appoggia ai sintetizzatori del sistema operativo. Il secondo viene importato e utilizzato solo se il primo fallisce.

[1]un socket è un'interfaccia di programmazione fornita dai protocolli TCP e UDP per la comunicazione a flusso e a datagramma, rispettivamente, nel livello di trasporto, che fa parte dello stack TCP/IP

Algoritmo 1 Funzione Text to Speech precedente

```
1: function TEXT2SPEECH(input, language)
2:     result ← ""
3:     print ("Provo con Google API")
4:     try:
5:         result ← Esegui text2speech di Google con (input, language)
6:     except errore:
7:         print ("Errore nell'utilizzo di Google API:", errore)
8:         Importa il modulo text2speech_pyttsx4
9:         print ("Provo con pyttsx4")
10:        try:
11:            result ← Esegui text2speech di pyttsx4 con (input)
12:        except errore:
13:            print ("Errore nell'utilizzo di pyttsx4:", errore)
14:    return result
```

Una volta restituito il risultato e convertito, eventualmente, nel formato corretto, si ottiene un file audio che viene inviato a NAO tramite Socket. Il processo è simile per entrambi i moduli, ma con qualche differenza nelle tempistiche, essendo il primo online.

A questo proposito, è importante sottolineare come le due opzioni siano strettamente collegate al tono di voce di NAO; infatti, potrebbe succedere che, durante la conversazione, la voce del robot cambi improvvisamente, proprio perché i due moduli offrono sintetizzatori e impostazioni di voce diversi.

Lo Speech To Text: l'implementazione precedente

La metodologia per l'implementazione dello Speech To Text è la stessa: un sistema di rollback che si aziona nel momento in cui il primo non restituisce una risposta. [12]

In questo caso, viene creato un oggetto `recognizer`, ottenuto dal modulo `speech_recognition`, attraverso il quale possiamo utilizzare diverse API. La prima scelta è ricaduta sull'API di Google, molto funzionale per ottenere una buona accuratezza, e `Vosk`, che rappresenta un'ottima alternativa per il riconoscimento offline.

Tramite l'oggetto `recognizer` si può accedere al microfono esterno, di cui si era già parlato, e impostare dei valori per la calibrazione. Viene poi richiamata la funzione `listen(source)` per registrare l'audio.

Da qui in poi la logica è analoga al Text To Speech, richiamando prima Google e in caso di errore Vosk. Una volta ottenuto l'output testuale, esso viene

Algoritmo 2 Speech To Text precedente

```
1: Definisci lang ← ""
2: if language.lower() == "italiano" OR language.lower() == "italian" then
3:     lang ← "it-IT"
4: else
5:     lang ← "en-EN"
6: Crea un riconoscitore
7: recognizer ← sr.Recognizer()
8: Acquisisci audio dal microfono
9: with sr.Microphone() as source:
10:    recognizer.energy_threshold ← 300
11:    recognizer.adjust_for_ambient_noise(source)
12: Registra l'audio
13: print("Parla...")
14: audio_data ← recognizer.listen(source)
15: Salva l'audio in un file temporaneo
16: audio_file ← "temp_audio.wav"
17: with open(audio_file, "wb") as f:
18:    f.write(audio_data.get_wav_data())
19: ...
```

elaborato direttamente dalla macchina locale e dato in pasto al ragionatore, per ottenere, eventualmente, una risposta da NAO.

Si vedrà nel capitolo successivo com'è stato, invece, ottimizzato il dialogo per eliminare l'uso di dispositivi esterni e del socket per l'invio del materiale.

3.2.2 Il modulo NLTK

È stata già esposta la procedura di funzionamento dell'idea di dialogo su NAO (Text To Speech e Speech To Text), ma come fa NAO a tradurre tutto ciò che viene detto, elaborandolo e trasformandolo in scelte da prendere?

Su questo punto viene in aiuto il *"Natural Language Toolkit"*, più comunemente conosciuto come NLTK, una suite di moduli Python che offre numerosi tipi di dati per l'NLP (Natural Language Processing), attività di elaborazione, oltre ad algoritmi animati, tutorial e insiemi di esercizi.[7]

Il modulo NLTK permette di costruire un modello di comparazione adatto allo scopo, utile a confrontare le risposte date con quelle che il robot si aspetta.

A tal proposito è stato costruito un modulo `compare`, che esegue dei confronti tra la frase detta e ciò che la macchina si aspetta, seguendo questo schema:[12]

1. Si esegue una "*Tokenizzazione*", cioè si trasforma la frase detta in singole parole che andranno inserite in una lista.

2. Si controlla se ci sono corrispondenze perfette.

3. Se non si trovano corrispondenze esatte, si valutano, per ogni parola, i synsets, cioè sinonimi e parole con lo stesso significato (auto e macchina, ad esempio).

4. Si cerca, poi, la coppia di parole con la corrispondenza migliore: se sono sinonimi allora c'è corrispondenza, altrimenti si utilizzano gli alberi semantici. Se si risale meno di 2 livelli, allora la corrispondenza è considerata accettabile, altrimenti il risultato sarà considerato insufficiente.

3.2.3 Il riconoscimento di emozioni tramite AI e il tracciamento della testa

NAO mette a disposizione 2 telecamere, una posizionata al livello della fronte e una tra gli occhi e il mento, utilizzabili a piacimento, per seguire ad esempio la testa dell'utente o riconoscere oggetti che ha davanti.

Nel corso del progetto, tramite l'uso di una rete neurale addestrata appositamente per questo scopo, è stato creato un modello in grado di riconoscere e classificare l'emozione della persona seduta davanti a NAO, tramite una delle due.[19]

Le emozioni che vengono riconosciute dal modello sono 8:

- Rabbia
- Disprezzo
- Disgusto
- Paura
- Felicità
- Neutralità
- Tristezza
- Sorpresa

Tramite queste, sarà possibile alterare il flusso del dialogo per intraprendere azioni diverse, in base all'emozione riconosciuta. L'integrazione nel sistema sarà infatti di fondamentale importanza ai fini del lavoro svolto, poiché il

soggetto, durante il corso della sessione, potrebbe annoiarsi o mostrare segni di fastidio. È quindi necessario monitorare lo stato del soggetto stesso durante le risposte che precedono l'inizio di un task.

Saranno fondamentali ulteriori valutazioni per capire esattamente in che modo le emozioni potranno essere integrate nel flusso del dialogo, ma per ora ci si limiterà ai casi base.

Oltre al riconoscimento degli stati emotivi, è stato implementato anche un meccanismo di tracciamento della testa, per seguire l'utente in caso di movimento, mantenendo il campo visivo di NAO centrato sull'utente.

Si sottolinea che il modello in questione, prima del lavoro recentemente sviluppato, era indipendente dal resto e non integrato nel flusso del dialogo.

3.2.4 La connessione a NAO

Nei primi capitoli è stato accennato a come poter implementare la struttura del lavoro, ma resta di sicuro ausilio osservare, ora e nel dettaglio, cosa serve sapere per comprendere al meglio le varie componenti che interagiscono tra di loro.

Come anticipato, è possibile creare dei propri script utilizzando le API che vengono fornite con NaoQi, il sistema operativo del robot, per controllarne i movimenti, i dialoghi, i sensori...

Le API di NaoQi sono disponibili in diversi linguaggi di programmazione, tra cui Python e C++, elementi essenziali nel nostro lavoro.

Alcuni moduli di NaoQi:

- ALAudioPlayer: gestisce la riproduzione di file audio dal NAO
- AlVideoDevice: gestisce l'acquisizione delle immagini dalla videocamera del NAO
- ALMotion: gestisce i movimenti del NAO, è possibile quindi controllare i suoi motori

I moduli sono disponibili sotto forma di Proxy, come oggetti che permettono di interagire con NAO in remoto.

Esempio di collegamento con un modulo:

In questo codice si può osservare come avviene la connessione ad un modulo di NaoQi tramite Proxy, in questo caso ALAudioPlayer.

La prima funzione permette di inizializzare tutti i Proxy che verranno utilizzati all'interno del programma, richiamati nei metodi che li utilizzano, così da assicurare che la connessione venga stabilita correttamente. La seconda permette, invece, di utilizzare il Proxy AudioPlayer per riprodurre un file audio (contenuto nel NAO).

Algoritmo 3 Inizializzazione e utilizzo di un Proxy

```
1: Dichiarazione di un puntatore condiviso
2: audioPlayerProxy ← null
3: function InitializeProxiesOnce(nao_ip)
4:     if audioPlayerProxy is null then
5:         audioPlayerProxy ← new AL::ALAudioPlayerProxy(nao_ip)
6: function PlayAudioFile(nao_path)
7:     if audioPlayerProxy exists then
8:         audioPlayerProxy → playFile(nao_path)
```

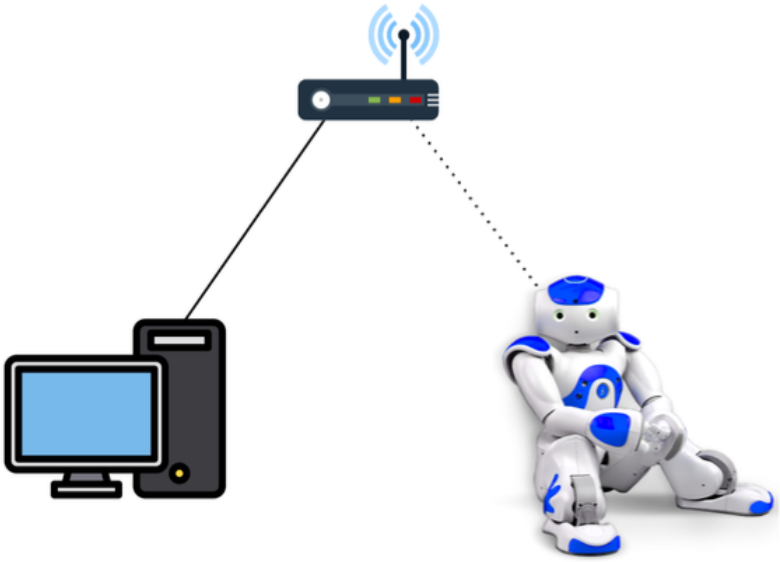


Figura 3.1: Immagine da: Tesi Di Mambro[19]

3.3 Un breve focus su ASP

In questo passaggio si riprende, per un momento, il capitolo precedente, quando è stato descritto come l'uso dell'xAI gestisca il flusso del programma. Al riguardo, è necessario capire meglio come questo avvenga praticamente. Ci si affida, quindi, al lavoro svolto in precedenza,[12] confidando negli script in Python che permettono di:

- chiedere, all'inizio della sessione, il nome dell'utente e memorizzarlo per tutta la durata
- gestire il flusso, scegliendo l'azione successiva secondo la risposta dell'utente
- scegliere il task ulteriore basandosi su uno storico, secondo la difficoltà e gli obiettivi da raggiungere.

Oltre agli script di gestione, ASP utilizza dei programmi chiamati *Knowledge Bases (KB)*, composti esclusivamente da **fatti**, cioè informazioni che conosciamo a priori.

Si ha:

- un KB con i tasks da proporre
- un KB con i goals, gli obiettivi da raggiungere
- un KB per capire come muoversi nel flusso
- un KB che contiene l'history, uno storico dell'esecuzione.

La scelta dell'azione conseguente da intraprendere si baserà sull'unione di questi elementi, evitando di proporre, ad esempio, task già trattati in precedenza e proponendone di nuovi non ancora svolti.

3.3.1 Il flusso del programma

L'obiettivo è quindi creare una conversazione tra robot e utente, ma come fa NAO a sapere cosa dire?

È qui che entra in gioco ASP, fattore tecnico già descritto e che per questo è pleonastico dover ancora esporre in dettaglio.

La struttura generale del flusso è questa:

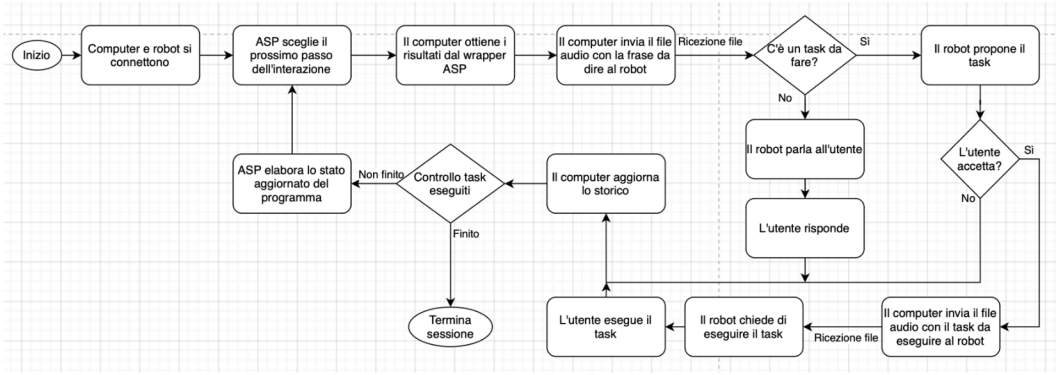


Figura 3.2: Immagine da: Tesi Corni[12]

È importante sapere che una macchina **NON** sa cosa dire e che scelte prendere. È necessario, quindi, implementare manualmente una struttura di dialogo, programmando il codice in modo da descrivere l'azione da eseguire.

3.4 Il programma principale: `main.py`

I pezzi descritti in questo capitolo compongono il programma principale, denominato `main.py`, che verrà descritto in questa sezione.

Il ciclo principale viene ripetuto fintanto che la sessione non arriva ad una conclusione, ed è formato da blocchi di `try-catch` per fare in modo di poter svolgere ogni operazione solo se prevista dallo script ASP.

Di seguito l'esempio di un blocco `try-catch` che verifica la sussistenza di ascolto dell'utente, sottolineando che il codice, di seguito evidenziato, non implementa la struttura spiegata nel capitolo 2, ma quella precedente: [12]

Algoritmo 4 Gestione ascolto utente in `main`

```
1: try:
2:     listen_user ← result['listen_user']
3: if listen_user = "yes" then
4:     if debug = 0 then
5:         response ← input("Risposta:").lower()
6:     else if debug = 1 then
7:         response ← s2t.record_with_mic(language).lower()
8:     else if debug = 2 then
9:         response ← s2t.record(language, soglia).lower()
10:    start_time ← time.time()
11:    if response = "error" then
12:        edge ← "unknown"
13:    else if next_state['statÈ] è in {"task", "ask_repeat"} then
14:        r ← cmp.compare_phrases(answer, response, language)
15:        edge ← ""
16:        if r < 0 then
17:            edge ← "stop"
18:            new_history ← {}
19:        else
20:            if r = 0 then
21:                edge ← "ok"
22:            else if r = 1 then
23:                edge ← "almost"
24:            else
25:                edge ← "no"
```

Il codice in esame ha il compito di gestire la risposta dell'utente nel flusso del dialogo, determinando il successivo stato in base alla variabile `listen_user`, che rappresenta la chiave in ASP. Ciò è essenziale per comprendere al meglio se il flusso prevede che l'utente parli o no, e qualora la chiave non sia presente, viene sollevata un'eccezione.

Se il valore di result è `yes`, allora si procede con l'acquisizione della voce con 3 modalità, in base al valore di debug: [2]

- debug = 0, il sistema comunica tramite input e output testuale
- debug = 1, il sistema registra tramite il microfono del proprio computer, senza utilizzare il NAO, e risponde tramite gli altoparlanti del computer

[2]debug, in tutto il codice, è una variabile che si riferisce alla modalità con cui viene riprodotto il flusso ed è possibile modificarla in fase di sviluppo

- debug = 2, si utilizzano i microfoni di NAO per registrare la voce e gli altoparlanti per rispondere; al riguardo, vedremo più avanti dettagli in merito.

Se la risposta ottenuta è, invece, "error", il sistema imposta `edge`, su `unknown`. Al contrario, se lo Speech To Text produce un risultato, se lo stato successivo è un Task o un "Ask_repeat", si confronta la risposta con quella attesa, utilizzando le modalità della funzione "compare" viste in precedenza:

- se l'esito è 0 la risposta è corretta e viene impostato lo stato "ok"
- se l'esito è 1 la risposta è quasi corretta e si imposta lo stato su "almost"
- se l'esito è maggiore di 1 la risposta è sbagliata e viene assegnato lo stato "no".

Il sistema rileva inoltre altre possibilità, non riportate nell'esempio: che la risposta possa essere sì o no, che l'utente indichi il nome di un task o che contenga un nome.

Nel caso in cui il valore di `listen_user` non sia "yes", il sistema non deve acquisire nessuna risposta dall'utente; in questo caso opta per lo stato successivo in base ai valori già disponibili.

Il resto del codice è molto simile, si determina il passaggio allo stato consecutivo in base alle informazioni accumulate in precedenza, basandosi sui dati della `history`, finché non si arriva alla fine della sessione.

Capitolo 4

Integrazione e Ottimizzazione del sistema

In questo capitolo verrà descritto come le singole componenti del progetto sono state integrate nel lavoro finale, per produrre un sistema stabile e funzionale. Verrà discusso, durante l'analisi degli script, di come siano state, nel frattempo, ottimizzate determinate porzioni con lo scopo di velocizzare l'intero flusso del programma.

Verranno analizzati, passo dopo passo, tutti gli script introdotti negli ultimi mesi di sviluppo, soffermandosi sui vari passaggi e le scelte implementative.

Il focus principale del lavoro è stato **integrare e ottimizzare le varie parti che compongono il progetto** (non considerando ASP), **e ottenere un sistema che non solo fosse funzionante e stabile, ma anche altamente fruibile**. Per questo motivo è stata rivolta una particolare attenzione all'usabilità, rendendo il sistema fluido e scorrevole, ma anche semplice e chiaro.

4.1 Il flusso della Demo

Per facilitare il lavoro sul sistema in maniera semplice e chiara, è stato sviluppato un programma indipendente dal Ragionatore, concepito come una "demo" del progetto finale. La scelta è ricaduta sulla creazione di una demo poiché la struttura dei task e del DFSA, nonostante sia stata modellizzata, non è stata ancora integrata nel ciclo del lavoro. Di conseguenza, per poter utilizzare i nuovi script implementati, assieme al riconoscimento delle emozioni, era necessario realizzare una struttura di prova sulla quale testarli.

Dopo aver effettuato gli `import` dei moduli utilizzati, che verranno descritti meglio dopo, vengono inizializzate le variabili utili alle funzioni, quali `nao_ip`, che rappresenta l'ip del robot NAO, `nao_path`, il percorso in cui salvare i file

audio del T2S, `nao_username` e `nao_password`, per stabilire una connessione SSH tra NAO e macchina locale, `language`, il linguaggio da utilizzare nello S2T e nel T2S e `negative_emotion`, una lista per memorizzare le emozioni negative. C'è poi una variabile speciale, `emotion_recognition`, che è possibile utilizzare per scopi di debug, utile a disattivare o attivare il riconoscimento di emozioni, tramite 0 e 1 rispettivamente.

La funzione che si occupa di definire il ciclo del programma è `reason()`, ed è divisa essenzialmente in due parti: durante la prima vengono proposte domande, molto banali, utili non tanto a testare specifiche abilità del soggetto, quanto più a dare una dimostrazione di come avviene un dialogo, nella sua interezza.

Figura 4.1: Dialogo con NAO: Demo

4.1.1 Prima sezione di `test_nao_main`

Si inizia chiedendo il nome dell'utente tramite la funzione `start_threads()`, che vedremo meglio nella sezione inerente al T2S, e aprendo una finestra di ascolto tramite `record_with_nao()`. Dopodiché vengono definite delle domande, con le risposte corrette associate, e proposte in ordine di scrittura all'utente. La risposta viene elaborata, assieme all'emozione rilevata (se il riconoscimento è attivo) e confrontata con la risposta corretta.

Algoritmo 5 Logica del ragionatore con gestione delle risposte

```
1: function REASON
2:     start_threads("Ciao. Come ti chiami?")
3:     (name, _) ← record_with_nao("Come ti chiami?")
4:     start_threads("Ciao " + name + ". è un piacere conoscerti! Voglio
   farti alcune domande per allenare la tua memoria.")
5:     questions ← [
6:         {"question": "Qual è la capitale dell'Italia?", "correct_answer":
   "roma"},
7:         {"question": "Che colore ha il cielo?", "correct_answer": "blu"},
8:         {"question": "Quante dita ha una mano?", "correct_answer": "5"},
9:         {"question": "Quanto fa 2 + 2?", "correct_answer": "4"},
10:        {"question": "Cosa viene dopo il 6?", "correct_answer": "7"},
11:        {"question": "Cane o leone, chi è più forte?", "correct_answer":
   "leone"}
12:        ...
13:    ]
14:    for all item in questions do
15:            start_threads(item["question"])
16:        if emotion_recognition == 1 then
17:                (answer, emotion) ← record_with_nao(item["question"])
18:        else
19:                answer ← record_with_nao(item["question"])
20:            r ← cmp.compare_phrases(answer, item["correct_answer"],
   language)
21:    ...
```

Da qui si aprono diversi scenari. Durante la funzione di ascolto, in alcuni casi, quando viene fatta una domanda, vengono analizzate anche le espressioni facciali dell'utente, per capire il suo stato d'animo in quel momento ed agire di conseguenza. Ad esempio, se appare felice durante la risposta, il NAO glielo farà notare, rafforzando il feedback positivo. Al contrario, se mostra

un'emozione negativa, gli verrà chiesto se desidera interrompere la sessione. Questo comportamento è completamente modificabile in base a come verrà scelto di gestire le emozioni in futuro.

4.1.2 Seconda sezione di `test_nao_main`

Una volta completate le domande di base, si entra nella seconda parte della Demo, in cui viene proposto il compito "Batti un colpo", discusso precedentemente. Per realizzare il task descritto è stato preso, direttamente dal database del compito, un esempio dello stesso. Prima di addentrarci nella seconda parte, però, verrà chiesto all'utente se è pronto per iniziare e, in caso di risposta negativa, gli verrà raccontata una barzelletta per metterlo a suo agio.

```
...
start_threads("Ora facciamo qualcosa di diverso. Ti racconterò una storia,
ogni volta che sentirai la parola BAMBINI, dovrai battere le mani...
clap ← 0
start_threads("Mario e Luca sono amici da quando erano bambini.")
result ← nao_interface.audio_recorder_clap(nao_ip, 1000, 5)
if result then
    clap ← clap +1
...
...
final_question ← {"question": "Dove si sono reincontrati...", "cor-
rect_answer": "bar"}
start_threads(final_question["question"])
(answer, _) ← record_with_nao(final_question["question"])
r ← cmp.compare_phrases(answer, final_question["correct_answer"],
language)
clap ← str(clap)
if r ∈ [0, 1] then
    start_threads("Ottimo! Hai risposto correttamente alla domanda e hai
battuto le mani " + clap + " volte su 3! Va benissimo così!")
else
    start_threads("Mi dispiace, la risposta corretta era " + fi-
nal_question["correct_answer"] + ".")
    start_threads("Hai battuto le mani " + clap + " volte su 3, va
benissimo così!")
...
```

Durante il compito verrà, quindi, raccontata una storia contenente una parola target. Ogni volta che questa viene pronunciata da NAO, l'utente dovrà

battere le mani. Alla fine della storia, verrà anche posta una domanda inerente al contenuto, dopodiché verrà fornito un feedback.

4.1.3 Rilevazione del battito di mani

Le funzioni utilizzate sono le stesse viste nella prima parte della demo, con l'aggiunta di un'altra che è stata appositamente creata per questo task: `audio_recorder_clap()`.

Analizziamola brevemente:

Algoritmo 6 Rilevamento del battito di mani tramite microfono NAO

```
1: function AUDIO_RECORDER_CLAP(nao_ip, soglia_clap, max_rec_time)  ▷ Restituisce un valore booleano
2:     InitializeProxiesOnce(nao_ip)
3:     EnableEnergyComputation()
4:     front_energy ← 0
5:     rec_time ← 0
6:     while rec_time < max_rec_time do
7:         print(rec_time)
8:         front_energy ← GetFrontEnergy()
9:         print("Front:", front_energy)
10:        if front_energy ≥ soglia_clap then
11:            return True
12:        Wait(100 ms)
13:        rec_time ← rec_time + 0.1
14:    DisableEnergyComputation()
15:    return False
```

La seguente funzione si trova nella libreria C++, utile ad interfacciarsi con i moduli di NAO. I parametri sono: `nao_ip`, per inizializzare il Proxy, `soglia_clap`, che rappresenta la soglia di energia oltre la quale viene rilevato il battito delle mani e `max_rec_time`, il tempo massimo di registrazione.

È doveroso soffermarsi sull'energia, un parametro cruciale quando si lavora con i microfoni di NAO. Tramite questo valore, recuperabile tramite la funzione `enable_energy_computation`, è possibile monitorare in tempo reale la presenza e l'intensità dei suoni rilevati. Grazie a queste informazioni è possibile intervenire direttamente sul flusso della registrazione o, come in questo caso, evitare di avviarla del tutto quando non è necessario trascrivere l'audio. È importante sottolineare che il valore della soglia deve essere definito manualmente dal programmatore, o da chi di competenza, prima dell'inizio della sessione. Questa scelta permette di adattare il riconoscimento del suono alle specifiche esigenze, garantendo un livello di sensibilità adeguato.

La funzione restituisce infine "True", se è stato rilevato un suono superiore alla soglia, "False" altrimenti; prima di questo, viene disattivato il rilevamento dell'energia dai microfoni.

4.1.4 La disconnessione sicura

Si anticipa un punto fondamentale del lavoro: il sistema opera in un ambiente MultiThread. Ciò significa che, per garantire un corretto funzionamento e migliorare la velocità, il flusso di lavoro è stato suddiviso in diverse parti, così da parallelizzare il maggior numero possibile di processi. Il MultiThreading consente di eseguire azioni che, altrimenti, richiederebbero più tempo, ma implica anche la necessità di una gestione rigorosa dei vari thread.

Per garantire, quindi, la corretta chiusura del programma, con i suoi sottoprocessi, è stato creato un meccanismo che consente, premendo 'e' durante l'esecuzione, di poter aspettare che tutto ciò che è in esecuzione venga correttamente terminato.

Per implementarlo è stato utilizzato il modulo `keyboard` della libreria `pynput`, che permette di avviare un thread che monitora la pressione di un tasto durante l'esecuzione ed eseguire l'azione desiderata una volta premuto.

Algoritmo 7 Press 'e' to terminate the program

```
1: global terminate
2: function press_e
3:     global terminate
4:     function on_press(key)
5:         if key = keyboard.KeyCode.from_char('e') then
6:             terminate = 1
7:     with keyboard.Listener(on_press=on_press) as listener
8:     listener.join()
9: thread_terminate = threading.Thread(target=press_e,
   daemon=True)
10: thread_terminate.start()
```

Nel caso corrente, alla pressione del tasto 'e' sulla tastiera, viene impostata a 1 una variabile di nome `terminate`. Durante l'esecuzione del codice, in particolare all'inizio di ogni funzione che prevede l'utilizzo dei moduli di NAO, viene verificato il valore di essa. Se è impostata a uno, viene invocata la funzione `clean_exit()`, che termina tutti i processi in esecuzione, con particolare attenzione al riconoscitore di emozioni che, oltre ad essere in esecuzione su un thread separato, presenta un ulteriore thread al suo interno che deve necessariamente essere terminato.

A proposito di questo, è importante notare come la funzione `clean_exit()` richiami un'ulteriore funzione al suo interno: `stop_emotion_recognition()`, che serve appunto a garantire che tutto ciò che è all'interno del riconoscitore venga correttamente terminato, assicurandosi anche di eseguire la disconnessione tramite `naoqi_disconnect()`, un metodo che deve essere necessariamente richiamato per poter disconnettere il programma dal modulo della videocamera del NAO e per liberare correttamente le risorse.

Algoritmo 8 Terminazione pulita del programma: `clean_exit`

```
1: function CLEAN_EXIT
2:     print("Terminazione in corso...")
3:     start_threads("A presto!")
4:     if emotion_recognition = 1 then
5:         stop_emotion_recognition()
6:     naoqi_getimages_disconnect()
7:     ssh_close()
8:     print("Programma chiuso!")
9:     exit(0)
```

Mentre la funzione `stop_emotion_recognition()` si trova nella classe dedicata al riconoscimento delle emozioni ed è strutturata in questo modo:

Algoritmo 9 Stop Emotion Recognition

```
1: function STOP_EMOTION_RECOGNITION(self)
2:     if self.head_tracking = 1 then
3:         self.ended = 1
4:         self.t1.join()
5:     print(sum(self.fps) / len(self.fps))
6:     frame_time ← sum(self.fps) / len(self.fps)
7:     print("FPS: " + str(1 / frame_time))
8:     interface_for_emotion_recognition.naoqi_getimages_disconnect()
9:     print("Programma emozioni terminato!")
```

Se, quindi, il thread che si occupa del tracciamento della testa è attivo, la variabile `ended` viene impostata a 1. Questa ha il compito di interrompere l'esecuzione del ciclo principale.

4.2 L'interfacciamento al NAO: `nao_interface`

In questa breve sezione si descrive un'interfaccia che è stata sviluppata, più per comodità che per esigenza, per poter richiamare le funzioni della libreria in C++ e alcune funzioni generiche, utili al programma.

L'interfaccia in questione è denominata `nao_interface` e utilizza `ctypes` e un collegamento alla libreria in C++ per poter recuperare le funzioni.

Vedremo solo un esempio di come utilizzare una funzione della libreria nell'interfaccia, poiché esse verranno approfondite nelle sezioni a loro dedicate.

Algoritmo 10 Esempio di una funzione in nao_interface

```
1: function AUDIO_RECORDER_FRONT(nao_ip, nao_path, soglia,
   max_silence_time, max_rec_time_with_silence, max_rec_time)
2:     b_nao_ip ← nao_ip.encode('utf-8')
3:     b_nao_path ← nao_path.encode('utf-8')
4:     naoqi_getimages.audio_recorder_front.argtypes ←
5:         [c_char_p, c_char_p, c_float, c_float, c_float, c_float]
6:     naoqi_getimages.audio_recorder_front(
7:         create_string_buffer(b_nao_ip),
8:         create_string_buffer(b_nao_path),
9:         soglia, max_silence_time,
10:        max_rec_time_with_silence, max_rec_time
11:    )
```

I parametri della funzione verranno esposti successivamente. Per ora ci si limiterà a descrivere brevemente il processo di conversione.

`naoqi_getimages` è la libreria, scritta in C++, da cui andremo a richiamare le funzioni.

I parametri `nao_ip` e `nao_path`, che sono stringhe in Python, vengono convertiti in byte string utilizzando il metodo `encode`. Questo rappresenta un passaggio necessario, poiché la funzione corrispondente in C++ richiede una rappresentazione di tipo `char*`, che in Python è gestita da `c_char` di `c_types`.

Il prossimo passaggio è specificare gli argomenti della funzione da richiamare, utilizzando i rispettivi tipi in Python con `c_types.` Lo stesso vale per il tipo di ritorno.

L'ultimo step è richiamare la funzione dalla libreria in C++, specificando gli argomenti, con la premura di aggiungere, per i `char*`, dei buffer che contengono i byte string precedentemente creati. Questo verrà fatto tramite l'uso di `create_string_buffer()`.

Quest'interfaccia facilita il lavoro e permette di non intasare il codice dei processi principali, mantenendo separata la logica di interazione con la libreria in C++.

4.3 Lo scambio di dati: SSH_Manager

Prima di procedere con la spiegazione del nuovo flusso del dialogo, è doveroso introdurre un cambiamento sostanziale nella nuova logica implementata per lo scambio di dati tra macchina locale e NAO. In precedenza, una volta elaborato il file tramite T2S per la riproduzione da NAO, veniva utilizzato un socket per l'invio di esso, che forzava l'apertura costante del software Choreographe, limitando l'indipendenza del sistema. L'obiettivo era, quindi, trovare qualcosa in grado di sostituire il socket, ma che fosse al contempo veloce e affidabile. Vista la possibilità di collegamento a NAO tramite SSH, la scelta è ricaduta su quello. L'invio dei dati avviene tramite il protocollo SFTP, sfruttando la libreria `paramiko`, sia per il trasferimento che per la gestione dei file in remoto. Il meccanismo scelto garantisce velocità di esecuzione e stabilità, evitando possibili blocchi dovuti al software Choreographe aperto in background. Ciò che verrà spiegato di seguito sarà utilizzato sia durante la fase di Text To Speech che durante la fase di Speech To Text. Le modalità con cui questo avverrà verranno spiegate nelle sezioni a loro dedicate. La classe che si occupa della gestione della sessione SSH è stata denominata `ssh_manager` e si occupa di fornire un'interfaccia stabile e affidabile per la connessione.

Il Singleton

Uno dei punti forti della classe è l'implementazione del pattern *Singleton*, che assicura che venga creata una sola istanza dell'oggetto per tutta l'esecuzione del programma, evitando, quindi, che vengano aperte più connessioni SSH contemporaneamente. Ciò garantisce una gestione centralizzata della connessione con il NAO.

Algoritmo 11 Implementazione Singleton per la gestione SSH

```
1: Variabile di classe: instance ← None
2: function NewInstance(hostname, username, password, retry_delay=5)
3:     if instance is None then
4:         instance ← Nuova istanza di SSHManager
5:         instance.hostname ← hostname
6:         instance.username ← username
7:         instance.password ← password
8:         instance.ssh ← Nuovo client SSH
9:         instance.sftp ← None
10:        instance.retry_delay ← retry_delay
11:        ssh_init(instance)
12:    return instance
```

All'inizio viene dichiarata una variabile `instance`, inizializzata a `None`, che funge da riferimento all'**unica** istanza della classe. Ogni volta che si tenta di creare un nuovo oggetto SSH_Manager, verrà richiamato il metodo `__new__`. Se l'istanza è ancora `None`, significa che l'oggetto non è mai stato creato in precedenza, quindi viene inizializzato con i relativi parametri, tra cui `retry_delay`, impostato a 5 secondi di default e che rappresenta il tempo di attesa per un'eventuale ri-connessione automatica. L'ultimo step è richiamare il metodo `ssh_init` per la connessione effettiva al robot, tramite il protocollo SSH.

4.3.1 La gestione della sessione SSH

Verranno ora descritte le 3 funzioni che si occupano di inizializzare, mantenere e chiudere la sessione SSH, con i suoi meccanismi di ri-connessione automatica in caso di errori.

Inizializzazione: `ssh_init`

L'inizializzazione di una sessione è molto delicata, in quanto, senza una connessione attiva, l'intero programma non avrà possibilità di funzionare correttamente, poiché non sarà in grado di scambiare materiale con il NAO. Per questo motivo, è stato implementato un ciclo che non verrà completato sin quando non si avrà la sicurezza che sia stata stabilita correttamente una connessione. Non mancano, ovviamente, dei blocchi `try-catch` per la gestione delle eccezioni.

Algoritmo 12 Inizializzazione della connessione SSH e SFTP

```
1: function SSH_INIT
2:     while True do
3:         try
4:             Imposta la policy per accettare chiavi host sconosciute
5:             Connetti al server SSH con hostname, username, password
6:             Apri connessione SFTP
7:             print "Connessioni SSH e SFTP aperte"
8:             break
9:         catch Errore e
10:            print "Errore nella connessione con NAO: e"
11:            print "Riprovo la connessione in retry_delay secondi..."
12:            wait retry_delay secondi
```

La logica `retry-delay` permette, inoltre, di non sovraccaricare il sistema effettuando continuamente nuovi tentativi di ri-connessione.

Controllo: `ssh_check`

Non verranno spese molte parole su questo metodo, è importante però sapere come sia stato implementato anche un meccanismo di controllo della sessione SSH, durante tutto il programma e ogni qualvolta si debba eseguire un'operazione che la coinvolga. La funzione si occupa esclusivamente di verificare lo stato della connessione, provvedendo a effettuare una ri-connessione nel caso in cui il riscontro sia negativo.

Algoritmo 13 Controllo e riconnessione SSH

```
1: function SSH_CHECK
2:     if La connessione SSH è attiva then
3:         Return True
4:     else
5:         print "Connessione SSH non attiva, provo a riconnettermi..."
6:         Chiama ssh_init() per tentare una nuova connessione
7:         Return True
```

La chiusura: `ssh_close`

Per una disconnessione sicura, è fondamentale garantire che tutte le risorse siano liberate correttamente e che la connessione venga chiusa in modo controllato. La funzione implementata gestisce questo processo, eseguendo

prima un controllo per assicurarsi che ci sia una connessione da chiudere. Successivamente procede alla chiusura della stessa.

Algoritmo 14 Chiusura della connessione SSH e SFTP

```
1: function SSH_CLOSE()
2:     if ssh_check() è vera then
3:         if sftp è attiva then
4:             Chiudi la connessione SFTP
5:         Chiudi la connessione SSH
6:         print "Connessioni SSH e SFTP chiuse"
7:     else
8:         print "Nessuna connessione da chiudere"
```

4.3.2 Invio, ricezione ed eliminazione

Si espongono ora i punti principali del Manager, fondamentali per il funzionamento del sistema. La libreria `paramiko` offre dei metodi avanzati per i trasferimenti sicuri. In particolare, si utilizzerà:

- Il metodo `post`, che consente l'invio dei file, da computer locale a NAO, durante la fase di Text To Speech.

- Il metodo `get`, che gestisce la ricezione dei file, da NAO a macchina locale, durante la fase di Speech To Text.

La fase di invio: `put_file_on_nao`

La funzione che si occupa dell'invio del file tramite SSH è chiamata `put_file_on_nao`. Essa utilizza il metodo `sftp.put(local_path, nao_path)`, prendendo come parametri il percorso locale del file, sulla macchina, e il percorso dove posizionare il file sul NAO per la riproduzione.

Algoritmo 15 Caricamento di un file su NAO: `put_file_on_nao`

```
1: function PUT_FILE_ON_NAO(local_path, nao_path)
2:     max_attempts ← 3
3:     attempts ← 0
4:     while attempts < max_attempts do
5:         Try
6:             If ssh_check() then
7:                 sftp.put(local_path, nao_path)
8:                 print "File local_path caricato con successo su nao_path"
9:                 Break
10:        Catch FileNotFoundError
11:            print "File locale local_path non trovato"
12:            Break
13:        Catch Exception e
14:            print "Errore nel caricare local_path: e. Provo a
    riconnettermi..."
15:            ssh_init()
16:            attempts ← attempts + 1
17:    if attempts = max_attempts then
18:        print "Impossibile inserire il file dopo max_attempts tentativi"
19:        Genera errore "Errore nell'inserire local_path"
```

Oltre al meccanismo di invio del file, è stato impostato anche un numero massimo di tentativi oltre il quale l'errore viene considerato definitivo, in modo da non sovraccaricare il sistema. Il blocco `try-catch` è utilizzato anche per quando non viene trovato il file specificato sul sistema.

Ricezione di un file: `get_file_from_nao`

Per la ricezione di un file, da NAO verso la macchina locale, il meccanismo implementato è lo stesso, con la differenza che viene usato il metodo `sft.get()` anziché `put()`. Esso è utilizzato durante la fase di Speech To Text, dopo la registrazione e la creazione di un file audio.

Algoritmo 16 Ricezione di un file da NAO: `get_file_from_nao`

```
1: function GET_FILE_FROM_NAO(nao_path, audio_path)
2:     start_time ← tempo attuale
3:     max_attempts ← 3
4:     attempts ← 0
5:     while attempts < max_attempts do
6:         Try
7:           If ssh_check() then
8:             sftp.get(nao_path, audio_path)
9:             print "Prelevato nao_path con successo"
10:            Break
11:        Catch FileNotFoundError
12:          print "File nao_path non trovato su NAO"
13:          Break
14:        Catch Exception e
15:          print "Errore nel prelevare nao_path: e. Provo a
    riconnettermi..."
16:          ssh_init()
17:          attempts ← attempts + 1
18:    if attempts = max_attempts then
19:        print "Impossibile prelevare il file dopo max_attempts tentativi"
20:        Genera errore "Errore nel prelevare nao_path"
21:    nao_interface.measure_time("File prelevato in: ", start_time)
```

Rimozione del file da NAO: `remove_file_from_nao`

Nel programma è stata inserita anche una funzione per rimuovere i file non più utili dal NAO, con implementazione simile alle due precedenti. Essa è utilizzata nella classe `Thread_Manager`, durante la fase di Text To Speech.

Algoritmo 17 Rimozione di un file da NAO: `remove_file_from_nao`

```
1: function REMOVE_FILE_FROM_NAO(nao_path)
2:     start_time ← tempo attuale
3:     max_attempts ← 3
4:     attempts ← 0
5:     while attempts < max_attempts do
6:         Try
7:           If ssh_check() then
8:             sftp.remove(nao_path)
9:             print "file nao_path rimosso con successo da NAO"
10:            Break
11:        Catch FileNotFoundError
12:          print "File nao_path non trovato su NAO"
13:          Break
14:        Catch Exception e
15:          print "Errore nella rimozione del file nao_path: e. Provo a
    riconnettermi..."
16:          ssh_init()
17:          attempts ← attempts + 1
18:    if attempts = max_attempts then
19:        print "Impossibile rimuovere il file dopo max_attempts tentativi"
20:        Genera errore "Errore nel rimuovere nao_path"
21:    nao_interface.measure_time("File rimosso da NAO in: ",
    start_time)
```

4.4 Il Text To Speech

Il Text To Speech è un punto delicato al quale lavorare, perché ne consegue non solo la velocità del sistema, ma anche il modo in cui il NAO comunica (il tono della voce, la velocità...). È stato descritto in precedenza come è stato implementato il processo di sintesi vocale, dalla trasformazione del testo in un file audio all'invio dello stesso. Prima di approfondire il nuovo flusso, è utile ribadire gli strumenti utilizzati per la conversione da testo ad audio e formulare alcune considerazioni.

L'obiettivo del lavoro non era solo quello di implementare un meccanismo stabile e veloce, ma anche quello di creare un programma funzionale, poiché un punto importante che è stato considerato è proprio l'usabilità.

I moduli utilizzati in precedenza erano 2: Google, online, e pyttsx4, offline, implementati in modo che uno fosse l'alternativa dell'altro in caso di fallimento. I due moduli fanno sì che NAO utilizzi due voci completamente diverse, a seconda del modulo in uso.

Tuttavia, con l'introduzione del nuovo flusso, si è deciso di eliminare l'uso di pyttsx4, pur lasciando la funzione nella classe. Questo cambiamento è stato motivato dalla necessità di evitare il passaggio brusco tra due voci differenti; infatti, il cambio radicale di voce risulterebbe innaturale per l'utente, interrompendo l'esperienza di interazione con NAO. Di conseguenza, è stato scelto, sacrificando il Text To Speech offline, di utilizzare solo Google, per migliorare la coerenza e l'efficacia del sistema. Nulla vieta, in futuro, di implementare un sistema di backup.

Il codice che si occupa dell'invio del testo a Google, per la trascrizione, è rimasto molto simile al precedente, con l'unica aggiunta di un controllo sulla lunghezza del file audio, poiché, durante lo sviluppo, è stato trovato un bug con l'utilizzo di `AudioSegment`, di `pydub`, in particolare con la funzione `from_file`. Il bug in questione causa il mancato invio del file audio, causando un crash del programma, o mandandolo in un loop, con la pressione dei tasti 'q' ed 'e'.

Per quanto riguarda, invece, il flusso di invio del file a NAO e la successiva riproduzione, è stato implementato un meccanismo di parallelizzazione utilizzando i thread, ottenendo un miglioramento del sistema e, conseguentemente, dell'efficienza e della velocità.

4.4.1 La classe Thread_Manager

L'uso dei thread è stato fondamentale per parallelizzare l'esecuzione delle operazioni coinvolte, evitando colli di bottiglia e attese indefinite. In particolare, le operazioni che sono state parallelizzate sono:

- La creazione, da un testo, del file audio, tramite Google
- L'invio del file audio a NAO tramite SSH
- La riproduzione del file audio da NAO

Prima di addentrarsi nell'implementazione, si riassume brevemente il flusso, analizzando di cosa si occupano i thread.

Dopo i consueti import delle librerie utilizzate, si utilizza, come negli altri file descritti, un Singleton che ci assicura che venga creata una sola istanza dell'oggetto. Dopodiché, c'è una funzione che gestisce l'inizializzazione dei

parametri, quali `nao_path`, il percorso in cui verranno salvati i file sul NAO, `nao_ip`, l'ip del robot per l'inizializzazione del Proxy, il campo `language`, per impostare la lingua del Text To Speech. Oltre ai parametri della funzione, vengono inizializzati anche ulteriori campi: le due code per i Worker, la lista per contenere i Thread e l'`SSH_Manager` per l'invio dei file.

Algoritmo 18 ThreadManager: inizializzazione

```
1: class ThreadManager
2:     _instance ← None
3:     _lock ← threading.Lock()
4: def __new__(cls, nao_path, nao_ip, language):
5: if cls._instance is None then
6:         with cls._lock:
7:             cls._instance ← super(ThreadManager, cls).__new__()
8:             cls._instance._initialize(nao_path, nao_ip, language)
9:     return cls._instance
10: def _initialize(self, nao_path, nao_ip, language):
11:     self.nao_path ← nao_path
12:     self.nao_ip ← nao_ip
13:     self.language ← language
14:     self.simple_queue_requests ← queue.Queue()
15:     self.simple_queue_files ← queue.Queue()
16:     self.threads ← []
17:     self.ssh_manager ← SSHManager(nao_ip, "nao", "nao")     ▷ Evita la
    variabile globale
```

Si hanno, innanzitutto, 3 Thread chiamati "worker" nella classe, che utilizzano una `simple_queue` per scambiarsi i dati e sincronizzarsi tra loro.

1. Il primo Worker ha il compito di generare un file audio a partire da una frase di input, con in più, un controllo per verificare se in input si riceve già un file audio pronto per la riproduzione. Se l'input è una frase testuale, questa viene suddivisa in frasi più piccole, secondo un criterio che vedremo dopo, utilizzando una funzione di `nao_interface` chiamata `sentence_splitter()`, e per ciascuna viene generato un file audio, inserito nella coda.

2. Il secondo Worker si occupa di prelevare i file audio dalla coda e trasferirli sul robot NAO tramite SSH. Quando questo è stato completato, il file, con il suo percorso completo, viene messo in un'altra coda e inviato al terzo Worker. Quando dalla coda di Worker2 viene prelevato `None`, vuol dire che non ci sono più file da trasferire.

3. Il terzo Worker ha il compito di riprodurre i file audio che sono stati trasferiti sul NAO, in ordine di immissione, utilizzando la funzione di `nao_interface`, `audio_play_file()`. L'audio già riprodotto viene poi eliminato da NAO tramite la funzione precedentemente descritta, `remove_file_from_nao()`. Questo è possibile poiché la funzione di riproduzione del file audio è bloccante, quindi, finché il file audio non viene del tutto riprodotto, il Thread non continua la sua esecuzione.

4. La funzione `start_threads`, richiamata nel ciclo principale, si occupa, invece, dell'avvio dei Thread e della loro corretta terminazione.

Vediamo ora l'implementazione dei singoli passaggi.

Worker1

Il Thread Worker1 ha, innanzitutto, 2 parametri essenziali: `prog`, che rappresenta il progressivo per distinguere e ordinare i file audio che vengono inseriti nella queue, chiamata `simple_queue_requests`, e `phrase`, che rappresenta la frase input da trasformare.

Il primo step è verificare che phrase non rappresenti il percorso di un file audio e che quindi non finisca con una delle estensioni per riconoscerlo, questo per poter far riprodurre al NAO non solo delle frasi, ma anche qualunque audio si voglia.

Se il controllo restituisce `False` si procede con la scomposizione della frase, tramite la funzione `sentence_splitter()`, implementata in questo modo:

Algoritmo 19 Funzione per dividere le frasi: `sentence_splitter`

```
1: Function sentence_splitter(input_text):
2:     Inizia il timer (start_time)
3:     Rimuovi spazi all'inizio e alla fine del testo (input_text)
4:     Dividi e ricomponi le frasi con i separatori [".", "?", "!"] (list)
5:     Stampa la lista delle frasi
6: Function split_and_readd_punctuation(text, separators):
7:     Crea una lista iniziale con il testo completo (list)
8:     For each separator (sep) in separators:
9:         Crea una lista temporanea (temp_list)
10:        For each sentence in list:
11:            Dividi la frase usando il separatore seguito da uno spazio (parts)
12:            Aggiungi il separatore a tutte le parti della lista tranne l'ultima
    (parts)
13:            Aggiungi le parti alla lista temporanea (temp_list)
14:        Aggiorna la lista principale con la lista temporanea (list)
15:    Restituisci la lista delle frasi
```

Avviene quindi uno `split` della frase ogni qualvolta si trova un punto, seguito da uno spazio, un punto esclamativo, seguito da uno spazio e un punto interrogativo, seguito da uno spazio. Si esegue, inoltre, una ricomposizione della frase per garantire l'intonazione corretta durante il processo di Text To Speech.

Il metodo prosegue trasformando ogni elemento della lista, che rappresenta ciascun pezzo scomposto della frase, in file audio, assegnando loro un progressivo. Ogni file audio viene quindi inserito nella coda. Una volta completato il processo, viene inserito `None` nella coda per indicare la conclusione.

Algoritmo 20 Worker1: Trasformazione di un testo

```
1: function WORKER1(self, thread_id, prog, phrase)
2:     ''Genera i file audio e li mette in coda per il
   trasferimento.''
3:     start_time ← tempo corrente
4:   if file_esiste(phrase) e phrase finisce con '.mp3', '.wav', o '.ogg' then
5:         Aggiungi phrase alla coda
6:   else
7:         lista ← sentence_splitter(phrase)
8:       for ogni elemento in lista do
9:             filename ← generare_audio(element, lingua, prog)
10:            prog ← prog + 1
11:            Aggiungi filename alla coda
12:            measure_time("Fine worker1: ", start_time)
13:    Aggiungi None alla coda                          ▷ Fine dei file
```

Worker2

La funzione Worker2 è responsabile dell'invio dei file audio al NAO, per la riproduzione. Il ciclo principale inizia prelevando i file dalla coda finché non viene letto `None`. C'è poi un'operazione di concatenazione di percorsi, per costruire il percorso completo del file. Una volta ottenuto il percorso completo, si procede al trasferimento vero e proprio, tramite la funzione `put_file_on_nao()` della classe `SSH_Manager`. Dopodiché si inserisce il file in un'ulteriore coda, chiamata `simple_queue_files`, essenziale per sincronizzare Worker2 con Worker3.

Algoritmo 21 Worker2: Trasferimento file al NAO

```
1: function WORKER2(self, thread_id)  ▷ Trasferisce i file generati al NAO.
2:     Inizia il timer (start_time)
3:     while True do
4:         filename ← self.simple_queue_requests.get()
5:         if filename è None then
6:             self.simple_queue_files.put(None)
7:             break
8:         print "File:", filename
9:         full_path ← unisci il percorso self.nao_path e filename
10:        self.ssh_manager.put_file_on_nao(filename, full_path)
11:        misura il tempo con nao_interface.measure_time("Fine worker2:
    ", start_time)
12:        self.simple_queue_files.put(filename)
```

Worker3

Worker3 ha come unico compito quello di riprodurre i file audio in ordine di creazione. Il ciclo principale inizia prelevando il file dalla coda, viene poi recuperato il percorso completo, dopodiché avviene la riproduzione vera e propria tramite la funzione `audio_play_file()` di `nao_interface`. Alla fine del processo, il file audio viene eliminato da NAO.

Algoritmo 22 Worker3: Riproduzione dei file audio sul NAO

```
1: function WORKER3(self, thread_id)     ▷ Riproduce i file audio sul NAO.
2:     Inizia il timer (start_time)
3:     first_audio_played ← False
4:     while True do
5:         item ← self.simple_queue_files.get()
6:         if item è None then
7:             break                    ▷ Interrompi il ciclo se il valore è None
8:         if first_audio_played è False then
9:             first_audio_played ← True
10:            nao_interface.measure_time("Tempo per riprodurre il primo
    file audio", start_time)
11:        full_path ← unisci il percorso self.nao_path e item
12:        nao_interface.audio_play_file(self.nao_ip, full_path)
13:        self.ssh_manager.remove_file_from_nao(full_path)
```

4.4.2 Considerazioni sul Text To Speech

È importante, oltre che alla comprensione dell'esecuzione del processo, capire anche in che modo avviene la parallelizzazione, analizzando come e quando ogni Thread inizia e finisce.

Si descrive il flusso di lavoro con un esempio:

La frase da scomporre è: **"Ciao. Io sono NAO! Come ti chiami?"**

I thread vengono avviati contemporaneamente tramite la funzione e si creano 3 esecuzioni differenti e parallele.

Si inizia da Worker1, che divide la frase, producendo, secondo i criteri di scomposizione descritti, 3 file audio: `file_1`, `file_2` e `file_3`. Non appena il primo file viene inserito nella coda, viene attivato il secondo thread, Worker2. Quest'ultimo, che fino a quel momento stava monitorando la coda, invia il file a NAO insieme al suo percorso completo. Nel frattempo, Worker3, che è nella stessa situazione di Worker2 (con almeno un elemento nella coda), inizia la riproduzione del file audio. Poiché il lavoro è eseguito in parallelo ma sincronizzato tramite le code, il ciclo di produzione, trasferimento e riproduzione dei file si ripete anche per i 2 file rimanenti.

Questo approccio permette di ottenere una fase di Text To Speech sempre veloce ed efficiente. Senza la parallelizzazione, ad esempio, con una frase molto lunga, si sarebbe costretti ad attendere il completamento di ogni fase, o peggio ancora, senza la scomposizione della frase, si dovrebbe aspettare che l'intera frase venga elaborata prima di passare alla fase successiva. In questo modo, si perderebbe la fluidità e la scioltezza necessaria per un dialogo naturale.

4.5 Lo Speech To Text

Si analizza ora su un altro punto fondamentale e fortemente collegato al precedente: lo Speech To Text, che rappresenta il modo in cui NAO riesce a pronunciare le frasi all'utente.

Come anticipato nel capitolo precedente, il processo di Speech To Text era dipendente da un microfono esterno e da Choreographe, che rappresentava il mezzo tramite il quale aprire un canale con cui inviare il file audio scomposto.

La prima decisione presa riguarda la separazione delle responsabilità. Prima di tutto è stata fatta un'analisi dettagliata delle limitazioni di NAO, testando uno ad uno i microfoni che il robot mette a disposizione.

L'obiettivo era eliminare del tutto la dipendenza dal microfono esterno e dal software Choreographe, valutando in primis la possibilità di registrare tramite i microfoni di NAO e, in secondo luogo, utilizzando un altro metodo di invio del materiale. Per il secondo punto, è stato già discusso della rimozione del socket a favore dello scambio di dati tramite SSH. Per il primo, sono stati condotti alcuni test registrando file audio con la stessa frase, prima utilizzando tutti e quattro i canali contemporaneamente, poi singolarmente con ciascun microfono. I file sono stati dapprima ascoltati per valutare il rumore di fondo e la presenza di disturbi, quindi trascritti tramite il sistema di Text To Speech per verificare la corrispondenza tra la frase trascritta e quella effettivamente pronunciata. Si è quindi arrivati alla conclusione di rimuovere il microfono esterno, poiché, dai test effettuati, è emerso che il NAO fosse capace di registrare audio di qualità decente, in modo da essere comprensibile per lo Speech To Text. In particolare, la registrazione migliore proviene dal microfono frontale di NAO. Quindi, ai fini del progetto, si utilizzerà solo quello.

4.5.1 La classe per lo Speech To Text: `speech_to_text.py`

Vediamo ora cosa è cambiato dalla precedente implementazione. Si inizia col dire che, come già descritto, lo Speech To Text avveniva utilizzando due moduli principali: Google e Vosk, avviando il secondo solo in caso di fallimento del primo. Ora, la funzione `record()` svolge un processo simile, ma avviando contemporaneamente sia Google che Vosk, tramite l'uso di due thread, utilizzando Vosk solo se Google impiega troppo tempo per elaborare una risposta (secondo un timer che può essere modificato) o se non produce un risultato valido. Questo è stato fatto poiché è stato riscontrato che Google restituisce risultati più attendibili rispetto a Vosk.

Ciò che prima era un insieme di funzioni richiamabili in altri codici, è ora stato reso una classe, con una sezione dedicata all'inizializzazione, per poter risparmiare tempo durante l'esecuzione, la funzione principale `record` e le due funzioni `google_speech_to_text()` e `vosk_speech_to_text()`.

L'inizializzazione

Ogni volta che viene creata un'istanza dell'oggetto Speech To Text, viene invocato il metodo responsabile dell'inizializzazione, che si occupa, innanzitutto, di impostare i vari parametri, utili all'SSH_Manager. Si mostri particolare attenzione al parametro `lang`, che è essenziale per impostare la lingua della trascrizione vocale. In base a questo, Vosk caricherà i modelli necessari.

Algoritmo 23 Inizializzazione `speech_to_text`

```
1: function INITIALIZE(nao_ip, nao_username, nao_password, lang)
2:     self.nao_ip ← nao_ip
3:     self.nao_username ← nao_username
4:     self.nao_password ← nao_password
5:     self.ssh_manager ← Create SSHManager(nao_ip, nao_username,
   nao_password)
6:     self.nao_path ← "/tmp/recording.wav"
7:     self.audio_path ← "/tmp/recording.wav"
8:     if lang = "italiano" then
9:         lang ← "it-IT"
10:    self.lang ← lang
11:    if self.lang = "it-IT" then
12:        self.vosk_model ← Create vosk.Model("vosk-model...")
13:    else
14:        self.vosk_model ← Create vosk.Model("vosk-model...")
15:    self.vosk_rec ← None
16:    self.google_thread ← None
17:    self.vosk_thread ← None
```

Google: `google_speech_to_text`

La funzione che richiama l'API di Google si occupa di utilizzare un oggetto `Recognizer` per richiamare la funzione `recognize_google()`, che effettua la trascrizione del file audio. Viene innanzitutto caricato il file in una variabile `audio`, tramite il metodo `from_file()` di `Audio_Segment`. La durata di esso viene ottenuta per questioni di ottimizzazione, poiché consente di determinare il tempo effettivo della registrazione, escludendo i momenti di silenzio. Questo valore viene poi utilizzato dalla funzione `record()` di `recognizer` per acquisire solo i dati audio rilevanti. Si ottiene, poi, il risultato della trascrizione, se presente.

Algoritmo 24 Funzione `google_speech_to_text`

```
1: function GoogleSpeechToText(audio_file, result_container, max_silence_time)
2:     audio ← AudioSegment.from_file(audio_file)
3:     duration ← len(audio)
4:     print("Audio file duration for STT:", duration)
5:     start_time ← time.perf_counter()
6:     recognizer ← sr.Recognizer()
7:     duration_without_silence ← max(0, duration - max_silence_time)
8:     With sr.AudioFile(audio_file) as source:
9:         audio_data ← recognizer.record(source, duration_without_silence)
10:    try:
11:        result_container["google"] ← recognizer.recognize_google(audio_data, lang=self.lang)
12:    catch: sr.UnknownValueError
13:        result_container["google"] ← "error"
14:    catch: sr.RequestError
15:        print("Could not request results from Google Cloud; check your internet connection.")
16:        result_container["google"] ← "error"
17:    nao_interface.measure_time("Google Speech-to-Text time:", start_time)
```

Vosk: `vosk_speech_to_text`

La funzione che si occupa di gestire la trascrizione audio con Vosk è strutturata in maniera diversa. I parametri della funzione sono: `audio_file`, il percorso del file audio e `result_container`, dove viene memorizzato il risultato. Dopo una verifica sull'esistenza del file audio, viene creato un modello `Kaldi_Recognizer`, se questo non esiste, e viene configurato per riconoscere anche le parole parziali e complete. Si entra poi nell'effettivo ciclo di lettura del file audio, frame per frame, tramite `wf.readframes(...)`. Ogni frammento viene passato a Vosk, per il riconoscimento, tramite `AcceptWaveForm`. Se non si verificano errori, il risultato viene estratto e inserito nel container di riferimento, per la restituzione del risultato.

Algoritmo 25 Funzione `vosk_speech_to_text`

```
1: function VOSK_SPEECH_TO_TEXT(audio_file, result_container)
2:     start_time ← time.perf_counter()
3:     if non esiste il file audio then
4:         return "error"
5:     apri audio_file con wave.open() come wf
6:     if self.vosk_rec è None then
7:         self.vosk_rec ← vosk.KaldiRecognizer(self.vosk_model,
   wf.getframerate())
8:         self.vosk_rec.SetWords(True)
9:         self.vosk_rec.SetPartialWords(True)
10:    rec ← self.vosk_rec
11:    text ← []
12:    try
13:    while True do
14:        data ← wf.readframes(...)
15:        if lunghezza di data è 0 then
16:            break
17:        if rec.AcceptWaveform(data) then
18:            text.append(json.loads(rec.Result())["text"])
19:    nao_interface.measure_time("Tempo per S2T Vosk: ",
   start_time)
20:    text.append(json.loads(rec.FinalResult())["text"])
21:    catch ValueError
22:    print("[ERRORE] Decoding JSON!")
23:    result ← " ".join(text).strip()
24:    if result è vuoto then
25:        result_container["vosk"] ← "error"
26:    else
27:        result_container["vosk"] ← result
28:    nao_interface.measure_time("Tempo per S2T Vosk end: ",
   start_time)
```

La funzione per la registrazione: `record`

La funzione `record()` è quella che verrà richiamata nel contesto di registrazione e riconoscimento vocale, eseguendo la registrazione dell'audio, trasferendo il file al sistema e avviando il riconoscimento tramite i motori di Google e Vosk. Rappresenta quindi il fulcro dello Speech To Text implementato nel sistema. Dopo le consuete inizializzazioni, avviene la registrazione della voce tramite

la funzione `audio_recorder_front()` di `nao_interface`, che vedremo successivamente. Viene poi definito il percorso del file audio, essenziale per poter salvare la registrazione.

Algoritmo 26 Funzione `record`

```
1: function RECORD(self, soglia)
2:     start_time ← time.perf_counter()
3:     max_silence_time, max_rec_time_with_silence, max_rec_time ← 0.7, 4.0,
   10.0
4:     nao_interface.audio_recorder_front(...)
5:     audio_file ← "/tmp/recording.wav"
6:     self.ssh_manager.get_file_from_nao(self.nao_path, audio_file)
```

Si verifica poi se ci sono esecuzioni attive, da una precedente esecuzione, dopodiché si creano i rispettivi container e i relativi thread, come anticipato in precedenza.

```
if self.google_thread è diverso da None then
    if self.google_thread è attivo then
        Join thread self.google_thread
    else
        Imposta google_disabled a True

if self.vosk_thread è diverso da None then
    Join thread self.vosk_thread

Crea result_container con chiavi "google" e "vosk"

Crea un nuovo thread self.google_thread per eseguire
self.google_speech_to_text con audio_file e result_container come argomenti

Crea un nuovo thread self.vosk_thread per eseguire self.vosk_speech_to_text
con audio_file e result_container come argomenti

if google_disabled è False then
    Avvia il thread self.google_thread

Avvia il thread self.vosk_thread
```

Inizia ora l'attesa del risultato di Google. Si noti che l'attesa è modificabile ed è stata inserita per migliorare le tempistiche complessive del sistema. Si prende poi il risultato di Google, se ne ha prodotto uno e ha rispettato i tempi; altrimenti si sceglie Vosk.

Attendere 3 secondi per il risultato di Google
Join thread self.google_thread con timeout di 3 secondi

if google_disabled è **True** o il risultato di google è **None** o il risultato di google è "error" **then**
    Stampa "Google ha fallito, prendo Vosk"
    Join thread self.vosk_thread
    Assegna final_result a result_container["vosk"]
else
    Assegna final_result a result_container["google"]

Misura tempo per la funzione record con start_time

Ritorna final_result

La funzione verrà quindi richiamata nel ciclo principale della Demo ogni qualvolta si dovrà effettuare una registrazione della voce dell'utente, per trascriverla in testo.

4.5.2 La registrazione da NAO: `audio_recorder_front`

La funzione `audio_recorder_front()` si trova nella libreria in C++. Essa viene richiamata tramite l'interfaccia discussa in precedenza, `nao_interface`. Per poter utilizzare i microfoni di NAO, per la registrazione, è stato scelto di utilizzare il modulo `ALAudioRecorder`, combinato con il modulo `ALAudioDevice`, per la gestione dell'energia. Il primo fornisce due funzioni essenziali per delimitare l'inizio e la fine della registrazione; il secondo fornisce, invece, la possibilità di leggere un valore, denominato energia, di cui si è discusso in precedenza, che equivale al livello di intensità del suono catturato dai microfoni.

Funzione per la calibrazione della Soglia

La calibrazione corretta dei parametri gioca, quindi, un ruolo fondamentale nella buona riuscita della registrazione. È necessario, prima dell'inizio di ogni sessione, effettuare una calibrazione della soglia, che è necessario impostare in base al tipo di soggetto.

A proposito di essa, per la sua impostazione corretta, è stata creata una funzione che, tramite il calcolo della media e della deviazione standard, permette di impostare automaticamente il valore ottimale.

Funzioni ausiliarie per Media e Deviazione Standard

Algoritmo 27 Calcolo della Media di un Vettore

```
1: function CalculateMean(values)
2:     sum ← 0.0
3:     for all value in values do
4:         sum ← sum + value
5:     return sum / length(values)
```

Algoritmo 28 Calcolo della Deviazione Standard di un Vettore

```
1: function CalculateStandardDeviation(values, mean)
2:     variance ← 0.0
3:     for all value in values do
4:         variance ← variance + (value − mean)^2
5:     return sqrt(variance/length(values))
```

Implementazione della funzione di Calibrazione

I parametri della funzione sono: `nao_ip`, per l'inizializzazione dei Proxy utilizzati, `calibration_time`, che rappresenta il tempo massimo in secondi per cui viene effettuata la calibrazione, `sample_interval`, che specifica l'intervallo di tempo tra una misurazione e l'altra, espresso in secondi (un intervallo più piccolo aumenta la frequenza di campionamento e raccoglie più dati), e `threshold_multiplier`, che rappresenta il fattore di moltiplicazione usato per calcolare la soglia finale e determina di quanto il valore della soglia deve essere sopra il rumore medio.

Per il calcolo effettivo della soglia è necessario attivare la lettura dell'energia tramite la funzione `enableEnergyComputation()`, e richiamare le due funzioni viste in precedenza.

Algoritmo 29 Calibrazione della soglia: calibrate_threshold

```
1: function CalibrateThreshold(nao_ip, calibration_time, sample_interval, threshold_multiplier)
2:     InitializeProxiesOnce(nao_ip)
3:     channels ← [0, 0, 1, 0]
4:     audioDeviceProxy.enableEnergyComputation()
5:     energy_samples ← []
6:     rec_time ← 0.0
7:     print "Calibrazione in corso..."
8:     while rec_time < calibration_time do
9:         front_energy ← audioDeviceProxy.getFrontMicEnergy()
10:        energy_samples.append(front_energy)
11:        rec_time ← rec_time + sample_interval
12:        Attendi per sample_interval secondi
13:    mean_energy ← CalculateMean(energy_samples)
14:    std_dev_energy ← CalculateStandardDeviation(energy_samples, mean_energy)
15:    optimal_threshold ← mean_energy + (threshold_multiplier × std_dev_energy)
16:    print "Calibrazione completata. Soglia ottimale: ", optimal_threshold
17:    return optimal_threshold
```

Implementazione di `audio_recorder_front`

Prima di analizzare l'implementazione della funzione, è necessario spiegare i suoi parametri:

- nao_ip, corrisponde all'ip del NAO, necessario per inizializzare i Proxy
- nao_path, è il percorso del NAO in cui viene salvata la registrazione
- soglia, rappresenta un valore che determina il livello minimo di volume (ampiezza del segnale audio) sotto il quale il suono non viene considerato come voce. Se il volume del suono registrato è inferiore a questa soglia, viene ignorato
- max_silence_time, è il tempo massimo di silenzio per cui, se non viene rilevata più voce, si interrompe la registrazione. Viene "attivato" solo se l'utente ha già parlato almeno una volta.
- max_rec_time_with_silence, è il tempo massimo di silenzio per cui, se non viene rilevata nemmeno una volta la voce, si interrompe la registrazione. Viene attivato se l'utente non ha mai parlato

- max_rec_time, rappresenta il tempo massimo assoluto della registrazione, necessario per non registrare per un tempo indefinito.

Si attiva la registrazione solo sul canale frontale, ignorando gli altri, ed eseguiamo prima di tutto una `stopMicrophonesRecording()` per garantire una corretta attivazione della registrazione.

Tramite `startMicrophonesRecording()`, invece, si avvia l'effettiva registrazione dal microfono, passando come parametri il percorso del nao, il formato del file, che dovrà essere in `.wav`, la frequenza di campionamento, che sarà `16kHz` e i canali da cui registrare.

Dopo aver inizializzato le variabili, si entra nel ciclo principale delle operazioni, con varie possibilità di uscita, in particolare, usciamo dal ciclo se:

- il tempo di silenzio raggiunge il tempo massimo di silenzio, se l'utente ha parlato almeno una volta

- il tempo di registrazione raggiunge il tempo massimo di registrazione con silenzio, se l'utente non ha mai parlato

- il tempo di registrazione raggiunge il tempo massimo assoluto di registrazione

La condizione all'interno del ciclo serve per rilevare il silenzio, in particolare, se l'energia rilevata è sopra la soglia, vuol dire che l'utente sta parlando e, se è la prima volta che la soglia viene superata, imposta la variabile `already_spoken` a `True` e, in ogni caso, viene azzerata `silence_time`. Se, invece, l'energia rilevata è sotto la soglia, vuol dire che l'utente non sta parlando e `silence_time` viene incrementato di `100ms`.

Sempre all'interno del ciclo principale, viene incrementato `rec_time` ogni `100ms` e viene fatta una pausa di `100ms` per indicare il tempo che scorre.

Algoritmo 30 Registrazione audio dal microfono frontale di NAO: audio_recorder_front

```
1: function AUDIO_RECORDER_FRONT(nao_ip, nao_path, soglia, max_silence_time, max_rec_time_with_silence, max_rec_time)
2:     InitializeProxiesOnce(nao_ip)
3:     channels ← [0, 0, 1, 0]
4:     audioRecorderProxy.stopMicrophonesRecording()
5:     audioDeviceProxy.enableEnergyComputation()
6:     audioRecorderProxy.startMicrophonesRecording(nao_path, "wav", 16000, channels)
7:     front_energy ← 0.0
8:     rec_time ← 0.0
9:     silence_time ← 0.0
10:    already_spoken ← false
11:    while (silence_time < max_silence_time and already_spoken) or (rec_time < max_rec_time_with_silence and not already_spoken) do
12:        if rec_time ≥ max_rec_time then
13:            break
14:        front_energy ← audioDeviceProxy.getFrontMicEnergy()
15:        print("Front: ", front_energy)
16:        print("Silence Time: ", silence_time)
17:        if front_energy ≥ soglia then
18:            if not already_spoken then
19:                already_spoken ← true
20:            silence_time ← 0.0
21:        else
22:            silence_time ← silence_time +0.1
23:        rec_time ← rec_time +0.1
24:        qi.os.msleep(100)
25:    audioRecorderProxy.stopMicrophonesRecording()
26:    audioDeviceProxy.disableEnergyComputation()
```

Alla fine della funzione, la registrazione viene interrotta, così come la rilevazione dell'energia.

4.5.3 Considerazioni sullo Speech To Text

È stato quindi descritto come funziona il nuovo meccanismo di Speech To Text, assieme alla registrazione dal microfono frontale di NAO.

Questo nuovo sistema garantisce una rilevazione del silenzio corretta e controllata, modificabile in futuro secondo le esigenze del sistema.

Il processo di registrazione è gestito in modo da garantire che venga rilevato il parlato quando l'intensità del suono supera una soglia predefinita. Una volta che il parlato viene rilevato, il sistema registra fino al momento in cui non si rileva più suono per un determinato periodo, quindi la registrazione viene interrotta.

Inoltre, la funzionalità di calibrazione consente di determinare la soglia ottimale per il rilevamento del parlato, tenendo conto del rumore di fondo presente durante il processo di calibrazione. Ciò assicura che il sistema sia in grado di adattarsi ai vari ambienti e alle diverse condizioni acustiche, migliorando così l'affidabilità e la precisione del riconoscimento vocale.

4.6 Il riconoscimento delle emozioni

Un altro punto fondamentale del progetto è l'integrazione del riconoscitore delle emozioni tramite videocamera, già implementato in precedenza.

Il nuovo script presenta le stesse caratteristiche del precedente, con l'unica differenza che è stata creata una classe apposita per poterlo utilizzare dove necessario.

L'unica aggiunta che è stata fatta riguarda la logica del riconoscimento delle emozioni più frequenti. Il focus principale è stato l'usabilità, poiché l'uso proposto dal dipartimento di Medicina e Chirurgia prevede il riconoscimento di esse solo in determinati contesti, quali ad esempio la risposta a domande specifiche (la proposta di un task, ad esempio). Per questo motivo, era essenziale garantire che l'emozione rilevata corrispondesse a quella del momento esatto in cui l'utente rispondeva, quindi per un periodo di tempo molto breve. Dai test effettuati è emerso che, nella maggior parte dei casi, l'output prevalente era "neutrale", mentre quella che avrebbe potuto riflettere meglio la reale emozione dell'utente veniva spesso trascurata. Questo è dovuto al fatto che, naturalmente, si tende ad esprimere maggiormente emozioni neutrali, ma, nel caso specifico del test, non era quella di cui si aveva bisogno. Per questo motivo è stata implementata un'ulteriore funzione che ha la possibilità di assegnare un peso ridotto ad una o più emozioni, in modo da essere considerate meno rispetto a quelle più interessanti nel contesto descritto.

È importante sottolineare, inoltre, che il modello utilizzato nel lavoro non raggiunge un'accuratezza del 100%, il che significa che, pur mantenendo un'espressione neutrale, potrebbe occasionalmente rilevare un'emozione diversa. In particolare, dai pochi test effettuati, è emerso che le emozioni "Disprezzo" e "Sorpresa" sono quelle che si confondono maggiormente. Pertanto, è necessario effettuare un'adeguata calibrazione, modificando il peso delle emozioni quando ritenuto opportuno.

Implementazione di `most_common_value_2`

La funzione proposta ha l'obiettivo di determinare l'emozione predominante in una lista. Gli è stato dato il suffisso "2" poiché è stata comunque mantenuta la precedente funzione nel codice. Come detto prima, vengono tenuti in considerazione pesi ridotti, in particolare per le emozioni "Neutrale" e "Sorpresa", ma è possibile aggiungere altre emozioni a cui applicare pesi personalizzati, se necessario. Nei parametri, oltre alla lista di input e alle emozioni con i relativi pesi, è stata aggiunta anche una soglia (threshold) che consente di effettuare un ulteriore controllo alla fine del processo. Prima di tutto, la funzione conta le occorrenze delle emozioni in una lista, passata come input, e ogni emozione rilevata è memorizzata in un dizionario con nome come chiave e il numero di occorrenze come valore associato. Se nella lista sono contenute le emozioni passate come parametri, a queste viene applicato un peso diverso, rendendole meno (o più) influenti delle altre.

Algoritmo 31 Funzione per determinare il valore più comune con pesi per emozioni specifiche: `most_common_value_2`

1: **function** MOST_COMMON_VALUE_2(self, lst, neutral = 5, surprise = 7, neutral_weight = 0.8, surprise_weight = 0.6, threshold = 0.10)
2: **Inizializza** weighted_counts come un dizionario vuoto
3: **for** item in lst **do**
4: **Incrementa** weighted_counts[item] di 1.0
5: **if** neutral è presente in weighted_counts **then**
6: **Moltiplica** weighted_counts[neutral] per neutral_weight
7: **if** surprise è presente in weighted_counts **then**
8: **Moltiplica** weighted_counts[surprise] per surprise_weight
9: **for** emotion, weight in weighted_counts.items() **do**
10: **Stampa** "(Prima della normalizzazione: Emozione emotion: weight)"
11: **Calcola** total_weight come la somma dei valori in weighted_counts

A questo punto viene effettuata una normalizzazione per questioni di leggibilità; ora tutti i pesi sommati daranno 1. Dopo aver effettuato un ordinamento crescente, viene effettuato un ulteriore controllo sull'emozione "Neutrale", per verificare se, anche dopo aver applicato il peso ridotto, risulta la più frequente. Se la differenza tra questa e la seconda emozione più frequente è inferiore a `threshold`, passato come parametro, viene considerata la seconda.

for emotion in weighted_counts **do**
Normalizza weighted_counts[emotion] dividendo per total_weight
for emotion, weight in weighted_counts.items() **do**
Stampa "(Emozione dopo emotion: round(weight, 2))"
Ordina weighted_counts per valore, in ordine decrescente, in sorted_items
Assegna most_common a sorted_items[0][0]
Assegna second_common a sorted_items[1][0] se len(sorted_items) > 1, altrimenti None
if most_common è uguale a neutral **then**
Assegna neutral_final_weight a weighted_counts[neutral]
Assegna second_final_weight a weighted_counts.get(second_common, 0)
if la differenza tra neutral_final_weight e second_final_weight è inferiore a threshold **then**
Assegna most_common a second_common
Ritorna most_common

Questa funzione verrà richiamata nel codice principale tramite:

Algoritmo 32 Riconoscimento emozioni

```
1: function DETECT_START
2:     Emozioni_rilevate ← lista_vuota
3:     print("Inizio riconoscimento emozioni.")
4: function DETECT_END
5:     print("Riconoscimento emozioni terminata.")
6:   if Emozioni_rilevate non è vuota then
7:         Emozione_comune ← most_common_value_2(Emozioni_rilevate)
8:         print("Emozione più frequente rilevata: " + emozioni_dict[Emozione_comune])
9:         Restituisci emozioni_dict[Emozione_comune]
10:  else
11:        print("Nessuna emozione rilevata.")
12:        Restituisci Nessuna
```

4.6.1 Considerazioni sul rilevamento delle emozioni

Il riconoscimento delle emozioni avrà un ruolo cruciale nel flusso del dialogo, permettendo di modificarlo in base alle esigenze specifiche del contesto.

Per quanto riguarda il tracciamento della testa dell'utente, non sono state apportate modifiche. È importante sottolineare che nel codice principale del programma, il riconoscitore delle emozioni viene avviato su un thread separato.

Questo, a sua volta, attiverà il tracciamento della testa su un altro thread, consentendo di eseguire operazioni aggiuntive, come la registrazione della voce, in parallelo.

4.7 Il programma in locale

Questa breve sezione funge solo a scopo informativo, ed è dedicata alla realizzazione di un programma, speculare a quello per NAO, ma che è possibile avviare da qualunque computer, senza l'utilizzo del robot.

Il programma realizzato replica in modo fedele il flusso di lavoro previsto per il robot, ma senza utilizzare i moduli di NAO. Questo approccio rende il sistema facilmente adattabile a diversi ambienti e più accessibile a tutti i membri del team di sviluppo, che possono lavorare su funzionalità e miglioramenti senza doversi preoccupare delle limitazioni fisiche del robot.

Inoltre, l'utilizzo di un sistema simile, ma non vincolato al robot, permette di testare in parallelo diverse implementazioni, ottimizzare il codice e correggere eventuali bug in modo più immediato. Una volta che una funzionalità è stata testata e ottimizzata nel programma per computer, può essere facilmente trasferita e adattata al robot senza particolari difficoltà, semplicemente utilizzando i suoi moduli e gestendo l'invio dei dati.

Infine, questa versione del programma può essere utile anche per scenari in cui il robot non è disponibile per motivi logistici, permettendo comunque di continuare lo sviluppo e i test senza interruzioni.

Non si vedrà l'implementazione specifica, poiché sarebbe una ripetizione di quanto detto fino ad ora. È importante sapere come il flusso sia identico a quello su NAO, senza utilizzare, ovviamente, i suoi moduli e le sue funzioni.

4.8 Considerazioni su ottimizzazioni e usabilità

Conclusa la descrizione di tutti i nuovi script implementati, è necessario fare alcune considerazioni tecniche.

Durante la scrittura del codice ci si è imbattuti in importanti problemi legati principalmente alla stabilità e alle tempistiche. Infatti, il dialogo, per essere il più naturale possibile, deve risultare fluido e chiaro, senza interruzioni o ritardi evidenti. Ogni interazione deve quindi apparire come una conversazione continua e senza intoppi, in modo da evitare disorientamenti da parte del soggetto.

Replicare in tutto e per tutto l'interazione umana è impossibile, soprattutto perché ci sono dei tempi che non dipendono dall'implementazione degli script,

ma dalla rete, per citarne uno(i tempi di risposta di Google, ad esempio). È possibile, però, focalizzarsi sui punti più delicati del sistema, analizzandone i tempi e verificando dove poter interagire per ridurli.

4.8.1 Gestione delle tempistiche

In ogni script, dove è stato considerato fondamentale, è stato inserito un meccanismo di gestione del tempo, per valutare le criticità e verificare dove eseguire miglioramenti e ottimizzazioni.

Tramite la funzione `measure_time()`, di `nao_interface`, è stato possibile identificare i colli di bottiglia presenti nel codice, valutando principalmente le due fasi più importanti del dialogo: Text To Speech e Speech to Text.

All'inizio del lavoro i tempi risultavano molto lunghi, con attese che variavano dai 4 ai 6 secondi, circa, dal momento in cui l'utente smetteva di parlare, fino alla successiva frase pronunciata da NAO.

Text To Speech

Per questo motivo è stato deciso, in primis, di parallelizzare il flusso del Text To Speech, come descritto in precedenza, eseguendo uno "split" della frase, in modo da risparmiare sui tempi di attesa di Google.

Durante questa fase, oltre ai tempi tecnici necessari per ottenere una risposta da Google, inizialmente variabili in base alla lunghezza della frase, si osserva ora una maggiore costanza, poiché ogni frase viene suddivisa equamente. Ciò che risultava particolarmente lento era l'invio dei file tramite SSH, da macchina a NAO. Questo, come anche l'attesa di Google, deriva principalmente dalla rete, che nonostante le ottimizzazioni, risulta essere il nemico principale del tempo. Ma c'è una soluzione anche a questo: utilizzare il cavo LAN; infatti, NAO ha la possibilità di essere collegato, tramite cavo LAN, direttamente al router. Inizialmente, questi tempi erano elevati per un dialogo, variando tra 300 ms e 600 ms, mentre ora si attestano tra 50 ms e 100 ms . Si tratta di un'ottimizzazione significativa, sebbene implichi la presenza di un cavo aggiuntivo.

Speech To Text

Per quanto riguarda la fase di Speech To Text, è stato deciso di non parallelizzare il flusso, ma focalizzarsi piuttosto sulla gestione ottimale della funzione di registrazione, in modo da renderla il più reattiva possibile. Anche in questo caso, si è limitati dall'attesa di Google e Vosk. Ma, a differenza di prima, non si è vincolati da file che potrebbero risultare troppo lunghi, poiché le risposte da dare a NAO risultano essere brevi e concise. Questo assicura di avere file

audio di breve durata e, conseguentemente, risparmiare sui tempi di risposta di Google. Per Vosk la situazione è diversa, poiché essendo offline, non ha tempi alti e nulla vieta, in futuro e se risulta essere una buona alternativa, di considerarlo come unica alternativa.

Anche in questo caso, l'utilizzo di un cavo LAN è consigliato per accorciare i tempi di invio del file.

4.8.2 Stabilità e gestione delle risorse

Sulla stabilità se ne è già parlato in precedenza; in particolare, è stata descritta una funzione in grado di terminare correttamente il programma, con i suoi thread.

Per quanto riguarda invece la gestione delle risorse, è importante far presente come, per i percorsi dei file di Text To Speech e Speech To Text, siano state utilizzate le cartelle temporanee sia del computer che di NAO, in modo da eliminare i file non appena si esegue un riavvio di entrambi. Per quanto riguarda il Text To Speech, inoltre, è stato comunque aggiunto un meccanismo di eliminazione dei file, poiché nominando i file audio secondo un progressivo (che si resetta ogni volta che vengono avviati i thread), è necessario eliminarli singolarmente per non utilizzare troppa memoria.

Per lo Speech To Text il discorso è diverso, poiché ogni nuovo file registrato presenta lo stesso nome del precedente, quindi viene semplicemente sovrascritto, evitando di appesantire NAO.

Conclusione

Nella tesi realizzata è stata data priorità ad una panoramica sui SAR, riservando un focus particolare a NAO e analizzando, al contempo, il contesto generale dell'intero progetto. Successivamente, sono stati ricostruiti e messi insieme le parti che lo compongono, descrivendo le implementazioni precedenti e focalizzando pregi e difetti. L'ultimo capitolo, in particolare, si concentra sul fulcro del lavoro, svolto e sviluppato negli ultimi mesi e che, come già ribadito, non sarebbe stato possibile senza l'aiuto del dipartimento di Medicina e Chirurgia e il dipartimento di Ingegneria, con l'attenta supervisione del prof. Monica. Il risultato è un sistema certamente non ancora del tutto maturo ma tale, tuttavia, da rappresentare un'ottima e imprescindibile base progettuale a favore di chi prenderà, in seguito, l'incarico di portare avanti il progetto. Con l'integrazione dei vari pezzi e l'ottimizzazione di questi ultimi, si realizza, sin da oggi, l'opportunità di riuscire ad essere più che mai pronti, in modo efficace e risoluto, a compiere un passo in avanti verso il lavoro finale, su cui manca ancora la messa in opera del ragionatore, completato dai vari task e perfezionato nei dialoghi, scelti in funzione di una totale ottimizzazione della performance e in conformità alle migliori attese pronosticate.

Il sistema, oltre ad essere stabile e funzionale, è anche facilmente adattabile alle specifiche esigenze, soprattutto per quanto riguarda la parte di Speech To Text, che, per quanto osservato, rappresenta il momento più delicato dell'intero processo di dialogo. Con i parametri della funzione di registrazione è possibile, poi, regolare diversi aspetti fondamentali, utili a garantire una trascrizione accurata e reattiva, soprattutto per quanto riguarda i tempi. Questa flessibilità consente di adattare il sistema a diversi casi d'uso; ad esempio, in un contesto in cui il soggetto parla lentamente. In questo caso è possibile aumentare la tolleranza ai momenti di pausa, evitando interruzioni premature della registrazione.

Per il riconoscimento delle emozioni, invece, saranno necessari ulteriori test, per capire quali, tra le diverse espressioni, dovrebbero essere considerate meno rilevanti o maggiormente enfatizzate rispetto ad altre. Dai risultati emersi, oltre all'emozione "Neutra", gli atteggiamenti espressivi più frequenti risultano essere "sorpresa" e "disprezzo", ma sarà possibile, comunque, ottenere dati più

concreti solo in considerazione di ulteriori prove.

Più in generale, le avanzate capacità dei SAR, in particolare di NAO, aprono innumerevoli opportunità nell'ambito dell'assistenza nei processi interattivi. Ciò che è stato sviluppato finora rappresenta, infatti, solo l'inizio delle reali potenzialità di questo robot. In prospettiva, quindi, è facile presumere che le aspettative future, in merito, si riveleranno molto promettenti e al di sopra di ogni immaginazione possibile.

A partire da questi aspetti il prossimo passo è, sicuramente, l'integrazione del ragionatore, per poter condurre finalmente a compimento il progetto. Oltre al ragionatore, sarebbe, nondimeno, un lavoro altrettanto importante convogliare e associare i movimenti del dispositivo robotico alle emozioni, manifestate e manifestabili da NAO stesso. Infatti, secondo l'opinione dei collaboratori, appartenenti al Dipartimento prima citato, e che lavorano nell'ambito della riabilitazione cognitiva, oltre alla rilevazione degli aspetti emotivi, è necessario che NAO possa essere in grado di esprimere ancor meglio l'intero ventaglio di emozioni esprimibili, poiché questo aspetto fondamentale sarebbe d'aiuto nel miglioramento dei processi di interazione, evitando, di conseguenza, che il soggetto si annoi. I gesti, in questo caso, rappresentano il miglior modo per esprimere il senso compiuto delle suggestioni e dei feedback emozionali riproducibili.

Un'altra funzionalità da aggiungere al lavoro è il cambio di colore degli occhi, che assieme ai gesti, rappresenta un ottimo indicatore per capire e trasmettere le emozioni di NAO.

Inoltre, come già menzionato nelle tesi sviluppate in precedenza sul tema oggetto di questo lavoro, la creazione di una GUI per l'interfacciamento alle funzioni e al ragionatore rappresenterebbe la svolta più significativa e dirimente, fornendo un accesso ancor più intuitivo e immediato alle funzionalità del sistema, oltre a dimostrarsi un mezzo utile per gli operatori, permettendo loro di aggiungere in autonomia eventuali modifiche.

Bibliografia

[1] Felisari A. Modellazione di un dialogo per la riabilitazione cognitiva con xai. Tesi di laurea triennale, UNIPR, 2024.

[2] Jordan A., Al-Hindawi A., Tiffany Ng, and Vizcaychipi M. P. Scoping review on the use of socially assistive robot technology in elderly care. *BMJ Open*, 8(2), 2018.

[3] Langer A., Feingold-Polak R., Mueller O., Kellmeyer P., and Levy-Tzedek S. Trust in socially assistive robots: Considerations for use in rehabilitation. *Neuroscience e Biobehavioral Reviews*, 104:231–239, 2019.

[4] De Carolis B., Palestra G., and Pino O. Facial expression recognition from nao robot within a memory training program for individuals with mild cognitive impairment. *Elettronico*, 2474:37–42, 2019.

[5] De Carolis B., Ferilli S., and Palestra G. Simulating empathic behavior in a social assistive robot. *Multimedia Tools and Applications*, 76(4):5073–5094, 2017.

[6] Aldebaran. Naoqi developer guide.

[7] Loper E. and Bird S. NLTK: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, pages 63–70, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[8] Zhou E., Shi Z., Qiao X., Matarić M. J., and Bittner A. K. Designing a socially assistive robot to support older adults with low vision. In H. Li et al., editors, *Social Robotics. ICSR 2021. Lecture Notes in Computer Science*, volume 13086. Springer, Cham, 2021.

[9] Python Software Foundation. ctypes – a foreign function interface for python, 2024.

[10] Kalita L. Socket programming. *International Journal of Computer Science and Information Technologies*, 5(3):4802–4807, 2014.

[11] Saponaro L. Socially assistive robot come strumenti nell'intervento contro la fragilità dell'anziano: uno studio pilota. Tesi di laurea triennale, UNIPR, 2023.

[12] Corni M. Il dialogo terapeutico modellato con ia spiegabile su robot nao. Tesi di laurea triennale, UNIPR, 2022-2023.

[13] Adriana D. e Aruffo D.;Ettore e Evaristo Marigliano V. e Servello. La riabilitazione cognitiva, 2012.

[14] Giuseppe Palestra and Olimpia Pino. Detecting emotions during a memory training assisted by a social robot for individuals with mild cognitive impairment (mci). *Multimedia Tools and Applications*, 79(47):35829–35844, 2020.

[15] Ronald C. Petersen. Mild cognitive impairment as a diagnostic entity. *J Intern Med*, 256(3):183–194, Sep 2004.

[16] O. Pino, G. Palestra, R. Trevino, et al. The humanoid robot nao as trainer in a memory program for elderly people with mild cognitive impairment. *International Journal of Social Robotics*, 12:21–33, 2020.

[17] ResearchGate. Secure autonomous cloud brained humanoid robot assisting rescuers in hazardous environments - scientific figure on researchgate, 2025.

[18] Wikipedia Contributors. Nao (robot).

[19] Di Mambro S. Riconoscimento automatico di emozioni con robot umanoide nao per applicazioni di riabilitazione cognitiva. Master's thesis, UNIPR, 2022-2023.

[20] Deepti Saraswat, Pronaya Bhattacharya, Ashwin Verma, Vivek Kumar Prasad, Sudeep Tanwar, Gulshan Sharma, Pitshou N Bokoro, and Ravi Sharma. Explainable ai for healthcare 5.0: opportunities and challenges. *IEEe Access*, 10:84486–84517, 2022.

[21] NoCamels Team. study-socially-assistive-robot-helps-rehab-for-stroke-patients, 2024.

[22] Lawson M. V. *Finite Automata*, pages 117–143. Birkhäuser Boston, Boston, MA, 2005.

[23] Lifschitz V. *Answer set programming*, volume 3. Springer Heidelberg, 2019.

Ringraziamenti

Se siete arrivati fin qui vuol dire che avete letto tutta la tesi, complimenti!

Voglio iniziare ringraziando il Prof. Alessandro dal Palù, i cui insegnamenti sono stati essenziali durante tutto il lavoro. Di professori così se ne incontrano pochi e se l'ho scelta come tutor è perché la stimo molto. Ringrazio inoltre i miei correlatori: il Professore Riccardo Monica, che mi ha sostenuto e mi ha consigliato durante la realizzazione del progetto, e la Professoressa Olimpia Pino, i cui consigli riguardo all'interfacciamento con i soggetti sono stati fondamentali.

Grazie ad Aurora, con cui ho collaborato durante la prima fase del progetto e che si è rivelata una compagna ed un'amica eccezionale. Grazie a Benedetta e Rachele, del dipartimento di Medicina e Chirurgia, laboratorio di Psicologia cognitiva, che hanno fornito il materiale e il cui lavoro ha un grandissimo valore. In bocca al lupo per tutto.

Passiamo ora a coloro che hanno reso possibile tutto ciò: la mia famiglia, a cui voglio un bene infinito. A mamma, che ho tartassato di chiamate sin dal primo giorno, anche solo per sentire la sua voce, non sai quante volte ero giù e il solo sentirti mi ha risollevato il morale. A papà, che non ringrazierò mai abbastanza per tutto ciò che ogni giorno fa per me, sei la mia fonte d'ispirazione e l'uomo che vorrei diventare. A mia sorella Sofia, che mi ha insegnato cosa vuol dire avere una persona che ci sarà per tutta la vita, ti devo tutte le risate che ci facciamo quando stiamo insieme. A mia nonna Nancy, che mi guarda da lassù. A te dedico ogni esame passato, perché so che è anche un po' merito tuo. Spero di avervi reso orgogliosi, non sarebbe mai stato possibile grazie a voi, ed è solo uno dei mille obiettivi che raggiungeremo insieme.

Voglio ringraziare i miei parenti di San Marco dei Cavoti, tutti i miei cugini, gli zii, coloro che mi hanno cresciuto. A tutte le domeniche che abbiamo passato e trascorreremo insieme.

Un grazie speciale va anche ai parenti di Francavilla Fontana, tutti i chilometri che ci separano non bastano a tenere distante l'affetto che ci lega. Grazie a Nonna Antonietta, che so che farebbe qualunque cosa per me, e non mi ha mai fatto mancare nulla. Grazie a Nonno Antonio, per tutte le volte che mi hai portato nella tua amata campagna e per tutte le volte che mi fai sentire

speciale quando varco la porta di casa e sento: "Antonio!", con infinito stupore. A tutti gli zii, ai cugini (i nuovi e i "vecchi") e a chiunque mi abbia supportato.

Grazie a tutti i miei nuovi amici di Parma, con cui ho preparato gran parte degli esami e senza i quali non sarei riuscito ad affrontare questo percorso con la stessa felicità. Alle serate che abbiamo passato insieme e a quelle che non abbiamo passato (perché paccavo).

Grazie agli amici di Benevento, che mi accompagnano da quando sono piccolo e che spero rimarranno per sempre nella mia vita. A Nazzareno, il mio coinquilino e amico dal primo anno di liceo, con cui ho condiviso questi 3 anni. Ti ringrazio per tutto ciò che hai fatto per me. A Francesco, che sono grato di avere nella mia vita; ci sarò sempre per te e so che per te è lo stesso. A Nicola, non ho mai riso con nessuno come con te. Rappresentate un punto fermo da ormai diversi anni. A Ludovica, che per me c'è sempre.

Un ringraziamento speciale va ad una persona con cui ho condiviso tanto e che rimarrà sempre nel mio cuore. È doveroso ringraziarti, in quanto più di tutti sai cosa ho provato durante questo percorso, e più di tutti mi hai aiutato durante i momenti di difficoltà. Grazie, Alessia.

Grazie a coloro che ho conosciuto durante il percorso, a chi mi ha svegliato la mattina, prima di un esame, a chi è stato sveglio fino a tardi per sentirmi ripetere.

Per ultimo, ma non per importanza, voglio ringraziare NAO, il robot su cui ho lavorato e che mi ha permesso di sviluppare le mie capacità e di ampliare le mie conoscenze.

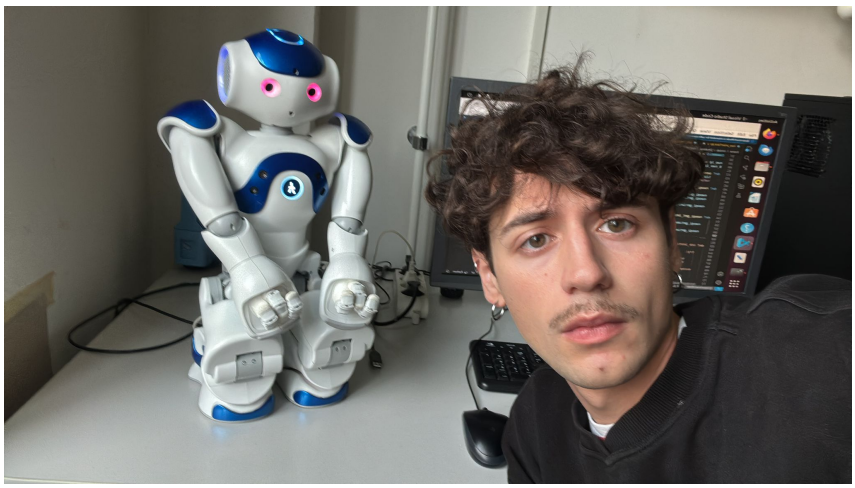


Figura 4.2: Io e NAO