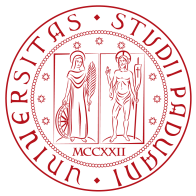


Università di Padova

Dipartimento di Ingegneria dell'Informazione

Tesi di laurea triennale in Ingegneria Informatica

Progettazione e sviluppo di un modulo di controllo per una sedia a rotelle implementato tramite Arduino

Relatore
Prof. Luca Tonin

Co-relatore
Piero Simonetto

Laureanda
Anita Giacomin

Numero di matricola
2016262

Anno Accademico
2022-2023

Abstract

La tesi è volta a descrivere la progettazione e l'implementazione di un modulo per il controllo del movimento di una sedia a rotelle motorizzata.
L'obiettivo principale del lavoro è stato quello di interfacciare il sistema proprietario della sedia con lo standard ROS tramite un Arduino, al fine di semplificare lo sviluppo di diverse metodologie di controllo a più alto livello, ad esempio tramite la decodifica di segnali cerebrali captati da un caschetto elettroencefalografico.
La soluzione sviluppata è poi stata implementata in laboratorio, verificando come il sistema fosse effettivamente in grado di controllare in modo efficace la velocità e la direzione di una sedia a rotelle, dimostrando così la fattibilità e l'utilità di una soluzione basata su Arduino e ROS per il controllo di dispositivi di assistenza alla mobilità.

Indice

Lista delle figure

Lista delle tablelle

Lista degli acronimi

FTC . . . . . Fundamental Theorem of Calculus

EEG . . . . . Elettroencefalogramma

BCI . . . . . Brain-Computer Interface

ROS . . . . . Robot Operating System

RPC . . . . . Remote Procedure Call

IDE . . . . . Integrated Development Environment

1 Introduzione

Il settore medico dell'ingegneria è stato, negli ultimi anni, oggetto di continuo avanzamento, grazie alla nascita di nuove tecnologie, metodologie e dispositivi dalle più varie applicazioni. Non è da meno la robotica assistiva, la quale sfrutta la robotica e l'intelligenza artificiale per migliorare la qualità della vita delle persone con disabilità motorie. In particolare, le tecnologie che consentono il controllo e l'interazione delle sedie a rotelle motorizzate sono diventate sempre più rilevanti e promettenti [2].
L'inclusione delle persone con disabilità motoria avviene non solo a livello di infrastrutture pubbliche ma soprattutto a livello individuale, con la progettazione di apparecchiature pensate per soddisfare le diverse esigenze riscontrabili. Il numero di sistemi di controllo presenti sul mercato riflette i vari bisogni dei consumatori, da quelli che necessitano di joystick particolarmente sensibili e facilmente manipolabili (Figure 1.1a), a quelli che, al contrario, operano con forza eccessiva e che cercano un dispositivo più resistente [3]; chi non ha capacità di manipolazione manuale può usufruire di pedali appositi o di Head Arrays (un set di sensori posizionati in prossimità della testa e attivati dal movimento del capo, della lingua o persino dal respiro (Figure 1.1b)) [4], ma anche touchpad, comandi vocali, pulsanti direzionali e molto altro [5].
Per facilitare ulteriormente l'utilizzo delle sedie motorizzate, sono stati elaborati sistemi di guida semi-autonoma: una modalità di funzionamento strutturata gerarchicamente, composta da sensori [6], processori, segnali e funzioni che permettono manovre complesse, come per esempio la prevenzione delle collisioni o il back-tracing [7].
Ad oggi, lo sviluppo di algoritmi di intelligenza artificiale e computer vision ha permesso la ricerca di sistemi di controllo alternativi per le sedie a rotelle motorizzate, consentendo agli utenti con disabilità

(a) Micro Joystick

(b) Head Array

Figure 1.1: Esempi di dispositivi di controllo di sedie a rotelle motorizzate

di guidare il veicolo senza impulso motorio [8].
Il movimento volontario degli occhi o delle mani, così come l'espressività facciale o lo sposta- mento del capo in una direzione piuttosto che in un'altra, o addirittura la sola intenzione di movimento, generano pattern di segnali cerebrali che possono essere riconosciuti e sfruttati come sistema di controllo [9]. Utilizzando la tecnologia di eye-tracking, i movimenti oculari vengono catturati e tradotti in comandi direzionali per la sedia a rotelle. Allo stesso modo, i movimenti delle mani possono essere rilevati attraverso sensori di gesti, consentendo agli utenti di controllare la sedia attraverso movimenti precisi.

1.1 Brain-computer interfaces per il controllo di sedie a rotelle intelligenti

Le Brain Computer Interfaces (BCI), una sotto-categoria delle interfacce neurali, sono sistemi che operano acquisendo e analizzando segnali cerebrali e li decodificano per identificare specifici comandi [10]. Questi ultimi vengono infine trasmessi ad un computer, il quale elabora i messaggi ricevuti, eliminando eventuali rumori e traducendoli in comandi per un determinato dispositivo esterno come ad esempio un braccio robotico o una sedia a rotelle [11][12].
I metodi per captare i segnali elettromagnetici con cui il cervello umano lavora possono essere più o

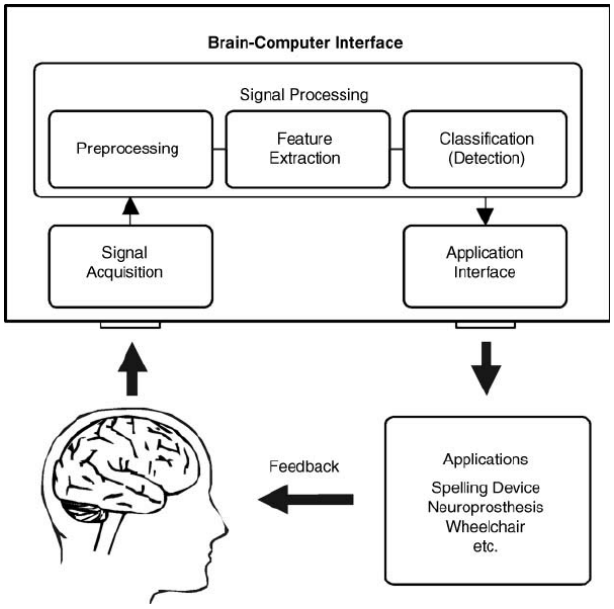


Figure 1.2: Componenti e feedback-loop delle BCI [1]

meno invasivi a seconda del livello di informazione che si vuole ottenere. Non sempre bastano degli elettrodi posizionati sullo scalpo del soggetto poiché i tessuti del cranio e del cuoio capelluto attenuano i segnali biologici e si potrebbe avere perdita di informazioni importanti; in alcuni casi si effettua un'operazione chirurgica per impiantare microarray intracorticali che registrano l'attività dei singoli neuroni [10].

Le componenti di una BCI sono principalmente quattro: l'acquisizione dei segnali, l'estrazione delle caratteristiche legate allo specifico compito mentale effettuato dal soggetto, la classificazione (funzioni di traduzione) e l'interfaccia applicativa [13]. Inoltre, in ogni BCI è presente un feedback (generalmente visivo) che permette all'utente di adattarsi e imparare ad utilizzare il sistema (Figure 1.2).

Per quanto riguarda la tecnologia assistiva, le BCI costituiscono promettenti strumenti di controllo nella navigazione spaziale delle sedie a rotelle motorizzate [12]. Essere in grado di effettuare spostamenti in modo autonomo è un importante fattore nella qualità della vita e nell'indipendenza personale di chi convive con una grave disabilità motoria.

L'intenzione di movimento genera pattern di segnali cerebrali che possono essere riconosciuti e sfruttati come sistema di controllo [9]. Ciò richiede una stretta collaborazione tra l'utente e il sistema BCI, poiché l'individuo deve apprendere come generare pattern di attività cerebrale distintivi per i diversi comandi di movimento.

L'approccio descritto è possibile grazie a dei sistemi basati su elettroencefalogramma (EEG), i quali

possono interpretare i segnali elettrofisiologici del cervello per tradurli in azioni specifiche per la sedia a rotelle, consentendo agli utenti di guidare la sedia attraverso, ad esempio, l'immaginazione motoria [14].
Questi sviluppi rappresentano un passo avanti significativo nell'innovazione dell'assistenza alla mobilità per le persone con disabilità, offrendo opzioni di controllo personalizzate e adattabili alle capacità individuali.

1.2 Motivazioni e obbiettivi della tesi

Per poter usufruire dei metodi di controllo alternativi è necessario stabilire una connessione tra essi e la sedia motorizzata. Questo collegamento è spesso realizzato tramite computer, sul quale si possono progettare e sviluppare dei software su misura che permettono di interagire in modo specifico con il sistema sedia - metodo di controllo.
L'obiettivo di questa tesi è quello di creare un'interfaccia tra una sedia a rotelle motorizzata e un computer, utilizzando il framework di comunicazione Robot Operating System (ROS) e un processore Arduino come piattaforma hardware. L'uso di ROS offre un ambiente flessibile e scalabile per lo sviluppo di sistemi robotici complessi, mentre Arduino fornisce una piattaforma di prototipazione rapida e accessibile per l'interfacciamento hardware. Saranno trattati la progettazione e lo sviluppo del collegamento (hardware e software) ROS-Arduino, la configurazione del sistema della sedia a rotelle, l'ambiente e la gestione dei messaggi ROS e la traduzione di questi ultimi in comandi per il modulo R-Net che monitora e coordina il mezzo finale. Il progetto è stato realizzato in collaborazione con il team di neurorobotica dello IAS-Lab, il laboratorio di Sistemi Autonomi Intelligenti dell'Università di Padova.

2
Materiali e metodi

In questo capitolo verranno presentati gli elementi hardware e software usati per il sistema di controllo. Per rendere maggiormente modulare il sistema, le varie componenti sono state progettate per lavorare su livelli indipendenti, i quali possono comunicare reciprocamente senza dover conoscere il funzionamento della struttura rimanente.

2.1 La sedia a rotelle

La componente su cui si basa il progetto è la sedia a rotelle (Figure 2.1). Essa è formata da un telaio resistente con quattro ruote adatte anche a terreni non regolari, azionate da due motori, a trazione anteriore, dotati di freno di stazionamento; la seduta, lo schienale, le ginocchiere e il poggiatesta sono regolabili e permettono l'adattamento della sedia all'utilizzatore [15].

Il sistema di controllo utilizzato dalla sedia è R-net, compatibile con differenti moduli di alimentazione, di joystick o di seduta intelligente. I moduli possono essere combinati in modo flessibile per garantire la massima performance ad un costo limitato [16].

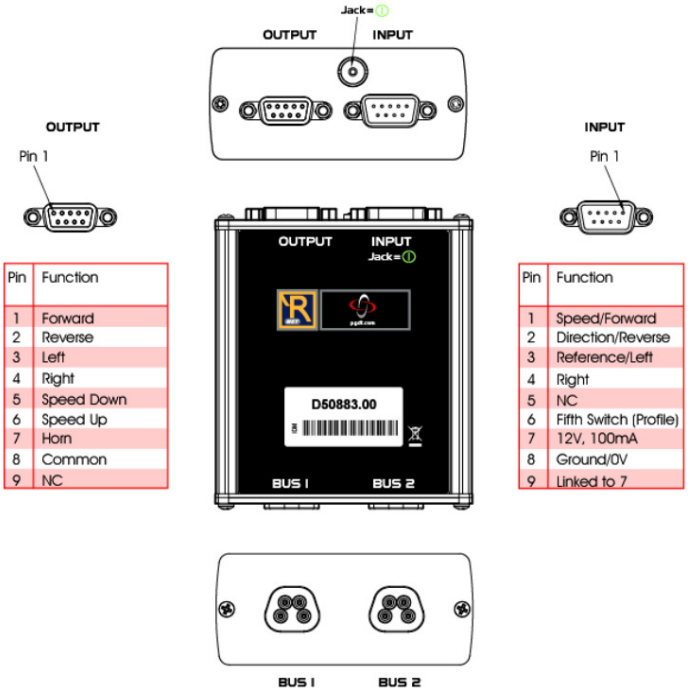
La sedia ha sei profili di utilizzo, differenti per velocità massima consentita nelle quattro direzioni, indicate come percentuale della velocità massima totale a cui può andare fisicamente il dispositivo(Table 3.4).

L'interfaccia tra il dispositivo di controllo integrato nella sedia motorizzata e l'Arduino su cui è stato implementato il programma è costituita dal modulo R-Net input/output, configurabile tramite switch interno al dispositivo stesso [17]. Può lavorare con segnali digitali o proporzionali grazie a i connettori D a nove poli, oltre ad avere una porta jack che permette di collegare un interruttore on/off esterno

Figure 2.1: La sedia a rotelle dello IAS-Lab

Table 2.1: Valori di percentuale della velocità massima del motore della sedia a cui sono impostati i profili

	Pr1	Pr2	Pr3	Pr4	Pr5	Pr6
Forward	25	40	60	100	100	80
Reverse	12	15	20	25	30	30
Turn	12	12	20	20	20	20



Pin	Function
1	Forward
2	Reverse
3	Left
4	Right
5	Speed Down
6	Speed Up
7	Horn
8	Common
9	NC

Pin	Function
1	Speed/Forward
2	Direction/Reverse
3	Reference/Left
4	Right
5	NC
6	Fifth Switch (Profile)
7	12V, 100mA
8	Ground/0V
9	Linked to 7

Figure 2.2: Le porte e lo schema dei pin del modulo R-NET Input/Output

(Figure 2.2) [17]; supporta inoltre sia il controllo 4-way che quello diagonale (Figure 2.3) [18]. Il primo tipo di controllo si riferisce a un sistema che consente di muoversi in quattro direzioni principali: avanti, indietro, destra e sinistra; il secondo si riferisce all'abilità di spostarsi in direzioni intermedie tra le quattro direzioni principali, offrendo una maggiore flessibilità nei movimenti rispetto al controllo 4-way. Il modulo R-Net è stato configurato in modalità input con segnali d'ingresso di tipo digitale, in modo tale da essere compatibile con i segnali pulse-width modulation inviati dalla scheda Arduino. La configurazione è stata effettuata grazie ai software specifici R-Net OEM Generic e R-Net Configurator rilasciati da PG Drives Technology. I pin del dispositivo hanno una funzione diversa a seconda dell'impostazione (Figure 2.4). I primi quattro pin ricevono gli impulsi relativi ai comandi per il movimento nelle direzioni avanti, indietro, sinistra e destra. Il pin 8 è l'ingresso di riferimento a zero volt, connesso all'uscita ground (GND) dell'Arduino. Tutti gli altri pin non sono stati utilizzati nel collegamento .

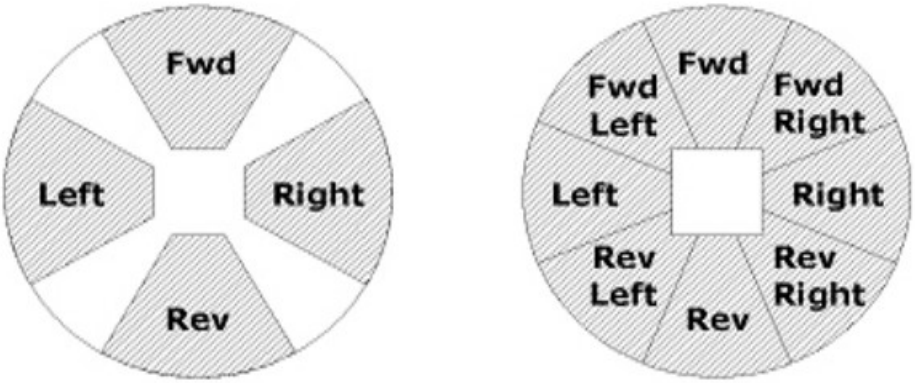
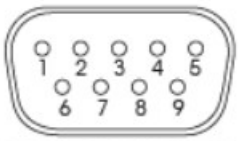


Figure 2.3: I due tipi di controllo: 4-way (sinistra) e diagonale (destra)



VIEWED FROM BELOW

Pin	Analog Function	Digital Function
1	Joystick Speed	Forward
2	Joystick Direction	Reverse
3	Joystick Reference	Left
4	-	Right
5	NC	NC
6	-	Fifth Switch
7	12V, 100mA	12V, 100mA
8	Joystick Ground	0V
9	Connected to 7	Connected to 7

Figure 2.4: Funzione dei pin di ingresso in base alla configurazione del dispositivo

2.2 Ambiente ROS

ROS è un framework di sviluppo open source per applicazioni robotiche [19], creato negli ultimi anni 2000 alla Stanford University, in collaborazione con Willow Garage [20]. ROS fornisce una comunicazione strutturata di alto livello tra i diversi cluster del sistema robotico, come ad esempio un computer, un braccio robotico o un insieme di sensori, ognuno rappresentato da un nodo [21]; comprende inoltre strumenti e librerie, organizzati in pacchetti, volti alla creazione di sistemi dal design modulare e distribuibile [22].

2.2.1 Nomenclatura in ROS

L'architettura ROS è formata da più **nodi**, le unità fondamentali di esecuzione, processi indipendenti che eseguono attività specifiche all'interno del sistema robotico [23]. L'organizzazione dei nodi è rappresentata secondo un grafo e i nodi-moduli comunicano usando un flusso di topic o di servizi Remote Procedure Call (RPC) e un **Parameter Server** [23], un dizionario dei parametri comuni utilizzati nel processo, accessibile tramite APIs [24]. Il Parameter Server è fornito dal **Master**, il quale, oltre a gestire l'archiviazione delle variabili che costituiscono il sistema, nomina e registra i vari nodi del grafo e ne traccia l'attività di publish/subscribe a topic e servizi, in modo da consentire agli stessi di individuarsi reciprocamente e comunicare con una modalità peer-to-peer [25].
La comunicazione consiste nello scambio di **messaggi**, strutture dati con campi definiti [26]. Sono supportati i tipi standard (interi, floating point, booleani, ecc.), ma anche array e costanti oppure composizione e annidamento di altri messaggi o array di messaggi [21]. Il canale di comunicazione e scambio di messaggi relativi a uno specifico argomento è chiamato **topic**. I nodi che sono interessati a un certo tipo di dati si iscrivono al topic corrispondente; una volta che un nodo pubblica dati su quel topic, tutti i nodi sottoscritti riceveranno automaticamente il messaggio passato, senza essere consapevoli degli altri moduli che operano sullo stesso canale [21] (Figure 2.5).

I sistemi distribuiti spesso necessitano interazioni di tipo richiesta/risposta, definite da coppie di messaggi, il primo appunto per la richiesta e il secondo che contiene l'esito, e gestite tramite **servizi** con un nome specifico [27].

2.2.2 Caratteristiche dell'ecosistema ROS

L'ecosistema ROS ha quattro aspetti chiave [19]:

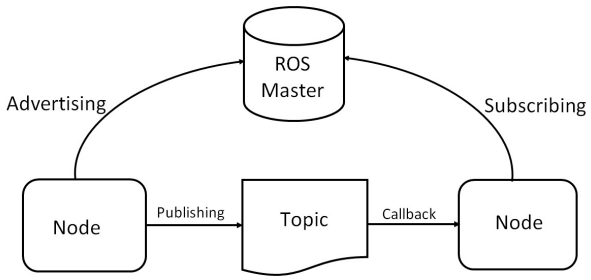


Figure 2.5: Schema di interazione dei nodi in ROS

1. **Comunicazione**: la categorizzazione dei nodi in publisher-subscriber consente una comunicazione flessibile e asincrona tra le componenti del sistema.

2. **Architettura**: la suddivisione in nodi consente di separare le funzionalità del robot in moduli indipendenti e interconnessi, favorendo la modularità e la riusabilità del codice.

3. **Librerie e strumenti**: ROS fornisce una vasta gamma di librerie e strumenti per supportare lo sviluppo robotico. Questi includono librerie per la manipolazione di dati geometrici, la navigazione, la percezione, la pianificazione del movimento e altro ancora [19]. Inoltre, ROS offre strumenti per la visualizzazione dei dati, la registrazione delle attività dei robot e l'analisi del comportamento.

4. **Community**: è presente una comunità di sviluppatori molto attiva e numerosa. Ciò significa che ci sono molti pacchetti e moduli disponibili che possono essere utilizzati direttamente o personalizzati per le esigenze specifiche di un robot. Inoltre, esistono forum di discussione, documentazione e risorse online che supportano gli sviluppatori nella risoluzione di problemi e lo scambio di conoscenze.

2.3 Modulo Arduino

Il modulo R-Net I/O è direttamente collegato al processore Arduino. Arduino è una piattaforma open-source basata su un microcontrollore, progettata per agevolare lo sviluppo di progetti elettronici interattivi[28]. Gli utenti possono facilmente creare e caricare codice in linguaggio Arduino Sketch sulla scheda. Questa può essere collegata a un computer tramite una porta seriale o USB per la programmazione e la comunicazione con il mondo esterno[29].

2.3.1 Arduino Hardware

La scheda fisica è composta da molti elementi, i principali sono i seguenti[29]:

- Microcontrollore: riceve e invia informazioni o comandi al circuito corrispondente.
- Porta USB: viene utilizzata per caricare il programma sul microcontrollore e dispone di un'alimentazione regolata di 5 volt che alimenta anche la scheda Arduino.
- Pulsante di reset: riavvia il microcontrollore, facendo ripartire il programma al suo interno.
- Pin analogici: una serie di pin che possono lavorare con segnali analogici, con un range di voltaggio da 0V a 5V.
- Pin digitali: Lavorano con i valori binari LOW (0) e HIGH (1), rispettivamente 0V e 5V. Il segnale viene discretizzato e mappato in valori interi da 0 a 255.
- Pin di Ground digitali e analogici: sono utilizzati come riferimento di tensione comune. Questi pin sono collegati al polo negativo dell'alimentazione elettrica e fungono da punto di riferimento a tensione zero per tutti i componenti elettronici nel circuito. Rappresentano inoltre il percorso di ritorno per la corrente elettrica, chiudendo il circuito e consentendo il flusso di energia.

Nella realizzazione del progetto, abbiamo utilizzato dei pin digitali della scheda, i quali hanno la capacità di funzionare secondo la modalità Pulse-Width Modulation (PWM). Il PWM è una tecnica utilizzata per simulare un segnale continuo utilizzando un segnale discreto quadrato, caratterizzato dai soli due valori HIGH e LOW, controllandone l'intensità aumentando il suo "duty-cycle", ovvero la durata dell'impulso[30].

2.3.2 Arduino Software

L'Arduino IDE (Integrated Development Environment) è composto da editor e compilatore, così da poter scrivere il codice che dichiara le istruzioni che informano l'hardware delle azioni da svolgere e come attuarle[29], lo compila e lo carica sulla scheda[28]. Il linguaggio utilizzato per gli *sketch*, così sono chiamati i programmi specifici di Arduino, è una versione semplificata del C++, di cui ne condivide la sintassi[31]. Il codice in tutti gli sketch deve contenere due funzioni:

1. **setup()**: include le istruzioni per impostare le condizioni iniziali delle variabili del programma, della scheda e dei suoi pin. Viene chiamata una sola volta, dopo aver lanciato il programma[29].
2. **loop()**: è la routine che esegue il codice al suo interno ripetutamente, fino a che non è raggiunta una condizione di stop interna o esterna al programma.

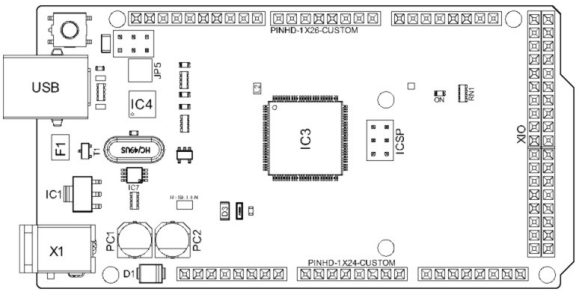


Figure 2.6: Topologia della scheda Arduino Mega 2560

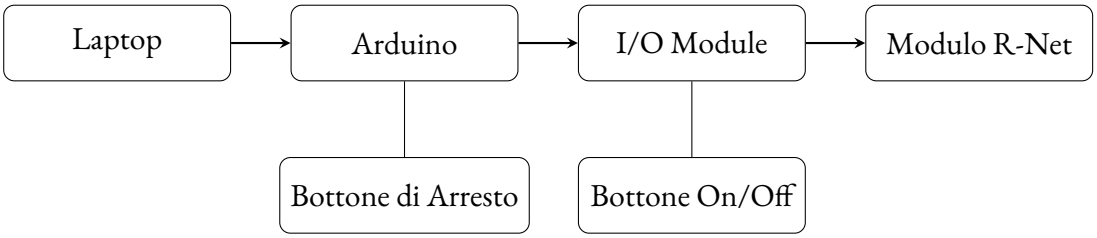
Prima della funzione **setup()** è necessario inizializzare le variabili globali usate all'interno del programma o importare gli eventuali pacchetti che servono per le computazioni[31].

2.3.3 Arduino Mega 2560 Rev3

Il modello della scheda Arduino usata per il progetto è Arduino Mega 2560 (Figure 2.6). Dispone di 54 ingressi/uscite digitali (di cui 14 possono essere utilizzate come uscite PWM), 4 porte seriali UART, 16 ingressi analogici, una connessione USB, un jack di alimentazione e un pulsante di reset[32]. Per connettere la scheda al computer si utilizza un cavo USB. Ha una tensione di funzionamento di 5V, ovvero la tensione massima che possono avere i segnali di input e di output[28]. La tensione di alimentazione deve essere compresa tra i 7V e i 12V. Il microcontrollore opera ad una frequenza di clock di 16MHz.

2.4 Interfaccia hardware sedia a rotelle e laptop

Il collegamento fisico tra il modulo di controllo e il sistema principale della sedia a rotelle ha la seguente struttura:

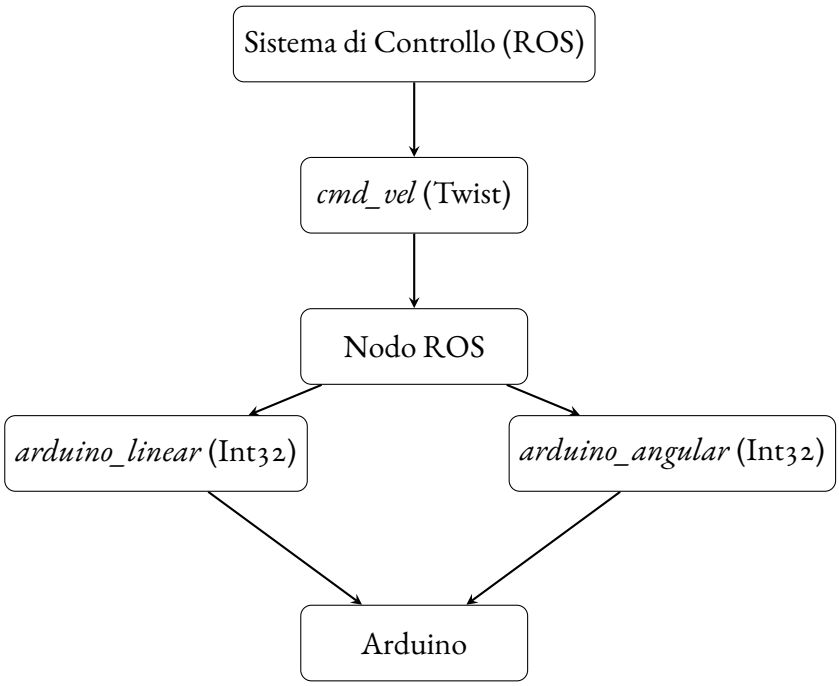


Il computer ha in memoria le cartelle e i file che servono per gestire la comunicazione con gli altri nodi ROS, fornisce potenza all'Arduino e vi carica il codice. Computer e Arduino sono collegati tramite cavo USB-A - USB-B.
Per avere un meccanismo facilmente accessibile di arresto forzato del motore, abbiamo aggiunto un pulsante twist-release, un dispositivo che, se premuto, crea un cortocircuito che segnala alla scheda Arduino la volontà di annullare i segnali di output e fermare così il movimento della sedia. La connessione, e quindi il moto del veicolo, viene ripresa quando il pulsante è rilasciato tramite torsione dello stesso sul proprio asse principale.
Il microprocessore invia i segnali output tramite i vari pin della scheda, in cui sono inseriti i cavi che trasmettono il messaggio al modulo R-Net I/O. Alla porta jack di questo dispositivo é stato collegato un bottone di spegnimento forzato manuale, che accende e spegne il sistema della sedia tramite cortocircuito.
Infine, il modulo R-Net input/output regola la comunicazione con il dispositivo di controllo integrato nella sedia. I cavi di collegamento sono stati creati su misura in laboratorio, utilizzando pinze per terminali DuPont e strumenti di saldatura; l'unità di protezione della scheda Arduino è stata invece modellata tramite software OpenSCAD e prodotta grazie ad una stampante 3D.

2.5 Interfaccia software sedia a rotelle e laptop

In questa sezione verrà presentato il codice implementato in laboratorio, poi testato sulla sedia. Il codice è diviso in due file: il primo gestisce il nodo ROS caricato su computer, il quale serve per convertire i comandi di input in numeri interi compatibili con i segnali PWM; il secondo è il programma specifico per Arduino, che comunica direttamente con il sistema che controlla il motore.
Per non aggiungere complessità al circuito Arduino - Modulo I/O abbiamo optato per la comunicazione digitale. Questa converte la presenza di potenziale elettrico nel pin di ingresso in comandi per il movimento della sedia. Per modulare la velocità è necessario quindi gestire la frequenza con cui generare il segnale HIGH da Arduino per presentarlo al modulo R-Net. Per fare ciò, abbiamo deciso di usare i pin PWM di output della scheda Arduino, potendone facilmente gestire il ciclo HIGH-LOW di funzionamento.
Volendo gestire il codice in modo che sia il più indipendente possibile abbiamo fatto in modo che la scheda Arduino accetti i valori corrispondenti alla frequenza PWM nel range 0-255 e che sia presente nella trasmissione del comando dal computer alla sedia un nodo interno al pc, antecedente al microcontrollore, che ha come scopo convertire il messaggio ROS relativo alla velocità desiderata in frequenza PWM. I valori specifici usati nel codice sono stati trovati sperimentalmente e sono modellati

appositamente per la sedia a rotelle del laboratorio. La modalità con cui abbiamo trovato i parametri è descritta nel Capitolo 3.
Qui sotto è riportato lo schema del flusso di messaggi di cui è composto il sistema:



2.5.1 Codice Nodo ROS

Il codice C++ sviluppato è il nodo che supervisiona l'elaborazione di dati di velocità per un la sedia motorizzata all'interno del framework ROS.
L'obiettivo primario di questo codice è la trasformazione dei messaggi di ingresso, acquisiti tramite network ROS, in segnali di uscita che saranno passati al microprocessore Arduino. L'input è costituito da una struttura di tipo Twist, che rappresenta le componenti spaziali delle velocità lineare e angolare a cui si deve muovere la sedia. Questi messaggi devono essere mappati in valori interi secondo una funzione ben precisa. Il risultato di tale operazione è il valore, positivo o negativo, del segnale pulse-width modulation che la scheda Arduino dovrà leggere dal topic ROS corrispondente e, successivamente, dividerlo nelle quattro diverse direzioni ed inviare al modulo R-Net I/O.

1. Inizialmente, vengono inclusi gli header delle librerie ROS, fornendo l'infrastruttura necessaria per comunicare all'interno dell'ecosistema ROS:

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <std_msgs/Int32.h>
#include <math.h>
```

<ros.h> La libreria principale del framework ROS, utilizzata per comunicare con il sistema ROS più ampio e consentire ai vari dispositivi di creare una rete di scambio dati, comandi e informazioni, il tutto in tempo reale.

<geometry_msgs/Twist.h> Questo pacchetto ha come funzione la rappresentazione di informazioni geometriche, come posizioni, orientamenti, velocità e accelerazioni, all'interno dell'ecosistema ROS. La libreria definisce una struttura di dati chiamata Twist, capace di memorizzare informazioni riguardanti la velocità di movimento spaziale. In particolare, la struttura include due vettori tridimensionali: il primo per la velocità lineare e il secondo per la velocità angolare, rispettivamente lungo e attorno agli assi x, y e z.

<std_msgs/Int32.h> La libreria contiene definizioni standard per vari tipi di messaggi, nello specifico la struttura 'Int32', utilizzata per rappresentare un numero intero a 32 bit, che è un tipo di dato numerico utilizzato comunemente nella programmazione. La struttura ha un unico campo 'data', la variabile che effettivamente andrà a memorizzare il messaggio.

math.h Fornisce un insieme di funzioni matematiche per eseguire operazioni comuni, come calcoli trigonometrici, esponenziali, logaritmiche e altre operazioni matematiche avanzate.

2. Le variabili statiche e costanti che seguono sono essenziali per il funzionamento dell'algoritmo, delineando i limiti del segnale di output oltre che le variabili che servono per memorizzare il messaggio di ingresso e il risultato della mappatura nel dominio degli interi.

```
float maxVel=1;
int minVal=255;

std_msgs::Int32 x_velocity;
std_msgs::Int32 y_velocity;

geometry_msgs::Twist mapped_velocity;
float linear_vel = 0;
float angular_vel = 0;
```

3. Le funzioni di mappatura **map_values_linear()** e **map_values_angular()** sono la parte fondamentale del codice del nodo, consentono di convertire il comando di ingresso contenente

la velocità desiderata in valori interi compatibili con i segnali PWM di Arduino, garantendo un'interfaccia coerente con il controllo fisico della sedia. In base alla direzione del movimento voluto, viene per prima cosa osservato se la velocità è contenuta nel range del profilo e, successivamente, viene mappata secondo le funzioni descritte nel prossimo capitolo, assieme al metodo con cui abbiamo ricavato la mappa.

```
int map_values_linear(float x){
    if(!neg_lin_vel){ //avanti
        if(x>=0.56) return maxVel;
        else if(x==0) return minVal;

        if(x <= 0.38){
            return round(36.6667 - 0.235702*sqrt(600*x+11));
        else
            return round(30.25 + 0.25*sqrt(449-800*x));
    }
    else{ //indietro
        if(x>=0.27) return maxVel;
        else if(x==0) return minVal;

        if(x <= 0.13)
            return round(17.5353 + 0.000137255*sqrt(2.7246*
                pow(10,10) - 7.28571*pow(10,10)*x));
        else
            return round(29.4999 + 0.000177087*sqrt(2.49005*
                pow(10,9) - 8.92858*pow(10,9)*x));
    }
}

int map_values_angular(float x){
    if(!neg_ang_vel){ //sinistra
        if(x>=0.806) return maxVel;
        else if(x==0) return minVal;

        if(x <= 0.73)
            return round(41.8148 - 0.0000734499*sqrt(2.72294*
                pow(10,10)*x + 9.99985*pow(10,8)));
 else
 return round(27.6 + 0.0000347833*sqrt(1.03944*
                pow(10,11) - 1.28571*pow(10,11)*x));
    }
```

```
    else{
        if(x>=0.74) return maxVel;
        else if(x==0) return minVal;

        if(x <= 0.34)
            return round(44.5203 - 0.000216842*sqrt(7.29168*
                pow(10,9)*x + 9.03179*pow(10,8)));
        else
 return round(27.9062 + 0.0000978282*sqrt(3.41492*
                pow(10,9) - 4.57142*pow(10,9)*x));
    }
}
```

4. La funzione **callback()** è il cuore della comunicazione col sistema ROS che gestisce i comandi per la sedia motorizzata. Questa funzione è richiamata automaticamente quando un messaggio è pubblicato nel topic **"cmd_vel"**. Gestisce i comandi di velocità lineare e angolare e calcola i valori di velocità adatti per il dispositivo Arduino chiamando i metodi di mappatura. La direzione in cui il comando muove la sedia viene salvata grazie alle variabili booleane **neg_lin_vel** e **neg_ang_vel**.

```
void callback(const geometry_msgs::Twist& velocity){
    linear_vel = velocity.linear.x;
    angular_vel = velocity.angular.z;

    if(linear_vel < 0)
        neg_lin_vel = true;
    else
        neg_lin_vel = false;

    if(angular_vel < 0)
        neg_ang_vel = true;
    else
        neg_ang_vel = false;

    x = map_values_linear(abs(linear_vel));
y = map_values_angular(abs(angular_vel));

    if(neg_lin_vel)
        x *=-1;
    if(neg_ang_vel)
```

```
        y *= -1;

    x_velocity.data = x ;
    y_velocity.data = y;
}
```

5. La funzione **main()** avvia l'esecuzione del nodo ROS. Inizializza il nodo, stabilisce i collegamenti con i topic e avvia un ciclo while() che rende continua la comunicazione con i nodi della rete del sistema. Durante ciascuna iterazione del ciclo, le velocità elaborate vengono pubblicate sui topic **"arduino_linear"** e **"arduino_angular"**, consentendo ad Arduino, e quindi alla sedia, di reagire in tempo reale alle richieste di movimento.

```
int main(int argc, char** argv){
    ros::init(argc,argv,"profile_mapping");
    ros::NodeHandle nh;
    ros::Subscriber sub = nh.subscribe("cmd_vel", 10,
            callback);

    ros::Publisher pub_l = nh.advertise<std_msgs::Int32>(
            "arduino_linear", 1);
    ros::Publisher pub_a = nh.advertise<std_msgs::Int32>(
            "arduino_angular", 1);

    x_velocity.data=minVal;
    y_velocity.data=minVal;

    ros::Rate rate(50);

    while(ros::ok()){
        pub_l.publish(x_velocity);
        pub_a.publish(y_velocity);

        ros::spinOnce();
        rate.sleep();
    }
    return 0;
}
```

2.5.2 Codice Arduino

Il codice Arduino implementa un nodo di controllo per un sistema di movimento bidirezionale controllato tramite PWM (Pulse Width Modulation) utilizzando il framework di comunicazione ROS. Questo codice è progettato per comunicare con un nodo ROS sul computer a cui è collegato.
Ecco una spiegazione dettagliata delle diverse parti del codice:

1. Inclusione delle librerie ROS:

```
#include <ros.h>
#include <std_msgs/Int32.h>
```

   Le prime righe di codice sono le direttive che servono per includere le librerie **ros.h** e **<std_msgs/Int32.h>** necessarie per utilizzare ROS con l'Arduino, descritte in precedenza.

2. Inizializzazione del nodo ROS:

```
ros::NodeHandle  nh;
```

   Viene istanziato un oggetto **NodeHandle** che rappresenta il nodo ROS del microprocessore.

3. Definizione dei pin e delle variabili:

```
const int forwardPin = 4;
const int reversePin = 5;
const int leftPin = 6;
const int rightPin = 7;
```

   I pin digitali 4, 5, 6 e 7 sono assegnati a quattro variabili, utilizzate per generare il segnale PWM che controlla il movimento della sedia nelle quattro direzioni. I seguenti pin forceStopPin e removeStopPin sono utilizzati per gestire l'arresto di emergenza e l'annullamento dello stesso.

4. Gestione dell'arresto forzato della sedia motorizzata:

```
const int forceStopPin = 2;
const int removeStopPin = 3;
volatile bool forcedStop = false;
```

```
...

void checkForceStop(){
    forcedStop = true;
}
void removeForceStop(){
    forcedStop = false;
}
```

L'arresto forzato è ottenuto annullando i segnali di output della scheda Arduino verso il modulo R-Net I/O. Viene definita una variabile volatile forcedStop che indica se è stato attivato lo stato di arresto. Le funzioni checkForceStop e removeForceStop vengono richiamate rispettivamente quando il pulsante di arresto di emergenza viene premuto o rilasciato.

5. Definizione di variabili di velocità e direzione:

```
int pwm_linear = minVal;
int pwm_angular = minVal;

int xVoltage = minVal;
int yVoltage = minVal;

bool forward;
bool left;
```

Le variabili **pwm_linear** e **pwm_angular** memorizzano i valori di velocità lineare e angolare in termini di segnali PWM. Le variabili **xVoltage** e **yVoltage** memorizzano le tensioni per il movimento in avanti/indietro e sinistra/destra. Le variabili booleane forward e left indicano la direzione del movimento.

6. Callback per i messaggi di velocità lineare e angolare: Queste callback vengono chiamate quando i messaggi con i valori di velocità lineare e angolare vengono ricevuti da ROS. I valori vengono memorizzati nelle variabili **pwm_linear** e **pwm_angular**.

```
void callback_linear(const std_msgs::Int32& lin_msg){
    pwm_linear = lin_msg.data;

    if(pwm_linear >= 0)
        forward = true;
```

```
    else
        forward = false;

    xVoltage = abs(pwm_linear);
}

void callback_angular(const std_msgs::Int32& ang_msg){
    pwm_angular = ang_msg.data;

    if(pwm_angular >= 0)
        left = true;
    else
        left = false;

    yVoltage = abs(pwm_angular);
}

ros::Subscriber<std_msgs::Int32> sub_l("arduino_linear",
        callback_linear);
ros::Subscriber<std_msgs::Int32> sub_a("arduino_angular",
        callback_angular);
```

7. Funzione setup: Viene inizializzato il nodo ROS e vengono effettuate le configurazioni iniziali dei pin e delle variabili. La comunicazione seriale con il nodo ROS avviene alla velocità di 115200 bps.

```
void setup(){
    pinMode(forwardPin,OUTPUT);
    pinMode(reversePin,OUTPUT);
    pinMode(leftPin,OUTPUT);
    pinMode(rightPin,OUTPUT);

    pinMode(forceStopPin, INPUT_PULLUP);
    pinMode(removeStopPin, INPUT_PULLUP);

    forcedStop = false;

    attachInterrupt(digitalPinToInterrupt(forceStopPin),
            checkForceStop, FALLING);
    attachInterrupt(digitalPinToInterrupt(removeStopPin),
```

```
            removeForceStop, FALLING);

    forward = true;
    left = true;

    xVoltage = minVal;
    yVoltage = minVal;

    nh.getHardware()->setBaud(115200);
    nh.initNode();
    nh.subscribe(sub_l);
    nh.subscribe(sub_a);

    analogWrite(forwardPin, 255);
    analogWrite(leftPin, 255);
    analogWrite(reversePin, 255);
    analogWrite(rightPin, 255);
}
```

8. Loop principale: Nel loop principale, se lo stato di arresto forzato è attivo, tutti i segnali PWM vengono impostati al valore minimo per annullare il movimento della sedia. Altrimenti, i segnali PWM vengono impostati a seconda della direzione e della velocità memorizzate nelle variabili dopo che sono state chiamate in modo automatico le callback.

```
void loop(){
    if(forcedStop){
        analogWrite(forwardPin, minVal);
        analogWrite(leftPin, minVal);
        analogWrite(reversePin, minVal);
        analogWrite(rightPin, minVal);
    }
    else{
        if(forward){
            analogWrite(reversePin, minVal);
            analogWrite(forwardPin, xVoltage);
        }
        else {
            analogWrite(forwardPin, minVal);
            analogWrite(reversePin, xVoltage);
        }
```

```
        if(left){
            analogWrite(rightPin, minVal);
            analogWrite(leftPin, yVoltage);
        }
        else{
            analogWrite(leftPin, minVal);
            analogWrite(rightPin, yVoltage);
        }
    }
    delay(10);
    nh.spinOnce();
}
```

3
Risultati e validazione

In questo capitolo saranno presentati i risultati ottenuti dalla progettazione e realizzazione del sistema di controllo descritto nel capitolo precedente. Verranno esaminati i segnali di output generati dalla scheda Arduino durante il funzionamento del sistema e verranno analizzate le performance della sedia motorizzata in termini di velocità. I risultati saranno presentati attraverso tabelle e grafici dettagliati per una migliore comprensione e valutazione delle prestazioni del sistema.

3.1 Mappatura dei segnali di Input del nodo ROS

Per implementare la comunicazione Arduino-RNET viene necessario conoscere il protocollo di comunicazione verso il bridge. Il protocollo viene espresso nel manuale fornito dalla casa di produzione, ma presenta alcune lacune essendo esso indipendente dalla macchina su cui viene installato.
In particolare non viene presentata nessuna tabella di conversione tra segnale in entrata e velocità della sedia (essendo che questa dipende da fattori come il modello della sedia, il profilo inserito, lo stato dei pneumatici ecc). Per ovviare a questa mancanza abbiamo optato per ricreare la tabella di conversione tramite una tecnica di ingegneria inversa, i cui passaggi e i risultati ottenuti sono trattati nei seguenti sottocapitoli.
Il nodo ROS descritto nel capitolo precedente riceve in input i comandi relativi alla velocità desiderata a cui la sedia deve muoversi.
Il programma, non appena legge i messaggi pubblicati nel topic, controlla la positività dei valori e chiama i due metodi di mappatura. A seconda della direzione a cui il comando di input fa riferimento,

verrà usata una funzione diversa per calcolare il valore intero che verrà trasferito alla scheda Arduino. Le funzioni sono state sviluppate, per il momento, solamente per il primo profilo.

3.1.1 Approccio di ricerca delle funzioni di mappatura

I valori inseriti nelle quattro funzioni descritte sono strettamente specifici per la sedia a rotelle attorno a cui cui è stato progettato il codice, ciò significa che il programma potrebbe non avere la stessa precisione nel trasformare i comandi di input se usato per altri veicoli dello stesso tipo. La ragione dietro alla necessità di specificità nella creazione dei metodi di mappatura è la mancanza di informazioni e documentazione rispetto al modulo I/O e al dispositivo di controllo della sedia a rotelle.

Per prima cosa abbiamo scollegato virtualmente il nodo ROS dal programma Arduino e abbiamo fornito in input a quest'ultimo, uno ad uno tramite messaggi scritti sul topic, i numeri interi necessari per far muovere la sedia. Da subito abbiamo notato come gli interi intorno e superiori a 41 non portassero ad alcun movimento da parte dei motori, indipendentemente dalla direzione specificata.
Successivamente abbiamo ricostruito la mappa delle velocità dai dati ricavati dall'odometria montata sulla sedia, provando uno ad uno come input i valori interi in ordine decrescente, fino a trovare il numero per il quale la sedia raggiunge il limite massimo di velocità consentita dal profilo. Ogni direzione differisce dalle altre nel range di numeri interi su cui opera e di velocità associate ad essi. Le mappe cosi create sono diverse per le varie direzioni, creando la necessita di funzioni custom per ogni direzione.
Dopo aver ottenuto le tabelle di valori riportate in seguito, abbiamo prima di tutto diviso ogni set di dati in due gruppi e calcolato per ognuno la funzione di interpolazione quadratica, in modo tale da avere una conversione più precisa da valore intero a valore di velocità trovato in laboratorio di quella che si otterrebbe calcolando la funzione su tutti i dati disponibili. Abbiamo poi identificato tutte le corrette funzioni inverse, permettendoci di avere una trasformazione accurata dalla velocità voluta all'intero che, se passato al modulo R-Net I/O, porta la sedia a quella stessa velocità. Con livelli di interpolazione più alti, la funzione sarebbe stata maggiormente verosimile ai dati raccolti ma avrebbe aumentato in modo non indifferente la complessità di calcolo della funzione inversa. Come ultima cosa abbiamo testato quanto fatto, verificando la correttezza delle funzioni sviluppate.

3.1.2 Avanti

La tabella per il primo profilo riguardante la velocità lineare nel verso frontale è la seguente:

Table 3.1: Mappa valori interi di Arduino - velocità associate, direzione lineare in avanti

Valore Arduino	Velocità sperimentale (m/s)
36	0
35	0.05
34	0.21
33	0.38
32	0.5
31	0.55
30	0.56

Le funzione di interpolazione calcolate sono, per i valori di velocità da 0 a 0.38 m/s:

$$f(x) = 36.6667 - 0.235702 \cdot \sqrt{600x + 11}$$

Per velocità superiori a 0.38m/s, fino a 0.56m/s:

$$f(x) = 30.25 + 0.25 \cdot \sqrt{449 - 800x}$$

Queste funzioni corrispondono al grafico Figure 3.1. La prima funzione è rappresentata da una linea tratteggiata, mentre la seconda da un segno continuo, in modo da differenziarle graficamente. Per la velocità nulla, il valore di output del nodo ROS è impostato di default a 255, mentre per velocità superiori al limite impostato dal profilo, ovvero 0.56m/s, l'output sarà 1.

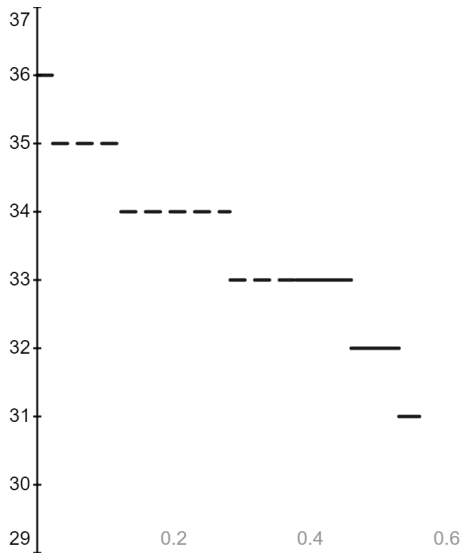


Figure 3.1: Mappa delle velocità - avanti

3.1.3 Indietro

La tabella per il primo profilo riguardante la velocità lineare in retromarcia è la seguente:

Table 3.2: Mappa valori interi di Arduino - velocità associate, direzione lineare indietro

Valore Arduino	Velocità sperimentale (m/s)
41	0
40	0.0009
39	0.05
38	0.066
37	0.09
36	0.13
35	0.17
34	0.21
33	0.23
32	0.26
31	0.27

Le funzione di interpolazione calcolate sono, per i valori di velocità da 0 a 0.13m/s:

$$f(x) = 17.5353 + 0.000137255 \cdot \sqrt{2.7246 \cdot 10^{10} - 7.28571 \cdot 10^{10} \cdot x}$$

Per velocità superiori a 0.13m/s, fino a 0.27m/s:

$$f(x) = 29.4999 + 0.000177087 \cdot \sqrt{2.49005 \cdot 10^{9} - 8.92858 \cdot 10^{9} \cdot x}$$

Queste funzioni corrispondono al grafico Figure 3.2. La prima funzione è rappresentata da una linea tratteggiata, mentre la seconda da un segno continuo, in modo da differenziarle graficamente. Per la velocità nulla, il valore di output del nodo ROS è impostato di default a 255, mentre per velocità superiori al limite impostato dal profilo, ovvero 0.27m/s, l'output sarà 1.

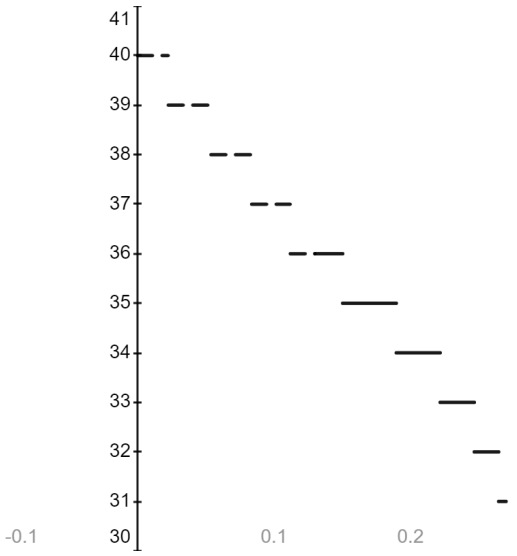


Figure 3.2: Mappa delle velocità - indietro

3.1.4 Sinistra

La tabella per il primo profilo riguardante la velocità angolare, girando verso sinistra, è la seguente:

Table 3.3: Mappa valori interi di Arduino - velocità associate, direzione angolare verso sinistra

Valore Arduino	Velocità sperimentale (m/s)
40	0
39	0.01
38	0.05
37	0.1
36	0.21
35	0.3
34	0.37
33	0.5
32	0.6
31	0.73
30	0.78
29	0.795
28	0.806

Le funzione di interpolazione calcolate sono, per i valori di velocità da 0 a 0.73m/s:

$$f(x) = 41.8148 - 0.0000734499 \cdot \sqrt{2.72294 \cdot 10^{10} \cdot x + 9.99985 \cdot 10^{8}}$$

Per velocità superiori a 0.73m/s, fino a 0.806m/s:

$$f(x) = 27.6 + 0.0000347833 \cdot \sqrt{1.03944 \cdot 10^{11} - 1.28571 \cdot 10^{11} \cdot x}$$

Queste funzioni corrispondono al grafico Figure 3.3. La prima funzione è rappresentata da una linea tratteggiata, mentre la seconda da un segno continuo, in modo da differenziarle graficamente. Per la velocità nulla, il valore di output del nodo ROS è impostato di default a 255, mentre per velocità superiori al limite impostato dal profilo, ovvero 0.806m/s, l'output sarà 1.

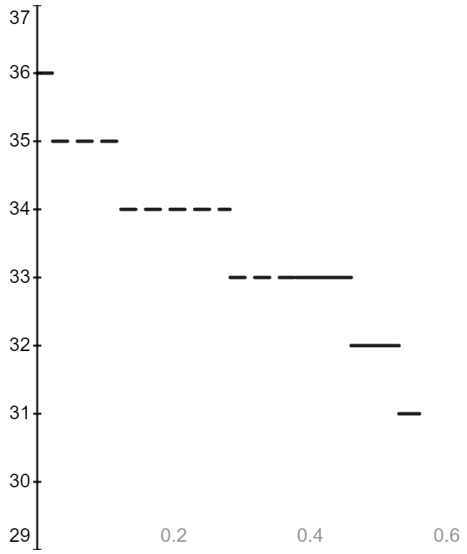


Figure 3.3: Mappa delle velocità - sinistra

3.1.5 Destra

La tabella per il primo profilo riguardante la velocità angolare, girando verso destra, è la seguente:

Table 3.4: Mappa valori interi di Arduino - velocità associate, direzione angolare verso destra

Valore Arduino	Velocità sperimentale (m/s)
38	0
37	0.04
36	0.085
35	0.15
34	0.2
33	0.25
32	0.34
31	0.51
30	0.66
29	0.7
28	0.74

Le funzione di interpolazione calcolate sono, per i valori di velocità da 0 a 0.34m/s:

$$f(x) = 44.5203 - 0.000216842 \cdot \sqrt{7.29168 \cdot 10^9 \cdot x + 9.03179 \cdot 10^8}$$

Per velocità superiori a 0.34m/s, fino a 0.74m/s:

$$f(x) = 27.9062 + 0.0000978282 \cdot \sqrt{3.41492 \cdot 10^9 - 4.57142 \cdot 10^9 \cdot x}$$

Queste funzioni corrispondono al grafico Figure 3.4. La prima funzione è rappresentata da una linea tratteggiata, mentre la seconda da un segno continuo, in modo da differenziarle graficamente. Per la velocità nulla, il valore di output del nodo ROS è impostato di default a 255, mentre per velocità superiori al limite impostato dal profilo, ovvero 0.74m/s, l'output sarà 1.

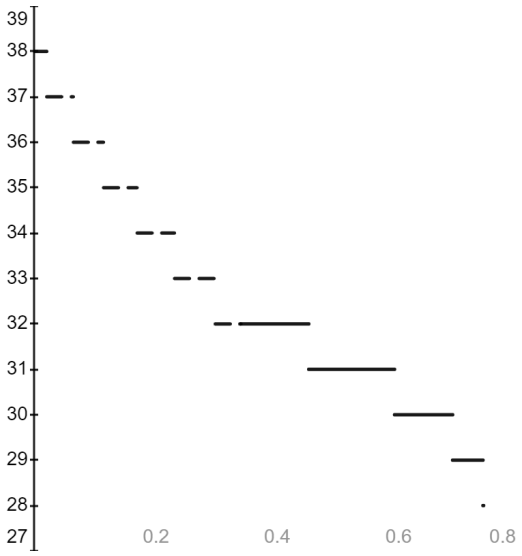
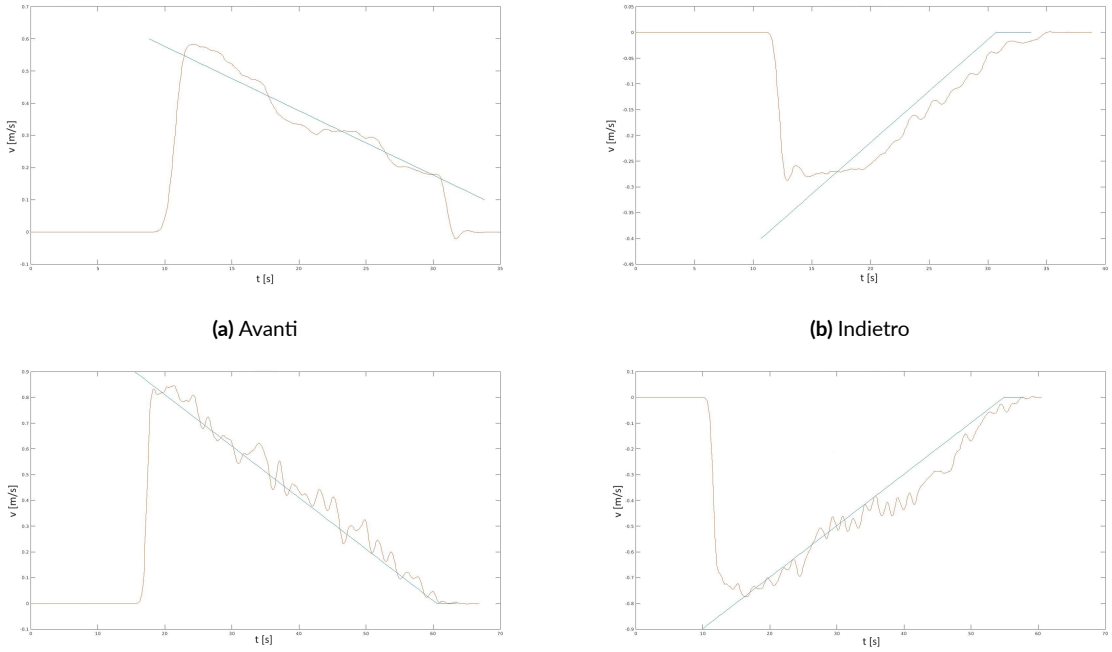


Figure 3.4: Mappa delle velocità - destra

3.2 Validazione della mappatura

La fase finale del progetto ha previsto la verifica delle funzioni sopra descritte.
Abbiamo creato un nodo ROS che ci ha permesso di pubblicare sul topic 'cmd_vel', da cui il codice di cui sopra ricava i valori di input, i comandi di velocità. Il valore pubblicato sul topic parte dalla velocità massima e decresce fino alla velocità nulla. L'incremento è di 0.1m/s ogni secondo, in modo tale da poter valutare l'accuratezza della mappa.
Abbiamo misurato poi tramite odometria la velocità effettiva a cui andava il veicolo e utilizzato i metodi di plot di matlab per confrontarla con il valore aspettato. Qui sotto sono riportati i grafici che descrivono i dati raccolti: la linea rossa rappresenta la velocità effettiva misurata tramite odometria, mentre quella blu è l'andamento del valore del comando di input.



(a) Avanti **(b)** Indietro

(c) Sinistra **(d)** Destra

Figure 3.5: Confronto tra la velocità di input e quella effettiva

Dai grafici si può notare un leggero ritardo nella prima fase del movimento, dovuto all'iniziale accelerazione della sedia per raggiungere la velocità imposta, ma anche il limite superiore alla velocità sia lineare che angolare, visibile soprattutto nel grafico 'Indietro' (Figure 3.5b). Oltre a questo, la velocità misurata con gli strumenti odometrici segue quella imposta dal comando di input. La misurazione del movimento in avanti è risultata difficoltosa a causa di uno spazio limitato in cui effettuare i test, oltre alla conformazione delle ruote della sedia, le quali spesso portavano a una modifica, per quanto leggera, della direzione durante il moto.

4
Conclusioni

In questa tesi, abbiamo presentato il design e lo sviluppo di un sistema di controllo per una sedia a rotelle motorizzata, utilizzando il framework ROS per permettere una connessione veloce e indipendente con gli altri dispositivi del sistema robotico e il microcontrollore Arduino per una maggiore modulatità del progetto.
Durante il corso dello sviluppo, abbiamo affrontato diverse sfide tecniche, tra cui la progettazione dell'hardware, la comunicazione tra le componenti del sistema e l'accuratezza della conversione dei comandi nelle velocità desiderate.
Abbiamo condotto test sperimentali per valutare le prestazioni del sistema, durante i quali esso si è dimostrato robusto e responsivo. In particolare, ci siamo concentrati sulla misurazione della velocità della sedia a rotelle utilizzando dati di odometria. I risultati hanno mostrato che il sistema è in grado di fornire una buona stima della velocità della sedia in un ambiente controllato. Tuttavia, è importante notare che questi test sono stati condotti solo per uno dei profili di utilizzo possibili, e ulteriori test sarebbero necessari per valutare il sistema in diverse condizioni, sviluppando di conseguenza le altre funzioni relative ai profili mancanti.

4.1 Lavori futuri

Ci sono ancora diverse sfide e opportunità per lavori futuri da considerare:

1. Estensione dei Profili di Utilizzo: Attualmente, il sistema è stato testato solo per uno specifico

profilo di utilizzo. Sarebbe importante estendere i test per coprire una gamma più ampia di situazioni e condizioni ambientali per garantire la sua robustezza.

2. Integrazione di Tecnologie Sensibili all'Ambiente: L'integrazione di dispositivi aggiuntivi, come telecamere o sensori di prossimità, potrebbe consentire una migliore percezione dell'ambiente circostante e una maggiore sicurezza nella navigazione.

3. Test tramite casco EEG: Per valutare appieno l'utilità del sistema per gli utenti finali, sarebbe opportuno condurre test di guida del veicolo tramite casco a segnali elettroencefalografici.

In definitiva, la ricerca in questo campo è destinata a proseguire, con l'obiettivo di rendere le tecnologie di assistenza sempre più accessibili ed efficaci, affinché tutti possano godere di una maggiore autonomia e indipendenza.

Bibliografia

[1] D. Pal, S. Palit, and A. Dey, "Brain computer interface: A review," in *Computational Advancement in Communication, Circuits and Systems*, M. Mitra, M. Nasipuri, and M. R. Kanjilal, Eds. Singapore: Springer Singapore, 2022, pp. 25–35.

[2] R. E. Cowan, B. J. Fregly, M. L. Boninger, L. Chan, M. M. Rodgers, and D. J. Reinkensmeyer, "Recent trends in assistive technology for mobility," *Journal of neuroengineering and rehabilitation*, vol. 9, no. 1, pp. 1–8, 2012.

[3] Special controls for electrical wheelchairs, mo-vis. [Online]. Available: https://www.mo-vis.com/

[4] Switched head arrays, switch-it. [Online]. Available: https://www.sunrisemedical.com/power-wheelchairs/switch-it-electronics/head-controls/switched-head-arrays

[5] J. Pineau, R. West, A. Atrash, J. Villemure, and F. Routhier, "On the feasibility of using a standardized test for evaluating a speech-controlled smart wheelchair," *International Journal of Intelligent Control and Systems*, vol. 16, no. 2, pp. 124–131, 2011.

[6] H. Seki, S. Kobayashi, Y. Kamiya, M. Hikizu, and H. Nomura, "Autonomous/semi-autonomous navigation system of a wheelchair by active ultrasonic beacons," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 1366–1371 vol.2.

[7] U. Borgolte, H. Hoyer, C. Bühler, H. Heck, and R. Hoelper, "Architectural concepts of a semi-autonomous wheelchair," *Journal of Intelligent and Robotic Systems*, vol. 22, pp. 233–253, 1998.

[8] Y. Rabhi, M. Mrabet, F. Fnaiech *et al.*, "Intelligent control wheelchair using a new visual joystick," *Journal of healthcare engineering*, vol. 2018, 2018.

[9] K. Tanaka, K. Matsunaga, and H. Wang, "Electroencephalogram-based control of an electric wheelchair," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 762–766, 2005.

[10] J. J. Shih, D. J. Krusienski, and J. R. Wolpaw, "Brain-computer interfaces in medicine," in *Mayo clinic proceedings*, vol. 87, no. 3. Elsevier, 2012, pp. 268–279.

[11] L. Tonin and J. d. R. Millán, "Noninvasive brain–machine interfaces for robotic devices," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 191–214, 2021. [Online]. Available: https://doi.org/10.1146/annurev-control-012720-093904

[12] L. Tonin, S. Perdikis, T. D. Kuzu, J. Pardo, B. Orset, K. Lee, M. Aach, T. A. Schildhauer, R. Martínez-Olivera, and J. del R. Millán, "Learning to control a bmi-driven wheelchair for people with severe tetraplegia," *iScience*, vol. 25, no. 12, p. 105418, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S258900422201690X

[13] A. Riehle and E. Vaadia, *Motor Cortex in Voluntary Movements: A Distributed System for Distributed Functions*, 1st ed. CRC Press, 2004, ch. Human Brain-Computer Interface, p. 36. [Online]. Available: https://doi.org/10.1201/9780203503584

[14] K. Matsuzawa and C. Ishii, "Control of an electric wheelchair with a brain-computer interface headset," in *2016 International Conference on Advanced Mechatronic Systems (ICAMechS)*. IEEE, 2016, pp. 504–509.

[15] Vassilli wheelchair 18-70-hi-lo-vario. [Online]. Available: https://vassilli.it/prodotto/18-70-hi-lo-vario/

[16] R-net control system. [Online]. Available: https://www.cw-industrial.com/en-gb/medical-mobility/rnet

[17] curtiss-wright r-net input/output module. [Online]. Available: https://www.cw-industrial.com/en-gb/products/mobility-vehicle-solutions/r-net/input-output-module

[18] R-net i/o module technical manual. [Online]. Available: https://www.cw-industrial.com/en-gb/products/mobility-vehicle-solutions/r-net/input-output-module

[19] ROS - robot operating system. [Online]. Available: https::://www.ros.org/blog/why-ros/

[20] A. Hussein, F. García, and C. Olaverri-Monreal, "ROS and unity based framework for intelligent vehicles control and simulation," in *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, 2018, pp. 1–6.

[21] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[22] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, and M. Carreras, "Rosplan: Planning in the robot operating system," in *Proceedings of the international conference on automated planning and scheduling*, vol. 25, 2015, pp. 333–341.

[23] ROS nodes. [Online]. Available: http://wiki.ros.org/Nodes

[24] Parameter server ROS. [Online]. Available: http://wiki.ros.org/Parameter%20Server

[25] ROS master. [Online]. Available: http://wiki.ros.org/Master

[26] ROS messages. [Online]. Available: http://wiki.ros.org/Messages

[27] ROS services. [Online]. Available: http://wiki.ros.org/Services

[28] F. Griso, "Progetto e realizzazione di un controllore di velocità per motorino," 2022.

[29] Y. A. Badamasi, "The working principle of an arduino," in *2014 11th International Conference on Electronics, Computer and Computation (ICECCO)*, 2014, pp. 1–4.

[30] B. Mokrytzki, "Pulse width modulated inverters for ac motor drives," *IEEE Transactions on Industry and General Applications*, no. 6, pp. 493–503, 1967.

[31] G. Organtini, *Scientific Arduino Programming*. Sapieza Univesità di Roma Rome, Italy, 2015.

[32] Arduino mega 2560. [Online]. Available: https://store.arduino.cc/products/arduino-mega-2560-rev3

Ringraziamenti

Desidero esprimere la mia profonda gratitudine a tutte le persone che hanno contribuito al completamento di questa tesi. Senza di loro, questo lavoro non sarebbe stato possibile.

Innanzitutto, voglio ringraziare il mio relatore, il Professore Luca Tonin, per avermi offerto la possibilità di lavorare a questo interessante progetto. Un ringraziamento speciale va anche al mio corelatore, il Dottor Piero Simonetto, per aver seguito da vicino le diverse fasi di questo lavoro. Il loro supporto e la loro guida sono stati fondamentali per la realizzazione di questa tesi.

Desidero ringraziare i miei genitori, mio fratello Marco e mia sorella Elena, i quali mi hanno permesso di intraprendere e portare a termine questo cammino e che mi hanno sempre aiutata ed incoraggiata. Grazie ai miei nonni e a tutta la mia famiglia, per avere creduto in me dall'inizio alla fine.

A Tommaso, che è stato la mia roccia in questo viaggio. Ha condiviso con me le gioie e le sfide, mi ha sostenuto quando ne avevo bisogno e ha condiviso la mia felicità in ogni successo. Grazie per essere sempre stato al mio fianco.

Grazie anche a Veronica, per essere stata la miglior guida e punto di riferimento che potessi avere nel mio percorso universitario.

Ringrazio inoltre tutti gli amici, sia quelli vecchi che quelli nuovi, oltre che i colleghi del gruppo Al.Organization, per aver alleggerito questi tre anni e avere sempre offerto conforto e gioia nei momenti difficili.

In conclusione, questa laurea è il risultato di un lungo percorso, e ogni singola persona qui menzionata ha contribuito in modo significativo al mio successo. Sono grata a ognuno di voi per aver reso questo percorso speciale ancora più memorabile. Il vostro affetto e il vostro supporto sono i veri premi di questo traguardo.

Grazie di cuore.

Anita