2212—

Università degli Studi di Trieste

Dipartimento di Ingegneria e Architettura

Corso di Studi in Ingegneria Industriale

Sperimentazione di un approccio data-driven per il parcheggio automatico di un robot mobile

Laureando:
Lorenzo MARZI

Relatore:
prof. Gianfranco FENU
Correlatore:
prof. Erica SALVATO
prof. Felice Andrea PELLEGRINO

Anno Accademico 2022-2023

Indice

Capitolo 1

Introduzione

Lo scopo principale di questa tesi è la validazione sperimentale di un approccio *data-driven* per l'esecuzione automatica di una manovra di parcheggio di un robot mobile.

In particolare, si vuole utilizzare la procedura proposta in [3], nel quale gli autori presentano un approccio di sintesi di controllori periodici stabilizzanti e variabile nel tempo a partire dai dati. La procedura richiede che il sistema da controllare sia lineare e che si conosca un numero n (dimensione dello spazio degli stati) di traiettorie ad anello aperto, in grado di portare lo stato in un opportuno intorno di 0 a partire da n condizioni iniziali.

In questa tesi si vuole utilizzare tale approccio di controllo *model-free* per sintetizzare un controllore che permetta al robot di eseguire la manovra di parcheggio automatico. I dati utilizzati per la sintesi del controllore sono la velocità e la posizione registrati durante una manovra di esempio precedentemente fornita da un utente esperto.

Il robot utilizzato in questo lavoro di tesi è il Thymio II, un robot educativo programmabile e controllabile a distanza.

Il lavoro sperimentale condotto può essere principalmente suddiviso in due parti: (i) la generazione delle n traiettorie, (ii) la sintesi e l'applicazione del controllore a partire da queste ultime.

Per la generazione delle n traiettorie, si è deciso di sfruttare le proprietà strutturali del robot e di implementare un algoritmo in grado di inseguire una manovra di esempio di parcheggio "disegnata". Durante l'esecuzione di tale manovra vengono raccolti i dati utili a descrivere l'operazione di parcheggio. In dettaglio, vengono acquisite le velocità dei motori delle ruote motrici e, a partire da queste, si utilizza un algoritmo di odometria per la ricostruzione passo passo della posizione e dell'orientazione del robot nel piano. Le n traiettorie sono poi il risultato dell'applicazione di un algoritmo di odometria inversa su particolari perturbazioni nella traiettoria dell'esperto.

Per la sintesi e l'applicazione del controllore a partire dalle n traiettorie di esempio si è invece creato un algoritmo in grado di replicare i risultati teorici presentati in [3]. Il controllore così ottenuto è in grado di fornire, sulla

base della posizione e dell'orientazione ottenute applicando lo stesso algoritmo di odometria, le velocità ai motori utili a realizzare la manovra di parcheggio richiesta.

La tesi segue la seguente struttura. Nel Capitolo 2 si fornisce una breve descrizione del robot Thymio II. L'algoritmo per la generazione delle traiettorie di esempio e le formule di odometria sono presentati nel Capitolo 3. Il Capitolo 4 riassume i risultati teorici presentati in [3] utili allo scopo della tesi. La procedura e i risultati sperimentali sono invece presentati nel Capitolo 5. La tesi si conclude con alcuni commenti su possibili sviluppi futuri nel Capitolo 6.

Capitolo 2

Thymio II

In questo capitolo si propone un'introduzione al robot mobile Thymio II, rappresentato in figura 2.1. Questo è il robot che ci ha affiancato durante lo svolgimento della tesi, e sulla sue caratteristiche è stata basata la scelta dei diversi algoritmi implementati. Si offre, pertanto, una breve panoramica sulle caratteristiche di questo robot sviluppato dalla casa produttrice Mobsya. Thymio II è stato progettato per essere utilizzato in ambienti educativi, mettendo a disposizione un'opzione economica per l'insegnamento della robotica agli studenti. Per dettagli si rimanda alla documentazione [4] e al report tecnico [8].

2.1 Software e comunicazione wireless

Software

Thymio è accompagnato da "Thymio Suite" e comprende numerosi programmi per interfacciarsi al robot, tra questi si cita "Thymio Device Manager". Quest'ultimo serve come ponte di collegamento tra l'applicazione da sviluppare ed il robot Thymio.

Il software del robot si fonda su Aseba, un'architettura basata su eventi; questo significa che porzioni di codice vengono eseguite in maniera asincrona alla ricezione di uno specifico evento. In particolare, se ne possono distinguere di due tipi:

1. **eventi globali**, sono trasmessi tra nodi[1] appartenenti allo stesso network,
2. **eventi locali**, sono trasmessi all'interno del nodo.

L'evento è caratterizzato da un identificatore e, opzionalmente, da un pacchetto dati. In pratica, è necessario definire una funzione chiamata *event-handler* che è caratterizzata dal nome dell'evento e dal codice da eseguire. Quando viene ricevuto un evento con lo stesso nome, il codice associato viene eseguito.

[1] Il robot all'interno del network.

Figura 2.1: Thymio II

Comunicazione wireless

L'interfaccia wireless è stata progettata per consentire all'utente di utilizzarla senza preoccuparsi dei dettagli tecnici. Pertanto, si vuole solo segnalare che non viene garantita l'affidabilità nella comunicazione. Informazioni riguardanti la definizione del protocollo, costruito ad hoc sopra al layer fisico IEE 802.15.4, sono presenti nella pubblicazione [7].

2.2 Attuatori, sensori e variabili

Si è deciso di presentare esclusivamente i sensori e gli attuatori utili alla sperimentazione. Per completezza, si riporta la figura 2.2, che fornisce una panoramica globale.

Motori

I due motori elettrici, uno per ciascuna delle due ruote, sono comandati separatamente da due controllori interni di tipo PID. Si può accedere sia alla velocità di riferimento che alla velocità effettiva della ruota tramite le seguenti variabili:

1. $motor.right.target \in [-500, 500]$, velocità di riferimento per la ruota destra, in modo analogo per la ruota sinistra($motor.left.target$).

2. $motor.right.speed \in [-500, 500]$, velocità attuale per la ruota sinistra, in modo analogo per la ruota destra($motor.left.speed$).

Il valore è aggiornato con una frequenza di 100 Hz e, contemporaneamente, viene emesso un evento locale di nome 'motor'. Questo consentirà di implementare in maniera semplice la logica per l'acquisizione delle misure di velocità (di riferimento ed effettiva) per ciascun motore.

Si osserva che i valori delle variabili sono considerati adimensionali, per comodità si dirà che le velocità sono espresse in unità di misura del robot. Nella parte sperimentale, si presenterà un metodo pratico per associare il loro valore

a delle variabili con unità di misura fisiche. Più precisamente, si determinerà la relazione per calcolare la velocità lineare del centro della ruota espressa in cm/s.

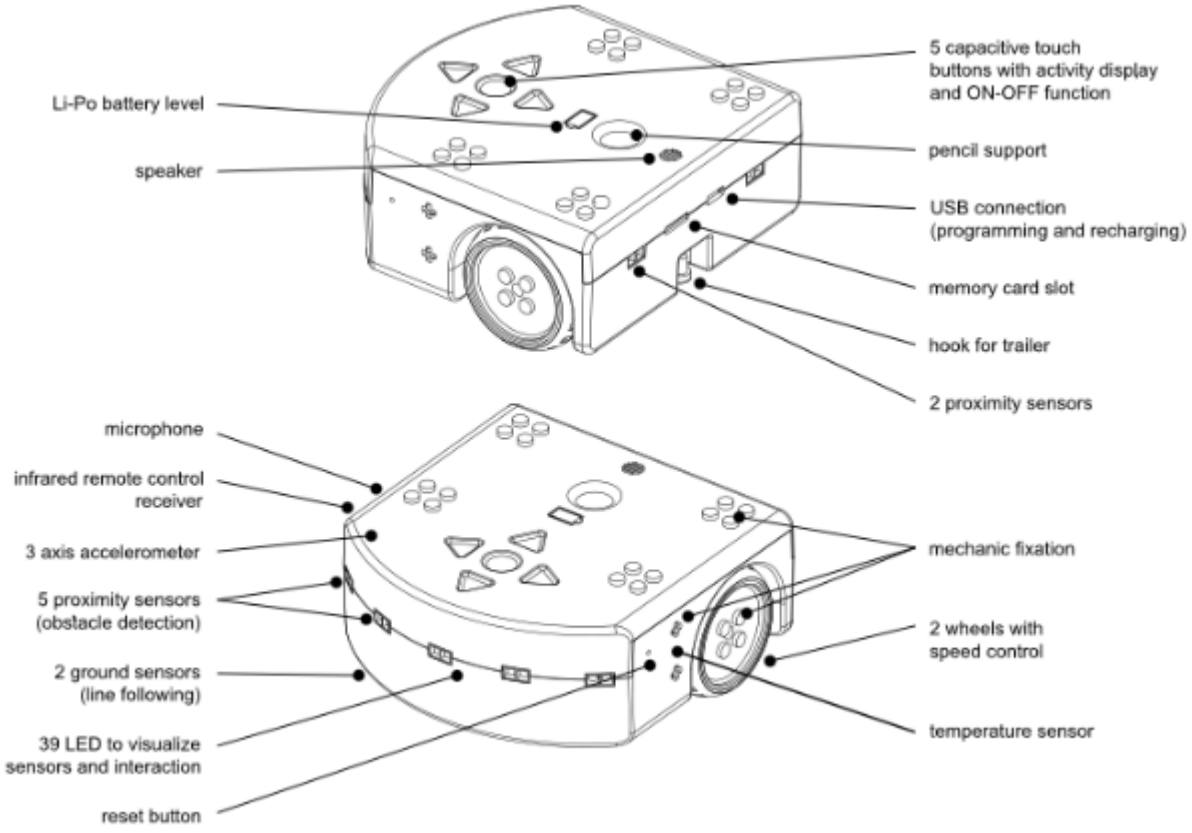

Figura 2.2: sensori ed attuatori robot thymio II

Sensori ad infrarossi

I due sensori ad infrarossi presenti sul robot, illustrati in figura 2.3, consentono di misurare il grado di riflessione della superficie, permettendo una distinzione precisa tra superfici di colore nero e bianco. Questa peculiarità suggerisce una modalità per poter far eseguire al robot una manovra di parcheggio d'esempio, necessaria per determinare la legge di controllo utilizzata durante questo lavoro. Si tratterà, infatti, di un algoritmo di inseguimento di una linea nera.

Il valore del sensore è accessibile tramite la variabile $prox.ground.delta \in [0, 1024]$, la quale si aggiorna ad una frequenza di 10 Hz emettendo un evento locale di nome "prox".

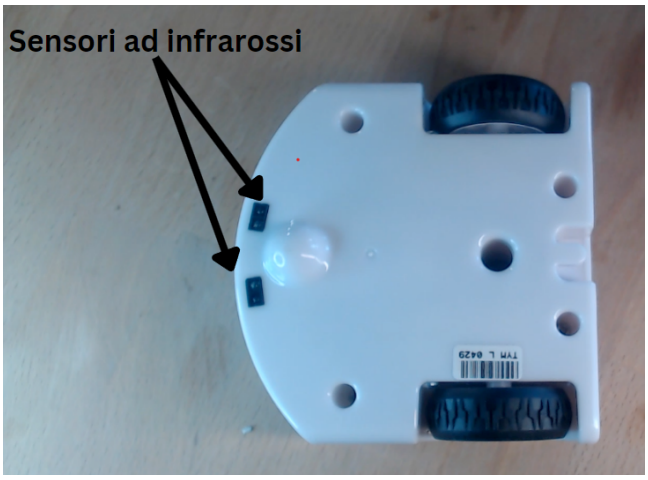

Figura 2.3: Posizione sensori ad infrarossi

Il valore della variabile corrisponde all'intensità della luce ad infrarossi riflessa dalla superficie. Quindi, come si vedrà nella sezione 3.1, un valore alto corrisponde ad una zona particolarmente riflessiva (colore bianco), mentre un valore basso ad una zona poco riflessiva (colore nero).

Capitolo 3

Verso la generazione del set di dati

In questo capitolo si presentano gli algoritmi fondamentali per la raccolta dati necessari alla definizione del controllore. In particolare, il robot, operante in modalità inseguimento-linea-nera è in grado di eseguire la traiettoria proposta dall'utente esperto. I sensori forniscono le velocità dei motori, che costituiranno gli input, mentre le formule di odometria consentono di stimare lo stato del sistema.

3.1 Inseguimento linea nera

L'algoritmo di inseguimento deve permette al robot di navigare su un percorso predefinito, individuato da una linea nera su uno sfondo bianco. L'obbiettivo può essere raggiunto usando solamente un sensore ad infrarossi, come descritto in [1], permettendo di sfruttare l'altro sensore per altri compiti.

Nel seguito si espone prima il comportamento del sensore al cambiare della superficie e dopo come tale comportamento viene sfruttato per far seguire al robot la linea nera. Per una maggiore chiarezza, nella sezione del sensore si è riportato il caso pratico in esame; l'algoritmo sviluppato è valido per un qualunque sensore che abbia un comportamento simile.

3.1.1 Sensore

Il sensore permette di identificare il valore medio della riflessività della superficie di un quadrato 6x6 mm. La figura 3.1 rappresenta i valori assunti dalla variabile del sensore in funzione della sua distanza dalla linea nera. In particolare, l'asse delle x denota la distanza del sensore dal confine tra la superficie nera e quella bianca.

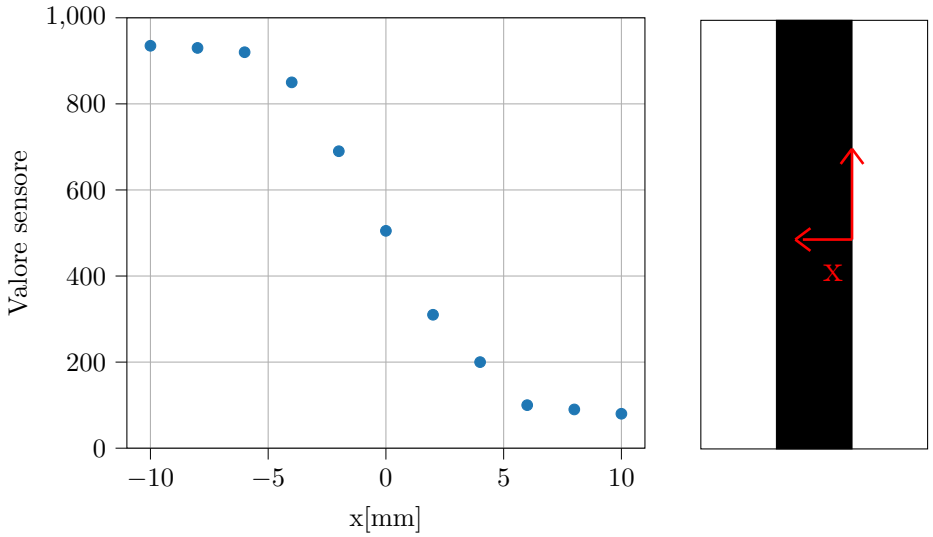

Figura 3.1: Il valore del sensore al variare della sua distantza (x) dal bordo

Si osservano i seguenti casi:

1. $x = 0$, il sensore è posizionato sul bordo della linea nera.
2. $x > 0$, il sensore è posizionato internamente alla linea nera ad una distanza pari ad x.
3. $x < 0$, il sensore è posizionato nella zona bianca(o alta riflessività) ad una distanza pari ad $|x|$.

Quindi un valore basso della variabile corrisponde ad una superficie mediamente poco riflessiva (zona nera), la zona bianca è invece identificata da un valore alto della variabile.

3.1.2 Algoritmo di controllo

Per prima cosa, si presenta l'algoritmo [1] in modo intuitivo per poi passare ad una descrizione più dettagliata. Il concetto alla base è semplice: l'algoritmo deve essere in grado di guidare il robot in modo tale che il sensore tenda ad un certo valore di riferimento. Quest'ultimo rispecchia il valore catturato dal sensore nel momento in cui si trova sul bordo della linea nera, quindi una zona composta in egual parte da superficie bianca e superficie nera.

In funziona dello scostamento massimo ammissibile (definito poi) ,E, del valore misurato,y, e del valore di riferimento,u, l'algoritmo segue due strade:

1. $|u - y| < E \Rightarrow$ **il robot si trova sul percorso**.
   Le velocità delle ruote vengono regolate in modo da seguire la traiettoria. Per esempio, si consideri il caso in cui sia $u - y < 0$; questo implica che il sensore, pur essendo vicino alla linea nera, si sta dirigendo verso una zona più chiara. Di conseguenza, definendo un valore dipendente dallo scostamento $u - y$, è possibile riallineare il robot con il bordo della linea sottraendo questo valore alla velocità della ruota sinistra e aggiungendolo alla velocità della ruota destra.

2. $|u - y| > E \Rightarrow$ **il robot ha perso il percorso.**
   In questo caso, le velocità vengono regolate in modo che il robot ruoti su se stesso, permettendo di ritrovare il bordo nero.

Si noti che la scelta di aggiungere o sottrarre il valore alla ruota destra o sinistra è completamente arbitraria. Se si opta per l'alternativa opposta, il robot seguirà la medesima traiettoria ma lungo l'altro margine della linea nera.

Algoritmo in dettaglio

Si passa ora a descrivere in maniera dettagliata l'algoritmo, per prima cosa si definisce il valore di riferimento come:

$$u[k] = \frac{maxi[k] - mini[k]}{2}$$

Dove maxi[k] rappresenta il valore massimo misurato fino al campione k-esimo, mentre mini[k] rappresenta il valore minimo. Questa definizione permette di svincolare il valore di riferimento dal sensore,consentendo al sistema di calibrarsi in modo autonomo. Tale valore in genere si assesta in breve tempo, e può essere considerato costante già dopo poche iterazioni dell'algoritmo.

Successivamente, si definisce lo scostamento massimo come:

$$E[k] = 0.45 \cdot (maxi[k] - mini[k]).$$

Si nota che gli stessi ragionamenti presentati per $u[k]$ sono estendibili a $E[k]$. A questo punto, studiamo il caso in cui il robot è considerato sul percorso, in termini matematici si ha:

$$y[k] \in [u[k] - E[k], u[k] + E[k].$$

In questo caso il comando delle velocità della ruote è ottenuto dalla seguente legge di controllo:

$$\begin{pmatrix} v_l[k] \\ v_r[k] \end{pmatrix} = \begin{pmatrix} v_b \\ v_b \end{pmatrix} + \begin{pmatrix} K_p \\ -K_p \end{pmatrix} \cdot e[k] + \begin{pmatrix} K_I \\ -K_I \end{pmatrix} \cdot \sum_{j=0}^{k} e[j] \tag{3.1}$$

Dove:

- v_l è la velocità della ruota sinistra[1].
- v_r è la velocità della ruota destra[2].
- v_b è la velocità di base del robot, da impostare arbitrariamente ma nei limiti concessi dal robot.
- K_p è la costante associata alla parte proporzionale del controllore.
- K_I è la costante associata alla parte integrale del controllore.
- $e \coloneqq u - y$ è lo scostamento.

La parte integrativa è di notevole importanza, infatti non è possibile selezionare una costante K_p in modo che il robot segue sia tratti rettilinei che curvilinei.

Per comprendere meglio questa situazione, immaginiamo di prendere in considerazione un controllore proporzionale e di aver tarato la costante K_p su un tratto rettilineo. Dato che in curva sono necessarie variazioni di velocità più significative il valore di K_p non è sufficientemente elevato per impostare velocità adeguate, il che porta il robot a perdere continuamente[3] il percorso. Considerazioni analoghe si possono fare per il caso di taratura su tratto curvilineo ed inseguimento su tratto rettilineo. Il contributo integrativo riesce a risolvere il problema. Infatti, nel caso in cui il robot sta per perdere la traiettoria il segno $e[k]$ è costante e quindi il contributo integrativo risulta sufficientemente elevato per ridirezionare il robot.

Si vede ora il caso $y[k] \notin [u[k] - E[k], u[k] + E[k]$.
Il sensore del robot è considerato troppo lontano dal bordo, la legge di controllo è la seguente:

$$\begin{pmatrix} v_l[k] \\ v_r[k] \end{pmatrix} = \begin{pmatrix} e[k]/2 \\ -e[k]/2 \end{pmatrix} \tag{3.2}$$

Il moto di rotazione è stato reso proporzionale allo scostamento, permettendo una più graduata risposta allo smarrimento del bordo.

[1] motor.left.target
[2] motor.right.target
[3] questo è dovuto all'intervento della seconda parte dell'algoritmo, si noterà un comportamento a zigzag

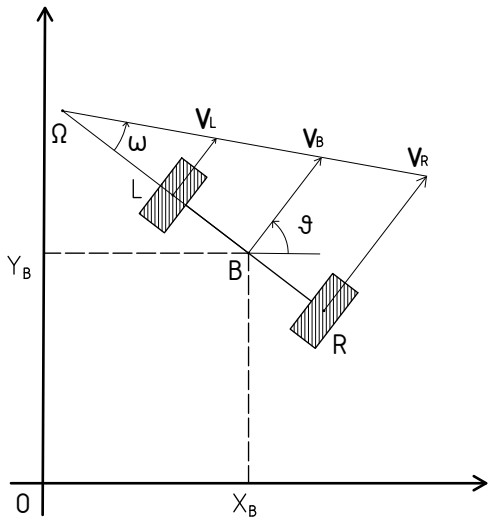

Figura 3.2: sistema di riferimento e coordinate scelte per rappresentare la configurazione del robot.

3.2 Odometria

L'odometria è uno strumento della robotica che permette di stimare la posizione di un robot mobile a partire dal valore dei suoi sensori interi. Le formule proposte permettono di ricostruire la traiettoria seguita dal robot durante una sua qualunque manovra a patto di conoscere il suo campo di velocità in ogni istante di tempo.
Si espone innanzitutto il modello adottato per il robot Thymio, successivamente si procede con il calcolo del campo di velocità e infine il calcolo della configurazione del robot.

3.2.1 Modello utilizzato

Il robot mobile è modellizzato come un corpo rigido vincolato ad eseguire un moto piano, questo permette di conoscere il campo di velocità a partire dai vettori velocità di due soli suoi punti. Considerando il moto delle ruote come un rotolamento puro, si può ricavare le velocità del centro (v_c) delle due ruote a partire dal valore della velocità angolare della ruota(w_r), normalmente ottenuta da un encoder, e dal suo diametro (D) come $v_c = \frac{2 \cdot w_r}{D}$.

Il campo delle velocità è definito da $\vec{v}_k = \vec{w} \times (\vec{x}_{k\Omega})$ dove:

- $\vec{v}_k$, è la velocità di un punto k generico appartenente al robot.
- $\vec{w}$, è la velocità angolare.
- $\vec{x}_{k\Omega}$, è la posizione del punto k rispetto al centro d'istantanea rotazione Ω.

La configurazione del robot è univocamente determinata da tre coordinate indipendenti, si è scelto:

- $\begin{pmatrix} x_B \\ y_B \end{pmatrix}$, coordinate cartesiane del punto di mezzeria del segmento che congiunge i due centri delle ruote.
- θ, angolo ottenuto dall'intersezione di un asse solidale al robot e da un asse fisso nello spazio.

Le coordinate ed il sistema di riferimento sono rappresentate in figura 3.2.
A questo punto sfruttando le seguenti relazioni :

- $\vec{v}_L = \vec{w} \times (\vec{x}_{L\Omega})$ velocità del centro della ruota sinistra,
- $\vec{v}_R = \vec{w} \times (\vec{x}_{R\Omega})$ velocità del centro della ruota destra,
- distanza (d), nota, tra punto L ed R,

si ricava:

$$\begin{aligned} w &= \frac{v_R - v_L}{d}, \\ v_B &= \frac{v_R + v_L}{2}. \end{aligned} \tag{3.3}$$

Così, una volta ottenuti i valori delle velocità v_R e v_L dai sensori, si dispone delle velocità associate alle coordinate indipendenti selezionate.
A questo punto, attraverso l'integrazione, è possibile risalire alle configurazioni assunte dal robot. In particolare, si riporta un metodo di integrazione utile nei casi in cui le velocità non siano note in modo continuo nel tempo, ma provengano da un campionamento(T_s), come avverrà nel caso pratico.

Basandosi sull'algoritmo descritto in [5], si è giunti alla seguente soluzioni[4]:

$$\begin{aligned} x_B[k+1] &= x_B[k] + \frac{v_B[k]}{w[k]} \cdot (\sin\theta[k+1] - \sin\theta[k]), \\ y_B[k+1] &= y_B[k] + \frac{v_B[k]}{w[k]} \cdot (\cos\theta[k+1] - \cos\theta[k]), \\ \theta[k+1] &= \theta[k] + w[k]T_s, \end{aligned} \tag{3.4}$$

[4]Le condizioni iniziali sono considerate note.

le quali permettono di calcolare esattamente la lunghezza del percorso compiuta dal robot assunto che nell'intervallo di tempo [kTs,(k+1)Ts] la velocità angolare w ed la posizione del centro d'istantanea rotazione siano costanti. Nel caso in cui il campo di velocità sia traslatorio ($w = 0$), non compatibile con 3.4, si usa la seguente soluzione:

$$\begin{aligned} x_B[k+1] &= x_B[k] + v_B[k]T_s \cos\theta[k], \\ y_B[k+1] &= y_B[k] + v_B[k]T_s \sin\theta[k], \\ \theta[k+1] &= \theta[k]. \end{aligned} \tag{3.5}$$

In questo caso è stato assunto costante il vettore velocità $\vec{v}_B$.

Capitolo 4

Il controllore

Ora verrà esposta la teoria necessaria alla prova sperimentale, l'obbiettivo è di esporre una procedura per il calcolo di un controllore in grado di replicare una certa traiettoria ben definita. Innanzitutto, si introduce il problema e la soluzione da un punto di vista più generale come presentato in [3], poi si particolarizza al caso di interesse.

4.1 Problema

La prima parte di questa sezione è un riassunto dell'articolo [3]. Si consideri il seguente sistema lineare a tempo continuo periodico e tempo variante:

$$\dot{x(t)} = A(t)x(t) + B(t)u(t), \tag{4.1}$$

dove $A(t) \in \mathbb{R}^{n\times n}$ e $B(t) \in \mathbb{R}^{n\times m}$ tali che:

$$A(t+T) = A(t),\ B(t+T) = B(t),$$

$x(t) \in \mathbb{R}^n$ $u(t) \in \mathbb{R}^m$ rappresentano rispettivamente lo stato e l'ingresso al tempo $t \in \mathbb{R}$.

Viene assunto che un operatore esperto fornisca n traiettorie di stato a partire da n diversi stati iniziali $x_i(0), i = 1, ..., n$. Le traiettoria devono avere lo stato finale in un intorno di 0, quindi $\forall i$ si ha $x_i(T) \approx 0$. I corrispondenti ingressi sono indicati come $u_i(t)$. Infine, si raggruppano le traiettorie in:

$$\begin{aligned} X(t) &:= [x_1(t)\ x_2(t)\ ...\ x_n(t)] \\ U(t) &:= [u_1(t)\ u_2(t)\ ...\ u_n(t)] \end{aligned} \tag{4.2}$$

Assunto per ipotesi le seguenti affermazioni:

1. lo stato finale è piccolo rispetto alla traiettoria, secondo il criterio: $x_i(0) = p \cdot x_i(T)$ dove $\|p\|_1 < 1$.

2. le condizioni iniziali delle n traiettorie devono essere indipendenti, ovvero $rank[X(0)] = n$.

Considerato il sistema retroazionato $u(t) = K(t)x(t)$, dove $K(t+T) = K(t)$ e

$$K(t) := U(t)X^{-1}(t). \tag{4.3}$$

Si può dimostrare:

- 4.3 è ben definito
- il sistema di controllo ad anello chiuso composto da 4.1 e 4.3 è asintoticamente stabile.
- lo stato del sistema converge a zero.

Particolarizzazione

L'evoluzione dello stato di un sistema continuo è rappresentabile da un'equazione del tipo:

$$\dot{x}(t) = f(x(t), u(t))$$

Ora si procede a determinare esplicitamente questa relazione, identificando pertanto la funzione f.
Nel nostro caso, lo stato è la configurazione del robot nel piano, mentre per l'ingresso si considerano direttamente le velocità ottenute da 3.3, quindi secondo le scelte fatta in 3.2.1, si ha:

$$x(t) = \begin{pmatrix} x_B(t) \\ y_B(t) \\ \theta(t) \end{pmatrix} \quad u(t) = \begin{pmatrix} v_B(t) \\ w(t) \end{pmatrix}$$

Aiutandosi con la figura 3.2 si ottiene facilmente:

$$f(x_B, y_B, \theta, v_B, w) = \begin{pmatrix} v_B \cos\theta \\ v_B \sin\theta \\ w \end{pmatrix}$$

In particolare, f è una funzione non lineare questo porta a non poter esprimere la rappresentazione come in 4.1. Per risolvere questo problema si può seguire il procedimento in [2] andando a linearizzare il sistema attorno a delle traiettorie di stato ed ingresso nominali, così da ottenere una rappresentazione del tipo:

$$\dot{z}(t) = A(t)z(t) + B(t)\tilde{u}(t)$$

Dove $z(t) = x(t) - \bar{x}(t)$, $\tilde{u}(t) = u(t) - \bar{x}(t)$ mentre $\bar{x}(t)$ e $\bar{u}(t)$ sono lo stato e l'ingresso nominale.
Ai fini del calcolo del controllore non è necessario valutare esplicitamente le matrici A(t) e B(t), infatti si può direttamente applicare 4.3 con

$$\tilde{U}(t) = U(t) - \bar{U}(t), \; Z(t) = X(t) - \bar{X}(t). \tag{4.4}$$

Dove $U(t) \in \mathbb{R}^{2\times 3}$, $X(t) \in \mathbb{R}^{3\times 3}$ ed hanno lo stesso significato di 4.2, mentre si definisce:

$$\bar{X}(t) := [\bar{x}(t)\ \bar{x}(t)\ \bar{x}(t)]\ , \bar{U}(t) := [\bar{u}(t)\ \bar{u}(t)\ \bar{u}(t)]$$

Il controllore ottenuto è applicabile al sistema originale, tuttavia il sistema retroazionato sarà localmente asintoticamente stabile [2]. Questo significa che nel caso in cui lo stato del robot $x(t)$ si discosta in maniera eccessiva dalla traiettoria nominale $\bar{x}(t)$ il controllore fornisce degli input al sistema che portano l'allontanamento dello stato.

In vista della parte sperimentale, è utile considerare il seguente aspetto: nel caso in cui il set di dati sia fornito tramite un campionamento, tutte le considerazioni espresse precedentemente rimangono valide. Infatti, la legge di definizione del controllore dipende solamente dall'istante di tempo in cui ci si trova. Quindi, si ottiene la seguente definizione del controllore e legge di controllo campionata:

$$\begin{aligned} K[k] &= \tilde{U}[k]\tilde{X}^{-1}[k] \\ u[k] &= \bar{u}[k] + K[k](x[k] - \bar{x}[k]). \end{aligned} \tag{4.5}$$

Capitolo 5

Sperimentazione

La prova sperimentale è stata suddivisa in tre fasi distinte. Inizialmente, si è stabilita la comunicazione tra il PC e il robot, analizzando la perdita di pacchetti in diverse modalità di funzionamento. Successivamente, è stato implementato l'algoritmo di inseguimento, il quale, insieme alle formule di odometria, ha permesso di configurare un setup per generare i dati necessari alla definizione del controllore data-driven. Il lavoro sperimentale si è concluso con la realizzazione e l'attuazione del controllore in diverse condizioni iniziali disturbate, verificando così le prestazioni della legge di controllo.

Tutto il lavoro è stato svolto utilizzando due linguaggi di programmazione: Python ed Aseba. Si allega il link al repository su GitHub contenente la sintassi dei programmi utilizzati.
`https://github.com/units-dronelab/TESI_Marzi_Lorenzo`

5.1 Comunicazione robot-PC

La comunicazione è stata implementata utilizzando la libreria tdmclient [6], che consente di operare a un livello elevato e stabilire una connessione di tipo TCP tra lo script Python ed il 'Thymio Device Manager', sarà quest'ultimo ad occuparsi di interagire con il robot.
La trasmissione delle informazioni avviene attraverso lo scambio di eventi globali. Per chiarire, consideriamo l'esempio dell'invio di un evento dal PC al robot. Una volta deciso il nome e i dati da inviare, è necessario fare essenzialmente due cose: notificare a 'Thymio Device Manager' la presenza del nuovo evento e caricare sul robot un programma con un *event handler*[1] per gestire tale evento. A questo punto, quando l'evento viene emesso dal PC, la funzione all'interno del robot verrà richiamata. In questo modo, ad esempio, sarà possibile inviare i comandi derivati dalla legge di controllo.

[1] vedere sezione 2.1

L'emissione dell'evento dal robot al PC segue la stessa logica, ma in questo caso la libreria consente di definire una singola funzione che viene richiamata ogni volta che un evento viene ricevuto. Per distinguere i vari eventi, è sufficiente utilizzare un'istruzione condizionale (if).

Si passa ora a valutare le prestazioni della connessione: sono stati condotti due tipi di esperimenti. Il primo esperimento consisteva nella modalità unidirezionale, in cui il robot inviava dati e il computer riceveva passivamente. Nel secondo esperimento, invece, è stata stabilita una comunicazione bidirezionale.

La tabella 5.1 riporta i dati ottenuti durante gli esperimenti condotti in modalità unidirezionale collegamento wireless, con l'invio di eventi a una frequenza di 100 Hz. Questa frequenza è stata scelta in quanto rappresenta la massima frequenza utile per l'invio dei dati, in coincidenza con la frequenza di aggiornamento delle variabili del motore tramite l'evento locale "motor". Per misurare il tempo, è stata definita una variabile chiamata "cicli 100 Hz", la quale viene incrementata di 1 ogni volta che si riceve l'evento locale "motor". La durata della misura è considerata pari a 1 minuto quando viene ricevuto un valore pari a 6000, indicando così il completamento della misurazione.

Numero misure	12
Tempo misura	60 secondi
Frequenza invio	100 Hz
Campioni aspettati	72000
Campioni misurati	70931
Buchi trovati	$1008 \times 1 + 25 \times 2 + 1 \times 3 + 2 \times 4 = 1069$

Tabella 5.1: Dati in comunicazione unidirezionale wireless

Si registra una perdita media di circa 1.5 campioni ogni 100 (1.5%), situazione compatibile con la perdita di pacchetti evidenziata in [7]. Non sono stati rilevati buchi significativi con perdite di più pacchetti consecutivi; al massimo, si è verificata una perdita consecutiva di 4 pacchetti. Si è poi ripetuta la procedura con la connessione di tipo USB ottenendo i risultati riportati in tabella 5.2 Si evidenzia che in questa prova non si è rilevata alcuna perdita di pacchetti,

Numero misure	5
Tempo misura	60 secondi
Frequenza invio	100 Hz
Campioni aspettati	3000
Campioni misurati	3000
Buchi trovati	0

Tabella 5.2: Dati in comunicazione unidirezionale usb

questo porta a ritenere valido lo script di phyton implementato e confermare che la perdita di pacchetti è associata alla modalità di comunicazione wireless.

La modalità bidirezionale in wireless non è stata stabilita con successo a 100 Hz; si è riscontrata una perdita di pacchetti significativa, approssimativamente del 25 % , rendendo impraticabile il controllo in tempo reale. Di conseguenza, è stata presa la decisione di ridurre la frequenza operativa a 10 Hz. A questa frequenza, la perdita di pacchetti è stata ridotta all'ordine del 5%. Anche se non ottimale, questa configurazione ha consentito comunque un controllo accettabile del robot. Per quanto riguarda la connessione USB, a 10 Hz non sono state osservate perdite di pacchetti.

5.2 Set-up generazione traiettorie

Lo scopo di questa fase è stato di creare un set-up per ottenere i movimenti dell'ingresso e dello stato a partire da un qualunque percorso. Sono stati installati gli algoritmi proposti nel capitolo 3 ed è stata implementata un'interfaccia grafica utilizzando la libreria tkinter.

L'algoritmo di inseguimento, grazie alla sua semplicità, è stato implementato direttamente sul processore del robot. Questa scelta ha permesso di evitare le problematiche legate alla perdita di pacchetti, che si verificano nella comunicazione bidirezionale con il PC. Più in dettaglio, le leggi di controllo 3.1 e 3.2 sono state implementate nel linguaggio di programmazione Aseba. Dove le costanti di K_I e K_p sono state calcolate tramite la procedura sperimentale di Ziegler, come suggerito in [1].

L'algoritmo viene richiamato alla ricezione dell'evento locale "prox" [2], che avviene ad una frequenza di 10 Hz. L'invio dei dati avviene alla ricezione dell'evento locale "motor" questa scelta ha permesso di acquisire sia le velocità generate dall'algoritmo di inseguimento(10Hz) che le velocità attuali del robot(100Hz), consentendo così di catturare anche il transitorio del sistema. Il pacchetto dati è infine completato con l'invio del valore della variabile "cicli 100 Hz", definita nella sezione 5.1.

Prima di applicare le formule di odometria, è stato necessario convertire le velocità espresse in unità di misura del robot in unità di misura fisiche. Per fare ciò, è stato realizzato un semplice esperimento: le variabili di riferimento del motore sono state impostate, e il robot è stato fatto percorrere una linea retta di 30 cm, delimitata dall'inizio e dalla fine da una linea nera. Durante il percorso, ogni volta che il robot attraversava la linea nera, veniva registrato il tempo. In questo modo è stato possibile calcolare la velocità fisica del robot, stabilendo così una corrispondenza con la velocità in unità di misura del robot.

Tuttavia, è emerso che questa relazione non è lineare, soprattutto a basse velocità. Pertanto, è stato necessario creare una tabella discreta 5.3 per ottenere una corrispondenza accurata tra i valori delle velocità.

[2] si veda sezione 2.1

Velocità robot	**Costante proporzionalità [cm/s]**
≤ 10	0.049
≤ 17	0.043
≤ 25	0.040
≤ 50	0.038
≤ 75	0.036
≤ 100	0.035
≤ 150	0.034
≤ 200	0.033
≤ 500	0.033

Tabella 5.3: costanti per la conversione della velocità da unità immaginarie a fisiche

A questo punto, è possibile applicare le formule di odometria. A partire dal set di dati delle velocità espresse in unità del robot, si effettua la trasformazione in velocità fisiche. Successivamente, si applica l'equazione 3.3. Infine, attraverso un'istruzione condizionale, si distinguono due casi: quando $w \neq 0$ e $w = 0$, applicando rispettivamente le equazioni 3.4 e 3.5. In questo modo si ottiene la traiettoria dello stato del robot durante l'inseguimento.

Ricapitolando, arrivati a questo punto, si è in grado di generare lo stato del robot e gli ingressi corrispondenti per qualsiasi traiettoria definita dalla linea nera.

5.3 Controllore

Seguendo le indicazioni fornite nel capitolo 4, la prima fase è stata quella di definire la traiettoria di stato nominale. A questo punto, sfruttando il set-up precedentemente sviluppato è possibile far eseguire al robot tale manovra ed ottenere i dati necessari per la definizione del controllore.

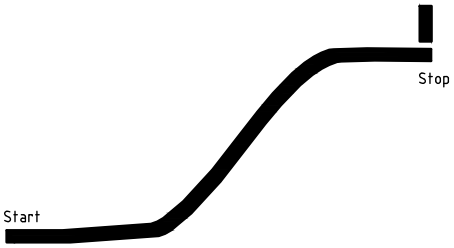

Figura 5.1: Traiettoria scelta per l'inseguimento

La figura 5.1 illustra la traiettoria selezionata, si evidenzia in particolare la presenza del segnale di fine percorso. Tale segnale è rilevato dal secondo sensore libero, il quale interrompe l'algoritmo di inseguimento, e le velocità vengono impostate a zero. Questa strategia consente di ottenere una partenza e una frenata del robot ben definite.

A causa dell'elevato livello di rumore presente nelle misure di velocità, è stata presa la decisione di effettuare quattro misurazioni distinte e successivamente calcolarne la media. Nei grafici 5.2 risulta evidente come l'effetto del disturbo sia ridotto notevolmente attraverso la media dei dati, rappresentata in nero.

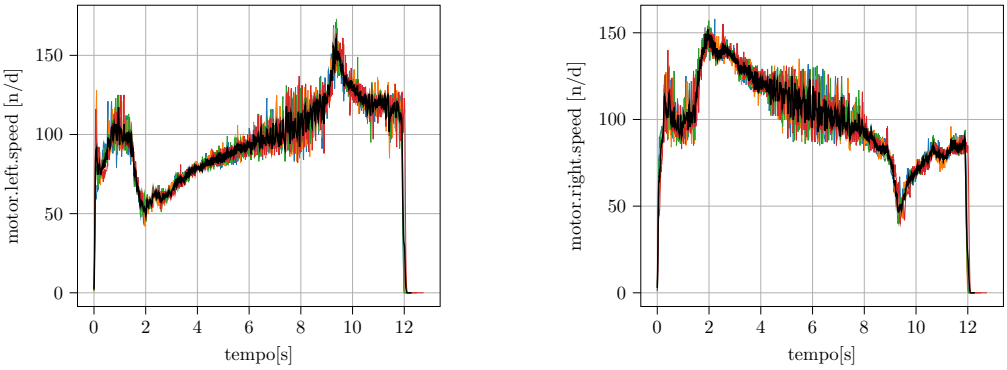

Figura 5.2: velocità nominale, rappresentata in nero, ottenuta dalla media di quattro misurazioni, rappresentate negli altri colori.

Prima di applicare l'odometria, è stato necessario compiere un ulteriore passo. L'algoritmo di inseguimento funziona correttamente solo se il sensore del robot è posizionato avanti rispetto alle due ruote. Per ottenere la manovra di parcheggio, tipicamente eseguita in retromarcia, è stato sufficiente notare che invertendo i segni delle velocità acquisite si ottiene una traiettoria speculare rispetto a quella seguita dal robot.

Arrivati a questo punto, si dispongono dei dati che rappresentano $\bar{x}[k]$ e $\bar{u}[k]$.Come descritto nella sottosezione particolarizzazione di 4.1, è necessario ottenere un numero di traiettorie pari alla dimensione del vettore di stato x(t). La scelta è stata quella di non ottenere direttamente queste traiettorie tramite misurazioni, ma utilizzando uno script in Python.In particolare, per il calcolo di ogni traiettoria si è applicato un disturbo allo stato finale della traiettoria nominale e ai valori delle velocità nominali. Successivamente, è stata eseguita un'integrazione all'indietro, facendo uso delle leggi di odometria, al fine di ottenere l'intera traiettoria disturbata. È stato fondamentale calibrare questi disturbi in modo adeguato: devono essere di entità tale da mantenere la stessa forma della traiettoria voluta e garantire il rispetto delle ipotesi del controllore.

Le due figure in 5.3 rappresentano gli input $U[k]$ per il calcolo $\tilde{U}[k]$ in 4.5, si nota che per via dei disturbi scelti alcune traiettorie non sono distinguibili.

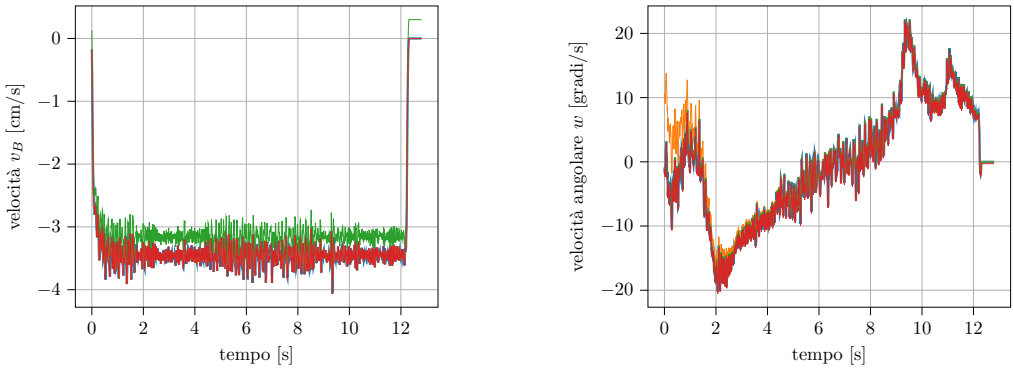

Figura 5.3: traiettorie di ingresso usate per la definizione del controllore. Non tutte le velocità sono distinguibili, questo è dovuto alla scelta dei disturbi

Invece, le due figure in 5.4 rappresentano i corrispondenti stati $X[k]$ per il calcolo di $\tilde{X}[k]$. Si nota la similitudine con lo stato nominale, evidenziato in nero, ed il rispetto dell'indipendenza delle condizioni iniziali.

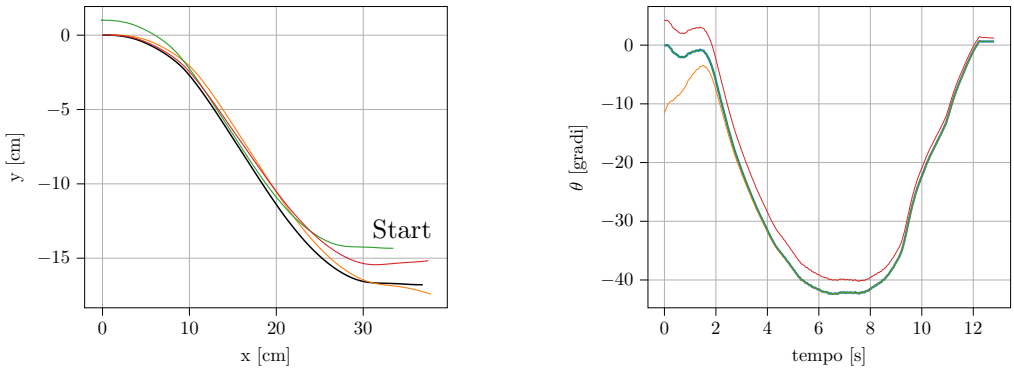

Figura 5.4: traiettorie di stato usate per la definizione del controllore. La nominale è in nero, il punto finale è l'intorno di 0.

Così, con la conoscenza della matrice K, è possibile implementare la legge di controllo definita in Eq. 4.5. Il controllo è completato dall'integrazione delle equazioni di odometria nel loop di comunicazione, permettendo di risalire allo stato $x[k]$. E' importante notare che il controllore è stato addestrato sulla velocità fisica, quindi prima di poter inviare le velocità target al robot è stato necessario convertirle ed effettuare un arrotondamento, dovuto al fatto che aseba lavora solamente con valori interi.

Prima di passare all'esposizione dei risultati si propone la figura 5.5, la quale ricapitola lo schema di funzionamento.

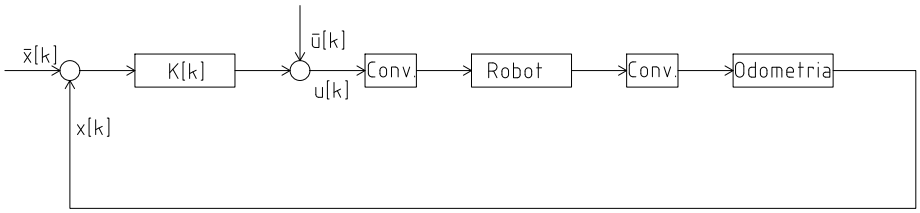

Figura 5.5: schema a blocchi che espone la logica del sistema di controllo.

5.4 Risultati ottenuti

Il controllore K è stato progettato considerando i dati campionati a 100 Hz, illustrati nelle figure 5.3 e 5.4. Tuttavia, non è stato possibile implementare la legge di controllo a questa frequenza. Infatti, la complessità [3] del sistema di controllo ha reso impossibile l'implementazione direttamente a bordo del robot. Pertanto, è stato necessario optare per una comunicazione bidirezionale tra lo script Python e il robot. Come evidenziato nella sezione dedicata alla comunicazione, si è deciso di operare a 10 Hz, rappresentando un adeguato compromesso tra la perdita di pacchetti e le prestazioni richieste per l'esecuzione della manovra di parcheggio. Di conseguenza, è stato effettuato un sotto campionamento del controllore K a tale frequenza.

Per valutare le prestazioni della legge di controllo, sono stati effettuati i seguenti esperimenti suddivisi in base alla perturbazione applicata alla condizione iniziale e al metodo di connessione utilizzato (usb o wireless).

Usb con leggera perturbazione

Questo esperimento ha consentito di testare il controllo nelle condizioni ideali, ovvero senza perdita di pacchetti. Le figure 5.6 confrontano lo stato del robot durante la prova con quello nominale, ed è evidente come il controllore sia riuscito a guidare il robot verso la traiettoria nominale, raggiungendo infine uno stato del sistema vicino allo stato nullo come descritto nel capitolo 4.

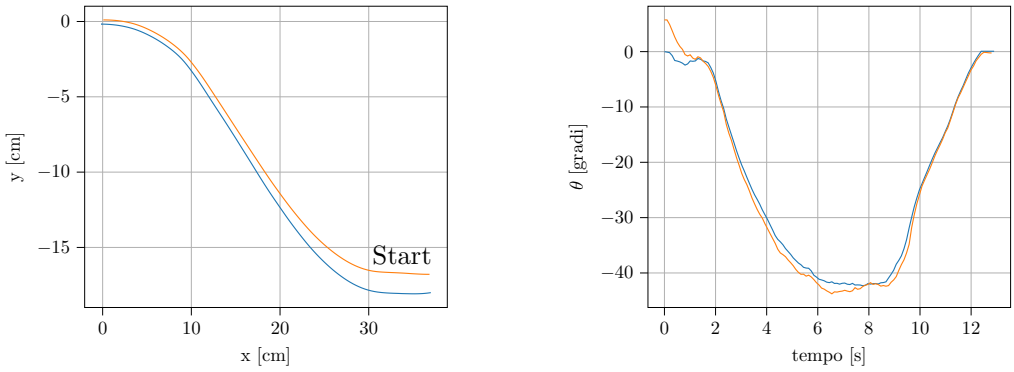

Figura 5.6: confronto tra la traiettoria nominale, in arancione, e la traiettoria ottenuta dal controllore in collegamento usb in condizioni di leggera perturbazione

[3] complessità relativa al linguaggio di programmazione Aseba

Nelle figure 5.7 è visualizzato come la velocità delle ruote del robot riesca a seguire le velocità ottenute dal controllo.

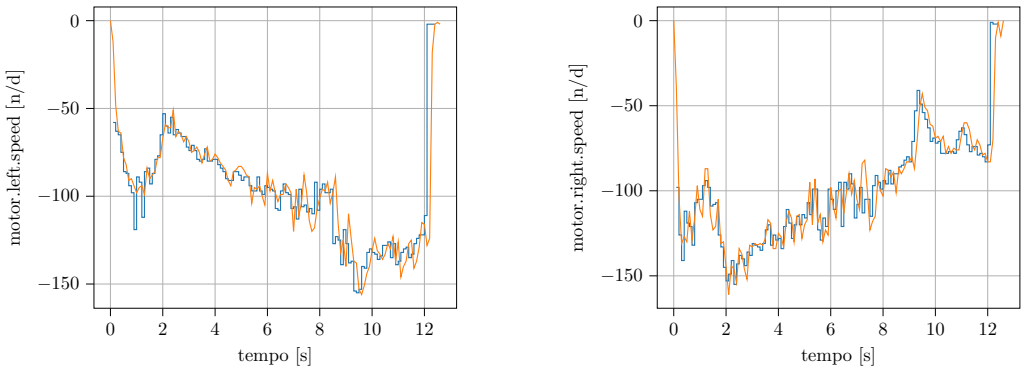

Figura 5.7: la velocità delle ruote del robot, in arancione, segue la velocità comandata dal controllore, in blu.

Connessione wireless con leggera perturbazione

L'esperimento è stato condotto con lo stesso disturbo sulle condizioni iniziali del caso usb, è quindi in grado di dimostrare come il sistema di controllo sia in grado di gestire la lieve perdite di pacchetti, portando risultati soddisfacenti.

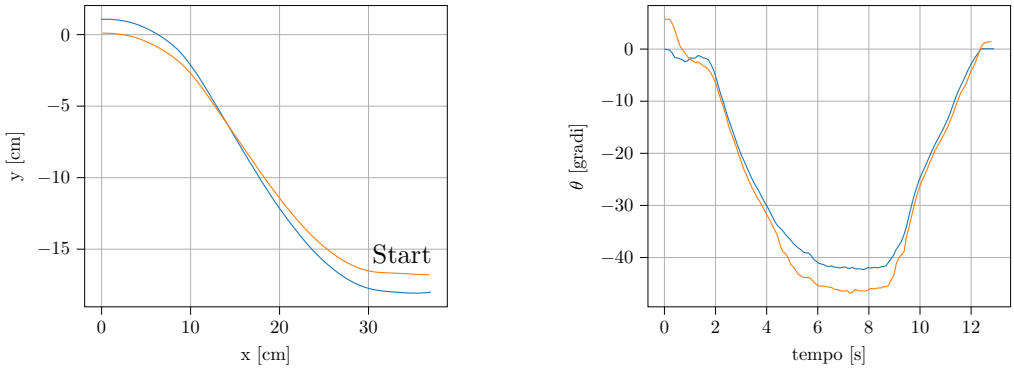

Figura 5.8: confronto tra la traiettoria nominale, in rosso, e la traiettoria ottenuta dal controllore in collegamento wireless in condizioni di leggera perturbazione

Connessione wireless con intensa perturbazione

Si è infine deciso di applicare una perturbazione intensa sulle condizioni iniziali. Per 'intensa', si intende una condizione iniziale molto distante da quelle utilizzate per la definizione del controllore, rappresentate in figura 5.4. Le figure 5.9 mostrano che anche in questo caso il sistema di controllo è in grado di guidare lo stato nella direzione corretta.

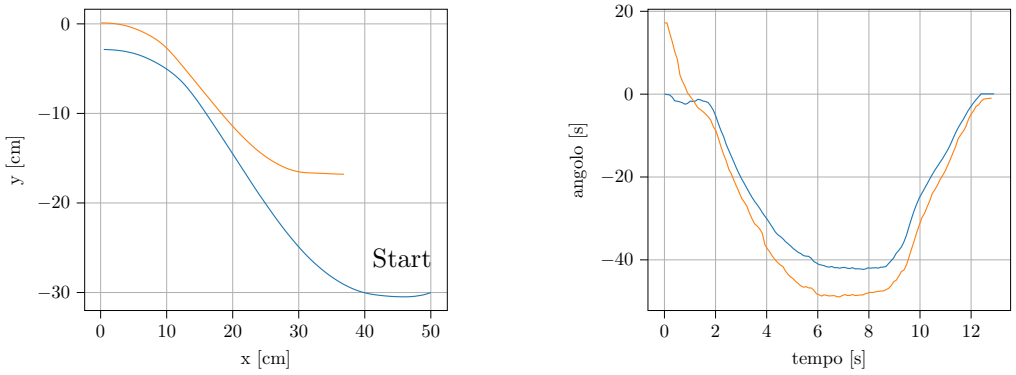

Figura 5.9: confronto tra la traiettoria nominale, in arancione, e la traiettoria ottenuta dal controllore in collegamento wireless in condizioni di intensa perturbazione

Capitolo 6

Conclusione

In conclusione, in questa tesi è stata descritta la procedura sperimentale condotta per applicare la tecnica di controllo *data-driven* presentata in [3] all'esecuzione automatica di una manovra di parcheggio di un robot mobile: il Thymio II.

In particolare, dai grafici 5.6, 5.8, e 5.9 si può osservare come la legge di controllo consente di effettuare manovre partendo da diverse condizioni iniziali. Inoltre, dalla similitudine della figura 5.6 ottenuta in collegamento usb e dalla figura 5.8 ottenuta con la connessione wireless, si evince che il controllore è in grado di gestire disturbi dovuti ad un perdita di pacchetti del 5%, garantendo il mantenimento di prestazioni più che accettabili.

Questo lavoro può essere considerato un buon punto di partenza per lo sviluppo di sistemi di parcheggio ben più complessi. In possibili lavori futuri, sarebbe interessante trovare una soluzione alternativa all'utilizzo dell'algoritmo di odometria per la stima puntuale della posizione e dell'orientazione nel robot nello spazio. Ad esempio, l'utilizzo di una telecamera in grado di eseguire il *tracking* della posizione potrebbe portare a stime più precise e puntuali dello stato del sistema e, di conseguenza, ad un miglioramento della precisione dei risultati sperimentali ottenuti. Sarebbe opportuno, inoltre, considerare come i dati ottenibili dai sensori di prossimità del robot possano intervenire nella realizzazione di controllori che, oltre alla sola realizzazione di manovre di parcheggio, siano anche in grado di operare in ambienti reali dove possono verificarsi collisioni con altri agenti mobili.

Bibliografia

[1] aseba wikidot. Thymio follows a black edge. http://aseba.wikidot.com/en:thymioedgefollower.

[2] Jay A. Farrell and Marios M. Polycarpou. Linearizing around a trajectory. https://www.globalspec.com/reference/21381/160210/chapter-5-1-2-linearizing-around-a-trajectory.

[3] Felice Andrea Pellegrino Franco Blanchini, Gianfranco Fenu and Erica Salvato. Data-driven periodic control in continuous-time. Technical report, UNITS, 2023.

[4] Mobsya. Aseba e thymio. http://aseba.wikidot.com/en:asebausermanual.

[5] Prof. Giuseppe Oriolo. corso "autonomous and mobile robotics", odometric localization. http://www.diag.uniroma1.it/~oriolo/amr/slides/Localization1_Slides.pdf.

[6] Yves Piguet. Communication with thymio ii robot via the thymio device manager. https://pypi.org/project/tdmclient/.

[7] Philippe Rétornaz, Fanny Riedo, Stéphane Magnenat, Florian Vaussard, Michael Bonani, and Francesco Mondada. Seamless multi-robot programming for the people: Aseba and the wireless thymio ii robot. In *2013 IEEE International Conference on Information and Automation (ICIA)*, pages 337–343. IEEE, 2013.

[8] Fanny Riedo. Thymio. Technical report, EPFL, 2015.