Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria dell'Automazione
Dipartimento di Elettronica, Informazione e Bioingegneria

Controllo MPC di una Sedia a Rotelle Autonoma

Relatore: Prof. Luca Bascetta
Correlatore: Prof. Matteo Matteucci

Tesi di laurea di:
Pietro Ilacqua Matr. 800787

Anno Accademico 2013/2014

A mio padre

Ringraziamenti

Vorrei comprendere in questi ringraziamenti tutti coloro che hanno avuto un ruolo di rilievo nel completare il mio percorso universitario.

Ringrazio i miei genitori, per avermi consentito di perseguire la formazione nel migliore dei modi senza avermi mai fatto mancare nulla.

Ringrazio chi è stato mio compagno nel portare a termine i complicati progetti nel corso di questi anni: Niccolò, Claudio e Flavio.

Ringrazio Stefano per le stressanti serate di studio concluse felicemente al Bar Giulia o al Tempio.

Ringrazio chi è stato mio compagno nelle lunghe notti di studio in Patio a Milano o alla Hartley a Southampton: Mattia, Paolo e Marco.

Ringrazio Manuel per avermi sempre dato consigli vincenti e mirabolanti appunti.

Ringrazio Gian e Federico, perchè la competitività con loro mi ha sempre spinto a dare il massimo.

Ringrazio Neri e Zanzi, per essere stati bravissimi compagni di stanza oltre che di avventure.

Ringrazio Bracalo e Menga per esser stati repentini nel celebrare con me la consegna della tesi.

Sommario

L'ambito dove si colloca il lavoro svolto è quello del controllo di traiettorie per robot mobili.

Più precisamente il controllo trattato è quello che viene chiamato controllo predittivo, o più brevemente MPC (Model Predictive Control).

Lo scopo della tesi è quello di applicare un controllo di tipo predittivo per ottenere un efficace inseguimento di traiettoria per il robot, che nel caso in questione è una sedia a rotelle autonoma chiamata LURCH (Let Unleashed Robot Crawl the House), presente nell'AirLab del Politecnico di Milano.

Abstract

The area this thesis is about is the trajectory control for mobile robots.

More precisely the control we deal with is the MPC (Model Predictive Control).

The aim of the thesis is to apply a predictive control to achieve an effective trajectory following for the robot, which in the specific case is an autonomous wheelchair called LURCH (Let Unleashed Robot Crawl the House), kept in the AirLab of Politecnico di Milano.

Indice

Introduzione **1**

1 Descrizione del problema affrontato **2**
1.1 Introduzione 2
1.2 Descrizione del sistema sperimentale [5] 2
1.2.1 Sedia a rotelle 2
1.2.2 Sensori 2
1.2.2.1 Scanner laser 2
1.2.2.2 Encoder 3
1.2.3 Computer 4
1.2.4 Touch screen 5
1.3 Motivazioni del lavoro 5
1.4 Presentazione del problema affrontato 6

2 Controllo MPC **8**
2.1 Introduzione 8
2.2 La feedback-linearizzazione del modello uniciclo 9
2.2.1 Equazioni dinamiche del modello uniciclo 9
2.2.2 Feedback linearizzazione rispetto ad un punto B 9
2.3 Applicazione della tecnica MPC all'inseguimento di traiettoria del robot 11
2.3.1 Formulazione del problema MPC come problema di ottimizzazione quadratica 12
2.4 Obstacle avoidance 14
2.4.1 Distanza dall'ostacolo 14
2.4.2 Algoritmo per definire i vincoli 15
2.4.3 Aggiunta dei vincoli alla formulazione del problema di ottimizzazione 16
2.4.4 Introduzione della slack variable 17

3 Implementazione **19**
3.1 Introduzione 19
3.2 Architettura ROS del sistema 19

3.2.1 Topic 20
3.2.2 Parameter server 20
3.2.3 Nodi 21
3.2.3.1 Nodo per il passaggio della traiettoria 22
3.2.3.2 Nodo del controllo 22
3.2.3.3 Nodo del simulatore 23
3.3 Matlab 23

4 Risultati di simulazione **24**

Conclusioni **37**

Bibliografia **38**

Elenco delle figure

1.1 Sedia a rotelle Rabbit 3
1.2 Scanner laser Hokuyo URG-04LX 3
1.3 Zotac ZBOX ID83 4
1.4 Sistema sperimentale completo 5
1.5 Passaggio al controllo MPC 7

2.1 Posizione del punto B nel piano 10
2.2 Vettori velocità utilizzati 10
2.3 Posizione relativa di robot e ostacolo 15

3.1 Architettura ROS: nodi e messaggi 21
3.2 Architettura ROS: Parameter Server 21

4.1 Test 1: Inseguimento di traiettoria 25
4.2 Test 1: Velocità, distanza e ritardo 25
4.3 Test 2: Inseguimento di traiettoria 26
4.4 Test 2: Velocità, distanza e ritardo 26
4.5 Test 3: Inseguimento di traiettoria 27
4.6 Test 3: Velocità, distanza e ritardo 27
4.7 Test 4: Inseguimento di traiettoria 28
4.8 Test 4: Velocità, distanza e ritardo 28
4.9 Test 5: Inseguimento di traiettoria 29
4.10 Test 5: Velocità, distanza e ritardo 29
4.11 Test 6: Inseguimento di traiettoria 30
4.12 Test 6: Velocità, distanza e ritardo 30
4.13 Test 7: Inseguimento di traiettoria 31
4.14 Test 7: Velocità, distanza e ritardo 31
4.15 Test 8: Inseguimento di traiettoria 32
4.16 Test 8: Velocità, distanza e ritardo 32
4.17 Test 9: Inseguimento di traiettoria 33
4.18 Test 9: Velocità, distanza e ritardo 33
4.19 Test 10: Inseguimento di traiettoria 34

4.20 Test 10: Velocità, distanza e ritardo 34
4.21 Test 11: Inseguimento di traiettoria 35
4.22 Test 11: Velocità, distanza e ritardo 35
4.23 Test su ambiente verosimile: Inseguimento di traiettoria 36
4.24 Test su ambiente verosimile: Velocità, distanza e ritardo 36

Elenco delle tabelle

4.1 Valori dei parametri utilizzati nei test 24

Introduzione

L'ambito dove si colloca il lavoro svolto è quello del controllo di traiettorie per robot mobili. Più precisamente il controllo trattato è quello che viene chiamato controllo predittivo, o più brevemente MPC (Model Predictive Control).

Lo scopo della tesi è quello di applicare un controllo di tipo predittivo per ottenere un efficace inseguimento di traiettoria per il robot, che nel caso in questione è una sedia a rotelle autonoma chiamata LURCH (Let Unleashed Robot Crawl the House), presente nell'AirLab del Politecnico di Milano.

A tal scopo dunque sono state studiate delle tesi precedenti riguardanti il controllo su robot modellizzati come unicicli [7], e riguardo l'attuale implementazione del controllo su LURCH [5].

Dopodichè utilizzando il framework già utilizzato su LURCH, ovvero un middleware ampiamente utilizzato in applicazioni robotiche chiamato ROS (Robot Operating System), è stato implementato un controllo MPC per il *trajectory following* e l'*obstacle avoidance* di un robot modellizzato come uniciclo.

Infine sono stati effettutati dei test di simulazione di traiettorie sugli algoritmi implementati in ROS, poi portati in Matlab per facilitare l'operazione di testing.

La tesi è strutturata nel modo seguente.

Nella prima sezione si descrive il problema affrontato, soffermandosi sul sistema sperimentale coinvolto e sulle motivazioni che hanno portato alla decisione di intraprendere il lavoro

Nella seconda sezione si parla nel dettaglio della teoria del controllo applicato, quindi sulle trasformazioni matematiche delle variabili da adottare ed in generale di tutte le conoscenze necessarie per procedere, in seguito, all'implementazione.

Nella terza parte si tratta proprio dell'implementazione. In particolar modo del framework, dei linguaggi di programmazione utilizzati e dell' architettura del sistema informatico.

Nell'ultima parte sono esposti dei risultati grafici di test degli algoritmi di controllo implementati.

Capitolo 1

Descrizione del problema affrontato

1.1 Introduzione

Si presenta in questo capitolo il problema affrontato.

Prima di presentare il problema tuttavia si parla del sistema sperimentale a disposizione, ovvero della sedia a rotelle autonoma presente nell'AirLab del Politecnico di Milano chiamata LURCH(Let Unleashed Robot Crawl the House), e delle motivazioni che hanno portato al lavoro svolto.

1.2 Descrizione del sistema sperimentale [5]

1.2.1 Sedia a rotelle

La sedia usata come base per lo sviluppo del sistema si chiama Rabbit, ed è prodotta dalla compagnia tedesca Otto Bock. E' una sedia utilizzabile per ambienti sia indoor che outdoor, dal momento che possiede due grandi ruote posteriori con pneumatici tassellati.

Il movimento avviene grazie a due motori indipendenti che operano su ciascuna delle ruote posteriori, mentre le ruote anteriori sono libere di girare e non sono controllabili. I motori sono di circa 200W ciascuno e sono alimentati da due batterie da 12V - 70Ah collegate in serie e posizionate sotto il sedile.

1.2.2 Sensori

1.2.2.1 Scanner laser

Per percepire l'ambiente in maniera precisa e affidabile, il robot è equipaggiato con due scanner laser Hokuyo URG-04LX.

Figura 1.1: Sedia a rotelle Rabbit

Questi sensori sono capaci di percepire ostacoli su un piano con range di campo visivo di 240ř ed una risoluzione di 0.36ř. La massima distanza rilevabile è 5.36 m.

I due laser sono montati sui lati del poggiapiedi. Per massimizzare il range angolare di scansione, i due sensori sono posizionati a circa 90ř rispetto all'asse longitudinale del robot. In questo modo l'area centrale antestante la sedia a rotelle è coperta da ambedue i sensori laser, aggiungendo così robustezza al rilevamento di ostacoli: se uno scanner non funziona, l'altro manterrà un minimo livello di sicurezza.

Figura 1.2: Scanner laser Hokuyo URG-04LX

1.2.2.2 Encoder

La sedia è inoltre equipaggiata con due encoder, uno per ogni ruota motrice.

Questi sensori sono capaci di misurare il numero di giri completati dalle ruote, espressi in *tick* (sezioni rilevabili di un giro), al fine di estrarre informazioni sull'odometria.

L'interfaccia tra gli encoder ed il computer è fatta con una scheda elettronica, chiamata *odometry board*, che manda messaggi con una frequenza di 50 Hz, specificando:

- un contatore rappresentante il numero dei messaggi mandati dall'inizio
- il numero totale di *tick* della ruota sinistra dall'inizio
- il numero totale di *tick* della ruota destra dall'inizio

La comunicazione è possibile mediante una connessione seriale.

1.2.3 Computer

Il computer elabora i segnali dei sensori per estrarre informazioni a riguardo dell'ambiente e della posizione del robot. Inoltre permette i comportamenti intelligenti quali la navigazione autonoma e l' *obstacle avoidance*, e automaticamente manda comandi alla scheda dei motori.

Si tratta di uno Zotac ZBOX ID83 equipaggiato con un processore Intel Core i3 3120M con due core a 2.5 GHz.

Per quanto riguarda le caratteristiche di memoria ha 4 GB di RAM e un SSD da 64 GB.

Per quanto riguarda la parte grafica è fornito di una scheda video integrata Intel HD 4000, che supporta OpenGL 4.0.

Il sistema operativo *on-board* è Ubuntu-Linux.

Figura 1.3: Zotac ZBOX ID83

1.2.4 Touch screen

Per fornire un feedback visivo o per eventualmente utilizzare comandi è presente anche un touch screen. Il monitor, Xenarc 700YYV, ha un display da 7 con una risoluzione 800x480, e speaker integrati.

Figura 1.4: Sistema sperimentale completo, ovvero LURCH

1.3 Motivazioni del lavoro

Illustriamo di seguito come attualmente avviene il controllo della velocità su LURCH, passando attraverso i suoi componenti principali: Il pianificatore locale di ROS, il controllore PID ed il meccanismo di *obstacle avoidance*, evidenziando a fine di ciascun paragrafo i lati negativi di tali scelte progettuali.

Base local planner [4] Il *base_local_planner* package di ROS fornise un controllore che guida una base mobile in un piano. Questo controllore serve a connettere il *local planner* del percorso al robot. Usando una mappa, il *local planner* crea una traiettoria cinematica per il robot per andare da un punto di partenza ad uno di arrivo e nel contempo crea, almeno localmente intorno al robot, una funzione di costo rappresentata da una griglia. Il ruolo del controllore è di usare questa funzione per determinare le velocità lungo gli assi e la velocità angolare da mandare al robot.

L'idea di base è la seguente:

1. Si campiona discretamente lo spazio di controllo del robot
2. Per ogni velocità campionata, si simula in avanti per prevedere cosa succederebbe al robot se assumesse quella velocità per un certo periodo di tempo

3. Si valuta la funzione di costo per ogni traiettoria simulata, includendo nella cifra di costo la vicinanza alla traiettoria stabilita, la vicinanza al punto di arrivo della traiettoria stabilita e la vicinanza di ostacoli.

4. Si prende la traiettoria che rende la funzione di costo minima e la si usa come ingresso per il robot

La mancanza di efficacia nell'utilizzare il *base_ local_ planner* consiste nel fatto che il riferimento di velocità viene stabilito senza tenere conto della traiettoria da seguire.

Controllore PID [5] Il controllore PID riceve in ingresso l'errore tra il *setpoint* di velocità generato dal *local planner* e la misura sul sistema reale, e genera le variabili di controllo adatte per portare la velocità al *setpoint*.

La misura del profilo velocità del sistema non è tuttavia troppo fedele vista la scarsa risoluzione degli encoder che portano ad una funzione del tempo tutt'altro che morbida. Questo rende necessario l'utilizzo di un feedback a media mobile su un prefissato numero di campioni.

Obstacle avoidance [5] La funzione di costo del *local planner* già di per sè cerca di mantenere la traiettoria del robot lontana da ostacoli. Tuttavia per gestire i casi in cui l'ostacolo non fosse aggiunto in tempo alla funzione di costo o per un malfunzionamento la sedia a rotelle dovesse trovarsi in rotta di collisione con un ostacolo esiste anche un meccanismo di emergenza.

Si trova logicamente tra il planner ed il PID e funziona semplicemente in questa maniera: se esiste un ostacolo al di sotto di una certa distanza nella direzione del *setpoint* di velocità passato dal *local planner*, il *setpoint* di velocità viene mandato a 0, altrimenti resta invariato.

1.4 Presentazione del problema affrontato

Il problema del controllo di velocità implementato attualmente su LURCH sta nel fatto che lo schema non tiene conto della traiettoria che il robot deve seguire. Si basa essenzialmente sulla velocità misurata ed sull riferimento della velocità ad ogni istante, senza tenere conto della traiettoria desiderata.

L'idea per migliorare il controllo e' di sostituire il controllo di velocità attuale, basato principalmente sul package ROS *base_ local_ planer,* con un controllo predittivo, altresì chiamato MPC.

Il controllo MPC permetterebbe di ottenere ad ogni istante variabili di controllo ottimizzate per l'inseguimento della traiettoria e non necessiterebbe di un feedback di velocità per il funzionamento.

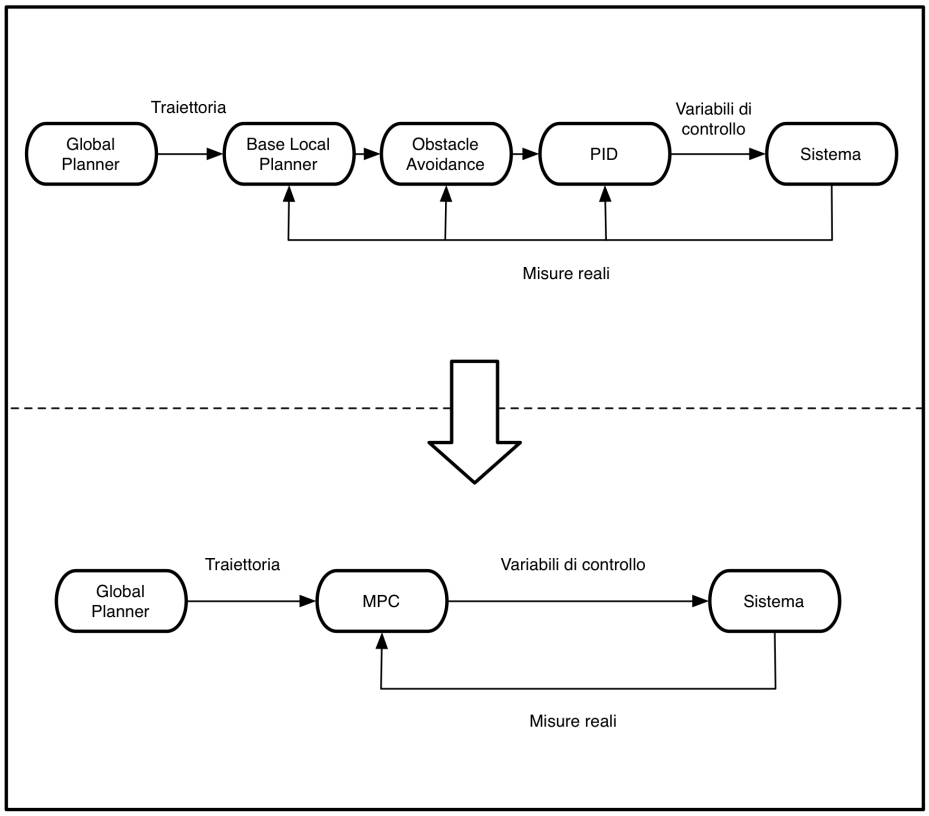

Figura 1.5: Passaggio logico dal controllo attuale con il *base local planner* standard ed il PID al controllo MPC

Inoltre grazie alla sua forma di problema di ottimizzazione matematica, è possibile impostare l'*obstacle avoidance* imponendo semplicemente dei vincoli sul problema di ottimizzazione.

Capitolo 2

Controllo MPC

2.1 Introduzione

La tecnica MPC, anche chiamata controllo predittivo, è una tecnica di controllo ampiamente utilizzata in diversi contesti.

Questa tecnica consiste nel formulare il problema di controllo come un problema di ottimizzazione matematica costruito sul modello dinamico del sistema da controllare, ed in particolare sulla predizione dell'evoluzione del sistema in certo orizzonte futuro.

Il problema di ottimizzazione viene poi risolto online a ciascun istante di campionamento, ottenendo la sequenza di ingressi ottimi da fornire al sistema.

Congiuntamente alla tecnica MPC viene sfruttato il principio *Receiding Horizon*. Grazie a questo principio ad ogni istante di tempo l'orizzonte predittivo viene fatto scorrere in avanti, ed il problema di controllo ottimo su orizzonte finito è risolto nuovamente sulla base di una nuova misura dello stato reale.

Un gran vantaggio della tecnica MPC consiste nel poter includere direttamente vincoli sulle variabili di stato e sulle variabili di controllo nel problema di ottimizzazione.

La tecnica MPC può essere applicata sia a sistemi lineari che non lineari [6].

Nel nostro caso il sistema da controllare, modellizzato tramite il modello uniciclo, è di tipo non lineare. Tuttavia nel caso lineare il problema di ottimizzazione può essere tradotto in un problema quadratico, risultando così di più agevole soluzione e di minore complessità computazionale, principale limite della tecnica MPC. Per questo motivo il sistema viene linearizzato mediante una tecnica chiamata *feedback-linearizzazione*, in quanto sfrutta nella linearizzazione un feedback della misura reale del sistema.

In questo capitolo si parlerà dapprima della linearizzazione da applicare al sistema, quindi della tecnica MPC applicata al sistema linearizzato, e infine dei vincoli da imporre al problema di ottimizzazione per l'*obstacle avoiding*.

2.2 La feedback-linearizzazione del modello uniciclo

2.2.1 Equazioni dinamiche del modello uniciclo

Prima di parlare del controllo esaminiamo il modello del sistema da controllare. Si tratta di quello che viene chiamato modello uniciclo.

Le equazioni del modello uniciclo sono

$$\begin{aligned}\dot{x}(t) &= v(t) \cdot cos(\theta(t)) \\ \dot{y}(t) &= v(t) \cdot sen(\theta(t)) \\ \dot{\theta}(t) &= \omega(t)\end{aligned}$$

dove le variabili di stato $x(t)$, $y(t)$, $\theta(t)$ rappresentano l'ascissa, l'ordinata e l'orientamento rispetto alle asse delle ordinate del punto medio dell'asse delle ruote della sedia a rotelle, mentre le variabili di controllo $v(t)$ e $\omega(t)$ rappresentano la velocità lineare e la velocità angolare del suddetto punto.

Ricordiamo che nel caso di guida differenziale le variabili di controllo possono essere espresse in funzione delle velocità delle singole ruote.

$$\begin{aligned}v(t) &= \frac{v_r(t) + v_l(t)}{2} \\ \omega(t) &= \frac{v_r(t) - v_l(t)}{l}\end{aligned}$$

dove $v_r(t)$ rappresenta la velocità della ruota destra, $v_l(t)$ la velocità della ruota sinistra, l la distanza tra le due ruote.

2.2.2 Feedback linearizzazione rispetto ad un punto B

Scegliendo un punto B a distanza $b > 0$ dal centro dell'asse delle ruote posto sulla retta su cui giace il vettore velocità $v(t)$.

La posizione del punto B è data da

$$\begin{aligned}x_B(t) &= x(t) + b \cdot cos(\theta(t)) \\ y_B(t) &= y(t) + b \cdot sen(\theta(t))\end{aligned}$$

Derivando rispetto al tempo si ottiene

$$\begin{aligned}\dot{x}_B(t) &= \dot{x}(t) - b \cdot \dot{\theta}(t) \cdot sen(\theta(t)) \\ &= v \cdot cos(\theta) - b \cdot \omega(t) \cdot sen(\theta(t)) \\ \dot{y}_B(t) &= \dot{y}(t) - b \cdot \dot{\theta}(t) \cdot cos(\theta(t)) \\ &= v \cdot sin(\theta(t)) + b \cdot \omega(t) \cdot cos(\theta(t))\end{aligned}$$

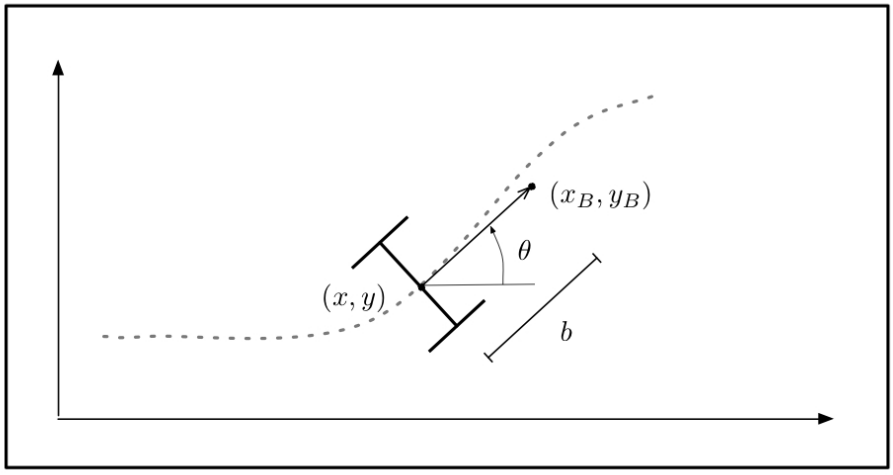

Figura 2.1: Posizione del punto B nel piano

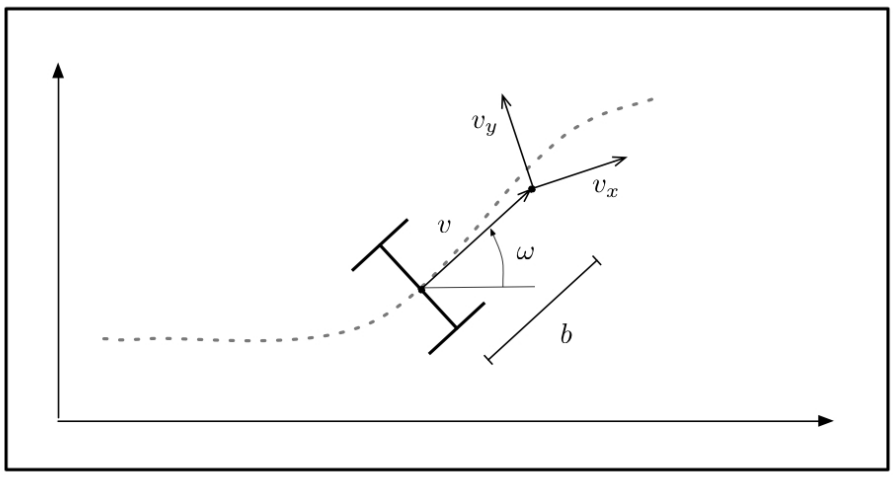

Figura 2.2: Vettori velocità utilizzati

Si introduce il cambiamento di variabili

$$v_x(t) = v(t) \cdot cos(\theta(t)) - b \cdot \omega(t) \cdot sen(\theta(t))$$
$$v_y(t) = v(t) \cdot sin(\theta(t)) + b \cdot \omega(t) \cdot cos(\theta(t))$$

ovvero

$$\begin{bmatrix} v_x(t) \\ v_y(t) \end{bmatrix} = \begin{bmatrix} cos(\theta(t)) & -b \cdot sen(\theta(t)) \\ sen(\theta(t)) & b \cdot cos(\theta(t)) \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix}$$

dove la matrice

$$T(\theta(t)) = \begin{bmatrix} cos(\theta(t)) & -b \cdot sen(\theta(t)) \\ sen(\theta(t)) & b \cdot cos(\theta(t)) \end{bmatrix}$$

è non singolare $\forall\theta$ a patto che $b > 0$.

La relazione inversa è quindi

$$\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} cos(\theta(t)) & sen(\theta(t)) \\ -\frac{sen(\theta(t))}{b} & \frac{cos(\theta(t))}{b} \end{bmatrix} \begin{bmatrix} v_x(t) \\ v_y(t) \end{bmatrix}$$

Effettuato questo cambiamento di variabili il sistema diventa lineare, ovvero

$$\dot{x_B}(t) = v_x(t)$$
$$\dot{y_B}(t) = v_y(t)$$

2.3 Applicazione della tecnica MPC all'inseguimento di traiettoria del robot

Le equazioni dinamiche prese in considerazione per il controllo, discretizzate a passo di campionamento τ, diventano

$$x_B(t+1) = x_B(t) + \tau \cdot v_x(t)$$
$$y_B(t+1) = y_B(t) + \tau \cdot v_y(t)$$

che riconduciamo all'equazione vettoriale

$$\xi(t+1) = A\xi(t) + Bu(t)$$

dove

$$\xi(t) = \begin{bmatrix} x_B(t) \\ y_B(t) \end{bmatrix}, \; u(t) = \begin{bmatrix} v_x(t) \\ v_y(t) \end{bmatrix}$$
$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \; B = \begin{bmatrix} \tau & 0 \\ 0 & \tau \end{bmatrix}$$

Introducendo i vettori di riferimento per le posizioni future e le velocità future, rispettivamente $\xi_{RIF}(t)$ e $u_{RIF}(t)$, la cifra di merito che vogliamo minimizzare è

$$J = \sum_{k=t}^{t+N-1} \left(\|\xi(k) - \xi_{RIF}(k)\|_Q^2 + \|u(k) - u_{RIF}(k)\|_R^2 \right) + \|\xi(t+N) - \xi_{RIF}(t+N)\|_S^2$$

dove l'intero positivo N è l'orizzonte di predizione, mentre $Q = Q^T \geq 0$, $R = R^T > 0$, $P = P^T \geq 0$ sono matrici di dimensioni opportune che pesano rispettivamente lo stato, il controllo e lo stato finale. Nel caso in questione scegliamo le matrici tutte e tre diagonali

$$Q = \begin{bmatrix} q & 0 \\ 0 & q \end{bmatrix} \ , \ R = \begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix} \ , \ P = \begin{bmatrix} p & 0 \\ 0 & p \end{bmatrix}$$

Avendo scelto Q ed R la matrice P è scelta in relazione alle altre utilizzando il metodo del *quasi infinite horizon*, ovvero si determina il controllo LQ su tempo infinito con legge di controllo

$$u(t) = -Kx(t)$$

e la matrice P è la soluzione dell'equazione stazionaria di Riccati

$$(A - BK)^T P (A - BK) - P = -\left(Q + K^T R K\right)$$

dove il guadagno di Kalman K nel caso in questione viene scelto con un banale assegnamento di autovalori

$$K = \begin{bmatrix} \frac{1}{2\tau} & 0 \\ 0 & \frac{1}{2\tau} \end{bmatrix}$$

Quindi la matrice P risulta

$$P = \begin{bmatrix} \frac{4}{3}\left(q + \frac{r}{2\tau^2}\right) & 0 \\ 0 & \frac{4}{3}\left(q + \frac{r}{2\tau^2}\right) \end{bmatrix}$$

2.3.1 Formulazione del problema MPC come problema di ottimizzazione quadratica

Dovendo fare in modo che l'ottimizzazione possa essere inserita nel problema di minimizzazione quadratica del solutore

$$min\ U_t^T H U_t + 2 \cdot f^T U_t + cost$$

dobbiamo riscrivere la funzione

$$J = \sum_{k=t}^{t+N-1} \left(\|\xi(k) - \xi_{RIF}(k)\|_Q^2 + \|u(k) - u_{RIF}(k)\|_R^2 \right) + \|\xi(t+N) - \xi_{RIF}(t+N)\|_S^2$$

in maniera compatta:

$$J = \left\| \Xi_t - \Xi_t^{RIF} \right\|_{\mathcal{Q}}^2 + \left\| U_t - U_t^{RIF} \right\|_{\mathcal{R}}^2$$

dove

$$\Xi_t = \begin{bmatrix} \xi(t) \\ \vdots \\ \xi(t+N) \end{bmatrix} \ , \ U_{t=} \begin{bmatrix} u(t) \\ \vdots \\ u(t+N-1) \end{bmatrix}$$

$$\mathcal{Q} = \begin{bmatrix} Q & & & 0 \\ & \ddots & & \\ & & Q & \\ 0 & & & P \end{bmatrix} \ , \ \mathcal{R} = \begin{bmatrix} R & & 0 \\ & \ddots & \\ 0 & & R \end{bmatrix}$$

Sfruttando l'equazione del sistema dinamico

$$\xi(t+1) = A\xi(t) + Bu(t)$$

riscriviamo Ξ_t come

$$\Xi_t = \underbrace{\begin{bmatrix} I \\ A \\ A^2 \\ \vdots \\ A^N \end{bmatrix}}_{\mathcal{A}} \xi(t) + \underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 \\ B & 0 & \cdots & 0 \\ AB & B & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ A^{N-1}B & A^{N-2}B & \cdots & B \end{bmatrix}}_{\mathcal{B}} U_t$$

Sostituendo nella cifra di merito si ha

$$\begin{aligned} J &= \left\| \mathcal{A}\xi(t) + \mathcal{B}U_t - \Xi_t^{RIF} \right\|_{\mathcal{Q}}^2 + \left\| U_t - U_t^{RIF} \right\|_{\mathcal{R}}^2 \\ &= \left(\mathcal{A}\xi(t) + \mathcal{B}U_t - \Xi_t^{RIF} \right)^T \mathcal{Q} \left(\mathcal{A}\xi(t) + \mathcal{B}U_t - \Xi_t^{RIF} \right) + \left(U_t - U_t^{RIF} \right)^T \mathcal{R} \left(U_t - U_t^{RIF} \right) \\ &= U_t^T \left(\mathcal{B}^T \mathcal{Q}\mathcal{B} + \mathcal{R} \right) U_t + 2 \cdot \left(\left(A\xi(t) - \Xi_t^{RIF} \right) \mathcal{Q}\mathcal{B} - U_t^{RIF}\mathcal{R} \right) U_t + cost \end{aligned}$$

quindi le matrici H ed f^T per il solutore risultano essere

$$H = \left(\mathcal{B}^T \mathcal{Q}\mathcal{B} + \mathcal{R} \right)$$

$$f^T = \left(\left(A\xi(t) - \Xi_t^{RIF} \right) \mathcal{Q}\mathcal{B} - \left(U_t^{RIF} \right)^T \mathcal{R} \right)$$

Vincoli sull'attuazione Al problema di ottimizzazione si possono aggiungere vincoli sulle variabili.

Nel caso in questione è utile aggiungere un vincolo sulle velocità U_t. Nello specifico è utile limitare in modulo ogni elemento del vettore U_t al di sotto di una certa velocità massima V_{max}, onde evitare che per qualsiasi ragione il problema di ottimizzazione dia soluzioni in velocità non realmente raggiungibili dalla sedia a rotelle.

Quindi ad ogni istante t, per $i = 0, ..., N-1$

$$\begin{aligned} v_x\,(t+i) &\leq V_{max} \\ v_y\,(t+i) &\leq V_{max} \\ -v_x\,(t+i) &\leq V_{max} \\ -v_y\,(t+i) &\leq V_{max} \end{aligned}$$

2.4 Obstacle avoidance

E' necessario aggiungere al problema di *trajectory following* anche quello di *obstacle avoidance*, cioè bisogna far si che nell'inseguire la traiettoria il robot non incorra in collisioni.

E' possibile raggiungere tale scopo immettendo degli ulteriori vincoli allo stesso problema di ottimizzazione utilizzato per il *trajectory following*, di modo da non aumentare eccessivamente l'onere computazionale dell'algoritmo di controllo.

Per semplicità gli ostacoli considerati saranno politopi regolari di n lati, circoscritti a una circonferenza di raggio r e centro (x_c, y_c). Questa scelta non toglie generalità al problema in quanto qualsiasi ostacolo può essere inscritto in questi politopi.

Sempre per semplicità viene anche adottata l'ipotesi di ostacoli fissi ed in posizione nota.

2.4.1 Distanza dall'ostacolo

Per definire la distanza dall'ostacolo prendiamo in considerazione la distanza da ogni lato, intesa più precisamente come distanza tra il punto corrispondente al centro dell'asse delle ruote della sedia a rotelle e la retta costruita sul lato del politopo circoscritto.

Essendo la rette passanti per i lati del politopo

$$(y - y_c)\,sen\,(\theta_i) + (x - x_c)\,cos\,(\theta_i) - r = 0\ ,\ \theta_i = \frac{2\pi}{n} \cdot i\ ,\ i = 1, ..., n$$

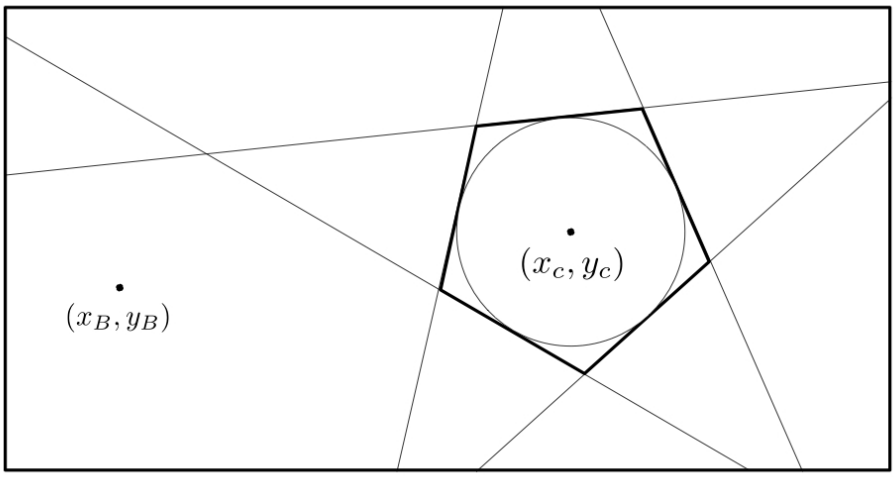

Figura 2.3: Posizione relativa di robot e ostacolo

il vettore con le distanze (normalizzate rispetto a r) del punto $\xi(t)$ è

$$\rho(t) = \underbrace{\begin{bmatrix} \frac{cos(\theta_1)}{r} & \frac{sen(\theta_1)}{r} \\ \vdots & \vdots \\ \frac{cos(\theta_n)}{r} & \frac{sen(\theta_n)}{r} \end{bmatrix}}_{D} \cdot (\xi(t) - \xi_c) - \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

dove $\xi_c = \begin{bmatrix} x_c \\ y_c \end{bmatrix}$.

Gli elementi di $\rho(t)$ maggiori di zero corrisponderanno ai lati per cui $\xi(t)$ si trova dalla parte opposta rispetto a ξ_c, al contrario elementi di $\rho(t)$ minori di zero corrisponderanno ai lati per cui $\xi(t)$ e ξ_c si trovano nello stesso semipiano tra i due divisi dalla retta giacente sul lato in questione.

2.4.2 Algoritmo per definire i vincoli

Ad ogni istante di tempo l'utilizzo della tecnica MPC con il principio *Receiding Horizon* ci fornisce predizioni sullo stato negli N istanti futuri $\xi(t+1|t), ..., \xi(t+N|t)$. Al tempo $t-1$ quindi disponiamo delle predizioni $\xi(t|t-1), ..., \xi(t+N-1|t-1)$ che possiamo utilizzare per calcolare i corrispettivi vettori ρ:

$$\rho(t|t-1) = D(\xi(t|t-1) - \xi_c) - \vec{1}$$
$$\vdots$$
$$\rho(t+N-1|t-1) = D(\xi(t+N-1|t-1) - \xi_c) - \vec{1}$$

dove $\vec{1}$ è una colonna di n elementi uguali a 1.

Se all'istante $t-1$ il punto $\xi(t)$ si trova al di fuori del politopo che racchiude l'ostacolo allora almeno uno degli elementi di ciascun vettore $\rho(k|t-1)\ ,\ k=t-1,...,t+N-1$ sarà maggiore di zero. Per ciascun vettore tra tutti gli elementi se ne sceglie il maggiore:

$$\rho_{\bar{i}}(k|t-1) = d_{\bar{i}}(k|t-1)(\xi(k|t-1)-\xi_c)-1\ ,\ k=t-1,...,t+N-1$$

dove

$$\bar{i} = argmax_{i=1,...,n}\rho_{\bar{i}}(k|t-1)-1\ ,\ k=t-1,...,t+N-1$$

e $d_{\bar{i}}(k|t-1)$ è la riga corrispondente all'indice $\bar{i}(k|t-1)$ della matrice D.

Qualitativamente l'indice $\bar{i}(k|t-1)$ è interpretabile come il lato del politopo lungo il quale muoversi all'istante k.

Quindi all'istante t i vincoli da imporre sono

$$\begin{aligned}
d_{\bar{i}}(t|t-1)(\xi(t|t-1)-\xi_c) &\geq 1\\
d_{\bar{i}}(t|t-1)(\xi(t+1|t-1)-\xi_c) &\geq 1\\
&\vdots\\
d_{\bar{i}}(t+N-1|t-1)(\xi(t+N-1|t-1)-\xi_c) &\geq 1\\
d_{\bar{i}}(t+N-1|t-1)(\xi(t+N-1|t-1)-\xi_c) &\geq 1
\end{aligned}$$

Da notare che l'ultima riga viene ripetuta non avendo a disposizione la previsione per l'istante $t+N$, ed avendo necessità che anche all'istante $t+N$ il vettore di stato si trovi all'esterno delle delimitazioni dell'ostacolo.

2.4.3 Aggiunta dei vincoli alla formulazione del problema di ottimizzazione

Per introdurre i vincoli di obstacle avoidance all'interno del problema di ottimizzazione è necessario esprimerli esplicitando il vettore argomento U_t.

Ricordando che

$$\Xi_t = \mathcal{A}\xi(t) + \mathcal{B}U_t$$

possiamo scrivere

$$\bar{D}\left(\mathcal{A}\xi(t) + \mathcal{B}U_t - \begin{bmatrix}\xi_c\\ \vdots\\ \xi_c\end{bmatrix}\right) - \begin{bmatrix}1\\ \vdots\\ 1\end{bmatrix} \geq 0$$

dove

$$\bar{D} = \begin{bmatrix} d_{\bar{i}}\left(t|t-1\right)\left(\xi(t|t-1) - \xi_c\right) \\ d_{\bar{i}}\left(t|t-1\right)\left(\xi(t+1|t-1) - \xi_c\right) \\ \vdots \\ d_{\bar{i}}\left(t+N-1|t-1\right)\left(\xi(t+N-1|t-1) - \xi_c\right) \\ d_{\bar{i}}\left(t+N-1|t-1\right)\left(\xi(t+N-1|t-1) - \xi_c\right) \end{bmatrix}$$

Con le necessarie operazioni per mettere in evidenza il vettore argomento U_t

$$\underbrace{\left(-\bar{D}\mathcal{B}\right)}_{A_{obst}} U_t \leq \underbrace{-\bar{D}\begin{bmatrix} \xi_c \\ \vdots \\ \xi_c \end{bmatrix} + \bar{D}\mathcal{A}\xi\left(t\right) - \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}}_{b_{obst}}$$

2.4.4 Introduzione della slack variable

Può capitare che dopo l'aggiunta dei vincoli sugli ostacoli si possa andare incontro a problemi di *infeasibility*. Infatti nel caso in cui il robot si sposti in prossimità degli ostacoli potrebbe capitare che per la non-linearità del sistema o per la non-idealità nel calcolo di integrazione la traiettoria prevista si trovi all'interno dell'area delimitata dai vincoli degli ostacoli.

Per evitare problemi di questo tipo possiamo aggiungere una variabile chiamata *slack variable*, la quale ha lo scopo di rilassare i vincoli in situazioni limite e quindi far si che il problema sia comunque *feasible*.

Inoltre pesando la variabile in maniera opportuna non si incorre nel rischio di collisioni con gli ostacoli, se il raggio della circonferenza inscritta al politopo che racchiude l'ostacolo è sufficientemente più grande dell'ostacolo reale.

Il nuovo funzionale di costo per l'ottimizzazione è così modificato:

$$\bar{J}\left(t\right) = J\left(t\right) + \Gamma s^2$$

dove s è la *slack variable*, che rappresenta una nuova variabile decisionale, e Γ il suo peso nella funzione di costo.

Le matrici per la risoluzione del problema di ottimizzazione devono di conseguenza essere modificate:

$$\bar{H} = \begin{bmatrix} & & & 0 \\ & H & & \vdots \\ & & & 0 \\ 0 & \cdots & 0 & \Gamma \end{bmatrix}$$

$$\bar{f}^T = \begin{bmatrix} f^T & 0 \end{bmatrix}$$

E anche i vincoli riguardanti gli ostacoli devono essere modificati, aggiungendo la *slack variable* al termine di destra:

$$A_{obst}U_t \leq b_{obst} + s \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

E' importante ricordare che la *slack variable* deve essere non-negativa, dunque è necessario aggiungere al problema di ottimizzazione anche un vincolo che imponga

$$s \geq 0$$

Capitolo 3

Implementazione

3.1 Introduzione

Per l'implementazione è stato scelto di usare il framework ROS, un framework ampiamente utilizzato nell'ambito della robotica.

ROS (Robot Operating System) è un sistema operativo open-source per robots, sviluppato dallo Stanford Artificial Intelligence Laboratory. Più precisamente ROS è un *middleware*, in quanto fornisce uno strato di comunicazione su un sistema operativo ospite Linux.

Grazie al suo paradigma di *publish-and-subscribe* e alla alta portabilità, ROS garantisce un'ottima evolvibilità e riusabilità dei suoi moduli software. [5]

Come linguaggio di programmazione per i moduli di ROS è stato utilizzato Python, linguaggio di alto livello piuttosto adatto ad interfacciarsi con ROS mediante la libreria *rospy*.

Per quanto riguarda il solutore per il problema di ottimizzazione dell'MPC è stato utilizzato CPLEX, un programma di ottimizzazione della IBM che fornisce ampia documentazione sulle API per Python.

Dopo l'implementazione in ROS, ai fini di facilitare il debbugging e il testing l'implementazione ROS è stata portata in Matlab.

In questo capitolo si parlerà dell'architettura ROS del sistema, e poi dell'implementazione per i test in Matlab.

3.2 Architettura ROS del sistema

Prima di spiegare l'architettura ROS del sistema in questione illustriamo brevemente qualche concetto base di ROS [4]:

- Nodo: un nodo è un eseguibile che usa ROS per comunicare con altri nodi. In una tipica applicazione robotica ogni nodo è responsabile di uno specifico *task*, spesso relativo ad un particolare componente hardware.

- Messaggi: sono dei ROS data type usati al momento del *subscribing* o *publishing* di un topic.
  Supportano tipi primitivi come integers, float, ecc... e strutture dati più complesse date dalla combinazione di questi.

- Topic: i nodi possono pubblicare messaggi a un topic oltre che iscriversi a un topic per ricevere messaggi.
  In generale più nodi possono pubblicare messaggi sullo stesso topic e sottoscrivere allo stesso topic.

- Parameter Server: il *Parameter Server* può memorizzare integers, floats, boolean, dictionaries, e liste.
  Qualsiasi nodo può attingere dal *Parameter Server* per ricevere dati o anche modificarne esistenti e crearne di nuovi.

3.2.1 Topic

I topic utilizzati nell'architettura ROS sono tre: /trj, /v_omega e /x_y_theta.

/trj E' il topic su cui vengono pubblicati dei messaggi di tipo /nav_msg/Path.msg [4] , contenenti la traiettoria da inseguire.

In particolare la traiettoria da seguire è una struttura dati composta da ascissa, ordinata, velocità e tempo per ogni punto della traiettoria.

/x_y_theta E' il topic su cui vengono pubblicate le misure reali del sistema x, y e θ, sottoforma di un array di tre float.

/v_omega E' il topic su cui vengono pubblicate le variabili di controllo v ed ω, sfruttando il tipo di messaggio geometry_msg/Twist.msg [4] .

3.2.2 Parameter server

Il *Parameter Server* viene solitamente utilizzato per passare ai nodi parametri che non variano durante l'esecuzione ma che può far comodo cambiare senza addentrarsi nel codice.

Nel nostro caso la maggiore utilità dell'archiviare i parametri nel parameter server sta nel momento in cui bisogna studiare la taratura dei parametri dell'MPC, come ad esempio

- la distanza di *feedback-linearizzazione* b
- il passo di campionamento τ
- l'orizzonte di predizione N

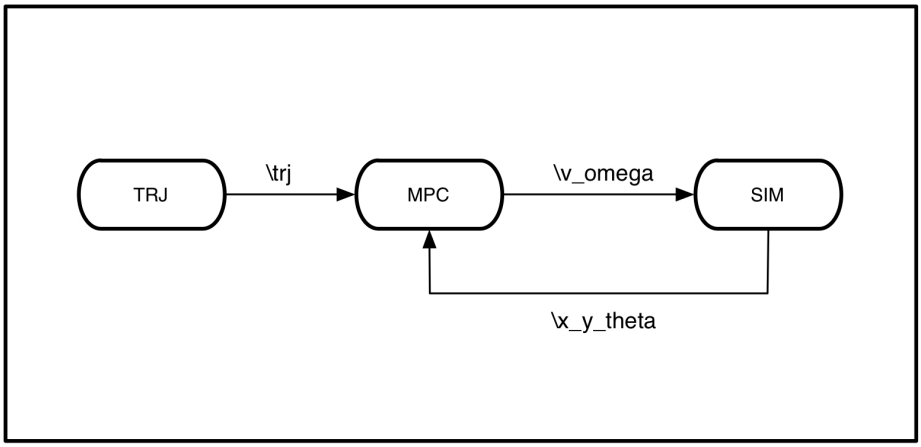

Figura 3.1: Architettura ROS: nodi e messaggi

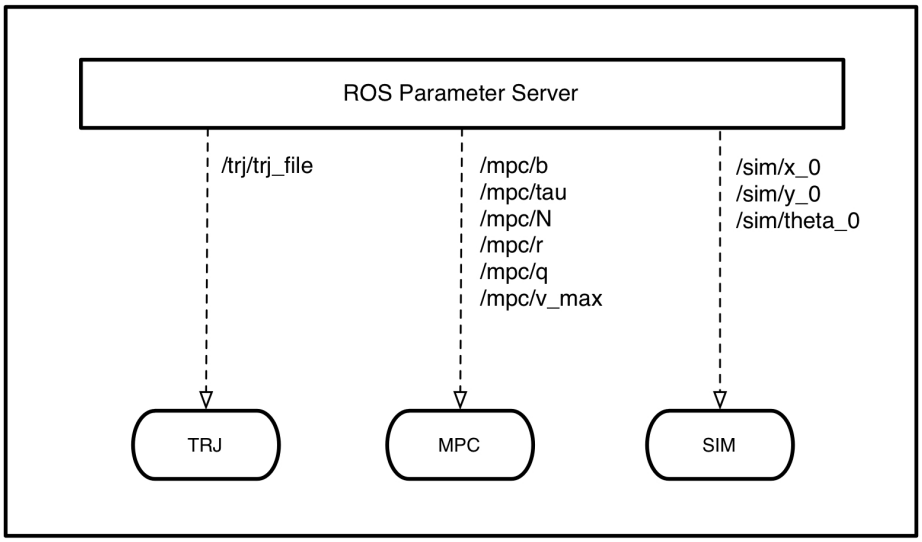

Figura 3.2: Architettura ROS: Parameter Server

- gli elementi delle diagonali delle matrici di peso per la posizione e la velocità q ed r
- il limite sull'attuazione V_{max}

Inoltre tiene in archivio stringhe che rimandano a file contenti traiettorie e la posizione iniziale della sedia.

3.2.3 Nodi

I topic utilizzati nell'architettura ROS sono tre, e si occupano del passaggio della traiettoria, del controllo e del simulatore.

3.2.3.1 Nodo per il passaggio della traiettoria

Il nodo che nelle figure viene nominato TRJ è un nodo che ha l'utilità di sostituire quello che nella realtà dovrebbe essere il pianificatore globale che crea la traiettoria, per testare il nodo MPC del controllo.

Infatti la funzione che svolge all'interno del sistema implementato è quello di ricevere dal *Parameter Server* una stringa con il nome di un file di testo contenente i dati sulla traiettoria da inseguire, impacchettare i dati nella maniera corretta e pubblicarli sul topic /trj.

3.2.3.2 Nodo del controllo

Il nodo del controllo, chiamato nelle figure MPC, è la parte principale dell'implementazione.

Sottoscrive a due topic: /trj e /x_y_theta. Di conseguenza contiene due callback corrispondenti alle due differenti sottoscrizioni. Riportiamo di seguito le operazione svolte dalle due callback.

Callback alla ricezione di un nuovo messaggio da /trj Come detto in precedenza /trj gestisce messaggi di tipo /nav_msg/Path.msg.

La callback per prima cosa estrae le informazioni dalla struttura dati del messaggio e le reinserisce in una matrice dove la prima dimensione è data da x, y, v e t, e la seconda dimensione dai punti della traiettoria.

Dopodichè vengono costruite delle spline cubiche per approssimare x, y e v rispetto ad un ascissa curvilinea funzione del tempo. Queste spline vengono poi utilizzate nell'altra callback, relativa all'anello di controllo, per trovare a ogni istante il punto della spline per costruire i vettori riferimento chiamati nel capitolo precedente Ξ^{REF} e U^{REF}.

Callback alla ricezione di un nuovo messaggio da /x_y_theta Alla ricezione di un nuovo messaggio da /x_y_theta, corrispondente ad una nuova posizione del robot sul piano, la callback per prima cosa trasforma le variabili come spiegato nella sezione della *feedback-linearizzazione* nel capitolo precedente.

Quindi ricerca il punto più vicino sulla spline della traiettoria nello spazio per utilizzarlo come $\xi^{REF}(t)$ e costruirvi su di esso il vettore di riferimento della posizione

$$\Xi^{REF} = \begin{bmatrix} \xi^{REF}(t) \\ \vdots \\ \xi^{REF}(t+N) \end{bmatrix}$$

e similmente U_t^{REF}, utilizzando come riferimento all'istante t una trasformazione coerente con la *feedback-linearizzazione* del punto della spline della velocità corrispondente alla stessa ascissa curvilinea di $\xi^{REF}(t)$.

Costruiti i riferimenti vengono costruite le matrici H ed f^T e usate come input per la risoluzione del problema di ottimizzazione, assieme ai vincoli per l'attuazione e per l'obstacle avoidance.

A problema risolto per gli N istanti successivi, le due soluzioni $v_x(t)$ e $v_y(t)$ vengono trasformate in $v(t)$ ed $\omega(t)$ come spiegato nel capitolo precedente e pubblicate sul topic /v_omega.

3.2.3.3 Nodo del simulatore

Il nodo chiamato SIM è fondamentalmente un integratore del modello del sistema dinamico uniciclo.

Sottoscrive al topic /v_omega per aggiornare le informazioni sulle variabili di controllo alla classe integratore che sfrutta (scipy.integrate.ode con il metodo di integrazione 'vode' [1]), svolge un passo di integrazione e pubblica sul topic /x_y_theta il risultato dell'integrazione.

3.3 Matlab

Come detto in precedenza solo a scopo di testing e di taratura dei parametri è stato portato il codice scritto in linguaggio Python per ROS in Matlab.

Non essendoci più necessità di creare un software a se stante per il controllore, i precedenti tre nodi vengono in Matlab riuniti in un unico script, che

1. riceve la traiettoria da file e ne calcola le spline
2. risolve il problema di ottimizzazione MPC grazie al solutore Quadprog [3]
3. integra in avanti grazie alla funzione ode15s [2]
4. visualizza i risultati graficamente

Capitolo 4

Risultati di simulazione

In questo capitolo si presentano alcuni risultati delle simulazioni eseguite su Matlab al variare di alcuni parametri o delle traiettorie, come riportato qui di seguito.

Test 1 Inseguimento di una traiettoria sinusoidale partendo da un punto appartenente alla traiettoria senza ostacoli sul percorso.

Test 2 Inseguimento di una traiettoria sinusoidale partendo da un punto non appartenente alla traiettoria senza ostacoli sul percorso.

Test dal 3 al 9 Inseguimento di una traiettoria sinusoidale con ostacoli sul percorso al variare dei parametri come da Tabella 4.1.

Test dal 10 al 11 Inseguimento di una traiettoria ellittica con ostacoli sul percorso al variare dei parametri come da Tabella 4.1.

	Γ	τ	b	N	q	r
Test 3	10000	0.5	0.5	10	1	0.01
Test 4	10	0.5	0.5	10	1	0.01
Test 5	10000	0.1	0.5	10	1	0.01
Test 6	10000	0.3	0.5	10	1	0.01
Test 7	10000	0.5	0.1	10	1	0.01
Test 8	10000	0.5	2	10	1	0.01
Test 9	10000	0.5	0.5	2	1	0.01
Test 10	10000	0.5	0.5	10	1	0.01
Test 11	10000	0.1	0.5	10	1	0.01

Tabella 4.1: Valori dei parametri utilizzati nei test

Test in ambiente verosimile Inseguimento di una traiettoria in una mappa verosimile di una stanza 12×14 metri.

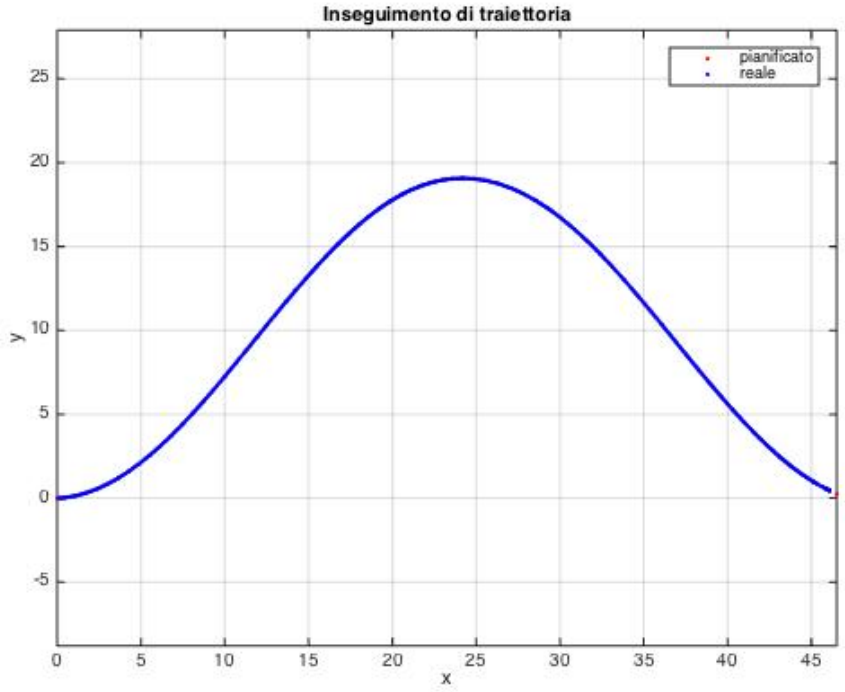

Figura 4.1: Test 1: Inseguimento di traiettoria

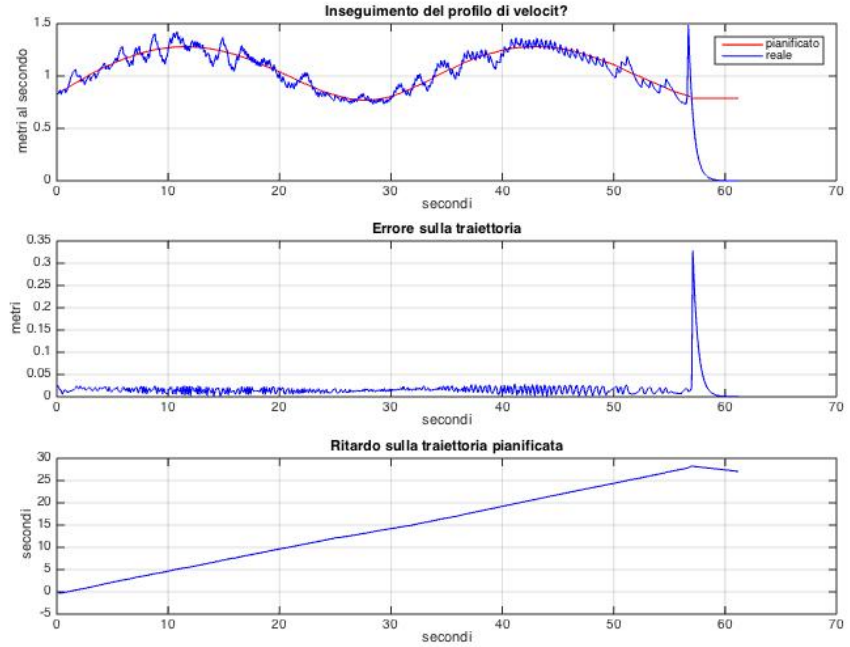

Figura 4.2: Test 1: Velocità, distanza e ritardo

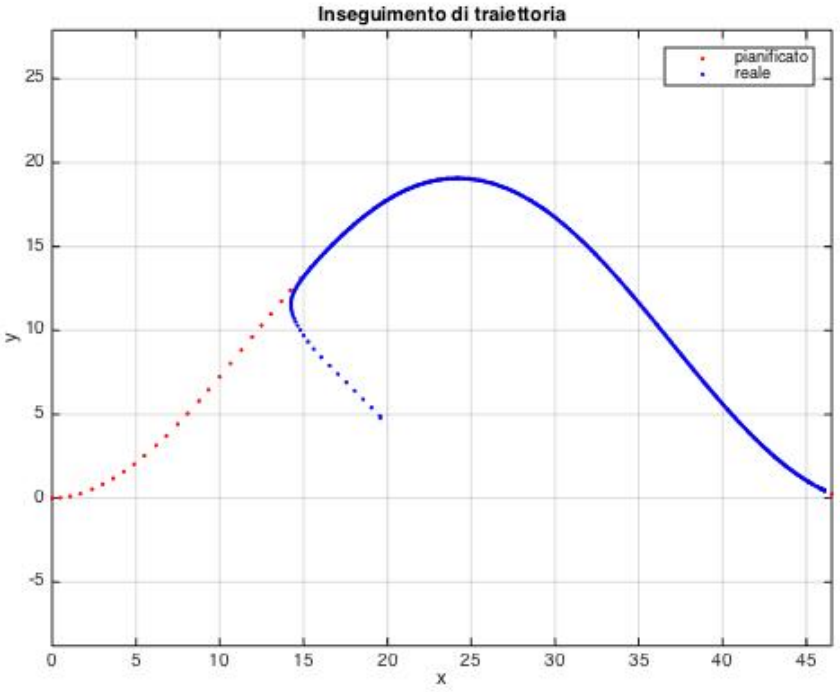

Figura 4.3: Test 2: Inseguimento di traiettoria

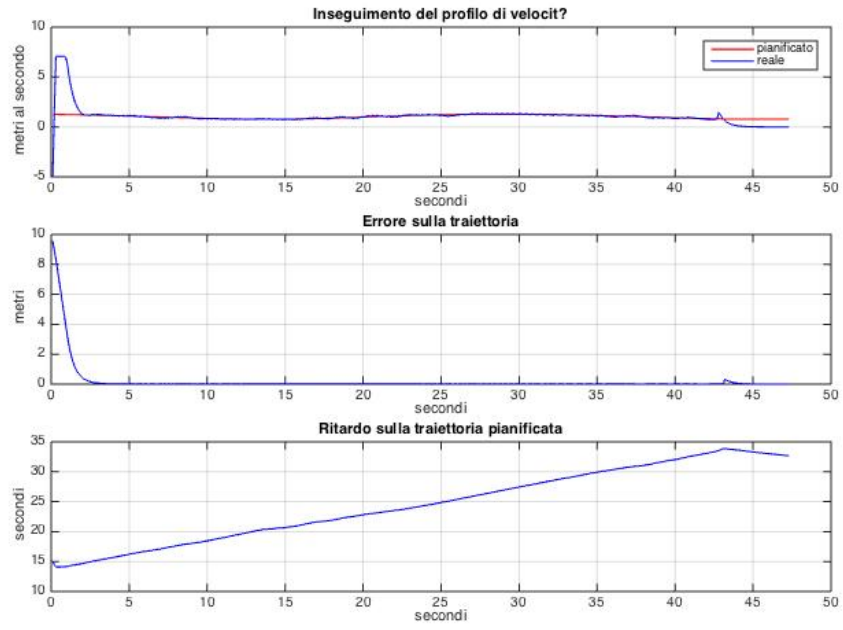

Figura 4.4: Test 2: Velocità, distanza e ritardo

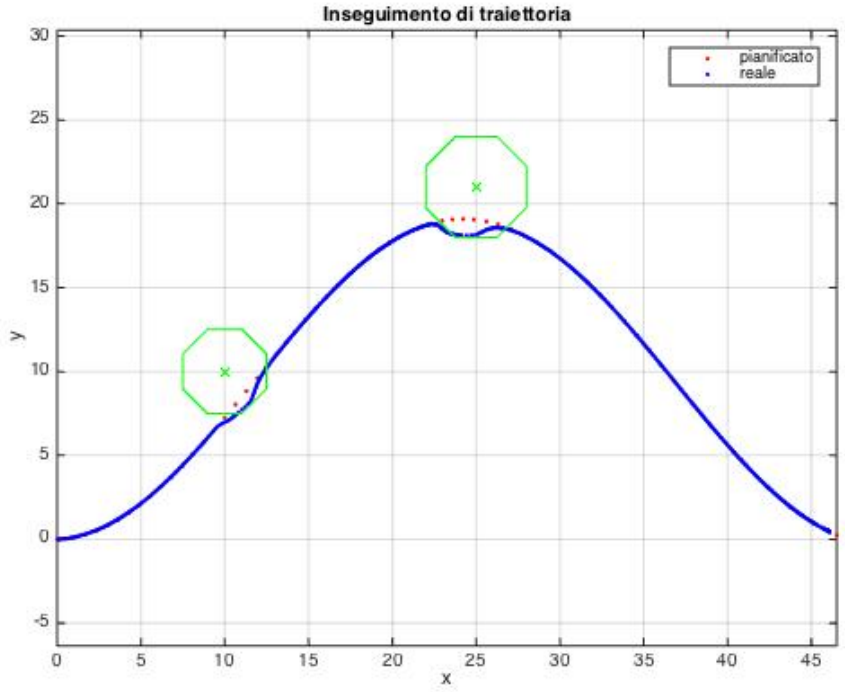

Figura 4.5: Test 3: Inseguimento di traiettoria

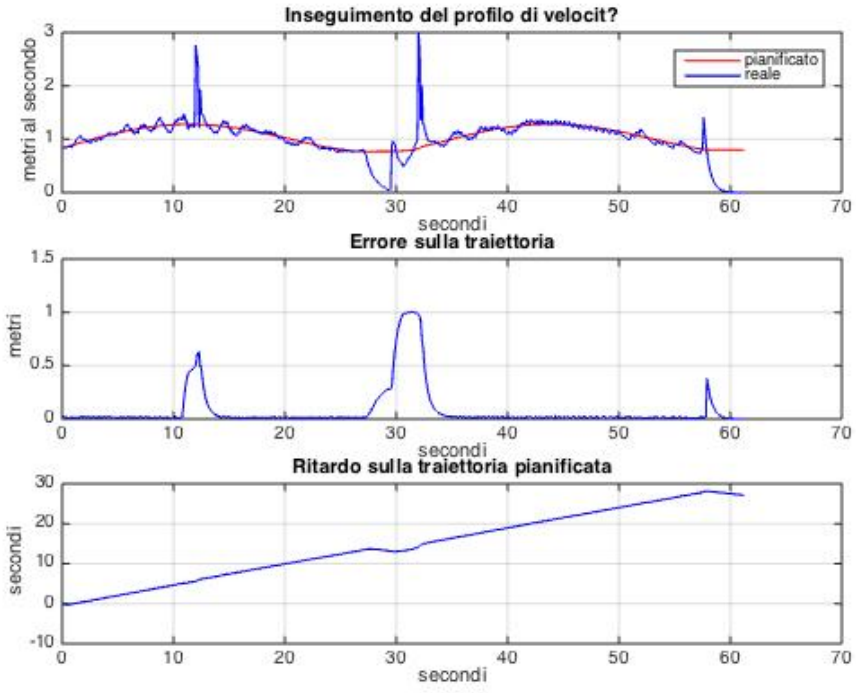

Figura 4.6: Test 3: Velocità, distanza e ritardo

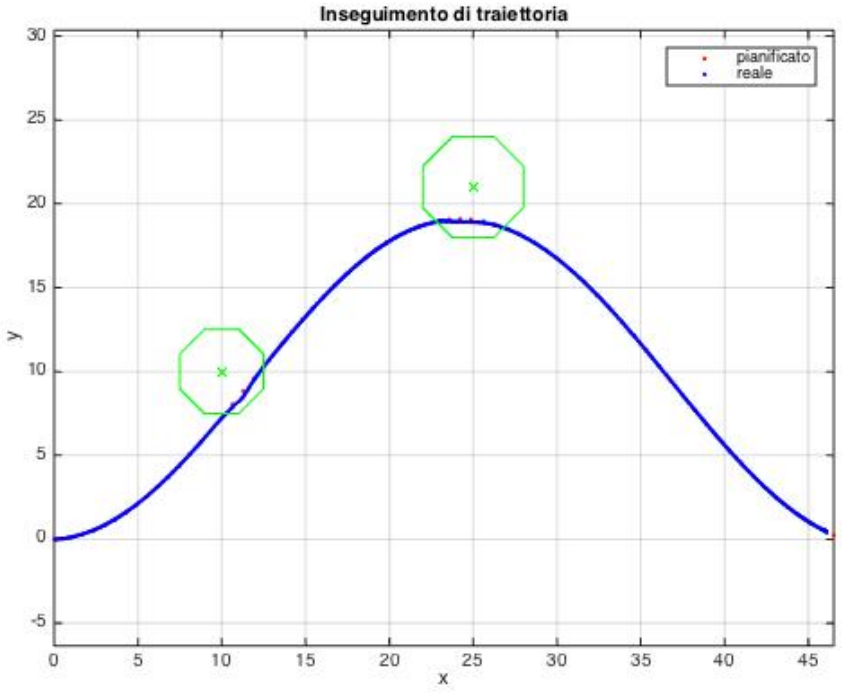

Figura 4.7: Test 4: Inseguimento di traiettoria

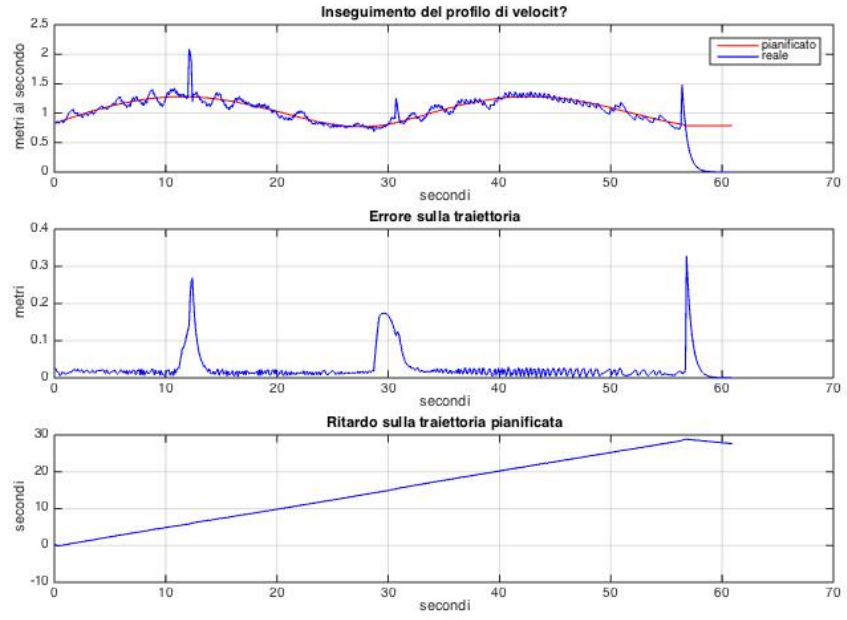

Figura 4.8: Test 4: Velocità, distanza e ritardo

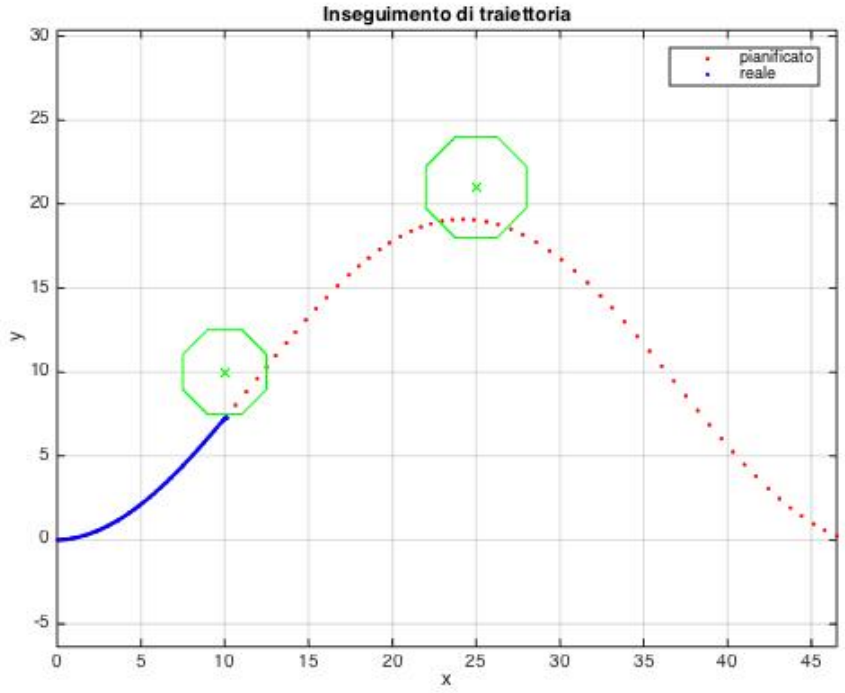

Figura 4.9: Test 5: Inseguimento di traiettoria

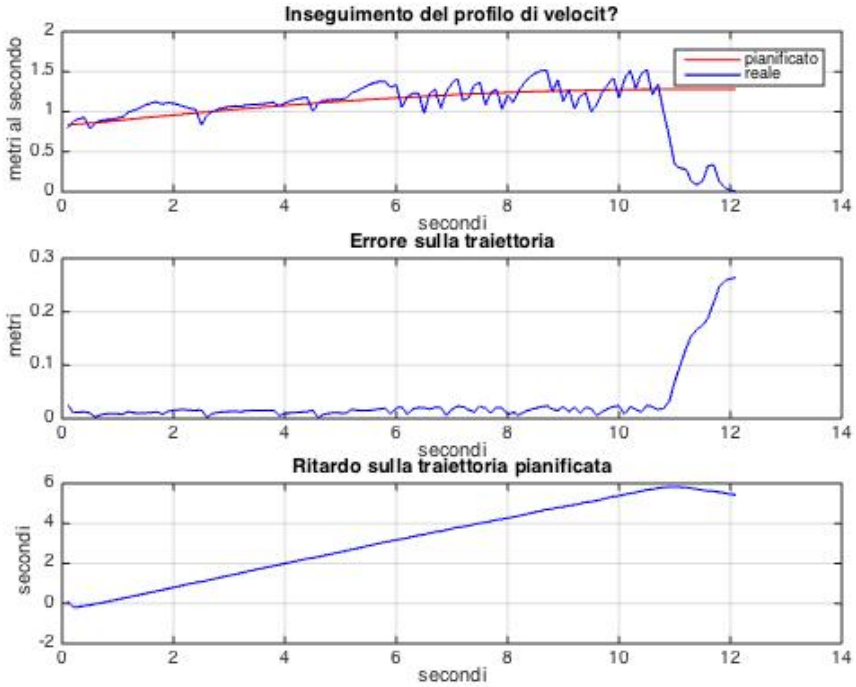

Figura 4.10: Test 5: Velocità, distanza e ritardo

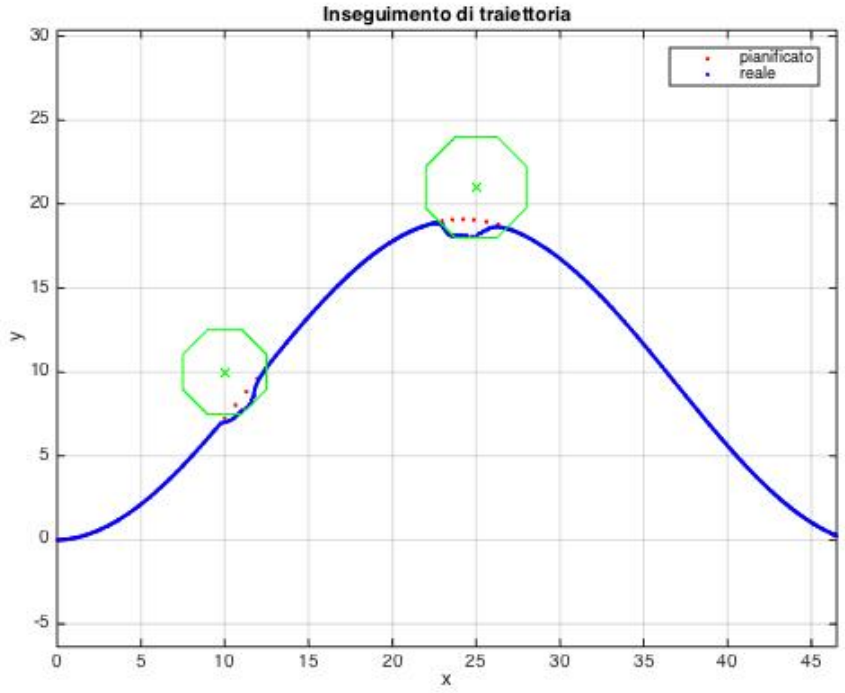

Figura 4.11: Test 6: Inseguimento di traiettoria

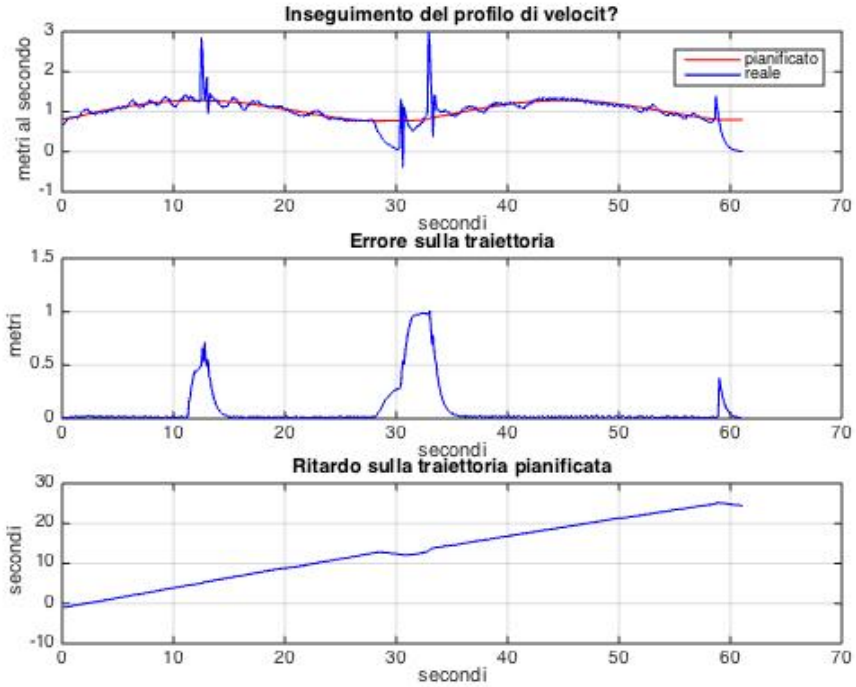

Figura 4.12: Test 6: Velocità, distanza e ritardo

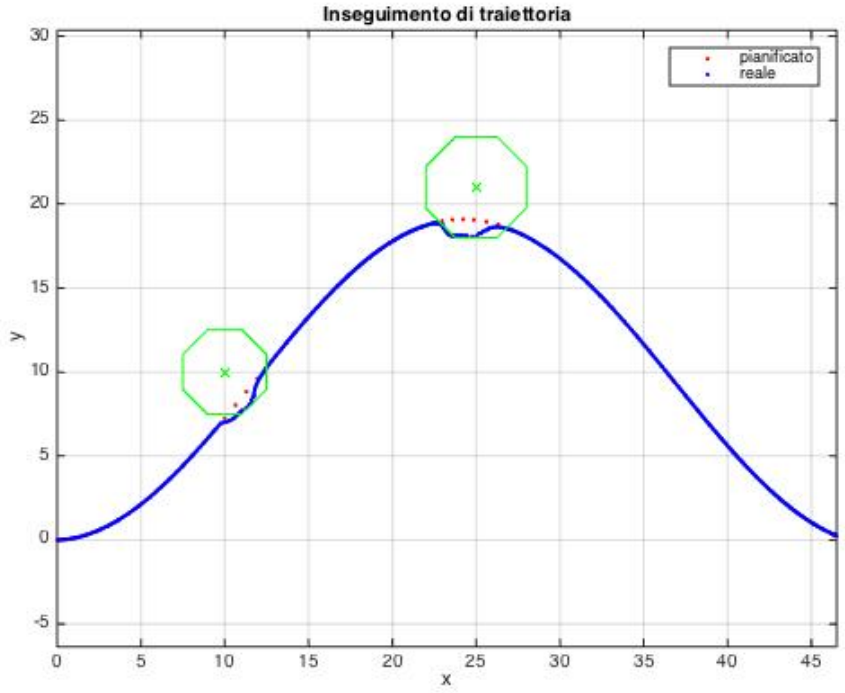

Figura 4.13: Test 7: Inseguimento di traiettoria

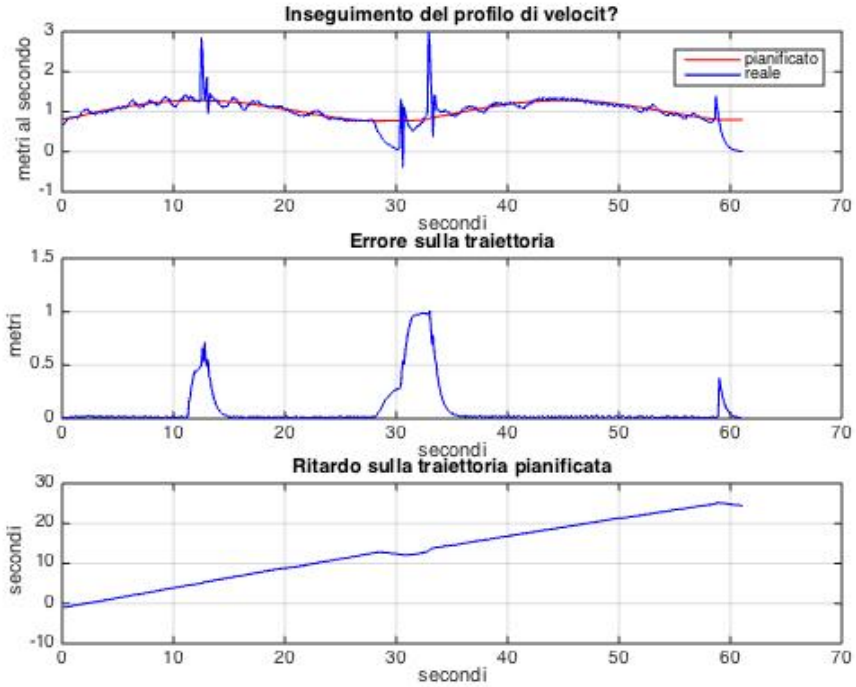

Figura 4.14: Test 7: Velocità, distanza e ritardo

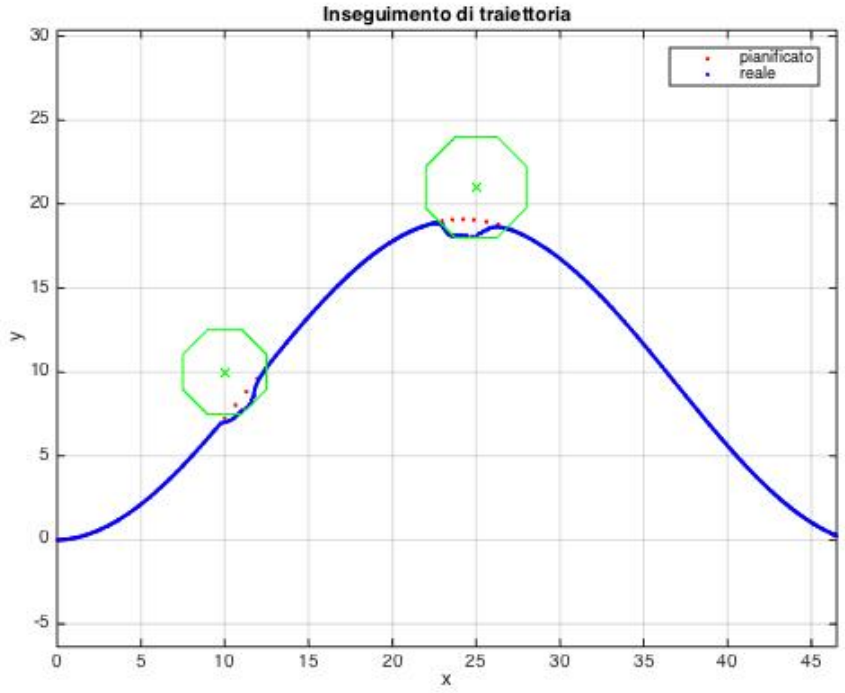

Figura 4.15: Test 8: Inseguimento di traiettoria

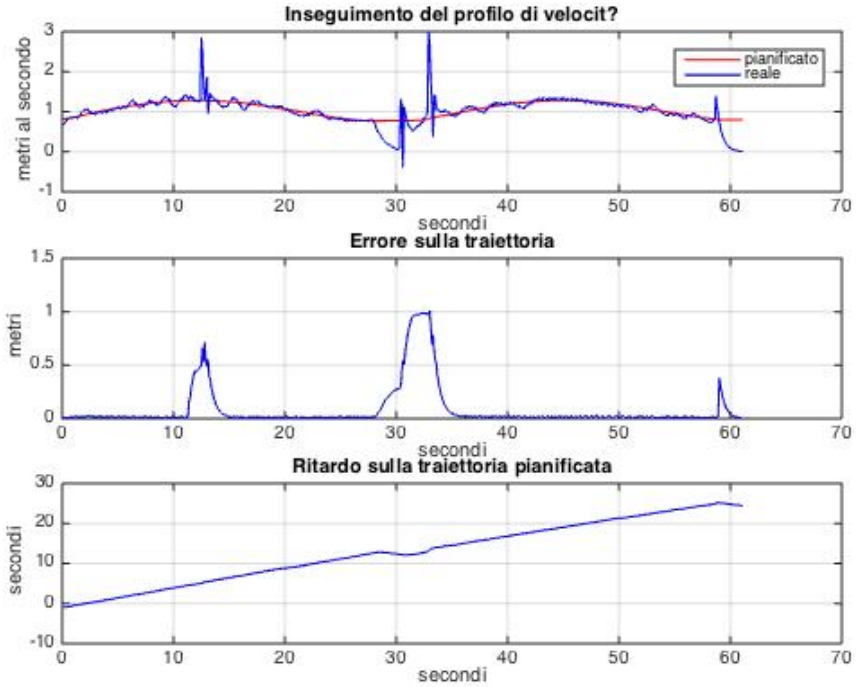

Figura 4.16: Test 8: Velocità, distanza e ritardo

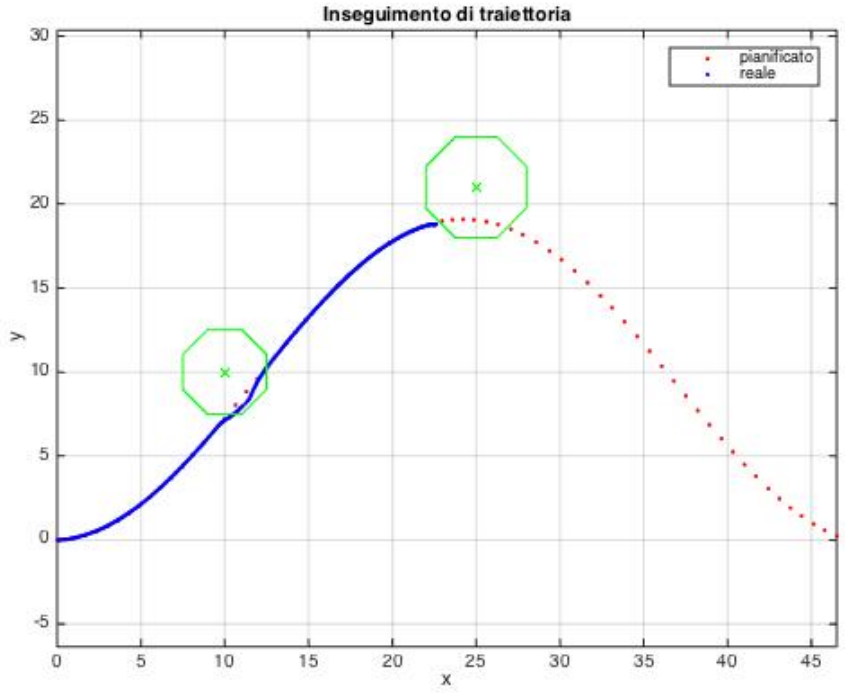

Figura 4.17: Test 9: Inseguimento di traiettoria

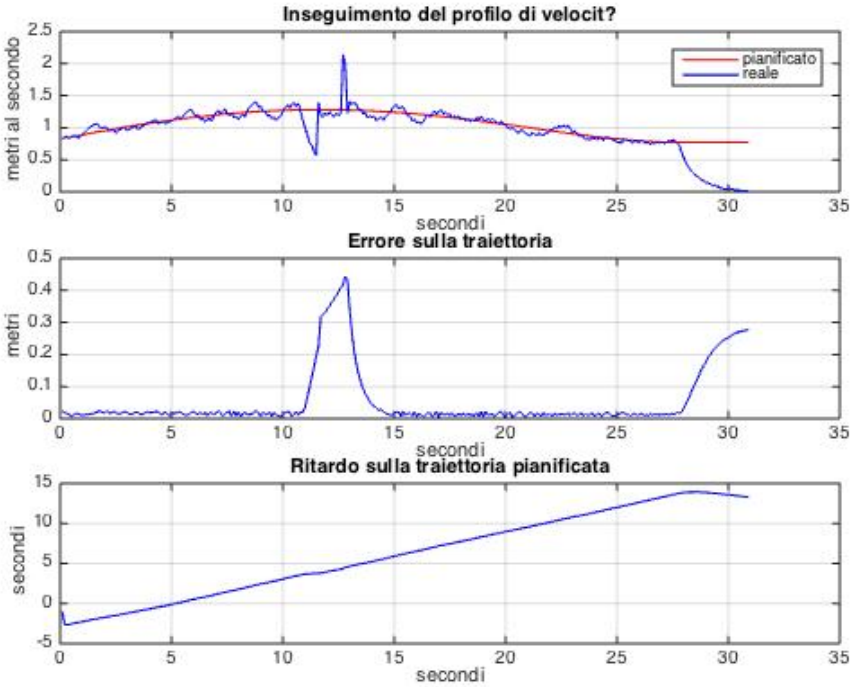

Figura 4.18: Test 9: Velocità, distanza e ritardo

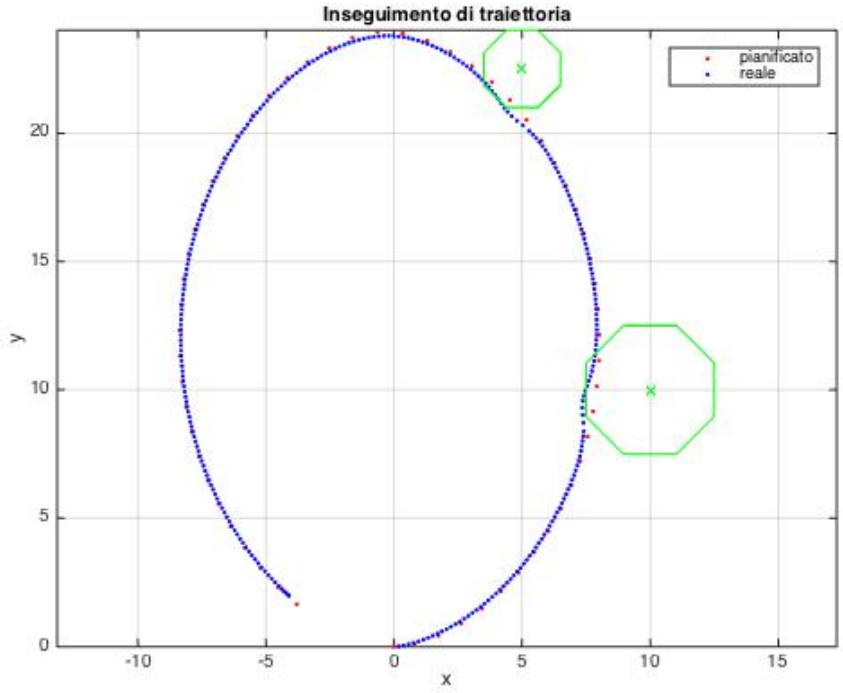

Figura 4.19: Test 10: Inseguimento di traiettoria

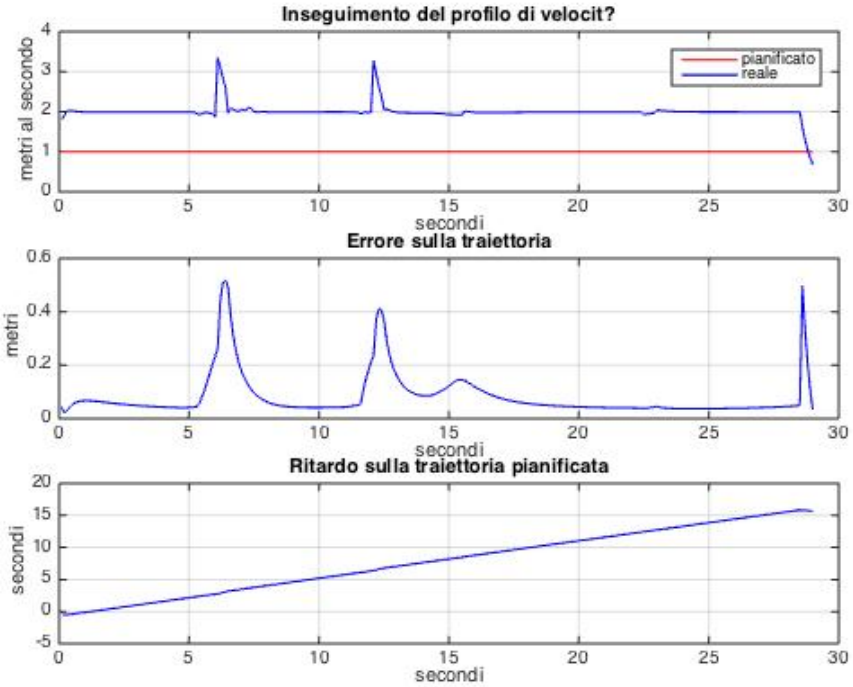

Figura 4.20: Test 10: Velocità, distanza e ritardo

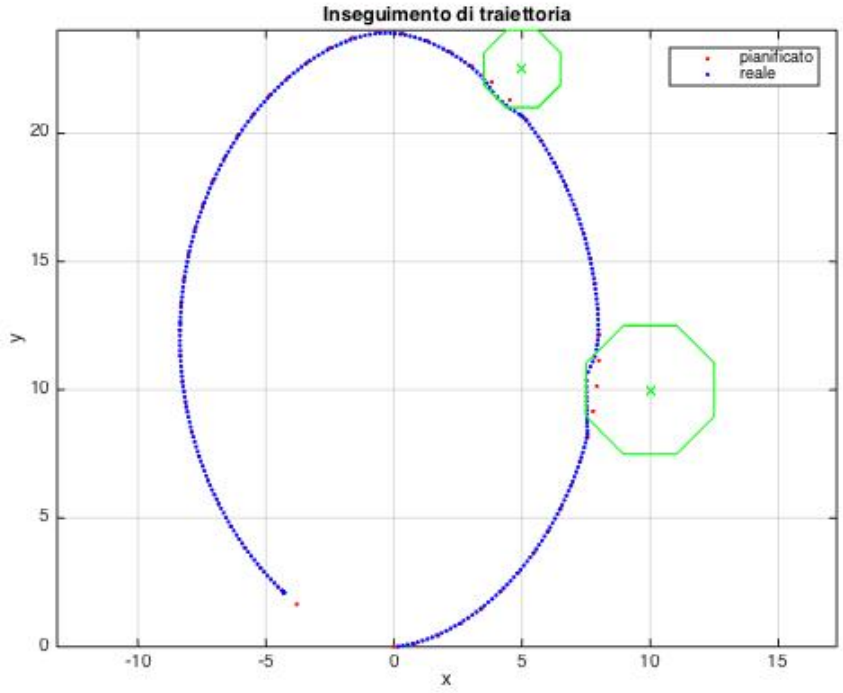

Figura 4.21: Test 11: Inseguimento di traiettoria

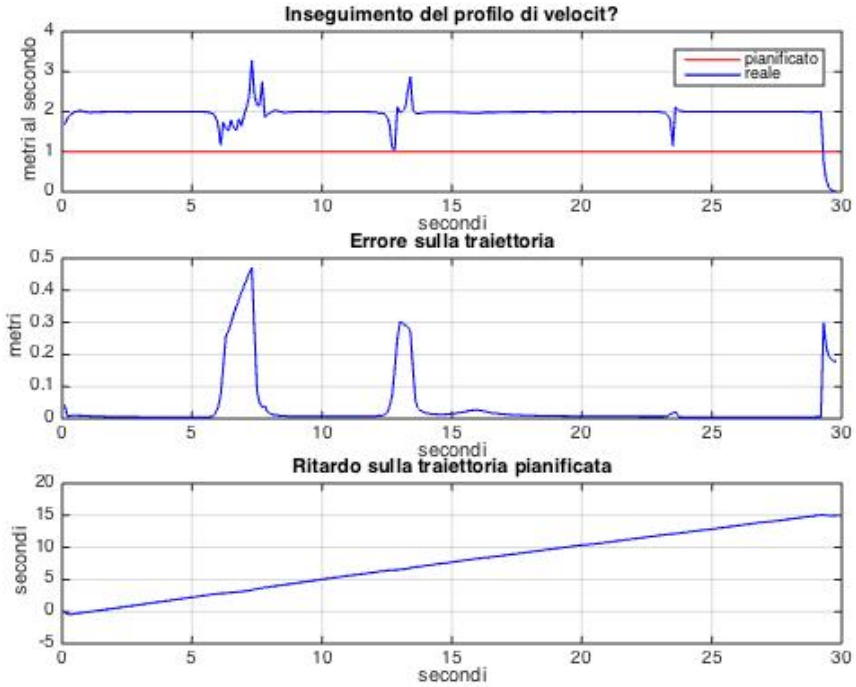

Figura 4.22: Test 11: Velocità, distanza e ritardo

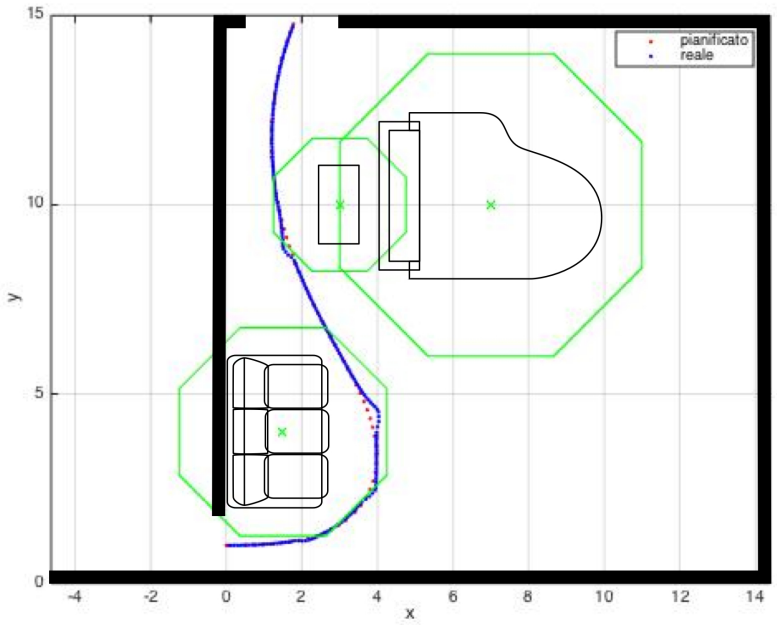

Figura 4.23: Test su ambiente verosimile: Inseguimento di traiettoria

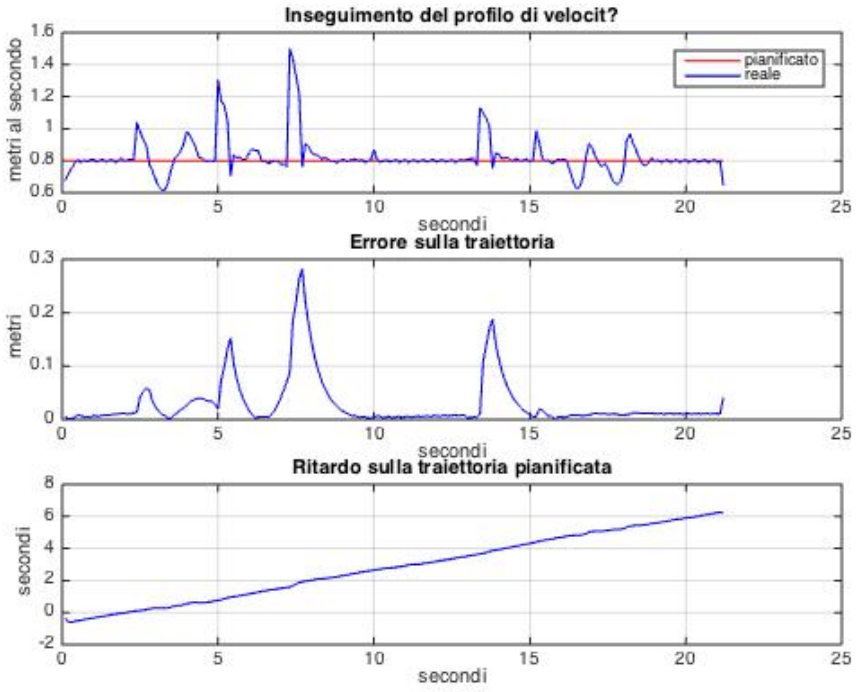

Figura 4.24: Test su ambiente verosimile: Velocità, distanza e ritardo

Conclusioni

In questo lavoro di tesi abbiamo analizzato la possibilità di applicare un controllo MPC mirato al *trajectory following* per una sedia a rotelle autonoma.

Siamo partiti da tecniche già utilizzate in un altro lavoro di tesi [7] per linseguimento di traiettorie di riferimento di un robot con struttura ad uniciclo.

In seguito abbiamo affrontato il problema dell*obstacle avoidance*, modellizzandolo in vincoli da aggiungere allo stesso problema di ottimizzazione per l'MPC del *trajectory following*.

Uno degli scopi di questo lavoro di tesi era quello di migliorare le operazioni di *trajectory following* e di *obstacle avoidance* attualmente implementati su LURCH. Sulla base teorica il controllo MPC è più efficace di quello attualmente implementato, ed anche le simulazioni effettuate sembrerebbero evidenziare dei miglioramenti per quanto riguarda il *trajectory following*, confrontandole con le simulazioni relative al sistema di controllo attuale [5]. Tuttavia non possiamo ancora affermare con certezza quale metodo sia più efficace.

Sicuramente nei lavori futuri si dovrà testare il controllo MPC su LURCH e si dovrà analizzare nel dettaglio il confronto tra i due sistemi di controllo.

Bibliografia

[1] http://docs.scipy.org.

[2] http://it.mathworks.com/help/matlab/ref/ode15s.html.

[3] http://it.mathworks.com/help/optim/ug/quadprog.html.

[4] http://wiki.ros.org.

[5] Luca Calabrese. Robust odometry, localization and autonomous navigation on a robotic wheelchair. Master's thesis, Politecnico di Milano, 2012-2013.

[6] Lalo Magni e Riccardo Scattolini. *Complementi di Controlli Automatici*. Pitagora Editrice Bologna, 2006.

[7] Luca Giulioni. Studio e applicazione di tecniche di controllo predittivo per il posizionameno e il coordinamento di agenti mobili. Master's thesis, Politecnico di Milano, 2011-2012.